9.0

Desenvolvendo Aplicativos para IBM MQ

Nota

Antes de usar estas informações e o produto suportado por elas, leia as informações em “Avisos” na página 1375.

Esta edição se aplica à versão 9 liberação 0 do IBM® MQ e a todas as liberações e modificações subsequentes até que seja indicado de outra forma em novas edições.

Ao enviar informações para a IBM, você concede à IBM um direito não exclusivo de usar ou distribuir as informações da maneira que julgar apropriada, sem incorrer em qualquer obrigação para com você

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.

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

- LearnMQ *(aprenda os conceitos básicos, execute uma demo, codifique um app, siga tutoriais mais avançados)*
- Downloads do desenvolvedor do MQ *(incluindo edições de desenvolvedor e versões de avaliação grátis)*

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 6
- “Considerações de design para aplicativos IBM MQ” na página 47

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
- ActiveX (deprecated; use .NET)

Consulte também “Aplicativos orientados a objetos” na página 11.

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)
- C++
- Visual Basic

Consulte “Gravando e implementando programas do IBM MQ .NET” na página 550.

ULW O IBM MQ também suporta a API do MQ Light, que implementa o protocolo do OASIS AMQP 1.0. Há APIs do sistema de mensagens para as linguagens a seguir:

- Node.js
- Ruby
- Java
- Python
- Maven (projeto esqueleto; usa a api do Java)
- Gradle (projeto esqueleto; usa a api do Java)

Consulte também “Desenvolvendo aplicativos clientes AMQP” na página 678.

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:

- V 9.0.4 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 694 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)
- IBM i RPG (apenasIBM MQ for IBM i)
- z/OS PL/I (apenasIBM MQ for z/OS)

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 711. 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.

Conceitos relacionados

“Desenvolvendo aplicativos do Microsoft Windows Communication Foundation (WCF) com o IBM MQ” na página 1266
O canal customizado do Microsoft Windows Communication Foundation (WCF) para o IBM MQ envia e recebe mensagens entre clientes e serviços do WCF.

Tarefas relacionadas

“Desenvolvendo aplicativos para o MQ Telemetry” na página 1242

Informações relacionadas

IBM Message Service Client for .NET

Conceitos de desenvolvimento de aplicativos

É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

Antes de começar a projetar e gravar seus aplicativos IBM MQ, familiarize-se com os conceitos básicos do IBM MQ, consulte os tópicos em Visão geral técnica. Para obter informações sobre os tipos de aplicativos

em que é possível gravar para o IBM MQ, consulte “Desenvolvendo aplicativos para o IBM MQ” na página 5.

Use os seguintes links para descobrir sobre conceitos específicos do IBM MQ para desenvolvimento de aplicativos:

Conceitos relacionados

“Usando o MQI em um aplicativo cliente” na página 913
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.

“Programas de Saída de Canal para Canais de Mensagens” na página 965
Essa coleção de tópicos contém informações sobre os programas de saída do canal IBM MQ para canais do sistema de mensagens.

“Considerações de design para aplicativos IBM MQ” na página 47
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 715
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

“Desenvolvendo aplicativos MQI com o IBM MQ” na página 711
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.

“Aplicativos orientados a objetos” na página 11
O IBM MQ fornece suporte para o .NET, ActiveX, C++, Java e JMS. 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).

“Usando o IBM MQ classes for JMS” na página 75
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

“Usando a Interface de modelo de objeto do componente (IBM MQ Automation Classes for ActiveX)” na página 583
O IBM MQ Automation Classes for ActiveX (MQAX) são os componentes ActiveX que fornecem as classes que podem ser usadas em seu aplicativo para acessar o IBM MQ.

“Usando o IBM MQ classes for Java” na página 323
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 do .NET” na página 535
IBM MQ classes for .NET permite que um programa gravado na estrutura de programação do .NET se conecte ao IBM MQ como um IBM MQ MQI client ou se conecte diretamente a um servidor IBM MQ.

“Desenvolvendo aplicativos C++” na página 505
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 1005

É possível escrever um aplicativo IBM MQ em uma das várias linguagens processuais e executar o aplicativo em várias plataformas diferentes.

Tarefas relacionadas

"Desenvolvendo serviços da web com o IBM MQ" na página 1306
É possível desenvolver aplicativos IBM MQ para serviços da web usando o transporte IBM MQ para SOAP.

"Usando os programas processuais de amostra do IBM MQ" na página 1075
Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

Informações relacionadas

Cenários de Suporte Transacional

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

Multi

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 a partir de dentro do CICS for IBM i IBM i, TXSeries para sistemas AIX, HP-UX, Solaris e Windows.
- Use o enfileiramento de mensagens a partir de dentro do Encina para o AIX, HP-UX, Solaris e sistemas Windows.
- Use o enfileiramento de mensagens de dentro do Tuxedo para sistemas AIX, AT & T, HP-UX, Solarise 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 “Criando aplicativos em lote do z/OS” na página 1041 e “Considerações em lote do z/OS” na página 726 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 filas compartilhadas.

z/OS As diferenças entre os ambientes suportados e suas limitações são mais bem explicadas em “Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887.

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

“Considerações de design para aplicativos IBM MQ” na página 47
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 715
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

“Usando o IBM MQ classes for JMS” na página 75

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

"Usando a Interface de modelo de objeto do componente (IBM MQ Automation Classes for ActiveX)" na página 583
O IBM MQ Automation Classes for ActiveX (MQAX) são os componentes ActiveX que fornecem as classes que podem ser usadas em seu aplicativo para acessar o IBM MQ.

"Usando o IBM MQ classes for Java" na página 323
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 do .NET" na página 535
IBM MQ classes for .NET permite que um programa gravado na estrutura de programação do .NET se conecte ao IBM MQ como um IBM MQ MQI client ou se conecte diretamente a um servidor IBM MQ.

"Desenvolvendo aplicativos do Microsoft Windows Communication Foundation (WCF) com o IBM MQ" na página 1266
O canal customizado do Microsoft Windows Communication Foundation (WCF) para o IBM MQ envia e recebe mensagens entre clientes e serviços do WCF.

"Desenvolvendo aplicativos C++" na página 505
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 1005
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 1075
Estes programas de amostra são gravados em linguagens processuais e demonstram usos típicos do Message Queue Interface (MQI). Programas IBM MQ em diferentes plataformas.

Informações relacionadas

Assegurando

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.

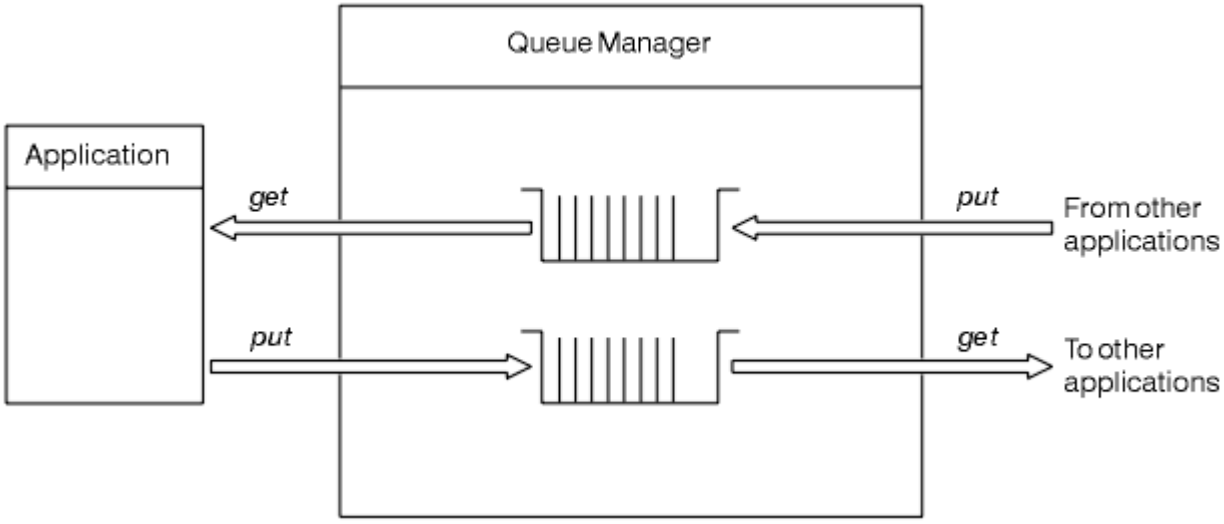


Figura 1. Filas, mensagens e aplicativos

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 1005.
- 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.

Aplicativos orientados a objetos

O IBM MQ fornece suporte para o .NET, ActiveX, C++, Java e JMS. 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 12.

.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).

ActiveX

O IBM MQ ActiveX é comumente conhecido como o MQAX. O MQAX é incluído como parte do IBM MQ for Windows. O suporte para ActiveX foi estabilizado no nível IBM WebSphere MQ 6.0. Para obter informações sobre como codificar programas usando o IBM MQ Object Model em ActiveX Usando o Component Object Model Interface (WebSphere MQ Classes de Automação para ActiveX).

V 9.0.0 No IBM MQ 9.0, o suporte para o MicrosoftActive X foi descontinuado. As classes do IBM MQ para .NET são a tecnologia de substituição recomendada. Para obter mais informações, consulte Desenvolvendo aplicativos .NET.

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).

Java

Consulte Usando IBM MQ classes for Java para obter informações sobre codificação de programas usando o IBM MQ Object Model no Java. IBM não fará primoramentos no IBM MQ classes for Java e eles estão funcionalmente estabilizados no nível enviado em 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 “Escolhendo usar IBM MQ classes for Java ou IBM MQ classes for JMS” na página 50.

JMS

IBM MQ também fornece classes que implementam a especificação do Java Message Service (JMS). Para obter detalhes sobre o IBM MQ classes for JMS, consulte Usando IBM MQ classes for JMS. Para obter informações sobre as diferenças entre o IBM MQ classes for Java e IBM MQ classes for JMS para ajudá-lo a decidir qual usar, consulte “Escolhendo usar IBM MQ classes for Java ou IBM MQ classes for JMS” na página 50.

O IBM Message Service Client for C/C++ e o IBM Message Service Client 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 mais informações, consulte Introdução ao IBM Message Service Client for .NET.

Conceitos relacionados

“Desenvolvendo aplicativos MQI com o IBM MQ” na página 711
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.

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

Informações relacionadas

Visão geral técnica

Referência de desenvolvimento de aplicativos

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 do 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 14 representa uma mensagem e mostra como ele é logicamente dividido em propriedades de mensagem e dados do aplicativo.

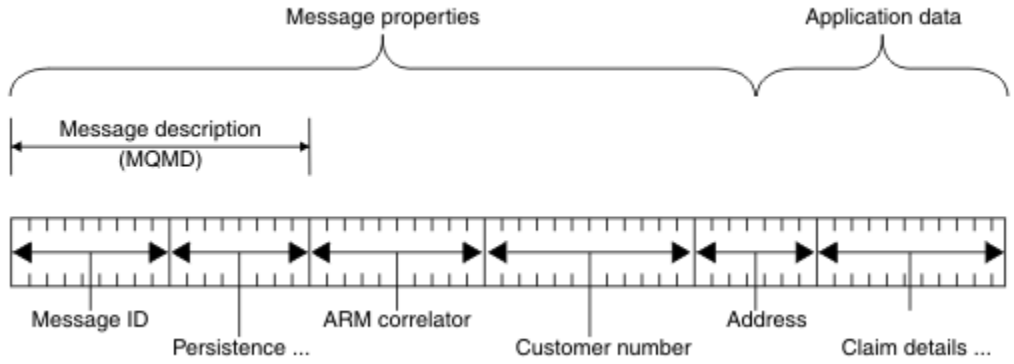


Figura 2. Representação de uma mensagem

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.

ULW No UNIX, 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 1024.

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 754.

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 44.

Existem diferentes versões do descritor de mensagens. Informações adicionais para agrupamento e segmentação de mensagens (consulte "Grupos de mensagens" na página 41) 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. Os gerenciadores de filas também podem retornar códigos de razão MQI no campo *Feedback* . O IBM MQ define um intervalo 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*

Não suportado no IBM MQ for z/OS.

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.

Para obter uma descrição de mensagens segmentadas, consulte “Segmentação de mensagem” na página 789.

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 988). 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 988 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 741), 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 *usr.myproperty* do IBM MQ (uma sequência de caracteres) fica acessível a um aplicativo JMS usando a chamada `message.getStringProperty('myproperty')` do JMS. 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". Inversamente, um conjunto de propriedades de usuário em um aplicativo JMS pode ser acessado em um aplicativo IBM MQ ao incluir o prefixo "usr." ao 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.

Ao fluir para um gerenciador de filas anterior ao IBM WebSphere MQ 7.0, as propriedades da mensagem, exceto aquelas no descritor de mensagens, consideram o comprimento da mensagem. Portanto, é necessário aumentar o valor do atributo *MaxMsgLength* de canais indo para um sistema anterior à IBM WebSphere MQ 7.0, conforme necessário, para compensar o fato de que mais dados podem ser enviados para cada mensagem. Como alternativa, é possível diminuir a fila ou o gerenciador de filas *MaxMsgLength*, para que o nível geral de dados que estão sendo enviados ao redor do sistema permaneça o mesmo.

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
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

Propriedade	Sinônimo para
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 de propriedade começando com " mq " em qualquer mistura de letra minúscula ou maiúscula e não começando com "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 no meio. Não é possível ter um ponto vazio na hierarquia. Da mesma forma, um nome de propriedade não pode terminar em um caractere ".".
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 o 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 29 mostra o processo de seleção usando a chamada MQSUB.

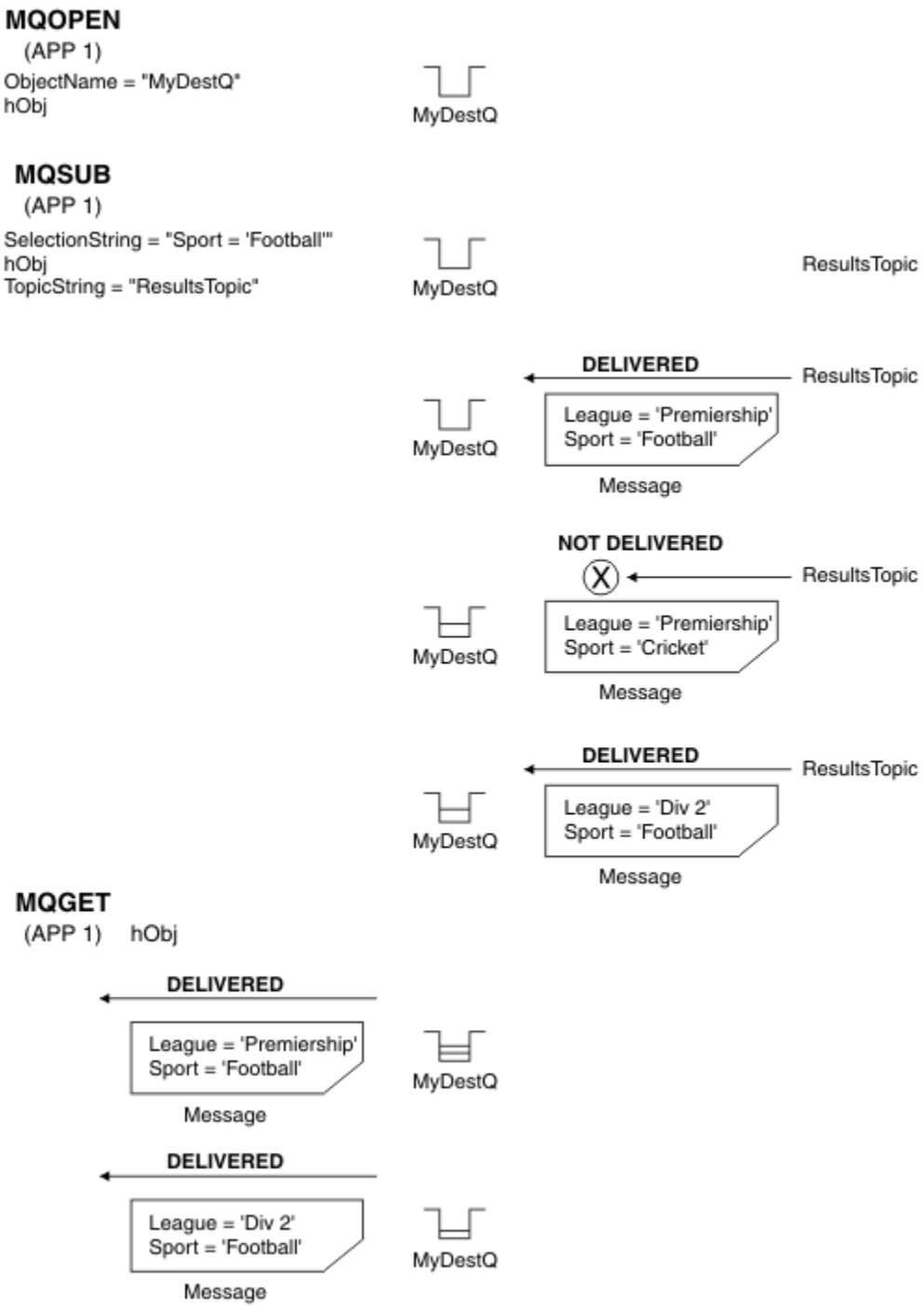


Figura 3. Seleção usando a chamada MQSUB

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 30 mostra o processo de seleção usando a chamada MQOPEN.

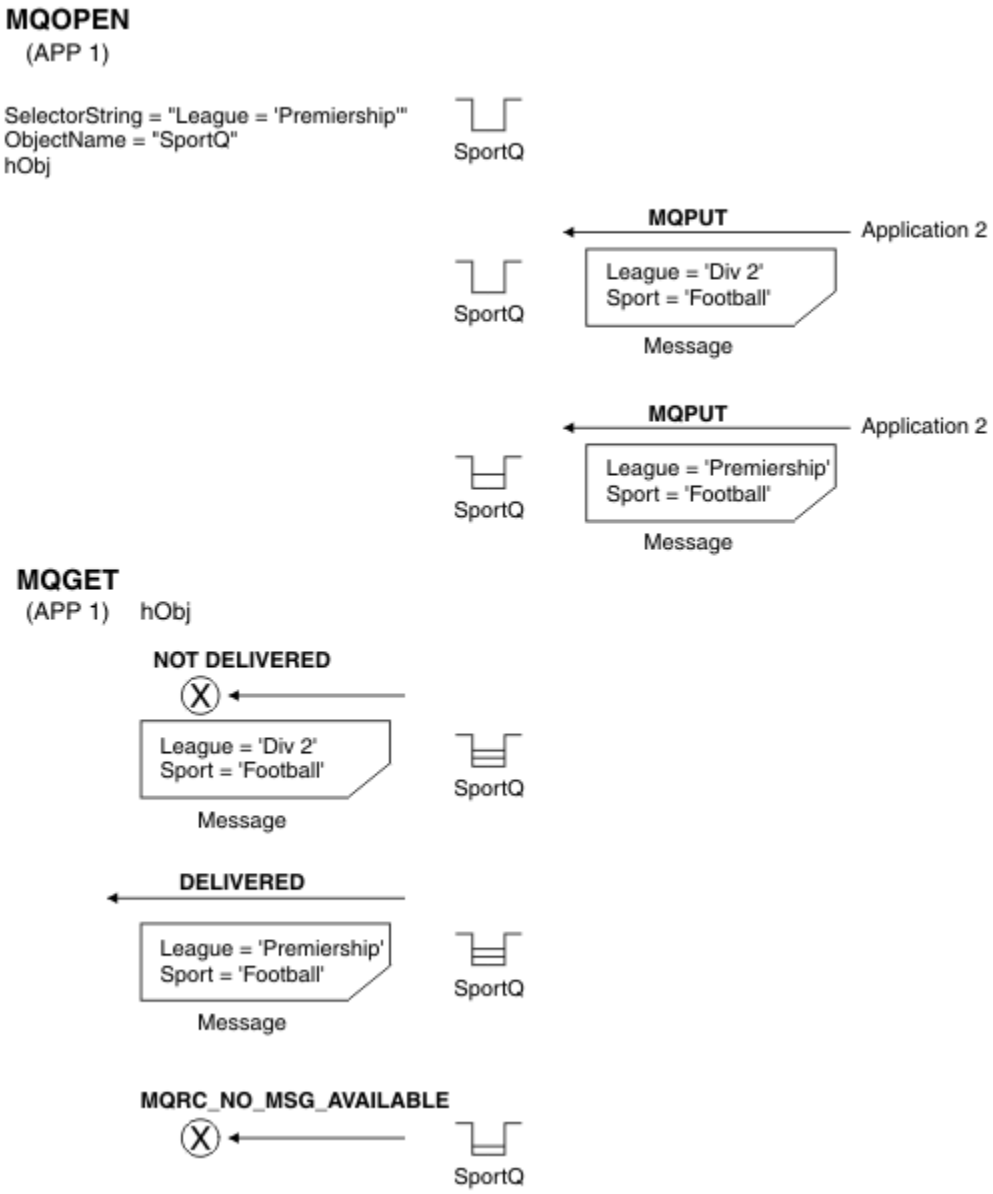


Figura 4. Seleção usando a chamada MQOPEN

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 37
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 128.

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 37
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 24 e “Restrições de nome da propriedade” na página 25 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 128 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` to `\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. 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. 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. 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. 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 23
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.

Informações 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 utilizado por muitas linguagens de programação para representar 'não igual a' é !=. Essa representação não é um sinônimo válido para <>; por exemplo, a <> b é válido, enquanto que 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 lógica/disjunção; por exemplo, a `&& b` deve ser especificado como a `E 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 escrita como `(b> 20) E (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 31
Visão geral do comportamento de seleção do IBM MQ.

Informações 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. V 9.0.0 Caracteres além de 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 propriedades 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.

Informações 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 40 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.

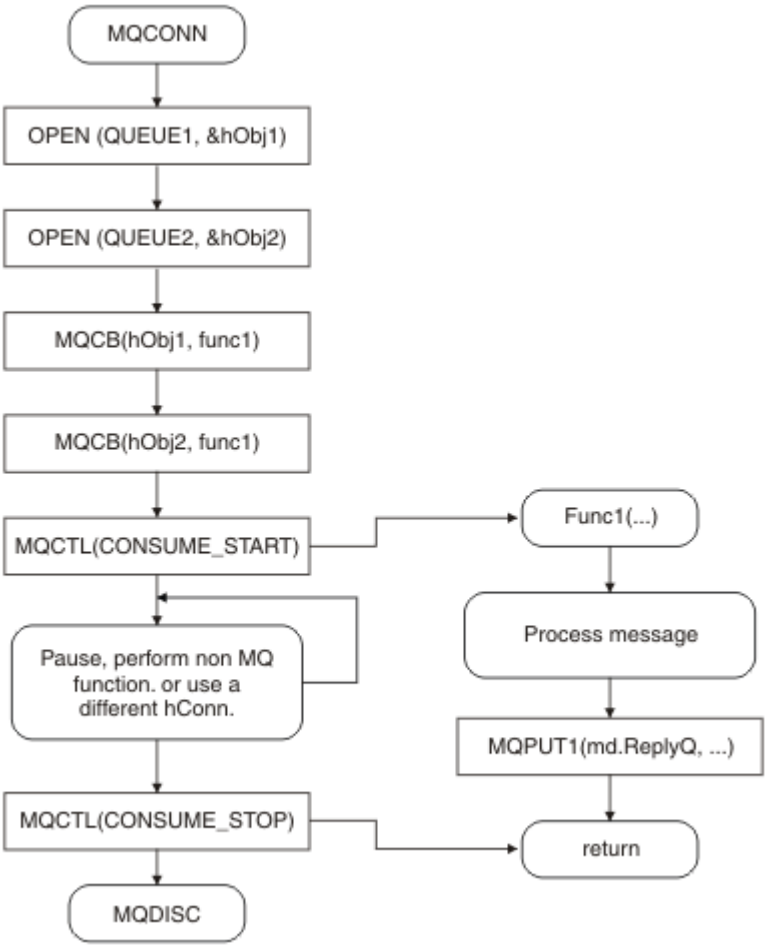


Figura 5. Aplicativo padrão acionado por mensagens consumindo de duas filas

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 41Esse 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.

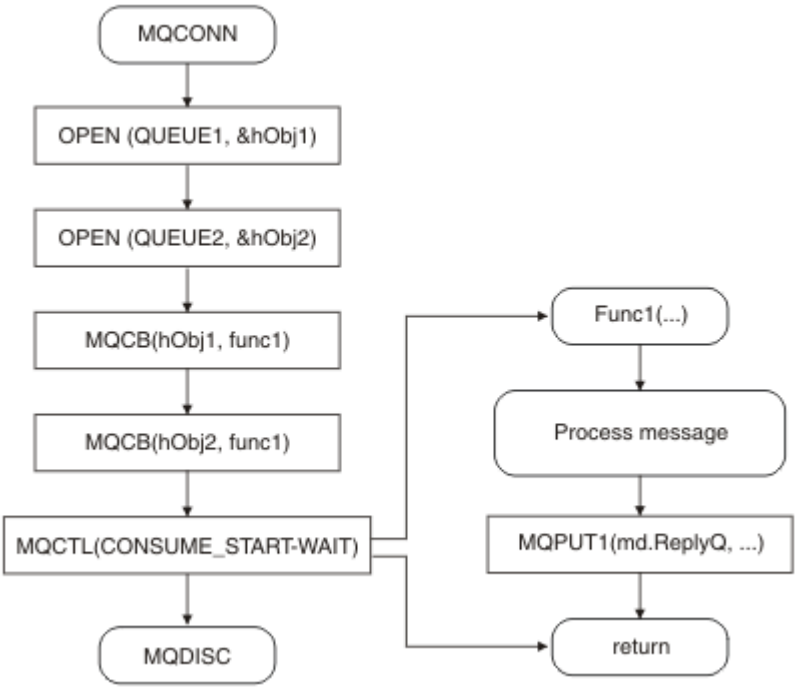


Figura 6. Aplicativo acionado por mensagens de encadeamento único consumindo de duas filas

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 771). 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 41 mostra um grupo de mensagens lógicas:

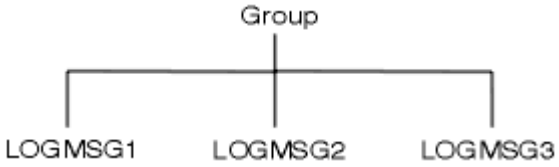


Figura 7. Grupo de mensagens lógicas

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.

Segmentar

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 789.

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 42 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 42 mostra um grupo de mensagens lógicas, algumas das quais são segmentadas:

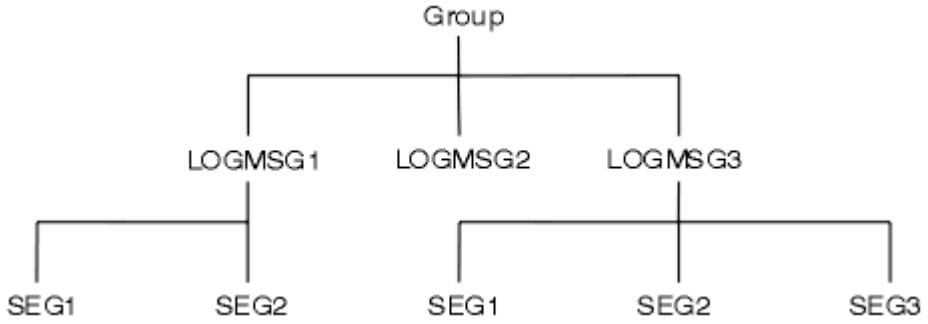


Figura 8. Mensagens segmentadas

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 789
Use estas informações para aprender sobre a segmentação de mensagens. Esse recurso não é suportado no IBM MQ for z/OS ou por aplicativos que utilizam o IBM MQ classes for JMS.

Referências relacionadas

"Ordenação lógica e física" na página 771
As mensagens em filas podem ocorrer (dentro de cada nível de prioridade) em ordem *física* ou *lógica*.

Informações relacionadas
MQMD - descritor de mensagem

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 851.
- ULW z/OS 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 1055 para 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 restauração para filas privadas sobreviva a reinicializações do gerenciador de filas, configure o atributo *HardenGetBackout* para MQQA_BACKOUT_HARDENED; caso contrário, se o gerenciador de filas tiver que reiniciar, 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 851.

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 1059).

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 756.

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 756
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 747
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.

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 1034, 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 530, 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 1037, mas quando definir o projeto Visual Basic, digite `MqType=3` no campo **Argumentos de compilação condicional**.
- Se o componente usar o IBM MQ Automation Classes para ActiveX (MQAX), defina uma variável de ambiente, GMQ_MQ_LIB, com o valor `mqic32xa.dll`.

  É possível definir a variável de ambiente a partir de seu aplicativo ou ela pode ser definida de modo que seu escopo seja para todo o sistema. Entretanto, defini-la como para todo o sistema pode causar

que qualquer aplicativo MQAX existente que não defina a variável de ambiente a partir do aplicativo se comporte incorretamente.

Usando o IBM MQ com o WebSphere Application Server

Use este tópico para entender o uso do IBM MQ com o WebSphere Application Server.

Aplicativos que são escritos em Java que estão em execução no WebSphere Application Server podem usar a especificação Java Messaging 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 com relação ao 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 Mapeando mensagens do JMS para mensagens do IBM MQ.

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 inúmeras linguagens de programação processuais e orientadas por objetos para escrever 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 filas compartilhadas 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 filas compartilhadas 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 51.

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 sistema de mensagens publicar/assinar, consulte Sistema de mensagens 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

É possível que você queira iniciar alguns programas automaticamente ou fazer com que eles esperem até que uma determinada mensagem chegue em uma fila (usando o recurso de *acionamento* do IBM MQ. Consulte “Iniciando aplicativos IBM MQ usando acionadores” na página 863). 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 o "Executando aplicativos no ambiente do IBM MQ MQI client" na página 920 para obter informações adicionais.

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 52 IBM i e "Considerações de design e desempenho de aplicativos IBM i" na página 56.

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 54.

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 851 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 "Usando e escrevendo aplicativos no IBM MQ for z/OS" na página 887.

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

"Considerações de design e desempenho de aplicativos z/OS" na página 58
O design do aplicativo é um dos fatores mais importantes que afetam o desempenho. Use este tópico para entender alguns dos fatores de design envolvidos no desempenho.

"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 6

É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

"Escrevendo um aplicativo processual para enfileiramento" na página 715
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

"Desenvolvendo aplicativos do .NET" na página 535
IBM MQ classes for .NET permite que um programa gravado na estrutura de programação do .NET se conecte ao IBM MQ como um IBM MQ MQI client ou se conecte diretamente a um servidor IBM MQ.

"Desenvolvendo aplicativos C++" na página 505
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 o IBM MQ classes for JMS" na página 75
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

"Usando o IBM MQ classes for Java" na página 323
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.

"Usando a Interface de modelo de objeto do componente (IBM MQ Automation Classes for ActiveX)" na página 583
O IBM MQ Automation Classes for ActiveX (MQAX) são os componentes ActiveX que fornecem as classes que podem ser usadas em seu aplicativo para acessar o IBM MQ.

Informações relacionadas

Visão Geral Técnica do IBM MQ

Escolhendo usar IBM MQ classes for Java ou IBM MQ classes for JMS

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. Cada abordagem tem suas vantagens.

Nota: 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.

O IBM MQ classes for Java contém a Message Queue Interface (MQI), a API nativa do IBM MQ e usa o mesmo modelo de objeto que outras interfaces orientadas por objeto, enquanto que o IBM MQ classes for Java Message Service implementa as interfaces do Java Message Service (JMS) da Oracle.

Se você estiver familiarizado com o IBM MQ em ambientes diferentes de Java, usando linguagens processuais ou orientadas por objetos, será possível transferir seu conhecimento existente para o ambiente Java usando o IBM MQ classes for Java. É possível também explorar o intervalo completo de recursos do IBM MQ, nem todos estão disponíveis no IBM MQ classes for JMS.

Se não estiver familiarizado com o IBM MQ ou já tiver experiência com o JMS, poderá achar mais fácil usar a API familiar do JMS para acessar os recursos do IBM MQ, usando o IBM MQ classes for JMS. JMS também é uma parte integrante da plataforma Java Platform, Enterprise Edition (Java EE). Os aplicativos Java EE podem usar beans acionados por mensagem (MDBs) para processar mensagens de forma assíncrona. O JMS também é o mecanismo padrão para o Java EE interagir com os sistemas de mensagens assíncronos, como o IBM MQ. Cada servidor de aplicativos compatível com o Java EE deve incluir um provedor do JMS, por isso, é possível usar o JMS para se comunicar entre servidores

de aplicativos diferentes ou transportar um aplicativo de um provedor do JMS para outro, sem qualquer mudança no aplicativo.

"Usando o IBM MQ classes for Java" na página 323
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.

"Usando o IBM MQ classes for JMS" na página 75
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

Cenários: WebSphere Application Server com o IBM MQ
Cenários: perfil Liberty do WebSphere Application Server com o IBM MQ

Técnicas de design para mensagens

Considere os aspectos fornecidos nestas informações para ajudá-lo a projetar mensagens.

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 UNIX 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.

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 794).
- 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 “Sinalização” na página 795.
- Estabelecendo uma saída de retorno de chamada para ser acionada quando uma mensagem chegar; consulte “Consumo assíncrono de mensagens do IBM MQ” na página 39.

- 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 44.

Iniciando programas IBM MQ automaticamente

Use o *acionamento* do IBM MQ 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 863. 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 16.

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 56
- “Problemas de desempenho específicos” na página 57

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 47.

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 os identificadores de mensagem e de correlação (*MsgId* e *CorrelId*) no descritor de mensagens para especificar uma mensagem específica, o gerenciador de filas deverá procurar na fila até localizar a 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 forem de comprimentos diferentes, para determinar o tamanho de uma mensagem, seu aplicativo pode usar a chamada MQGET com o campo *BufferLength* configurado como zero para zero de forma que, mesmo se a chamada falhar, ele retornará 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. O IBM 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. Pode existir uma condição de armazenamento séria` é possível que você esteja preenchendo o espaço associado aos gerenciadores de filas 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 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 47
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 Considerações de design e desempenho de aplicativos z/OS

O design do aplicativo é um dos fatores mais importantes que afetam o desempenho. Use este tópico para entender alguns dos fatores de design envolvidos no desempenho.

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 de programas que fazem chamadas MQI são demonstrados nas seções a seguir.

Para obter mais informações sobre design do aplicativo, consulte "Considerações de design para aplicativos IBM MQ" na página 47.

Efeito do comprimento da mensagem

Embora o IBM MQ for z/OS 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, envie somente os dados essenciais em uma mensagem. Por exemplo, em uma solicitação de débito de uma conta bancária, as únicas informações que podem precisar ser passadas do cliente para o aplicativo do servido são o número da conta e o valor do débito.

Efeito de persistência de mensagem

Mensagens persistentes são 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.

Dados para mensagens persistentes são gravados em buffers de log. Esses buffers são gravados nos conjuntos de dados de log quando:

- Ocorre uma confirmação
- Uma mensagem é obtida ou colocada fora do ponto de sincronização
- Buffers WRTHRSH estão cheios

O processamento muitas mensagens em uma unidade de trabalho pode causar menos entrada/saída do que se as mensagens fossem processadas uma para cada unidade de trabalho ou fora do ponto de sincronização.

Procurando uma mensagem específica

A chamada MQGET geralmente recupera a primeira mensagem de uma fila. Se você usar os identificadores de mensagem e de correlação (**MsgId** e **CorrelId**) no descritor de mensagens para especificar uma mensagem específica, o gerenciador de filas procurará na fila até localizar a mensagem. Usando MQGET dessa forma afeta o desempenho de seu aplicativo, pois, para localizar uma mensagem específica, o IBM MQ pode precisar verificar a fila inteira.

É possível usar o atributo de fila **IndexType** para especificar que deseja que o gerenciador de filas mantenha um índice que possa ser usado para aumentar a velocidade de operações MQGET na fila. No entanto, há uma pequena redução de desempenho para a manutenção de um índice, portanto, gere um somente se precisar usá-lo. É possível optar por construir um índice de identificadores de mensagem ou de identificadores de correlação ou optar por não construir um índice para filas em que as mensagens são recuperadas sequencialmente. Tente ter muitos valores de chaves diferentes, não muitas com o mesmo valor. Por exemplo, Balance1, Balance2 e Balance3, não três com Balance. Para filas compartilhadas, deve-se ter o **IndexType** correto. Para obter detalhes do atributo de fila **IndexType**, consulte IndexType.

Para evitar afetar o tempo de reinicialização do gerenciador de filas usando filas indexadas, use o parâmetro QINDXBLD(NOWAIT) na macro CSQ6SYSP. Isso permite que o reinício do gerenciador de filas seja concluído sem precisar esperar a conclusão da construção do índice de fila.

Para obter uma descrição completa do atributo **IndexType** e outros atributos de objetos, consulte Atributos de objetos.

Filas que contêm mensagens de comprimentos diferentes

Obtenha uma mensagem usando um tamanho de buffer que corresponda ao tamanho esperado da mensagem. Se você receber o código de retorno indicando que a mensagem é muito longa, obtenha um buffer maior. Quando a obtenção falhar desta maneira, o comprimento de dados retornado é o tamanho dos dados da mensagem não convertidos. Se você especificar MQGMO_CONVERT na chamada MQGET e os dados se expandirem durante a conversão, eles ainda podem não se ajustar no buffer, neste caso, será necessário aumentar ainda mais o tamanho do buffer.

Se você emitir o MQGET com um comprimento de buffer igual a zero, ele retorna o tamanho da mensagem e o aplicativo poderá, então, obter um buffer desse tamanho e emitir o get novamente. Se você tiver vários aplicativos processando a fila, outro aplicativo pode já ter processado a mensagem quando o aplicativo original tiver emitido get novamente. Se, ocasionalmente, você tiver mensagens grandes, poderá precisar obter um buffer grande apenas para essas mensagens e liberá-lo após a mensagem tiver sido processada. Isso deve ajudar a reduzir problemas de armazenamento virtual se todos os aplicativos tiverem buffers grandes.

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 **MaxMsgL** da fila. Esse método poderia usar grandes quantidades de armazenamento, no entanto, o valor de **MaxMsgL** pode ter até 100 MB, o máximo permitido por IBM MQ for z/OS.

Nota: É possível reduzir o parâmetro **MaxMsgL** após mensagens grandes terem sido colocadas na fila. Por exemplo, é possível colocar uma mensagem 100 MB e, em seguida, configurar **MaxMsgL** para 50 bytes. Isso significa que ainda é possível obter mensagens maiores do que o aplicativo esperava.

Frequência de pontos de sincronização

Os programas que emitem muitas 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. Isso tem implicações em termos de armazenamento e em termos de encadeamentos ligados a tarefas que estão tentando obter mensagens.

Como regra, se tiver vários aplicativos processando uma fila, geralmente terá o melhor desempenho quando tiver

- 100 mensagens curtas (menores que 1 KB) ou
- Uma mensagem para mensagens maiores (100 KB)

para cada ponto sincronização. Se houver somente um aplicativo processando a fila, deve-se ter mais mensagens para cada unidade de trabalho.

É possível limitar o número de mensagens que uma tarefa pode obter ou colocar em uma única unidade de recuperação com o atributo do gerenciador de filas **MAXUMSGS**. Para obter informações sobre esse atributo, consulte o comando **ALTER QMGR** em Comandos de Script (MQSC).

Vantagens 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.

Quantas mensagens um gerenciador de filas pode conter

Filas locais

O número de mensagens locais que um gerenciador de filas pode reter é basicamente o tamanho dos conjuntos de páginas. É possível ter até 100 conjuntos de páginas (embora seja recomendável que o conjunto de páginas 0 e o conjunto de páginas 1 sejam para objetos e filas relacionados ao sistema). É possível usar um conjunto de páginas com formato estendido e aumentar a capacidade de um conjunto de páginas.

Filas Compartilhadas

A capacidade para filas compartilhadas depende do tamanho do recurso de acoplamento (CF). O IBM MQ usa estruturas de lista de CF em que unidades de armazenamento fundamentais são entradas e elementos. Cada mensagem é armazenada como uma entrada e vários elementos contendo o MQMD associado e outros dados de mensagem. O número de elementos consumidos por uma única mensagem depende do tamanho da mensagem e, para CFLEVEL(5), das regras de transferência em vigor no momento de MQPUT. Menos elementos são necessários quando os dados da mensagem são transferidos para o Db2 ou SMDS. Acesso a dados da mensagem é mais lento quando a mensagem tiver sido transferida. Consulte Performance Supportpac MP1H para comparação adicional de desempenho e sobrecarga de CPU associados à transferência de mensagens.

O que afeta o desempenho

Desempenho pode significar com que rapidez as mensagens podem ser processadas e também pode significar quanto da CPU é necessário por mensagem.

O que afeta a rapidez com que mensagens podem ser processadas

Para mensagens persistentes, o maior impacto é a velocidade dos conjuntos de dados de log. A velocidade dos conjuntos de dados de log depende do DASD em que se encontram. Portanto, deve-se tomar cuidado para colocar o conjunto de dados do log em volumes de baixo uso para reduzir a contenção. O striping dos logs do MQ melhora o desempenho do log quando há várias páginas gravadas por E/S. O Z High Performance Fibre connection (zHPF) também tem um desempenho significativo para o tempo de resposta de E/S quando o subsistema de E/S está ocupado.

Quando houver uma solicitação para obter e colocar uma mensagem, o acesso à fila é bloqueado durante a solicitação para preservar a integridade da fila. Para propósitos de planejamento, considere a fila bloqueada por toda a solicitação. Portanto, se o tempo de um put for 100 microssegundos

e houver mais de 10.000 solicitações por segundo, poderá observar atrasos. Você pode alcançar melhor que isso na prática, mas é uma boa regra geral. É possível usar filas diferentes para melhorar o desempenho.

Possíveis razões para isso podem ser:

- usar uma fila de resposta comum que toda transação do CICS usa
- cada transação do CICS recebe uma resposta exclusiva para colocar na fila
- uma resposta para uma fila para região do CICS e todas as transações na região do CICS usam esta fila.

A resposta depende do número de solicitações por segundo e o tempo de resposta das solicitações.

Se as mensagens precisarem ser lidas a partir de um conjunto de páginas, elas serão mais lentas em comparação a quando as mensagens estão no buffer pool. Se você tiver mais mensagens do que cabem em um buffer pool, então, elas serão derramadas para o disco. Portanto, é necessário assegurar que o buffer pool seja grande o suficiente para as suas mensagens de curta duração. Se houver mensagens que você processará muitas horas mais tarde, elas provavelmente serão derramadas para o disco, portanto, deve-se esperar que um get para essas mensagens seja mais lento do que se estivessem no buffer pool.

Para uma fila compartilhada, a velocidade das mensagens depende da velocidade do Recurso de acoplamento. Um CF dentro do processador físico provavelmente será mais rápido do que um CF externo. O tempo de resposta do CF depende de se o CF está muito ocupado. Por exemplo, nos sistemas Hursley, quando o CF estava 17% ocupado, o tempo de resposta era de 14 microssegundos. Quando o CF estava 95% ocupado, o tempo de resposta era de 45 microssegundos.

Se suas solicitações do MQ usam muita CPU, isso pode afetar a rapidez com que as mensagens são processadas. Pois, se a Partição lógica (LPAR) for restrita para a CPU, os aplicativos serão atrasados esperando a CPU.

Quanto da CPU por mensagem

Em geral, mensagens maiores usam mais CPU, portanto, evite mensagens grandes (x MB), se possível.

Ao obter mensagens específicas das filas, a fila deve ser indexada de forma que o gerenciador de filas possa ir diretamente para a mensagem (e assim evitar potencialmente uma varredura completa da fila). Se a fila não for indexada, então, a fila será verificada desde o início procurando a mensagem. Se houver 1000 mensagens na fila, pode ser necessário verificar todas as 1000 mensagens. O resultado é muito uso desnecessário da CPU.

Canais que usam TLS têm um custo adicional devido à criptografia da mensagem.

No MQ V7, é possível selecionar mensagens por uma sequência de seletor, além de **CORRELID** ou **MSGID**. Toda mensagem precisa ser verificada, portanto, se houver muitas mensagens na fila, isso custa caro.

É mais eficiente para um aplicativo executar OPEN PUT PUT CLOSE do que PUT1 PUT1.

Acionamento no CICS

Quando a taxa de chegada de mensagens para uma fila acionada for baixa, é eficiente usar o acionador primeiro. Quando a taxa de chegada de mensagens for mais de 10 mensagens por segundo, é mais eficiente acionar a primeira transação, em seguida, fazer a transação processar uma mensagem e obter a próxima mensagem e assim por diante. Se uma mensagem não chegou em um período curto (digamos entre 0,1 e 1 segundo), a transação será finalizada. Em alto rendimento, poderá precisar de várias transações em execução para processar as mensagens e para evitar um acúmulo de mensagens. Para cada mensagem do acionador produzida, isso requer um put e um get de uma mensagem do acionador, que, efetivamente, duplica o custo da mensagem.

Quantas conexões ou usuários simultâneos são suportados

Cada conexão usa o armazenamento virtual dentro do gerenciador de filas, portanto, quanto mais usuários simultâneos mais armazenamento usado. Se precisar de um buffer pool muito grandes e de

um grande número de usuários, então, você poderá ser limitado para armazenamento virtual e talvez seja necessário reduzir o tamanho de seus buffer pools.

Se segurança estiver sendo usada, o gerenciador de filas armazena informações em cache no gerenciador de filas por um longo período. A quantia de armazenamento virtual usada no gerenciador de filas é afetada.

O **CHINIT** pode suportar até aproximadamente 10.000 conexões. Isso é limitado pelo armazenamento virtual. Se uma conexão usar mais armazenamento, por exemplo, usando por TLS, o armazenamento por conexão aumentará, o que, portanto, significa que o **CHINIT** pode suportar menos conexões. Se você estiver processando mensagens grandes, elas precisarão de mais armazenamento para buffers no **CHINIT**, portanto, o **CHINIT** pode suportar menos mensagens.

As conexões com um gerenciador de filas remotas são mais eficientes do que as conexões de cliente. Por exemplo, toda solicitação do cliente MQ requer dois fluxos de rede (um para a solicitação e um para a resposta). Com um canal para um gerenciador de filas remotas, pode haver 50 envios pela rede antes que uma resposta retorne. Se estiver considerando uma rede de cliente grande, pode ser mais eficiente usar um gerenciador de filas concentrador em uma caixa distribuída e ter um canal de entrada e saída do concentrador.

Outras coisas que afetam o desempenho

O conjunto de dados do log deve ter um tamanho de pelo menos 1.000 cilindros. Se os logs forem menores que isso, a atividade de ponto de verificação pode estar muito frequente. Em um sistema ocupado, um ponto de verificação geralmente deve ser a cada 15 minutos ou mais, em rendimentos muito altos, pode ser menor que isso. Quando um ponto de verificação ocorre, os buffer pools são verificados a as mensagens 'antigas' e as páginas mudadas são gravadas no disco. Se os pontos de verificação forem muito frequentes, isso pode afetar o desempenho. O valor de LOGLOAD também pode afetar a frequência do ponto de verificação. Se o gerenciador de filas for finalizado de forma anormal, então, na reinicialização poderá precisar fazer a leitura de três pontos de verificação para trás. O melhor intervalo de ponto de verificação é um equilíbrio entre a atividade quando um ponto de verificação é feito e a quantia de dados de log que pode precisar ser lida quando o gerenciador de filas for reiniciado.

Há uma sobrecarga significativa incorrida ao iniciar um canal. Geralmente, é melhor iniciar um canal e deixá-lo conectado, em vez de iniciar e parar o canal frequentemente.

Informações relacionadas

MP1H: IBM MQ for z/OS 9.0 Relatório de Desempenho

z/OS Os aplicativos IMS e IMS bridge no IBM MQ for z/OS

Essas informações ajudam a gravar aplicativos IMS usando IBM MQ.

- Para usar pontos de sincronização e as chamadas MQI em aplicativos IMS, consulte "Gravando aplicativos do IMS usando o IBM MQ" na página 63.
- Para escrever aplicativos que usam a ponte IBM MQ - IMS, consulte "Escrevendo aplicativos de ponte IMS" na página 67.

Use os links a seguir para descobrir mais sobre os aplicativos IMS e de ponte IMS no IBM MQ for z/OS:

- "Gravando aplicativos do IMS usando o IBM MQ" na página 63
- "Escrevendo aplicativos de ponte IMS" na página 67

Conceitos relacionados

"Visão geral da Message Queue Interface" na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

"Conectando-se e desconectando-se de um gerenciador de filas" na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

Gravando aplicativos do IMS usando o IBM MQ

Há considerações adicionais ao usar o IBM MQ em aplicativos do IMS. Estas incluem chamadas API do MQ que podem ser usadas e o mecanismo usado para o ponto de sincronização.

Use os seguintes links para descobrir mais sobre a gravação de aplicativos do IMS no IBM MQ for z/OS:

- “Pontos de sincronização em aplicativos IMS” na página 63
- “Chamadas MQI em aplicativos IMS” na página 64

Restrições

Há restrições sobre quais chamadas API do IBM MQ podem ser usadas por um aplicativo que usa o adaptador do IMS.

As seguintes chamadas API do IBM MQ não são suportadas em um aplicativo que usa o adaptador do IMS:

- MQCB
- MQCB_FUNCTION
- MQCTL

Conceitos relacionados

“Escrevendo aplicativos de ponte IMS” na página 67
Este tópico contém informações sobre como escrever aplicativos para usar a ponte IBM MQ - IMS.

Pontos de sincronização em aplicativos IMS

Em um aplicativo IMS, você estabelece um ponto de sincronização usando chamadas do IMS, como GU (get unique) para o IOPCB e o CHKP (checkpoint).

Para voltar todas as mudanças desde o ponto de verificação anterior, é possível usar a chamada IMS ROLB (rollback). Para obter informações adicionais, consulte a seguinte documentação:

- IMS 13 Programação de Aplicativos APG SC19-3646
- IMS 13 APIs de Programação de Aplicativos APR SC19-3647

O gerenciador de filas é um participante de um protocolo two-phase commit; o gerenciador de ponto de sincronização do IMS é o coordenador.

Todos os identificadores abertos são fechados pelo adaptador do IMS em um ponto de sincronização (exceto em um ambiente BMP de lote ou não acionado por mensagem). Isso ocorre porque um usuário diferente poderia iniciar a próxima unidade de trabalho e a verificação de segurança do IBM MQ é executada quando as chamadas MQCONN, MQCONNX e MQOPEN são feitas, não quando as chamadas MQPUT ou MQGET são feitas.

No entanto, em uma ambiente Wait-for-Input (WFI) ou pseudo Wait-for-Input (PWFI), o IMS não notifica o IBM MQ para fechar os identificadores até a próxima mensagem chegar ou um código de status QC ser retornado ao aplicativo. Se o aplicativo estiver esperando na região do IMS e qualquer um desses identificadores pertencer a filas acionadas, o acionamento não ocorrerá porque as filas estão abertas. Por essa razão, os aplicativos em execução em um ambiente WFI ou PWFI devem explicitamente MQCLOSE os identificadores de fila antes de executar GU no IOPCB para a próxima mensagem.

Se um aplicativo IMS (um BMP ou um MPP) emitir a chamada MQDISC, as filas abertas serão fechadas mas nenhum ponto de sincronização implícito será tomado. Se o aplicativo finalizar normalmente, quaisquer filas abertas serão fechadas e uma confirmação implícita ocorrerá. Se o aplicativo finalizar de forma anormal, quaisquer filas abertas serão fechadas e uma restauração implícita ocorrerá.

Chamadas MQI em aplicativos IMS

Use estas informações para aprender sobre o uso de chamadas MQI em aplicativos Server e Enquiry.

Esta seção abrange o uso de chamadas de MQI nos tipos de aplicativos IMS a seguir:

- “Aplicativos do servidor” na página 64
- “Aplicativos de consulta” na página 66

Aplicativos do servidor

Aqui está um esboço do modelo de servidor de aplicativos MQI:

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

O programa de amostra CSQ4ICB3 mostra a implementação, em C/370, de um BMP usando este modelo. O programa estabelece comunicação com o IMS primeiro e, em seguida, com o 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
```

A inicialização do IMS determina se o programa foi chamado como um BMP direcionado por mensagens ou orientado por lote e controla as manipulações de fila e conexão do gerenciador de filas do IBM MQ adequadamente:

```
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
```

A inicialização do IBM MQ se conecta ao gerenciador de filas e abre as filas. Em um BMP orientado a mensagens, isso é chamado após cada ponto de sincronização do IMS ser obtido; em um BMP orientado para lote, isso é chamado apenas durante a inicialização do programa:

```
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
```

A implementação do modelo de servidor em um MPP é influenciada pelo fato de que o MPP processa uma única unidade de trabalho por chamada. Isso ocorre porque, quando um ponto de sincronização (GU) é obtido, a conexão e as manipulações de filas estão fechadas e a próxima mensagem do IMS é entregue. Essa limitação pode ser superada parcialmente por um dos seguintes itens:

- **Processamento de muitas mensagens em uma única unidade de trabalho**

Isso envolve:

– Lendo uma mensagem

– Processar as atualizações necessárias

– Colocar a resposta

em um loop até que todas as mensagens tenham sido processadas ou até que um número máximo configurado de mensagens tenha sido processado, no momento em que um ponto de sincronização é obtido.

Apenas determinados tipos de aplicativo (por exemplo, uma atualização simples do banco de dados ou consulta) podem ser abordados dessa forma. Embora as mensagens de resposta de MQI possam ser colocadas com a autoridade do originador da mensagem MQI que está sendo tratada, as implicações de segurança de quaisquer atualizações de recurso do IMS precisam ser tratadas com cuidado.

- **Processamento de uma mensagem por chamada do MPP e assegurar o planejamento múltiplo de MPP para processar todas as mensagens disponíveis.**

  Use o IBM MQ IMS trigger monitor program (CSQQTRMN) para planejar a transação de MPP quando houver mensagens na fila do IBM MQ e nenhum aplicativo atendendo.

  Se o monitor acionador iniciar o MPP, o nome do gerenciador de filas e o nome da fila serão transmitidos para o programa, conforme mostrado no trecho de código COBOL a seguir:

```
* 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 '='.
```

O modelo do servidor, que espera-se ser uma tarefa de longa execução, é melhor suportado em uma região de processamento de lote, embora o BMP não possa ser acionado usando CSQQTRMN.

Aplicativos de consulta

Um aplicativo IBM MQ típico que inicia uma consulta ou atualização funciona da seguinte forma:

- Reúna dados do usuário
- Coloque uma ou mais mensagens do IBM MQ
- Obtenha as mensagens de resposta (você pode ter que esperar por elas)
- Forneça uma resposta ao usuário

Como as mensagens colocadas nas filas do IBM MQ não são disponibilizadas a outros aplicativos IBM MQ até serem confirmadas, elas devem ser colocadas fora do ponto de sincronização ou o aplicativo IMS deve ser dividido em duas transações.

Se a consulta envolver a colocação de uma única mensagem, é possível usar a opção *sem ponto de sincronização*; no entanto, se a consulta for mais complexa ou atualizações de recursos estiverem

envolvidas, você poderá ter problemas de consistência, se a falha ocorrer e você não usar ponto de sincronização.

Para resolver isso, é possível dividir transações do IMS MPP usando chamadas MQI com um comutador de mensagem de programa para programa; consulte *IMS/ESA Programação de aplicativo: comunicação de dados* para obter informações sobre isso. Isso permite que um programa de consulta seja implementado em um 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
```

Escrevendo aplicativos de ponte IMS

Este tópico contém informações sobre como escrever aplicativos para usar a ponte IBM MQ - IMS.

Para obter informações sobre a ponte IBM MQ - IMS, consulte A ponte IMS.

Use os links a seguir para descobrir mais sobre como escrever aplicativos de ponte IMS no IBM MQ for z/OS:

- “Como a ponte IMS lida com mensagens” na página 67
- “Escrevendo programas de transação do IMS por meio do IBM MQ” na página 910

Conceitos relacionados

“Gravando aplicativos do IMS usando o IBM MQ” na página 63
Há considerações adicionais ao usar o IBM MQ em aplicativos do IMS. Estas incluem chamadas API do MQ que podem ser usadas e o mecanismo usado para o ponto de sincronização.

Como a ponte IMS lida com mensagens

Ao usar a ponte IBM MQ - IMS para enviar mensagens para um aplicativo IMS, é necessário construir suas mensagens em um formato especial.

Deve-se colocar suas mensagens também nas filas do IBM MQ que foram definidas com uma classe de armazenamento que especifica o grupo XCF e o nome do membro do sistema IMS de destino. Elas são conhecidas como filas de ponte MQ-IMS ou simplesmente filas de **ponte**.

A ponte do IBM MQ-IMS vai requerer acesso de entrada exclusivo (MQOO_INPUT_EXCLUSIVE) para a fila de pontes se ela for definida com QSGDISP(QMGR) ou se for definida com QSGDISP(SHARED) juntamente com a opção NOSHARE.

Um usuário não precisa se conectar ao IMS antes de enviar mensagens para um aplicativo IMS. O ID do usuário no campo *UserIdentifier* da estrutura MQMD é usado para verificação de segurança. O nível de verificação é determinado quando o IBM MQ se conecta ao IMS e está descrito em Controle de acesso de aplicativo para a ponte IMS. Isso permite que uma pseudo conexão seja implementada.

A ponte IBM MQ - IMS aceita os tipos de mensagem a seguir:

- Mensagens contendo os dados de transação do IMS e uma estrutura MQIIH (descrita em MQIIH):

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

Nota:

1. Os colchetes, [ ], representam diversos segmentos opcionais.
2. Configure o campo *Format* da estrutura MQMD para MQFMT_IMS para usar a estrutura MQIIH.

- Mensagens contendo dados de transação do IMS, mas nenhuma estrutura MQIIH:

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

O IBM MQ valida os dados da mensagem para assegurar que a soma dos bytes LL mais o comprimento de MQIIH (se estiver presente) seja igual ao comprimento da mensagem.

Quando a ponte IBM MQ - IMS obtém mensagens das filas de ponte, ela processa as mesmas da seguinte forma:

- Se a mensagem contiver uma estrutura MQIIH, a ponte verifica MQIIH (consulte MQIIH), constrói os cabeçalhos OTMA e envia a mensagem para o IMS. O código de transação for especificado na mensagem de entrada. Se for um LTERM, o IMS responde com uma mensagem DFS1288E. Se o código de transação representar um comando, o IMS executa o comando; caso contrário, a mensagem será enfileirada no IMS para a transação.
- Se a mensagem contiver dados de transação do IMS, mas nenhuma estrutura MQIIH, a ponte IMS faz as suposições a seguir:
  - O código de transação está nos bytes 5 a 12 dos dados do usuário
  - A transação está em modo de não conversação
  - A transação está em modo de confirmação 0 (commit-then-send)
  - O *Format* no MQMD é usado como o *MFSMapName* (na entrada)
  - O modo de segurança é MQISS_CHECK

  A mensagem de resposta também é construída sem uma estrutura MQIIH, usando o *Format* para o MQMD do *MFSMapName* da saída do IMS.

A ponte IBM MQ - IMS usa um ou dois Tpipes para cada fila do IBM MQ:

- Um Tpipe sincronizado é usado para todas as mensagens usando o modo de Confirmação 0 (COMMIT_THEN_SEND) (elas são mostradas com SYN no campo de status do comando IMS /DIS TMEMBER client TPIPE xxxx)
- Um Tpipe não sincronizado é usado para todas as mensagens que estiverem usando o modo de Confirmação 1 (SEND_THEN_COMMIT)

Os Tpipes são criados pelo IBM MQ quando são usados pela primeira vez. Um Tpipe não sincronizado existe até que o IMS seja reiniciado. Tpipes sincronizados existem até o IMS passar por cold start. Não é possível excluir esses Tpipes.

Consulte os tópicos a seguir para obter informações adicionais sobre como a ponte IBM MQ - IMS lida com mensagens:

- “Mapeando mensagens do IBM MQ para tipos de transação do IMS” na página 69
- “Se a mensagem não puder ser colocada na fila do IMS” na página 69
- “Códigos de feedback de ponte IMS” na página 70
- “Os campos de MQMD em mensagens da ponte IMS” na página 70
- “O campos de MQIIH em mensagens da ponte IMS” na página 71
- “Mensagens de resposta a partir do IMS” na página 72
- “Usando PCBs de resposta alternativos em transações do IMS” na página 72

- “Enviando mensagens não solicitadas a partir do IMS” na página 73
- “Segmentação de mensagem” na página 73
- “Conversão de Dados” na página 73

Conceitos relacionados

“Escrevendo programas de transação do IMS por meio do IBM MQ” na página 910
A codificação necessária para manipular transações do IMS por meio do IBM MQ depende do formato de mensagem necessário para a transação do IMS e do intervalo de respostas que pode retornar. No entanto, há vários pontos a considerar quando seu aplicativo manipula as informações de formatação de tela do IMS.

Mapeando mensagens do IBM MQ para tipos de transação do IMS
Uma tabela que descreve o mapeamento das mensagens do IBM MQ para tipos de transação do IMS.

Tabela 4. Mapeando mensagens do IBM MQ para tipos de transação do IMS

tipo de mensagem doIBM MQ	**Commit-then-send (modo 0) - usa Tpipes do IMS sincronizados**	**Send-then-commit (modo 1) – usa Tpipes do IMS não sincronizados**
Mensagens persistentes do IBM MQ	• Transações de função integral recuperáveis • Transações irrecuperáveis são rejeitadas pelo IMS	• Transações Fastpath • Transações conversacionais • Transações de função integral
Mensagens do IBM MQ não persistentes	• Transações de função integral irrecuperáveis • As transações recuperáveis são permitidas com o IMS V8, o APAR PQ61404 e todas as versões mais recentes do IMS	• Transações Fastpath • Transações conversacionais • Transações de função integral

Nota: Comandos IMS não podem usar mensagens IBM MQ persistentes com o modo confirmar 0. Consulte o *IMS/ESA Guia do usuário de Abrir o Acesso ao Gerenciador de Transações* para obter mais informações.

Se a mensagem não puder ser colocada na fila do IMS
Aprenda sobre ações a serem executadas se a mensagem não puder ser colocada na fila do IMS.

Se a mensagem não puder ser colocada na fila do IMS, a ação a seguir será executada pelo IBM MQ:

- Se uma mensagem não puder ser colocada no IMS porque a mensagem é inválida, a mensagem será colocada na fila de mensagens não entregues e uma mensagem será enviada ao console do sistema.
- Se a mensagem for válida, mas for rejeitada pelo IMS, o IBM MQ enviará uma mensagem de erro ao console do sistema, a mensagem inclui o sense code do IMS e a mensagem do IBM MQ é colocada na fila de mensagens não entregues. Se o sense code do IMS for 001A, o IMS envia uma mensagem do IBM MQ que contém a razão da falha para a fila de resposta.

  Nota: Nas circunstâncias listadas anteriormente, se o IBM MQ não puder colocar a mensagem na fila de mensagens não entregues, a mensagem será retornada para a fila de origem do IBM MQ. Uma mensagem de erro é enviada ao console do sistema e nenhuma mensagem adicional é enviada a partir dessa fila.

  Para reenviar as mensagens, execute ***um*** dos seguintes:

  – Pare e reinicie os Tpipes no IMS que correspondem à fila
  – Mude a fila para GET(DISABLED) e novamente para GET(ENABLED)
  – Pare e reinicie o IMS ou o OTMA
  – Pare e reinicie o subsistema IBM MQ

- Se a mensagem for rejeitada pelo IMS para algo diferente de um erro da mensagem, a mensagem do IBM MQ é retornada à fila de origem, o IBM MQ para o processamento da fila e uma mensagem de erro é enviada ao console do sistema.

  Se uma mensagem de relatório de exceções for necessária, a ponte a coloca na fila de resposta com a autoridade do originador. Se a mensagem não puder ser colocada na fila, a mensagem de relatório será colocada na fila de mensagens não entregues com a autoridade da ponte. Se ela não puder ser colocada na DLQ, ela será descartada.

Códigos de feedback de ponte IMS
Os sense codes do IMS são geralmente saída no formato hexadecimal em mensagens do console do IBM MQ, como CSQ2001I (por exemplo, sense code 0x001F). Os códigos de feedback do IBM MQ, conforme visto no cabeçalho de mensagens não entregues colocadas em fila de mensagens não entregues, são números decimais.

Os códigos de feedback de ponte IMS estão no intervalo de 301 a 399 ou de 600 a 855 para sense code 0x001A NACK. Eles são mapeados dos sense codes IMS-OTMA da seguinte forma:

1. O sense code IMS-OTMA é convertido de um número hexadecimal para um número decimal.
2. 300 é incluído ao número resultante do cálculo em 1, fornecendo o código do IBM MQ *Feedback*.
3. O sense code IMS-OTMA 0x001A, decimal 26 é um caso especial. Um código *Feedback* no intervalo de 600 a 855 é gerado.
   a. O código de razão IMS-OTMA é convertido de um número hexadecimal para um número decimal.
   b. 600 é incluído ao número resultante do cálculo em a, fornecendo o código do IBM MQ *Feedback*.

Para obter informações sobre sense codes IMS-OTMA, consulte Sense codes OTMA para mensagens NAK.

Os campos de MQMD em mensagens da ponte IMS
Conheça os campos MQMD em mensagens a partir da ponte IMS.

O MQMD da mensagem de origem é executado por IMS na seção Dados do Usuário dos cabeçalhos OTMA. Se a mensagem se originar no IMS, isso será construído pela Saída de Resolução de Destino do IMS. O MQMD de uma mensagem recebida do IMS é construído da seguinte maneira:

StrucID
" " MD

Versão
MQMD_VERSION_1

Relatório
MQRO_NONE

MsgType
MQMT_REPLY

Expiração
Se MQIIH_PASS_EXPIRATION é definido no campo Sinalizadores da MQIIH, este campo contém o tempo de expiração restante, caso contrário ele é configurado como MQEI_UNLIMITED

Feedback
MQFB_NONE

Codificação
MQENC.Native (a codificação do sistema z/OS)

CodedCharSetId
MQCCSI_Q_MGR (o CodedCharSetID do sistema z/OS)

Formato
MQFMT_IMS se o MQMD.Format da mensagem de entrada é MQFMT_IMS, caso contrário, IOPCB.MODNAME

Priority
MQMD.Priority da mensagem de entrada

Persistence
Depende no modo de confirmação: MQMD.Persistence da mensagem de entrada se a persistência de CM-1; corresponder à capacidade de recuperação do IMS se CM-0

MsgId
MQMD.MsgId se MQRO_PASS_MSG_ID, caso contrário, Nova MsgId (o padrão)

ID de Correlação
MQMD.CorrelId da mensagem de entrada se MQRO_PASS_CORREL_ID, caso contrário, MQMD.MsgId da mensagem de entrada (o padrão)

BackoutCount
0

ReplyToQ
Espaços em branco

ReplyToQMgr
Espaços (configurado como nome de qmgr local pelo gerenciador de filas durante o MQPUT)

UserIdentifier
MQMD.UserIdentifier da mensagem de entrada

AccountingToken
MQMD.AccountingToken da mensagem de entrada

ApplIdentityData
MQMD.ApplIdentityData da mensagem de entrada

PutApplType
MQAT_XCF se nenhum erro, caso contrário, MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> se nenhum erro, caso contrário, nome do QMGR

PutDate
Data quando a mensagem foi colocada

PutTime
Hora quando a mensagem foi colocada

ApplOriginData
Espaços em branco

O campos de MQIIH em mensagens da ponte IMS
Aprenda sobre os campos MQIIH em mensagens da ponte IMS.

O MQIIH de uma mensagem recebida do IMS é construído como segue:

StrucId
"IIH "

Versão
1

StrucLength
84

Codificação
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Formato
MQIIH.ReplyToFormat da mensagem de entrada, se MQIIH.ReplyToFormat não estiver em branco, caso contrário, IOPCB.MODNAME

Sinalizadores
0

LTermOverride
nome de LTERM (de Tpipe) do cabeçalho do OTMA

MFSMapName
nome do Mapa do cabeçalho OTMA

ReplyToFormat
Espaços em branco

Autenticador
MQIIH.Authenticator da mensagem de entrada, se a mensagem de resposta está sendo colocada em uma fila de ponte MQ-IMS, caso contrário, em branco.

TranInstanceId
ID da conversa/Token do servidor do cabeçalho do OTMA, se ele estiver na conversa. Nas versões do IMS anteriores à V14, esse campo sempre seria nulo se não estivesse na conversa. Do IMS V14 em diante, esse campo pode ser configurado pelo IMS, mesmo que não esteja na conversa.

TranState
"C" se na conversa, caso contrário, em branco

CommitMode
modo de Commit a partir do cabeçalho OTMA ("0" ou "1")

SecurityScope
Em branco

Reservado
Em branco

Mensagens de resposta a partir do IMS
Quando uma transação do IMS ISRTs para seu IOPCB, a mensagem é roteada de volta para o LTERM ou o TPIPE de origem.

Estas são vistas no IBM MQ como mensagens de resposta. Mensagens de resposta do IMS são colocadas na fila de resposta especificada na mensagem original. Se a mensagem não puder ser colocada na fila de resposta, ela será colocada na fila de mensagens não entregues usando a autoridade da ponte. Se a mensagem não puder ser colocada na fila de mensagens não entregues, um reconhecimento negativo será enviado para IMS para dizer que a mensagem não pode ser recebida. A responsabilidade pela mensagem é então retornada ao IMS. Se estiver usando o modo de confirmação 0, mensagens desse Tpipe não serão enviadas à ponte e permanecerão na fila IMS; ou seja, nenhuma mensagem adicional será enviada até a reinicialização. Se estiver sendo usado o modo de confirmação 1, outros trabalhos poderão continuar.

Se a resposta tiver uma estrutura MQIIH, seu tipo de formato será MQFMT_IMS; se não, seu tipo de formato será especificado pelo nome do MOD IMS usado ao inserir a mensagem.

Usando PCBs de resposta alternativos em transações do IMS
Quando uma transação IMS usa PCBs de resposta alternativa (ISRTs para o ALTPCB ou emite uma chamada CHNG para um PCB modificável), a saída de pré-roteamento (DFSYPRX0) é chamada para determinar se a mensagem deve ser redirecionada.

Se a mensagem tiver que ser roteada novamente, a saída de resolução de destino (DFSYDRU0) será chamada para confirmar o destino e preparar as informações de cabeçalho. Consulte Usando saídas do OTMA no IMS e A saída de pré-roteamento DFSYPRX0 para obter informações sobre esses programas de saída.

A menos que seja tomada ação nas saídas, toda saída de transações do IMS iniciada a partir de um gerenciador de filas do IBM MQ, seja para o IOPCB ou para um ALTPCB, será retornada ao mesmo gerenciador de filas.

Enviando mensagens não solicitadas a partir do IMS
Para enviar mensagens do IMS para uma fila IBM MQ, é necessário chamar uma transação IMS que ISRTs para um ALTPCB.

É necessário escrever saídas de gravar pré-roteamento e de resolução de destino para rotear mensagens não solicitadas a partir do IMS e construir os dados do usuário do OTMA, de modo que o MQMD da mensagem possa ser construído corretamente. Consulte O saída de pré-roteamento DFSYPRX0 e A saída do usuário de resolução de destino para obter informações sobre esses programas de saída.

Nota: A ponte IBM MQ - IMS não sabe se uma mensagem que recebe é uma resposta ou uma mensagem não solicitada. Ela manipula a mensagem da mesma maneira em cada caso, construindo o MQMD e o MQIIH da resposta com base no UserData do OTMA que chegou com a mensagem

Mensagens não solicitadas podem criar novos Tpipes. Por exemplo, se uma transação existente do IMS tiver mudado para um novo LTERM (por exemplo, PRINT01), mas a implementação requerer que a saída seja entregue por meio do OTMA, um novo Tpipe (chamado PRINT01, neste exemplo) será criado. Por padrão, este é um Tpipe não sincronizado. Se a implementação requerer que a mensagem seja recuperável, configure a sinalização da saída de resolução de destino. Consulte o *Guia de customização do IMS* para obter mais informações.

Segmentação de mensagem
É possível definir IMS transações como esperando entradas de um único segmento ou de vários segmentos.

O aplicativo IBM MQ de origem deve construir a entrada do usuário seguindo a estrutura MQIIH como um ou mais segmentos de dados LLZZ. Todos os segmentos de uma mensagem do IMS devem estar contidos em uma única mensagem do IBM MQ enviada com um único MQPUT.

O comprimento máximo de um segmento de dados LLZZ é definido por IMS/OTMA (32767 bytes). O comprimento total da mensagem do IBM MQ é a soma dos bytes LL, mais o comprimento da estrutura MQIIH.

Todos os segmentos da resposta estão contidos em uma única mensagem do IBM MQ.

Há uma restrição adicional na limitação de 32 KB em mensagens com formato MQFMT_IMS_VAR_STRING. Quando os dados em uma mensagem CCSID de ASCII combinado são convertidos para uma mensagem CCSID de EBCDIC combinado, um byte shift-in ou shift-out é incluído toda cada vez que há uma transição entre os caracteres de SBCS e DBCS. A restrição de 32 KB se aplica ao tamanho máximo da mensagem. Ou seja, como o campo LL na mensagem não pode exceder 32 KB, a mensagem não deve exceder 32 KB incluindo todos os caracteres shift-in e shift-out. O aplicativo que está construindo a mensagem deve permitir isso.

Conversão de Dados
A conversão de dados é executada pelo recurso de enfileiramento distribuído (que pode chamar quaisquer saídas necessárias) ou pelo agente de enfileiramento intragrupo (que não suporta o uso de saídas) quando coloca uma mensagem em uma fila de destino que possui informações XCF definidas para sua classe de armazenamento. A conversão de dados não ocorre quando uma mensagem é entregue a uma fila por publicação/assinatura.

Todas as saídas necessárias devem estar disponíveis para o recurso de enfileiramento distribuído no conjunto de dados referido pela instrução CSQXLIB DD. Isso significa que é possível enviar mensagens para um aplicativo IMS usando a ponte IBM MQ - IMS de qualquer plataforma IBM MQ.

Se houver erros de conversão, a mensagem será colocada na fila não convertida; isso resulta, finalmente, no fato de ela ser tratada como um erro pela ponte IBM MQ - IMS, pois a ponte não pode reconhecer o formato de cabeçalho. Se ocorrer um erro de conversão, uma mensagem de erro será enviada ao console do z/OS.

Consulte “Escrevendo saídas de conversão de dados” na página 988 para obter informações detalhadas sobre a conversão de dados em geral.

Enviando mensagens para a ponte IBM MQ - IMS

Para assegurar que a conversão seja executada corretamente, deve-se informar ao gerenciador de filas qual é o formato da mensagem.

Se a mensagem tiver uma estrutura MQIIH, o *Format* no MQMD deve ser configurado como o formato MQFMT_IMS integrado e o *Formato* no MQIIH deve ser configurado como o nome do formato que descreve seus dados da mensagem. Se não houver MQIIH, configure o *Format* no MQMD como seu nome de formato.

Se seus dados (diferente da LLZZs) forem todos dados de caracteres (MQCHAR), use como seu nome de formato (no MQIIH ou MQMD, conforme apropriado) o formato MQFMT_IMS_VAR_STRING integrado. Caso contrário, use seu próprio nome de formato, nesse caso deve-se também fornecer uma saída de conversão de dados para o seu formato. A saída deve identificar a conversão de LLZZs em sua mensagem, além dos dados em si (mas ela não precisa identificar nenhum MQIIH no início da mensagem).

Se seu aplicativo usar *MFSMapName*, será possível usar mensagens com o MQFMT_IMS em vez disso e definir o nome de mapa transmitido para a transação do IMS no campo MFSMapName da MQIIH.

Recebendo mensagens da ponte IBM MQ - IMS

Se uma estrutura MQIIH estiver presente na mensagem original que você está enviando ao IMS, uma também estará presente na mensagem de resposta.

Para assegurar que sua resposta seja convertida corretamente:

- Se você tiver uma estrutura MQIIH em sua mensagem original, especifique o formato desejado para sua mensagem de resposta no campo *ReplytoFormat* de MQIIH da mensagem original. Este valor é colocado no campo *Format* de MQIIH da mensagem de resposta. Isso será particularmente útil se todos os dados de saída estiverem no formato LLZZ<dados de caracteres>.
- Se você não tiver uma estrutura MQIIH em sua mensagem original, especifique o formato desejado para a mensagem de resposta como o nome MFS MOD no ISRT do aplicativo IMS para o IOPCB.

Desenvolvendo aplicativos JMS e Java

O IBM MQ fornece duas interfaces de idioma do Java: IBM MQ classes for Java Message Service e IBM MQ classes for Java.

No IBM MQ, há duas APIs alternativas para uso em aplicativos Java:

IBM MQ classes for JMS
IBM MQ classes for Java Message Service (JMS) é o provedor do JMS que é fornecido com o IBM MQ. 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.

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.

Nota:

Os IBM MQ classes for Java são estabilizados funcionalmente no nível enviado em IBM MQ 8.0 Para obter mais informações, veja Estabilização de classes do IBM MQ para Java.

O IBM MQ classes for Java não é suportado no IMS.

O IBM MQ classes for Java não é suportado no WebSphere Application Server 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.

Usando o IBM MQ classes for JMS

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

A especificação do JMS define um conjunto de interfaces que os aplicativos podem usar para executar operações de mensagens. A versão mais recente da especificação é JMS 2.0. O pacote javax.jms define as interfaces do JMS e um provedor do 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.

A especificação do JMS espera que os objetos ConnectionFactory e Destination sejam objetos administrados. Um administrador cria e mantém objetos administrados em um armazenador central, e um aplicativo JMS recupera esses objetos usando o Java Naming Directory Interface (JNDI). IBM MQ classes for JMS suporta o uso de objetos administrados, e um administrador pode utilizar a ferramenta de administração IBM MQ JMS ou IBM MQ Explorer para criar e manter objetos administrados.

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

Liberações anteriores do IBM MQ classes for JMS contêm extensões que são implementadas em objetos como os objetos 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. Esta liberação do IBM MQ classes for JMS mantêm estas extensões, que são agora conhecidas como extensões do IBM MQ JMS. É possível continuar usando, sem mudança, quaisquer aplicativos que usam essas extensões.

As extensões do IBM JMS

Esta liberação do IBM MQ classes for JMS fornece um conjunto mais genérico de extensões para a API do JMS, que não são específicos para IBM MQ como o sistema de mensagens. Essas extensões são conhecidas como extensões do IBM JMS, e possuem os seguintes objetivos amplos:

- Para fornecer um nível superior de consistência entre provedores IBM JMS
- Facilitar a gravação de um aplicativo de ligação entre dois sistemas de mensagens IBM
- Para facilitar o transporte de um aplicativo de um provedor IBM JMS para outro

As extensões fornecem a função que é semelhante àquela fornecida em IBM Message Service Client for C/C++ e IBM Message Service Client for .NET.

No IBM MQ 8.0, o IBM MQ classes for JMS é construído com o Java 7. O ambiente de tempo de execução do Java 7 suporta a execução das versões anteriores do arquivo de classe.

V 9.0.0.6 IBM MQ 9.0.5 foi a liberação Continuous Delivery final para IBM MQ 9.0. Portanto, a partir do IBM MQ 9.0.0 Fix Pack 6 em diante, as informações do Javadoc para o IBM MQ classes for JMS são atualizadas para refletir o comportamento do IBM MQ classes for JMS apenas para recursos disponíveis para Long Term Support clientes.

Conceitos relacionados

“O modelo do JMS” na página 125
O modelo JMS define um conjunto de interfaces que os aplicativos Java podem usar para executar operações do sistema de mensagens... IBM MQ classes for JMS, como um provedor JMS , define como os objetos JMS estão relacionados aos conceitos IBM MQ . A especificação do JMS espera que determinados objetos do JMS sejam objetos administrados. O JMS 2.0 introduz uma API simplificada, enquanto também retém a API clássica, a partir do JMS 1.1.

“Usando a funcionalidade do JMS 2.0” na página 303
JMS 2.0 apresenta várias novas áreas de funcionalidade para o IBM MQ classes for JMS.

Informações relacionadas
Interfaces de linguagem do IBM MQ Java

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.

IBM MQ classes for JMS é uma das duas APIs alternativas que os aplicativos Java podem usar para acessar os recursos do IBM MQ. A outra API é IBM MQ classes for Java. Embora os aplicativos existentes que usam o IBM MQ classes for Java continuem a ser totalmente suportados, novos aplicativos deverão usar o IBM MQ classes for JMS (consulte “Opção de API” na página 77).

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.

- 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 o JNDI para recuperar connection factories e destinos armazenados como objetos administrados e usar somente as interfaces definidas no pacote javax.jms para executar operações do 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. Consulte a IBM nota técnica Usando Interfaces Java do WebSphere MQ em J2EE/JEE para obter detalhes de IBM MQ classes for Java considerações em Java EE.

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 estão agora mais facilmente implementável, independente ou em estruturas de gerenciamento de software, como Maven. Para obter mais informações, consulte a IBM nota técnica Obtendo as classes do WebSphere MQ para JMS.

Opção de API

Novos aplicativos devem usar o IBM MQ classes for JMS em vez do IBM MQ classes for Java.

IBM MQ classes for JMS fornecem acesso aos recursos do sistema de mensagens de publicação/assinatura e ponto a ponto 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. Controle completo do MQMD e cargas úteis das mensagens do MQ estão disponíveis. 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.

Recursos 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. Os aplicativos existentes que usam o IBM MQ classes for Java continuam a ser suportados completamente.

Se você precisar de acesso aos recursos do IBM MQ que não estão disponíveis através do IBM MQ classes for JMS, será possível criar uma Solicitação para Aprimoramento (RFE). IBM pode então avisar se a implementação é possível na implementação do IBM MQ classes for JMS ou se há uma melhor prática que possa 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.

Informações relacionadas

Processo de envio da IBM RFE
Processo de revisão de especificação do JMS Java
Usando as interfaces Java do WebSphere MQ em ambientes J2EE/JEE
Obtendo as classes do WebSphere MQ para JMS
Usando JMS para enviar mensagens PCF
Rastreando aplicativos do IBM MQ classes for JMS
Resolução de problemas do Java e do JMS

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, UNIX and Linux System Services (USS).

V 9.0.0 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.

V 9.0.0 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. Você pode ser capaz de ativar manualmente essa conformidade (novamente, veja a discussão em Modo compatível com FIPS 140 para SunJSSE para alguns ponteiros), mas o IBM não pode fornecer orientação sobre isso atualmente.

É possível usar endereços Internet Protocol Versão 6 (IPv6) 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 do serviço de sistema de arquivos nos arquivos `fscontext.jar` e `providerutil.jar` é fornecido com o 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.

Instalação e configuração do IBM MQ classes for JMS

Esta seção descreve os diretórios e arquivos que são criados ao instalar o IBM MQ classes for JMS e informa como configurar o IBM MQ classes for JMS após a instalação.

Conceitos relacionados

“O que é instalado para IBM MQ classes for JMS” na página 79
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.

“Executando aplicativos IBM MQ classes for JMS no Java Security Manager” na página 95
IBM MQ classes for JMS pode ser executado 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.

“Usando o adaptador de recursos do IBM MQ” na página 418

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.

“Configuração de pós-instalação para os aplicativos IBM MQ classes for JMS” na página 97
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.

“O IVT ponto a ponto para IBM MQ classes for JMS” na página 100
Um programa de teste de verificação de instalação (IVT) ponto a ponto é fornecido com o IBM MQ classes for JMS. 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.

“O IVT de publicar/assinar para IBM MQ classes for JMS” na página 104
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.

“Configurando o adaptador de recursos para comunicação de saída” na página 451
Para configurar a comunicação de saída, defina as propriedades de um objeto ConnectionFactory e um objeto de destino administrado.

“Suporte para OSGi” na página 112
OSGi fornece uma estrutura que suporta a implementação de aplicativos como pacotes configuráveis. Nove pacotes configuráveis do OSGi são fornecidos como parte do IBM MQ classes for JMS.

Tarefas relacionadas

“Verificando a instalação do adaptador de recursos” na página 469
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.

Referências relacionadas

“Scripts fornecidos com IBM MQ classes for JMS” na página 111
Um número de scripts será fornecido para auxiliar com tarefas comuns que precisam ser executadas ao usar o IBM MQ classes for JMS.

Informações relacionadas

Resolução de problemas do IBM MQ classes for JMS ..

Determinação de problemas para o adaptador de recursos do IBM MQ

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 relocalizáveis descritos neste tópico, copiar os arquivos JAR do IBM MQ classes for JMS ou as bibliotecas nativas para outras máquinas ou para um local diferente em uma máquina na qual o IBM MQ classes for JMS foi instalado não é suportado.
- Além disso, incluir o arquivo `com.ibm.mq.allclient.jar` ou o IBM MQ classes for JMS nos archives do aplicativo (como archives de aplicativos corporativos ou arquivos EAR) não é suportado.

É necessário, portanto, evitar o empacotamento de arquivos jar IBM MQ em seus aplicativos (arquivos EAR no WebSphere Application Server), caso contrário, pode-se encontrar problemas inesperados associados à execução de códigos de versão anterior não corrigidos.

Diretórios de instalação

O Tabela 5 na página 80 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**
AIX UNIX 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*`/mqm/V9R0M0/java`
	MQ_INSTALLATION_PATH`/opt/mqm/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:

- O arquivos JAR IBM MQ classes for JMS, 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 84.
- Os scripts adicionais que são descritos em "Scripts fornecidos com IBM MQ classes for JMS" na página 111. Estes 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`:

  – ULW No UNIX, 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" na página 112.
- O adaptador de recursos do IBM MQ, que pode ser implementado em qualquer servidor de aplicativos compatível com o Java Platform, Enterprise Edition 7 (Java EE 7).

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 418

- 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:

- O transporte IBM MQ para SOAP, que fornece um transporte do JMS para SOAP, é instalado no diretório *MQ_INSTALLATION_PATH*\java\lib\soap. Para obter informações adicionais sobre o transporte de IBM MQ para SOAP, consulte "Desenvolvendo serviços da web com o transporte do IBM MQ para SOAP" na página 1307.

  A partir da IBM MQ 9.0, o transporte IBM MQ para SOAP foi descontinuado.

V 9.0.5 V 9.0.0.3 O arquivo JSON4J.jar e o pacote com.ibm.msg.client.mqlight não são necessários para IBM MQ classes for Java e IBM MQ classes for JMS. No IBM MQ 9.0.0 Fix Pack 3 e IBM MQ 9.0.5, as mudanças a seguir serão, portanto, feitas no arquivo com.ibm.mq.allclient.jar:

- A referência ao arquivo JSON4J.jar é removida da instrução de caminho da classe no arquivo manifest para o arquivo com.ibm.mq.allclient.jar.
- O pacote com.ibm.msg.client.mqlight não é mais incluído no arquivo com.ibm.mq.allclient.jar.

Aplicativos de amostra

Alguns aplicativos de amostra são fornecidos com o IBM MQ classes for JMS. O Tabela 6 na página 81 mostra onde os aplicativos de amostra são instalados em cada plataforma.

Tabela 6. Diretórios de amostras

Plataforma	Diretório
AIX UNIX 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*/mqm/V9R0M0/java/samples/jms
	MQ_INSTALLATION_PATH/opt/mqm/samp/jms

O *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.

O "Definindo Variáveis de Ambiente" na página 82 descreve o caminho de classe que é necessário para executar aplicativos simples 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 do IBM MQ classes for JMS" na página 87.

Arquivos JAR relocalizáveis

Dentro de uma empresa, os seguintes arquivos podem ser movidos para os sistemas que precisam executar o IBM MQ classes for JMS:

- `-com.ibm.mq.allclient.jar`
- `-com.ibm.mq.traceControl.jar`
- `-jms.jar`
- `-fscontext.jar`
- `-providerutil.jar`
- O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS

Os arquivos `fscontext.jar` e `providerutil.jar` serão necessários se seu aplicativo executar consultas JNDI usando o contexto de sistema de arquivos.

O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS são necessários. Para obter mais informações, consulte Suporte para JREs não IBM. São necessários os seguintes arquivos JAR:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.0.0.12 `bcutil-jdk15on.jar`

O arquivo `com.ibm.mq.allclient.jar` contém o IBM MQ classes for JMS, o IBM MQ classes for Java e as Classes PCF e Cabeçalhos. Se este arquivo for movido 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 desse arquivo seja anunciado ao Suporte IBM se você estiver obtendo uma correção temporária.

Para determinar a versão do arquivo `com.ibm.mq.allclient.jar`, use o seguinte comando:

```
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:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.0.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

O arquivo `com.ibm.mq.traceControl.jar` é usado para controlar dinamicamente o rastreio a aplicativos IBM MQ classes for JMS. Para obter mais informações, consulte Controlando o rastreio em um processo em execução usando classes do IBM MQ for Java e classes do IBM MQ for JMS.

Informações relacionadas

Problemas na implementação do adaptador de recursos

Definindo Variáveis de Ambiente

Antes de poder compilar e executar aplicativos IBM MQ classes for JMS, a configuração para a variável de ambiente CLASSPATH deve incluir o arquivo IBM MQ classes for JMS Java archive (JAR). Dependendo de

seus requisitos, pode ser necessário incluir outros arquivos JAR no caminho de classe. Para executar os scripts fornecidos com o IBM MQ classes for JMS, outras variáveis de ambiente devem ser configuradas.

Sobre esta tarefa

Importante: A configuração da opção *-Xbootclasspath* do Java para incluir o IBM MQ classes for JMS não é suportada.

Para compilar e executar aplicativos IBM MQ classes for JMS, use a configuração de CLASSPATH para sua plataforma conforme mostrado em Tabela 7 na página 83. A configuração inclui o diretório de amostras para que seja possível compilar e executar os aplicativos de amostra do IBM MQ classes for JMS. Como alternativa, é possível especificar o caminho de classe no comando **java** em vez de usar a variável de ambiente.

Tabela 7. Configuração de CLASSPATH para 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.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Solaris Linux HP-UX HP-UX, Linux e Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/samples/jms/samples:

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

O manifest do arquivo JAR `com.ibm.mqjms.jar` contém referências à 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 da 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 seu aplicativo usa o JTA, deve-se também incluir `jta.jar` em seu caminho da classe.

Nota: Esses arquivos JAR adicionais são necessários somente para compilar seus aplicativos, não para executá-los.

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 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 do IBM MQ classes for JMS estão armazenadas, conforme mostrado em Tabela 8 na página 85.

Procedimento

- Windows
  No Windows, após instalar IBM MQ, execute o comando **setmqenv**.
  Se esse comando não for executado primeiro, a mensagem de erro a seguir poderá aparecer quando você estiver emitindo um comando **dspmqver**:

  V 9.0.2
  ```
  AMQ8351: o ambiente IBM MQ Java não foi configurado
  corretamente ou o recurso IBM MQ JRE não foi instalado.
  ```

  Nota: V 9.0.2 Essa mensagem deverá ser esperada se você não instalou o IBM MQ Java Runtime Environment (JRE).
- Em qualquer outra plataforma, configure você mesmo as variáveis de ambiente:
  - Linux UNIX Para configurar as variáveis de ambiente se você estiver usando uma JVM de 32 bits em sistemas UNIX, ou Linux, é possível usar o script setjmsenv.
  - Linux UNIX Para configurar as variáveis de ambiente, se você estiver usando uma JVM de 64 bits em um sistema UNIX ou Linux, é possível usar o script setjmsenv64. Esses scripts estão no diretório *MQ_INSTALLATION_PATH*`/java/bin`, em que *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

  É possível usar o script setjmsenv ou setjmsenv64 de uma variedade de maneiras: é possível usá-lo 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 iniciada durante o processo de verificação do JMS digitando `. ./setjmsenv` ou `. ./setjmsenv64`

  IBM i No IBM i, deve-se 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. Consulte Configurando o IBM MQ com Java e JMS para obter informações adicionais.

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).

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 9.0 pode ser usada para se conectar a qualquer gerenciador de filas do IBM MQ 9.0 (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 8. 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 HP-UX Linux Linux (plataformas POWER, x86-64 e zSeries s390x) Solaris Solaris (plataformas x86-64 e SPARC)	*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 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
```

- Configurando o ambiente de shell de forma 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
  ```

  Solaris Linux HP-UX
  ```
  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 do IBM MQ classes for JMS
Um arquivo de configuração do IBM MQ classes for JMS especifica propriedades que são usadas para configurar o IBM MQ classes for JMS.

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 do IBM MQ classes for JMS é o de um arquivo de propriedades padrão do Java. 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 o local de um arquivo de configuração do IBM MQ classes for JMS. 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 JMS. 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
```

Este comando identifica o arquivo de configuração IBM MQ classes for JMS como o arquivo `D:\mydir\mjms.config` no sistema Windows local.

Quando um aplicativo for iniciado, o IBM MQ classes for JMS 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 JMS usará os valores padrão para todas as propriedades.

Um arquivo de configuração do IBM MQ classes for JMS 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 JMS. 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 MQI client especificando-o como uma propriedade em um arquivo de configuração do IBM MQ classes for JMS. Para substituir um atributo em um arquivo de configuração do IBM MQ MQI client, use uma entrada com o seguinte formato no arquivo de configuração do IBM MQ classes for JMS:

```
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 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 JMS. Se você especificar ou substituir outros atributos, isso não terá efeito. Especificamente, observe que `ChannelDefinitionFile` e `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 264 para obter detalhes de como usar o CCDT com o IBM MQ classes for JMS.

Tabela 9. Qual sub-rotina do arquivo de configuração do cliente contém qual atributo

Sub-rotina	**Atribuir**
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
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 do IBM MQ MQI client, consulte Configurando um cliente usando um arquivo de configuração

Sub-rotina de rastreio do ambiente Java Standard
Use a sub-rotina de Configurações de rastreio do ambiente Java Standard para configurar o recurso de rastreio do IBM MQ classes for JMS.

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:

- Para o IBM MQ classes for JMS for IBM MQ 9.0.0 Fix Pack 1 ou anterior, o rastreio é gravado em um arquivo chamado `mqjms_%PID%.trc`.
- V 9.0.0.2 A partir da IBM MQ 9.0.0 Fix Pack 2, 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%.trc`.
- V 9.0.0.2 Na IBM MQ 9.0.0 Fix Pack 2, se o aplicativo tiver carregado o IBM MQ classes for JMS do arquivo JAR realocável `com.ibm.mq.allclient.jar`, o rastreio será gravado em um arquivo chamado `mqjavaclient_%PID%.trc`.
- V 9.0.0.10 No IBM MQ 9.0.0 Fix Pack 10, 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 denominado `mqjava_%PID%.cl%u.trc`.
- V 9.0.0.10 Na IBM MQ 9.0.0 Fix Pack 10, se o aplicativo tiver carregado o IBM MQ classes for JMS do arquivo JAR realocável `com.ibm.mq.allclient.jar`, o rastreio será gravado em um arquivo chamado `mqjavaclient_%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 estão executando o rastreio em diferentes carregadores de classe do Java.

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 JMS.

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 JMS 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 JMS.

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 JMS 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 JMS 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 JMS 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_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.

FALSE
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 do sistema de mensagens do IBM MQ, a propriedade `receiveConversionCCSID` pode ser configurada para substituir o valor de CCSID padrão 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. IBM MQ 8.0 inclui a propriedade `com.ibm.mq.jms.SupportMQExtensions`, que pode ser configurada como *TRUE*, para reverter esses comportamentos alterados para implementações anteriores. 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 309
A especificação JMS 2.0 introduz mudanças na forma como determinados comportamentos funcionam. O IBM MQ 8.0 e posterior inclui a propriedade `com.ibm.mq.jms.SupportMQExtensions`, que pode ser definida como *TRUE* para reverter esses comportamentos modificados de volta para implementações anteriores.

A configuração de STEPLIB para IBM MQ classes for JMS no z/OS
No z/OS, o STEPLIB usado no tempo de execução deve conter as bibliotecas SCSQAUTH e SCSQANLE do IBM MQ. Especifique essas bibliotecas na JCL de inicialização ou usando o arquivo `.profile`.

No UNIX and Linux System Services, é possível incluí-los usando uma linha em seu `.profile` , conforme mostrado no fragmento de código a seguir, substituindo `thlqual` pelo qualificador de conjunto de dados de alto nível que você escolheu ao instalar o IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

Em outros ambientes, geralmente é necessário editar a JCL de inicialização para incluir SCSQAUTH e SCSQANLE na concatenação STEPLIB:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS e ferramentas de gerenciamento de software
Ferramentas de gerenciamento de software como Apache Maven podem ser usadas com o IBM MQ classes for JMS.

Muitas das grandes organizações de desenvolvimento usam essas ferramentas para gerenciar centralmente os repositórios de bibliotecas de terceiros.

Os IBM MQ classes for JMS 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 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.

No IBM MQ 8.0, os seguintes arquivos JAR precisam ser instalados no repositório:

- `com.ibm.mq.allclient.jar`.
- `jms.jar` é necessário se você estiver usando o IBM MQ classes for JMS.
- `fscontext.jar` é necessário se você estiver usando o IBM MQ classes for JMS e acessar objetos administrados JMS que são armazenados em um contexto de JNDI do sistema de arquivos.
- `providerutil.jar` se você estiver usando o IBM MQ classes for JMS e acessar objetos administrados JMS que são armazenados em um contexto de JNDI do sistema de arquivos.

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 "O que é instalado para IBM MQ classes for JMS" na página 79 e Suporte para JREs não IBM.

Executando aplicativos IBM MQ classes for JMS no Java Security Manager

IBM MQ classes for JMS pode ser executado 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, este arquivo está no diretório `lib/security/java.policy` relativo 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.

Importante:

Sempre que possível, o termo *lista de permissões* substitui o termo *lista de desbloqueio*. Para IBM MQ 9.0 e liberações posteriores, isso inclui os nomes de propriedade do sistema Java mencionados neste tópico (**`com.ibm.mq.jms.*`**). Você não tem que alterar nenhuma configuração existente. Os nomes de propriedades do sistema anteriores também continuam funcionando.

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.
- ProprityPermission (leitura) nas propriedades **`com.ibm.mq.jms.allowlist`**, **`com.ibm.mq.jms.allowlist.discover`** e **`com.ibm.mq.jms.allowlist.mode`**.

Para Continuous Delivery, a lista de permissões ClassName é suportada a partir do IBM MQ 9.0.1. Para obter informações adicionais, consulte "Conceitos de listagem de permissões" na página 116.

V9.0.0.1 Na liberação Long Term Support , a listagem de permissões ClassName é suportada com APAR IT14385e a partir do IBM MQ 9.0.0 Fix Pack 1.

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. Esse arquivo precisará ser customizado, para especificar os locais de determinados arquivos e diretórios: *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual IBM MQ está instalado, *MQ_DATA_DIRECTORY* representa o local do diretório de dados 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.

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 mais informações sobre o modo de migração do provedor de sistema de mensagens IBM MQ, consulte Configurando a propriedade **PROVIDERVERSION** do JMS

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).

Um aplicativo em execução no WebSphere Application Server no z/OS pode se conectar a um gerenciador de filas no modo de ligações ou de cliente, mas um aplicativo em execução em qualquer outro ambiente no z/OS pode se conectar a um gerenciador de filas apenas no modo de ligações. 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 84 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_SERIALIZE_CONN_TAG_QSG

- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- 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 733

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.

ULW No UNIX, 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 (IVT) ponto a ponto é fornecido com o IBM MQ classes for JMS. 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.

Um script é fornecido para executar o programa IVT. O script é chamado IVTRun nos sistemas UNIX and Linux e IVTRun.bat no Windows e está no subdiretório `bin` do diretório de instalação do IBM MQ classes for JMS.

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 1086 observando que o canal a ser usado use como padrão SYSTEM.DEF.SVRCONN e a fila a ser usada é SYSTEM.DEFAULT.LOCAL.QUEUE e, em seguida, insira o comando a seguir:

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

Nenhum script equivalente é fornecido em sistemas z/OS, mas é possível executar o IVT no modo de ligações, chamando a classe Java diretamente, usando o comando a seguir:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

O caminho de classe deve conter com.ibm.mqjms.jar.

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, 2023. 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

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 77. Para obter informações sobre como configurar a ferramenta de administração do IBM MQ JMS, consulte Configurando a ferramenta de administração do 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 nos sistemas UNIX and Linux e IVTSetup.bat no Windows e está no subdiretório `bin` do diretório de instalação do 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 baseado em um servidor LDAP
- `file:/directoryPath` , para um serviço de diretório baseado 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 baseado em um servidor LDAP. Esse é o valor-padrão.
- `com.sun.jndi.fscontext.RefFSContextFactory`, para um serviço de diretório baseado 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 nos sistemas UNIX and Linux e IVTTidy.bat no Windows e está no subdiretório `bin` do diretório de instalação do 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 “Definindo Variáveis de Ambiente” na página 82.
- 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`.

  nome do host
  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 1086 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 chamado PSIVTRun nos sistemas UNIX and Linux e PSIVTRun.bat no Windows e está no subdiretório `bin` do diretório de instalação do 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

1086, 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 com número de versão v 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, 2023. All
Rights Reserved.
IBM MQ classes for Java(tm) 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 77. Para obter informações sobre como configurar a ferramenta de administração do IBM MQ JMS, consulte Configurando a ferramenta de administração do 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 nos sistemas UNIX and Linux e IVTSetup.bat no Windows e está no subdiretório `bin` do diretório de instalação do 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 baseado em um servidor LDAP
- `file:/directoryPath` , para um serviço de diretório baseado 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 baseado em um servidor LDAP. Esse é o valor-padrão.
- `com.sun.jndi.fscontext.RefFSContextFactory`, para um serviço de diretório baseado 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 nos sistemas UNIX and Linux e IVTTidy.bat no Windows e está no subdiretório `bin` do diretório de instalação do 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 “Definindo Variáveis de Ambiente” na página 82.
- 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`.

nome do host
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 1086.

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 mais informações, consulte "Gravando aplicativos do IBM MQ classes for JMS" na página 125. As amostras podem estar sujeitas a mudanças em futuras liberações do IBM MQ.

A Tabela 10 na página 108 mostra o local de instalação dos aplicativos de amostra do IBM MQ classes for JMS em cada plataforma:

Tabela 10. Diretórios de instalação para os aplicativos de amostra do IBM MQ classes for JMS

Plataforma	Diretório
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*/samp/jms/samples
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms\samples
IBM i IBM i	/qibm/proddata/mqm/java/samples/jms/samples
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

Nesse diretório, há subdiretórios que contêm um ou mais aplicativos de amostra, conforme mostrado em Tabela 11 na página 109.

Tabela 11. 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.
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.

Tabela 11. Aplicativos de Amostra do IBM MQ classes for JMS (continuação)	
Nome da amostra	**Descrição**
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.
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 12 na página 110 mostra o local do script em cada plataforma:

Tabela 12. Localização do script `runjms`	
Plataforma	**Diretório**
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

Tabela 12. Localização do script `runjms` (continuação)	
Plataforma	**Diretório**
Windows	*MQ_INSTALLATION_PATH*\java\bin\runjms.bat
IBM i	/qibm/proddata/mqm/java/bin/runjms ou /qibm/proddata/mqm/java/bin/runjms64
z/OS	*MQ_INSTALLATION_PATH*java/bin/runjms

Para usar o script `runjms` para chamar um aplicativo de amostra, conclua as etapas a seguir:

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 no 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 79
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

Um número de scripts será fornecido para auxiliar com tarefas comuns que precisam ser executadas ao usar o IBM MQ classes for JMS.

Tabela 13 na página 111 lista todos os scripts e seus usos. Os scripts estão no subdiretório `bin` do diretório de instalação do IBM MQ classes for JMS.

Tabela 13. Scripts fornecidos com IBM MQ classes for JMS	
Utilitário	**Usar**
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.

Tabela 13. Scripts fornecidos com IBM MQ classes for JMS (continuação)	
Utilitário	**Usar**
formatLog [1]	Esse script é mantido para compatibilidade com liberações anteriores, mas não executa nenhuma função. Agora a saída de log é 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 100.
JMSAdmin [1]	Executa a ferramenta de administração do IBM MQ JMS, conforme descrito em Iniciando a ferramenta de administração.
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 104.
PSReportDump.class	Essa classe é mantida para compatibilidade com liberações anteriores, mas não executa nenhuma função.
setjmsenv	Configura as variáveis de ambiente para executar um aplicativo IBM MQ classes for JMS em uma máquina virtual Java (JVM) de 32 bits em sistemas UNIX and Linux, conforme descrito em "Definindo Variáveis de Ambiente" na página 82.
setjmsenv64	Configura as variáveis de ambiente para executar um aplicativo IBM MQ classes for JMS em uma JVM de 64 bits em sistemas UNIX and Linux, conforme descrito em "Definindo Variáveis de Ambiente" na página 82.

Nota:

1. No Windows, o nome do arquivo tem a extensão de arquivo .bat.

Suporte para OSGi

OSGi fornece uma estrutura que suporta a implementação de aplicativos como pacotes configuráveis. Nove pacotes configuráveis do OSGi são fornecidos como parte do IBM MQ classes for JMS.

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 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 do IBM MQ para JMS, consulte 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 do IBM MQ classes for JMS, consulte 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.osgi.allclient_*version_number*.jar
Esse arquivo JAR permite que aplicativos usem o IBM MQ classes for JMS e o IBM MQ classes for Java e também inclui o código para manipular mensagens do PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
Este arquivo JAR fornece os pré-requisitos para `com.ibm.mq.osgi.allclient_`*`version_number`*`.jar` onde *`version_number`* é o número da versão de IBM MQ que está instalada.

Os pacotes configuráveis são instalados no subdiretório `java/lib/OSGi` de sua instalação da pasta IBM MQ ou `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. Para versões do produto anteriores ao IBM MQ 8.0, essa restrição de usar o IBM MQ classes for JMS ou o IBM MQ classes for Java se aplica.

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.

Obtendo o IBM MQ classes for JMS separadamente

O IBM MQ classes for JMS está disponível em um arquivo JAR autoextrator que poderá ser transferido por download da Fix Central se você desejar obter apenas os arquivos JAR do IBM MQ classes for JMS para implementação em uma ferramenta de gerenciamento de software ou para usar com aplicativos clientes independentes.

Antes de começar

Antes de iniciar essa tarefa, certifique-se de que tenha um Java runtime environment (JRE) instalado em sua máquina e que o JRE tenha sido 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.

Sobre esta tarefa

Antes do IBM MQ 8.0, o IBM WebSphere MQ classes for Java ou o IBM WebSphere MQ classes for JMS não estão disponíveis como download separado. Para o IBM WebSphere MQ 7.5 ou anterior, se você estiver desenvolvendo e executando os aplicativos de linguagem Java que usam o IBM WebSphere MQ classes for Java ou o IBM WebSphere MQ classes for JMS, será necessário instalá-los executando uma instalação completa do servidor ou instalando um dos SupportPacs do cliente no sistema em que o aplicativo estiver sendo desenvolvido e no sistema em que o aplicativo será executado. Isso instala muito mais arquivos do que os arquivos IBM WebSphere MQ classes for Java e IBM WebSphere MQ classes for JMS.

No entanto, a partir do IBM MQ 8.0, os arquivos a seguir estão disponíveis dentro de um arquivo JAR autoextrator, que minimiza o tamanho do download e da instalação, além do tempo necessário para executar a instalação:

- O IBM MQ classes for JMS
- O IBM MQ classes for Java
- O adaptador de recursos do IBM MQ
- Os pacotes configuráveis OSGi do IBM MQ

Ao executar o arquivo JAR executável, ele exibe o contrato de licença do IBM MQ, que deve ser aceito. Ele solicita um diretório no qual instalar o IBM MQ classes for Java, o IBM MQ classes for JMS, o adaptador de recursos e os pacotes configuráveis OSGi. Se o diretório de instalação selecionado não existir, ele será criado e os arquivos de programa serão instalados. No entanto, se o diretório existir, um erro será relatado e nenhum arquivo será instalado.

Procedimento

1. Faça download do arquivo JAR do IBM MQ Java da Fix Central.

   Para localizar a versão mais recente que está disponível para download, insira a frase "Java" na caixa **Procura de texto**. O nome do arquivo a ser transferido por download está no formato *V.R.M.F*-WS-MQ-Install-Java-All.jar, em que *V.R.M.F* é o número da versão do produto, por exemplo, 9.0.0.0.

   Se não for possível localizar o arquivo, certifique-se de que **Produto selecionado** seja WebSphere MQ e **Versão** seja 9.0.

2. Inicie a instalação do diretório para o qual você transferiu o arquivo por download.

   Para iniciar a instalação, insira um comando no formato a seguir:

   ```
   java -jar V.R.M.F-WS-MQ-Install-Java-All.jar
   ```

   em que *V.R.M.F* é o número da versão do produto, por exemplo, 9.0.0.0, e *V.R.M.F*-WS-MQ-Install-Java-All.jar é o nome do arquivo que foi transferido por download do Fix Central.

   Por exemplo, para instalar o IBM MQ classes for JMS para IBM MQ 9.0.0.0, você usaria o comando a seguir:

   ```
   java -jar 9.0.0.0-WS-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 9.0, 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, se você selecionar 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, pressionar x ignora a exibição dos termos de licença adicionais.

      Após a exibição dos termos de licença adicionais ou imediatamente, se você selecionar 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, a seleção de 2 termina a instalação imediatamente.

      Se você selecionar 1, a mensagem a seguir aparecerá:

      ```
      Insira um diretório para arquivos do produto ou mantenha em branco para aceitar o valor
      padrão.
      O diretório de destino padrão é H:\WMQ

      Diretório de destino para arquivos do produto?
      ```

4. Especifique o diretório de instalação para o adaptador de recursos:
   - Se desejar instalar os arquivos do produto no local padrão, pressione Enter sem especificar um valor.
   - Se desejar instalar os arquivos do produto em um local diferente do padrão, especifique o nome do diretório no qual você deseja instalar os arquivos do produto e, em seguida, pressione Enter para iniciar a instalação.

   O nome do diretório que for especificado ainda não deverá existir, caso contrário, quando você iniciar a instalação, um erro será relatado e nenhum arquivo será instalado.

   Contanto que ele ainda não exista, o diretório de instalação selecionado será criado e os arquivos de programa serão instalados nesse diretório. Durante a instalação, um novo diretório com o nome `wmq` é criado no diretório de instalação selecionado. São criados três subdiretórios, `JavaEE`, `JavaSE` e `OSGi` no diretório `wmq`, com o conteúdo a seguir:

   ```
   .\JavaEE:
   wmq.jmsra.ivt.ear
   wmq.jmsra.rar

   .\JavaSE:
   com.ibm.mq.allclient.jar
   com.ibm.mq.traceControl.jar
   fscontext.jar
   jms.jar
   providerutil.jar

   .\OSGi:
   com.ibm.mq.osgi.allclient_V.R.M.F.jar
   com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar
   ```

   em que *V.R.M.F* é a Versão, Liberação, Modificação e o número do Fix Pack.

V 9.0.5 V 9.0.0.3 Antes do IBM MQ 9.0.0 Fix Pack 3 e IBM MQ 9.0.5, os arquivos que são instalados no diretório JavaSE incluem o arquivo JSON4J.jar. No entanto, esse arquivo JAR não é necessário e por isso foi removido do arquivo *V.R.M.F*-WS-MQ-Install-Java-All.jar na IBM MQ 9.0.0 Fix Pack 3 e na IBM MQ 9.0.5. Além disso, a partir de IBM MQ 9.0.0 Fix Pack 3 e IBM MQ 9.0.5, há duas mudanças no com.ibm.mq.allclient.jar file:

- A referência ao arquivo JSON4J.jar é removida da instrução de caminho da classe no arquivo manifest para o arquivo com.ibm.mq.allclient.jar.
- O pacote com.ibm.msg.client.mqlight não é mais incluído no arquivo com.ibm.mq.allclient.jar.

Quando a instalação for concluída, uma mensagem de confirmação será exibida, conforme mostrado no exemplo a seguir:

```
Extraindo arquivos para H:\WMQ\wmq
Todos os arquivos do produto foram extraídos com sucesso.
```

V 9.0.0.1 Listagem de permissões no IBM MQ classes for JMS

O mecanismo de serialização e desserialização de objetos do Java foi identificado como um risco de segurança em potencial. A listagem de permissões no IBM MQ classes for JMS fornece alguma proteção contra alguns riscos de serialização.

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. Um aplicativo notável de serialização está no Java Message Service (JMS) ObjectMessages que usa a serialização para conter 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.

Listagem de permissões no IBM MQ classes for JMS

Consulte:

- “Conceitos de listagem de permissões” na página 116 para obter uma visão geral da listagem de permissões
- “Configurando e usando uma lista de permissões do JMS” na página 120 para obter informações sobre como configurar uma lista de permissões
- “Listagem de permissões no WebSphere Application Server” na página 122 para obter informações sobre como configurar uma lista de permissões no WebSphere Application Server.

Conceitos relacionados

“Executando aplicativos IBM MQ classes for JMS no Java Security Manager” na página 95
IBM MQ classes for JMS pode ser executado 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.

V 9.0.0.1 *Conceitos de listagem de permissões*

No IBM MQ classes for JMS, o suporte para a listagem de permissões de classes na implementação da interface do ObjectMessage JMS fornece uma mitigação em potencial com relação a alguns dos riscos de segurança que potencialmente se relacionam ao mecanismo de serialização e desserialização de objeto do Java.

Listagem de permissões no IBM MQ classes for JMS

Importante:

Sempre que possível, o termo *lista de permissões* substitui o termo *lista de desbloqueio*. Para IBM MQ 9.0 e liberações posteriores, isso inclui os nomes de propriedade do sistema Java mencionados neste tópico (**com.ibm.mq.jms.***). Você não tem que alterar nenhuma configuração existente. Os nomes de propriedades do sistema anteriores também continuam funcionando.

O IBM MQ classes for JMS suporta a listagem de permissões de classes na implementação da interface ObjectMessage do JMS.

A lista de permissões define quais classes do Java podem ser serializadas com ObjectMessage.setObject() e desserializadas com ObjectMessage.getObject().

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.

Produzindo a lista de permissões

Importante: O IBM MQ classes for JMS não pode ser distribuído 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
Essa propriedade pode ser especificada de uma das seguintes maneiras:

- O nome do caminho do arquivo que contém a lista de permissões, no 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

- 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 esta propriedade for configurada para true e a lista de permissões não tiver sido especificada usando a propriedade com.ibm.mq.jms.allowlist, o mecanismo de allowlist é 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
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.

Por isso, `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 da 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 em uma java.lang.IllegalArgumentException sendo lançada.
- 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, já que ele é o caractere de comentário padrão em muitos shells UNIX ou 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.
- ProprityPermission (leitura) nas propriedades **`com.ibm.mq.jms.allowlist`**, **`com.ibm.mq.jms.allowlist.discover`** e **`com.ibm.mq.jms.allowlist.mode`**.

Informações adicionais

V9.0.0.1

Consulte “Configurando e usando uma lista de permissões do JMS” na página 120 e “Listagem de permissões no WebSphere Application Server” na página 122 para obter mais informações sobre listas de permissões.

Conceitos relacionados

"Executando aplicativos IBM MQ classes for JMS no Java Security Manager" na página 95
IBM MQ classes for JMS pode ser executado 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.

V 9.0.0.1 *Configurando e usando uma lista de permissões do JMS*

Essas informações indicam como funciona uma lista de permissões e como você configura uma usando a funcionalidade contida no IBM MQ classes for JMS 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*. Para IBM MQ 9.0 e liberações posteriores, isso inclui os nomes de propriedade do sistema Java mencionados neste tópico (**com.ibm.mq.jms.***). Você não tem que alterar 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 116

Sobre esta tarefa

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 do 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 os ObjectMessages com as propriedades de sistema Java especificadas a seguir:

   ```
   -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.

V 9.0.0.1 *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*. Para IBM MQ 9.0 e liberações posteriores, isso inclui os nomes de propriedade do sistema Java mencionados neste tópico (**com.ibm.mq.jms.***). Você não tem que alterar 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. Essa funcionalidade foi incluída no adaptador de recursos como parte do APAR IT14385.

Consulte “Usando o IBM MQ e o WebSphere Application Server juntos” na página 477 para obter informações adicionais sobre o uso dos dois produtos.

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 do JMS” na página 120.

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 de *Descoberta* e registram os detalhes do ObjectMessages do JMS, que o servidor de aplicativos processa, no arquivo `C:\allowlist.txt`

A configuração a seguir:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

faz com que o servidor de aplicativos carregue o arquivo `C:/allowlist.txt` e use 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 95
IBM MQ classes for JMS pode ser executado 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 de IBM MQ 8.0, as conversões de sequência em IBM MQ classes for JMS eram 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.

Configurar

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`.

Atualmente, a decodificação assume o comportamento padrão para `CharsetDecoders`, que relata erros lançando 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.

UnmappableCharacterConfigurações de ação e bytes de substituição em IBM MQ Classes para 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 327
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 do IBM MQ classes for JMS

Após uma breve introdução ao modelo do JMS, este tópico fornece orientação detalhada sobre como gravar aplicativos do IBM MQ classes for JMS.

O modelo do JMS

O modelo JMS define um conjunto de interfaces que os aplicativos Java podem usar para executar operações do sistema de mensagens... IBM MQ classes for JMS, como um provedor JMS , define como os objetos JMS estão relacionados aos conceitos IBM MQ . A especificação do JMS espera que determinados objetos do JMS sejam objetos administrados. O JMS 2.0 introduz uma API simplificada, enquanto também retém a API clássica, a partir do JMS 1.1.

A especificação do JMS e o pacote javax.jms definem um conjunto de interfaces que os aplicativos Java podem usar para executar operações do sistema de mensagens.

No IBM MQ 8.0, o produto suporta a versão JMS 2.0 do padrão JMS, que introduz uma API simplificada, enquanto também retém a API clássica do JMS 1.1.

API simplificada

O JMS 2.0 introduz 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.

JMSProducer
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 enviada da seguinte forma:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

O JMS 2.0 também introduz assinaturas compartilhadas em que mensagens podem ser compartilhadas entre múltiplos 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.

Session
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 126 mostra esses objetos e seus relacionamentos.

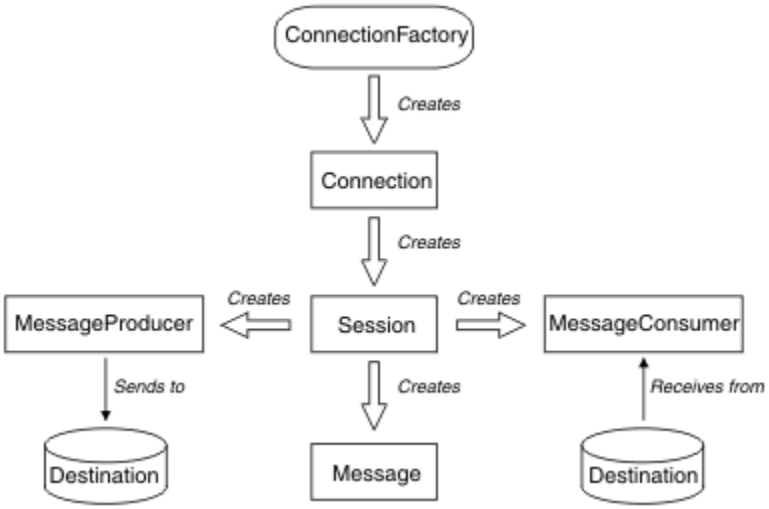


Figura 9. Os objetos do JMS e seus relacionamentos

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.

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 14 na página 127 lista as interfaces independentes de domínio do JMS e suas interfaces específicas de domínio correspondentes.

Tabela 14. 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
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

O JMS 2.0 retém todas as interfaces específicas do domínio e, portanto, os aplicativos existentes ainda podem usar essas interfaces. Para novos aplicativos, no entanto, considere usar as interfaces independentes de domínio do JMS 1.1 ou a API simplificada do JMS 2.0.

No IBM MQ classes for JMS, os objetos do JMS são relacionadas aos conceitos do IBM MQ das seguintes formas:

- 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 o IBM MQ classes for JMS, todas as regras normais do 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).

No IBM MQ classes for JMS, a implementação da interface Destination é uma superclasse abstrata de Queue e Topic, e, portanto, uma instância de Destination é um objeto Queue ou um objeto Topic. 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.

No IBM MQ classes for JMS, portanto, os objetos dos tipos a seguir podem ser objetos administrados:

- ConnectionFactory

- QueueConnectionFactory
- TopicConnectionFactory
- Fila
- Tópico
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Conceitos relacionados

"Usando a funcionalidade do JMS 2.0" na página 303
JMS 2.0 apresenta várias novas áreas de funcionalidade para o IBM MQ classes for JMS.

Informações relacionadas

Interfaces de linguagem do IBM MQ Java

Mensagens do JMS

As mensagens do JMS são compostas por cabeçalho, propriedades e corpo. O JMS define cinco tipos de corpo de 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.

Corpo
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 contendo um identificador de mensagem IBM MQ padrão, as classes IBM MQ para JMS usam 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 contendo um identificador de correlação IBM MQ padrão, as classes IBM MQ para JMS usam o **MatchOption** MQMO_MATCH_CORREL_ID para obter a mensagem com o identificador de correlação especificado da 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 MQMO_MATCH_MSG_ID e MQMO_MATCH_CORREL_ID **MatchOptions**, 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 133 mostra como a estrutura de uma mensagem do JMS é transformada em uma mensagem do IBM MQ e de volta:

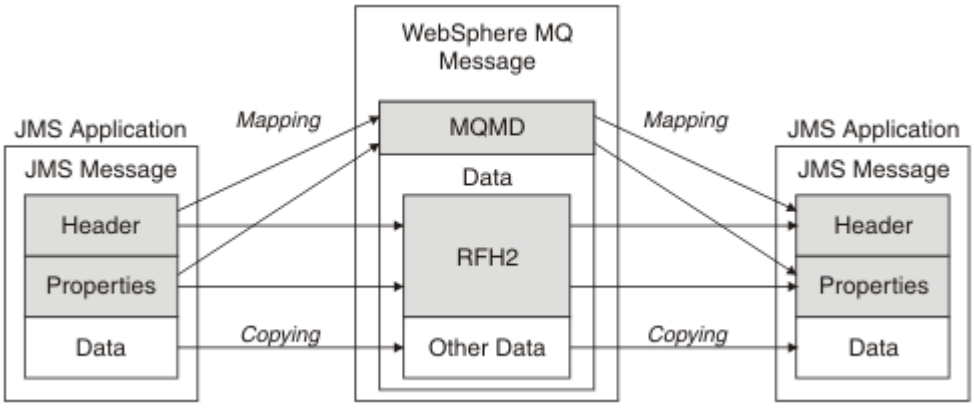


Figura 10. Como as mensagens são transformadas entre o JMS e o IBM MQ usando o cabeçalho MQRFH2

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.)

O NameValueCCSID obtém valores da seguinte tabela:

V 9.0.0

Tabela 15. 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

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 16. Nome da propriedade mcd, sinônimo, tipo de dados e pasta

Sinônimo da Propriedade	Nome da Propriedade	Tipo de Dados	Pasta (Folder)
	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

jms contém campos de cabeçalho JMS e propriedades do JMSX que não podem ser totalmente expressadas no MQMD. A pasta jms está sempre presente em uma JMS MQRFH2.

A pasta usr

usr contém propriedades de JMS definidas pelo aplicativo associadas à mensagem. A pasta usr estará presente somente se 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 17. Nome da propriedade mqext, sinônimo, tipo de dados e pasta

Sinônimo da Propriedade	Nome da Propriedade	Tipo de Dados	Pasta (Folder)
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 18. Nome da propriedade `mqps`, sinônimo, tipo de dados e pasta

Sinônimo da Propriedade	Nome da Propriedade	Tipo de Dados	Pasta (Folder)
`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 19 na página 136 mostra uma lista completa de nomes de propriedades.

Tabela 19. 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	cadeia de caracteres
JMSExpiration	grande	jms	Expand.	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Sequência	jms	Cid	cadeia de caracteres
JMSReplyTo	Destino	jms	Rto	cadeia de caracteres
JMSTimestamp	grande	jms	Tms	i8

Tabela 19. Pastas MQRFH2 e propriedades usadas pelo JMS (continuação)				
Nome do campo JMS	**Tipo de Java**	**Nome da pasta MQRFH2**	**Nome da Propriedade**	**Tipo/valores**
JMSType	Sequência	mcd	Tipo, Configuração, Fmt	cadeia de caracteres
JMSXGroupID	Sequência	jms	Gid	cadeia de caracteres
JMSXGroupSeq	int	jms	Seq.	i4
xxx (definido pelo usuário)	Qualquer	usr	xxx	qualquer
		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, porque o tipo de dados é predefinido. Se for incluído, um ou mais caracteres de espaço deve ser incluído antes da tag `dt=`.

name
é o nome da propriedade; consulte Tabela 19 na página 136.

datatype
deve corresponder, após compactação, a um dos tipos de dados listados em Tabela 20 na página 138.

value
é uma representação de sequência do valor a ser transmitido, usando as definições em Tabela 20 na página 138.

Um valor nulo é codificado usando a seguinte sintaxe:

```
<name dt='datatype' xsi:nil='true'></name>
```

Não use `xsi:nil='false'`.

Tabela 20. Tipos de dados de propriedade	
Tipo de Dados	**Definição**
cadeia de caracteres	Qualquer sequência de caracteres excluindo < e &
booleano	O caractere `0` ou `1` (`0` = false, `1` = true)
bin.hex	Dígitos hexadecimais que representam octetos
i1	Um número, expresso usando dígitos `0..9`, com sinal opcional (nenhuma fração ou expoente). Deve estar no intervalo de -128 a 127, inclusive
i2	Um número, expresso usando dígitos `0..9`, com sinal opcional (nenhuma fração ou expoente). Deve estar no intervalo de -32768 a 32767, inclusive
i4	Um número, expresso usando dígitos `0..9`, com sinal opcional (nenhuma fração ou expoente). Deve estar no intervalo de -2147483648 a 2147483647, inclusive
i8	Um número, expresso usando dígitos `0..9`, com sinal opcional (nenhuma fração ou expoente). Deve estar no intervalo de -9223372036854775808 a 9223372036854775807, inclusive
int	Um número, expresso usando dígitos `0..9`, com sinal opcional (nenhuma fração 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.

Tabela 21 na página 138 lista os campos de cabeçalho JMS e Tabela 22 na página 139 lista as propriedades JMS que são mapeadas diretamente para campos MQMD. Tabela 23 na página 139 lista as propriedades específicas do provedor e os campos MQMD para os quais elas são mapeadas.

Tabela 21. 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

Tabela 21. Mapeamento de campos de cabeçalho do JMS para campos MQMD (continuação)

Campo de cabeçalho doJMS	Tipo de Java	Campo MQMD	tipo C
JMSExpiration	long	Expiração	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	Sequência de caracteres	MsgID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Sequência de caracteres	ID de Correlação	MQBYTE24

Tabela 22. Mapeamento de propriedades do JMS para campos MQMD

Propriedade JMS	Tipo de Java	Campo MQMD	tipo C
JMSXUserID	Sequência de caracteres	UserIdentifier	MQCHAR12
JMSXAppID	Sequência de caracteres	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	Sequência de caracteres	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tabela 23. 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

Tabela 23. 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_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 de caracteres	Formatar "1" na página 140	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Codificação	MQLONG
JMS_IBM_Character_Set	Sequência de caracteres	CodedCharacterSetId "2" na página 140	MQLONG
JMS_IBM_PutDate	Sequência de caracteres	PutDate	MQCHAR8
JMS_IBM_PutTime	Sequência de caracteres	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 24 na página 141 mostra como os campos de cabeçalho do JMS são mapeados para campos MQMD/RFH2 no momento de send() ou publish(). O Tabela 25 na página 141 mostra como as propriedades do JMS são mapeadas para campos MQMD/RFH2 no momento de send () ou publish(). O Tabela 26 na página 142 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 24. 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
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 25. 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

Tabela 26. 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	Encoding		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
Este é armazenado como uma sequência que serializa as características importantes do objeto de destino, de forma que um recebimento do JMS possa reconstruir um objeto de destino equivalente. O campo MQRFH2 é codificado como URI (consulte "Identificadores uniformes de recursos (URIs)" na página 204 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 204 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 221.

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. Nenhum valor CorrelId é codificado no MQRFH2.

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*. O aplicativo deve configurar o valor dessas propriedades para as constantes padrão do IBM MQ MQRO_ (incluído no com.ibm.mq.MQC). Então, por exemplo, para solicitar COD com dados completos, o aplicativo deve configurar JMS_IBM_Report_COD para o valor CMQC.MQRO_COD_WITH_FULL_DATA.

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

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.

Tabela 27 na página 146 mostra como os campos de cabeçalho JMS são mapeados para campos MQMD/MQRFH2 nos tempos de get () ou receive (). O Tabela 28 na página 146 mostra como os campos de propriedade JMS são mapeados para campos MQMD/MQRFH2 nos tempos de get () ou receive (). O Tabela 29 na página 147 mostra como propriedades específicas do provedor JMS são mapeadas.

Tabela 27. 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 146
JMSDeliveryMode	Persistence "2" na página 146	jms.Dlv "2" na página 146
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate "2" na página 146 PutTime "2" na página 146	jms.Tms "2" na página 146
JMSCorrelationID	CorrelId "2" na página 146	jms.Cid "2" na página 146
JMSReplyTo	ReplyToQ "2" na página 146 ReplyToQMgr "2" na página 146	jms.Rto "2" na página 146
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 28. 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 147	jms.Gid "1" na página 147
JMSXGroupSeq	MsgSeqNumber "1" na página 147	jms.Seq "1" na página 147

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 29. 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 147	Codificação	
JMS_IBM_Character_Set "1" na página 147	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 148 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.

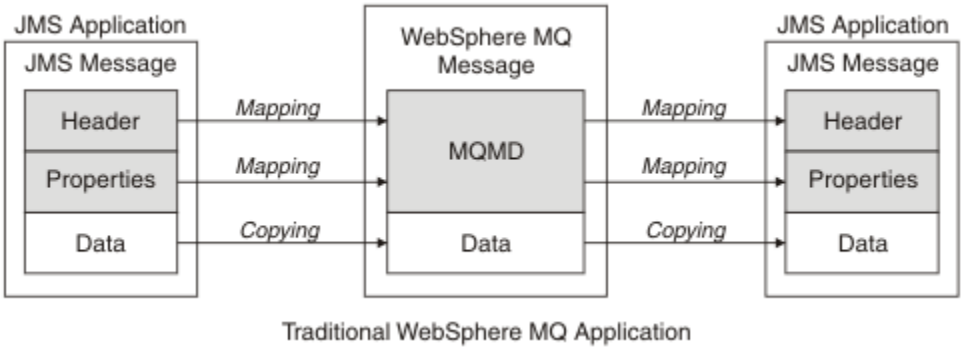


Figura 11. Como as mensagens do JMS são transformadas em mensagens do IBM MQ sem o cabeçalho MQRFH2

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 20 na página 138. 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 20 na página 138. 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 mensagem em JMS é realizada ao enviar e receber mensagens. O IBM MQ executa a maior parte da conversão de dados 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 os números e o texto são representados. O IBM MQ gerencia a codificação de textos e números em mensagens do JMS, exceto para `JMSObjectMessage`. Consulte "JMSObjectMessage" na página 157. 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 dos 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 151
2. “Conversão de Dados do Aplicativo” na página 152
3. “Conversão de dados do gerenciador de filas” na página 152
4. “Conversão de dados do canal de mensagens” na página 153

Onde a conversão deve ser executada?

A seção, “Escolha uma abordagem para a conversão de mensagem: receiver makes good” na página 153, 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 30 na página 154.

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 164.

[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 executar a conversão de dados de caracteres explícitos usando a classe `java.nio.charset.Charset`; consulte os exemplos em Figura 14 na página 156 e Figura 15 na página 156. 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 que corresponda 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 172.

Conversão de dados do gerenciador de filas

No IBM MQ 7.0, a conversão da página de códigos pode ser executada por um gerenciador de filas 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 156. 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 153.

Antes de 7.0, as conversões eram sempre executadas pelo cliente do JMS A conversão de dados do clienteJMS será restrita ao converter as sequencias de números, texto do tipo e comprimento conhecido para o cliente JMS. Não é possível converter as estruturas de dados; consulte “Trocando um registro formatado com um aplicativo não JMS” na página 172. No 7.0, até o fix pack 7.0.1.5, se a conversão puder ser executada pelo gerenciador de fila, ela sempre será executada pelo gerenciador de filas De 7.0.1.5 em diante, o comportamento de conversão padrão é revertido para o mesmo que 6.0e todas as conversões são executadas pelo cliente JMS . A partir de 7.0.1.5 ou 7.0.1.4 com APAR IC72897, é possível configurar uma nova opção de destino, `WMQ_RECEIVE_CONVERSION`, para controlar onde a conversão será executada e `WMQ_RECEIVE_CCSID`, para configurar a página de códigos de destino; consulte Figura 12 na página 153.

[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 o receptor não JMS, o exemplo se aplica ao IBM MQ pré e pós V7.0. Com o IBM MQ 7.0, a saída de conversão pode ser chamada para um programa JMS de recebimento também

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, use 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 172.

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 bytes 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 depender de `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 UNIX. `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 172. para obter exemplos de converter mensagens trocadas com aplicativos não JMS.

Examples

Tabela de tipos de mensagens e tipos de conversões

Tabela 30. Tipos de mensagens e tipos de conversão

	Tipo de conversão			
Tipo de Mensagem	**text**	**Numérica**	**Outro**	**Nenhum**
`JMSObjectMessage`				`getObject` `setObject`

Tabela 30. Tipos de mensagens e tipos de conversão (continuação)

	Tipo de conversão			
Tipo de Mensagem	**text**	**Numérica**	**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. Fragmento de código de amqsget0.c

Enviando e recebendo texto em uma JMSBytesMessage

O código em Figura 14 na página 156 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 156. 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. Desde 7.0, JMS clientes que recebem mensagens também

usam a conversão de dados do gerenciador de filas. A partir da 7.0.1.5 ou 7.0.1.4 com a APAR IC72897, a conversão de dados do gerenciador de filas 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 160 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 160.

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 160.

Uma JMSTextMessage pode ser enviada em um corpo da mensagem de estilo MQ, sem um cabeçalho MQRFH2 do 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 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 uma JMSTextMessage, isso causará uma exceção, `ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to 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 162. Os detalhes são descritos em "Conversão e codificação de mensagem do clienteJMS" na página 164. Cada primitivo é identificado; consulte "O corpo da mensagem do JMS" na página 148.

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 161, 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 161 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 162. Os detalhes são descritos em "Conversão e tipos de mensagem doJMS" na página 157.

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 161.

Uma JMSBytesMessage enviada a um aplicativo MQ é geralmente enviada em um corpo da mensagem de estilo MQ, sem um cabeçalho MQRFH2 do JMS. 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 161, 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 161 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 161. 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 162 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 162 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 31. Tipos de mensagens e tipos de conversão

	Tipo de conversão			
Tipo de Mensagem	**text**	**Numérica**	**Outro**	**Nenhum**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

Tabela 31. Tipos de mensagens e tipos de conversão (continuação)				
	Tipo de conversão			
Tipo de Mensagem	**text**	**Numérica**	**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. Desde 7.0, JMS clientes que recebem mensagens também usam a conversão de dados do gerenciador de filas. A partir da 7.0.1.5 ou 7.0.1.4 com a APAR IC72897, a conversão de dados do gerenciador de filas 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 32 na página 164 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 32. Tipos de mensagens e tipos de conversão

	Tipo de conversão			
Tipo de Mensagem	**text**	**Numérica**	**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 de Java SE.

Tabela 32. Tipos de mensagens e tipos de conversão (continuação)				
	Tipo de conversão			
Tipo de Mensagem	**text**	**Numérica**	**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

text

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 167 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 168 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 168 ao fragmento de código incorreto em Figura 25 na página 166. 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`.

Numérica

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 168 mostra um aplicativo codificando um número no formato de destino e enviando-o em um `JMSStreamMessage`. Compare o exemplo em Figura 30 na página 168 ao exemplo em Figura 31 na página 168. 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 169.

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 167 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 UNIX, `UTF-8`. A troca entre o Windows e o UNIX requer a seleção explícita de uma página de códigos 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 JMSStreamMessage usando o conjunto de caracteres de destino

Examples

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.

[5] `SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage)` currently accepts only numeric character set identifiers.

```
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 169 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 169. 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. Desde 7.0, JMS clientes que recebem mensagens também usam a conversão de dados do gerenciador de filas. A partir da 7.0.1.5 ou 7.0.1.4 com a APAR IC72897, a conversão de dados do gerenciador de filas 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. Desde 7.0, JMS clientes que recebem mensagens também usam a conversão de dados do gerenciador de filas. A partir da 7.0.1.5 ou 7.0.1.4 com a APAR IC72897, a conversão de dados do gerenciador de filas 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 recurso de conversão do gerenciador de filas não esteve disponível para um aplicativo JMS recebendo uma mensagem até 7.0.

É possível usar a conversão do gerenciador de filas, antes de 7.0, com um aplicativo cliente do JMS que envia uma mensagem O cliente JMS constrói um registro formatado, configura os atributos `CodedCharacterSetId`, `Encoding` e `Format` correspondentes aos dados colocados na mensagem. Um aplicativo de recebimento não JMS lê a mensagem usando `GMO_CONVERT` e faz com que uma saída de conversão de dados escrita pelo usuário seja chamada. A saída de conversão de dados é uma biblioteca compartilhada que tem o nome configurado no campo `Format`.

Desde 7.0, o gerenciador de filas é capaz de converter mensagens que são enviadas para clientes do JMS Da 7.0.0.0 à 7.0.1.4, inclusive, a conversão do gerenciador de filas sempre á chamada para clientes JMS. Na 7.0.1.5 ou na 7.0.1.4 com o APAR IC72897 aplicado, a conversão do gerenciador de filas é controlada configurando a propriedade de destino `WMQ_RECEIVE_CONVERSION` para `WMQ_RECEIVE_CONVERSION_QMGR` ou `WMQ_RECEIVE_CONVERSION_CLIENT_MSG`. `WMQ_RECEIVE_CONVERSION_CLIENT_MSG` é a configuração padrão, correspondente ao comportamento de IBM WebSphere MQ 6.0, que não suportou a conversão de dados do gerenciador de filas para clientes do JMS 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 171,

```
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. Fragmento 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 989
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 132.

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 175, 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 173 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 175, 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 175.
- 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 175 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 174. Com as modificações, `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 175.
   - 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`.

Examples

```
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 do 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. Desde 7.0, JMS clientes que recebem mensagens também usam a conversão de dados do gerenciador de filas. A partir da 7.0.1.5 ou 7.0.1.4 com a APAR IC72897, a conversão de dados do gerenciador de filas é opcional.

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`.

Informações relacionadas

Utilitário para criação de código de saída de conversão

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 182 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 7.0, a partir de 7.0.1.5, é necessário ativar a conversão do gerenciador de filas Ele está desativado por padrão. Em 7.0, até 7.0.1.4 a conversão do gerenciador de filas é ativada por padrão, e essa linha de código 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 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;
```

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 javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Resultados

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 174 no “Trocando um registro formatado com um aplicativo não JMS” na página 172.
  - 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 Figura 39 na página 183.

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 definições do JMS a seguir em um campo 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 `TryMyRecord`.

Classes usadas para executar o exemplo

As classes listadas nas figuras de Figura 39 na página 183 a Figura 44 na página 187 também estão disponíveis em um arquivo compactado; faça download de jm25529_.zip ou de jm25529_.tar.gz.

```
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());
    }
}
```

Figura 39. TryMyRecord

```
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; }
}
```

Figura 40. RECORD

```
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; }
}
```

Figura 41. MyRecord

```
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); }
    }
```

Figura 42. EndPoint

```
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); }
}
```

Figura 43. MyProducer

```
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();  }
}
```

Figura 44. MyConsumer

Criando e configurando connection factories e destinos em um aplicativo IBM MQ classes for JMS

Um aplicativo IBM MQ classes for JMS pode criar connection factories e destinos, recuperando-os como objetos administrados a partir de uma Naming and Directory Interface (JNDI) Java, usando extensões do IBM JMS ou usando 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. 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 usando a JMS ferramenta de administraçãoou IBM MQ Explorer conforme descrito em Configurando objetos JMS usando 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.

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.

Informações relacionadas

Configurando os recursos do JMS

Usando JNDI para recuperar objetos administrados em um aplicativo JMS

Para recuperar objetos administrados de um namespace do Java Naming and Directory Interface (JNDI), um aplicativo JMS 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. O administrador pode usar a ferramenta de administração do IBM MQ JMS ou IBM MQ Explorer para criar e manter objetos administrados em um namespace do JNDI. Para obter mais informações, consulte Configurando connection factories e destinos em um namespace do JNDI.

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:

```
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 baseado em um servidor LDAP
- `file:/`*`directoryPath`* , para um serviço de diretório baseado 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 baseado em um servidor LDAP
- `com.sun.jndi.fscontext.RefFSContextFactory`, para um serviço de diretório baseado 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.

Informações relacionadas

Configurando objetos do JMS usando o IBM MQ Explorer

Configurando objetos do JMS usando a ferramenta de administração

Configurando os recursos do JMS no WebSphere Application Server

Usando as extensões IBM JMS

O IBM MQ classes for JMS contém um conjunto de extensões para a API do JMS chamado extensões do IBM JMS. Um aplicativo pode usar estas extensões para criar connection factories e os destinos dinamicamente no tempo de execução e para configurar as propriedades de objetos do IBM MQ classes for JMS. 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

Os pacotes podem ser localizados em `com.ibm.mqjms.jar` que está localizado em *MQ_INSTALLATION_PATH*`/java/lib`.

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 do IBM MQ classes for JMS
- 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 o IBM MQ classes for JMS

Com relação à criação de connection factories e destinos dinamicamente no tempo de execução e à configuração e obtenção de suas propriedades, as extensões do 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 contém as definições de constantes que um aplicativo pode usar ao configurar as propriedades de objetos do IBM MQ classes for JMS 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 supõem que as instruções de importação a seguir foram executadas:

```
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:

```
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 191.

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 204. 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 191.

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 204. 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 191.

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.

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 no pacote com.ibm.mq.jms também herdam os métodos da interface JmsPropertyContext. Um aplicativo pode, portanto, usar esses métodos para configurar as propriedades dos objetos MQConnectionFactory, MQQueue e MQTopic.

Quando um aplicativo cria um objeto do IBM MQ classes for JMS, quaisquer 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 tenta configurar uma propriedade para um valor que não é um valor válido para a propriedade, o IBM MQ classes for JMS lança uma exceção JMSException. Se um aplicativo tentar obter uma propriedade que não foi configurada, o comportamento será conforme descrito na especificação JMS. O IBM MQ classes for JMS lança uma exceção NumberFormatException para tipos de dados primitivos e retorna nulo para tipos de dados de referência.

Além das propriedades predefinidas de um objeto do IBM MQ classes for JMS, um aplicativo pode configurar suas próprias propriedades. Essas propriedades definidas pelo aplicativo são ignoradas pelo IBM MQ classes for JMS.

Para obter mais informações sobre as propriedades de objetos IBM MQ classes for JMS, 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 JmsPropertyContext para configurar ou obter a propriedade de um objeto do IBM MQ classes for JMS, o valor da propriedade pode ser implicitamente convertido 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 são mostradas em Tabela 33 na página 193.

Tabela 33. 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
caractere	Sequência
duplo	Sequência
float	double, String
int	long, String
grande	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 tenta converter uma sequência que não esteja no formato correto, o IBM MQ classes for JMS lançará uma exceção NumberFormatException.
- Se um aplicativo tentar uma conversão não suportada, o IBM MQ classes for JMS lançará uma exceção MessageFormatException.

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, o IBM MQ classes for JMS lançará uma exceção NumberFormatException.

- 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, o IBM MQ classes for JMS lançará uma exceção NumberFormatException.

- 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 191. 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 usa os métodos da interface JmsPropertyContext para configurar e obter as propriedades de objetos do IBM MQ classes for JMS, o aplicativo pode especificar os nomes e os valores de propriedades de qualquer uma das maneiras a seguir. 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, o IBM MQ classes for JMS armazena nomes e valores de propriedades como valores literais definidos na interface com.ibm.msg.client.wmq.WMQConstants. 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, o IBM MQ classes for JMS precisa converter os nomes e valores do formato de entrada especificado para o formato canônico. De forma semelhante, se um aplicativo obtém propriedades usando uma das outras duas maneiras de especificar nomes e valores de propriedades, o IBM MQ classes for JMS deverá converter os nomes do formato de entrada especificado para o formato canônico e converter os valores do formato canônico para o formato de saída necessário. Precisar executar essas conversões pode ter implicações no desempenho.

Nomes e valores de propriedades retornados por exceções, em arquivos de rastreio, ou no log do IBM MQ classes for JMS sempre estão 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 Map para acessar as propriedades de um objeto do IBM MQ classes for JMS.

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 197.

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.

Para uma conexão em tempo real com um broker, um aplicativo pode usar o seguinte código:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_DIRECT);
factory.setHostName("HOST2");
factory.setPort(1507);
```

Esse código supõe que o broker está sendo executado em um sistema com o nome do host HOST2 e atendendo no número de porta 1507.

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 204. Para obter uma maneira alternativa de configurar as propriedades de um objeto MQQueue, consulte “Configurando as propriedades de destinos” na página 199.

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 204. Para obter uma maneira alternativa de configurar as propriedades de um objeto MQTopic, consulte “Configurando as propriedades de destinos” na página 199.

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.

Construindo uma conexão em um aplicativo JMS

Para construir uma conexão, um aplicativo do 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();
```

Criando uma sessão em um aplicativo JMS

Para criar uma sessão, um aplicativo JMS usa o método createSession() de um objeto Connection.

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.

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 204.

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 204.

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, o 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 curinga +, #, *, e? em nomes de tópicos. Nomes de tópicos contendo esses caracteres podem causar resultados inesperados quando assinados. Consulte Usando sequências de tópicos.

***propertyName1*, *propertyName2*, ...**
Os nomes das propriedades do objeto Fila ou Tópico criados pelo aplicativo. Tabela 34 na página 205 lista os nomes de propriedade 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 Fila ou Tópico criados pelo aplicativo. Tabela 34 na página 205 lista os valores de propriedade 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 34 na página 205 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 34. 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.
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

Tabela 34. Nomes de propriedades e valores válidos para uso nos URIs de fila e tópico (continuação)		
Nome da Propriedade	**Descrição**	**Valores Válidos**
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 230.
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" na página 188
- "Usando as extensões IBM JMS" na página 189
- "Usando as extensões IBM MQ JMS" na página 196
- "Criando destinos em um aplicativo JMS" na página 202

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
- Mapear
- Object
- Fluxo
- text

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 do JMS" na página 128.

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 301.

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" na página 188
- "Usando as extensões IBM JMS" na página 189
- "Usando as extensões IBM MQ JMS" na página 196
- "Criando destinos em um aplicativo JMS" na página 202

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 128.

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 do IBM WebSphere MQ 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:

```
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 214.

V 9.0.0.2 V 9.0.2 *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.

Para o Continuous Delivery, no IBM MQ 9.0.2, quando uma mensagem é consumida de uma fila que contém um cabeçalho RFH2 com a pasta MQPS, o valor associado à chave Sud, caso exista, é incluído como uma propriedade de Sequência no objeto de Mensagem 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 javax.jms.Message.getStringProperty método(java.lang.String) para obter os dados do usuário de assinatura.

No IBM MQ 9.0.0 Fix Pack 2, a constante JMS_IBM_SUBSCRIPTION_USER_DATA também está disponível no Long Term Support.

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:

```
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 133

Informações relacionadas

Definindo uma assinatura administrativa

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 manipuladas 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. Nessa situação, o IBM MQ classes for JMS então move a mensagem para a fila especificada pelo atributo **BOQNAME** . Se a mensagem não puder ser colocada na fila de restauração, então, ela será movida para a fila de mensagens não entregues 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:

- Ao usar o IBM MQ classes for JMS para a IBM MQ 9.0.0 Fix Pack 5 e anterior, 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.
- V 9.0.0.6 Ao usar o IBM MQ classes for JMS para a IBM MQ 9.0.0 Fix Pack 6 e mais recente, o usuário que está executando o aplicativo de recebimento deve ter acesso de 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, deve-se conceder ao usuário que estiver executando o aplicativo as autoridades `put` e `passall`.

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, então o IBM MQ classes for JMS tentará 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, então o IBM MQ classes for JMS tentará 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 313
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 ser capaz de manipular exceções lançadas por chamadas API de um JMS ou entregues a um manipulador de exceções.

IBM MQ classes for JMS relata problemas de tempo de execução lançando as exceções. JMSException é a classe raiz para exceções lançadas pelos métodos JMS e capturar exceções JMSException fornece uma maneira genérica de manipular todas as exceções relacionados ao JMS.

Cada exceção JMSException encapsula as seguintes informações:

- Uma mensagem de exceção específica do provedor, que um aplicativo obtém chamando o método Throwable.getMessage().
- Um código de erro específico do provedor, que um aplicativo obtém chamando o método JMSException.getErrorCode().
- Uma exceção vinculada. Uma exceção lançada por uma chamada de API do JMS é geralmente o resultado de um problema de nível inferior, que é relatado por outra exceção vinculada a esta exceção. Um aplicativo obtém uma exceção vinculada chamando o método JMSException.getLinkedException() ou Throwable.getCause().

A maioria das exceções lançadas pelo IBM MQ classes for JMS são instâncias de subclasses do JMSException. Essas subclasses implementam a interface com.ibm.msg.client.jms.JmsExceptionDetail, que fornece as seguintes informações adicionais:

- Uma explicação da mensagem de exceção, que um aplicativo obtém chamando o método JmsExceptionDetail.getExplanation().

- Uma resposta do usuário recomendada para a exceção, que um aplicativo obtém chamando o método JmsExceptionDetail.getUserAction().
- As chaves para as inserções de mensagem na mensagem de exceção. Um aplicativo obtém um agente iterativo 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 que um aplicativo consiga acessar esse nome. Um aplicativo obtém a inserção de mensagem correspondente a uma chave especificada, chamando o método JmsExceptionDetail.getValue().

Todos os métodos na interface JmsExceptionDetail podem retornar 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 do javax.jms.InvalidDestinationException e implementa a interface com.ibm.msg.client.jms.JmsExceptionDetail.

Exceções Vinculadas

Uma exceção vinculada fornece informações adicionais sobre um problema de tempo de execução. Portanto, para cada exceção JMSException 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(). Para uma exceção JMSException, o método getLinkedException() delega na realidade ao 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
|
+- -->
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 sistema de mensagens do IBM MQ.
- A próxima exceção, uma instância de com.ibm.mq.jmqi.JmqiException, é lançada pela interface comum do Java para o MQI.
- A exceção final, uma instância de java.net.ConnectionException, é lançada pela biblioteca de classes do Java.

Para obter mais informações sobre a arquitetura em camadas do IBM MQ classes for JMS, consulte Classes do IBM MQ para arquitetura JMS.

Usando o código semelhante ao seguinte código, um aplicativo pode iterar por meio desta cadeia para extrair todas as informações apropriadas:

```
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 um aplicativo sempre deve verificar o tipo de cada exceção em uma cadeia porque o tipo da exceção pode variar e as exceções dos diferentes tipos encapsulam diferentes informações.

Obtendo informações específicas do IBM MQ sobre um problema

As instâncias de com.ibm.mq.MQException e com.ibm.mq.jmqi.JmqiException contêm informações específicas do IBM MQ sobre um problema.

Uma exceção MQException encapsula as seguintes informações:

- Um código de conclusão, que um aplicativo obtém chamando o método getCompCode()
- Um código de razão, que um aplicativo obtém chamando o método getReason()

Uma exceção JmqiException também encapsula um código de conclusão e um código de razão. No entanto,, além disso, uma exceção JmqiException encapsula as informações em uma mensagem AMQ*nnnn* ou CSQ*nnnn*, se houver uma associada à exceção. Ao chamar os métodos apropriados da exceção, um aplicativo pode obter diversos componentes desta mensagem, como a gravidade, a explicação e a resposta do usuário.

Para obter exemplos de como usar os métodos mencionados nesta seção, consulte o código de amostra em “Exceções Vinculadas” na página 218.

Fazendo upgrade de versões anteriores do IBM MQ classes for JMS

Comparado com versões anteriores do IBM MQ classes for JMS, a maioria dos códigos de erro e mensagens de exceção foi mudada em IBM WebSphere MQ 7.0. A razão para estas mudanças é que a partir do IBM WebSphere MQ 7.0, IBM MQ classes for JMS possui uma arquitetura em camadas e exceções que são lançadas de diferentes camadas no código.

Por exemplo, se um aplicativo tenta se conectar a um gerenciador de filas que não existe, uma versão anterior do IBM MQ classes for JMS emitiu uma exceção JMSException com as informações a seguir:

```
MQJMS2005: Failed to create MQQueueManager for 'localhost:QM_test'.
```

Esta exceção continha uma exceção MQException vinculada às seguintes informações:

```
MQJE001: Completion Code 2, Reason 2058
```

Por comparação nas mesmas circunstâncias, IBM MQ classes for JMS em IBM WebSphere MQ 7.0 lança uma exceção JMSException com as seguintes informações:

```
Message : JMSWMQ0018: Failed to connect to queue manager 'QM_test' with
connection mode 'Client' and host name 'localhost'.
Class : class com.ibm.msg.client.jms.DetailedJMSException
Error Code : JMSWMQ0018
Explanation : null
User Action : Check the queue manager is started and if running in client mode,
check there is a listener running. Please see the linked exception
for more information.
```

Esta exceção contém uma exceção MQException vinculada às seguintes informações:

```
Message : JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED')
          reason '2058' ('MQRC_Q_MGR_NAME_ERROR').
Class : class com.ibm.mq.MQException
Completion Code : 2
Reason Code : 2058
```

Se o seu aplicativo analisar ou testar mensagens de exceção retornadas pelo método Throwable.getMessage() ou códigos de erro retornados pelo método JMSException.getErrorCode() e você estiver fazendo upgrade de uma liberação antes do IBM WebSphere MQ 7.0, provavelmente precisará ser modificado para usar IBM MQ classes for JMS em IBM WebSphere MQ 7.0.

Listeners de Exceção

Um aplicativo pode registrar um listener de exceção com um objeto de Conexão. Subsequentemente, se ocorrer um problema que transforma a conexão inutilizável, o IBM MQ classes for JMS entregará uma exceção ao listener de exceção chamando seu método onException(). O aplicativo então 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.

No IBM MQ 8.0.0 Fix Pack 2, para manter o comportamento para aplicativos JMS atuais que configuram um MessageListener JMS e um ExceptionListener JMS e para assegurar que o IBM MQ classes for JMS seja consistente com a especificação JMS, o valor padrão para a propriedade ConnectionFactory de JMS ASYNC_EXCEPTIONS é mudado para ASYNC_EXCEPTIONS_CONNECTIONBROKEN para o IBM MQ classes for JMS. Como resultado, por padrão, somente as exceções correspondentes a códigos de erro de conexão quebrada são entregues para o ExceptionListener JMS de um aplicativo.

V 9.0.0.1 O APAR IT14820, incluído do IBM MQ 9.0.0 Fix Pack 1, atualiza o IBM MQ classes for JMS para que:

- Um ExceptionListener registrado por um aplicativo é chamado para qualquer exceção de conexão quebrada, independentemente se o aplicativo está usando consumidores de mensagem síncronos ou assíncronos.
- Um ExceptionListener registrado por um aplicativo será chamado se um soquete TCP/IP usado por uma sessão do JMS for quebrado.
- As exceções que não são de conexão quebrada (por exemplo, MQRC_GET_INHIBITED) que surgem durante a entrega de mensagens são entregues a um ExceptionListener do aplicativo quando o aplicativo está usando consumidores de mensagens assíncronas e o ConnectionFactory do JMS usado pelo aplicativo tem a propriedade ASYNC_EXCEPTIONS configurada com o valor ASYNC_EXCEPTIONS_ALL.

Nota: Um ExceptionListener somente será chamado uma vez para uma exceção de conexão quebrada, mesmo se duas conexões TCP/IP (uma usada por uma conexão do JMS e uma usada por uma sessão do JMS) estiveram quebradas.

Para qualquer outro tipo de problema, uma exceção JMSException é lançada pela chamada API atual do JMS.

Se um aplicativo não registrar um listener de exceção com um objeto Connection, quaisquer exceções que teriam sido entregues ao listener de exceção serão gravadas no IBM MQ classes for JMS para o log do JMS.

Informações relacionadas

Classes do IBM MQ para a arquitetura JMS

ASYNCEXCEPTION

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 para eles. IBM MQ classes for JMS fornece atributos de mensagem que permitem aos aplicativos JMS configurar campos MQMD e assim ativar aplicativos JMS para "conduzir" 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 224.

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 35. 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 36. 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	• **NO** 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.	• **False** • True	setMQMDWriteEnabled
WMQ_MQMD_READ_ENABLED	• **NO** 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.	• **False** • True	setMQMDReadEnabled

Tabela 36. 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 _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 37. 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	Report
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	Encoding
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 de caracteres	Format
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

Tabela 37. Propriedade nomes, descrições e tipos (continuação)

Propriedade	Descrição	Tipo de Java	Link para descrição completa
JMS_IBM_MQMD_ReplyToQ	Nome da fila de resposta	Sequência de caracteres	ReplyToQ
JMS_IBM_MQMD_ReplyToQMgr	Nome do gerenciador de filas de resposta	Sequência de caracteres	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	Identificador de usuários	Sequência de caracteres	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Símbolo de contabilidade	Objeto (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	dados do aplicativo relacionados à identidade	Sequência de caracteres	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 de caracteres	PutApplName
JMS_IBM_MQMD_PutDate	Data quando a mensagem foi colocada	Sequência de caracteres	PutDate
JMS_IBM_MQMD_PutTime	Hora quando a mensagem foi colocada	Sequência de caracteres	PutTime
JMS_IBM_MQMD_ApplOriginData	Os dados do aplicativo relacionados à origem	Sequência de caracteres	ApplOriginData
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 45 na página 228 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 45. MQMD e dados de mensagens inconsistentemente codificados

Um dos snippets de código em Figura 46 na página 228 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 46. 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 38. 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 39. 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 Camada JMS 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 264.

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, os CipherSuites são distintos dos CipherSpecs e, portanto, têm nomes diferentes. “CipherSpecs e CipherSuites TLS no IBM MQ classes for JMS” na página 235 contém uma tabela mapeamento os CipherSpecs suportados por IBM MQ para seus CipherSuites equivalentes como conhecidos pela 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 a Tabela de definição de canal de cliente (CCDT). Para obter mais informações, consulte “Usando uma tabela de definição de canal do cliente com o IBM MQ classes for JMS” na página 264.

Propriedade de objeto SSLFIPSREQUIRED
Se você precisar de uma conexão para usar um CipherSuite que seja suportado pelo provedor JSSE FIPS do IBM Java (IBM JSSEFIPS), configure a propriedade SSLFIPSREQUIRED do connection factory como YES.

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.

Informações relacionadas
Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI
Federal Information Processing Standards (FIPS) para UNIX, 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. Os CRLs são tipicamente hospedados em servidores LDAP. 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.

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 o 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.

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 mais alto de IBM JRE fornece o suporte apropriado

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* da versão que você está usando.

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	no

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLSv1.2	no

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLSv1.2	no

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	no

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na página 259	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLSv1	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLSv1	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	sim

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1	no

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLSv1.2	no

Tabela 40. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)				
CipherSpec	**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	TLSv1.2	no

Notes:

1. 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.

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

É possível configurar se seu aplicativo usa IBM Java CipherSuite padrão para os mapeamentos IBM MQ CipherSpec ou o Oracle CipherSuite para mapeamentos IBM MQ CipherSpec. Portanto, será possível usar CipherSuites TLS se seu aplicativo usar um IBM JRE ou um Oracle JRE. A Propriedade de sistema Java `com.ibm.mq.cfg.useIBMCipherMappings` controla quais mapeamentos são usados. A propriedade pode ser um dos valores a seguir:

true
Use os mapeamentos do IBM Java CipherSuite para IBM MQ CipherSpec.

Esse valor é o valor padrão.

false
Use os mapeamentos do Oracle CipherSuite para IBM MQ CipherSpec.

Para obter mais informações sobre como usar o IBM MQ Java e os Ciphers TLS, consulte o post do blog do MQdev MQ Java, Ciphers TLS, JREs & APARs nãoIBM IT06775, IV66840, IT09423, IT10837.

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 41. 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 260	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. 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.

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 de Canais do arquivo de configuração do cliente IBM MQ
2. O caminho de classe especificado pela propriedade de sistema **com.ibm.mq.exitClasspath** do Java. Observe que essa propriedade agora está descontinuada.
3. O diretório de saídas do IBM MQ, conforme mostrado em Tabela 42 na página 263. 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 42. O diretório de saídas do IBM MQ

Plataforma	Diretório
Linux UNIX UNIX 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 você não será capaz de influenciar a escolha do carregador de classes pai e, portanto, o carregador de classes deverá ser definido configurando a propriedade de sistema **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** do Java 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 classes do IBM MQ para aplicativos JMS no gerenciador de segurança Java.

As interfaces `MQSendExit`, `MQReceiveExite` `MQSecurityExit` fornecidas com versões anteriores a IBM WebSphere MQ 7.0 ainda são suportadas.. Se você usar saídas de canal que implementem essas interfaces, `com.ibm.mq.jar` deverá 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 965. Deve-se armazenar programas de saída de canal gravados em C ou C++ no diretório mostrado em Tabela 42 na página 263.

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 264.

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 264.

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 do IBM MQ classes for JMS” na página 87 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 DEFINE CHANNEL 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 260.

  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.

A reconexão de cliente automática está disponível apenas para aplicativos IBM MQ classes for JMS independentes. O uso de reconexão automática do cliente nos servidores de aplicativos do Java Platform, Enterprise Edition não é suportado.

Reconexão automática do cliente JMS usando CONNECTIONNAMELIST
Se um aplicativo IBM MQ classes for JMS independente usar um Connection Factory que tenha a propriedade CONNECTIONNAMELIST configurada, o aplicativo estará elegível para usar reconexão automática do cliente.

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 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 43. Valores possíveis da propriedade CLIENTRECCECTOPTIONS.

CLIENTRECONNECTOPTIONS	**Comportamento do IBM MQ classes for JMS**
QUALQUER	Usar o valor da propriedade CONNECTIONNAMELIST para abrir uma conexão com uma combinação de nome do host e porta e conectar-se 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 "*".
ASDEF	Usar o valor de DefRecon para determinar se a reconexão automática do cliente está disponível.
DISABLED	Não execute nenhuma reconexão automática do cliente e retorne uma JMSException para o aplicativo.
QMGR	Usar o valor da propriedade CONNECTIONNAMELIST para abrir uma conexão com uma combinação de nome do host e porta e conectar-se ao gerenciador de filas especificado pela propriedade QMANAGER.

Ao executar reconexão automática do cliente, o IBM MQ classes for JMS usa as informações na propriedade CONNNECTIONNAMELIST do 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.

Esse processo de reconexão automática do cliente continua até o IBM MQ classes for JMS reconectar-se com sucesso 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.

Conectando-se a gerenciadores de filas de várias instâncias usando CONNECTIONNAMELIST
A reconexão de cliente automática pode ser usada pelos aplicativos do IBM MQ classes for JMS que se conectam a um gerenciador de filas de várias instâncias.

Se a instância do gerenciador de filas que um aplicativo estiver usando ficar indisponível, o IBM MQ classes for JMS poderá automaticamente tentar se conectar à instância de espera em nome do aplicativo. O aplicativo é bloqueado enquanto a reconexão automática do cliente está ocorrendo e retoma quando o IBM MQ classes for JMS estabelece uma conexão com o gerenciador de filas de espera.

Para ativar a reconexão automática do cliente para um gerenciador de filas de várias instâncias, configure as propriedades a seguir no Connection Factory usado pelo aplicativo IBM MQ classes for JMS:

CHANNEL
O nome de um canal de conexão do servidor definido no gerenciador de filas.

QMANAGER

O nome do gerenciador de filas de várias instâncias.

CONNECTIONNAMELIST=host1(port1), host2(port2).

A primeira entrada da lista deve conter o nome do host e a porta usados para contatar a instância principal do gerenciador de filas. A segunda entrada deve conter o nome do host e a porta do sistema no qual a instância do gerenciador de filas em espera está localizada.

CLIENTRECONNECTOPTIONS=QMGR.

Isso assegura que o IBM MQ classes for JMS tente se reconectar a um gerenciador de filas que possua o mesmo nome que o gerenciador de filas ao qual o aplicativo foi conectado anteriormente.

Reconexão automática do cliente JMS com CCDTs
Se um aplicativo IBM MQ classes for JMS independente usar um Connection Factory que tenha a propriedade CCDTURL configurada, o aplicativo estará elegível para usar reconexão automática do cliente.

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 usando 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 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 ao se conectar a um gerenciador de filas.

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 usando 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. Possível para a propriedade CLIENTRECCECTOPTIONS	
CLIENTRECONNECTOPTIONS	**Comportamento do IBM MQ classes for JMS**
QUALQUER	Abrir a tabela de definição de canal de cliente especificada pela propriedade CCDTURL, escolher uma entrada na tabela e, em seguida, usar 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.
DISABLED	Não execute nenhuma reconexão automática do cliente e retorne uma JMSException para o aplicativo.
QMGR	Abrir a tabela de definição de canal do cliente especificada pela propriedade CCDTURL, localizar as entradas na tabela que correspondem ao nome do gerenciador de filas especificado pela propriedade QMANAGER e, em seguida, usar essas entradas para iniciar um canal de conexão do cliente para esse gerenciador de filas.

Ao executar reconexão automática do cliente, o IBM MQ classes for JMS usa 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.

Esse processo de reconexão automática do cliente continua até o IBM MQ classes for JMS reconectar com sucesso 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 evita 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 ter sido reconectado automaticamente depende de seu design. Leia os tópicos relacionados para entender como projetar aplicativos que podem usar a funcionalidade de reconexão automática.

Conectando-se a gerenciadores de filas de várias instâncias usando CCDTs
A reconexão do cliente automática pode ser usada pelos aplicativos IBM MQ classes for JMS que se conectam a um gerenciador de filas de várias instâncias.

Se a instância do gerenciador de filas que um aplicativo estiver usando ficar indisponível, o IBM MQ classes for JMS poderá automaticamente tentar se conectar à instância de espera em nome do aplicativo. O aplicativo será bloqueado enquanto a reconexão automática do cliente estiver ocorrendo e continuará quando o IBM MQ classes for JMS estabelecer uma conexão com o gerenciador de filas de espera.

Para ativar a reconexão automática do cliente para um gerenciador de filas de várias instâncias, configure as propriedades a seguir no Connection Factory usado pelo aplicativo IBM MQ classes for JMS:

QMANAGER=O nome do gerenciador de filas de várias instâncias.

CCDTURL=URI

O URI para uma tabela de definição de canal de cliente que contém duas entradas para o gerenciador de filas de várias instâncias; uma para a instância primária e uma para a instância em espera.

Usando a reconexão automática do cliente em ambientes Java SE e Java EE
Informações sobre como usar a reconexão automática de cliente do IBM MQ e os gerenciadores de filas de várias instâncias dentro de um ambiente do Java SE e do Java EE.

Gerenciadores de Filas de Várias Instâncias são instâncias do mesmo gerenciador de filas configurado em servidores diferentes. 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.

Informações relacionadas
Gerenciadores de Filas de Várias Instâncias

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**, disponível no IBM WebSphere MQ 7.0.1 Fix Pack 3 e mais recente, usa duas propriedades adicionais do connection factory, **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 de 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 as CCDTs são usadas pelo IBM MQ classes for JMS, consulte Usando uma tabela de definição de canal de cliente com classes do IBM MQ para JMS.

Usando a propriedade CLIENTRECONNECTOPTIONS para ativar a reconexão automática do cliente no 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 do IBM MQ (`mqclient.ini`).

DISABLED
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**.

ANY
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 do IBM MQ e o provedor de sistemas de mensagens do WebSphere Application Server IBM MQ fornecem suporte para a reconexão automática do cliente.

As opções disponíveis para fornecer a reconexão automática do cliente em um ambiente do Java EE são:

- Especificação de ativação
- Portas do listener do WebSphere Application Server
- Enterprise JavaBeans e aplicativos baseados na web
- Os aplicativos em execução dentro de contêineres do cliente

Nota: A reconexão automática do cliente com especificações de ativação usando a funcionalidade fornecida pelo 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.

Esse mecanismo é controlada 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 431.

O uso de reconexão automática do cliente dentro de um método `onMessage()` do aplicativo de 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.

Suporte para reconexão automática do cliente em ambientes Java EE
Em ambientes Java EE, como o WebSphere Application Server, o adaptador de recursos do IBM MQ e o provedor de sistemas de mensagens do WebSphere Application Server IBM MQ fornecem suporte para a reconexão automática do cliente. 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 da reconexão automática do cliente em ambientes Java EE				
	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 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 278.

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 **reconnectionRetryCount** do IBM MQ define o número de tentativas consecutivas de reconexão que devem ser feitas antes que uma especificação de ativação seja interrompida e requeira 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 puder se conectar ao gerenciador de filas, ela esperará o período de tempo especificado pela propriedade **reconnectionRetryInterval** do adaptador de recursos do IBM MQ antes de tentar novamente.

A propriedade do adaptador de recursos **reconnectionRetryCount** do IBM MQ define o número de tentativas consecutivas de reconexão que devem ser feitas antes que uma especificação de ativação seja interrompida e requeira 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.

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 278 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

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 274 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, o **CONNECTIONNAMELIST** será usado 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

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.

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 que são 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

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 280 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 usa a propriedade **CONNECTIONNAMELIST** da mesma maneira que 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

CCDTURL

Se o ambiente do Java EE fornecer um conjunto de conexões para JMS conexões, consulte "Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões" na página 280 para obter informações sobre como isso afeta o comportamento da propriedade **CCDTURL** .

Se o ambiente Java EE não fornecer um conjunto de conexões JMS, o aplicativo usa a propriedade **CCDTURL** da mesma maneira que 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.

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 o "Implementando a lógica de reconexão em um aplicativo Java EE" na página 281 para obter informações adicionais.

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

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 280 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 usa a propriedade **CONNECTIONNAMELIST** da mesma maneira que aplicativos Java SE .

CCDTURL

Se o ambiente do Java EE fornecer um conjunto de conexões para JMS conexões, consulte “Usando CONNECTIONNAMELIST ou CCDT em um conjunto de conexões” na página 280 para obter informações sobre como isso afeta o comportamento da propriedade **CCDTURL** .

Se o ambiente Java EE não fornecer um conjunto de conexões JMS, o aplicativo usa a propriedade **CCDTURL** da mesma maneira que 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.

A primeira vez que o connection factory é usado para criar uma conexão, o ambiente Java EE usa o **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 a partir do connection factory, o ambiente Java EE retornará a conexão 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 IBM MQ.

Devido à maneira como o **CONNECTIONNAMELIST** e os CCDTs são usados para criar conexões do JMS , também é possível ter um conjunto de conexões que contenha conexões com 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 do JMS é criada para esse sistema IBM MQ e é retornada ao 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
Os aplicativos Enterprise JavaBeans e baseados na web que quiserem se reconectar automaticamente, se um gerenciador de filas falhar, precisarão implementar a sua própria lógica de reconexão.

As opções a seguir fornecem mais informações sobre como é possível atingir isso:

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 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());
    }
}
```

em que `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 uma forma de conjunto de conexões fora do Java EE ajuda a reduzir a carga geral resultante, por exemplo, de alguns aplicativos independentes usando estruturas ou sendo implementados em ambientes de nuvem e também de um número maior de conexões do cliente no `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
Java EE servidores de aplicativos fornecem a funcionalidade de definição do conjunto de conexões que pode ser usada pelos 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 47 na página 285 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.

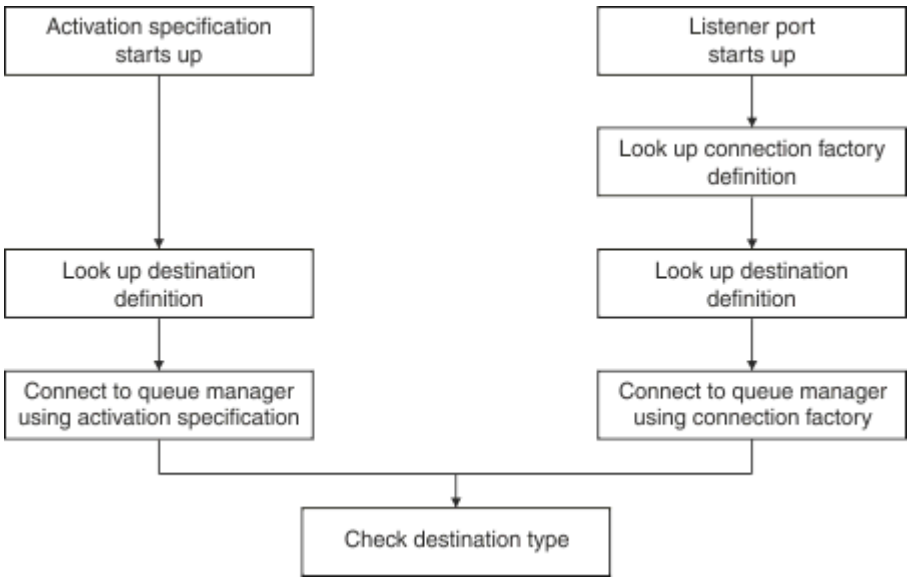


Figura 47. Modo normal

Ao usar o provedor de sistemas de mensagens do IBM MQ, os aplicativos que executam sistemas de mensagens não enviadas (como enterprise Java beans e servlets) e o componente de 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 289 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 292 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 293 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 48 na página 286 mostra o relacionamento entre esses objetos.

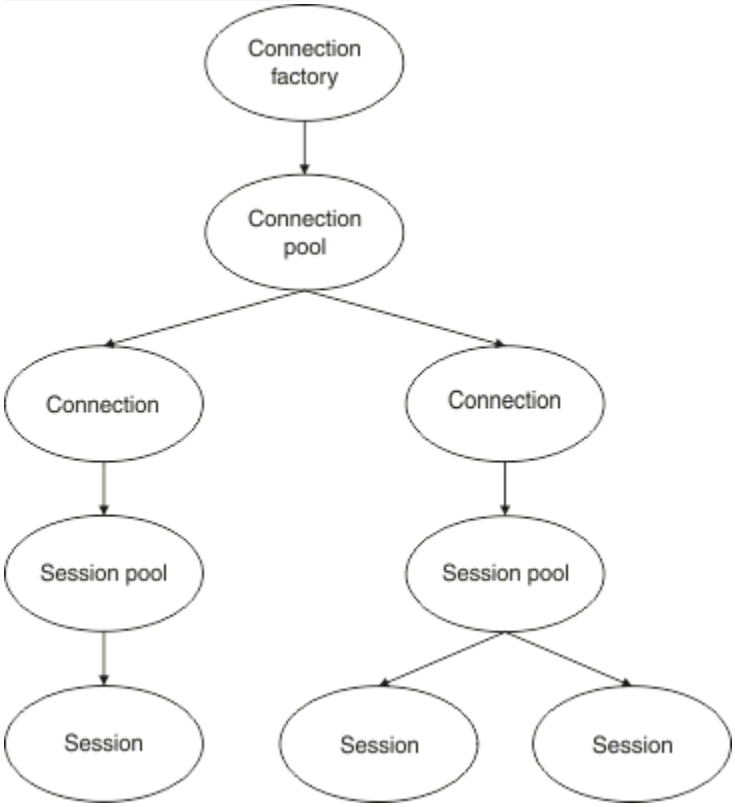


Figura 48. Conjuntos de conexões e conjuntos de sessões

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 291 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 293 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.

Figura 49 na página 289 mostra como o conjunto de conexões funciona para WebSphere Application Server 7.5 e 8.0.

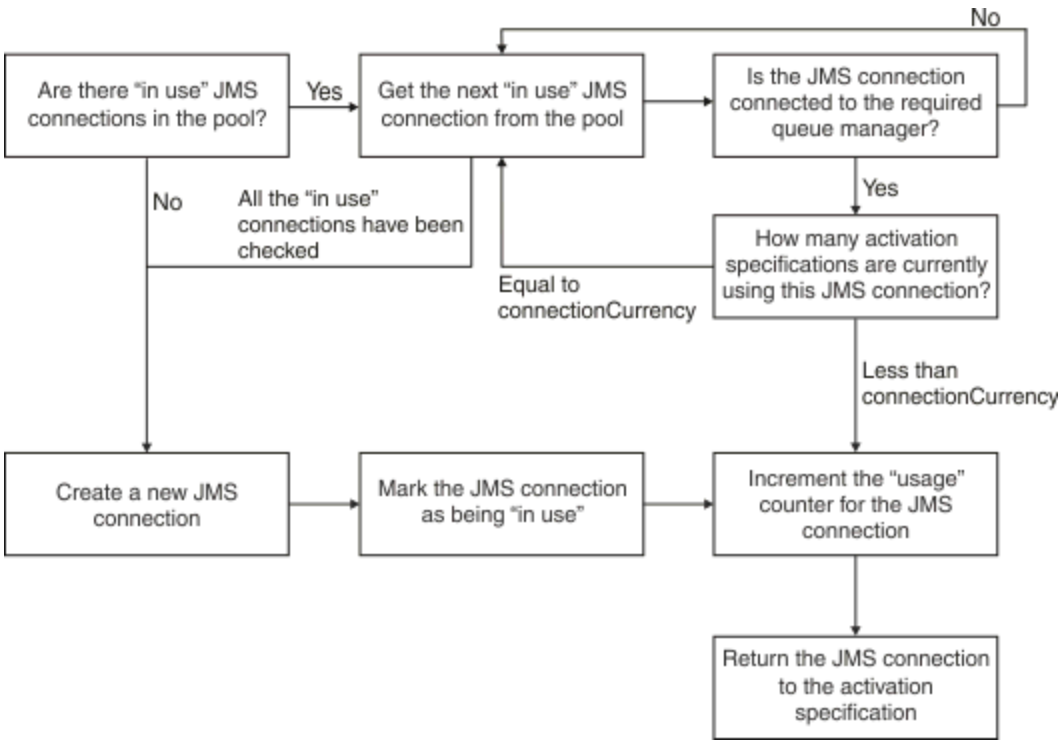


Figura 49. WebSphere Application Server 7.5 e 8.0 -como o conjunto de conexões funciona..

Figura 50 na página 290 mostra como o conjunto de conexões funciona para o WebSphere Application Server 8.5

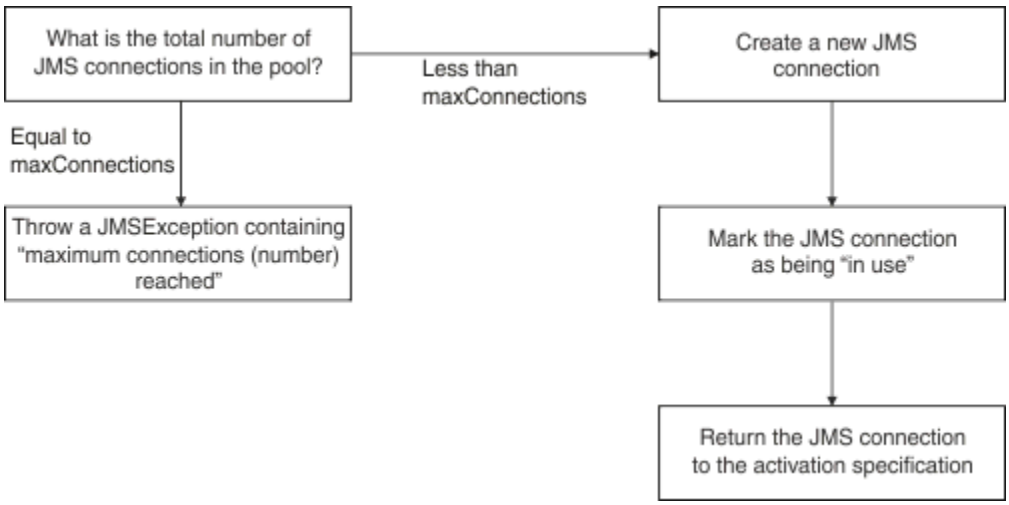


Figura 50. WebSphere Application Server 8.5 -como o conjunto de conexões funciona

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 é iniciada, a porta tenta 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 IBM MQ a partir 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 *sessão* do JMS, que é criada por meio de uma *conexão* do JMS. É a *sessão* do JMS que é inscrita nas transações e não a *conexão* do JMS.

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 JMS conexões de dentro de EJBs, assegure que qualquer trabalho JMS seja explicitamente confirmado na sessão JMS , antes que o método EJB que executa as operações JMS saia.

Exemplos de encadeamento de manutenção do conjunto
Usando o exemplo do Enterprise Java Bean (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 291 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 mostra como o encadeamento de manutenção usa as propriedades **Reap time** e **Unused timeout** para evitar conexões antigas, quando a propriedade **Aged timeout** está 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 os conteúdos do conjunto livre por 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 290 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

Suponhamos que o primeiro listener esteja parado e sua conexão `c1` seja colocada no conjunto livre. 180 segundos mais tarde, o encadeamento de manutenção do conjunto acorda, varre os conteúdos do conjunto livre por `jms / CF1` e descobre que `c1` está no conjunto livre por mais tempo do que o valor da propriedade **Unused timeout** para o 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 está no conjunto de conexões livres por mais tempo que o valor da propriedade **Unused timeout**. No entanto, uma vez que o fechamento de `c2` deixaria o conjunto de conexões livres com menos que o número configurado de Conexões mínimas, 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 do JMS estabelecerá uma conversa (também conhecida como **hconn**) com 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 desse factory estabelecerá uma nova conversa (também conhecida como**hconn**) para 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 no IBM MQ classes for JMS.

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 foram configurados para se conectarem 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).

Informações relacionadas

Usando o IBM MQ classes for JMS

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
- Session
- 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.

Monitoring

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 Java EE, haverá um grande número de opções, incluindo o:

- 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 o IBM MQ classes for JMS" na página 75
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

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 do 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 a instância do canal mesmo.

Há um relacionamento um-a-um entre as instâncias do canal e conexões TCP/IP. Uma conexão TCP/IP é criada para cada instância do canal.

Se um canal for definido com o parâmetro **SHARECNV** configurado para um valor maior que 1, esse número de conversações pode compartilhar uma instância do canal. Para ativar um connection factory ou uma especificação de ativação para usar esta função, configure a propriedade **SHARECONVALLOWED** como YES.

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 compartilhará uma conexão já estabelecida.

Calculando o número de instâncias do canal

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 são criadas pelo aplicativo, *jms_sessions* é o número de sessões JMS que são criadas pelo aplicativo e *SHARECNV_value* é o valor da propriedade **SHARECNV** no canal que é usado pela especificação de ativação.

Examples

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 para 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 a propriedade **maxPoolDepth** configurada para 10. O canal que a especificação de ativação está configurada para usar tem a propriedade **SHARECNV** configurada como 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** 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 conversa 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 482

O IBM WebSphere MQ 7.0 introduziu um novo recurso chamado "compartilhando conversas". Usando esse recurso, várias conversas podem compartilhar instâncias do canal MQI, também 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 do JMS PUTASYNCALLOWED (nome abreviado - PAALD).

  PUTASYNCALLOWED controla se os aplicativos JMS podem colocar mensagens assincronamente, se a fila ou o tópico subjacente que o destino do JMS representa, permite essa opção.
- A propriedade de fila ou de tópico DEFPRESP do IBM MQ (Tipo de resposta de colocação padrão).

  DEFPRESP especifica se os aplicativos que colocam mensagens na fila ou publicam mensagens para o tópico, podem usar a funcionalidade de colocação assíncrona.

A tabela a seguir mostra os valores possíveis das propriedades PUTASYNCALLOWED e DEFPRESP e quais valores são necessários para que a funcionalidade de colocação assíncrona seja ativada:

Tabela 46. As propriedades PUTASYNCALLOWED e DEFPRESP determinam se as mensagens são colocadas de forma assíncrona.

Propriedade de destino doJMS	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**PUTASYNCALLOWED = AS_DEST or AS_Q_DEF or AS_T_DEF**
Propriedade de filas doIBM MQ			
DEFPRESP=SYNC	Funcionalidade de colocação assíncrona não ativada	Funcionalidade de colocação assíncrona ativada	Funcionalidade de colocação assíncrona não ativada
DEFPRESP=ASYNC	Funcionalidade de colocação assíncrona não ativada	Funcionalidade de colocação assíncrona ativada	Funcionalidade de colocação assíncrona ativada

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.

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:

- O READAHEADALLOWED da propriedade de destino do JMS (nome abreviado - RAALD). READAHEADALLOWED controla se os aplicativos JMS podem 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 quais valores são necessários para a funcionalidade de leitura antecipada a ser ativada:

Tabela 47. As propriedades READAHEADALLOWED e DEFREADA que determinam se a leitura antecipada será usada ao receber ou procurar mensagens não persistentes fora de uma transação.

Propriedade de destino doIBM MQ	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST or AS_Q_DEF or AS_T_DEF**
Propriedade de filas doIBM MQ			
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.

Usando a funcionalidade do JMS 2.0
JMS 2.0 apresenta várias novas áreas de funcionalidade para o IBM MQ classes for JMS.

Quando você estiver desenvolvendo um aplicativo JMS para IBM MQ 8.0 ou mais recente, talvez seja necessário considerar o impacto dessa funcionalidade em seu gerenciador de filas.

Informações relacionadas
Interfaces de linguagem do IBM MQ Java

Atraso de entrega doJMS 2.0
Com o JMS 2.0, será 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.

No IBM MQ 8.0 e mais recente, 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. Estas restrições se aplicam igualmente a `MessageProducers` e `JMSProducers`, mas as `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 uma 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.

Authorization

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 No 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 No IBM MQ for z/OS, SYSTEM.DDELAY.LOCAL.QUEUE é usada como uma fila temporária para mensagens enviadas com o atraso de entrega para as filas local e compartilhada. 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.

Atraso de entrega não é protegido pelo OPMODE. Se você usar atraso de entrega com um gerenciador de filas do IBM MQ 8.0 e, em seguida, migrar de volta para uma liberação anterior quaisquer mensagens no SYSTEM.DDELAY.LOCAL.QUEUE são abandonadas, a menos que você trate manualmente com elas.

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

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 outra razão, ele poderá ser reiniciado alterando o atributo PUT da fila temporária a partir de ENABLED para DISABLED e de volta para ENABLED novamente. Se você precisar parar o processador de atraso de entrega por qualquer razão, configure o atributo PUT da fila temporária para DISABLED.

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 em relação a especificação JMS 2.0.

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
No IBM MQ 8.0 ou mais recente, há dois métodos para dar 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** para `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

A especificação do JMS 2.0 apresenta assinaturas compartilhadas, que permite que 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 estão ativadas pela chamada relevante para a API do JMS 2.0.

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 do JMS 2.0 não define nenhuma maneira padrão de conduzir um MDB a partir de um sharedSubscription, então o IBM MQ 8.0 ou mais recente fornece a propriedade de especificação de ativação sharedSubscription para este propósito. Para obter mais informações sobre essa propriedade,

consulte “Configurando o adaptador de recursos para comunicação de entrada” na página 433 e “Exemplos de como definir a propriedade sharedSubscription” na página 450.

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 requisito para criar quaisquer objetos separadamente no lado do gerenciador de filas além da configuração usual do JMS, quaisquer objetos necessários são criados dinamicamente.

Decidindo entre assinaturas compartilhadas ou clonadas

Quando você estiver determinando se usar assinaturas compartilhadas ou clonadas, considere os benefícios de ambas. Sempre que possível, use assinaturas compartilhadas, pois elas têm um comportamento definido pela especificação, em vez de uma extensão específica do IBM MQ.

A tabela a seguir contém algumas das questões a serem consideradas quando estiver decidindo entre assinaturas compartilhadas e clonadas:

Tabela 48. Contraprestações quando você estiver escolhendo entre assinaturas compartilhadas e clonadas

Assinaturas compartilhadas	**Assinaturas clonadas**
As assinaturas compartilhadas são uma parte padrão da especificação do JMS 2.0.	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.

Tentativas de alterar a capacidade de compartilhamento de uma assinatura existente

V 9.0.0.1

Se uma assinatura for criada como compartilhada, ela não poderá ser mudada posteriormente para não compartilhada ou vice-versa.

No IBM MQ 9.0.0 Fix Pack 1, o gerenciador de filas do IBM MQ foi atualizado de modo que se um aplicativo JMS 2.0 criar uma assinatura durável não compartilhada e outro aplicativo não JMS 2.0 tentar continuar a assinatura, o código de razão a seguir será retornado:

```
2432 (MQRC_SUB_ALREADY_EXISTS
```

Referências relacionadas

“Exemplos de como definir a propriedade sharedSubscription” na página 450
É possível definir a propriedade sharedSubscription de uma especificação de ativação dentro de um arquivo `server.xml` do WebSphere Application Server Liberty. Como alternativa, é possível definir a propriedade dentro de um bean acionado por mensagens (MDB) usando anotações.

Informações relacionadas

Assinantes e assinaturas

Durabilidade da assinatura
Usando assinaturas compartilhadas do JMS 2.0
CLONESUPP

Propriedade SupportMQExtensions
A especificação JMS 2.0 introduz mudanças na forma como determinados comportamentos funcionam. O IBM MQ 8.0 e posterior inclui a propriedade `com.ibm.mq.jms.SupportMQExtensions`, que pode ser definida como *TRUE* para reverter esses comportamentos modificados de volta para implementações anteriores.

Três áreas de funcionalidade são revertidas configurando `SupportMQExtensions` para *True*:

Prioridade da mensagem
Uma prioridade por ser designada às mensagens, *0 – 9*. Antes do JMS 2.0, as mensagens também poderiam usar o valor *-1*, indicando que uma prioridade padrão da fila é usada. O JMS 2.0 não permite que uma prioridade de mensagem de *-1* seja definida. 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.

A propriedade `com.ibm.mq.jms.SupportMQExtensions` é uma propriedade booleana contida em `com.ibm.mqjms.jar`. 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 do IBM MQ classes for JMS” na página 87
Um arquivo de configuração do IBM MQ classes for JMS especifica propriedades que são usadas para configurar o IBM MQ classes for JMS.

Referências relacionadas

“Propriedades usadas para configurar o comportamento do cliente JMS” na página 93
Use essas propriedades para configurar o comportamento do cliente JMS.

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 418.

IBM MQ classes for JMS suporta os Application Server Facilities (ASF) especificados no *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 310
- “Princípios gerais para o sistema de mensagens de publicação/assinatura usando o ASF” na página 311
- “Removendo mensagens da fila em ASF” na página 312
- Manipulando mensagens suspeitas no ASF. Consulte o “Manipulando mensagens suspeitas no IBM MQ classes for JMS” na página 214.

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 312). 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)`
- PutApplName is "`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 314. 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 214.

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 214
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 314 e "Códigos de razão e de feedback em ASF" na página 314.

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 conclusão de 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 51 na página 316 resume os princípios da funcionalidade ServerSessionPool e ServerSession.

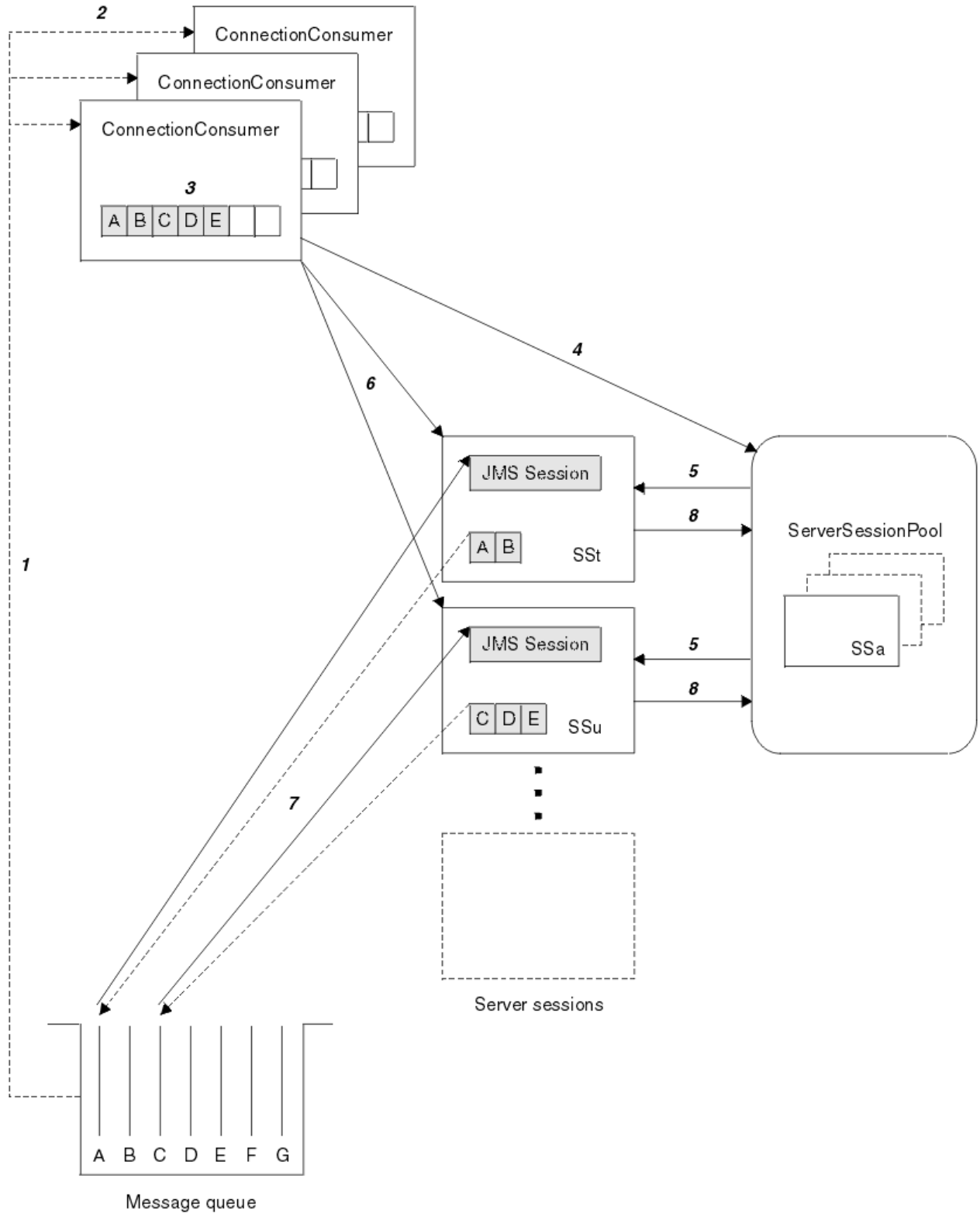


Figura 51. Funcionalidade ServerSessionPool e ServerSession

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 Usando o IBM MQ classes for JMS em um servidor JVM CICS OSGi

O IBM MQ 8.0 incluiu suporte para usar o IBM MQ classes for JMS em determinadas versões do servidor CICS Open Services Gateway initiative (OSGi) Java Virtual Machine (JVM).

Atenção: Verifique os requisitos do sistema para o sistema do CICS usado por sua empresa. Consulte Requisitos detalhados do sistema para o CICS Transaction Server para obter informações detalhadas.

Este tópico é uma introdução sobre como configurar o IBM MQ classes for JMS em um ambiente do servidor JVM.

Consulte Usando IBM MQ classes for JMS em um servidor JVM OSGi na documentaçãoCICS para obter detalhes sobre como configurar seu sistema.

Restrições gerais

As restrições a seguir se aplicam ao usar o IBM MQ classes for JMS em um servidor JVM CICS OSGi:

- Conexões de modo cliente não são suportadas.
- As conexões são suportadas apenas para IBM WebSphere MQ 7.1ou IBM MQ 8.0 ou gerenciadores de filas posteriores. O atributo **PROVIDERVERSION** no connection factory deve ser não especificado ou um valor maior ou igual a sete.
- O uso de qualquer um dos connection factories XA, por exemplo `com.ibm.mq.jms.MQXAConnectionFactory`, não é suportado.
- O uso de IBM MQ classes for JMS em um servidor JVM CICS OSGi é suportado somente no CICS 5.2 ou posterior. Se você está usando o CICS 5.2, deve-se aplicar o APAR PI32151.

V 9.0.1 Antes de IBM MQ 9.0.1, o uso do IBM MQ classes for JMS em um ambiente do servidor JVM Liberty não é suportado

Informações relacionadas

Configurando a propriedade JMS **PROVIDERVERSION**

z/OS V 9.0.1 Usando o IBM MQ classes for JMS em um servidor JVM (Java virtual machine) do CICS Liberty

No IBM MQ 9.0.1, programas Java em execução em um servidor JVM do CICS Liberty podem usar o IBM MQ classes for JMS para acessar o IBM MQ.

Deve-se estar usando o IBM MQ 9.0.1 ou versão mais recente do adaptador de recursos do IBM MQ que pode ser obtido no Fix Central (veja “Instalando o adaptador de recursos no Liberty” na página 426).

Há dois tipos de JVMs do Perfil do Liberty disponíveis no CICS 5.3 e mais recente. Os tipos de conexão possíveis para o IBM MQ são restritos, como a seguir:

CICS Liberty Standard

- O adaptador de recursos do IBM MQ pode se conectar a qualquer versão de serviço do IBM MQ no modo CLIENT

- O adaptador de recursos do IBM MQ pode se conectar a qualquer versão de serviço do IBM MQ for z/OS no modo BINDINGS quando não há nenhuma conexão do CICS (definição de recurso CICS MQCONN ativa) com o mesmo gerenciador de filas na mesma região CICS.

CICS Liberty integrado

- O adaptador de recursos do IBM MQ pode se conectar a qualquer versão de serviço do IBM MQ no modo CLIENT.
- A conexão no modo BINDINGS não é suportada.

Para obter mais informações, consulte Usando o IBM MQ classes for JMS em um servidor Liberty JVM na documentação do CICS para obter detalhes sobre como instalar e configurar seu sistema.

Usando o IBM MQ classes for JMS no IMS

O IBM MQ 8.0.0 Fix Pack 4 incluiu suporte para usar o IBM MQ classes for JMS no IMS versões 13 e mais recente.

Verifique os requisitos do sistema para o sistema IMS usado por sua empresa. Consulte Informações Gerais de Planejamento para IMS 13 para obter informações adicionais

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 322 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.

Regiões dependentes do IMS suportadas

Os tipos de regiões dependentes a seguir são suportados:

- MPR
- BMP
- IFP
- JMP (Apenas as Java virtual machines (JVMs) de 31 bits, as JVMs de 64 bits não são suportadas)
- JBP (Apenas as JVMs de 31 bits, as JVMs de 64 bits não são suportadas)

Exceto quando especificamente mencionado nos tópicos a seguir, o IBM MQ classes for JMS se comporta da mesma forma em todos os tipos de regiões.

Java Virtual Machines suportadas

O IBM MQ classes for JMS requer o Java Platform, Standard Edition 7 (Java SE 7) ou mais recente.

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.
- Conexões são suportadas somente para os gerenciadores de filas do IBM MQ 8.0 que estão usando o modo `Normal` ou `Version 8` do provedor de sistema de mensagens do IBM MQ.

  O atributo **PROVIDERVERSION** no connection factory deve ser não especificado ou um valor maior ou igual a sete.
- O uso de qualquer um dos connection factories XA, por exemplo `com.ibm.mq.jms.MQXAConnectionFactory`, não é suportado.

Informações relacionadas

Definindo IBM MQ para IMS

Configurando o adaptador do IMS para uso com o IBM MQ classes for JMS

IBM MQ classes for JMS usam o mesmo adaptador do IBM MQ-IMS que o 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 do 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 IBM MQ classes for JMS bibliotecas nativas:

```
LIBPATH=>
/java/java71_31/J7.1/bin/j9vm:>
/java/java71_31/J7.1/bin:>
/ims13/dbdc/imsjava/classic/lib:>
/ims13/dbdc/imsjava/lib:>
/mqm/V8R0M0/java/lib
```

2. Inclua o IBM MQ classes for JMS no caminho de classe da JVM, usado pela 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:

```
-Djava.class.path=/ims13/dbdc/imsjava/imsutm.jar:/ims13/dbdc/imsjava/imsudb.jar:
/mqm/V8R0M0/java/lib/com.ibm.mq.allclient.jarLIBPATH_SUFFIX=MQ_INSTALLATION_PATH
```

   Nota: Embora haja muitos arquivos jar diferentes disponíveis no diretório que contém o IBM MQ classes for JMS, você precisa somente do arquivo `com.ibm.mq.allclient.jar`.

3. Pare e reinicie qualquer região dependente do IMS que estará usando o IBM MQ classes for JMS.

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 em um aplicativo IBM MQ classes for JMS” na página 187 para obter detalhes.

Observe que essas três abordagens são todas válidas em um ambiente do IMS.

Informações relacionadas

Configurando o adaptador do IMS

Definindo 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 só pode ser concluída chamando os métodos de confirmação ou rollback em uma instância do objeto `com.ibm.ims.dli.tm.Transaction` ou pela tarefa IMS terminando normalmente nesse caso, a UOW é 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 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.

Consulte “Pontos de sincronização em aplicativos IMS” na página 63 para obter mais informações.

Para ilustrar esse ponto, considere o código a seguir em execução em um ambiente do JMP. A segunda chamada para `mp.send()` resulta em um `JMSException` como o código `messageQueue.getUnique(inputMessage)` resulta em todas as conexões IBM MQ abertas e manipulações de objetos sendo encerradas

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 encerrada 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

Problemas podem ocorrer ao intercalar código Java e código nativo que usam a funcionalidade do IBM MQ e quando a conexão com o IBM MQ não for fechada antes de sair do código nativo ou Java.

Por exemplo, no pseudocódigo a seguir, uma manipulação de conexões para um gerenciador de filas é estabelecida originalmente 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 manipulações de conexões do IBM MQ entre o código nativo e Java (por exemplo, o exemplo anterior funcionaria se não houvesse uma chamada MQDISC), em geral, a melhor prática é fechar qualquer manipulação de conexões antes de mudar de código Java para 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. A tentativa de criar um contexto JMS , ao especificar um nome de usuário e senha, resulta em um `JMSRuntimeException` sendo lançado.

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.

Informações relacionadas

Configurando a Segurança em z/OS

Restrições de API do JMS

A partir de uma JMS perspectiva de especificação, IBM MQ classes for JMS trate IMS como um Java servidor de aplicativos compatível com EE, que sempre tem 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 319.

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:

Os IBM MQ classes for Java são estabilizados funcionalmente no nível enviado em IBM MQ 8.0 Para obter mais informações, veja Estabilização de classes do IBM MQ para Java.

O IBM MQ classes for Java não é suportado no IMS.

O IBM MQ classes for Java não é suportado no WebSphere Application Server 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 duas APIs alternativas que os aplicativos Java podem usar para acessar os recursos do IBM MQ. A outra API é IBM MQ classes for JMS.

No IBM MQ 8.0, o IBM MQ classes for Java é construído com o Java 7.

O ambiente de tempo de execução do Java 7 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 348.

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 333 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 348.

Informações relacionadas

Interfaces de linguagem do IBM MQ Java

Rastreando aplicativos do IBM MQ classes for Java

Resolução de problemas do Java e do JMS
Usando o IBM MQ classes for 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 JMS. Recursos 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. Para obter informações adicionais, consulte “Por que devo usar o IBM MQ classes for JMS?” na página 76.

Nota: O IBM MQ classes for Java é funcionalmente estabilizado no nível enviado no IBM MQ 8.0. Para obter mais informações, veja Estabilização de classes do IBM MQ para 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, UNIX and Linux System Services (USS)

Se você precisar de conexões TLS para usar módulos criptográficos que tenham sido certificados por FIPS 140-2, será necessário o provedor JSSE FIPS do IBM Java (IBMJSSEFIPS) Cada IBM JDK e JRE na versão 1.4.2 ou posterior 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. Os 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. Por exemplo, o WebSphere Application Server 7.0, que envia um nível de RA do IBM MQ de 7.0.1, pode ser instalado na mesma máquina que um gerenciador de filas do IBM WebSphere MQ 6.

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 de IBM MQ 8.0, as conversões de sequência em IBM MQ classes for Java eram 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.

Configurar

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`.

Atualmente, a decodificação assume o comportamento padrão para `CharsetDecoders`, que relata erros lançando 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 123
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.

Instalação e configuração do 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:

- Multi Instalando o IBM MQ
- z/OS Instalando o IBM MQ for z/OS produto

O IBM MQ classes for Java está contido nos arquivos Java archive (JAR), `com.ibm.mq.jar` e `com.ibm.mq.jmqi.jar`.

O suporte para cabeçalhos de mensagem padrão, como Programmable Command Format (PCF), está contido no arquivo JAR `com.ibm.mq.headers.jar`.

O suporte para Programmable Command Format (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 325. Para obter mais informações, consulte Usando Interfaces Java do WebSphere MQ em Ambientes J2EE/JEE..

Importante: Além dos arquivos JAR relocalizáveis descritos neste tópico, copiar os arquivos JAR do IBM MQ classes for Java ou as bibliotecas nativas para outras máquinas ou para um local diferente em uma máquina na qual o IBM MQ classes for Java foi instalado não é suportado. Além disso, incluir o arquivo `com.ibm.mq.allclient.jar` ou o IBM MQ classes for Java nos archives do aplicativo (como archives de aplicativos corporativos ou arquivos EAR) não é suportado.

V 9.0.5 V 9.0.0.3 O arquivo `JSON4J.jar` e o pacote `com.ibm.msg.client.mqlight` não são necessários para IBM MQ classes for Java e IBM MQ classes for JMS. No IBM MQ 9.0.0 Fix Pack 3 e IBM MQ 9.0.5, as mudanças a seguir serão, portanto, feitas no arquivo `com.ibm.mq.allclient.jar`:

- A referência ao arquivo `JSON4J.jar` é removida da instrução de caminho da classe no arquivo manifest para o arquivo `com.ibm.mq.allclient.jar`.
- O pacote `com.ibm.msg.client.mqlight` não é mais incluído no arquivo `com.ibm.mq.allclient.jar`.

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:

- `com.ibm.mq.allclient.jar`
- `com.ibm.mq.traceControl.jar`
- O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS

O provedor de segurança Bouncy Castle e os arquivos JAR de suporte do CMS são necessários. Para obter mais informações, consulte Suporte para JREs não IBM. São necessários os seguintes arquivos JAR:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.0.0.12 `bcutil-jdk15on.jar`

O arquivo `com.ibm.mq.allclient.jar` contém o IBM MQ classes for JMS, o IBM MQ classes for Java e as Classes PCF e Cabeçalhos. Se este arquivo for movido 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 desse arquivo seja anunciado ao Suporte IBM se você estiver obtendo uma correção temporária.

Para determinar a versão do arquivo `com.ibm.mq.allclient.jar`, use o comando:

```
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:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.0.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.0.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C: /Arquivos de Programas /IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

O arquivo `com.ibm.mq.traceControl.jar` é usado para controlar dinamicamente o rastreio para aplicativos IBM MQ classes for JMS. Para obter mais informações, consulte Controlando o rastreio em um processo em execução usando classes do IBM MQ for Java e classes do IBM MQ for JMS.

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 330 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	*MQ_INSTALLATION_PATH*/java/lib
Solaris Linux HP-UX Solaris Linux HP-UX HP-UX, Linux e Solaris	*MQ_INSTALLATION_PATH*/java/lib
IBM i	/QIBM/ProdData/mqm/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib

Tabela 49. Diretórios de instalação do IBM MQ classes for Java (continuação)	
Plataforma	**Diretório**
z/OS 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 331 mostra onde os aplicativos de amostra são instalados. As amostras do 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 AIX AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Solaris Linux HP-UX Solaris Linux HP-UX HP-UX, Linux e Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i IBM i IBM i	/QIBM/ProdData/mqm/java/samples
Windows Windows Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS z/OS 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

O IBM MQ classes for JMS requer um Java Runtime Environment (JRE) Java 7 (ou superior). Um JRE adequado é instalado com o IBM MQ. O Tabela 51 na página 331 mostra onde este JRE é instalado. Para executar programas Java como as amostras fornecidas, usando esse JRE, chame explicitamente *JRE_LOCATION*`/bin/java` ou inclua *JRE_LOCATION*`/bin` no ambiente PATH (ou equivalente) para sua plataforma, em que *JRE_LOCATION* é o diretório fornecido em Tabela 51 na página 331.

Tabela 51. diretórios do JRE	
Plataforma	**Diretório**
AIX AIX AIX	*MQ_INSTALLATION_PATH*/java/jre
Solaris Linux HP-UX Solaris Linux HP-UX HP-UX, Linux e Solaris	*MQ_INSTALLATION_PATH*/java/jre
IBM i IBM i IBM i	/QIBM/ProdData/mqm/java/jre
Windows Windows Windows	*MQ_INSTALLATION_PATH*\java\jre

Tabela 51. diretórios do JRE (continuação)	
Plataforma	**Diretório**
z/OS z/OS 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.

No IBM MQ classes for Java, para os aplicativos a serem executados, o caminho de classe 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 ambiente CLASSPATH.

Importante: Configurar a opção Java *-Xbootclasspath* para incluir o IBM MQ classes for Java não é suportado.

Tabela 52 na página 332 mostra a configuração de CLASSPATH apropriada a ser usada em cada plataforma para executar aplicativos IBM MQ classes for Java, incluindo os aplicativos de amostra.

Tabela 52. Configurando CLASSPATH para executar aplicativos IBM MQ classes for Java, incluindo os aplicativos de amostra do IBM MQ classes for Java	
Plataforma	**Configuração de CLASSPATH**
AIX AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Solaris Linux HP-UX Solaris Linux HP-UX HP-UX, Linux e Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Windows Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples/pcf

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Se você compilar usando a opção -Xlint, poderá ver uma mensagem que avise 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.

UNIX No UNIX, 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.

Linux UNIX No UNIX and Linux, esses 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. Consulte Configurando o IBM MQ com Java e JMS para obter informações adicionais.

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 330.

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 333 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	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)
HP-UX Linux (POWER, x86-64 e Plataformas zSeries s390x) Solaris (plataformas x86-64 e SPARC)	*MQ_INSTALLATION_PATH*/java/lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliotecas de 64 bits)

Tabela 53. O local das bibliotecas do IBM MQ classes for Java para cada plataforma (continuação)	
Plataforma	**Diretório que contém as bibliotecas do IBM MQ classes for Java**
Linux (plataforma x86)	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\Java\lib (bibliotecas de 32 bits) *MQ_INSTALLATION_PATH*\Java\lib64 (bibliotecas de 64 bits)
z/OS z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (bibliotecas de 32 bits e de 64 bits)

Nota:

1. Solaris Linux HP-UX AIX No AIX, HP-UX, Linux (plataforma Power) ou Solaris, use as bibliotecas de 32 bits ou de 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.osgi.allclient_*version_number*.jar
Esse arquivo JAR permite que aplicativos usem o IBM MQ classes for JMS e o IBM MQ classes for Java e também inclui o código para manipular mensagens do PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
Esse arquivo JAR fornece os pré-requisitos para o com.ibm.mq.osgi.allclient_*version_number*.jar.

em que *version_number* é o número da versão do IBM MQ que foi instalado.

Os pacotes configuráveis são instalados no subdiretório `java/lib/OSGi` de sua instalação da pasta IBM MQ ou `java\lib\OSGi` no Windows.

A partir do IBM MQ 8.0, use os pacotes configuráveis `com.ibm.mq.osgi.allclient_8.0.0.0.jar` e `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` para todos os 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 anteriores a IBM MQ 8.0, a restrição de uso de IBM MQ classes for JMS ou IBM MQ classes for Java se aplica

Nove outros pacotes configuráveis também são instalados no subdiretório `java/lib/OSGi` de sua instalação do IBM MQ ou 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 *Instalação do IBM MQ classes for Java no z/OS*

No z/OS, o STEPLIB usado no tempo de execução deve conter as bibliotecas SCSQAUTH e SCSQANLE do IBM MQ.

No UNIX and Linux System Services, é possível incluir essas bibliotecas usando uma linha em seu `.profile`, conforme mostrado no seguinte exemplo, substituindo `thlqual` pelo qualificador do conjunto de dados de alto nível que você escolheu ao instalar o IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

Em outros ambientes, normalmente é necessário editar a JCL de inicialização para incluir SCSQAUTH na concatenação STEPLIB:

```
 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.

V 9.0.0.2 V 9.0.3 No IBM MQ 9.0.3 e IBM MQ 9.0.0 Fix Pack 2, um arquivo de configuração de amostra, `mqjava.config`, é fornecido no subdiretório `bin` do diretório de instalação do 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 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 `ChannelDefinitionFile` e `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 353 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	**Atribuir**
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
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 do IBM MQ MQI client, veja Configurando um cliente usando um arquivo de configuração.

Informações relacionadas

Rastreando classes do IBM MQ para aplicativos Java

Sub-rotina de rastreio do ambiente Java Standard
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:

- Para o IBM MQ classes for Java for IBM MQ 9.0.0 Fix Pack 1 ou anterior, o rastreio é gravado em um arquivo chamado `mqjms_%PID%.trc`.
- V 9.0.0.2 Na IBM MQ 9.0.0 Fix Pack 2, 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%.trc`.
- V 9.0.0.2 Na IBM MQ 9.0.0 Fix Pack 2, se o aplicativo tiver carregado o IBM MQ classes for Java do arquivo JAR realocável `com.ibm.mq.allclient.jar`, o rastreio será gravado em um arquivo chamado `mqjavaclient_%PID%.trc`.
- V 9.0.0.10 Na IBM MQ 9.0.0 Fix Pack 10, 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`.
- V 9.0.0.10 Na IBM MQ 9.0.0 Fix Pack 10, se o aplicativo tiver carregado o IBM MQ classes for Java do arquivo JAR realocável `com.ibm.mq.allclient.jar`, o rastreio será gravado em um arquivo chamado `mqjavaclient_%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 estão executando o rastreio em diferentes carregadores de classe do Java.

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_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 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 "O que é instalado para IBM MQ classes for Java" na página 329 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 1086 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.

Um gerenciador de filas do IBM MQ pode enviar notificações para clientes conectados solicitando um encerramento controlado de conversas (manipulações de conexão), por exemplo, quando o gerenciador de filas está sendo colocado em modo quiesce. Se um encadeamento dentro de um cliente Java receber uma dessas notificações ao mesmo tempo que outro encadeamento dentro do cliente solicitar uma

nova conversa, um conflito poderá ocorrer, visto que ambos os encadeamentos precisam de acesso ao "connectionsLock" interno no objeto RemoteConnectionSpecification.

V 9.0.5 V 9.0.0.3 Em IBM MQ 9.0.0 Fix Pack 3 e IBM MQ 9.0.5, o conflito dentro do cliente do IBM MQ Java é corrigido. Se seu aplicativo Java usar o Java Security Manager, deve-se incluir a permissão a seguir no arquivo `java.security.policy` usado pelo aplicativo, caso contrário, as exceções serão lançadas 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";

V 9.0.0.3// 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.

Executando aplicativos IBM MQ classes for Java em CICS Transaction Server
Um aplicativo IBM MQ classes for Java pode ser executado como uma transação sob o CICS Transaction Server.

Para executar um aplicativo IBM MQ classes for Java como uma transação sob o CICS Transaction Server for z/OS, execute as etapas a seguir:

1. Defina o aplicativo e a transação para o CICS usando a transação CEDA fornecida.
2. Assegure-se de que o adaptador IBM MQ CICS esteja instalado em seu sistema CICS. z/OS (Consulte Usando o IBM MQ com o CICS para obter detalhes.)
3. Certifique-se de que o ambiente de JVM especificado em CICS inclua as entradas CLASSPATH e LIBPATH apropriadas.
4. Inicie a transação usando qualquer um dos seus processos normais.

Para obter mais informações sobre a execução de transações CICS Java, consulte a documentação do sistema CICS.

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 1086. 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 323.

   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 332.
4. Mude o diretório para *MQ_INSTALLATION_PATH*/mqm/samp/wmqjava/samples, em que *MQ_INSTALLATION_PATH* é o caminho para sua 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 333).

   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 mais informações, consulte “Gravando aplicativos do IBM MQ classes for Java” na página 348. As amostras podem estar sujeitas a mudanças em futuras liberações do IBM MQ Java.

A Tabela 55 na página 345 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
UNIX UNIX 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 345 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 de mensagem introduzida na IBM WebSphere MQ 7.0.
MQPubSubApiSample.java	Demonstra o uso da API de publicação/assinatura introduzida na IBM WebSphere MQ 7.0.
MQSample.java	Programa simples para demonstrar a colocação e o recebimento de uma mensagem de uma fila.
MQSampleMessageManager.java	Classe do utilitário para manipulação de mensagens nas amostras Java base do IBM MQ.
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 346 mostra o local do script em cada plataforma:

Tabela 57. Localização do script `runjms`	
Plataforma	**Diretório**
UNIX UNIX 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 etapas a seguir:

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 79
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 programa não for concluído com êxito, execute o programa de verificação da instalação e siga o conselho fornecido nas mensagens de diagnóstico. Este programa está descrito em “Verificando a Instalação do IBM MQ classes for Java” na página 343.

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 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 do *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 333).

Para obter mais informações sobre como usar o rastreio, consulte Rastreando aplicativos IBM MQ classes for Java.

V 9.0.4 z/OS MQ Adv. VUE Java e conectividade do cliente JMS para aplicativos em lote em execução no z/OS

Um aplicativo JMS ou IBM MQ classes for Java no z/OS pode se conectar a um gerenciador de filas no z/OS, que tem o atributo **ADVCAP**(ENABLED), usando uma conexão do cliente.

Um valor de **ADVCAP**(ENABLED) se aplica apenas a um gerenciador de filas do z/OS , licenciado como IBM MQ Advanced for z/OS, Value Unit Edition (consulte IBM MQ identificadores do produto e informações de exportação), executando com **QMGRPROD** configurado como ADVANCEDVUE.

Veja DISPLAY QMGR para obter mais informações sobre **ADVCAP** e START QMGR para obter mais informações sobre **QMGRPROD**.

Observe que lote é o único ambiente suportado; não há suporte para o JMS for CICS ou JMS for IMS.

Um aplicativo IBM MQ classes for JMSou IBM MQ classes for Javano z/OS não pode se conectar, usando a conexão do modo cliente, a um gerenciador de filas que não está em execução no z/OSou a um gerenciador de filas que não tem a opção **ADVCAP**(ENABLED) ....

Se um aplicativo JMS tenta se conectar usando o modo cliente e o aplicativo não tem permissão para fazer isso, a mensagem de exceção JMSFMQ0005 é emitida.

Se um aplicativo IBM MQ classes for Java no z/OS tentar se conectar usando o modo cliente, e não tiver permissão para isso, um MQRC_ENVIRONMENT_ERROR será retornado.

Suporte ao Advanced Message Security (AMS)

V 9.0.5

No IBM MQ 9.0.5, IBM MQ classes for JMS ou IBM MQ classes for Java, os aplicativos clientes poderão usar o AMS ao se conectarem aos gerenciadores de filas do IBM MQ Advanced for z/OS, Value Unit Edition em sistemas z/OS remotos.

Um novo tipo de armazenamento de chaves, `jceracfks`, é suportado em `keystore.conf` no z/OS somente, 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`.

Ao usar `jceracfks` com AMS, o armazenamento de chave 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
```

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, você grava aplicativos do Java que contêm chamadas que enviam 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 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 733.

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 98

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), mas se esta informação não estiver disponível, o texto `WebSphere MQ Client for Java` é 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	**Atribuir**
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 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 715.

  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 374.
- 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 a localização de um respondente OCSP. O IBM MQ classes for Java não pode usar as informações do OCSP em um arquivo de 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 367. 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 371.

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 “Classe MQMessage” na página 621 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 .

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 24.

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 Java é inerentemente multiencadeado. O IBM MQ classes for Java permite que um objeto gerenciador de filas seja compartilhado entre vários encadeamentos, mas garante que todo o acesso ao gerenciador de filas de destino esteja 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 IBM MQ MQI client baseado em C, isso causaria um problema, pois há limitações no compartilhamento de manipuladores por vários encadeamentos. O IBM MQ classes for Java flexibiliza essa restrição, permitindo que um objeto gerenciador de filas (e seus objetos de fila, tópico e 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 370 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 370 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, HP-UX, Linux e Solaris	`/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 353.

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 WebSphere MQ 7.0, 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.

Usando as classes de saída externas

Em versões anteriores a IBM WebSphere MQ 7.0, três classes foram fornecidas para permitir que você use as saídas de canal escritas em linguagens diferentes de Java:

- MQExternalSecurityExit, que implementa a interface MQSecurityExit
- MQExternalSendExit, que implementa a interface MQSendExit
- MQExternalReceiveExit, que implementa a interface MQReceiveExit

O uso dessas classes permanece válido, mas o novo método é preferencial.

Para usar uma saída de segurança que não é escrita em Java, um aplicativo primeiro precisava criar um objeto MQExternalSecurityExit. O aplicativo especificou, como parâmetros no construtor MQExternalSecurityExit, o nome da biblioteca que contém a saída de segurança, o nome do ponto de entrada para a saída de segurança e os dados do usuário a serem passados para a saída de segurança quando foi chamado. Programas de saída do canal que não são gravados em Java foram armazenados no diretório mostrado em Tabela 58 na página 370.

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 dos dados que fluem em um canal pode melhorar o desempenho do canal e reduzir o tráfego de rede. Os IBM MQ classes for Java usam a função de compactação integrada ao 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));
:
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 376).

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 oferece compatibilidade com o Oracle 12C para uma instalação do servidor IBM MQ 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`

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 o Db2, o nome de cada arquivo de objeto é jdbcdb2.o e, para o 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 usar 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 comutação de 32 bits, insira o comando no diretório /java/lib/jdbc de sua instalação do IBM MQ. Para criar uma biblioteca de comutação de 64 bits, insira 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 o Db2 e `oracle` para o Oracle.

Aqui está um exemplo de um comando **make**:

```
make db2
```

Note os seguintes pontos:

- 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 o Db2, o nome de cada biblioteca de comutação é jdbcdb2 e, para o 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 comutação de 32 bits é armazenada no diretório /java/lib/jdbc e uma biblioteca de comutação de 64 bits é armazenada no diretório /java/lib64/jdbc.
- Como é possível instalar o Oracle em qualquer lugar de um sistema, os makefiles usam a variável de ambiente ORACLE_HOME para localizar onde o Oracle está instalado.

Depois de ter criado as bibliotecas de comutação para o Db2, o Oracle, ou ambos, deve-se declará-las a seu gerenciador de filas. Se o arquivo de configuração do gerenciador de filas (qm.ini) já contiver sub-rotinas XAResourceManager para bancos de dados Db2 ou Oracle, deve-se substituir a entrada SwitchFile em cada sub-rotina por uma das 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 ainda não contiver sub-rotinas XAResourceManager para bancos de dados Db2 ou Oracle ou se você desejar incluir sub-rotinas XAResourceManager adicionais, consulte Administrando para obter 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 ou 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 353.

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 386 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 o TLS com um CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
```

```
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Embora o canal tenha um CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA, o aplicativo Java deve especificar um CipherSuite de SSL_RSA_WITH_AES_128_CBC_SHA. Consulte "CipherSpecs e CipherSuites TLS no IBM MQ classes for Java" na página 386 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 informações adicionais, consulte "Usando uma tabela de definição de canal do cliente com o IBM MQ classes for Java" na página 353

Se você precisar de uma conexão do cliente para usar um CipherSuite que seja suportado pelo provedor JSSE FIPS do IBM Java (IBMJSSEFIPS), um aplicativo poderá configurar o campo sslFipsObrigatório na classe MQEnvironment para `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.

Informações relacionadas

Federal Information Processing Standards (FIPS) para UNIX, 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 que começa com QMGR. e pelo menos dois nomes da 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 para usar através da classe java.security.cert.CertStore. O IBM MQ classes for Java então verifica os certificados com relação ao CRL especificado.

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 386.

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.

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 o IBM MQ. Os CipherSuites listados incluem aqueles suportados pelo Oracle Java JRE. Para obter mais informações sobre como configurar seu aplicativos para usar um Oracle Java JRE, veja “Configurando seu aplicativo para usar mapeamentos do IBM Java ou Oracle Java CipherSuite” na página 411.

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.

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 mais alto de IBM JRE fornece o suporte apropriado

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* da versão que você está usando.

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLSv1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLSv1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na página 410	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLSv1	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLSv1	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	sim

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**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	TLSv1	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)

CipherSpec	**CipherSuite equivalente (IBM JRE)**	**CipherSuite equivalente (Oracle JRE)**	**Protocolo**	**compatível com o FIPS 140-2**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLSv1.2	no

Tabela 59. CipherSpecs suportados pelo IBM MQ e seus CipherSuites equivalentes (continuação)				
CipherSpec	**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	TLSv1.2	no

Notes:

1. 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.

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

É possível configurar se seu aplicativo usa IBM Java CipherSuite padrão para os mapeamentos IBM MQ CipherSpec ou o Oracle CipherSuite para mapeamentos IBM MQ CipherSpec. Portanto, será possível usar CipherSuites TLS se seu aplicativo usar um IBM JRE ou um Oracle JRE. A Propriedade de sistema Java `com.ibm.mq.cfg.useIBMCipherMappings` controla quais mapeamentos são usados. A propriedade pode ser um dos valores a seguir:

true
Use os mapeamentos do IBM Java CipherSuite para IBM MQ CipherSpec.

Esse valor é o valor padrão.

false
Use os mapeamentos do Oracle CipherSuite para IBM MQ CipherSpec.

Para obter mais informações sobre como usar o IBM MQ Java e os Ciphers TLS, consulte o post do blog do MQdev MQ Java, Ciphers TLS, JREs & APARs nãoIBM IT06775, IV66840, IT09423, IT10837.

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 411	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. 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 IBM MQ classes for Java. Para obter mais informações, consulte “IBM MQ classes for Java bibliotecas” na página 333.

Informações relacionadas

Rastreando aplicativos do IBM MQ classes for Java

Rastreando o Adaptador de recursos IBM MQ

V 9.0.4 z/OS MQ Adv. VUE *Java e conectividade do cliente JMS para aplicativos em lote em execução no z/OS*

Um aplicativo JMSou IBM MQ classes for Javano z/OS pode se conectar a um gerenciador de filas no z/OS, que possui o atributo **ADVCAP**(ENABLED), usando uma conexão do cliente.

Um valor de **ADVCAP**(ENABLED) se aplica apenas a um gerenciador de filas do z/OS , licenciado como IBM MQ Advanced for z/OS, Value Unit Edition (consulte IBM MQ identificadores do produto e informações de exportação), executando com **QMGRPROD** configurado como ADVANCEDVUE.

Veja DISPLAY QMGR para obter mais informações sobre **ADVCAP** e START QMGR para obter mais informações sobre **QMGRPROD**.

Observe que lote é o único ambiente suportado; não há suporte para o JMS for CICS ou JMS for IMS.

Um aplicativo IBM MQ classes for JMSou IBM MQ classes for Javano z/OS não pode se conectar, usando a conexão do modo cliente, a um gerenciador de filas que não está em execução no z/OSou a um gerenciador de filas que não tem a opção **ADVCAP**(ENABLED) ....

Se um aplicativo JMS tenta se conectar usando o modo cliente e o aplicativo não tem permissão para fazer isso, a mensagem de exceção JMSFMQ0005 é emitida.

Se um aplicativo IBM MQ classes for Java no z/OS tentar se conectar usando o modo cliente, e não tiver permissão para isso, um MQRC_ENVIRONMENT_ERROR será retornado.

Suporte ao Advanced Message Security (AMS)

V 9.0.5

No IBM MQ 9.0.5, IBM MQ classes for JMS ou IBM MQ classes for Java, os aplicativos clientes poderão usar o AMS ao se conectarem aos gerenciadores de filas do IBM MQ Advanced for z/OS, Value Unit Edition em sistemas z/OS remotos.

Um novo tipo de armazenamento de chaves, `jceracfks`, é suportado em `keystore.conf` no z/OS somente, 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`.

Ao usar `jceracfks` com AMS, o armazenamento de chave 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
```

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 414.

- 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 414 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 Windows, UNIX ou Linux IBM MQ fosse usado, com exceção das seguintes restrições e variações menores.

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.

Diversas diferenças entre o IBM MQ classes for Java no z/OS e em outras plataformas
O IBM MQ for z/OS se comporta de modo diferente do IBM MQ em outras plataformas em algumas áreas.

BackoutCount
Um gerenciador de filas do z/OS retorna um BackoutCount máximo de 255, mesmo se a mensagem tiver sido restaurada mais de 255 vezes.

Prefixo de fila dinâmico padrão
Quando conectado a um gerenciador de filas do z/OS usando uma conexão de ligações, o prefixo de fila dinâmico padrão é CSQ.*. Caso contrário, o prefixo de fila dinâmico padrão é AMQ.*.

Construtor MQQueueManager
A conexão do cliente não é suportada no z/OS. A tentativa de se conectar com outras opções do cliente resulta em uma MQException com MQCC_FAILED e MQRC_ENVIRONMENT_ERROR.

O construtor MQQueueManager também pode falhar com MQRC_CHAR_CONVERSION_ERROR (se falhar ao inicializar conversão entre as páginas de código IBM-1047 e ISO8859-1) ou MQRC_UCS2_CONVERSION_ERROR (se falhar ao inicializar conversão entre a página de códigos do gerenciador de filas e Unicode). Se seu aplicativo falhar com um desses códigos de razão, assegure que o componente National Language Resources do Language Environment esteja instalado e assegure que as tabelas de conversão corretas estejam disponíveis.

Tabelas de conversão para Unicode são instaladas como parte do recurso opcional C/C++ do z/OS. Consulte o *Guia de programação de C/C++ do z/OS*, SC09-4765, para obter mais informações sobre como ativar as conversões UCS-2.

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()

Este método pode ser usado apenas em relação a um gerenciador de filas do IBM MQ no UNIX, Linux ou sistemas Windows no modo de ligações. Caso contrário, ele falha com MQRC_ENVIRONMENT_ERROR.

Consulte “Coordenação JTA/JDBC usando o IBM MQ classes for Java” na página 377 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 "Diversas diferenças entre o IBM MQ classes for Java no z/OS e em outras plataformas" na página 415 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

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 do IBM MQ 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 325.

Qual versão do adaptador de recursos usar

A versão do Java Platform, Enterprise Edition (Java EE) do servidor de aplicativos que você está usando determina a versão do adaptador de recursos que deve ser usado:

Java EE 7
O adaptador de recursos do IBM MQ 8.0 e do IBM MQ 9.0 suporta 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 419).

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 IBM WebSphere MQ 7.0 ou posterior usando o transporte BINDINGS ou CLIENT ou a um gerenciador de filas IBM WebSphere MQ 6.0 usando apenas o transporte 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.

Java EE 5 e Java EE 6

O adaptador de recursos do IBM WebSphere MQ 7.5 suporta Java EE Connector Architecture (JCA) v1.5 e fornece suporte ao JMS 1.1. Para fornecer integração total com o WebSphere Application Server Liberty, o adaptador de recursos do IBM WebSphere MQ 7.5 é atualizado para a APAR IC92914 do IBM WebSphere MQ 7.5.0 Fix Pack 2. Esse adaptador de recursos retém a compatibilidade total com outros servidores de aplicativos Java EE 5 e mais recentes (consulte o adaptador de recursos WebSphere MQ 7.1 e a instrução de suporte mais recente).

Usando o adaptador de recursos com o WebSphere Application Server traditional

V 9.0.0 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.

Nota: Um adaptador de recursos do IBM MQ 9.0 pode se conectar no modo de transporte CLIENT ou BINDINGS a qualquer gerenciador de filas do IBM MQ dentro do serviço.

Usando o adaptador de recursos com o WebSphere Application Server Liberty

Para conectar-se ao IBM MQ a partir do WebSphere Application Server 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 da versão do Java EE do servidor de aplicativos.

Para obter mais informações sobre como fazer download e instalar o adaptador de recursos, veja “Instalando o adaptador de recursos no Liberty” na página 426.

Conceitos relacionados

“Configurando o adaptador de recursos para comunicação de entrada” na página 433
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 451
Para configurar a comunicação de saída, defina as propriedades de um objeto ConnectionFactory e um objeto de destino administrado.

“Usando o IBM MQ classes for JMS” na página 75
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) é o provedor do JMS que é fornecido com o IBM MQ. Bem como a implementação das interfaces definidas no pacote javax.jms, o IBM MQ classes for JMS fornece dois conjuntos de extensões para a API do JMS.

“Usando o IBM MQ classes for Java” na página 323
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.

Informações 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

Informações relacionadas para WebSphere Application Server Versão 8.5.5

Mantendo o adaptador de recursos do IBM MQ

Implementando os aplicativos JMS para o perfil Liberty para usar o provedor do sistema de mensagens do IBM MQ

Instrução de suporte do adaptador de recursos do IBM MQ

O adaptador de recursos que é fornecido com o IBM MQ 8.0 ou mais recente implementa a especificação do 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 é mantida no website da Oracle.

Implementação em WebSphere Application Server 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, portanto, o adaptador de recursos do IBM MQ 8.0 ou mais recente pode ser implementado neles.

O WebSphere Application Server Liberty possui o recurso wmqJmsClient-1.1 para permitir trabalhar com os adaptadores de recursos do JMS 1.1 e o recurso wmqJmsClient-2.0 para permitir trabalhar com os adaptadores de recursos do JMS 2.0. O adaptador de recursos do IBM MQ 8.0 ou mais recente deve ser implementado com o recurso wmqJmsClient-2.0.

Informações sobre essa configuração estão no Cenário Conectando o perfil Liberty WebSphere Application Server para o IBM MQ.

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. O adaptador de recursos pode se conectar a qualquer gerenciador de filas que está em execução em uma versão suportada do IBM MQ ou do IBM WebSphere MQ. Para obter mais informações, consulte “Conectividade com os gerenciadores de filas IBM MQ 8.0 e 9.0” na página 420.

O adaptador de recursos do IBM MQ 9.0 não pode ser implementado em versões anteriores do WebSphere Application Server, pois essas versões não são certificadas pelo Java EE 7.

Usando o adaptador de recursos com outros servidores de aplicativos

Para todos os outros servidores de aplicativos compatíveis com o Java EE 7, os problemas que ocorrem após a conclusão bem-sucedida do Teste de verificação de instalação (IVT) do adaptador de recursos do IBM MQ podem ser relatados para o IBM para obter a investigação de rastreio de produto do IBM MQ e outras informações de diagnóstico do IBM MQ. Caso o teste de verificação de instalação (IVT) do adaptador de recursos do IBM MQ não possa ser executado com sucesso, é provável que qualquer problema encontrado tenha sido causado pela implementação incorreta ou por definições incorretas de recursos que são específicas do servidor de aplicativos. Além disso, os problemas devem ser investigados usando a documentação do servidor de aplicativos e/ou a organização de suporte para esse servidor de aplicativos.

JavaRuntime

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 Client. Esses JREs estão listados no Requisitos do sistema para IBM MQ. (Selecione qual sistema operacional ou componente de relatório que você deseja ver, em seguida, siga o link do **Java** listado na guia **Software Suportado**).

Conectividade com os gerenciadores de filas IBM MQ 8.0 e 9.0

O intervalo completo da funcionalidade do JMS 2.0 está disponível ao se conectar a um gerenciador de filas IBM MQ 8.0 ou 9.0 usando o adaptador de recursos do IBM MQ 9.0 que foi implementado em um servidor de aplicativos certificado do Java EE 7 . Para usar essa funcionalidade, 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 informações adicionais, consulte **PROVIDERVERSION** unspecified.

Conectividade com o gerenciador de filas do IBM WebSphere MQ 7.5 ou anterior

É suportada para implementar o adaptador de recursos do IBM MQ 9.0 em um servidor de aplicativos certificado Java EE 7 que suporta JMS 2.0 e se conecta a esse adaptador de recursos para um gerenciador de filas que está executando o IBM WebSphere MQ 7.5 ou anterior. A funcionalidade que está disponível é limitada pela capacidade do gerenciador de filas. Para obter mais informações, consulte ../com.ibm.mq.con.doc/q123360_.dita#q123360_.

Extensões de MQ

A especificação do JMS 2.0 introduz mudanças em como certos comportamentos funcionam. Como o IBM MQ 8.0 e o 9.0 implementam essa especificação, há mudanças no comportamento entre essas duas liberações e versões anteriores do IBM WebSphere MQ. O IBM MQ 8.0 e 9.0 IBM MQ classes for JMS incluem suporte para a Java propriedade de sistema `com.ibm.mq.jms.SupportMQExtensions` que, quando configurado como TRUE, faz com que essas duas liberações revertam esses comportamentos para aqueles de IBM WebSphere MQ 7.5 ou anterior... O valor padrão da propriedade é FALSE.

O adaptador de recursos do IBM MQ 9.0 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 false 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 mais informações, consulte “Propriedade SupportMQExtensions” na página 309.

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 Application Server 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 nota técnica do IBM Usando as interfaces Java do WebSphere MQ em ambientes J2EE/JEE para obter detalhes das considerações do IBM MQ classes for Java no Java EE.
- Ao executar dentro do servidor de aplicativos do WebSphere Application Server 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 novas tentativas de conexão introduzida no IBM WebSphere MQ 7.0.1 não é suportada para conexões de saída quando em execução em um contêiner de 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.

Informações relacionadas

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Federal Information Processing Standards (FIPS) para UNIX, Linux e 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 IBM MQ ou IBM WebSphere MQ com WebSphere Application Server 6.0 ou 6.1..

O WebSphere Application Server 7.0 e o WebSphere Application Server 8.0 incluem uma versão do adaptador de recursos do IBM WebSphere MQ 7.0.

O WebSphere Application Server 8.5.5 inclui uma versão do adaptador de recursos do IBM WebSphere MQ 7.1.

V 9.0.0 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 não pode ser implementado em versões anteriores do WebSphere Application Server, pois essas versões não são certificadas pelo Java EE 7.

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 do IBM MQ pode ser instalado no WebSphere Application Server Liberty WebSphere Application Server 8.5.5 Fix Pack 2 ou mais recente, usando o recurso wmqJmsClient-1.1 ou wmqJmsClient-2.0, dependendo da versão do adaptador de recursos que está sendo instalada. 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 Java do WebSphere MQ em Ambientes 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.
- V 9.0.0 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.

Restrições ao usar os recursos do Liberty

Com o Liberty WebSphere Application Server 8.5.5 Fix Pack 2 para WebSphere Application Server 8.5.5 Fix Pack 5 inclusive, somente o recurso wmqJmsClient-1.1 estava disponível e somente o JMS 1.1 poderia ser usado.Liberty O WebSphere Application Server 8.5.5 Fix Pack 6 incluiu o recurso wmqJmsClient-2.0 para que o JMS 2.0 pudesse ser usado.

No entanto, o recurso que você deve usar depende da versão do adaptador de recursos que está sendo usada:

- O IBM WebSphere MQ 7.5.0 Fix Pack 6 e o adaptador de recursos do IBM WebSphere MQ 7.5 mais recente podem ser usados somente com o recurso wmqJmsClient-1.1.
- 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.

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 JCA genérico:

  - O JMS 1.1 e JCA 1.6 devem ser usados somente com os adaptadores de recursos do IBM WebSphere MQ 7.5.0 Fix Pack 6 e IBM WebSphere MQ 7.5 mais recente.
  - 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:

```
<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`:

```
<!-- 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 a volta de classloaderProviderRef para o adaptador de recursos por meio do id `mqJms`; isso é para permitir que classes específicas do IBM MQ sejam 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 recursos IBM MQ deve ser ativado usando propriedades do sistema Java ou um arquivo de configuração IBM MQ classes for JMS , conforme descrito em Rastreio IBM MQ classes para aplicativos JMS. Para obter detalhes sobre como configurar propriedades do sistema Java em Liberty, consulte a WebSphere Application Server 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>`.

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 do IBM MQ é fornecido como um arquivo archive de recursos (RAR) chamado `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 o adaptador de recursos é instalado como parte da instalação do produto IBM MQ, o `wmq.jmsra.rar` é instalado com IBM MQ classes for JMS no diretório mostrado em Tabela 61 na página 425.

Tabela 61. O diretório contendo wmq.jmsra.rar para cada plataforma

Plataforma	**Diretório**
UNIX 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 Application Server Liberty	O WebSphere Application Server Liberty não contém o adaptador de recursos do IBM MQ, por isso, deve-se obtê-lo separadamente no Fix Central.

Tabela 62. Instalação do adaptador de recursos em um servidor de aplicativos (continuação)	
Servidor de aplicativos	**Pré-instalado ou precisa ser instalado?**
Outros servidores de aplicativos Java EE	Obtenha o adaptador de recursos separadamente no Fix Central, como para o WebSphere Application Server Liberty.

Procedimento

- Se estiver conectando ao IBM MQ do WebSphere Application Server Liberty ou a outro servidor de aplicativos Java EE, faça o download e instale o adaptador de recursos do IBM MQ conforme descrito em "Instalando o adaptador de recursos no Liberty" na página 426.
- Linux UNIX

  Para conexões de ligações em UNIX 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 84.

  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 Application Server 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 do WebSphere Application Server Liberty ou outros servidores de aplicativos Java 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.

Sobre esta tarefa

O arquivo JAR para o adaptador de recursos no qual é possível fazer 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 *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 Application Server Liberty.

Procedimento

1. Faça download do adaptador de recursos do IBM MQ por meio do Fix Central:
   a) Clique em **Localizar produto**, em seguida, inclua as informações para a sua instalação do IBM MQ nos campos a seguir:
      - No campo **Seletor de produto**, digite MQ, em seguida, selecione **WebSphere MQ** na lista exibida.
      - No campo **Versão instalada**, clique na seta, em seguida, selecione o número da versão na lista exibida, por exemplo **9.0.0.0**.
      - No campo **Plataforma**, clique na seta e selecione sua plataforma, por exemplo **Windows 64 bits, x86**.

      Clique em **Continuar**.
   b) Certifique-se de que **Procurar correções** esteja selecionado e, em **Opções de consulta adicional**, limpe **Mostrar correções que se aplicam a esta versão** e selecione **Mostrar correções que me levam para esta versão**, em seguida, clique em **Continuar**.

      O Fix Central procurará as correções disponíveis para o produto, versão e plataforma selecionados, por exemplo **WebSphere, o WebSphere MQ (9.0.0.0, Windows 64 bits, x86)**.
   c) Localize o adaptador de recursos na lista exibida de correções disponíveis.

      Por exemplo:

      ```
      release level: 9.0.0.0-IBM-MQ-Install-Java-All
      9.0.0.0 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 a Versão, Liberação, Modificação e o número do 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 do IBM MQ para IBM MQ 9.0.0.0, você usaria o comando a seguir:

   ```
   java -jar 9.0.0.0-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:

   ```
   Antes de poder usar, extrair ou instalar o IBM MQ 9.0, 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 no diretório de instalação selecionado e os arquivos a seguir são, então, instalados no diretório `wmq`:

   - O programa de teste de verificação de instalação, `wmq.jmsra.ivt`.
   - O arquivo RAR do IBM MQ, `wmq.jmsra.rar`.

5. Configure o adaptador de recursos em WebSphere Application Server Liberty.

   As etapas que devem ser executadas para configurar o adaptador de recursos no Liberty são as seguintes. Para obter mais informações, consulte a WebSphere Application Server documentação do produto.

   a) Inclua o recurso wmqJmsClient-2.0 no arquivo `server.xml` para permitir trabalhar com o adaptador de recursos do IBM MQ 9.0.

      O recurso que você inclui (wmqJmsClient-1.1 ou wmqJmsClient-2.0) depende de qual versão do adaptador de recursos você instalou. O adaptador de recursos do IBM MQ 9.0 deve ser implementado com o recurso wmqJmsClient-2.0. Para obter mais informações, consulte “Qual versão do adaptador de recursos usar” na página 418.

   b) Inclua uma referência no arquivo `wmq.jmsra.rar` que você instalou.

   Nota: Para versões do Liberty até o WebSphere Application Server 8.5.5 Fix Pack 1, se um EJB for implementado usando apenas a configuração dentro do `ejb-jar.xml`, a versão do WebSphere Application Server que o Perfil do Liberty estiver 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.

Uma configuração de exemplo para suportar servlets e MDBs, com o JNDI, pode ser semelhante ao seguinte:

```
 <featureManager>
    <feature>wmqJmsClient-1.1</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"/>
```

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 431.
- Defina as propriedades de um objeto ActivationSpec.

  Estas propriedades determinam como um MDB é ativado para comunicação de entrada. Para obter mais informações, consulte “Configurando o adaptador de recursos para comunicação de entrada” na página 433.
- 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 451.
- 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 mais informações, consulte "Configurando o adaptador de recursos para comunicação de saída" na página 451.

- Opcional: Defina um plano de implementação para o adaptador de recursos.

  O arquivo RAR do adaptador de recursos do 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 231.

- 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 do IBM MQ classes for JMS" na página 87). 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 469.

  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 432 ou se estiver usando o WebSphere Application Server Liberty, veja “Configuração do WebSphere Application Server Liberty” na página 433. 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 431 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).
V 9.0.0 V 9.0.0 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 309.

Tabela 63. Propriedades do Objeto ResourceAdapter que estão associadas ao conjunto de conexões (continuação)

Nome da propriedade	Tipo	Valor padrão	Descrição
V 9.0.0 V 9.0.0 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 UNIX 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 Application Server Liberty

O adaptador de recursos é configurado usando elementos XML no arquivo `server.xml`, conforme mostrado no exemplo a seguir:

```
<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 de unidade de mensagem (MDB) recebe mensagens do JMS a partir de uma fila do 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.

Novas propriedades de ActivationSpec no JMS 2.0

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, suponha que um connection factory seja definido no JNDI e o nome JNDI desse objeto seja especificado na propriedade connectionFactoryLookup para uma especificação de ativação. Todas as propriedades do connection factory definidas no JNDI são usadas em preferência às propriedades em Tabela 64 na página 434.

Se um destino for definido no JNDI e o nome JNDI estiver configurado na propriedade destinationLookup do ActivationSpec, os valores serão usados em preferência aos valores em Tabela 65 na página 444. Para obter mais informações sobre como essas duas propriedades são usadas, veja “Propriedades ActivationSpec connectionFactoryLookup e destinationLookup” na página 448.

Propriedades usadas para criar uma conexão do JMS com um gerenciador de filas do IBM MQ

Todas as propriedades no Tabela 64 na página 434 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
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

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
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** • NENHUMA • 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
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 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
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 448.
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.

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
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.
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.

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
messageCompression	Sequência	• **NONE** • Uma lista de um ou mais dos seguintes valores separados por caracteres em branco: RLE ZLIBFAST ZLIBHIGH	Uma lista de técnicas que podem ser usadas para compactar os dados da mensagem em uma conexão
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 de 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.

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
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	cadeia de caracteres	• **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	• **false** • true	Se uma conexão TLS deve usar um CipherSuite que seja suportado pelo provedor JSSE FIPS IBM Java (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 de 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

Notes:

1. Essa propriedade pode ser usada com IBM MQ classes for JMS em IBM WebSphere MQ 7.0 Ela não afeta um aplicativo conectado a um gerenciador de filas do IBM WebSphere MQ 7.0, a menos que a propriedade **providerVersion** esteja configurada para um número de versão menor que 7.
2. Para obter informações importantes sobre o uso da propriedade sslFipsRequired, consulte "Limitações do adaptador de recursos do IBM MQ" na página 421.
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 262.

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 444 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 mais informações, consulte "Propriedades ActivationSpec connectionFactoryLookup e destinationLookup" na página 448.
destinationType	Sequência	• javax.jms.Queue • javax.jms.Topic	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.
maxPoolDepth	int	• **10** • Um inteiro positivo	O número máximo de sessões do servidor no conjunto de sessões do servidor usado pelo consumidor de conexão

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
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 Application Server 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 423.
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.	Indica se o MDB tem permissão para usar a leitura antecipada para obter as mensagens não persistentes do destino em um buffer interno antes de recebê-las
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 450.
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 objeto javax.jms.Queue ou javax.jms.Topic no namespace JNDI do servidor de aplicativos.	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 a propriedade **ccdtURL** somente, deixando a propriedade chamada **channel** com seu valor padrão de SYSTEM.DEF.SVRCONN, nenhuma exceção será lançada e a tabela de definição de canal do cliente identificada pela propriedade **ccdtURL** será usada para iniciar uma conexão JMS.

Propriedades ActivationSpec connectionFactoryLookup e destinationLookup

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 434 e Tabela 65 na página 444.

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 434. 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 declara que cada JMS Connection criado a partir de 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** for 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 a partir de 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**.

A propriedade useJNDI é descontinuada em IBM MQ 9.0 , pois a propriedade **destinationLookup** é a especificação JMS 2.0 equivalente 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 às propriedades de objetos do IBM MQ classes for JMS ou parâmetros de métodos IBM MQ classes for JMS. No entanto, três propriedades de ajuste e uma propriedade de usabilidade não possuem equivalentes no IBM MQ classes for JMS:

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 444.

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:

```
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 425.

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 214.

Informações relacionadas

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Federal Information Processing Standards (FIPS) para UNIX, Linux e Windows

Configurando recursos JMS no WebSphere Application Server

Exemplos de como definir a propriedade sharedSubscription

É possível definir a propriedade sharedSubscription de uma especificação de ativação dentro de um arquivo `server.xml` do WebSphere Application Server Liberty. Como alternativa, é possível definir a propriedade dentro de um bean acionado por mensagens (MDB) usando anotações.

Exemplo: definindo dentro de um arquivo `server.xml` do Liberty

Dentro de um arquivo `server.xml` do WebSphere Application Server Liberty, defina uma especificação de ativação, conforme 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:

```
/**
 * 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

“Assinaturas clonadas e compartilhadas” na página 307
No IBM MQ 8.0 ou mais recente, há dois métodos para dar a vários consumidores acesso à mesma assinatura. Esses dois métodos são através do uso de assinaturas clonadas ou compartilhadas.

Informações relacionadas

Assinantes e assinaturas

Durabilidade da assinatura

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:

```
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 452 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**).
brokerCCSubQueue [1]	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.
brokerControlQueue [1]	Sequência	• **SYSTEM.BROKER.CONTROL.QUEUE** • Um nome da fila	O nome da fila de controle do broker.
brokerPubQueue [1]	Sequência	• **SYSTEM.BROKER.DEFAULT.STREAM** • Um nome da fila	O nome da fila para onde as mensagens publicadas são enviadas (a fila de fluxo).

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	Tipo	Valores válidos (valor padrão em negrito)	Descrição
brokerQueueManager [1]	Sequência	• **"" (empty string)** • Um nome do gerenciador de filas	O nome do gerenciador de filas no qual o broker está em execução.
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. Consulte a propriedade BROKERSUBQ para obter mais informações.
brokerVersion [1]	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.
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 [1]	int	• **3 600 000** • Um inteiro positivo	O intervalo, em milissegundos, entre as execuções em segundo plano do utilitário de limpeza de publicação/assinatura.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	Tipo	Valores válidos (valor padrão em negrito)	Descrição
cleanupLevel [1]	Sequência	• **SAFE** • NENHUMA • 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.
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.

Tabela 66. Propriedades de um objeto ConnectionFactory (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: • 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	Uma lista de técnicas que podem ser usadas para compactar os dados da mensagem em uma conexão.
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 de 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 [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. 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	cadeia de caracteres	• **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.
pubAckInterval [1]	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 [1]	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 [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.

Tabela 66. Propriedades de um objeto ConnectionFactory (continuação)

Nome da propriedade	Tipo	Valores válidos (valor padrão em negrito)	Descrição
sslFipsRequired [2]	Booleana	• **false** • true	Se uma conexão TLS deve usar um CipherSuite que seja suportado pelo provedor JSSE FIPS IBM Java (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.
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 **SUBSTORE** tiver o valor QUEUE.
subscriptionStore [1]	Sequência	• **BROKER** • MIGRATE • FILA	Determina onde o IBM MQ classes for JMS armazena os dados persistentes sobre assinaturas ativas.
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.

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 sequência de tópico no formato "TEMP/ *TEMPTOPICPREFIX* / *unique_id* " ou se esta 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 de 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.

Notes:

1. Essa propriedade pode ser usada com IBM WebSphere MQ classes for JMS em IBM WebSphere MQ 7.0 , mas não afeta um aplicativo conectado a um gerenciador de fila do IBM WebSphere MQ 7.0 , a menos que a propriedade providerVersion seja configurada para um número de versão menor que 7
2. Para obter informações importantes sobre o uso da propriedade sslFipsRequired, consulte “Limitações do adaptador de recursos do IBM MQ” na página 421.
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 262.

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 463 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 determinada pelo atributo **NonPersistentMessageClass** da fila IBM MQ , de acordo com a explicação em "Mensagens persistentes do JMS" na página 230	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 466 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 467 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 [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.
brokerDurSubQueue [1]	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
brokerPubQueue [1]	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 [1]	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 [1]	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.

Nota:

1. Essa propriedade pode ser usada com IBM WebSphere MQ classes for JMS em IBM WebSphere MQ 7.0 , mas não afeta um aplicativo conectado a um IBM WebSphere MQ 7.0 gerenciador de filas,

a menos que a propriedade providerVersion do objeto ConnectionFactory seja configurada para um número de versão inferior a 7..

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
```

Informações relacionadas

Especificando que Apenas CipherSpecs Certificados por FIPS São Usados no Tempo de Execução no Cliente de MQI

Federal Information Processing Standards (FIPS) para UNIX, Linux, and Windows

Configurando recursos JMS no WebSphere Application Server

V 9.0.0.5 *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 Application Server 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 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 Application Server 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 202
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 451
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 archive corporativo (EAR) chamado `wmq.jmsra.ivt.ear`. Esse arquivo é instalado com o IBM MQ classes for JMS no mesmo diretório que o arquivo RAR do adaptador de recursos IBM MQ, `wmq.jmsra.rar`. Para obter informações sobre onde esses arquivos estão instalados, consulte “Instalando o adaptador de recursos do IBM MQ” na página 425.

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 Application Server Liberty, inclua as entradas a seguir no arquivo `server.xml` do servidor de aplicativos:

```
<!-- 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/
   ```

   Figura 52 na página 471 mostra a página inicial do programa IVT.

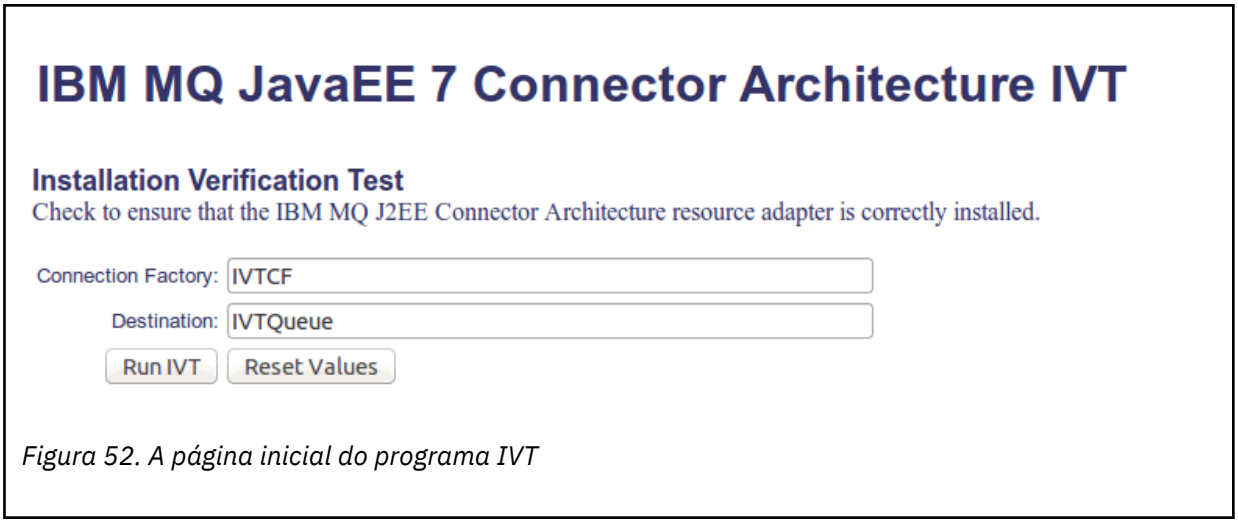


Figura 52. A página inicial do programa IVT

2. Para executar o teste, clique em **Run IVT**.

   Figura 53 na página 471 mostra a página que é exibida se o IVT for bem-sucedido.

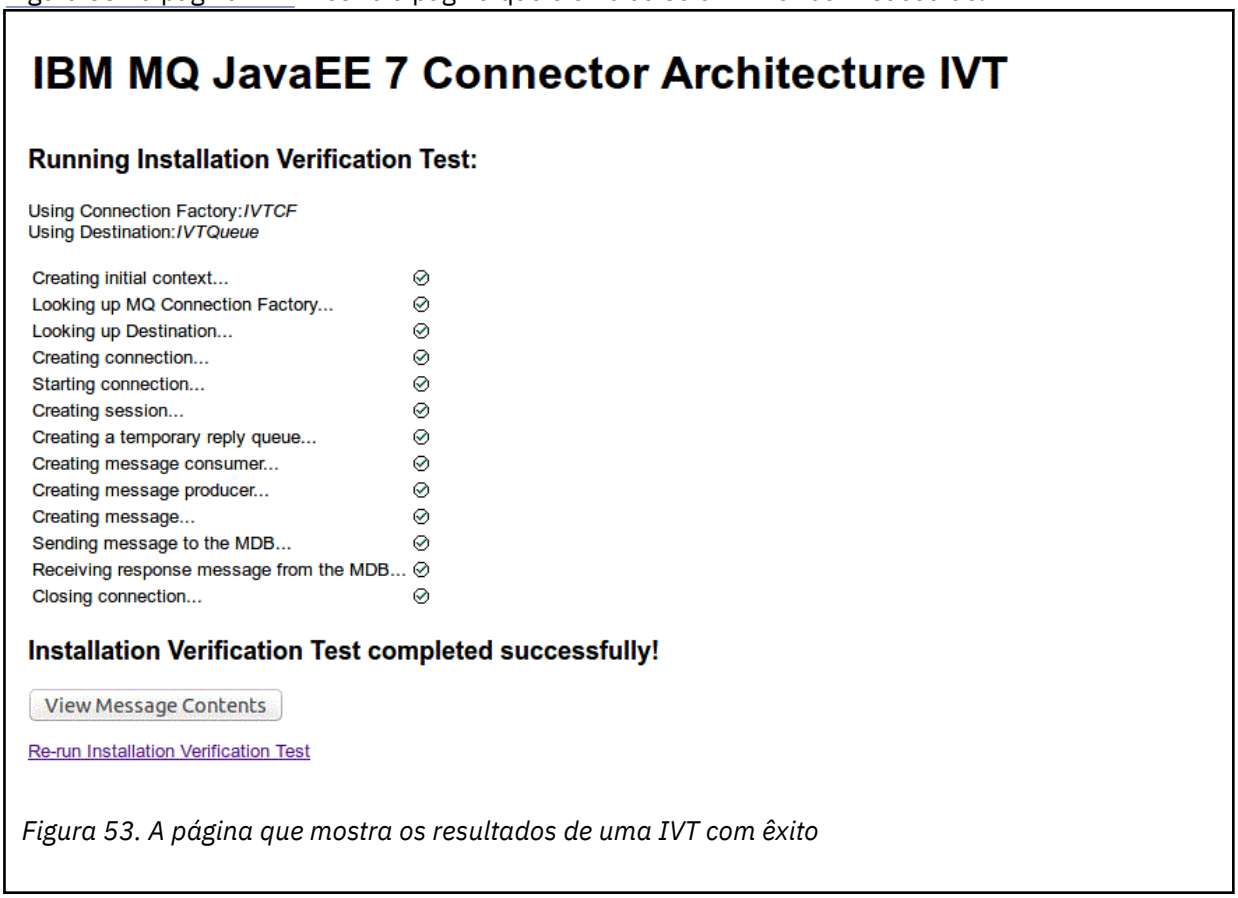


Figura 53. A página que mostra os resultados de uma IVT com êxito

Se o IVT falhar, uma página como essa mostrada em Figura 54 na página 472 será exibida. Para obter informações adicionais sobre a causa da falha, clique em **Visualizar rastreio de pilha**.

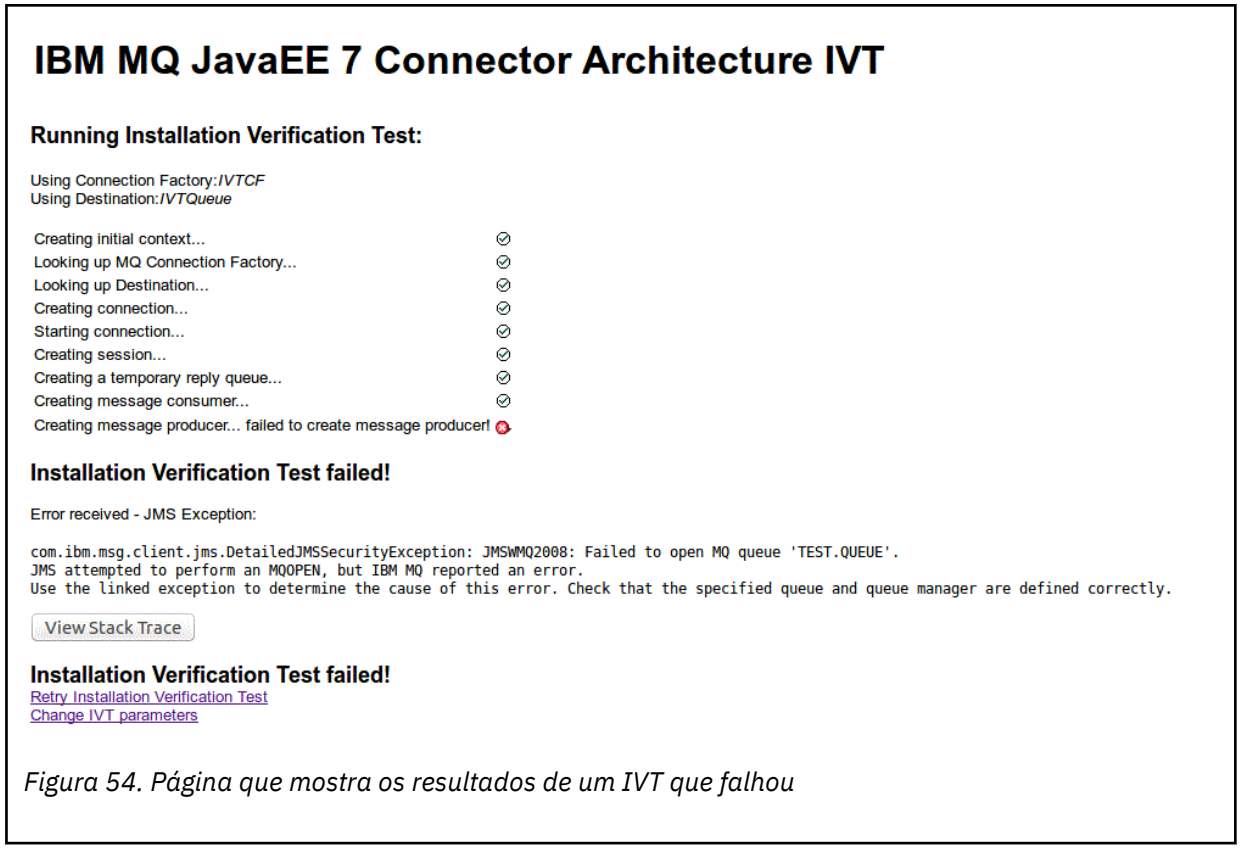


Figura 54. Página que mostra os resultados de um IVT que falhou

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).

Sobre esta tarefa

Importante: Estas instruções são para GlassFish Server versão 4.

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 do Windows e navegue para o diretório *GlassFish*/bin, em que *GlassFish* é o diretório no qual o GlassFish Server versão 4 está instalado.
   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** e, em seguida, navegue até o local do arquivo `wmq.jmsra.rar`. Selecione o arquivo e 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 `wmq.jmsra.ivt.ear` e selecione-o.

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:

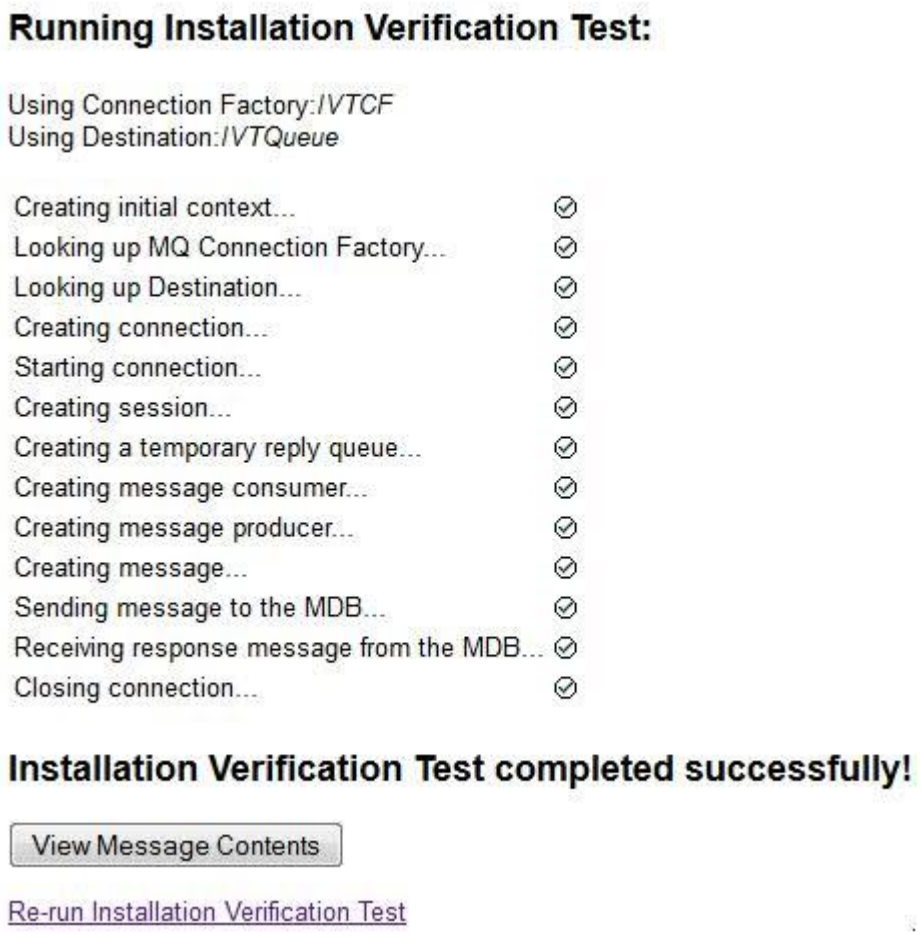


Figura 55. Saída de IVT bem-sucedida

V 9.0.0.1 V 9.0.2 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).

Sobre esta tarefa

Importante: Essas instruções são para o Wildfly V10.

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 1086.

   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 em seu 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 exibindo 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 mais informações, consulte “Verificando a instalação do adaptador de recursos” na página 469. 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 escritos em Java que estão em execução no WebSphere Application Server podem usar a especificação Java Messaging 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.

Informações 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.
- No WebSphere Application Server Liberty, consulte Implementando aplicativos JMS no Liberty para usar o provedor de sistema 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 STEPLIB do WebSphere Application Server for z/OS na documentação do produto do 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

WebSphere Application Server versões anteriores a 7.0 poderiam ser configuradas para usar o IBM WebSphere MQ classes for JMS localizado em uma instalação externa do IBM WebSphere MQ para se conectar a um gerenciador de filas configurando a variável WebSphere 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 9.0. Consulte o “Configurando as bibliotecas do Java Native Interface (JNI)” na página 84 para obter informações adicionais.
   - z/OS No z/OS, deve-se especificar as bibliotecas corretas do IBM MQ na concatenação STEPLIB do WebSphere Application Server.

     Veja bibliotecas IBM MQ e WebSphere Application Server for z/OS STEPLIB para detalhes das bibliotecas IBM MQ que você precisa.

     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, um WebSphere Application Server 8.5, com um Adaptador de Recursos IBM WebSphere MQ 7.1 ainda pode gerenciar uma conexão de ligação 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 419.

A tabela a seguir mostra quais tipos de transporte podem ser usados para se conectar ao IBM MQ de todas as versões do WebSphere Application Server.

Versão do IBM MQ ou IBM WebSphere MQ	Transporte de LIGAÇÕES	Transporte de CLIENTE
IBM MQ 9.0	Suportado. • O IBM MQ 9.0 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 9.0. • 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.	Suportado
IBM MQ 8.0	Suportado. • O IBM MQ 8.0 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 8.0. • 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.	Suportado
IBM WebSphere MQ 7.5	Suportado. • O IBM WebSphere MQ 7.5 deve ser instalado na mesma máquina que o servidor de aplicativos. • O WebSphere Application Server deve ser configurado para carregar as bibliotecas	Suportado

Versão do IBM MQ ou IBM WebSphere MQ	Transporte de LIGAÇÕES	Transporte de CLIENTE
	nativas fornecidas com o IBM WebSphere MQ 7.5. • z/OS No z/OS, se você deseja conectar um connection factory do WebSphere Application Server a um gerenciador de filas do IBM WebSphere MQ no modo de ligações, as bibliotecas do IBM WebSphere MQ corretas devem ser especificadas na concatenação STEPLIB do WebSphere Application Server.	
IBM WebSphere MQ 7.1	Suportado. • O IBM WebSphere MQ 7.1 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 WebSphere MQ 7.1. • z/OS No z/OS, se você deseja conectar um connection factory do WebSphere Application Server a um gerenciador de filas do IBM WebSphere MQ no modo de ligações, as bibliotecas do IBM WebSphere MQ corretas devem ser especificadas na concatenação STEPLIB do WebSphere Application Server.	Suportado

A tabela a seguir mostra as versões do WebSphere Application Server que o adaptador de recursos do IBM MQ suporta para execução.

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	O adaptador de recursos pode ser executado em: • Qualquer versão compatível do Java EE 7 do WebSphere Application Server 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 Application Server Liberty

Versão do adaptador de recursos do IBM MQ	Em qual versão do WebSphere Application Server esta versão do adaptador de recursos pode ser executada?
	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.
IBM WebSphere MQ 7.5	O adaptador de recursos pode ser utilizado em servidores de aplicativos compatíveis com o J2EE 1.4 ou mais recente. A versão do adaptador de recursos IBM WebSphere MQ incluída em WebSphere Application Server 8.0 e 7.0 deve ser usada nestes ambientes. O IBM WebSphere MQ 7.5.0 Fix Pack 2 e o APAR IC92914 incluem suporte para implementar o adaptador de recursos no WebSphere Application Server Liberty.
IBM WebSphere MQ 7.1	O adaptador de recursos pode ser utilizado em servidores de aplicativos compatíveis com o J2EE 1.4 ou mais recente. A versão do adaptador de recursos IBM WebSphere MQ incluída em WebSphere Application Server 8.0 e 7.0 deve ser usada nestes ambientes.

Conceitos relacionados

“Instrução de suporte do adaptador de recursos do IBM MQ” na página 419
O adaptador de recursos que é fornecido com o IBM MQ 8.0 ou mais recente implementa a especificação do 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.

Informações relacionadas

Requisitos do Sistema para IBM MQ

Determinando o número de conexões TCP/IP criadas do WebSphere Application Server para o IBM MQ

O IBM WebSphere MQ 7.0 introduziu um novo recurso chamado "compartilhando conversas". Usando esse recurso, várias conversas podem compartilhar instâncias do canal MQI, também conhecido como uma conexão TCP/IP.

Sobre esta tarefa

Aplicativos em execução dentro do WebSphere Application Server 7 e 8, que usam o modo normal do provedor de sistemas de mensagens do IBM MQ , usarão automaticamente esse recurso. 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 examinar o número de conversas que são criadas por WebSphere Application Server 7 e 8, é possível determinar o número de instâncias de canal que são 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 298
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 ser criadas por meio de um connection factory é especificado pela propriedade **Maximum connections** do conjunto de conexões do connection factory. 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 a conexão ser encerrada quando o valor da propriedade **Unused timeout** do conjunto livre é 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 Sessões JMS que podem ser criadas por meio de uma única conexão JMS é determinado pela propriedade **Maximum connections** do conjunto de sessões do connection factory. O valor padrão desta propriedade é 10.

Uma conversa começa quando uma sessão JMS é criada pela primeira vez. A conversa permanecerá ativa até que a sessão JMS seja encerrada porque permaneceu no conjunto livre mais tempo 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 o conjunto de conexões **Maximum connections** e as propriedades **Maximum connections** do conjunto de sessões, o número máximo de conversas que podem existir entre o WebSphere Application Server e o 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 de especificação de ativação **Maximum server sessions** especifica o número máximo de sessões do servidor que podem estar ativas em um determinado 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 conversas que podem existir entre o WebSphere Application Server e o 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 483.

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. 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 483.

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 do listener simples que está usando o valor padrão para a propriedade **Maximum sessions**, o número máximo de conversas que podem existir entre o WebSphere Application Server e o IBM MQ para essa porta do 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 483.

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, 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, 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
```

Use aliases de autenticação para proteger a conexão do WebSphere Application Server com 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, um connection factory do provedor de sistemas de mensagens do IBM MQ definido no escopo da Célula do WebSphere Application Server contém informações sobre como se conectar ao gerenciador de filas (`myQM`) usando o transporte BINDINGS. Esse connection factory recebe o nome JNDI `jms/myCF` para identificá-lo exclusivamente.

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 localizados em Programas de saída de segurança do canal.

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 493
- “Usando o connection factory por meio de uma consulta indireta” na página 493

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 70. 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 71. 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 *Aplicativo* e o connection factory é configurado para usar um canal IBM MQ que possui a propriedade MCAUSER desconfigurada	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 *Aplicativo* e o connection factory é configurado para usar um canal IBM MQ que tem a propriedade MCAUSER configurada como um identificador de usuário	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 71. 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 "*Contêiner* e o connection factory é configurado para usar um canal IBM MQ que possui a propriedade MCAUSER desconfigurada	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 a connection factory que possui a propriedade **res-auth** configurada como "*Contêiner* e a connection factory é configurada para usar um canal IBM MQ que possui a propriedade MCAUSER configurada para um identificador de usuário	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 "*Contêiner* e o aplicativo foi configurado com um alias de autenticação e o connection factory está configurado para usar um canal IBM MQ que possui a propriedade MCAUSER desconfigurada	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.
O descritor de implementação do aplicativo contém uma Referência de Recurso para o connection factory que tem a propriedade **res-auth** configurada como *Contêiner* e o aplicativo foi configurado com um alias de autenticação e o connection factory está configurado para usar um canal IBM MQ que possui o MCAUSER configurado para um identificador de usuário	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");
```

Notes:

- 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 tiver executado uma procura direta de um connection factory e, em seguida, chamado esse método, o nome do usuário e a senha que foram transmitidos para o método `createConnection()` para 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 localizados em Programas de saída de segurança do canal.

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");
```

É necessário o prefixo `java:comp/env` para o nome JNDI local, para informar ao servidor de aplicativos que o aplicativo corporativo está executando uma procura indireta.

Quando o aplicativo é implementado, o usuário mapeia o JNDI nome 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 JMS usando o nome local JNDI , 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 usar `ConnectionFactory.createConnection()` e **`res-auth`** for configurado como *Aplicativo*, a identidade do usuário padrão será transmitida para 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` especificar um nome de usuário e uma senha.

Substituindo o alias de autenticação ao usar uma consulta indireta

Se um aplicativo usar uma referência de recurso que tenha a propriedade **`res-auth`** configurada como *Contêiner*, será possível substituir o alias de autenticação usado quando as conexões do JMS forem 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 aplicativos de bean acionado por mensagens implementados em um cluster WebSphere Application Server 7.0 e 8.0 e configurados para executar no modo normal do provedor de sistemas de mensagens do IBM WebSphere 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.

O IBM WebSphere MQ 7.0 introduziu um novo recurso chamado **Asynchronous consume**, que permite que os aplicativos consumam mensagens assincronamente de uma fila usando APIs chamadas **MQCB** e **MQCTL**.

Os aplicativos de bean acionados por mensagens em execução dentro do WebSphere Application Server 7.0 e do 8.0, que usam o modo normal do provedor de sistemas de mensagens do IBM WebSphere MQ , farão uso automaticamente desse 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 WebSphere MQ 7.0 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.

Antes do IBM WebSphere MQ 7.0.1 Fix Pack 6, os gerenciadores de filas favorecerão o primeiro membro de cluster para configurar seu consumidor assíncrono no Destino JMS. Esse membro de cluster será o primeiro a ser notificado quando uma mensagem adequada chegar ao destino JMS. Subsequentemente, o primeiro membro de cluster a iniciar o aplicativo de bean acionado por mensagens processará a maioria de mensagens adequadas que chegar ao destino JMS.

Quando o WebSphere Application Server estiver se conectando a um gerenciador de filas IBM WebSphere MQ 7.0.1 Fix Pack 6 ou posterior, as mensagens que chegam em um destino JMS serão distribuídas mais uniformemente a todos os consumidores assíncronos registrados naquele destino JMS. Para aplicativos de bean acionado por mensagens implementados dentro de um cluster WebSphere Application Server 7.0 e 8.0 , isso significa que as mensagens serão distribuídas mais uniformemente entre os membros do cluster.

Informações relacionadas

Configurando a propriedade 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.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 496.
- 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 497.

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 330).

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 358
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 363
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 496).

Procedimento

1. Crie um JMS Queue Destination que represente o SYSTEM.ADMIN.COMMAND.QUEUE.

   Os aplicativos IBM MQ JMS enviam as mensagens PCF ao SYSTEM.ADMIN.COMMAND.QUEUE e precisam acessar 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 do JMS” na página 128
As mensagens do JMS são compostas por cabeçalho, propriedades e corpo. O JMS define cinco tipos de corpo de 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: No IBM MQ 8.0, `ldap.jar`, `jndi.jar` e `jta.jar` fazem parte do JDK.

Usando os comandos de CL

O CLASSPATH configurado, é para teste com o MQ baseado em Java, JMS com JNDI e JMS sem JNDI.

Se você não usar um arquivo `.profile` em seu diretório `/home/Userprofile`, será necessário configurar as variáveis de ambiente no 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, faça o seguinte:

```
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.mqjms.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.

```
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.mqjms.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, 2023. 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, 2023. 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, 2023.
All Rights Reserved.
WebSphere MQ classes for Java(tm) 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, 2023.
All Rights Reserved.
WebSphere MQ classes for Java(tm) 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 mudanças apropriadas no arquivo `JMSAdmin.config`. Para editar esse arquivo, use o comando **EDTF** (Editar Arquivo) a partir da linha de comando 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, 2023. All Rights Reserved.
Starting WebSphere MQ classes for Java(tm) Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java(tm) 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, 2023. All Rights Reserved.
MQSeries classes for Java(tm) Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
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
$
```

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.

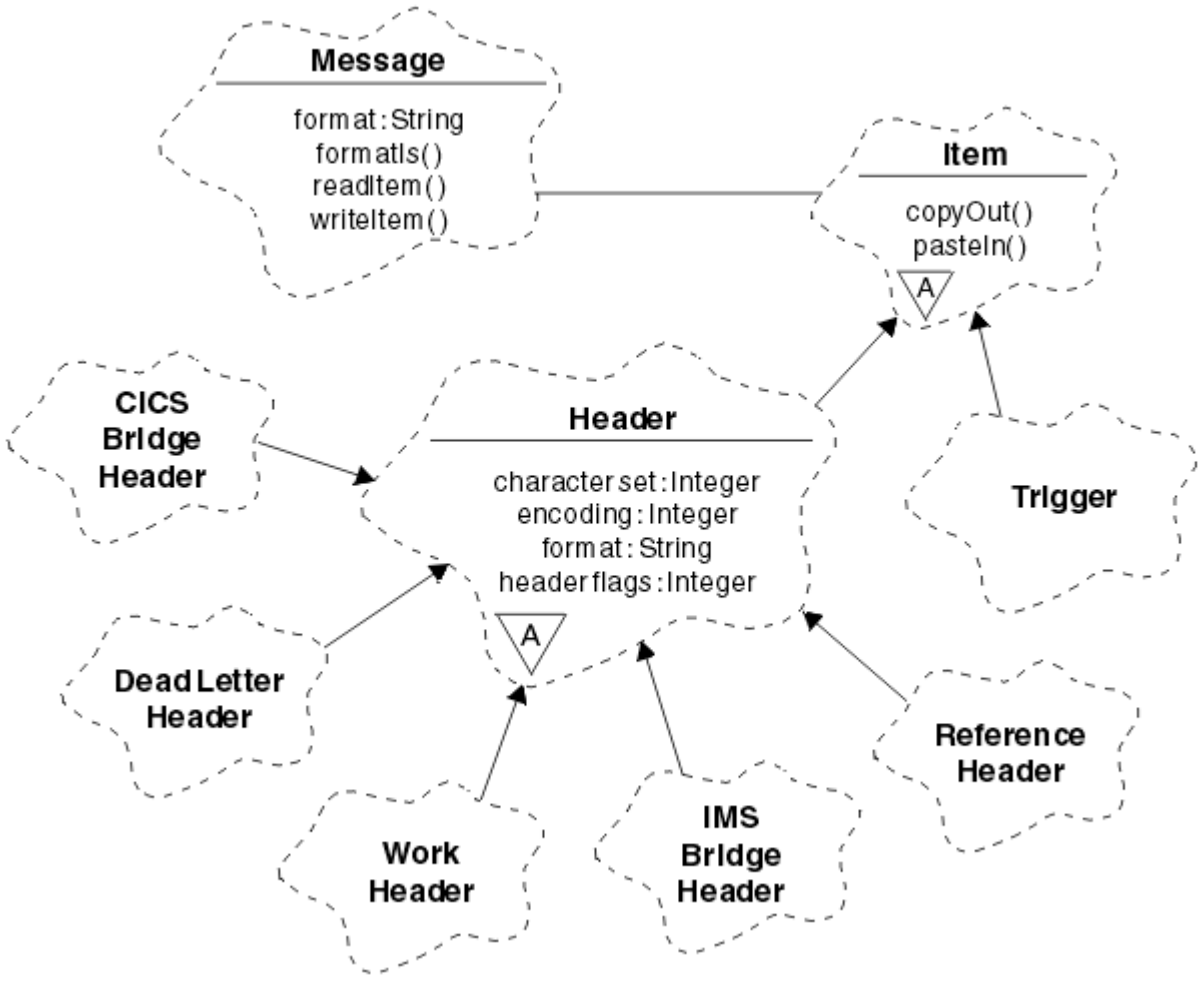


Figura 56. Classes IBM MQ C++ (manipulação de item)

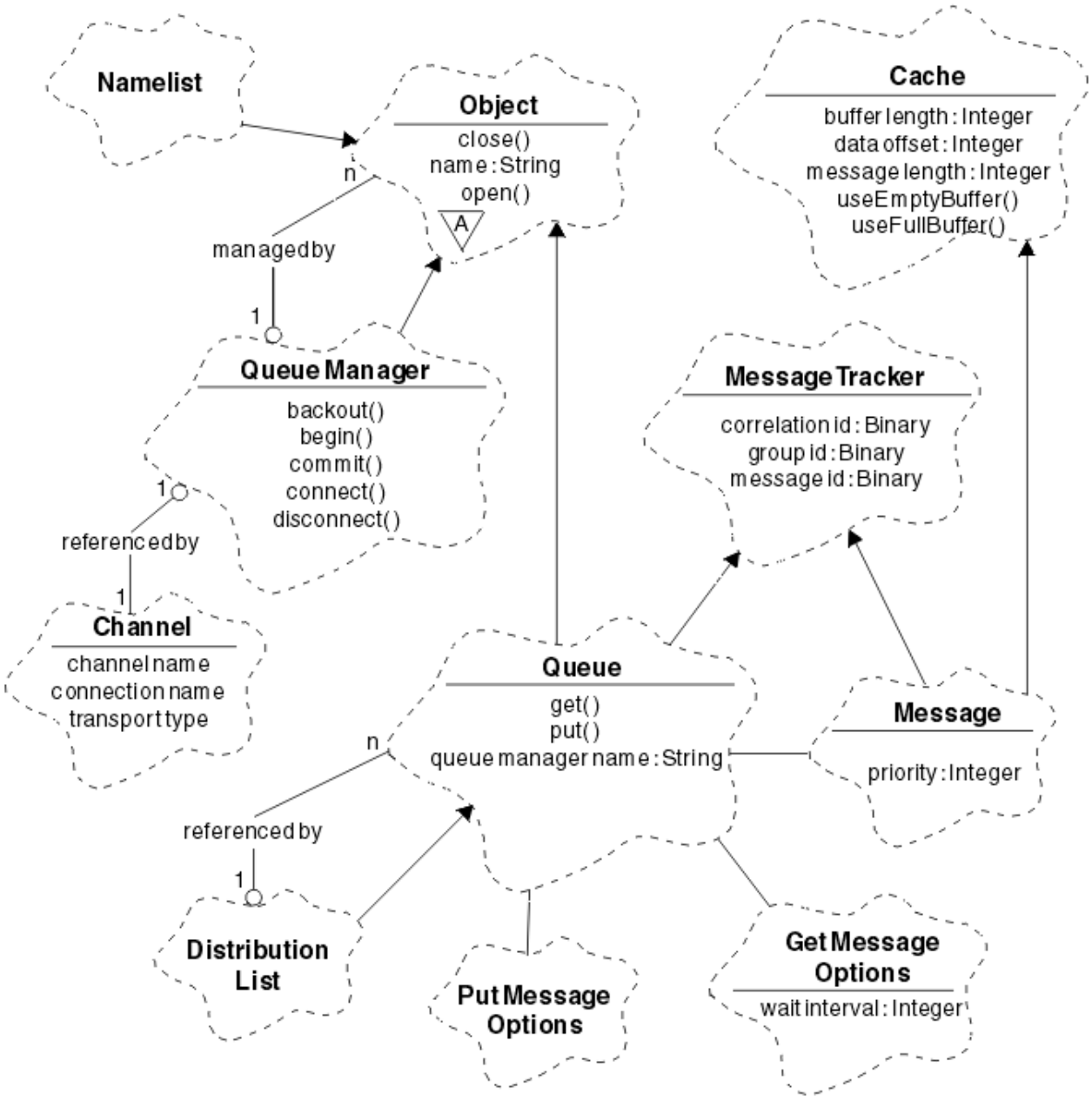


Figura 57. Classes IBM MQ C++ (gerenciamento de fila)

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 57 na página 507) e as classes de manipulação de item (consulte Figura 56 na página 506):

- 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 518), 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

"Programas de amostra C++" na página 508
Quatro programas de amostra são fornecidos, para demonstrar a obtenção e a colocação de mensagens.

"contraprestações sobre linguagem C++" na página 512
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 517
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 523
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.

Informações relacionadas

Visão geral técnica

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 72 na página 509.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Tabela 72. Local de programas de amostra

Ambiente	Diretório que contém a origem	Diretório que contém o construído programas
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
IBM i IBM i IBM i	/QIBM/ProdData/mqm/samp/	(consulte a nota “1” na página 509)
HP-UX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/ samp/bin/ah (consulte a nota “2” na página 509)
z/OS z/OS z/OS	thlqual.SCSQCPPS	Nenhum
Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/ samp/bin/as
Linux	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\ samples	*MQ_INSTALLATION_PATH*\tools\cplus\ samples\bin\vn (consulte a nota “3” na página 509)

Notes:

1. Os programas construídos usando o compilador ILE C++ para o IBM i estão na biblioteca QMQM. Os arquivos de inclusão estão em /QIBM/ProdData/mqm/inc.
2. Os programas construídos usando o compilador HP ANSI C++ estão localizados no diretório *MQ_INSTALLATION_PATH*/samp/bin/ah. Veja informações adicionais na publicação “Construindo programas C++ no HP-UX” na página 524.
3. Os programas construídos usando o Microsoft 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 530.

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

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 “Construindo programas C++ no z/OS Batch, RRS Batch e CICS” na página 532 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 73. arquivos de cabeçalho C/C++	
Nome do arquivo	**Conteúdo**
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.

Se você 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 ao seu programa, como qualquer outro erro de 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.

Construindo programas C++ no AIX

Desenvolva programas IBM MQ C++ no AIX usando o compilador XL C Enterprise Edition.

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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -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 -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

Construindo programas C++ no HP-UX

Construa programas C++ do IBM MQ no HP-UX usando os compiladores aC++ ou aCC.

No HP-UX Itanium, o IBM MQ suporta apenas o tempo de execução padrão. Use o compilador aCC.

- libimqi23bh.sl fornece as classes C++ do IBM MQ para o tempo de execução Padrão.
- Para compatibilidade com liberações anteriores, um link simbólico é fornecido de libimqi23ah.sl para libimqi23bh.sl.

IA64 (IPF)

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: IA64 (IPF)

Aplicativo não encadeado de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqic
```

Aplicativo encadeado de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqic_r -lpthread
```

Aplicativo não encadeado de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh
-lmqic
```

Aplicativo encadeado de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqic_r
-lpthread
```

Servidor: IA64 (IPF)

Aplicativo não encadeado de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqm
```

Aplicativo encadeado de 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqm_r -lpthread
```

Aplicativo não encadeado de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqm
```

Aplicativo encadeado de 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqm_r
-lpthread
```

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)
```

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
```

System x (32 bits)

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: System x (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: System x (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
```

Construindo programas C++ no Solaris

Construir IBM MQ programas C++ no Solaris usando o compilador Sun ONE.

SPARC

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: SPARC

Aplicativo de 32 bits

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicativo de 64 bits

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Servidor: SPARC

Aplicativo de 32 bits

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Aplicativo de 64 bits

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Cliente: x86-64

Aplicativo de 32 bits

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicativo de 64 bits

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Servidor: x86-64

Aplicativo de 32 bits

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Aplicativo de 64 bits

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

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.

No IBM MQ 8.0.0 Fix Pack 4, o produto envia clientes redistribuíveis que contêm 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 no *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
```

Bibliotecas de clientes C++ construídas usando o compilador do Microsoft Visual Studio 2015

V 9.0.1

V 9.0.1 A partir do IBM MQ 9.0.1, o produto fornece bibliotecas do cliente C++ que são construídas com o compilador C++ Microsoft Visual Studio 2015 . Aplicativos que são construídos usando uma liberação do IBM MQ 9.0.1 ou posterior podem usar essas bibliotecas. Essas bibliotecas são fornecidas além das bibliotecas C++ do IBM MQ 9.0.1 existentes que são construídas com o compilador Microsoft Visual Studio 2012 C++.

Ambas as versões de 32 bits e de 64 bits das bibliotecas C++ do IBM MQ são fornecidas. Enquanto as bibliotecas de 32 bits são instaladas na pasta `bin\vs2015`, as bibliotecas de 64 bits são instaladas nas pastas `bin64\vs2015`.

Por padrão, o IBM MQ está configurado para usar as bibliotecas do Microsoft Visual Studio 2012. Para usar as bibliotecas Microsoft Visual Studio 2015 , deve-se configurar a variável de ambiente `MQ_PREFIX_VS_LIBRARIES` usando o comando **setmqenv**ou **setmqinst** .

É possível compilar seus próprios aplicativos de sistema de mensagens com o Microsoft Visual Studio 2017 e vinculá-los com as bibliotecas IBM MQ C/C++ Microsoft Visual Studio 2015 fornecidas.

Usando o IBM MQ com o compilador C++ do Microsoft Visual Studio 2015.

No Microsoft Visual Studio 2015, os módulos de mesclagem do Microsoft instalados com o IBM MQ não contêm mais o código de tempo de execução C inteiro.

Para usar o compilador C++ do Microsoft Visual Studio 2015, deve-se instalar a atualização da base de conhecimento do Microsoft, KB3118401, na página da web Windows 10 Universal C Runtime, se você estiver usando uma versão do Windows antes do Windows 10.

Essa página inclui requisitos do sistema e instruções de instalação.

Se você não instalar o KB3118401, os programas C++ construídos com relação ao Microsoft Visual Studio 2015 (IBM e aqueles para a sua própria empresa) falharão ao serem ativados, geralmente com a mensagem a seguir aparecendo:

```
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.
```

Usando bibliotecas C++ do IBM MQ denominadas de forma diferente

No IBM MQ 8.0.0 Fix Pack 4, o produto fornece algumas bibliotecas de cliente C++ adicionais que são nomeadas de forma diferente. Essas bibliotecas são construídas com o compilador Microsoft Visual Studio 2012 C++. As bibliotecas do Microsoft Visual Studio 2015 também estão disponíveis. Essas bibliotecas são fornecidas além das bibliotecas C++ existentes que também são construídas com o compilador C++ do Microsoft Visual Studio 2012 ou Microsoft Visual Studio 2015. Como essas bibliotecas C++ adicionais do IBM MQ têm nomes diferentes, é possível executar aplicativos C++ do IBM MQ construídos usando o C++ do IBM MQ e compilados com o Microsoft Visual Studio 2012 e versões anteriores do produto na mesma máquina.

As bibliotecas adicionais do Microsoft Visual Studio 2012 são nomeadas como a seguir:

- `imqb23vnvs2012.dll`
- `imqc23vnvs2012.dll`
- `imqs23vnvs2012.dll`
- `imqx23vnvs2012.dll`

As bibliotecas adicionais do Microsoft Visual Studio 2015 são nomeadas como 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 são instaladas sob a pasta `bin` e as bibliotecas de 64 bits são instaladas sob a pasta `bin64`. Bibliotecas de importação correspondentes são instaladas sob os diretórios `Tools\lib` e `Tools\lib64`.

Se seu aplicativo usar arquivos `imq*vs2012.lib`, deve-se compilá-lo usando o compilador Microsoft Visual Studio 2012. Para executar aplicativos C++ do IBM MQ que são compilados com o Microsoft Visual Studio 2012 e aplicativos que são compilados com uma versão anterior do produto na mesma máquina, a variável de ambiente PATH deve ser prefixada conforme mostrado nos exemplos a seguir:

- Para aplicativos de 32 bits:

```
SET PATH=installation folder\bin\vs2008;%PATH%
```

- Para aplicativos de 64 bits:

```
SET PATH=installation folder\bin64\vs2008;%PATH%
```

Informações relacionadas

Windows: mudanças para IBM MQ 8.0

Construindo programas C++ no z/OS Batch, RRS Batch e CICS

Construção dos programas C++ do IBM MQ no z/OS para o lote, lote RRS ou ambientes CICS e execução dos programas de amostra.

É possível gravar programas C++ para três dos ambientes que o IBM MQ for z/OS suporta:

- Batch
- RRS batch
- CICS

Compile, pré-vinculação e vincular

Crie um aplicativo z/OS compilando, pré-vinculando e linkeditando seu código-fonte C++.

IBM MQ C++ for z/OS é implementado como DLLs do z/OS para o C++ IBM para linguagem z/OS. Ao usar as DLLs, você irá concatenar as bibliotecas auxiliares de módulos de definição fornecidas com a saída do compilador no tempo de pré-link. Isso permite que o vinculador verifique suas chamadas para as funções de membro do C++ IBM MQ.

Nota: Há três conjuntos de bibliotecas auxiliares de módulos para cada um dos três ambientes.

Para construir um aplicativo C++ IBM MQ for z/OS, crie e execute a JCL. Use o seguinte procedimento:

1. Se seu aplicativo for executado no CICS, use o procedimento fornecido pelo CICS para converter os comandos do CICS em seu programa.

Além disso, para os aplicativos CICS é necessário:

a. Inclua a biblioteca SCSQLOAD para a concatenação DFHRPL.

b. Defina o grupo CSQCAT1 CEDA usando o membro IMQ4B100 na biblioteca SCSQPROC.

c. Instale o CSQCAT1.

2. Compile o programa para produzir o código do objeto. A JCL para sua compilação deve incluir instruções que tornam os arquivos de definição de dados de produto disponíveis para o compilador. As definições de dados são fornecidas nas bibliotecas do IBM MQ for z/OS a seguir:

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Certifique-se de especificar a opção do compilador `/cxx`.

   Nota: O nome **thlqual** é o qualificador de alto nível da biblioteca de instalação do IBM MQ no z/OS.

3. Pré-vincule o código do objeto criado na etapa "2" na página 533, incluindo a seguinte definição das bibliotecas auxiliares de módulos, que são fornecidas em **thlqual**.SCSQDEFS:

   a. imqs23dm e imqb23dm para lote

   b. imqs23dr e imqb23dr para lote RRS

   c. imqs23dc e imqb23dc para CICS

   Estas são as DLLs correspondentes.

   a. imqs23im e imqb23im para lote

   b. imqs23ir e imqb23ir para lote RRS

   c. imqs23ic e imqb23ic para CICS

4. Pré-edite o código do objeto criado na etapa "3" na página 533, para produzir um módulo de carregamento e armazene-o na biblioteca de carregamento de seu aplicativo.

Para executar programas em lote ou em lote RRS, inclua as bibliotecas **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD na concatenação do conjunto de dados JOBLIB ou STEPLIB.

Para executar um programa CICS, primeiro faça com que seu administrador do sistema o defina para CICS como uma transação e um programa do IBM MQ. É possível, então, executá-lo da maneira usual.

Execute os programas de amostra

Os programas são descritos em "Programas de amostra C++" na página 508.

Os aplicativos de amostra são fornecidos na forma de origem apenas. Os arquivos são:

Tabela 74. Arquivos do programa de amostra doz/OS

Amostra	Programa de origem (na biblioteca thlqual.SCSQCPPS)	JCL (na biblioteca thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

Para executar as amostras, compile e linkedite-os como qualquer programa C++ (consulte "Construindo programas C++ no z/OS Batch, RRS Batch e CICS" na página 532). Use a JCL fornecido para construir e executar uma tarefa em lote. Deve-se inicialmente customizar a JCL, seguindo o comentário incluído nele.

Construindo programas C++ no z/OS UNIX System Services

Construa programas IBM MQ C++ no z/OS for Unix System Services.

Para construir um aplicativo sob o shell do UNIX System Services, deve-se fornecer ao compilador acesso aos arquivos include IBM MQ (localizados em `thlqual.SCSQC370` e `hlqual.SCSQHPPS`) e vincular a dois dos sidedecks DLL (localizados em `thlqual.SCSQDEFS`). No tempo de execução, o aplicativo precisa de acesso aos conjuntos de dados IBM MQ `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`, e um dos conjuntos de dados específicos de linguagem, como `thlqual.SCSQANLE`[6].

Compilando

1. Copie a amostra para o HFS usando o comando **oput** do TSO ou use FTP. O restante deste exemplo assume que você tenha copiado a amostra para um diretório chamado `/u/fred/sample` e denominado-a `imqwrld.cpp`.
2. Efetue login no shell do UNIX System Services e mude para o diretório no qual você colocou a amostra.
3. Configure o compilador C++ para que possa aceitar a biblioteca auxiliar de módulos DLL e arquivos .cpp como entrada:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Compile e vincule o programa de amostra. O comando a seguir vincula o programa às bibliotecas auxiliares de módulos em lote; as bibliotecas auxiliares de módulos em lote RRS podem ser usadas em vez disso. O caractere \ é usado para dividir o comando em mais de uma linha. Não insira esse caractere; insira o comando como uma única linha:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

Para obter mais informações sobre o comando **oput** do TSO, consulte a *Referência de comando do z/OS UNIX System Services*.

Também é possível usar o utilitário make para simplificar a construção de programas C++. Aqui está um makefile de amostra para construir o programa de amostra C++ HELLO WORLD. Ele separa os estágios de compilação e vinculação. Configure o ambiente como na etapa “3” na página 534 antes executar make.

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Consulte *Ferramentas de programação do z/OS UNIX System Services* para obter mais informações sobre como usar make.

Executando

1. Efetue login no shell do UNIX System Services e mude para o diretório em que você construiu a amostra.
2. Configure a variável de ambiente STEPLIB para incluir os conjuntos de dados do IBM MQ:

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

[6] É possível vincular a qualquer um dos sidedecks listados em " Pré-vincular o código do objeto para executar o serviço do sistema UNIX em qualquer um dos três ambientes, “Construindo programas C++ no z/OS Batch, RRS Batch e CICS” na página 532

3. Execute a amostra:

```
/u/fred/sample:> ./imqwrld
```

Desenvolvendo aplicativos do .NET

IBM MQ classes for .NET permite que um programa gravado na estrutura de programação do .NET se conecte ao IBM MQ como um IBM MQ MQI client ou se conecte diretamente a um servidor IBM MQ.

Se você tiver aplicativos que usem Microsoft .NET Framework e desejar aproveitar os recursos do IBM MQ, deve-se usar o IBM MQ classes for .NET.

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();
```

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 detalhes sobre essas classes, consulte Classes e interfaces do IBM MQ .NET.

Antes de poder compilar qualquer aplicativo que você grave, deve-se ter um .NET Framework instalado. Para obter instruções sobre como instalar o IBM MQ classes for .NET e o .NET Framework, veja “Instalando o IBM MQ classes for .NET” na página 536.

Conceitos relacionados

“Opções para conexão a um gerenciador de filas” na página 536
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.

“Gravando e implementando programas do IBM MQ .NET” na página 550
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.

“Desenvolvendo aplicativos do Microsoft Windows Communication Foundation (WCF) com o IBM MQ” na página 1266
O canal customizado do Microsoft Windows Communication Foundation (WCF) para o IBM MQ envia e recebe mensagens entre clientes e serviços do WCF.

“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.

Informações relacionadas

Visão geral técnica

Resolução de problemas do IBM MQ .NET

Introdução ao IBM MQ classes for .NET

IBM MQ classes for .NET permite que um programa gravado na estrutura de programação do .NET se conecte ao IBM MQ como um IBM MQ MQI client ou se conecte diretamente a um servidor IBM MQ.

Opções para conexão 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 informações adicionais sobre elas, consulte “Conexões do Cliente Gerenciadas” na página 550.

Instalando o IBM MQ classes for .NET

O IBM MQ classes for .NET, incluindo amostras, é instalado com o IBM MQ. Há um pré-requisito do Microsoft.NET Framework.

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 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 um ambiente com diversas instalações, 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 você primeiro desinstale o pacote de suporte. O recurso IBM MQ classes for .NET instalado com o IBM MQ contém a mesma funcionalidade que o pacote de suporte.

Aplicativos de amostra, incluindo arquivos de origem, também são fornecidos; consulte “Aplicativos de amostra” na página 537.

Para executar o IBM MQ classes for .NET em plataformas de 64 bits ou de 32 bits, deve-se ter instalado o Microsoft.NET Framework V3.5 ou posterior.

Nota: Se o Microsoft.NET Framework v3.5 ou superior não for instalado antes da instalação do IBM MQ 8.0, a instalação do produto IBM MQ continuará sem erro, mas o IBM MQ classes for .NET não ficará disponível. Se o .NET Framework for instalado após a instalação do IBM MQ 8.0, os conjuntos do IBM MQ.NET deverão ser registrados executando o script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd , em que *WMQInstallDir* é o diretório no qual o IBM MQ 8.0 está instalado Este script instala as montagens necessárias no Global Assembly Cache (GAC). Um conjunto de arquivos amqi*.log gravando as ações executadas é criado no diretório %TEMP%.

Para obter informações sobre como usar o canal customizado do IBM MQ para o Microsoft WCF com o .NET 3, consulte "Desenvolvendo aplicativos do Microsoft Windows Communication Foundation (WCF) com o IBM MQ" na página 1266

Aplicativos de amostra

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.

Cinco 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.

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

Para construir os aplicativos de amostra, um arquivo em lote foi 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
```

com

```
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
```

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” na página 541
- 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

Novas amostras do produto, `SimpleXAPut` e `SimpleXAGet`, estão disponíveis em `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 544

Transações distribuídas no modo gerenciado

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

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 a montagem 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 `WebSphere MQ\tools\dotnet\samples\cs\base`

Procedimento

Os aplicativos podem ser executados com os parâmetros da linha de comandos 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

Esta seção descreve o processo de recuperação de transações no IBM MQ .NET XA usando o modo gerenciado.

Visão Geral

No processamento de transações distribuído, as transações podem ser concluídas com sucesso. No entanto, pode haver cenários em que uma transação pode falhar por várias razões. Essas razões podem incluir uma falha do sistema, falha de hardware, erro de rede, dados incorretos ou inválidos, erros de aplicativo 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.

Modelo de recuperação

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 no IBM MQ .NET é um processo de dois estágios.

1. O registro de informações de recuperação da transação.
   - 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.
2. Recuperando Transações usando um WmqDotnetXAMonitor aplicativo do monitor.
   - WmqDotnetXAMonitor é um aplicativo gerenciado pelo .NET que processa mensagens em SYSTEM.DOTNET.XARECOVERY.QUEUE e recupera transações incompletas

   Se o MCA não conseguir colocar a mensagem na fila de destino, ele gera um relatório de exceções que contém a mensagem original e a coloca em uma fila de transmissão a 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)

SYSTEM.DOTNET.XARECOVERY.QUEUE

Essa é uma fila do sistema que contém informações de recuperação da transação de transações incompletas. Essa fila é criada quando um gerenciador de filas é criado.

Nota: Não é necessário excluir a fila SYSTEM.DOTNET.XARECOVERY.QUEUE.

Aplicativo WMQDotnetXAMonitor

O aplicativo IBM MQ .NET XA Monitor monitora um determinado gerenciador de filas e recupera transações incompletas, se houver alguma. A seguir, são consideradas como transações incompletas e são recuperadas:

Transações Incompletas

- 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.

  Aplicativo do monitor deve ser executado a partir do mesmo sistema no qual o aplicativo cliente IBM MQ .NET está em execução. Se houver aplicativos em execução em sistemas múltiplos que se conectam ao mesmo gerenciador de filas, o aplicativo monitor deverá ser executado a partir de todos os sistemas. Embora cada máquina cliente tenha um aplicativo monitor em execução para recuperar o aplicativo, cada monitor deve ser capaz de identificar a mensagem que corresponde à transação que o MS DTC local do monitor atual estava coordenando para que possa realistá-la e concluí-la.

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 Aplicativo usando um único DTC e uma única instância do gerenciador de filas:** nesse caso de uso, ao se conectar ao gerenciador de filas e executar a Unidade de trabalho (UoW) sob a transação e se a transação falhar e torna-se incompleta, o aplicativo do monitor recuperará a transação e irá concluí-la.

  Nesse caso de uso, haverá uma única instância de aplicativo do monitor em execução, visto que um único gerenciador de filas está associado às transações.
- **Vários aplicativos IBM MQ usando um único DTC e uma única instância do gerenciador de filas:** nesse caso de uso, há mais de um aplicativo WMQ sob um único DTC e todos estão se conectando ao mesmo gerenciador de filas e executando UoW sob transações.

  Se as transações falharem e se tornarem incompletas, o aplicativo de monitor irá recuperá-los e conclui as transações referentes a todos os aplicativos.

  Nesse caso de uso, um único aplicativo de monitor é executado, como um gerenciador de filas é usado em transações.

- **Vários aplicativos IBM MQ, vários DTCs, instâncias diferentes do gerenciador de filas:** nesse caso de uso, há mais de um aplicativo WMQ sob diferentes DTCs (ou seja, cada aplicativo está sendo executado em uma máquina diferente) e se conectando a gerenciadores de filas diferentes.

  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.

  Nesse caso de uso, haverá apenas uma instância de aplicativo de monitor em execução por cliente (usuário ou computador) à medida que cada cliente tenha seu próprio gerenciador de filas usado em transações.
- **Vários aplicativos IBM MQ, vários DTCs, várias mesmas instâncias do gerenciadores de filas:** nesse caso de uso, há mais de um aplicativo WMQ sob DTCs diferentes (ou seja cada aplicativo está sendo executado em uma máquina diferente) e todos estão se conectando ao 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.

  Nesse caso de uso, haverá apenas uma única instância de aplicativo de monitor em execução por cliente (usuário ou computador), à medida que cada cliente tenha sua própria associação do gerenciador de filas usada em transações.
- **Vários aplicativos IBM MQ, único DTC, instâncias diferentes do gerenciadores de filas:** nesse caso de uso, há mais de um aplicativo WMQ sob um único DTC (ou seja, em um computador, há mais de um aplicativo WMQ em execução) e se conectando a gerenciadores de filas diferentes.

  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 de monitor não estiver sendo executado em segundo plano, será possível iniciá-lo.

Usando o aplicativo WMQDotnetXAMonitor
O aplicativo WMQDotnetXAMonitor deve ser executado manualmente. Ele pode ser iniciado a qualquer momento. Será possível iniciá-lo quando você vir as mensagens em SYSTEM.DOTNET.XARECOVERY.QUEUE ou mantê-lo em execução em segundo plano antes de executar qualquer outro trabalho transicional com os aplicativos escritos usando classes IBM MQ .NET.

Utilize o comando a seguir para iniciar o aplicativo de monitor

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
```

Em que

- **-m *QueueManagerName***
  Nome do gerenciador de filas.

  Opcional

  -n *ConnectionName*
  Nome da conexão em formato do host (porta). O nome da conexão pode conter mais de um nome de conexão. Vários nomes de conexão devem ser fornecidos em uma lista separada por vírgulas, por exemplo `localhost (1414), localhost (1415), localhost (1416)`. O aplicativo de monitor executa a recuperação para cada um dos nomes de conexão especificados na lista separada por vírgula.

-c *ChannelName*
Nome do canal.

-i
Conclusão da ramificação heurística.

Opcional

O aplicativo de monitor executa as ações a seguir:

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, o monitor de XA procura na fila mensagens e verifica se a mensagem satisfaz os critérios de transação incompleta.
3. Se qualquer uma das mensagens satisfizer os critérios de transação incompleta, o monitor a retira e recupera as informações de recuperação da transação.
4. Em seguida, determina se as informações de recuperação estão relacionadas ao MS DTC local. Se estiverem, então, ele continua para recuperar a transação. Caso contrário, ele volta para procurar a próxima mensagem.
5. Ele então faz chamadas ao gerenciador de filas para recuperar a transação incompleta.

Configurações do arquivo de configuração de aplicativo WmqDotNETXAMonitor
Para monitorar o aplicativo, as entradas também podem ser fornecidas usando o arquivo de configuração de aplicativo. Um arquivo de configuração de aplicativo de amostra é enviado com o IBM MQ .NET. Esse arquivo pode ser modificado de acordo com seus requisitos.

O arquivo de configuração de aplicativo tem a precedência mais alta ao considerar os valores de entrada. Se ambos os valores de entrada forem fornecidos tanto na linha de comandos como no arquivo de configuração de aplicativo, então os valores da configuração do aplicativo serão consideradas.

Arquivo de configuração de aplicativo de amostra.

```
<?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>
```

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.

A documentação do IBM MQ contém informações apenas nas linguagens C++, C# e Visual Basic.

Essa coleta de tópicos fornece informações para ajudar na gravação de aplicativos para interagir com os 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.

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. Esse arquivo geralmente é chamado de `mqclient.ini`, mas é possível especificar um nome de arquivo diferente. Para obter informações adicionais sobre o arquivo de configuração do cliente, veja Configurando um cliente usando um arquivo de configuração.

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 75. Atributos do arquivo de configuração do cliente que são relevantes para o IBM MQ classes for .NET

Sub-rotina	**Atribuir**
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ReconDelay
CHANNELS	DefRecon
CHANNELS	MQReconnectTimeout
CHANNELS	ServerConnectionParms
CHANNELS	Put1DefaultAlwaysSync
CHANNELS	PasswordProtection
ClientExitPath	Caminho Padrão das Saídas
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
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 arquivo de configuração de aplicativo .NET 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, 2023
// ============================================================================
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 de porta
- Saídas do canal
- Parâmetros de SSL
- ID do Usuário e Senha

Para obter informações completas sobre a classe MQEnvironment, consulte a classe MQEnvironment.NET

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 de caracteres

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 .NET é inerentemente multiencadeado. O IBM MQ classes for .NET permite que um objeto gerenciador de filas seja compartilhado entre vários encadeamentos, mas garante que todo o acesso ao gerenciador de filas de destino esteja 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 as classes do .NET para IBM MQ. 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 os ChannelDefinitionDirectory chaves para especificar o diretório no qual 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.

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 978).

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 560 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 560 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.

Informações relacionadas

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 a TLS para o cliente não gerenciado do .NET é baseado no C MQI e GSKit. O C MQI manipula as operações do TLS e o GSKit implementa os protocolos de soquete seguro do 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_SHA";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
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 iniciado por QMGR e pelo menos dois nomes de Unidades organizacionais, 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.

Suporte de TLS para o cliente .NET gerenciado

O cliente .NET gerenciado usa as bibliotecas do 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 do .NET Framework é .NET Framework v3.5. O nível de suporte ao Algoritmo de cifra é baseado no nível do .NET Framework que o aplicativo está usando.

- Para aplicativos baseados no .NET Framework nível 3.5 e v4.0, os protocolos de soquete seguro disponíveis são SSL v3.0 e TSL v1.0.
- Para aplicativos baseados no .NET Framework nível 4.5, os protocolos de soquete seguro disponíveis são SSLv3.0, TLS v1.1 e TLSv1.2.

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 567.

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 567 e “Mapeamentos CipherSpec para o cliente gerenciado .NET” na página 569.

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 570.

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 570.

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 571.

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 reconfiguração ou renegociação de chave secreta, para consistência com outros clientes IBM MQ, o cliente .NET gerenciado permite que aplicativos configurem SSLKeyResetCount. Para obter informações adicionais, consulte “Reconfiguração ou renegociação de chave secreta” na página 572.

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 572.

Suporte de saída de segurança do IBM MQ
A classe SSLStream fornece suporte limitado para saídas de segurança do IBM MQ. Consultar certificados locais e remotos para obter SSLPeerNamePtr(Subject DN) e SSLRemCertIssNamePtr (Issuer DN) é possível porque 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 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 566.

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 569.

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 SSL3.0 ou TLS1.0 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 um CipherSpec for configurado na política de grupo do Windows, o mesmo CipherSpec deverá ser configurado para o valor da propriedade SSLCipherSpec no canal SVRCONN e no aplicativo. Se a política de grupo do Windows estiver configurada para o padrão, ou seja, a política de grupo não estiver ativada/editada para a configuração CipherSpec, os aplicativos deverão configurar o mesmo valor padrão do CipherSpec a partir da configuração TLS da política de grupo do Windows na classe MQEnvironment ou nas propriedades de hashtable do construtor de MQQueueManager.

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 556
Antes de usar a conexão do cliente para se conectar a um gerenciador de filas, deve-se configurar o ambiente do IBM MQ.

Informações relacionadas

Especificando CipherSpecs

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 falha com 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 SSLCipherSpec, a versão de SSLProtocol pode ser TLS v1.0 ou TLS v1.2, dependendo da versão do Microsoft.NET Framework que está sendo usada.

Certifique-se de fornecer o valor SSLCipherSpec correto, já que a especificação de um valor incorreto pode resultar no uso dos protocolos SSL3.0 ou TLS1.0.

Tabela 76. 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
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

Tabela 76. Tabela de mapeamento de IBM MQ e Microsoft.NET (continuação)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Versão do TLS**
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_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

Notes:

1. 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.

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.
- "*SYSTEM": 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 os certificados de clientes, o cliente .NET gerenciado pelo IBM MQ acessa o keystore do Windows e carrega todos os certificados do cliente correspondidos pelo rótulo certificado ou pela sequência.

Ao selecionar um certificado para usar, o cliente .NET gerenciado pelo IBM MQ sempre usará o primeiro certificado correspondente para o handshake de TLS SSLStream.

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.

Informações relacionadas

Classe MQEnvironment do .NET

Conectando um Cliente a um Gerenciador de Filas de Forma Segura

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 de 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.

Informações relacionadas
Classe MQEnvironment do .NET

Reconfiguração ou renegociação de chave secreta
A classe SSLStream não suporta reconfiguração ou renegociação de chave secreta. No entanto, para ficar consistente com outros clientes do IBM MQ, o cliente .NET gerenciado pelo IBM MQ permite que os aplicativos configurem 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.

Informações relacionadas
Reconfigurando as chaves secretas 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 definida 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 uma das ferramentas a seguir fornecidas com o IBM MQ:

      Use a GUI **strmqikm** ou, na linha de comandos, use **runmqckm** ou **runmqakm**. Para obter mais informações sobre o uso dessas ferramentas, consulte Usando **runmqckm**, **runmqakm** e **strmqikm** para gerenciar certificados digitais.
   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 o GSKit, conforme descrito em Conectando um cliente a um gerenciador de filas de forma segura.
   b) Configure os atributos TLS do canal SVRCONN:

      - Configure **SSLCAUTH** para "REQUIRED/OPTIONAL".
      - Configure **SSLCIPH** para um CipherSpec apropriado.

      Para obter mais informações, consulte “Ativando o TLS para o cliente .NET não gerenciado” na página 565.
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 de SSLProtocol pode ser TLS v1.0 ou TLS v1.2, 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 574.

Informações relacionadas

Classe MQEnvironment do .NET

KeyResetCount (MQLONG)

Federal Information Processing Standards (FIPS) para UNIX, Linux, and Windows

Gravando um aplicativo simples

Dicas para escrever um aplicativo .NET TLS simples gerenciado por IBM MQ, incluindo exemplos para configurar as propriedades SSL para as connection factories, criando uma instância, conexão, sessão e destino do gerenciador de filas e enviando 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 572.

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

Se você não incluir uma seção de configuração de diagnósticos do sistema no arquivo de configuração do aplicativo, o cliente .NET gerenciado por IBM MQ não capturará nenhum evento, rastreio ou ponto de depuração relativos 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.

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 a localização dos aplicativos de amostra. *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual IBM MQ está instalado.

Tabela 77. 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: See Features that can be used only with the primary installation on Windows for important information.

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, 2023.                               */
/*********************************************************************/
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, 2023.                               */
/*********************************************************************/

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, 2023.                               */
/*********************************************************************/

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

No IBM MQ 8.0.0 Fix Pack 2, o cliente IBM MQ .NET oferece a habilidade de compactar e implementar uma montagem do IBM MQ .NET sem a necessidade de usar a instalação completa IBM MQ do cliente em sistemas de produção para executar seus aplicativos.

Sobre esta tarefa

A partir de IBM MQ 8.0.0 Fix Pack 2, você pode construir seus aplicativos IBM MQ .NET em uma máquina onde o cliente completo IBM MQ é instalado e posteriormente empacotar o conjunto IBM MQ .NET, ou seja, `amqmdnet.dll`, juntamente com seu aplicativo e implementá-lo em sistemas de produção.

Os aplicativos do que você constrói e implementa podem ser aplicativos tradicionais Windows .NET, 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

- Consulte o conjunto `amqmdnet.dll` em seu aplicativo da mesma maneira que você fez com liberações anteriores.

  Configure a propriedade **CopyLocal** do conjunto `amqmdnet` para `True` para assegurar que o conjunto `amqmdnet` seja copiado no 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

- Para iniciar e parar o rastreio, use o arquivo de configuração de aplicativo e um arquivo de configuração de rastreio específico do IBM MQ.

  Nota: As seguintes etapas para gerar o rastreio se aplicam ao cliente gerenciado redistribuível do .NET, bem como o cliente independente do .NET.

  Deve-se usar o arquivo de configuração de aplicativo e um arquivo de configuração de rastreio específico IBM MQ porque, como não há nenhuma instalação completa do cliente IBM MQ , as ferramentas padrão que são usadas para iniciar e parar o rastreio **strmqtrc** e **endmqtrc**não estão disponíveis.

  Arquivo de configuração do aplicativo (`app.config` ou `web.config`)
  Os aplicativos precisam definir a propriedade **MQTRACECONFIGFILEPATH** sob a seção `<appSettings>` do arquivo de configuração de aplicativo, ou seja, o arquivo `app.config` ou `web.config` . (O nome real do arquivo de configuração de aplicativo depende do nome do aplicativo.) O valor da propriedade **MQTRACECONFIGFILEPATH** especifica o caminho para o local do arquivo de configuração de rastreio específico do IBM MQ, `mqtrace.config`, conforme mostrado no exemplo a seguir:

  ```
  <appSettings>
  <add key="MQTRACECONFIGFILEPATH" value="C:\MQTRACECONFIG"/>
  </appSettings>
  ```

  O rastreio está desativado se o arquivo `mqtrace.config` não foi localizado no caminho que está especificado no arquivo de configuração do aplicativo especificado. No entanto, o First Failure Support Technology (FFST) e logs de erro são criados no diretório do aplicativo, se o aplicativo tiver autoridade de gravar no diretório atual.

  arquivo de configuração de rastreio específico doIBM MQ (`mqtrace.config`)

  O arquivo `mqtrace.config` é um arquivo XML que define propriedades para iniciar e parar o rastreio, o caminho para os arquivos de rastreio e o caminho para os logs de erro. A tabela a seguir descreve essas propriedades.

Tabela 78. As propriedades definidas no arquivo mqtrace.config	
Atribuir	**Descrição**
MQTRACELEVEL	0: interrompe o rastreio – esse é o valor padrão. 1: inicia o rastreio com menos detalhes. 2: inicia o rastreio com detalhes completos - recomendado.
MQTRACEPATH	Aponta para uma pasta na qual os arquivos de rastreio serão criados. O diretório atual do aplicativo é usado se o caminho estiver em branco ou o atributo **MQTRACEPATH** não estiver definido.
MQERRORPATH	Aponta para uma pasta na qual os arquivos de log de erros serão criados. O diretório atual do aplicativo é usado se o caminho estiver em branco ou o atributo **MQERRORPATH** não estiver definido.

O exemplo a seguir mostra um arquivo `mqtrace.config` de amostra:

```
<?xml version="1.0" encoding="utf-8"?>
<traceSettings>
  <MQTRACELEVEL>2</MQTRACELEVEL>
  <MQTRACEPATH>C:\MQTRACEPATH</MQTRACEPATH>
  <MQERRORPATH>C:\MQERRORLOGPATH</MQERRORPATH>
</traceSettings>
```

O rastreio pode ser iniciado e interrompido dinamicamente quando um aplicativo está em execução, alterando o valor do atributo **MQTRACELEVEL** no arquivo `mqtrace.config`.

O aplicativo em execução deve ter permissões de criação e gravação para a pasta especificada pelo atributo **MQTRACELEVEL** para gerar arquivos de rastreio. Os aplicativos que estão em execução em um ambiente Microsoft Azure PaaS também devem garantir que as permissões de acesso semelhantes, desde que os aplicativos da web que usam um conjunto IBM MQ .NET em execução no Microsoft Azure PaaS pode não ter permissões de criação e gravação. A geração do rastreio, first failure data capture (FDC) e logs de erro falhará se o aplicativo não tiver as permissões necessárias de criação e gravação para a pasta especificada.

Ativando redirecionamento de ligação

- 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

“Usando o aplicativo WMQDotnetXAMonitor” na página 548
O aplicativo WMQDotnetXAMonitor deve ser executado manualmente. Ele pode ser iniciado a qualquer momento. Será possível iniciá-lo quando você vir as mensagens em SYSTEM.DOTNET.XARECOVERY.QUEUE ou mantê-lo em execução em segundo plano antes de executar qualquer outro trabalho transicional com os aplicativos escritos usando classes IBM MQ .NET.

Informações relacionadas

Componentes e recursos do IBM MQ

Clientes redistribuíveis

Aplicativo de tempo de execução do.NET - Windows somente

Usando a Interface de modelo de objeto do componente (IBM MQ Automation Classes for ActiveX)

O IBM MQ Automation Classes for ActiveX (MQAX) são os componentes ActiveX que fornecem as classes que podem ser usadas em seu aplicativo para acessar o IBM MQ.

V 9.0.0 No IBM MQ 9.0, o suporte para o MicrosoftActive X foi descontinuado. As classes do IBM MQ para .NET são a tecnologia de substituição recomendada. Para obter mais informações, consulte Desenvolvendo aplicativos .NET.

MQAX requer um ambiente do IBM MQ e um aplicativo correspondentes IBM MQ com a qual se comunicar.

Ele fornece ao aplicativo ActiveX a capacidade de executar transações e acesso a dados em qualquer um de seus sistemas corporativos que você pode acessar através do IBM MQ.

IBM MQ Classes de automação para ActiveX:

- Conceda o acesso às funções e recursos da API do IBM MQ, permitindo a interconectividade completa para outras plataformas do IBM MQ.
- Conformidade com as convenções normais esperadas de um componente ActiveX.
- Conformidade para o modelo de objeto do IBM MQ, também disponível para .NET, C++, Java e LotusScript.

São fornecidas amostras iniciais do MQAX. É possível usar essas amostras inicialmente para verificar se a instalação do MQAX é bem-sucedida e se você tem o ambiente básico do IBM MQ no local. As amostras também demonstram como o MQAX pode ser usado.

Script de ActiveX e COM

O Modelo de objeto do componente (COM) é um modelo de programação baseado no objeto definido por Microsoft. Ele especifica como os componentes de software podem ser fornecidos de maneira que os permita localizar e se comunicam uns com os outros independentemente da linguagem do computador na qual estão gravados ou seus locais.

ActiveX é um conjunto de tecnologias, baseadas em COM, que integra o desenvolvimento de aplicativos, componentes reutilizáveis e tecnologias da Internet nas plataformas Microsoft Windows. Os componentes ActiveX fornecem interfaces que podem ser acessadas dinamicamente por aplicativos. Um cliente de script ActiveX é um aplicativo, por exemplo um compilador, que pode construir ou executar um problema ou script que usa as interfaces fornecidas pelos componentes ActiveX (ou COM).

Suporte ao ambiente do IBM MQ

IBM MQ Automation Classes for ActiveX pode ser usado apenas com os clientes de script ActiveX de **32 bits**.

O componente COM pode ser usado apenas para aplicativos de **32 bits**. Se você deseja gravar um aplicativo COM de 64 bits, será possível usar a interface do .NET.

Para executar o MQAX em um ambiente do servidor IBM MQ, deve-se ter o Windows 2000 ou mais recente instalado em seu sistema.

Para executar o MQAX em um ambiente do IBM MQ MQI client você precisa do IBM MQ MQI client no Windows 2000 ou mais recente instalado em seu sistema:

O IBM MQ MQI client requer acesso a pelo menos um servidor IBM MQ. Quando o servidor IBM MQ MQI client e IBM MQ estiverem instalados em seu sistema, os aplicativos MQAX sempre executarão no servidor. A interface ActiveX para a MQAI está disponível apenas nos ambientes do servidor IBM MQ.

Design e programação usando o IBM MQ Automation Classes for ActiveX

Projetando aplicativos MQAX que acessam aplicativos não ActiveX

O IBM MQ Automation Classes fornece acesso às funções da API do IBM MQ. É possível, portanto, se beneficiar de todas as vantagens que o uso do IBM MQ pode trazer ao aplicativo Windows.

O design geral de seu aplicativo é o mesmo que para qualquer aplicativo IBM MQ, portanto considere todos os aspectos de design descritos na seção "Considerações de design para aplicativos IBM MQ" na página 47.

Para usar o IBM MQ Automation Classes, codifique os programas Windows em seu aplicativo usando uma linguagem que suporte a criação e o uso de objetos COM. Por exemplo, o Visual Basic, Java e outros clientes de script ActiveX. As classes podem, então, ser facilmente integradas em seu aplicativo porque os objetos do IBM MQ necessários podem ser codificados usando a sintaxe nativa da linguagem de implementação.

Usando o IBM MQ Automation Classes for ActiveX

Ao projetar um aplicativo ActiveX que usa o IBM MQ Automation Classes for ActiveX, o item mais importante da informação será a mensagem enviada ou recebida do sistema remoto do IBM MQ. Portanto, deve-se conhecer o formato dos itens inseridos na mensagem. Para obter um script MQAX para um trabalho, ele e o aplicativo IBM MQ que capta ou envia a mensagem devem conhecer a estrutura da mensagem.

Se você estiver enviando uma mensagem com um aplicativo MQAX e deseja executar a conversão de dados no final MQAX, deve-se também saber:

- A página de códigos usada pelo sistema remoto
- A codificação usada pelo sistema remoto

Para manter seu código portátil, é uma boa prática configurar a página de códigos e codificação, mesmo se eles estiverem atualmente o mesmo nos sistemas emissor e receptor.

Ao considerar, como estruturar a implementação do design de sistema, lembre-se de que seus scripts MQAX executados na mesma máquina que aquele na qual você tem o gerenciador de filas do IBM MQ ou o cliente IBM MQ instalado.

Dicas e sugestões de programação

As dicas e sugestões a seguir não estão em nenhuma ordem significativa. Elas são assuntos que, se relevantes para o trabalho que você está realizando, podem economizar tempo.

Propriedades do Descritor de Mensagens

Se você manipular propriedades do descritor de mensagens em um programa, pode ser melhor usar equivalentes hexadecimais dos campos.

As informações nesta seção se referem às propriedades a seguir:

- AccountingToken
- CorrelationId
- GroupId
- MessageId

Quando um aplicativo IBM MQ é o originador de uma mensagem e IBM MQ gera essas propriedades, é melhor usar as propriedades AccountingTokenHex, CorrelationIdHex, GroupIdHex e MessageIdHex se você deseja consultar seus valores ou manipulá-los de qualquer maneira, incluindo transmiti-los de volta em uma mensagem para IBM MQ. A razão para isso é que os valores gerados do IBM MQ são sequências de bytes que têm qualquer valor entre 0 a 255 inclusive, eles não são sequências de caracteres imprimíveis.

Onde seu script MQAX é o originador de uma mensagem, é possível usar as propriedades AccountingToken, CorrelationId, GroupId e MessageId ou seus equivalentes hexadecimais.

Constantes doIBM MQ

As constantes do IBM MQ são fornecidas como membros do enum do IBM MQ na biblioteca MQAX200.

Constantes de sequência do IBM MQ

As constantes de sequência do IBM MQ não estão disponíveis ao usar o IBM MQ Automation Classes for ActiveX. Deve-se usar a sequência de caracteres explícita para aqueles mostrados na lista a seguir e quaisquer outros que possam ser necessários. Os comandos devem ser preenchidos com oito caracteres usando espaços:

Tabela 79. Constantes de sequência do IBM MQ e suas sequências de caracteres correspondentes.

Constante de cadeia	**Sequência de caracteres correspondente**
MQFMT_NONE	" "
MQFMT_ADMIN	"MQADMIN "
MQFMT_CHANNEL_COMPLETED	"MQCHCOM "
MQFMT_CICS	"MQCICS "
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD "
MQFMT_DIST_HEADER	"MQHDIST "
MQFMT_EVENT	"MQEVENT "
MQFMT_IMS	"MQIMS "
MQFMT_IMS_VAR_STRING	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "
MQFMT_REF_MSG_HEADER	"MQHREF "

Tabela 79. Constantes de sequência do IBM MQ e suas sequências de caracteres correspondentes. (continuação)	
Constante de cadeia	**Sequência de caracteres correspondente**
MQFMT_RF_HEADER	"MQHRF "
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	"MQXMIT "

Constantes de sequência nula

As constantes do IBM MQ, usadas para a inicialização de quatro propriedades MQMessage, MQMI_NONE (24 caracteres NULL), MQCI_NONE (24 caracteres NULL), MQGI_NONE (24 caracteres NULL) e MQACT_NONE (32 caracteres NULL), não são suportadas pelo IBM MQ Automation Classes for ActiveX. Configurá-las para sequências vazias tem o mesmo efeito.

Por exemplo, para configurar os vários IDs de um MQMessage para estes valores: *mymessage*. **MessageId** = "" *mymessage*. **CorrelationId** = "" *mymessage*. **AccountingToken** = ""

Recebendo uma mensagem do IBM MQ

Há várias maneiras de receber uma mensagem do IBM MQ:

- Pesquisando emitindo um GET seguido por uma Espera, usando a função do Visual Basic TIMER.
- Emitindo um GET com a opção de Espera; você especifica a duração de espera, configurando a propriedade WaitInterval. Considere isto quando, mesmo se você configurar seu sistema para executar no ambiente multiencadeado, o software em execução no momento pode executar apenas um único encadeamento. Isso evita que o sistema trave indefinidamente.

  Outros encadeamentos operam não afetados. No entanto, se seus outros encadeamentos requererem acesso a IBM MQ, eles requererão uma segunda conexão com o IBM MQ usando o gerenciador de filas MQAX adicionais e objetos de fila.

Emitir um GET com a opção de Espera e configurar o WaitInterval para MQWI_UNLIMITED faz com que o sistema trave até a conclusão da chamada GET, se o processo for um único encadeamento.

Usando a conversão de dados

Duas formas de conversão de dados são suportadas pelo IBM MQ Automation Classes for ActiveX - codificação numérica e conversão do conjunto de caracteres.

Codificação numérica

Se você configurar a propriedade MQMessage Encoding, os seguintes métodos serão convertidos entre diferentes sistemas de codificação numérica:

- Método ReadDecimal2
- Método ReadDecimal4
- Método ReadDouble
- Método ReadDouble4
- Método ReadFloat
- Método ReadInt2
- Método ReadInt4
- Método ReadLong

- Método ReadShort
- Método ReadUInt2
- Método WriteDecimal2
- Método WriteDecimal4
- Método WriteDouble
- Método WriteDouble4
- Método WriteFloat
- Método WriteInt2
- Método WriteInt4
- Método WriteLong
- Método WriteShort
- Método WriteUInt2

A propriedade Codificação pode ser configurada e interpretada usando as constantes fornecida pelo IBM MQ. Figura 58 na página 587 mostra um exemplo destas:

```
/* Encodings for Binary Integers */
MQENC_INTEGER_UNDEFINED
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
MQENC_DECIMAL_UNDEFINED
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
MQENC_FLOAT_UNDEFINED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_S390
```

Figura 58. Constantes fornecidas para codificação de IBM MQ

Por exemplo, para enviar um número inteiro a partir de um sistema Intel para um sistema operacional System/390 em codificação System/390:

```
Dim msg As New MQMessage  'Define an IBM MQ message for our use..
Print msg. Encoding   'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg. Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg. Encoding   'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg. WriteLong (local_num)       'Write the number into the message
```

Conversão do conjunto de caracteres

A conversão do conjunto de caracteres será necessária ao enviar uma mensagem de um sistema para outro em que as páginas de códigos são diferentes. A conversão da página de códigos é usada por:

- Método ReadString
- Método ReadNullTerminatedString
- Método WriteString
- Método WriteNullTerminatedString
- Propriedade MessageData

Deve-se configurar a propriedade MQMessage CharacterSet para um valor de conjunto de caracteres suportados (CCSID).

IBM MQ Automation Classes for ActiveX usa tabelas de conversão para executar a conversão do conjunto de caracteres.

Por exemplo, para converter sequências automaticamente para a página de códigos 437:

```
Dim msg As New MQMessage                'Define an IBM MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

O método WriteString recebe os dados da sequência (`A character string` no exemplo) como uma sequência Unicode. Ele então converte esses dados de Unicode para a página de códigos 437 usando a tabela de conversão 34B001B5.TBL.

Os caracteres na sequência Unicode que não são suportados pela página de códigos 437 serão fornecidos o caractere de substituição padrão da página de códigos 437.

Da mesma forma, ao usar o método ReadString, a mensagem recebida terá um conjunto de caracteres estabelecido pelo valor do IBM MQ Message Descriptor (MQMD) e haverá uma conversão a partir dessa página de códigos em Unicode antes de ser transmitido de volta para sua linguagem de script.

Encadeamentos

IBM MQ Automation Classes for ActiveX implementam um modelo de encadeamento livre em que os objetos podem ser usados entre os encadeamentos.

Enquanto MQAX permite o uso de objetos MQQueue e MQQueueManager, o IBM MQ não permite atualmente o compartilhamento de identificadores entre diferentes encadeamentos.

Tenta usá-los em outro resultado de encadeamento em um erro e IBM MQ retorna um código de retorno MQRC_HCONN_ERROR.

Nota: Há apenas um objeto MQSession por processo. O uso do MQSession CompletionCode e ReasonCode não é recomendado em ambientes multiencadeados. Os valores de erro MQSession podem ser sobrescritos por um segundo encadeamento entre um erro sendo levantado e verificado no primeiro encadeamento. Os encadeamentos são serializados para a duração de cada chamada de método ou acesso de propriedade. Portanto, emitindo um Get com a opção Espera faz com que outros encadeamentos acessem objetos MQAX para serem suspensos até que a operação seja concluída.

Manipulação de Erros

Estas informações descrevem propriedades do objeto MQAX, como funciona a manipulação de erros, regras que descrevem como exceções levantadas são manipuladas e obtenção de uma propriedade.

Cada objeto MQAX inclui propriedades para manter informações de erro e um método para reconfigurar ou limpar as mesmas. As propriedades são:

- CompletionCode
- ReasonCode
- ReasonName

O método é:

- ClearErrorCodes

Como a manipulação de erros funciona

O script ou aplicativo MQAX chama o método de um objeto MQAX ou acessa ou atualiza uma propriedade do objeto MQAX:

1. O ReasonCode e o CompletionCode no objeto em questão são atualizados.

2. O ReasonCode e CompletionCode no objeto MQSession também são atualizados com as mesmas informações.

   Nota: Consulte "Encadeamentos" na página 588 para obter as restrições sobre o uso de códigos de erro MQSession em aplicativos encadeados.

   Se o CompletionCode for maior ou igual à propriedade ExceptionThreshold de MQSession, MQAX lançará uma exceção (número 32000). Use isso dentro de seu script usando a instrução On Error (ou equivalente) para processá-la.

3. Use a função Error para recuperar a sequência de erros associada, que tem o formato:

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   Para obter mais informações sobre como usar as instruções On Error, consulte a documentação para a linguagem de script ActiveX.

   Usar CompletionCode e ReasonCode no objeto MQSession é conveniente para manipuladores de erros simples.

   A propriedade ReasonName retorna o nome simbólico do IBM MQ para o valor atual do ReasonCode.

Levantando exceções

As regras a seguir descrevem como exceções levantadas são manipuladas:

- Sempre que uma propriedade ou um método configurar o código de conclusão para um valor maior ou igual ao limite de exceção (geralmente configurado como 2), uma exceção será levantada.
- Todas as chamadas de método e conjuntos de propriedades configuram o código de conclusão.

Obtendo uma propriedade

Este é um caso especial, pois CompletionCode e ReasonCode não são sempre atualizados:

- Se uma propriedade get for bem-sucedida, o objeto e o ReasonCode e o CompletionCode do objeto MQSession permanecem inalterados.
- Se uma propriedade get falhar com um CompletionCode de aviso, o ReasonCode e o CompletionCode permanecerão inalterados.
- Se uma propriedade get falhar com um CompletionCode de erro, o ReasonCode e o CompletionCode serão atualizados para refletirem os valores verdadeiros e o processamento de erro continua conforme descrito.

A classe MQSession tem um método *ReasonCodeName* que pode ser usado para substituir um código de razão do IBM MQ por um nome simbólico. Isso é especialmente útil durante o desenvolvimento de programas em que podem ocorrer erros inesperados. No entanto, o nome não é ideal para apresentação aos usuários.

Cada classe também tem uma propriedade *ReasonName*, que retorna o nome simbólico do código de razão atual para essa classe.

Referência do IBM MQ Automation Classes for ActiveX

Esta seção descreve as classes do IBM MQ Automation Classes for ActiveX (MQAX), desenvolvido para ActiveX. As classes permitem gravar aplicativos ActiveX que podem acessar outros aplicativos em execução em seus ambientes não ActiveX, usando o IBM MQ.

Interface do IBM MQ Automation Classes for ActiveX

IBM MQ Automation Classes for ActiveX fornece constantes ActiveX numéricas predefinidas (como MQMT_REQUEST) necessárias para usar as classes.

As classes de automação ActiveX consistem no seguinte:

- “Classe MQSession” na página 591
- “Classe MQQueueManager” na página 594
- “Classe MQQueue” na página 606
- “Classe MQMessage” na página 621
- “Classe MQPutMessageOptions” na página 644
- “Classe MQGetMessageOptions” na página 646
- “Classe MQDistributionList” na página 648
- “Classe MQDistributionListItem” na página 653

Além disso, IBM MQ Automation Classes for ActiveX fornece constantes ActiveX numéricas predefinidas (como MQMT_REQUEST) necessárias para usar as classes. Essas são fornecidas na enumeração MQ na biblioteca MQAX200. As constantes é um subconjunto daqueles definidos nos arquivos de cabeçalho do IBM MQ C (cmqc*.h) com alguns códigos de razão adicionais do IBM MQ Automation Classes for ActiveX.

Sobre as classes do IBM MQ Automation Classes for ActiveX

Leia estas informações juntamente com os tópicos de referência em Desenvolvendo referência de aplicativos.

See Features that can be used only with the primary installation on Windows for important information.

A classe MQSession fornece um objeto raiz que contém o status da última ação executada em qualquer um dos objetos MQAX. Consulte o “Manipulação de Erros” na página 588 para obter informações adicionais.

As classes MQQueueManager e MQQueue fornecem acesso a objetos subjacentes do IBM MQ. Acessos a propriedade ou métodos para essas classes no resultado geral em chamadas que estão sendo feitas entre o IBM MQ MQI.

As classes MQMessage, MQPutMessageOptions e MQGetMessageOptions contêm as estruturas de dados do MQMD, MQPMO e MQGMO e são usadas para ajudá-lo a enviar mensagens para filas e recuperar mensagens delas.

A classe MQDistributionList contém uma coleta de filas - local, remoto ou alias para saída. A classe MQDistributionListItem contém as estruturas MQOR, MQRR e MQPMR e as associa a uma lista de distribuição de propriedade.

Transmissão de parâmetros

Os parâmetros em chamadas de métodos são todos transmitidos por valor, exceto se esse parâmetro for um objeto, em cujo caso é uma referência transmitida.

As definições de classe fornecidas listam o Tipo de dados para cada parâmetro ou propriedade. Para muitos clientes ActiveX, como Visual Basic, se a variável usada não for do tipo requerido, o valor será automaticamente convertido para ou a partir do tipo requerido - o fornecimento de tal conversão é possível. Isso segue regras padrão do cliente; MQAX não fornece essa conversão.

Muitos dos métodos têm parâmetros de sequência de comprimento fixo ou retorna uma sequência de caracteres de comprimento fixo. As regras de conversão são as seguintes:

- Se o usuário fornecer uma sequência de comprimento fixo de comprimento incorreto, como um parâmetro de entrada ou como um valor de retorno, o valor será truncado ou preenchido com espaços à direita, conforme necessário.
- Se o usuário fornecer uma sequência de comprimento variável de comprimento incorreto como um parâmetro de entrada, o valor será truncado ou preenchido com espaços à direita.

- Se o usuário fornecer uma sequência de comprimento variável do comprimento incorreto como um valor de retorno, a sequência será ajustada para o comprimento necessário (porque retornar um valor inutilizará o valor anterior na sequência de qualquer forma).
- As sequências fornecidas como parâmetros de entrada podem conter Nulos integrados.

Estas classes podem ser localizadas na biblioteca MQAX200.

Métodos de acesso ao objeto

Esses métodos não estão relacionados diretamente com qualquer chamada única do IBM MQ. Cada um desses métodos cria um objeto no qual as informações de referência são mantidas, seguido pela conexão ou abertura de um objeto IBM MQ:

Quando uma conexão for estabelecida em um gerenciador de filas, ela irá manter o atributo 'connection handle' gerado pelo IBM MQ.

Quando uma fila for aberta, ela irá manter o atributo 'object handle' gerado pelo IBM MQ.

Esses atributos do IBM MQ não estão diretamente disponíveis para o programa MQAX.

Erros

Os erros sintáticos na transmissão de parâmetro podem ser detectados no tempo de compilação e no tempo de execução pelo cliente ActiveX. Os erros podem ser capturados usando On Error no Visual Basic.

Classes do IBM MQ ActiveX todas contêm duas propriedades especiais somente leitura - ReasonCode e CompletionCode. Essas propriedades podem ser lidas a qualquer momento.

Uma tentativa de acessar qualquer outra propriedade ou emitir qualquer chamada de método pode gerar um erro do IBM MQ.

Se um conjunto de propriedades ou chamada de método for bem-sucedido, o ReasonCode do objeto de propriedade será configurado para MQRC_NONE e CompletionCode para MQCC_OK.

Se o acesso de propriedade ou chamada de método não for bem-sucedido, os códigos de razão e de conclusão serão configurados nesses campos.

Classe MQSession

Esta é a classe principal para IBM MQ Automation Classes for ActiveX.

Há sempre apenas um objeto MQSession por processo do cliente ActiveX. Uma tentativa de criar um segundo objeto cria uma segunda referência ao objeto original.

Criação

Novo cria um novo objeto MQSession.

Sintaxe

Dim *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

Propriedades

- “propriedade CompletionCode” na página 592.
- “Propriedade ExceptionThreshold” na página 592.
- “propriedade ReasonCode” na página 592.
- “Propriedade ReasonName” na página 593.

Método

- “Método AccessGetMessageOptions” na página 593.
- “Método AccessMessage” na página 593.
- “Método AccessPutMessageOptions” na página 593.
- “Método AccessQueueManager” na página 593.
- “método ClearErrorCodes” na página 594.
- “Método ReasonCodeName” na página 594.

propriedade CompletionCode

Somente leitura. Retorna o código de conclusão IBM MQ configurado pelo método ou conjunto de propriedades mais recente emitido com relação a qualquer objeto do IBM MQ.

Ele é reconfigurado para MQCC_OK quando um método ou um conjunto de propriedades é chamado com êxito em relação a qualquer objeto MQAX.

Um manipulador de eventos de erro pode inspecionar essa propriedade para diagnosticar o erro, sem ter que saber qual objeto foi envolvido.

Usando o CompletionCode e ReasonCode no objeto MQSession é muito conveniente para manipuladores de erro simples.

Nota: Consulte “Encadeamentos” na página 588 para obter as restrições sobre o uso de códigos de erro MQSession em aplicativos encadeados.

Definido em:
Classe MQSession

Tipo de dados:
Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe:
Para obter: *completioncode & = MQSession*.CompletionCode

Propriedade ExceptionThreshold

Leitura/gravação. Define o nível de erro do IBM MQ para o qual MQAX emitirá uma exceção. Usa como padrão MQCC_FAILED. Um valor maior do que MQCC_FAILED efetivamente impede o processamento de exceções, deixando o programador executar verificações no CompletionCode e ReasonCode.

Definido em: classe MQSession

Tipo de dados: longo

Valores:

- Qualquer, mas considere MQCC_WARNING, MQCC_FAILED ou superior.

Sintaxe:

Para obter: *ExceptionThreshold& = MQSession*. **ExceptionThreshold**

Para configurar: *MQSession.* **ExceptionThreshold** = *ExceptionThreshold$*

propriedade ReasonCode

Somente leitura. Retorna o código de razão definido pelo método ou conjunto de propriedades mais recente emitido em relação a qualquer objeto do IBM MQ.

Um manipulador de eventos de erro pode inspecionar essa propriedade para diagnosticar o erro, sem ter que saber qual objeto foi envolvido.

Usando o CompletionCode e ReasonCode no objeto MQSession é muito conveniente para manipuladores de erro simples.

Nota: Consulte “Encadeamentos” na página 588 para obter as restrições sobre o uso de códigos de erro MQSession em aplicativos encadeados.

Definido em: classe MQSession

Tipo de dados: longo

Valores:

- Consulte Reason (MQLONG) e os valores MQAX adicionais listados em “Códigos de razão para o IBM MQ Automation Classes for ActiveX” na página 660.

Sintaxe: para obter: *reasoncode & = MQSession*. **ReasonCode**

Propriedade ReasonName

Somente leitura. Retorna o nome simbólico do código de razão mais recente. Por exemplo, "MQRC_QMGR_NOT_AVAILABLE".

Nota: Consulte “Encadeamentos” na página 588 para obter as restrições sobre o uso de códigos de erro MQSession em aplicativos encadeados.

Definido em: classe MQSession

Tipo de dados: sequência

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasonname$ = MQSession* **.ReasonName**

Método AccessGetMessageOptions

Cria um novo objeto MQGetMessageOptions.

Definido em:
Classe MQSession

Sintaxe:
gmo = MQSession **.AccessGetMessageOptions()**

Método AccessMessage

Cria um novo objeto MQMessage.

Definido em:
Classe MQSession

Sintaxe:
msg = MQSession **.AccessMessage()**

Método AccessPutMessageOptions

Cria um novo objeto MQPutMessageOptions.

Definido em:
Classe MQSession

Sintaxe:
pmo = MQSession **.AccessPutMessageOptions()**

Método AccessQueueManager

Cria um novo objeto MQQueueManager e conecta-o a um gerenciador de filas real por meio do servidor IBM MQ MQI client ou IBM MQ. Assim como executar uma conexão, esse método também executa uma abertura para o objeto do gerenciador de filas.

Quando ambos os servidores IBM MQ MQI client e IBM MQ estão instalados em seu sistema, os aplicativos MQAX serão executados com relação ao servidor por padrão. Para executar MQAX com relação ao cliente, a biblioteca de ligações do cliente deve ser especificada na variável de ambiente `GMQ_MQ_LIB`, por exemplo, configure `GMQ_MQ_LIB=mqic.dll`.

Para uma instalação somente do cliente, não é necessário configurar a variável de ambiente `GMQ_MQ_LIB`. Quando essa variável não estiver configurada, o IBM MQ tenta carregar amqzst.dll. Se esse DLL não estiver presente (como é o caso em uma instalação somente cliente), o IBM MQ tenta carregar mqic.dll.

Se bem-sucedido, configura o ConnectionStatus do MQQueueManager para TRUE.

Um gerenciador de filas pode ser conectado a no máximo um objeto MQQueueManager por instância ActiveX.

Se a conexão com o gerenciador de filas falhar, um evento de erro será levantado e o ReasonCode e o CompletionCode do objeto MQSession serão configurados.

Definido em: classe MQSession

Sintaxe: *set qm = MQSession* **.AccessQueueManager** (*Name$*)

Parâmetro: *Name$* String. Nome do Gerenciador de filas ao qual ser conectado.

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode.

Definido em: classe MQSession

Sintaxe:

```
Call MQSession.ClearErrorCodes()
```

Método ReasonCodeName

Retorna o nome do código de razão com o valor numérico determinado. É útil fornecer indicações mais claras de condições de erro aos usuários. O nome ainda está um tanto quanto criptografado (por exemplo, ReasonCodeName(2059) é **MQRC_Q_MGR_NOT_AVAILABLE**), portanto, quando possível, erros devem ser capturados e substituídos com texto descritivo apropriado para o aplicativo.

Definido em: classe MQSession

Sintaxe: *errname $= MQSession* **.ReasonCodeNome** (*reasonCode&*)

Parâmetro: *reasoncode &* longo. O código de razão para o qual o nome simbólico é necessário.

Classe MQQueueManager

Esta classe representa uma conexão com um gerenciador de filas. O gerenciador de filas pode ser executado localmente (um servidor IBM MQ) ou remotamente com acesso fornecido pelo cliente do IBM MQ. Um aplicativo deve criar um objeto desta classe e conectá-lo a um gerenciador de filas. Quando um objeto dessa classe é destruído, ele é automaticamente desconectado de seu gerenciador de filas.

Restrição

Os objetos da classe MQQueue são associadas a essa classe.

Novo cria um novo objeto MQQueueManager e configura todas as propriedades como os valores iniciais. Como alternativa, use o método AccessQueueManager da classe MQSession.

Criação

Novo cria um **novo** objeto MQQueueManager e configura todas as propriedades como os valores iniciais. Como alternativa, use o método AccessQueueManager da classe MQSession.

Sintaxe

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

Propriedades

- “Propriedade TriggerInterval” na página 603.

Methods

- “Método AccessQueue” na página 603.
- “Método AddDistributionList” na página 604.
- “Método Backout” na página 604.
- “Método Begin” na página 605.
- “método ClearErrorCodes” na página 605.
- “Método Commit” na página 605.
- “Método Connect” na página 605.
- “Método Disconnect” na página 605.

Acesso à propriedade

As propriedades a seguir podem ser acessadas a qualquer momento.

- “Propriedade AlternateUserId” na página 596.
- “propriedade CompletionCode” na página 598.
- “Propriedade ConnectionStatus” na página 599.
- “propriedade ReasonCode” na página 602.

As propriedades restantes poderão ser acessadas apenas se o objeto estiver conectado a um gerenciador de filas e o ID do usuário estiver autorizado a consultar em relação a esse gerenciador de filas. Se um ID de usuário alternativo for configurado e o ID do usuário atual estiver autorizado a usá-lo, o ID de usuário alternativo será marcado para autorização para consulta no lugar

Se essas condições não se aplicarem, IBM MQ Automation Classes for ActiveX tentará se conectar ao gerenciador de filas e abri-lo para consulta automaticamente. Se isto for malsucedido, a chamada configurará um CompletionCode de MQCC_FAILED e um dos seguintes ReasonCodes:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

Propriedade AlternateUserId

Leitura/gravação. O ID do usuário alternativo a ser utilizado para validar o acesso aos atributos do gerenciador de filas.

Esta propriedade não deve ser configurada se IsConnected for TRUE.

Esta propriedade não pode ser configurada enquanto o objeto está aberto.

`Defined in:` Classe MQQueueManager

`Data Type:` Sequência de 12 caracteres

`Syntax:` Para obter: *altuser $= MQQueueManager* **.AlternateUserId** Para configurar: *MQQueueManager* **.AlternateUserId** = *altuser $*

Propriedade AuthorityEvent

Somente leitura. O atributo de MQI AuthorityEvent.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: para obter: *authevent = MQQueueManager* **.AuthorityEvent**

Propriedade BeginOptions

Leitura/gravação. Estas são as opções que se aplicam ao método Begin. Inicialmente MQBO_NONE.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Valores:

- MQBO_NONE

Sintaxe: Para obter: *beginoptions & =MQQueueManager* **BeginOptions**

Para configurar: *MQQueueManager* **.BeginOptions** = *beginoptions &*

Propriedade ChannelAutoDefinition

Somente leitura. Isso controla se a definição de canal automática é permitida.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Valores:

- MQCHAD_DISABLED
- MQCHAD_ENABLED

Sintaxe: Para obter: *channelautodef & = MQQueueManager* **ChannelAutoDefinition**

Propriedade ChannelAutoDefinitionEvent

Somente leitura. Isso controla se os eventos de definição de canal automáticos são gerados.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *channelautoderrovent & =MQQueueManager* **ChannelAutoDefinitionEvent**

Propriedade ChannelAutoDefinitionExit

Somente leitura. O nome do usuário utilizado para saída de definição automática do canal.

Definido em:
Classe MQQueueManager

Tipo de dados:
Sequência

Sintaxe: Para obter: *channelautodefexit$= MQQueueManager*. **ChannelAutoDefinitionExit**

Propriedade CharacterSet

Somente leitura. O atributo CodedCharSetId de MQI.

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Para obter: *characterset & = MQQueueManager* **.CharacterSet**

Propriedade CloseOptions

Leitura/gravação. Opções usadas para controlar o que acontece quando o gerenciador de filas é fechada. O valor inicial é MQCO_NONE.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Valores:

- MQCO_NONE

Sintaxe: Para obter: *closeopt & = MQQueueManager* **.CloseOptions**

Para configurar: *MQQueueManager* **.CloseOptions** = *closeopt &*

Propriedade CommandInputQueueName

Somente leitura. O atributo de MQI CommandInputQName.

Definido em: classe MQQueueManager

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *commandinputqname$ = MQQueueManager* **.CommandInputQueueName**

Propriedade CommandLevel

Somente leitura. Retorna a versão e o nível da implementação do gerenciador de filas do IBM MQ (atributo CommandLevel de MQI)

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Para obter: *level & = MQQueueManager* **.CommandLevel**

propriedade CompletionCode

Somente leitura. Retorna o código de conclusão configurados pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode & = MQQueueManager* **.CompletionCode**

Propriedade ConnectionHandle

Somente leitura. A manipulação de conexões para o objeto do gerenciador de filas IBM MQ.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Sintaxe: Para obter: *hconn & = MQQueueManager.* **ConnectionHandle**

Propriedade ConnectionStatus

Somente leitura. Indica se o objeto está conectado a seu gerenciador de filas ou não.

Definido em: classe MQQueueManager

Tipo de dado: booleano

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxe: para obter: *status = MQQueueManager* **.ConnectionStatus**

Propriedade ConnectOptions

Leitura/Gravação. Essas opções se aplicam ao método Connect. Inicialmente MQCNO_NONE.

Definido em:
Classe MQQueueManager

Tipo de dados:
Long

Valores:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

Sintaxe: Para obter: *conexões & =MQQueueManager*. **ConnectOptions**

Para configurar: *MQQueueManager* **.ConnectOptions** = *connectoptions &*

Propriedade DeadLetterQueueName

Somente leitura. O atributo de MQI DeadLetterQName.

Definido em: classe MQQueueManager

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *dlqname$ = MQQueueManager* **.DeadLetterQueueName**

Propriedade DefaultTransmissionQueueName

Somente leitura. O atributo de MQI DefXmitQName.

Definido em: classe MQQueueManager

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *defxmitqname$ = MQQueueManager* **.DefaultTransmissionQueueName**

Propriedade Description

Somente leitura. O atributo QMgrDesc do MQI.

Definido em: classe MQQueueManager

Tipo de dado: Sequência de 64 caracteres

Sintaxe: para obter: *description$ = MQQueueManager* **.Description**

Propriedade DistributionLists

Somente leitura. Essa é a capacidade do gerenciador de filas para suportar listas de distribuição.

Definido em:
Classe MQQueueManager

Tipo de dados:
Booleana

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxe: Para obter: *distributionlists= MQQueueManager*. **DistributionLists**

Propriedade InhibitEvent

Somente leitura. O atributo de MQI InhibitEvent.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *inibir & = MQQueueManager* **.InhibitEvent** .

Propriedade IsConnected

Um valor que indica se o gerenciador de filas está atualmente conectado.

Somente leitura.

Definido em: classe MQQueueManager

Tipo de dado: booleano

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxe: para obter: *isconnected = MQQueueManager* **.IsConnected**

Propriedade IsOpen

Um valor que indica se o gerenciador de filas está atualmente aberto para consulta.

Somente leitura.

Definido em:
Classe MQQueueManager

Tipo de dados:
Booleana

Valores:

- TRUE (-1)

- FALSE (0)

Sintaxe: Para obter: *IsOpen = MQQueueManager*. **IsOpen**

Propriedade LocalEvent

Somente leitura. O atributo de MQI LocalEvent.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *localevent & = MQQueueManager* **.LocalEvent**

Propriedade MaximumHandles

Somente leitura. O atributo MaxHandles MQI.

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Para obter: *maxidentificadores & = MQQueueManager* **.MaximumHandles**

Propriedade MaximumMessageLength

Somente leitura. O atributo Gerenciador de Filas MaxMsgLength MQI.

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Para obter: *maxmessagelength & = MQQueueManager* **.MaximumMessageLength**

Propriedade MaximumPriority

Somente leitura. O atributo MaxPriority do MQI.

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Para obter: *maxpriority & = MQQueueManager* **.MaximumPriority**

Propriedade MaximumUncommittedMessages

Somente leitura. O atributo MQI MaxUncommittedMsgs.

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Obter: *maxuncommitted & = MQQueueManager* **.MaximumUncommittedMensagens**

Propriedade Name

Leitura/gravação. O atributo QMgrName do MQI. Esta propriedade não pode ser gravada quando o MQQueueManager está conectado.

Definido em: classe MQQueueManager

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *name$ = MQQueueManager* **.name**

Para configurar: *MQQueueManager* **.name** = *name$*

Nota: O Visual Basic reserva a propriedade "Name" para uso na interface visual. Portanto, ao usar no Visual Basic, utilize letras minúsculas, ou seja, "name".

Propriedade ObjectHandle

Somente leitura. A manipulação de objetos para o objeto do gerenciador de filas do IBM MQ.

Definido em:
Classe MQQueueManager

Tipo de Dados
Long

Sintaxe: Para obter: *hobj & = MQQueueManager.* **ObjectHandle**

Propriedade PerformanceEvent

Somente leitura. O atributo de MQI PerformanceEvent.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *perfevent & = MQQueueManager*.PerformanceEvent

Propriedade Platform

Somente leitura. O atributo de Plataforma MQI.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

Sintaxe: Para obter: *platform & = MQQueueManager* **Plataforma**

propriedade ReasonCode

Somente leitura. Retorna o código de razão definido pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter: *reasoncode & = MQQueueManager* **.ReasonCode**

Propriedade ReasonName

Somente leitura. Retorna o nome simbólico do código de razão mais recente. Por exemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definido em: classe MQQueueManager

Tipo de dados: sequência

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasonname$ = MQQueueManager* **.ReasonName**

Propriedade RemoteEvent

Somente leitura. O atributo de MQI RemoteEvent.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *remoteevent & = MQQueueManager* **.RemoteEvent**

Propriedade StartStopEvent

Somente leitura. O atributo de MQI StartStopEvent.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *strstpevent & = MQQueueManager* **.StartStopEvento**

Propriedade SyncPointAvailability

Somente leitura. O atributo SyncPoint MQI.

Definido em: classe MQQueueManager

Tipo de dados: longo

Valores:

- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

Sintaxe: Para obter: *syncpointavailability & = MQQueueManager* **.SyncPointDisponibilidade**

Propriedade TriggerInterval

Somente leitura. O atributo de MQI TriggerInterval.

Definido em: classe MQQueueManager

Tipo de dados: longo

Sintaxe: Para obter: *trigint & = MQQueueManager* **.TriggerInterval**

Método AccessQueue

Cria um objeto MQQueue e associa-o a este objeto MQQueueManager configurando a propriedade de referência de conexão da fila. Ele configura as propriedades Nome, OpenOptions, DynamicQueueName e AlternateUserId do objeto MQQueue para os valores fornecidos e tentativas para abri-lo.

Se a abertura for mal sucedida, a chamada falhará. Um evento de erro é emitido em relação ao objeto. O ReasonCode e CompletionCode e o MQSession ReasonCode e CompletionCode do objeto são configurados.

Os parâmetros DynamicQueueName, QueueManagerName e AlternateUserId são opcionais e padronizados para "".

O OpenOption MQOO_INQUIRE deve ser especificado além das outras opções, se as propriedades da fila deverem ser lidas.

Não configure o QueueManagerName ou configure-o para "" se a fila a ser aberta for local. Caso contrário, configure-o para o nome do gerenciador de filas remotas que possui a fila e uma tentativa é feita para abrir uma definição local da fila remota. Para obter mais informações sobre a resolução de nome de fila remota e alias do gerenciador de filas, consulte O que são aliases?.

Se a propriedade Nome estiver configurada para um nome de fila modelo, especifique o nome da fila dinâmica a ser criada no parâmetro DynamicQueueName$. Se o valor fornecido no parâmetro DynamicQueueName$ for "", o valor configurado no objeto de fila e usado na chamada aberta será "AMQ.* ". Consulte “Criando filas dinâmicas” na página 748 para obter mais informações sobre a nomenclatura de filas dinâmicas.

Definição

Definido em: classe MQQueueManager.

Sintaxe

Sintaxe: set queue = MQQueueManager. **AccessQueue** (Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Parâmetros

Name$ String. Nome da fila do IBM MQ.

OpenOptions: Long. Opções a serem usadas quando a fila for aberta. Consulte OpenOptions (MQLONG).

QueueManagerName$ String. Nome do gerenciador de filas que possui a fila a ser aberta. Um valor "" indica que o gerenciador de filas é local.

*DynamicQueueName$* String. O nome designado à fila dinâmica no momento em que a fila é aberta quando o parâmetro Name$ especifica uma fila modelo.

AlternateUserId$ String. O ID do usuário alternativo usado para validar o acesso ao abrir a fila.

Método AddDistributionList

Cria um novo objeto MQDistributionList e configura sua referência de conexão para o gerenciador de filas proprietário.

Definido em:
Classe MQQueueManager

Sintaxe: *set distributionlist* = **MQQueueManager.** *AddDistributionList*

Método Backout

Faz saída quaisquer entradas de mensagens não consolidadas e obtenções que ocorreram como parte de uma unidade de trabalho desde a última sincronização.

Definido em: classe MQQueueManager

Sintaxe:

```
Call MQQueueManager.Backout()
```

Método Begin

Inicia uma unidade de trabalho que é coordenada pelo gerenciador de filas. O iniciar opções afetam o comportamento deste método.

Definido em:
Classe MQQueueManager

Sintaxe:

```
Call MQQueueManager.Begin()
```

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode para a classe MQQueueManager e a classe MQSession.

Definido em:
Classe MQQueueManager

Sintaxe:
Call *MQQueueManager* **.ClearErrorCodes()**

Método Commit

Confirma quaisquer inserções de mensagem e obtenções que ocorreram como parte de uma unidade de trabalho desde o último ponto de sincronização.

Definido em: classe MQQueueManager

Sintaxe:

```
Call MQQueueManager.Commit()
```

Método Connect

Conecta o objeto MQQueueManager a um gerenciador de filas real por meio do IBM MQ MQI client ou servidor. Além de estabelecer a conexão, este método também abre o objeto de gerenciador de filas para que ele possa ser consultado.

Configura IsConnected para TRUE.

Um máximo de um objeto MQQueueManager por instância do ActiveX é permitido para se conectar a um gerenciador de filas.

Definido em: classe MQQueueManager

Sintaxe:

```
Call MQQueueManager.Connect()
```

Método Disconnect

Desconecta o objeto MQQueueManager do gerenciador de filas.

Configura IsConnected para FALSE.

Todos os objetos de Filas associados ao objeto MQQueueManager tornam-se inutilizáveis e não podem ser reabertos.

Qualquer mudança não confirmada (envios e recebimentos de mensagens) é confirmada.

Definido em: classe MQQueueManager

Sintaxe:

```
Call MQQueueManager.Disconnect()
```

Classe MQQueue

Essa classe representa acesso a uma fila do IBM MQ. Essa conexão é fornecida por um objeto MQQueueManager associado. Quando um objeto desta classe é destruído, é fechado automaticamente.

Restrição

A classe MQQueue é contida pela classe MQQueueManager.

Criação

New cria um novo objeto MQQueue e define todas as propriedades para os valores iniciais. Como alternativa, use o método AccessQueue da classe MQQueueManager.

Sintaxe

```
Dim que As New MQQueue Set que = New MQQueue
```

Propriedades

Methods

- “Método Get” na página 619
- “Método Open” na página 620
- “Método Put” na página 621

Acesso à propriedade

Se o objeto da fila não estiver conectado a um gerenciador de filas, será possível ler as seguintes propriedades:

- “propriedade CompletionCode” na página 609
- “Propriedade OpenStatus” na página 615
- “propriedade ReasonCode” na página 616

e será possível ler e gravar em:

- “Propriedade AlternateUserId” na página 608
- “Propriedade CloseOptions” na página 609
- “Propriedade ConnectionReference” na página 610
- “Propriedade Name” na página 614
- “Propriedade OpenOptions” na página 614

Se o objeto de fila estiver conectado a um gerenciador de filas, será possível ler todas as propriedades.

Propriedades do atributo da fila

Propriedades não listadas na seção anterior são todas atributos da fila subjacente do IBM MQ. Eles podem ser acessados apenas se o objeto for conectado a um gerenciador de filas e o ID de Usuário do usuário estiver autorizado a consultar ou configurar com relação a essa fila. Se um ID de usuário alternativo for definido e o ID do usuário atual estiver autorizado a usá-lo, o ID de usuário alternativo será verificado para autorização ao invés.

A propriedade deve ser adequada para o QueueType fornecido. Consulte Atributos para filas para obter mais informações.

Se essas condições não se aplicarem, o acesso de propriedade irá configurar um CompletionCode de MQCC_FAILED e um dos ReasonCodes a seguir:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode é MQCC_WARNING)

Abrindo uma fila

A única maneira de criar um objeto MQQueue é usando o método AccessQueue de MQQueueManager ou em Novo. Um objeto MQQueue aberto permanece aberto (OpenStatus=TRUE) até que ele seja fechado ou excluído ou até que o objeto do gerenciador de filas criador seja excluído ou a conexão é perdida ao gerenciador de filas. O valor das propriedades CloseOptions de MQQueue controlam o comportamento da operação de encerramento que ocorre quando o objeto MQQueue é excluído.

O método MQQueueManager AccessQueue abre a fila usando o parâmetro OpenOptions. O método MQQueue.Open abre a fila usando a propriedade OpenOptions. IBM MQ valida o OpenOptions em relação à autorização do usuário como parte do processo de fila aberto.

Propriedade AlternateUserId

Leitura/gravação. O ID do usuário alternativo utilizado para validar o acesso à fila quando ela é aberta.

Esta propriedade não pode ser configurado enquanto o objeto está aberto (ou seja, quando IsOpen for TRUE).

Definido em: classe MQQueue

Tipo de dados: sequência de 12 caracteres

Sintaxe: para obter: *altuser$ = MQQueue* **.AlternateUserId**

Para configurar: *MQQueue.* **AlternateUserId** = *altuser$*

Propriedade BackoutRequeueName

Somente leitura. O atributo de MQI BackOutRequeueQName.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *backoutrequeuename$ = MQQueue* **.BackoutRequeueName**

Propriedade BackoutThreshold

Somente leitura. O atributo BackoutThreshold MQI.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- Veja BackoutThreshold (MQLONG).

Sintaxe: Para obter: *backoutthreshold & = MQQueue.* **BackoutThreshold**

Propriedade BaseQueueName

Somente leitura. O nome da fila ao qual o alias é resolvido.

Válido apenas para filas de alias.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *baseqname$ = MQQueue* **.BaseQueueName**

Propriedade CloseOptions

Leitura/Gravação. As opções utilizadas para controlar o que acontece quando a fila for fechada.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE e MQCO_DELETE_PURGE são válidos apenas para filas dinâmicas.

Sintaxe: para obter: *closeopt & = MQQueue* **.CloseOptions**

Para configurar: *MQQueue* **.CloseOptions** = *closeopt &*

propriedade CompletionCode

Somente leitura. Retorna o código de conclusão configurados pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode & = MQQueue* **.CompletionCode**

Propriedade ConnectionReference

Leitura/gravação. Define o objeto do gerenciador de filas ao qual um objeto da fila pertence. A referência de conexão não pode ser gravada enquanto uma fila está aberta.

Definido em: classe MQQueue

Tipo de dado: MQQueueManager

Valores:

- Uma referência a um objeto do Gerenciador de Filas ativo do IBM MQ

Sintaxe: para configurar: *set MQQueue* **.ConnectionReference** = *ConnectionReference*

Para obter: *set ConnectionReference = MQQueue* **.ConnectionReference**

Propriedade CreationDateTime

Somente leitura. Data e hora em que essa fila foi criada.

Definido em: classe MQQueue

Tipo de dado: Variante de tipo 7 (data/hora) ou EMPTY

Sintaxe: para obter: *datetime = MQQueue* **.CreationDateTime**

Propriedade CurrentDepth

Somente leitura. O número de mensagens que estão atualmente na fila.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *currentdepth & = MQQueue* **.CurrentDepth**

Propriedade DefaultInputOpenOption

Somente leitura. Controla o modo que a fila é aberta se o OpenOptions especificar MQOO_INPUT_AS_Q_DEF.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

Sintaxe: Para obter: *defaultinop & = MQQueue* **.DefaultInputOpenOption**

Propriedade DefaultPersistence

Somente leitura. A persistência padrão para mensagens em uma fila.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *defpersistence & = MQQueue* **.DefaultPersistence**

Propriedade DefaultPriority

Somente leitura. A prioridade padrão para mensagens em uma fila.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *defpriority & = MQQueue* **.DefaultPriority**

Propriedade DefinitionType

Somente leitura. O tipo de definição de fila.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQDT_PREDEFINED
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

Sintaxe: Para obter: *deftype & = MQQueue* **.DefinitionType**

Propriedade DepthHighEvent

Somente leitura. O atributo de MQI QDepthHighEvent.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *depthhighevent & = MQQueue.* **DepthHighEvent**

Propriedade DepthHighLimit

Somente leitura. O atributo de MQI QDepthHighLimit.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *depthhighlimit & = MQQueue.* **DepthHighLimit**

Propriedade DepthLowEvent

Somente leitura. O atributo de MQI QDepthLowEvent.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *depthlowevent & = MQQueue.* **DepthLowEvent**

Propriedade DepthLowLimit

Somente leitura. O atributo MQI QDepthLowLimit.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *depthlowlimit & = MQQueue.* **DepthLowLimit**

Propriedade DepthMaximumEvent

Somente leitura. O atributo de MQI QDepthMaxEvent.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQEVR_DISABLED
- MQEVR_ENABLED

Sintaxe: Para obter: *depthmaximumevent & = MQQueue.* **DepthMaximumEvent**

Propriedade Description

Somente leitura. Uma descrição da fila.

Definido em: classe MQQueue

Tipo de dado: Sequência de 64 caracteres

Sintaxe: para obter: *description$ = MQQueue* **.Description**

Propriedade DynamicQueueName

Leitura/Gravação, somente leitura quando a fila for aberta.

Isso controla o nome da fila dinâmica usado quando uma fila modelo é aberta. Ele pode ser configurado com um curinga pelo usuário como um conjunto de propriedades (apenas quando a fila estiver fechada) ou como um parâmetro para MQQueueManager.AccessQueue().

O nome real da fila dinâmica é localizado pela consulta de QueueName.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Valores:

- Qualquer nome de fila IBM MQ válido.

Sintaxe: para configurar: *MQQueue* **.DynamicQueueName** = *dynamicqueuename$*

Para obter: *dynamicqueuename$* = *MQQueue* **.DynamicQueueName**

Propriedade HardenGetBackout

Somente leitura. Se para manter uma contagem de retorno precisa.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQA_BACKOUT_HARDENED
- MQQA_BACKOUT_NOT HARDENED

Sintaxe: Para obter: *hardengetback & = MQQueue* **.HardenGetBackout**

Propriedade InhibitGet

Leitura/gravação. O atributo de MQI InhibitGet.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQA_GET_INHIBITED
- MQQA_GET_ALLOWED

Sintaxe: Para obter: *getstatus & = MQQueue* **.InhibitGet** .

Para configurar: *MQQueue* **.InhibitGet** = *getstatus &* .

Propriedade InhibitPut

Leitura/gravação. O atributo InhibitPut MQI.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

Sintaxe: Para obter: *putstatus & = MQQueue* **.InhibitPut**

Para configurar: *MQQueue* **.InhibitPut** = *putstatus &*

Propriedade InitiationQueueName

Somente leitura. Nome da fila de iniciação.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *initqname$ = MQQueue* **.InitiationQueueName**

Propriedade IsOpen

Retorna se a fila está aberta.

Somente leitura.

Definido em: classe MQQueue

Tipo de dado: booleano

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxe: para obter: *open* = *MQQueue* **.IsOpen**

Propriedade MaximumDepth

Somente leitura. Profundidade máxima da fila.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: para obter: *maxdepth & = MQQueue* **.MaximumDepth**

Propriedade MaximumMessageLength

Somente leitura. Comprimento máximo de mensagem permitido em bytes para esta fila.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *maxmlength & = MQQueue* **.MaximumMessageComprimento**

Propriedade MessageDeliverySequence

Somente leitura. Sequência de entrega de mensagens.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQMDS_PRIORITY
- MQMDS_FIFO

Sintaxe: Para obter: *messdelseq & = MQQueue* **.MessageDeliverySequência**

Propriedade Name

Leitura/gravação. O atributo MQI Queue. Esta propriedade não pode ser gravada após o MQQueue estar aberto.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *name$ = MQQueue* **.name**

Para configurar: *MQQueue* **.name** = *name$*

Nota: O Visual Basic reserva a propriedade "Name" para uso na interface visual. Portanto, ao usar no Visual Basic, utilize letras minúsculas, ou seja, "name".

Propriedade ObjectHandle

Somente leitura. A manipulação de objetos para o objeto da fila do IBM MQ.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *hobj & = MQQueue.* . **ObjectHandle**

Propriedade OpenInputCount

Somente leitura. Número de aberturas para entrada.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter:

```
openincount& = MQQueue.OpenInputCount
```

Propriedade OpenOptions

Leitura/gravação. Opções a serem usadas para abrir a fila.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- Consulte OpenOptions (MQLONG).

Sintaxe: Para obter:

```
openopt& = MQQueue.OpenOptions
```

Para configurar: *MQQueue.* **OpenOptions** = *openopt &*

Propriedade OpenOutputCount

Somente leitura. Número de aberturas para saída.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter:

```
openoutcount& = MQQueue.OpenOutputCount
```

Propriedade OpenStatus

Somente leitura. Indica se a fila for aberta ou não. O valor inicial é TRUE após o método AccessQueue ou FALSE após New.

Definido em: classe MQQueue

Tipo de dado: booleano

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxe: Para obter:

```
status& = MQQueue.OpenStatus
```

Propriedade ProcessName

Somente leitura. O atributo ProcessName de MQI.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *procname$* **=** *MQQueue* **.ProcessName**

Propriedade QueueManagerName

Leitura/gravação. O nome do gerenciador de filas do IBM MQ.

Definido em: classe MQQueue

Tipo de dados: sequência

Sintaxe: para obter: *QueueManagerName$* = *MQQueue* **.QueueManagerName**

Para configurar: *MQQueue* **.QueueManagerName** = *QueueManagerName$*

Propriedade QueueType

Somente leitura. O atributo QType MQI.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQT_ALIAS
- MQQT_LOCAL
- MQQT_MODEL
- MQQT_REMOTE

Sintaxe: Para obter: *queuetype & = MQQueue* **.QueueType**

propriedade ReasonCode

Somente leitura. Retorna o código de razão definido pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter: *reasoncode & = MQQueue* **.ReasonCode**

Propriedade ReasonName

Somente leitura. Retorna o nome simbólico do código de razão mais recente. Por exemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definido em: classe MQQueue

Tipo de dados: sequência

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasonname$ = MQQueue* **.ReasonName**

Propriedade RemoteQueueManagerName

Somente leitura. Nome do gerenciador de filas remoto. Válido apenas para filas remotas.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *remqmanname$ = MQQueue* **.RemoteQueueManagerName**

Propriedade RemoteQueueName

Somente leitura. O nome da fila como é conhecido no gerenciador de filas remotas. Válido apenas para filas remotas.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *remqname$ = MQQueue* **.RemoteQueueName**

Propriedade ResolvedQueueManagerName

Somente leitura. O nome do gerenciador de filas de destino final como conhecido para o gerenciador de filas locais.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *resqmanname$ = MQQueue* **.ResolvedQueueManagerName**

Propriedade ResolvedQueueName

Somente leitura. O nome da fila de destino final conforme conhecido para o gerenciador de filas locais.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *resqname$ = MQQueue* **.ResolvedQueueName**

Propriedade RetentionInterval

Somente leitura. O período de tempo pelo qual a fila deve ser retida.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *retinterval & = MQQueue* **.RetentionInterval**

Propriedade Scope

Somente leitura. Controla se uma entrada para esta fila também existe em um diretório de células.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQSCO_Q_MGR
- MQSCO_CELL

Sintaxe: Para obter: *scope & = MQQueue* **.Scope**

Propriedade ServiceInterval

Somente leitura. O MQI atributo QServiceInterval.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *serviceinterval & = MQQueue.* . **ServiceInterval**

Propriedade ServiceIntervalEvent

Somente leitura. O atributo MQI QServiceIntervalEvent.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

Sintaxe: Para obter: *serviceintervalevent & = MQQueue.* **ServiceIntervalEvent**

Propriedade Shareability

Somente leitura. Compartilhamento de Filas.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

Sintaxe: Para obter: *shareability & = MQQueue* **.Shareability**

Propriedade TransmissionQueueName

Somente leitura. Nome da fila de transmissão. Válido apenas para filas remotas.

Definido em: classe MQQueue

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *transqname$ = MQQueue* **.TransmissionQueueName**

Propriedade TriggerControl

Leitura/gravação. Acionador de controle.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQTC_OFF
- MQTC_ON

Sintaxe: Para obter: *trigcontrol & = MQQueue* **.TriggerControl**

Para configurar: *MQQueue* **.TriggerControl** = *trigcontrol &*

Propriedade TriggerData

Leitura/gravação. Dados do acionador.

Definido em: classe MQQueue

Tipo de dado: Sequência de 64 caracteres

Sintaxe: para obter: *trigdata$ = MQQueue* **.TriggerData**

Para configurar: *MQQueue* **.TriggerData** = *trigdata$*

Propriedade TriggerDepth

Leitura/gravação. O número de mensagens que precisam estar na fila antes de uma mensagem do acionador ser gravada.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *trigdepth & = MQQueue* **.TriggerDepth** .

Para configurar: *MQQueue* **.TriggerDepth** = *trigdepth &*

Propriedade TriggerMessagePriority

Leitura/gravação. Prioridade da mensagem limite para acionadores.

Definido em: classe MQQueue

Tipo de dados: longo

Sintaxe: Para obter: *trigmesspriority & = MQQueue* **.TriggerMessagePriority**

Para configurar: *MQQueue* **.TriggerMessagePriority** = *trigmesspriority &*

Propriedade TriggerType

Leitura/gravação. Tipo de acionador.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

Sintaxe: Para obter: *trigtype & = MQQueue* **.TriggerType** .

Para configurar: *MQQueue* **.TriggerType** = *Trigtype &*

Propriedade Usage

Somente leitura. Indica para que a fila é utilizada.

Definido em: classe MQQueue

Tipo de dados: longo

Valores:

- MQUS_NORMAL
- MQUS_TRANSMISSION

Sintaxe: Para obter: *use & = MQQueue* **Uso**

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e o ReasonCode para MQRC_NONE para as classes MQQueue e MQSession.

Definido em: classe MQQueue

Sintaxe:

```
Call MQQueue.ClearErrorCodes()
```

Método Close

Fecha uma fila utilizando os valores atuais dos CloseOptions.

Definido em: classe MQQueue

Sintaxe:

```
Call MQQueue.Close()
```

Método Get

Recupera uma mensagem da fila.

Este método toma um objeto MQMessage como parâmetro, usando alguns dos campos do MQMD do objeto como parâmetros de entrada. Em particular, os campos MessageId e CorrelId são usados,

então é importante assegurar que esses campos estejam configurados conforme necessário. Para mais informações sobre esses campos, consulte MsgId (MQBYTE24) e CorrelId (MQBYTE24).

Se o método falhar, o objeto MQMessage permanecerá inalterado. Se for bem-sucedido, as partes de MQMD e de Dados da mensagem do objeto MQMessage serão substituídas pelo MQMD e os Dados da mensagem a partir da mensagem recebida. As propriedades de controle de MQMessage serão configuradas como segue

- **MessageLength** é configurado para o comprimento da mensagem do IBM MQ
- **DataLength** é configurado para o comprimento da mensagem do IBM MQ
- **DataOffset** é configurado como zero

Definido em:
Classe MQQueue

Sintaxe:

```
Call MQQueue.Get(Message, GetMsgOptions, GetMsgLength)
```

Parâmetros
Mensagem:

Objeto MQMessage representando a mensagem a ser recuperada.

GetMsgOptions:

Objeto MQGetMessageOptions opcional para controlar a operação get. Se esse parâmetro não for especificado, o MQGetMessageOptions padrão será usado.

GetMsgLength:

Valor opcional de 2 ou 4 bytes de comprimento para controlar o comprimento máximo da mensagem do IBM MQ que é recuperada da fila.

Se a opção MQGMO_ACCEPT_TRUNCATED_MSG for especificada, o GET é bem-sucedido com código de conclusão de MQCC_WARNING e um código de razão de MQRC_TRUNCATED_MSG_ACCEPTED se o tamanho da mensagem exceder o comprimento especificado.

MessageData contém os primeiros bytes de dados de GetMsgLength.

Se MQGMO_ACCEPT_TRUNCATED_MSG ***não for*** especificado e o tamanho da mensagem exceder o comprimento especificado, o código de conclusão MQCC_FAILED juntamente ao código de razão MQRC_TRUNCATED_MESSAGE_FAILED serão retornados.

Se o conteúdo do buffer de mensagens estiverem indefinidos, o comprimento total da mensagem será configurado para o comprimento total da mensagem que teria sido recuperada.

Se o parâmetro de comprimento da mensagem não for especificado, o comprimento do buffer de mensagem é ajustado automaticamente para pelo menos o tamanho da mensagem de entrada.

Método Open

Abre uma fila usando os valores atuais de:

1. QueueName
2. QueueManagerName
3. AlternateUserId
4. DynamicQueueName

Definido em:
Classe MQQueue

Sintaxe:

```
Call MQQueue.Open()
```

Método Put

Coloca uma mensagem na fila.

Este método utiliza um objeto MQMessage como parâmetro. As propriedades do Descritor de Mensagens (MQMD) desse objeto podem ser alteradas como resultado deste método. Os valores que elas possuem imediatamente após esse método ter executado são os valores que foram colocados em IBM MQ.

Modificações ao objeto MQMessage após o Put ter sido concluído não afetam a mensagem real na fila do IBM MQ.

Definido em:
Classe MQQueue

Sintaxe:

```
Call MQQueue.Put(Message, PutMsgOptions)
```

Parâmetros

Mensagem

Um objeto MQMessage que representa a mensagem a ser colocada.

PutMsgOptions

O objeto MQPutMessageOptions contendo opções para controlar a operação de entrada. Se eles não forem especificados, PutMessageOptions padrão são usadas.

Classe MQMessage

Esta classe representa uma mensagem do IBM MQ. Ele inclui propriedades para encapsular o descritor de mensagens (MQMD) IBM MQ, e fornece um buffer para reter os dados da mensagem definidos pelo aplicativo.

A classe inclui métodos de gravação para copiar dados a partir de um aplicativo do ActiveX para um objeto do MQMessage. Da mesma forma, a classe inclui métodos de leitura para copiar dados de um objeto do MQMessage para um aplicativo do ActiveX. A classe gerencia a alocação e desalocação de memória para o buffer automaticamente. O aplicativo não tem de declarar o tamanho do buffer quando um objeto MQMessage é criado porque o buffer aumenta para acomodar os dados gravados nele.

Não é possível colocar uma mensagem em uma fila do IBM MQ se o tamanho do buffer exceder a propriedade MaximumMessageLength dessa fila.

Após ter sido construído, um objeto MQMessage pode ser colocado em uma fila do IBM MQ usando o método MQQueue.Put. Este método usa uma cópia do MQMD e as partes de dados da mensagem do objeto e coloca essa cópia na fila. O aplicativo pode, portanto, modificar ou excluir um objeto MQMessage após o Put, sem afetar a mensagem na fila do IBM MQ. O gerenciador de filas pode ajustar alguns dos campos no MQMD ao copiar a mensagem na fila do IBM MQ.

Uma mensagem recebida pode ser lida em um objeto MQMessage usando o método MQQueue.Get. Isso substitui quaisquer dados de MQMD ou de mensagem que podem já ter estado no objeto MQMessage com valores da mensagem que chega. Ele ajusta o tamanho do buffer de dados do objeto MQMessage para corresponder ao tamanho dos dados da mensagem de entrada.

Restrição

As mensagens são contidas pela classe MQSession.

Criação

New cria um objeto MQMessage. Suas propriedades do Descritor de mensagens são inicialmente configuradas como valores padrão e seu buffer de dados da mensagem está vazio.

Sintaxe

```
Dim msg As New MQMessage
```

ou

```
Set msg = New MQMessage
```

Propriedades

As propriedades de controle são:

- “propriedade CompletionCode” na página 624
- “Propriedade DataLength” na página 624
- “Propriedade DataOffset” na página 625
- “Propriedade MessageLength” na página 625
- “propriedade ReasonCode” na página 625
- “Propriedade ReasonName” na página 625

As propriedades do Descritor de mensagens são:

- “Propriedade AccountingToken” na página 626
- “Propriedade AccountingTokenHex” na página 626
- “Propriedade ApplicationIdData” na página 626
- “Propriedade ApplicationOriginData” na página 626
- “Propriedade BackoutCount” na página 627
- “Propriedade CharacterSet” na página 627
- “Propriedade CorrelationId” na página 627
- “Propriedade CorrelationIdHex” na página 627
- “Propriedade Encoding” na página 628
- “Propriedade Expiry” na página 629
- “Propriedade Feedback” na página 629
- “Propriedade Format” na página 629
- “Propriedade GroupId” na página 629
- “Propriedade GroupIdHex” na página 629
- “Propriedade MessageData” na página 630
- “Propriedade MessageFlags” na página 630
- “Propriedade MessageId” na página 630
- “propriedade MessageIdHex” na página 631
- “Propriedade MessageSequenceNumber” na página 631
- “Propriedade MessageType” na página 631
- “Propriedade Offset” na página 631
- “Propriedade OriginalLength” na página 632
- “Propriedade Persistence” na página 632

- “Propriedade Priority” na página 632
- “Propriedade PutApplicationName” na página 632
- “Propriedade PutApplicationType” na página 633
- “Propriedade PutDateTime” na página 633
- “Propriedade ReplyToQueueManagerName” na página 633
- “Propriedade ReplyToQueueName” na página 633
- “Propriedade Report” na página 633
- “Propriedade TotalMessageLength” na página 634
- “Propriedade UserId” na página 634

Methods

- “método ClearErrorCodes” na página 634
- “Método ClearMessage” na página 634
- “Método Read” na página 634
- “Método ReadBoolean” na página 635
- “Método ReadByte” na página 635
- “Método ReadDecimal2” na página 635
- “Método ReadDecimal4” na página 635
- “Método ReadDouble” na página 635
- “Método ReadDouble4” na página 636
- “Método ReadFloat” na página 636
- “Método ReadInt2” na página 636
- “Método ReadInt4” na página 636
- “Método ReadLong” na página 637
- “Método ReadNullTerminatedString” na página 637
- “Método ReadShort” na página 637
- “Método ReadString” na página 637
- “Método ReadUInt2” na página 638
- “Método ReadUnsignedByte” na página 638
- “Método ReadUTF” na página 638
- “Método ResizeBuffer” na página 639
- “Método Write” na página 639
- “Método WriteBoolean” na página 639
- “Método WriteByte” na página 640
- “Método WriteDecimal2” na página 640
- “Método WriteDecimal4” na página 640
- “Método WriteDouble” na página 640
- “Método WriteDouble4” na página 641
- “Método WriteFloat” na página 641
- “Método WriteInt2” na página 641
- “Método WriteInt4” na página 641
- “Método WriteLong” na página 642
- “Método WriteNullTerminatedString” na página 642

- “Método WriteShort” na página 642
- “Método WriteString” na página 643
- “Método WriteUInt2” na página 643
- “Método WriteUnsignedByte” na página 643
- “Método WriteUTF” na página 643

Acesso à propriedade

Todas as propriedades podem ser lidas a qualquer momento.

As propriedades de controle são somente leitura, exceto para DataOffset que são de leitura/gravação. As propriedades do Descritor de mensagens são todas de leitura/gravação, exceto BackoutCount e TotalMessageLength, que são somente leitura.

No entanto, observe que algumas das propriedades MQMD podem ser modificadas pelo gerenciador de filas quando a mensagem é colocada em uma fila do IBM MQ. Consulte os campos no MQMD para obter detalhes de como eles podem ser modificados.

Conversão de Dados

É possível passar dados binários para uma mensagem IBM MQ definindo a propriedade **CharacterSet** para corresponder ao Identificador de conjunto de caracteres codificados do gerenciador de filas (MQCCSI_Q_MGR) e transmitir os dados para a mensagem como uma sequência. Se a sequência precisar incluir números de código Unicode ou ASCII, é possível usar a função chr$ para convertê-los em formato de sequência.

Os métodos de leitura e gravação executam a conversão de dados. Eles convertem entre os formatos internos ActiveX e os formatos de mensagem IBM MQ conforme definido pelas propriedades Encoding e CharacterSet do descritor de mensagens. Ao gravar uma mensagem, configure os valores de Encoding e CharacterSet que correspondem às características do destinatário da mensagem antes de emitir um método Write. Ao ler uma mensagem, esta etapa não é normalmente necessária porque esses valores terão sido configurados a partir daqueles no MQMD recebido.

Esta é uma etapa de conversão de dados adicionais que ocorre após qualquer conversão executada pelo método MQQueue.Get.

propriedade CompletionCode

Somente leitura. Retorna o código de conclusão IBM MQ configurado pelo acesso de propriedade ou método mais recente emitido para esse objeto.

Definido em: classe MQMessage

Tipo de dados: longo

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode & = MQMessage* **.CompletionCode**

Propriedade DataLength

Somente leitura. Essa propriedade retorna o valor:

```
MQMessage.MessageLength - MQMessage.DataOffset
```

Ele pode ser usado antes de um método Read para verificar se o número esperado de caracteres está realmente presente no buffer.

O valor inicial é zero.

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *bytesleft & = MQMessage* **.DataLength**

Propriedade DataOffset

Leitura/gravação. A posição atual na parte Dados da mensagem do objeto de mensagem.

O valor é expresso como um deslocamento de bytes desde o início do buffer de dados da mensagem; o primeiro caractere no buffer corresponde a um valor DataOffset de zero.

Um método de leitura ou gravação inicia sua operação no caractere referenciado pelo DataOffset. Esses métodos processam dados no buffer sequencialmente a partir dessa posição e atualizam o DataOffset para apontar para o byte (se houver) imediatamente após o último byte processado.

DataOffset pode aceitar somente valores no intervalo zero a MessageLength inclusive. Quando DataOffset = MessageLength, ele está apontando para o final, que é o primeiro caractere inválido do buffer. Métodos de gravação são permitidos nesta situação - eles estendem os dados no buffer e aumentam o MessageLength no número de bytes incluídos. Ler além do final do buffer não é válido.

O valor inicial é zero.

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *currpos & = MQMessage* **.DataOffset**

Para configurar: *MQMessage* **.DataOffset** = *currpos &*

Propriedade MessageLength

Somente leitura. Retorna o comprimento total da parte Dados da Mensagem do objeto de mensagem em caracteres, independentemente do valor de DataOffset.

O valor inicial é zero. É definido para inserir o Comprimento da Mensagem após uma chamada de método Get que referenciou este objeto de mensagem. Ele é incrementado se o aplicativo usa um método de gravação para incluir dados no objeto. Ele não é afetado por métodos Read.

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *msglength & = MQMessage* **.MessageLength** .

propriedade ReasonCode

Somente leitura. Retorna o código de razão definido pelo método mais recente ou acesso de propriedade emitidos para este objeto.

Definido em: classe MQMessage

Tipo de dados: longo

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasoncode & = MQMessage* **.ReasonCode**

Propriedade ReasonName

Somente leitura. Retorna o nome simbólico do código de razão mais recente. Por exemplo, "MQRC_QMGR_NOT_AVAILABLE". **Definido em:** classe MQMessage

Tipo de dados: sequência

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasonname$ = MQMessage* **.ReasonName**

Propriedade AccountingToken

Leitura/gravação. O AccountingToken MQMD – parte do Contexto de identidade da mensagem.

Seu valor inicial é todos nulos.

Definido em: classe MQMessage

Tipo de dado: sequência de 32 caracteres

Sintaxe: para obter: *actoken$ = MQMessage* **.AccountingToken**

Para configurar: *MQMessage.* **AccountingToken** = *actoken$*

Consulte “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando deve-se usar AccountingTokenHex no lugar da propriedade AccountingToken.

Propriedade AccountingTokenHex

Leitura/gravação. O AccountingToken MQMD – parte do Contexto de identidade da mensagem.

Cada dois caracteres representam o equivalente hexadecimal de um caractere ASCII único. Por exemplo, o par de caracteres "6" e "1" representa o caractere único "A", o par de caracteres "6" e "2" representa o caractere único "B" e assim por diante.

Deve-se fornecer 64 caracteres hexadecimais válidos.

Seu valor inicial é "0...0"

Definido em: classe MQMessage

Tipo de dados: sequência de 64 caracteres hexadecimais representando 32 caracteres ASCII

Sintaxe: para obter: *actokenh$ = MQMessage* **.AccountingTokenHex**

Para configurar: *MQMessage.* **AccountingTokenHex** = *actokenh$*

Consulte “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando deve-se usar AccountingTokenHex no lugar da propriedade AccountingToken.

Propriedade ApplicationIdData

Leitura/gravação. O MQMD ApplIdentityData - parte do Contexto de Identidade da mensagem.

Seu valor inicial é todos os espaços em branco.

Definido em: classe MQMessage

Tipo de dado: sequência de 32 caracteres

Sintaxe: para obter: *applid$ = MQMessage* **.ApplicationIdData**

Para configurar: *MQMessage* **.ApplicationIdData** = *applid$*

Propriedade ApplicationOriginData

Leitura/gravação. O MQMD ApplOriginData - parte do contexto de origem da mensagem.

Seu valor inicial é todos os espaços em branco.

Definido em: classe MQMessage

Tipo de dado: Sequência de 4 caracteres

Sintaxe: para obter: *applor$ = MQMessage* **.ApplicationOriginData**

Para configurar: *MQMessage* **.ApplicationOriginData** = *applor$*

Propriedade BackoutCount

Somente leitura. O BackoutCount de MQMD.

Seu valor inicial é 0

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *backoutct & = MQMessage* **.BackoutCount**

Propriedade CharacterSet

Leitura/gravação. O MQMD CodedCharSetId.

Seu valor inicial é o valor especial **MQCCSI_Q_MGR**.

Se o CharacterSet estiver configurado como **MQCCSI_Q_MGR**, a página de códigos para o código do idioma atual será usada para a conversão de caracteres no método WriteString. Para aplicativos do servidor, a página de códigos usada é a página de códigos do gerenciador de filas. Para aplicativos clientes, é a página de códigos do idioma atual padrão.

Por exemplo:

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

em que 'n' é maior ou igual a zero e menor ou igual a 255, resulta na gravação de um único byte de valor 'n' no buffer.

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *:30ccid& = MQMessage* **.CharacterSet**

Para configurar: *MQMessage* **.CharacterSet** = *ccid &*

exemplo

Se desejar que a sequência seja gravada na página de códigos 437, emita:

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

Configure o valor que deseja em CharacterSet antes de emitir quaisquer chamadas WriteString.

Propriedade CorrelationId

Leitura/gravação. O CorrelationId a ser incluído no MQMD de uma mensagem quando colocada em uma fila. Além disso, o ID a ser correspondido em relação ao obter uma mensagem de uma fila.

Seu valor inicial é nulo.

Definido em: classe MQMessage

Tipo de dado: sequência de 24 caracteres

Sintaxe: para obter: *correlid$ = MQMessage* **.CorrelationId** para configurar: *MQMessage* **.CorrelationId** = *correlid$*

Veja “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando se deve usar CorrelationIdHex em vez da propriedade CorrelationId.

Propriedade CorrelationIdHex

Leitura/gravação. O CorrelationId a ser incluído no MQMD de uma mensagem quando colocada em uma fila. Além disso, o CorrelationId a ser correspondido em relação ao obter uma mensagem de uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representa o caractere único "A", o par de caracteres "6" e "2" representa o caractere único "B" e assim por diante.

Deve-se fornecer 48 caracteres hexadecimais válidos.

Seu valor inicial é "0...0".

Definido em: classe MQMessage

Tipo de dados: sequência de 48 caracteres hexadecimais representando 24 caracteres ASCII

Sintaxe: para obter: *correlidh$ = MQMessage* **.CorrelationIdHex**

Para configurar: *MQMessage* **.CorrelationIdHex** = *correlidh$*

Veja “Propriedades do Descritor de Mensagens” na página 585 para obter uma discussão de quando se deve usar CorrelationIdHex em vez da propriedade CorrelationId.

Propriedade Encoding

Leitura/gravação. O campo MQMD que identifica a representação usada para valores numéricos nos dados de mensagem do aplicativo.

Seu valor inicial é o valor especial MQENC_NATIVE, que varia por plataforma.

Essa propriedade é usada pelos métodos a seguir:

- Método ReadDecimal2
- Método ReadDecimal4
- Método ReadDouble
- Método ReadDouble4
- Método ReadFloat
- Método ReadInt2
- Método ReadInt4
- Método ReadLong
- Método ReadShort
- Método ReadUInt2
- Método WriteDecimal2
- Método WriteDecimal4
- Método WriteDouble
- Método WriteDouble4
- Método WriteFloat
- Método WriteInt2
- Método WriteInt4
- Método WriteLong
- Método WriteShort
- Método WriteUInt2

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *encoding & = MQMessage* **Codificação** Para configurar: *MQMessage* **.Codificação** = *codificação &*

Se estiver se preparando para gravar dados no buffer de mensagem, é necessário configurar esse campo para que corresponda às características da plataforma do gerenciador de filas de recebimento se o gerenciador de filas de recebimento for incapaz de executar sua própria conversão de dados.

Propriedade Expiry

Leitura/gravação. O campo de tempo de expiração do MQMD, esperado em décimos de segundo.

Seu valor inicial é o valor especial MQEI_UNLIMITED

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *expirar & = MQMessage* **Expiração**

Para configurar: *MQMessage* . **Expiração** = *expiração &*

Propriedade Feedback

Leitura/gravação. O campo feedback MQMD.

Seu valor inicial é o valor especial MQFB_NONE.

Definido em: classe MQMessage

Tipo de dados: longo

Valores:

- Veja Feedback.

Sintaxe: Para obter: *feedback & = MQMessage* **Feedback**

Para configurar: *MQMessage* . **Feedback** = *feedback &*

Propriedade Format

Leitura/gravação. O campo de formato MQMD. Fornece o nome de um formato integrado ou definido pelo usuário que descreve a natureza dos dados da mensagem.

Seu valor inicial é o valor especial MQFMT_NONE.

Definido em: classe MQMessage

Tipo de dados: Sequência de 8 caracteres

Sintaxe: para obter: *format$ = MQMessage* **.Format**

Para configurar: *MQMessage* **.Format** = *format$*

Propriedade GroupId

Leitura/gravação. O GroupId a ser incluído na MQPMR de uma mensagem quando colocar em uma fila. Além disso, o ID a ser correspondido em relação ao obter uma mensagem de uma fila. Seu valor inicial é todos nulos.

Definido em:
Classe MQMessage

Tipo de dados:
Sequência de 24 caracteres

Sintaxe: Para obter: *groupid MQMessage = $*. **ID do Grupo**

Para configurar: *MQMessage*. **GroupId** = *groupid$*

Veja “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando se deve usar GroupIdHex em vez da propriedade GroupId.

Propriedade GroupIdHex

Leitura/gravação. O GroupId a ser incluído na MQPMR de uma mensagem quando colocar em uma fila. Além disso, o ID a ser correspondido em relação ao obter uma mensagem de uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representam o caractere único "A", o par de caracteres "6" e "2" representam o caractere único "B" e assim por diante.

Deve-se fornecer 48 caracteres hexadecimais válidos.

Seu valor inicial é "0...0".

Definido em:
Classe MQMessage

Tipo de dados:
Sequência de 48 caracteres hexadecimais que representam 24 caracteres ASCII.

Sintaxe: Para obter: *groupidh MQMessage = $*. **GroupIdHex**

Para configurar: *MQMessage*. **GroupIdHex** = *groupidh$*

Veja “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando se deve usar GroupIdHex em vez da propriedade GroupId.

Propriedade MessageData

Leitura/gravação. Recupera ou define todo o conteúdo de uma mensagem como uma sequência de caracteres.

Definido em: classe MQMessage

Data Type: Variant

Nota: O tipo de dados usado por esta propriedade é variante, mas o MQAX espera que esta seja um tipo variante de sequência. Se você passar em uma variante de tipo diferente desta, o erro MQRC_OBJECT_TYPE_ERROR será retornado.

Sintaxe: para obter: *String$ = MQMessage* **.MessageData**

Para configurar: *MQMessage* **.MessageData** = *String$*

Propriedade MessageFlags

Leitura/Gravação. Sinalizações de mensagem especificando informações de controle de segmentação. O valor inicial é 0.

Definido em:
Classe MQMessage

Tipo de dados:
Long

Valores:
Veja MsgFlags (MQLONG).

Sintaxe: Para obter: *messageflags & = MQMessage* **MessageFlags**

Para configurar: *MQMessage*. **MessageFlags** = *messageflags &*

Propriedade MessageId

Leitura/gravação. O MessageId a ser incluído no MQMD de uma mensagem quando colocada em uma fila. Além disso, o ID a ser correspondido em relação ao obter uma mensagem de uma fila.

Seu valor inicial é todos nulos.

Definido em: classe MQMessage

Tipo de dado: sequência de 24 caracteres

Sintaxe: para obter: *messageid$ = MQMessage* **.MessageId**

Para configurar: *MQMessage* **.MessageId** = *messageid$*

Consulte “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando deve-se usar MessageIdHex no lugar da propriedade MessageId.

propriedade MessageIdHex

Leitura/gravação. O MessageId a ser incluído no MQMD de uma mensagem quando colocada em uma fila. Além disso, o MessageId a ser correspondido em relação a quando obter uma mensagem de uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representa o caractere único "A", o par de caracteres "6" e "2" representa o caractere único "B" e assim por diante.

Deve-se fornecer 48 caracteres hexadecimais válidos.

Seu valor inicial é "0...0".

Definido em: classe MQMessage

Tipo de dados: sequência de 48 caracteres hexadecimais representando 24 caracteres ASCII

Sintaxe: para obter: *messageidh$ = MQMessage* **.MessageIdHex**

Para configurar: *MQMessage* **.MessageIdHex** = *messageidh$*

Consulte “Propriedades do Descritor de Mensagens” na página 585 para obter mais informações sobre quando deve-se usar MessageIdHex no lugar da propriedade MessageId.

Propriedade MessageSequenceNumber

Leitura/Gravação. Informações de sequência que identificam uma mensagem em um grupo. O valor inicial é 1.

Definido em:
Classe MQMessage

Tipo de dados:
Long

Valores:
Consulte MsgSeqNumber (MQLONG).

Sintaxe: Obter: *sequencenumber & = MQMessage*. **SequenceNumber**

Para configurar: *MQMessage*. **SequenceNumber** = *sequencenumber &*

Propriedade MessageType

Leitura/gravação. O campo MQMD MsgType.

Seu valor inicial é MQMT_DATAGRAM.

Definido em: classe MQMessage

Tipo de dados: longo

Valores:

- Consulte MsgType (MQLONG).

Sintaxe: Para obter: *msgtype & = MQMessage* **.MessageType**

Para configurar: *MQMessage* **.MessageType** = *msgtype &*

Propriedade Offset

Leitura/Gravação. O deslocamento em uma mensagem segmentada. O valor inicial é 0.

Definido em:
Classe MQMessage

Tipo de dados:
Long

Valores:
Veja Offset (MQLONG).

Sintaxe: Obter: *offset & = MQMessage*. **Offset**

Para configurar: *MQMessage*. **Deslocamento** = *deslocamento &*

Propriedade OriginalLength

Leitura/Gravação. O comprimento original de uma mensagem segmentada. O valor inicial é MQOL_UNDEFINED.

Definido em:
Classe MQMessage

Tipo de dados:
Long

Valores:
Veja OriginalLength (MQLONG).

Sintaxe: Para obter: *originallength & = MQMessage* **OriginalLength**

Para configurar: *MQMessage*. **OriginalLength** = *originallength &*

Propriedade Persistence

Leitura/gravação. A configuração de persistência da mensagem.

Seu valor inicial é MQPER_PERSISTENCE_AS_Q_DEF.

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: para obter: *persist & = MQMessage* **.Persistência**

Para configurar: *MQMessage* . **Persistência** = *persistente &*

Propriedade Priority

Leitura/gravação. A prioridade da mensagem.

Seu valor inicial é o valor especial MQPRI_PRIORITY_AS_Q_DEF

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: para obter: *priority & = MQMessage* **Prioridade**

Para configurar: *MQMessage* . **Prioridade** = *prioridade &*

Propriedade PutApplicationName

Leitura/gravação. O MQMD PutApplName - parte do contexto de Origem da Mensagem.

Seu valor inicial é todos os espaços em branco.

Definido em: classe MQMessage

Tipo de dado: Sequência de 28 caracteres

Sintaxe: para obter: *putapplnm$ = MQMessage* **.PutApplicationName**

Para configurar: *MQMessage* **.PutApplicationName** = *putapplnm$*

Propriedade PutApplicationType

Leitura/gravação. O MQMD PutApplType – parte do contexto de origem da mensagem.

Seu valor inicial é MQAT_NO_CONTEXT

Definido em: classe MQMessage

Tipo de dados: longo

Valores:

- Consulte PutApplType (MQLONG).

Sintaxe: Para obter: *putappltp & = MQMessage* **.PutApplicationTipo**

Para configurar: *MQMessage* **.PutApplicationType** = *putappltp &*

Propriedade PutDateTime

Leitura/gravação. Esta propriedade combina os campos MQMD PutDate e PutTime. Esses são partes do contexto de Origem de mensagens que indicam quando a mensagem foi colocada.

A Extensão ActiveX converte entre o formato de data/hora ActiveX e os formatos Data e Hora usados em um IBM MQ MQMD. Se for recebida uma mensagem que possui um PutDate ou PutTime inválido, a propriedade PutDateTime após o método get é configurada como EMPTY.

Seu valor inicial é EMPTY.

Definido em: classe MQMessage

Tipo de Dado: Variante de tipo 7 (data/hora) ou EMPTY.

Sintaxe: para obter: *datetime = MQMessage* **.PutDateTime**

Para configurar: *MQMessage* **.PutDateTime** = *datetime*

Propriedade ReplyToQueueManagerName

Leitura/gravação. O campo MQMD ReplyToQMgr.

Seu valor inicial é todos os espaços em branco

Definido em: classe MQMessage

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *replytoqmgr$ = MQMessage* **.ReplyToQueueManagerName**

Para configurar: *MQMessage* **.ReplyToQueueManagerName** = *replytoqmgr$*

Propriedade ReplyToQueueName

Leitura/gravação. O campo MQMD ReplyToQ.

Seu valor inicial é todos os espaços em branco

Definido em: classe MQMessage

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *replytoq$ = MQMessage* **.ReplyToQueueName**

Para configurar: *MQMessage* **.ReplyToQueueName** = *replytoq$*

Propriedade Report

Leitura/gravação. As opções de relatório da mensagem.

Seu valor inicial é MQRO_NONE.

Definido em: classe MQMessage

Tipo de dados: longo

Valores:

- Consulte Relatório.

Sintaxe: Para obter: *report & = MQMessage* **Relatório**

Para configurar: *MQMessage* . **Relatório** = *relatório &*

Propriedade TotalMessageLength

Somente leitura. Recupera o comprimento da última mensagem recebida pela chamada MQGET. Se a mensagem não foi truncada, este valor é igual ao valor da propriedade MessageLength.

Definido em: classe MQMessage

Tipo de dados: longo

Sintaxe: Para obter: *totalmessagelength & = MQMessage* **.TotalMessageComprimento**

Propriedade UserId

Leitura/gravação. O MQMD UserIdentifier - parte do Contexto de Identidade da mensagem.

Seu valor inicial é todos os espaços em branco.

Definido em: classe MQMessage

Tipo de dados: sequência de 12 caracteres

Sintaxe: para obter: *userid$ = MQMessage* **.UserId**

Para configurar: *MQMessage* **.UserId** = *userid$*

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode para a classe MQMessage e a classe MQSession.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.ClearErrorCodes()
```

Método ClearMessage

Esse método limpa a parte do buffer de dados do objeto MQMessage. Quaisquer Dados de Mensagem no buffer de dados que forem perdidos, como MessageLength, DataLength e DataOffset, são todos configurados para zero.

A parte do Descritor de mensagens (MQMD) não é afetada; um aplicativo pode precisar modificar alguns dos campos do MQMD antes de reutilizar o objeto MQMessage. Para definir os campos de MQMD, use Novo de volta para substituir o objeto por uma nova instância.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.ClearMessage()
```

Método Read

Lê uma sequência de bytes do buffer de mensagem em uma matriz de bytes. O DataOffset é aumentado e o Data Lenght diminuído no número de bytes lidos.

Definido em:
Classe MQMessage

Syntax: Data = MQMessage. **Leitura** (len &)

Parâmetros:
len &: longo. Comprimento de dados em bytes a serem lidos.

Método ReadBoolean

Lê um valor booleano de 1 byte da posição atual no buffer de mensagem e retorna um valor booleano de 2 bytes TRUE(-1)/FALSE(0). DataOffset é incrementado em um e dados de Comprimento será diminuído por um.

Definido em:
Classe MQMessage

Sintaxe: *value = MQMessage*. **ReadBoolean**

Método ReadByte

Iniciando com o byte referenciado por DataOffset, o método ReadByte lê 1 byte do buffer Dados da mensagem e o retorna como um valor de número inteiro (2 bytes assinados) no intervalo de -128 a 127.

O método falha se MQMessage.DataLength for inferior a 1 quando for emitido.

DataOffset é incrementado em 1 e DataLength é decrementado em 1 se o método for bem-sucedido.

O byte de dados da mensagem é assumido para ser um número inteiro binário assinado.

Definido em:
Classe MQMessage

Sintaxe:
integerv% = MQMessage **.ReadByte**

Método ReadDecimal2

Lê um número decimal compactado de 2 bytes e retorna-o como um valor de número inteiro de 2 bytes assinado. DataOffset é incrementado por dois e Data Length é reduzida por dois.

Definido em:
Classe MQMessage

Sintaxe: *value% = MQMessage*. **ReadDecimal2**

Método ReadDecimal4

Lê um número decimal compactado de 4 bytes e retorna-o como um valor de número inteiro de 4 bytes assinado. DataOffset é incrementado em quatro e Dados Comprimento será diminuído por quatro.

Definido em:
Classe MQMessage

Sintaxe:

```
Call value& = MQMessage.ReadDecimal4
```

Método ReadDouble

Iniciando com o byte referenciado por DataOffset, o método ReadDouble lê 8 bytes do buffer Dados da mensagem e os retorna como um valor de ponto flutuante duplo (8 bytes assinados).

O método falha se MQMessage.DataLength for menor que 8 quando ele é emitido.

DataOffset é incrementado em 8 e DataLength é decrementado em 8 se o método for bem-sucedido.

Os 8 caracteres de dados da mensagem são considerados como sendo um número de vírgula flutuante binário. A codificação é especificada pela propriedade MQMessage.Encoding. Observe que a conversão a partir do formato de System/360 não é suportada.

Definido em:
Classe MQMessage

Sintaxe:
doublev# = MQMessage **.ReadDouble**

Método ReadDouble4

Os métodos ReadDouble4 e WriteDouble4 são alternativas para ReadFloat e WriteFloat. Isso ocorre porque eles suportam valores de mensagem do ponto flutuante do System/390 de 4 bytes que são muito grandes para converter no formato de ponto flutuante IEEE de 4 bytes.

Iniciando com o byte referido por DataOffset, o método ReadDouble4 lê 4 bytes do buffer Dados da mensagem e os retorna como um valor de ponto flutuante duplo (8 bytes assinados).

O método falhará se MQMessage.DataLength for menor que 4 quando emitido.

DataOffset será incrementado em 4 e DataLength será decrementado em 4 se o método for bem-sucedido.

Os 4 caracteres de dados da mensagem são considerados como sendo um número de vírgula flutuante binário. A codificação é especificada pela propriedade MQMessage.Encoding. Observe que a conversão a partir do formato de System/360 não é suportada.

Definido em:
Classe MQMessage

Sintaxe:
doublev# = MQMessage. **ReadDouble4**

Método ReadFloat

Iniciando com o byte referido por DataOffset, o método ReadFloat lê 4 bytes do buffer de dados da mensagem e os retorna como um único (assinado 4-byte) valor de ponto flutuante.

O método falhará se MQMessage.DataLength for menor que 4 quando emitido.

DataOffset será incrementado em 4 e DataLength será decrementado em 4 se o método for bem-sucedido.

Os 4 caracteres de dados da mensagem são considerados como sendo um número de vírgula flutuante. A codificação é especificada pela propriedade MQMessage.Encoding. Observe que a conversão a partir do formato de System/360 não é suportada.

Definido em:
Classe MQMessage

Sintaxe:
singlev! = MQMessage **.ReadFloat**

Método ReadInt2

O método é idêntico ao método ReadShort.

Sintaxe:
integerv% = MQMessage **.ReadInt2**

Método ReadInt4

Este método é idêntico ao método ReadLong.

Sintaxe:
bigint & = MQMessage **.ReadInt4**

Método ReadLong

Iniciando com o byte referido por DataOffset, o método ReadLong lê 4 bytes a partir do buffer de Dados de Mensagem e os retorna como um valor de número inteiro Longo (4 bytes assinado).

O método falhará se MQMessage.DataLength for menor que 4 quando emitido.

DataOffset será incrementado em 4 e DataLength será decrementado em 4 se o método for bem-sucedido.

Os quatro caracteres de dados da mensagem são considerados como sendo um número inteiro binário. A codificação é especificada pela propriedade MQMessage.Encoding.

Definido em:
Classe MQMessage

Sintaxe:
bigint & = MQMessage **.ReadLong**

Método ReadNullTerminatedString

Esse método é para ser usado no lugar de ReadString se a sequência possivelmente contiver caracteres nulos integrados.

Esse método lê o número especificado de bytes do buffer de dados de mensagem começando com o byte referido por DataOffset e retorna o mesmo como uma sequência do ActiveX. Se a sequência contiver um nulo integrado antes do fim, então, o comprimento da sequência retornada é reduzido para refletir somente os caracteres antes do nulo.

DataOffset é incrementado e DataLength é diminuído pelo valor especificado, independentemente de se a sequência contém caracteres nulos integrados.

Os caracteres nos dados da mensagem são considerados como uma sequência na página de códigos especificada pela propriedade MQMessage.CharacterSet. A conversão para representação do ActiveX é executada para o aplicativo.

Definido em:
Classe MQMessage

Sintaxe: *string$ = MQMessage*. **ReadNullTerminatedString(comprimento &)**

Parâmetros:
comprimento & *Longo.* O comprimento do campo de sequência em bytes.

Método ReadShort

Iniciando com o byte referido por DataOffset, o método ReadShort lê 2 bytes do buffer de Dados de Mensagem e os retorna como um valor de número inteiro (2 bytes assinado).

O método falhará se MQMessage.DataLength for menor que 2 quando emitido.

DataOffset será incrementado em 2 e DataLength será decrementado em 2 se o método for bem-sucedido.

Os dois caracteres de dados da mensagem são considerados como sendo um número inteiro binário. A codificação é especificada pela propriedade MQMessage.Encoding.

Definido em:
Classe MQMessage

Sintaxe:
integerv% = MQMessage **.ReadShort**

Método ReadString

Esse método lê n bytes do buffer de Dados de mensagem começando com o byte referido por DataOffset e o retorna como uma sequência do ActiveX.

O método falha se MQMessage.DataLength for menor que n quando emitido.

DataOffset é incrementado por n e DataLength é reduzido por n se o método for bem-sucedido.

Os n caracteres dos dados da mensagem são considerados como uma sequência na página de códigos especificada pela propriedade MQMessage.CharacterSet. A conversão para representação do ActiveX é executada para o aplicativo.

Definido em: classe MQMessage

Sintaxe: *stringv $= MQMessage* **.ReadString** *(comprimento &*)

Parâmetro
length & longo. O comprimento do campo de sequência em bytes.

Método ReadUInt2

Iniciando com o byte referido por DataOffset, o método ReadUInt2 lê 2 bytes do buffer de Dados de Mensagem e os retorna como um valor de número inteiro Longo (4 bytes assinado).

O método falhará se MQMessage.DataLength for menor que 2 quando emitido.

DataOffset será incrementado em 2 e DataLength será decrementado em 2 se o método for bem-sucedido.

Os 2 bytes de dados da mensagem são considerados como sendo um número inteiro binário não assinado. A codificação é especificada pela propriedade MQMessage.Encoding.

Definido em:
Classe MQMessage

Sintaxe:
bigint & = MQMessage **.ReadUInt2**

Método ReadUnsignedByte

Começando com o byte referenciado pelo DataOffset, o método ReadUnsignedByte lê 1 byte do buffer Dados da mensagem e retorna-o como um valor de Número inteiro (2 bytes assinados) no intervalo de 0 a 255.

O método falha se MQMessage.DataLength for inferior a 1 quando for emitido.

DataOffset é incrementado em 1 e DataLength é decrementado em 1 se o método for bem-sucedido.

O 1 caractere dos dados da mensagem é assumido para ser um número inteiro binário não assinado.

Definido em:
Classe MQMessage

Sintaxe:
integerv% = MQMessage **.ReadUnsignedByte**

Método ReadUTF

Esse método lê um formato de sequência UTF da mensagem, iniciando a partir do byte referido por **DataOffset** e retorna a sequência de formato UTF como uma sequência ActiveX. A sequência de formato UTF que está sendo lida consiste em 2 bytes de dados que indicam o comprimento da sequência seguidos pelos dados de caracteres UTF.

O método falha se **MQMessage.DataLength** for menor que o comprimento da sequência quando ele é emitido.

DataOffset é incrementado pelo comprimento da sequência e **DataLength** será diminuído pelo comprimento da sequência se o método for bem-sucedido.

Definido em:
Classe MQMessage

Sintaxe:

```
value$ = MQMessage.ReadUTF
```

Método ResizeBuffer

Esse método altera a quantia de armazenamento atualmente alocada internamente para conter o buffer de Dados de mensagem. Ela fornece ao aplicativo algum controle sobre o gerenciamento de buffer automático, em que, se o aplicativo souber que irá lidar com uma mensagem grande, poderá assegurar que um buffer grande o suficiente esteja alocado. O aplicativo não precisa usar essa chamada – se não usar, o código de gerenciamento de buffer automático aumentará o tamanho do buffer para ajustar.

Se você redimensionar o buffer a ser menor que o MessageLength atual, corre o risco de perder dados. Se perder dados, o método retorna um CompletionCode igual a MQCC_WARNING e um ReasonCode igual a MQRC_DATA_TRUNCATED.

Se redimensionar o buffer para ser menor que o valor da propriedade **DataOffset**, a:

- Propriedade **DataOffset** mudará para apontar para o fim do novo buffer
- Propriedade **DataLength** será configurada para zero
- Propriedade **MessageLength** mudará para o novo tamanho do buffer

Definido em:
Classe MQMessage

Sintaxe: *MQMessage* **.ResizeBuffer** (*Comprimento &*)

Parâmetro:
Length& Long. Tamanho necessário em caracteres.

Método Write

Grava uma sequência de bytes para o buffer de mensagem a partir de uma matriz de byte na posição referida por Deslocamento de dados. Se necessário, o comprimento do buffer (MQMessage.MQMessageLength) é estendido para acomodar o comprimento total da matriz de bytes. DataOffset é incrementado pelo número de bytes gravados se o método for bem-sucedido.

Definido em:
Classe MQMessage

Sintaxe:

```
Call MQMessage.Write(value)
```

Parâmetros:
data: uma matriz de bytes ou uma referência variante para uma matriz de bytes

Método WriteBoolean

Grava um valor booleano de 1 byte na posição atual no buffer de mensagem de um valor booliano de 2 bytes. DataOffset é incrementado em um.

Definido em:
Classe MQMessage

Sintaxe:

```
Call MQMessage.WriteBoolean(value)
```

Parâmetro:
value: Boolean (2-bytes). Valor a ser gravado.

Método WriteByte

Este método usa um valor inteiro de 2 bytes assinados e o grava no buffer Dados da mensagem como um número binário de 1 byte na posição referenciada por DataOffset. Ele substitui quaisquer dados já na posição no buffer, e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

DataOffset é incrementado em um se o método for bem-sucedido.

O valor especificado deve estar no intervalo de -128 a 127. Se não estiver, o método retorna com CompletionCode MQCC_FAILED e ReasonCode MQRC_WRITE_VALUE_ERROR.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteByte(value%)
```

Parameter: *value%* Integer. Valor a ser gravado.

Método WriteDecimal2

Grava um número inteiro de 2 bytes assinado como um número decimal compactado de 2 bytes. DataOffset é incrementado por dois.

Definido em:
Classe MQMessage

Sintaxe:

```
Call MQMessage.WriteDecimal2(value%)
```

Parâmetro:
value% Integer. Valor a ser gravado.

Método WriteDecimal4

Grava um número inteiro de 4 bytes assinado como um número decimal compactado de 4 bytes. DataOffset é incrementado por quatro.

Definido em:
Classe MQMessage

Sintaxe:

```
Call MQMessage.WritedDecimal4(value&)
```

Parâmetro:
valor & Long. Valor a ser gravado.

Método WriteDouble

Este método usa um valor de vírgula flutuante de 8 bytes assinados e o grava no buffer Dados da mensagem como um número de vírgula flutuante de 8 bytes começando na posição referenciada por DataOffset. Ele substitui quaisquer dados que já estejam nessas posições no buffer e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

DataOffset é incrementado em 8 se o método for bem-sucedido.

O método converte para a representação de ponto flutuante especificada pela propriedade MQMessage.Encoding. *A conversão para o formato System/360 não é suportada.*

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteDouble(value#)
```

Parâmetro:
value# Double. Valor a ser gravado.

Método WriteDouble4

Consulte “Método ReadDouble4” na página 636 para obter uma descrição de quando ReadDouble4 e WriteDouble4 devem ser usados no lugar de ReadFloat e WriteFloat.

Este método obtém um valor de ponto flutuante de 8 bytes assinado e o grava no buffer de Dados de Mensagem como um número flutuante de 4 bytes começando na posição referida por DataOffset.

DataOffset será incrementado em 4 se o método for bem-sucedido.

Ele substitui quaisquer dados que já estejam nessas posições no buffer e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

O método converte para a representação de ponto flutuante especificada pela propriedade MQMessage.Encoding. *A conversão para o formato System/360 não é suportada.*

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteDouble4(value#)
```

Parâmetro: *value#* Double. Valor a ser gravado.

Método WriteFloat

Este método usa um valor de vírgula flutuante de 4 bytes assinados e o grava no buffer Dados da mensagem como um número de vírgula flutuante de 4 bytes iniciando no caractere referido por DataOffset. Ele substitui quaisquer dados que já estejam nessas posições no buffer e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

DataOffset será incrementado em 4 se o método for bem-sucedido.

O método converte para a representação binária especificada pela propriedade MQMessage.Encoding. *A conversão para o formato System/360 não é suportada.*

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteFloat(value!)
```

Parameter *value!* Flutuante. Valor a ser gravado.

Método WriteInt2

Este método é idêntico ao método WriteShort.

Sintaxe:

```
Call MQMessage.WriteInt2(value%)
```

Parameter *value%* Integer. Valor a ser gravado.

Método WriteInt4

Este método é idêntico ao método WriteLong.

Sintaxe:

```
Call MQMessage.WriteInt4(value&)
```

Parâmetro *valor &* longo. Valor a ser gravado.

Método WriteLong

Este método usa um valor inteiro de 4 bytes assinado e grava-o no buffer de Dados da mensagem como um número binário de 4 bytes iniciando no byte referenciado por DataOffset. Ele substitui quaisquer dados que já estejam nessas posições no buffer e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

DataOffset será incrementado em 4 se o método for bem-sucedido.

O método converte para a representação binária especificada pela propriedade MQMessage.Encoding.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteLong(value&)
```

Parâmetro *valor &* longo. Valor a ser gravado.

Método WriteNullTerminatedString

Este método executa uma WriteString normal e preenche os bytes restantes até o comprimento especificado com nulo. Se o número de bytes gravados pela sequência de gravação inicial for igual ao comprimento especificado, nenhum nulo será gravado. Se o número de bytes exceder o comprimento especificado, um erro (código de razão MQRC_WRITE_VALUE_ERROR) será configurado.

DataOffset será incrementado pelo comprimento especificado se o método for bem-sucedido.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteNullTerminatedString(value$, length&)
```

Parâmetros:
value$ String. Valor a ser gravado.

length& Long. O comprimento do campo de sequência em bytes.

Método WriteShort

Este método usa um valor inteiro de 2 bytes assinado e grava-o no buffer de Dados da mensagem como um número binário de 2 bytes iniciando no byte referenciado por DataOffset. Ele substitui quaisquer dados que já estão nessas posições no buffer e estenderá o comprimento do buffer (MQMessage.MessageLength) se necessário.

DataOffset é incrementado em 2 se o método for bem-sucedido.

O método converte para a representação binária especificada pela propriedade MQMessage.Encoding.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteShort(value%)
```

Parameter *value%* Integer. Valor a ser gravado.

Método WriteString

Esse método usa uma sequência de ActiveX e a grava no buffer de Dados da mensagem iniciando no byte referido por DataOffset. Ele substitui quaisquer dados que já estão nessas posições no buffer e estenderá o comprimento do buffer (MQMessage.MessageLength) se necessário.

DataOffset é incrementado pelo comprimento da sequência em bytes se o método for bem-sucedido.

O método converte os caracteres na página de códigos especificada pela propriedade MQMessage.CharacterSet.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteString(value$)
```

Parameter *value$* String. Valor a ser gravado.

Método WriteUInt2

Este método usa um valor de número inteiro de 4 bytes assinados e o grava no buffer Dados da mensagem como um número binário de 2 bytes não assinados, iniciando no byte referido por DataOffset. Ele substitui quaisquer dados que já estejam nessas posições no buffer e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

DataOffset é incrementado em 2 se o método for bem-sucedido.

O método converte para a representação binária especificada pela propriedade MQMessage.Encoding. O valor especificado deve estar no intervalo de 0 a 2**16-1. Se não for, o método retorna com CompletionCode MQCC_FAILED e ReasonCode MQRC_WRITE_VALUE_ERROR.

Definido em: classe MQMessage

Sintaxe:

```
Call MQMessage.WriteUInt2(value&)
```

Parâmetro *valor &* longo. Valor a ser gravado.

Método WriteUnsignedByte

Esse método pega um valor inteiro de 2 bytes assinado e grava no buffer de Dados da mensagem como um número binário não assinado de 1 byte iniciando no caractere referido por DataOffset. Ele substitui quaisquer dados que já estejam nessas posições no buffer e estende o comprimento do buffer (MQMessage.MessageLength), se necessário.

DataOffset é incrementado em 1 se o método for bem-sucedido.

O valor especificado deve estar no intervalo de 0 a 255. Se não for, o método retorna com CompletionCode MQCC_FAILED e ReasonCode MQRC_WRITE_VALUE_ERROR.

Definido em:
Classe MQMessage

Sintaxe:

```
Call MQMessage.WriteUnsignedByte(value%)
```

Parameter *value%* Integer. Valor a ser gravado.

Método WriteUTF

Este método obtém uma sequência ActiveX e a grava no buffer de dados de mensagem na posição atual no formato UTF. Os dados gravados consistem em um comprimento de 2 bytes seguido pelos dados de caracteres. DataOffset será incrementado pelo comprimento da sequência se o método for bem-sucedido.

Definido em:
Classe MQMessage

Sintaxe: Call *MQMessage*. **WriteUTF** (value$)

Parâmetro:
value$ String. Valor a ser gravado.

Classe MQPutMessageOptions

Essa classe encapsula as várias opções que controlam a ação de colocar uma mensagem em uma fila do IBM MQ.

Restrição

A classe MQPutMessageOptions é contida pela classe MQSession.

Criação

New cria um novo objeto MQPutMessageOptions e define todas as suas propriedades para os valores iniciais.

Alternativamente, use o método AccessPutMessageOptions da classe MQSession.

Sintaxe

Dim *pmo* **As New MQPutMessageOptions** ou

Set *pmo* = **New MQPutMessageOptions**

Propriedades

- “propriedade CompletionCode” na página 644.
- “Propriedade Options” na página 645.
- “propriedade ReasonCode” na página 645.
- “Propriedade ReasonName” na página 645.
- “Propriedade RecordFields” na página 645.
- “Propriedade ResolvedQueueManagerName” na página 646.
- “Propriedade ResolvedQueueName” na página 646.

Methods

- “método ClearErrorCodes” na página 646.

propriedade CompletionCode

Somente leitura. Retorna o código de conclusão configurados pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQPutMessageOptions

Tipo de dados: longo

Valores:

- MQCC_OK

- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode & = PutOpts* **.CompletionCode**

Propriedade Options

Leitura/gravação. O campo Opções MQPMO. O valor inicial deste campo é MQPMO_NONE. Para obter mais informações, consulte Opções MQPMO.

Definido em: classe MQPutMessageOptions.

Tipo de dados: longo

Sintaxe: Para obter: *opções & = PutOpts* **Opções**

Para configurar: *PutOpts* **.Opções** = *opções &*

As opções MQPMO_PASS_IDENTITY_CONTEXT e MQPMO_PASS_ALL_CONTEXT não são suportadas.

propriedade ReasonCode

Somente leitura. Retorna o código de razão definido pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQPutMessageOptions

Tipo de dados: longo

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: Obter: *Reasoncode & = PutOpts* **.ReasonCode**

Propriedade ReasonName

Somente leitura. Retorna o nome simbólico do código de razão mais recente. Por exemplo, "MQRC_QMGR_NOT_AVAILABLE".

Definido em: classe MQPutMessageOptions

Tipo de dados: sequência

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasonname$ = PutOpts* **.ReasonName**

Propriedade RecordFields

Leitura/gravação. Sinalizadores que indicam quais campos devem ser customizados em uma base por fila ao colocar uma mensagem em uma lista de distribuição. O valor inicial é zero.

Essa propriedade corresponde às sinalizações PutMsgRecFields na estrutura MQI MQPMO. No MQI, esses sinalizadores controlam quais campos (na estrutura MQPMR) estão presentes e são usados pelo MQPUT. Em um objeto MQPutMessageOptions, esses campos estão sempre presentes, e os sinalizadores, portanto, afetam apenas os campos que são usados pelo Put.

Definido em:
Classe MQPutMessageOptions

Tipo de dados:
Long

Sintaxe: Para obter: *recordfields & = PutOpts* **.RecordFields**

Para configurar: *PutOpts*. **RecordFields** = *recordfields &*

Propriedade ResolvedQueueManagerName

Somente leitura. O campo MQPMO ResolvedQMgrName. Consulte ResolvedQMgrName (MQCHAR48) para obter detalhes. O valor inicial é todos os espaços em branco.

Definido em: classe MQPutMessageOptions

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *qmgr$ = PutOpts* **.ResolvedQueueManagerName**

Propriedade ResolvedQueueName

Somente leitura. O campo MQPMO ResolvedQName. Veja ResolvedQName (MQCHAR48) para obter detalhes. O valor inicial é todos os espaços em branco.

Definido em: classe MQPutMessageOptions

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *qname$ = PutOpts* **.ResolvedQueueName**

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode para a classe MQPutMessageOptions e a classe MQSession.

Definido em:
Classe MQPutMessageOptions

Sintaxe:
Call *PutOpts* **.ClearErrorCodes()**

Classe MQGetMessageOptions

Esta classe encapsula as várias opções que controlam a ação de obter uma mensagem de uma fila do IBM MQ.

Restrição

A classe MQGetMessageOptions está contido pela classe MQSession.

Criação

New cria um novo objeto MQGetMessageOptions e define todas as suas propriedades para valores iniciais.

Como alternativa, use o método AccessGetMessageOptions da classe MQSession.

Propriedades

Methods

- “método ClearErrorCodes” na página 648

Sintaxe

Dim *gmo* **As New MQGetMessageOptions** ou

Set *gmo* = **New MQGetMessageOptions**

propriedade CompletionCode

Somente leitura. Retorna o código de conclusão configurados pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQGetMessageOptions.

Tipo de dados: longo

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode & = GetOpts* **.CompletionCode** .

Propriedade MatchOptions

Leitura/gravação. Opções que controlam os critérios de seleção usados para MQGET. O valor inicial é MQMO_MATCH_MSG_ID MQMO_MATCH_CORREL_ID.

Definido em:
Classe MQGetMessageOptions

Tipo de dados:
Long

Valores:
Veja MatchOptions (MQLONG).

Sintaxe: Para obter: *matchoptions & = GetOpts* **MatchOptions**

Para configurar: *GetOpts*. **MatchOptions** = *matchoptions &*

Propriedade Options

Leitura/gravação. O campo Options de MQGMO. Consulte Opções para obter detalhes. valor inicial é MQGMO_NO_WAIT.

Definido em: classe MQGetMessageOptions.

Tipo de dados: longo

Sintaxe: Para obter: *opções & = GetOpts* **Opções** Para configurar: *GetOpts* **Opções** = *opções &*

propriedade ReasonCode

Somente leitura. Retorna o código de razão definido pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em: classe MQGetMessageOptions

Tipo de dados: longo

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter: *reasoncode & = GetOpts* **.ReasonCode** .

Propriedade ReasonName

Somente leitura. Retorna o nome simbólico do código de razão mais recente. Por exemplo, "MQRC_QMGR_NOT_AVAILABLE". **Definido em:** classe MQGetMessageOptions

Tipo de dados: sequência

Valores:

- Consulte conclusão da API e códigos de razão.

Sintaxe: para obter: *reasonname$ = MQGetMessageOptions* **.ReasonName**

Propriedade ResolvedQueueName

Somente leitura. O campo MQGMO ResolvedQName. Veja ResolvedQName (MQCHAR48) para obter detalhes. O valor inicial é todos os espaços em branco.

Definido em: classe MQGetMessageOptions

Tipo de dados: sequência de 48 caracteres

Sintaxe: para obter: *qname$ = GetOpts* **.ResolvedQueueName**

Propriedade WaitInterval

Leitura/gravação. O campo MQGMO WaitInterval. O tempo máximo, em milissegundos, que o Get aguarda uma mensagem adequada chegar - se a ação de espera foi solicitada pela propriedade Options. Este campo tem um valor inicial de 0. Para obter detalhes de opções MQGMO, consulte MQGMO.

Definido em: classe MQGetMessageOptions

Tipo de dados: longo

Sintaxe: Para obter: *wait & = GetOpts* **.WaitInterval**

Para configurar: *GetOpts* **.WaitInterval** = wait &

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode para a classe MQGetMessageOptions e a classe MQSession.

Definido em:
Classe MQGetMessageOptions

Sintaxe:
Call *GetOpts* **.ClearErrorCodes()**

Classe MQDistributionList

Esta classe encapsula uma coleta de filas – local, remota ou de alias para saída.

Criação

new cria um objeto MQDistributionList novo.

Como alternativa, utilize o método AddDistributionList da classe MQQueueManager

Propriedades

- “Propriedade AlternateUserId” na página 649
- “Propriedade CloseOptions” na página 649
- “propriedade CompletionCode” na página 649

- “Propriedade ConnectionReference” na página 650
- “Propriedade FirstDistributionListItem” na página 650
- “Propriedade IsOpen” na página 650
- “Propriedade OpenOptions” na página 650
- “propriedade ReasonCode” na página 651
- “Propriedade ReasonName” na página 651

Método

- “Método AddDistributionListItem” na página 651
- “método ClearErrorCodes” na página 651
- “Método Close” na página 652
- “Método Open” na página 652
- “Método Put” na página 652

Sintaxe

Dim *distlist.* **A s New MQDistributionList** ou **Set** *distlist* = **New MQDistributionList**

Propriedade AlternateUserId

Leitura/gravação. O ID do usuário alternativo utilizado para validar o acesso à lista de filas quando elas forem abertas.

Definido em:
Classe MQDistributionList

Tipo de dados:
Sequência de 12 caracteres

Sintaxe: Para obter: *altuser$= MQDistributionList*. **AlternateUserId**

Para configurar: *MQDistributionList*. **AlternateUserId** = *altuser$*

Propriedade CloseOptions

Leitura/gravação. As opções usadas para controlar o que acontece quando a lista de distribuição é fechada. O valor inicial é MQCO_NONE.

Definido em:
Classe MQDistributionList

Tipo de dados:
Long

Valores:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

Sintaxe: Para obter: *closeopt & = MQDistributionList* **CloseOptions**

Para configurar: *MQDistributionList*. **CloseOptions** = *closeopt &*

propriedade CompletionCode

Somente leitura. O código de conclusão configurados pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em:
Classe MQDistributionList

Tipo de dados:
Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode & = MQDistributionList* **CompletionCode**

Propriedade ConnectionReference

Leitura/gravação. O gerenciador de filas ao qual a lista de distribuição pertence.

Definido em:
Classe MQDistributionList

Tipo de dados:
MQQueueManager

Sintaxe: Para obter: *set queuemanager = MQDistributionList*. **ConnectionReference**

Para configurar: *set MQDistributionList*. **ConnectionReference** = *queuemanager*

Propriedade FirstDistributionListItem

Somente leitura. O objeto de item da lista de distribuição primeira associado à lista de distribuição.

Definido em:
Classe MQDistributionList

Tipo de dados:
MQDistributionListItem

Valores:

Sintaxe: Para obter: *set distributionlistitem = MQDistributionList*. **FirstDistributionListItem**

Propriedade IsOpen

Somente leitura.

Definido em:
Classe MQDistributionList

Tipo de dados:
Booleana

Valores:

- TRUE (-1)
- FALSE (0)

Sintaxe: Para obter: *IsOpen = MQDistributionList*. **IsOpen**

Propriedade OpenOptions

Leitura/gravação. Opções a serem utilizadas quando a lista de distribuição é aberta.

Definido em:
Classe MQDistributionList

Tipo de dados:
Long

Valores:
Veja opções de MQPMO.

Sintaxe: Para obter: *openopt & = MQDistributionList* **OpenOptions**

Para configurar: *MQDistributionList.* **OpenOptions** = openopt &

propriedade ReasonCode

Somente leitura. O código de razão definido pelo último método ou acesso de propriedade emitidos com relação ao objeto.

Definido em:
Classe MQDistributionList

Tipo de dados:
Long

Valores:
Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter: *reasoncode & = MQDistributionList* **ReasonCode**

Propriedade ReasonName

Somente leitura. O nome simbólico para a ReasonCode. Por exemplo "MQRC_QMGR_NOT_AVAILABLE".

Definido em:
Classe MQDistributionList

Tipo de dados:
Sequência

Valores:
Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter: *reasonname$= MQDistributionList*. **ReasonName**

Método AddDistributionListItem

Use esse método para criar um novo objeto `MQDistributionListItem` e associá-lo ao objeto de lista de distribuição. O parâmetro de nome de fila é obrigatório.

Este método insere um novo item da lista de distribuição como o primeiro item em uma lista existente. Especificamente, esse método cria a configuração a seguir:

- Na lista de distribuição, ele configura a propriedade **`FirstDistributionListItem`** para apontar para o novo item da lista de distribuição.
- No novo item da lista de distribuição, ele configura as propriedades a seguir:
  - Ele configura a propriedade **`DistributionList`** para apontar para a lista de distribuição.
  - Ele configura a propriedade **`PreviousDistributionListItem`** como nula.
  - Ele configura a propriedade **`NextDistributionListItem`** para apontar para o item da lista de distribuição que era primeira anteriormente ou para null se não havia itens anteriores na lista.

Não é possível usar esse método para incluir um novo item quando a lista de distribuição é aberta.

Definido em:
Classe MQDistributionList

Sintaxe: set distributionlistitem = *MQDistributionList* **.AddDistributionListItem** (QName$, QMgrName$)

Parâmetros:

QName$ String. Nome da fila do IBM MQ.

QMgrName$ String. Nome do gerenciador de filas do IBM MQ.

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode para a classe MQDistributionList e a classe MQSession.

Definido em:
Classe MQDistributionList

Sintaxe:

```
Call MQDistributionList.ClearErrorCodes()
```

Método Close

Fecha uma lista de distribuição utilizando o valor atual de opções de fechamento.

Definido em:
Classe MQDistributionList

Sintaxe:
Call *MQDistributionList.* **Close** ()

Método Open

Abre cada uma das filas especificadas pelas propriedades **QueueName** e (quando aplicável) **QueueManagerName** dos itens da lista de distribuição associados ao objeto atual usando o valor atual de AlternateUserId.

Definido em:
Classe MQDistributionList

Sintaxe:

```
Call MQDistributionList.Open()
```

Método Put

Coloca uma mensagem em cada uma das filas identificado pelo itens da lista de distribuição associado com a lista de distribuição.

Definido em:

Classe MQDistributionList

Sintaxe

Chame MQDistributionList. **Put** (Mensagem, PutMsgOpções &)

Parâmetros

Message Objeto MQMessage que representa a mensagem a ser colocada.

PutMsgOptions Objeto MQPutMessageOptions que contém opções para controlar a operação de entrada. Se não for especificado, PutMessageOptions padrão são utilizados.

Este método utiliza um objeto MQMessage como parâmetro. A seguinte lista de distribuição de item propriedades podem ser alteradas como resultado deste método:

- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex

- GroupId
- GroupIdHex
- Feedback
- AccountingToken
- AccountingTokenHex

Classe MQDistributionListItem

Esta classe encapsula a MQOR, MQRR, e MQPMR as estruturas e os associa com uma lista de distribuição de propriedade.

Criação

Use o método AddDistributionListItem da classe MQDistributionList

Propriedades

Methods

- “Propriedade AccountingToken” na página 654.
- “Propriedade AccountingTokenHex” na página 654.
- “propriedade CompletionCode” na página 654.
- “Propriedade CorrelationId” na página 655.
- “Propriedade CorrelationIdHex” na página 655.
- “Propriedade DistributionList” na página 655.
- “Propriedade Feedback” na página 655.
- “Propriedade GroupId” na página 656.
- “Propriedade GroupIdHex” na página 656.
- “Propriedade MessageId” na página 656.
- “propriedade MessageIdHex” na página 656.
- “Propriedade NextDistributionListItem” na página 657.
- “Propriedade PreviousDistributionListItem” na página 657.
- “Propriedade QueueManagerName” na página 657.
- “Propriedade QueueName” na página 657.
- “propriedade ReasonCode” na página 657.
- “Propriedade ReasonName” na página 658.
- “método ClearErrorCodes” na página 658.

Propriedades:

- Propriedade AccountingToken
- Propriedade AccountingTokenHex
- propriedade CompletionCode
- Propriedade CorrelationId
- Propriedade CorrelationIdHex
- Propriedade DistributionList
- Propriedade Feedback
- Propriedade GroupId

- Propriedade GroupIdHex
- Propriedade MessageId
- propriedade MessageIdHex
- Propriedade NextDistributionListItem
- Propriedade PreviousDistributionListItem
- Propriedade QueueManagerName
- Propriedade QueueName
- propriedade ReasonCode
- Propriedade ReasonName

Métodos:

- método ClearErrorCodes

Criação:

Use o método AddDistributionListItem da classe MQDistributionList

Propriedade AccountingToken

Leitura/gravação. O AccountingToken a serem incluídos na MQPMR de uma mensagem quando colocada em uma fila. Seu valor inicial é todos nulos.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 32 caracteres

Sintaxe: Para obter: *accountingtoken MQDistributionListItem = $*. **AccountingToken**

Para configurar: *MQDistributionListItem*. **AccountingToken** = *accountingtoken$*

Propriedade AccountingTokenHex

Leitura/gravação. O AccountingToken a serem incluídos na MQPMR de uma mensagem quando colocada em uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representam o caractere único "A", o par de caracteres "6" e "2" representam o caractere único "B" e assim por diante.

Deve-se fornecer 64 caracteres hexadecimais válidos.

Seu valor inicial é "0...0".

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 64 caracteres hexadecimais reqpresenting 32 caracteres ASCII.

Sintaxe: Para obter: *accountingtokenh MQDistributionListItem = $*. **AccountingTokenHex**

Para configurar: *MQDistributionListItem*. **AccountingTokenHex** = *accountingtokenh$*

propriedade CompletionCode

Somente leitura. O código de conclusão configurados pela última abertura ou pedido put emitidos com relação ao objeto de lista de distribuição de propriedade.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Long

Valores:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Sintaxe: Para obter: *completioncode$ = MQDistributionListItem*. **CompletionCode**

Propriedade CorrelationId

Leitura/gravação. O CorrelId a serem incluídas na MQPMR de uma mensagem quando colocada em uma fila. Seu valor inicial é todos nulos.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 24 caracteres

Sintaxe: Para obter: *correlid MQDistributionListItem = $*. **CorrelationId**

Para configurar: *MQDistributionListItem*. **CorrelationId** = *correlid$*

Propriedade CorrelationIdHex

Leitura/gravação. O CorrelId a serem incluídas na MQPMR de uma mensagem quando colocada em uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representam o caractere único "A", o par de caracteres "6" e "2" representam o caractere único "B" e assim por diante.

Deve-se fornecer 48 caracteres hexadecimais válidos.

Seu valor inicial é "0..0".

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 48 caracteres hexadecimais que representam 24 caracteres ASCII.

Sintaxe: Para obter: *correlidh MQDistributionListItem = $*. **CorrelationIdHex**

Para configurar: *MQDistributionListItem*. **CorrelationIdHex** = *correlidh$*

Propriedade DistributionList

Somente leitura. A lista de distribuição com o qual este item da lista de distribuição está associado.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
MQDistributionList

Sintaxe: Para obter: *set distributionlist = MQDistributionListItem*. **DistributionList**

Propriedade Feedback

Leitura/gravação. O valor de feedback a serem incluídas na MQPMR de uma mensagem quando colocado em uma fila.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Long

Valores:
Veja Feedback (MQLONG).

Sintaxe: Obter: *feedback & = MQDistributionListItem*. **Feedback**

Para configurar: *MQDistributionListItem*. **Feedback** = *feedback &*

Propriedade GroupId

Leitura/gravação. O GroupId a ser incluído na MQPMR de uma mensagem quando colocar em uma fila. Seu valor inicial é todos nulos.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 24 caracteres

Sintaxe: Para obter: *groupid MQDistributionListItem = $*. **ID do Grupo**

Para configurar: *MQDistributionListItem*. **GroupId** = *groupid$*

Propriedade GroupIdHex

Leitura/gravação. O GroupId a ser incluído na MQPMR de uma mensagem quando colocar em uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representam o caractere único "A", o par de caracteres "6" e "2" representam o caractere único "B" e assim por diante.

Deve-se fornecer 48 caracteres hexadecimais válidos.

Seu valor inicial é "0..0".

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 48 caracteres hexadecimais reqpresenting 24 caracteres ASCII.

Sintaxe: Para obter: *groupidh MQDistributionListItem = $*. **GroupIdHex**

Para configurar: *MQDistributionListItem*. **GroupIdHex** = *groupidh$*

Propriedade MessageId

Leitura/gravação. O MessageId a serem incluídas na MQPMR de uma mensagem quando colocado em uma fila. Seu valor inicial é todos nulos.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 24 caracteres

Sintaxe: Para obter: *messageid MQDistributionListItem = $*. **MessageId**

Para configurar: *MQDistributionListItem*. **MessageId** = *messageid$*

propriedade MessageIdHex

Leitura/gravação. O MessageId a serem incluídas na MQPMR de uma mensagem quando colocado em uma fila.

Cada dois caracteres da sequência reprsentam o equivalente hexadecimal de um único caractere ASCII. Por exemplo, o par de caracteres "6" e "1" representam o caractere único "A", o par de caracteres "6" e "2" representam o caractere único "B" e assim por diante.

Deve-se fornecer 48 caracteres hexadecimais válidos.

Seu valor inicial é "0..0".

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 48 caracteres hexadecimais que representam 24 caracteres ASCII.

Sintaxe: Para obter: *messageidh MQDistributionListItem = $*. **MessageIdHex**

Para configurar: *MQDistributionListItem*. **MessageIdHex** = *messageidh$*

Propriedade NextDistributionListItem

Somente leitura. O objeto de item da lista de distribuição próxima associado à lista de distribuição da mesma.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
MQDistributionListItem

Sintaxe: Para obter: *set distributionlistitem = MQDistributionListItem.* **NextDistributionListItem**

Propriedade PreviousDistributionListItem

Somente leitura. O objeto de item da lista de distribuição anterior associado à lista de distribuição da mesma.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
MQDistributionListItem

Sintaxe: Para obter: *set distributionlistitem = MQDistributionListItem*. **PreviousDistributionListItem**

Propriedade QueueManagerName

Leitura/gravação. O nome do gerenciador de filas do IBM MQ.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 48 caracteres.

Sintaxe: Para obter: *qmname MQDistributionListItem = $*. **QueueManagerName**

Para configurar: *MQDistributionListItem*. **QueueManagerName** = *qmname$*

Propriedade QueueName

Leitura/gravação. O nome da fila do IBM MQ.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência de 48 caracteres.

Sintaxe: Para obter: *qname MQDistributionListItem = $*. **QueueName**

Para configurar: *MQDistributionListItem*. **QueueName** = *qname$*

propriedade ReasonCode

Somente leitura. O código de razão definido pelo último abrir ou colocar emitida para o objeto da lista de distribuição de propriedade.

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Long

Valores:
Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter:

```
reasoncode& = MQDistributionListItem.ReasonCode
```

Propriedade ReasonName

Somente leitura. O nome simbólico para a ReasonCode. Por exemplo "MQRC_QMGR_NOT_AVAILABLE".

Definido em:
Classe MQDistributionListItem

Tipo de dados:
Sequência

Valores:
Consulte conclusão da API e códigos de razão.

Sintaxe: Para obter: *reasonname MQDistributionListItem = $*. **ReasonName**

método ClearErrorCodes

Reconfigura o CompletionCode para MQCC_OK e MQRC_NONE para ReasonCode para a classe MQDistributionListItem e a classe MQSession.

Definido em:
Classe MQDistributionListItem

Sintaxe:

```
Call MQDistributionListItem.ClearErrorCodes
```

Rastreando o IBM MQ Automation Classes for ActiveX

Informações sobre o recurso de rastreio fornecido para o IBM MQ Automation Classes for ActiveX, armadilhas comuns e ajuda sobre como evitá-las.

V 9.0.0 No IBM MQ 9.0, o suporte para o MicrosoftActive X foi descontinuado. As classes do IBM MQ para .NET são a tecnologia de substituição recomendada. Para obter informações adicionais, consulte Desenvolvendo aplicativos .NET.

A seção a seguir explica o recurso de rastreio fornecido e detalha armadilhas comuns, com a ajuda sobre como evitá-las:

- “Controlando o rastreio para o IBM MQ Automation Classes for ActiveX” na página 658
- “quando seu script do IBM MQ automation classes for ActiveX falhar” na página 660
- “Códigos de razão para o IBM MQ Automation Classes for ActiveX” na página 660
- “Ferramenta de nível do código” na página 664

Controlando o rastreio para o IBM MQ Automation Classes for ActiveX

O IBM MQ Automation Classes for ActiveX (MQAX) inclui um recurso de rastreio para ajudar a organização de serviço a identificar o que está acontecendo quando você tem um problema.

Ele mostra os caminhos seguidos quando você executa o script MQAX. A menos que tenha um problema, execute com o rastreio desativado para evitar qualquer uso desnecessário de recursos do sistema.

Existem três variáveis de ambiente que você configura para controlar o rastreio:

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

Nota: Especificar *qualquer* valor para a variável **OMQ_TRACE** ativa o recurso de rastreio. Mesmo se você configurar a variável **OMQ_TRACE** para OFF, o rastreio ainda estará ativo. Para desativar o rastreio, não especifique um valor para **OMQ_TRACE**.

1. Clique em **Iniciar**
2. Clique em **Painel de Controle**
3. Clique duas vezes em **Sistema**
4. Clique em **Avançado**
5. Clique em **Ambiente**
6. Na seção intitulada **Variáveis do usuário para (nome do usuário)**, clique em **Novo**
7. Insira o nome da variável e um valor válido nos campos apropriados e clique em **OK**
8. Clique em **OK** para fechar a janela Variáveis de ambiente
9. Clique em **OK** para fechar a janela Propriedades do sistema
10. Feche a janela Painel de Controle

Ao decidir onde você deseja que os arquivos de rastreio sejam gravados, assegure-se de que você tenha autoridade suficiente para gravar e ler a partir do disco.

Com o rastreio ativado, isso desacelera a execução do MQAX, mas não afeta o desempenho de seus ambientes do ActiveX ou do IBM MQ. Quando você não precisar mais de um arquivo de rastreio, será possível excluí-lo.

Você não pode mudar o status da variável OMQ_TRACE enquanto MQAX está em execução.

Nome e diretório do arquivo de rastreio

O nome do arquivo de rastreio assume o formato `OMQ`*nnnnn*`.trc`, em que *nnnnn* é o ID do processo de ActiveX em execução no momento.

Comando:	**Efeito**
SET OMQ_TRACE_PATH = drive:\directory	Configura o diretório de rastreio no qual o arquivo de rastreio é gravado.
SET OMQ_TRACE_PATH =	Remove qualquer configuração existente para o diretório de rastreio. Quando o diretório de rastreio não está configurado, o diretório de trabalho atual (quando ActiveX é iniciado) é usado.
ECHO %OMQ_TRACE_PATH%	Exibe a configuração atual para o diretório de rastreio no Windows.
SET OMQ_TRACE = xxxxxxxx	Ativa o rastreio. Você ativa o rastreio colocando um ou mais caracteres após o sinal de '='. Por exemplo: SET OMQ_TRACE=yes e SET OMQ_TRACE=no. Em ambos os exemplos, o rastreio está ativado Esta configuração só é efetiva para uma janela/sessão única.

Comando:	Efeito
SET OMQ_TRACE=	Desativa o rastreio.
ECHO %OMQ_TRACE%	Exibe o conteúdo da variável de ambiente no Windows.
SET	Exibe o conteúdo de todas as variáveis de ambiente em Windows.
SET OMQ_TRACE_LEVEL = 9	Configura o nível de rastreio para 9 Valores maiores que 9 não produzem informações adicionais no arquivo de rastreio.

quando seu script do IBM MQ automation classes for ActiveX falhar

Se seu script do IBM MQ Automation Classes for ActiveX falha, há várias fontes de informações que podem ser consultadas.

Primeiro relatório de sintoma da falha

Independentemente do recurso de rastreio, para erros inesperados e internos, um relatório de sintoma da Primeira falha pode ser produzido.

Esse relatório está localizado em um arquivo denominado `OMQ`*nnnnn*`.fdc`, em que *nnnnn* é o número do processo ActiveX que está em execução no momento. Localize esse arquivo no diretório ativo a partir do qual você iniciou o ActiveX ou no caminho especificado na variável de ambiente OMQ_PATH.

Outras fontes de informações

IBM MQ fornece diversos logs de erro e informações de rastreio, dependendo da plataforma envolvida. Consulte seu log de eventos do aplicativo Windows.

Códigos de razão para o IBM MQ Automation Classes for ActiveX

Códigos de razão para o IBM MQ Automation Classes for ActiveX (MQAX) que podem ocorrer além de códigos de razão do IBM MQ MQI.

Os códigos de razão a seguir podem ocorrer além daqueles documentados para a MQI do IBM MQ. Para outros códigos, consulte seu log de eventos do aplicativo IBM MQ.

Código de razão	Explanation
MQRC_LIBRARY_LOAD_ERROR (6000)	Uma ou mais das bibliotecas do IBM MQ não puderam ser carregadas. Verifique se todas as bibliotecas do IBM MQ estão no caminho da procura correto no sistema que você está usando. Por exemplo, certifique-se de que os diretórios que contêm as bibliotecas do IBM MQ estejam em PATH.
MQRC_CLASS_LIBRARY_ERROR (6001)	Uma das chamadas do IBM MQ `classlibrary` retornou um valor inesperado de **ReasonCode** ou **CompletionCode**. Verifique o First Failure Symptom Report para obter detalhes. Anote o último método/propriedade e classe usados e informe o Suporte IBM sobre o problema.

Código de razão	Explanation
MQRC_STRING_LENGTH_TOO_BIG (6002)	Foi feita uma tentativa de gravar uma sequência em formato UTF com um comprimento maior que 65.535 bytes no buffer de mensagem.
MQRC_WRITE_VALUE_ERROR (6003)	É usado um valor fora do intervalo; por exemplo, `msg.WriteByte (240)`.
MQRC_PACKED_DECIMAL_ERROR (6004)	Foi feita uma tentativa de ler um número decimal compactado do buffer de mensagem, mas os dados no ponteiro de dados não estão em um formato de dados compactado válido.
MQRC_FLOAT_CONVERSION_ERROR (6005)	Foi feita uma tentativa de ler um número de vírgula flutuante único ou duplo do buffer de mensagem, mas os dados no ponteiro de dados não estão em um formato de vírgula flutuante apropriado.
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	Um objeto aberto não tem as configurações de **OpenOptions** corretas e requer uma ou mais opções adicionais. Uma reabertura implícita é necessária, mas o fechamento foi impedido, porque a fila aberta para entrada e fechamento exclusivos representaria uma janela de oportunidade para outros, potencialmente, para obter acesso à fila. Configure os valores de **OpenOptions** explicitamente para cobrirem todas as eventualidades, de forma que a reabertura explícita não seja necessária.
MQRC_REOPEN_INQUIRE_ERROR (6101)	Um objeto aberto não tem as configurações de **OpenOptions** corretas e requer uma ou mais opções adicionais. Uma reabertura implícita é necessária, mas o fechamento foi evitado porque uma ou mais características do objeto precisam ser verificadas dinamicamente antes do fechamento e os valores de **OpenOptions** ainda não incluem a opção **MQOO_INQUIRE**. Configure os valores de **OpenOptions** explicitamente para incluir a opção **MQOO_INQUIRE**
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	Um objeto aberto não tem as configurações de **OpenOptions** corretas e requer uma ou mais opções adicionais. Uma reabertura implícita é necessária, mas o fechamento foi evitado, porque a fila está aberta com a opção **MQOO_SAVE_ALL_CONTEXT** e uma chamada `Get` destrutiva foi executada anteriormente. Isso fez com que informações de estado retidas fossem associadas à fila aberta e essas informações seriam destruídas pelo fechamento. Configure os valores de **OpenOptions** explicitamente para cobrirem todas as eventualidades, de forma que a reabertura explícita não seja necessária.

Código de razão	Explanation
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	Um objeto aberto não tem as configurações de **OpenOptions** corretas e requer uma ou mais opções adicionais. Uma reabertura implícita é necessária, mas o fechamento foi evitado porque a fila é uma fila local com tipo de definição **MQQDT_TEMPORARY_DYNAMIC**, que seria destruída pelo fechamento. Configure os valores de **OpenOptions** explicitamente para cobrirem todas as eventualidades, de forma que a reabertura explícita não seja necessária.
MQRC_ATTRIBUTE_LOCKED (6104)	Foi feita uma tentativa para mudar o valor ou o atributo de um objeto enquanto o objeto está aberto. Determinados atributos, como **AlternateUserId**, não podem ser mudados enquanto um objeto é aberto.
MQRC_CURSOR_NOT_VALID (6105)	O cursor de procura de uma fila aberta foi invalidado desde que foi usado pela última vez por uma reabertura implícita. Configure os valores de **OpenOptions** explicitamente para cobrirem todas as eventualidades, de forma que a reabertura explícita não seja necessária.
MQRC_ENCODING_ERROR (6106)	A codificação do próximo item de mensagem precisa ser a codificação **MQENC_NATIVE** para leitura.
MQRC_STRUCID_ERROR (6107)	A estrutura do ID do próximo item de mensagem, que é derivada dos 4 caracteres começando no ponteiro de dados, está ausente ou é inconsistente com o tipo de variável em que o item está sendo lido.
MQRC_NULL_POINTER (6108)	Um ponteiro nulo é fornecido quando um ponteiro não nulo for necessário ou implícito. Isso pode ser causado pelo uso de declarações explícitas para objetos do IBM MQ que são usadas a partir do Visual Basic ou do Excel como parâmetros para chamadas. Por exemplo: • No Visual Basic, o `dim msg as Object` funciona corretamente, enquanto o `dim msg as MqMessage` pode não funcionar corretamente. • No Visual Basic, com uma fila definida e configurada, o `dim msg as MqMessageq.put msg` funciona corretamente, enquanto que a partir do Excel esse comando gera uma exceção `MQRC_NULL_POINTER`.
MQRC_NO_CONNECTION_REFERENCE (6109)	O objeto `MQQueue` perdeu sua conexão com o objeto `MQQueueManager`. Isso ocorrerá se o gerenciador de filas estiver desconectado. Exclua o objeto `MQQueue`.
MQRC_NO_BUFFER (6110)	Nenhum buffer está disponível. Para um objeto `MQMessage`, um buffer não pode ser alocado porque há uma inconsistência interna no estado do objeto.

Código de razão	Explanation
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	O comprimento dos dados binários é inconsistente com o comprimento do atributo de destino. Zero é um comprimento correto para todos os atributos. 24 é o comprimento correto de um atributo **CorrelationId** e de um atributo **MessageId**. 32 é um comprimento correto de um atributo **AccountingToken**.
MQRC_BUFFER_NOT_AUTOMATIC (6112)	Um buffer definido pelo usuário e gerenciado não pode ser redimensionado. Como buffers de mensagens são gerenciados pelo sistema, isso indica uma inconsistência interna.
MQRC_INSUFFICIENT_BUFFER (6113)	Não há espaço disponível suficiente no buffer após o ponteiro de dados para acomodar a solicitação. Isso pode ser porque o buffer não pode ser redimensionado.
MQRC_INSUFFICIENT_DATA (6114)	Há dados insuficientes após o ponteiro de dados para acomodar a solicitação de leitura. Reduza o buffer para o tamanho correto e leia os dados novamente.
MQRC_DATA_TRUNCATED (6115)	Dados foram truncados ao serem copiados de um buffer para outro. Isso pode ser porque o buffer de destino não pode ser redimensionado ou porque há um problema direcionando um ou outro buffer ou porque um buffer está sendo reduzido por uma substituição menor.
MQRC_ZERO_LENGTH (6116)	Um comprimento zero é fornecido quando um comprimento positivo for necessário ou implícito.
MQRC_NEGATIVE_LENGTH (6117)	Um comprimento negativo é fornecido quando um comprimento zero ou positivo for necessário.
MQRC_NEGATIVE_OFFSET (6118)	Um deslocamento negativo foi fornecido quando um deslocamento zero ou positivo é necessário.
MQRC_INCONSISTENT_FORMAT (6119)	O formato do próximo item de mensagem é inconsistente com o tipo de variável no qual o item está sendo lido.
MQRC_INCONSISTENT_OBJECT_STATE (6120)	Há uma inconsistência entre esse objeto, que está aberto, e o objeto `MQQueueManager` referido, que não está conectado.
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	A referência de contexto `MQPutMessageOptions` não faz referência a um objeto `MQQueue` válido. O objeto foi destruído anteriormente.
MQRC_CONTEXT_OPEN_ERROR (6122)	A referência de contexto `MQPutMessageOptions` se refere a um objeto `MQQueue` que não pôde ser aberto para estabelecer um contexto. Isso pode ser porque o objeto `MQQueue` tem opções abertas inapropriadas. Inspecione o código de razão do objeto de referência para estabelecer a causa.

Código de razão	Explanation
MQRC_STRUC_LENGTH_ERROR (6123)	O comprimento de uma estrutura de dados interna é inconsistente com seu conteúdo. Para obter um cabeçalho MQRMH, o comprimento é insuficiente para conter os campos fixos e todos os dados de deslocamento.
MQRC_NOT_CONNECTED (6124)	Um método falhou porque uma conexão necessária com um gerenciador de filas não está disponível e uma conexão não pode ser estabelecida implicitamente.
MQRC_NOT_OPEN (6125)	Um método falhou porque um objeto do IBM MQ não está aberto e a abertura não pode ser feita implicitamente.
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	Um MQDistributionList falhou em abrir, porque não há objetos MQDistributionListItem na lista de distribuição. Ação corretiva: inclua pelo menos um objeto MQDistributionListItem na lista de distribuição.
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	Um método falhou porque o objeto está aberto e as opções abertas são inconsistentes com a operação necessária. Ação corretiva: abra o objeto com as opções de abertura apropriadas, em seguida, tente novamente.
MQRC_WRONG_VERSION (6128)	Um método falhou porque um número de versão especificado ou encontrado é incorreto ou não é suportado.

Ferramenta de nível do código

A Equipe de serviço da IBM pode perguntar qual nível de código que você instalou.

Para descobrir isso, execute o programa utilitário 'MQAXLEV'.

No prompt de comandos, mude para o diretório que contém o MQAX200.dll ou inclua o comprimento do caminho completo e insira:

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

em que MQAXLEV.OUT é o nome do arquivo de saída.

Se você não especificar um arquivo de saída, o detalhe será exibido na tela.

Um arquivo de saída de exemplo da ferramenta de nível de código é detalhado no exemplo a seguir:

Arquivo de saída de exemplo da ferramenta de nível de código

```
5639-B43 (C) Copyright IBM Corp. 1996, 2023. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 *****
lib/mqole/mqole.cpp, mqole, p000, p000 L981119       1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212      1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212   1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216        1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212       1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212       1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212      1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212      1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212      1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219        1.11 99/02/18 12:12:59
```

Interface do ActiveX para a MQAI

Para obter uma visão geral breve de interfaces COM e seu uso na MQAI, consulte “Usando a Interface de modelo de objeto do componente (IBM MQ Automation Classes for ActiveX)” na página 583.

A MQAI permite que os aplicativos construam e enviem comandos Programmable Command Format (PCF) sem obter e formatar diretamente os buffers de comprimento variável necessários para PCF. Para obter mais informações sobre MQAI, veja O IBM MQ Administration Interface (MQAI). A classe MQAI ActiveX MQBag contém os pacotes de dados suportados pela MQAI de maneira que seja possível usar em qualquer linguagem que suporte a criação de objetos COM; por exemplo, Visual Basic, C++, Java e outros clientes de script ActiveX.

A interface MQAI ActiveX é para uso com as classes MQAX que fornecem uma interface COM para a MQI. Para obter mais informações sobre as classes MQAX, consulte “Projetando aplicativos MQAX que acessam aplicativos não ActiveX” na página 584.

A interface ActiveX fornece uma única classe chamada MQBag. Essa classe é usada para criar pacotes de dados MQAI e suas propriedades e métodos são usados para criar e trabalhar com itens de dados dentro de cada pacote. O método MQBag Execute envia os dados de pacote para um gerenciador de filas do IBM MQ como uma mensagem PCF e coleta as respostas.

Para obter mais informações sobre a classe MQBag, suas propriedades e seus métodos, consulte “A classe MQBag” na página 665.

A mensagem PCF é enviada para o objeto de gerenciador de filas especificado, opcionalmente usando filas de resposta e solicitação especificadas. As respostas são retornadas em um novo objeto MQBag. O conjunto completo de comandos e respostas é descrito em Definições dos Programmable Command Formats. Os comandos podem ser enviados para qualquer gerenciador de filas na rede do IBM MQ selecionando as filas de resposta e solicitação apropriadas.

A classe MQBag

A classe MQBag é usada para criar objetos MQBag, conforme necessário. Quando instanciada, a classe MQBag retornará uma nova referência de objeto MQBag.

Crie um objeto MQBag em Visual Basic como a seguir:

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

Propriedade MQBag

As propriedades de objetos MQBag são explicadas na lista a seguir:

- “Propriedade Item” na página 666.

- “Propriedade Count” na página 667.
- “Propriedade Options” na página 668.

Métodos MQBag

Os métodos dos objetos MQBag são explicados na lista a seguir:

- “Método Add” na página 668.
- “Método AddInquiry” na página 669.
- “Método Clear” na página 669.
- “Método Execute” na página 670.
- “Método FromMessage” na página 670.
- “Método ItemType” na página 671.
- “Método Remove” na página 671.
- “Método Selector” na página 672.
- “Método ToMessage” na página 673.
- “Método Truncate” na página 673.

Manipulação de Erros

Se um erro for detectado durante uma operação em um objeto MQBag, incluindo os erros retornados para o pacote por um objeto subjacente MQAX ou MQAI, uma exceção de erro será levantada. A classe MQBag suporta a interface COM ISupportErrorInfo de forma que as informações a seguir estejam disponíveis para sua rotina de manipulação de erros:

- Número do erro: composto do código de razão do IBM MQ para o erro detectado e um código de recurso COM. O campo recurso, como padrão para COM, indica a área de responsabilidade para o erro. Para erros detectados pelo IBM MQ é sempre FACILITY_ITF.
- Erro de origem: identifica o tipo e a versão do objeto que detectou o erro. Para erros detectados durante as operações de MQBag, a origem do erro é sempre MQBag.MQBag1.
- Descrição do erro: a sequência que fornece o nome simbólico para o código de razão do IBM MQ.

Como acessar as informações de erros depende de sua linguagem de script; por exemplo, no Visual Basic, as informações são retornadas no objeto Err e o código de razão é IBM MQ obtida subtraindo o vbObjectError constante a partir de Err.Number.

ReasonCode = Err.Number - vbObjectError

Se o método MQBag Execute enviar uma mensagem de PCF e uma resposta for recebida, a operação será considerada bem-sucedida embora o comando enviado possa ter falhado. Neste caso, o próprio pacote de resposta contém os códigos de razão de conclusão e erro, conforme descrito em Definições dos Programmable Command Formats.

Propriedade Item

Finalidade

A propriedade Item representa um item em um pacote. Ela é usada para configurar ou consultar sobre o valor de um item. O uso dessa propriedade corresponde às seguintes chamadas MQAI:

- "mqSetString"
- "mqSetInteger"
- "mqInquireInteger"
- "mqInquireString"
- "mqInquireBag"

no Referência de formatos de comando programáveis.

Formato

Item (Selector, ItemIndex, Value)

Parâmetros

Seletor (VARIANT) - entrada
Seletor do item a ser configurado ou consultado.

Ao consultar sobre um item, MQSEL_ANY_USER_SELECTOR será o padrão. Ao configurar um item, MQIA_LIST ou MQCA_LIST serão o padrão.

Se `Selector` não for do tipo longo, resultará em MQRC_SELECTOR_TYPE_ERROR.

Esse parâmetro é opcional.

ItemIndex (LONG) - entrada
Este valor identifica a ocorrência do item do seletor especificado que deve ser configurado ou consultado. MQIND_NONE é o padrão.

Esse parâmetro é opcional.

Valor (VARIANT)-entrada/saída
O valor retornado ou o valor a ser configurado. Ao consultar sobre um item, o valor de retorno pode ser de tipo longo, sequência ou MQBag. No entanto, ao configurar um item, o valor deve ser do tipo longo ou sequência, caso contrário resulta em MQRC_ITEM_VALUE_ERROR.

Chamada de linguagem Visual Basic

Ao consultar sobre um valor de um item em um pacote:

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

Para referências MQBag:

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

Para configurar o valor de um item em um pacote:

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

Propriedade Count

Finalidade

A propriedade Count representa o número de itens de dados dentro de um pacote. Esta propriedade corresponde à chamada MQAI, "mqCountItems," no Referência de formatos de comando programáveis.

Formato

Contagem (*`Selector, Value`*)

Parâmetros

Seletor (VARIANT) - entrada
Seletor dos itens de dados a serem incluídos na contagem.

MQSEL_ALL_USER_SELECTORS é o padrão.

Se `Selector` não for do tipo longo, MQRC_SELECTOR_TYPE_ERROR será retornado.

Valor (LONG)-saída
O número de itens no pacote incluído por *`Selector`*.

Chamada de linguagem Visual Basic

Para retornar o número de itens em um pacote:

```
ItemCount = mqbag.Count([Selector])
```

Propriedade Options

Finalidade

A propriedade Options define as opções para o uso de um pacote. Esta propriedade corresponde ao parâmetro **`Options`** da chamada MQAI, "mqCreateBag," no Referência de formatos de comando programáveis.

Formato

Opções (*`Options`*)

Parâmetros

Opções (LONG)-entrada/saída
As opções de pacote.

Nota: As opções de pacote devem ser configuradas ***antes*** de os itens de dados serem incluídos ou definidos dentro do pacote. Se as opções forem mudadas quando o pacote não estiver vazio, MQRC_OPTIONS_ERROR será o resultado. Isso se aplica mesmo se o pacote for subsequentemente removido.

Chamada de linguagem Visual Basic

Ao consultar sobre as opções de um item em um pacote:

```
Options = mqbag.Options
```

Para configurar uma opção de um item em um pacote:

```
mqbag.Options = Options
```

Métodos MQBag

Os métodos dos objetos MQBag são explicados nas seguintes páginas.

Método Add

Finalidade

O método Add inclui um item de dados em um pacote. Este método corresponde às chamadas MQAI, “mqAddInteger” e "mqAddString," no Referência de formatos de comando programáveis.

Formato

Incluir (*Value, Selector*)

Parâmetros

Valor (VARIANT)-entrada
Valor de número inteiro ou sequência do item de dados.

Seletor (VARIANT) - entrada
O seletor que identifica o item a ser incluído.

Dependendo do tipo de `Value`, MQIA_LIST ou MQCA_LIST é o padrão. Se o parâmetro **`Selector`** não for do tipo longo, MQRC_SELECTOR_TYPE_ERROR será o resultado.

Chamada de linguagem Visual Basic

Para incluir um item em um pacote:

```
mqbag.Add(Value,[Selector])
```

Método AddInquiry

Finalidade

O método AddInquiry inclui um seletor que especifica o atributo a ser retornado quando um pacote de administração é enviado para executar um comando INQUIRE. Este método corresponde à chamada MQAI, "mqAddInquiry,", no Referência de formatos de comando programáveis.

Formato

AddInquiry *(Inquiry*)

Parâmetros

Consulta (LONG)-entrada
O seletor do atributo IBM MQ a ser retornado pelo comando de administração INQUIRE.

Chamada de linguagem Visual Basic

Para usar o método AddInquiry:

```
mqbag.AddInquiry(Inquiry)
```

Método Clear

Finalidade

O método Limpar exclui todos os itens de dados de um pacote. Esse método corresponde à chamada MQAI, "mqClearBag,", no Referência de formatos de comando programáveis.

Formato

Remover

Chamada de linguagem Visual Basic

Para excluir todos os itens de dados de um pacote:

```
mqbag.Clear
```

Método Execute

Finalidade

O método Execute envia uma mensagem de comando de administração para o servidor de comandos e aguarda por quaisquer mensagens de resposta. Este método corresponde à chamada MQAI, "mqExecute,", no Referência de formatos de comando programáveis.

Formato

Execute *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag*)

Parâmetros

QueueManager (MQQueueManager) - entrada
O gerenciador de filas ao qual o aplicativo está conectado.

Comando (LONG)-entrada
O comando a ser executado.

OptionsBag (MQBag)-entrada
O pacote que contém opções que afetam o processamento da chamada.

RequestQ (MQQueue)-entrada
A fila na qual a mensagem de comando de administração será colocada.

ReplyQ (MQQueue)-entrada
A fila na qual todas as mensagens de resposta são recebidas.

ReplyBag (MQBag)-saída
Uma referência de pacote que contém dados de mensagens de resposta.

Chamada de linguagem Visual Basic

Para enviar uma mensagem de comando de administração e aguardar a quaisquer mensagens de resposta:

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

Método FromMessage

Finalidade

O método FromMessage carrega dados a partir de uma mensagem em um pacote. Este método corresponde à chamada MQAI, "mqBufferToBag," no Referência de formatos de comando programáveis.

Formato

FromMessage *(Message, OptionsBag*)

Parâmetros

Mensagem (MQMessage)-entrada
A mensagem que contém os dados a serem convertidos.

OptionsBag (MQBag)-entrada
Opções para controlar o processamento da chamada.

Chamada de linguagem Visual Basic

Para carregar dados de uma mensagem para um pacote:

```
mqbag.FromMessage(Message,[OptionsBag])
```

Método ItemType

Finalidade

O método ItemType retorna o tipo do valor em um item especificado em um pacote. Este método corresponde à chamada MQAI, "mqInquireItemInfo," no Referência de formatos de comando programáveis.

Formato

ItemType (*Selector, ItemIndex, ItemType*)

Parâmetros

Seletor (VARIANT) - entrada
Seletor identificando o item a ser consultado.

MQSEL_ANY_USER_SELECTOR é o padrão. Se o parâmetro **Selector** não for do tipo longo, MQRC_SELECTOR_TYPE_ERROR será o resultado.

ItemIndex (LONG) - entrada
Índice de itens a serem consultados.

MQIND_NONE é o padrão.

ItemType (LONG)-saída
Tipo de dados do item especificado.

Nota: Deve-se especificar o parâmetro **Selector**, o parâmetro **ItemIndex** ou ambos. Se nenhum parâmetro estiver presente, resulta em MQRC_PARAMETER_MISSING.

Chamada de linguagem Visual Basic

Para retornar o tipo de um valor:

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

Método Remove

Finalidade

O método remove exclui um item de um pacote. Este método corresponde à chamada MQAI, "mqDeleteItem," no Referência de formatos de comando programáveis.

Formato

Remover *(Selector, ItemIndex)*

Parâmetros

Seletor (VARIANT) - entrada
Seletor identificando o item a ser excluído.

MQSEL_ANY_USER_SELECTOR é o padrão. Se o parâmetro **Selector** não for do tipo longo, MQRC_SELECTOR_TYPE_ERROR será o resultado.

ItemIndex (LONG) - entrada
Índice do item a ser excluído.

MQIND_NONE é o padrão.

Nota: Deve-se especificar o parâmetro **Selector**, o parâmetro **ItemIndex** ou ambos. Se nenhum parâmetro estiver presente, resulta em MQRC_PARAMETER_MISSING.

Chamada de linguagem Visual Basic

Para excluir um item de um pacote:

```
mqbag.Remove([Selector],[ItemIndex])
```

Método Selector

Finalidade

O método Selector retorna o seletor de um item especificado dentro de um pacote. Este método corresponde à chamada MQAI, "mqInquireItemInfo," no Referência de formatos de comando programáveis.

Formato

Seletor *(Selector, ItemIndex, OutSelector)*

Parâmetros

Seletor (VARIANT) - entrada
Seletor identificando o item a ser consultado.

MQSEL_ANY_USER_SELECTOR é o padrão. Se o parâmetro **Selector** não for do tipo longo, MQRC_SELECTOR_TYPE_ERROR será o resultado.

ItemIndex (LONG) - entrada
Índice do item a ser consultado.

MQIND_NONE é o padrão.

OutSelector (VARIANT)-saída
Seletor do item especificado.

Nota: Deve-se especificar o parâmetro **Selector**, o parâmetro **ItemIndex** ou ambos. Se nenhum parâmetro estiver presente, resulta em MQRC_PARAMETER_MISSING.

Chamada de linguagem Visual Basic

Para retornar o seletor de um item:

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

Método ToMessage

Finalidade

O método ToMessage retorna uma referência a um objeto MQMessage.. A referência contém dados de um pacote. Este método corresponde à chamada MQAI, "mqBagToBuffer", no Referência de formatos de comando programáveis.

Formato

ToMessage *(OptionsBag, Message)*

Parâmetros

OptionsBag (MQBag)-entrada
Um pacote que contém opções que controlam o processamento do método

Mensagem (MQMessage)-saída
Uma referência do objeto MQMessage que contém dados do pacote.

Chamada do Visual Basic Language

Para usar o método ToMessage :

```
Set Message = mqbag.ToMessage([OptionsBag])
```

Método Truncate

Finalidade

O método Truncar reduz o número de itens do usuário em um pacote. Este método corresponde à chamada MQAI, "mqTruncateBag," no Referência de formatos de comando programáveis.

Formato

Truncar *(ItemCount)*

Parâmetros

ItemCount (LONG) - entrada
O número de itens do usuário permanece no pacote após o truncamento ocorrer.

Chamada de linguagem Visual Basic

Para reduzir o número de itens do usuário em um pacote:

```
mqbag.Truncate(ItemCount)
```

Sobre as amostras IBM MQ Automation Classes for ActiveX Starter

Este apêndice descreve as amostras do IBM MQ Automation Classes for ActiveX Starter e explica como usá-las.

O IBM MQ for Windows fornece os seguintes programas de amostra do Visual Basic:

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

Essas amostras são executadas no Visual Basic 4 ou no Visual Basic 5. Você os encontrará no diretório ... \tools\mqax\samples\vb.

No mesmo diretório, também podem ser encontradas amostras de Microsoft Excel e html. São elas:

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

Nota: Se estiver usando o Visual Basic 5, **deve-se** selecionar e instalar o Visual Basic do componente grid32.ocx.

O que é demonstrado nas amostras

As amostras demonstram como usar o IBM MQ Automation Classes for ActiveX para:

- Conecta-se a um gerenciador de filas
- Acessar uma fila
- Colocar uma mensagem em uma fila
- Obter uma mensagem de uma fila

A parte central da amostra do Visual Basic é mostrada nas páginas a seguir.

"Preparando para executar as amostras" na página 675 e

"Manipulação de erros nas amostras" na página 675

Executando as amostras do ActiveX Starter

Antes de executar as amostras do IBM MQ Automation Classes for ActiveX Starter, verifique se há um gerenciador de filas padrão em execução e se as definições de fila necessárias foram criadas. Para obter detalhes de criação e execução de um gerenciador de filas e criar uma fila, consulte Administrando. A amostra usa a fila SYSTEM.DEFAULT.LOCAL.QUEUE , que deve ser definido em qualquer servidor IBM MQ normalmente configurado

As diferentes maneiras de usar pacotes de dados são conforme o mostrado na lista a seguir:

- Conecta-se a um gerenciador de filas
- Acessar uma fila
- Colocar uma mensagem em uma fila
- Obter uma mensagem de uma fila

Para obter informações sobre as amostras do iniciador MQAX para Microsoft Basic 4 ou mais recente, consulte "Executando a amostra MQAXTRIV" na página 675

Para obter informações sobre uma amostra que permite procurar propriedades e métodos de gerenciadores de filas e objetos de fila, consulte "Iniciando a amostra MQAXCLSS" na página 677

Para obter informações sobre a amostra MQAXDLST, "A amostra MQAXDLST" na página 677

Para obter informações sobre como executar o MQAX starter sample for Microsoft Excel 95 ou posterior, MQAXTRIV.XLS, consulte "Executando a amostra MQAXTRIV.XLS" na página 677.

Para obter informações adicionais sobre como executar a demonstração de Banco com o MQAX.XLS, consulte "Executando a demonstração do Banco com MQAX.XLS" na página 677

Para obter informações sobre amostras iniciais usando um navegador WWW compatível com ActiveX, consulte “Amostra do Starter usando um navegador WWW compatível com ActiveX” na página 677

Preparando para executar as amostras

Para executar qualquer uma das amostras, é necessário um dos seguintes, dependendo de qual das amostras que você pretende executar.

- Microsoft Visual Basic 4 (ou posterior)
- Microsoft Excel 95 (ou mais recente)
- Um navegador da web

Também é necessário:

- Um gerenciador de filas do IBM MQ em execução.
- Uma fila do IBM MQ já definida.

Manipulação de erros nas amostras

A maioria das amostras fornecidas no pacote IBM MQ Automation Classes for ActiveX exibe pouca ou nenhuma manipulação de erros. Para obter mais informações sobre a manipulação de erros, consulte “Manipulação de Erros” na página 588.

Executando a amostra MQAXTRIV

1. Inicie o gerenciador de filas.
2. No Windows Explorer ou File Manager, selecione o ícone para a amostra, MQAXTRIV.VBP (arquivo do Visual Basic Project) e abra o arquivo.

   O programa Visual Basic é iniciado e abre o arquivo, MQAXTRIV.VBP.
3. No Visual Basic, pressione a tecla de função 5 (F5) para executar a amostra.
4. Clique em qualquer lugar no formulário da janela, **Testador trivial MQAX**.

Se tudo estiver funcionando corretamente, o plano de fundo da janela deverá ser mudado para verde. Se houver um problema com sua configuração, o plano de fundo da janela deverá ser mudado para vermelho e as informações sobre o erro serão exibidas.

A figura a seguir mostra a parte central da amostra do Visual Basic.

```
Option Explicit

Private Sub Form_Click()

'*****************************************************************************
'* This simple example illustrates how to put and get an IBM MQ message to
'* and from an IBM MQ message queue. The data from the message returned by the
'* get is read and compared with that from the original message.
'*****************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* put message options
Dim GetOptions As MQGetMessageOptions   '* get message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*****************************************************************************
'* Handle errors
'*****************************************************************************
On Error GoTo HandleError

'*****************************************************************************
'* Initialize the current position for the form
'*****************************************************************************
```

```
CurrentX = 0
CurrentY = 0

'*****************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*****************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                        MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*****************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*****************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions

'*****************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*****************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*****************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*****************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127) '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*****************************************************************************
'* Handle errors
'*****************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "IBM MQ Completion Code = " & MQSess.CompletionCode
    Print "IBM MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
Else
    Print "Visual Basic error: " & Err
```

```
    Print Error(Err)
End If

Exit Sub

End Sub
```

Iniciando a amostra MQAXCLSS

Esta amostra permite navegar pelas propriedades e métodos de gerenciadores de filas e objetos de fila.

1. Inicie o gerenciador de filas.
2. Abra o arquivo, MQAXCLSS.VBP, clicando duas vezes no ícone de documento no Windows Explorer ou clicando em Arquivo - Abrir no menu de arquivo no Visual Basic.
3. Inicie a amostra.
4. Insira o gerenciador de filas apropriado e os nomes de filas e, em seguida, clique nos botões correspondentes.

A amostra MQAXDLST

A amostra em Visual Basic MQAXDLST demonstra o uso de uma lista de distribuição para enviar a mesma mensagem para duas filas com um put. Para executar a amostra, faça o mesmo que para a amostra MQAXCLSS.

Amostra do MQAX Starter para Microsoft Excel 95 ou posterior

Esta seção explica como executar o MQAX starter sample for Microsoft Excel 95 ou posterior, MQAXTRIV.XLS.

Executando a amostra MQAXTRIV.XLS

1. Inicie o gerenciador de filas.
2. No Explorer ou File Manager, selecione o ícone para a amostra MQAX MQAXTRIV.XLS.
3. Clique no botão na planilha.
4. A tela será atualizada com uma mensagem de sucesso (ou de falha).

Executando a demonstração do Banco com MQAX.XLS

Siga estas etapas para executar a demonstração do Banco.

1. Inicie o gerenciador de filas.
2. Execute o arquivo de comandos MQSC do IBM MQ, BANK.TST. Isso configura as definições de filas necessárias do IBM MQ.

   Para descobrir como usar um arquivo de comandos MQSC, consulte Comandos de script (MQSC).
3. Execute MQAXBSRV.VBP. Este programa de amostra é o servidor simulando um aplicativo backend e precisa ser executado com o Microsoft Excel.
4. Execute MQAX.XLS. Esta amostra é a demonstração do IBM MQ cliente.
5. Selecione um cliente na lista.
6. Clique em **Enviar**.

Após uma pausa curta (aproximadamente 3 segundos), os campos serão preenchidos com valores e um gráfico de barras será exibido.

Amostra do Starter usando um navegador WWW compatível com ActiveX

Nota: Para executar essa amostra, deve-se estar executando um navegador da web compatível com ActiveX. O Microsoft Internet Explorer (mas não o Netscape Navigator) é um navegador da web compatível.

Executando a amostra HTML

Esta amostra demonstra como é possível chamar MQAX a partir de ambos os VBScript e JavaScript.

1. Inicie o gerenciador de filas.
2. Abra o arquivo, "MQAXTRIV.HTM", em seu navegador da web compatível com ActiveX.

   É possível fazer isso clicando duas vezes no ícone do arquivo no Windows Explorer ou escolher Arquivo - Abrir no menu Arquivo de seu navegador da web compatível com ActiveX.
3. Siga as instruções na tela.

ULW V 9.0.0 Desenvolvendo aplicativos clientes AMQP

O suporte do IBM MQ para APIs AMQP, incluindo a API do MQ Light, permite que um administrador do IBM MQ crie um canal 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 no UNIX, no Linux ou no Windows; ele não está disponível no IBM i ou no z/OS.

A API do MQ Light é baseado no protocolo Oasis AMQP 1.0. Há APIs do sistema de mensagens para Node.js, Java, Ruby e Python.

Um aplicativo desenvolvido para usar a API do MQ Light pode ser conectado a um tempo de execução do MQ Light, a um gerenciador de filas do IBM MQ com um canal AMQP ou a uma instância de um serviço do MQ Light no IBM Cloud (formerly Bluemix).

Desenvolvendo clientes AMQP

A API MQ Light visa fazer com que seja mais fácil para o protótipo e desenvolver aplicativos de negócios rapidamente. Há APIs MQ Light para Node.js, Java, Ruby e Python, disponíveis em https://github.com/mqlight.

Fazendo Download de Clientes AMQP de Amostra

O IBM MQ não envia clientes MQ Light, mas é possível fazer download e instalar os clientes MQ Light a seguir:

Node.js
Instale a API Node.js do MQ Light em seu diretório ativo usando o npm: `npm install mqlight@1.0`

Java
Faça o download do pacote mqlight-distribution para a versão requerida da Maven Central e extraia o conteúdo. É possível localizar as versões disponíveis dos pacotes mqlight-distribution no Maven Central.

Rubi
Instale a API Ruby do MQ Light em seu diretório ativo usando o gem: `gem install mqlight --pre`

Python
Instale a API Python do MQ Light em seu diretório ativo usando o pip: `pip install mqlight --pre`

O cliente do MQ Light faz download de tudo, incluindo várias amostras, que demonstram os recursos diferentes de sistema de mensagens:

- Enviar amostra

- Receber amostra
- Amostra de exercício da UI

Também é possível fazer o download de outros clientes AMQP de software livre com base nas bibliotecas Apache Qpid. Para obter mais informações, consulte https://qpid.apache.org/index.html

Protegendo clientes AMQP

Para obter informações sobre como proteger aplicativos MQ Light , consulte Protegendo clientes AMQP.

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. É possível implementar aplicativos MQ Light em um gerenciador de filas do IBM MQ. Consulte “Implementando os aplicativos do MQ Light em um ambiente local do IBM MQ” na página 694 .

Ao implementar um cliente AMQP, é possível trocar mensagens com aplicativos IBM MQ. Por exemplo, se você usar o cliente MQ Light Node.js para enviar uma mensagem de sequência do JavaScript, o aplicativo IBM MQ receberá uma mensagem do MQ, em que o campo de formato 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 for iniciado, será possível testá-lo conectando um aplicativo MQ Light usando qualquer um dos métodos a seguir:

- Usando o cliente IBM MQ Light para Node.js e o Java.
- Usando o cliente IBM MQ Light para Ruby e Python.
- Usando outro cliente AMQP 1.0. Por exemplo, Apache Qpid Proton.

Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW MQ Light e AMQP (Advanced Message Queuing Protocol)

A API IBM MQ Light é baseada no protocolo de ligação do 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.

A API do sistema de mensagens MQ Light baseia-se no AMQP 1.0. A API fornece a maioria dos recursos do sistema de mensagens necessários para a maioria dos fluxos de publicação/assinatura e de mensagens ponto-a-ponto.

A API do MQ Light tem 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 MQ Light, consulte os websites a seguir:

- Para obter a documentação da API do Node.js, consulte https://www.npmjs.org/package/mqlight
- Para obter a documentação da API do Ruby, consulte https://www.rubydoc.info/github/mqlight/ruby-mqlight
- Para obter a documentação da API do Python, consulte https://python-mqlight.readthedocs.org
- Para a documentação da API Java, consulte https://mqlight.github.io/java-mqlight

Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW 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 os clientes do MQ Light ou outros clientes compatíveis com o AMQP 1.0 a um canal AMQP do IBM MQ. 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:

Nomes de links

Os canais do AMQP esperam que o nome de um link AMQP siga um dos três formatos a seguir:

- Um tópico simples (para publicação e assinatura)
  - Mensagens de publicação: 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`.
  - Assinar um tópico para receber mensagens: uma sequência de tópicos simples (por exemplo, um nome de link de `"/sports/football"` faz com que a 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` se diferencia de uma assinatura específica de um cliente AMQP em particular de uma assinatura compartilhada entre clientes.
- Um tópico detalhado compartilhado (para assinatura)

– Uma sequência de tópicos detalhada que descreve uma assinatura compartilhada no formulário: "share:*share name*:*topic string*" (por exemplo: "share:bbc:/sports/football").

Para obter mais informações sobre a maneira que as mensagens AMQP são mapeadas e de mensagens do IBM MQ, consulte Mapeando campos do AMQP em campos do IBM MQ (mensagens recebidas).

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 um ID de contêiner de um performativo aberto do AMQP contenha um identificador de cliente do MQ Light. O comprimento máximo de um identificador de cliente do MQ Light é 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 somente um estado de entrega para quadros de disposição de Aceito.

Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW V 9.0.0 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:

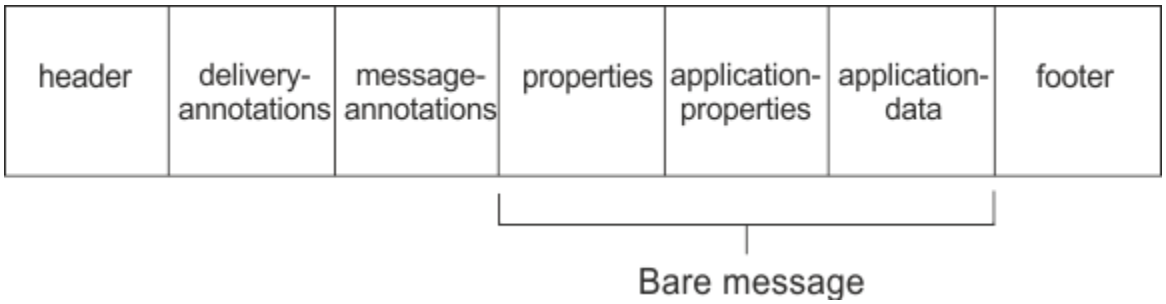
- **ID de mensagem** - identificador de mensagem do aplicativo
- **user-id** - ID do usuário da criação
- **para** - endereço do nó para o qual a mensagem é destinada
- **assunto** - o assunto da mensagem
- **responder para** - o nó para o qual a resposta é enviada
- **ID de correlação** - identificador 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.

Informações relacionadas
Criando e utilizando canais AMQP
Protegendo clientes AMQP

ULW V 9.0.0 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 80. 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 81. Mapeamentos de campo propriedades

Nome	Value
message-id	O `MQMD.MsgId` é configurado como binário.

Tabela 81. Mapeamentos de campo propriedades (continuação)	
Nome	**Value**
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.

Informações relacionadas

MQMD - descritor de mensagem
Criando e utilizando canais AMQP
Protegendo clientes AMQP

ULW V 9.0.0 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 mensagem

Tabela 82. 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
Encoding	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/.
BackoutCount	0
ReplyToQ	""
ReplyToQMgr	""
UserIdentifier	Configure como o identificador do usuário autenticado que conectou ao canal AMQP
AccountingToken	MQACT_NONE

Tabela 82. Descritor de mensagens para a mensagem AMQP (continuação)	
Campo	**Value**
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 de 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 83.			
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.
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.

Tabela 83. (continuação)

Nome da Propriedade	Nome do MQRFH2	Tipo	Descrição
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.
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.

Tabela 83. (continuação)

Nome da Propriedade	Nome do MQRFH2	Tipo	Descrição
AMQPContentEncoding	mq_amqp.Prp.Cne	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 pela Especificação de linguagem Java para identificadores Java.
  - 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.

carga útil

- Para um AMQP bodycom 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 bodycom 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.

Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW V 9.0.0 Confiabilidade de entrega de mensagens com AMQP

Há quatro recursos da API (interface de programação de aplicativos) do IBM MQ Light que permitem controlar a confiabilidade da entrega de mensagens de e para os aplicativos MQ Light e AMQP.

São elas:

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 o cliente ou assine a opção **qos** como *QOS_AT_MOST_ONCE* ou *QOS_AT_LEAST_ONCE*.

Se você estiver usando um cliente AMQP diferente, configure o atributo **`settled`** do quadro de transferência (para publicadores) ou do quadro de disposição (para assinantes) como *true* ou *false*, dependendo da qualidade de serviço que você deseja atingir.

A qualidade de serviço determina quando uma mensagem é descartada do lado `sending` de uma conversa.

Publicação

Se um publicador escolher **QOS** *0* (no máximo uma vez), ele 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 uma confirmação 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 uma confirmação do assinante antes de descartar sua cópia da mensagem.

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 de o gerenciador de filas descartar 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 opção de assinatura **ttl** do MQ Light é usada para especificar o tempo (em milissegundos) que uma assinatura continuará a existir após a desconexão do aplicativo. 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 AMQP usar **QOS** *0* (no máximo uma vez) e criar uma assinatura não durável, o canal 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 AMQP configurar o cabeçalho **durable** AMQP como *true*, o canal 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 *NO* (não persistente).

Atenção: As versões mais recentes do cliente MQ Light não suportam a configuração do cabeçalho durável do AMQP.

Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW 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.

Informações relacionadas

Criando e utilizando canais AMQP

ULW Clientes AMQP comunicando-se por meio do IBM MQ

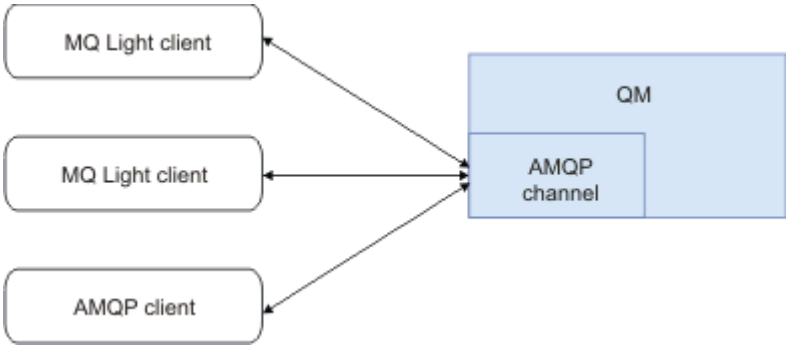
É possível usar o IBM MQ como o provedor de sistema de mensagens do IBM MQ Light ou de qualquer outro aplicativo que esteja em conformidade com o AMQP 1.0. Embora qualquer cliente AMQP 1,0 pode se conectar a um canal AMQP, alguns recursos AMQP não são suportados, por exemplo transações ou múltiplas sessões.

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 seguinte cenário, apenas as mensagens de envio e recebidas dos aplicativos são aplicativos do MQ Light ou 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.



Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW Clientes AMQP trocando mensagens com aplicativos IBM MQ

Ao definir e iniciar um canal AMQP, MQ Light ou aplicativos AMQP 1.0 podem publicar mensagens que são recebidas pelos aplicativos do 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.

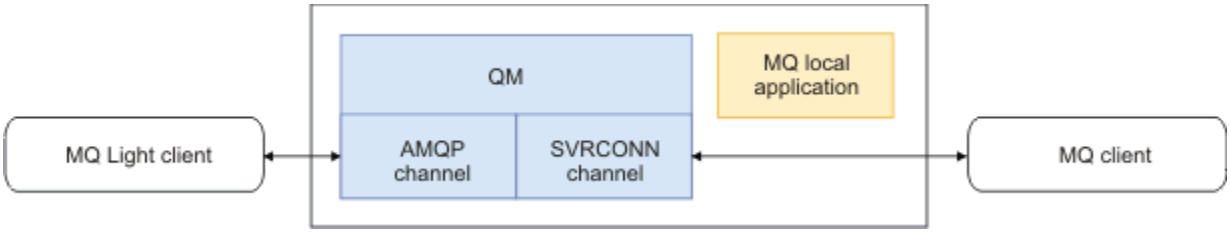
Dados da mensagem AMQP, atributos e propriedades são configurados na mensagem MQ recebida pelo aplicativo MQ. Para obter mais informações sobre os mapeamentos de mensagem do AMQP para MQ, consulte Mapeando campos do AMQP em campos do IBM MQ (mensagens recebidas).

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.

MQ Light ou aplicativos AMQP 1.0 também podem consumir mensagens que são publicadas por aplicativos do MQ existente. 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 os mapeamentos de mensagem do MQ para AMQP, consulte Mapeando campos do AMQP em campos do IBM MQ (mensagens recebidas).



Informações relacionadas

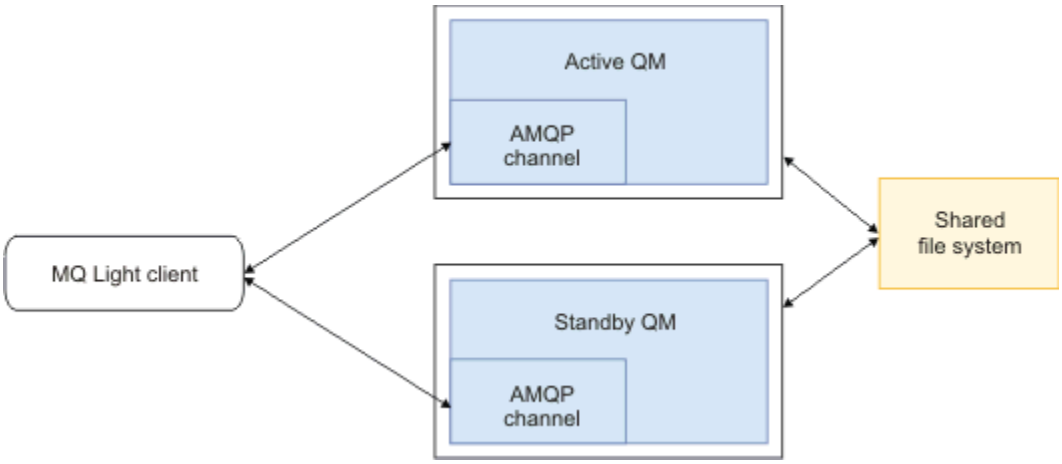
Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW Configurando um cliente AMQP para alta disponibilidade

É possível configurar os aplicativos MQ Light ou AMQP 1.0 para conectar à instância ativa de um gerenciador de filas de várias instâncias do IBM MQ e executar failover para a instância do gerenciador de filas de várias instâncias e 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 o MQ Light API com uma função customizada, que será chamado se o cliente perde sua conexão com o servidor. A função pode se conectar a um endereço IP alternativo, por exemplo, um gerenciador de filas MQ em espera, 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.



Informações relacionadas

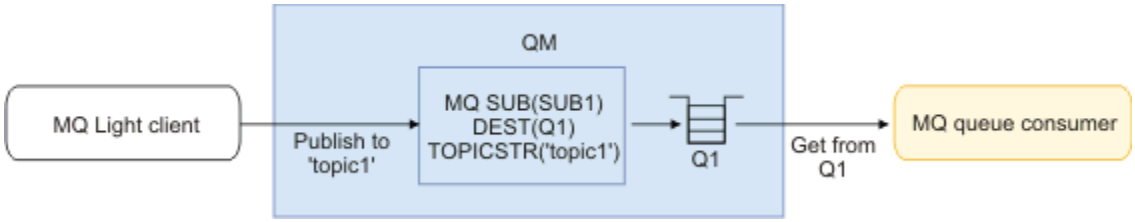
Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW 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 deseja que um aplicativo MQ Light ou AMQP 1.0 envie mensagens para um aplicativo IBM MQ existente que esteja configurado para ler de uma fila, deve-se definir uma assinatura administrada do IBM MQ 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.



Informações relacionadas

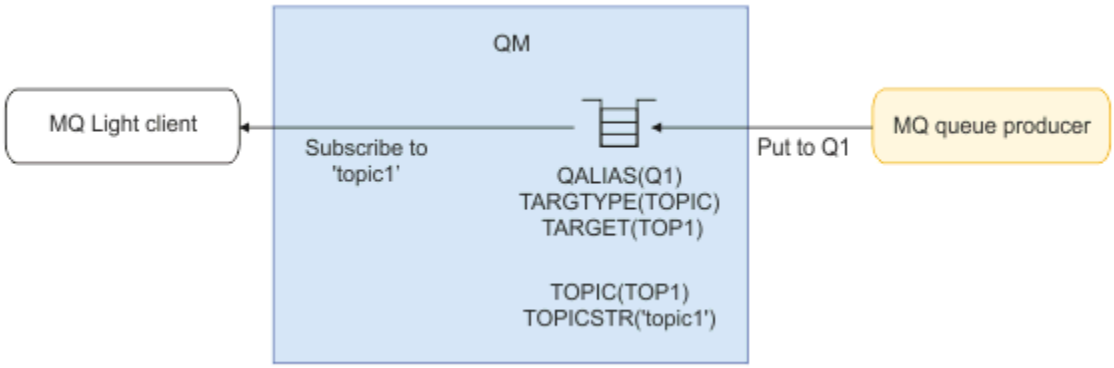
Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW 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 deseja que um aplicativo MQ Light ou AMQP 1.0 receba mensagens de um aplicativo IBM MQ existente que esteja configurado para colocar mensagens em uma fila, deve-se 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.



Informações relacionadas

Criando e utilizando canais AMQP

Protegendo clientes AMQP

ULW Cliente AMQP submetendo solicitações para e consumindo respostas de um servidor de aplicativos

Um MQ Light ou outro 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 respostas. 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.

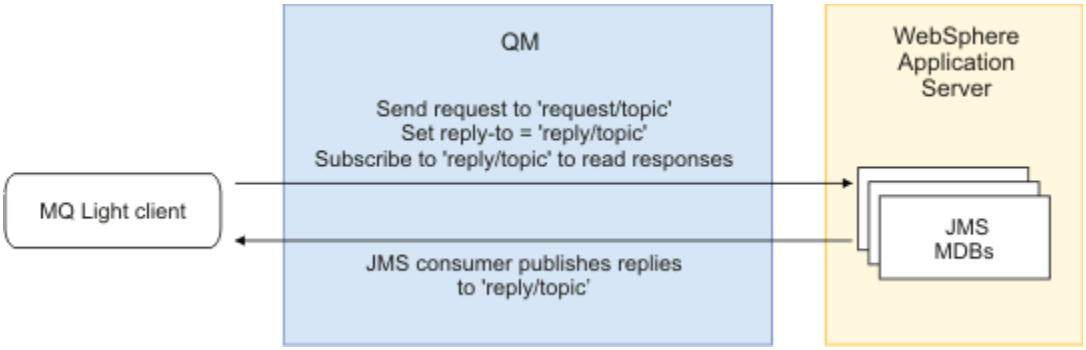
A propriedade JMS é **JMSReplyTo**. A sequência de resposta do AMQP deve ser um dos tipos a seguir:

- 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 de 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 do AMQP para a uma propriedade **JMSReplyTo**, consulte Mapeando campos do AMQP em campos IBM MQ (mensagens recebidas)



Informações relacionadas
Criando e utilizando canais AMQP
Protegendo clientes AMQP

ULW Implementando os aplicativos do MQ Light em um ambiente local do IBM MQ

O IBM MQ suporta a API de sistema de mensagens do IBM MQ Light, portanto, é possível usar o IBM MQ para implementar seu aplicativo MQ Light em um ambiente IBM MQ no local.

É possível implementar aplicativos MQ Light em um gerenciador de filas do IBM MQ, permitindo que seus aplicativos MQ Light se comuniquem com aplicativos corporativos existentes já conectados ao IBM MQ, conforme ilustrado no diagrama a seguir:

Os aplicativos MQ Light podem compartilhar um espaço de tópico com aplicativos IBM MQ existentes, o que permite que eles interajam com os sistemas corporativos existentes.

Informações relacionadas
Criando e utilizando canais AMQP
Protegendo clientes AMQP

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 suportadas pelo IBM MQ dentre as quais é possível escolher para enviar e receber mensagens do IBM MQ usando REST são as opções a seguir:

- IBM MQ messaging REST API
- IBM MQ ponte para HTTP
- IBM z/OS Connect EE
- IBM Integration Bus
- DataPower

V 9.0.4 IBM MQ messaging REST API

O messaging REST API vem como padrão com o IBM MQ do IBM MQ 9.0.4 e é ativado por padrão. É possível usar o messaging REST API para enviar e receber mensagens do IBM MQ em formato de texto sem formatação.

Os aplicativos podem emitir um HTTP POST para enviar uma mensagem para o IBM MQ ou um HTTP DELETE para obter destrutivamente uma mensagem do IBM MQ. 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. Os usuários devem ser autenticados para o servidor mqweb e devem ser um membro da função MQWebUser.

Para obter informações adicionais, consulte Sistema de mensagens usando a REST API.

IBM MQ ponte para HTTP

A ponte IBM MQ para HTTP é um aplicativo JEE que pode ser instalado em um ambiente de tempo de execução adequado e fornece uma REST API básica sobre as filas e tópicos hospedados por um único gerenciador de filas do MQ.

Os aplicativos podem emitir um HTTP POST para a ponte para enviar uma mensagem para o IBM MQ, um HTTP GET para procurar uma mensagem do IBM MQ ou um HTTP DELETE para obter destrutivamente uma mensagem do IBM MQ. O suporte é fornecido para vários cabeçalhos de HTTP diferentes que podem ser usados para configurar propriedades de mensagens.

Nota: A partir do IBM MQ 8.0, o IBM MQ bridge for HTTP está descontinuado. O IBM messaging REST API fornecido do IBM MQ 9.0.4 deve ser usado como alternativa.

Para obter informações adicionais, veja Desenvolvendo serviços da web com o IBM MQ bridge for HTTP.

IBM z/OS Connect EE

O IBM z/OS Connect EE (zCEE) é um produto z/OS que permite construir APIs de REST em cima de ativos existentes do z/OS, como transações do CICS ou IMS e filas e tópicos do IBM MQ. O ativo do z/OS existente é ocultado do usuário. Isso permite que o REST ative os ativos sem mudá-los ou qualquer um dos aplicativos existentes que os usam.

Com o zCEE, é possível construir facilmente uma REST API que enviará e receberá dados JSON e opcionalmente transformará esses dados nas estruturas de linguagem mais tradicionais esperadas por muitos aplicativos de mainframe. Por exemplo, copybooks COBOL.

Um editor de API baseado no Eclipse pode ser usado para construir uma API RESTful abrangente, fazendo uso de parâmetros de consulta e segmentos de caminho de URL, manipulando o formato JSON conforme ele flui no tempo de execução do zCEE.

O zCEE pode ser usado para expor filas e tópicos do IBM MQ como serviços. Dois tipos de serviços diferentes são suportados:

- Serviços unidirecionais: a função fornecida é semelhante àquela fornecida com o IBM MQ bridge for HTTP, em que um HTTP POST envia uma mensagem, um HTTP DELETE obtém destrutivamente uma mensagem. Os benefícios principais sobre a ponte são o suporte de conversão de dados integrado e que é possível usar o editor de API para construir uma API RESTful mais abrangente.
- Serviços bidirecionais: isso fornece uma REST API sobre um par de filas usadas por um aplicativo backend de estilo solicitação-resposta. Os responsáveis pela chamada emitem um HTTP POST para o serviço bidirecional e enviam dados JSON. Esses dados são colocados em uma fila de solicitações, na qual eles são processados pelo aplicativo backend e uma resposta é colocada na fila de resposta. Essa resposta é recuperada e enviada de volta para o responsável pela chamada como o corpo de resposta de POST.

zCEE é suportado no IBM MQ 8 e posterior. Para obter informações adicionais, veja IBM MQ for z/OS Service Provider for z/OS Connect.

IBM Integration Bus

O IBM Integration Bus é a tecnologia de integração principal da IBM que pode ser usada para conectar aplicativos e sistemas juntos, 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 Serviço de gateway multiprotocolo.

V 9.0.4 Sistema de mensagens usando a REST API

É possível usar a messaging REST API para enviar e receber mensagens do IBM MQ. As informações são atualmente 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 manualmente 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. O responsável pela chamada deve ser autenticado para o servidor mqweb e deve ser um membro da função MQWebUser. O responsável pela chamada também deve estar autorizado a acessar a fila especificada. Para obter mais informações sobre segurança para o REST API, consulte Segurança do IBM MQ Console e do REST API.

Procedimento

- “Introdução ao messaging REST API” na página 697
- “Usando o messaging REST API” na página 699
- “Limitações do sistema de mensagens da REST API” na página 700
- REST API manipulação de erros
- DescobertaREST API
- Suporte ao Idioma NacionalREST API

Informações relacionadas

Referência do sistema de mensagens da REST API

V 9.0.4 Introdução ao messaging REST API

Para poder iniciar o messaging REST API, deve-se instalar os componentes corretos, ativar a REST API, configurar a segurança e iniciar o servidor mqweb.

Antes de começar

IBM i Em IBM i, os comando devem estar em execução em QSHELL

Sobre esta tarefa

O procedimento para esta tarefa tem o foco na introdução rápida ao messaging REST API. As etapas para configurar a segurança descrevem como configurar um registro do usuário básico, mas existem outras opções para configurar usuários e funções. Para obter mais informações sobre como configurar a segurança para a messaging REST API, consulte IBM MQ Console e a segurança da REST API.

Nota: Deve-se ser um usuário privilegiado para acessar o arquivo `mqwebuser.xml`.

Procedimento

1. Instale o IBM MQ Console e o componente REST API:
   - AIX No AIX, instale o conjunto de arquivos `mqm.web.rte`.
   - Linux No Linux, instale o componente `MQSeriesWeb`. Para obter mais informações sobre como instalar componentes e recursos no Linux, consulte Tarefas de instalação do Linux.
   - Windows No Windows, instale o recurso `Web Administration`. Para obter mais informações sobre como instalar componentes e recursos no Windows, consulte Tarefas de instalação do Windows.
   - z/OS No z/OS, instale o recurso `IBM MQ for z/OS Unix System Services Components`. Para obter mais informações sobre como instalar componentes e recursos no z/OS, consulte Tarefas de instalação do z/OS.
2. Configure usuários e funções para poder utilizar o messaging REST API:
   a) Copie o arquivo `basic_registry.xml` do diretório *MQ_INSTALLATION_PATH* `/web/mq/samp/configuration`.
   b) Coloque o arquivo XML de amostra no diretório apropriado:
      - ULW No UNIX, Linux, and Windows: *MQ_DATA_DIRECTORY*`/web/installations/`*installationName*`/servers/mqweb`
      - z/OS No z/OS: *WLP_user_directory*`/servers/mqweb`

      em que *WLP_user_directory* é o diretório que foi especificado quando o script **crtmqweb.sh** foi executado para criar a definição do servidor do mqweb.
   c) Renomeie o arquivo XML de amostra para `mqwebuser.xml`.

      Nota: Esse arquivo renomeado substitui um arquivo existente que também é usado para o IBM MQ Console. Portanto, se você mudou o arquivo `mqwebuser.xml` para o IBM MQ Console, copie suas mudanças no novo arquivo XML antes de renomeá-lo.
   d) Opcional: Edite o arquivo `mqwebuser.xml` para incluir usuários e grupos. Designe esses usuários e grupos à função `MQWebUser` para que sejam autorizados a usar o messaging REST API. As funções `MQWebAdmin` e `MQWebAdminRO` não são aplicáveis para o messaging REST API Também é possível mudar as senhas para os usuários que são definidos por padrão e codificar as novas senhas. Assegure-se de que os usuários estejam autorizados a acessar a fila especificada. Para obter mais informações, veja Configurando usuários e funções.
3. Ative as conexões remotas com o servidor mqweb usando o comando **setmqweb**:

```
setmqweb properties -k httpHost -v hostname
```

Em que *hostname* especifica o endereço IP, o nome do host do servidor de nomes de domínio (DNS) com o sufixo de nome de domínio ou o nome do host do DNS do servidor no qual o IBM MQ está instalado. Use um asterisco, *, para especificar todas as interfaces de rede disponíveis.

Atenção: V 9.0.4 z/OS

Antes de emitir os comandos **setmqweb** ou **dspmqweb** no z/OS, deve-se configurar a variável de ambiente WLP_USER_DIR, de modo que a variável aponte para a configuração do servidor mqweb.

Para fazer isto, emita o seguinte comando:

```
export WLP_USER_DIR=WLP_user_directory
```

em que *WLP_user_directory* é o nome do diretório que é transmitido para `crtmqweb.sh`. Por exemplo:

```
export WLP_USER_DIR=/var/mqm/web/installation1
```

Para obter mais informações, consulte Criar a definição do servidor Liberty.

4. Inicie o servidor mqweb que suporta o REST API:
   - ULW Como um usuário privilegiado, insira o comando a seguir na linha de comandos:

     `strmqweb`
   - z/OS No z/OS, inicie o procedimento que você criou na Tarefa 29: criar um procedimento para o servidor IBM WLP.

Como proceder a seguir

1. Escolha como os usuários do messaging REST API são autenticados com o servidor mqweb. Não é necessário usar o mesmo método para todos os usuários. Estão disponíveis as seguintes opções:
   - Permitir que os usuários se autentifiquem usando a autenticação de HTTP básica. Nesse caso, um nome de usuário e uma senha são codificados, mas não criptografados, e enviados com cada solicitação de REST API para autenticar e autorizar o usuário para essa solicitação. Para que essa autenticação seja segura, deve-se usar uma conexão segura. Ou seja, deve-se usar HTTPS. Para obter mais informações, veja Usando a autenticação básica HTTP com a REST API.
   - Permitir que os usuários se autentifiquem usando a autenticação do token. Nesse caso, um usuário fornece um ID de usuário e uma senha para o recurso REST API `login` com o método HTTP POST. É gerado um token LTPA que permite que o usuário permaneça com login efetuado e autorizado por um período de tempo configurado. Para que essa autenticação seja segura, deve-se usar uma conexão segura. Ou seja, deve-se usar HTTPS. Para obter mais informações, veja Usando a autenticação baseada em token com a REST API.
   - Permitir que os usuários se autentifiquem usando certificados de cliente. Nesse caso, o usuário não usa um ID do usuário ou senha para efetuar login no messaging REST API, mas usa o certificado de cliente. Para obter mais informações, veja Usando a autenticação por certificado de cliente com a REST API.
2. Configure as definições da REST API, incluindo a ativação de conexões HTTP e a mudança do número da porta. Para obter mais informações, veja Configurando IBM MQ Console e a REST API.
3. Opcionalmente, configure o Cross Origin Resource Sharing para a REST API. Por padrão, não é possível acessar a REST API por meio de recursos da web não hospedados no mesmo domínio que a REST API. Ou seja, as solicitações de origem cruzada não estão ativadas. É possível configurar o Cross Origin Resource Sharing (CORS) para permitir solicitações de origem cruzada de URLs especificadas. Para obter mais informações, veja Configurando o CORS para a REST API.

4. Use o REST API. Para obter mais informações, consulte “Usando o messaging REST API” na página 699 e a Referência do sistema de mensagens da REST API.

Nota: É possível parar o servidor mqweb a qualquer momento usando o comando **endmqweb**. No entanto, se o servidor mqweb não estiver em execução, não será possível usar a REST API ou o IBM MQ Console.

V 9.0.4 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 de autenticação, consulte Segurança do IBM MQ Console e do REST API.
- A REST API faz distinção entre maiúsculas e minúsculas. Por exemplo, um HTTP POST na URL a seguir resulta em um erro se o gerenciador de filas é denominado `qmgr1`.

  ```
  /ibmmq/rest/v1/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- 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 porcentagem, %, deve ser codificado como %25.

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 699 e “1.b” na página 700.

Procedimento

1. Construa a URL:
   a) Inicie com a URL de prefixo a seguir:

   ```
   https://host:port/ibmmq/rest/v1/messaging
   ```

host
Especifica o nome do host ou endereço IP no qual a messaging REST API está disponível.

O valor padrão é `localhost`.

port
Especifica o número da porta HTTPS usada pela messaging REST API.

O valor padrão é 9443.

Se você ativar as conexões HTTP, será possível usar HTTP no lugar de HTTPS. Para obter mais informações sobre como ativar HTTP, veja Configurando as portas HTTP e HTTPS.

Para obter mais informações sobre como determinar a URL do prefixo, consulte Determinando a URL da REST API.

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 do sistema de mensagens, os segmentos da variável podem ser identificados na URL pelas chaves que o cercam *{ }*. Para obter mais informações, consulte /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

Por exemplo, para interagir com a fila *Q1* associada com o gerenciador de filas *QM1*, inclua `/qmgr` e `/queue` na URL de prefixo para criar a URL a seguir:

```
https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/Q1/message
```

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/v1/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 9.0.4 Limitações do sistema de mensagens da REST API

Antes de usar o messaging REST API, considere as limitações a seguir:

- A API não suporta atualmente o sistema de mensagens de publicar/assinar. O messaging REST API suporta apenas o sistema de mensagens ponto a ponto síncrono.
- A API não suporta atualmente a procura de filas. As mensagens são recebidas destrutivamente da fila do IBM MQ usando o método HTTP DELETE. Para obter informações adicionais, veja DELETE.
- Ao enviar uma mensagem, somente conteúdo baseado em texto UTF-8 pode ser usado. As mensagens são colocadas como mensagens formatadas pelo IBM MQ MQSTR. Para obter informações adicionais, veja POST.

  Ao receber uma mensagem, somente mensagens formatadas pelo IBM MQ MQSTR são suportadas. Subsequentemente, todas as mensagens são recebidas 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. Para obter informações adicionais, veja Manipulando mensagens suspeitas em classes do IBM MQ para JMS.

- Se você usar Advanced Message Security (AMS) com o messaging REST API, observe que todas as mensagens serão criptografadas usando o contexto do servidor mqweb, não o contexto do usuário que postar a mensagem.
- As novas linhas em sequências de entrada são removidas na operação HTTP POST Aplicativos REST

  não deve usar novas linhas em mensagens enviadas ou publicadas usando a API REST, pois elas serão perdidas.

Desenvolvendo serviços da web com o IBM MQ bridge for HTTP

Com o IBM MQ bridge for HTTP, os aplicativos clientes podem trocar mensagens com IBM MQ sem a necessidade de instalar um IBM MQ MQI client. É possível chamar IBM MQ a partir de qualquer plataforma ou linguagem com recursos HTTP.

Sobre esta tarefa

Nota: A partir do IBM MQ 8.0, o IBM MQ bridge for HTTP está descontinuado. O IBM messaging REST API fornecido do IBM MQ 9.0.4 deve ser usado como alternativa.

Introdução ao IBM MQ bridge for HTTP

O IBM MQ bridge for HTTP é um aplicativo da web Java, Enterprise Environment (JEE). Clientes HTTP podem enviar solicitações **POST**, **GET** e **DELETE** a ele para colocar, procurar e excluir mensagens a partir de filas do IBM MQ. O IBM MQ bridge for HTTP não é adequado para uso com mensagens, se a entrega garantida for necessária.

Benefícios

Com o IBM MQ bridge for HTTP, é possível enviar e receber mensagens do IBM MQ usando HTTP a partir de uma ampla variedade de ambientes:

- Ambientes que suportam HTTP, mas não o IBM MQ.
- Ambientes que têm espaço de armazenamento insuficiente para instalar um IBM MQ MQI client.
- Ambientes que são muito numerosos para instalar o IBM MQ MQI client em cada sistema que requer acesso ao IBM MQ.
- Os aplicativos baseados na web a partir dos quais deseja enviar ou receber mensagens sem codificar sua própria ponte para o IBM MQ.
- Aplicativos baseados na web que você deseja aprimorar, usando técnicas assíncronas, como AJAX. IBM MQ bridge for HTTP disponibiliza filas e tópicos do IBM MQ usando Representation State Transfer (REST) sobre HTTP.

O suporte a HTTP pode ser usado com topologias de sistema de mensagens ponto a ponto e de publicar/assinar.

Como o suporte a HTTP funciona?

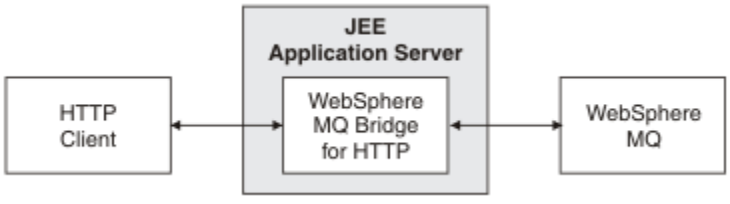


Figura 59. IBM MQ bridge for HTTP

O aplicativo da web IBM MQ bridge for HTTP recebe solicitações de HTTP de um ou mais clientes. Ele interage com o IBM MQ em seu nome e retorna respostas HTTP a eles.

O IBM MQ bridge for HTTP é um servlet JEE conectado ao IBM MQ usando um adaptador de recursos. O servlet HTTP manipula três tipos diferentes de solicitações de HTTP: **POST**, **GET** e **DELETE**.

Tabela 84. Verbos IBM MQ bridge for HTTP

Solicitação de HTTP	**Resultado**
POST	Coloca uma mensagem em uma fila ou tópico.
RECEBER	Procura a primeira mensagem em uma fila. Alinhado ao protocolo HTTP, **GET** não exclui a mensagem da fila. Não use **GET** com mensagens de publicar/assinar.
EXCLUIR	Obtém e exclui uma mensagem de uma fila ou tópico.

Exemplo de HTTP POST

HTTP **POST** coloca uma mensagem em uma fila, ou uma publicação para um tópico. A amostra do Java **HTTPPOST** é um exemplo de uma solicitação de HTTP **POST** de uma mensagem para uma fila. Em vez de usar Java, seria possível criar uma solicitação de HTTP **POST** usando um formulário do navegador ou um kit de ferramentas AJAX em vez disso.

A figura a seguir mostra uma solicitação de HTTP para colocar uma mensagem em uma fila chamada `myQueue`. Essa solicitação contém o cabeçalho HTTP `x-msg-correlId` para configurar o ID de correlação da mensagem IBM MQ.

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

*Figura 60. Exemplo de uma solicitação de HTTP **POST** para uma fila*

A figura a seguir mostra a resposta enviada de volta ao cliente. Não há conteúdo de resposta.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Figura 61. Exemplo de uma resposta de HTTP POST

Exemplo de HTTP DELETE

HTTP **DELETE** recebe uma mensagem de uma fila e exclui a mensagem, ou recupera e exclui uma publicação. A amostra do Java **HTTPDELETE** é um exemplo de uma solicitação de HTTP **DELETE** lendo uma mensagem de uma fila. Em vez de usar Java, seria possível criar uma solicitação de HTTP **DELETE** usando um formulário do navegador ou um kit de ferramentas AJAX em vez disso.

A figura a seguir mostra uma solicitação de HTTP para excluir a próxima mensagem na fila chamada `myQueue`. Em resposta, o corpo da mensagem é retornado ao cliente. Em termos do IBM MQ, o HTTP **DELETE** é um get destrutivo.

A solicitação contém o cabeçalho de solicitação HTTP `x-msg-wait`, que instrui a ponte IBM MQ para HTTP por quanto tempo esperar pela chegada de uma mensagem na fila. A solicitação também contém o

cabeçalho da solicitação `x-msg-require-headers`, que especifica que o cliente deve receber o ID de correlação da mensagem na resposta.

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Figura 62. Exemplo de uma solicitação de HTTP **DELETE***

A figura a seguir mostra a resposta retornada ao cliente. O ID de correlação é retornado ao cliente, conforme solicitado no `x-msg-require-headers` da solicitação.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

*Figura 63. Exemplo de uma resposta HTTP **DELETE***

Exemplo de HTTP GET

HTTP **GET** recebe uma mensagem de uma fila. A mensagem permanece na fila. Em temos do IBM MQ, HTTP **GET** é uma solicitação de procura. Seria possível criar uma solicitação de HTTP **GET** usando um cliente Java, um formulário do navegador ou um kit de ferramentas AJAX.

A figura a seguir mostra uma solicitação de HTTP para procurar a próxima mensagem na fila chamada `myQueue`.

A solicitação contém o cabeçalho da solicitação de HTTP `x-msg-wait`, que instrui o IBM MQ bridge for HTTP quanto tempo esperar para uma mensagem chegar à fila. A solicitação também contém o cabeçalho da solicitação `x-msg-require-headers`, que especifica que o cliente deve receber o ID de correlação da mensagem na resposta.

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Figura 64. Exemplo de uma solicitação de HTTP **GET***

A figura a seguir mostra a resposta retornada ao cliente. O ID de correlação é retornado ao cliente, conforme solicitado no `x-msg-require-headers` da solicitação.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

*Figura 65. Exemplo de uma resposta HTTP **GET***

Instalando, configurando e verificando o IBM MQ bridge for HTTP

Obtenha o IBM MQ bridge for HTTP instalando o Java Messaging and Web Services a partir de um IBM MQ MQI client ou materiais de instalação do servidor. Implemente IBM MQ bridge for HTTP em um servidor de aplicativos adequado.

Antes de começar

Verifique os produtos com pré-requisitos em Requisitos do sistema para IBM MQ. O processo de instalação não verifica a presença e a disponibilidade do software obrigatório para executar o IBM MQ bridge for HTTP. Deve-se verificar se os pré-requisitos estão instalados.

O IBM MQ bridge for HTTP é um aplicativo Java EE 4. Para obter informações sobre os servidores de aplicativos suportados, veja Requisitos do sistema para IBM MQ.

Sobre esta tarefa

O IBM MQ bridge for HTTP é fornecido como um arquivo `.war`, `WMQHTTP.war`.

- No UNIX e no Linux:
  - `WMQHTTP.war` é incluído como parte da opção de instalação do Java Messaging and Web Services. A opção está disponível nos materiais de instalação de servidor e clientes.
  - O `WMQHTTP.war` é instalado no *mqmtop*`/java/http/WMQHTTP.war` *mqmtop* é diretório em que o IBM MQ está instalado.
  - O `WMQHTTP.samples` é instalado no *mqmtop*`/java/http/samples` *mqmtop* é diretório em que o IBM MQ está instalado.
- z/OS No z/OS:
  - `WMQHTTP.war` é incluído como parte do recurso do IBM MQ z/OS UNIX System Services Components.
  - O `WMQHTTP.war` é instalado no *PathPrefix*`/usr/lpp/mqm/V7R0M0/HTTPBridge/`, em que *PathPrefix* é um prefixo opcional definido pelo cliente

Execute as seguintes etapas de instalação para instalar o IBM MQ bridge for HTTP, implementá-lo e configurá-lo e verificar a configuração. Os detalhes das etapas de configuração variam em diferentes servidores de aplicativos. Use o "Implementando e verificando IBM MQ bridge for HTTP no WebSphere Application Server 6.1.0.9" na página 705 como um modelo para as etapas a seguir em seu servidor de aplicativos.

Procedimento

1. Obtenha `WMQHTTP.war` instalando o IBM MQ MQI client ou servidor.
2. Copie `WMQHTTP.war` para um servidor a partir do qual ele pode ser implementado em um servidor de aplicativos.
3. Implemente `WMQHTTP.war` em um servidor de aplicativos.

4. Se necessário, instale o IBM MQ como um adaptador de recursos em seu servidor de aplicativos.

   Descubra se o IBM MQ já está configurado como um provedor de sistemas de mensagens em seu servidor de aplicativos. Use a ferramenta de gerenciamento ou administração fornecida com seu servidor de aplicativos para procurar o IBM MQ. O IBM MQ pode ser localizado no caminho a seguir, **Recursos > JMS > Provedores de sistemas de mensagens**.
5. Configure um connection factory no servidor de aplicativos para se conectar a um gerenciador de fila que usa o transporte IBM MQ MQI client[7].
6. Configure o aplicativo da web `WMQHTTP.war` no servidor de aplicativos para usar a connection factory
7. Verifique a configuração.
   a) Configure o gerenciador de filas denominado na connection factory e uma fila local.
   b) Coloque uma mensagem na fila local.
   c) Crie o canal de conexão do servidor denominado na connection factory, com a autoridade para ler e gravar a fila local.
   d) Inicie o gerenciador de filas e o listener.
   e) Inicie o servidor de aplicativos e `WMQHTTP.war`, se eles ainda não estiverem em execução.
   f) Abra um navegador e digite `http://`*`hostname: web port`*`/`*`Context root`*`/msg/queue/`*`local queue`*

Resultados

A janela do navegador exibe a mensagem colocada na fila local.

Como proceder a seguir

1. Tente o exemplo, "Implementando e verificando IBM MQ bridge for HTTP no WebSphere Application Server 6.1.0.9" na página 705.
2. Execute os aplicativos Java HTTP de amostra.

Implementando e verificando IBM MQ bridge for HTTP no WebSphere Application Server 6.1.0.9

Use o exemplo a seguir para preparar uma implementação do IBM MQ bridge for HTTP para executar os programas Java HTTP de amostra. A implementação está em WebSphere Application Server 6.1.0.9

Antes de começar

1. Siga as instruções em "Instalando, configurando e verificando o IBM MQ bridge for HTTP" na página 704, para copiar `WMQHTTP.war` para um servidor acessível à sua instalação do WebSphere Application Server.
2. Configure um gerenciador de filas, e uma fila, para usar para testar a configuração:
   - No exemplo, o gerenciador de filas é configurado como usando os valores em Tabela 85 na página 705:

Tabela 85. Configuração do Gerenciador de Filas	
Object	**Value**
Nome do host	`itso-01`
Gerenciador de Filas	`QM1`
Fila local	`HTTPTESTQ`

[7] Inicialmente, pelo menos, configure o transporte do cliente Alguns servidores de aplicativos podem se conectar ao IBM MQ usando direto ou as conexões do modo de ligações.

Tabela 85. Configuração do Gerenciador de Filas (continuação)	
Object	**Value**
Canal de conexão do servidor	MYSVRCON. Configure um ID do usuário do MCA com autoridade suficiente para ler e gravar em HTTPTESTQ.
Porta listener	1414

3. Inicie o gerenciador de filas e o listener
4. Coloque uma mensagem de teste em HTTPTESTQ. Por exemplo:
   a. Inicie o IBM MQ Explorer.
   b. Na lista de filas locais para QM1, clique com o botão direito em **HTTPTESTQ > Colocar mensagem de teste > digite `First Message` > Colocar mensagem > Fechar**
5. Inicie o servidor de aplicativos e efetue sign on no Integrated Solutions Console.

Sobre esta tarefa

O exemplo mostra as etapas a serem executadas se estiver executando WebSphere Application Server 6.1.0.9 como seu servidor de aplicativos. Se você estiver executando uma versão diferente do WebSphere Application Server ou executando um servidor de aplicativos diferente, as etapas serão diferentes. O WebSphere Application Server 6.1.0.9 é pré-configurado com o IBM MQ instalado como um provedor de mensagens, usando as bibliotecas IBM MQ MQI client . Se o IBM MQ não for pré-configurado como um provedor de sistemas de mensagens ou se você desejar usar as ligações do servidor IBM MQ, será necessário instalar e configurar o IBM MQ adaptador de recursos para JEE em seu servidor de aplicativos.

Siga as instruções para implementar IBM MQ bridge for HTTP no WebSphere Application Server 6.1.0.9e verifique a implementação usando um navegador:

Procedimento

1. Na área de janela de navegação, clique em **Recursos > Provedores JMS > Provedor de sistemas de mensagens do IBM MQ**.

   É possível configurar no nível do Nó, da Célula ou do Servidor, dependendo de sua implementação do WebSphere Application Server. O exemplo usa a implementação no nível do Servidor.
2. Em **Propriedades adicionais**, clique em **Connection factories > Novo**.
3. No formulário de provedores JMS, forneça as informações em Tabela 86 na página 706 ou alternativas de sua preferência, clique em **Aplicar > Salvar**.

Tabela 86. Configure ou modifique os campos a seguir	
Campo	**Value**
Nome	WMQHTTPBridge
Nome JNDI	jms/WMQHTTPJCAConnectionFactory
Gerenciador de Filas	QM1
Host	itso-01
Port	1414
Canal	MYSVRCON
Tipo de transporte	CLIENTE

4. Na área de janela de navegação, clique em **Aplicativos > Instalar novo aplicativo**.
5. Insira o caminho para `WMQHTTP.war` no formulário e forneça uma raiz de contexto, clique em **Avançar**.

a) A raiz de contexto é opcional. `mq` é a raiz de contexto padrão para os aplicativos HTTP de amostra.

b) A raiz de contexto forma parte da URI identificando o IBM MQ bridge for HTTP. É possível omitir a raiz de contexto ou mudá-la posteriormente.

6. Na página **Selecionar opções de instalação** do assistente de instalação, não é necessário mudar nenhum dos padrões, clique em **Avançar**.
7. Na página **Mapear módulos para servidores**, selecione um Cluster ou um Servidor, marque a caixa Selecionar, clique em **Aplicar > Avançar**.
8. Na página **Mapear Referências de Recursos para Recursos** , no formulário **javax.jms.ConnectionFactory** , clique em **Procurar ...** na linha IBM MQ bridge for HTTP ..
9. Na página **Aplicativos corporativos > Recursos disponíveis**, selecione **WMQHTTPBridge**, clique em **Aplicar**.
10. De volta ao formato **javax.jms.ConnectionFactory**, selecione o método de autenticação.
    a) Para o exemplo, escolha **Nenhum**, clique em **Aplicar**. As outras opções requerem configuração adicional.
11. Marque a caixa de seleção **Selecionar** para o IBM MQ bridge for HTTP, clique em **Avançar > Avançar > Concluir > Salvar**
12. Na área de janela de navegação, clique em **Aplicativos > Aplicativos corporativos**.
13. Marque a caixa de seleção para `WMQHTTP.war`, clique em **Iniciar**.
14. Abra a janela do navegador. Digite `http://itso-01:9080/mq/msg/queue/HTTPTESTQ`, usando o nome do host e a porta apropriados.

Resultados

A janela do navegador exibirá `First Message`, se a configuração for bem-sucedida.

Como proceder a seguir

Execute os aplicativos Java HTTP de amostra.

Publicação/assinatura usando o IBM MQ bridge for HTTP

O IBM MQ bridge for HTTP usa a interface de publicação/assinatura do IBM MQ classes for JMS. HTTP **POST** cria uma publicação. HTTP **DELETE** cria uma assinatura gerenciada não durável. Deve-se configurar publicação/assinatura para JMS antes de usar o URI de tópico.

A publicação / assinatura é totalmente integrada ao IBM WebSphere MQ 7 Antes da versão 7, um broker de publicação / assinatura separado manipulava as publicações e assinaturas Ele é chamado de publicação / assinatura "enfileirada" , para distingui-lo da publicação / assinatura totalmente integrada na versão 7 O IBM WebSphere MQ 7 emula a publicação / assinatura enfileirada usando a publicação / assinatura integrada. A emulação permite que aplicativos de publicação/assinatura enfileirada existentes coexistam com aplicativos integrados em execução no mesmo gerenciador de filas. Os aplicativos de publicação/assinatura enfileirada também podem interoperar com aplicativos integrados, compartilhando os mesmos tópicos. No IBM WebSphere MQ 6, o broker foi fornecido com o IBM MQ; antes do IBM WebSphere MQ 6 estava disponível como um SupportPack.

Configuração

O IBM MQ bridge for HTTP usa a interface do JMS para publicação e assinatura. Na versão 7, é possível controlar se o IBM MQ classes for JMS usa a publicação/assinatura enfileirada ou integrada, usando a propriedade `PROVIDERVERSION` JMS.

Uma consideração adicional é que é possível usar as bibliotecas do IBM MQ MQI client com o IBM MQ bridge for HTTP ou servidor de bibliotecas. As bibliotecas do clienteIBM WebSphere MQ 6 apenas suportam publicação/assinatura enfileirada, enquanto as bibliotecas da versão 7 suportam publicação/assinatura enfileirada e integrada. A maioria dos servidores da web ou de aplicativos que usam IBM MQ como um provedor de sistemas de mensagens fazem isso usando as bibliotecas do cliente. Para usar

publicação / assinatura integrada, as bibliotecas do IBM MQ MQI client e do servidor devem estar pelo menos na versão 7. Se uma delas estiver executando uma versão anterior do WebSphere de 7, você deverá configurar a publicação / assinatura enfileirada; consulte Tabela 87 na página 708. Verifique quais bibliotecas estão instaladas ou configuradas com o servidor da web ou servidor de aplicativos que você está usando.

Tabela 87. Modos de configuração de publicação/assinatura

	Cliente V6 ou anterior	**Cliente V7 ou posterior**
Servidor V6 ou anterior	1. Execute o script `\java\bin\MQJMS_PSQ.mqsc`	Não Suportado
Servidor V7 ou posterior	1. Execute o script `\java\bin\MQJMS_PSQ.mqsc` 2. Configure o gerenciador de filas como `PSMODE=ENABLED`	1. Se PROVIDERVERSION = 7 a. Configure o gerenciador de filas como `PSMODE=ENABLED` ou `PSMODE=COMPAT` 2. If PROVIDERVERSION = 6 a. Configure o gerenciador de filas como `PSMODE=ENABLED`

Publicar

Envie uma solicitação de HTTP **POST** com o URI:

```
http://hostname: port/context_root/msg/topic/topicString
```

O conteúdo da mensagem é publicado usando a sequência de tópicos *`topicString`*.

Assinar

Envie uma solicitação de HTTP **DELETE** com o URI:

```
http://hostname: port/context_root/msg/topic/topicString
```

O IBM MQ bridge for HTTP cria uma assinatura não durável gerenciada para a sequência de tópicos *`topicString`*. A assinatura é excluída assim que uma publicação é retornada ou até que o intervalo de espera configurado pelo cabeçalho de entidade customizado, `x-msg-wait`, expire.

Executando as amostras do IBM MQ bridge for HTTP

As amostras do IBM MQ bridge for HTTP estão disponíveis para uso somente no sistema operacional Windows. As amostras mostram como enviar comandos HTTP **POST** e HTTP **DELETE** para o IBM MQ bridge for HTTP a partir de programas Java.

Antes de começar

Verifique a instalação da ponte do IBM MQ para HTTP executando a etapa “7” na página 705 em “Instalando, configurando e verificando o IBM MQ bridge for HTTP” na página 704.

As amostras de HTTP são instaladas nos diretórios mostrados em Tabela 88 na página 709. Em cada caso, o código-fonte é instalado no subdiretório `/src`.

Tabela 88. Local de amostras HTTP	
Plataforma	**Local**
Windows Windows	*MQ_INSTALLATION_PATH*/tools/http/samples
z/OS z/OS z/OS z/OS	*PathPrefix*/usr/lpp/mqm/V7R0M0/http/samples
IBM i IBM i IBM i IBM i	*MQ_INSTALLATION_PATH*/java/samples/http
Todas as outras plataformas	*MQ_INSTALLATION_PATH*/samp/http

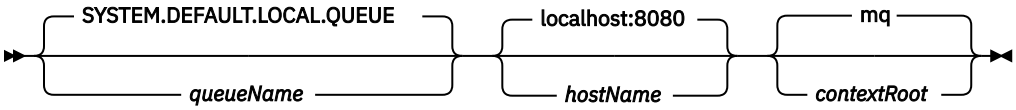
MQ_INSTALLATION_PATH representa o diretório no qual o IBM MQ está instalado.

Sobre esta tarefa

As amostras simulam os aplicativos de amostra do IBM MQ AMQSPUT e AMQSGET. Eles ilustram as funções a seguir em um ambiente de sistema de mensagens ponto a ponto:

- **HTTPPOST** - Envia solicitações HTTP **POST** em um aplicativo Java para colocar mensagens em uma fila do IBM MQ, usando o IBM MQ bridge for HTTP e manipula as respostas.
- **HTTPDELETE** - Envia solicitações HTTP **DELETE** em um aplicativo Java para obter mensagens de uma fila do IBM MQ, usando o IBM MQ bridge for HTTP e manipula as respostas que contém a mensagem do IBM MQ.

Parâmetros para HTTPPOST e HTTPDELETE



Para executar a amostra **HTTPPOST**, conclua as etapas a seguir:

Procedimento

1. Em uma janela de comandos, navegue até o diretório de amostras HTTP.
2. Execute a amostra **HTTPPOST**.

   ```
   java -classpath . HTTPPOST [parameters]
   ```

   Quando a amostra **HTTPPOST** iniciar, a saída a seguir será exibida:

   ```
   HTTP POST Sample start
   Target server is ' hostName '
   Target queue is ' queueName '
   Target context-root is ' contextRoot '
   ```

3. No prompt de comandos, digite o texto que deseja para formar o corpo de sua mensagem.
4. Pressione Enter para postar a mensagem na fila do IBM MQ.
   a) Se desejar enviar outra mensagem, insira mais algum texto.
      O texto forma o corpo de uma segunda mensagem do IBM MQ.
   b) Pressione Enter para postar a mensagem na fila do IBM MQ.
5. Pressione Enter duas vezes para terminar **HTTPPOST**.
   A seguinte saída é exibida:

   ```
   HTTP POST Sample end
   ```

Como proceder a seguir

A amostra **HTTPDELETE** executa um get destrutivo de todas as mensagens colocadas na fila do IBM MQ.

Execute a amostra **HTTPDELETE** concluindo as etapas a seguir:

1. Em uma janela de comandos, navegue até *MQ_INSTALLATION_PATH*/tools/samples. *MQ_INSTALLATION_PATH* representa o diretório no qual o IBM MQ está instalado.
2. Execute a amostra **HTTPDELETE**.

```
java -classpath . HTTPPOST [parameters]
```

Quando a amostra **HTTPDELETE** iniciar, a saída a seguir será exibida:

```
HTTP DELETE Sample start
Target server is ' host:port '
Target queue is ' your queue name '
Target context-root is ' your context-root '
message
message
...
```

Considerações de segurança para o IBM MQ bridge for HTTP

As considerações de segurança da web padrão se aplicam a autenticação de um cliente do navegador da web. A autorização para os recursos do IBM MQ está no nível do usuário que está executando o servlet do IBM MQ bridge for HTTP e não no cliente do navegador da web individual. As contraprestações de segurança padrão do IBM MQ se aplicam ao IBM MQ.

Os dados que fluem de um navegador da web para um aplicativo IBM MQ usando o IBM MQ bridge for HTTP e de volta, levam três etapas:

Conexão do cliente
A partir do navegador para o IBM MQ bridge for HTTP por meio de uma conexão TCP/IP usando HTTP.

Conexão do adaptador de recursos com o IBM MQ
A conexão é do IBM MQ bridge for HTTP para um gerenciador de filas do IBM MQ. É uma conexão do cliente, através de TCP/IP, ou uma conexão local de ligações do IBM MQ. Quando a conexão for estabelecida, a solicitação de HTTP será colocada em uma fila local padrão ou uma fila de transmissão.

A partir da fila local do IBM MQ em um ou mais canais, para a fila de destino.
Aplique as técnicas padrão para proteger filas, tópicos, gerenciadores de filas e canais.

A resposta usa as etapas em reverso.

Conexão do cliente

Assegure as conexões entre clientes HTTP e o servidor de aplicativos usando o contêiner da web. Use as técnicas do servidor HTTP padrão, como ao usar HTTPS. Consulte a documentação de seu servidor de aplicativos para obter informações.

Conexão do adaptador de recursos com o IBM MQ

A conexão entre o adaptador de recursos e o gerenciador de filas é autorizada usando apenas um único ID do usuário. Designe um único ID do usuário para identificar solicitações a partir do IBM MQ bridge for HTTP. O ID do usuário deve ter restringido às autorizações do IBM MQ apenas para os recursos que os usuários externos devem ter acesso. Deve-se autenticar o cliente real separadamente e estabelecer a confiança para interações sucessivas com o cliente, usando técnicas padrão para segurança da web.

Proteja a conexão entre o adaptador de recursos e o gerenciador de filas usando o ID de usuário único. Ao restringir as autoridades, o ID do usuário não terá mais do que necessário para ler e gravar mensagens em filas e tópicos. O IBM MQ bridge for HTTP é um ponto de ataque entre a Internet e sua intranet.

Como você protege a conexão entre seu adaptador de recursos e o IBM MQ é dependente de seu adaptador de recursos específico. Consulte a documentação para o adaptador de recursos.

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 711.

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 11
O IBM MQ fornece suporte para o .NET, ActiveX, C++, Java e JMS. 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).

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

Informações relacionadas

Visão geral técnica

Referência de desenvolvimento de aplicativos

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)

Idioma	Definições de dados
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 936.

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

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 712
- "Arquivos de módulo Visual Basic" na página 713
- "Arquivos de cópia COBOL" na página 713
- z/OS "Macros de linguagem do montador System/390" na página 714
- z/OS "Arquivos de inclusão PL/I" na página 714

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	QMQM/H
IBM i IBM i	
UNIX	*MQ_INSTALLATION_PATH*/inc/
Sistemas Windows	*MQ_INSTALLATION_PATH*\Tools\c\include

Plataforma	Diretório de instalação ou biblioteca
z/OS	**thlqual**.SCSQC370
z/OS z/OS	

em que *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Nota: Para o UNIX, 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
Outras plataformas UNIX	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i IBM i	QMQM/QCBLLESRC
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 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 “Preparando seu programa para execução” na página 1040). Inclua apenas CMQP se você pretender vincular as chamadas do IBM MQ dinamicamente (consulte “Chamando dinamicamente o stub do IBM MQ” na página 1046). 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 715
- “Conectando-se e desconectando-se de um gerenciador de filas” na página 730
- “Abrindo e fechando objetos” na página 739
- “Colocando mensagens em uma fila” na página 750
- “Obtendo mensagens de uma fila” na página 765
- “Escrevendo aplicativos de publicar/assinar” na página 806
- “Consultando e configurando atributos de objeto” na página 848
- “Confirmando e fazendo backup de unidades de trabalho” na página 851
- “Iniciando aplicativos IBM MQ usando acionadores” na página 863
- “Trabalhando com MQI e clusters” na página 883
- z/OS “Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
- z/OS “Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

“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 47
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

“Construindo um aplicativo processual” na página 1005
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 1055
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 1075
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 serviços da web com o IBM MQ” na página 1306
É possível desenvolver aplicativos IBM MQ para serviços da web usando o transporte IBM MQ para SOAP.

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 "Usando os programas de amostra para z/OS" na página 1182.

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.

IBM MQ for Windows e IBM MQ em sistemas UNIX and Linux também fornecem:

- Chamadas através das quais o IBM MQ for Windows e o IBM MQ em programas de sistemas UNIX and Linux podem confirmar e recuperar 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 1076.
- 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:

Conceitos relacionados

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.

Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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. 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, confirmando ou revertendo as atualizações feitas durante a unidade de trabalho. O controlador de compromisso


IBM i é usado para coordenar unidades globais de trabalho 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

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 “Gerenciamento de transações e serviços do gerenciador de recurso recuperável” na página 856.)

Nota: SRRCMIT e SRRBACK são comandos nativos RRS, não são chamadas MQI.

Chamadas do IBM i

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 do IBM MQ em plataformas Windows, UNIX and Linux

ULW

Os produtos a seguir fornecem as chamadas MQCMIT e MQBACK:

- IBM MQ for Windows
- IBM MQ em sistemas UNIX and Linux

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. Exceto no IBM MQ for z/OS, essa chamada é usada apenas a partir de uma saída de conversão de dados.

Consulte MQXCNVC - Converter caracteres para a sintaxe usada com a chamada MQXCNVC e “Escrevendo saídas de conversão de dados” na página 988 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 711.

Estruturas

As estruturas, usadas com as chamadas MQI listadas em “Chamadas MQI” na página 717, são fornecidas em arquivos de definição de dados para cada uma das linguagens de programação suportadas. IBM i z/OS IBM MQ for z/OS e IBM MQ for IBM i fornecem arquivos que contêm constantes para serem usadas ao preencher alguns dos campos dessas estruturas. Para obter mais informações sobre isso, consulte Definições de dados do IBM MQ.

Consulte Resumo dos tipos de dados da estrutura para obter um resumo das estruturas.

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 1005. Para obter informações sobre como vincular para arquivos de biblioteca C++, consulte Usando C++ *IBM MQ Usando C++*.

AIX *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 89. Arquivos de biblioteca para aplicativos AIX não encadeados

Arquivo de bilioteca	**Ambiente**
libmqm.a	Servidor para C
libmqic.a & **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	Client for C++
libimqs23ia.a	Servidor para C++

Em um ambiente encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 90. 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 bilioteca	**Ambiente**
libmqm_r.a	Servidor para C
libmqic_r.a & **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	Client for C++
libimqs23ia_r.a	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.

HP-UX *IBM MQ for HP-UX*

No IBM MQ for HP-UX, 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.

Plataforma IA64 (IPF)

Em um aplicativo não encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 91. Arquivos de biblioteca para aplicativos HP-UX não encadeados

Arquivo de bilioteca	**Ambiente**
libmqm.so	Servidor para C
libmqic.so & **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
libmqcxa64.so	Interface XA alternativa do cliente
libimqi23ah.so	C++
libmqmcbrt.o	IBM MQ biblioteca de tempo de execução para suporte do Micro Focus COBOL
libmqmcb.so	Servidor para COBOL
libmqicb.so	Cliente para COBOL

Em um ambiente encadeado, vincule-se a uma das bibliotecas a seguir:

Tabela 92. Arquivos de biblioteca para aplicativos HP-UX encadeados

Arquivo de bilioteca	**Ambiente**
libmqm_r.so	Servidor para C
libmqmzf_r.so & **libmqm_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
libimqi23ah_r.so	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.

IBM i *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 93. Arquivos de biblioteca para aplicativos IBM i não encadeados	
Arquivo de bilioteca	**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 94. Arquivos de biblioteca para aplicativos IBM i encadeados	
Arquivo de bilioteca	**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 *IBM MQ for Linux*

No IBM MQ for 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 95. Arquivos de biblioteca para aplicativos Linux não encadeados	
Arquivo de bilioteca	**Ambiente**
libmqm.so	Servidor para C
libmqic.so & **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
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 96. Arquivos de biblioteca para aplicativos Linux encadeados	
Arquivo de bilioteca	**Ambiente**
libmqm_r.so	Servidor para C
libmqic_r.so & **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.

Solaris *IBM MQ for Solaris*

No IBM MQ for Solaris, deve-se vincular o seu programa aos arquivos de biblioteca MQI fornecidos para o ambiente no qual você está executando o seu aplicativo além daqueles fornecidos pelo sistema operacional.

Tabela 97. Arquivos de biblioteca para aplicativos do Solaris	
Arquivo de biblioteca	**Ambiente**
libmqm.so	Servidor e cliente para C
libmqmzse.so	Para C
libmqic.so	Cliente para C
libmqmcs.so	Serviços comuns 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
libmqcxa64.so	Interface XA alternativa do cliente
libimqc23as.a	Client for C++
libimqs23as.a	Servidor para C++

Windows *IBM MQ for Windows*

No IBM MQ for Windows, 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:

Tabela 98. Arquivos de biblioteca para aplicativos do Windows	
Arquivo de biblioteca	**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)

Tabela 98. Arquivos de biblioteca para aplicativos do Windows (continuação)	
Arquivo de biblioteca	**Ambiente**
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)
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 580 na seção “Desenvolvendo aplicativos do .NET” na página 535 para obter mais informações.

Estes arquivos são enviados para compatibilidade com liberações anteriores:

```
mqic32.lib
mqic32xa.lib
```

z/OS *Programas stub do IBM MQ for z/OS*

Antes de ser possível executar um programa escrito com o IBM MQ for z/OS, deve-se editar o link para o programa stub fornecido com o IBM MQ for z/OS para o ambiente no qual você está executando o aplicativo.

O programa stub fornece o primeiro estágio do processamento de suas chamadas em solicitações que o IBM MQ for z/OS pode processar.

O IBM MQ for z/OS fornece os programas stub a seguir:

CSQBSTUB
Programa stub para programas z/OS em lote

CSQBRRSI
Programa stub para programas z/OS em lote usando RRS por meio da MQI

CSQBRSTB
Programa stub para programas z/OS em lote usando RRS diretamente

CSQCSTUB
Programa stub para programas CICS

CSQQSTUB
Programa stub para programas IMS

CSQXSTUB
Programa stub para saídas não CICS de enfileiramento distribuído

CSQASTUB
Programa stub para saídas de conversão de dados

Atenção: Se você usar um programa stub diferente daquele listado para um ambiente específico, ele pode ter resultados imprevisíveis.

Nota: Se usar o programa stub do CSQBRSTB, edite o link com ATRSCSS a partir de SYS1.CSSLIB. (SYS1.CSSLIB também é conhecido como *Callable Services Library*). Para obter mais informações sobre RRS, consulte “Gerenciamento de transações e serviços do gerenciador de recurso recuperável” na página 856.

Como alternativa, é possível chamar dinamicamente o stub a partir se seu programa. Essa técnica está descrita em “Chamando dinamicamente o stub do IBM MQ” na página 1046.

No IMS, também pode ser necessário usar um módulo de interface de linguagem especial que é fornecido pelo IBM MQ.

Não execute aplicativos que são editados por link com CSQBSTUB e CSQQSTUB na mesma região MPP do IMS. Isso pode causar problemas, como mensagens DFS3607I ou CSQQ005E. A primeira chamada MQCONN em um espaço de endereço determina qual interface é usada, portanto, transações CSQQSTUB e CSQBSTUB devem ser executadas em diferentes regiões de mensagens do IMS.

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 para 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 *Considerações em lote do z/OS*

z/OS programas em lote que chamam o MQI podem estar tanto no estado de supervisor quanto de problema.

No entanto, devem atender as condições a seguir:

- Elas devem estar em modo de tarefa, não no modo de bloco de solicitação de serviço (SRB).
- Elas devem estar no modo de controle de espaço de endereço principal (ASC) (não no modo de ASC de registro de acesso).
- Não devem estar no modo de memória cruzada. O número de espaço de endereço principal (ASN) deve ser igual ao ASN secundário e ao ASN inicial.
- Eles não devem ser usados como programas de saída MPF.
- Nenhum bloqueio do z/OS pode ser mantido.
- Não pode haver rotinas de recuperação de função (FRRs) na pilha de FRR.
- Qualquer chave de palavra de status do programa (PSW) pode estar em vigor para a chamada MQCONN ou MQCONNX (desde que a chave seja compatível com o uso de armazenamento que está na chave TCB), mas as chamadas subsequentes que usam a manipulação de conexões retornada por MQCONN ou MQCONNX:
  - Devem ter a mesma chave PSW que foi usada na chamada MQCONN ou MQCONNX
  - Devem ter parâmetros acessíveis (para gravar, quando apropriado) sob a mesma chave PSW
  - Devem ser emitidas sob a mesma tarefa (TCB), mas não em qualquer subtarefa da tarefa
- Podem estar no modo de endereçamento de 24 bits ou de 31 bits. No entanto, se o modo de endereçamento de 24 bits estiver em vigor, endereços de parâmetros devem ser interpretados como endereços válidos de 31 bits.

Se alguma dessas condições não for atendida, uma verificação de programa pode ocorrer. Em alguns casos, a chamada falhará e um código de razão será retornado.

Linux UNIX *considerações do UNIX and Linux*

Considerações de que você precisa estar ciente.

Anote os pontos a seguir ao desenvolver aplicativos UNIX and Linux.

Linux UNIX *A chamada de sistema fork em sistemas UNIX and Linux*

Observe estas considerações ao usar uma chamada do sistema fork em aplicativos IBM MQ.

Se seu aplicativo desejar usar `fork`, o processo pai desse aplicativo deverá chamar `fork` antes de fazer quaisquer chamadas IBM MQ , por exemplo, `MQCONN`ou criar um objeto IBM MQ usando **`ImqQueueManager`**.

Se seu aplicativo desejar criar um processo-filho após fazer quaisquer chamadas de IBM MQ , o código do aplicativo deverá usar um `fork()` com `exec()` para assegurar que o filho seja uma nova instância e não uma cópia exata do pai.

Se seu aplicativo não usar `exec()`, a chamada API IBM MQ feita dentro do processo-filho retornará MQRC_ENVIRONMENT_ERROR.

Linux UNIX *Manipulação de sinais do UNIX and Linux*

Isso não se aplica ao IBM MQ for z/OS ou IBM MQ for Windows.

Em geral, os sistemas UNIX, Linux e IBM i foram movidos 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.

UNIX *Aplicativos não encadeados*

Não aplicável no Solaris já que todos os aplicativos são considerados encadeados mesmo se eles usarem somente um único encadeamento.

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 UNIX *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 UNIX 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)

Considerações adicionais ao usar a manipulação de sinal do UNIX na MQI
Observe estas considerações ao usar a manipulação de sinal do UNIX.

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 733.

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.

Manipuladores de saída de VMS

Os usuários podem instalar manipuladores de saída para um aplicativo IBM MQ usando o serviço de sistema **SYS$DCLEXH**.

O manipulador de saída recebe controle quando uma imagem sai. Uma saída de imagem normalmente ocorre ao chamar o serviço Exit ($EXIT) ou Force Exit ($FORCEX). $FORCEX interrompe o processo de destino no modo de usuário. Em seguida, todos os manipuladores de saída de modo de usuário (estabelecidos por $DCLEXH) começam a executar na ordem inversa de estabelecimento. Para obter mais detalhes sobre os manipuladores de saída e $FORCEX, consulte o *Manual de conceitos de programação de VMS* e o *Manual do VMS System Services*.

Se você chamar uma função MQI a partir de um manipulador de saída, o comportamento da função depende da maneira que a imagem foi finalizada. Se a imagem tiver sido finalizada enquanto outra função MQI estava ativa, um `MQRC_CALL_IN_PROGRESS` será retornado.

É possível chamar uma função MQI a partir de um manipulador de saída se nenhuma outra função MQI estiver ativa e upcalls forem desativados para o aplicativo IBM MQ. Se upcalls estiverem ativados para o aplicativo IBM MQ, ele falha com o código de razão `MQRC_HCONN_ERROR`.

O escopo de uma chamada MQCONN ou MQCONNX é geralmente o encadeamento que o emitiu. Se upcalls forem ativados, o manipulador de saída é executado como um encadeamento separado e as manipulações de conexões não podem ser compartilhadas.

Manipuladores de saída são iniciados no contexto interrompido do processo de destino. Depende do aplicativo assegurar se as ações tomadas por um manipulador sejam seguras e confiáveis, para o contexto interrompido de forma assíncrona do qual são chamada.

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 “Pontos de sincronização em aplicativos IMS” na página 855.

Os programas IMS podem se conectar, consecutivamente ou simultaneamente, a vários gerenciadores de filas no mesmo TCB.

z/OS CICS Transaction Server for 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 732
- “Conectando-se a um gerenciador de filas usando a chamada MQCONNX” na página 733
- “Desconectando programas de um gerenciador de filas usando MQDISC” na página 738

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Abrindo e fechando objetos” na página 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765

Use estas informações para aprender como obter mensagens de uma fila.

"Consultando e configurando atributos de objeto" na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

"Confirmando e fazendo backup de unidades de trabalho" na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"Usando e escrevendo aplicativos no IBM MQ for z/OS" na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

"Os aplicativos IMS e IMS bridge no IBM MQ for z/OS" na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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:

- Para o IBM MQ for z/OS, no ambiente em lote, o gerenciador de filas padrão é especificado no módulo CSQBDEFV.
- Para sistemas IBM MQ for Windows, IBM i, UNIX e Linux, o gerenciador de filas padrão é especificado no arquivo mqs.ini.

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.

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 740).

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 737 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.[8]

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 do grupo de filas compartilhadas nos z/OS de filas compartilhadas.

A saída de MQCONNX é:

- Uma manipulação de conexões (Hconn)
- Um código de conclusão
- Um código de razão

A manipulação de conexões é usada em chamadas MQI subsequentes.

Uma descrição de todos os parâmetros de MQCONNX é fornecida em MQCONNX. O campo *Options* permite que sejam configurados STANDARD_BINDING, FASTPATH_BINDING, SHARED_BINDING ou ISOLATED_BINDING para qualquer versão do MQCNO. Também é possível fazer conexões compartilhadas (independente de encadeamento) usando uma chamada MQCONNX. Consulte “Conexões compartilhadas (independentes de encadeamento) com MQCONNX” na página 737 para obter mais informações sobre estas.

MQCNO_STANDARD_BINDING

Por padrão, MQCONNX (como MQCONN) implica em dois encadeamentos lógicos no qual o aplicativo IBM MQ e o agente do gerenciador de filas locais são executados em processos separados. O aplicativo IBM MQ solicita os serviços da operação IBM MQ e o agente do gerenciador de filas locais a solicitação. Isso é definido pela opção MQCNO_STANDARD_BINDING na chamada MQCONNX.

[8] Ao usar aplicativos multiencadeados com sistemas IBM MQ on UNIX and Linux , é necessário assegurar que os aplicativos tenham um tamanho de pilha suficiente para os 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).

Se você especificar MQCNO_STANDARD_BINDING, a chamada MQCONNX usará MQCNO_SHARED_BINDING ou MQCNO_ISOLATED_BINDING, dependendo do valor do atributo **DefaultBindType** do gerenciador de filas, que é definido em qm.ini.

Esse é o valor-padrão.

Se estiver vinculando à biblioteca `mqm`, uma conexão do servidor padrão usando o tipo de ligação padrão será tentada primeiro. Se o carregamento da biblioteca do servidor subjacente tiver falhado, uma conexão do cliente será tentada ao invés.

- Se a variável de ambiente MQ_CONNECT_TYPE for especificada, uma das seguintes opções pode ser fornecida para mudar o comportamento de MQCONN ou MQCONNX, se MQCNO_STANDARD_BINDING for especificado. (A exceção a isto é se MQCNO_FASTPATH_BINDING estiver especificado com MQ_CONNECT_TYPE configurado como LOCAL ou STANDARD para permitir que as conexões de atalho façam o downgrade pelo administrador sem uma mudança relacionada com o aplicativo:

Value	Significado
CLIENTE	É tentada apenas uma conexão do cliente.
FASTPATH	Esse valor era suportado em liberações anteriores, mas agora ele será ignorado se for especificado.
LOCAL	É tentada apenas uma conexão do servidor. Conexões de atalho são transferidas por downgrade para uma conexão padrão do servidor.
STANDARD	Suportado para compatibilidade com liberações anteriores. Esse valor agora é tratado como LOCAL.

- Se a variável de ambiente MQ_CONNECT_TYPE não estiver configurada quando MQCONN for chamado, uma conexão padrão de servidor usando o tipo de ligação padrão será tentada. Se o carregamento da biblioteca do servidor tiver falhado, uma conexão do cliente será tentada.

MQCNO_FASTPATH_BINDING

Aplicativos confiáveis significa que o aplicativo IBM MQ e o agente do gerenciador de filas locais se tornam 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. Isso é definido pela opção MQCNO_FASTPATH_BINDING na chamada MQCONNX.

É necessário vincular aplicativos confiáveis para as bibliotecas encadeadas do IBM MQ. Para obter instruções sobre como configurar um aplicativo IBM MQ para executar como confiável, consulte Opções de MQCNO.

Esta opção fornece o melhor desempenho.

Nota: Esta opção compromete a integridade do gerenciador de filas: não há proteção de sobrescrever seu armazenamento. Isso também se aplica se o aplicativo contiver erros que possam ser expostos para mensagens e outros dados no gerenciador de filas. Considere esses problemas antes de usar essa opção.

MQCNO_SHARED_BINDING

Especifique essa opção para que o aplicativo e o agente do gerenciador de filas locais sejam executados em processos separados. Isso mantém a integridade do gerenciador de filas, ou seja, protege o gerenciador de filas de programas com erros. No entanto, o aplicativo e o agente do gerenciador de filas locais compartilham alguns recursos.

Essa opção é intermediária entre MQCNO_FASTPATH_BINDING e MQCNO_ISOLATED_BINDING, tanto em termos de proteger a integridade do gerenciador de filas quanto em termos de desempenho das chamadas do MQI.

MQCNO_SHARED_BINDING será ignorado se o gerenciador de filas não suportar esse tipo de ligação. O processamento continuará, embora a opção não tenha sido especificada.

Se um aplicativo tiver sido conectado ao gerenciador de filas locais usando MQCNO_SHARED_BINDING, o gerenciador de filas poderá ser parado enquanto o aplicativo estiver em execução. Se o gerenciador de filas for reiniciado enquanto o aplicativo ainda estiver em execução, a tentativa de iniciar o gerenciador de filas falhará com o erro AMQ7018 já que o aplicativo ainda estará mantendo os recursos necessários pelo gerenciador de filas.

Para iniciar o gerenciador de filas, deve-se parar o aplicativo.

MQCNO_ISOLATED_BINDING

Especifique essa opção para que o aplicativo e o agente do gerenciador de filas locais sejam executados em processos separados, como para MQCNO_SHARED_BINDING. Nesse caso, no entanto, o processo de aplicativo e o agente do gerenciador de filas locais são isolados uns dos outros, pois não compartilham recursos.

Esta é a opção mais segura para proteger a integridade do gerenciador de filas, mas ela fornece o desempenho mais lento de chamadas MQI.

MQCNO_ISOLATED_BINDING será ignorada se o gerenciador de filas não suportar esse tipo de ligação. O processamento continuará, embora a opção não tenha sido especificada.

MQCNO_CLIENT_BINDING

Especifique essa opção para que o aplicativo tente apenas uma conexão do cliente. Essa opção tem as seguintes limitações:

- z/OS MQCNO_CLIENT_BINDING é ignorado no z/OS.
- MQCNO_CLIENT_BINDING é rejeitado com MQRC_OPTIONS_ERROR se for especificado com qualquer opção de ligação MQCNO que não MQCNO_STANDARD_BINDING.
- MQCNO_CLIENT_BINDING não está disponível para Java porque ele tem seus próprios mecanismos para escolher o tipo de ligação.
- Se a variável de ambiente MQ_CONNECT_TYPE não estiver configurada quando MQCONN for chamado, uma conexão padrão de servidor usando o tipo de ligação padrão será tentada. Se o carregamento da biblioteca do servidor tiver falhado, uma conexão do cliente será tentada.

MQCNO_LOCAL_BINDING

Especifique essa opção para que o aplicativo tente uma conexão do servidor. Se MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING ou MQCNO_SHARED_BINDING também estiverem especificadas, então ao invés a conexão será desse tipo e será documentada nesta seção. Caso contrário, uma conexão do servidor padrão será tentada usando o tipo de ligação padrão. MQCNO_LOCAL_BINDING tem as seguintes limitações:

- z/OS MQCNO_LOCAL_BINDING é ignorado em z/OS.
- MQCNO_LOCAL_BINDING é rejeitado com MQRC_OPTIONS_ERROR se for especificado com qualquer opção de reconexão MQCNO além de MQCNO_RECONNECT_AS_DEF.
- MQCNO_LOCAL_BINDING não está disponível para Java porque ele tem seus próprios mecanismos para escolher o tipo de ligação.
- Se a variável de ambiente MQ_CONNECT_TYPE não estiver configurada quando MQCONN for chamado, uma conexão padrão de servidor usando o tipo de ligação padrão será tentada. Se o carregamento da biblioteca do servidor tiver falhado, uma conexão do cliente será tentada.

z/OS No z/OS essas opções são toleradas, mas apenas um limite padrão de conexão é executado.

z/OS MQCNO Versão 3, for z/OS, permite quatro opções diferentes:

MQCNO_SERIALIZE_CONN_TAG_QSG

Isso permite que um aplicativo solicite que apenas uma instância de um aplicativo seja executada a qualquer momento em um grupo de filas compartilhadas. Isso é obtido registrando o uso de uma tag de conexão com um valor que é especificado ou derivado pelo aplicativo. A tag é uma sequência de caracteres de 128 bytes especificada no MQCNO Versão 3.

MQCNO_RESTRICT_CONN_TAG_QSG

Isso é usado quando um aplicativo consiste em mais de um processo (ou um TCB), cada um dos quais pode se conectar a um gerenciador de filas. A conexão é permitida apenas se não houver uso atual da identificação ou se o aplicativo de pedido estiver dentro do mesmo escopo de processamento. Este é o espaço de endereço do MVS dentro do mesmo grupo de filas compartilhadas que o proprietário da tag.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

Isso é semelhante a MQCNO_SERIALIZE_CONN_TAG_QSG, mas apenas o gerenciador de filas local é interrogado para ver se a identificação solicitada já está em uso.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

Isso é semelhante a MQCNO_RESTRICT_CONN_TAG_QSG, mas apenas o gerenciador de filas local é interrogado para ver se a identificação solicitada já está em uso.

Restrições para aplicativos confiáveis

As restrições a seguir se aplicam a aplicativos confiáveis:

- Deve-se desconectar explicitamente aplicativos confiáveis do gerenciador de filas.
- Deve-se parar aplicativos confiáveis antes de terminar o gerenciador de filas 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.
- No IBM MQ em sistemas UNIX and Linux, deve-se usar mqm como o userID e 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 MQ for 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. Nesse caso, uma abordagem é usar o perfil IBM i trocando funções de API, QSYGETPH, QWTSETP e QSYRLSPH, para mudar temporariamente o usuário atual da tarefa para QMQM enquanto programas MQ são executados. Detalhes dessas funções, juntamente com um exemplo de seu uso, são fornecidos na seção APIs de segurança da *IBM i Referência de API do sistema*.
  2. Não cancele aplicativos confiáveis usando System-Request Option 2 ou finalizando as tarefas nas quais eles estão sendo executados usando ENDJOB.
- No IBM MQ for HP-UX, aplicativos multiencadeados de atalho provavelmente precisarão configurar um tamanho de pilha maior que o padrão. Use um tamanho de 256 KB.
- No IBM MQ for Windows, aplicativos confiáveis de 64 bits não são suportados. Se você tentar executar um aplicativo confiável de 64 bits, ele feito downgrade para uma conexão limite padrão.

- No IBM MQ em sistemas UNIX and Linux, 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.

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: Não suportado no IBM MQ for z/OS.

Nas plataformas IBM MQ diferentes do IBM MQ for z/OS, uma conexão feita com MQCONN está disponível somente 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 737, 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.

Variável de ambiente MQCONNX
Use estas informações para conhecer as opções chamada MQCONNX diferente e como eles são usados com MQ_CONNECT_TYPE. Observe que MQ_CONNECT_TYPE apenas tem qualquer efeito para ligações STANDARD. Para outras ligações, MQ_CONNECT_TYPE é ignorado.

Em IBM MQ for IBM i, IBM MQ for Windows e IBM MQ em sistemas UNIX and Linux, 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 99. 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.

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:

- 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.
- 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.

Exceto no CICS no z/OS, depois de MQDISC ser chamado, a manipulação de conexões (Hconn) não será mais válida e não será possível emitir chamadas MQI adicionais até chamar novamente o MQCONN ou

MQCONNX. 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.

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 759).

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 730.

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 760 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 740
- "Criando filas dinâmicas" na página 748
- "Abrindo filas remotas" na página 748
- "Fechando objetos usando a chamada MQCLOSE" na página 749

Conceitos relacionados

"Visão geral da Message Queue Interface" na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

"Conectando-se e desconectando-se de um gerenciador de filas" na página 730
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 750
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" na página 765
Use estas informações para aprender como obter mensagens de uma fila.

"Consultando e configurando atributos de objeto" na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

"Confirmando e fazendo backup de unidades de trabalho" na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"Usando e escrevendo aplicativos no IBM MQ for z/OS" na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

"Os aplicativos IMS e IMS bridge no IBM MQ for z/OS" na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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 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 programas, 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 733 e "Conexões compartilhadas (independentes de encadeamento) com MQCONNX" na página 737. 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 lote do z/OS
Nos ambientes IMS e de lote, é possível usar o identificador na mesma tarefa, mas não em quaisquer subtarefas.

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 761 em "Listas de distribuição" na página 760.

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 100 na página 742 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 100 na página 742 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 100. 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 100. 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 100. 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)

Notes:

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. Ao usar os 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 no lugar do nome do gerenciador de filas locais no Tabela 100 na página 742.

   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 refere-se 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 884 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) TARGQ(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 800.

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 760.
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 101 na página 746 para ver como a definição desses atributos afeta o tipo de acesso que você receberá quando usar esta opção.

Tabela 101. 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 101 na página 746 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 760.

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 848).

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 44
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 756
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 *Opção de MQOPEN para quiesce do gerenciador de filas*
Se você usar a chamada MQOPEN quando o gerenciador de filas estiver em um estado quiesce, a chamada poderá falhar, dependendo do ambiente que estiver sendo usado.

No ambiente do CICS no z/OS, se você usar a chamada MQOPEN quando o gerenciador de filas estiver em um estado quiesce, a chamada sempre falhará.

Em outros ambientes do z/OS, ambientes do IBM i, de sistemas Windows e de sistemas UNIX and Linux, a chamada falhará quando o gerenciador de filas estiver em quiesce somente se a opção MQOO_FAIL_IF_QUIESCING da chamada MQOPEN for usada.

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 uma fila de clusters, os campos *ResolvedQName* e *ResolvedQMgrName* da estrutura MQOD são preenchidos com os nomes da fila remota e o gerenciador de filas remotas está localizado na definição de fila remota ou com a fila de clusters remotos escolhida.

Use a opção MQOO_RESOLVE_LOCAL_Q da chamada MQOPEN para preencher *ResolvedQName* na estrutura MQOD com o nome da fila local que foi aberta. *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 para a 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 751).

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 conforme conhecida para o gerenciador de filas *locais*.

   Nota: Deixe o campo *ObjectQMgrName* em branco neste caso.
2. No campo *ObjectName* da estrutura MQOD, especifique o nome da fila remota, conforme conhecido para o gerenciador de filas *remotas*. No campo *ObjectQMgrName*, especifique um dos seguintes:
   - 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, conforme é conhecido pelo gerenciador de filas *remotas*.

   Nota: Configure o campo *ObjectQMgrName* para o nome do gerenciador de filas remotas (não pode ser deixado em branco neste caso).

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 100 na página 742.

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.
- 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 manipulaçao de conexões usada para abri-la ou, como alternativa, para os aplicativos CICS no z/OS, é possível especificar a constante MQHC_DEF_HCONN (que possui 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 751
- “Colocando mensagens em uma fila remota” na página 756
- “Configurando propriedades de uma mensagem” na página 756
- “Controlando informações de contexto da mensagem” na página 756
- “Colocando uma mensagem em uma fila usando a chamada MQPUT1” na página 759
- “Listas de distribuição” na página 760
- “Alguns casos em que as chamadas put falham” na página 765

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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 751
- “Definindo mensagens usando a estrutura MQMD” na página 751
- “Especificando opções usando a estrutura MQPMO” na página 752
- “Os dados em sua mensagem” na página 754
- “Efetuando put de mensagens: usando identificadores de mensagens” na página 756

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 do IBM MQ” na página 14 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 760). 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.

Além disso, a maneira que a chamada opera com pontos de sincronização é determinada pela plataforma. O padrão do controle de ponto de sincronização é sim no z/OS; para outras plataformas, é não.

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 44. 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 756.

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 760). 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 764.

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 764 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 763 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 748. 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 757
- "Transmitindo contexto de usuário" na página 757
- "Transmitindo todo o contexto" na página 758
- "Configurando contexto de identidade" na página 758
- "Configurando contexto do usuário" na página 758
- "Configurando todo o contexto" na página 758

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 44
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 747

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 741). 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 752).

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.

Listas de distribuição

Não suportado no IBM MQ for z/OS. As listas de distribuição permitem que você coloque uma mensagem para 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.

V 9.0.0.1 V 9.0.1

Atenção: As listas de distribuição não suportam o uso de filas de alias que apontam para objetos do tópico. Em IBM MQ 9.0.1 e IBM MQ 9.0.0 Fix Pack 1, 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 66 na página 761 mostra como as listas de distribuição funcionam.

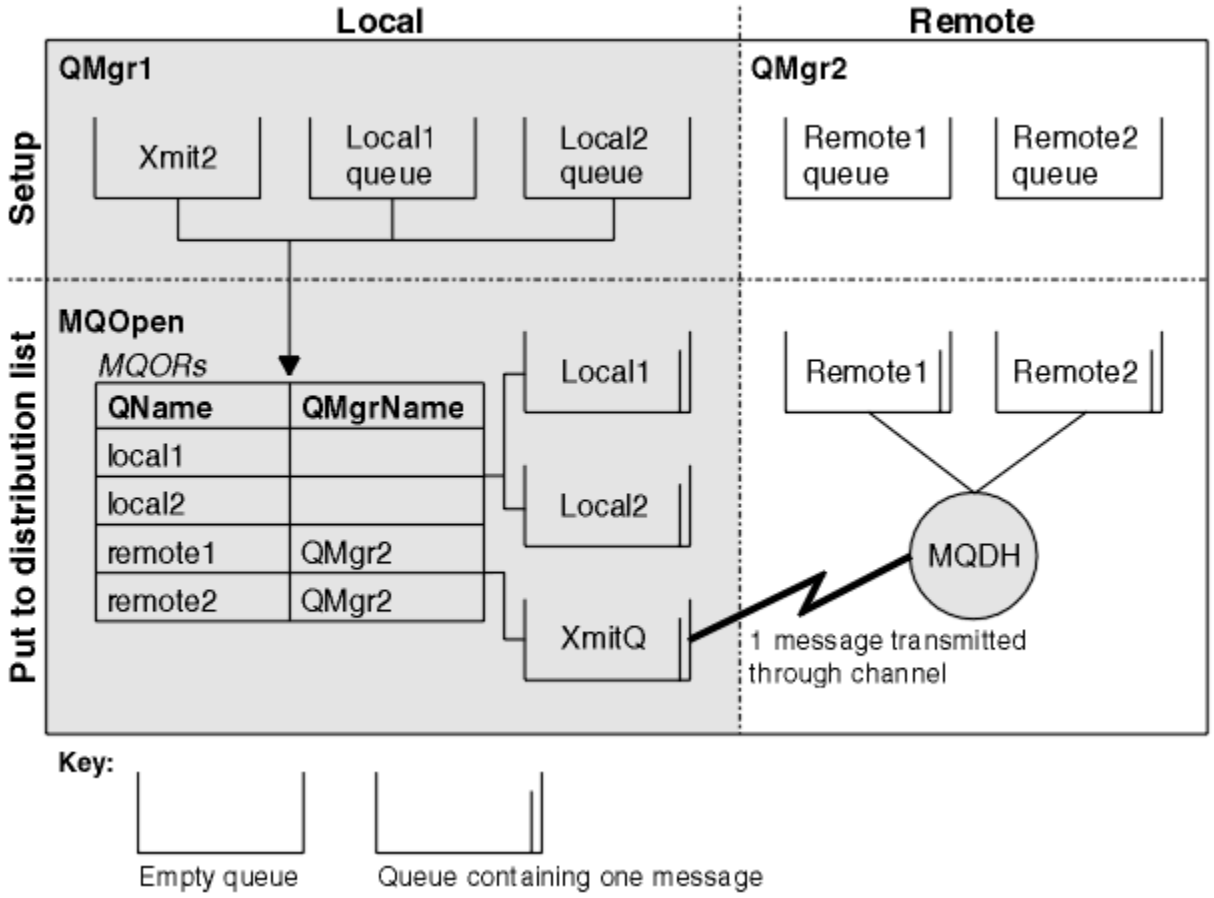


Figura 66. Como as listas de distribuição funcionam

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 750 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 67 na página 763 mostra como é possível abrir uma lista de distribuição em C.

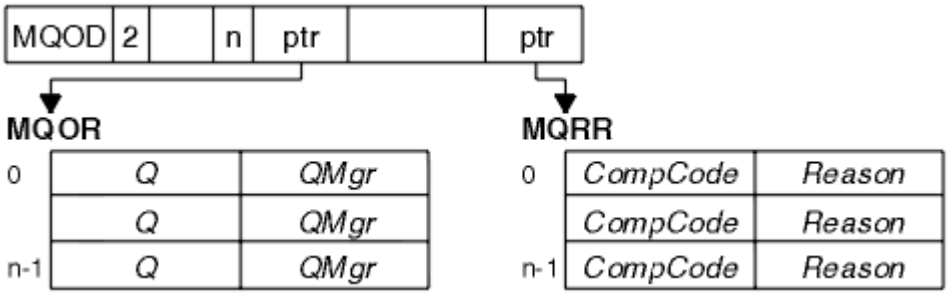


Figura 67. Abrindo uma lista de distribuição em C

Figura 68 na página 763 mostra como é possível abrir uma lista de distribuição em COBOL.

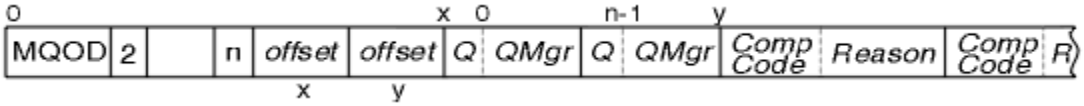


Figura 68. Abrindo uma lista de distribuição em COBOL

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 739 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 750 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 752 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 1107 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 69 na página 764 mostra como é possível colocar uma mensagem em uma lista de distribuição em C.

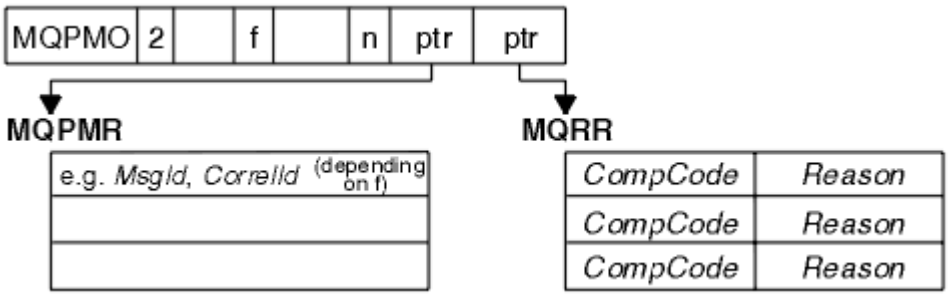


Figura 69. Colocando uma mensagem em uma lista de distribuição em C

Figura 70 na página 764 mostra como é possível colocar uma mensagem em uma lista de distribuição em COBOL.

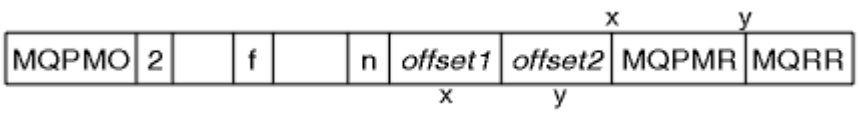


Figura 70. Colocando uma mensagem em uma lista de distribuição em COBOL

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 730 e “Abrindo e fechando objetos” na página 739.

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 766
- “A ordem em que as mensagens são recuperadas de uma fila” na página 771
- “Obtendo uma mensagem específica” na página 782
- “Melhorando o desempenho de mensagens não persistentes” na página 783
- z/OS “Tipo de índice” na página 788
- “Manipulando mensagens com mais de 4 MB de comprimento” na página 788
- “Esperando mensagens” na página 794
- z/OS “Sinalização” na página 795
- “Ignorando restauração” na página 796
- “Conversão de Dados do Aplicativo” na página 799
- “Procurando mensagens em uma fila” na página 800
- “Alguns casos em que a chamada MQGET falha” na página 806

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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 767
- “Descrevendo mensagens usando a estrutura MQMD e a chamada MQGET” na página 767
- “Especificando opções MQGET usando a estrutura MQGMO” na página 768
- “Especificando o tamanho da área de buffer” na página 770

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 do IBM MQ” na página 14 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 782.

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 771 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 794)
- 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 796)
- 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 799)
- 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	Não
Sistemas UNIX and Linux	Não
z/OS z/OS z/OS	Sim
Sistemas Windows	Não

- 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 uma mensagem chegar 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 a 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 794 z/OS e “Sinalização” na página 795 .

Signal1
Signal1 é suportado apenas em z/OS 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 mais informações, consulte “Sinalização” na página 795.

Signal2
O campo *Signal2* é reservado em todas as plataformas e seu valor não é significativo.

z/OS Para obter mais informações, consulte “Sinalização” na página 795.

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. Nesse 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 789.

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 22). 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 782.

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 "Tipo de índice" na página 788.

Ordenação lógica e física

As mensagens em filas podem ocorrer (dentro de cada nível de prioridade) em 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 41. 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 71 na página 772:

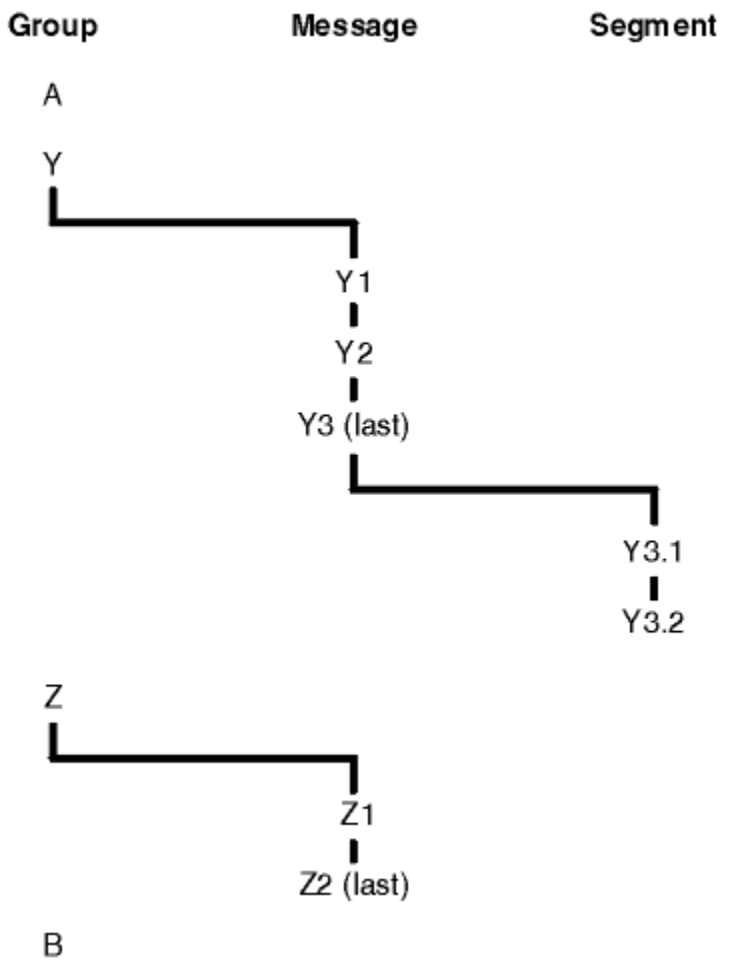


Figura 71. Ordem lógica em uma fila

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 72 na página 773:

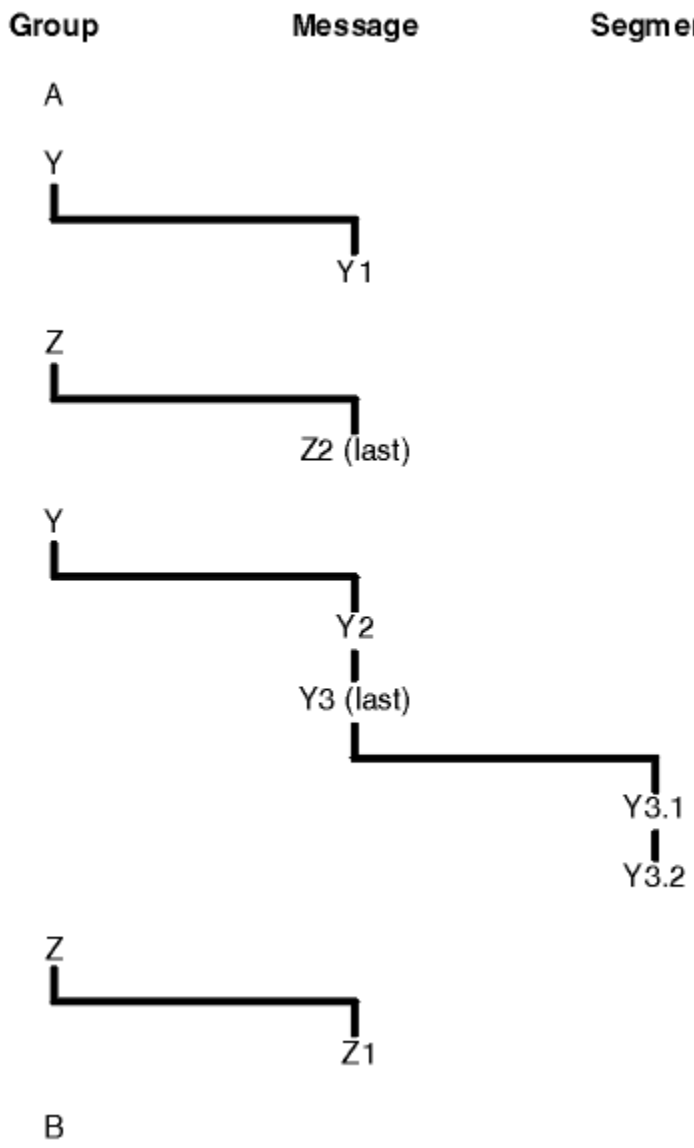


Figura 72. Ordem física em uma fila

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: 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 72 na página 773 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 102. 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	*GroupId*	*MsgSeqNumber*	*Offset*
Sim	Não	Não	Não	Não	Não	MQGI_NONE	1	0
Sim	Não	Não	Sim	Não	Não	Nova ID de grupo	1	0

Tabela 102. 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	Não	Sim	Qualquer um	Não	Não	Nova ID de grupo	1	0
Sim	Não	Sim	Qualquer um	Não	Sim	ID de grupo anterior	1	Deslocamento anterior + comprimento do segmento anterior
Sim	Sim	Qualquer um	Qualquer um	Não	Não	Nova ID de grupo	1	0
Sim	Sim	Qualquer um	Qualquer um	Sim	Não	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
Não	Não	Não	Não	Qualquer um	Qualquer um	MQGI_NONE	1	0
Não	Não	Não	Sim	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	1	0
Não	Não	Sim	Qualquer um	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	1	Valor no campo
Não	Sim	Não	Qualquer um	Qualquer um	Qualquer um	Novo ID do grupo se MQGI_NONE, caso contrário valor no campo	Valor no campo	0
Não	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 103 na página 778 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 103. 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 41.

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 791 e “Colocando e obtendo um grupo que abrange unidades de trabalho” na página 780.

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 104 na página 782 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 104. 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 105 na página 783 mostra qual mensagem é recuperada para as configurações possíveis desses campos.

Tabela 105. 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

Notes:

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 771 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 das operações MQGET na fila. Para obter mais informações, consulte "Tipo de índice" na página 788.

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 27.

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:

- Tanto o cliente quanto o gerenciador de filas remotas devem estar no IBM WebSphere MQ 7 ou mais recente.
- 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 787

Se você estiver usando AIX 5.3 ou posterior, considere configurar seu parâmetro de ajuste para usar o desempenho total para mensagens não persistentes.

Tarefas relacionadas

"Ativando e desativando leia mais adiante" na página 786
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 785
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:

- Tanto o cliente quanto o gerenciador de filas remotas devem estar no IBM WebSphere MQ 7 ou mais recente.
- 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 106. Opções de MQGET e leia mais adiante

	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]	Encoding 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_MSG • 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:

- Tanto o cliente quanto o gerenciador de filas remotas devem estar no IBM WebSphere MQ 7 ou mais recente.
- 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 AIX 5.3 ou posterior, considere configurar seu parâmetro de ajuste para usar o desempenho total 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 principalmente ao executar em máquinas com múltiplos processadores, é altamente recomendável configurar AIXTHREAD_SCOPE=S no ID mqm `.profile` ou configurar AIXTHREAD_SCOPE=S no ambiente antes de iniciar o aplicativo, para obter um desempenho melhor e um 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
```

Tipo de índice

O atributo da fila, *IndexType*, especifica o tipo de índice que o gerenciador de filas mantém para aumentar a velocidade de operações MQGET na fila.

Nota: Suportado somente no IBM MQ for z/OS.

Você tem cinco opções:

Valor	Descrição
NONE	Nenhum índice é mantido. Use isso ao recuperar mensagens sequencialmente (consulte “Priority” na página 771).
GROUPID	Um índice de identificadores de grupo é mantido. Deve-se usar esse tipo de índice quando se deseja a ordenação lógica de grupos de mensagens (consulte “Ordenação lógica e física” na página 771).
MSGID	Um índice dos identificadores de mensagem é mantido. Use isso ao recuperar mensagens usando o campo *MsgId* como um critério de seleção na chamada MQGET (consulte “Obtendo uma mensagem específica” na página 782).
MSGTOKEN	Um índice de tokens de mensagem é mantido.
CORRELID	Um índice de identificadores de correlação é mantido. Use isso ao recuperar mensagens usando o campo *CorrelId* campo como um critério de seleção na chamada MQGET (consulte “Obtendo uma mensagem específica” na página 782).

Nota:

1. Se estiver indexando usando a opção MSGID ou a opção CORRELID, configure os parâmetros **MsgId** ou **CorrelId** relativos no MQMD. Não é benéfico configurar ambas.
2. A procura usa o mecanismo de índice para localizar uma mensagem se uma fila corresponder a todas as condições a seguir:
   - Tem tipo de índice MSGID, CORRELID ou GROUPID
   - É procurado com o mesmo tipo de id
   - Tem mensagens somente de uma prioridade
3. Evite filas (indexadas por *MsgId* ou *CorrelId*) que contêm milhares de mensagens porque isso afeta o tempo de reinício. (Isso não se aplica a mensagens não persistentes, pois elas são excluídas na reinicialização.)
4. MSGTOKEN é usado para definir filas gerenciadas pelo gerenciador de carga de trabalho do z/OS.

Para obter uma descrição completa do atributo **IndexType**, consulte IndexType. Para obter informações adicionais sobre o atributo **IndexType**, consulte “Considerações de design e desempenho de aplicativos z/OS” na página 58.

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. Nos sistemas IBM MQ diferentes do IBM MQ for z/OS, 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.

Se você estiver manipulando mensagens grandes, será possível alterar esses atributos independentemente em plataformas diferentes do z/OS. É possível configurar o valor de atributo do gerenciador de filas no intervalo de 32768 bytes até 100 MB.

Atenção: No IBM MQ for z/OS , o atributo **MaxMsgLength** do gerenciador de filas é codificado permanentemente em 100 MB

Em todas as plataformas, é possível configurar o valor do atributo de fila na faixa de 0 até 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 no IBM MQ for z/OS ou por aplicativos que utilizam o IBM MQ classes for JMS.

Aumentar o comprimento máximo da mensagem, conforme explicado no tópico "Aumentar o comprimento máximo da mensagem" na página 789 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 41.

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).

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 780.

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.

Mensagens de referência
Use estas informações para saber mais sobre as mensagens de referência.

Nota: Não suportado no IBM MQ for z/OS.

Esse método permite 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 fazer isso, você especifica 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 983.

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 1076.

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 com que seu programa aguarde muito tempo, portanto use esta constante com cautela. Aplicativos IMS não devem especificar um intervalo de espera ilimitado porque isso evitaria que o sistema IMS finalizasse. (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 “Sinalização” na página 795). Para obter mais informações sobre as circunstâncias nas quais essas ações ocorrem, consulte MQGMO.

Sinalização

Sinalização é suportada somente no IBM MQ for z/OS.

Sinalização é uma opção na chamada MQGET para permitir que o sistema operacional notifique (ou *sinalize*) um programa quando uma mensagem esperada chega em uma fila. É semelhante à função *get with wait* descrita no tópico “Esperando mensagens” na página 794 porque permite que seu programa continue com outro trabalho enquanto espera o sinal. No entanto, se você usar sinalização, será possível liberar o encadeamento de aplicativos e contar com o sistema operacional para notificar o programa quando uma mensagem chegar.

Para configurar um sinal

Para configurar um sinal, faça o seguinte na estrutura MQGMO usada em sua chamada MQGET:

1. Configure a opção MQGMO_SET_SIGNAL no campo *Options*.
2. Configure a vida útil máxima do sinal no campo *WaitInterval*. Isso configura o período de tempo (em milissegundos) que deseja que o IBM MQ para monitore a fila. Use o valor MQWI_UNLIMITED para especificar uma vida ilimitada.

   Nota: Os aplicativos IMS não devem especificar um intervalo de espera ilimitado, pois isso evitaria que o sistema IMS fosse finalizado. (Quando o IMS é finalizado, ele requer que todas as regiões dependentes sejam finalizadas.) Em vez disso, os aplicativos IMS podem examinar o estado do ECB em intervalos regulares (consulte a etapa 3). Um programa pode ter sinais configurados em vários identificadores de filas ao mesmo tempo:
3. Especifique o endereço do *Event Control Block* (ECB) no campo *Signal1* campo. Isso o notifica sobre o resultado de seu sinal. O armazenamento ECB deve permanecer disponível até que a fila seja fechada.

Nota: Não é possível usar a opção MQGMO_SET_SIGNAL com a opção MQGMO_WAIT.

Quando a mensagem chega

Quando uma mensagem adequada chega, um código de conclusão é retornado ao ECB.

O código de conclusão descreve um dos seguintes:

- A mensagem para a qual você configurou o sinal para chegou na fila. A mensagem não está reservada para o programa que solicitou um sinal, portanto, o programa deve emitir uma chamada MQGET novamente para obter a mensagem.

  Nota: Outro aplicativo pode obter a mensagem no tempo entre o recebimento do sinal e a emissão de outra chamada MQGET.
- O intervalo de espera configurado expirou e a mensagem para a qual você configurou o sinal não chegou na fila. O IBM MQ cancelou o sinal.
- O sinal foi cancelado. Isso ocorre, por exemplo, se o gerenciador de filas parar ou o atributo da fila for mudado, de forma que as chamadas MQGET não serão mais permitidas.

Quando uma mensagem adequada já estiver na fila, a chamada MQGET será concluída da mesma maneira que uma chamada MQGET sem sinalização. Além disso, se um erro for detectado imediatamente, a chamada será concluída e os códigos de retorno serão configurados.

Quando a chamada é aceita e nenhuma mensagem está imediatamente disponível, o controle é retornado ao programa para que ele possa continuar com outro trabalho. Nenhum dos campos de saída no descritor de mensagens são configurados, mas o parâmetro **CompCode** é configurado para MQCC_WARNING e o parâmetro **Reason** é configurado para MQRC_SIGNAL_REQUEST_ACCEPTED.

Para obter informações sobre o que o IBM MQ pode retornar para seu aplicativo quando faz uma chamada MQGET usando sinalização, consulte MQGET.

Se o programa não tiver nenhum outro trabalho para fazer enquanto está esperando o ECB ser postado, ele pode esperar o ECB usando:

- Para o programa CICS Transaction Server for z/OS, o comando EXEC CICS WAIT EXTERNAL
- Para programas em lote e IMS, a macro z/OS WAIT

Se o estado da fila ou do gerenciador de filas mudar enquanto o sinal estiver configurado (ou seja, o ECB ainda não foi postado), as ações a seguir ocorrem:

- Se o gerenciador de filas entrar no estado quiesce e você usou a opção MQGMO_FAIL_IF_QUIESCING, o sinal será cancelado. O ECB é postado com o código de conclusão MQEC_Q_MGR_QUIESCING. Sem essa opção, o sinal permanece configurado.
- Se o gerenciador de filas for forçado a parar ou for cancelado, o sinal será cancelado. O sinal é entregue com o código de conclusão MQEC_WAIT_CANCELED.
- 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 sejam inibidas, o sinal será cancelado. O sinal é entregue com o código de conclusão MQEC_WAIT_CANCELED.

Nota:

1. Se mais de um programa tiver configurado um sinal 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. As regras que o gerenciador de filas segue ao decidir quais aplicativos ativar são as mesmas que para aplicativos em espera; para obter mais informações, consulte a descrição do campo *Options* da estrutura MQGMO em MQGMO - Opções de get-message.
2. Se houver mais de uma chamada MQGET esperando a mesma mensagem, com uma combinação de opções de espera e de sinal, cada chamada em espera será considerada igualmente. Para obter mais informações, consulte a descrição do campo *Options* da estrutura MQGMO em MQGMO – Opções de get-message.
3. Sob algumas condições, é possível que uma chamada MQGET recupere uma mensagem e que um sinal (resultante da chegada da mesma mensagem) seja entregue. Isso significa que quando o programa emite outra chamada MQGET (porque o sinal foi entregue), pode não haver nenhuma mensagem disponível. Projete seu programa para testar para essa situação.

Para obter informações sobre como configurar um sinal, consulte a descrição da opção MQGMO_SET_SIGNAL e do campo *Signal1* em Signal1.

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.

Nota: Suportado somente no IBM MQ for z/OS.

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 73 na página 798 ilustra uma sequência típica de etapas que um programa de aplicativo pode conter quando uma solicitação MQGET é necessária para ignorar a restauração.

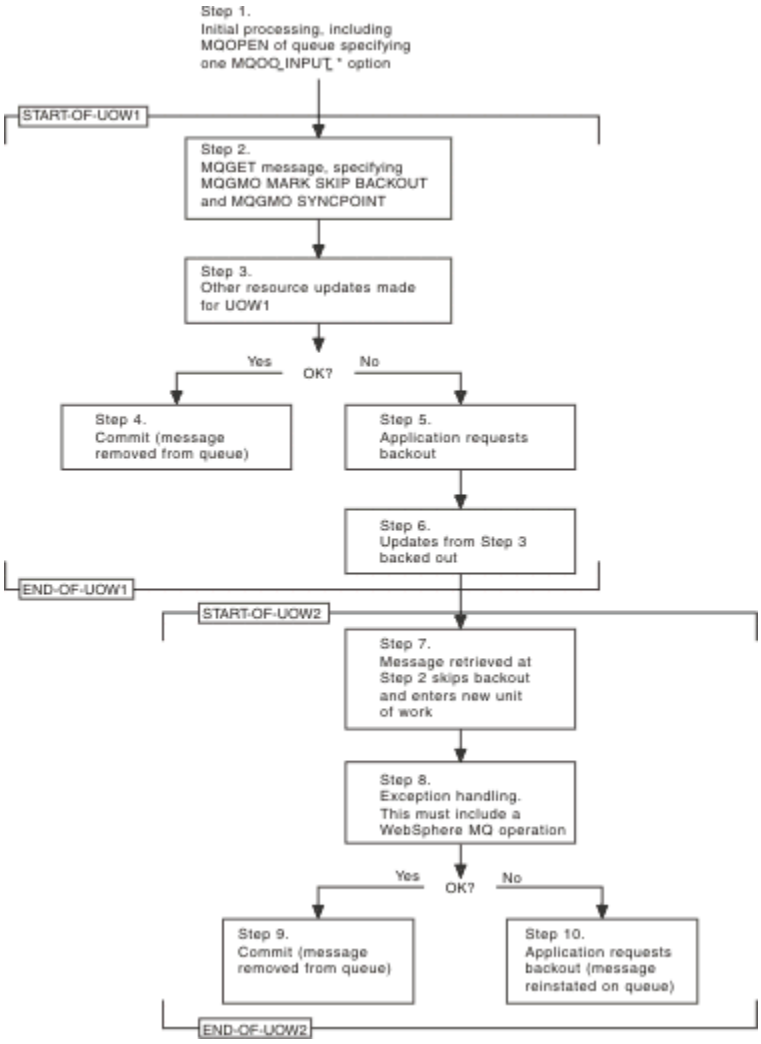


Figura 73. Ignorando a restauração usando MQGMO_MARK_SKIP_BACKOUT

As etapas em Figura 73 na página 798 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 73 na página 798, 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 988. 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 771.)

- "Filas na sequência FIFO (primeira a entrar, primeira a sair)" na página 801
- "Filas em sequência de prioridade" na página 801
- "Mensagens Não Confirmadas" na página 801
- "Mudança na sequência da fila" na página 802
- "Usando o índice da fila" na página 802

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

Ao procurar em uma fila indexada que contenha somente mensagens de uma única prioridade (persistentes, não persistentes ou ambas), o gerenciador de filas usa o índice para procurar quando determinadas formas de procura são usadas.

Nota: Suportado somente no IBM MQ for z/OS.

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, chame MQGET novamente, mas no campo *Options* da estrutura MQGMO, especifique MQGMO_MSG_UNDER_CURSOR.Neste caso, a chamada 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 771 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 803, 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 807
- “Escrevendo aplicativos de assinante” na página 814
- “Ciclos de vida de publicar/assinar” na página 831
- “Propriedades de mensagem de publicação/assinatura” na página 836
- “Ordenação de mensagens” na página 837
- “Interceptando publicações” na página 838
- “Opções de publicação” na página 846
- “Opções de Assinatura” na página 846

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 6

É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

"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 47
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 715
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

"Construindo um aplicativo processual" na página 1005
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 1055
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 1075
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 serviços da web com o IBM MQ" na página 1306
É possível desenvolver aplicativos IBM MQ para serviços da web usando o transporte IBM MQ para SOAP.

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.

Gravar um aplicativo publicador simples do IBM MQ é como gravar um aplicativo ponto a ponto do IBM MQ que coloca mensagens em uma fila (Tabela 107 na página 807). A diferença é você MQPUT mensagens para um tópico, não para uma fila.

Tabela 107. Padrão do programa IBM MQ ponto a ponto versus de publicação/assinatura.

Salto	**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 808), 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 811), 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 814
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.

Informações relacionadas

DEFINE TOPIC
DISPLAY TOPIC
DISPLAY TPSTATUS

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 75 na página 809

```
#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 74. 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 75. 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 abrangem um espectro a partir de unicamente usar tópicos fixos, como neste exemplo, e tópicos variáveis, como no próximo. O “Exemplo 2: publicador em um tópico de variável” na página 811 também demonstra combinar o uso de tópicos e sequências de caracteres de tópicos.

Conceitos relacionados

“Exemplo 2: publicador em um tópico de variável” na página 811
Um programa do WebSphere MQ para ilustrar a publicação em um tópico definido programaticamente.

“Escrevendo aplicativos de assinante” na página 814
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 77 na página 812.

```
#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 76. 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 77. 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 tópicos resolvidos não é necessariamente o mesmo objeto do tópico transmitido para MQOPEN pelo publicador. O IBM MQ usa a árvore implícita na sequência de tópicos resolvidos para resolver 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 78 na página 813). 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 ele significa que não há nenhum nome de tópico e a sequência de tópicos é fornecida inteiramente pelo programa. A saída em Figura 77 na página 812 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

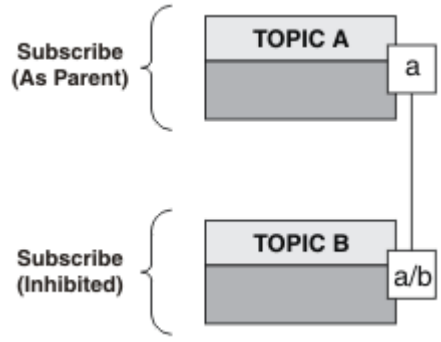


Figura 78. Associações de objeto de tópico

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 808
Um programa IBM MQ para ilustrar a publicação em um tópico definido administrativamente.

"Escrevendo aplicativos de assinante" na página 814
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 108 na página 814 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.
3. No terceiro estilo, o aplicativo assinante opta por abrir e fechar a fila subjacente que é utilizada para as publicações bem como emitir assinaturas para preencher a fila com as publicações.

Uma maneira de entender esses estilos é estudar os programas C de exemplo listados em Tabela 108 na página 814 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 807.

Tabela 108. Padrões de programa IBM MQ ponto a ponto vs. de assinatura.

Salto	**consumidor de mensagens do MQ**	**"Exemplo 1: consumidor de publicação do MQ" na página 815**	**"Exemplo 2: assinante do MQ gerenciado" na página 817**	**"Exemplo 3: assinante do MQ não gerenciado" na página 822**
Conecta-se a um gerenciador de filas	MQCONN	MQCONN	MQCONN	MQCONN
Abrir fila	MQOPEN	MQOPEN		MQOPEN
Assinar			MQSUB	MQSUB

Tabela 108. Padrões de programa IBM MQ ponto a ponto vs. de assinatura. (continuação)				
Salto	**consumidor de mensagens do MQ**	**"Exemplo 1: consumidor de publicação do MQ" na página 815**	**"Exemplo 2: assinante do MQ gerenciado" na página 817**	**"Exemplo 3: assinante do MQ não gerenciado" na página 822**
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 831.

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 822

Nota: O estilo de codificação compacto destina-se à capacidade de leitura e não ao uso na produção.

Os resultados são mostrados em Figura 80 na página 816.

```
#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 79. 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 80. 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 817
O assinante do MQ gerenciado é o padrão preferencial para a maioria dos aplicativos de assinante. O exemplo *não* requer nenhuma definição administrativa de filas, tópicos ou assinaturas.

“Exemplo 3: assinante do MQ não gerenciado” na página 822
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. O exemplo demonstra maneiras diferentes de combinar assinaturas e filas.

“Gravando aplicativos de publicador” na página 807
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. 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 o "Exemplo 2: publicador em um tópico de variável" na página 811.

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 83 na página 820.

```
#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 81. 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 82. 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 83. 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 822.
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 815
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 822
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. O exemplo demonstra maneiras diferentes de combinar assinaturas e filas.

"Gravando aplicativos de publicador" na página 807
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. 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 usar assinaturas compartilhadas, cada aplicativo que está compartilhando a assinatura abre a mesma fila gerenciada e obtém as mensagens dela.

Uma assinatura *gerenciada* é aquela manipulada pelo IBM MQ , que faz os registros e as remoções de registro por você. Já em uma assinatura *não gerenciada*, o aplicativo é responsável por especificar a fila na qual as as assinaturas são armazenadas.

O gerenciador de filas abre implicitamente a fila de assinaturas gerenciadas duráveis para um assinante de tal forma que o processamento compartilhado da fila não seja possível. Além disso, não é possível criar mais de uma assinatura para cada fila gerenciada e você pode achar mais difícil gerenciar as filas, pois tem menos controle sobre os nomes das filas. 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 86 na página 828 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 109 na página 824) se os estados da assinatura e da fila fornecidos como entrada para MQSUB forem inconsistentes com o valor de `sd.Options`. Figura 92 na página 831 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 85 na página 827 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 109 na página 824 que mostra os resultados de passar diferentes identificadores de filas para MQSUB e os resultados de executar o programa de exemplo mostrado em Figura 87 na página 829 para Figura 92 na página 831.

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 109 na página 824, 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 109 na página 824 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 109. Erros de MQSUB com diferentes identificadores de filas e combinações de assinaturas

	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 109. Erros de MQSUB com diferentes identificadores de filas e combinações de assinaturas (continuação)

	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 84. 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 85. 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 86. 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 87 na página 829, 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 87. Publicar 130 em NYSE/IBM/PRICE

Em Figura 88 na página 829, 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 92 na página 831. 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 88. Receber a publicação retida

Em (Figura 89 na página 830), 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 89. Continuar assinatura

No Figura 90 na página 830, 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 90. Receber publicação retida com a nova assinatura não durável não gerenciada

No Figura 91 na página 830, 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 91. Assinaturas sobrepostas

No Figura 92 na página 831, 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 92. Tópicos de Assinatura não podem ser mudados

Conceitos relacionados

“Exemplo 1: consumidor de publicação do MQ” na página 815
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 817
O assinante do MQ gerenciado é o padrão preferencial para a maioria dos aplicativos de assinante. O exemplo *não* requer nenhuma definição administrativa de filas, tópicos ou assinaturas.

“Gravando aplicativos de publicador” na página 807
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[9] 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 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

[9] O publicar deve enviar os registros de passageiros como mensagens persistentes para evitar outras possíveis falhas, é claro.

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 93 na página 833 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.

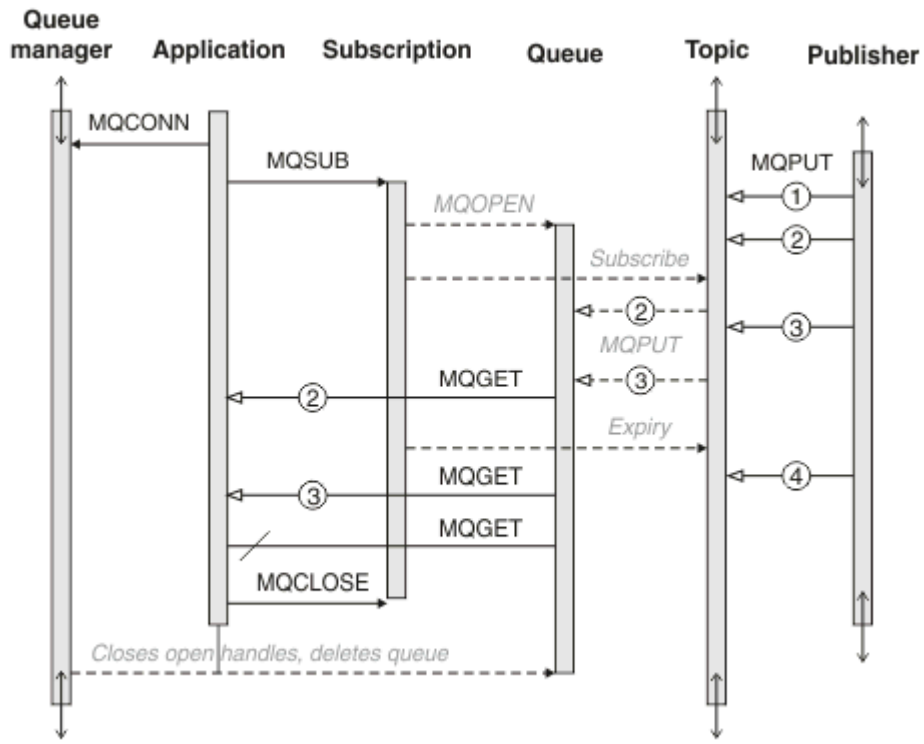


Figura 93. Linhas de vida do assinante gerenciado não durável

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.

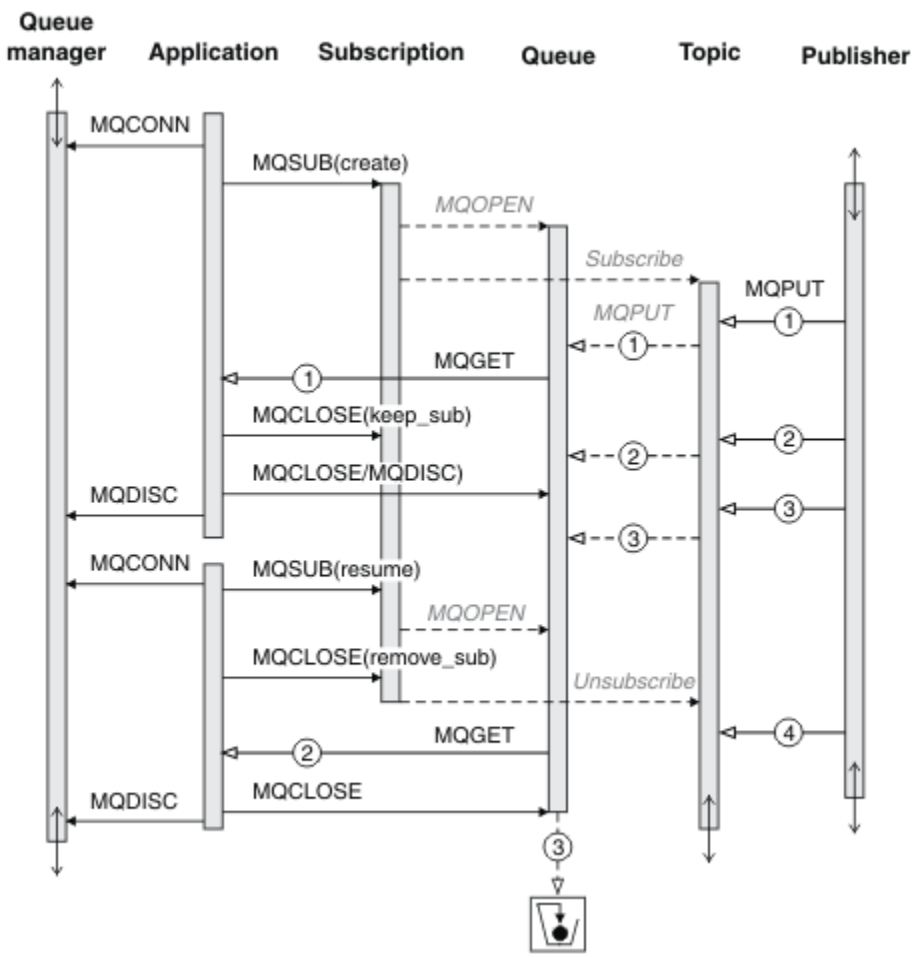


Figura 94. Linhas de vida do assinante gerenciado durável

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.

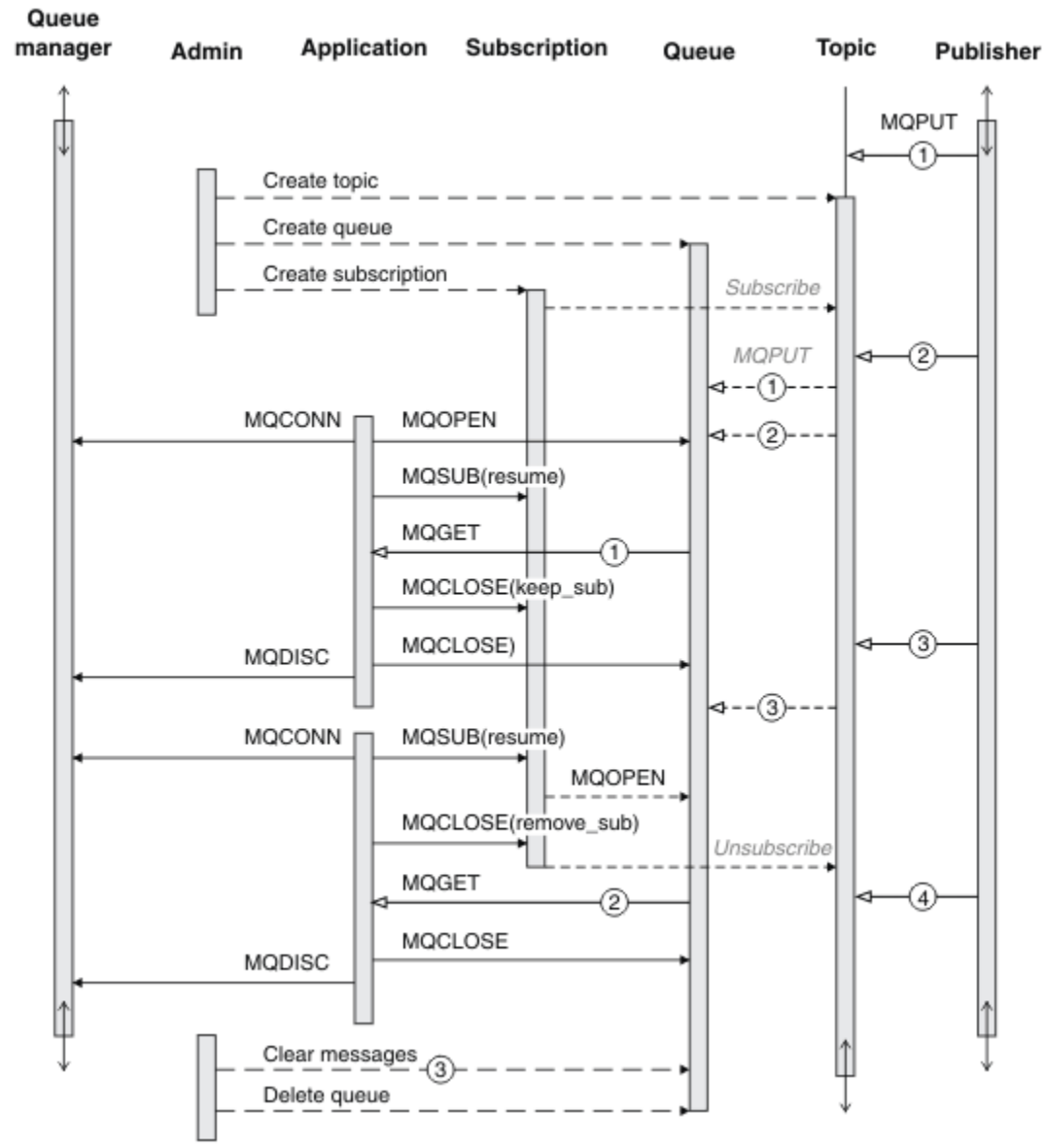


Figura 95. Linhas de vida do assinante não gerenciado durável

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 44. 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 44. 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 110 na página 837 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 110. 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.

Informações 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.

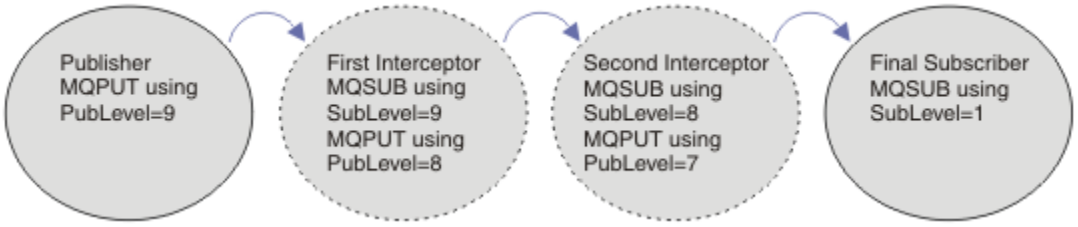


Figura 96. Sequência de assinantes de interceptação

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 111 na página 839.

Tabela 111. Opções de assinatura para assinantes de interceptação

Opção de assinatura	**Notes**
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 111 na página 839 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 112 na página 840 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 “Examples” na página 840.

Tabela 112. Valores de MQPUT para mensagens publicadas novamente	
Publicar mensagem novamente usando MQPUT	**Informações na mensagem de publicação**
MQOD. ObjectString	Propriedade de mensagem MQTopicString
MQPMO. Options	Propriedade de 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 112 na página 840. 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.

Examples

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 97 na página 840 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 97. Diretivas do pré-processador

Figura 98 na página 841 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 98. Declarações

Inicializações que não são facilmente executadas nas declarações são mostradas em Figura 99 na página 842. 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.

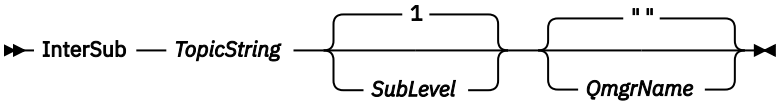
MQPRI_PRIORITY_AS_PUBLISHED
A publicação é colocada na fila de publicação com a mesma prioridade de mensagem que foi publicada.

```
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 99. Inicializações

Figura 100 na página 843 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 100. Preparando para interceptar publicações

O fragmento de código principal, Figura 101 na página 844, 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 101. Interceptar publicação e publicar novamente

O fragmento de código final é mostrado em Figura 102 na página 844.

```
    } 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 102. 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 103 na página 845 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.

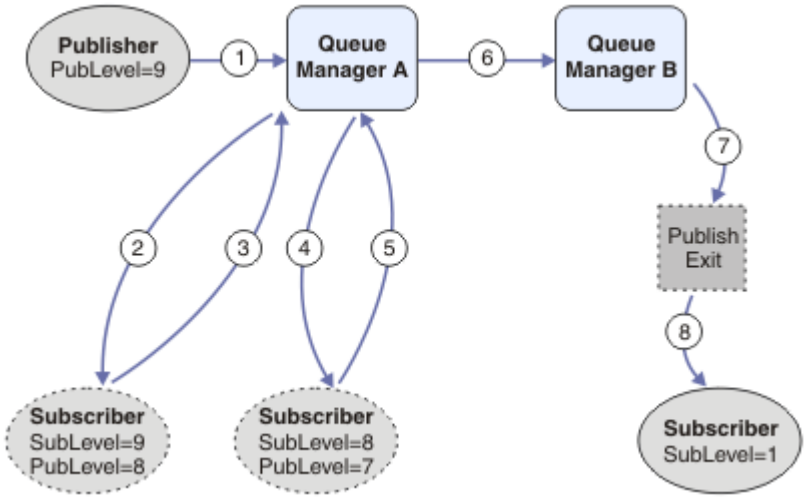


Figura 103. Interceptação e saída Publish em um cluster

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 104 na página 846 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 104 na página 846, o resultado é que a publicação não é interceptada pelo assinante usando `SubLevel` 9 no gerenciador de filas B.

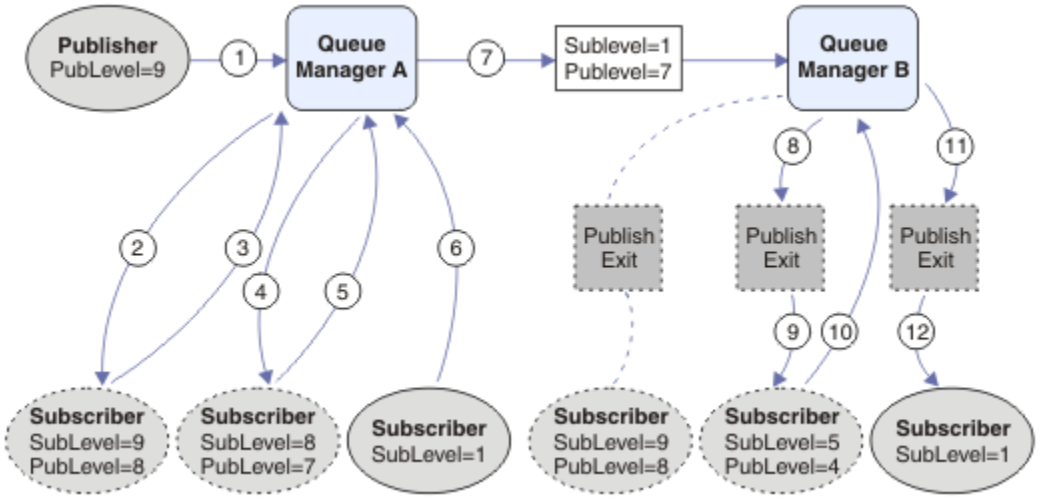


Figura 104. Implementação complexa de assinantes de interceptação

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.

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:

- ULW O recurso MQSC, que é descrito em Referência de 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 113. 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 730 e "Abrindo e fechando objetos" na página 739.

Use os seguintes links para descobrir mais sobre consultar e configurar atributos do objeto:

- "Consultando sobre os atributos de um objeto" na página 849
- "Alguns casos em que a chamada MQINQ falha" na página 850
- "Configurando os atributos da fila" na página 851

Conceitos relacionados

"Visão geral da Message Queue Interface" na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

"Conectando-se e desconectando-se de um gerenciador de filas" na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

"Colocando mensagens em uma fila" na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" na página 765
Use estas informações para aprender como obter mensagens de uma fila.

"Confirmando e fazendo backup de unidades de trabalho" na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"Usando e escrevendo aplicativos no IBM MQ for z/OS" na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

"Os aplicativos IMS e IMS bridge no IBM MQ for z/OS" na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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 884 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)
- *DistList* (não no z/OS)
- *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 853.

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 853
- z/OS “Pontos de sincronização em aplicativos IBM MQ for z/OS” na página 854
- IBM i “Pontos de sincronização em CICS para os aplicativos IBM i” na página 857
- “Pontos de sincronização em IBM MQ for Multiplatforms” na página 857
- IBM i “Interfaces para o gerenciador de ponto de sincronização externa do IBM i” na página 862

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Iniciando aplicativos IBM MQ usando acionadores” na página 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- HP-UX IBM MQ for HP-UX
- Linux IBM MQ for Linux
- Solaris IBM MQ for Solaris
- Windows IBM MQ for Windows
- z/OS CICS Transaction Server for 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:

- IBM i IBM MQ for IBM i
- UNIX IBM MQ em UNIX
- Windows IBM MQ for Windows
- 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 860 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 1075.

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*.

Exceto no lote do z/OS com RRS, se um programa emitir a chamada MQDISC enquanto houver solicitações não confirmadas, ocorre um ponto de sincronização implícito. 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 sob o z/OS UNIX System Services and Language Environment (LE), manipulação de condição padrão é chamada para finalizações anormais de tarefas 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 obter mais informações, consulte Introdução ao Language Environment Condition Handling.

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 796.

Mudanças nos atributos da fila (pela chamada MQSET ou por comandos) não são afetadas pela confirmação ou restauração das unidades de trabalho.

z/OS *Pontos de sincronização em aplicativos IBM MQ for z/OS*

Este tópico explica como usar pontos de sincronização no gerenciador de transações (CICS e IMS) e aplicativos em lote.

z/OS *Pontos de sincronização em aplicativos CICS Transaction Server for z/OS*

Em um aplicativo CICS, você estabelece um ponto de sincronização usando o comando EXEC CICS SYNCPOINT.

Para restaurar todas as mudanças para o ponto de sincronização anterior, é possível usar o comando EXEC CICS SYNCPOINT ROLLBACK. Para obter mais informações, veja o *CICS Application Programming Reference*.

Se outros recursos recuperáveis estiverem envolvidos na unidade de trabalho, o gerenciador de filas (em conjunto com o gerenciador do ponto de sincronização do CICS) participa de um protocolo two-phase commit; caso contrário, o gerenciador de filas executa um processo single-phase commit.

Se um aplicativo CICS emitir a chamada MQDISC, nenhum ponto de sincronização implícito será tomado. Se o aplicativo for fechado normalmente, quaisquer filas abertas serão fechadas e uma confirmação implícita ocorrerá. Se o aplicativo for fechado de forma anormal, quaisquer filas abertas serão fechadas e uma restauração implícita ocorrerá.

z/OS *Pontos de sincronização em aplicativos IMS*

Em um aplicativo IMS, estabeleça um ponto de sincronização usando chamadas IMS, como GU (GET exclusivo) para o IOPCB e CHKP (ponto de verificação).

Para voltar todas as mudanças desde o ponto de verificação anterior, é possível usar a chamada IMS ROLB (rollback). Para obter mais informações, consulte a documentação do IMS.

O gerenciador de filas (em conjunto com o gerenciador de ponto de sincronização do IMS) participa de um protocolo two-phase commit se outros recursos recuperáveis também estiverem envolvidos na unidade de trabalho.

Todos os identificadores abertos são fechados pelo adaptador do IMS em um ponto de sincronização (exceto em um ambiente BMP de lote ou não acionado por mensagem). Isso ocorre porque um usuário diferente poderia iniciar a próxima unidade de trabalho e a verificação de segurança do IBM MQ é executada quando as chamadas MQCONN, MQCONNX e MQOPEN são feitas, não quando as chamadas MQPUT ou MQGET são feitas.

No entanto, em uma ambiente Wait-for-Input (WFI) ou pseudo Wait-for-Input (PWFI), o IMS não notifica o IBM MQ para fechar os identificadores até a próxima mensagem chegar ou um código de status QC ser retornado ao aplicativo. Se o aplicativo estiver esperando na região do IMS e qualquer um desses identificadores pertencer a filas acionadas, o acionamento não ocorrerá porque as filas estão abertas. Por essa razão aplicativos em execução em um ambiente WFI ou PWFI devem efetuar explicitamente MQCLOSE nos identificadores de fila antes de executar GU em IOPCB para a próxima mensagem.

Se um aplicativo IMS (um BMP ou um MPP) emitir a chamada MQDISC, as filas abertas serão fechadas mas nenhum ponto de sincronização implícito será tomado. Se o aplicativo for fechado normalmente, quaisquer filas abertas serão fechadas e uma confirmação implícita ocorrerá. Se o aplicativo for fechado de forma anormal, quaisquer filas abertas serão fechadas e uma restauração implícita ocorrerá.

z/OS *Pontos de sincronização em aplicativos em lote do z/OS*

Para aplicativos em lote, é possível usar as chamadas de gerenciamento do ponto de sincronização do IBM MQ: MQCMIT e MQBACK. Para compatibilidade com versões anteriores, CSQBCMT e CSQBBAK estão disponíveis como sinônimos.

Nota: Se precisar confirmar ou recuperar atualizações em recursos gerenciados por gerenciadores de recursos diferentes, como IBM MQ e Db2, dentro de uma única unidade de trabalho, é possível usar RRS. Para obter informações adicionais, consulte “Gerenciamento de transações e serviços do gerenciador de recurso recuperável” na página 856.

Confirmando mudanças usando a chamada MQCMIT

Como entrada, deve-se fornecer a manipulação de conexões (*Hconn*) que é retornada pela chamada MQCONN ou MQCONNX.

A saída de MQCMIT é um código de conclusão e um código de razão. A chamada será concluída com um aviso se o ponto de sincronização tiver sido concluído, mas o gerenciador de filas tiver recuperado as operações put e get desde o ponto de sincronização anterior.

A conclusão bem-sucedida da chamada MQCMIT indica ao gerenciador de filas que o aplicativo atingiu um ponto de sincronização e que todas as operações put e get feitas desde o ponto de sincronização anterior se tornaram permanentes.

Nem todas as respostas de falha significam que o MQCMIT não foi concluído. Por exemplo, o aplicativo pode receber MQRC_CONNECTION_BROKEN.

Há uma descrição da chamada MQCMIT em MQCMIT.

Recuperando mudanças usando a chamada MQBACK

Como entrada, você deve fornecer um identificador de conexão (*Hconn*). Use o identificador que é retornado pela chamada MQCONN ou MQCONNX.

A saída de MQBACK é um código de conclusão e um código de razão.

A saída indica ao gerenciador de filas que o aplicativo atingiu um ponto de sincronização e que todos gets e puts feitos desde o último ponto de sincronização foram restaurados.

Há uma descrição da chamada MQBACK em MQBACK.

Gerenciamento de transações e serviços do gerenciador de recurso recuperável

Gerenciamento de transação e serviços do gerenciador de recurso recuperável (RRS) é um recurso do z/OS para fornecer suporte ao ponto de sincronização de duas fases entre gerenciadores de recursos participantes.

Um aplicativo pode atualizar os recursos recuperáveis gerenciadas por vários gerenciadores de recursos do z/OS, como IBM MQ e Db2, e, em seguida, confirmar ou recuperar essas atualizações como uma única unidade de trabalho. RRS fornece o status necessário da unidade de trabalho que está se conectando durante a execução normal, coordena o processamento do ponto de sincronização e fornece informações apropriadas de status da unidade de trabalho durante a reinicialização do subsistema.

Suporte de participante do IBM MQ for z/OS RRS permite que aplicativos IBM MQ nos ambientes de lote, TSO e de procedimento armazenado do Db2 atualizem recursos do IBM MQ e não do IBM MQ (por exemplo, Db2) dentro de uma única unidade de trabalho lógica. Para obter informações sobre o suporte ao participante do RRS, consulte *z/OS MVS Programação: Recuperação de Recursos*.

Seu aplicativo IBM MQ pode usar MQCMIT e MQBACK ou as chamadas RRS equivalentes, SRRCMIT e SRRBACK. Consulte o “O adaptador em lote RRS” na página 890 para obter informações adicionais.

Disponibilidade de RRS

Se RRS não estiver ativo em seu sistema z/OS, qualquer chamada do IBM MQ emitida de um programa vinculado com um stub do RRS (CSQBRSTB ou CSQBRRSI) retorna MQRC_ENVIRONMENT_ERROR.

Procedimentos armazenados do Db2

Se você usar procedimentos armazenados do Db2 com RRS, esteja ciente do seguinte:

- Os procedimentos armazenados do Db2 que usam RRS devem ser gerenciados pelo workload manager (gerenciado por WLM).
- Se um procedimento armazenado gerenciado pelo Db2 contiver chamadas do IBM MQ e estiver vinculado a um stub RRS (CSQBRSTB ou CSQBRRSI), a chamada MQCONN ou MQCONNX retornará MQRC_ENVIRONMENT_ERROR.
- Se um procedimento armazenado gerenciado pelo WLM contiver chamadas do IBM MQ e estiver vinculado a um stub não RRS, a chamada MQCONN ou MQCONNX retornará MQRC_ENVIRONMENT_ERROR, a menos que ela seja a primeira chamada do IBM MQ executada desde que o espaço de endereço do procedimento armazenado foi iniciado.

- Se seu procedimento armazenado do Db2 contiver chamadas do IBM MQ e estiver vinculado a um stub não RRS, os recursos do IBM MQ atualizados nesse procedimento armazenado não serão confirmados até que o espaço de endereço do procedimento armazenado ser finalizado ou até um procedimento armazenado subsequente executar um MQCMIT (usando um stub Lote/TSO do IBM MQ).
- Várias cópias do mesmo procedimento armazenado podem ser executadas simultaneamente no mesmo espaço de endereço. Assegure que seu programa esteja codificado de maneira reentrante se desejar que o Db2 use uma única cópia de seu procedimento armazenado. Caso contrário, você pode receber MQRC_HCONN_ERROR em qualquer chamada do IBM MQ em seu programa.
- Não codifiquer MQCMIT ou MQBACK em um procedimento armazenado do Db2 gerenciador por WLM.
- Projete todos os programas para serem executados em 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

ULW 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

V 9.0.5

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 Multiplatforms
Use unidades de trabalho globais quando também precisar incluir atualizações nos recursos pertencentes a outros gerenciadores de recursos.

Aqui, a coordenação pode ser interna ou externas para o gerenciador de filas:

Coordenação de ponto de sincronização interno

A coordenação do gerenciador de filas de unidades de trabalho globais não é suportada pelo IBM MQ for IBM i ou pelo IBM MQ for z/OS. Ela não é suportada em um ambiente do IBM MQ MQI client.

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, o IBM MQ em sistemas UNIX and Linux e o IBM MQ for Windows registram seu interesse no resultado da unidade de trabalho com o coordenador do ponto de sincronização para que possam confirmar ou retroceder quaisquer operações get ou put 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 V 9.0.5 *Ponto de sincronização implícito no Multiplatforms*

O IBM MQ 9.0.5 inclui um ponto de sincronização implícito para mensagens persistentes colocadas 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. O gerenciador de recursos para o IBM MQ no UNIX, Linux, and Windows é denominado MQSeries_XA_RMI. IBM i Para 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 114. 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 737).
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. 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 controle de compromisso nativo do IBM i como um coordenador do ponto de sincronização externa.

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 for 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 fornecido pelo atributo local-queue do *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 UNIX, 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 de 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 870.

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 874).

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 881).

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 “Monitores de Acionadores” na página 878.

Para entender como o acionamento funciona, considere Figura 105 na página 865, que é um exemplo do tipo de acionador FIRST (MQTT_FIRST).

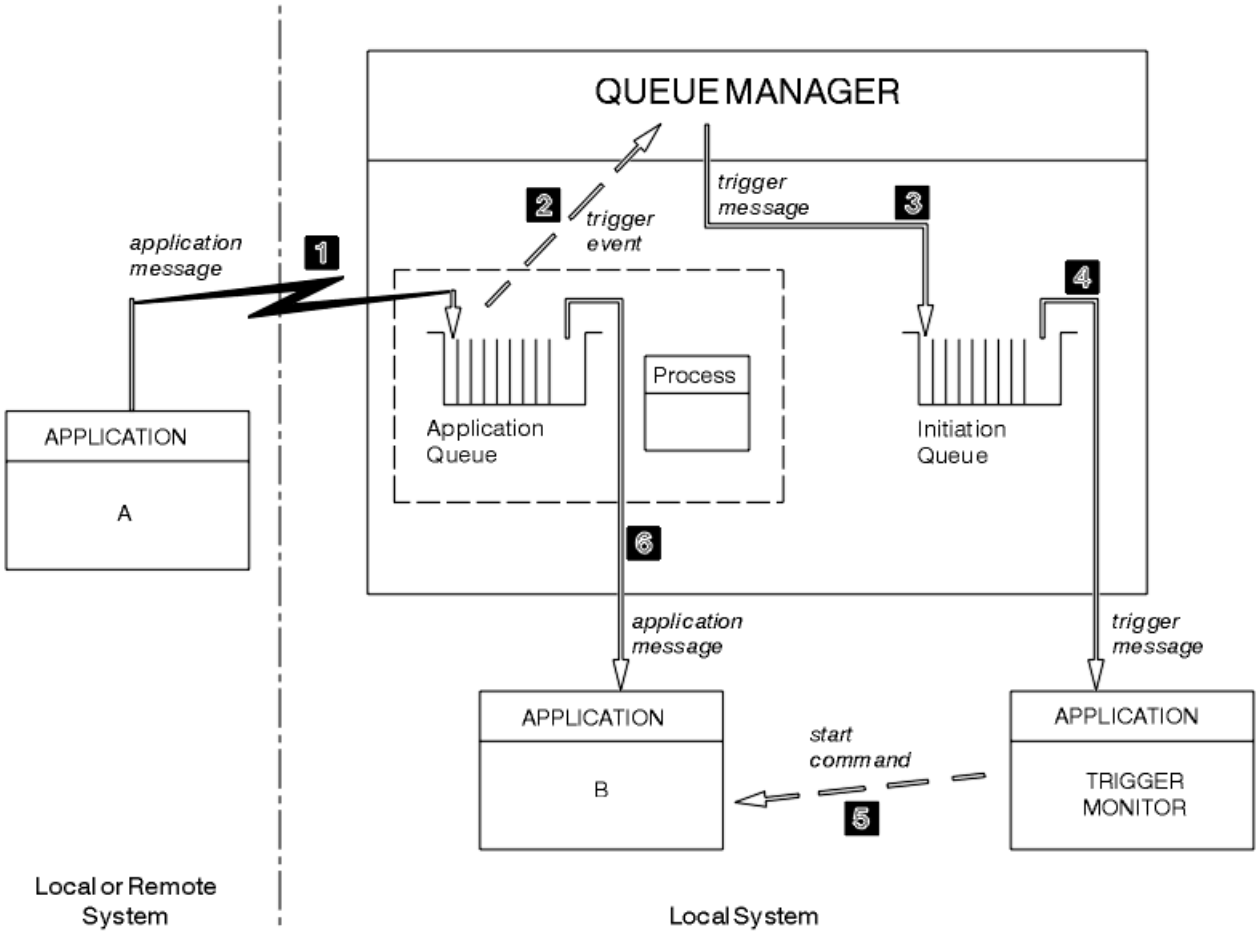


Figura 105. Fluxo de mensagens do aplicativo e do acionador

No Figura 105 na página 865, 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 106 na página 867.

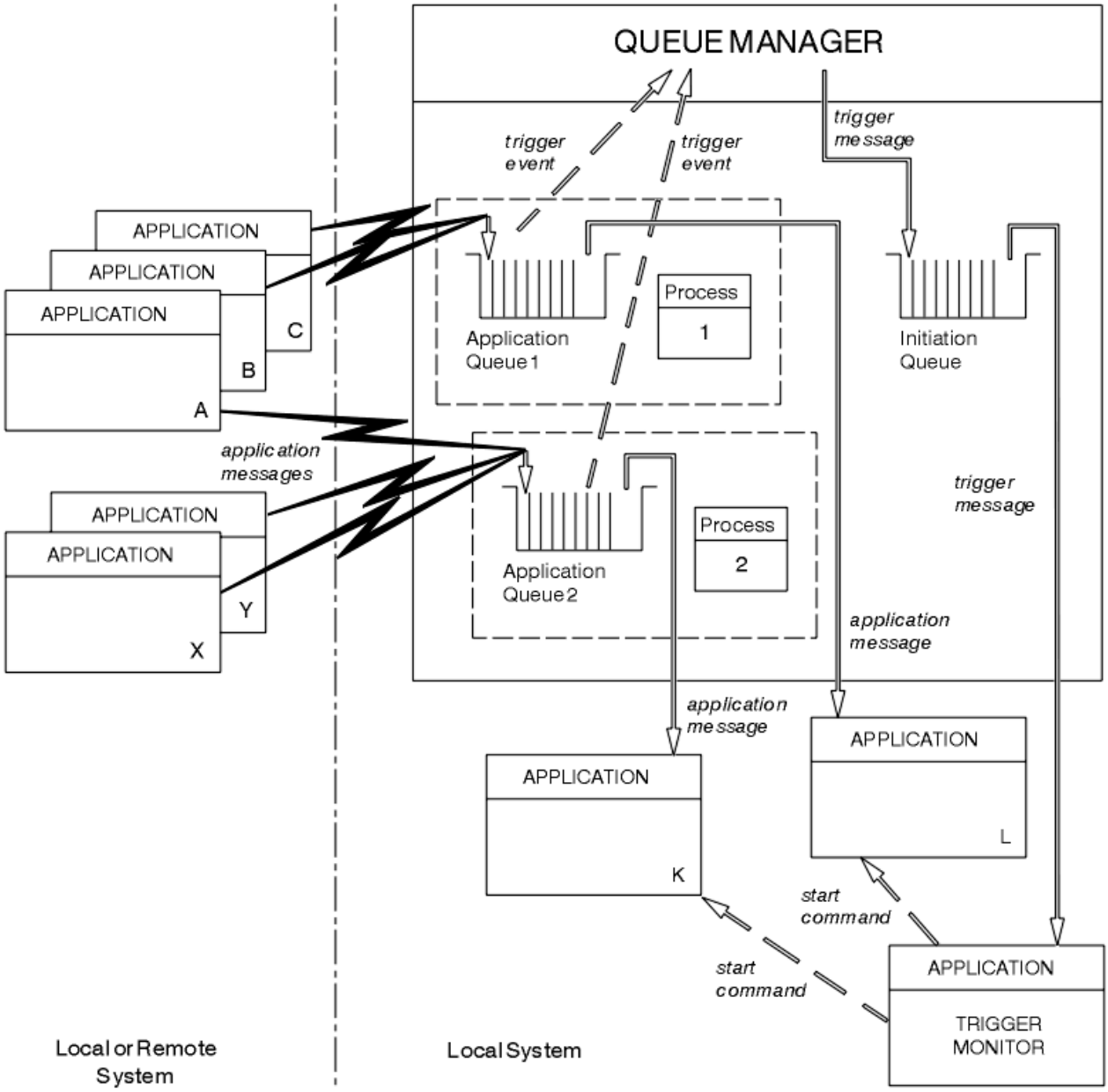


Figura 106. Relacionamento de filas no acionamento

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 868
- “Condições para um evento acionador” na página 870
- “Controlando eventos acionadores” na página 874
- “Projetando um aplicativo que usa filas acionadas” na página 876
- “Monitores de Acionadores” na página 878

- “Propriedades de mensagens do acionador” na página 881
- “Quando o acionamento não funciona” na página 882

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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. Execute um dos dois procedimentos:
   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 UNIX, Linux, and Windows	Use DEFINE PROCESS ou use SYSTEM.DEFAULT.PROCESS e modifique usando ALTER PROCESS
z/OS z/OS z/OS	Use DEFINE PROCESS (consulte o código de amostra na etapa “3” na página 869) ou use as operações e os painéis de controle.
IBM i IBM i IBM i	Use um programa CL que contenha código como na etapa “3” na página 869.

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 no IBM MQ for z/OS, se ele 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 UNIX, 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 “Monitores de Acionadores” na página 878 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 874.

Condições para um evento acionador

O gerenciador de filas cria uma mensagem do acionador quando as condições detalhadas neste tópico são satisfeitas.

Referências a filas compartilhadas neste tópico significam filas compartilhadas em um grupo de filas compartilhadas, disponível apenas no IBM MQ for z/OS.

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:

   a. 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.

b. 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:

   a. 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.

   b. 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. 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 870 a “10” na página 872 (excluindo “3” na página 870) 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 870) e as condições “6” na página 871 a “10” na página 872 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 43.

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 870) e as condições "4" na página 871 a "10" na página 872 (excluindo "8" na página 872) 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 870) em qualquer uma das filas para as quais essa é a fila de inicialização e as condições "4" na página 871 a "10" na página 872 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 870) na fila e as condições "4" na página 871 a "10" na página 872, excluindo a "5" na página 871, 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 870) em qualquer uma das filas do aplicativo para a qual esta é a fila de inicialização e as condições "4" na página 871 a "10" na página 872 (excluindo a "8" na página 872) 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 874.

    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.

Nota:

1. A partir da etapa "12" na página 872 (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.

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 876.

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 870.

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.
- 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.)
- 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.

Monitores de 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.

Windows UNIX *Monitores acionadores em sistemas UNIX e Windows*

Este tópico contém informações sobre monitores acionadores fornecidos em sistemas UNIX e 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 1076 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.

Para executar o aplicativo iniciado em segundo plano em sistemas Windows , no campo *ApplId* , prefixe o nome do aplicativo com um comando START. Por exemplo:

```
START ?B AMQSECHA
```

UNIX Para executar o aplicativo iniciado no plano de fundo no UNIX, coloque um & no final do *EnvData* da definição PROCESS.

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.

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 se aplica somente aos sistemas Windows, UNIX e Linux x86-64.

É fornecido como um programa CICS; defina-lo 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, o registro) e o 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 a partir da fila de inicialização e usadas para iniciar transações CICS , usando EXEC CICS START, supondo que APPL_TYPE na mensagem do acionador seja MQAT_CICS A leitura de mensagens da fila de inicialização é executada sob o controle de ponto de sincronização 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.

A seguir estão as versões disponíveis do monitor acionador:

Versão	Usar
amqltmc0	TXSeries 5.1 para sistemas AIX, HP-UX, Linux x86-64 e Oracle Solaris
amqltmc4	TXSeries 5.1 para Windows
amqltmcc	Versão ligada ao cliente do monitor acionador do CICS

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, envia 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.

   IBM i Use DEFINE QLOCAL ou ALTER QLOCAL. 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 IBM MQ for IBM i comando CL **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 881
- “Reinicialização do gerenciador de filas e mensagens do acionador” na página 881
- “Mensagens do acionador e mudanças nos atributos do objeto” na página 881
- “Formato de mensagens do acionador” na página 881

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. Em qualquer produto IBM MQ, exceto no IBM MQ for z/OS, 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.

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 monitor acionador fornecido por CICS (CKTI) ou o monitor acionador fornecido por IBM MQ for z/OS(CSQQTRMN) não utilizam esse campo, mas outros monitores acionadores podem optar por utilizá-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 884
- “MQPUT, MQPUT1 e clusters” na página 885
- “MQINQ e clusters” na página 885
- “MQSET e clusters” na página 886
- “Códigos de retorno” na página 886

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Usando e escrevendo aplicativos no IBM MQ for z/OS” na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando 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, MQBND_BIND_NOT_FIXED ou MQBND_BIND_ON_GROUP. MQBND_BIND_ON_OPEN ou MQBND_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 745
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.

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')

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 do tópico especificando PUBSCOPE (ALL) ou SUBSCOPE (ALL). A definição de tópico do cluster não pode ser resolvida corretamente porque é necessária uma resposta do gerenciador de filas de repositório completo mas nenhum 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 Usando e escrevendo aplicativos no IBM MQ for z/OS

Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

Estas informações explicam os recursos do IBM MQ disponíveis para programas em execução em cada um dos ambientes suportados. Além disso,

- Para obter informações sobre como usar o IBM MQ-CICS bridge, veja Usando o IBM MQ com o CICS .
- Para obter informações sobre o uso do IMS e a ponte IMS, consulte “Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62.

Use os links a seguir para descobrir mais sobre como usar e escrever aplicativos no IBM MQ for z/OS:

- “Funções IBM MQ for z/OS dependentes de ambiente” na página 888
- “Recursos de depuração, suporte do ponto de sincronização e suporte de recuperação” na página 889
- “A interface do IBM MQ for z/OS com o ambiente de aplicativo” na página 890
- “Gravando aplicativos z/OS UNIX System Services” na página 891
- “Programação de aplicativos com filas compartilhadas” na página 895

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730
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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

“Colocando mensagens em uma fila” na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

“Obtendo mensagens de uma fila” na página 765
Use estas informações para aprender como obter mensagens de uma fila.

“Consultando e configurando atributos de objeto” na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

“Confirmando e fazendo backup de unidades de trabalho” na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

“Trabalhando com MQI e clusters” na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

“Os aplicativos IMS e IMS bridge no IBM MQ for z/OS” na página 62
Essas informações ajudam a gravar aplicativos IMS usando IBM MQ.

Funções IBM MQ for z/OS dependentes de ambiente

Use essas informações ao considerar funções do IBM MQ for z/OS.

As principais diferenças a serem consideradas entre funções do IBM MQ em ambientes nos quais o IBM MQ for z/OS é executado são:

- O IBM MQ for z/OS fornece os seguintes monitores acionadores:
  - CKTI para uso no ambiente do CICS
  - CSQQTRMN para uso no ambiente do IMS

  Deve-se gravar seu próprio módulo para iniciar aplicativos em outros ambientes.
- Syncpointing usando a consolidação de duas fases é suportado no CICS e IMS ambientes. Ele também é suportado no z/OS de ambiente em lote usando o gerenciamento de transações e serviços do gerenciador de recurso recuperável (RRS). A single-phase commit é suportada no ambiente do z/OS pelo próprio IBM MQ.

- Para obter ambientes do IMS e lote, MQI fornece chamadas para conectar programas ao gerenciador de filas e para desconectá-los do gerenciador de filas. Os programas podem se conectar a mais de um gerenciador de filas.
- Um sistema CICS pode conectar-se a apenas um gerenciador de filas. É possível fazer com que isso aconteça quando o CICS é iniciado se o nome do subsistema estiver definido na tarefa de inicialização do sistema CICS. As chamadas de conexão e desconexão MQI são toleradas, mas não têm efeito sobre o ambiente do CICS.
- A saída cruzada da API permite que um programa intervenha no processamento de todas as chamadas MQI. Essa saída está disponível no ambiente do CICS apenas.
- No CICS em sistemas multiprocessadores, alguma vantagem de desempenho é obtida, pois as chamadas MQI podem ser executadas sob vários TCBs z/OS. Para obter mais informações, consulte o Planejando em z/OS . *Guia de conceitos e planejamento do IBM MQ for z/OS*.

Esses recursos são resumidos em Tabela 115 na página 889.

Tabela 115. Recursos ambientais do z/OS

	CICS	**IMS**	**Batch/TSO**
Monitor acionador fornecido	Sim	Sim	No
Two-phase commit	Sim	Sim	Sim
Single-phase commit	Sim	No	Sim
Conectar/desconectar chamadas MQI	tolerado	Sim	Sim
saída de cruzamento de API	Sim	No	No

Nota: Two-phase commit é suportado no ambiente utilizando Batch/TSO RRS.

Recursos de depuração, suporte do ponto de sincronização e suporte de recuperação

Use estas informações para aprender sobre os recursos de depuração, o suporte do ponto de sincronização e o suporte de recuperação do programa.

Recursos de depuração do programa

O IBM MQ for z/OS fornece um recurso de rastreio que pode ser usado para depurar seus programas em todos os ambientes.

Além disso, no ambiente do CICS, é possível usar:

- O recurso de diagnóstico de execução do CICS (CEDF)
- The CICS Trace Control Transaction (CETR)
- A saída cruzada da API do IBM MQ for z/OS

Na plataforma z/OS, é possível usar qualquer ferramenta de depuração interativa disponível suportada pela linguagem de programação que você está usando.

Suporte do ponto de sincronização

Sincronizar o início e o fim de unidades de trabalho é necessário em um ambiente de processamento de transações de modo que o processamento de transação possa ser usado com segurança.

Isso é totalmente suportado pelos ambientes do IBM MQ for z/OS no CICS e no IMS. Suporte completo significa cooperação entre os gerenciadores de recursos para que unidades de trabalho possam ser consolidadas ou recuperadas em uníssono, sob o controle do CICS ou do IMS. Exemplos de gerenciadores de recursos são Db2, CICS File Control, IMS e IBM MQ for z/OS.

Os aplicativos em lote do z/OS podem usar IBM MQ for z/OS chamadas para fornecer um recurso single-phase commit. Isso significa que um conjunto de operações de fila definido pelo aplicativo pode ser confirmado ou recuperado, sem fazer referência a outros gerenciadores de recursos.

Two-phase commit também é suportado no ambiente de lote do z/OS usando gerenciamento de transação e serviços do gerenciador de recurso recuperável (RRS). Para obter informações adicionais, consulte Pontos de sincronização em aplicativos em lote do z/OS.

Suporte à recuperação

Se a conexão entre um gerenciador de filas e um sistema CICS ou IMS for interrompida durante uma transação, algumas unidades de trabalho podem não ser recuperadas com sucesso.

No entanto, essas unidades de trabalho serão resolvidas pelo gerenciador de filas (sob o controle do gerenciador de ponto de sincronização) quando sua conexão com o sistema CICS ou IMS for restabelecida.

A interface do IBM MQ for z/OS com o ambiente de aplicativo

Para permitir que aplicativos em execução em ambientes diferentes enviem e recebam mensagens por meio de uma rede de enfileiramento de mensagens, o IBM MQ for z/OS fornece um *adaptador* para cada um dos ambientes que ele suporta.

Esses adaptadores são a interface entre programas aplicativos e subsistemas IBM MQ for z/OS. Eles permitem que os programas usem a MQI.

O adaptador em lote

Use essas informações para aprender sobre o adaptador em lote e o protocolo de confirmação que ele suporta.

O *adaptador em lote* fornece acesso aos recursos do IBM MQ for z/OS para programas em execução no:

- Modo de tarefa (TCB)
- Problema ou estado do supervisor
- Modo de controle de espaço de endereço principal

Os programas não devem estar em modo de memória cruzada.

Conexões entre os programas de aplicativo e o IBM MQ for z/OS estão no nível de tarefa. O adaptador fornece um encadeamento de conexão única a partir de um bloco de controle de tarefas (TCB) para o IBM MQ for z/OS.

O adaptador suporta um protocolo single-phase commit para mudanças feitas nos recursos pertencentes ao IBM MQ for z/OS; ele não suporta protocolos multiphase commit.

O adaptador em lote RRS

Use estas informações para saber mais sobre o adaptador em lote RRS e os dois adaptadores em lote RRS fornecidos pelo IBM MQ.

O adaptador de gerenciamento de transação e de serviços do gerenciador de recurso recuperável (RRS):

- Usa z/OS RRS para o controle de confirmação.
- Suporta conexões simultâneas com vários subsistemas IBM MQ em execução em uma única instância do z/OS a partir de uma única tarefa.
- Fornece controle de compromisso coordenado para todo o z/OS (usando o z/OS RRS) para recursos recuperáveis acessados por meio de gerenciadores recuperáveis compatíveis com o z/OS RRS para:
  - Os aplicativos que se conectam ao IBM MQ usando o adaptador em lote RRS.
  - Procedimentos armazenados do Db2 em execução em um espaço de endereço de procedimentos armazenados do Db2 que é gerenciado por um workload manager (WLM) no z/OS.
- Suporta a capacidade de alternar um encadeamento em lote do IBM MQ entre TCBs.

O IBM MQ for z/OS fornece dois adaptadores em lote RRS:

CSQBRSTB
Esse adaptador requer que você mude qualquer instrução MQCMIT para SRRCMIT e qualquer instrução MQBACK para SRRBACK em seu aplicativo IBM MQ. (Se você codificar MQCMIT ou MQBACK em um aplicativo vinculado com CSQBRSTB, receberá MQRC_ENVIRONMENT_ERROR.)

CSQBRRSI
Esse adaptador permite que seu aplicativo IBM MQ use MQCMIT e MQBACK ou SRRCMIT e SRRBACK.

Nota: CSQBRSTB e CSQBRRSI são fornecidos com atributos de ligação AMODE(31) RMODE(ANY). Se seu aplicativo carregar o stub abaixo da linha de 16 MB, primeiro religue o stub com RMODE(24).

migração

É possível migrar aplicativos IBM MQ Lote/TSO existentes para usarem coordenação de RRS com poucas ou nenhuma mudança.

Se você editar por link seu aplicativo IBM MQ com o adaptador CSQBRRSI, MQCMIT e MQBACK criam o ponto de sincronização de sua unidade de trabalho entre o IBM MQ e todos os outros gerenciadores de recursos ativados por RRS. Se você editar por link seu aplicativo IBM MQ com o adaptador CSQBRSTB, mude MQCMIT para SRRCMIT e MQBACK para SRRBACK. A última abordagem é preferível; ela claramente indica que o ponto de sincronização não está restrito a apenas recursos do IBM MQ.

O adaptador IMS
Se você estiver usando o adaptador IMS a partir de um sistema IBM MQ for z/OS, certifique-se de que o IMS pode obter armazenamento suficiente para acomodar mensagens com até 100 MB de comprimento.

Nota para usuários

O *adaptador IMS* fornece acesso aos recursos do IBM MQ for z/OS para:

- Programas de processamento de mensagens on-line (MPPs)
- Programas de atalho interativo (IFPs)
- Programas de processamento de mensagens em lote (BMPs)

Para usar esses recursos, os programas devem estar em execução no modo tarefa (TCB) e estado de problema; eles não devem estar no modo de memória cruzada ou no modo de registro de acesso.

O adaptador fornece um encadeamento de conexão a partir de um bloco de controle de tarefas (TCB) para IBM MQ. O adaptador suporta um protocolo two-phase commit para mudanças feitas nos recursos de propriedade do IBM MQ for z/OS, com o IMS atuando como coordenador de ponto de sincronização.

O adaptador também fornece um programa de monitor acionador que pode iniciar programas automaticamente quando certas condições acionadoras em uma fila forem atendidas. Para obter mais informações, consulte “Iniciando aplicativos IBM MQ usando acionadores” na página 863.

Se você estiver gravando programas em lote DL/I, siga a orientação fornecida neste tópico para programas em lote do z/OS.

Gravando aplicativos z/OS UNIX System Services

O adaptador de lote suporta conexões do gerenciador de filas a partir de espaços de endereço em lote e TSO:

Se considerarmos um espaço de endereço em lote, o adaptador suporta conexões a partir de diversos TCBs dentro desse espaço de endereço como a seguir:

- Cada TCB pode se conectar a diversos gerenciadores de filas usando a chamada MQCONN ou MQCONNX (mas um TCB pode ter somente uma instância de uma conexão com um determinado gerenciador de filas a qualquer momento).
- Diversos TCBs podem se conectar ao mesmo gerenciador de filas (mas o identificador de gerenciador de filas retornado em qualquer chamada MQCONN ou MQCONNX é ligado ao TCB emitindo e não pode ser usado por nenhum outro TCB).

O z/OS UNIX System Services suporta dois tipos de chamadas pthread_create:

1. Os encadeamentos pesados, executados um para cada TCB, que são ATTACHed e DETACHed no início e final do encadeamento pelo z/OS.
2. Encadeamentos de peso médio, executados um para cada TCB, mas o TCB pode ser um de um conjunto de TCBs de longa execução. O aplicativo deve executar toda a limpeza de aplicativo necessária, pois, se conectado a um servidor, o término de encadeamento padrão que pode ser fornecido pelo servidor no término da tarefa (TCB) **nem** sempre é direcionado.

Encadeamentos leves não são suportados. (Se um aplicativo cria seus encadeamentos permanentes que despacham suas próprias solicitações de trabalho, o ***aplicativo*** é responsável pela limpeza de quaisquer recursos antes de iniciar a próxima solicitação de trabalho.)

O IBM MQ for z/OS suporta encadeamentos z/OS UNIX System Services usando o Adaptador em Lote da seguinte forma:

1. Os encadeamentos pesados são totalmente suportados como conexões em lote. Cada encadeamento é executado em seu próprio TCB, que é anexado e desanexado no início e no fim do encadeamento. Caso o encadeamento termine antes da emissão de uma chamada MQDISC, o IBM MQ for z/OS executará sua limpeza de tarefa padrão, o que inclui confirmar qualquer unidade de trabalho pendente se o encadeamento tiver sido finalizado normalmente ou retroceder se o encadeamento tiver sido finalizado de forma anormal.
2. Encadeamentos de peso médio são totalmente suportados, mas se o TCB for reutilizado por outro encadeamento, o aplicativo deverá assegurar que uma chamada MQDISC, precedida por um MQCMIT ou MQBACK, seja emitida antes do início do próximo encadeamento. Isso significa que se o aplicativo tiver estabelecido um Manipulador de Interrupção de Programa e o aplicativo for finalizado de forma anormal, o Manipulador de interrupção deverá emitir chamadas MQCMIT e MQDISC antes de reutilizar o TCB para outro encadeamento.

Nota: Os modelos de encadeamento **não** suportam o acesso a recursos comuns do IBM MQ a partir de diversos encadeamentos.

A saída cruzada da API para o z/OS

Este tópico contém do informações sobre a interface de programação sensível ao produto.

Uma saída é um ponto no código fornecido pelo IBM em que é possível executar o seu próprio código. O IBM MQ for z/OS fornece um *Saída cruzada da API* que é possível usar para interceptar chamadas para o MQI e para monitorar ou modificar a função das chamadas MQI. Esta seção descreve como usar a saída cruzada da API, e descreve o programa de saída de amostra que é fornecido com o IBM MQ for z/OS.

Esta seção é aplicável somente a usuários do CICS TS V3.1 e anterior. Os usuários do CICS TS V3.2 e posterior devem consultar a seção Integração do CICS com o IBM MQ na documentação do produto CICS.

Nota

A saída cruzada da API é chamada somente pelo adaptador CICS do IBM MQ for z/OS. O programa de saída é executado no espaço de endereço do CICS.

Escrevendo seu próprio programa de saída
É possível usar o programa de saída de cruzamento de API de amostra (CSQCAPX) que é fornecido com o IBM MQ for z/OS como uma estrutura para seu próprio programa.

Isso é descrito no "O programa de saída cruzada da API de amostra, CSQCAPX" na página 893.

Ao gravar um programa de saída, para localizar o nome de uma chamada MQI emitida por um aplicativo, examine o campo *ExitCommand* da estrutura MQXP. Para localizar o número de parâmetros na chamada, examine o campo *ExitParmCount*. É possível usar o campo de 16 bytes *ExitUserArea* para armazenar o endereço de qualquer armazenamento dinâmico que o aplicativo obtém. Esse campo é retido nas chamadas da saída e tem a mesma duração que uma tarefa do CICS.

Se estiver usando o CICS Transaction Server V3.2, deve-se gravar seu programa de saída para que seja thread-safe e declarar seu programa de saída como thread-safe. Se estiver usando liberações anteriores

do CICS, também será recomendado gravar e declarar seus programas de saída como thread-safe para que estejam prontos para migrar para o CICS Transaction Server V3.2.

O programa de saída pode suprimir a execução de uma chamada MQI, retornando MQXCC_SUPPRESS_FUNCTION ou MQXCC_SKIP_FUNCTION no campo *ExitResponse*. Para permitir que a chamada seja executada (e o programa de saída chamado novamente após a chamada ser concluída), seu programa de saída deve retornar MQXCC_OK.

Quando chamado após uma chamada MQI, um programa de saída pode inspecionar e modificar os códigos de conclusão e de razão definidos pela chamada.

Observações de Uso

Aqui estão alguns pontos gerais a serem considerados ao gravar seu programa de saída:

- Por motivos de desempenho, grave seu programa em linguagem de montagem. Se gravá-lo em qualquer uma das outras linguagens suportadas pelo IBM MQ for z/OS, deve-se fornecer seu próprio arquivo de definição de dados.
- Edite a vinculação do seu programa como AMODE(31) e RMODE(ANY).
- Para definir o bloco de parâmetros de saída para seu programa, use a macro do montador de linguagem, CMQXPA.
- Especifique CONCURRENCY(THREADSAFE) quando você definir seu programa de saída e todos os programas que seu programa de saída chamar.
- Se você estiver usando o CICS Transaction Server para o recurso de proteção de armazenamento do z/OS, seu programa deverá ser executado na chave de execução do CICS. Ou seja, deve-se especificar EXECKEY(CICS) ao definir seu programa de saída e quaisquer programas para os quais ele transmite o controle. Para obter informações sobre programas de saída do CICS e o recurso de proteção de armazenamento do CICS, consulte o *CICS Guia de Customização*.
- Seu programa pode usar todas as APIs (por exemplo, IMS, Db2 e CICS) que um programa de saída do usuário relacionado à tarefa do CICS possa usar. Ele também pode usar qualquer uma das chamadas MQI exceto MQCONN, MQCONNX e MQDISC. No entanto, quaisquer chamadas MQI dentro do programa de saída não chamarão o programa de saída uma segunda vez.
- Seu programa pode emitir os comandos EXEC CICS SYNCPOINT ou EXEC CICS SYNCPOINT ROLLBACK. No entanto, esses comandos confirmam ou retrocedem ***todas*** as atualizações feitas pela tarefa até o ponto em que a saída foi usada e, portanto, seu uso não é recomendado.
- O seu programa tem que ser finalizado, emitindo um comando EXEC CICS RETURN. Ele não deve transferir o controle com um comando XCTL.
- As saídas são gravadas como extensões para o código do IBM MQ for z/OS. Certifique-se de que sua saída não interrompa programas ou transações IBM MQ for z/OS que usam a MQI. Esses geralmente são indicados com um prefixo CSQ ou CK.
- Se o CSQCAPX for definido para o CICS, o sistema CICS tentará carregar o programa de saída quando o CICS se conectar ao IBM MQ for z/OS. Se essa tentativa for bem-sucedida, a mensagem CSQC301I será enviada para o painel CKQC ou para o console do sistema. Se o carregamento for malsucedido (por exemplo, se o módulo de carregamento não existir em nenhuma das bibliotecas na concatenação de DFHRPL), a mensagem CSQC315 será enviada para o painel CKQC ou para o console do sistema.
- Como os parâmetros na área de comunicação são endereços, o programa de saída deve estar definido como local para o sistema CICS (ou seja, não como um programa remoto).

O programa de saída cruzada da API de amostra, CSQCAPX
O programa de saída da amostra é fornecido como um programa de linguagem assembler. O arquivo de origem (CSQCAPX) é fornecido na biblioteca **thlqual**.SCSQASMS (em que **thlqual** é o qualificador de alto nível usado por sua instalação). Esse arquivo de origem inclui o pseudocódigo que descreve a lógica do programa.

O programa de amostra contém o código de inicialização e um layout que é possível usar quando você estiver escrevendo seus próprios programas de saída.

A amostra mostra como:

- Configurar o bloco de parâmetros de saída
- Direcionar a chamada e blocos de parâmetros de saída
- Determinar para qual chamada MQI a saída está sendo chamada
- Determinar se a saída está sendo chamada antes ou depois do processamento da chamada MQI
- Colocar uma mensagem em uma fila de armazenamento temporário do CICS
- Usar a macro DFHEIENT para aquisição de armazenamento dinâmico para manter reentrância
- Usar DFHEIBLK para o bloco de controle da interface exec do CICS
- Efetuar trap de condições de erro
- Retornar o controle ao responsável pela chamada

Design do programa de saída de amostra

O programa de saída de amostra grava mensagens em uma fila de armazenamento temporário do CICS (CSQ1EXIT) para mostrar a operação da saída.

As mensagens mostram se a saída está sendo chamada antes ou após a chamada MQI. Se a saída for chamada após a chamada, a mensagem conterá o código de conclusão e o código de razão retornados pela chamada. A amostra usa constantes denominadas da macro CMQXPA para verificar o tipo de entrada (ou seja, antes ou após a chamada).

A amostra não executa nenhuma função de monitoramento, mas simplesmente coloca mensagens com um registro de data e hora em uma fila do CICS indicando o tipo de chamada que ele está processando. Isso fornece uma indicação do desempenho da MQI, assim como o funcionamento correto do programa de saída.

Nota: O programa de saída de amostra emite seis chamadas CICS EXEC para cada chamada MQI que é feita enquanto o programa estiver em execução. Se você usar esse programa de saída, o desempenho do IBM MQ for z/OS será degradado.

Preparando e usando a saída cruzada da API
A saída de amostra é fornecida somente no formato de origem.

Para usar a saída de amostra ou um programa de saída que você tenha escrito, crie uma biblioteca de carregamento como você faria para qualquer outro programa CICS, conforme descrito em "Criando aplicativos CICS no z/OS" na página 1044.

- Para o CICS Transaction Server for z/OS e o CICS for MVS/ESA, ao atualizar o conjunto de dados da definição do sistema CICS (CSD), as definições que você precisa estão no membro **thlqual**.SCSQPROC(CSQ4B100).

  Nota: As definições usam um sufixo MQ. Se esse sufixo já for usado em sua empresa, ele deve ser mudado antes do estágio de montagem.

Se usar as definições padrão do programa CICS fornecidas, o programa de saída CSQCAPX será instalado em um estado ***desativado***. Isso ocorre porque usar o programa de saída pode produzir uma redução significativa no desempenho.

Para ativar a saída cruzada da API temporariamente:

1. Emita o comando **CEMT S PROGRAM(CSQCAPX) ENABLED** a partir do terminal principal do CICS.
2. Execute a transação CKQC e use a opção 3 no menu suspenso Conexão para alterar o status da saída cruzada da API para **Ativado**.

Se deseja executar o IBM MQ for z/OS com a saída cruzada da API permanentemente ativada, com o CICS Transaction Server for z/OS e o CICS for MVS/ESA, execute um dos seguintes procedimentos:

- Mude a definição de CSQCAPX no membro CSQ4B100, mudando STATUS(DISABLED) para STATUS(ENABLED). É possível atualizar a definição CSD do CICS usando o programa em lote DFHCSDUP fornecido pelo CICS.

- Altere a definição CSQCAPX no grupo CSQCAT1 mudando o status de DISABLED para ENABLED.

Em ambos os casos, deve-se reinstalar o grupo. É possível fazer isso com cold start do sistema CICS usando a transação CICS CEDA para reinstalar o grupo enquanto o CICS está em execução.

Nota: Usar CEDA pode causar um erro se qualquer uma das entradas no grupo estiver atualmente em uso.

Fim das informações da interface de programação sensível ao produto.

Programação de aplicativos com filas compartilhadas

Este tópico fornece informações sobre alguns dos fatores que você precisa para levar em conta ao projetar novos aplicativos para utilizar filas compartilhadas, e quando migrar aplicativos existentes para o ambiente compartilhado de filas.

Serializando seus aplicativos

Determinados tipos de aplicativos podem ter que garantir que as mensagens sejam recuperadas de uma fila em exatamente a mesma ordem em que chegaram na fila.

Por exemplo, se o IBM MQ estiver sendo usado para sombrear as atualizações do banco de dados em um sistema remoto, uma mensagem que descreve a atualização de um registro deve ser processada após uma mensagem que descreve a inserção desse registro. Em um ambiente de enfileiramento local, isso é frequentemente obtido pelo aplicativo que obtém as mensagens abrindo a fila com a opção MQOO_INPUT_EXCLUSIVE, evitando, assim, que qualquer outro aplicativo de get processe a fila ao mesmo tempo.

O IBM MQ permite que os aplicativos abram filas compartilhada exclusivamente da mesma maneira. No entanto, se o aplicativo estiver funcionando a partir de uma partição de uma fila (por exemplo, todas as atualizações do banco de dados estão na mesma fila, mas aquelas da tabela A têm um identificador de correlação A e aquelas da tabela B um identificador de correlação B) e os aplicativos desejarem obter mensagens de atualizações da tabela A e de atualizações da tabela B simultaneamente, o mecanismo simples de abrir a fila exclusivamente não é possível.

Se esse tipo de aplicativo precisar tirar proveito da alta disponibilidade de filas compartilhadas, você pode decidir que outra instância do aplicativo que acessa as mesmas filas compartilhadas, em execução em um gerenciador de filas secundário, deve assumir o controle se o aplicativo de get ou gerenciador de filas primário falhar.

Se o gerenciador de filas primário falhar, duas coisas acontecem:

- A recuperação peer-to-peer de fila compartilhada assegura que quaisquer atualizações incompletas do aplicativo principal são concluídas ou restauradas.
- O aplicativo secundário assumirá o processamento da fila.

O aplicativo secundário pode iniciar antes que todas as unidades de trabalho incompletas tenham sido manipuladas, o que pode levar o aplicativo secundário a recuperar as mensagens fora de sequência. Para resolver esse tipo de problema, o aplicativo pode optar por ser um *aplicativo serializado*.

Um aplicativo serializado usa a chamada MQCONNX para se conectar ao gerenciador de filas, especificando um tag de conexão ao se conectar que é exclusivo do aplicativo. Quaisquer unidades de trabalho executadas pelo aplicativo são marcadas com a sinalização de conexão. O IBM MQ garante que as unidades de trabalho dentro do grupo de compartilhamento de filas com a mesma sinalização de conexão sejam serializadas (de acordo com as opções de serialização na chamada MQCONNX).

Isso significa que, se o aplicativo principal usar a chamada MQCONNX com uma tag de conexão `Database shadow retriever` e o aplicativo de controle secundário tentar usar a chamada MQCONNX com uma tag de conexão idêntica, o aplicativo secundário não poderá se conectar ao segundo IBM MQ até que quaisquer unidades de trabalho primárias pendentes tenham sido concluídas, neste caso, por recuperação peer-ro-peer.

Considere usar a técnica de aplicativo serializado para aplicativos que dependem da sequência exata de mensagens em uma fila. Principalmente:

- Aplicativos que não devem reiniciar após uma falha de aplicativo ou de gerenciador de filas até que todas as operações de confirmação e restauração da execução anterior do aplicativo sejam concluídas.

  Nesse caso, a técnica de aplicativo serializado é aplicável somente se o aplicativo funcionar no ponto de sincronização.
- Os aplicativos que não devem iniciar enquanto outra instância do mesmo aplicativo já está em execução.

  Nesse caso, a técnica de aplicativo serializado técnica é necessária somente se o aplicativo não puder abrir a fila para entrada exclusiva.

Nota: O IBM MQ garante preservar a sequência de mensagens somente quando determinados critérios são atendidos. Eles são descritos na descrição de MQGET.

Os aplicativos que não são adequados para uso com filas compartilhadas
Alguns recursos do IBM MQ não são suportados ao usar filas compartilhadas, portanto, aplicativos que usam esses recursos não são adequados para o ambiente de filas compartilhadas.

Considere os seguintes pontos ao projetar seus aplicativos de filas compartilhadas:

- A fila de indexação é limitada para filas compartilhadas. Se desejar usar o identificador de mensagem ou identificador de correlação para selecionar a mensagem que você deseja obter da fila, a fila deve ser indexada com o valor correto. Se estiver selecionando mensagens pelo identificador de mensagem sozinho, a fila precisa de um tipo de índice de MQIT_MSG_ID (embora também seja possível usar MQIT_NONE). Se estiver selecionando mensagens apenas pelo identificador de correlação, a fila deve ter um tipo de índice de MQIT_CORREL_ID.
- Não é possível usar filas dinâmicas temporárias como filas compartilhadas. No entanto, é possível filas dinâmicas permanentes. Os modelos para filas dinâmicas compartilhadas têm um DEFTYPE de SHAREDYN (dinâmica compartilhada), embora sejam criados e destruídos da mesma maneira que as filas PERMDYN (dinâmicas permanente).

Decidindo se filas de não aplicativos devem ou não ser compartilhadas
Use estas informações ao considerar compartilhar filas que não forem de aplicativo.

Existem outras filas além de filas de aplicativo que você pode desejar considerar compartilhar:

Filas de inicialização

Se você definir uma fila de inicialização compartilhada, não será necessário ter um monitor acionador em execução em cada gerenciador de filas no grupo de filas compartilhadas, desde que haja pelo menos um monitor acionador em execução. (Também será possível usar uma fila de inicialização compartilhada mesmo se houver um monitor acionador em execução em cada gerenciador de filas no grupo de filas compartilhadas.)

Se você tiver uma fila do aplicativo compartilhada e usar o tipo de acionador EVERY (ou um tipo de acionador FIRST com o intervalo de acionador pequeno que se comporta como um tipo de acionador EVERY) sua fila de inicialização deverá sempre ser uma fila compartilhada. Para obter mais informações sobre quando usar uma fila de inicialização compartilhada, consulte Tabela 116 na página 897.

filas SYSTEM.*

É possível definir as filas SYSTEM.ADMIN.* usadas para manter as mensagens de eventos como filas compartilhadas. Isso pode ser útil para verificar balanceamento de carga se ocorrer uma exceção. Cada mensagem de evento criada pelo IBM MQ contém um identificador de correlação indicando qual gerenciador de filas a produziu.

Deve-se definir as filas SYSTEM.QSG.* usadas para canais compartilhados e enfileiramento intragrupo como filas compartilhadas.

Também é possível mudar as definições da SYSTEM.DEFAULT.LOCAL.QUEUE a serem compartilhadas ou definir sua própria definição de fila compartilhada padrão. Isso é descrito na seção **Definindo objetos do sistema** no Planejando em z/OS . *IBM MQ for z/OS Guia de conceitos e planejamento*.

Não é possível definir quaisquer outras filas SYSTEM.* como filas compartilhadas.

Migrando seus aplicativos existentes para usar filas compartilhadas
Códigos de razão, acionamento e a chamada API MQINQ podem funcionar de forma diferente em um ambiente de fila compartilhada.

A migração de suas filas existentes para filas compartilhadas é descrita no Administrando IBM MQ for z/OS *IBM MQ for z/OS Guia de Administração do Sistema*.

Ao migrar seus aplicativos existentes, considere os aspectos a seguir, que podem funcionar de maneira diferente no ambiente de fila compartilhada:

Códigos de Razão
Ao migrar seus aplicativos existentes para usar filas compartilhadas, verifique os novos códigos de razão que podem ser emitidos.

Acionando
Se você estiver usando uma fila do aplicativo compartilhada, o acionamento funcionará apenas em mensagens confirmadas (em um aplicativo de filas não compartilhadas, o acionamento funcionará em todas as mensagens).

Se você usar o acionamento para iniciar aplicativos, você pode desejar usar uma fila de inicialização compartilhada. O Tabela 116 na página 897 descreve o que você precisa considerar ao decidir qual tipo de fila de inicialização utilizar.

Tabela 116. Quando usar uma fila de inicialização compartilhada

	Fila do aplicativo não compartilhada	**Fila do aplicativo compartilhada**
Fila de inicialização não compartilhada	Como para liberações anteriores.	Se você usar um tipo de acionador de FIRST ou DEPTH, será possível usar uma fila de inicialização não compartilhada com uma fila aplicativo compartilhada. Mensagens do acionador extra podem ser geradas, mas essa configuração é boa para acionar aplicativos de longa execução (como o CICS bridge) e fornece alta disponibilidade.) Para o tipo de acionador FIRST ou DEPTH, uma mensagem do acionador aciona uma instância do aplicativo em cada gerenciador de filas que está executando um monitor acionador e que ainda não tenha a fila do aplicativo aberta para entrada. Uma mensagem do acionador é gerada para cada gerenciador de filas; se houver mais do que um monitor acionador em execução na fila de inicialização não compartilhada local, em um determinado gerenciador de filas, eles competirão para processar a mensagem.

Tabela 116. Quando usar uma fila de inicialização compartilhada (continuação)		
	Fila do aplicativo não compartilhada	**Fila do aplicativo compartilhada**
Fila de inicialização compartilhada	Não use uma fila de inicialização compartilhada com uma fila de aplicativo não compartilhada.	Se você tiver uma fila do aplicativo compartilhada que tenha um tipo de acionador EVERY, use uma fila de inicialização compartilhada ou você pode perder as mensagens do acionador em determinadas circunstâncias; por exemplo, um gerenciador de filas com falha. Para o tipo de acionador FIRST ou DEPTH, uma mensagem do acionador será gerada por cada gerenciador de filas que possui a fila de inicialização denominada aberta para entrada. **Nota:** Para o tipo de acionador FIRST ou DEPTH, se uma instância de monitor acionador estiver ocupada, isso deixará o potencial para monitores acionadores menos ocupados processarem mais de uma mensagem do acionador da fila de inicialização compartilhada. Assim, diversas instâncias do aplicativo de servidor podem ser iniciadas em relação a um determinado gerenciador de filas. Observe que essas diversas instâncias são iniciadas como resultado do processamento de diversas mensagens do acionador. Normalmente, para o tipo de acionador FIRST ou DEPTH, se uma instância de aplicativo já estiver servindo uma fila do aplicativo, outra mensagem do acionador não será gerada pelo gerenciador de filas ao qual o aplicativo está conectado.

MQINQ

Ao usar a chamada MQINQ para exibir informações sobre uma fila compartilhada, os valores do número de chamadas MQOPEN que possuem a fila aberta para entrada e saída estão relacionados apenas ao gerenciador de filas que emitiu a chamada. Nenhuma informação é produzida sobre outros gerenciadores de filas no grupo de filas compartilhadas que têm a fila aberta.

z/OS Os aplicativos IMS e IMS bridge no IBM MQ for z/OS

Essas informações ajudam a gravar aplicativos IMS usando IBM MQ.

- Para usar pontos de sincronização e as chamadas MQI em aplicativos IMS, consulte “Gravando aplicativos do IMS usando o IBM MQ” na página 63.
- Para escrever aplicativos que usam a ponte IBM MQ - IMS, consulte “Escrevendo aplicativos de ponte IMS” na página 67.

Use os links a seguir para descobrir mais sobre os aplicativos IMS e de ponte IMS no IBM MQ for z/OS:

- “Gravando aplicativos do IMS usando o IBM MQ” na página 63
- “Escrevendo aplicativos de ponte IMS” na página 67

Conceitos relacionados

“Visão geral da Message Queue Interface” na página 715
Aprenda sobre os componentes de Message Queue Interface (MQI).

“Conectando-se e desconectando-se de um gerenciador de filas” na página 730

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 739
Essas informações fornecem um insight sobre abrir e fechar objetos do IBM MQ.

"Colocando mensagens em uma fila" na página 750
Use estas informações para aprender como colocar mensagens em uma fila.

"Obtendo mensagens de uma fila" na página 765
Use estas informações para aprender como obter mensagens de uma fila.

"Consultando e configurando atributos de objeto" na página 848
Atributos são as propriedades que definem as características de um objeto do IBM MQ.

"Confirmando e fazendo backup de unidades de trabalho" na página 851
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 863
Aprenda sobre acionadores e como iniciar aplicativos IBM MQ usando acionadores.

"Trabalhando com MQI e clusters" na página 883
Há opções especiais em chamadas e códigos de retorno relacionadas ao armazenamento em cluster.

"Usando e escrevendo aplicativos no IBM MQ for z/OS" na página 887
Aplicativos IBM MQ for z/OS podem ser compostos por programas executados em muitos ambientes diferentes. Isso significa que eles podem aproveitar os recursos disponíveis em mais de um ambiente.

Gravando aplicativos do IMS usando o IBM MQ

Há considerações adicionais ao usar o IBM MQ em aplicativos do IMS. Estas incluem chamadas API do MQ que podem ser usadas e o mecanismo usado para o ponto de sincronização.

Use os seguintes links para descobrir mais sobre a gravação de aplicativos do IMS no IBM MQ for z/OS:

- "Pontos de sincronização em aplicativos IMS" na página 63
- "Chamadas MQI em aplicativos IMS" na página 64

Restrições

Há restrições sobre quais chamadas API do IBM MQ podem ser usadas por um aplicativo que usa o adaptador do IMS.

As seguintes chamadas API do IBM MQ não são suportadas em um aplicativo que usa o adaptador do IMS:

- MQCB
- MQCB_FUNCTION
- MQCTL

Conceitos relacionados

"Escrevendo aplicativos de ponte IMS" na página 67
Este tópico contém informações sobre como escrever aplicativos para usar a ponte IBM MQ - IMS.

Pontos de sincronização em aplicativos IMS

Em um aplicativo IMS, você estabelece um ponto de sincronização usando chamadas do IMS, como GU (get unique) para o IOPCB e o CHKP (checkpoint).

Para voltar todas as mudanças desde o ponto de verificação anterior, é possível usar a chamada IMS ROLB (rollback). Para obter informações adicionais, consulte a seguinte documentação:

- IMS 13 Programação de Aplicativos APG SC19-3646
- IMS 13 APIs de Programação de Aplicativos APR SC19-3647

O gerenciador de filas é um participante de um protocolo two-phase commit; o gerenciador de ponto de sincronização do IMS é o coordenador.

Todos os identificadores abertos são fechados pelo adaptador do IMS em um ponto de sincronização (exceto em um ambiente BMP de lote ou não acionado por mensagem). Isso ocorre porque um usuário diferente poderia iniciar a próxima unidade de trabalho e a verificação de segurança do IBM MQ é executada quando as chamadas MQCONN, MQCONNX e MQOPEN são feitas, não quando as chamadas MQPUT ou MQGET são feitas.

No entanto, em uma ambiente Wait-for-Input (WFI) ou pseudo Wait-for-Input (PWFI), o IMS não notifica o IBM MQ para fechar os identificadores até a próxima mensagem chegar ou um código de status QC ser retornado ao aplicativo. Se o aplicativo estiver esperando na região do IMS e qualquer um desses identificadores pertencer a filas acionadas, o acionamento não ocorrerá porque as filas estão abertas. Por essa razão, os aplicativos em execução em um ambiente WFI ou PWFI devem explicitamente MQCLOSE os identificadores de fila antes de executar GU no IOPCB para a próxima mensagem.

Se um aplicativo IMS (um BMP ou um MPP) emitir a chamada MQDISC, as filas abertas serão fechadas mas nenhum ponto de sincronização implícito será tomado. Se o aplicativo finalizar normalmente, quaisquer filas abertas serão fechadas e uma confirmação implícita ocorrerá. Se o aplicativo finalizar de forma anormal, quaisquer filas abertas serão fechadas e uma restauração implícita ocorrerá.

Chamadas MQI em aplicativos IMS
Use estas informações para aprender sobre o uso de chamadas MQI em aplicativos Server e Enquiry.

Esta seção abrange o uso de chamadas de MQI nos tipos de aplicativos IMS a seguir:

- "Aplicativos do servidor" na página 900
- "Aplicativos de consulta" na página 902

Aplicativos do servidor

Aqui está um esboço do modelo de servidor de aplicativos MQI:

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

O programa de amostra CSQ4ICB3 mostra a implementação, em C/370, de um BMP usando este modelo. O programa estabelece comunicação com o IMS primeiro e, em seguida, com o 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
```

A inicialização do IMS determina se o programa foi chamado como um BMP direcionado por mensagens ou orientado por lote e controla as manipulações de fila e conexão do gerenciador de filas do IBM MQ adequadamente:

```
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
```

A inicialização do IBM MQ se conecta ao gerenciador de filas e abre as filas. Em um BMP orientado a mensagens, isso é chamado após cada ponto de sincronização do IMS ser obtido; em um BMP orientado para lote, isso é chamado apenas durante a inicialização do programa:

```
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
```

A implementação do modelo de servidor em um MPP é influenciada pelo fato de que o MPP processa uma única unidade de trabalho por chamada. Isso ocorre porque, quando um ponto de sincronização (GU) é obtido, a conexão e as manipulações de filas estão fechadas e a próxima mensagem do IMS é entregue. Essa limitação pode ser superada parcialmente por um dos seguintes itens:

- **Processamento de muitas mensagens em uma única unidade de trabalho**

Isso envolve:

- Lendo uma mensagem
- Processar as atualizações necessárias
- Colocar a resposta

em um loop até que todas as mensagens tenham sido processadas ou até que um número máximo configurado de mensagens tenha sido processado, no momento em que um ponto de sincronização é obtido.

Apenas determinados tipos de aplicativo (por exemplo, uma atualização simples do banco de dados ou consulta) podem ser abordados dessa forma. Embora as mensagens de resposta de MQI possam ser colocadas com a autoridade do originador da mensagem MQI que está sendo tratada, as implicações de segurança de quaisquer atualizações de recurso do IMS precisam ser tratadas com cuidado.

- **Processamento de uma mensagem por chamada do MPP e assegurar o planejamento múltiplo de MPP para processar todas as mensagens disponíveis.**

  Use o IBM MQ IMS trigger monitor program (CSQQTRMN) para planejar a transação de MPP quando houver mensagens na fila do IBM MQ e nenhum aplicativo atendendo.

  Se o monitor acionador iniciar o MPP, o nome do gerenciador de filas e o nome da fila serão transmitidos para o programa, conforme mostrado no trecho de código COBOL a seguir:

```
* 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 '='.
```

O modelo do servidor, que espera-se ser uma tarefa de longa execução, é melhor suportado em uma região de processamento de lote, embora o BMP não possa ser acionado usando CSQQTRMN.

Aplicativos de consulta

Um aplicativo IBM MQ típico que inicia uma consulta ou atualização funciona da seguinte forma:

- Reúna dados do usuário
- Coloque uma ou mais mensagens do IBM MQ
- Obtenha as mensagens de resposta (você pode ter que esperar por elas)
- Forneça uma resposta ao usuário

Como as mensagens colocadas nas filas do IBM MQ não são disponibilizadas a outros aplicativos IBM MQ até serem confirmadas, elas devem ser colocadas fora do ponto de sincronização ou o aplicativo IMS deve ser dividido em duas transações.

Se a consulta envolver a colocação de uma única mensagem, é possível usar a opção *sem ponto de sincronização*; no entanto, se a consulta for mais complexa ou atualizações de recursos estiverem

envolvidas, você poderá ter problemas de consistência, se a falha ocorrer e você não usar ponto de sincronização.

Para resolver isso, é possível dividir transações do IMS MPP usando chamadas MQI com um comutador de mensagem de programa para programa; consulte *IMS/ESA Programação de aplicativo: comunicação de dados* para obter informações sobre isso. Isso permite que um programa de consulta seja implementado em um 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
```

Escrevendo aplicativos de ponte IMS

Este tópico contém informações sobre como escrever aplicativos para usar a ponte IBM MQ - IMS.

Para obter informações sobre a ponte IBM MQ - IMS, consulte A ponte IMS.

Use os links a seguir para descobrir mais sobre como escrever aplicativos de ponte IMS no IBM MQ for z/OS:

- “Como a ponte IMS lida com mensagens” na página 67
- “Escrevendo programas de transação do IMS por meio do IBM MQ” na página 910

Conceitos relacionados

“Gravando aplicativos do IMS usando o IBM MQ” na página 63
Há considerações adicionais ao usar o IBM MQ em aplicativos do IMS. Estas incluem chamadas API do MQ que podem ser usadas e o mecanismo usado para o ponto de sincronização.

Como a ponte IMS lida com mensagens

Ao usar a ponte IBM MQ - IMS para enviar mensagens para um aplicativo IMS, é necessário construir suas mensagens em um formato especial.

Deve-se colocar suas mensagens também nas filas do IBM MQ que foram definidas com uma classe de armazenamento que especifica o grupo XCF e o nome do membro do sistema IMS de destino. Elas são conhecidas como filas de ponte MQ-IMS ou simplesmente filas de **ponte**.

A ponte do IBM MQ-IMS vai requerer acesso de entrada exclusivo (MQOO_INPUT_EXCLUSIVE) para a fila de pontes se ela for definida com QSGDISP(QMGR) ou se for definida com QSGDISP(SHARED) juntamente com a opção NOSHARE.

Um usuário não precisa se conectar ao IMS antes de enviar mensagens para um aplicativo IMS. O ID do usuário no campo *UserIdentifier* da estrutura MQMD é usado para verificação de segurança. O nível de verificação é determinado quando o IBM MQ se conecta ao IMS e está descrito em Controle de acesso de aplicativo para a ponte IMS. Isso permite que uma pseudo conexão seja implementada.

A ponte IBM MQ - IMS aceita os tipos de mensagem a seguir:

- Mensagens contendo os dados de transação do IMS e uma estrutura MQIIH (descrita em MQIIH):

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

Nota:

1. Os colchetes, [ ], representam diversos segmentos opcionais.
2. Configure o campo *Format* da estrutura MQMD para MQFMT_IMS para usar a estrutura MQIIH.

- Mensagens contendo dados de transação do IMS, mas nenhuma estrutura MQIIH:

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

O IBM MQ valida os dados da mensagem para assegurar que a soma dos bytes LL mais o comprimento de MQIIH (se estiver presente) seja igual ao comprimento da mensagem.

Quando a ponte IBM MQ - IMS obtém mensagens das filas de ponte, ela processa as mesmas da seguinte forma:

- Se a mensagem contiver uma estrutura MQIIH, a ponte verifica MQIIH (consulte MQIIH), constrói os cabeçalhos OTMA e envia a mensagem para o IMS. O código de transação for especificado na mensagem de entrada. Se for um LTERM, o IMS responde com uma mensagem DFS1288E. Se o código de transação representar um comando, o IMS executa o comando; caso contrário, a mensagem será enfileirada no IMS para a transação.
- Se a mensagem contiver dados de transação do IMS, mas nenhuma estrutura MQIIH, a ponte IMS faz as suposições a seguir:
  - O código de transação está nos bytes 5 a 12 dos dados do usuário
  - A transação está em modo de não conversação
  - A transação está em modo de confirmação 0 (commit-then-send)
  - O *Format* no MQMD é usado como o *MFSMapName* (na entrada)
  - O modo de segurança é MQISS_CHECK

  A mensagem de resposta também é construída sem uma estrutura MQIIH, usando o *Format* para o MQMD do *MFSMapName* da saída do IMS.

A ponte IBM MQ - IMS usa um ou dois Tpipes para cada fila do IBM MQ:

- Um Tpipe sincronizado é usado para todas as mensagens usando o modo de Confirmação 0 (COMMIT_THEN_SEND) (elas são mostradas com SYN no campo de status do comando IMS /DIS TMEMBER client TPIPE xxxx)
- Um Tpipe não sincronizado é usado para todas as mensagens que estiverem usando o modo de Confirmação 1 (SEND_THEN_COMMIT)

Os Tpipes são criados pelo IBM MQ quando são usados pela primeira vez. Um Tpipe não sincronizado existe até que o IMS seja reiniciado. Tpipes sincronizados existem até o IMS passar por cold start. Não é possível excluir esses Tpipes.

Consulte os tópicos a seguir para obter informações adicionais sobre como a ponte IBM MQ - IMS lida com mensagens:

- "Mapeando mensagens do IBM MQ para tipos de transação do IMS" na página 69
- "Se a mensagem não puder ser colocada na fila do IMS" na página 69
- "Códigos de feedback de ponte IMS" na página 70
- "Os campos de MQMD em mensagens da ponte IMS" na página 70
- "O campos de MQIIH em mensagens da ponte IMS" na página 71
- "Mensagens de resposta a partir do IMS" na página 72
- "Usando PCBs de resposta alternativos em transações do IMS" na página 72

- “Enviando mensagens não solicitadas a partir do IMS” na página 73
- “Segmentação de mensagem” na página 73
- “Conversão de Dados” na página 73

Conceitos relacionados

“Escrevendo programas de transação do IMS por meio do IBM MQ” na página 910
A codificação necessária para manipular transações do IMS por meio do IBM MQ depende do formato de mensagem necessário para a transação do IMS e do intervalo de respostas que pode retornar. No entanto, há vários pontos a considerar quando seu aplicativo manipula as informações de formatação de tela do IMS.

Mapeando mensagens do IBM MQ para tipos de transação do IMS
Uma tabela que descreve o mapeamento das mensagens do IBM MQ para tipos de transação do IMS.

Tabela 117. Mapeando mensagens do IBM MQ para tipos de transação do IMS

tipo de mensagem doIBM MQ	**Commit-then-send (modo 0) - usa Tpipes do IMS sincronizados**	**Send-then-commit (modo 1) – usa Tpipes do IMS não sincronizados**
Mensagens persistentes do IBM MQ	• Transações de função integral recuperáveis • Transações irrecuperáveis são rejeitadas pelo IMS	• Transações Fastpath • Transações conversacionais • Transações de função integral
Mensagens do IBM MQ não persistentes	• Transações de função integral irrecuperáveis • As transações recuperáveis são permitidas com o IMS V8, o APAR PQ61404 e todas as versões mais recentes do IMS	• Transações Fastpath • Transações conversacionais • Transações de função integral

Nota: Comandos IMS não podem usar mensagens IBM MQ persistentes com o modo confirmar 0. Consulte o *IMS/ESA Guia do usuário de Abrir o Acesso ao Gerenciador de Transações* para obter mais informações.

Se a mensagem não puder ser colocada na fila do IMS
Aprenda sobre ações a serem executadas se a mensagem não puder ser colocada na fila do IMS.

Se a mensagem não puder ser colocada na fila do IMS, a ação a seguir será executada pelo IBM MQ:

- Se uma mensagem não puder ser colocada no IMS porque a mensagem é inválida, a mensagem será colocada na fila de mensagens não entregues e uma mensagem será enviada ao console do sistema.
- Se a mensagem for válida, mas for rejeitada pelo IMS, o IBM MQ enviará uma mensagem de erro ao console do sistema, a mensagem inclui o sense code do IMS e a mensagem do IBM MQ é colocada na fila de mensagens não entregues. Se o sense code do IMS for 001A, o IMS envia uma mensagem do IBM MQ que contém a razão da falha para a fila de resposta.

  Nota: Nas circunstâncias listadas anteriormente, se o IBM MQ não puder colocar a mensagem na fila de mensagens não entregues, a mensagem será retornada para a fila de origem do IBM MQ. Uma mensagem de erro é enviada ao console do sistema e nenhuma mensagem adicional é enviada a partir dessa fila.

  Para reenviar as mensagens, execute ***um*** dos seguintes:

  – Pare e reinicie os Tpipes no IMS que correspondem à fila
  – Mude a fila para GET(DISABLED) e novamente para GET(ENABLED)
  – Pare e reinicie o IMS ou o OTMA
  – Pare e reinicie o subsistema IBM MQ

- Se a mensagem for rejeitada pelo IMS para algo diferente de um erro da mensagem, a mensagem do IBM MQ é retornada à fila de origem, o IBM MQ para o processamento da fila e uma mensagem de erro é enviada ao console do sistema.

  Se uma mensagem de relatório de exceções for necessária, a ponte a coloca na fila de resposta com a autoridade do originador. Se a mensagem não puder ser colocada na fila, a mensagem de relatório será colocada na fila de mensagens não entregues com a autoridade da ponte. Se ela não puder ser colocada na DLQ, ela será descartada.

Códigos de feedback de ponte IMS
Os sense codes do IMS são geralmente saída no formato hexadecimal em mensagens do console do IBM MQ, como CSQ2001I (por exemplo, sense code 0x001F). Os códigos de feedback do IBM MQ, conforme visto no cabeçalho de mensagens não entregues colocadas em fila de mensagens não entregues, são números decimais.

Os códigos de feedback de ponte IMS estão no intervalo de 301 a 399 ou de 600 a 855 para sense code 0x001A NACK. Eles são mapeados dos sense codes IMS-OTMA da seguinte forma:

1. O sense code IMS-OTMA é convertido de um número hexadecimal para um número decimal.
2. 300 é incluído ao número resultante do cálculo em 1, fornecendo o código do IBM MQ *Feedback*.
3. O sense code IMS-OTMA 0x001A, decimal 26 é um caso especial. Um código *Feedback* no intervalo de 600 a 855 é gerado.
   a. O código de razão IMS-OTMA é convertido de um número hexadecimal para um número decimal.
   b. 600 é incluído ao número resultante do cálculo em a, fornecendo o código do IBM MQ *Feedback*.

Para obter informações sobre sense codes IMS-OTMA, consulte Sense codes OTMA para mensagens NAK.

Os campos de MQMD em mensagens da ponte IMS
Conheça os campos MQMD em mensagens a partir da ponte IMS.

O MQMD da mensagem de origem é executado por IMS na seção Dados do Usuário dos cabeçalhos OTMA. Se a mensagem se originar no IMS, isso será construído pela Saída de Resolução de Destino do IMS. O MQMD de uma mensagem recebida do IMS é construído da seguinte maneira:

StrucID
" " MD

Versão
MQMD_VERSION_1

Relatório
MQRO_NONE

MsgType
MQMT_REPLY

Expiração
Se MQIIH_PASS_EXPIRATION é definido no campo Sinalizadores da MQIIH, este campo contém o tempo de expiração restante, caso contrário ele é configurado como MQEI_UNLIMITED

Feedback
MQFB_NONE

Codificação
MQENC.Native (a codificação do sistema z/OS)

CodedCharSetId
MQCCSI_Q_MGR (o CodedCharSetID do sistema z/OS)

Formato
MQFMT_IMS se o MQMD.Format da mensagem de entrada é MQFMT_IMS, caso contrário, IOPCB.MODNAME

Priority
MQMD.Priority da mensagem de entrada

Persistence
Depende no modo de confirmação: MQMD.Persistence da mensagem de entrada se a persistência de CM-1; corresponder à capacidade de recuperação do IMS se CM-0

MsgId
MQMD.MsgId se MQRO_PASS_MSG_ID, caso contrário, Nova MsgId (o padrão)

ID de Correlação
MQMD.CorrelId da mensagem de entrada se MQRO_PASS_CORREL_ID, caso contrário, MQMD.MsgId da mensagem de entrada (o padrão)

BackoutCount
0

ReplyToQ
Espaços em branco

ReplyToQMgr
Espaços (configurado como nome de qmgr local pelo gerenciador de filas durante o MQPUT)

UserIdentifier
MQMD.UserIdentifier da mensagem de entrada

AccountingToken
MQMD.AccountingToken da mensagem de entrada

ApplIdentityData
MQMD.ApplIdentityData da mensagem de entrada

PutApplType
MQAT_XCF se nenhum erro, caso contrário, MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> se nenhum erro, caso contrário, nome do QMGR

PutDate
Data quando a mensagem foi colocada

PutTime
Hora quando a mensagem foi colocada

ApplOriginData
Espaços em branco

O campos de MQIIH em mensagens da ponte IMS
Aprenda sobre os campos MQIIH em mensagens da ponte IMS.

O MQIIH de uma mensagem recebida do IMS é construído como segue:

StrucId
"IIH "

Versão
1

StrucLength
84

Codificação
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Formato
MQIIH.ReplyToFormat da mensagem de entrada, se MQIIH.ReplyToFormat não estiver em branco, caso contrário, IOPCB.MODNAME

Sinalizadores
0

LTermOverride
nome de LTERM (de Tpipe) do cabeçalho do OTMA

MFSMapName
nome do Mapa do cabeçalho OTMA

ReplyToFormat
Espaços em branco

Autenticador
MQIIH.Authenticator da mensagem de entrada, se a mensagem de resposta está sendo colocada em uma fila de ponte MQ-IMS, caso contrário, em branco.

TranInstanceId
ID da conversa/Token do servidor do cabeçalho do OTMA, se ele estiver na conversa. Nas versões do IMS anteriores à V14, esse campo sempre seria nulo se não estivesse na conversa. Do IMS V14 em diante, esse campo pode ser configurado pelo IMS, mesmo que não esteja na conversa.

TranState
"C" se na conversa, caso contrário, em branco

CommitMode
modo de Commit a partir do cabeçalho OTMA ("0" ou "1")

SecurityScope
Em branco

Reservado
Em branco

Mensagens de resposta a partir do IMS
Quando uma transação do IMS ISRTs para seu IOPCB, a mensagem é roteada de volta para o LTERM ou o TPIPE de origem.

Estas são vistas no IBM MQ como mensagens de resposta. Mensagens de resposta do IMS são colocadas na fila de resposta especificada na mensagem original. Se a mensagem não puder ser colocada na fila de resposta, ela será colocada na fila de mensagens não entregues usando a autoridade da ponte. Se a mensagem não puder ser colocada na fila de mensagens não entregues, um reconhecimento negativo será enviado para IMS para dizer que a mensagem não pode ser recebida. A responsabilidade pela mensagem é então retornada ao IMS. Se estiver usando o modo de confirmação 0, mensagens desse Tpipe não serão enviadas à ponte e permanecerão na fila IMS; ou seja, nenhuma mensagem adicional será enviada até a reinicialização. Se estiver sendo usado o modo de confirmação 1, outros trabalhos poderão continuar.

Se a resposta tiver uma estrutura MQIIH, seu tipo de formato será MQFMT_IMS; se não, seu tipo de formato será especificado pelo nome do MOD IMS usado ao inserir a mensagem.

Usando PCBs de resposta alternativos em transações do IMS
Quando uma transação IMS usa PCBs de resposta alternativa (ISRTs para o ALTPCB ou emite uma chamada CHNG para um PCB modificável), a saída de pré-roteamento (DFSYPRX0) é chamada para determinar se a mensagem deve ser redirecionada.

Se a mensagem tiver que ser roteada novamente, a saída de resolução de destino (DFSYDRU0) será chamada para confirmar o destino e preparar as informações de cabeçalho. Consulte Usando saídas do OTMA no IMS e A saída de pré-roteamento DFSYPRX0 para obter informações sobre esses programas de saída.

A menos que seja tomada ação nas saídas, toda saída de transações do IMS iniciada a partir de um gerenciador de filas do IBM MQ, seja para o IOPCB ou para um ALTPCB, será retornada ao mesmo gerenciador de filas.

Enviando mensagens não solicitadas a partir do IMS
Para enviar mensagens do IMS para uma fila IBM MQ, é necessário chamar uma transação IMS que ISRTs para um ALTPCB.

É necessário escrever saídas de gravar pré-roteamento e de resolução de destino para rotear mensagens não solicitadas a partir do IMS e construir os dados do usuário do OTMA, de modo que o MQMD da mensagem possa ser construído corretamente. Consulte O saída de pré-roteamento DFSYPRX0 e A saída do usuário de resolução de destino para obter informações sobre esses programas de saída.

Nota: A ponte IBM MQ - IMS não sabe se uma mensagem que recebe é uma resposta ou uma mensagem não solicitada. Ela manipula a mensagem da mesma maneira em cada caso, construindo o MQMD e o MQIIH da resposta com base no UserData do OTMA que chegou com a mensagem

Mensagens não solicitadas podem criar novos Tpipes. Por exemplo, se uma transação existente do IMS tiver mudado para um novo LTERM (por exemplo, PRINT01), mas a implementação requerer que a saída seja entregue por meio do OTMA, um novo Tpipe (chamado PRINT01, neste exemplo) será criado. Por padrão, este é um Tpipe não sincronizado. Se a implementação requerer que a mensagem seja recuperável, configure a sinalização da saída de resolução de destino. Consulte o *Guia de customização do IMS* para obter mais informações.

Segmentação de mensagem
É possível definir IMS transações como esperando entradas de um único segmento ou de vários segmentos.

O aplicativo IBM MQ de origem deve construir a entrada do usuário seguindo a estrutura MQIIH como um ou mais segmentos de dados LLZZ. Todos os segmentos de uma mensagem do IMS devem estar contidos em uma única mensagem do IBM MQ enviada com um único MQPUT.

O comprimento máximo de um segmento de dados LLZZ é definido por IMS/OTMA (32767 bytes). O comprimento total da mensagem do IBM MQ é a soma dos bytes LL, mais o comprimento da estrutura MQIIH.

Todos os segmentos da resposta estão contidos em uma única mensagem do IBM MQ.

Há uma restrição adicional na limitação de 32 KB em mensagens com formato MQFMT_IMS_VAR_STRING. Quando os dados em uma mensagem CCSID de ASCII combinado são convertidos para uma mensagem CCSID de EBCDIC combinado, um byte shift-in ou shift-out é incluído toda cada vez que há uma transição entre os caracteres de SBCS e DBCS. A restrição de 32 KB se aplica ao tamanho máximo da mensagem. Ou seja, como o campo LL na mensagem não pode exceder 32 KB, a mensagem não deve exceder 32 KB incluindo todos os caracteres shift-in e shift-out. O aplicativo que está construindo a mensagem deve permitir isso.

Conversão de Dados
A conversão de dados é executada pelo recurso de enfileiramento distribuído (que pode chamar quaisquer saídas necessárias) ou pelo agente de enfileiramento intragrupo (que não suporta o uso de saídas) quando coloca uma mensagem em uma fila de destino que possui informações XCF definidas para sua classe de armazenamento. A conversão de dados não ocorre quando uma mensagem é entregue a uma fila por publicação/assinatura.

Todas as saídas necessárias devem estar disponíveis para o recurso de enfileiramento distribuído no conjunto de dados referido pela instrução CSQXLIB DD. Isso significa que é possível enviar mensagens para um aplicativo IMS usando a ponte IBM MQ - IMS de qualquer plataforma IBM MQ.

Se houver erros de conversão, a mensagem será colocada na fila não convertida; isso resulta, finalmente, no fato de ela ser tratada como um erro pela ponte IBM MQ - IMS, pois a ponte não pode reconhecer o formato de cabeçalho. Se ocorrer um erro de conversão, uma mensagem de erro será enviada ao console do z/OS.

Consulte “Escrevendo saídas de conversão de dados” na página 988 para obter informações detalhadas sobre a conversão de dados em geral.

Enviando mensagens para a ponte IBM MQ - IMS

Para assegurar que a conversão seja executada corretamente, deve-se informar ao gerenciador de filas qual é o formato da mensagem.

Se a mensagem tiver uma estrutura MQIIH, o *Format* no MQMD deve ser configurado como o formato MQFMT_IMS integrado e o *Formato* no MQIIH deve ser configurado como o nome do formato que descreve seus dados da mensagem. Se não houver MQIIH, configure o *Format* no MQMD como seu nome de formato.

Se seus dados (diferente da LLZZs) forem todos dados de caracteres (MQCHAR), use como seu nome de formato (no MQIIH ou MQMD, conforme apropriado) o formato MQFMT_IMS_VAR_STRING integrado. Caso contrário, use seu próprio nome de formato, nesse caso deve-se também fornecer uma saída de conversão de dados para o seu formato. A saída deve identificar a conversão de LLZZs em sua mensagem, além dos dados em si (mas ela não precisa identificar nenhum MQIIH no início da mensagem).

Se seu aplicativo usar *MFSMapName*, será possível usar mensagens com o MQFMT_IMS em vez disso e definir o nome de mapa transmitido para a transação do IMS no campo MFSMapName da MQIIH.

Recebendo mensagens da ponte IBM MQ - IMS

Se uma estrutura MQIIH estiver presente na mensagem original que você está enviando ao IMS, uma também estará presente na mensagem de resposta.

Para assegurar que sua resposta seja convertida corretamente:

- Se você tiver uma estrutura MQIIH em sua mensagem original, especifique o formato desejado para sua mensagem de resposta no campo *ReplytoFormat* de MQIIH da mensagem original. Este valor é colocado no campo *Format* de MQIIH da mensagem de resposta. Isso será particularmente útil se todos os dados de saída estiverem no formato LLZZ<dados de caracteres>.
- Se você não tiver uma estrutura MQIIH em sua mensagem original, especifique o formato desejado para a mensagem de resposta como o nome MFS MOD no ISRT do aplicativo IMS para o IOPCB.

Escrevendo programas de transação do IMS por meio do IBM MQ
A codificação necessária para manipular transações do IMS por meio do IBM MQ depende do formato de mensagem necessário para a transação do IMS e do intervalo de respostas que pode retornar. No entanto, há vários pontos a considerar quando seu aplicativo manipula as informações de formatação de tela do IMS.

Quando uma transação do IMS é iniciado a partir de uma tela 3270, a mensagem passa pelos Serviços de formato de mensagens do IMS. Isso pode remover todas as dependências do terminal a partir do fluxo de dados visto pela transação. Quando uma transação é iniciada por meio do OTMA, o MFS não está envolvido. Se a lógica do aplicativo for implementada no MFS, ela deve ser recriada no novo aplicativo.

Em algumas transações do IMS, o aplicativo de usuário final pode modificar determinados comportamentos da tela 3270, por exemplo, destacar um campo que teve dados inválidos inseridos. Esse tipo de informação é comunicado pela inclusão de um campo de atributo de dois bytes na mensagem do IMS para cada campo da tela que precisa ser modificado pelo programa.

Portanto, se você estiver codificando um aplicativo para imitar um 3270, precisa levar em consideração esses campos quando construindo ou recebendo mensagens.

Pode ser necessário codificar informações em seu programa para processar:

- Qual tecla é pressionada (por exemplo, Enter e PF1)
- Onde o cursor está quando a mensagem é passada para seu aplicativo
- Se os campos de atributos foram configurados pelo aplicativo IMS
  - Intensidade alta, normal ou zero
  - Cor
  - Se o IMS está esperando que o campo de volta na próxima vez que for pressionado Enter
- Se o aplicativo IMS usou caracteres nulos (X'3F') em algum campo.

Se sua mensagem do IMS contiver somente dados de caracteres (além do segmento de dados LLZZ) e você estiver usando uma estrutura MQIIH, configure o formato MQMD para MQFMT_IMS e o formato MQIIH para MQFMT_IMS_VAR_STRING.

Se a mensagem do IMS contiver apenas dados de caractere (além do segmento de dados LLZZ) e você **não** estiver usando uma estrutura MQIIH, configure o formato MQMD como MQFMT_IMS_VAR_STRING e assegure-se de que o aplicativo IMS especifique MODname MQFMT_IMS_VAR_STRING ao responder. Se ocorrer um problema (por exemplo, usuário não autorizado a usar a transação) e o IMS envia uma mensagem de erro, essa tem um MODname do formulário DFSMOx, em que x é um número no intervalo de 1 a 5. Isto é colocado no MQMD.Format.

Se a mensagem do IMS contiver dados binários, compactados ou de vírgula flutuante (além do segmento de dados LLZZ), codifique suas próprias rotinas de conversão de dados. Consulte *Programação de aplicativos do IMS/ESA: Transaction Manager* para obter informações sobre formatação de tela do IMS.

Considere os tópicos a seguir ao escrever código para manipular transações do IMS por meio do IBM MQ.

- "Escrevendo aplicativos IBM MQ para chamar transações de conversação do IMS" na página 911
- "Escrevendo programas que contém os comandos do IMS" na página 911
- "Acionando" na página 912

Escrevendo aplicativos IBM MQ para chamar transações de conversação do IMS

Use estas informações como um guia para considerações ao escrever o aplicativo IBM MQ para chamar transações de conversações do IMS.

Ao escrever um aplicativo que chame uma conversa do IMS, considere o seguinte:

- Inclua uma estrutura MQIIH com seu aplicativo de mensagens.
- Configure *CommitMode* em MQIIH para MQICM_SEND_THEN_COMMIT.
- Para chamar uma nova conversa, configure *TranState* em MQIIH para MQITS_NOT_IN_CONVERSATION.
- Para chamar a segunda etapa e as subsequentes de uma conversa, configure *TranState* para MQITS_IN_CONVERSATION e configure *TranInstanceId* para o valor desse campo retornado na etapa anterior da conversa.
- Não há uma maneira fácil no IMS para localizar o valor de um *TranInstanceId*, caso você perca a mensagem original enviada a partir do IMS.
- O aplicativo deve verificar o *TranState* de mensagens do IMS para verificar se a transação do IMS finalizou a conversa.
- É possível usar /EXIT para finalizar uma conversa. Deve-se também citar o *TranInstanceId*, configurar *TranState* para MQITS_IN_CONVERSATION e usar a fila do IBM MQ na qual a conversa está sendo realizada.
- Não é possível usar /HOLD ou /REL para reter ou liberar uma conversa.
- Conversas chamadas por meio da ponte IBM MQ - IMS são finalizadas se o IMS for reiniciado.

Escrevendo programas que contém os comandos do IMS

Um programa de aplicativo pode construir uma mensagem do IBM MQ com o formato LLZZ*command*, em vez de uma transação, em que *command* tem o formato /DIS TRAN PART ou /DIS POOL ALL.

A maioria dos comandos do IMS pode ser emitida dessa maneira; consulte *IMS V11 Communications and Connections* para obter detalhes. A saída de comando é recebida como a mensagem de resposta do IBM MQ no formato de texto como seria enviada para um terminal 3270 para exibição.

OTMA implementou um formato especial do comando da transação de exibição do IMS, que retorna um formulário arquitetado da saída. O formato exato é definido no *IMS V11 Communications and Connections*. Para chamar este formulário a partir de uma mensagem IBM MQ, construa os dados da mensagem como antes, por exemplo /DIS TRAN PART, e configure o campo TranState no MQIIH como

MQITS_ARCHITECTED. O IMS processa o comando e retorna a resposta no formulário arquitetado. Uma resposta arquitetada contém todas as informações que poderiam ser localizadas no formato de texto da saída, além de uma informação adicional: se a transação está definida como recuperável ou não recuperável.

Acionando

A ponte IBM MQ - IMS não suporta mensagens do acionador.

Se você definir uma fila de inicialização que usa uma classe de armazenamento com os parâmetros XCF, mensagens colocadas nessa fila serão rejeitadas quando chegarem à ponte.

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 918.

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 920.

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 913.

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 933
- “Preparando e executando aplicativos Microsoft Transaction Server” na página 46
- “Preparando e executando aplicativos IBM MQ JMS” na página 936

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

“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 47
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 715
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 806
Inicie a escrever aplicativos de publicar/assinar IBM MQ.

“Construindo um aplicativo processual” na página 1005

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 1055
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 1075
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 serviços da web com o IBM MQ" na página 1306
É possível desenvolver aplicativos IBM MQ para serviços da web usando o transporte IBM MQ para SOAP.

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 917
É possível usar a chamada MQCONNX para especificar uma estrutura de definição de canal (MQCD) na estrutura MQCNO.

Informações 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. Use 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 348.

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 este CCSID é um CCSID não suportado ou não é o CCSID necessário, ele poderá ser substituído pela variável de ambiente MQCCSID usando um destes comandos:

- Windows

```
SET MQCCSID=850
```

- UNIX

```
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. Veja O que é um cliente transacional estendido? para mais 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:

- Tanto o cliente quanto o gerenciador de filas remotas devem estar no IBM WebSphere MQ 7 ou mais recente.
- 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 informações adicionais, consulte “Melhorando o desempenho de mensagens não persistentes” na página 783

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 1094.

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 informações adicionais, consulte Usando a estrutura MQCNO

em uma chamada MQCONNX. 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 965.

Manipulações de conexões compartilhadas no MQCONNX
É possível compartilhar identificadores entre diferentes encadeamentos dentro do mesmo processo, usando identificadores de conexão compartilhados.

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.

Para obter mais informações, consulte “Usando MQCONNX” na página 917.

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 118. Linguagens de Programação Suportadas nos Ambientes do Cliente

plataforma do cliente	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Sim	Sim	Sim			
HP-UX	Sim	Sim	Sim			
IBM i	Sim		Sim		Sim	
Linux	Sim	Sim	Sim			
Solaris	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.

HP-UX **HP-UX**
A biblioteca libmqic.sl para aplicativos não encadeados ou a biblioteca libmqic_r.sl 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.

Solaris **Solaris**
libmqic.so.

Se desejar usar os programas em uma máquina que tenha somente o IBM MQ MQI client for Solaris instalado, deve-se recompilar os programas para vinculá-los à biblioteca do cliente:

```
$ /opt/SUNWspro/bin/cc -o prog_name prog_name c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

Os parâmetros devem ser inseridos na ordem correta, conforme mostrado.

Windows **Windows**
MQIC32.LIB.

ULW *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.

HP-UX **HP-UX**
Vincule seu aplicativo COBOL não encadeado com a biblioteca libmqicb.sl ou o aplicativo COBOL encadeado com libmqicb_r.sl.

HP-UX **Linux**
Vincule seu aplicativo COBOL não encadeado com a biblioteca libmqicb.so ou o aplicativo COBOL encadeado com libmqicb_r.so.

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.

Solaris **Solaris**
Vincule seu aplicativo COBOL não encadeado com a biblioteca libmqicb.so ou o aplicativo COBOL encadeado com libmqicb_r.so.

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.

V 9.0.0 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 da dll do cliente:

MQIC32.dll
Windows 7, Windows 8, Windows 2008 e Windows 2012

Conceitos relacionados

"Codificação no Visual Basic" na página 1070
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 1037
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 999.
2. Se a variável de ambiente MQSERVER for configurada, o canal que ela definir será usado.
3. Se um arquivo `mqclient.ini` estiver definido e contiver um ServerConnectionParms, o canal que ele definir será usado. Para obter mais informações, consulte Configurando um cliente usando um arquivo de configuração e Sub-rotina CHANNELS do arquivo de configuração do cliente.
4. Se as variáveis de ambiente MQCHLLIB e MQCHLTAB forem configuradas, a tabela de definição de canal de cliente para a qual elas apontarem será usada. V 9.0.0 Em alternativa, a partir de IBM MQ 9.0, 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 de cliente para a qual ele apontar será usada. Para obter mais informações, veja Acesso de endereço da web à tabela de definição de canal do cliente.
5. Se um arquivo `mqclient.ini` estiver definido e contiver atributos ChannelDefinitionDirectory e ChannelDefinitionFile, esses atributos serão usados para localizar a tabela de definição de canal de cliente. Para obter mais informações, consulte Configurando um cliente usando um arquivo de configuração e Sub-rotina CHANNELS do arquivo de configuração do cliente.
6. Finalmente, se as variáveis de ambiente não estiverem configuradas, o cliente procurará por uma tabela de definição de canal de cliente com um caminho e nome que são estabelecidos no `DefaultPrefix` no arquivo `mqs.ini`. Se a procura por uma tabela de definição de canal de cliente falhar, o cliente usará os caminhos a seguir:
   - Linux UNIX No UNIX 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`
   - 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.

Informações relacionadas

Tabela de Definições de Canal do Cliente

V 9.0.0 Acesso de endereço da web à tabela de definição de canal de cliente

MQSERVER

MQCHLLIB

MQCHLTAB

V 9.0.0 MQCCDTURL

Configurando as Conexões entre o Servidor e o Cliente

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 MQSERVER é usada para definir um canal mínimo. MQSERVER especifica o local do IBM MQ do servidor 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 definição de canal de 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 que contém a tabela de definição de canal de cliente (CCDT).

V 9.0.0 No IBM MQ 9.0, a variável de ambiente MQCCDTURL fornece o recurso equivalente para configurar uma combinação das variáveis de ambiente MQCHLLIB e MQCHLTAB. O MQCCDTURL permite fornecer uma URL de arquivo, ftp ou http como um valor único do qual uma tabela de definição de canal de 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 Usando a estrutura MQCNO em uma chamada MQCONNX.

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.

V 9.0.0 No IBM MQ 9.0, a CCDT pode ser hospedada em um local central que seja acessível por meio de um URI, removendo a necessidade de atualizar individualmente a CCDT para cada cliente implementado.

Informações relacionadas
Tabela de Definições de Canal do Cliente

V 9.0.0 Acesso de endereço da web à tabela de definição de canal de cliente

Acessando Definições de Canal de Conexão 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 107 na página 924 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.

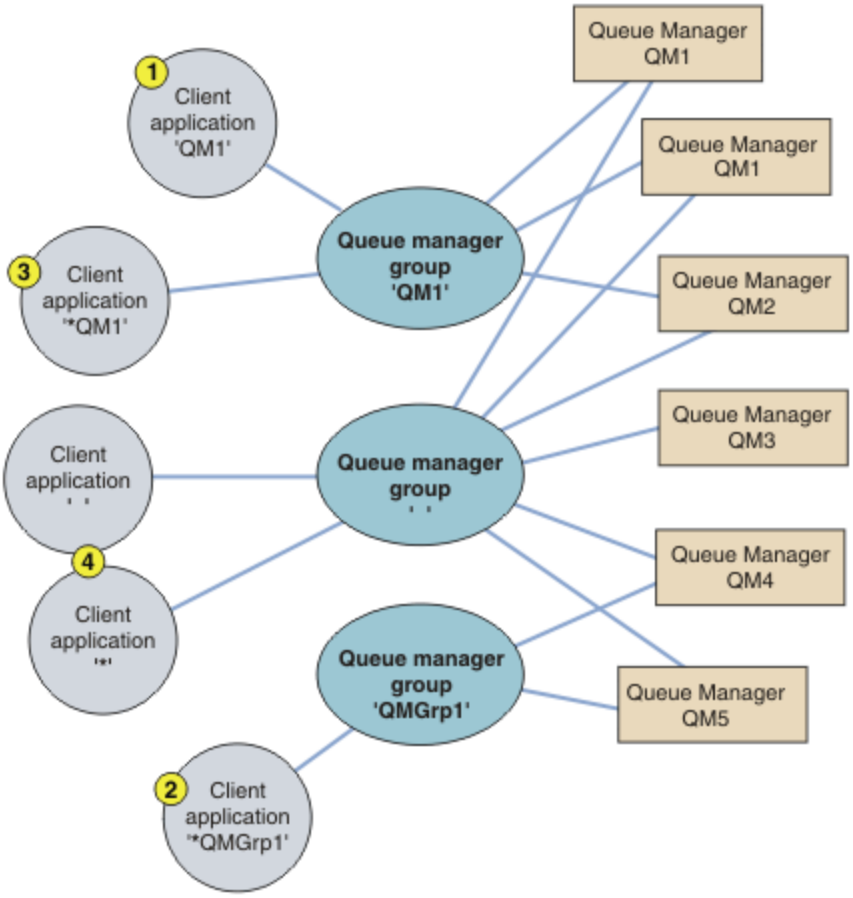


Figura 107. Grupos de gerenciadores de filas

Quatro exemplos de como usar essa tabela de conexões do cliente são descritos com a ajuda dos aplicativos clientes numerados em Figura 107 na página 924.

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 *Conectando-se a grupos de filas compartilhadas*

É possível conectar o seu aplicativo a um gerenciador de filas que faz parte de um grupo de filas compartilhadas. Isso pode ser feito usando o nome do grupo de filas compartilhadas em vez do nome do gerenciador de filas na chamada MQCONN ou MQCONNX.

Os grupos de filas compartilhadas têm um nome de até quatro caracteres. O nome deve ser exclusivo em sua rede e deve ser diferente de qualquer nome de gerenciador de filas.

A definição de canal do cliente deve usar a interface genérica do grupo de filas compartilhadas para conectar-se a um gerenciador de filas disponível no grupo. Para obter mais informações, consulte Conectando um cliente a um grupo de filas compartilhadas. Uma verificação é feita para assegurar que o gerenciador de filas o listener se conecta seja um membro do grupo de filas.

Para obter mais informações sobre as filas compartilhadas, consulte Filas compartilhadas e grupos de compartilhamento de filas.

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.

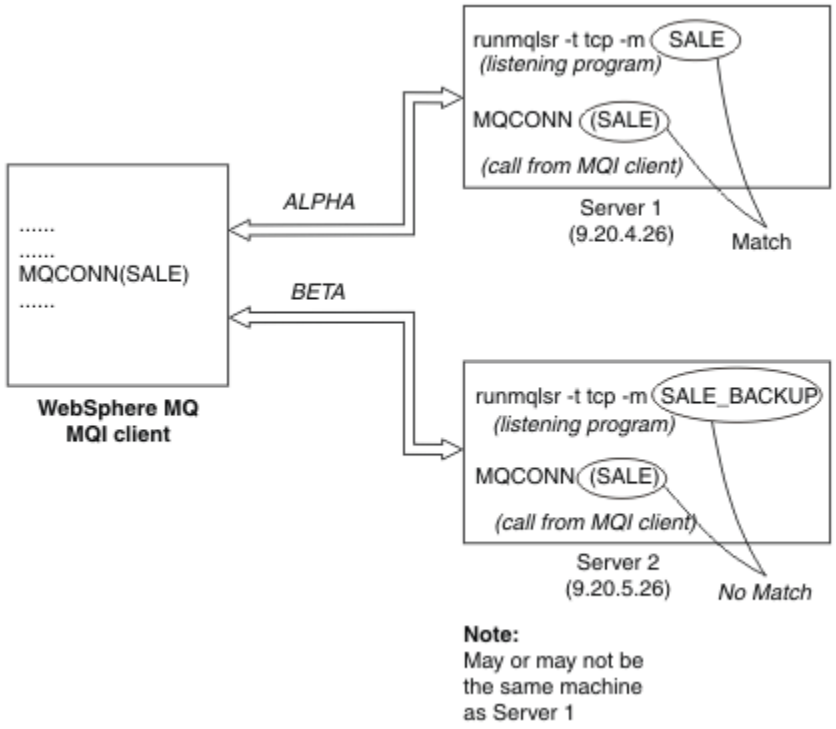


Figura 108. Exemplo de MQCONN

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. O 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 com 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. O nome do gerenciador de filas inclui um asterisco (*)" na página 930.

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. Especificamente, se nenhuma opção de controle do ponto de sincronização MQPMO for especificada em um aplicativo cliente que está colocando mensagens em uma fila acionada que é 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 em execução em um cliente Windows e desejar enviar uma solicitação a um servidor em outro sistema operacional, MQAT_WINDOWS_NT deverá ser definido, caso contrário, o outro sistema operacional usará suas definições padrão e o processo falhará.

Monitor acionador

O monitor acionador fornecido por produtos não z/OSIBM MQ é executado nos ambientes do cliente para os sistemas IBM i IBM i, UNIX, Linux, and Windows.

Para executar o monitor acionador, emita um destes comandos:

- IBM i No IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ULW Nas plataformas Windows, UNIX and Linux:

```
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 UNIX 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.

Windows UNIX 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:
  - UNIX Para aplicativos escritos em C no UNIX, 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 119 na página 934 e Tabela 120 na página 934, 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 119. Bibliotecas do sistema do cliente no UNIX	
Biblioteca para um sistema do servidor IBM MQ	**Biblioteca equivalente para uso em um sistema do cliente IBM MQ**
libmqmxa	libmqcxa

Tabela 120. 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.

Windows UNIX *Programas de amostra CICS e Tuxedo*

Os programas de amostra CICS e Tuxedo para uso nos sistemas UNIX e Windows.

Tabela 121 na página 934 lista os programas de amostra CICS e Tuxedo que são fornecidos para uso nos sistemas do cliente UNIX. Tabela 122 na página 934 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 1097 e “Usando as amostras do TUXEDO no UNIX e Windows” na página 1141.

Tabela 121. Programas de amostra para sistemas cliente UNIX		
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 122. 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	-

Tabela 122. Programas de amostra para sistemas cliente Windows (continuação)		
Descrição	**Origem**	**Módulo executável**
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	-

Windows UNIX *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.

Explanation

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.

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 1034, 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 530, 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 1037, mas quando definir o projeto Visual Basic, digite `MqType=3` no campo **Argumentos de compilação condicional**.
- Se o componente usar o IBM MQ Automation Classes para ActiveX (MQAX), defina uma variável de ambiente, GMQ_MQ_LIB, com o valor `mqic32xa.dll`.

  É possível definir a variável de ambiente a partir de seu aplicativo ou ela pode ser definida de modo que seu escopo seja para todo o sistema. Entretanto, defini-la como para todo o sistema pode causar que qualquer aplicativo MQAX existente que não defina a variável de ambiente a partir do aplicativo se comporte incorretamente.

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 o IBM MQ classes for JMS" na página 75.

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.

Informações relacionadas

Programas de Saída de Canal para Canais MQI

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

ULW Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows

É possível escrever e compilar saídas sem se vincular a nenhuma biblioteca do IBM MQ no UNIX, Linux e Windows.

Sobre esta tarefa

Este tópico se aplica aos sistemas UNIX, 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 UNIX, 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 UNIX 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 957.
   - 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 968.
   - 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 996 e “Escrevendo uma saída de conversão de dados para o IBM MQ em sistemas UNIX and Linux” na página 992.

- 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 988.

- 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 999.
- 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 1000.
- 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 1002.

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 1075.

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 959.
     - Saídas de canal, saídas de publicação, saídas de carga de trabalho do cluster: "Compilando programas de saída de canal em sistemas Windows,UNIX and Linux" na página 986.
     - Saídas de conversão de dados: "Escrevendo saídas de conversão de dados" na página 988.
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 installation2 do IBM MQ usa a saída de /exits/installation2. A instalação installation1 do IBM MQ usa a saída do diretório /exits.

5. Se necessário, configure a saída:
   - Serviços instaláveis: "Configurando serviços e componentes" na página 947.
   - Saídas de API: "Configurando saídas de API" na página 963.
   - Saídas do canal: "Configurando saídas do canal" na página 987.
   - Saídas de publicação: "Configurando saídas de publicação" na página 1002.
   - Saídas de pré-conexão: Sub-rotina PreConnect do arquivo de configuração do cliente.

ULW *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.

No IBM MQ 8.0, **runmqsc** é um exemplo de um programa que não 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

"Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows" na página 936
É possível escrever e compilar saídas sem se vincular a nenhuma biblioteca do IBM MQ no UNIX, Linux e Windows.

ULW Serviços e componentes instaláveis para UNIX, Linux e 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 123 na página 940 resume os serviços e componentes que podem ser usados.

Tabela 123. 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.

Informações 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 944 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 109 na página 941 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.

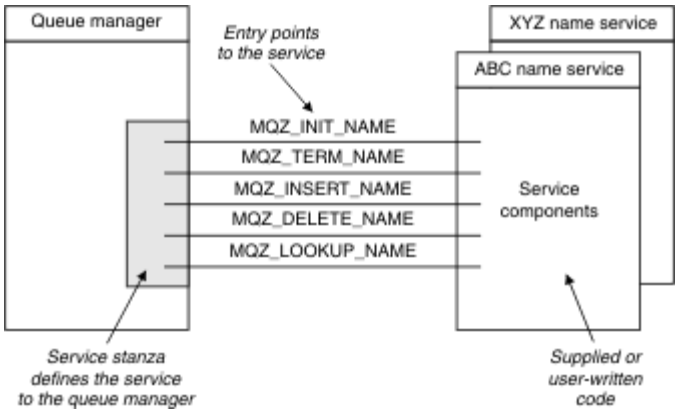


Figura 109. Entendendo serviços, componentes e pontos de entrada

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 UNIX 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 UNIX 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 estão descritos em Referência de comandos de controle do IBM MQ.

O OAM trabalha com a *entidade* de um diretor ou grupo:

- Linux UNIX Nos sistemas UNIX 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 124 na página 947 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 124 na página 947). 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 124. 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

Configure 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.

1. 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.

   Cada serviço usado deve ter uma sub-rotina *`Service`*, que define o serviço para o gerenciador de filas.

   Para cada componente dentro de um serviço, deve haver uma sub-rotina *`ServiceComponent`*. Isso identifica o nome e o caminho do módulo que contém o código para esse componente.

   Para obter mais informações, consulte “Formato da sub-rotina Service” na página 947 e “Formato da sub-rotina do componente de serviço” na página 948

   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.

Formato da sub-rotina Service

A sub-rotina Service contém o nome do serviço e o número de pontos de entrada definidos para o serviço.

O formato da sub-rotina é o seguinte:

```
Service:
  Name=service_name
```

```
    EntryPoints=entries
    SecurityPolicy=policy
```

em que:

service_name
O nome do serviço. Isso é definido pelo serviço.

entries
O número de pontos de entrada definidos para o serviço. Inclui os pontos de entrada de inicialização e finalização.

policy

Linux UNIX Nos sistemas UNIX and Linux: `usuário`, `grupo` ou `padrão`. O valor especifica se o gerenciador de filas usa autorização baseada em usuário ou em grupo. Os valores não fazem distinção entre maiúsculas e minúsculas. Se você não incluir esse atributo, o valor padrão usado será autorização baseada em grupo. Reinicie o gerenciador de filas para que as mudanças sejam efetivadas. Consulte também "Configurando as sub-rotinas do serviço de autorização no UNIX and Linux" na página 949.

Windows Nos sistemas Windows: `NTSIDsRequired` (o Identificador de Segurança do Windows) ou `Padrão`. Se você não especificar `NTSIDsRequired`, o valor `Padrão` será usado. Este atributo é válido apenas se **Name** tiver um valor de `AuthorizationService`. Consulte também "Configurando as sub-rotinas do serviço de autorização no Windows" na página 949.

Formato da sub-rotina do componente de serviço
As sub-rotinas **Service** e **ServiceComponent** podem ocorrer em qualquer ordem.

O formato da sub-rotina do componente de serviço é:

```
ServiceComponent:
  Service=service_name
  Name=component_name
  Module=module_name
  ComponentDataSize=size
```

em que:

service_name
O nome do serviço. Isso deve corresponder ao `Name` especificado em uma sub-rotina de serviço.

component_name
Um nome descritivo do componente de serviço. Isso deve ser exclusivo e conter apenas os caracteres que forem válidos para os nomes de objetos do IBM MQ (por exemplo, nomes de fila). Este nome ocorre em mensagens do operador geradas pelo serviço. Recomendamos que seja usado um nome que comece com uma marca comercial da empresa ou sequência distintiva semelhante.

module_name
O nome do módulo a conter o código para esse componente.

size
O tamanho em bytes da área de dados do componente passado ao componente em cada chamada. Especifique zero se nenhum dado de componente for requerido.

As sub-rotinas **Service** e **ServiceComponent** podem ocorrer em qualquer ordem e as chaves de sub-rotina sob eles também podem ocorrer em qualquer ordem. Para qualquer uma dessas sub-rotinas, todas as chaves de sub-rotina devem estar presentes. Se uma chave de sub-rotina for duplicada, a última será usada.

No momento da inicialização, o gerenciador de filas processa cada entrada do componente de serviço no arquivo de configuração na sua vez. Em seguida, ele carrega o módulo de componente especificado, chamando o ponto de entrada do componente (que deve ser o ponto de entrada para a inicialização do componente), passando a ele um identificador de configuração.

Linux UNIX *Configurando as sub-rotinas do serviço de autorização no UNIX and Linux*

No UNIX and Linux, cada gerenciador de filas tem seu próprio arquivo de configuração do gerenciador de filas.

Por exemplo, o caminho padrão e o nome do arquivo do arquivo de configuração do gerenciador de filas para o gerenciador de filas QMNAME é `/var/mqm/qmgrs/QMNAME/qm.ini`.

As sub-rotinas *Service* e *ServiceComponent* para o componente de autorização padrão são incluídas em `qm.ini` automaticamente, mas podem ser substituídas por `mqsnoaut`. 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 em IBM MQ for AIX. O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

```
Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
Module= MQ_INSTALLATION_PATH/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

Figura 110. sub-rotinas de serviço de autorização UNIX and Linux em qm.ini

A sub-rotina do componente de serviço (`MQSeries.UNIX.auth.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.

Windows *Configurando as sub-rotinas do serviço de autorização no Windows*

No IBM MQ for Windows, cada gerenciador de filas tem sua própria sub-rotina no registro.

A sub-rotina *Service* e a sub-rotina *ServiceComponent* para o componente de autorização padrão são incluídas no registro automaticamente, mas podem ser substituídas usando `mqsnoaut`. Qualquer outra sub-rotina *ServiceComponent* deve ser incluída manualmente.

Também é possível incluir o atributo `SecurityPolicy` usando os serviços do IBM MQ O atributo `SecurityPolicy` se aplicará apenas se o serviço especificado na sub-rotina *Service* for o serviço de autorização, ou seja, o OAM padrão O atributo `SecurityPolicy` permite especificar a política de segurança para cada gerenciador de filas. Os valores possíveis são:

Default
: Especifique `Default` se desejar que a política de segurança padrão tenha efeito. Se um identificador de segurança do Windows (NT SID) não for passado para o OAM para um ID do usuário específico, uma tentativa será feita para obter o SID apropriado procurando os bancos de dados de segurança relevantes.

NTSIDsRequired
: Exige que um SID NT seja passado ao OAM ao executar as verificações de segurança.

Para obter informações sobre o formato da sub-rotina Service, consulte “Formato da sub-rotina Service” na página 947. Para obter mais informações gerais sobre segurança, consulte Configurando a segurança nos sistemas Windows, UNIX and Linux.

A sub-rotina do componente de serviço, `MQSeries.WindowsNT.auth.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.

Linux UNIX *Configurando sub-rotinas de serviço de nomes: sistemas UNIX and Linux*

Nos sistemas UNIX and Linux, cada gerenciador de filas tem seu próprio arquivo de configuração de gerenciador de filas.

Os exemplos a seguir de sub-rotinas no arquivo de configuração do UNIX and Linux para o serviço de nomes especificam um componente de serviço de nomes fornecido pela empresa ABC (fictícia).

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

Figura 111. Sub-rotinas de serviço de nomes em qm.ini (para sistemas UNIX and Linux)

Nota: Windows Nos sistemas Windows, as informações da sub-rotina de serviço de nomes são armazenadas no Registro.

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**.

Informações 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 125 na página 951 resume o suporte para o serviço de autorização.

Tabela 125. 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 apenas em versões de IBM MQ anteriores à 6.0	• 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. Cada serviço usado deve ter uma sub-rotina *Service*, que define o serviço para o gerenciador de filas.

Para cada componente dentro de um serviço, deve haver uma sub-rotina *ServiceComponent*. Isso identifica o nome e o caminho do módulo que contém o código para esse componente.

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).

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.

Formato da sub-rotina Service

O formato da sub-rotina **Service** é:

```
Service:
  Name=service_name
  EntryPoints=entries
```

em que:

service_name
O nome do serviço. Isso é definido pelo serviço.

entradas
O número de pontos de entrada definidos para o serviço. Inclui os pontos de entrada de inicialização e finalização.

Formato da sub-rotina do componente de serviço

O formato da sub-rotina **Service component** é:

```
ServiceComponent:
  Service=service_name
  Name=component_name
  Module=module_name
  ComponentDataSize=size
```

em que:

service_name
O nome do serviço. Isso deve corresponder ao *Name* especificado em uma sub-rotina de serviço.

component_name
Um nome descritivo do componente de serviço. Isso deve ser exclusivo e conter apenas os caracteres que forem válidos para os nomes de objetos do IBM MQ (por exemplo, nomes de fila). Este nome ocorre em mensagens do operador geradas pelo serviço. Recomendamos que seja usado um nome que comece com uma marca comercial da empresa ou sequência distintiva semelhante.

module_name
O nome do módulo a conter o código para esse componente. Especifique um nome de caminho completo.

tamanho
O tamanho em bytes da área de dados do componente passado ao componente em cada chamada. Especifique zero se nenhum dado de componente for requerido.

Essas duas sub-rotinas podem ocorrer em qualquer ordem e as chaves de sub-rotina sob elas também podem ocorrer em qualquer ordem. Para qualquer uma dessas sub-rotinas, todas as chaves de sub-rotina devem estar presentes. Se uma chave de sub-rotina for duplicada, a última será usada.

No momento da inicialização, o gerenciador de filas processa cada entrada do componente de serviço no arquivo de configuração na sua vez. Em seguida, ele carrega o módulo componente do especificado, chamando o ponto de entrada do componente (que deve ser o ponto de entrada para a inicialização do componente), passando a ele um identificador de configuração.

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 954
- "Definindo o serviço para o sistema operacional" na página 954
- "Configurando as sub-rotinas do serviço de autorização" na página 955
- "Interface de serviço de autorização no IBM i" na página 955

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 952 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 112. 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 954 para obter mais informações.

Escrevendo e compilando saídas de API

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: Não suportado no 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 deseje 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:

- 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.
- Para *flexibilidade*, é possível responder a mudanças rápidas em 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.
- Para *monitoramento* do uso de uma fila ou gerenciador de filas, é possível rastrear o fluxo de aplicativos e mensagens, registrar os erros nas chamadas de API, configurar 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 de API a serem executadas são identificadas em sub-rotinas nos sistemas IBM i, Windows, UNIX and Linux. Este tópico cobre 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 957 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 957 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 963 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 “Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows” na página 936

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.

Escrevendo saídas de API

É possível escrever saídas para cada chamada de API usando a linguagem de programação C.

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.

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 1092.

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 “Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows” na página 936.

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.

Nota: Para que as propriedades de mensagem de saída sejam processadas em:

- Uma função MQ_GET_EXIT após, você deve definir uma função MQ_GET_EXIT anterior para a saída.
- Uma função MQ_CALLBACK_EXIT anterior, deve-se definir uma função MQ_CB_EXIT anterior para a saída.

Um programa de amostra, amqsaem0.c, é fornecido para ilustrar o uso de identificadores de mensagens em saídas de API.

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 1092. Para plataformas diferentes dos sistemas Windows, é possível localizar 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. 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 para usuários:

1. A orientação sobre a programação de aplicativos de 64 bits é listada em Padrões de codificação em plataformas de 64 bits

Com a introdução de clientes Multicast, as saídas de API e as saídas de conversão de dados precisam ser capazes de executar no lado do cliente, pois algumas mensagens podem não passar pelo gerenciador de filas. As bibliotecas a seguir agora fazem parte dos pacotes do cliente, assim como dos pacotes do servidor:

Tabela 126. Bibliotecas que agora estão nos pacotes do cliente e do servidor

Sistema operacional	**Bibliotecas**
Windows	32 bits e 64 bits: mqm.dll e mqm.pdb
Linux & HP-UX	32 bits e 64 bits: libmqm.s e libmqm_r.so
AIX	32 bits e 64 bits: libmqm.a e libmqm_r.a
Solaris	32 bits e 64 bits: libmqm.so
IBM i IBM i	LIBMQM & LIBMQM_R

Compilando saídas de API no Unix e sistemas Linux

Exemplos de como compilar Saídas de API em sistemas UNIX 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.

Em 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
```

Na plataforma HP-UX Itanium

Aplicativos de 32 bits

Não encadeado

Compile o código-fonte da Saída de API:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o código-fonte da Saída de API

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe
rm amqsaxe.o
```

Encadeado

Compile o código-fonte da Saída de API:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o código-fonte da Saída de API

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe_r
rm amqsaxe.o
```

Aplicativos de 64 bits

Não encadeado

Compile o código-fonte da Saída de API:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o código-fonte da Saída de API

```
ld -b amqsaxe.o +ee MQStart -o /var/mqm/exits64/amqsaxe
rm amqsaxe.o
```

Encadeado

Compile o código-fonte da Saída de API:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o código-fonte da Saída de API

```
ld -b amqsaxe.o +ee MQStart -o /var/mqm/exits64/amqsaxe_r
rm amqsaxe.o
```

Em 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
```

Em Solaris

Compile o código-fonte de saída de API emitindo um dos seguintes comandos:

Aplicativos de 32 bits

Plataforma SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

Aplicativos de 64 bits

Plataforma SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

em sistemas Windows

Compile e vincule o programa de saída de 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 113 na página 962 ou Figura 114 na página 962 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 113. 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 114. Compile e vincule o `amqsaxe0.c` no Windows de 64 bits

Conceitos relacionados

"O programa de amostra de saída de API" na página 1092

A saída API de amostra gera um rastreio de MQI para um arquivo especificado pelo usuário com um prefixo definido na variável de ambiente MQAPI_TRACE_LOGFILE.

Em 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:

- Usando o IBM MQ Explorer (em Windows e Linux (plataformas x86 e x86-64))
- Usando o comando **amqmdain** (em Windows)
- Usando os arquivos mqs.ini e qm.ini diretamente (Em sistemas Windows, IBM i, UNIX and Linux).

  O arquivo mqs.ini contém informações relevantes para todos os gerenciadores de filas em um nó específico. É possível localizá-lo no diretório `/var/mqm` no UNIX and Linux IBM i, no diretório `/QIBM/UserData/mqm` no IBM i e no WorkPath especificado na chave `HKLM\SOFTWARE\IBM\WebSphere MQ` nos sistemas Windows.

  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:

  Nos sistemas UNIX and Linux:

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  IBM i Nos sistemas IBM i:

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  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.

  Você pode editar um arquivo de configuração:

  – Automaticamente, usando os comandos que alteram 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 totalmente o atributo).

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 do IBM MQ, em Saídas.

Quando algum gerenciador de filas for iniciado, os atributos nesta sub-rotina serão lidos e, em seguida, substituídos pelas saídas de API definidas em qm.ini.

ApiExitTemplate

Definido em mqs.ini e no IBM MQ Explorer na página de propriedades do IBM MQ, em 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, em Saídas.

Quando o gerenciador de filas é iniciado, as saídas de 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 988.

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 127 na página 966 mostra os tipos de saída do canal que ficam disponíveis para cada tipo de canal.

Tabela 127. 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 115 na página 967.

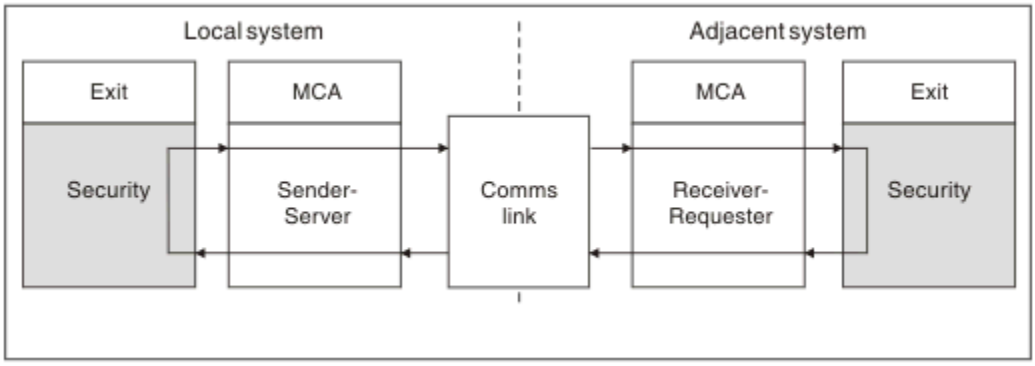


Figura 115. Loop da saída de segurança

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 116 na página 967.

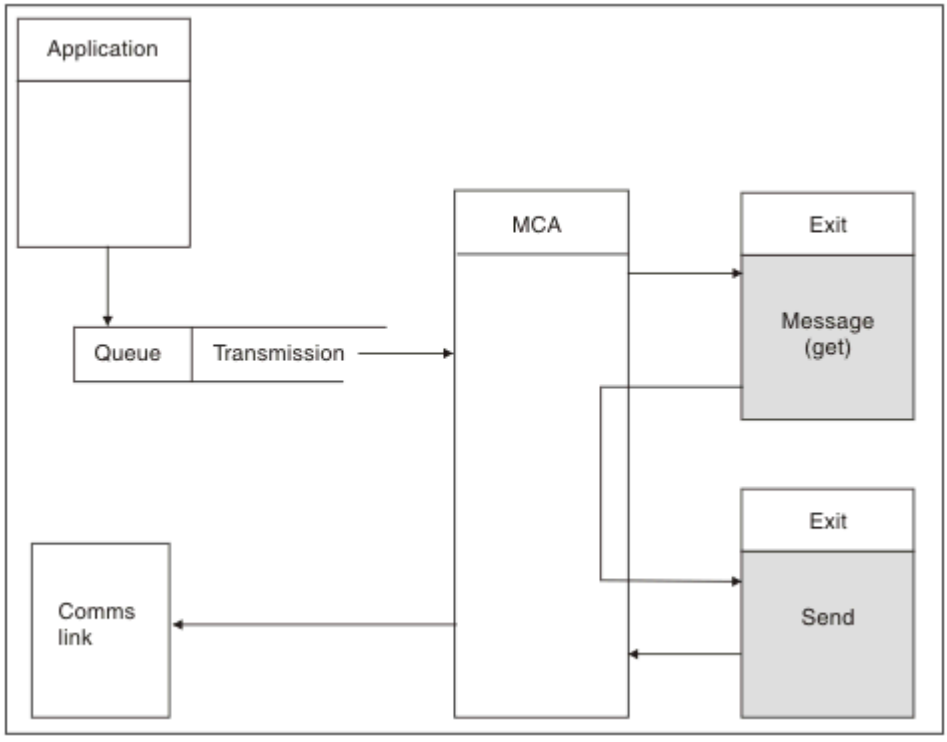


Figura 116. Exemplo de uma saída de envio na extremidade do emissor do canal de mensagens

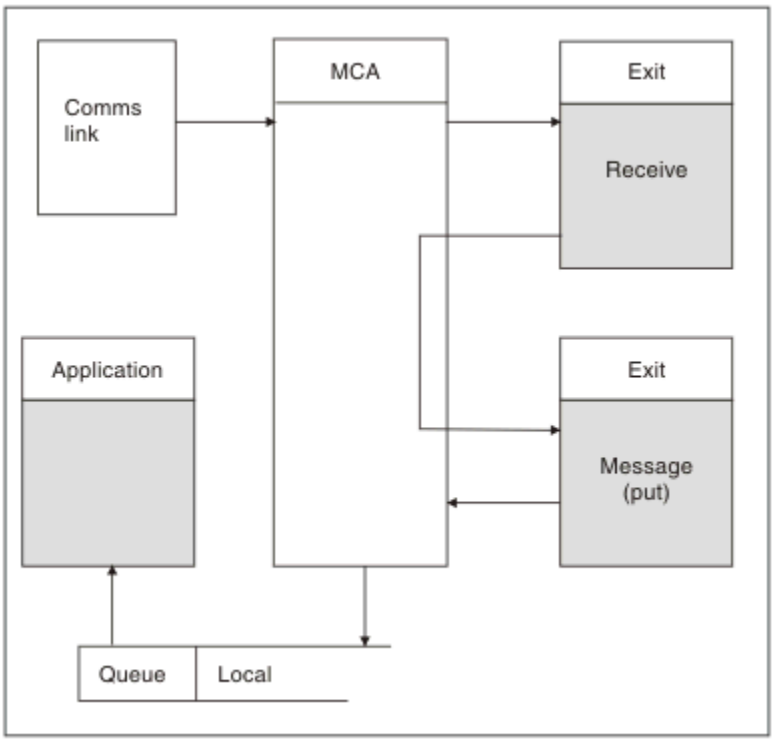


Figura 117. Exemplo de uma saída de recebimento no receptor final do canal de mensagens

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 117 na página 968. (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

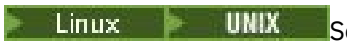
de filas corretamente. Por exemplo, uma saída de canal que cria uma nova conexão ao gerenciador de filas em cada chamada sem desconectar resulta em um acúmulo de manipulações de conexão e um aumento no número de encadeamentos de agente.

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.



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 16 KB

  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 informações adicionais sobre 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 965 para os detalhes da chamada.

z/OS *Gravando programas de saída de canal em z/OS*

É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para z/OS.

As saídas são iniciadas como se por um LINK do z/OS, 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

O módulos editados por link devem ser colocados no conjunto de dados especificado pela instrução CSQXLIB DD do procedimento do espaço de endereço do inicializador de canais; os nomes dos módulos de carregamento são especificados como os nomes de saída na definição de canal.

Ao gravar saídas de canal para o z/OS, as regras a seguir se aplicam:

- As saídas devem ser escritas no assembler ou C; se C for usado, ele deve se adequar com o ambiente de programação de sistemas C para saídas de sistema, descritas no Guia de Programação C/C++ z/OS.
- As saídas são carregadas a partir das bibliotecas não autorizadas definidas por uma instrução CSQXLIB DD. Desde que CSQXLIB tenha DISP=SHR, as saídas podem ser atualizadas enquanto o inicializador de canal estiver sendo executado. A nova versão é usada quando o canal é reiniciado.
- As saídas devem ser reentrantes e capazes de executar em qualquer lugar no armazenamento virtual.
- As saídas devem reconfigurar o ambiente, no retorno, para aquela entrada at.
- As saídas devem liberar qualquer armazenamento obtido ou assegurar que seja liberado por uma chamada de saída subsequente.

  Para armazenamento que deve persistir entre chamadas, use o serviço STORAGE do z/OS ou a função de biblioteca 4kmalc para System Programming C.

  Para obter mais informações sobre esta função, consulte 4kmalc () -- Alocar Armazenamento Alinhado à Página.
- Todas as chamadas MQI do IBM MQ, exceto MQCMIT ou CSQBCMT e MQBACK ou CSQBBAK podem ser usadas. Elas devem estar contidas após MQCONN (com um nome do gerenciador de filas em branco). Se essas chamadas forem usadas, a saída deve editada por link com o stub CSQXSTUB.

  A exceção a essa regra é que as saídas de canal de segurança podem emitir chamadas MQI de confirmação e restauração. Para emitir essas chamadas, codifique os verbos CSQXCMT e CSQXBAK no lugar de MQCMIT ou CSQBCMT e MQBACK ou CSQBBAK.
- Todas as saídas que usam o stub CSQXSTUB a partir do IBM WebSphere MQ 7.0 ou posterior devem ser editadas por link em uma biblioteca de carregamento CSQXLIB com formato PDS-E.
- Saídas não devem usar nenhum serviço do sistema que cause uma espera, porque usar serviços do sistema poderia afetar gravemente a manipulação de alguns ou todos os outros canais. Muitos canais são executados sob um único TCB normalmente. Se você fizer algo em uma saída que cause uma espera e não usar MQXWAIT, isso faz com que todos esses canais esperem. Fazer canais esperar não causa nenhum problema funcional, mas pode ter um efeito adverso no desempenho. A maioria das SVCs envolvem esperas, portanto, deve-se evitá-las, exceto as SVCs a seguir:
  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

  Em geral, portanto, evite SVCs, PCs e E/S. Em vez disso, use a chamada MQXWAIT.

- Saídas não emitem ESTAEs ou SPIEs, exceto em quaisquer subtarefas às quais se conectam, porque a manipulação de erros das mesmas pode interferir na manipulação de erros executada pelo IBM MQ. Isso significa que o IBM MQ pode não ser capaz de se recuperar de um erro ou que seu programa de saída pode não receber todas as informações de erro.
- A chamada MQXWAIT (consulte MQXWAIT) fornece um serviço espera que espera E/S e outros eventos; se esse serviço for usado, as saídas não devem usar a pilha de ligação.

  Para E/S e outros recursos que não fornecem recursos não de bloqueio ou um ECB para esperar, uma subtarefa separada deve ser ANEXADO e sua conclusão esperada por MQXWAIT; devido ao processamento que esta técnica envolve, esse recurso deve ser usado somente pela saída de segurança.

- A chamada MQI MQDISC não causa uma confirmação implícita no programa de saída. Uma confirmação do processo do canal é executada somente quando o protocolo do canal determina.

As amostras de saída a seguir são fornecidas com o IBM MQ for z/OS:

CSQ4BAX0
Essa amostra é escrita em assembler e ilustra o uso de MQXWAIT.

CSQ4BCX1 e CSQ4BCX2
Essas amostras são escritas em C e ilustram como acessar os parâmetros.

CSQ4BCX3 e CSQ4BAX3
Essas amostras são escritas em C e assembler, respectivamente.

A amostra CSQ4BCX3 (que está pré-compilada em SCSQAUTH LOADLIB) deve funcionar sem mudanças necessárias na própria saída. É possível criar um LOADLIB (por exemplo, chamado MY.TEST.LOADLIB) e copiar o membro SCSQAUTH(CSQ4BCX3) para ele.

Para configurar uma saída de segurança em uma conexão do cliente, execute o procedimento a seguir:

1. Estabeleça um segmento OMVS válido para o ID do usuário que o inicializador de canais usa.

   Isso permite que o inicializador de canais do IBM MQ for z/OS use TCP/IP com a interface de soquete UNIX System Services (USS) para facilitar o processamento de saída. Observe que não é necessário definir um segmento de OMVS para o ID do usuário de qualquer cliente de conexão.
2. Assegure que o código de saída em si seja executado somente em um ambiente controlado por programa.

   Isso significa que tudo que for carregado no espaço de endereço CHINIT deve ser carregado a partir de uma biblioteca controlada por programa (o que significa todas as bibliotecas no STEPLIB) e quaisquer bibliotecas denominadas em CSQXLIB e

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   Para configurar uma biblioteca como controlada por programa, use um comando semelhante a este exemplo:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   Em seguida, será possível ativar ou atualizar o ambiente controlado por programa emitindo o comando:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```
3. Inclua a saída LOADLIB em CSQXLIB DD (no procedimento iniciado CHINIT), emitindo o comando a seguir:

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

Isso ativa a saída para o canal denominado.

4. O gerenciador de segurança externa (ESM) lista quaisquer outras bibliotecas para serem controladas por programa, mas observe que nenhum das bibliotecas de ESM ou C precisa ser sob controle do programa.

Consulte IBM MQ for z/OS - canal de conexão do servidor para obter informações adicionais sobre como configurar uma saída de segurança usando a amostra CSQ4BCX3.

CSQ4BCX4
Essa amostra é escrita em C e demonstra o uso dos campos **RemoteProduct** e **RemoteVersion** em MQCXP.

Conceitos relacionados

"Gravando programas de saída de canal em IBM i" na página 972
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 UNIX, Linux, and Windows" na página 973
É possível usar as informações a seguir para ajudar a escrever programas de saída de canal para sistemas UNIX, Linux, and Windows.

Informações relacionadas

Canal de conexão do servidor IBM MQ for z/OS

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 UNIX, Linux, and Windows" na página 973

É possível usar as informações a seguir para ajudar a escrever programas de saída de canal para sistemas UNIX, Linux, and Windows.

“Gravando programas de saída de canal em z/OS” na página 970
É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para z/OS.

ULW *Escrevendo programas de saída de canal no UNIX, Linux, and Windows*
É possível usar as informações a seguir para ajudar a escrever programas de saída de canal para sistemas UNIX, Linux, and Windows.

Siga as instruções descritas em “Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows” na página 936. 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 118 na página 973 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 118. Código-fonte de amostra para uma saída do canal

Ao escrever saídas de canal para o Windows usando o Visual C++, deve-se escrever seu próprio arquivo DEF. Um exemplo de como fazer isso é mostrado em Figura 119 na página 973. Para obter informações adicionais sobre como escrever programas de saída do canal, consulte “Gravando programas de saída do canal” na página 968.

```
EXPORTS
ChannelExit
```

Figura 119. Arquivo DEF de amostra para o Windows

Conceitos relacionados

“Gravando programas de saída de canal em IBM i” na página 972
É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para IBM i.

“Gravando programas de saída de canal em z/OS” na página 970
É possível usar as informações a seguir para ajudá-lo a escrever e compilar programas de saída de canal para 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 de 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.

Para saídas de segurança gravadas antes do IBM WebSphere MQ 7.1 vale a pena observar que versões anteriores do IBM MQ consultavam o provedor de soquetes seguros subjacentes (por exemplo, GSKit) para determinar o Subject Distinguished Name (SSLPEER) e Issuer Distinguished Name (SSLCERTI) do certificado do parceiro remoto. No IBM WebSphere MQ 7.1, suporte foi incluído para um intervalo de novos atributos de segurança. Para acessar esses atributos, o IBM WebSphere MQ 7.1 obtém a codificação DER do certificado e a usa para determinar o Subject e Issuer DN. 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

No IBM WebSphere MQ 7.1, um atributo SERIALNUMBER também é incluído no DN do assunto e contém o número de série do certificado do parceiro remoto. Além disso, alguns atributos DN são retornados em uma sequência diferente das liberações anteriores. Consequentemente, a composição dos campos SSLPEER e SSLCERTI está alterada no IBM WebSphere MQ 7.1 de liberações anteriores e recomenda-se, portanto, que qualquer saída de segurança ou aplicativo dependente desses campos sejam examinados e atualizados.

Os filtros de nome de mesmo nível existentes do IBM MQ especificados por meio do campo SSLPEER de uma definição de canal não foram afetados e continuam a operar da mesma maneira que em liberações anteriores. Isso ocorre porque o algoritmo correspondente do nome de peer do IBM MQ foi atualizado para processar filtros SSLPEER existentes sem a necessidade de alterar as definições de canal. Essa mudança mais provavelmente afetará as saídas de segurança e os aplicativos que dependem dos valores de Subject DN e Issuer DN retornados pela interface de programação do PCF.

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 120 na página 975 a Figura 123 na página 977. 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 120 na página 975:

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.

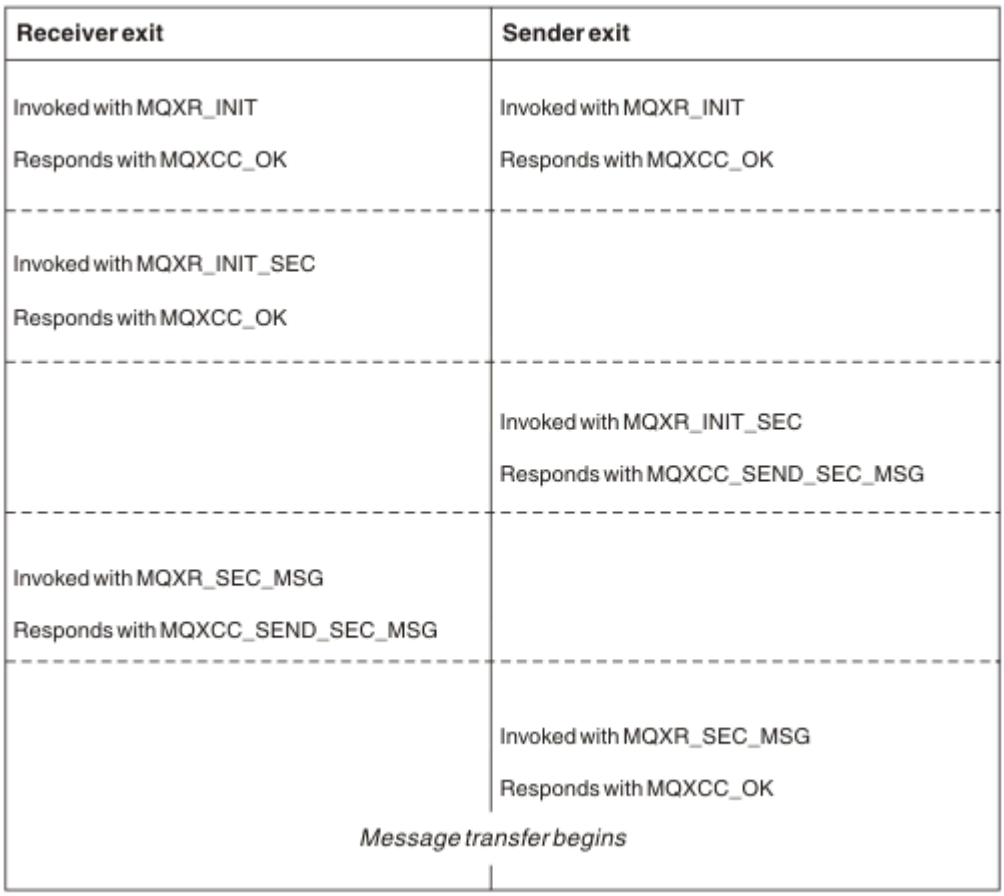


Figura 120. Troca iniciada pelo emissor com acordo

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 121. 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 122. 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 123. 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.

Informações relacionadas

Registros de Autenticação de Canal

Conceitos de TLS (Transport Layer Security)

Escrevendo uma saída de segurança

É possível escrever uma saída de segurança usando o código de estrutura básica da saída de segurança.

Figura 124 na página 978 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 124. 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).

Tipo de Canal
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).

Identificador de usuário do MCA
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 1155

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 normalmente tem 32 Kb de comprimento. 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 envio e recebimento de programas de saída para serem executados 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.

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. Ela identifica o objeto, recupera-o na maneira apropriada, anexa-o ao cabeçalho e passa-o ao MCA para transmissão ao 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” na página 792 e “Executando as amostras Reference Message” na página 1123 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 UNIX Se você fornecer uma saída de mensagem no UNIX 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.

Compilando programas de saída de canal em sistemas Windows,UNIX and Linux
Use os exemplos a seguir para ajudá-lo a compilar programas de saída do canal para sistemas Windows, UNIX and Linux.

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
```

Sistemas UNIX and Linux

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.

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
```

Comandos de amostra do compilador e do vinculador para saídas de canal no HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b exit.o +ee MQStart +ee ChannelExit -o
/var/mqm/exits64/exit -L/usr/lib/pa20_64 -lpthread
$ rm exit.o
```

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
```

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
```

Comandos de amostra do compilador e vinculador para saídas de canal no Solaris:

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

No cliente, uma saída de 32 bits ou 64 bits pode ser usada. Essa saída deve ser vinculada a mqic_r.

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 de sua plataforma IBM MQ; consulte MQCD ou Comandos de script (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.

Com a introdução de clientes Multicast, as saídas de API e as saídas de conversão de dados precisam ser capazes de executar no lado do cliente, pois algumas mensagens podem não passar pelo gerenciador de filas. As bibliotecas a seguir agora fazem parte dos pacotes do cliente, assim como dos pacotes do servidor:

Tabela 128. Bibliotecas que agora estão nos pacotes do cliente e do servidor

Sistema operacional	**Bibliotecas**
Windows	32 bits e 64 bits: mqm.dll e mqm.pdb
Linux & HP-UX	32 bits e 64 bits: libmqm.s e libmqm_r.so
AIX	32 bits e 64 bits: libmqm.a e libmqm_r.a
Solaris	32 bits e 64 bits: libmqm.so
IBM i IBM i	LIBMQM & LIBMQM_R

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 990.
   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 990. 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 990.

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.

Gravando um programa de saída de conversão de dados para o IBM MQ for z/OS

Informações sobre as etapas a serem consideradas ao gravar programas de saída de conversão de dados para o IBM MQ for z/OS.

Siga estas etapas:

1. Use a estrutura básica fornecida CSQ4BAX9 (para ambientes não CICS) ou CSQ4CAX9 (para CICS) como seu ponto de início.
2. Execute o utilitário CSQUCVX.
3. Siga as instruções no prólogo de CSQ4BAX9 ou de CSQ4CAX9 para incorporar as rotinas geradas pelo utilitário CSQUCVX na ordem em que as estruturas ocorrem na mensagem que você deseja converter.

4. O utilitário supõe que as estruturas de dados não estejam compactadas, que o alinhamento implícito dos dados seja respeitado e que as estruturas iniciem em um limite de palavra inteira, com bytes sendo ignorados conforme necessário (como entre o ID e VERSION no exemplo em Sintaxe válida). Se as estruturas estiverem compactadas, omita as macros CMQXCALA geradas. Portanto, considere declarar suas estruturas de forma que todos os campos sejam denominados e nenhum byte ignorado; no exemplo, em Sintaxe válida, inclua um campo "MQBYTE DUMMY;" entre ID e VERSION.
5. A saída fornecida retorna um erro se o buffer de entrada for mais curto que o formato da mensagem a ser convertido. Embora a saída converta o máximo de campos preenchidos possível, o erro faz uma mensagem não convertida ser retornada ao aplicativo. Se deseja permitir que buffers de entrada curta sejam convertidos até onde for possível, incluindo campos parciais, mude o valor de TRUNC= na macro CSQXCDFA para YES; nenhum erro será retornado, portanto, o aplicativo receberá uma mensagem convertida. O aplicativo deve manipular o truncamento.
6. Inclua qualquer outro código de processamento especial necessário.
7. Renomeie o programa para o nome de seu formato de dados.
8. Compile e edite o link de seu programa como um programa de aplicativo em lote (a menos que seja para uso com aplicativos CICS). As macros no código gerado pelo utilitário estão na biblioteca **thlqual**.SCSQMACS.

   Se a mensagem contiver dados de caractere, o código gerado chamará MQXCNVC. Se sua saída usar essa chamada, edite o link da mesma com o programa stub de saída CSQASTUB. O stub é independente da linguagem e do ambiente. Como alternativa, é possível carregar o stub dinamicamente usando o nome da chamada dinâmica CSQXCNVC. Consulte “Chamando dinamicamente o stub do IBM MQ” na página 1046 para obter mais informações.

   Coloque o módulo com link editado em sua biblioteca de carregamento de aplicativo e em um conjunto de dados que é referenciado pela instrução CSQXLIB DD do procedimento de tarefa iniciado pelo inicializador de canais.
9. Se a saída for para ser usadas por aplicativos CICS, compile e edite o link da mesma como um programa de aplicativo CICS, incluindo CSQASTUB, se necessário. Coloque-a em sua biblioteca de programas aplicativos CICS. Defina o programa para o CICS comumente, especificando EXECKEY(CICS) na definição.

Nota: Embora as bibliotecas de tempo de execução LE/370 sejam necessárias para executar o utilitário CSQUCVX (consulte a etapa “2” na página 991), elas não são necessárias para editar o link ou executar a saída de conversão de dados em si (consulte as etapas “8” na página 992 e “9” na página 992).

Consulte “Escrevendo aplicativos de ponte IMS” na página 67 para obter informações sobre conversão de dados na ponte IBM MQ - IMS.

Escrevendo uma saída de conversão de dados para o IBM MQ em sistemas UNIX and Linux

Informações sobre as etapas a serem consideradas ao escrever programas de saída de conversão de dados para o IBM MQ em sistemas UNIX and 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 992. 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 992.

   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 993. 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 992.
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 UNIX and Linux” na página 993.
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 o `crtmqcvx` usar estruturas empacotados, 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.

Compilação das saídas de conversão de dados em sistemas UNIX and Linux
Exemplos de como compilar uma saída de conversão de dados em sistemas UNIX 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

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
```

Plataforma HP-UX Itanium

Compile e vincule o código-fonte de saída emitindo um dos seguintes conjuntos de comandos:

Aplicativos de 32 bits

Não encadeado

Compile o código-fonte de saída:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o objeto de saída:

```
ld +b: -b MYFORMAT.o +ee MQStart -o \
    /var/mqm/exits/MYFORMAT -L/usr/lib/hpux32
rm MYFORMAT.o
```

Encadeado

Compile o código-fonte de saída:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o objeto de saída:

```
ld +b: -b MYFORMAT.o +ee MQStart -o \
    /var/mqm/exits/MYFORMAT_r -L/usr/lib/hpux32 \
    -lpthread
rm MYFORMAT.o
```

Aplicativos de 64 bits

Não encadeado

Compile o código-fonte de saída:

```
 c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o objeto de saída:

```
ld -b MYFORMAT.o +ee MQStart \
    -o  /var/mqm/exits64/MYFORMAT \
    -L/usr/lib/hpux64
rm MYFORMAT.o
```

Encadeado

Compile o código-fonte de saída:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Vincule o objeto de saída:

```
ld -b MYFORMAT.o +ee MQStart  \
    -o  /var/mqm/exits64/MYFORMAT_r  \
    -L/usr/lib/hpux64 -lpthread
rm MYFORMAT.o
```

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
```

Solaris

Compile o código-fonte de saída emitindo um dos seguintes comandos:

Aplicativos de 32 bits

Plataforma SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

Aplicativos de 64 bits

Plataforma SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

Plataforma x86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

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 996.

   amqsvfc0.c está em *MQ_INSTALLATION_PATH*`\Tools\C\Samples`, em que *MQ_INSTALLATION_PATH* é o diretório no qual o 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 997.
   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 997. 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 997.
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.

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 arquivos	**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, este parâmetro é a sequência de filtros fornecida para a chamada de API do 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);
```

Parâmetro

```
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 “Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows” na página 936, 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 MQRC_PUBLISH_EXIT_ERROR do IBM MQ.

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 “Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows” na página 936.

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 UNIX 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 "Programação de saída de carga de trabalho do cluster para IBM MQ for z/OS" na página 1004.

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 "Escrevendo saídas e serviços instaláveis no UNIX, Linux e Windows" na página 936 e use o programa de amostra, "Saída de carga de trabalho do cluster de amostra" na página 1003, 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 968.

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)
```

Informações 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 chamado `amqswlm0.c`. Ele pode ser localizado em:

Tabela 129. Local do programa de saída de carga de trabalho do cluster de amostra para multiplataformas

Plataforma	**Caminho de arquivo..**
AIX, HP-UX, Sun Solaris	*MQ_INSTALLATION_PATH*/samp
Windows	*MQ_INSTALLATION_PATH* \Tools\c\Samples
IBM i 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 UNIX No UNIX 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.

Programação de saída de carga de trabalho do cluster para IBM MQ for z/OS

Saídas de carga de trabalho do cluster são chamadas como se por um comando z/OS **LINK**. As saídas estão sujeitas a diversas regras rígidas de programação. Evite usar a maioria dos comandos SVC que envolvem esperas ou usar um STAE ou ESTAE em uma saída de carga de trabalho.

Saídas de carga de trabalho são chamadas como se por um 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

- Chave de armazenamento 8
- Máscara da chave de programa 8
- Chave TCB 8

Coloque os módulos com link editado no conjunto de dados especificado pela instrução `CSQXLIB DD` do procedimento do espaço de endereço do gerenciador de filas. Os nomes dos módulos de carregamento são especificados como os nomes de saída de carga de trabalho na definição do gerenciador de filas.

Ao escrever saídas de carga de trabalho para o IBM MQ for z/OS, as regras a seguir se aplicam:

- Deve-se escrever saídas em assembler ou C. Se você usa C, ele deverá estar em conformidade com o ambiente de programação de sistemas C para saídas de sistema, descritas no *z/OS Guia De Programação C/C++*, SC09-4765.
- Se estiver usando a chamada `MQXCLWLN`, linkedite com `CSQMFCLW`, fornecido em *`thlqual`*`.SCSQLOAD`.
- As saídas são carregadas a partir das bibliotecas não autorizadas definidas por uma instrução `CSQXLIB DD`. Supondo que `CSQXLIB` tenha `DISP=SHR`, as saídas podem ser atualizadas enquanto o gerenciador de filas estiver em execução, com a nova versão usada no próximo encadeamento `MQCONN` que o gerenciador de filas iniciar.
- As saídas devem ser reentrantes e capazes de executar em qualquer lugar no armazenamento virtual.
- As saídas devem reconfigurar o ambiente ao retornarem a essa entrada.
- As saídas devem liberar qualquer armazenamento obtido ou assegurar que o armazenamento seja liberado por uma chamada de saída subsequente.
- Nenhuma chamada de MQI é permitida.
- As saídas não devem usar nenhum serviço do sistema que possa causar uma espera, porque uma espera degrada gravemente o desempenho do gerenciador de filas. Em geral, portanto, evite um SVC, PC ou E/S.
- As saídas não devem emitir um ESTAE ou SPIE, exceto de quaisquer subtarefas que conectarem.

Nota: Não há restrições absolutas no que é possível fazer em uma saída. No entanto, a maioria dos SVCs envolve esperas, portanto, evite-os, exceto para os comandos a seguir:

- **`GETMAIN`** / **`FREEMAIN`**
- **`LOAD`** / **`DELETE`**

Não use ESTAEs e ESPIEs porque suas manipulações de erros podem interferir na manipulação de erros executada pelo IBM MQ. O IBM MQ pode não ser capaz de recuperar-se de um erro ou seu programa de saída pode não receber todas as informações do erro.

O parâmetro do sistema `EXITLIM` limita o tempo de execução de uma saída. O valor padrão para `EXITLIM` é 30 segundos. Se você vir o código de retorno `MQRC_CLUSTER_EXIT_ERROR`, 2266 `X'8DA'`, sua saída pode estar em loop. Se achar que a saída precisa de mais de 30 segundos para concluir, aumente o valor de `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 711 para obter uma descrição completa.

Ao executar aplicativos cliente ou servidor encadeados, configure a variável de ambiente AIXTHREAD_SCOPE=S

Preparando programas C no AIX

Este tópico contém informações sobre a vinculação de bibliotecas necessárias para preparar programas C no AIX.

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.

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 você desejar usar os programas em uma máquina que tenha apenas o IBM MQ MQI client for AIX instalado, recompile os programas para vinculá-los à biblioteca do cliente (`-lmqic`).

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 bilioteca	**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 bilioteca	**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.

Se você desejar usar os programas em uma máquina que tenha apenas o IBM MQ MQI client for AIX instalado, recompile os programas para vinculá-los à biblioteca do cliente (`-lmqic`).

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 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 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 às bibliotecas `libmqmxa.a` (ou `libmqmxa64.a` se seu gerenciador de transações tratar o tipo 'longo' como 64 bits) e `libmqz.a` em um aplicativo não encadeado e às bibliotecas `libmqmxa_r.a` (ou `libmqmxa64_r.a`) e `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 no IBM MQ em sistemas UNIX and Linux pode ser conectado de cada vez.
5. Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

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 bilioteca	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 um programa de amostra

Consulte a documentação do Micro Focus COBOL para obter uma descrição das variáveis de ambiente que precisa configurar.

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 130 na página 1010.

Tabela 130. 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 *amqzsc* do IBM MQ, fornecida 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.

Suporte do 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 `libmqmcbrt.o` do IBM MQ 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 ativa a biblioteca COBOL de tempo de execução do CICS para chamar IBM MQ nos sistemas UNIX and 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" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export 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.SAMPLECICS.WORKQUEUE no gerenciador de filas padrão e as coloca 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.

Construindo seu aplicativo processual no HP-UX

Estas informações descrevem as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao construir aplicativos IBM MQ for HP-UX para execução no HP-UX.

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 HP-UX 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 HP-UX para a linguagem que você está usando. Familiarize-se com o conteúdo desses arquivos. Consulte “Arquivos de definição de dados doIBM MQ” na página 711 para obter uma descrição completa.

Em todo este tópico, usamos o caractere de barra invertida (\) para dividir os comandos longos em mais de uma linha. Não insira esse caractere; insira cada comando como uma única linha.

Preparando programas C no HP-UX

Este tópico contém informações a serem consideradas ao preparar programas C no HP-UX; com exemplos para a plataforma IA64 (IPF).

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Trabalhe em seu ambiente normal. Programas C pré-compilados são fornecidos no diretório *MQ_INSTALLATION_PATH* /samp/bin.

Para obter informações adicionais sobre a programação de aplicativos de 64 bits, consulte Padrões de codificação em plataformas de 64 bits.

Para usar TLS, IBM MQ MQI clients no HP-UX devem ser construídos usando encadeamentos POSIX.

Alguns exemplos a serem considerados são:

- “Plataforma IA64 (IPF)” na página 1012
- “Vinculando bibliotecas” na página 1014

Plataforma IA64 (IPF)

Exemplos de construção de amqsput0, cliexit e srvexit na plataforma IA64(IPF).

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo cliente em um ambiente de 32 bits não encadeado:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo cliente em um ambiente de 32 bits encadeado:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic_r -lpthread
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo cliente em um ambiente de 64 bits não encadeado:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo cliente em um ambiente de 64 bits encadeado:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo do servidor em um ambiente de 32 bits não encadeado:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo do servidor em um ambiente de 32 bits encadeado:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm_r -lpthread
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo do servidor em um ambiente de 64 bits não encadeado:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

O exemplo a seguir constrói o programa de amostra amqsput0 como um aplicativo do servidor em um ambiente de 64 bits encadeado:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
  -L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

O exemplo a seguir constrói uma saída de cliente cliexit em um ambiente de 32 bits não encadeado:

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic
```

O exemplo a seguir constrói uma saída de cliente cliexit em um ambiente de 32 bits encadeado:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

O exemplo a seguir constrói uma saída de cliente cliexit em um ambiente de 64 bits não encadeado:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

O exemplo a seguir constrói uma saída de cliente cliexit em um ambiente de 64 bits encadeado:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -I MQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

O exemplo a seguir constrói uma saída de servidor srvexit em um ambiente de 32 bits não encadeado:

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm
```

O exemplo a seguir constrói uma saída de servidor srvexit em um ambiente de 32 bits encadeado:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -L MQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

O exemplo a seguir constrói uma saída de servidor srvexit em um ambiente de 64 bits não encadeado:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH
MQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

O exemplo a seguir constrói uma saída de servidor srvexit em um ambiente de 64 bits encadeado:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -I MQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-L MQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

Vinculando bibliotecas

É necessário vincular seus programas a uma das bibliotecas fornecidas pelo IBM MQ.

A tabela a seguir mostra qual biblioteca usar em diferentes ambientes:

Plataforma de hardware	Ambiente encadeado ou não encadeado	Tipo de programa/saída	Arquivo de bilioteca
IA64 (IPF)	Encadeamento	Servidor e cliente para C	libmqm_r.so
IA64 (IPF)	Encadeamento	Cliente para C	libmqic_r.so
IA64 (IPF)	Não encadeado	Servidor e cliente para C	libmqm.so
IA64 (IPF)	Não encadeado	Cliente para C	libmqic.so

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 para obter mais informações), será necessário vincular-se à biblioteca `libmqmzf.sl`.
3. Se estiver produzindo um aplicativo para coordenação externa por um gerenciador de transações compatível com XA, como o IBM TXSeries Encina ou o BEA Tuxedo, você precisa vincular às bibliotecas `libmqmxa.sl` (ou `libmqmxa64.sl` se seu gerenciador de transações tratar do tipo 'long' como 64 bits) e `libmqz.sl` em um aplicativo não encadeado e às bibliotecas `libmqmxa_r.sl` (ou `libmqmxa64_r.sl`) e `libmqz_r.sl` em um aplicativo encadeado.
4. Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

Preparando programas COBOL em HP-UX

Saiba sobre como preparar programas COBOL no HP-UX, usando o Micro Focus Server Express com o IBM MQ na Plataforma IA64 (IPF), e executar programas no ambiente do IBM MQ MQI client.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Observações de uso:

1. 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
   ```

2. Copy books 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 aplicativos de 32 bits e:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

para aplicativos de 64 bits.

Compile o programa usando o compilador Micro Focus. Os arquivos de cópia que declaram as estruturas estão em *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB= MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="COBCPY_VALUE"
```

Compilando programas de 32 bits:

```
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Compilando programas de 64 bits:

```
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

em que `amqsput` é um programa de amostra

Certifique-se de ter especificado os tamanhos de pilha de tempo de execução adequados; 16 KB é o mínimo recomendado.

É necessário vincular seus programas à biblioteca apropriada fornecida pelo IBM MQ. A tabela a seguir mostra qual biblioteca usar em diferentes ambientes

Plataforma de hardware	Tipo de programa/saída	Arquivo de bilioteca
IA64 (IPF)	Servidor para COBOL	libmqmcb.so
IA64 (IPF)	Cliente para COBOL	libmqicb.so
IA64 (IPF)	Aplicativos encadeados	libmqmcb_r.so

Usando o Micro Focus Server Express com IBM MQ na plataforma IA64 (IPF)

Consulte “Modelos de espaço de endereço suportados pelo IBM MQ for HP-UX no IA64 (IPF)” na página 1017 para obter detalhes sobre a utilização do Micro Focus Server Express junto com o IBM MQ na plataforma HP/IPF.

Programas para execução no ambiente do IBM MQ MQI client

Se estiver usando LU 6.2 para conectar o cliente de MQI a um servidor, vincule seu aplicativo a libsna.a, parte do produto SNAplusAPI. Use as opções `-lV3` e `-lstr` em seu comando de compilação e vinculação.

- A opção `-lV3` fornece ao programa acesso à biblioteca de sinalização AT & T (o SNAplusAPI usa sinais AT & T)
- A opção `-lstr` vincula seu programa ao componente de fluxos

Preparando programas CICS no HP-UX

Aprenda a construir programas de transação do CICS no HP-UX.

Para construir a amostra de transação do CICS, amqscic0.ccs, execute o comando a seguir:

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

Um módulo do comutador XA é fornecido para permitir que você vincule o CICS ao IBM MQ:

Tabela 131. Código essencial para aplicativos CICS (HP-UX)		
Descrição	**C (origem)**	**C (exec)**
Rotina de inicialização de XA	amqzscix.c	amqzsc

É possível localizar mais informações sobre o suporte a transações do CICS no Administrando.

Suporte do TXSeries CICS

IBM MQ no HP-UX suporta TXSeries CICS usando a interface XA. Assegure-se de que os aplicativos CICS estejam vinculados à versão encadeada das bibliotecas MQ.

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`.

Transação de amostra do CICS C

A origem de C de amostra para uma transação CICS IBM MQ é fornecida por AMQSCIC0.CCS. A transação lê mensagens a partir da fila de transmissão SYSTEM.SAMPLE.CICS.WORKQUEUE no gerenciador de filas padrão e as coloca na fila local com o nome da fila 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.

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`. Após mover essas 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 ativa a biblioteca COBOL de tempo de execução do CICS para chamar IBM MQ nos sistemas UNIX and 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 HP-UX
- 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
```

Modelos de espaço de endereço suportados pelo IBM MQ for HP-UX no IA64 (IPF)

O HP-UX fornece modelos de espaço de endereço que podem ser explorados pelos aplicativos IBM MQ.

HP-UX suporta dois modelos de espaço de endereço:

- MGAS - Mostly global address space (esse é o padrão e é usado pelo IBM MQ)
- MPAS - Mostly private address space

Os aplicativos que se conectam ao IBM MQ podem usar os modelos de espaço de endereço MPAS ou MGAS. Os aplicativos construídos usando o modelo MPAS que se conectam ao IBM MQ usando a memória compartilhada pode incorrer em um custo de desempenho menor devido à ineficiência no mapeamento de páginas de memória compartilhada usada pelo IBM MQ no espaço de endereço virtual do programa MPAS.

Os aplicativos em COBOL construídos usando o Micro Focus Server Express usam o modelo MPAS por padrão.

É possível usar o programa **chatr** para verificar e mudar o modelo de endereçamento usado por um programa.

Se você encontrar problemas de conexão com o IBM MQ a partir de programas MPAS de 32 bits, considere o uso do modelo de endereçamento MGAS ou a construção de seu aplicativo como um aplicativo MPAS de 64 bits em vez de um aplicativo MPAS de 32 bits.

Mais detalhes sobre os modelos de espaço de endereço MGAS e MPAS podem ser encontrados na documentação do HP-UX.

Linux Construindo seu aplicativo processual no Linux

Essas informações descrevem as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao construir aplicativos do IBM MQ for Linux para execução

C e C++ são suportados. Para obter informações sobre a preparação de seus programas C++, consulte Usando C++.

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 bilioteca	**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 bilioteca	**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 para obter mais informações), será necessário vincular-se à biblioteca `libmqmzf.so`.
3. Se 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 ao `libmqmxa.so` (ou `libmqmxa64.so` se seu gerenciador de transações tratar o tipo 'long' como 64 bits) e as bibliotecas `libmqz.so` em um aplicativo não encadeado e para as bibliotecas `libmqmxa_r.so` (ou `libmqmxa64_r.so`) e `libmqz_r.so` em um aplicativo encadeado.
4. Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

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
```

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
```

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 bilioteca	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 um programa de amostra

Consulte a documentação Micro Focus COBOL para uma descrição das variáveis de ambiente que você precisa.

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 711 para obter uma descrição completa.

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 usar o comando do IBM i,CRTCMOD. Certifique-se de que a biblioteca que contém os arquivos de inclusão (QMQM) esteja na lista de bibliotecas quando você compilar.

Deve-se, então, vincular a saída do compilador com o programa de serviço usando o comando CRTPGM.

Um exemplo do comando para um ambiente não encadeado é:

Tabela 132. Exemplo de CRTPGM no ambiente não encadeado

Comando:	**Tipo de programa/saída**
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM)`	Servidor ou cliente para C

em que *pgmname* é o nome do seu programa.

Um exemplo do comando para um ambiente encadeado é:

Tabela 133. Exemplo de CRTPGM no ambiente encadeado

Comando:	**Tipo de programa/saída**
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM_R)`	Servidor ou cliente para C

em que *pgmname* é o nome do seu programa.

As tabelas a seguir listam as bibliotecas que são necessárias ao preparar programas C no IBM i em um ambiente não encadeado e em um ambiente encadeado.

Tabela 134. Ambiente não encadeado

Arquivo de bilioteca	**Tipo de programa/saída**
LIBMQM	Servidor para C

Tabela 134. Ambiente não encadeado (continuação)	
Arquivo de bilioteca	**Tipo de programa/saída**
LIBMQIC & LIBMQM	Cliente para C

Tabela 135. Ambiente encadeado	
Arquivo de bilioteca	**Tipo de programa/saída**
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.

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.

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 CBL do CRTCICSC ou CRTCICS 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 1075 e consulte a Referência de programação de aplicativos do IBM i (ILE/RPG).

Contraprestações sobre a programação de SQL

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)
```

Solaris Construindo seu aplicativo processual no Solaris

Estas informações descrevem as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao construir aplicativos IBM MQ for Solaris para execução no Solaris.

As linguagens de programação COBOL, C e C++ são suportadas. Para obter informações sobre a preparação de seus programas C++, consulte Usando C++.

Além de codificar as chamadas MQI em seu código fonte, deve-se incluir os arquivos de inclusão apropriados. Familiarize-se com o conteúdo desses arquivos. Consulte “Arquivos de definição de dados doIBM MQ” na página 711 para obter uma descrição completa.

Em todo este tópico, o caractere de barra invertida (\) é usado para dividir os comandos longos em mais de uma linha. Não insira este caractere, insira cada comando como uma única linha.

Preparando programas C no Solaris

Programas C pré-compilados são fornecidos no diretório *MQ_INSTALLATION_PATH* `/samp/bin`.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Para obter informações adicionais sobre a programação de aplicativos de 64 bits, consulte Padrões de codificação em plataformas de 64 bits.

Se desejar usar os programas em uma máquina que possui apenas o IBM MQ MQI client for Solaris instalado, compile os programas para vinculá-los à biblioteca do cliente (`-lmqic`).

Se você usar o compilador não suportado `/usr/ucb/cc`, seu aplicativo poderá compilar e vincular com êxito. No entanto, quando você executa o aplicativo, ele falha quando tenta se conectar ao gerenciador de filas.

Nota: Clientes SSL e TLS do Solaris x86 de 32 bits configurados para operação compatível com FIPS 140-2 falham ao executar em sistemas Intel. Essa falha ocorre porque o arquivo de biblioteca GSKit-Crypto Solaris x86 de 32 bits compatível com o FIPS 140-2 não é carregado no chipset Intel. Nos sistemas afetados, o erro AMQ9655 é relatado no log de erros do cliente. Para resolver esse problema, desative a conformidade com o FIPS 140-2 ou recompile o aplicativo cliente de 64 bits, porque o código de 64 bits não é afetado.

Vinculando bibliotecas

Deve-se vincular com as bibliotecas do IBM MQ que são apropriadas ao seu tipo de aplicativo:

Arquivos de biblioteca	Tipo de programa/saída
libmqm.so	Servidor para C
libmqic.so & libmqm.so	Cliente para C

Nota:

1. Se você estiver gravando um serviço instalável (para obter informações adicionais, consulte o Administrando), vincule-se à biblioteca `libmqmzf.so` .
2. Se estiver produzindo um aplicativo para coordenação externa por um gerenciador de transações compatível com XA, como IBM TXSeries Encina ou BEA Tuxedo, você deverá se vincular às bibliotecas `libmqmxa.so` (ou `libmqmxa64.so` se seu gerenciador de transações tratar o tipo ‘longo' como 64 bits) e `libmqz.so`.

3. Deve-se vincular as bibliotecas do IBM MQ antes de quaisquer outras bibliotecas de produto.

Construindo aplicativos em x86-64
Este tópico contém exemplos dos comandos usados para construir programas em diversos ambientes na plataforma x86-64.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo do cliente C, 32 bits

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Aplicativo do cliente C, 64 bits

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

Aplicativo do servidor C, 32 bits

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Aplicativo do servidor C, 64 bits

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

Aplicativo cliente C++, 32 bits

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

Aplicativo do cliente C++, 64 bits

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicativo do servidor C++, 32 bits

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Aplicativo do servidor C++, 64 bits

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Saída do cliente C, 32 bits

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
```

```
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

Saída do cliente C, 64 bits

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

Saída do servidor C, 32 bits

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

Saída do servidor C, 64 bits

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

Construindo aplicativos no SPARC
Este tópico contém exemplos dos comandos usados para construir programas em vários ambientes na plataforma SPARC.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Aplicativo do cliente C, 32 bits

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Aplicativo do cliente C, 64 bits

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Aplicativo do servidor C, 32 bits

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Aplicativo do servidor C, 64 bits

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Aplicativo cliente C++, 32 bits

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic
-lsocket -lnsl -ldl
```

Aplicativo do cliente C++, 64 bits

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplicativo do servidor C++, 32 bits

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Aplicativo do servidor C++, 64 bits

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Saída do cliente C, 32 bits

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

Saída do cliente C, 64 bits

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

Saída do servidor C, 32 bits

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

Saída do servidor C, 64 bits

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

Preparando programas COBOL em Solaris

Saiba sobre como preparar programas COBOL no Solaris.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

1. 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
   ```

2. Copy books 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 aplicativos de 32 bits e:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   para aplicativos de 64 bits.

Compile os programas usando o compilador Micro Focus. Os arquivos de cópia que declaram as estruturas estão em *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB= MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="COBCPY_VALUE"
```

Compilando programas de 32 bits:

- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb`

  Servidor para COBOL
- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb`

  Cliente para COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r`

  Servidor encadeado para COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r`

  Cliente encadeado para COBOL

Compilando programas de 64 bits:

- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb`

  Servidor para COBOL
- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb`

  Cliente para COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r`

  Servidor encadeado para COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r`

  Cliente encadeado para COBOL

em que *amqs0put0.cbl* é um programa de amostra.

Deve-se vincular seu programa com um dos seguintes:

- libmqmcb.so

  Servidor para COBOL
- libmqicb.so

  Cliente para COBOL

Preparando programas CICS no Solaris

Saiba sobre como preparar programas CICS no Solaris.

Um módulo do comutador XA é fornecido para permitir que você vincule o CICS ao IBM MQ:

Tabela 136. Código essencial para aplicativos CICS (Solaris)		
Descrição	**C (origem)**	**C (exec)**
Rotina de inicialização de XA	amqzscix.c	amqzsc - TXSeries para Solaris

Sempre vincule suas transações à biblioteca do IBM MQ thread-safe libmqm.so.

É possível localizar mais informações sobre o suporte a transações do CICS no Administrando.

Suporte do TXSeries CICS
O IBM MQ for Solaris suporta o TXSeries CICS usando a interface XA.

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 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
   ```

   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, mova essas funções dos aplicativos COBOL para sua própria biblioteca, por exemplo, `myMQ.so`. Após mover essas 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 ativa a biblioteca COBOL de tempo de execução do CICS para chamar IBM MQ nos sistemas UNIX and 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 TXSeries para Solaris
   - 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

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" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export 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 CICS IBM MQ é fornecida por AMQSCIC0.CCS. A transação lê mensagens da fila de transmissão SYSTEM.SAMPLECICS.WORKQUEUE no gerenciador de filas padrão e as coloca 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.

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 ActiveX, C, C++, COBOL e Visual Basic são suportadas. Para obter informações sobre como preparar seus programas ActiveX, consulte Usando o Component Object Model Interface (WebSphere MQ Classes de Automação para ActiveX). 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 711 para obter uma descrição completa.

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 137. 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 desejar criar a versão de 64 bits, deve-se assegurar que seu ambiente esteja configurado para usar os arquivos de 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.

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 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, poderá mover do IBM MQ for Windows para IBM MQ em sistemas UNIX and Linux, em que o uso de letras minúsculas 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.

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 bilioteca	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.

V 9.0.0 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 1070 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 1070
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 920
É possível vincular os aplicativos Microsoft Visual Basic com o código IBM MQ MQI client no 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 1155.

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 138. 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 1039). 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 Construindo seu aplicativo processual no z/OS

As publicações CICS, IMS e z/OS descrevem como construir aplicativos que são executados nesses ambientes.

Esta coleção de tópicos descreve as tarefas adicionais e as mudanças nas tarefas padrão que devem ser executadas ao criar aplicativos IBM MQ for z/OS para esses ambientes. As linguagens de programação COBOL, C, C++, Assembler e PL/I são suportadas. (Para obter informações sobre a construção de aplicativos C++, consulte Usando C++.)

As tarefas que devem ser executadas para criar um aplicativo IBM MQ for z/OS executável dependem da linguagem de programação em que o programa é gravado e do ambiente no qual o aplicativo será executado.

Além de codificar as chamadas MQI em seu programa, inclua as instruções de linguagem apropriadas para incluir o arquivo de definição de dados do IBM MQ for z/OS para a linguagem que você está usando. Familiarize-se com o conteúdo desses arquivos. Consulte “Arquivos de definição de dados doIBM MQ” na página 711 para obter uma descrição completa.

Nota

O nome **thlqual** é o qualificador de alto nível da biblioteca de instalação no z/OS.

Preparando seu programa para execução

Após ter escrito o programa para o seu aplicativo IBM MQ para criar um aplicativo executável, será necessário compilá-lo ou montá-lo, em seguida, editar o link do código do objeto resultante com o programa stub que o IBM MQ for z/OS fornece para cada ambiente que suporta.

Como você prepara seu programa depende tanto no ambiente (de lote, CICS, IMS (BMP ou MPP), serviços do sistema Linux ou UNIX) no qual o aplicativo é executado e da estrutura dos conjuntos de dados em sua instalação do z/OS.

O “Chamando dinamicamente o stub do IBM MQ” na página 1046 descreve um método alternativo de fazer chamadas MQI em seus programas de forma que não seja necessário editar o link de um stub do IBM MQ. Esse método não está disponível para todas as linguagens e ambientes.

Não edite o link de um nível mais alto do programa stub do que aquele da versão do IBM MQ for z/OS em que seu programa está em execução. Por exemplo, um programa em execução em MQSeries para OS/390, 5.2 não deve ser editado por link com um programa stub fornecido com IBM MQ for z/OS 7.

Criando aplicativos C de 64 bits

No z/OS, aplicativos C de 64 bits são construídos usando o compilador LP64 e opções do componente de ligação. O arquivo de cabeçalho *cmqc.h* do IBM MQ for z/OS reconhece quando essa opção é fornecida ao compilador e gera tipos de dados e estruturas do IBM MQ apropriados para a operação de 64 bits.

O código C construído com essa opção deve ser construído para usar as bibliotecas de vínculo dinâmico (DLLs) apropriadas para a semântica de coordenação necessária. Ligando o código compilado com o deck de lado apropriado definido em Nome do deck de lado necessário para cada semântica de coordenação mostra a DLL específica necessária.

Tabela 139. Nome do deck de lado necessário para cada semântica de coordenação	
Coordenação	**Nome do deck de lado**
MQI de confirmação de fase única	CSQBMQ2X
Duas confirmações de fase com coordenação RRS, usando verbos RRS	CSQBRR2X
Duas confirmações de fase com coordenação RRS, usando verbos MQI	CSQBRI2X

Use o procedimento EDCQCB JCL, fornecido com o *z/OS XL C/C++*, para construir um programa IBM MQ de confirmação de fase única como uma tarefa em lote, conforme a seguir:

```
//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
```

Para construir um programa coordenado pelo RRS no z/OS Unix System Services, compile e vincule, conforme a seguir:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Criando aplicativos em lote do z/OS*

Aprenda como construir aplicativos em lote do z/OS e as etapas a considerar ao fazer isso.

Para construir um aplicativo para IBM MQ for z/OS que é executado em lote no z/OS, crie uma linguagem de controle de tarefas (JCL) que execute estas tarefas:

1. Compile (ou monte) o programa para produzir código do objeto. O JCL da sua compilação deve incluir instruções SYSLIB que tornam os arquivos de definição dos dados de produto disponíveis para o compilador. As definições de dados são fornecidas nas bibliotecas do IBM MQ for z/OS a seguir:
   - Para COBOL, **thlqual**.SCSQCOBC
   - Para linguagem assembler, **thlqual**.SCSQMACS
   - Para C, **thlqual**.SCSQC370
   - Para PL/I, **thlqual**.SCSQPLIC
2. Para um aplicativo C, pré-vincule o código do objeto criado na etapa “1” na página 1041.
3. Para os aplicativos PL/I, use a opção do compilador EXTRN(SHORT).
4. Linkedite o código do objeto criado na etapa “1” na página 1041 (ou na etapa “2” na página 1041 para um aplicativo C) para produzir um módulo de carregamento. Ao linkeditar o código, deve-se incluir um dos programas stub em lote do IBM MQ for z/OS (CSQBSTUB ou um dos programas stub RRS: CSQBRRSI ou CSQBRSTB).

   CSQBSTUB
   single-phase commit fornecido pelo IBM MQ for z/OS

   CSQBRRSI
   two-phase commit fornecido pelo RRS usando o MQI

   CSQBRSTB
   two-phase commit fornecido diretamente pelo RRS

Notas:

a. Ao usar o CSQBRSTB, deve-se também linkeditar seu aplicativo com ATRSCSS a partir do SYS1.CSSLIB. Figura 125 na página 1042 e Figura 126 na página 1042 mostram fragmentos de código JCL para fazer isso. Os stubs são independentes do idioma e são fornecidos na biblioteca **thlqual**.SCSQLOAD.

b. Ao executar o aplicativo no ambiente de linguagem, é necessário assegurar a linkedição com a DLL do ambiente de linguagem, como alternativa, conforme descrito em "Criando aplicativos em lote do z/OS usando o Language Environment" na página 1042.

5. Armazene o módulo de carregamento em uma biblioteca de carregamento do aplicativo.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Figura 125. Os fragmentos de código JCL para linkeditar o módulo de objeto no ambiente de lote usando 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)
⋮
/*
```

Figura 126. Os fragmentos de JCL para linkeditar o módulo de objeto no ambiente de lote usando two-phase commit

Para executar um programa em lote ou RRS, deve-se incluir as bibliotecas **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD na concatenação dos conjuntos de dados JOBLIB ou STEPLIB.

Para executar um programa TSO, deve-se incluir as bibliotecas **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD ao STEPLIB usado pela sessão do TSO.

Para executar um programa em lote UNIX System Services a partir do shell UNIX System Services, inclua as bibliotecas **thlqual**.SCSQAUTH e **thlqual**.SCSQLOAD à especificação STEPLIB em seu $HOME?.profile desse modo:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Criando aplicativos em lote do z/OS usando o Language Environment*

IBM MQ for z/OS fornece um conjunto de bibliotecas de link dinâmico (DLLs) que deverá ser usado ao linkeditar seus aplicativos.

Existem duas variantes das bibliotecas que permitem que o aplicativo use uma das seguintes interfaces de chamada:

- A interface de chamada do ambiente de linguagem de 31 bits.

- A interface de chamada de XPLINK de 31 bits. z/OS XPLINK é uma convenção de chamada de alto desempenho disponível para aplicativos C.

Para usar as DLLs, o aplicativo é ligado ou vinculado às *bibliotecas auxiliares de módulos*, em vez de stubs fornecidos com versões anteriores. As bibliotecas auxiliares de módulos são localizadas na biblioteca SCSQDEFS (em vez da biblioteca SCSQLOAD).

Tabela 140. Variantes de bibliotecas de vínculo dinâmico

Confirmar	**DLL do Language Environment de 31 bits**	**DLL do XPLINK de 31 bits**	**Nome do stub equivalente**
Bibliotecas MQI de confirmação de fase 1	CSQBMQ1	CSQBMQ1X	CSQBSTUB
confirmação de fase 2 com coordenação RRS usando verbos de controle de transação RRS	CSQBRR1	CSQBRR1X	CSQBRSTB
confirmação de duas fases com coordenação RRS usando verbos de controle de transação MQI	CSQBRI1	CSQBRI1X	CSQBRRSI

Nota: Todas as bibliotecas auxiliares de módulos contêm uma definição do ponto de entrada de conversão de dados, MQXCNVC, anteriormente resolvido ao incluir o CSQASTUB.

Problemas comuns:

- A seguinte mensagem aparecerá no registro de tarefas, se seu aplicativo usar o consumo de mensagem assíncrona (chamadas MQCB, MQCTL ou MQSUB) e a interface DLL anterior não for usada:

  ```
  CSQB001E Programas de ambiente de linguagem em execução no USS ou no lote do z/OS devem usar a
  interface DLL para IBM MQ
  ```

  Solução: reconstrua seu aplicativo usando as bibliotecas auxiliares de módulos em vez de stubs como detalhado anteriormente.

- No momento da construção do programa, aparece a seguinte mensagem

  ```
  IEW2469E Os Atributos de uma referência a MQAPI-NAME a partir da seção your-code não
  correspondem aos atributos de
  símbolo de destino
  ```

  Razão: isso significa que você compilou seu programa XPLINK com a versão V701 (ou mais recente) do cmqc.h, mas não está ligando às bibliotecas auxiliares de módulos.

  Solução: mude o arquivo de construção de seu programa para ligação com relação à biblioteca auxiliar de módulos apropriada a partir do SCSQDEFS, em vez de um stub do SCSQLOAD

A JCL de amostra a seguir demonstra como é possível compilar e linkeditar m programa C para usar a interface de chamada da DLL do Language Environment com 31 bits:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
```

```
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

Nota: A compilação usa a opção **DLL**. A linkeditação usa a opção **DYNAM=DLL** e as referências da biblioteca **CSQBMQ1**.

A JCL de amostra a seguir demonstra como é possível compilar e linkeditar um programa C para usar a interface de chamada da DLL do XPLINK de 31 bits:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

Nota: A compilação usa as opções **XPLINK** e **DLL**. A linkedição usa a opção **DYNAM=DLL** e faz referência à biblioteca **CSQBMQ1X**.

Assegure-se de incluir a DLL de opção de compilação em cada programa no módulo. As mensagens como IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED são uma indicação de que é necessário verificar se todos os programas foram compilados com a opção DLL.

Criando aplicativos CICS no z/OS
Use essas informações ao criar aplicativos CICS no z/OS.

Para construir um aplicativo para o IBM MQ for z/OS executado sob o CICS, deve-se:

- Converta os comandos do CICS em seu programa para a linguagem na qual o restante do seu programa está escrito.
- Compile ou monte a saída do conversor para produzir código do objeto.
  - Para programas PL/I, use a opção do compilador EXTRN(SHORT).
  - Para aplicativos C, se o aplicativo não estiver usando XPLINK, use a opção do compilador DEFINE(MQ_OS_LINKAGE=1).
- Edite o link do código do objeto para criar um módulo de carregamento.

O CICS fornece um procedimento para executar essas etapas em sequência para cada uma das linguagens de programação que suporta.

- Para o Servidor de Transação CICS para z/OS, o *CICS Servidor de Transação para z/OS Guia de Definição do Sistema* descreve como utilizar esses procedimentos e o *Guia de Programação de Aplicativos CICS/ESA* fornece mais informações sobre o processo de tradução.

Deve-se incluir:

- Na instrução SYSLIB do estágio de compilação (ou montagem), as instruções que tornam os arquivos de definição de dados do produto disponíveis para o compilador. As definições de dados são fornecidas nas bibliotecas do IBM MQ for z/OS a seguir:
  - Para COBOL, **thlqual**.SCSQCOBC

  - Para linguagem assembler, **thlqual**.SCSQMACS
  - Para C, **thlqual**.SCSQC370
  - Para PL/I, **thlqual**.SCSQPLIC
- Em sua JCL de edição de link, o programa stub do IBM MQ for z/OS CICS (CSQCSTUB). Figura 127 na página 1045 mostra fragmentos de código JCL para fazer isso. O stub é independente da linguagem e é fornecido na biblioteca **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Figura 127. Fragmentos de JCL para editar o link do módulo de objeto no ambiente CICS

- Para versões do CICS posteriores ao CICS TS 3.2 ou se deseja usar as APIs de propriedade de mensagem do IBM MQ ou as APIs do IBM MQ MQCB, MQCTL, MQSTAT, MQSUB ou MQSUBR, deve-se editar o link de seu código de objeto com o stub fornecido pelo CICS, DFHMQSTB, não com o fornecido pelo IBM MQ, CSQCSTUB. Para obter mais informações sobre como construir programas IBM MQ para o CICS, consulte Programa stub de API para acessar chamadas MQI do IBM MQ na documentação do produto CICS.

Quando tiver concluído estas etapas, armazene o módulo de carregamento em uma biblioteca de carregamento de aplicativos e defina o programa para CICS da maneira usual.

Antes de executar um programa CICS, o administrador do sistema deve defini-lo para o CICS como um programa e transação do IBM MQ. Será possível executá-lo comumente.

Criando aplicativos IMS (BMP ou MPP)
Use estas informações ao criar aplicativos IMS (BMP ou MPP).

Se estiver construindo programas DL/I em lote, consulte “Criando aplicativos em lote do z/OS” na página 1041. Para construir outros aplicativos que são executados no IMS (como um BMP ou um MPP), crie JCL que execute estas tarefas:

1. Compile (ou monte) o programa para produzir código do objeto. O JCL da sua compilação deve incluir instruções SYSLIB que tornam os arquivos de definição dos dados de produto disponíveis para o compilador. As definições de dados são fornecidas nas bibliotecas do IBM MQ for z/OS a seguir:
   - Para COBOL, **thlqual**.SCSQCOBC
   - Para linguagem assembler, **thlqual**.SCSQMACS
   - Para C, **thlqual**.SCSQC370
   - Para PL/I, **thlqual**.SCSQPLIC
2. Para um aplicativo C, faça um prelink do módulo de objeto criado na etapa “1” na página 1045.
3. Para programas PL/I, use a opção do compilador EXTRN(SHORT).
4. Para um aplicativo C, se o aplicativo não estiver usando XPLINK, use a opção do compilador DEFINE(MQ_OS_LINKAGE=1).
5. Edite o link do código do objeto criado na etapa “1” na página 1045 (ou na etapa “2” na página 1045 para um aplicativo C/370) para produzir um módulo de carregamento:
   a. Inclua o módulo da interface de linguagem do IMS (DFSLI000).
   b. Inclua o programa stub do IBM MQ for z/OS IMS (CSQQSTUB). Figura 128 na página 1046 mostra fragmentos de JCL para fazer isso. O stub é independente da linguagem e é fornecido na biblioteca **thlqual**.SCSQLOAD.

Nota: Se você estiver usando COBOL, selecione a opção do compilador NODYNAM para ativar o editor de ligação para resolver as referências a CSQQSTUB, a menos que pretenda usar o link dinâmico conforme descrito em "Chamando dinamicamente o stub do IBM MQ" na página 1046.

6. Armazene o módulo de carregamento em uma biblioteca de carregamento do aplicativo.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Figura 128. Fragmentos de JCL para editar o link do módulo de objeto no ambiente IMS

Antes de executar um programa IMS, o administrador do sistema deve defini-lo para o IMS como um programa e transação do IBM MQ; será possível executá-lo comumente.

Construindo aplicativos z/OS UNIX System Services
Use estas informações ao construir aplicativos z/OS UNIX System Services.

Para construir um aplicativo C para o IBM MQ for z/OS que é executado no UNIX System Services, compile e vincule seu aplicativo da maneira a seguir:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

em que **thlqual** é o qualificador de alto nível usado por sua instalação.

Para executar o programa C, é necessário incluir o seguinte em seu arquivo `.profile`; isso deve estar em seu diretório raiz:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Observe que você precisa sair do UNIX System Services e entrar no UNIX System Services novamente para que a mudança seja reconhecida.

Se desejar executar vários shells, inclua a palavra export no início da linha, ou seja:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Quando isso for concluído com êxito, será possível vincular as chamadas CSQBSTUB e IBM MQ .

O "Chamando dinamicamente o stub do IBM MQ" na página 1046 descreve um método alternativo de fazer chamadas MQI em seus programas de forma que não seja necessário editar o link de um stub do IBM MQ. Esse método não está disponível para todas as linguagens e ambientes.

Não edite o link de um nível mais alto do programa stub do que aquele da versão do IBM MQ for z/OS em que seu programa está em execução. Por exemplo, um programa em execução no IBM WebSphere MQ for z/OS 7.1 não deve ser linkeditado com um programa stub fornecido com o IBM MQ for z/OS 8.0.

Chamando dinamicamente o stub do IBM MQ

Em vez de linkeditar o programa stub IBM MQ com o seu código de objeto, é possível chamar dinamicamente o stub de dentro do seu programa.

É possível fazer isso nos ambientes de lote, IMS e CICS. Esse recurso não é suportado no ambiente RRS. Se o seu programa de aplicativo usar RRS para coordenar atualizações, consulte "Considerações sobre RRS" na página 1051.

No entanto, este método:

- Aumenta a complexidade dos seus programas
- Aumenta o armazenamento necessário por seus programas no tempo de execução
- Reduz o desempenho dos seus programas
- Significa que não será possível usar os mesmos programas em outros ambientes

Se você chamar o stub dinamicamente, o programa stub apropriado e seus aliases devem estar disponíveis no tempo de execução. Para assegurar isso, inclua o conjunto de dados do IBM MQ for z/OS SCSQLOAD:

- Para lote e IMS, na concatenação de STEPLIB do JCL.
- Para CICS, na concatenação de DFHRPL do CICS.

Para o IMS, assegure-se de que a biblioteca que contém o stub dinâmico (construído conforme descrito nas informações sobre como instalar o adaptador do IMS em Configurando o adaptador do IMS) esteja à frente do conjunto de dados SCSQLOAD na concatenação STEPLIB da JCL da região.

Use os nomes mostrados em Tabela 141 na página 1047 ao chamar o stub dinamicamente. Em PL/I, somente declare os nomes de chamada usados em seu programa.

Tabela 141. Nomes de chamada para link dinâmico

Chamada MQI	Nomes de chamada dinâmica de lote (não RRS)	Nomes de chamada dinâmica do CICS	Nomes de chamada dinâmica do IMS
MQBACK	CSQBBACK	não é suportado	Não Suportado
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Não Suportado
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	não é suportado	Não Suportado
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Não Suportado
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

Tabela 141. Nomes de chamada para link dinâmico (continuação)			
Chamada MQI	**Nomes de chamada dinâmica de lote (não RRS)**	**Nomes de chamada dinâmica do CICS**	**Nomes de chamada dinâmica do IMS**
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Nota: 1. Essas chamadas de API estão disponíveis somente ao usar o CICS TS 3.2 ou posterior e o CSQCSTUB fornecido com o CICS deve ser usado. Para o CICS TS 3.2, o APAR PK66866 deve ser aplicado. Para o CICS TS 4.1, o APAR PK89844 deve ser aplicado.

Para obter exemplos de como usar essa técnica, consulte as figuras a seguir:

- Lote e COBOL: veja Figura 129 na página 1048
- CICS e COBOL: veja Figura 130 na página 1048
- IMS e COBOL: veja Figura 131 na página 1049
- Lote e assembler: veja Figura 132 na página 1049
- CICS e assembler: veja Figura 133 na página 1049
- IMS e assembler: veja Figura 134 na página 1049
- Lote e C: Figura 135 na página 1050
- CICS e C: veja Figura 136 na página 1050
- IMS e C: veja Figura 137 na página 1050
- Lote e PL/I: veja Figura 138 na página 1050
- IMS e PL/I: veja Figura 139 na página 1051

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figura 129. Link dinâmico usando COBOL no ambiente de lote

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figura 130. Link dinâmico usando COBOL no ambiente CICS

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                    PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figura 131. Link dinâmico usando COBOL no ambiente IMS

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

Figura 132. Link dinâmico usando linguagem assembly no ambiente de lote

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Figura 133. Link dinâmico usando linguagem assembly no ambiente CICS

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

Figura 134. Link dinâmico usando linguagem assembly no ambiente IMS

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 135. Link dinâmico usando linguagem C no ambiente de lote

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 136. Link dinâmico usando linguagem C no ambiente CICS

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figura 137. Link dinâmico usando linguagem C no ambiente IMS

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Figura 138. Link dinâmico usando PL/I no ambiente de lote

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Figura 139. Link dinâmico usando PL/I no ambiente IMS

Considerações sobre RRS
Considere usar estas informações se o seu programa de aplicativo usar RRS para coordenar as atualizações.

O IBM MQ fornece dois stubs diferentes para programas em lote que precisam de coordenação RRS; consulte "O adaptador em lote RRS" na página 890. A diferença no comportamento das chamadas de API posteriores é determinada no tempo de MQCONN pelo adaptador em lote a partir de informações transmitidas pela rotina de stub na chamada API MQCONN ou MQCONNX. Isso significa que as chamadas API dinâmicas estão disponíveis para programas em lote que precisa de coordenação RRS, desde que a conexão inicial com o IBM MQ tenha sido feita usando o stub apropriado. O exemplo a seguir ilustra isso:

```
       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.
```

Depurando seus programas

Use essas informações para saber sobre depuração do TSO e programas CICS e obter um insight sobre rastreio do CICS.

Os principais auxílios para depuração de programas de aplicativo IBM MQ for z/OS são os códigos de razão retornados por cada chamada de API. Para obter uma lista deles, incluindo ideias para ação corretiva, consulte:

- mensagens, conclusão e códigos de razão do IBM MQ for z/OS para IBM MQ for z/OS
- Mensagens e códigos de razão para todas as outras plataformas IBM MQ

Este tópico também sugere outras ferramentas de depuração para uso em ambientes específicos.

Depurando os programas TSO

As ferramentas de depuração interativa a seguir estão disponíveis para os programas TSO:

- Ferramenta TEST
- Ferramenta de depuração interativa VS COBOL II
- Ferramenta de depuração interativa INSPECT para programas C e PL/I

Depurando programas CICS

É possível usar o CICS Execution Diagnostic Facility (CEDF) para testar seus programas CICS interativamente sem precisar modificar o programa ou o procedimento de preparação do programa.

Para obter mais informações sobre a EDF, consulte o *Guia de programação de aplicativos do CICS Transaction Server for z/OS CICS*.

Rastreio do CICS

Você provavelmente também achará útil usar a transação Controle de rastreio do CICS (CETR) para controlar a atividade de rastreio do CICS.

Para obter mais informações sobre o CETR, consulte o manual *CICS Transaction Server for z/OS CICS - Transações fornecidas*.

Para determinar se o rastreio do CICS está ativo, exiba o status da conexão usando o painel CKQC. Esse painel também mostra o número de rastreio.

Para interpretar as entradas de rastreio do CICS, consulte Tabela 142 na página 1052.

A entrada de rastreio do CICS para esses valores é AP0 *xxx* (em que *xxx* é o número de rastreio especificado quando o adaptador do CICS foi ativado). Todas as entradas de rastreio, exceto CSQCTEST, são emitidas por CSQCTRUE. CSQCTEST é emitida por CSQCRST e CSQCDSP.

Tabela 142. Entradas de rastreio do adaptador do CICS

Nome	Descrição	Sequência de rastreio	Dados de rastreio
CSQCABNT	Finalização anormal	Antes de emitir END_THREAD ABNORMAL para IBM MQ. Isso ocorre por causa do término da tarefa e uma restauração implícita pôde ser executada pelo aplicativo. Uma solicitação de ROLLBACK é incluída na chamada END_THREAD neste caso.	Informações da unidade de trabalho. É possível usar essas informações ao descobrir sobre o status do trabalho. (Por exemplo, ele pode ser verificado com relação à saída produzida pelo comando DISPLAY THREAD ou o utilitário de impressão de log do IBM MQ for z/OS.)
CSQCBACK	Restauração do ponto de sincronização	Antes de emitir BACKOUT para IBM MQ for z/OS. Isso é devido a uma solicitação de restauração explícita do aplicativo.	Informações da unidade de trabalho.
CSQCCCRC	Código de conclusão e código de razão	Após retorno mal sucedido da chamada de API.	Código de conclusão e código de razão.

Tabela 142. Entradas de rastreio do adaptador do CICS (continuação)

Nome	Descrição	Sequência de rastreio	Dados de rastreio
CSQCCOMM	Confirmação do ponto de sincronização	Antes de emitir COMMIT para IBM MQ for z/OS. Isso pode ser devido a uma solicitação de single-phase commit ou a segunda fase de uma solicitação de two-phase commit. A solicitação é devido a uma solicitação explícita do ponto de sincronização a partir do aplicativo.	Informações da unidade de trabalho.
CSQCEXER	Resolução de execução	Antes de emitir EXECUTE_RESOLVE para IBM MQ for z/OS.	As informações da unidade de trabalho que está emitindo EXECUTE_RESOLVE. Essa é a última unidade de trabalho em dúvida no processo de ressincronização.
CSQCGETW	Espera de GET	Antes de emitir espera do CICS.	Endereço do ECB a ser esperado.
CSQCGMGD	Dados da mensagem GET	Após retorno bem-sucedido de MQGET.	Até 40 bytes dos dados da mensagem.
CSQCGMGH	Identificador da mensagem GET	Antes de emitir MQGET para IBM MQ for z/OS.	Manipulação de objetos.
CSQCGMGI	ID de mensagem Get	Após retorno bem-sucedido de MQGET.	ID da mensagem e ID de correlação da mensagem.
CSQCINDL	Lista de em dúvida	Após retorno bem-sucedido do segundo INQUIRE_INDOUBT.	A lista de unidades de trabalho em dúvida.
CSQCINDO	Somente uso de IBM		
CSQCINDS	Tamanho da lista de em dúvida	Após o retorno bem-sucedido do primeiro INQUIRE_INDOUBT e a lista de em dúvida não está vazia.	Comprimento da lista. Dividido por 64 fornece o número de unidades de trabalho em dúvida.
CSQCINQH	Identificador INQ	Antes de emitir MQINQ para IBM MQ for z/OS.	Manipulação de objetos.
CSQCLOSH	Identificador de CLOSE	Antes de emitir MQCLOSE para IBM MQ for z/OS.	Manipulação de objetos.
CSQCLOST	Disposição perdida	Durante o processo de ressincronização, o CICS informa ao adaptador que foi reiniciado, portanto, nenhuma informação de disposição referente à unidade de trabalho que está sendo ressincronizada está disponível.	ID da unidade de trabalho conhecido do CICS para a unidade de trabalho que está sendo ressincronizada.

Tabela 142. Entradas de rastreio do adaptador do CICS (continuação)

Nome	Descrição	Sequência de rastreio	Dados de rastreio
CSQCNIND	Disposição não em dúvida	Durante o processo de ressincronização, o CICS informa o adaptador que a unidade de trabalho que está sendo ressincronizada não deveria estar em dúvida (ou seja, possivelmente ainda está em execução).	ID da unidade de trabalho conhecido do CICS para a unidade de trabalho que está sendo ressincronizada.
CSQCNORT	Finalização normal	Antes de emitir END_THREAD NORMAL para IBM MQ for z/OS. Isso é devido ao final da tarefa e, portanto, o aplicativo pode executar uma confirmação implícita do ponto de sincronização. Uma solicitação COMMIT é incluída na chamada END_THREAD neste caso.	Informações da unidade de trabalho.
CSQCOPNH	Identificador OPEN	Após retorno bem-sucedido de MQOPEN.	Manipulação de objetos.
CSQCOPNO	Objeto OPEN	Antes de emitir MQOPEN para IBM MQ for z/OS.	Nome do objeto.
CSQCPMGD	Dados da mensagem PUT	Antes de emitir MQPUT para IBM MQ for z/OS.	Até 40 bytes dos dados da mensagem.
CSQCPMGH	Identificador de mensagem PUT	Antes de emitir MQPUT para IBM MQ for z/OS.	Manipulação de objetos.
CSQCPMGI	ID da mensagem PUT	Após MQPUT bem-sucedido a partir do IBM MQ for z/OS.	ID da mensagem e ID de correlação da mensagem.
CSQCPREP	Preparação do ponto de sincronização	Antes de emitir PREPARE para IBM MQ for z/OS na primeira fase do processamento de two-phase commit. Essa chamada também pode ser emitida a partir do componente de enfileiramento distribuído como uma chamada de API.	Informações da unidade de trabalho.
CSQCP1MD	Dados da mensagem PUTONE	Antes de emitir MQPUT1 para IBM MQ for z/OS.	Até 40 bytes de dados da mensagem.
CSQCP1MI	ID da mensagem PUTONE	Após retorno bem-sucedido de MQPUT1.	ID da mensagem e ID de correlação da mensagem.
CSQCP1ON	Nome do objeto PUTONE	Antes de emitir MQPUT1 para IBM MQ for z/OS.	Nome do objeto.
CSQCRBAK	Restauração resolvida	Antes de emitir RESOLVE_ROLLBACK para IBM MQ for z/OS.	Informações da unidade de trabalho.

Tabela 142. Entradas de rastreio do adaptador do CICS (continuação)

Nome	Descrição	Sequência de rastreio	Dados de rastreio
CSQCRCMT	Confirmação resolvida	Antes de emitir RESOLVE_COMMIT para IBM MQ for z/OS.	Informações da unidade de trabalho.
CSQCRMIR	Resposta de RMI	Antes de retornar à RMI (interface de gerenciador de recursos) do CICS de uma chamada específica.	Valor da resposta de RMI arquitetado. Seu significado depende do tipo da chamada. Esses valores estão documentados no *Guia de Customização do CICS Servidor de Transação z/OS*. Para determinar o tipo de chamada, consulte as entradas de rastreio anteriores produzidas pelo componente RMI do CICS.
CSQCRSYN	Ressincronização	Antes do processo de ressincronização ser iniciado para a tarefa.	ID da unidade de trabalho conhecido do CICS para a unidade de trabalho que está sendo ressincronizada.
CSQCSETH	Identificador de SET	Antes de emitir MQSET para IBM MQ for z/OS.	Manipulação de objetos.
CSQCTASE	Somente uso de IBM		
CSQCTEST	Teste de rastreio	Usado na chamada EXEC CICS ENTER TRACE para verificar o número de rastreio fornecido pelo usuário ou o status de rastreio da conexão.	Sem dados.
CSQCDCFF	Somente uso de IBM		

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 para 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 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 851) 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 “Gerenciamento de transações e serviços do gerenciador de recurso recuperável” na página 856.

IBM i 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 UNIX, 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 851). 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. ULW Com IBM MQ em UNIX, 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 UNIX, Linux, and Windows, consulte Administrando. 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, UNIX, 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, Windows, UNIX and Linux, 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 796 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 16.

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 965.

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 16 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 1060.

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* é MQAT_CICS e o campo *PutApplName* é o *ApplId* do sistema CICS seguido pelo nome de 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 1061 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 27.

A tabela a seguir lista todos os principais conceitos de MQI e como eles se relacionam com a Multicast:

Tabela 143. 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
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

Tabela 143. conceitos de MQI e como eles se relacionam com multicast (continuação)		
Conceito de MQI	**Ação ao tentar usar multicast**	**Código de razão**
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 988 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 144. 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

Tabela 144. Descrições de códigos de eventos multicast (continuação)

Código de evento	Descrição	Dados adicionais
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
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

Tabela 144. Descrições de códigos de eventos multicast (continuação)

Código de evento	Descrição	Dados adicionais
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	
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.

- “Valores iniciais para as estruturas dinâmicas” na página 1069
- “Uso de C++” na página 1070

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. Por exemplo, o símbolo ¬ representa um 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: 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 informações adicionais, consulte Desenvolvendo aplicativos .NET.

V 9.0.0 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 1037
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 920
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.

Codificação na linguagem assembler do System/390 (Message Queue Interface)

Observe as informações nas seções a seguir ao codificar programas IBM MQ for z/OS na linguagem assembler.

- "Nomes" na página 1072

Nomes

Os nomes de parâmetros nas descrições de chamadas e os nomes de campos nas descrições de estruturas são mostrados em uma combinação de maiúsculas e minúsculas. No macros da linguagem assembler fornecidas com o IBM MQ, todos os nomes estão em maiúsculas.

Usando as chamadas MQI

A MQI é uma interface de chamada, portanto, programas em linguagem assembler devem observar a convenção de ligação do SO.

Especificamente, antes de emitir uma chamada MQI, programas em linguagem assembler devem registrar o ponto R13 em uma área de salvamento de pelo menos 18 palavras completas. Essa área de salvamento fornece armazenamento para o programa chamado. Ela armazena os registros do responsável pela chamada antes de seus conteúdos serem destruídos e restaura o conteúdo dos registros do responsável pela chamada no retorno.

Nota: Isso é importante para programas em linguagem assembler do CICS que usam a macro DFHEIENT para configurar o armazenamento dinâmico, mas que optaram por substituir o DATAREG padrão de R13 por outros registos. Quando a CICS Resource Manager Interface recebe o controle do stub, ela salva o conteúdo atual dos registros no endereço para o qual R13 está apontando. Deixar de reservar uma área de salvamento para esse propósito fornece resultados imprevisíveis e provavelmente irá causar uma finalização anormal de tarefa no CICS.

Declarando constantes

A maioria das constantes é declarada como equivalente na macro CMQA.

No entanto, as constantes a seguir não podem ser definidas como equivalentes e não são incluídas quando você chama a macro usando as opções padrão:

- 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

Para incluí-las, inclua a palavra-chave EQUONLY=NO ao chamar a macro.

CMQA é protegido contra várias declarações, de modo que é possível incluí-la muitas vezes. No entanto, a palavra-chave EQUONLY terá efeito somente na primeira vez que a macro for incluída.

Especificando o nome de uma estrutura

Para permitir que mais de uma instância de uma estrutura seja declarada, a macro que gera os prefixos de estrutura coloca como prefixo do nome de cada campo uma sequência que pode ser especificada pelo usuário e um caractere sublinhado (_).

Especifique a sequência quando você chamar a macro. Se não especificar uma sequência, a macro usa o nome da estrutura para construir o prefixo:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

As declarações de estrutura em Descrições de chamada mostram o prefixo padrão.

Especificando a forma de uma estrutura

As macros podem gerar declarações de estrutura em uma de duas formas, controladas pelo parâmetro DSECT:

DSECT=YES
Uma instrução DSECT da linguagem assembler é usada para iniciar uma nova seção de dados; a definição de estrutura segue imediatamente a instrução DSECT. Nenhum armazenamento é alocado, portanto, nenhuma inicialização é possível. O rótulo na chamada da macro é usado como o nome da seção de dados; se nenhum rótulo for especificado, o nome da estrutura será usado.

DSECT=NO
Instruções DC da linguagem assembler são usadas para definir a estrutura na posição atual na rotina. Os campos são inicializados com valores, que é possível especificar codificando os parâmetros relevantes na chamada da macro. Campos para os quais nenhum valor é especificado na chamada de macro são inicializados com valores padrão.

DSECT=NO será assumido se o parâmetro DSECT não for especificado.

Controlando a listagem

É possível controlar a aparência da declaração de estrutura na listagem da linguagem assembler com o parâmetro LIST:

LIST=YES
A declaração de estrutura aparece na listagem de linguagem assembler.

LIST=NO
A declaração de estrutura não aparece na listagem de linguagem assembler. Isso é assumido se o parâmetro LIST não for especificado.

Especificando valores iniciais para campos

É possível especificar o valor a ser usado para inicializar um campo em uma estrutura codificando o nome desse campo (sem o prefixo) como um parâmetro na chamada da macro, acompanhado pelo valor requerido.

Por exemplo, para declarar uma estrutura do descritor de mensagens com o campo *MsgType* inicializado com MQMT_REQUEST e o campo *ReplyToQ* inicializado com a sequência MY_REPLY_TO_QUEUE, use o código a seguir:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

Se você especificar uma constante denominada (ou equivalente) como um valor na chamada da macro, use a macro CMQA para definir a constante denominada. Não se deve colocar entre aspas simples (' ') valores que são sequências de caracteres.

Escrevendo programas reentrantes

O IBM MQ usa suas estruturas para entrada e saída. Se deseja que seu programa permaneça reentrante:

1. Defina versões de armazenamento de funcionamento das estruturas como DSECTs ou defina as estruturas sequenciais dentro de um DSECT já definido. Em seguida, copie o DSECT para o armazenamento que é obtido usando:
   - Para programas em lote e TSO, as macros STORAGE ou GETMAIN em assembler do z/OS
   - Para o CICS, o armazenamento de funcionamento DSECT (DFHEISTG) ou o comando EXEC CICS GETMAIN

   Para inicializar corretamente essas estruturas de armazenamento de funcionamento, copie uma versão constante da estrutura correspondente para a versão de armazenamento de funcionamento.

   Nota: As estruturas MQMD e MQXQH têm cada uma mais de 256 bytes de comprimento. Para copiar essas estruturas para o armazenamento, use a instrução assembler MVCL.
2. Reserve espaço no armazenamento usando o formato de LIST (MF=L) da macro CALL. Ao usar a macro CALL para fazer uma chamada MQI, use o formato EXECUTE (MF=E) da macro, usando o armazenamento reservado anteriormente, conforme mostrado no exemplo em "Usando CEDF" na página 1074. Para obter mais exemplos de como fazer isso, consulte os programas de amostra de linguagem assembler conforme enviados com o IBM MQ.

Use a opção RENT de linguagem assembler para ajudá-lo a determinar se seu programa é reentrante.

Para obter informações sobre como gravar programas reenterable, consulte *z/OS MVS Application Development Guide: Assembler Language Programs*.

Usando CEDF

Se quiser usar a transação fornecida pelo CICS, CEDF (Recurso de diagnóstico de execução do CICS) para ajudá-lo a depurar seu programa, inclua a palavra-chave ,VL em cada instrução CALL, por exemplo:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

O exemplo anterior é código da linguagem assembler reentrante, em que PARMAREA é uma área no armazenamento de funcionamento que você especificou.

Usando as chamadas MQI

A MQI é uma interface de chamada, portanto, programas em linguagem assembler devem observar a convenção de ligação do SO. Especificamente, antes de emitir uma chamada MQI, programas em linguagem assembler devem registrar o ponto R13 em uma área de salvamento de pelo menos 18 palavras completas. Essa área de salvamento fornece armazenamento para o programa chamado. Ela armazena os registros do responsável pela chamada antes de seus conteúdos serem destruídos e restaura o conteúdo dos registros do responsável pela chamada no retorno.

Nota: Isso é importante para programas em linguagem assembler do CICS que usam a macro DFHEIENT para configurar o armazenamento dinâmico, mas que optaram por substituir o DATAREG padrão de R13

por outros registos. Quando a CICS Resource Manager Interface recebe o controle do stub, ela salva o conteúdo atual dos registros no endereço para o qual R13 está apontando. Deixar de reservar uma área de salvamento adequada para esse propósito fornece resultados imprevisíveis e provavelmente irá causar uma finalização anormal de tarefa no 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).

Codificando em PL/I (z/OS somente)

Informações úteis ao codificar para IBM MQ em PL/I.

Estruturas

Estruturas são declarados com o atributo BASED e, portanto, não ocupam qualquer armazenamento, a menos que o programa declare uma ou mais instâncias de uma estrutura.

Uma instância de uma estrutura pode ser declarada usando o atributo `like`, por exemplo:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

Os campos de estrutura são declarados com o atributo INITIAL; quando o atributo `like` for usado para declarar uma instância de uma estrutura, essa instância herdará os valores iniciais definidos para essa estrutura. Você precisa configurar somente os campos nos quais o valor necessário for diferente do valor inicial.

PL/I não faz distinção entre maiúsculas e minúsculas, portanto, os nomes de chamadas, campos de estrutura e constantes podem ser codificados em minúsculas, maiúsculas ou em uma combinação de maiúsculas e minúsculas.

Constantes nomeadas

As constantes denominadas são declaradas como variáveis de macro; como resultado, constantes denominadas que não são referidas pelo programa não ocupam qualquer armazenamento no procedimento compilado.

No entanto, a opção do compilador que faz com que a origem seja processada pelo pré-processador de macro deve ser especificada quando o programa for compilado.

Todas as variáveis de macro são variáveis de caracteres, mesmo aquelas que representam valores numéricos. Embora isso possa parecer contraintuitivo, não resulta em qualquer conflito de tipo de dados após as variáveis de macro terem sido substituídas pelo processador de macro, por exemplo:

```
%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:

- Programas de amostra para sistemas distribuídos e IBM i.
- Programas de amostra para z/OS.

Procedimento

- Use os seguintes links para descobrir mais sobre os programas de amostra:
  - “Usando os programas de amostra em multiplataformas” na página 1076
  - z/OS “Usando os programas de amostra para z/OS” na página 1182

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

“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 47
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 715
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

“Escrevendo aplicativos de publicar/assinar” na página 806
Inicie a escrever aplicativos de publicar/assinar IBM MQ.

“Construindo um aplicativo processual” na página 1005
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 1055
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

“Desenvolvendo serviços da web com o IBM MQ” na página 1306
É possível desenvolver aplicativos IBM MQ para serviços da web usando o transporte IBM MQ para SOAP.

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 1086.

Procedimento

- Use os seguintes links para descobrir mais sobre os programas de amostra:
  - "Recursos demonstrados nos programas de amostra em multiplataformas" na página 1078
  - "Os programas de amostra Publish/Subscribe" na página 1115
  - "Os programas de amostra Put" na página 1120
  - "O programa de amostra Distribution List" na página 1107
  - "Os programas de amostra Browse" na página 1095
  - "O programa de amostra Browser" na página 1096
  - "Os programas de amostra Get" na página 1109
  - "Os programas de amostra Reference Message" na página 1122
  - "Os programas de amostra Request" na página 1129
  - "Os programas de amostra Inquire" na página 1114
  - "O programa de amostra Inquire Properties of a Message Handle" na página 1115
  - "Os programas de amostra Set" na página 1135
  - "Os programas de amostra Echo" na página 1108
  - "O programa de amostra Data-Conversion" na página 1099
  - "Os programas de amostra Triggering" na página 1139
  - "O programa de amostra Asynchronous Put" na página 1094
  - "Amostras de coordenação de banco de dados" na página 1099
  - "A transação de amostra do CICS" na página 1097
  - "Usando as amostras do TUXEDO no UNIX e Windows" na página 1141
  - "Amostra do manipulador de filas de mensagens não entregues" na página 1106
  - "O programa de amostra Connect" na página 1098
  - "O programa de amostra de saída de API" na página 1092
  - "Usando a saída de segurança SSPI no Windows" na página 1155
  - "Executando as amostras usando filas remotas" na página 1156
  - "O programa de amostra de Cluster Queue Monitoring (AMQSCLM)" na página 1156
  - "Programa de amostra para Connection Endpoint Lookup (CEPL)" na página 1166

Conceitos relacionados

“Programas de amostra C++” na página 508
Quatro programas de amostra são fornecidos, para demonstrar a obtenção e a colocação de mensagens.

Tarefas relacionadas

“Usando os programas de amostra para z/OS” na página 1182
Os aplicativos processuais de amostra que são entregues com o IBM MQ for z/OS demonstram usos típicos do 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 “Usando os programas de amostra para z/OS” na página 1182.

Linux UNIX *Amostras para sistemas UNIX and Linux*

As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas UNIX and Linux.

Veja “Preparando e executando programas de amostra em UNIX and Linux” na página 1089 para descobrir onde os programas de amostra para o IBM MQ em sistemas UNIX and Linux são armazenados.

Tabela 145 na página 1078 A tabela lista quais arquivos de origem C e COBOL são fornecidos e se um executável do servidor ou cliente é incluído.

Tabela 145. Programas de amostra do IBM MQ em UNIX and Linux demonstrando o uso de MQI (C e COBOL)

Técnica	**C (origem) (“1” na página 1080)**	**COBOL (origem) (“2” na página 1080)**	**Servidor (executável C)**	**Cliente (executável C) (“3” na página 1080)**
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 (“4” na página 1080)	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

Tabela 145. Programas de amostra do IBM MQ em UNIX and Linux demonstrando o uso de MQI (C e COBOL) (continuação)

Técnica	**C (origem) (“1” na página 1080)**	**COBOL (origem) (“2” na página 1080)**	**Servidor (executável C)**	**Cliente (executável C) (“3” na página 1080)**
Colocando mensagens de referência em uma fila (“4” na página 1080)	amqsprma	sem amostra	amqsprm	amqsprmc
Obtendo mensagens de referência de uma fila (“4” na página 1080)	amqsgrma	sem amostra	amqsgrm	amqsgrmc
Saída do canal de mensagem de referência (“4” na página 1080)	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
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	(“5” na página 1080)	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 (“6” na página 1080)	amqscic0.ccs	sem amostra	amqscic0	sem amostra
Transação Encina (“4” na página 1080)	amqsxae0	sem amostra	amqsxae0	sem amostra
Transação TUXEDO para colocar mensagens (“7” na página 1080)	amqstxpx	sem amostra	sem amostra	sem amostra
Transação TUXEDO para obter mensagens (“7” na página 1080)	amqstxgx	sem amostra	sem amostra	sem amostra

Tabela 145. Programas de amostra do IBM MQ em UNIX and Linux demonstrando o uso de MQI (C e COBOL) (continuação)

Técnica	**C (origem) ("1" na página 1080)**	**COBOL (origem) ("2" na página 1080)**	**Servidor (executável C)**	**Cliente (executável C) ("3" na página 1080)**
Servidor para TUXEDO ("7" na página 1080)	amqstxsx	sem amostra	sem amostra	sem amostra
Manipulador da fila de devoluções	Diretório `./tools/c/Samples/dlq` ("8" na página 1081)	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
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

Notes:

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. As versões executáveis das amostras do IBM MQ MQI client não estão disponíveis no IBM MQ for HP-UX.
4. Suportado em IBM MQ for AIX, IBM MQ for HP-UX e IBM MQ for Solaris apenas.
5. No IBM MQ for AIX, no IBM MQ for HP-UXe no IBM MQ for Solaris, esse programa é chamado `amqsvfc0.c`
6. O CICS é suportado apenas pelo IBM MQ for AIX e IBM MQ for HP-UX.
7. TUXEDO não é suportado por IBM MQ for Linux em System p.

8. 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 UNIX and Linux, veja Requisitos do sistema para IBM MQ.

Windows *Amostras para o IBM MQ for Windows*

As técnicas demonstradas pelos programas de amostra para o IBM MQ for Windows.

Tabela 146 na página 1081 lista quais arquivos de origem C e COBOL são fornecidos e se um executável do servidor ou cliente é incluído.

Tabela 146. 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

Tabela 146. 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)**
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
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 1083)	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

Tabela 146. 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)**
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

Notes:

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 147 na página 1083 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 1037.

Tabela 147. 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 o IBM i*

As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas IBM i.

O Tabela 148 na página 1084 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 148. Programas de amostra demonstrando o uso do MQI (C e COBOL) no IBM i

Técnica	**C (origem) ("1" na página 1085)**	**COBOL (origem) ("2" na página 1085)**	**RPG (origem) ("3" na página 1085)**	**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

Tabela 148. Programas de amostra demonstrando o uso do MQI (C e COBOL) no IBM i (continuação)

Técnica	**C (origem) ("1" na página 1085)**	**COBOL (origem) ("2" na página 1085)**	**RPG (origem) ("3" na página 1085)**	**Cliente (executável C)(4)**
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
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

Notes:

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 o IBM MQ com Java e JMS para obter informações adicionais.

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.

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:

- “Preparando e executando programas de amostra em IBM i” na página 1088
- “Preparando e executando programas de amostra em UNIX and Linux” na página 1089
- “Preparando e executando programas de amostra em Windows” na página 1091

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 1087.

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 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) Windows UNIX
   Para sistemas UNIX e Windows:

   ```
   runmqlsr -t tcp -m qmgr-name -p nnnn
   ```

   b) IBM i
   Para o IBM i:

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. Se seu aplicativo usar SYSTEM.DEF.SVRCONN, então, esse canal já está definido. Se seu aplicativo usar outro canal, crie-o emitindo o comando do MQSC:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name é o nome de seu canal.
4. Crie uma regra de autenticação de canal permitindo que somente o endereço IP do seu sistema cliente use o canal emitindo o comando do MQSC:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   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 1087
5. Se seu aplicativo usar SYSTEM.DEFAULT.LOCAL.QUEUE, então, essa fila já está definida. Se seu aplicativo usar outra fila, crie-a emitindo o comando do MQSC:

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   queue-name é o nome de sua fila.
6. Conceda acesso para se conectar e consultar o gerenciador de filas:

   a) Windows IBM i UNIX
   Para sistemas IBM i IBM i, UNIX e Windows, emita os comandos do MQSC:

   ```
   SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
   AUTHADD(CONNECT, INQ)
   ```

   non-privileged-user-id é o ID do usuário obtido na etapa “1” na página 1087

7. Se seu aplicativo for de ponto a ponto, ou seja, usa filas, conceda acesso para permitir a consulta e a colocação e obtenção de mensagens usando sua fila pelo ID do usuário a ser usado, emitindo os comandos do MQSC:

   a) Windows IBM i UNIX

   Para sistemas IBM i IBM i, UNIX e Windows, emita os comandos do MQSC:

   ```
   SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
   PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
   ```

   queue-name é o nome de sua fila.

   non-privileged-user-id é o ID do usuário obtido na etapa “1” na página 1087

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:

   a) Windows IBM i UNIX

   Para sistemas IBM i IBM i, UNIX e Windows, emita os comandos do MQSC:

   ```
   SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
   PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
   ```

   non-privileged-user-id é o ID do usuário obtido na etapa “1” na página 1087

   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.

Informações relacionadas

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

IBM i Autoridades do IBM MQ no IBM i

Fornecendo acesso a um objeto do IBM MQ em sistemas UNIX ou Linux e Windows

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.

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 1086.
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 o IBM i” na página 1084
As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas IBM i.

UNIX *Preparando e executando programas de amostra em UNIX and Linux*

Antes de executar os programas de amostra no UNIX, 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 UNIX and Linux estão nos diretórios listados em Tabela 149 na página 1089 se os padrões tiverem sido usados no momento da instalação.

Tabela 149. Onde localizar as amostras para IBM MQ em sistemas UNIX and Linux

Conteúdo	**Diretório**
arquivos de origem	*MQ_INSTALLATION_PATH*/samp

Tabela 149. Onde localizar as amostras para IBM MQ em sistemas UNIX and Linux (continuação)	
Conteúdo	**Diretório**
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.

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 como criar um gerenciador de filas, veja Administrando o 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 1086.
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 UNIX and Linux, insira:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   Verifique o arquivo `sampobj.out` para assegurar que não haja nenhum erro.
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 sistemas UNIX and Linux” na página 1078

As técnicas demonstradas pelos programas de amostra para o IBM MQ em sistemas UNIX and 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 150 na página 1091, se os padrões foram usados no momento da instalação. A unidade de instalação é padronizada como <c:>.

Tabela 150. 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\Samples
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\Samples

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.

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 como criar um gerenciador de filas, veja Administrando o 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 1086.
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 assegurar que não haja nenhum erro. Esse arquivo encontra-se em seu diretório atual.

Nota:

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 o IBM MQ for Windows" na página 1081
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 1083
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 de 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 "Escrevendo e compilando saídas de API" na página 956.

Origem

amqsaxe0.c

Binário

amqsaxe

Configurando para a saída de amostra

1. Inclua o seguinte ao 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=/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 em C e um executável binário de cliente e servidor nas plataformas Windows, UNIX and Linux.

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 1078 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 1078 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 UNIX, Linux, and Windows
Considere usar este tópico quando estiver aprendendo sobre programas de amostra Procurar no UNIX, 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.

ULW *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 1078 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)

Esses programas também usam uma variável de ambiente chamada **MQSAMP_USER_ID** que deveria ser configurada como o ID do usuário a ser usado para autenticação de conexão. Quando isso é configurado, o programa solicita uma senha para acompanhar esse ID do usuário.

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 151. Valores permitidos para o parâmetro da propriedade

Value	Comportamento
0	Comportamento padrão, como era para o IBM WebSphere MQ 6. As propriedades que são entregues ao aplicativo dependem do atributo de fila **PropertyControl** do qual a mensagem é recuperada.

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 WebSphere MQ 6 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 .

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 1005 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 as coloca 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 1078 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 140 na página 1100 mostra como as amostras fornecidas são usadas para atualizar os bancos de dados:

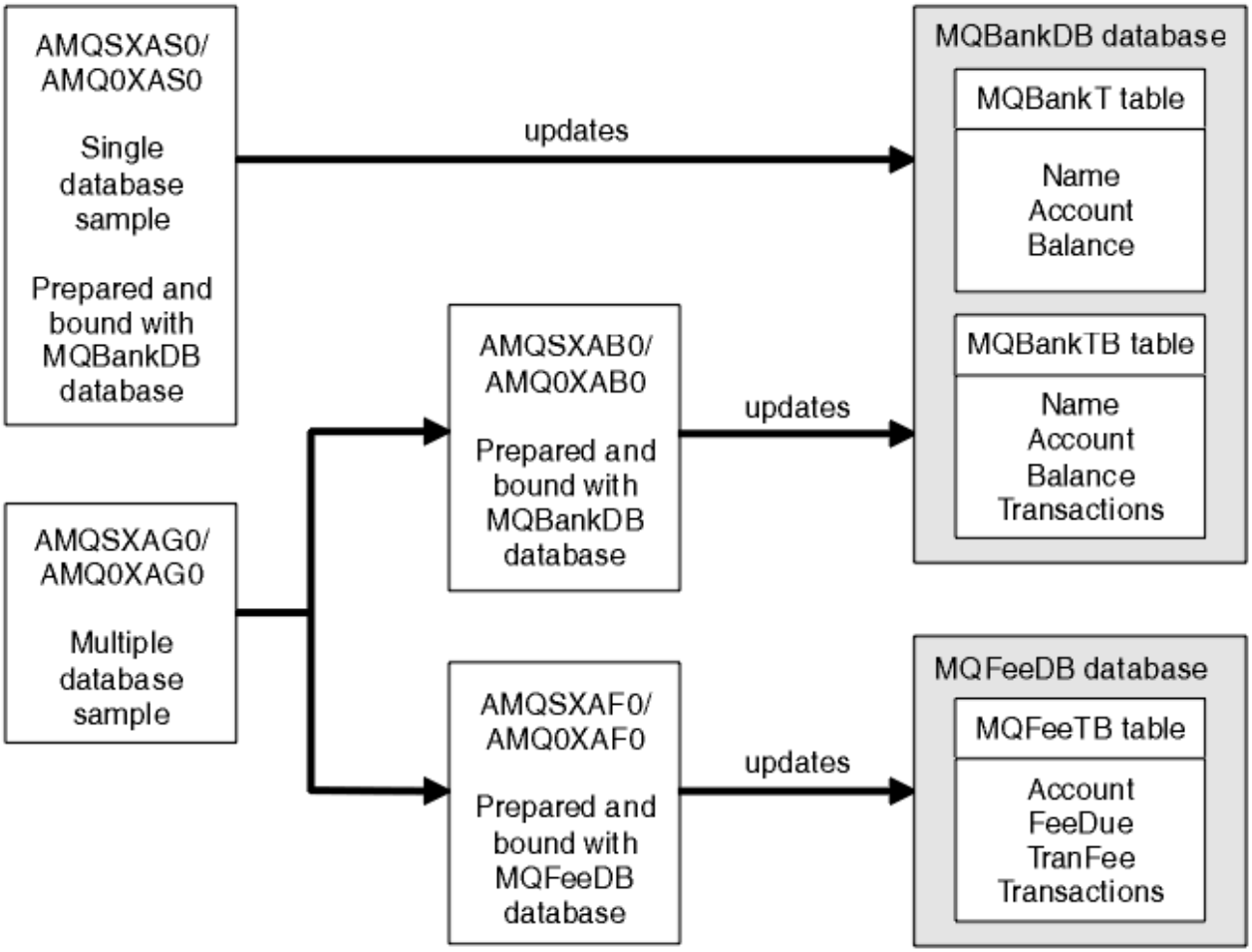


Figura 140. As amostras de coordenação de banco de dados

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 arquivos .SQB (em COBOL) e ligue-os em 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.

- No AIX, o caminho do diretório é:

```
/usr/lpp/db2_05_00
```

- No HP-UX e Solaris, o caminho do diretório é:

```
/opt/IBMdb2/V5.0
```

- Em sistemas Windows, o caminho do diretório depende do caminho escolhido ao instalar o produto. Se você escolher as configurações padrão, o caminho será:

```
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` de sua instalação do IBM MQ

  Nota: Esse arquivo pode ser encontrado nos diretórios a seguir:

  – Nos sistemas UNIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Nos sistemas Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Os arquivos `.c` obtidos pré-compilando os arquivos de origem `.sqc`, `amqsxas0.sqc, amqsxaf0.sqc` e `amqsxab0.sqc`
- Os arquivos `util.c` e `util.h` de sua instalação do Db2.

  Nota: Esses arquivos podem ser localizados no diretório:

  ```
  DB2TOP/samples/c
  ```

Crie os arquivos de objeto para cada arquivo `.c` usando o comando do compilador a seguir para a plataforma que você está usando:

- AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- HP-UX

```
cc -Aa +z -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Solaris

```
cc -Aa -KPIC -mt -I MQ_INSTALLATION_PATH
/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- 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

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- HP-UX Revision 11i

```
ld -E -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread -lcl
/lib/crt0.o util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Solaris

```
cc -mt -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm -lthread -lsocket -lc -lnsl -ldl util.o
amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- 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 vinculação a seguir para a plataforma que você está usando:

- AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- HP-UX Revision 11i

```
ld -E -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread
-lcl /lib/crt0.o util.o amqsxas0.o -o amqsxas0
```

- Solaris

```
cc -mt -L DB2TOP/lib -ldb2-L MQ_INSTALLATION_PATH/lib
-lqm -lthread -lsocket -lc -lnsl -ldl util.o
amqsxas0.o -o amqsxas0
```

- Sistemas Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Informações adicionais

Se você estiver trabalhando no AIX ou HP-UX 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 1105
- “Amostras em COBOL” na página 1105

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

Uma amostra de manipulador da fila de mensagens não entregues é fornecida, o nome da versão executável é amqsdlq. Se você deseja um manipulador da fila de devoluções diferente de RUNMQDLQ, a origem da amostra está 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 sim para ativar o rastreio

ODQ_MSG
Configure para o nome do arquivo que contém mensagens de erro e de informações. O arquivo fornecido é chamado 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 adequar a seus próprios requisitos. A amostra coloca mensagens no stdout, **não** no arquivo de log de erro do IBM MQ.

O Administrando ou o *Guia de gerenciamento de sistemas* para sua plataforma explica como o manipulador de mensagens não entregues funciona e como executá-lo.

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 1078 para obter os nomes desses programas.

Os programas são destinados a execução como programas acionados.

Nos sistemas IBM i, UNIX, 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 que o processo de acionamento funcione, certifique-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 1078 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, amqsget se conecta ao gerenciador de filas padrão e amqsgetc se conecta ao gerenciador de filas identificado por uma variável de ambiente ou o 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.

Esses programas também usam uma variável de ambiente chamada **MQSAMP_USER_ID** que deveria ser configurada como o ID do usuário a ser usado para autenticação de conexão. Quando isso for configurado, o programa solicitará uma senha para acompanhar esse ID do usuário.

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 de 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 tem 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 1112.

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 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 141 na página 1112.

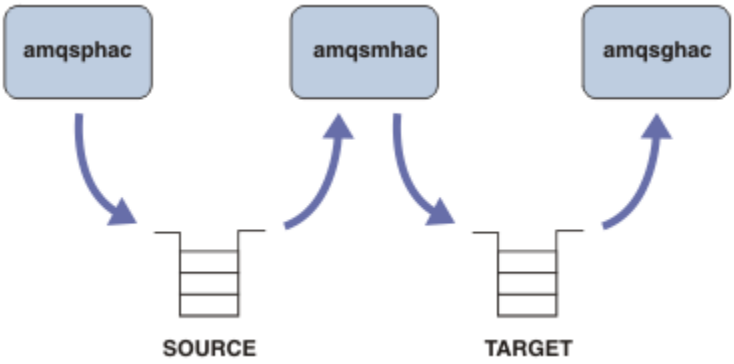


Figura 141. Amostras de clientes reconectáveis

Siga estas etapas para executar as amostras.

1. Crie um arquivo `hasamples.tst` que contenha os comandos:
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 1113 em “Configurar e controlar o gerenciador de filas” na página 1112.)
5. Digite o comando `endmqm -r -p QM1` para parar o gerenciador de filas e, em seguida, permitir 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.
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 1112 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
```

Informações relacionadas

Verificando o comportamento do sistema de arquivo compartilhado

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 1078 para obter os nomes desses programas.

Esses programas são projetados para execução como programas acionados, portanto única entrada é um estrutura MQTMC2 (mensagem do acionador) para sistemas IBM i, Windows, UNIX and Linux. 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:

- ULW Para UNIX, 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
Início de amostra AMQSIQMA
nome da propriedade MyProp valor MyValue
texto da mensagem Hello world!
Final da amostra AMQSIQMA
```

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 de 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 1086 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 comandos no mesmo diretório e execute o programa de assinante, `amqssub`, fornecendo a ele 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 da 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 152 na página 1116.

Tabela 152. Categorias de programas C de amostra Publish/Subscribe de legado

Categoria	Programas	Comentários
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 em `amqsgama.tst` e `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 em `amqsgama.tst` e `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.

Java

A amostra `MQPubSubApiSample.java` de Java combina publicador e assinantes em um único programa. Seus arquivos de origem e de classe compilada estão localizados 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 1086 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 1117 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 1086 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 campo **Nome do projeto**, digite `MQ Java Samples`. Clique em **Avançar**.
   e) No painel **Configurações de Java**, 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** > **Próximo** > **Procurar** ...e navegue até o caminho *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` em que *MQ_INSTALLATION_PATH* é o diretório de instalação do IBM MQ.
   k) No painel **Importar**, Figura 142 na página 1118, clique em `samples` (não selecione a caixa de seleção).
   l) Selecione `MQPubSubApiSample.java`. O campo **Na pasta** deve conter `MQ Java Samples/src`. Clique em **Concluir**.

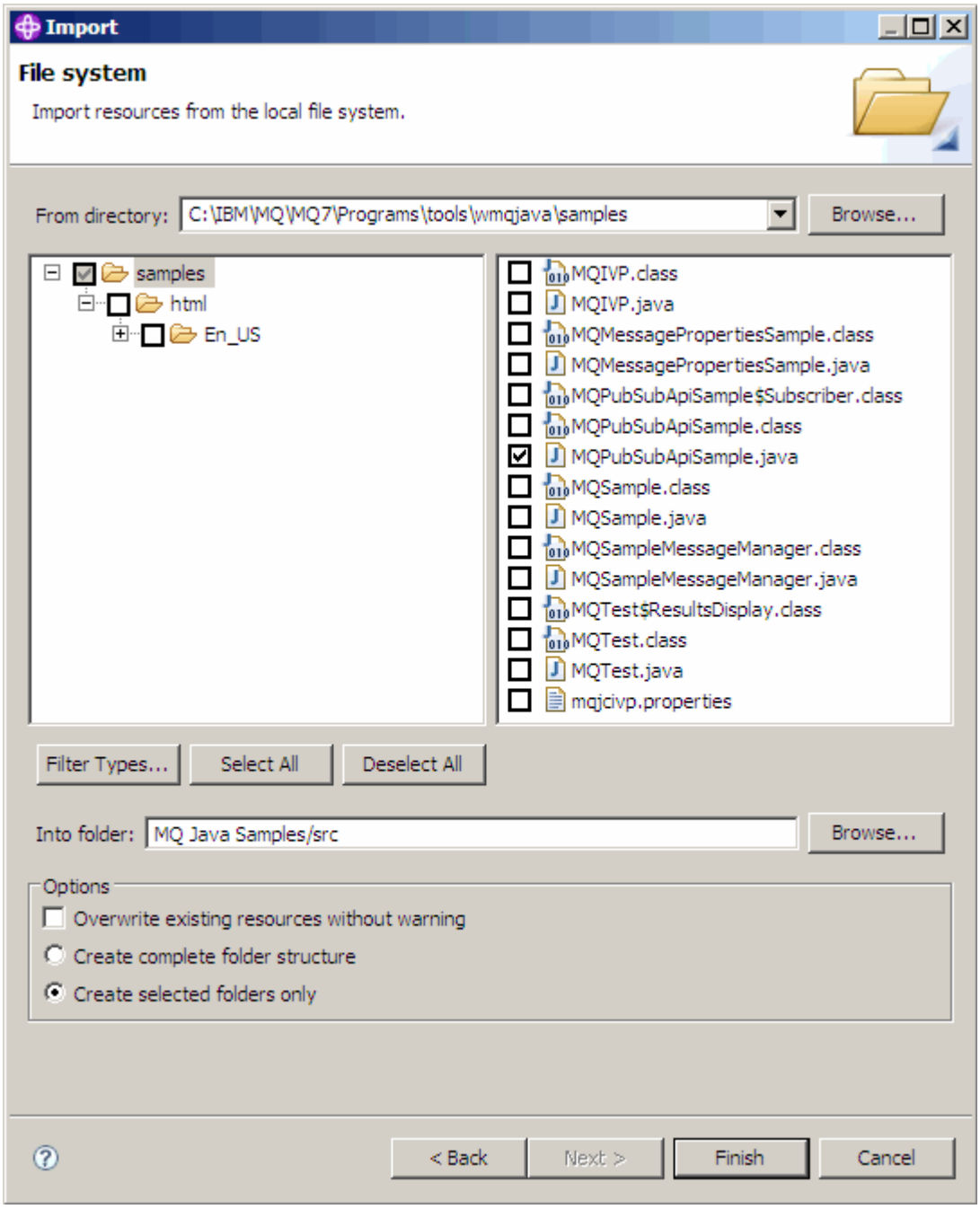


Figura 142. Importação do sistema de arquivos

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.

  - hostname
  - 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 UNIX Nos sistemas UNIX 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 UNIX Nos sistemas UNIX and Linux, assegure que o diretório `/var/mqm/trace` exista e exporte a variável de ambiente a seguir:

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows No Windows, assegure que o diretório `C:\temp` exista e configure a variável de ambiente a seguir:

     ```
     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 1078 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

ULW

A amostra amqqsput é o programa para colocar mensagens usando ligações locais e a amostra amqsputc é o programa para colocar usando ligações de cliente. 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, amqsput se conectará ao gerenciador de filas padrão e amqsputc se conectará ao gerenciador de filas identificado pela variável de ambiente MQSERVER ou pelo arquivo de definição de canal de 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.

Esses programas também usam uma variável de ambiente chamada **MQSAMP_USER_ID** que deveria ser configurada como o ID do usuário a ser usado para autenticação de conexão. Quando isso for configurado, o programa solicitará uma senha para acompanhar esse ID do usuário.

Para executar esses programas, insira uma das opções a seguir:

- `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

ULW

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.

Executando as amostras Reference Message
Use este exemplo para descobrir como executar o aplicativo de amostra Reference Message AMQSPRM 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:

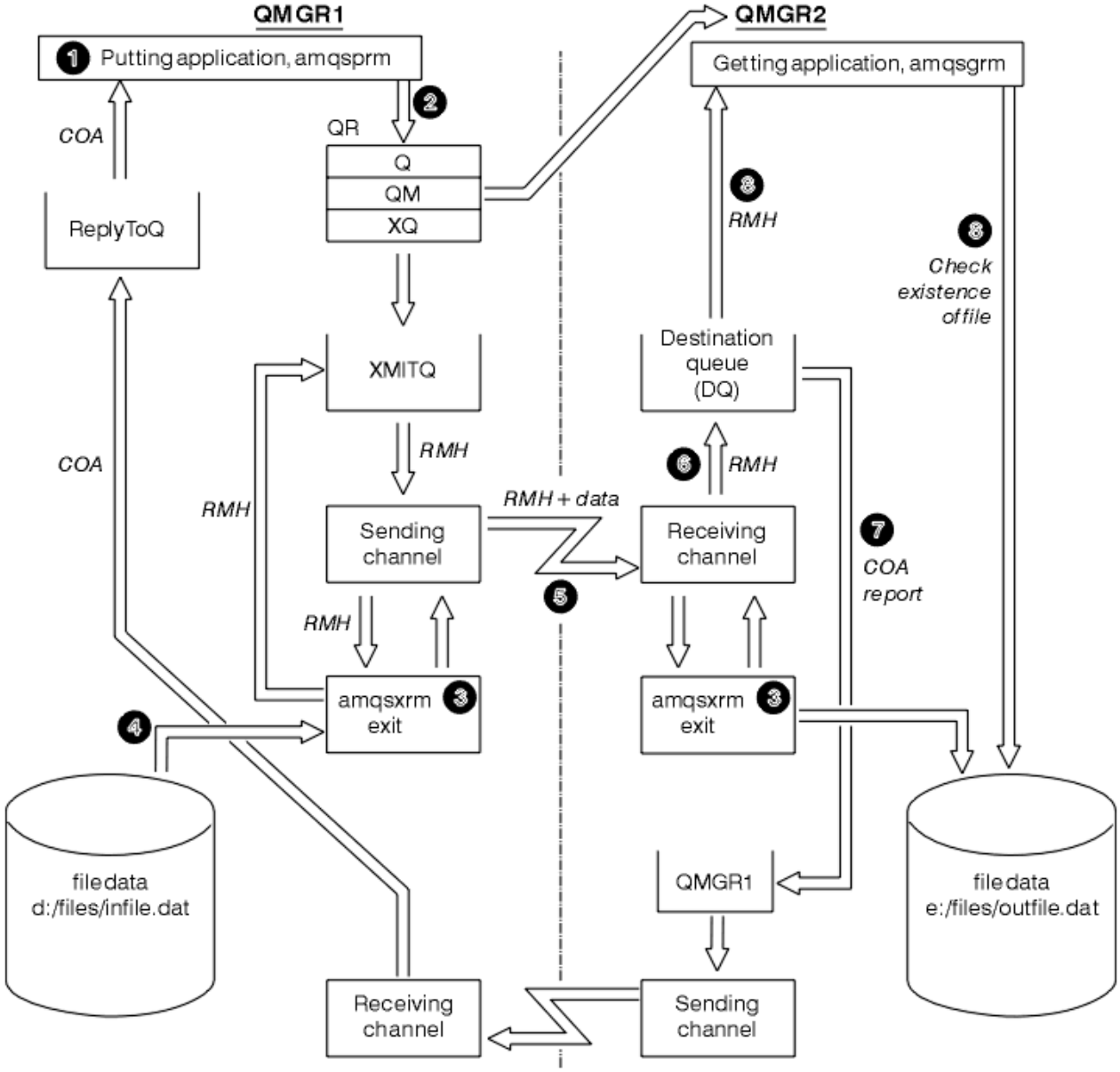


Figura 143. Executando as amostras Reference Message

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 Reference Message seja construída para enviar um arquivo com um tipo de objeto FLATFILE do gerenciador de filas QMGR1 para o QMGR2 e 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 a Mensagem de referência (AMQSGRM ou AMQSGRMA no IBM i) é acionado usando a fila de inicialização INITQ e o PROC do processo. 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.

As definições do MQSC usaram um estilo do AIX para definir as saídas, portanto, se você estiver usando o MQSC no IBM i, será necessário modificá-las apropriadamente. 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
```

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)
```

Sintaxe de comando do IBM i

Nota: IBM i Se você não especificar um nome de 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

a. Use o comando a seguir:

```
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 É possível usar uma barra (/) ou um traço (-) ao especificar parâmetros.

IBM i Por exemplo:

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Nota: Para as plataformas UNIX 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)')
```

Solaris HP-UX AIX Isso é definido no AIX, no HP-UX e no Solaris da seguinte forma:

```
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.

   Solaris HP-UX AIX No AIX, no HP-UX e no Solaris, mude o proprietário do grupo do diretório de destino para 'mqm' de forma 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 UNIX 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 fim do arquivo não for atingido, a Reference Message é colocada de volta na fila de transmissão após atualizar o 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 canais emissor e do servidor, se o campo *DataLogicalLength* na Message Reference de entrada for zero, a parte restante do arquivo, de *DataLogicalOffset* até o fim do arquivo, deve 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* será configurado para MQXCC_SUPPRESS_FUNCTION de forma que a mensagem sendo processada será colocada na fila de mensagens não entregues em vez de continuar para a fila de destino. Um código de feedback é retornado em MQCXP. *Feedback* e retornado ao aplicativo, que coloca a mensagem no campo *Feedback* do descritor de mensagens de uma mensagem de relatório. Isso ocorre porque o aplicativo de put solicitou relatórios de exceção configurando MQRO_EXCEPTION no campo *Report* do MQMD.

Se a codificação ou *CodedCharacterSetId* (CCSID) de Reference Message for diferente daquela do gerenciador de filas, Reference Message 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:

Em 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
```

Em HP-UX

HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -I MQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -L MQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

Em 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)`

Em 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
```

Em Solaris

Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -I  MQ_INSTALLATION_PATH/inc
-L  MQ_INSTALLATION_PATH/lib64 -R  MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm


-lsocket
-lnsl -ldl
```

Em 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 968
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 1078 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.

ULW *Executando a amostra de solicitação usando o acionamento no UNIX, Linux, and Windows*

No UNIX, 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 144 na página 1132 demonstra como usar as amostras Request e Inquire juntas.

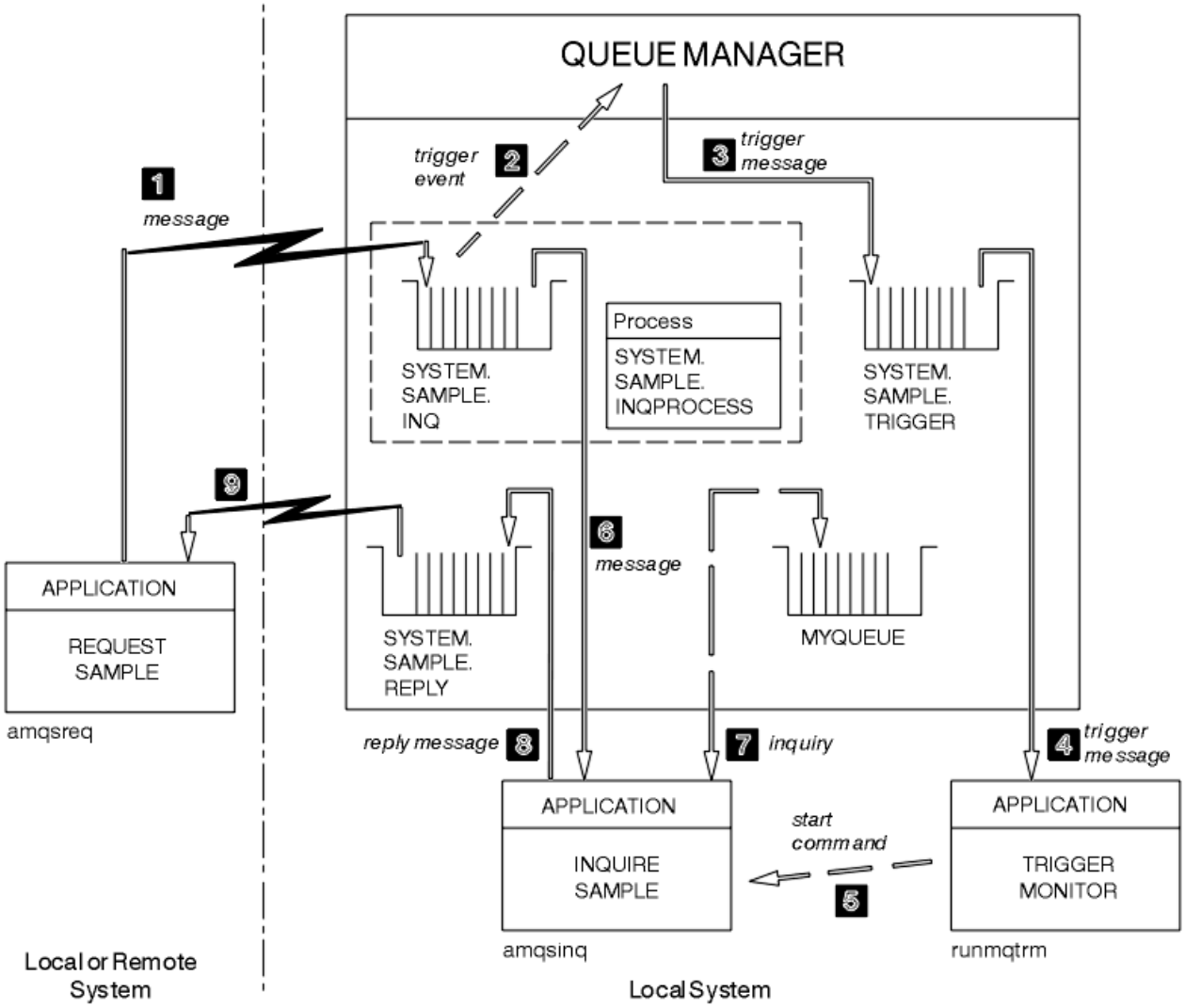


Figura 144. Amostras Request e Inquire usando acionamento

Em Figura 144 na página 1132, 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 144 na página 1132 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 1133.

Para obter mais informações sobre acionamento, consulte "Iniciando aplicativos IBM MQ usando acionadores" na página 863.

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 145 na página 1135. 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.

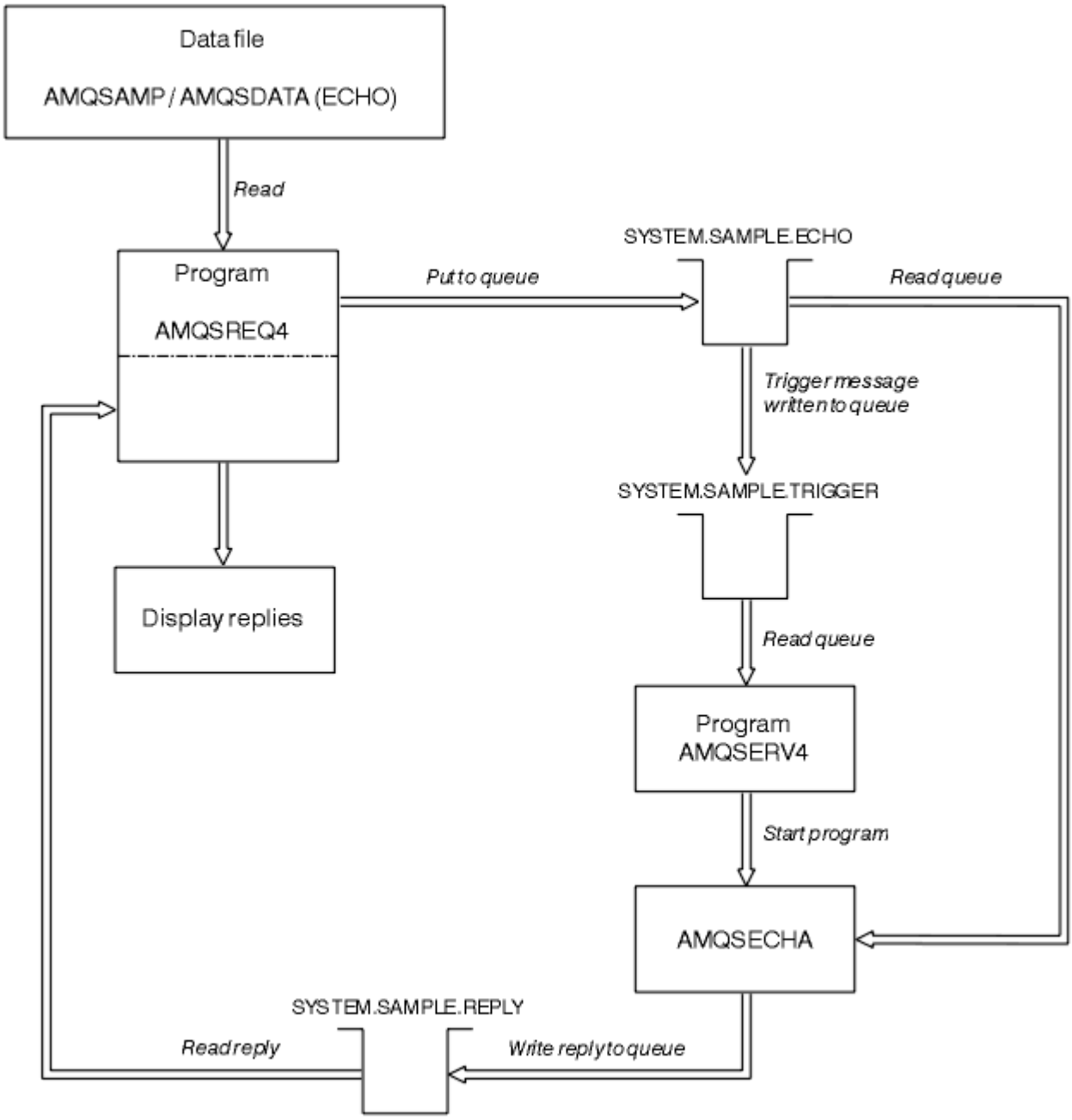


Figura 145. Fluxograma do programa IBM i Cliente/Server (Echo) de amostra

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 1078 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:

- ULW Para sistemas UNIX, 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

AMQSSLC é um programa C de amostras 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. Em seguida, ele simplesmente se conecta a um gerenciador de filas local e ignora as informações de conexão do cliente

AMQSSLC aceita os seguintes parâmetros, 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 KeyReposStem
O nome do stem do arquivo de repositório de chaves. Este é o caminho completo para o arquivo sem o sufixo .kdb. Por exemplo:

```
/home/user/client
C:\User\client
```

-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. Estes valores possuem 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.

Parâmetro de revogação de certificado do OCSP:

-o URL
A URL do respondente de OCSP

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 UNIX, Linux, and Windows.
3. Crie um repositório de chaves para o cliente. Chame-o de *clientkey.kdb*.
4. Crie um certificado pessoal para o gerenciador de filas. Para obter mais informações, consulte Criando um certificado pessoal autoassinado no UNIX, 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 UNIX, Linux, and Windows e Incluindo um certificado de autoridade de certificação (ou a parte pública de um certificado autoassinado) em um repositório de chaves, nos sistemas UNIX, Linux ou 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_SHA)
   ```

   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" -s TLS_RSA_WITH_AES_128_CBC_SHA
    -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_SHA
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 1078 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, UNIX, 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).

- ULW Nos sistemas UNIX, 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.

Windows UNIX *Usando as amostras do TUXEDO no UNIX e 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.

Windows UNIX *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 é construído e execute todos os comandos nesse 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 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 é construído e execute todos os comandos nesse 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.

HP-UX *Construindo o ambiente do servidor para HP-UX (32 bits)*

Como construir o ambiente do servidor para IBM MQ for HP-UX (32 bits).

Sobre esta tarefa

Nota: O ambiente do servidor TUXEDO de 32 bits pode ser construído somente na plataforma Itanium.

Procedimento

1. Crie um diretório (por exemplo, APPDIR) no qual o ambiente do servidor é construído e execute todos os comandos nesse diretório.
2. Exporte as variáveis de ambiente a seguir, em que TUXDIR é o diretório raiz para TUXEDO:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
```

```
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin: MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Inclua a linha a seguir no arquivo TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.so MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

4. Execute os comandos:

```
$ mkfldhdr MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc    MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

   Após a execução dos comandos mkfldhdr e viewc, o arquivo de cabeçalho amqstxvx.h é criado no diretório do aplicativo TUXEDO. Copie esse arquivo do diretório application do TUXEDO para o diretório include do TUXEDO e, em seguida, execute os comandos a seguir.

```
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -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.so \
       -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.so
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. Edite ubbstxcx.cfg e inclua detalhes do nome da máquina, de diretórios ativos e do 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.

HP-UX *Construindo o ambiente do servidor para HP-UX (64 bits)*

Como construir o ambiente do servidor para IBM MQ for HP-UX (64 bits).

Sobre esta tarefa

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Procedimento

1. Crie um diretório (por exemplo, APPDIR) no qual o ambiente do servidor é construído e execute todos os comandos nesse diretório.
2. Exporte as variáveis de ambiente a seguir, em que TUXDIR é o diretório raiz para TUXEDO:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin: MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib64:/lib64
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Na plataforma HP-UX IA64 (IPF), inclua a linha a seguir no arquivo TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

   Nota: As bibliotecas do IBM MQ enviadas na plataforma HP-UX IA64 (IPF) têm uma extensão de nome de arquivo .so.

4. Execute os comandos:

```
$ mkfldhdr MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc    MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

   Após a execução dos comandos mkfldhdr e viewc, o arquivo de cabeçalho amqstxvx.h é criado no diretório do aplicativo TUXEDO. Copie esse arquivo do diretório application do TUXEDO para o diretório include do TUXEDO e, em seguida, execute os comandos a seguir.

```
$ buildtms  -o MQXA -r MQSERIES_XA_RMI
```

   Na plataforma HP-UX IA64 (IPF):

```
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -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.so \
       -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.so
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

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.

Solaris *Construindo o ambiente do servidor para Solaris (32 bits)*

Como construir o ambiente do servidor para IBM MQ for Solaris (32 bits).

Sobre esta tarefa

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Procedimento

1. Crie um diretório (por exemplo, APPDIR) no qual o ambiente do servidor é construído e execute todos os comandos nesse diretório.
2. Exporte as variáveis de ambiente a seguir, em que TUXDIR é o diretório raiz para TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. Inclua o seguinte no arquivo TUXEDO udataobj/RM (RM deve incluir *MQ_INSTALLATION_PATH*/lib/libmqmcs e *MQ_INSTALLATION_PATH*/lib/libmqmzse).

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a MQ_INSTALLATION_PATH/lib/libmqmcs.so \
MQ_INSTALLATION_PATH/lib/libmqmzse.so
```

4. Execute os comandos:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
```

```
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
```

5. Edite ubbstxcx.cfg e inclua detalhes do nome da máquina, de diretórios ativos e do gerenciador de filas, conforme necessário:

```
$ tmloadcf  -y 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.

Solaris *Construindo o ambiente do servidor para Solaris (64 bits)*

Como construir o ambiente do servidor para IBM MQ for Solaris (64 bits).

Sobre esta tarefa

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Procedimento

1. Crie um diretório (por exemplo, APPDIR) no qual o ambiente do servidor é construído e execute todos os comandos nesse diretório.
2. Exporte as variáveis de ambiente a seguir, em que TUXDIR é o diretório raiz para TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib64
$ export LD_LIBRARY_PATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Inclua o seguinte no arquivo TUXEDO udataobj/RM (RM deve incluir *MQ_INSTALLATION_PATH*/lib/libmqmcs e *MQ_INSTALLATION_PATH*/lib/libmqmzse).

```
MQSERIES_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a MQ_INSTALLATION_PATH/lib64/libmqmcs.so \
MQ_INSTALLATION_PATH/lib64/libmqmzse.so
```

4. Execute os comandos:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
```

5. Edite `ubbstxcx.cfg` e inclua detalhes do nome da máquina, de diretórios ativos e do gerenciador de filas, conforme necessário:

```
$ tmloadcf  -y 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 153. Campos a serem mudados para caminhos de diretório	
Campo	**Caminho do 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 TUXEDO foi instalado, por exemplo `f:\tuxedo`.
APPDIR	O caminho do diretório a ser usado 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 146. 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 146 na página 1149. Ele é fornecido como `ubbstxcn.cfg` no diretório de amostras IBM MQ.

O makefile de amostra (consulte Figura 147 na página 1150) fornecido para IBM MQ for Windows é chamado `ubbstxmn.mak` e é mantido no diretório de amostras do 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 147. 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 146 na página 1149) para incluir detalhes sobre o nome da máquina e o gerenciador de filas ao qual 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 154. Campos a serem mudados para caminhos de diretório

Campo	**Caminho do 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 TUXEDO foi instalado, por exemplo `f:\tuxedo`.
APPDIR	O caminho do diretório a ser usado 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 148. 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 148 na página 1152. Ele é fornecido como `ubbstxcn.cfg` no diretório de amostras IBM MQ.

O makefile de amostra (consulte Figura 149 na página 1153) 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 149. 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 148 na página 1152) para incluir detalhes sobre o nome da máquina e o gerenciador de filas ao qual 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.

Windows UNIX *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.

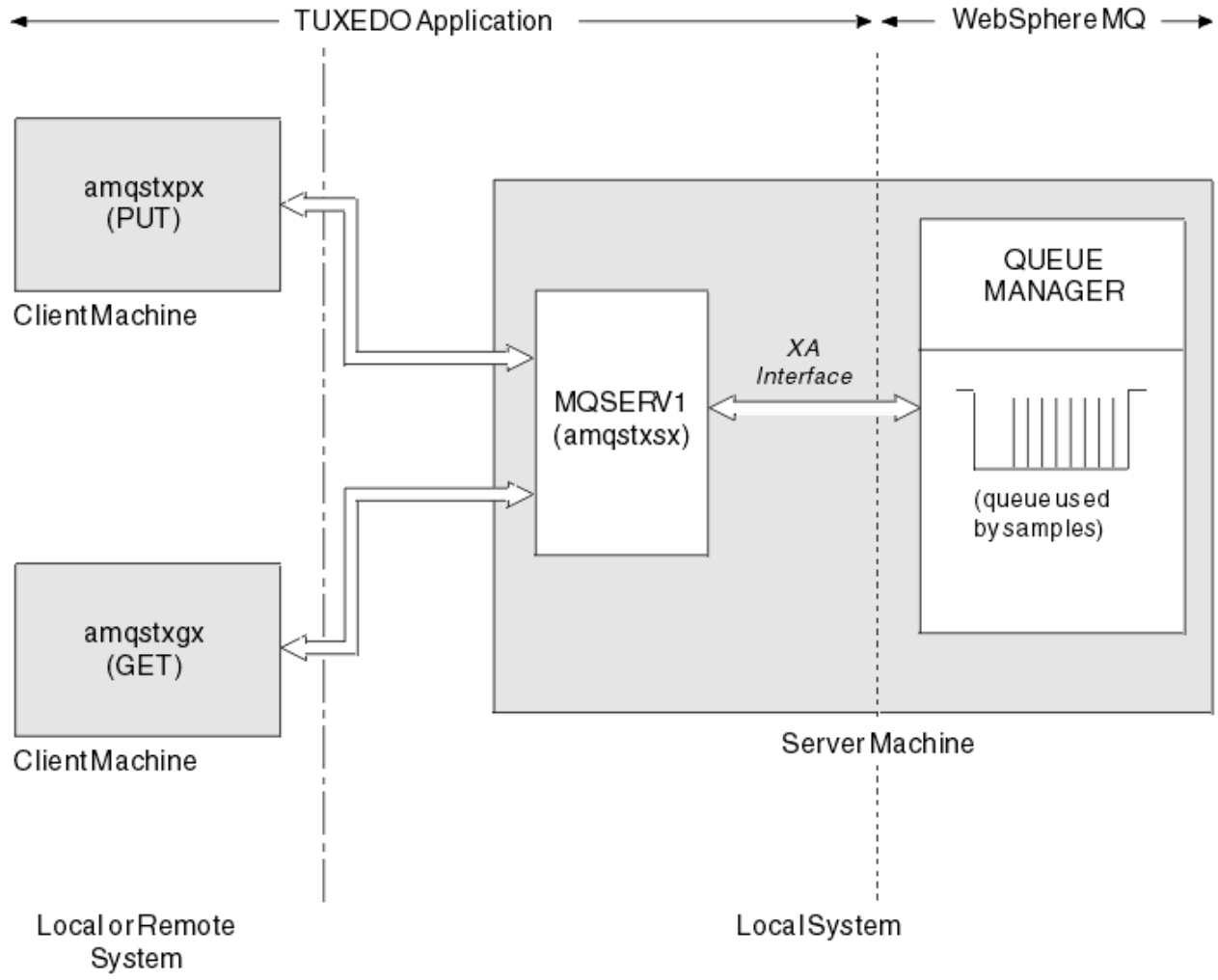


Figura 150. Como as amostras do TUXEDO funcionam juntas

Windows UNIX *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.

Windows UNIX *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 de 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`, 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 de cluster **CLWLPRTY** de cada instância de fila para determinar a 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[10]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) Registre as informações de relatório em `standard out`, além de nos 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.*
```

[10] 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 é preenchido, um novo arquivo RPT01 é criado. O antigo arquivo RPT01 é renomeado para RPT02. RPT02 é renomeado para RPT03. O antigo RPT03 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.

ULW *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.

Tivoli Directory Server 6.3 O cliente deve ser instalado para usar CEPL.

Um conhecimento de trabalho de administração do IBM MQ nas plataformas suportadas é necessário para usar essa amostra.

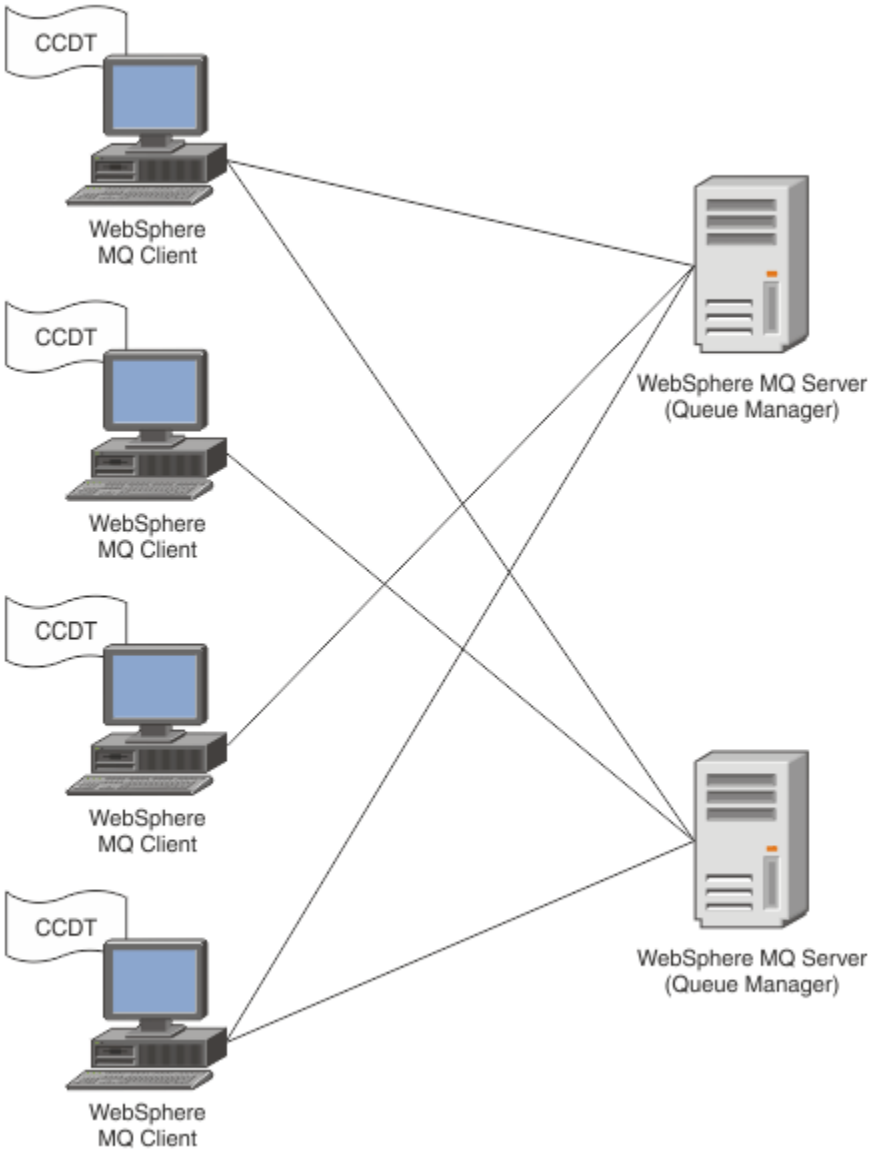
Windows Solaris Linux AIX *Apresentação*

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.

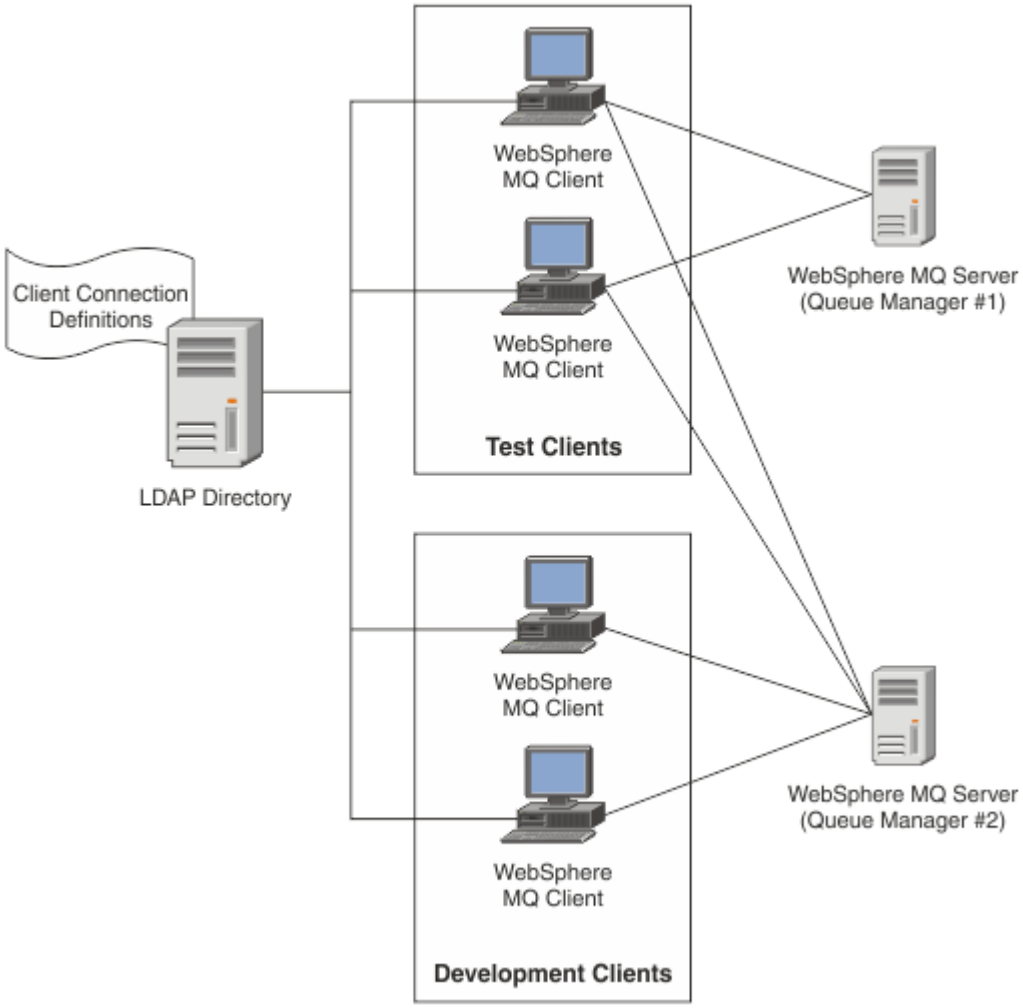


O arquivo da CCDT deve ser distribuído para cada cliente IBM MQ. Quando milhares de clientes podem existir local ou globalmente, logo se tornaria difícil manter e administrar. Uma abordagem mais flexível é necessária para ajudar a assegurar que cada cliente tenha as definições de cliente corretas disponíveis 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.

ULW *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 6.3 Client ou posterior

Sistemas operacionais suportados:

1. Windows (7/8/2008/2012)
2. Solaris (SPARC e x86-64)
3. AIX
4. Linux
   - RHEL v4 e v5 no System p
   - SUSE v9 e v10 no System p
   - RHEL v4 e v5 do System x de 32 bits e 64 bits
   - SUSE v9 e v10 System x de 32 bits e 64 bits
5. HP-UX HP IA64.

Nota: A amostra não está disponível para as plataformas a seguir:

- z/OS z/OS
- IBM i IBM i
- HP-UX HP PA-RISC

ULW *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 em `tools/samples/c/preconnexit/bin`. Para plataformas de 32 bits, o módulo de saída deve ser copiado para `exit/`*`installation_name`*`/` para que 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.

- Assegure que o IBM Tivoli Directory Server esteja em execução, em seguida, copie ou faça FTP do 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 de 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 será procurada.

Função: o nome do ponto de entrada funcional na biblioteca que contém o código de saída PreConnect. A definição de função segue o protótipo MQ_PRECONNECT_EXIT.

Dados: URI do repositório LDAP que contém definições de canal.

O fragmento a seguir é um exemplo das mudanças necessárias no arquivo `mqclient.ini`.

```
PreConnect:
Module=amqlcelp
Function=PreConnectExit
Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ULW *Visão geral da saída e do esquema*

Sintaxe e parâmetros usados para estabelecer uma conexão com um gerenciador de filas.

O IBM MQ 9.0 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, este parâmetro é a sequência de filtros fornecida para a chamada de API do 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.

ULW *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 Solaris 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, Linux, Solaris e Windows.

Compilando a origem

É possível compilar a origem com quaisquer bibliotecas do cliente LDAP, por exemplo, IBM Tivoli Directory Server 6.3 Bibliotecas do cliente Esta documentação supõe que você esteja usando bibliotecas do cliente Tivoli Directory Server 6.3 .

Nota: A biblioteca de saída preconnect é suportada com os servidores LDAP a seguir:

- IBM Tivoli Directory Server 6.3
- Novell eDirectory 8.2

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 6.3 que são compiladas com o compilador Microsoft Visual Studio 2003 , poderá obter avisos enquanto estiver compilando o IBM Tivoli Directory Server 6.3 bibliotecas do cliente com o compilador Microsoft Visual Studio 2012ou posterior..

Solaris Linux AIX **Compilando a saída em AIX, Linux ou Solaris**

O fragmento de código a seguir é para compilar a origem de saída no Linux. Algumas opções do compilador podem diferir no AIX ou Solaris.

```
#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)
```

ULW *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 MQNXP é preenchido com o atributo Data da sub-rotina PreConnect no 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.

Neste ponto, a saída se conecta e 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 **QMgrNameparameter** contiver um nome de gerenciador de filas específico, a procura retorna todas as definições de canal para as quais o valor do atributo LDAP **ibm-amqQueueManagerName** corresponde ao nome do gerenciador de filas fornecido.

Se o parâmetro **QMgrName** for '*' ou ' ' (em branco), então, a procura retorna todas as definições de canal para as quais o atributo do endpoint **ibm-amqIsClientDefault Connection** estiver configurado para 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** não são usados durante uma saída com o código de razão MQXR_TERM e podem ser NULL.

Informações relacionadas

Sub-rotina PreConnect do arquivo de configuração do cliente

ULW *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.

ULW *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 155. Atributos da sequência do diretório do canal do cliente IBM MQ

Atributo LDAP	Descrição	Propriedade do 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.	SENHA
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 155. Atributos da sequência do diretório do canal do cliente IBM MQ (continuação)		
Atributo LDAP	**Descrição**	**Propriedade do 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 156. Atributos de número inteiro do diretório do canal do cliente IBM MQ		
Atributo LDAP	**Descrição**	**Propriedade do 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 157. 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 158. Atributos da lista de canal do cliente IBM MQ		
Atributo LDAP	**Descrição**	**Propriedade do 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

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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 1179
Esse atributo booleano soluciona o problema de procura de entradas quando o atributo ibm-amqQueueManagerName não tiver sido definido.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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.

ULW *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 1177
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.

ULW *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.

ULW *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.

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, que é mapeado para a sequência de compactação subjacente RLE, ZLIBFAST e ZLIBHIGH.

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.

ULW *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.

ULW *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.

ULW *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.

ULW *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 Usando os programas de amostra para z/OS

Os aplicativos processuais de amostra que são entregues com o IBM MQ for z/OS demonstram usos típicos do Message Queue Interface (MQI).

Sobre esta tarefa

O IBM MQ for z/OS também fornece saídas de conversão de dados de amostra, descritas em "Escrevendo saídas de conversão de dados" na página 988.

Todos os aplicativos de amostra são fornecidos no formato de origem; diversos também são fornecidos no formato executável. Os módulos de origem incluem o pseudocódigo que descreve a lógica do programa.

Nota: Embora alguns dos aplicativos de amostra possuam interfaces orientadas ao painel básico, eles não pretendem demonstrar como projetar a aparência dos seus aplicativos. Para obter mais informações sobre como projetar interfaces orientadas ao painel para terminais não programáveis, veja o *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) e seu anexo (GG22-9508). Eles fornecem diretrizes para ajudar a projetar aplicativos consistentes dentro do aplicativo e entre outros aplicativos.

Procedimento

- Use os seguintes links para descobrir mais sobre os programas de amostra:
  - "Recursos demonstrados nos aplicativos de amostra para z/OS" na página 1182
  - "Preparando e executando aplicativos de amostra para o ambiente de lote no z/OS" na página 1189
  - "Preparando os aplicativos de amostra para o ambiente TSO no z/OS" na página 1192
  - "Preparando os aplicativos de amostra para o ambiente do CICS no z/OS" na página 1194
  - "Preparando o aplicativo de amostra para o ambiente do IMS no z/OS" na página 1198
  - "As amostras de Put no z/OS" na página 1199
  - "As amostras de Get no z/OS" na página 1201
  - "A amostra de procura no z/OS" na página 1204
  - "A amostra de impressão de mensagem no z/OS" na página 1206
  - "A amostra de atributos da fila no z/OS" na página 1210
  - "A amostra de gerenciador de e-mail no z/OS" na página 1211
  - "A amostra de verificação de crédito no z/OS" na página 1218
  - "A amostra Manipulador de mensagem no z/OS" na página 1230
  - "A amostra de postagem assíncrona no z/OS" na página 1234
  - "A amostra de consumo assíncrono em lote no z/OS" na página 1235
  - "A amostra de consumo assíncrono e publicar/assinar do CICS no z/OS" na página 1236
  - "A amostra de publicar/assinar no z/OS" na página 1239
  - "A amostra de propriedade de mensagem Set and Inquire no z/OS" na página 1241

Tarefas relacionadas

"Usando os programas de amostra em multiplataformas" na página 1076
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 *Recursos demonstrados nos aplicativos de amostra para z/OS*

Esta seção resume os recursos do MQI demonstrados em cada um dos aplicativos de amostra, mostra as linguagens de programação em que cada amostra é escrita e o ambiente no qual cada amostra é executada.

z/OS *Amostras de Put no z/OS*

As amostras PUT demonstram como colocar mensagens em uma fila usando a chamada MQPUT.

O aplicativo usa essas chamadas MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

O programa é entregue em COBOL e C e é executado no lote e o ambiente do CICS. Consulte Tabela 161 na página 1190 para o aplicativo em lote e Tabela 168 na página 1195 para o aplicativo CICS.

z/OS *Amostras de Get no z/OS*

As amostras Get demonstram como obter mensagens de uma fila usando a chamada de MQGET.

O aplicativo usa essas chamadas MQI:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

O programa é entregue em COBOL e C e é executado no lote e o ambiente do CICS. Consulte Tabela 161 na página 1190 para o aplicativo em lote e Tabela 168 na página 1195 para o aplicativo CICS.

z/OS *Amostras de procura no z/OS*

A Procurar de amostra demonstra como usar a opção Navegar para localizar uma mensagem, imprimi-lo, em seguida, percorrer as mensagens em uma fila.

O aplicativo usa essas chamadas MQI:

- MQCONN
- MQOPEN
- MQGET para procurar mensagens
- MQCLOSE
- MQDISC

O programa é entregue nas linguagens COBOL, assembler, PL/I e C. O aplicativo é executado no ambiente de lote. Veja Tabela 162 na página 1190 para o aplicativo em lote.

z/OS *Amostra de impressão de mensagem no z/OS*

A amostra de Imprimir Mensagem demonstra como remover uma mensagem de uma fila e imprimir os dados na mensagem, junto com todos os campos de seu descritor de mensagens. Ele pode, opcionalmente, exibir todas as propriedades da mensagem associadas a cada mensagem.

Ao remover caracteres de comentário de duas linhas no módulo de origem, é possível mudar o programa de forma que ele procure, em vez de remover, as mensagens em uma fila. Este programa pode utilmente ser utilizado para diagnosticar problemas com um aplicativo que está colocando mensagens em uma fila.

O aplicativo usa essas chamadas MQI:

- MQCONN
- MQOPEN
- MQGET para remover mensagens de uma fila (com uma opção para navegar)

- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

O programa é entregue na linguagem C. O aplicativo é executado no ambiente de lote. Veja Tabela 163 na página 1191 para o aplicativo em lote.

z/OS *Amostra de atributos da fila no z/OS*

A amostra de Atributos de Fila demonstra como consultar e configurar os valores de atributos de objetos IBM MQ for z/OS.

O aplicativo usa essas chamadas MQI:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

O programa é entregue nas linguagens COBOL, assembler e C. O aplicativo é executado no ambiente do CICS. Veja Tabela 169 na página 1196 para o aplicativo CICS.

z/OS *Amostra de gerenciador de e-mail no z/OS*

Considerações que devem ser observadas ao usar a amostra do Gerenciador de correio.

A amostra do Gerenciador de correio demonstra essas técnicas:

- Usando filas de alias
- Usando uma fila modelo para criar uma fila dinâmica temporária
- Usando filas de resposta
- Uso de pontos de sincronização no CICS e ambientes em lote
- Envio de comandos para a fila de entrada de comandos do sistema
- Testando códigos de retorno
- Envio de mensagens a gerenciadores de filas remotas, usando uma definição local de uma fila remota e colocando mensagens diretamente em uma fila nomeada em um gerenciador de filas remotas

O aplicativo usa essas chamadas MQI:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Três versões do aplicativo são fornecidas:

- Um aplicativo CICS gravado em COBOL
- Um aplicativo TSO gravado em COBOL
- Um aplicativo TSO gravado em C

Os aplicativos TSO usam o adaptador IBM MQ for z/OS em lote e incluem alguns painéis do ISPF.

Consulte Tabela 166 na página 1193 para o aplicativo TSO e Tabela 170 na página 1196 para o aplicativo CICS.

z/OS *Amostra de verificação de crédito no z/OS*

Essas informações contêm os pontos a serem considerados ao usar a amostra Credit Check.

A amostra Credit Check é um conjunto de programas que demonstra essas técnicas:

- Desenvolvendo um aplicativo que é executado em mais de um ambiente
- Usando uma fila modelo para criar uma fila dinâmica temporária
- Usando um identificador de correlação
- Configurando e transmitindo informações de contexto
- Usando persistência e prioridade da mensagem
- Iniciando programas usando o acionamento
- Usando filas de resposta
- Usando filas de alias
- Usando uma fila de mensagens não entregues
- Usando uma lista de nomes
- Testando códigos de retorno

O aplicativo usa essas chamadas MQI:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET para procurar e obter mensagens, usando as opções espera e sinal, e para obter uma mensagem específica
- MQINQ
- MQSET
- MQCLOSE

A amostra pode ser executada como um aplicativo do CICS independente. No entanto, para demonstrar como projetar um aplicativo de enfileiramento de mensagens que usa os recursos fornecidos por ambos os ambientes CICS e IMS, um módulo também é fornecido como um programa de processamento de mensagens em lote do IMS.

Os programas CICS são entregues em C e COBOL. O único programa IMS é entregue em C.

Consulte Tabela 171 na página 1197 para o aplicativo CICS e Tabela 173 na página 1199 para o aplicativo IMS.

z/OS *A amostra Manipulador de mensagem no z/OS*

A amostra Manipulador de Mensagens permite que você procure, encaminhe e exclua mensagens em uma fila.

O aplicativo usa essas chamadas MQI:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET

- MQCLOSE
- MQDISC

O programa é entregue em linguagens de programação C e COBOL. O aplicativo é executado sob TSO. VejaTabela 167 na página 1194para o aplicativo TSO.

z/OS *Amostras de saída de enfileiramento distribuído no z/OS*

Uma tabela de programas de origem de amostras de saída de enfileiramento distribuído.

Os nomes dos programas de origem das amostras de saída de enfileiramento distribuído são listados na tabela a seguir:

Tabela 159. Origem para o enfileiramento distribuído amostras de saída

Nome membro	Para idioma	Descrição	Fornecido na biblioteca
CSQ4BAX0	Assembler	Programa de origem	SCSQASMS
CSQ4BCX1	C	Programa de origem	SCSQC37S
CSQ4BCX2	C	Programa de origem	SCSQC37S
CSQ4BCX4	C	Programa de origem	SCSQC37S

Nota: programas de origem são, link-edit com CSQXSTUB.

z/OS *Amostras de saída de conversão de dados no z/OS*

Uma estrutura é fornecida para uma rotina de saída de conversão de dados, e uma amostra é enviada com o IBM MQ ilustrando a chamada MQXCNVC.

Os nomes dos programas de origem da conversão de dados de saída de amostras estão listados na tabela a seguir:

Tabela 160. A fonte para as amostras de saída de conversão de dados (apenas linguagem assembler)

Nome do membro	Descrição	Fornecido na biblioteca
CSQ4BAX8	Programa de origem	SCSQASMS
CSQ4BAX9	Programa de origem	SCSQASMS
CSQ4CAX9	Programa de origem	SCSQASMS

Nota: Os programas de origem são editados por link com CSQASTUB.

Consulte o “Escrevendo saídas de conversão de dados” na página 988 para obter informações adicionais.

z/OS *Amostras de publicar/assinar no z/OS*

Os programas de amostra Publish/Subscribe demonstram o uso dos recursos de publicação e assinatura no IBM MQ.

Há quatro programas de amostra de linguagem de programação C e dois COBOL que demonstram como programar a interface do IBM MQ Publish/Subscribe.

Os aplicativos usam estas chamadas de MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE

- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

Os programas de amostra Publish/Subscribe são entregues nas linguagens de programação C e COBOL. Os aplicativos de amostra são executados no ambiente de lote. Veja Amostras de publicação/assinatura para os aplicativos em lote.

z/OS *Configurando um gerenciador de filas para aceitar conexões do cliente no z/OS*

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 dos registros de autenticação de canal, então, 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 1187.

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 é a autoridade sob a qual a conexão do cliente será executada no gerenciador de filas.
2. Inicie um programa listener.
   a) Assegure-se de que seu inicializador de canais esteja iniciado. Se não estiver, inicie-o emitindo o comando **START CHINIT**.
   b) Inicie o programa listener emitindo o comando a seguir:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   em que *nnnn* é o número da porta escolhido.
3. Se seu aplicativo usar SYSTEM.DEF.SVRCONN, então, esse canal já está definido. Se seu aplicativo usar outro canal, crie-o emitindo o comando do MQSC:

```
DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Channel for use by sample programs')
```

channel-name é o nome de seu canal.

4. Crie uma regra de autenticação de canal permitindo que somente o endereço IP do seu sistema cliente use o canal emitindo o comando do MQSC:

```
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 1187

5. Se seu aplicativo usar o SYSTEM.DEFAULT.LOCAL.QUEUE, então essa fila já estará definida. Se seu aplicativo usar outra fila, crie-a emitindo o comando do 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 se conectar e consultar o gerenciador de filas:
   a) Assegure-se de que seu inicializador de canais esteja iniciado. Se não estiver, inicie o inicializador de canais emitindo o comando START CHINIT.
   b) Inicie um listener TCP, por exemplo, emita o comando a seguir:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

em que *nnnn* é o número da porta escolhido.

7. Se seu aplicativo for de ponto a ponto, ou seja, usa filas, conceda acesso para permitir a consulta e a colocação e obtenção de mensagens usando sua fila pelo ID do usuário a ser usado, emitindo os comandos do MQSC:

Emita os comandos do RACF:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

em que

qmgr-name é o nome do seu gerenciador de filas

queue-name é o nome de sua fila.

non-privileged-user-id é o ID do usuário obtido na etapa “1” na página 1187

8. Se seu aplicativo for um aplicativo publicar/assinar, ou seja, usar tópicos, conceda acesso para permitir publicação e assinatura usando seu tópico pelo ID do usuário a ser usado, emitindo os comandos do RACF a seguir:

```
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)
```

em que

qmgr-name é o nome do seu gerenciador de filas

non-privileged-user-id é o ID do usuário obtido na etapa “1” na página 1187

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. Para obter mais informações, veja Controlando o acesso de usuário a tópicos.

Como proceder a seguir

Agora, seu aplicativo cliente pode se conectar ao gerenciador de filas e colocar ou obter mensagens usando a fila.

Informações relacionadas

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

z/OS Autoridade para trabalhar com objetos do IBM MQ no z/OS

z/OS *Preparando e executando aplicativos de amostra para o ambiente de lote no z/OS*

Para preparar um aplicativo de amostra que é executado no ambiente em lote, execute as mesmas etapas que você faria ao construir qualquer aplicativo IBM MQ for z/OS de lote.

Essas etapas são listadas em “Criando aplicativos em lote do z/OS” na página 1041.

Como alternativa, quando fornecemos um formato executável de uma amostra, é possível executá-la a partir da biblioteca de carregamento thlqual.SCSQLOAD.

Nota: A versão da linguagem assembler da amostra Browse usa blocos de controle de dados (DCBs), portanto, deve-se editá-la por link usando RMODE(24).

Os membros da biblioteca para uso são listados em Tabela 161 na página 1190, Tabela 162 na página 1190, Tabela 163 na página 1191 e Tabela 164 na página 1191.

Deve-se editar a JCL de execução fornecida para as amostras que deseja usar (consulte Tabela 161 na página 1190, Tabela 162 na página 1190, Tabela 163 na página 1191 e Tabela 164 na página 1191).

A instrução PARM na JCL fornecida contém diversos parâmetros que você precisa modificar. Para executar os programas de amostra C, separe os parâmetros por espaços; para executar os programas de amostra em assembler, COBOL e PL/I, separe-os por vírgulas. Por exemplo, se o nome de seu gerenciador de filas for CSQ1 e você desejar executar o aplicativo com uma fila denominada LOCALQ1, na JCL de linguagem COBOL, PL/I e assembler, sua instrução PARM deve ser semelhante a esta:

```
PARM=(CSQ1,LOCALQ1)
```

Na JCL de linguagem C, sua instrução PARM deve ser semelhante a esta:

```
PARM=('CSQ1 LOCALQ1')
```

Agora, você está pronto para enviar as tarefas.

z/OS *Nomes dos aplicativos em lote de amostra no z/OS*

Um resumo dos programas que são fornecidos para aplicativos em lote de amostra.

Os programas de aplicativo de batch são resumidas nas seguintes tabelas:

- Amostras de put e get do Tabela 161 na página 1190
- Tabela 162 na página 1190 Amostra Browse
- Tabela 163 na página 1191 Amostra Print message
- Tabela 164 na página 1191 Amostras Publish/Subscribe
- Tabela 165 na página 1192 Outras amostras

Tabela 161. Lote Colocação e Obtenção de amostras

Nome do membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca	Arquivo executável fornecido na biblioteca
CSQ4BCJ1	C	Obtenha programa de origem	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Coloque programa de origem	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	JCL de execução de amostra para CSQ4BCJ1 e CSQBCK1	SCSQPROC	Nenhum
CSQ4BVJ1	COBOL	Obtenha programa de origem	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Coloque programa de origem	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	JCL de execução de amostra para CSQBVJ1 e CSQBVK1	SCSQPROC	Nenhum

Tabela 162. Procurar de amostra em lote

Nome do membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca	Arquivo executável fornecido na biblioteca
CSQ4BVA1	COBOL	Programa de origem	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	JCL de execução de amostra para CSQ4BVA1	SCSQPROC	Nenhum
CSQ4BAA1	Assembler	Programa de origem	SCSQASMS	SCSQLOAD
CSQ4BAAR	Assembler	JCL de execução de amostra para CSQ4BAA1	SCSQPROC	Nenhum
CSQ4BCA1	C	Programa de origem	SCSQC37S	SCSQLOAD

Tabela 162. Procurar de amostra em lote (continuação)

Nome do membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca	Arquivo executável fornecido na biblioteca
CSQ4BCAR	C	JCL de execução de amostra para CSQ4BCA1	SCSQPROC	Nenhum
CSQ4BPA1	PL/I	Programa de origem	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	JCL de execução de amostra para CSQ4BPA1	SCSQPROC	Nenhum

Tabela 163. amostra de impressão em Batch (apenas linguagem C)

Nome do membro	Descrição	Arquivo de origem fornecido na biblioteca	Arquivo executável fornecido na biblioteca
CSQ4BCG1	Programa de origem	SCSQC37S	SCSQLOAD
CSQ4BCGR	JCL de execução de amostra para CSQ4BCG1	SCSQPROC	Nenhum
CSQ4BCL1	programa de origem Procurar	SCSQC37S	SCSQLOAD
CSQ4BCLR	JCL de execução de amostra para CSQ4BCL1	SCSQPROC	Nenhum

Tabela 164. Amostras Publish/Subscribe

Nome do membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca	JCL em SCSQPROC	Arquivo executável fornecido na biblioteca
CSQ4BCP1	C	programa de origem de tópico para publicação	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	tópico para Assinar e o programa de origem mensagens	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	tópico para Subscrever usando um destino fornecido pelo usuário e obter mensagens de origem do programa	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	Subscrever usando um tópico para opções estendidas e obter o programa de origem mensagens	SCSQC37S	CSQ4BCPE	SCSQLOAD

Tabela 164. Amostras Publish/Subscribe (continuação)					
Nome do membro	**Para idioma**	**Descrição**	**Arquivo de origem fornecido na biblioteca**	**JCL em SCSQPROC**	**Arquivo executável fornecido na biblioteca**
CSQ4BVP1	COBOL	programa de origem de tópico para publicação	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	tópico para Assinar e o programa de origem mensagens	SCSQCOBS	CSQ4BVPS	SCSQLOAD

Tabela 165. Outras Amostras					
Nome do membro	**Para idioma**	**Descrição**	**Arquivo de origem fornecido na biblioteca**	**JCL em SCSQPROC**	**Arquivo executável fornecido na biblioteca**
CSQ4BCS1	C	programa de origem consumo assíncrono	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	Coloque Assíncrono e Verificar programa de origem status	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	Consultar as propriedades de mensagem de origem do programa	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	Configure programa de origem as propriedades de mensagem	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS *Preparando os aplicativos de amostra para o ambiente TSO no z/OS*

Para preparar um aplicativo de amostra que é executado no ambiente TSO, execute as mesmas etapas que você faria ao construir qualquer aplicativo IBM MQ for z/OS em lote.

Essas etapas são listadas em "Criando aplicativos em lote do z/OS" na página 1041. Os membros da biblioteca para uso são listados em Tabela 166 na página 1193.

Como alternativa, quando fornecemos um formato executável de uma amostra, é possível executá-la a partir da biblioteca de carregamento thlqual.SCSQLOAD.

Para o aplicativo de amostra Mail Manager, assegure-se de que as filas que ele usa estejam disponíveis em seu sistema. Elas são definidas no membro **thlqual**.SCSQPROC(CSQ4CVD). Para se assegurar de que essas filas sempre estejam disponíveis, você poderia incluir esses membros em seu conjunto de dados de entrada de inicialização CSQINP2 ou usar o programa CSQUTIL para carregar essas definições de filas.

z/OS *Nomes dos aplicativos de amostra do TSO no z/OS*

Informações sobre os nomes dos programas que são fornecidos para cada um dos aplicativos de amostra do TSO e as bibliotecas nas quais a origem, o JCL e, somente para a amostra Message Handler, os arquivos executáveis residem.

Os programas de aplicativo do TSO são resumidos nas tabelas a seguir:

- Tabela 166 na página 1193 Amostra Mail

- Tabela 167 na página 1194 Amostra Message Handler

Essas amostras usam painéis do ISPF. Deve-se, portanto, incluir o stub do ISPF, ISPLINK, quando você editar o link do programas.

Tabela 166. Amostra TSO Mail Manager

Nome do membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca
CSQ4CVD	independente	Definições de objeto do IBM MQ for z/OS	SCSQPROC
CSQ40	independente	Mensagens ISPF	SCSQMSGE
CSQ4RVD1	COBOL	CSQ4TVD1 CLIST para iniciar	SCSQCLST
CSQ4TVD1	COBOL	programa de origem para o programa Menu	SCSQCOBS
CSQ4TVD2	COBOL	Obter Fonte de programa para programa de correio	SCSQCOBS
CSQ4TVD4	COBOL	programa de origem para o programa Send Mail	SCSQCOBS
CSQ4TVD5	COBOL	programa de origem para o programa Apelido	SCSQCOBS
CSQ4VDP1-6	COBOL	Definições do Painel	SCSQPNLA
CSQ4VD0	COBOL	Definição de dados	SCSQCOBC
CSQ4VD1	COBOL	Definição de dados	SCSQCOBC
CSQ4VD2	COBOL	Definição de dados	SCSQCOBC
CSQ4VD4	COBOL	Definição de dados	SCSQCOBC
CSQ4RCD1	C	CSQ4TCD1 CLIST para iniciar	SCSQCLST
CSQ4TCD1	C	programa de origem para o programa Menu	SCSQC37S
CSQ4TCD2	C	Obter Fonte de programa para programa de correio	SCSQC37S
CSQ4TCD4	C	programa de origem para o programa Send Mail	SCSQC37S
CSQ4TCD5	C	programa de origem para o programa Apelido	SCSQC37S
CSQ4CDP1-6	C	Definições do Painel	SCSQPNLA
CSQ4TC0	C	Arquivo de inclusão	SCSQC370

Tabela 167. Amostra TSO Message Handler				
Nome do membro	**Para idioma**	**Descrição**	**Arquivo de origem fornecido na biblioteca**	**Arquivo executável fornecido na biblioteca**
CSQ4TCH0	C	Definição de dados	SCSQC370	Nenhum
CSQ4TCH1	C	Programa de origem	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	Programa de origem	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	Programa de origem	SCSQC37S	SCSQLOAD
CSQ4RCH1	C e COBOL	CLIST para iniciar CSQ4TCH1 ou CSQ4TVH1	SCSQCLST	Nenhum
CSQ4CHP1	C e COBOL	definição do Painel	SCSQPNLA	Nenhum
CSQ4CHP2	C e COBOL	definição do Painel	SCSQPNLA	Nenhum
CSQ4CHP3	C e COBOL	definição do Painel	SCSQPNLA	Nenhum
CSQ4CHP9	C e COBOL	definição do Painel	SCSQPNLA	Nenhum
CSQ4TVH0	COBOL	Definição de dados	SCSQCOBC	Nenhum
CSQ4TVH1	COBOL	Programa de origem	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	Programa de origem	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	Programa de origem	SCSQCOBS	SCSQLOAD

z/OS *Preparando os aplicativos de amostra para o ambiente do CICS no z/OS*

Antes de executar os programas de amostra do CICS, efetue logon no CICS usando um LOGMODE de 32702. Isso ocorre porque os programas de amostra foram escritos para usar uma tela 3270 de modo 2.

Para preparar um aplicativo de amostra que é executado no ambiente do CICS, execute as etapas a seguir:

1. Crie o mapa de descrição simbólico e o mapa da tela física para a amostra, montando a fonte de definição de tela BMS (fornecida na biblioteca **thlqual**.SCSQMAPS, em que **thlqual** é o qualificador de alto nível usado por sua instalação). Ao nomear os mapas, use o nome da fonte de definição de tela BMS (não disponível para programas de amostra Put e Get), mas omita o último caractere desse nome.
2. Execute as mesmas etapas que você faria ao construir qualquer aplicativo CICS IBM MQ for z/OS. Essas etapas são listadas em "Criando aplicativos CICS no z/OS" na página 1044. Os membros da biblioteca para uso são listados em Tabela 168 na página 1195, Tabela 169 na página 1196, Tabela 170 na página 1196 e Tabela 171 na página 1197.

   Como alternativa, quando fornecemos uma forma executável de uma amostra, é possível executá-la a partir da biblioteca de carregamento thlqual.SCSQCICS.
3. Identifique o conjunto de mapas, programas e transações para CICS atualizando o conjunto de dados de definição de sistema do CICS (CSD). As definições que você precisa estão no membro **thlqual**.SCSQPROC(CSQ4S100). Para obter orientação sobre como fazer isso, consulte *A seção CICS-*

IBM MQ Adaptador na documentação do produto CICS Transaction Server 4.1 para z/OS em: CICS Transaction Server 4.1 para z/OS, O adaptador CICS-IBM MQ.

Nota: Para o aplicativo de amostra de Credit Check, você obtém uma mensagem de erro neste estágio, se você ainda não tiver criado um conjunto de dados VSAM que a amostra usa.

4. Para os aplicativos de amostra Credit Check e Mail Manager, assegure-se de que as filas que eles usam estejam disponíveis em seu sistema. Para a amostra do Credit Check, elas são definidos no membro **thlqual**.SCSQPROC (CSQ4CVB) para COBOL e **thlqual**.SCSQPROC (CSQ4CCB) para C. Para a amostra do Mail Manager, elas são definidos no membro **thlqual**.SCSQPROC (CSQ4CVD). Para se assegurar de que essas filas sempre estejam disponíveis, você poderia incluir esses membros em seu conjunto de dados de entrada de inicialização CSQINP2 ou usar o programa CSQUTIL para carregar essas definições de filas.

   Para o aplicativo de amostra Queue Attributes, você poderia usar uma ou mais das filas que são fornecidas para os outros aplicativos de amostra. Como alternativa, você pode usar suas próprias filas. No entanto, no formulário que é fornecido, essa amostra funciona somente com filas que têm os caracteres CSQ4SAMP nos primeiros oito bytes de seu nome.

z/OS *Nomes dos aplicativos CICS de amostra no z/OS*

Este tópico apresenta fornece um resumo dos programas fornecidos para aplicativos de amostra do CICS.

Os programas de aplicativo CICS estão resumidos nas seguintes tabelas:

- Amostras de put e get do Tabela 168 na página 1195
- Amostra Queue Attributes do Tabela 169 na página 1196
- Amostra Mail Manager do Tabela 170 na página 1196 (somente COBOL)
- Amostra Credit Check do Tabela 171 na página 1197
- Amostras Asynchronous Consumption and Publish/Subscribe do Tabela 172 na página 1198

Tabela 168. Amostras de put e get do CICS

Nome membro	**Para idioma**	**Descrição**	**Arquivo de origem fornecido na biblioteca**	**Arquivo executável fornecido na biblioteca**
CSQ4CCK1	C	Coloque programa de origem	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Obtenha programa de origem	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Obtenha programa de origem	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Coloque programa de origem	SCSQCOBS	SCSQCICS
CSQ4S100	produção independente	Conjunto de dados de definição do sistema do CICS	SCSQPROC	Nenhum

Tabela 169. Amostra Queue Attributes do CICS

Nome membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca	Arquivo executável fornecido na biblioteca
CSQ4CVC1	COBOL	Programa de origem	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Definição de mensagem	SCSQCOBC	Nenhum
CSQ4VCMS	COBOL	definição de tela BMS	SCSQMAPS	SCSQCICS (denominado CSQ4ACM)
CSQ4CAC1	Assembler	Programa de origem	SCSQASMS	SCSQCICS
CSQ4AMSG	Assembler	Definição de mensagem	SCSQMACS	Nenhum
CSQ4ACMS	Assembler	definição de tela BMS	SCSQMAPS	SCSQCICS (denominado CSQ4ACM)
CSQ4CCC1	C	Programa de origem	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Definição de mensagem	SCSQC370	Nenhum
CSQ4CCMS	C	definição de tela BMS	SCSQMAPS	SCSQCICS (denominado CSQ4ACM)
CSQ4S100	produção independente	Conjunto de dados de definição do sistema do CICS	SCSQPROC	Nenhum

Tabela 170. Amostra Mail Manager do CICS (somente COBOL)

Nome membro	Descrição	Arquivo de origem fornecido na biblioteca
CSQ4CVD	Definições de objeto do IBM MQ for z/OS	SCSQPROC
CSQ4CVD1	Origem para o programa Menu	SCSQCOBS
CSQ4CVD2	Correio programa Obter Origem para	SCSQCOBS
CSQ4CVD3	Origem para o programa Exibir Mensagens	SCSQCOBS
CSQ4CVD4	Origem para o programa Sendmail	SCSQCOBS
CSQ4CVD5	Origem para o programa Apelido	SCSQCOBS
CSQ4VDMS	origem da definição BMS de tela	SCSQMAPS

Tabela 170. Amostra Mail Manager do CICS (somente COBOL) (continuação)

Nome membro	Descrição	Arquivo de origem fornecido na biblioteca
CSQ4S100	Conjunto de dados de definição do sistema do CICS	SCSQPROC
CSQ4VD0	Definição de dados	SCSQCOBC
CSQ4VD3	Definição de dados	SCSQCOBC
CSQ4VD4	Definição de dados	SCSQCOBC

Tabela 171. Amostra Credit Check do CICS

Nome membro	Para idioma	Descrição	Arquivo de origem fornecido na biblioteca
CSQ4CVB	produção independente	Definições de objeto do IBM MQ	SCSQPROC
CSQ4CCB	produção independente	Definições de objeto do IBM MQ	SCSQPROC
CSQ4CVB1	COBOL	Origem para a interface do programa	SCSQCOBS
CSQ4CVB2	COBOL	Origem para o gerenciador de crédito	SCSQCOBS
CSQ4CVB3	COBOL	Origem para programa de conta corrente	SCSQCOBS
CSQ4CVB4	COBOL	Origem para programa de distribuição	SCSQCOBS
CSQ4CVB5	COBOL	Origem para o programa agência-consulta	SCSQCOBS
CSQ4CCB1	C	Origem para a interface do programa	SCSQC37S
CSQ4CCB2	C	Origem para o gerenciador de crédito	SCSQC37S
CSQ4CCB3	C	Origem para programa de conta corrente	SCSQC37S
CSQ4CCB4	C	Origem para programa de distribuição	SCSQC37S
CSQ4CCB5	C	Origem para o programa agência-consulta	SCSQC37S
CSQ4CB0	C	Arquivo de inclusão	SCSQC370
CSQ4CBMS	C	origem da definição BMS de tela	SCSQMAPS
CSQ4VBMS	COBOL	origem da definição BMS de tela	SCSQMAPS
CSQ4VB0	COBOL	Definição de dados	SCSQCOBC
CSQ4VB1	COBOL	Definição de dados	SCSQCOBC
CSQ4VB2	COBOL	Definição de dados	SCSQCOBC
CSQ4VB3	COBOL	Definição de dados	SCSQCOBC
CSQ4VB4	COBOL	Definição de dados	SCSQCOBC
CSQ4VB5	COBOL	Definição de dados	SCSQCOBC
CSQ4VB6	COBOL	Definição de dados	SCSQCOBC
CSQ4VB7	COBOL	Definição de dados	SCSQCOBC

Tabela 171. Amostra Credit Check do CICS (continuação)			
Nome membro	**Para idioma**	**Descrição**	**Arquivo de origem fornecido na biblioteca**
CSQ4VB8	COBOL	Definição de dados	SCSQCOBC
CSQ4BAQ	produção independente	Origem para o conjunto de dados VSAM	SCSQPROC
CSQ4FILE	produção independente	JCL para construir conjunto de dados VSAM utilizado por CSQ4CVB3	SCSQPROC
CSQ4S100	produção independente	Conjunto de dados de definição do sistema do CICS	SCSQPROC

Tabela 172. Amostras Asynchronous Consumption and Publish/Subscribe do CICS		
Nome membro	**Descrição**	**Arquivo de origem fornecido na biblioteca**
CSQ4CVCN	Origem para o programa Consumo de Mensagem Simples	SCSQCOBS
CSQ4CVCT	Consumo de Mensagem de Controle de Origem para o programa	SCSQCOBS
CSQ4CVEV	Origem para o programa Manipulador de Eventos	SCSQCOBS
CSQ4CVPT	Colocar Mensagem do Cliente para o programa de origem	SCSQCOBS
CSQ4CVRG	Origem para o programa cliente de Registro	SCSQCOBS
CSQ4S100	Conjunto de dados de definição do sistema do CICS	SCSQPROC

z/OS *Preparando o aplicativo de amostra para o ambiente do IMS no z/OS*

Parte do aplicativo de amostra de Credit Check pode ser executada no ambiente IMS.

Para preparar esta parte do aplicativo para executar com a amostra do CICS, primeiro execute as etapas descritas em “Preparando os aplicativos de amostra para o ambiente do CICS no z/OS” na página 1194.

Em seguida, execute as etapas a seguir:

1. Execute as mesmas etapas que você faria ao construir qualquer aplicativo IMS IBM MQ for z/OS. Essas etapas são listadas em “Criando aplicativos IMS (BMP ou MPP)” na página 1045. Os membros da biblioteca para uso são listados em Tabela 173 na página 1199.
2. Identifique o programa de aplicativo e banco de dados para o IMS. As amostras são fornecidas com as instruções PSBGEN, DBDGEN, ACB definition, IMSGEN e IMSDALOC para permitir isso.
3. Carregue o banco de dados CSQ4CA customizando e executando a JCL de amostra fornecida para esse propósito (CSQ4ILDB). Essa JCL carrega o banco de dados com os dados do arquivo CSQ4BAQ. Atualize a região de controle do IMS com uma instrução DD para o banco de dados CSQ4CA.
4. Inicie o programa de conta corrente como um programa de processamento de mensagens em lote (BMP), configurando e executando a JCL de amostra fornecida para esse propósito. Essa JCL inicia um programa BMP orientado por lote. Para executar o programa como um programa BMP orientado por mensagem, remova os caracteres de comentário da linha na JCL que contém a instrução IN=.

z/OS *Nomes do aplicativo IMS de amostra no z/OS*

Essas informações fornecem uma tabela com a lista das fontes e JCLs que são fornecidos para o aplicativo de Crédito de verificação de amostra do IMS.

Tabela 173. Origem e JCL para o Crédito de verificação de IMS de amostra (apenas C)

Nome do membro	Descrição	Fornecido na biblioteca
CSQ4CVB	Definições de objeto do IBM MQ	SCSQPROC
CSQ4ICB3	Origem para programa de conta corrente	SCSQC37S
CSQ4ICBL	Origem para carregar o banco de dados de conta corrente	SCSQC37S
CSQ4CBI	Definição de dados	SCSQC370
CSQ4PSBL	PSBGEN JCL para o programa de carregamento do banco de dados	SCSQPROC
CSQ4PSB3	PSBGEN JCL para o programa de conta corrente	SCSQPROC
CSQ4DBDS	DBDGEN JCL para o banco de dados CSQ4CA	SCSQPROC
CSQ4GIMS	IMSDefinições de macro GEN para CSQ4IVB3 e CSQ4CA	SCSQPROC
CSQ4ACBG	Definição de Application control block (ACB) para CSQ4IVB3	SCSQPROC
CSQ4BAQ	Origem para banco de dados	SCSQPROC
CSQ4ILDB	JCL de execução de amostra para o trabalho de carregamento de banco de dados	SCSQPROC
CSQ4ICBR	JCL de execução de amostra para o programa de conta corrente	SCSQPROC
CSQ4DYNA	IMSDefinições de macro DALOC para o banco de dados	SCSQPROC

z/OS *As amostras de Put no z/OS*

Os programas de amostra Put colocam mensagens em uma fila usando a chamada MQPUT.

Os programas de origem são fornecidos em C e COBOL no lote e ambientes do CICS (consulte Tabela 161 na página 1190 e Tabela 168 na página 1195).

Design da amostra Put

O fluxo pela lógica do programa é:

1. Conectar-se ao gerenciador de filas usando a chamada MQCONN. Se essa chamada falhar, imprima os códigos de conclusão e de razão e pare o processamento.

   Nota: Se estiver executando a amostra em um ambiente do CICS, não será necessário emitir uma chamada MQCONN; se fizer isso, DEF_HCONN será retornado. É possível usar a manipulação de conexão MQHC_DEF_HCONN para as chamadas MQI que seguem.

2. Abra a fila usando a chamada MQOPEN com a opção MQOO_OUTPUT. Na entrada para essa chamada, o programa usa a manipulação de conexão que é retornada na etapa “1” na página 1201. Para a

estrutura do descritor de objeto (MQOD), ele usa os valores padrão para todos os campos, exceto o campo nome da fila, que é transmitido como um parâmetro para o programa. Se a chamada MQOPEN falhar, imprima os códigos de conclusão e de razão e pare o processamento.

3. Crie um loop dentro do programa emitindo chamadas MQPUT até que o número necessário de mensagens seja colocado na fila. Se uma chamada MQPUT falhar, o loop será abandonado no começo, nenhuma outra chamada MQPUT será tentada e os códigos de conclusão e de razão serão retornados.
4. Feche a fila usando a chamada MQCLOSE com a manipulação de objetos retornada na etapa “2” na página 1201. Se essa chamada falhar, imprima os códigos de conclusão e de razão.
5. Desconecte-se do gerenciador de filas usando a chamada MQDISC com a manipulação de conexões retornada na etapa “1” na página 1201. Se essa chamada falhar, imprima os códigos de conclusão e de razão.

   Nota: Se estiver executando a amostra em um ambiente do CICS, não será necessário emitir uma chamada MQDISC.

z/OS *As amostras de Put para o ambiente de lote no z/OS*

Use este tópico ao considerar amostras Put para o ambiente de lote.

Para executar as amostras, edite e execute a JCL de amostra, conforme descrito em “Preparando e executando aplicativos de amostra para o ambiente de lote no z/OS” na página 1189.

Os programas aceitam os parâmetros a seguir em um EXEC PARM, separados por espaços em C e por vírgulas em COBOL:

1. O nome do gerenciador de filas (4 caracteres)
2. O nome da fila de destino (48 caracteres)
3. O número de mensagens (até 4 dígitos)
4. O caractere de preenchimento a ser inserido na mensagem (1 caractere)
5. O número de caracteres a inserir na mensagem (até 4 dígitos)
6. A persistência da mensagem (1 caractere: P para persistente ou N para não persistente)

Se você inserir qualquer um destes parâmetros incorretamente, receberá mensagens de erro apropriadas.

Quaisquer mensagens das amostras são gravadas no conjunto de dados SYSPRINT.

Observações de uso

- Para manter as amostras simples, há algumas pequenas diferenças funcionais entre as versões das linguagens. No entanto, essas diferenças são minimizados se você usar o layout dos parâmetros mostrados na JCL de execução de amostra, CSQ4BCJR e CSQ4BVJR. Nenhuma das diferenças está relacionada à MQI.
- CSQ4BCK1 permite inserir mais de quatro dígitos para o número de mensagens enviadas e o comprimento das mensagens.
- Para os dois campos numéricos, insira qualquer dígito no intervalo de 1 a 9999. O valor inserido deve ser um número positivo. Por exemplo, para efetuar put de uma única mensagem, é possível inserir 1, 01, 001 ou 0001 como o valor. Se inserir valores não numéricos ou negativos, poderá receber um erro. Por exemplo, se inserir -1, o programa COBOL envia uma mensagem de 1 byte, mas o programa C recebe um erro.
- Para ambos os programas, CSQ4BCK1 e CSQ4BVK1, deve-se inserir P no parâmetro de persistência, ++PER++, se desejar que a mensagem seja persistente. Se deixar de fazer isso, a mensagem será não persistente.

z/OS *As amostras de Put para o ambiente do CICS no z/OS*

Use este tópico quando considerar amostras Put para o ambiente do CICS.

As transações aceitam os parâmetros a seguir separados por vírgulas:

1. O número de mensagens (até 4 dígitos)
2. O caractere de preenchimento a ser inserido na mensagem (1 caractere)
3. O número de caracteres a inserir na mensagem (até 4 dígitos)
4. A persistência da mensagem (1 caractere: P para persistente ou N para não persistente)
5. O nome da fila de destino (48 caracteres)

Se você inserir qualquer um destes parâmetros incorretamente, receberá mensagens de erro apropriadas.

Para a amostra em COBOL, chame a amostra Put no ambiente do CICS inserindo:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

Para a amostra em C, chame a amostra Put no ambiente do CICS inserindo:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Quaisquer mensagens das amostras são exibidas na tela.

Observações de uso

- Para manter as amostras simples, há algumas pequenas diferenças funcionais entre as versões das linguagens. Nenhuma das diferenças está relacionada à MQI.
- Se você inserir um nome de fila com mais de 48 caracteres, seu comprimento será truncado para o máximo de 48 caracteres, mas nenhuma mensagem de erro será retornada.
- Antes de inserir a transação, pressione a tecla CLEAR.
- Para os dois campos numéricos, insira qualquer número no intervalo de 1 a 9999. O valor inserido deve ser um número positivo. Por exemplo, para colocar uma única mensagem, é possível inserir o valor 1, 01, 001 ou 0001. Se inserir valores não numéricos ou negativos, poderá receber um erro. Por exemplo, se inserir -1, o programa em COBOL envia uma mensagem de 1 byte e o programa em C é encerrado de forma anormal com um erro de malloc().
- Para ambos os programas, CSQ4CCK1 e CSQ4CVK1, insira P no parâmetro persistence se desejar que a mensagem seja persistente. Para mensagens não persistentes, insira N no parâmetro persistence. Se inserir qualquer outro valor, você receberá uma mensagem de erro.
- As mensagens são colocadas no ponto de sincronização porque os valores padrão são usados para todos os parâmetros, exceto aqueles configurados durante a chamada de programa.

z/OS *As amostras de Get no z/OS*

Os programas de amostra GET obtêm mensagens de uma fila usando a chamada MQGET.

Os programas de origem são fornecidos em C e COBOL no lote e ambientes do CICS (consulte Tabela 161 na página 1190 e Tabela 168 na página 1195).

z/OS *Design da amostra de Get no z/OS*

Saiba sobre o design da amostra Get e algumas observações de uso a considerar.

O fluxo pela lógica do programa é:

1. Conectar-se ao gerenciador de filas usando a chamada MQCONN. Se essa chamada falhar, imprima os códigos de conclusão e de razão e pare o processamento.

   Nota: Se estiver executando a amostra em um ambiente do CICS, não será necessário emitir uma chamada MQCONN; se fizer isso, DEF_HCONN será retornado. É possível usar a manipulação de conexão MQHC_DEF_HCONN para as chamadas MQI que seguem.
2. Abrir a fila usando a chamada MQOPEN com as opções MQOO_INPUT_SHARED e MQOO_BROWSE. Na entrada para essa chamada, o programa usa a manipulação de conexão que é retornada na etapa “1”

na página 1201. Para a estrutura do descritor de objeto (MQOD), ele usa os valores padrão para todos os campos, exceto o campo nome da fila, que é transmitido como um parâmetro para o programa. Se a chamada MQOPEN falhar, imprima os códigos de conclusão e de razão e pare o processamento.

3. Criar um loop dentro do programa emitindo chamadas MQGET até que o número de mensagens requerido seja recuperado da fila. Se uma chamada MQGET falhar, o loop será abandonado no início, nenhuma outra chamada MQGET será tentada e os códigos de conclusão e de razão serão retornados. As opções a seguir são especificadas na chamada MQGET:
   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT ou MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST e MQGMO_BROWSE_NEXT

   Para obter uma descrição dessas opções, consulte MQGET. Para cada mensagem, o número da mensagem será impresso seguido pelo comprimento da mensagem e os dados da mensagem.
4. Feche a fila usando a chamada MQCLOSE com a manipulação de objetos retornada na etapa "2" na página 1201. Se essa chamada falhar, imprima os códigos de conclusão e de razão.
5. Desconecte-se do gerenciador de filas usando a chamada MQDISC com a manipulação de conexões retornada na etapa "1" na página 1201. Se essa chamada falhar, imprima os códigos de conclusão e de razão.

   Nota: Se estiver executando a amostra em um ambiente do CICS, não será necessário emitir uma chamada MQDISC.

Observações de uso

- Para manter as amostras simples, há algumas pequenas diferenças funcionais entre as versões das linguagens. No entanto, essas diferenças são minimizadas se você usar o layout dos parâmetros mostrados na execução de amostra JCL, CSQ4BCJR e CSQ4BVJR. Nenhuma das diferenças está relacionada à MQI.
- CSQ4BCJ1 permite inserir mais de quatro dígitos para o número de mensagens recuperadas.
- Mensagens maiores que 64 KB são truncadas.
- CSQ4BCJ1 pode exibir corretamente somente mensagens de caracteres, pois exibe somente até o primeiro caractere NULL (\0) ser exibido.
- Para o campo numérico de número de mensagens, insira qualquer dígito no intervalo de 1 a 9999. O valor inserido deve ser um número positivo. Por exemplo, para efetuar get de uma única mensagem, é possível inserir 1, 01, 001 ou 0001 como o valor. Se inserir valores não numéricos ou negativos, poderá receber um erro. Por exemplo, se inserir -1, o programa em COBOL recupera uma mensagem, mas o programa em C não recupera nenhuma mensagem.
- Para ambos os programas, CSQ4BCJ1 e CSQ4BVJ1, insira B no parâmetro get, ++GET++, se desejar procurar as mensagens.
- Para ambos os programas, CSQ4BCJ1 e CSQ4BVJ1, insira S no parâmetro syncpoint, ++SYNC++, para que as mensagens sejam recuperadas no ponto de sincronização.

z/OS *As amostras de Get para o ambiente de lote no z/OS*

Para executar as amostras, edite e execute a JCL de amostra, conforme descrito em "Preparando e executando aplicativos de amostra para o ambiente de lote no z/OS" na página 1189.

Os programas aceitam os parâmetros a seguir em um EXEC PARM, separados por espaços em C e por vírgulas em COBOL:

1. O nome do gerenciador de filas (4 caracteres)
2. O nome da fila de destino (48 caracteres)
3. O número de mensagens a obter (até 4 dígitos)

4. A opção browse/get message (1 caractere: B para procurar ou D para obter as mensagens destrutivamente)
5. O controle de ponto de sincronização (1 caractere: S para ponto de sincronização ou N para nenhum ponto de sincronização)

Se inserir qualquer um desses parâmetros incorretamente, você receberá mensagens de erro apropriadas.

A saída das amostras é gravada no conjunto de dados SYSPRINT:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *As amostras de Get para o ambiente do CICS no z/OS*

Considerações especiais para as amostras Get para o ambiente do CICS.

As transações aceitam os parâmetros a seguir em um EXEC PARM, separados por vírgulas:

1. O número de mensagens a obter (até quatro dígitos)
2. A opção browse/get message (um caractere: B para procurar ou D para obter as mensagens destrutivamente)
3. O controle de ponto de sincronização (um caractere: S para ponto de sincronização ou N para nenhum ponto de sincronização)
4. O nome da fila de destino (48 caracteres)

Se inserir qualquer um desses parâmetros incorretamente, você receberá mensagens de erro apropriadas.

Para a amostra em COBOL, chame a amostra Get no ambiente do CICS inserindo:

```
MVGT,9999,B,S,QUEUE.NAME
```

Para a amostra em C, chame a amostra Get no ambiente do CICS inserindo:

```
MCGT,9999,B,S,QUEUE.NAME
```

Quando as mensagens são recuperadas da fila, elas são colocadas em uma fila de armazenamento temporário do CICS com o mesmo nome que a transação do CICS (por exemplo, MCGT para a amostra C).

Aqui está uma saída de exemplo das amostras Get:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE **********************
```

Observações de uso

- Para manter as amostras simples, há algumas pequenas diferenças funcionais entre as versões das linguagens. Nenhuma das diferenças está relacionada à MQI.
- Se você inserir um nome de fila com mais de 48 caracteres, seu comprimento será truncado para o máximo de 48 caracteres, mas nenhuma mensagem de erro será retornada.
- Antes de inserir a transação, pressione a tecla CLEAR.
- CSQ4CCJ1 pode exibir corretamente somente mensagens de caracteres, pois exibe somente até o primeiro caractere NULL (\0) ser exibido.
- Para o campo numérico, insira qualquer número no intervalo de 1 a 9999. O valor inserido deve ser um número positivo. Por exemplo, para obter uma única mensagem, é possível inserir o valor 1, 01, 001 ou 0001. Se inserir um valor não numérico ou negativo, poderá receber um erro.
- Mensagens com mais de 24 526 bytes de comprimento em C e 9 950 bytes em COBOL são truncadas. Isso ocorre devido à maneira como as filas de armazenamento temporário do CICS são usadas.
- Para ambos os programas, CSQ4CCK1 e CSQ4CVK1, insira B no parâmetro GET se você deseja procurar as mensagens, caso contrário, insira D. Isso executa chamadas MQGET destrutivas. Se inserir qualquer outro valor, você receberá uma mensagem de erro.
- Para ambos os programas, CSQ4CCJ1 e CSQ4CVJ1, insira S no parâmetro syncpoint para recuperar as mensagens no ponto de sincronização. Se inserir N no parâmetro syncpoint, as chamadas MQGET serão emitidas fora do ponto de sincronização. Se inserir qualquer outro valor, você receberá uma mensagem de erro.

z/OS *A amostra de procura no z/OS*

A amostra Procurar é um aplicativo em lote que demonstra como procurar por mensagens em uma fila usando a chamada MQGET.

O aplicativo percorre todas as mensagens em uma fila, imprimindo os primeiros 80 bytes de cada uma. Você pode usar esse aplicativo para ver as mensagens em uma fila sem alterá-las.

Os programas de origem e a JCL de execução de amostra são fornecidos no COBOL, montador, linguagens PL/I e C (consulte Tabela 162 na página 1190).

Para iniciar o aplicativo, edite e execute a JCL de execução de amostra, conforme descrito em “Preparando e executando aplicativos de amostra para o ambiente de lote no z/OS” na página 1189. É possível procurar mensagens em uma de suas próprias filas, especificando o nome da fila na JCL de execução.

Quando você executa o aplicativo (e há algumas mensagens na fila), o conjunto de dados de saída é semelhante a isto:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

Se não houver mensagens na fila, o conjunto de dados conterá os títulos e a mensagem `End of report` apenas. Se um erro ocorrer com qualquer uma das chamadas MQI, os códigos de conclusão e de razão são incluídos no conjunto de dados de saída.

z/OS *Design da amostra de procura no z/OS*

O aplicativo de amostra Procurar utiliza um único módulo de programa; um é fornecido em cada uma das linguagens de programação suportadas.

O fluxo pela lógica do programa é:

1. Abrir um conjunto de dados de impressão e imprimir a linha de título do relatório. Verificar se os nomes do gerenciador de filas e da fila foram passados a partir da JCL. Se ambos os nomes foram passados, imprimir as linhas do relatório que contêm os nomes. Se não foram, imprimir uma mensagem de erro, fechar o conjunto de dados de impressão e parar o processamento.

   A forma como o programa testa os parâmetros que são passados da JCL depende da linguagem na qual o programa está escrito; para obter mais informações, consulte "Considerações de design dependente da linguagem no z/OS" na página 1206.
2. Conectar-se ao gerenciador de filas usando a chamada MQCONN. Se essa chamada não for bem-sucedida, imprimir os códigos de conclusão e de razão, fechar o conjunto de dados de impressão e parar o processamento.
3. Abrir a fila usando a chamada MQOPEN com a opção MQOO_BROWSE. Na entrada para essa chamada, o programa usa a manipulação de conexões retornada na etapa "2" na página 1205. Para a estrutura do descritor de objeto (MQOD), usa os valores padrão para todos os campos, exceto o nome da fila (que foi passado na etapa "1" na página 1205). Se essa chamada não for bem-sucedida, imprimir os códigos de conclusão e de razão, fechar o conjunto de dados de impressão e parar o processamento.
4. Procurar a primeira mensagem na fila, usando a chamada MQGET. Na entrada para essa chamada, o programa especifica:
   - As manipulações de conexões e de filas das etapas "2" na página 1205 e "3" na página 1205
   - Uma estrutura MQMD com todos os campos configurados para seus valores iniciais
   - Duas opções:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - Um buffer com tamanho de 80 bytes para conter os dados copiados da mensagem

   A opção MQGMO_ACCEPT_TRUNCATED_MSG permite que a chamada seja concluída, mesmo se a mensagem for maior do que o buffer de 80 bytes especificado na chamada. Se a mensagem for maior do que o buffer, a mensagem será truncada para caber no buffer e os códigos de conclusão e de razão serão configurados para mostrar isso. A amostra foi projetado para que as mensagens sejam truncadas em 80 caracteres para tornar o relatório fácil de ler. O tamanho do buffer é configurado por uma instrução `DEFINE`, portanto, é possível mudá-lo facilmente, se desejar.
5. Executar o loop a seguir até a chamada MQGET falhar:
   a. Imprimir uma linha do relatório que mostre:
      - O número de sequência da mensagem (essa é uma contagem das operações de procura).
      - O comprimento verdadeiro da mensagem (não o comprimento truncado). Esse valor é retornado no campo `DataLength` da chamada MQGET.
      - Os primeiros 80 bytes dos dados da mensagem.

   b. Reconfigurar os campos `MsqId` e `CorrelId` da estrutura MQMD para nulos

   c. Procurar a próxima mensagem, usando a chamada MQGET com estas duas opções:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. Se a chamada MQGET falhar, teste o código de razão para ver se a chamada falhou porque o cursor de procura chegou ao fim da fila. Nesse caso, imprimir a mensagem `End of report` e ir para a etapa "7" na página 1206; caso contrário, imprimir os códigos de conclusão e de razão, fechar o conjunto de dados de impressão e parar o processamento.

7. Feche a fila usando a chamada MQCLOSE com a manipulação de objetos retornada na etapa “3” na página 1205.
8. Desconecte-se do gerenciador de filas usando a chamada MQDISC com a manipulação de conexões retornada na etapa “2” na página 1205.
9. Fechar o conjunto de dados de impressão e parar o processamento.

z/OS *Considerações de design dependente da linguagem no z/OS*

Os módulos de origem são fornecidos para a amostra Browse em quatro linguagens de programação.

Há duas diferenças principais entre os módulos de origem:

- Ao testar os parâmetros transmitidos a partir da execução do JCL, os módulos de linguagem assembler, COBOL, e PL/I procuram pelo caractere de vírgula (,). Se o JCL transmite `PARM=(,LOCALQ1)`, o aplicativo tenta abrir a fila LOCALQ1 no gerenciador de filas padrão. Se não houver nenhum nome após a vírgula (ou nenhuma vírgula), o aplicativo retornará um erro. O módulo C não procura o caractere de vírgula. Se a JCL passar um parâmetro único (por exemplo, `PARM=('LOCALQ1'))`, o módulo C usa isso como um nome de fila no gerenciador de filas padrão.
- Para manter o módulo da linguagem assembler simples, usa o formato de data *yy/ddd* (por exemplo, 05/116) quando cria o relatório de impressão. Os outros módulos usam a data do calendário no formato *mm/dd/yy*.

z/OS *A amostra de impressão de mensagem no z/OS*

A amostra Imprimir Mensagem é um aplicativo em lote que demonstra como remover todas as mensagens de uma fila usando a chamada MQGET.

A amostra Print Message usa três parâmetros:

1. O nome do gerenciador de filas
2. O nome da fila de origem
3. Um parâmetro opcional para as propriedades

Também imprime, para cada mensagem, os campos do descritor de mensagens, seguido pelos dados da mensagem. O programa imprime os dados em hexadecimais e como caracteres (se forem imprimíveis). Se um caractere não for imprimível, o programa o substitui por um caractere de ponto (.) É possível usar o programa ao diagnosticar problemas com um aplicativo que está colocando mensagens em uma fila.

Os valores permitidos para o parâmetro da propriedade são:

Tabela 174. Valores permitidos para o parâmetro da propriedade

Value	**Comportamento**
0	Comportamento padrão, como era para o IBM WebSphere MQ 6. As propriedades que são entregues ao aplicativo dependem do atributo de fila **`PropertyControl`** do qual a mensagem é recuperada.

Tabela 174. 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 WebSphere MQ 6 estar incluída, caso contrário, as propriedades serão descartadas.

É possível mudar o aplicativo para que procure as mensagens, em vez de removê-las da fila. Para fazer isso, compile com a opção de -DBROWSE para definir a macro BROWSE, conforme indicado em "Design da amostra de impressão de mensagem no z/OS" na página 1208. Código executável é fornecido para você na biblioteca SCSQLOAD. O módulo CSQ4BCG0 é construído com -DBROWSE; o módulo CSQ4BCG1 lê destrutivamente a fila.

O aplicativo tem um programa de origem única, escrito na linguagem C. O código JCL de execução de amostra também é fornecido (consulte Tabela 163 na página 1191).

Para iniciar o aplicativo, edite e execute a JCL de execução de amostra, conforme descrito em "Preparando e executando aplicativos de amostra para o ambiente de lote no z/OS" na página 1189. Ao executar o aplicativo (e se houver algumas mensagens na fila), o conjunto de dados de saída é semelhante àquele em Figura 151 na página 1208.

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR   '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                 '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '   '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

Figura 151. Exemplo de um relatório do aplicativo de amostra Print Message

z/OS *Design da amostra de impressão de mensagem no z/OS*

O aplicativo de amostra Print Message usa um único programa escrito na linguagem C.

O fluxo pela lógica do programa é:

1. Verificar se os nomes do gerenciador de filas e da fila foram passados a partir da JCL. Se não tiverem sido, imprimir uma mensagem de erro e parar o processamento.
2. Conectar-se ao gerenciador de filas usando a chamada MQCONN. Se essa chamada não for bem-sucedida, imprimir os códigos de conclusão e de razão e parar o processamento; caso contrário, imprimir o nome do gerenciador de filas.
3. Abrir a fila usando a chamada MQOPEN com a opção MQOO_INPUT_SHARED.

   Nota: Se desejar que o aplicativo procure as mensagens em vez de removê-las da fila, compilar a amostra com -DBROWSE ou incluir `#define BROWSE` no início do código-fonte. Ao fazer isso, o pré-processador de macro inclui a linha no programa que seleciona a opção MQOO_BROWSE na compilação.

Na entrada para essa chamada, o programa usa a manipulação de conexões retornada na etapa "2" na página 1208. Para a estrutura do descritor de objeto (MQOD), usa os valores padrão para todos os campos, exceto o nome da fila (que foi passado na etapa "1" na página 1208). Se essa chamada não for bem-sucedida, imprimir os códigos de conclusão e de razão e parar o processamento; caso contrário, imprimir o nome da fila.

4. Se usar um identificador de mensagem para obter as propriedades de mensagem, usar MQCRTMH para criar esse identificador para uso com chamadas MQGET subsequentes. Se essa chamada não for bem-sucedida, imprimir os códigos de conclusão e de razão e parar o processamento.
5. Configurar as opções de get message para refletirem a ação de solicitação de quaisquer propriedades de mensagem.
6. Executar o loop a seguir até a chamada MQGET falhar:
   a. Inicializar o buffer para espaços em branco para que os dados da mensagem não sejam corrompidos por quaisquer dados que já estejam no buffer.
   b. Configurar os campos `MsgId` e `CorrelId` da estrutura MQMD para nulos para que a chamada MQGET selecione a primeira mensagem da fila.
   c. Obter uma mensagem da fila, usando a chamada MQGET. Na entrada para essa chamada, o programa especifica:
      - Os identificadores de conexão e de objeto das etapas "2" na página 1208 e "3" na página 1208.
      - Uma estrutura MQMD com todos os campos configurados para seus valores iniciais. (`MsgId` e `CorrelId` são reconfigurados para nulos para cada chamada MQGET.)
      - A opção MQGMO_NO_WAIT.

        Nota: Se desejar que o aplicativo procure as mensagens em vez de removê-las da fila, compilar a amostra com -DBROWSE ou incluir `#define BROWSE` no início do código-fonte. Ao fazer isso, o pré-processador de macro inclui a linha no programa que seleciona a opção MQGMO_BROWSE_NEXT para a compilação. Quando essa opção for usada em uma chamada com relação a uma fila para a qual nenhum cursor de procura foi anteriormente usado com a manipulação de objetos atual, o cursor de procura será posicionado de forma lógica da primeira mensagem.
      - Um buffer com tamanho de 64 KB para conter os dados copiados da mensagem.
   d. Chamar a sub-rotina printMD. Isso imprime o nome de cada campo no descritor de mensagens, seguido por seu conteúdo.
   e. Se você criou um identificador de mensagem na etapa "4" na página 1209, chamar a sub-rotina printProperties para exibir quaisquer propriedades de mensagem.
   f. Imprimir o comprimento da mensagem, seguido pelos dados da mensagem. Cada linha de dados da mensagem está neste formato:
      - Posição relativa (em hexadecimal) desta parte dos dados
      - 16 bytes de dados hexadecimais
      - Os mesmos 16 bytes de dados em formato de caractere, se imprimíveis (caracteres não imprimíveis são substituídos por pontos)
7. Se a chamada MQGET falhar, testar o código de razão para ver se a chamada falhou porque não há mais mensagens na fila. Nesse caso, imprimir a mensagem: No more messages; caso contrário, imprimir os códigos de conclusão e de razão. Em ambos os casos, ir para a etapa "9" na página 1210.

   Nota: A chamada MQGET falhará se localizar uma mensagem que tenha mais de 64 KB de dados. Para mudar o programa para manipular mensagens maiores, você poderia executar um dos seguintes:

   - Incluir a opção MQGMO_ACCEPT_TRUNCATED_MSG na chamada MQGET para que a chamada obtenha os primeiros 64 KB de dados e descarte o restante
   - Fazer o programa deixar a mensagem na fila quando localizar uma com essa quantia de dados
   - Aumentar o tamanho do buffer

8. Se você criou um identificador de mensagem na etapa “4” na página 1209, chamar MQDLTMH para excluí-lo.
9. Feche a fila usando a chamada MQCLOSE com a manipulação de objetos retornada na etapa “3” na página 1208.
10. Desconecte-se do gerenciador de filas usando a chamada MQDISC com a manipulação de conexões retornada na etapa “2” na página 1208.

z/OS *A amostra de atributos da fila no z/OS*

A amostra Queue Attributes é um aplicativo CICS de modo conversacional que demonstra o uso das chamadas MQINQ e MQSET.

Mostra como consultar sobre os valores dos atributos **InhibitPut** e **InhibitGet** de filas e como mudá-los para que os programas não possam colocar mensagens ou obter mensagens de uma fila. Você poderá querer *bloquear* uma fila dessa maneira quando você estiver testando um programa.

Para evitar interferência acidental com suas próprias filas, essa amostra funciona somente em um objeto de fila que tem os caracteres CSQ4SAMP nos primeiros oito bytes de seu nome. No entanto, o código-fonte inclui comentários para mostrar como remover essa restrição.

Programas de origem são fornecidos nas linguagens COBOL, assembler e C (consulte Tabela 169 na página 1196).

A versão da linguagem assembler da amostra usa código reentrante. Para fazer isso, você observará que o código para cada chamada MQI nessa versão da amostra inclui a palavra-chave MF; por exemplo:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(A palavra-chave VL significa que é possível usar a transação fornecida pelo Recurso de diagnóstico de função do CICS (CEDF) para depurar o programa.) Para obter informações adicionais sobre como escrever programas reentrantes, consulte Codificação em linguagem assembler do System/390.

Para iniciar o aplicativo, inicie seu sistema CICS e use as transações do CICS a seguir:

- Para COBOL, MVC1
- Para linguagem assembler, MAC1
- Para C, MCC1

É possível mudar o nome de qualquer uma dessas transações mudando o conjunto de dados CSD mencionado na etapa 3.

Design da amostra

Ao iniciar a amostra, ela exibirá um mapa de tela que tem campos para:

- Nome da fila
- Solicitação do usuário (as ações válidas são: inquire, allow ou inhibit)
- Status atual de operações put para a fila
- Status atual de operações get para a fila

Os dois primeiros campos são para entrada do usuário. Os dois últimos campos são preenchidos pelo aplicativo: eles mostram a palavra `INHIBITED` ou a palavra `ALLOWED`.

O aplicativo valida os valores que você inserir nos primeiros dois campos. Verifica se o nome da fila começa com os caracteres `CSQ4SAMP` e se você digitou uma das três solicitações válidas no campo Action. O aplicativo converte todas as entradas para maiúsculas, portanto, não é possível usar nenhuma fila com nome que contenha caracteres minúsculos.

Se você inserir `inquire` no campo **Ação**, o fluxo pela lógica do programa será:

1. Abrir a fila usando a chamada MQOPEN com a opção MQOO_INQUIRE

2. Chamar MQINQ usando os seletores MQIA_INHIBIT_GET e MQIA_INHIBIT_PUT
3. Fechar a fila usando a chamada MQCLOSE
4. Analisar os atributos retornados no parâmetro **IntAttrs** da chamada MQINQ e mover as palavras INHIBITED ou ALLOWED, conforme apropriado, para os campos relevantes da tela

Se você inserir `inhibit` no campo **Ação**, o fluxo pela lógica do programa será:

1. Abrir a fila usando a chamada MQOPEN com a opção MQOO_SET
2. Chamar MQSET usando os seletores MQIA_INHIBIT_GET e MQIA_INHIBIT_PUT e com os valores MQQA_GET_INHIBITED e MQQA_PUT_INHIBITED no parâmetro **IntAttrs**
3. Fechar a fila usando a chamada MQCLOSE
4. Mover a palavra INHIBITED para os campos relevantes da tela

Se você inserir `allow` no campo **Ação**, o aplicativo executará processamento similar àquele de uma solicitação para inibir. As únicas diferenças são as configurações dos atributos e as palavras exibidas na tela.

Quando o aplicativo abre a fila, ele usa o identificador de conexão padrão para o gerenciador de filas. (O CICS estabelece uma conexão com o gerenciador de filas ao iniciar o seu sistema CICS.) O aplicativo pode efetuar trap dos erros a seguir neste estágio:

- O aplicativo não está conectado ao gerenciador de filas
- A fila não existe
- O usuário não está autorizado a acessar a fila
- O aplicativo não está autorizado a abrir a fila

Por outros erros da MQI, o aplicativo exibe os códigos de conclusão e de razão.

z/OS *A amostra de gerenciador de e-mail no z/OS*

O aplicativo de amostra do Mail Manager é um conjunto de programas que demonstra o envio e o recebimento de mensagens, ambos dentro de um único ambiente e em ambientes diferentes. O aplicativo é um simples sistema de correspondência eletrônica que permite que os usuários troquem mensagens, mesmo que utilizem gerenciadores de filas diferentes.

O aplicativo demonstra como criar filas usando a chamada MQOPEN e colocando os comandos do IBM MQ for z/OS na fila de entrada de comandos do sistema.

Três versões do aplicativo são fornecidas:

- Um aplicativo CICS gravado em COBOL
- Um aplicativo TSO gravado em COBOL
- Um aplicativo TSO gravado em C

z/OS *Preparando a amostra de gerenciador de e-mail no z/OS*

O Mail Manager é fornecido em versões que são executadas em dois ambientes. A preparação que se deve realizar antes de executar o aplicativo depende do ambiente que deseja usar.

Os usuários podem acessar filas de mensagens e filas de apelidos a partir do TSO e do CICS, contanto que seus IDs de usuário de conexão sejam os mesmos em cada sistema.

Antes ser possível enviar mensagens para outro gerenciador de filas, deve-se configurar um canal de mensagens para esse gerenciador de filas. Para fazer isso, use a função de controle de canal do IBM MQ, descrita em Função de controle de canal.

Preparando a amostra para o ambiente do TSO

Siga estas etapas:

1. Preparar a amostra conforme descrito em “Preparando os aplicativos de amostra para o ambiente TSO no z/OS” na página 1192.

2. Customizar o CLIST fornecido para a amostra para definir:
   - O local dos painéis
   - O local do arquivo de mensagens
   - A localização dos módulos de carregamento
   - O nome do gerenciador de filas que você deseja usar com o aplicativo

   Um CLIST separado é fornecido para cada versão de linguagem da amostra:
   - Para a versão do COBOL: CSQ4RVD1
   - Para a versão do C: CSQ4RCD1
3. Assegure que as filas usadas pelo aplicativo estejam disponíveis no gerenciador de filas. (As filas são definidas no CSQ4CVD.)

Nota: VS COBOL II não suporta multitarefas com o ISPF. Isso significa que não é possível usar o aplicativo de amostra Mail Manager em ambos os lados de uma tela dividida. Se fizer isso, os resultados serão imprevisíveis.

z/OS *Executando a amostra de gerenciador de e-mail no z/OS*

Para iniciar a amostra no ambiente do CICS Transaction Server for z/OS, execute a transação MAIL. Se você ainda não tiver se conectado ao CICS, o aplicativo solicitará que insira um ID do usuário para o qual ele pode enviar seus e-mails.

Ao iniciar o aplicativo, ele abre sua fila de mensagens. Se essa fila não existir, o aplicativo cria uma para você. As filas de e-mail possuem nomes do formulário CSQ4SAMP.MAILMGR. *userid*, em que *userid* depende do ambiente:

No TSO
O ID do TSO do usuário

No CICS
A conexão do CICS do usuário ou o ID do usuário inserido pelo usuário quando solicitado na inicialização do Mail Manager

Todas as partes dos nomes de filas que o Mail Manager usa devem estar em letras maiúsculas.

O aplicativo apresenta, então, um painel de menu que tem opções para:

- Ler mensagem recebida
- Enviar mensagem
- Criar apelido

O painel de menu também mostra quantas mensagens estão esperando em sua fila de mensagens. Cada uma das opções de menu exibe um painel adicional:

Ler mensagem recebida
O Mail Manager exibe uma lista das mensagens que estão em sua fila de mensagens. (Somente as 99 primeiras mensagens na fila são exibidas.) Para obter um exemplo deste painel, consulte Figura 154 na página 1216. Ao selecionar uma mensagem nessa lista, o conteúdo da mensagem é exibido (consulte Figura 155 na página 1217).

Enviar mensagem
Um painel solicita que você insira:
- O nome do usuário a quem deseja enviar uma mensagem
- O nome do gerenciador de filas que possui a fila de mensagens
- O texto de sua mensagem

No campo de nome do usuário, é possível inserir um ID de usuário ou um apelido criado usando o Mail Manager. É possível deixar o campo do nome do gerenciador de filas em branco se a fila de mensagens do usuário pertencer ao mesmo gerenciador de filas que você está usando e deve deixá-lo em branco se tiver inserido um apelido no campo de nome de usuário:

- Se especificar somente um nome de usuário, o programa primeiramente assume que o nome é um apelido e envia a mensagem para o objeto definido por esse nome. Se não houver tal apelido, o programa tentará enviar a mensagem para uma fila local com esse nome.
- Se especificar um nome de usuário e um nome de gerenciador de filas, o programa envia a mensagem para a fila de mensagens definida por esses dois nomes.

Por exemplo, se desejar enviar uma mensagem para o usuário JONESM no gerenciador de filas remotas QM12, você poderia enviar uma mensagem a eles de uma de duas maneiras:

- Use ambos os campos para especificar o usuário JONESM no gerenciador de filas QM12.
- Defina um apelido (por exemplo, MARY) para esse usuário e envie uma mensagem a eles colocando MARY no campo de nome de usuário e nada no campo do nome do gerenciador de filas.

Criar apelido

É possível definir um nome fácil de lembrar que pode ser usado ao enviar uma mensagem para outro usuário que você entre em contato com frequência. Será solicitado que você insira o ID do usuário do outro usuário e o nome do gerenciador de filas que tem a fila de mensagens dele.

Apelidos são filas que possuem nomes do formulário CSQ4SAMP.MAILMGR. *userid.nickname*, em que *userid* é o seu próprio ID de usuário e *apelido* é o apelido que você deseja utilizar. Com nomes estruturados dessa maneira, os usuários podem ter cada um seu próprio conjunto de apelidos.

O tipo de fila que o programa cria depende de como você preenche os campos do painel Create Nickname:

- Se você especificar somente um nome de usuário ou se o nome do gerenciador de filas for o mesmo que o do gerenciador de filas para ao qual o Mail Manager está conectado, o programa cria uma fila de alias.
- Se você especificar um nome de usuário e um nome de gerenciador de filas (e o gerenciador de filas não for aquele ao qual o Mail Manager está conectado), o programa cria uma definição local de uma fila remota. O programa não verifica a existência da fila para a qual essa definição resolve, nem mesmo se o gerenciador de filas remotas existe.

Por exemplo, se o seu próprio ID do usuário for SMITHK e você criar um apelido chamado MARY para o usuário JONESM (que usa o gerenciador de filas remotas QM12), o programa de apelido cria uma definição local de uma fila remota denominada CSQ4SAMP.MAILMGR.SMITHK.MARY. Essa definição é resolvida para a fila de mensagens de Mary, que é CSQ4SAMP.MAILMGR.JONESM no gerenciador de filas QM12. Se você estiver usando o gerenciador de filas QM12, o programa cria uma fila de alias com o mesmo nome (CSQ4SAMP.MAILMGR.SMITHK.MARY).

A versão em C do aplicativo TSO faz melhor uso dos recursos de manipulação de mensagens do ISPF do que versão em COBOL. Você pode notar que mensagens de erro diferentes são exibidas pelas versões em C e em COBOL.

z/OS *Design da amostra de gerenciador de e-mail no z/OS*

As seções a seguir descrevem cada um dos programas que compõem o aplicativo de amostra Mail Manager.

Os relacionamentos entre os programas e os painéis que o aplicativo usa são mostrados em Figura 152 na página 1214 para a versão do TSO e Figura 153 na página 1215 para a versão do CICS Transaction Server for z/OS.

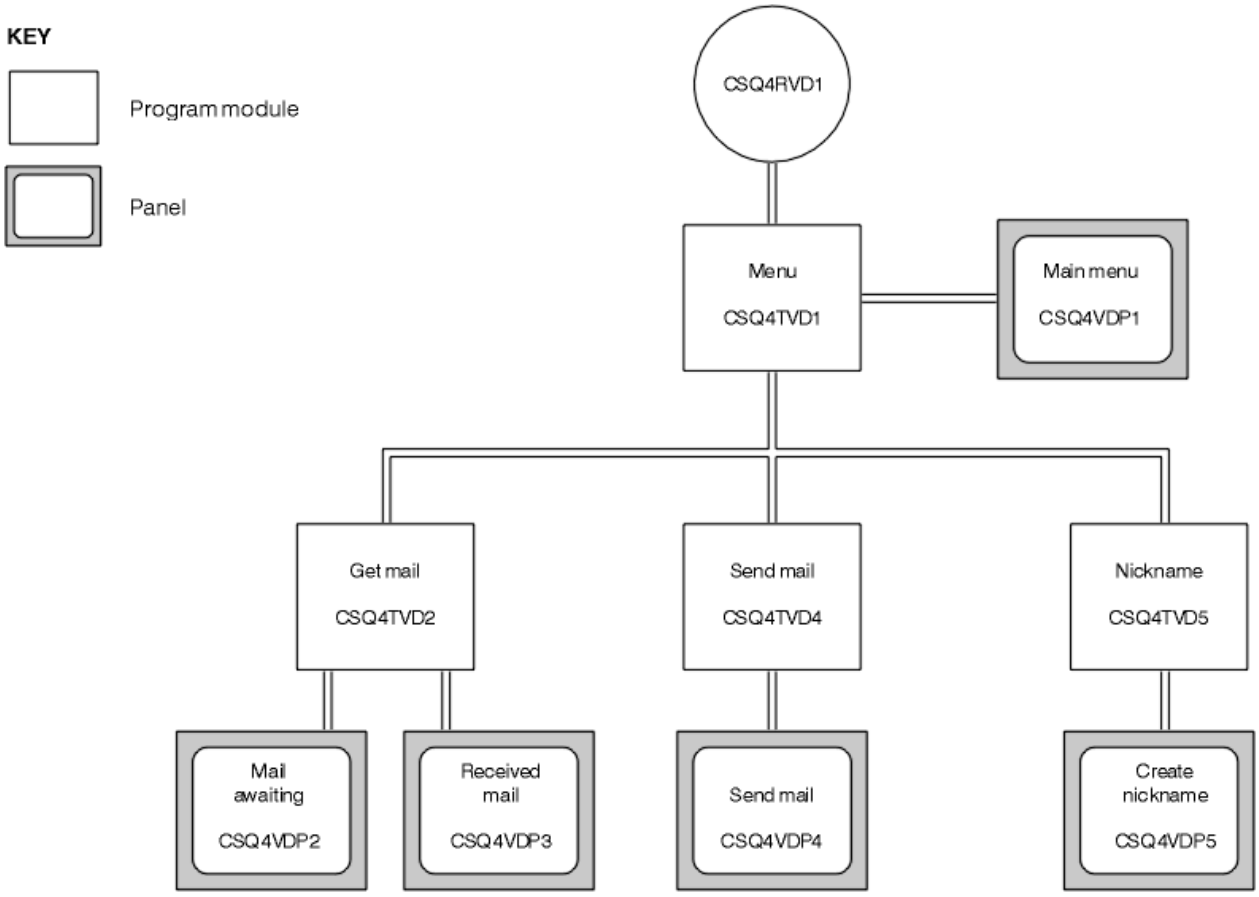


Figura 152. Programas e painéis para as versões do TSO do Mail Manager

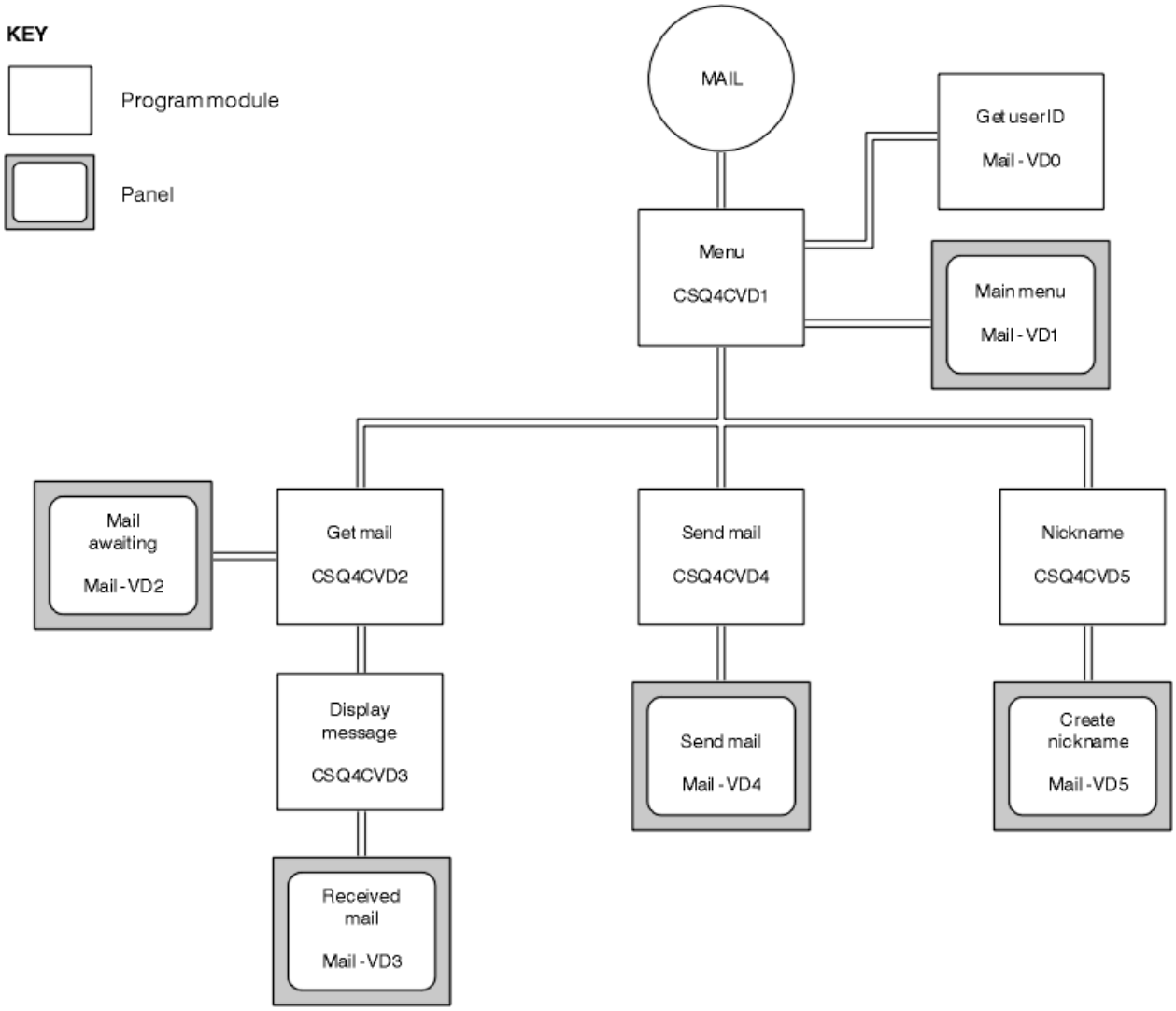


Figura 153. Programas e painéis para a versão do CICS do Mail Manager

z/OS *Programa de menu no z/OS*

No ambiente do TSO, o programa de menu é chamado por CLIST. No ambiente CICS, o programa é chamado por transação MAIL.

O programa de menu (CSQ4TVD1 para TSO, CSQ4CVD1 para CICS) é o programa inicial no conjunto. Ela exibe o menu (CSQ4VDP1 para TSO, VD1 para CICS) e chama os outros programas quando selecionados no menu.

O programa obtém primeiramente o ID do usuário:

- Na versão do CICS do programa, se o usuário tiver se conectado ao CICS, o ID do usuário será obtido usando o comando ASSIGN USERID do CICS. Se o usuário não tiver se conectado, o programa exibe o painel de conexão (CSQ4VD0) para solicitar ao usuário que insira um ID de usuário. Não há processamento de segurança dentro desse programa; o usuário pode fornecer qualquer ID do usuário.
- Na versão do TSO, o ID do usuário é obtido a partir do TSO em CLIST. Ele é passado para o programa de menu como uma variável no conjunto compartilhado ISPF.

Após o programa obter o ID do usuário, ele verifica para assegurar que o usuário tenha uma fila de mensagens (CSQ4SAMP.MAILMGR. *userid*). Se uma fila de mensagens não existir, o programa cria uma colocando uma mensagem na fila de entrada de comandos do sistema. A mensagem contém o comando DEFINE QLOCAL do IBM MQ for z/OS. A definição de objeto que esse comando usa configura a profundidade máxima da fila para 9999 mensagens.

O programa também cria uma fila dinâmica temporária para manipular respostas da fila de entrada de comandos do sistema. Para fazer isso, o programa usa a chamada MQOPEN, especificando SYSTEM.DEFAULT.MODEL.QUEUE como o modelo para a fila dinâmica. O gerenciador de filas cria a fila dinâmica temporária com um nome que tenha o prefixo CSQ4SAMP; o restante do nome será gerado pelo gerenciador de filas.

O programa abre, então, a fila de mensagens do usuário e localiza o número de mensagens na fila por consulta sobre a profundidade atual da fila. Para fazer isso, o programa usa a chamada MQINQ, especificando o seletor MQIA_CURRENT_Q_DEPTH.

O programa executa, então, um loop que exibe o menu e processa a seleção que o usuário faz. O loop é interrompido quando o usuário pressiona a tecla PF3. Quando uma seleção válida é feita, o programa adequado é iniciado; caso contrário, uma mensagem de erro será exibida.

z/OS *Programas get-mail e display-message no z/OS*

Nas versões do TSO do aplicativo, as funções get-mail e display-message são executadas pelo mesmo programa (CSQ4TVD2). Na versão CICS do aplicativo, essas funções são executadas por programas separados (CSQ4CVD2 e CSQ4CVD3).

O painel Mail Awaiting (CSQ4VDP2 para TSO, VD2 para CICS; consulte Figura 154 na página 1216 para um exemplo) mostra todas as mensagens que estão na fila de mensagens do usuário. Para criar essa lista, o programa usa a chamada MQGET para procurar todas as mensagens na fila, salvando informações sobre cada uma. Além das informações exibidas, o programa registra `MsgId` e `CorrelId` de cada mensagem.

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

Figura 154. Exemplo de um painel que mostra uma lista de mensagens em espera

No painel Mail Awaiting, o usuário pode selecionar uma mensagem e exibir os conteúdos da mensagem (consulte Figura 155 na página 1217 para obter um exemplo). O programa usa a chamada MQGET para remover essa mensagem da fila, usando `MsgId` e `CorrelId` que o programa observou ao procurar em todas as mensagens. Essa chamada MQGET é executada usando a opção MQGMO_SYNCPOINT. O programa exibe o conteúdo da mensagem, em seguida, declara um ponto de sincronização. Isso confirma a chamada MQGET, portanto, a mensagem agora não existe mais.

```
--------------------- IBM MQ for z/OS Sample Programs ---------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
----------------------------------- Message ------------------------------
| HELLO FROM JOHNJ                                |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
'-------------------------------------------------------------------------'
```

Figura 155. Exemplo de um painel mostrando os conteúdos de uma mensagem

Uma extensão óbvia para a função fornecida pelo Mail Manager é dar ao usuário a opção de deixar a mensagem na fila após visualizar seus conteúdos. Para fazer isso, você teria que voltar a chamada MQGET que remove a mensagem da fila, após exibir a mensagem.

z/OS *Programa sendmail no z/OS*

Quando o usuário concluir o painel Enviar E-Mail (CSQ4VDP4 para TSO, VD4 para CICS), o programa enviar-e-mail (CSQ4TVD4 para TSO, CSQ4CVD4 para CICS) coloca a mensagem na fila de e-mail do destinatário.

Para fazer isso, o programa usa a chamada MQPUT1. O destino da mensagem depende de como o usuário preencheu os campos no painel Send Mail:

- Se o usuário tiver especificado somente um nome de usuário, o primeiro programa assumirá que o nome seja um apelido e enviará a mensagem para o objeto definido por esse nome. Se não houver tal apelido, o programa tentará enviar a mensagem para uma fila local com esse nome.
- Se o usuário tiver especificado um nome de usuário e um nome de gerenciador de filas, o programa enviará a mensagem para a fila de mensagens definida por esses dois nomes.

O programa não aceita mensagens em branco e remove espaços em branco à esquerda de cada linha do texto da mensagem.

Se a chamada MQPUT1 for bem-sucedida, o programa exibe uma mensagem que mostra o nome do usuário e o nome do gerenciador de filas no qual a mensagem foi colocada. Se a chamada for mal sucedida, o programa verifica especificamente os códigos de razão que indicam que a fila ou o gerenciador de filas não existe; eles são MQRC_UNKNOWN_OBJECT_NAME e MQRC_UNKNOWN_OBJECT_Q_MGR. O programa exibe sua própria mensagem de erro para cada um desses erros; para outros erros, o programa exibe os códigos de conclusão e razão retornados pela chamada.

z/OS *Programa de apelido no z/OS*

Quando o usuário define um apelido, o programa de apelido (CSQ4TVD5 para TSO, CSQ4CVD5 para CICS) cria uma fila que tem o apelido como parte de seu nome.

O programa faz isso colocando uma mensagem na fila de entrada de comandos do sistema. A mensagem contém o comando DEFINE QALIAS ou DEFINE QREMOTE do IBM MQ for z/OS. O tipo de fila que o programa cria depende de como o usuário preencheu os campos no painel Create Nickname (CSQ4VDP5 para TSO, VD5 para CICS):

- Se o usuário tiver especificado somente um nome de usuário ou o nome do gerenciador de filas for o mesmo que o do gerenciador de filas com o qual o Mail Manager está conectado, o programa cria uma fila de alias.
- Se o usuário tiver especificado um nome de usuário e um nome de gerenciador de filas (e o gerenciador de filas não for aquele ao qual o Mail Manager está conectado), o programa cria uma definição local de uma fila remota. O programa não verifica a existência da fila para a qual essa definição resolve, nem mesmo se o gerenciador de filas remotas existe.

O programa também cria uma fila dinâmica temporária para manipular respostas da fila de entrada de comandos do sistema.

Se o gerenciador de filas não puder criar a fila de apelidos por uma razão que o programa espere (por exemplo, a fila já existe), o programa exibe sua própria mensagem de erro. Se o gerenciador de filas não puder criar a fila por uma razão que o programa não espere, o programa exibe até duas das mensagens de erro que são retornadas para o programa pelo servidor de comandos.

Nota: Para cada apelido, o programa de apelido cria somente uma fila de alias ou uma definição local de uma fila remota. As filas locais para as quais esses nomes de filas resolvem são criadas somente quando o ID do usuário que está contido no apelido é usado para iniciar o aplicativo Mail Manager.

z/OS *A amostra de verificação de crédito no z/OS*

O aplicativo de amostra Credit Check é um conjunto de programas que demonstra como usar muitos dos recursos fornecidos pelo IBM MQ for z/OS. Ele mostra como os vários programas de componentes de um aplicativo podem transmitir mensagens um para o outro usando técnicas de enfileiramento de mensagens.

A amostra pode ser executada como um aplicativo do CICS independente. No entanto, para demonstrar como projetar um aplicativo de enfileiramento de mensagens que usa os recursos fornecidos por ambos os ambientes CICS e IMS, um módulo também é fornecido como um programa de processamento de mensagens em lote do IMS. Essa extensão para a amostra é descrita em "A extensão IMS para a amostra de verificação de crédito no z/OS" na página 1229.

Também é possível executar a amostra em mais de um gerenciador de filas e enviar mensagens entre cada instância do aplicativo. Para fazer isso, consulte "A amostra de verificação de crédito com diversos gerenciadores de filas no z/OS" na página 1229.

Os programas CICS são entregues em C e COBOL. O programa único IMS é entregue apenas em C. Os conjuntos de dados fornecidos são mostrados em Tabela 171 na página 1197 e Tabela 173 na página 1199.

O aplicativo demonstra um método de avaliação de risco quando os clientes do banco solicitam empréstimos. O aplicativo mostra como um banco poderia funcionar de duas maneiras para processar solicitações de empréstimo:

- Ao lidar diretamente com um cliente, a equipe do banco deseja acesso imediato a informações de conta e de risco de crédito.
- Ao lidar com aplicativos gravados, a equipe do banco pode enviar uma série de solicitações para obter informações de conta e de risco de crédito e lidar com as respostas em um momento posterior.

O detalhes financeiros e de segurança no aplicativo foram mantidos simples para que as técnicas de enfileiramento de mensagens estejam claras.

z/OS *Preparando e executando a amostra de verificação de crédito no z/OS*

Para preparar e executar a amostra Credit Check, execute as etapas a seguir:

1. Crie o conjunto de dados VSAM que contém informações sobre algumas contas de exemplo. Faça isso editando e executando a JCL fornecida no conjunto de dados CSQ4FILE.
2. Execute as etapas em "Preparando os aplicativos de amostra para o ambiente do CICS no z/OS" na página 1194. (As etapas adicionais que devem ser executadas se desejar usar a extensão do IMS para

a amostra estão descritas em “A extensão IMS para a amostra de verificação de crédito no z/OS” na página 1229.)

3. Inicie o monitor acionador CKTI (fornecido com o IBM MQ for z/OS) com relação à fila CSQ4SAMP.INITIATION.QUEUE, usando a transação do CICS CKQC.
4. Para iniciar o aplicativo, inicie seu sistema CICS e use a transação MVB1.
5. Selecione consulta **Immediate** ou **Batch** no primeiro painel.

   Os painéis de consulta imediata e em lote são semelhantes; Figura 156 na página 1219 mostra o painel Immediate Inquiry.

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . _______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : ____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

Figura 156. Painel Immediate Inquiry para o aplicativo de amostra Credit Check

6. Insira um número de conta e valor de empréstimo nos campos apropriados. Consulte “Inserindo informações nos painéis de consulta” na página 1219 para orientação sobre quais informações devem ser inseridas nesses campos.

Inserindo informações nos painéis de consulta

O aplicativo de amostra Credit Check verifica se os dados inseridos no campo **Amount requested** dos painéis de consulta está no formato de números inteiros.

Se você inserir um dos números de conta a seguir, o aplicativo localiza o nome da conta apropriado, o saldo médio da conta e o índice de merecimento de crédito no conjunto de dados VSAM CSQ4BAQ:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

É possível inserir qualquer, ou nenhuma, informação nos outros campos. O aplicativo retém todas as informações inseridas e retorna as mesmas informações nos relatórios que gera.

z/OS *Design da amostra de verificação de crédito no z/OS*

Esta seção descreve o design de cada um dos programas que compõem o aplicativo de amostra Credit Check.

Para obter mais informações sobre de algumas das técnicas que foram consideradas durante o design do aplicativo, consulte “Considerações de design para a amostra de verificação de crédito no z/OS” na página 1226.

Figura 157 na página 1221 mostra os programas que compõem o aplicativo e também as filas que esses programas atendem. Nesta figura, o prefixo CSQ4SAMP foi omitido de todos os nomes de filas para tornar a figura mais fácil de entender.

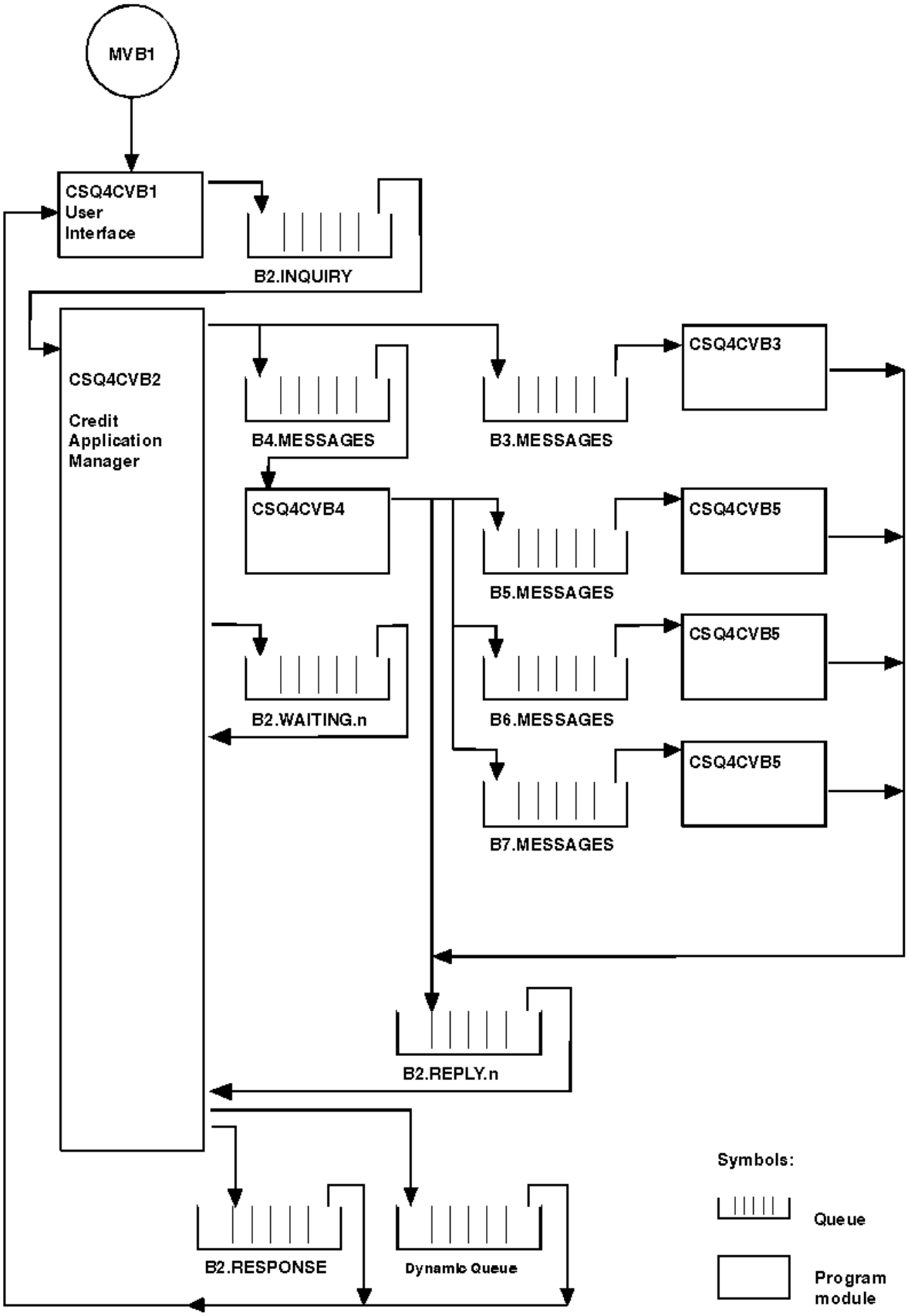


Figura 157. Programas e filas para o aplicativo de amostra Credit Check (somente programas COBOL)

z/OS *Programa de interface com o usuário (CSQ4CVB1) no z/OS*

Ao iniciar a transação do modo de conversação CICS MVB1, este inicia o programa de interface com o usuário para o aplicativo.

Esse programa coloca mensagens de consulta na fila CSQ4SAMP.B2.INQUIRY e obtém respostas a essas consultas de uma fila de resposta que especifica quando faz a consulta. A partir da interface com o usuário, é possível enviar consultas imediatas ou em lote:

- Para consultas imediatas, o programa cria uma fila dinâmica temporária que usa como uma fila de resposta. Isso significa que cada consulta tem sua própria fila de resposta.
- Para consultas em lote, o programa de interface com o usuário recebe respostas da fila CSQ4SAMP.B2.RESPONSE. Para simplificar, o programa obtém respostas para todas as suas consultas a partir de uma única fila de resposta. É fácil imaginar que um banco possa desejar usar uma fila de resposta separada para cada usuário de MVB1, de forma que cada um deles possa ver somente respostas para consultas que eles iniciaram.

Diferenças importantes entre as propriedades de mensagens usadas no aplicativo quando no modo em lote e imediato são:

- Para trabalho em lote, as mensagens têm uma baixa prioridade, portanto, elas são processadas após quaisquer solicitações de empréstimo inseridas no modo imediato. Além disso, as mensagens são persistentes, portanto, elas serão recuperadas se o aplicativo ou o gerenciador de filas precisar ser reiniciado.
- Para trabalho imediato, as mensagens têm uma alta prioridade, portanto, elas são processadas antes de quaisquer solicitações de empréstimo inseridas no modo em lote. Além disso, as mensagens não são persistentes, portanto, elas serão descartadas se o aplicativo ou o gerenciador de filas precisar ser reiniciado.

No entanto, em todos os casos, as propriedades de mensagens de solicitação de empréstimo são propagadas por todo o aplicativo. Portanto, por exemplo, todas as mensagens que resultam de uma solicitação de alta prioridade também terão uma alta prioridade.

z/OS *Gerenciador de aplicativo de crédito (CSQ4CVB2) no z/OS*

O programa Credit Application Manager (CAM) executa a maior parte do processamento para o aplicativo Credit Check.

O CAM é iniciado pelo monitor acionador CKTI (fornecido com IBM MQ for z/OS) quando ocorre um evento acionador em qualquer fila CSQ4SAMP.B2.INQUIRY ou fila CSQ4SAMP.B2.REPLY. *n*, em que *n* é um número inteiro que identifica uma de um conjunto de filas de respostas. A mensagem do acionador contém dados que incluem o nome da fila na qual o evento acionador ocorreu.

O CAM usa as filas com nomes de formato CSQ4SAMP.B2.WAITING.n para armazenar informações sobre consultas que está processando. As filas são denominadas de forma que sejam colocadas em pares com uma fila de resposta; por exemplo, a fila CSQ4SAMP.B2.WAITING.3 contém os dados de entrada para uma consulta específica e a fila CSQ4SAMP.B2.REPLY.3 contém um conjunto de mensagens de resposta (de programas que consultam bancos de dados), todas relacionadas a essa mesma consulta. Para entender as razões por trás desse design, consulte “Filas de consulta e resposta separadas no CAM” na página 1227.

Lógica de inicialização

Se o evento acionador ocorrer na fila CSQ4SAMP.B2.INQUIRY, o CAM abre a fila para acesso compartilhado. Em seguida, tenta abrir cada fila de resposta até que uma livre seja localizada. Se não puder localizar uma fila de resposta livre, a CAM registra o fato e é finalizado normalmente.

Se o evento acionador ocorrer na fila CSQ4SAMP.B2.REPLY.n, o CAM abre a fila para acesso exclusivo. Se o código de retorno relatar que o objeto já está em uso, o CAM é finalizado normalmente. Se algum outro erro ocorrer, o CAM registra o erro e é finalizado. O CAM abre a fila de espera correspondente e a fila de consulta, em seguida, inicia a obtenção e o processamento de mensagens. A partir da fila de espera, os CAM recupera detalhes das consultas parcialmente concluídas.

Por questão de simplicidade nesta amostra, os nomes das filas usadas são mantidos no programa. Em um ambiente de negócios, os nomes de filas provavelmente seriam mantidas em um arquivo acessado pelo programa.

Obtendo uma mensagem da fila de consulta

O CAM primeiro tenta obter uma mensagem da fila de consulta usando a chamada MQGET com a opção MQGMO_SET_SIGNAL. Se uma mensagem estiver disponível imediatamente, a mensagem será processada; se nenhuma mensagem estiver disponível, um sinal será configurado.

O CAM tenta, então, obter uma mensagem da fila de resposta, novamente usando a chamada MQGET com a mesma opção. Se uma mensagem estiver disponível imediatamente, a mensagem será processada; caso contrário, um sinal será configurado.

Quando ambos os sinais estiverem configurados, o programa espera até que um sinal seja postado. Se um sinal for postada para indicar que uma mensagem está disponível, a mensagem será recuperada e processada. Se o sinal expirar ou o gerenciador de filas estiver sendo finalizado, o programa será finalizado.

Processando a mensagem recuperada pelo CAM

Uma mensagem recuperada pelo CAM pode ser um de quatro tipos:

- Uma mensagem de consulta
- Uma mensagem de resposta
- Uma mensagem de propagação
- Uma mensagem inesperada ou indesejada

O CAM processa essas mensagens conforme descrito em “Processando a mensagem recuperada pelo CAM no z/OS” na página 1224.

Enviando uma resposta

Quando o CAM tiver recebido todas as respostas que está esperando para uma consulta, ele processa a resposta e cria uma mensagem de resposta única. Ele consolida em uma mensagem todos os dados de todas as mensagens de resposta que têm o mesmo `CorrelId`. Essa resposta é colocada na fila de resposta especificada na solicitação de empréstimo original. A mensagem de resposta é colocada na mesma unidade de trabalho que contém a recuperação da mensagem de resposta definitiva. Isso é para simplificar a recuperação, garantindo que nunca haja uma mensagem concluída na fila CSQ4SAMP.B2.WAITING.n.

Recuperação de consultas parcialmente concluídas

O CAM copia para a fila CSQ4SAMP.B2.WAITING.n todas as mensagens recebidas. Ele configura os campos do descritor de mensagens desta forma:

- *`Priority`* é determinado pelo tipo de mensagem:
  - Para mensagens de solicitação, priority = 3
  - Para datagramas, priority = 2
  - Para mensagens de resposta, priority = 1
- *`CorrelId`* é configurado para o *`MsgId`* da mensagem de solicitação de empréstimo
- Outros campos de MQMD são copiados daqueles da mensagem recebida

Quando uma consulta tiver sido concluída, as mensagens para uma consulta específica são removidas da fila de espera durante o processamento de resposta. Portanto, a qualquer momento, a fila de espera contém todas as mensagens relevantes para consultas em andamento. Essas mensagens são usadas para recuperar detalhes das consultas em andamento se o programa precisar ser reiniciado.

As prioridades diferentes são configuradas de forma que as mensagens de consulta sejam recuperadas antes de propagações ou mensagens de resposta.

z/OS *Processando a mensagem recuperada pelo CAM no z/OS*

Uma mensagem recuperada pelo Credit Application Manager (CAM) pode ser um de quatro tipos. A maneira na qual o CAM processa uma mensagem depende de seu tipo.

Uma mensagem recuperada pelo CAM pode ser um de quatro tipos:

- Uma mensagem de consulta
- Uma mensagem de resposta
- Uma mensagem de propagação
- Uma mensagem inesperada ou indesejada

O CAM processa essas mensagens da seguinte forma:

Mensagem de consulta

Mensagens de consulta vêm do programa de interface com o usuário. Ele cria uma mensagem de consulta para cada solicitação de empréstimo.

Para todas as solicitações de empréstimo, o CAM solicita o saldo médio da conta corrente do cliente. Ele faz isso colocando uma mensagem de solicitação na fila de alias CSQ4SAMP.B2.OUTPUT.ALIAS. Esse nome da fila é resolvido para fila CSQ4SAMP.B3.MESSAGES, que é processada pelo programa de conta corrente, CSQ4CVB3. Quando o CAM coloca uma mensagem nessa fila de alias, ele especifica a fila CSQ4SAMP.B2.REPLY.n apropriada para a fila de resposta. Uma fila de alias é usada aqui para que o programa CSQ4CVB3 possa facilmente ser substituído por outro programa que processe uma fila base com um nome diferente. Para fazer isso, você redefine a fila de alias para que seu nome seja resolvido para a nova fila. Além disso, você poderia designar diferentes autoridades de acesso à fila de alias e à fila base.

Se um usuário solicitar um empréstimo que seja maior que 10000 unidades, o CAM inicia verificações em outros bancos de dados também. Ele faz isso colocando uma mensagem de solicitação na fila CSQ4SAMP.B4.MESSAGES, que é processada pelo programa de distribuição, CSQ4CVB4. O processo que atende essa fila propaga a mensagem para filas atendidas por programas que têm acesso a outros registros, como histórico do cartão de crédito, contas de poupança e pagamentos de hipoteca. Os dados desses programas são retornados à fila de resposta especificada na operação put. Além disso, uma mensagem de propagação é enviada à fila de resposta por esse programa para especificar quantas mensagens de propagação foram enviadas.

Em um ambiente de negócios, o programa de distribuição provavelmente reformataria os dados fornecidos para corresponder ao formato necessário para cada um dos outros tipos de conta bancária.

Qualquer uma das filas referidas podem estar em um sistema remoto.

Para cada mensagem de consulta, o CAM inicia uma entrada na Inquiry Record Table (IRT) residente na memória. Esse registro contém:

- O `MsgId` da mensagem de consulta
- No campo ReplyExp, o número de respostas esperadas (igual ao número de mensagens enviadas)
- No campo ReplyRec, o número de respostas recebidas (zero neste estágio)
- No campo PropsOut, uma indicação de se uma mensagem de propagação é esperada

O CAM copia a mensagem de resposta para a fila de espera com:

- `Priority` configurado para 3
- `CorrelId` configurado para o `MsgId` da mensagem de consulta
- Os outros campos do descritor de mensagens configurados para aqueles da mensagem de consulta

Mensagem de propagação

Uma mensagem de propagação contém o número de filas para as quais o programa de distribuição encaminhou a consulta. A mensagem é processada da seguinte forma:

1. Incluir no campo ReplyExp do registro apropriado na IRT o número de mensagens enviadas. Essa informação está na mensagem.
2. Incrementar em 1 o campo ReplyRec do registro na IRT.
3. Decrementar em 1, o campo PropsOut do registro na IRT.
4. Copiar a mensagem para a fila de espera. O CAM configura `Priority` para 2 e os outros campos do descritor de mensagens para aqueles da mensagem de propagação.

Mensagem de Resposta

Uma mensagem de resposta contém a resposta para uma das solicitações para o programa de conta corrente ou para um dos programas de consulta de agência. As mensagens de resposta são processadas da seguinte forma:

1. Incrementar em 1 o campo ReplyRec do registro na IRT.
2. Copiar a mensagem para a fila de espera com `Priority` configurado para 1 e os outros campos do descritor de mensagens configurados para aqueles da mensagem de resposta.
3. Se ReplyRec = ReplyExp e PropsOut = 0, configure a sinalização MsgComplete.

Outras mensagens

O aplicativo não espera outras mensagens. No entanto, o aplicativo pode receber mensagens transmitidas pelo sistema ou responder mensagens com um `CorrelIds` desconhecido.

O CAM coloca essas mensagens na fila CSQ4SAMP.DEAD.QUEUE, onde elas podem ser examinadas. Se essa operação put falhar, a mensagem será perdida e o programa continuará. Para obter mais informações sobre o design desta parte do programa, consulte "Como a amostra manipula mensagens inesperadas" na página 1227.

z/OS *Programa de conta corrente (CSQ4CVB3) no z/OS*

O programa de conta corrente é iniciado por um evento acionador na fila CSQ4SAMP.B3.MESSAGES. Após ter aberto a fila, este programa recebe uma mensagem da fila usando a chamada MQGET com a opção de espera e com o intervalo de espera configurado para 30 segundos.

O programa procura no conjunto de dados VSAM CSQ4BAQ o número da conta na mensagem de solicitação de empréstimo. Ele recupera o nome da conta correspondente, o saldo médio e o índice merecimento de crédito ou observa que o número da conta não está no conjunto de dados.

O programa então coloca uma mensagem de resposta (usando a chamada MQPUT1) na fila de resposta denominada na mensagem de solicitação de empréstimo. Para essa mensagem de resposta, o programa:

- Copia o `CorrelId` da mensagem de solicitação de empréstimo
- Usa a opção MQPMO_PASS_IDENTITY_CONTEXT

O programa continua a obter mensagens da fila até o intervalo de espera expirar.

z/OS *Programa de distribuição (CSQ4CVB4) no z/OS*

O programa de distribuição é iniciado por um evento acionador na fila CSQ4SAMP.B4.MESSAGES.

Para simular a distribuição da solicitação de empréstimo para outras agências que têm acesso a registros, como, histórico de cartão de crédito, contas de poupança e pagamentos de hipoteca, o programa coloca uma cópia da mesma mensagem em todas as filas na lista de nomes CSQ4SAMP.B4.NAMELIST. Há três dessas filas, com nomes do formulário CSQ4SAMP.B *n*.MESSAGES, em que *n* é 5, 6 ou 7. Em um aplicativo de negócios, as agências poderiam estar em locais separados, portanto, essas filas poderiam ser filas remotas. Se desejar modificar o aplicativo de amostra para mostrar isso, consulte "A amostra de verificação de crédito com diversos gerenciadores de filas no z/OS" na página 1229.

O programa de distribuição executa as etapas a seguir:

1. Na lista de nomes, obtém os nomes das filas que o programa deve usar. O programa faz isso usando a chamada MQINQ para consultar sobre os atributos do objeto da lista de nomes.
2. Abre essas filas e também CSQ4SAMP.B4.MESSAGES.
3. Executa o loop a seguir até que não haja mais mensagens na fila CSQ4SAMP.B4.MESSAGES:
   a. Obter uma mensagem usando a chamada MQGET com a opção de espera e com o intervalo de espera configurado para 30 segundos.
   b. Colocar uma mensagem em cada fila listada na lista de nomes, especificando o nome da fila CSQ4SAMP.B2.REPLY.n apropriado para a fila de resposta. O programa copia o *CorrelId* da mensagem de solicitação de empréstimo para estas mensagens de cópia e usa a opção MQPMO_PASS_IDENTITY_CONTEXT na chamada MQPUT.
   c. Enviar uma mensagem do datagrama para a fila CSQ4SAMP.B2.REPLY.n para mostrar quantas mensagens colocou com sucesso.
   d. Declarar um ponto de sincronização.

z/OS *Programa de consulta de agência (CSQ4CVB5/CSQ4CCB5) no z/OS*

O programa de consulta de agência é fornecido como um programa COBOL e um programa C. Ambos os programas têm o mesmo design. Isso mostra que programas de diferentes tipos podem facilmente coexistir dentro de um aplicativo IBM MQ e que os módulos do programa que compõem tal aplicativo podem ser facilmente substituídos.

Uma instância do programa é iniciada por um evento acionador em qualquer uma destas filas:

- Para o programa COBOL (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- Para programa C (CSQ4CCB5), fila CSQ4SAMP.B8.MESSAGES

Nota: Se deseja usar o programa C, deve-se alterar a definição da lista de nomes CSQ4SAMP.B4.NAMELIST para substituir a fila CSQ4SAMP.B7.MESSAGES por CSQ4SAMP.B8.MESSAGES. Para fazer isso, é possível usar um dos seguintes:

- As operações e os painéis de controle do IBM MQ for z/OS
- O comando ALTER NAMELIST
- O utilitário CSQUTIL

Após ter aberto a fila apropriada, esse programa obtém uma mensagem da fila usando a chamada MQGET com a opção de espera e com o intervalo de espera configurado para 30 segundos.

O programa simula a procura de banco de dados de uma agência procurando no conjunto de dados VSAM CSQ4BAQ o número da conta que foi passado na mensagem de solicitação. Em seguida, constrói uma resposta que inclui o nome da fila que está servindo e um índice de merecimento de credito. Para simplificar o processamento, o índice de merecimento de crédito é selecionado aleatoriamente.

Ao efetuar put a mensagem de resposta, o programa usa a chamada MQPUT1 e:

- Copia o `CorrelId` da mensagem de solicitação de empréstimo
- Usa a opção MQPMO_PASS_IDENTITY_CONTEXT

O programa envia a mensagem de resposta à fila de resposta denominada na mensagem de solicitação de empréstimo. (O nome do gerenciador de filas que possui a fila de resposta também é especificado na mensagem de solicitação de empréstimo.)

z/OS *Considerações de design para a amostra de verificação de crédito no z/OS*

Considerações de design para a amostra Credit Check.

Este tópico contém informações sobre:

Filas de consulta e resposta separadas no CAM

O aplicativo poderia usar uma única fila para consultas e respostas, mas foi projetado para usar filas separadas pelas razões a seguir:

- Quando o programa estiver manipulando o número máximo de consultas, consultas adicionais podem ser deixadas na fila. Se uma fila única estiver sendo usada, isso teria de ser retirado da fila e armazenado em outro local.
- Outras instâncias do CAM poderiam ser iniciadas automaticamente para servir a mesma fila de consulta se o tráfego de mensagem estivesse alto o suficiente para garantir isso. Mas o programa deve acompanhar consultas em andamento e, para fazer isso, deve receber todas as respostas às consultas que iniciou. Se somente uma fila for usada, o programa precisaria procurar nas mensagens para ver se eram para esse programa ou para outro. Isso tornaria a operação muito menos eficiente.

  O aplicativo pode suportar diversos CAMs e pode recuperar soluções em andamento efetivamente usando filas de resposta e de espera em pares.
- O programa pode esperar em várias filas efetivamente usando sinalização.

Como a amostra manipula erros

O programa de interface com o usuário manipula erros relatando-os diretamente ao usuário.

Os outros programas não têm interfaces com o usuário, portanto, eles precisam manipular erros de outras maneiras. Além disso, em muitas situações (por exemplo, se uma chamada MQGET falhar), esses outros programas não conhecem a identidade do usuário do aplicativo.

Os outros programas colocam mensagens de erro em uma fila de armazenamento temporário do CICS chamada CSQ4SAMP. É possível procurar nessa fila usando a transação CEBR fornecida pelo CICS. Os programas também gravam mensagens de erro no log CSML do CICS.

Como a amostra manipula mensagens inesperadas

Ao projetar um aplicativo de enfileiramento de mensagens, deve-se decidir como manipular mensagens que chegam em uma fila inesperadamente.

As duas opções básicas são:

- O aplicativo não executa mais nenhum trabalho até ter processado a mensagem inesperada. Isso provavelmente significa que o aplicativo notifica um operador, se finaliza e assegura que não seja reiniciado automaticamente (ele pode fazer isso configurando o acionamento para desligado). Essa opção significa que todo o processamento para o aplicativo pode ser interrompido por uma única mensagem inesperada e a intervenção de um operador é necessária para reiniciar o aplicativo.
- O aplicativo remove a mensagem da fila que está atendendo, coloca a mensagem em outro local e continua o processamento. O melhor lugar para colocar essa mensagem é na fila de mensagens não entregues do sistema.

Se escolher a segunda opção:

- Um operador ou outro programa deve examinar as mensagens que são colocadas na fila de mensagens não entregues para descobrir de onde as mensagens são provenientes.
- Uma mensagem inesperada será perdida se não puder ser colocada na fila de mensagens não entregues.

- Uma mensagem inesperada longa será truncada se for maior que o limite de mensagens na fila de mensagens não entregues ou maior que o tamanho do buffer no programa.

Para assegurar que o aplicativo manipule suavemente todas as consultas com efeito mínimo de atividades externas, o aplicativo de amostra Credit Check usa a segunda opção. Para permitir que você mantenha a amostra separada de outros aplicativos que usam o mesmo gerenciador de filas, a amostra Credit Check não usa a fila de mensagens não entregues do sistema; em vez disso, usa sua própria fila de mensagens não entregues. Essa fila é denominada CSQ4SAMP.DEAD.QUEUE. A amostra trunca quaisquer mensagens maiores que a área de buffer fornecida para os programas de amostra. É possível usar o aplicativo de amostra Browse para procurar mensagens nessa fila ou usar o aplicativo de amostra do Print Message para imprimir as mensagens juntamente com seus descritores de mensagens.

No entanto, se você estender a amostra para execução em mais de um gerenciador de filas, mensagens inesperadas ou mensagens que não podem ser entregues poderiam ser colocadas na fila de mensagens não entregues do sistema pelo gerenciador de filas.

Como a amostra usa pontos de sincronização

Os programas no aplicativo de amostra Credit Check declaram pontos de sincronização para assegurar que:

- Somente uma mensagem de resposta seja enviada em resposta a cada mensagem esperada
- Várias cópias de mensagens inesperadas nunca sejam colocadas na fila de mensagens não entregues da amostra
- O CAM pode recuperar o estado de todas as consultas parcialmente concluídas obtendo mensagens persistentes a partir de sua fila de espera

Para conseguir isso, uma única unidade de trabalho é usada para cobrir a obtenção de uma mensagem, o processamento dessa mensagem e quaisquer operações put subsequentes.

Como a amostra usa informações de contexto da mensagem

Quando o programa da interface com o usuário (CSQ4CVB1) envia mensagens, ele usa a opção MQPMO_DEFAULT_CONTEXT. Isso significa que o gerenciador de filas gera informações de identidade e de contexto de origem. O gerenciador de filas obtém essas informações da transação que iniciou o programa (MVB1) e do ID do usuário que iniciou a transação.

Quando o CAM envia mensagens de consulta, ele usa a opção MQPMO_PASS_IDENTITY_CONTEXT. Isso significa que as informações de contexto de identidade da mensagem sendo colocadas são copiadas do contexto de identidade da mensagem de consulta original. Com essa opção, as informações de contexto de origem são geradas pelo gerenciador de filas.

Quando o CAM envia mensagens de resposta, ele usa a opção MQPMO_ALTERNATE_USER_AUTHORITY. Isso faz com que o gerenciador de filas use um ID de usuário alternativo para sua verificação de segurança quando o CAM abre uma fila de resposta. O CAM usa o ID do usuário do requisitante da mensagem original de consulta. Isso significa que os usuários têm permissão para ver as respostas para apenas aquelas consultas que eles originaram. O ID do usuário alternativo é obtido a partir das informações de contexto de identidade no descritor de mensagens da mensagem original de consulta.

Quando os programas de consulta (CSQ4CVB3/4/5) enviam mensagens de resposta, eles usam a opção MQPMO_PASS_IDENTITY_CONTEXT. Isso significa que as informações de contexto de identidade da mensagem sendo colocadas são copiadas do contexto de identidade da mensagem de consulta original. Com essa opção, as informações de contexto de origem são geradas pelo gerenciador de filas.

Nota: O ID do usuário associado às transações MVB3/4/5 requer acesso às filas B2.REPLY.n. Esses IDs de usuário podem não ser o mesmo que aqueles associados à solicitação que está sendo processada. Para contornar essa possível exposição de segurança, os programas de consulta poderiam usar a opção MQPMO_ALTERNATE_USER_AUTHORITY ao colocar suas respostas. Isso significaria que cada usuário individual do MVB1 precisa de autoridade para abrir as filas B2.REPLY.n.

Uso de identificadores de mensagem e correlação no CAM

O aplicativo tem de monitorar o progresso de todas as consultas em tempo real que ele está processando a qualquer momento. Para fazer isso, ele usa o identificador de mensagem exclusivo de cada mensagem de solicitação de empréstimo para associar todas as informações que tem sobre cada consulta.

O CAM copia o `MsgId` da mensagem de consulta para o `CorrelId` de todas as mensagens de solicitação que envia para essa consulta. Os outros programas na amostra (CSQ4CVB3 5) copiam o `CorrelId` de cada mensagem que eles recebem para o `CorrelId` de sua mensagem de resposta.

z/OS *A amostra de verificação de crédito com diversos gerenciadores de filas no z/OS*

É possível usar o aplicativo de amostra Credit Check para demonstrar o enfileiramento distribuído ao instalar a amostra em dois gerenciadores de filas e sistemas CICS (com cada gerenciador de filas conectado a um sistema CICS diferente).

Quando o programa de amostra é instalado e o monitor acionador (CKTI) está em execução em cada sistema, é necessário:

1. Configurar o link de comunicação entre os dois gerenciadores de filas. Para obter informações sobre como fazer isso, consulte Configurando enfileiramento distribuído.
2. Em um gerenciador de filas, crie uma definição local para cada uma das filas remotas (no outro gerenciador de filas) que deseja usar. Essas filas podem ser qualquer uma de CSQ4SAMP.B *n*.MESSAGES, em que *n* é 3, 5, 6 ou 7. (Essas são as filas que são atendidas pelo programa de conta corrente e o programa de consulta de agência.) Para obter informações sobre como fazer isso, consulte DEFINE QREMOTE e Filas DEFINE.
3. Mude a definição da lista de nomes (CSQ4SAMP.B4.NAMELIST) para que ela contenha os nomes das filas remotas que você deseja usar. Para obter informações sobre como fazer isso, consulte DEFINE NAMELIST.

z/OS *A extensão IMS para a amostra de verificação de crédito no z/OS*

Uma versão do programa de conta corrente é fornecido como um programa de processamento de mensagens em lote (BMP) do IMS. Ele é escrito na linguagem C.

O programa executa a mesma função que a versão do CICS, exceto que para obter as informações da conta, o programa lê um banco de dados do IMS em vez de um arquivo VSAM. Se você substituir a versão do CICS do programa de conta corrente pela versão do IMS, não verá diferença no método de uso do aplicativo.

Para preparar e executar a versão do IMS, deve-se:

1. Seguir as etapas em “Preparando e executando a amostra de verificação de crédito no z/OS” na página 1218.
2. Seguir as etapas em “Preparando o aplicativo de amostra para o ambiente do IMS no z/OS” na página 1198.
3. Alterar a definição da fila de alias CSQ4SAMP.B2.OUTPUT.ALIAS para resolver para a fila CSQ4SAMP.B3.IMS.MESSAGES (em vez de CSQ4SAMP.B3.MESSAGES). Para fazer isso, é possível usar um dos seguintes:
   - As operações e os painéis de controle do IBM MQ for z/OS
   - O comando ALTER QALIAS.

Outra maneira de usar o programa de conta corrente do IMS é fazer com que ele atenda uma das filas que recebe mensagens do programa de distribuição. No formulário entregue do aplicativo de amostra Credit Check, há três dessas filas (B5/6/7.MESSAGES), todas atendidas pelo programa de consulta de agência. Esse programa procura um conjunto de dados VSAM. Para comparar o uso do conjunto de dados VSAM e do banco de dados do IMS, você poderia fazer o programa de conta corrente do IMS atender uma dessas filas. Para fazer isso, deve-se mudar a definição da lista de nomes CSQ4SAMP.B4.NAMELIST para substituir uma das filas CSQ4SAMP.B *n*.MESSAGES com a fila CSQ4SAMP.B3.IMS.MESSAGES. É possível usar um dos seguintes:

- As operações e os painéis de controle do IBM MQ for z/OS
- O comando ALTER NAMELIST.

É possível executar, então, a amostra a partir da transação do CICS MVB1. O usuário não vê nenhuma diferença na operação ou resposta. O IMS BMP é interrompido após receber uma mensagem de parada ou após estar inativo por cinco minutos.

Design do programa de conta corrente do IMS (CSQ4ICB3)

Esse programa é executado como um BMP. Inicie o programa usando sua JCL antes de quaisquer mensagens do IBM MQ serem enviadas a ele.

O programa procura em um banco de dados do IMS o número da conta nas mensagens de solicitação de empréstimo. Ele recupera o nome de conta correspondente, o saldo médio e o índice merecimento de crédito.

O programa envia os resultados da procura do banco de dados para a fila de resposta denominada na mensagem do IBM MQ que está sendo processada. A mensagem retornada anexa o tipo de conta e os resultados da procura à mensagem recebida de forma que a transação que está construindo a resposta possa confirmar que a consulta correta está sendo processada. A mensagem está no formato de três grupos de 79 caracteres, conforme a seguir:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

Ao executar como um BMP orientado por mensagem, o programa drena a fila de mensagens do IMS, em seguida, lê mensagens da fila do IBM MQ for z/OS e processa as mesmas. Nenhuma informação é recebida da fila de mensagens do IMS. O programa reconecta ao gerenciador de filas após cada ponto de verificação porque os identificadores foram fechados.

Ao executar em um BMP orientado a lote, o programa continuará a ser conectado ao gerenciador de filas após cada ponto de verificação porque os identificadores não são fechados.

z/OS *A amostra Manipulador de mensagem no z/OS*

O aplicativo TSO da amostra do Message Handler permite procurar, encaminhar e excluir mensagens em uma fila. A amostra está disponível em C e em COBOL.

Preparando e executando a amostra

Siga estas etapas:

1. Preparar a amostra conforme descrito em “Preparando os aplicativos de amostra para o ambiente TSO no z/OS” na página 1192.
2. Customize o CLIST (CSQ4RCH1) fornecido para a amostra para definir o local dos painéis, o local do arquivo de mensagens e o local dos módulos de carregamento.

É possível usar CLIST CSQ4RCH1 para executar ambas as versões da amostra em C e em COBOL. A versão fornecida de CSQ4RCH1 executa a versão C e contém instruções sobre a customização necessária para a versão em COBOL.

Nota:

1. Não há definições de fila de amostra fornecidas com a amostra.
2. VS COBOL II não suporta multitarefas com o ISPF, portanto, não use o aplicativo de amostra Message Handler em ambos os lados de uma tela dividida. Se fizer isso, os resultados serão imprevisíveis.

z/OS *Usando a amostra de manipulador de mensagem no z/OS*

Tendo instalado a amostra e chamado a mesma a partir de CLIST CSQ4RCH1 customizado, a tela mostrada em Figura 158 na página 1231 será exibida.

```
----------------------- IBM MQ for z/OS -- Samples ------------------------
COMANDO ===>
User Id : JOHNJ



Enter information. Press ENTER :


Nome do Gerenciador de Filas: ________________________________________________:

Nome da Fila: _____________________________________________:









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Figura 158. Tela inicial da amostra Message Handler

Insira o gerenciador de filas e nome da fila a serem visualizados (distinção entre maiúsculas e minúsculas) e a tela da lista de mensagens será exibida (consulte Figura 159 na página 1231).

```
----------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

Figura 159. Tela da lista de mensagens para a amostra Message Handler

Essa tela mostra as primeiras 99 mensagens na fila e, para cada, mostra os campos a seguir:

Msg No
Número da mensagem

Put Date MM/DD/YYYY
Data em que a mensagem foi colocada na fila (GMT)

Put Time HH:MM:SS
Hora em que a mensagem foi colocada na fila (GMT)

Format Name
Campo MQMD.Format

UserIdentifier
Campo MQMD.UserIdentifier

Put Application Type
Campo MQMD.PutApplType

Put Application Name
Campo MQMD.PutApplName

O número total de mensagens na fila também é exibido.

Nessa tela uma mensagem pode ser escolhida, por número, não pela posição do cursor e, em seguida, exibida. Para obter um exemplo, consulte Figura 160 na página 1232.

```
---------------------- 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 :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                            `
PutApplType   : 000000002
PutApplName   : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000: C9C9 C840 0000 0001 0000 0054 0311 0311 ` IIH ............ `
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

Figura 160. Mensagem escolhida é exibida

Quando a mensagem tiver sido exibida, ela poderá ser excluída, deixada na fila ou encaminhada a outra fila. Os campos `Forward to Q Mgr` e `Forward to Queue` são inicializados com valores do MQMD, eles podem ser mudados antes de encaminhar a mensagem.

O plano de amostra permite que somente as mensagens com combinações exclusivas de `MsgId` / `CorrelId` sejam selecionadas e exibidas, porque a mensagem é recuperada usando o `MsgId` e o `CorrelId` como a chave. Se a chave não for exclusiva, a amostra não pode recuperar a mensagem escolhida com segurança.

Nota: Ao usar amostra SCSQCLST(CSQ4RCH1) para procurar mensagens, cada chamada faz com que a contagem de restauração da mensagem aumente. Se desejar mudar o comportamento desta amostra, copie a amostra e modifique o conteúdo conforme necessário. É necessário estar ciente de que outros aplicativos que dependem dessa contagem de restauração podem ser influenciados por esse aumento da contagem.

z/OS *Design da amostra de manipulador de mensagem no z/OS*

Este tópico descreve o design de cada um dos programas que compõem o aplicativo de amostra Message Handler.

Programa de validação de objeto

Solicita uma fila válida e o nome do gerenciador de filas.

Se você não especificar um nome do gerenciador de filas, o gerenciador de filas padrão será usado, se disponível. Somente filas locais podem ser usadas; um MQINQ é emitido para verificar o tipo de fila e um erro será relatado se a fila não for local. Se a fila não for aberta com sucesso ou a chamada MQGET for inibida na fila, as mensagens de erro são retornadas indicando o CompCode e o código de retorno Reason.

Programa de lista de mensagens

Exibe uma lista de mensagens em uma fila com informações sobre elas, como putdate, puttime e o formato da mensagem.

O número máximo de mensagens armazenado na lista é 99. Se houver mais mensagens na fila do que isso, a profundidade da fila atual também será exibida. Para escolher uma mensagem para exibição, digite o número da mensagem no campo de entrada (o padrão é 01). Se sua entrada não for válida, você receberá uma mensagem de erro apropriada.

Programa de conteúdo da mensagem

Exibe o conteúdo da mensagem.

O conteúdo é formatado e dividido em duas partes:

1. Descritor de Mensagens
2. Buffer de mensagem

O descritor de mensagens mostra os conteúdos de cada campo em uma linha separada.

O buffer de mensagem é formatado dependendo de seu conteúdo. Se o buffer contiver um cabeçalho de mensagens não entregues (MQDLH) ou um cabeçalho de fila de transmissão (MQXQH), eles serão formatados e exibidos antes do próprio buffer.

Antes dos dados do buffer serem formatados, uma linha de título mostra o comprimento do buffer da mensagem em bytes. O tamanho máximo do buffer é 32768 bytes e qualquer mensagem mais longa do que isso será truncada. O tamanho total do buffer é exibido junto com uma mensagem indicando que somente os primeiros 32768 bytes da mensagem são exibidos.

Os dados do buffer são formatados de duas maneiras:

1. Após o deslocamento no buffer ser impresso, os dados do buffer são exibidos em hexadecimal.
2. Os dados do buffer são, então, exibidos novamente como valores de EBCDIC. Se qualquer valor de EBCDIC não puder ser impresso, ele imprimirá um ponto (.).

É possível inserir D para excluir ou F para encaminhar no campo de ação. Se optar por encaminhar a mensagem, `forward-to queue` e `queue manager name` devem ser configurados corretamente. Os padrões para esses campos são lidos a partir dos campos ReplyToQ e ReplyToQMgr do descritor de mensagens.

Se encaminhar uma mensagem, qualquer bloco de cabeçalho armazenado no buffer será removido. Se a mensagem for encaminhada com sucesso, ela será removida da fila original. Se você inserir ações inválidas, mensagens de erro serão exibidas.

Um painel de ajuda de exemplo chamado CSQ4CHP9 também está disponível.

z/OS *A amostra de postagem assíncrona no z/OS*

O programa de amostra Asynchronous Put coloca mensagens em uma fila usando a chamada assíncrona MQPUT. A amostra também recupera as informações de status usando a chamada MQSTAT.

Os aplicativos de postagem assíncrona usam essas chamadas de MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

Os programas de amostra são fornecidos na linguagem de programação C.

Os aplicativos de postagem assíncrona são executados no ambiente de lote. Consulte Outras amostras para obter os aplicativos em lote.

Este tópico também fornece informações sobre o design do programa de consumo assíncromo e como executar a amostra CSQ4BCS2.

- “Executando a amostra CSQ4BCS2” na página 1234
- “Design do programa de amostra Asynchronous Put” na página 1234

Executando a amostra CSQ4BCS2

Este programa de amostra toma até seis 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, o CSQ4BCS2 se conectará ao gerenciador de filas padrão. O conteúdo da mensagem é fornecido por meio da entrada padrão (**SYSIN DD**).

Há um JCL de amostra para executar o programa, ele reside em CSQ4BCSP.

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 o programa não puder abrir a fila, o programa envia uma mensagem de erro que contenha o código de razão retornado pela chamada MQOPEN. Para manter o programa simples nesta e em subsequentes chamadas MQI, os valores padrão são usados para várias 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 contenha o texto dessa linha e coloca a mensagem na fila de destino de forma assíncrona. O programa continua até atingir o final da entrada ou até que a chamada MQPUT falhe. Se o programa atingir o final da entrada, ele fechará a fila usando a chamada MQCLOSE.

O programa, em seguida, emite a chamada MQSTAT, que retorna uma estrutura MQSTS e exibe mensagens que contenham o número de mensagens colocadas com êxito, o número de mensagens colocadas com um aviso, e o número de falhas.

Nota: Para observar o que acontece quando um erro MQPUT é detectado pela chamada MQSTAT, configure MAXDEPTH na fila de destino para um valor baixo.

z/OS *A amostra de consumo assíncrono em lote no z/OS*

O programa de amostra CSQ4BCS1 é entregue em C, ele demonstra o uso de MQCB e MQCTL para consumir mensagens de diversas filas de maneira assíncrona.

As amostras do Asynchronous Consumption são executadas no ambiente de lote. Consulte Outras amostras para obter os aplicativos em lote.

Também há uma amostra em COBOL que é executada no ambiente do CICS, consulte "A amostra de consumo assíncrono e publicar/assinar do CICS no z/OS" na página 1236.

Os aplicativos usam estas chamadas de MQI:

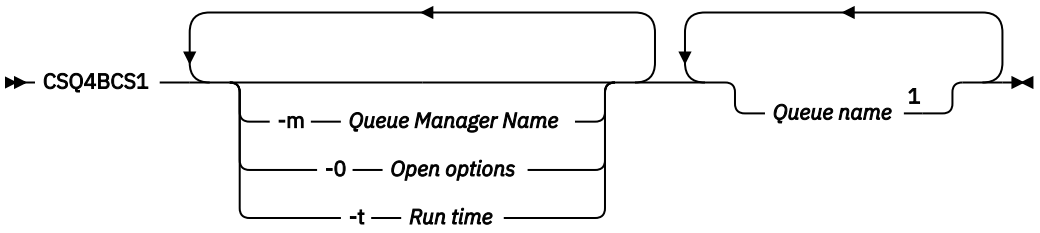
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

Este tópico também forneceu informações sobre os títulos a seguir:

- "Executando a amostra CSQ4BCS1" na página 1235
- "Design do programa de amostra Batch Asynchronous Consumption" na página 1235

Executando a amostra CSQ4BCS1

Esse programa de amostra segue a sintaxe a seguir:



Notas:

[1] No máximo 10 filas são suportadas por esta amostra. Forneça mais de um nome de fila para ler mensagens de várias filas.

Há uma JCL de amostra para executar este programa, ela reside em CSQ4BCSC.

Design do programa de amostra Batch Asynchronous Consumption

A amostra mostra como ler mensagens de diversas filas na ordem de chegada. Isso exigiria mais código usando MQGET síncrono. Com consumo assíncrono, nenhuma pesquisa é necessária e o gerenciamento de encadeamento e armazenamento é executado pelo IBM MQ. No programa de amostra, erros são gravados no console.

O código de amostra tem as etapas a seguir:

1. Defina a função de retorno de chamada de consumo de mensagem única.

```
void MessageConsumer(MQHCONN   hConn,
```

```
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Conecte-se ao gerenciador de filas.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Abra as filas de entrada e associe cada fila 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. É possível associar uma função de retorno de chamada diferente com cada fila.

4. Inicie o consumo das mensagens.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Espere até que o usuário pressione Enter, em seguida, pare o consumo de mensagens.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Por fim, desconecte-se do gerenciador de filas.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *A amostra de consumo assíncrono e publicar/assinar do CICS no z/OS*

Os programas de amostra Asynchronous Consumption and Publish/Subscribe demonstram o uso de consumo assíncrono e de recursos para publicar e assinar no CICS.

Um programa *Registration client* registra três manipuladores de Retorno de chamada (um identificador de evento e dois consumidores de mensagem) e inicia o Asynchronous Consumption. Um programa *Messaging client* coloca mensagens em uma fila ou publica mensagens adequadas de um console do CICS para consumo pelos dois Message Consumers (CSQ4CVCN e CSQ4CVCT).

Para fornecer controle de tempo de execução sobre o comportamento da amostra, um dos consumidores de mensagem pode ser instruído usando as mensagens que recebe, para SUSPEND, RESUME ou DEREGISTER qualquer um dos manipuladores de Retorno de chamada. Ele também pode ser usado para emitir MQCTL STOP para finalizar o Asynchronous Consumption sob o controle. O outro consumidor de mensagem é registrado para assinar um tópico.

Cada programa emite instruções COBOL DISPLAY nos pontos apropriados para exibir o comportamento da amostra.

Os aplicativos usam estas chamadas de MQI:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

Os programas são entregues na linguagem COBOL. Consulte Amostras de consumo assíncrono e de publicação/assinatura do CICS para os aplicativos CICS.

Este tópico também fornece as informações a seguir:

Instalação

Os nomes da Fila e do Tópico usados pelos Message Consumers são codificados permanentemente nos programas Registration e Messaging Client.

A Fila, **SAMPLE.CONTROL.QUEUE**, deve ser definida para o Gerenciador de filas associado à CICS região antes de executar a amostra. O Tópico, **News/Media/Movies**, pode ser definido se necessário ou será criado no tempo de execução sob o Administrative Object padrão se não existir.

Programas e definições de transações CICS podem ser instalados ao instalar um grupo: CSQ4SAMP.

Registration Client CSQ4CVRG

O programa Registration Client deve ser iniciado sob a transação CICS MVRG. Não aceita nenhuma entrada.

Quando iniciado, o Registration Client registra os manipuladores de Retorno de chamada a seguir usando MQCB:

- CSQ4CVEV como um Event Handler.
- CSQ4CVCN como um Message Consumer em um tópico, **News/Media/Movies**.
- CSQ4CVCT como um Message Consumer em uma Fila, **SAMPLE.CONTROL.QUEUE**.

O Registration Client passa uma estrutura de dados que contém os nomes de todos os três manipuladores de Retorno de chamada para CSQ4CVCT, juntamente com os identificadores de objeto associados aos dois consumidores de mensagem.

Tendo registrado os manipuladores de Retorno de chamada, o Registration Client emite um MQCTL START_WAIT para iniciar o Asynchronous Consumption e suspende até que o controle seja retornado a ele (por exemplo, por um dos manipuladores de Retorno de emitindo um MQCTL STOP).

Event Handler CSQ4CVEV

Quando acionado, o Event Handler exibe uma mensagem indicando o tipo de chamada (por exemplo, START). Quando acionado para o código de razão do IBM MQ CONNECTION_QUIESCING, o Event Handler emite um MQCTL STOP para finalizar o Asynchronous Consumption e retornar controle ao Registration Client.

Simple Message Consumer CSQ4CVCN

Quando acionado, esse Message Consumer exibe uma mensagem indicando o tipo de chamada (por exemplo, REGISTER). Quando acionado para o tipo de chamada MSG_REMOVED, o Message Consumer recupera a mensagem de entrada e a envia para o log de tarefas do CICS.

Control Message Consumer CSQ4CVCT

Quando acionado, esse Message Consumer exibe uma mensagem indicando o tipo de chamada (por exemplo, START). Quando acionado para o tipo de chamada MSG_REMOVED, o Message Consumer recupera a mensagem de entrada e a estrutura de dados passadas pelo Registration Client. Com base no conteúdo da mensagem, ele emite os comandos MQCB ou MQCTL apropriados para um dos seguintes:

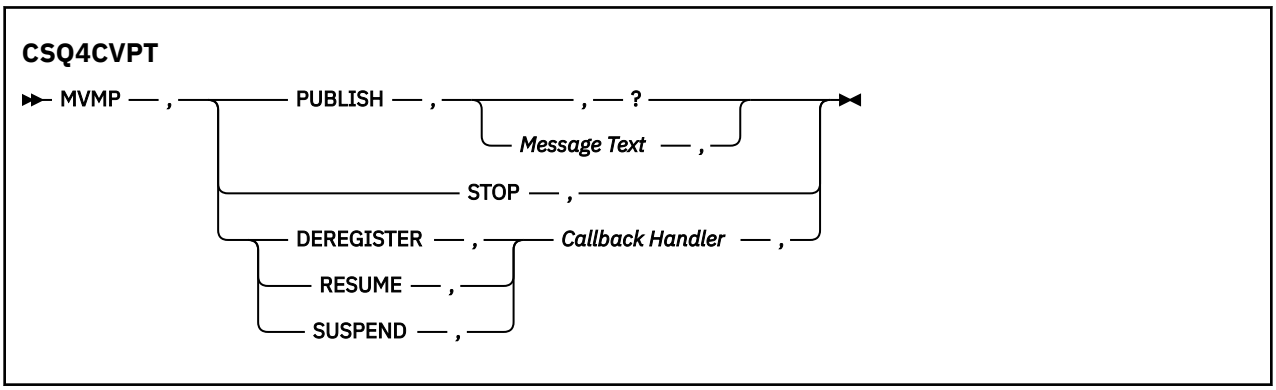
- STOP Asynchronous Consumption (retornando controle ao Registration Client).
- SUSPEND, RESUME ou DEREGISTER um manipulador de Retorno de chamada denominado (incluindo ele mesmo).

Messaging Client CSQ4CVPT

O Messaging Client tem duas funções:

- Ele publica uma mensagem em um tópico para consumo pelo Message Consumer CSQ4CVCN.
- Ele coloca uma mensagem de controle em uma fila para consumo pelo Control Message Consumer CSQ4CVCT, resultando em uma mudança em potencial no comportamento da amostra.

O programa Messaging Client deve ser iniciado a partir de um console do CICS sob uma transação do CICS e aceita entrada da linha de comandos com a sintaxe a seguir:



PUBLICAR
Publique o Message Text (ou uma mensagem padrão) como uma Retained Message para consumo pelo Simple Message Consumer.

PARAR
Pare o Asynchronous Consumption.

DEREGISTER
Cancele o registro do manipulador de Retorno de chamada.

RESUME
Continue o manipulador de Retorno de chamada.

SUSPEND
Suspenda o manipulador de Retorno de chamada.

Campos de entrada são posicionais e separados por vírgula. Palavras-chave e nomes de Manipulador de retorno de chamada não fazem distinção entre maiúsculas e minúsculas.

Exemplos

Tabela 175. Exemplos de entrada

exemplo	**Descrição**
MVMP,PUBLISH,,	Publicar uma mensagem padrão
MVMP, publicar, A `short message`	Publicar o texto fornecido
MVMP,STOP,	Parar consumo assíncrono

Tabela 175. Exemplos de entrada (continuação)	
exemplo	**Descrição**
MVMP,DEREGISTER,CSQ4CVEV,	Cancelar registro do manipulador de eventos
MVMP,resume,csq4cvcn,	Continuar o Simple Message Consumer
MVMP,SUSPEND,CSQ4CVEV,	Suspender o manipulador de eventos

Em que MVMP é a transação do CICS associada a CSQ4CVPT do programa Messaging Client.

Nota:

- Suspender ou cancelar o registro de todos os manipuladores de Retorno de chamada finaliza o START_WAIT emitido pelo Registration Client, retornando o controle a ele e encerrando a tarefa.
- Suspender ou cancelar o registro do Control Callback Handler não foi evitado deliberadamente, mas remove a capacidade de controlar ainda mais o comportamento da amostra.

z/OS *A amostra de publicar/assinar no z/OS*

Os programas de amostra Publish/Subscribe demonstram o uso dos recursos de publicação e assinatura no IBM MQ.

Há quatro programas de amostra de linguagem de programação C e dois COBOL que demonstram como programar a interface do IBM MQ Publish/Subscribe. Os programas são fornecidos na linguagem C e COBOL. Os aplicativos são executados no ambiente em lote; consulte Amostras Publish/Subscribe para os aplicativos em lote.

Há também amostras de COBOL executadas no ambiente CICS; consulte “A amostra de consumo assíncrono e publicar/assinar do CICS no z/OS” na página 1236.

Este tópico também fornece informações sobre como executar programas de amostra Publish/Subscribe. Esses programas de amostra incluem:

Executando a amostra CSQ4BCP1

Esse programa é escrito em C; ele publica mensagens em um tópico. Inicie uma das amostras de assinante antes de executar esse programa.

Esse programa usa até quatro parâmetros:

1. O nome da sequência de tópicos de destino (obrigatório).
2. O nome do gerenciador de filas (opcional).
3. Opções de abertura (opcional).
4. Opções de fechamento (opcional).

Se um gerenciador de filas não for especificado, CSQ4BCP1 se conectará ao gerenciador de filas padrão. Há uma JCL de amostra para executar o programa, ela reside em CSQ4BCPP.

O conteúdo da mensagem é fornecido por meio da entrada padrão (**SYSIN DD**).

Executando a amostra CSQ4BCP2

Esse programa é escrito em C; ele assina um tópico e imprime as mensagens recebidas.

Esse programa usa até três parâmetros:

1. O nome da sequência de tópicos de destino (obrigatório).
2. O nome do gerenciador de filas (opcional).
3. Opções de assinatura de MQSD (opcional).

Se um gerenciador de filas não for especificado, CSQ4BCP2 se conectará ao gerenciador de filas padrão. Há uma JCL de amostra para executar o programa, ela reside em CSQ4BCPS.

Executando a amostra CSQ4BCP3

Esse programa é escrito em C; ele assina um tópico usando uma fila de destino especificada pelo usuário e imprime as mensagens recebidas.

Esse programa usa até quatro parâmetros:

1. O nome da sequência de tópicos de destino (obrigatório).
2. O nome do destino (obrigatório).
3. O nome do gerenciador de filas (opcional).
4. Opções de assinatura de MQSD (opcional).

Se um gerenciador de filas não for especificado, CSQ4BCP3 se conectará ao gerenciador de filas padrão. Há uma JCL de amostra para executar o programa, ela reside em CSQ4BCPD.

Executando a amostra CSQ4BCP4

Esse programa é escrito em C; ele assina e recebe mensagens de um tópico, permitindo o uso de opções estendidas na chamada MQSUB, estendendo aquelas disponíveis na amostra MQSUB mais simples: CSQ4BCP2. Além da carga útil da mensagem, as propriedades de cada mensagem são recebidas e exibidas.

Esse programa aceita conjunto de variáveis de parâmetros:

- **-t** *Topic string* (obrigatório).
- **-o** *Topic object name* (obrigatório).
- **-m** *Queue manager name* (opcional).
- **-q** *Destination queue name* (opcional).
- **-w** *Wait interval on MQGET in seconds* (optional), em que *seconds* pode ter qualquer um dos valores a seguir:
  - unlimited: MQWI_UNLIMITED
  - none: nenhuma espera
  - *n*: intervalo de espera em segundos
  - Nenhum valor especificado: quando nenhum valor é especificado, o padrão é 30 segundos
- **-d** *Subscription name* (opcional). Cria ou continua a assinatura durável nomeada.
- **-k** (opcional). Mantém assinatura durável em MQCLOSE.

Se um gerenciador de filas não for especificado, CSQ4BCP4 se conectará ao gerenciador de filas padrão. Há uma JCL de amostra para executar o programa, ela reside em CSQ4BCPE.

Executando a amostra CSQ4BVP1

Esse programa é escrito em COBOL, ele publica mensagens em um tópico. Inicie uma das amostras de assinante antes de executar esse programa.

Esse programa não aceita parâmetros. **SYSIN DD** fornece o nome do tópico de entrada, o nome do gerenciador de filas e o conteúdo da mensagem.

Se um gerenciador de filas não for especificado, CSQ4BVP1 se conectará ao gerenciador de filas padrão. Há uma JCL de amostra para executar o programa, ela reside no CSQ4BVPP.

Executando a amostra CSQ4BVP2

Esse programa é escrito em COBOL, ele assina um tópico e imprime as mensagens recebidas.

Esse programa não aceita parâmetros. **SYSIN DD** fornece a entrada para o nome do tópico e o nome do gerenciador de filas.

Se um gerenciador de filas não for especificado, CSQ4BVP1 se conectará ao gerenciador de filas padrão. Há uma JCL de amostra para executar o programa, ela reside no CSQ4BVPP.

z/OS *A amostra de propriedade de mensagem Set and Inquire no z/OS*

Os programas de amostra de propriedade de mensagem demonstram a adição de propriedades definidas pelo usuário para um identificador de mensagens e a pesquisa das propriedades associadas a essa mensagem.

Os aplicativos usam estas chamadas de MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

Os programas são fornecidos na linguagem C. Os aplicativos são executados no ambiente em lote. Consulte Outras amostras para obter os aplicativos em lote.

O programa CSQ4BCM1 é usado para pesquisar as propriedades de um identificador de mensagens de uma fila de mensagens e é um exemplo do uso da chamada de API MQINQMP. A amostra obtém uma mensagem de uma fila e, em seguida, imprime todas as propriedades do identificador de mensagens.

O programa CSQ4BCM2 é usado para configurar as propriedades de um identificador de mensagens em uma fila de mensagens e é um exemplo do uso da chamada de API MQSETMP. A amostra cria um identificador de mensagens e o coloca no campo `MsgHandle` da estrutura MQGMO. Em seguida, ele coloca a mensagem em uma fila.

Outros exemplos de pesquisa e impressão de propriedades de mensagens estão incluídos nos programas de amostra CSQ4BCG1 e CSQ4BCP4.

Este tópico também fornece informações sobre a execução das amostras de propriedades de mensagens Set e Inquire sob os seguintes títulos:

- “Executando a amostra CSQ4BCM1” na página 1241
- “Executando a amostra CSQ4BCM2” na página 1242

Executando a amostra CSQ4BCM1

Esse programa usa até quatro 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).

Executando a amostra CSQ4BCM2

Este programa usa até seis 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).

Os nomes das propriedades, os valores e o conteúdo da mensagem são fornecidos por meio da entrada padrão (**SYSIN DD**). Há um JCL de amostra para executar o programa, ele reside em CSQ4BCMP.

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.

Informações relacionadas

MQ Telemetry

Casos de Uso de Telemetria

Instalando o MQ Telemetry

Administrando o MQ Telemetry

Referência do MQ Telemetry

Resolução de problemas 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 `/SDK/` e os conteúdos foram removidos:

- Apenas a origem do aplicativo de amostra (`mqttv3app.jar`) era 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.

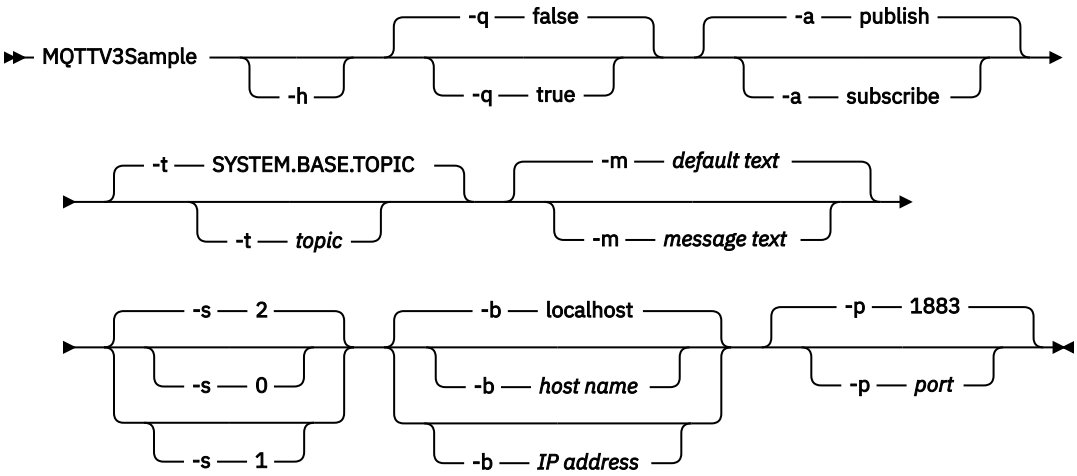
Programa MQTTV3Sample

Informações de referência sobre a sintaxe de amostra e parâmetros para o programa MQTTV3Sample.

Finalidade

O programa MQTTV3Sample pode ser usado para publicar uma mensagem e assinar um tópico.

Sintaxe MQTTV3Sample



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:

```
runMQTTV3Sample -a subscribe
```

Para publicar uma mensagem no Windows, use o comando:

```
runMQTTV3Sample
```

Para obter informações adicionais sobre como executar os scripts de amostra fornecidos, consulte “Programas de amostra doIBM MQ Telemetry Transport” na página 1242.

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 `/SDK/` e os conteúdos foram removidos:
    - Apenas a origem do aplicativo de amostra (`mqttv3app.jar`) era 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 as alterações mais recentes ao `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 1251. 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 retorna o controle para o cliente MQTT depois de ser chamado pelo método `deliveryComplete` , a entrega é 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 desreferencia o objeto `MqttMessage` quando a entrega é concluída e o objeto token de entrega quando o cliente se desconecta. 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 1248. 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 pela classe `MqttClientPersistence` antes da MQTT chamada do cliente `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 descarta suas referências ao tópico e à mensagem quando `messageArrived` retorna para o 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 `MqttCallback.deliveryComplete` retornar para o MQTT Cliente `token.waitForCompletion` continua. 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 176 na página 1247 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 176. 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 uso de `cleanSession=false` para `cleanSession=true`, todas as assinaturas anteriores para o cliente e quaisquer publicações que não foram recebidas serão descartados

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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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.

Na persistência de mensagem do MQTT há dois aspectos; como a mensagem será transferida e se ela está enfileirada no IBM MessageSight e 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 estiver marcada como `QoS` 1 ou 2, ela será enfileirada no IBM MessageSight e IBM MQ como uma mensagem persistente. Se estiver marcada como `QoS=0`, ela será enfileirada no IBM MessageSight e 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 Cliente MQTT para 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. Ele 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 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 uma Eclipse Rich Client Platform (RCP) ou um OSGi.
- `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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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`.

Publicando

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 ou IBM MessageSight é responsável por criar publicações para enviar para 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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:

- Executar 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 do IBM MQ pode remover uma assinatura.Listar assinaturas usando o IBM MQ Explorerou usando comandos **runmqsc** ou PCF Todas as assinaturas do cliente MQTT são nomeadas. Eles recebem um nome do 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 uso de `cleanSession=false` para `cleanSession=true`, todas as assinaturas anteriores para o cliente e quaisquer publicações que não foram recebidas serão descartados

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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 Cliente MQTT para 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.

Diferente da publicação/assinatura do IBM MQ, o protocolo `mqttv3` não terá um conceito de um objeto do 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 de tópicos do IBM MQ é descrita em Esquema de curinga baseado em tópico. A sintaxe dos filtros de tópicos que é possível construir com um cliente MQTT é descrita na classe `MqttClient` na documentação da API para o Cliente MQTT para 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 de estado de 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. Mude o modo de sessão limpa configurando `MqttConnectOptions.cleanSession` antes de se 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 do Microsoft Windows Communication Foundation (WCF) com o IBM MQ

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

Conceitos relacionados

“Introdução ao uso do canal customizado do IBM MQ para WCF com .NET 3” na página 1267
Visão geral das informações disponíveis para programadores que usam o canal customizado do IBM MQ para Windows Communication Foundation (WCF) com o .NET 3.

“Usando os canais customizados do IBM MQ para WCF” na página 1272
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 1292
As amostras do Windows Communication Foundation (WCF) fornecem alguns exemplos simples de como o canal customizado do IBM MQ pode ser usado.

“Determinação de problema no canal customizado do WCF para IBM MQ” na página 1298
É possível usar o rastreio do IBM MQ para coletar informações detalhadas sobre quais partes diferentes do código do IBM MQ está fazendo. Ao usar o Windows Communication Foundation (WCF), uma saída

de rastreio separada é gerada para o rastreio do canal customizado do WCF integrado com o rastreio de infra-estrutura do Microsoft WCF.

Introdução ao uso do canal customizado do IBM MQ para WCF com .NET 3

Visão geral das informações disponíveis para programadores que usam o canal customizado do IBM MQ para Windows Communication Foundation (WCF) com o .NET 3.

Qual é o canal customizado do IBM MQ para WCF?

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 1272.

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.

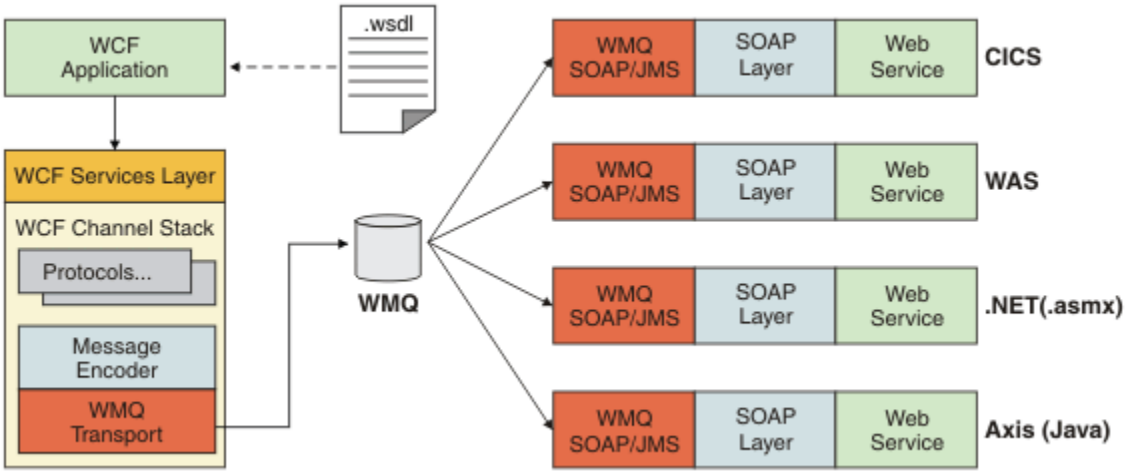
Os cenários típicos de uso padrão para o canal customizado do IBM MQ para o WCF são:

- Como uma interface para serviços da web hospedados no IBM MQ (SOAP sobre JMS).
- Como uma interface não SOAP para transmissão de mensagens nativas do IBM MQ.

Mensagens transportadas usando o SOAP no formato JMS

Um cenário do padrão de uso típico do canal customizado IBM MQ para o WCF é como uma interface para serviços da Web hospedados no IBM MQ (SOAP/JMS).

As mensagens são transportadas usando o SOAP no formato de mensagem JMS do IBM MQ, possibilitando que os clientes e serviços do WCF também chamem ou sejam chamados por outros aplicativos WebSphere Application Server ou ambientes de hospedagem que são compatíveis com este formato, incluindo serviços da Web e clientes em execução no CICS, Axis v1 (Java) e .asmx (.NET), conforme mostrado no diagrama a seguir:



Para obter detalhes sobre SOAP sobre JMS, consulte: “Desenvolvendo serviços da web com o transporte do IBM MQ para SOAP” na página 1307

Um exemplo de um cenário típico do diagrama seria:

1. Um serviço da Web hospedado no WebSphere Application Server e exposto no IBM MQ usando o suporte para SOAP no JMS no WebSphere Application Server.
2. O documento WSDL que descreve o serviço pode, então, ser usado pela ferramenta WCF para gerar um proxy de cliente e uma configuração que poderá criar uma pilha de canais do WCF apropriada, incluindo o canal customizado.
3. O aplicativo cliente pode, então, usar o proxy para iniciar o serviço da Web da mesma maneira que qualquer outro serviço da Web.

O canal, geralmente, pode ser usado com um codificador de mensagem de texto/SOAP do WCF, mas o canal pode ser unido a outros codificadores de mensagem do WCF se necessário. Usar codificadores alternativos pode também fornecer integração limitada com aplicativos nativos do IBM MQ que não suportam o SOAP no JMS, mas esta não é a função principal do canal.

Os principais benefícios de usar o canal customizado em um ambiente WCF são:

- Chamada assíncrona: Suportar o disparo e esquecer operações do cliente nas quais o cliente é separado da disponibilidade do serviço e dos recursos, como rotear novamente as respostas e o multi-hop.
- Características de escala confiável: Sistema de mensagens baseado em fila permite que a capacidade seja previsivelmente incluída em um sistema.
- Qualidade de serviço: As mensagens são tangíveis e rastreáveis e podem ser facilmente gerenciadas e administradas.

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 nas filas do IBM MQ MQ, esta mensagem poderá ser processada por qualquer serviço do WCF ou aplicativos não WCF como C ou aplicativos Java que estão em execução no Windows, UNIX ou z/OS.

Requisitos de software e instruções de instalação para o canal customizado do IBM MQ para WCF

Este tópico descreve os requisitos de software e informações de instalação 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 de software para o canal customizado do WCF para IBM MQ

Estas informações listam os requisitos de software para o canal customizado do WCF para IBM MQ.

Ambiente de tempo de execução

- Microsoft.NET Framework v3.5 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 8.0. Instala os conjuntos do .NET necessários para o canal customizado no Global Assembly Cache.

Nota: Se o Microsoft.NET Framework v3.5 ou superior não for instalado antes de instalar o IBM MQ 8.0 ou mais recente, então a instalação do produto IBM MQ continuará sem erro, mas o canal customizado do IBM MQ estará indisponível. Se o .NET Framework for instalado após a instalação do IBM MQ 8.0 ou mais recente, então o canal customizado do IBM MQ deverá ser ativado executando o script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd, em que *WMQInstallDir* é o diretório em que o IBM MQ 8.0 ou mais recente é instalado. Este script instala as montagens necessárias no Global Assembly Cache (GAC). Um conjunto de arquivos amqi*.log gravando as ações executadas é criado no diretório %TEMP%. Não será necessário executar novamente o script amqiRegisterdotNet.cmd se .NET for atualizado para v3.5 ou superior a partir de uma versão anterior, por exemplo, de .NET v2.0.

Ambiente de desenvolvimento

- Microsoft Visual Studio 2008 ou Windows Software Development Kit para o .NET 3.5 ou posterior.
- O Microsoft.NET Framework V3.5 ou superior deve ser instalado na máquina host para construir os arquivos de solução de amostra.

Nota: Se o Microsoft.NET Framework v3.5 ou superior não for instalado antes de instalar o IBM MQ 8.0 ou mais recente, então a instalação do produto IBM MQ continuará sem erro, mas o canal customizado do IBM MQ estará indisponível. Se o .NET Framework for instalado após a instalação do IBM MQ 8.0 ou mais recente, então o canal customizado do IBM MQ deverá ser ativado executando o script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd, em que *WMQInstallDir* é o diretório em que o IBM MQ 8.0 ou mais recente é instalado. Este script instala as montagens necessárias no Global Assembly Cache (GAC). Um conjunto de arquivos amqi*.log gravando as ações executadas é criado no diretório %TEMP%. Não será necessário executar novamente o script amqiRegisterdotNet.cmd se .NET for atualizado para v3.5 ou superior a partir de uma versão anterior, por exemplo, de .NET v2.0.

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 do IBM MQ 8.0 ou mais recente.

Canal customizado do IBM MQ para o WCF

O canal customizado do IBM MQ para WCF é instalado por padrão como parte da instalação do IBM MQ 8.0. O canal customizado e suas dependências estão contidos no componente `Java and .NET Messaging and Web Services`, que é instalado por padrão. Ao fazer upgrade para o IBM MQ 8.0 a partir de uma versão anterior, a atualização instala o canal customizado do IBM MQ para o WCF por padrão se o componente `Java and .NET Messaging and Web Services` foi instalado anteriormente 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 177. 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

V 9.0.5

No IBM MQ 9.0.5, 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"
```

Notes:

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 1292.

svcutil.exe.config

O `svcutil.exe.config` é um exemplo das definições de configuração necessárias para permitir que a ferramenta de geração de proxy do cliente Microsoft WCF `svcutil` reconheça 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 informações adicionais sobre como usar o `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 1289.

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:

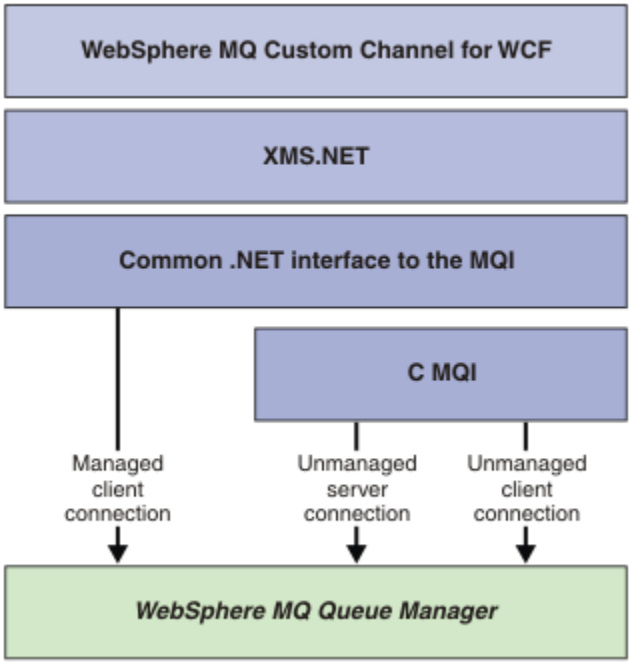


Figura 161. Arquitetura do WCF para a interface SOAP/JMS

Para o IBM WebSphere MQ 7.0.1 e posterior, 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 1278.

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:

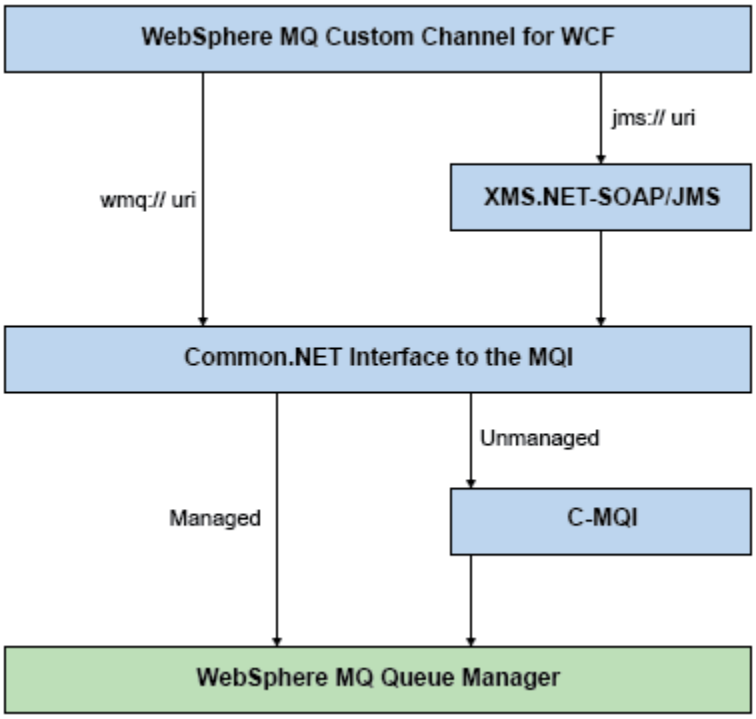


Figura 162. Arquitetura do WCF para a interface não SOAP/não JMS

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 WebSphere MQ 7.0 ou posterior pode ser usado como um canal customizado dentro do WCF no .NET Framework 3 da mesma forma que os canais integrados oferecidos pelo Microsoft.

As mensagens transportadas através do canal customizado são formatadas de acordo com a implementação do SOAP sobre JMS do IBM WebSphere MQ 7.0 ou mais recente. Os aplicativos podem então se comunicar com serviços hospedados pelo WCF ou pela infraestrutura de serviço WebSphere SOAP over JMS. Para obter mais informações sobre SOAP sobre JMS, consulte “Desenvolvendo serviços da web com o transporte do IBM MQ para SOAP” na página 1307.

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. Para obter a sintaxe deste parâmetro, veja Sintaxe de URI e parâmetros para a implementação de serviço da web.

initialContextFactory
O parâmetro initialContextFactory é necessário e deve ser configurado para "com.ibm.mq.jms.Nojndi" para ter compatibilidade com o WebSphere Application Server e outros produtos (consulte “Implementando um serviço ao WebSphere Application Server para usar o WebSphere Transport for SOAP” na página 1360).

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.

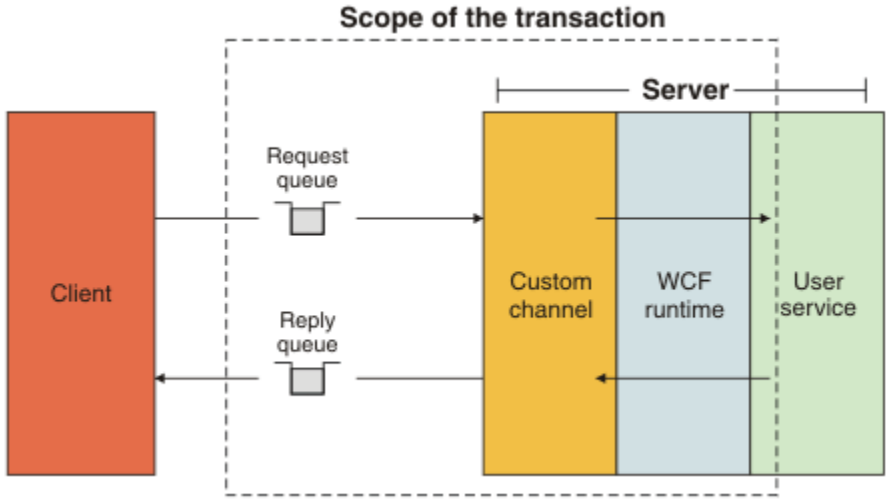
Uma mensagem de pedido é recebida e qualquer mensagem de resposta é enviada em um ponto de sincronização da transação local, que pode ser retrocedido no caso de falha de tempo de execução. Exemplos dessas falhas são: Uma exceção não manipulada lançada pelo serviço, falha para executar o dispatch da mensagem 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. Para obter informações adicionais sobre

como configurar propriedades do URI, veja: Sintaxe de URI e parâmetros para a implementação de serviço da web.

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 1276
- 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 1284
- 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.

O TLS pode ser especificado de uma das duas formas:

- Especifique o TLS diretamente no SOAP sobre o URI do JMS. Para obter uma descrição completa das opções do TLS, consulte TLS e o transporte do IBM MQ para SOAP
- 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 de 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.

Não é possível especificar a tabela de definição de canal diretamente no URI do SOAP sobre JMS. 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.

Informações relacionadas

Tabela de Definições 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 1274
- 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

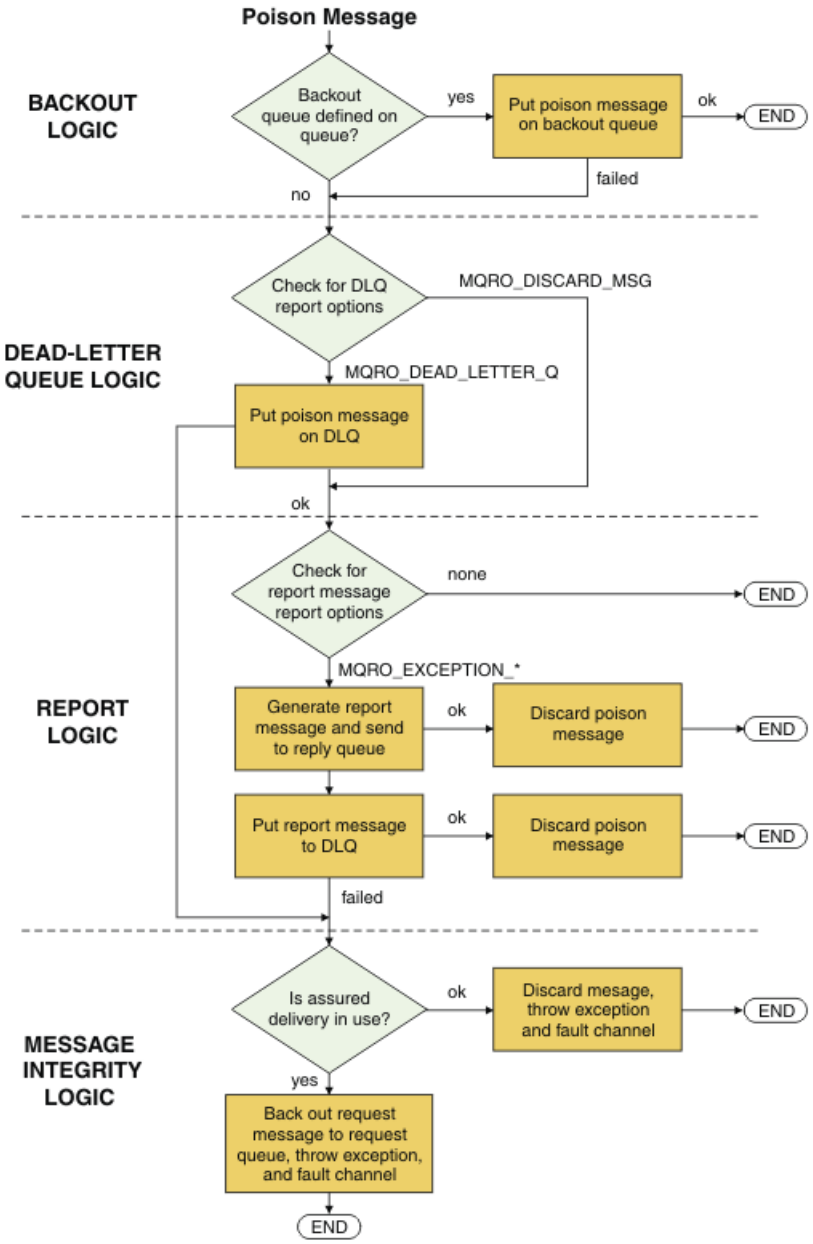
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

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 possíveis ações e as etapas realizadas se a manipulação de uma mensagem suspeita 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 550

Para obter informações adicionais sobre arquitetura WCF, veja: “Arquitetura do WCF” na página 1271

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 550.

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 de 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 1281
- Administrativamente, fornecendo os detalhes em um arquivo de configuração do aplicativo, conforme descrito no procedimento a seguir.

O arquivo de configuração do cliente ou do aplicativo de serviço é comumente denominado *yourappname*`.exe.config` em que *yourappname* é o nome de seu aplicativo. O arquivo de configuração do aplicativo é modificado mais facilmente usando a ferramenta do editor de configuração de serviço Microsoft chamada `SvcConfigEditor.exe` da seguinte maneira:

- Inicie a ferramenta do editor de configuração `SvcConfigEditor.exe`. O local de instalação 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 178. 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 179. 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 1280
2. Faça qualquer alteração necessária nos valores-padrão das propriedades conforme descrito em: "Opções de Configuração de Ligação do WCF" na página 1284

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 1280 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 180. Novos Campos de Terminal do Cliente	
Campo	**Value**
Nome	`Endpoint_WMQ`

Tabela 180. Novos Campos de Terminal do Cliente (continuação)	
Campo	**Value**
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 1283
Ligação	`customBinding`
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 181. 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 1283
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 1292.

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 1279.
- 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 1283.

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 &).

Para obter informações adicionais sobre como configurar propriedades de URI, consulte Sintaxe de URI e parâmetros para implementação de serviço da web

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. O formato de URI é conforme especificado para SupportPac 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 182. 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.

Tabela 182. 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
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 está conforme especificado no URI. 0 será o valor padrão se nenhum modo de conexão do cliente for configurado. 1 significa que o modo de conexão do cliente é um cliente não gerenciado. Usado somente se a conexão do cliente for usada.
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.

Tabela 182. 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	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 1274). 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 1279

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 espaço de nomes `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 1281

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 precisarem ser expostos (para que ferramentas como `svcutil` possam acessá-los diretamente a partir do serviço em execução, em vez de a partir 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

Os metadados estão agora 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, geralmente `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` para que svcutil reconheça o canal customizado do 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 do cliente do Microsoft Visual Studio.

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 1290

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.

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 1279

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 espaço de nomes `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 1281

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 1289

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 informações adicionais sobre como formatar o URI do terminal JMS, consulte *IBM MQ Transport for SOAP* na documentação de produto do 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 1289

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 informações adicionais sobre como formatar o URI do terminal JMS, consulte *IBM MQ Transport for SOAP* na documentação de produto do 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. Para obter informações sobre como instalar e configurar o ambiente, consulte: "Instalando o transporte da web do IBM MQ para SOAP" na página 1317

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 do IBM MQ, por exemplo: `C:\Program Files\IBM\MQ`
2. Assegure-se de que o compilador `javac` de Java esteja disponível e no PATH.
3. Copie o arquivo `axis.jar` do diretório `prereqs/axis` do CD de instalação do WebSphere para o diretório de produção do 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 em que 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 1290

A amostra foi configurada com nomes de recurso específicos conforme descrito no procedimento a seguir. Se for necessário mudar os nomes de recurso, então deve-se também mudar o valor correspondente no aplicativo cliente do 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 do 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 do 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. Para obter informações sobre como instalar e configurar o ambiente, consulte: "Instalando o transporte da web do IBM MQ para SOAP" na página 1317

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 do IBM MQ, por exemplo: `C:\Program Files\IBM\MQ`
2. Assegure-se de que o compilador `javac` de Java esteja disponível e no PATH.
3. Copie o arquivo `axis.jar` do diretório `prereqs/axis` do CD de instalação WebSphere no diretório de instalação IBM MQ.
4. Inclua no PATH: *MQ_INSTALLATION_PATH*`\Java\lib`, em que *MQ_INSTALLATION_PATH* representa o diretório em que 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 1290

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 do 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.

Referências relacionadas

"Manipulando Diferentes Nomes de Elemento de Resposta do SOAP" na página 1305

O WCF espera que o nome de um valor retornado esteja em um formato específico por padrão, mas um serviço pode não retornar um elemento com seu nome no formato esperado.

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 183. 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` do serviço da web no seu servidor de aplicativos do WebSphere da seguinte maneira:
   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 do 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 1289

A amostra foi configurada com nomes de recurso específicos conforme descrito no procedimento a seguir. Se você precisar mudar os nomes do recurso, 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. Para obter mais informações sobre a formatação do URI do terminal JMS, veja Sintaxe de URI e parâmetros para a implementação de serviço da web.

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 o 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.

Determinação de problema no canal customizado do WCF para IBM MQ

É possível usar o rastreio do IBM MQ para coletar informações detalhadas sobre quais partes diferentes do código do IBM MQ está fazendo. Ao usar o Windows Communication Foundation (WCF), uma saída de rastreio separada é gerada para o rastreio do canal customizado do WCF integrado com o rastreio de infra-estrutura do Microsoft WCF.

Ativar totalmente o rastreio para o canal customizado do WCF produz dois arquivos de saída:

1. O rastreio do canal customizado do WCF integrado com o rastreio de infra-estrutura do Microsoft WCF.
2. O rastreio do canal customizado do WCF integrado com o XMS do .NET.

Tendo duas saídas de rastreio, problemas podem ser controlados em cada interface usando as ferramentas apropriadas, por exemplo:

- Determinação de problemas do WCF usando conjunto de ferramentas adequado do Microsoft.
- O IBM MQ MQI client emite usando o formato de rastreio XMS.

Para simplificar a ativação do rastreio, o .NET 3 TraceSource e a pilha de rastreamento do .NET XMS são ambos controlados usando uma única interface conforme descrito em: "Configuração de rastreio WCF e os nomes do arquivo de rastreio: interface SOAP/JMS" na página 1299.

Hierarquia de Exceção do Canal Customizado do WCF

Os tipos de exceções lançados pelo canal customizado são consistentes com o WCF e são, geralmente, TimeoutException ou CommunicationException (ou uma subclasse de CommunicationException). Detalhes adicionais da condição de erro, onde disponíveis, são fornecidos usando exceções vinculadas ou internas.

Interface SOAP/JMS

As exceções a seguir são exemplos típicos e cada camada na arquitetura do canal contribui com uma exceção vinculada adicional, por exemplo, CommunicationsException tem uma XMSException vinculada, que tem uma MQException vinculada:

1. System.serviceModel.CommunicationsExceptions
2. IBM.XMS.XMSException
3. IBM.WMQ.MQException

As informações chave são capturadas e fornecidas na coleta de dados da CommunicationException mais alta na hierarquia. Esta captura e o fornecimento de dados evitam a necessidade de os aplicativos se vincularem a cada camada na arquitetura do canal para interrogar as exceções vinculadas e as informações adicionais que elas podem conter. Os nomes de chaves a seguir são definidos:

- IBM.XMS.WCF.ErrorCode: O código de mensagem de erro da exceção do canal customizado atual.
- IBM.XMS.ErrorCode: a mensagem de erro da primeira exceção XMS na pilha.
- IBM.WMQ.ReasonCode: o código de razão subjacente do IBM MQ.
- IBM.WMQ.CompletionCode: o código de conclusão subjacente do IBM MQ.

Interface não SOAP/não JMS

As exceções a seguir são exemplos típicos e cada camada na arquitetura do canal contribui com uma exceção vinculada adicional, por exemplo, CommunicationsException tem uma MQException vinculada:

1. System.ServiceModel.CommunicationsExceptions
2. IBM.WMQ.MQException

As informações chave são capturadas e fornecidas na coleta de dados da CommunicationException mais alta na hierarquia. Esta captura e o fornecimento de dados evitam a necessidade de os aplicativos se vincularem a cada camada na arquitetura do canal para interrogar as exceções vinculadas e as informações adicionais que elas podem conter. Os seguintes nomes de chave são definidos:

- IBM.WMQ.WCF.ErrorCode: o código de mensagem de erro da exceção do canal customizado atual.
- IBM.WMQ.ReasonCode: o código de razão subjacente do IBM MQ.
- IBM.WMQ.CompletionCode: o código de conclusão subjacente do IBM MQ.

Configuração de rastreio do WCF

Há duas opções para configurar o rastreio WCF. É possível configurar o rastreio programaticamente ou através de uma variável de ambiente.

Configuração de rastreio WCF e os nomes do arquivo de rastreio: interface SOAP/JMS

Quando o rastreio está totalmente ativado, ele produz dois arquivos de saída, um para diagnosticar problemas WCF e um arquivo detalhado para material de diagnóstico de rastreio interno. Para simplificar a ativação de rastreio, as pilhas de rastreio .NET 3 TraceSource e XMS .NET usam uma interface única.

Dois métodos de rastreio diferentes estão disponíveis para o canal customizado do WCF. Os dois métodos de rastreio são ativados de modo independente ou juntos. Cada método produz seu próprio arquivo de rastreio, portanto, quando ambos os métodos de rastreio foram ativados, dois arquivos de saída de rastreio são gerados.

Para manter a configuração e ativação o mais simples possível, a mesma interface é usada para controlar ambos os métodos de rastreio. O arquivo `app.config` deve ser editado para incluir a configuração de rastreio relevante conforme descrito na seção a seguir. Os usuários podem, então, incluir suas próprias seções equivalentes para combinar a saída com o rastreio de seus próprios aplicativos.

O rastreio do canal customizado do WCF não é ativado por padrão. É necessário, primeiro, criar um listener de rastreio e, em seguida, configurar o nível de rastreio necessário para a origem de rastreio selecionada no arquivo `app.config`.

Configurando o Canal Customizado do WCF com o Rastreio de Infraestrutura do WCF

Inclua a seção de código a seguir na seção `<system.diagnostics><sources>` no arquivo `app.config`:

```
<source name="IBM.XMS.WCF" switchValue="Verbose,ActivityTracing">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

A parte de código anterior usa o rastreio do canal usando o .NET 3 TraceSource. Todas as chamadas dos arquivos de configuração associados aos arquivos executáveis são controladas por esta parte do código.

Configurando o canal customizado do WCF com rastreio XMS .NET

A configuração do rastreio XMS .NET requer a inclusão de uma seção de código na seção `<system.diagnostics><sources>` no arquivo `app.config`. No entanto, a parte de código é incluída no elemento `<source>` extensível mostrado na seção Configurando o canal customizado do WCF com o rastreio de infraestrutura do WCF. Portanto, embora o código de rastreio de infraestrutura do WCF deva estar presente para que o rastreio de XMS .NET funcione, o rastreio de infraestrutura do WCF pode ser desativado se ele não for necessário, conforme descrito na seção Ativando o rastreio do WCF.

```
<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing"
 xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path"
 xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

Variáveis de Configuração de Rastreio do WCF

Tabela 184. Variáveis de Configuração de Rastreio do WCF

Variável	**Descrição**
nome	Especifique o nome como: `IBM.XMS.WCF`
switchValue	switchValue controla o nível de rastreio. Quando switchValue é configurado como `Off`, TraceSource da infraestrutura do WCF não é gerada. Qualquer outro valor, como `Verbose`, gera TraceSource. Para obter informações detalhadas do nível de rastreio do Microsoft, consulte sua documentação do WCF ou vá para a página da web de Rastreio do WCF Microsoft: https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx

Tabela 184. Variáveis de Configuração de Rastreio do WCF (continuação)

Variável	Descrição

xmsTraceSpecification=*ComponentName=type=state*	*ComponentName* é o nome da classe que você deseja rastrear. É possível usar um caractere curinga * neste nome. Por exemplo:
	`*=all=enabled`
	especifica que você deseja rastrear todas as classes, e
	`IBM.XMS.impl.*=all=enabled`
	especifica que você requer apenas o rastreio da API.
	type pode ser qualquer um dos seguintes tipos de rastreio:
	• all
	• debug
	• evento
	• EntryExit
	state pode ser ativado ou desativado.

xmsTraceFilePath="*filename*"	Se você não especificar um xmsTraceFilePath, ou se o xmsTraceFilePath estiver presente mas contiver uma sequência vazia, o arquivo de rastreio será colocado no diretório atual. Para armazenar o arquivo de rastreio em um diretório nomeado, especifique o nome de diretório no xmsTraceFilePath, por exemplo: `xmsTraceFilePath="c:\somepath"`
xmsTraceFileSize="*size*"	O tamanho máximo permitido do arquivo de rastreio. Quando um arquivo atinge este tamanho, ele é arquivado e renomeado. O máximo padrão é 20 KB, que é especificado como: `xmsTraceFileSize="20000000".`
xmsTraceFileNumber="*number*"	O número de arquivos de rastreio que devem ser retidos. O padrão é 4 (um arquivo ativo e três arquivos de archive). O número mínimo permitido é dois.

Tabela 184. Variáveis de Configuração de Rastreio do WCF (continuação)	
Variável	**Descrição**

xmsTraceFormat="*format*"	Existem dois níveis de xmsTraceFormat: `basic` e `advanced`. O formato de rastreio padrão é básico se você não especificar um xmsTraceFormat ou se o xmsTraceFormat estiver presente, mas contiver uma sequência de caracteres vazia. Os arquivos de rastreio são produzidos neste formato se você especificar:
	`xmsTraceFormat="basic"`
	Se você requerer um rastreio que seja compatível com as ferramentas do analisador de rastreio, deverá especificar:
	`traceFormat="advanced"`

Ativando o Rastreio do WCF

Existem quatro combinações para ativar e desativar os dois métodos de rastreio diferentes. As quatro combinações requerem a edição dos valores das seções de codificação descritas nas seções anteriores.

Há também uma variável de ambiente que pode ser configurada; para obter informações adicionais, veja “Ativando o Rastreio WCF com a Variável de Ambiente WCF_TRACE_ON” na página 1303.

Esta tabela e os valores mostrados dependem de as partes do código demonstrado anteriormente já terem sido incluídas no arquivo `app.config`.

Tabela 185. Combinações de Ativação de Rastreio de WCF.		
Tipo de Rastreio	**Valor Alterado**	**exemplo**
Rastreio XMS ativado. TraceSource de WCF ativada	O `switchValue` não é configurado como `Off`	`<source name="IBM.XMS.WCF" switchValue=" Verbose, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
Rastreio XMS ativado. TraceSource de WCF desativada	O `switchValue` está configurado como `Off` e uma `xmsTraceSpecification` foi fornecida	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

Tabela 185. Combinações de Ativação de Rastreio de WCF. (continuação)

Tipo de Rastreio	Valor Alterado	exemplo
Rastreio de XMS desativado. TraceSource de WCF ativada	Existem duas maneiras de atingir este resultado: • A variável `switchValue` não está configurada como `Off` e uma `xmsTraceSpecification` não foi incluída • A variável `switchValue` não está configurada como `Off` e o `xmsTraceSpecification` foi configurado como `disabled`	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
Rastreio de XMS desativado. TraceSource de WCF desativada	Há três maneiras de alcançar este resultado: • Nenhum elemento `<source>` no arquivo `app.config` • A variável `switchValue` está configurada como `Off` e um `xmsTraceSpecification` não foi incluído • A variável `switchValue` está configurada como `Off` e o `xmsTraceSpecification` foi configurado como `disabled`	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

Ativando o Rastreio WCF com a Variável de Ambiente WCF_TRACE_ON

Assim como os métodos anteriores descritos para ativar o rastreio WCF, o rastreio de XMS .NET também pode ser ativado usando a variável de ambiente WCF_TRACE_ON.

Configurar a variável de ambiente WCF_TRACE_ON como qualquer valor não nulo é o equivalente a configurar `xmstraceSpecification` como `*=all=enabled`, por exemplo: "`set WCF_TRACE_ON=true`"

Porém, se `xmstraceSpecification` for explicitamente configurado no arquivo `app.config`, a variável de ambiente WCF_TRACE_ON será substituída.

arquivos de saída de rastreio WCF e nomes de arquivos

Os arquivos de rastreio XMS são, tradicionalmente, denominados usando o formato de nome base e ID do processo de: `xms_trace_`*`pid`*`.log`, em que *pid* é o ID do processo.

Como os arquivos de rastreio XMS ainda podem ser produzidos em paralelo com os arquivos de rastreio de canal customizado do WCF, o rastreio de canal customizado do WCF integrado com os arquivos de saída de rastreio XMS .NET têm o formato a seguir para evitar confusão: `wcf xms_trace_`*`pid`*`.log`, em que *pid* é o ID do processo

O arquivo de saída de rastreio é criado no diretório de trabalho atual por padrão, mas este destino poderá ser redefinido, se necessário.

Configuração de rastreio do WCF: interface Não SOAP/Não JMS

Para a interface Não SOAP/Não JMS, é possível configurar o rastreio por meio de uma variável de ambiente ou programaticamente.

Existem duas maneiras de ativar o rastreio para a interface Não SOAP/Não JMS:

- Configurando WMQ_TRACE_ON como a variável de ambiente.

- Adicionando a seção de código a seguir na seção <system.diagnostics><sources> no arquivo app.config

```
<source name="IBM.WMQ.WCF" switchValue="Verbose, ActivityTracing"
xmsTraceSpecification="*=all=enabled"
xmsTraceFileSize="2000000" xmsTraceFileNumber="4"
xmsTraceFormat="advanced">
</source>
```

WCF XMS First Failure Support Technology (FFST)

É possível coletar informações detalhadas sobre o que várias partes do código IBM MQ estão fazendo usando o rastreio IBM MQ. O XMS FFST possui seus próprios arquivos de configuração e saída para o canal customizado do WCF.

Os arquivos de rastreio do XMS FFST são nomeados tradicionalmente usando o formato de nome base e ID de processo de: `xmsffdc` *`pid_date`*`.txt`, em que *pid* é o ID do processo e *date* é a data e hora.

Como os arquivos de rastreio do XMS FFST ainda podem ser produzidos em paralelo com arquivos do XMS FFST do canal customizado WCF, os arquivos de saída do XMS FFST do canal customizado WCF têm o seguinte formato para evitar confusão: `wcf ffdc` *`pid_date`*`.txt`, em que *pid* é o ID do processo e *date* é a data e hora.

Este arquivo de saída de rastreio é criado no diretório de trabalho atual por padrão, mas este destino pode ser redefinido se necessário.

O canal customizado do WCF com cabeçalho de rastreio XMS .NET é semelhante ao exemplo a seguir:

```
************* Início do Ambiente XMS WCF de Exibição *************
Nome do Produto  :- valor
Versão do WCF    :- valor
Nível  :- valor
*************  Fim do Ambiente XMS WCF de Exibição  *************
```

Os arquivos de rastreio FFST são formatados da forma padrão, sem qualquer formatação que seja específica para o canal customizado.

Informações da Versão do WCF

As informações da versão do WCF auxiliam na determinação de problema e estão incluídas nos metadados do conjunto do canal customizado.

O canal customizado do IBM MQ para metadados da versão do WCF pode ser recuperado de três maneiras diferentes:

- Usando o utilitário do IBM MQ dspmqver. Para obter informações sobre como usar o dspmqver, consulte: dspmqver
- Usando o diálogo de propriedades do Windows Explorer: no Windows Explorer, clique com o botão direito em **IBM.XMS.WCF.dll** > **Propriedades** > **Versão**.
- A partir das informações do cabeçalho de qualquer um dos arquivos de rastreio ou FFST dos canais. Para obter mais informações sobre o cabeçalho FFST, consulte: “WCF XMS First Failure Support Technology (FFST)” na página 1304

Sugestões e Dicas do WCF

As seguintes sugestões e dicas estão em ordem não significativa e podem ser incluídas quando novas versões da documentação forem liberadas. Elas são assuntos que podem economizar tempo se elas são relevantes para o trabalho que você está fazendo.

Externalizando Exceções do Host de Serviço do WCF

Para serviços hospedados usando o host de serviço do WCF, as exceções não manipuladas lançadas pelo serviço, internos do WCF ou pilha de canais não são externalizadas por padrão. Para ser informado sobre essas exceções, um manipulador de erros deve ser registrado.

O código a seguir fornece um exemplo de como definir o comportamento do serviço do manipulador de erros que pode ser aplicado como um atributo de um serviço:

```
using System.ServiceModel.Dispatcher;
using System.Collections.ObjectModel;
....
    public class ErrorHandlerBehaviorAttribute : Attribute, IServiceBehavior, IErrorHandler
    {
        //
        // IServiceBehavior Interface
        //
        public void AddBindingParameters(ServiceDescription serviceDescription,
           ServiceHostBase serviceHostBase, CollectionServiceEndpoint endpoints,
           BindingParameterCollection bindingParameters)
        {
        }
        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
            ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher channelDispatcher in serviceHostBase.ChannelDispatchers)
{
                channelDispatcher.ErrorHandlers.Add(this);
            }
        }
        public void Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
        {
        }

        //
        // IErrorHandler Interface
        //
        public bool HandleError(Exception e)
        {
            // Process the exception in the required way, in this case just outputting to the
console
            Console.Out.WriteLine(e);

            // Always return false to allow any other error handlers to run
            return false;
        }
        public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
        {
        }
    }
```

Manipulando Diferentes Nomes de Elemento de Resposta do SOAP

O WCF espera que o nome de um valor retornado esteja em um formato específico por padrão, mas um serviço pode não retornar um elemento com seu nome no formato esperado.

O WCF tem a convenção de esperar que o valor retornado seja denominado no seguinte formato: *methodName*`Resultado` em que *methodName* é o nome da operação de serviço. Por exemplo, para um serviço chamado `getQuote`, o WCF espera que a resposta seja chamada: `getQuoteResult`.

Porém, o serviço pode retornar um elemento com um nome que não esteja em conformidade com este formato.

Ao executar a ferramenta scvutil para gerar um cliente proxy, se o WSDL especificar um nome diferente, a interface do proxy incluirá parâmetros para instruir o WCF com o nome a procurar. Por exemplo:

```
[System.ServiceModel.OperationContractAttribute(Action = "", ReplyAction = "*")]
[System.ServiceModel.XmlSerializerFormatAttribute(Style =
System.ServiceModel.OperationFormatStyle.Rpc,
                                                  Use =
System.ServiceModel.OperationFormatUse.Encoded)]
```

```
[return: System.ServiceModel.MessageParameterAttribute(Name = "getQuoteReturn")]
float getQuote(string in0);
```

Se você criar sua própria interface (por exemplo, incluindo um método pedido-resposta em uma interface de proxy existente), será necessário assegurar a inclusão dos mesmos parâmetros na interface se o serviço retornar um nome diferente. Se você não fizer isso, o problema mais comum será que uma chamada para o método de serviço sempre retornará um valor nulo; se um objeto for retornado, o método retornará nulo, mas se um valor numérico como um número inteiro for retornado, o método retornará 0.

Desenvolvendo serviços da web com o IBM MQ

É possível desenvolver aplicativos IBM MQ para serviços da web usando o transporte IBM MQ para SOAP.

Sobre esta tarefa

Nota: A partir da IBM MQ 9.0, o transporte IBM MQ para SOAP foi descontinuado. Isso inclui os recursos de produto a seguir:

- Listener do IBM MQ Java
- IBM MQ .NET 1 e 2 Listener
- Cliente IBM MQ Java Axis2
- Cliente IBM MQ Java (descontinuação anunciada em IBM MQ 8.0)
- Clientes IBM MQ .NET 1 e 2 (descontinuação anunciada em IBM MQ 8.0)
- IBM MQ bridge for HTTP (descontinuação anunciada em IBM MQ 8.0)

O IBM MQ Transport for SOAP fornece um JMS Transport for SOAP. O IBM MQ Transport for SOAP também é integrado em outros ambientes como Microsoft Windows Communication Foundation, WebSphere Application Server e CICS Transaction Server.

Para obter informações adicionais sobre o transporte do IBM MQ para SOAP, consulte “Desenvolvendo serviços da web com o transporte do IBM MQ para SOAP” na página 1307.

Conceitos relacionados

“Conceitos de desenvolvimento de aplicativos” na página 6
É possível usar uma opção de linguagens orientadas a objeto ou processuais para gravar aplicativos IBM MQ. Use os links neste tópico para obter informações sobre conceitos do IBM MQ que são úteis para desenvolvedores de aplicativos.

“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 47
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 715
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 912
O que você precisa saber para escrever aplicativos clientes no IBM MQ usando uma linguagem processual.

“Escrevendo aplicativos de publicar/assinar” na página 806
Inicie a escrever aplicativos de publicar/assinar IBM MQ.

“Construindo um aplicativo processual” na página 1005
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 1055
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 1075
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 serviços da web com o transporte do IBM MQ para SOAP

O transporte do IBM MQ para SOAP fornece um transporte do JMS para SOAP. O IBM MQ Transport for SOAP também é integrado em outros ambientes como Microsoft Windows Communication Foundation, WebSphere Application Server e CICS Transaction Server.

Sobre esta tarefa

Nota: A partir da IBM MQ 9.0, o transporte IBM MQ para SOAP foi descontinuado. Isso inclui os recursos de produto a seguir:

- Listener do IBM MQ Java
- IBM MQ .NET 1 e 2 Listener
- Cliente IBM MQ Java Axis2
- Cliente IBM MQ Java (descontinuação anunciada em IBM MQ 8.0)
- Clientes IBM MQ .NET 1 e 2 (descontinuação anunciada em IBM MQ 8.0)
- IBM MQ bridge for HTTP (descontinuação anunciada em IBM MQ 8.0)

Introdução ao IBM MQ Transport for SOAP

O IBM MQ Transport for SOAP fornece um JMS Transport for SOAP. O emissor e o listener do IBM MQ SOAP fornecem um meio para chamar os serviços da web.

O listener do IBM MQ SOAP suporta serviços hospedados pelo .NET Framework 1, .NET Framework 2 e Axis 1.4. O emissor do IBM MQ SOAP suporta os clientes de serviços da web em execução no .NET Framework 1, .NET Framework 2, Axis 1.4 e Axis2. Os clientes podem ser um aplicativo cliente ou servidor do IBM MQ. O IBM MQ Transport for SOAP também é integrado em outros ambientes como Microsoft Windows Communication Foundation, WebSphere Application Server e CICS Transaction Server.

A integração em Microsoft Windows Communication Foundation é parte do suporte IBM MQ para o .NET Framework 3.

O IBM MQ Transport for SOAP é um conjunto de protocolos e ferramentas para transportar mensagens SOAP usando o JMS sobre IBM MQ. Ele é empacotado de maneiras diferentes para ambientes de aplicativos diferentes, conforme mostrado na Tabela 186 na página 1307.

Tabela 186. Ambientes de aplicativos do IBM MQ Transport for SOAP

	Integrado com componentes adicionais do IBM MQ	**Integrado a uma estrutura**
Fornecido como parte da instalação do IBM MQ	.NET Estrutura 1 .NET Estrutura 2 Eixo 1.4	Windows Communication Foundation (.NET Framework 3) Axis2 (Somente Cliente)..
Fornecido em outro pacote de software		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server para Multiplataformas

A integração do IBM MQ Transport for SOAP em uma estrutura de aplicativo simplifica o desenvolvimento e a implementação de serviços da web para IBM MQ.

Com componentes do IBM MQ SOAP adicionais, é possível interagir diretamente com os componentes do IBM MQ SOAP para desenvolver e implementar serviços. Use as ferramentas do IBM MQ SOAP para configurar e implementar os serviços da web e os clientes de serviço da web para IBM MQ.

Nos ambientes integrados, o desenvolvimento e a implementação é mais simples. É possível usar as mesmas ferramentas para desenvolvimento e implementação, como você faria para desenvolver e implementar um serviço da web SOAP HTTP. Deve-se ainda configurar as filas, canais e gerenciadores de filas do IBM MQ que você requer usando as ferramentas do IBM MQ.

É possível combinar e corresponder com clientes e servidores do IBM MQ SOAP a partir de qualquer um desses ambientes.

Benefícios

O IBM MQ Transport for SOAP oferece aos usuários existentes do IBM MQ, os seguintes benefícios principais:

Usando a rede do IBM MQ para se conectar aos serviços da web existentes.

Os serviços podem ser aqueles gravados ou serviços fornecidos como interfaces para outros aplicativos de software empacotados implementados.

O benefício vem a partir do uso de sua rede existente do IBM MQ para se conectar aos serviços da web. O transporte do IBM MQ tem a vantagem de ser um serviço de sistema de mensagens na fila confiável e gerenciado.

Gravando novos aplicativos ou convertendo seus aplicativos existentes, para usar SOAP em vez de interfaces do IBM MQ.

Geralmente, os aplicativos requerem um adaptador específico do IBM MQ para ser desenvolvido para integrar com outro aplicativo. Os adaptadores possuem duas partes: a parte do conector, que coloca e obtém mensagens para e a partir do transporte e a parte do adaptador que converte dados para e a partir de formatos específicos do aplicativo. A integração de cada par de aplicativos é um novo desafio.

O benefício de SOAP vem da padronização em SOAP para definir as interfaces de aplicativo e, em seguida, ter uma opção de transportes. Não é necessário gravar adaptadores específicos do aplicativo e é possível escolher se deseja usar o IBM MQ ou HTTP como o conector. Qual transporte você escolhe, depende de quais qualidades de serviço e conectividade você requer.

Para usuários SOAP sobre HTTP existentes, o benefício do IBM MQ Transport for SOAP vem do uso de um transporte assíncrono gerenciado e confiável. Os benefícios são duplos:

Um modelo de programação realmente assíncrono para disponibilidade e desempenho.
Ao usar uma interface de cliente assíncrona, os aplicativos de serviço e cliente não precisarão estar disponíveis ao mesmo tempo. As solicitações enviadas pelo cliente serão armazenadas até que o serviço esteja disponível para processá-las.

Uma rede gerenciada pronta construída projetada para ser confiável e disponível.

Ao escolher o IBM MQ como um transporte, você estará obtendo a vantagem de usar uma rede gerenciada que fornece o sistema de mensagens confiável.

Em contraste, transportes, como HTTP e FTP sobre TCP/IP não são gerenciados. Uma rede não gerenciada é ideal para conexões imprevisíveis: há menos tarefas de gerenciamento.

Resumo

O IBM MQ Transport for SOAP fornece os componentes a seguir:

- A ligação de transporte do SOAP/JMS é usada em documentos WSDL para ligar um serviço SOAP a um transporte do JMS. A implementação do IBM MQ da ligação do SOAP/JMS usa um URI que usa um dos dois formatos:

  Transporte IBM MQ para SOAP

  ```
  jms:/queue? &Name=Value&Name=Value...
  ```

  Formato de ligação do IBM MQ para recomendação do candidato do W3C

  ```
  jms:queue: qName ?connectionFactory=connectQueueManager
  (qMgrName)&Name=Value&Name=Value...
  ```

- O mapeamento de uma mensagem SOAP em uma mensagem do IBM MQ.
- Dois listeners SOAP do IBM MQ para receber solicitações SOAP, um para Java e um para .NET Framework 1 ou .NET Framework 2 Os listeners usam .NET ou Axis 1.4 para processar a solicitação SOAP.
- Dois emissores do IBM MQ SOAP criam solicitações SOAP do IBM MQ. os clientes de serviços da web se registram com um emissor para processar as solicitações SOAP `jms:`.
- A integração com o Windows Communication Foundation (WCF), às vezes conhecida como .NET 3, para enviar e receber mensagens do IBM MQ Transport for SOAP.
- A integração do cliente com Axis2, às vezes conhecida como JAX-WS, para enviar o IBM MQ Transport for SOAP ou as mensagens JMS do W3C SOAP.
- O comando **amqwdeployWMQService**, que cria o desenvolvimento, os componentes de tempo de execução e os scripts para implementar um serviço da web usando o IBM MQ Transport for SOAP.
- Java de amostra e cliente e serviço do código do .NET.
- Um script para configurar o caminho de classe e outros scripts de utilitário.

Nos ambientes integrados, o emissor e o listener são integrados em cada ambiente, pois são extensões para as ferramentas de desenvolvimento e implementação.

Integração do SOAP e IBM MQ

O IBM MQ Transport for SOAP estende SOAP e o tempo de execução das ferramentas de serviços da web com o IBM MQ como um transporte alternativo para HTTP para o SOAP. Não é necessário modificar serviços da web existentes para usar o IBM MQ Transport for SOAP como um transporte. O transporte usa um formato de URI customizado para SOAP/JMS. O formato de URI do W3C para SOAP/JMS é suportado em um modo limitado pelos clientes do Axis2.

Uma linha de código adicional precisa ser incluída em clientes nos ambientes do .NET Framework 1, .NET Framework 2 e Axis 1.4. Nenhum código adicional é necessário no Axis 2 e nos clientes Windows Communication Foundation (WCF). O listener do IBM MQ SOAP executa serviços no ambientes do .NET Framework 1, .NET Framework 2 e Axis 1.4. O IBM MQ Transport for SOAP é integrado em alguns outros ambientes de servidor de aplicativos, incluindo WCF, CICS e WebSphere Application Server.

O que é SOAP?

SOAP[11]descreve o formato padronizado das mensagens e protocolos de interação que os aplicativos usam para trocar solicitações, respostas e datagramas. SOAP é independente do transporte usado para transferir as mensagens e do ambiente de aplicativos que envia e recebe as mensagens. O W3C define SOAP 1.2 sucintamente:

SOAP 1.2 fornece a definição das informações baseadas em XML que podem ser usadas para trocar informações estruturadas e digitadas entre peers em um ambiente distribuído descentralizado.[12].

Para usar o SOAP, ele deve ser ligado a um transporte, como HTTP, e-mail ou IBM MQ.

11 Historicamente, o acrônimo era Simple Object Access Protocol.
12 W3C: SOAP 1.2 Parte 0

Uma estrutura de ligação de protocolo SOAP é o conjunto de regras para transportar uma mensagem SOAP na parte superior de outro protocolo, como HTTP. SOAP 1.2 Parte 2: Adjuntos (Segunda Edição) descreve a ligação SOAP HTTP.

A recomendação do candidato do W3C, 4 de junho de 2009, SOAP sobre Java Message Service 1.0, descreve a recomendação para a ligação SOAP do JMS. Como o JMS é uma especificação da API e não um protocolo de transporte, a recomendação do JMS SOAP não descreve o formato de conexão das mensagens SOAP do JMS. Ele descreve os protocolos de interação SOAP e a ligação da API do JMS. Consequentemente, ao usar a recomendação do SOAP JMS que deve-se ainda usar a mesma implementação do JMS para o cliente SOAP e o servidor SOAP. Ele ativa um aplicativo SOAP JMS para ser executado em qualquer implementação de JMS. Uma implementação do JMS poderá ser conectada a um servidor de aplicativos J2EE, se ambos o servidor e a implementação do JMS estiverem de acordo com a especificação JCA.IBM MQ JMS está em conformidade com a especificação JCA e pode ser conectado a um servidor de aplicativos compatível.

A ligação do transporte para SOAP do IBM MQ é como o padrão W3C proposto, mas não é o mesmo. Seu uso é descrito no tópico Configurações SOAP de MQRFH2. Ao contrário da recomendação do candidato do W3C, a ligação SOAP não está formalmente especificada. Com efeito, é a ligação HTTP e o endereço de serviço toma o formulário, `jms:/queue?name=value&name=value...`, em vez `http://authority/path?query#fragment`. `jms:` não é um esquema de URI oficialmente registrado pelo IANA.

O que é um serviço da web?

SOAP permite que os programas gravados em linguagens diferentes, em execução em diferentes plataformas, se comuniquem usando vários protocolos de transporte. SOAP é a especificação do protocolo. Um serviço da web é um aplicativo que fornece um serviço por meio de uma interface SOAP que pode ser acessado usando protocolos da Internet.

Um objetivo importante do SOAP é fornecer serviços que os clientes possam usar facilmente. Depois de ter projetado um cliente para usar um serviço, será possível programar a chamada para solicitar o serviço sem fazer referência à documentação externa. As interfaces de serviço são descritas em XML, em um documento WSDL. A consulta, `http://authority/path?wsdl`, retorna a descrição WSDL de um serviço SOAP.

Sugestão: Ao implementar um serviço da web para usar o IBM MQ, também implemente o serviço em HTTP para que a consulta WSDL padrão funcione.

Desenvolvendo servioços da web

Os serviços da web têm um cliente e uma parte do serviço. O serviço é gravado primeiro, iniciando a partir da descrição de interface em WSDL ou seguindo as regras para gravar a classe de serviço. Os kits de ferramentas de serviço da web têm utilitários para gerar WSDL a partir de uma definição de interface de uma classe; por exemplo, **`java2wsdl`** ou **`disco`** Eles também possuem ferramentas para gerar estruturas básicas de classe a partir de descrições de interface WSDL; por exemplo **`wsdl2java`**, **`wsimport`** ou **`wsdl`**. O primeiro é conhecido como desenvolvimento ascendente e o segundo como desenvolvimento descendente.

O comando **`amqwdeployWMQService`** no IBM MQ Transport for SOAP usa essas ferramentas para gerar WSDL, stubs do cliente e os proxies de cliente.

Os serviços da web são geralmente gravados usando um ambiente de desenvolvimento integrado destinado a um ambiente de servidor de aplicativos específico:

Eclipse IDE para Java EE Developers
Cria serviços da Web para o Axis 2 Suporta JAX-RPC e JAX-WS

Rational Application Developer 7.5
Cria os serviços da web para o WebSphere Application Server V7 e versões anteriores e também para o Axis. Suporta JAX-RPC e JAX-WS.

WebSphere Integration Developer 6.2
Cria os serviços da web para o WebSphere Process Server e WebSphere ESB. Suporta JAX-RPC e JAX-WS.

Visual Studio 2012
Cria os serviços da web para o .NET Framework 3.5 e anterior (Windows Communication Foundation)

É possível usar qualquer uma dessas ferramentas em combinação com o IBM MQ Transport for SOAP. Depois de ter desenvolvido um serviço para uso com o HTTP, use a ferramenta **amqwdeployWMQService** para implementar os serviços para usar o IBM MQ como um transporte. É possível gravar um novo cliente usando a saída da ferramenta ou modificar seus clientes existentes para usar o IBM MQ Transport for SOAP.

Se o IBM MQ Transport for SOAP for integrado ao ambiente de aplicativos, então não será necessário usar a ferramenta **amqwdeployWMQService** ou modificar o código do cliente. A camada SOAP do cliente direciona as solicitações do cliente que possuem um URI com o prefixo `jms:` para o IBM MQ Transport for SOAP. A camada SOAP do servidor chama o IBM MQ Transport for SOAP para aguardar as solicitações SOAP `jms:` e retorna respostas para o IBM MQ Transport for SOAP.

Geralmente, os serviços .NET foram desenvolvidos de forma ascendente usando anotações de serviço da web no código e os serviços Java de forma descendente, usando definições de interface WSDL. A diferença em abordagens está diminuindo, pois o Java Standard Edition 6 suporta JAX-WS 2.0e usa anotações para qualificar a definição de interfaces de serviço. Ele agora é tão fácil de desenvolver os serviços do Java de baixo para cima e vice-versa. Qual abordagem escolhida é uma questão de método de desenvolvimento.

O cliente de serviços da web é gravado após o serviço usando a definição de serviço WSDL e os proxies e stubs de cliente gerados. Em alguns aplicativos, a definição de serviço não será conhecida quando o cliente for gravado. O cliente recupera o WSDL de serviço e cria solicitações de serviço dinamicamente. Mais comumente a definição de serviço é conhecida, mas o endereço no qual o serviço é implementado, não é. O kit de ferramentas de serviço da web gera interfaces para o cliente usar para fazer solicitações de serviço. O cliente fornecerá o endereço de serviço quando for necessário. No terceiro caso, o WSDL contém todas as informações que um cliente precisa. O WSDL contém a interface e o endereço do serviço. O código gerado pelo kit de ferramentas de serviço da web tem todas as informações necessárias pelo cliente para fazer solicitações de um serviço.

É possível usar qualquer um destes três estilos o IBM MQ Transport for SOAP.

Ambientes de aplicativo de serviço da web

Os kits de ferramentas de serviço da web requerem um mapeamento a partir da definição WSDL de um serviço aos fluxos de bytes transferidos em solicitações e respostas SOAP. O fluxo de bytes é definido pela especificação SOAP e está contido no envelope SOAP. O envelope SOAP é mostrado em Figura 163 na página 1311.

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->

<!-- headers... -->

</soap:Header>
<soap:Body>
<!-- payload or fault message -->

</soap:Body>
</soap:Envelope>
```

Figura 163. Envelope SOAP

O mapeamento a partir do envelope SOAP para a ligação entre linguagens e novamente é a peça padronizada e o proprietário da peça. O mapeamento é fundamental para a arquitetura do .NET e é fornecido como parte do Common Language Runtime (CLR). O mapeamento é padronizada em Java pelas especificações de JAX. Como os mapeamentos do Java são padronizados, os clientes de serviço da web

do Java e os serviços são móveis entre diferentes ambientes de aplicativos baseados no Java. JAX-RPC (às vezes chamado de JAX-WS 1.0) é o mapeamento mais usado nos dias de hoje. Ele é suportado por Axis 1.4. JAX-WS (às vezes chamado de JAX-WS 2.0) é um padrão muito melhorado e provavelmente substituirá o JAX-RPC rapidamente. O JAX-WS é suportado por Axis 2.0. IBM MQ 7.0.1 não suporta o JAX-WS e Axis 2.

O IBM MQ Transport for SOAP não altera o conteúdo do envelope SOAP e o conteúdo não afeta o transporte. As ligações entre linguagens afetam o IBM MQ Transport for SOAP. IBM MQ 7.0.1 suporta .NET Framework 1, .NET Framework 2 e Axis 1.4 usando o código e os utilitários fornecidos com o IBM MQ Transport for SOAP. O suporte para o WebSphere Transport for SOAP no .NET Framework 3 e 3.5 é implementado usando o canal customizado do IBM MQ for Windows Communication Foundation.

Outros ambientes de tempo de execução e desenvolvimento SOAP podem enviar o suporte para o IBM MQ Transport for SOAP e suportar diferentes linguagens. Por exemplo, os serviços da web em execução no CICS suportam linguagens como COBOL e PL/1.

Nota: O mapeamento usado não faz diferença para a interoperabilidade de serviços da web. É possível combinar e corresponder os clientes e serviços gravados usando os mapeamentos do .NET, JAX-RPC e JAX-WS.

O que é o IBM MQ Transport for SOAP?

O IBM MQ Transport for SOAP é uma ligação SOAP e um kit de ferramentas de serviços da web. Juntos, eles permitem que os aplicativos troquem mensagens SOAP usando o IBM MQ em vez de HTTP. Figura 164 na página 1312 mostra o IBM MQ como uma alternativa para o HTTP como um transporte SOAP.

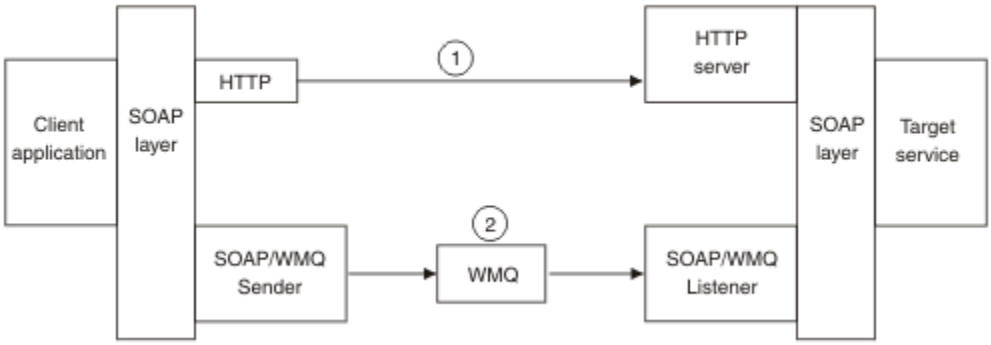


Figura 164. Visão geral do IBM MQ Transport for SOAP

SOAP sobre HTTP é mostrado como (1) no diagrama. A camada SOAP do cliente converte uma solicitação em uma mensagem SOAP e o componente HTTP envia sobre TCP/IP. O componente do servidor HTTP atende as solicitações de HTTP, geralmente na porta 80 do TCP/IP. Se a solicitação for para um serviço SOAP, o componente do servidor HTTP chamará a camada SOAP para converter a solicitação SOAP em chamada de método. Ele então retorna a resposta.

SOAP sobre IBM MQ é mostrado como (2). O aplicativo cliente registra o componente de emissor SOAP do IBM MQ como um identificador para o protocolo `jms:` com a camada SOAP. A camada SOAP transmite as mensagens SOAP endereçados ao `jms:` para o emissor SOAP do IBM MQ. O emissor usa o URI na mensagem para colocar a mensagem na fila de solicitações com as qualidades de serviço necessárias. O listener do IBM MQ SOAP correspondente aguarda por mensagens em sua fila de solicitações e chama a camada SOAP para processar solicitações e respostas de retorno.

O listener e remetente SOAP são programas normais do IBM MQ. Eles podem ser conectados no mesmo gerenciador de filas, como em Figura 165 na página 1313 ou conectados a gerenciadores de filas diferentes; consulte Figura 166 na página 1314. O cliente pode ser conectado por uma conexão do cliente.

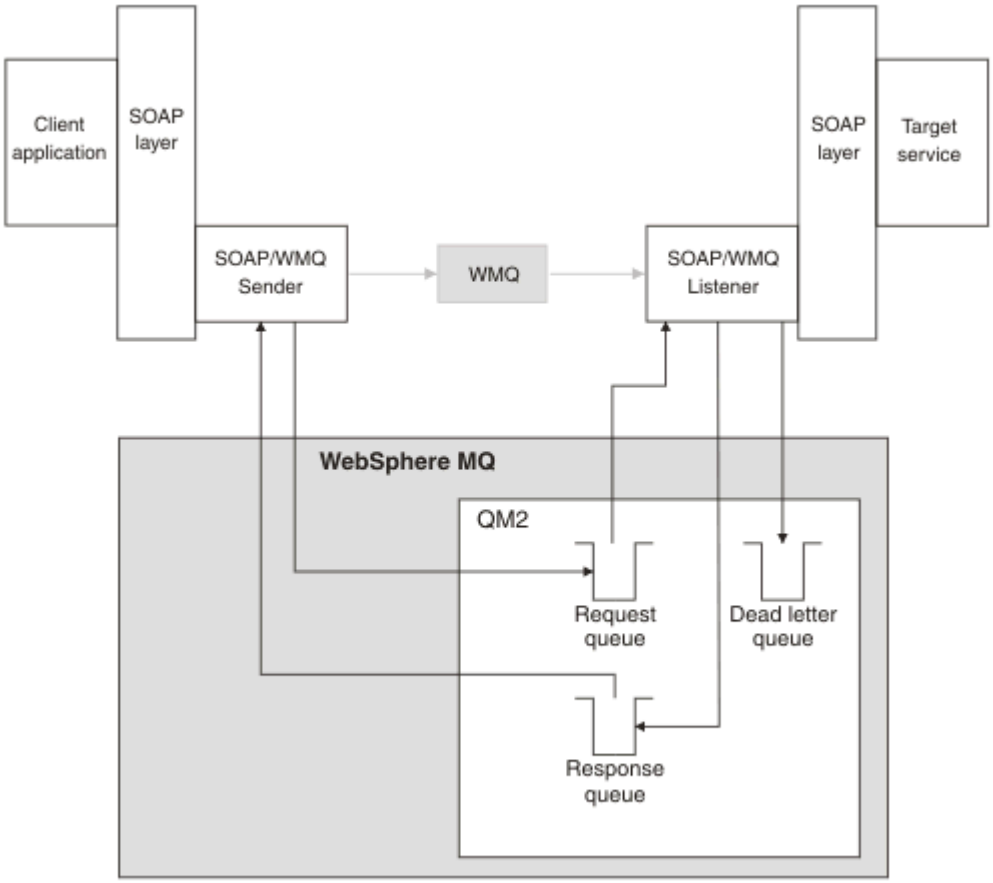


Figura 165. Filas usadas por SOAP/IBM MQ (gerenciador de filas único)

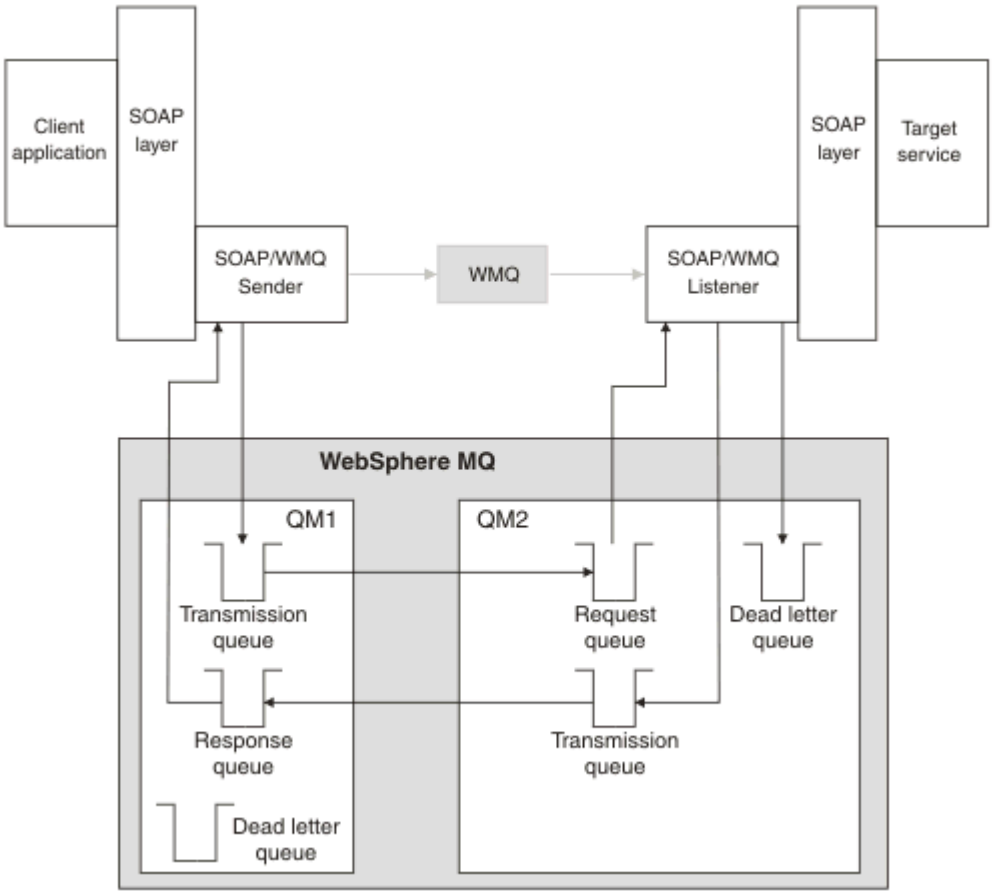


Figura 166. Filas usadas pelo SOAP/IBM MQ (gerenciadores de filas separados)

Recomendação do candidato do W3C para a ligação SOAP para JMS.

A recomendação de candidato do W3C define a ligação SOAP sobre JMS; SOAP sobre Java Message Service 1.0. O Esquema de URI para Java(tm) Message Service 1.0 também é útil para seus exemplos[13].

Algumas estruturas de aplicativos, como WebSphere Application Server v7, têm suporte para a recomendação do candidato do W3C. Envie solicitações SOAP formatadas com um URI compatível com a recomendação do candidato do W3C usando o cliente Axis2; consulte W3C SOAP sobre URI do JMS para o cliente Axis 2 do IBM MQ. O cliente do Axis2 envia uma solicitação SOAP formatada com um W3C ou um IBM MQ Transport for SOAP com base no URI na solicitação SOAP.

O suporte a clientes do Axis2 para a recomendação W3C é introduzido no fix pack 7.0.1.3. O suporte para outros clientes e para os listeners SOAP fornecidos pelo IBM MQ não é fornecido.

Conceitos relacionados

A implementação do WebSphere Transport for SOAP no .NET Framework 1, .NET 2 e Axis 1.4
Você pode desejar gravar seu próprio emissor e listener do IBM MQ SOAP. Use a implementação do IBM MQ Transport for SOAP no .NET Framework 1, .NET Framework 2 e Axis 1.4 como um guia.

Sistema de mensagens confiável dos serviços da web do IBM MQ Transport for SOAP
O sistema de mensagens confiável de serviços da web é um protocolo para trocar de forma confiável as solicitações e respostas de serviço da web em uma conexão não confiável. É adequado para resolver problemas de interrupção de conexão de curta duração.

[13] Procure *Esquema de URI para JMS*, nas referências de especificação W3C, para o rascunho mais recente.

A implementação do WebSphere Transport for SOAP no .NET Framework 1, .NET 2 e Axis 1.4

Você pode desejar gravar seu próprio emissor e listener do IBM MQ SOAP. Use a implementação do IBM MQ Transport for SOAP no .NET Framework 1, .NET Framework 2 e Axis 1.4 como um guia.

1. Um programa cliente usa a estrutura de serviços da web apropriada da mesma maneira que faria para o transporte HTTP. Ele também deve registrar o prefixo `jms:`. O prefixo é registrado usando o método `com.ibm.mq.soap.Register.extension()` Java ou o método CLR `IBM.WMQSOAP.Register.Extension()`.
2. O Axis 1.4 ou a estrutura do .NET Framework 1 ou 2 serializa a chamada em uma mensagem da solicitação SOAP exatamente como para SOAP/HTTP.
3. Um serviço do IBM MQ é identificado por um URI prefixado com `jms:`. Quando a estrutura identificar o URI `jms:`, ele chamará o código do emissor de transporte do IBM MQ; `com.ibm.mq.soap.transport.jms.WMQSender` (para Axis 1.4) ou ou `IBM.WMQSOAP.MQWebRequest` (para .NET1 e 2). Se a estrutura encontrar um URI com um prefixo `http:`, ela chamará o emissor padrão de SOAP sobre HTTP.
4. A mensagem SOAP é transportada pelo emissor de SOAP do IBM MQ usando a fila de solicitações. O **`SimpleJavaListener`** (for Java) ou **`amqwSOAPNETListener`** (for .NET) recebe a mensagem de solicitações.

   Os listeners do IBM MQ SOAP são processos independente e multiencadeados com um número customizável de encadeamentos.
5. O listener do IBM MQ SOAP lê a solicitação SOAP recebida e a transmite para a infraestrutura de serviço da web apropriada.
6. A infraestrutura de serviço da web analisa a mensagem de solicitação SOAP e invoca o serviço. O procedimento é o mesmo para uma mensagem que chegou em um transporte de HTTP.
7. A infraestrutura formata a resposta em uma mensagem de resposta SOAP e a retorna para o listener do IBM MQ SOAP.
8. O listener coloca a mensagem na fila de resposta e a mensagem é transferida para o emissor do IBM MQ SOAP. O emissor a passa para a infraestrutura de serviço da web do cliente.
9. A infraestrutura do cliente analisa a mensagem SOAP de resposta e manipula o resultado de volta para o aplicativo cliente.

Cada contexto de aplicativos é servido por uma fila de solicitação separada do IBM MQ.

O contexto de aplicativos é controlado no Axis 1.4, assegurando-se de que o listener do IBM MQ SOAP e o serviço sejam executados no diretório apropriado. Axis 1.4 configura o CLASSPATH correto para o diretório.

O contexto de aplicativos é controlado no .NET pelo listener do IBM MQ SOAP executando o serviço em um contexto criado por uma chamada para `ApplicationHost.CreateApplicationHost`. A chamada especifica o diretório de execução de destino. Cada serviço, então, opera no diretório no qual ele foi implementado.

`amqwdeployWMQService` gera as filas de solicitações e respostas. Ele também gera a infraestrutura necessária para manipular as filas e implementar serviços para o Axis 1.4.

Conceitos relacionados

Integração do SOAP e IBM MQ

Sistema de mensagens confiável dos serviços da web do IBM MQ Transport for SOAP

O sistema de mensagens confiável de serviços da web é um protocolo para trocar de forma confiável as solicitações e respostas de serviço da web em uma conexão não confiável. É adequado para resolver problemas de interrupção de conexão de curta duração.

Sistema de mensagens confiável dos serviços da web do IBM MQ Transport for SOAP

O sistema de mensagens confiável de serviços da web é um protocolo para trocar de forma confiável as solicitações e respostas de serviço da web em uma conexão não confiável. É adequado para resolver problemas de interrupção de conexão de curta duração.

IBM MQ for SOAP obtém vantagem de usar um IBM MQ gerenciado e uma rede confiável para transmitir as mensagens SOAP. Transportes como HTTP e FTP não são gerenciados. As redes não gerenciadas são ideais para conexões imprevisíveis, em que as dificuldades e os custos de gerenciamento de conexões excedem os benefícios de não perder solicitações e respostas.

Para superar o problema de perda de arquivos quando interrupções de conexões em redes não gerenciadas, serviços como FTP gerenciado construírem uma camada de gerenciamento na parte superior de FTP. A camada de gerenciamento assume a responsabilidade de verificar se os arquivos foram transferidos com sucesso a partir de usuários e retransmite arquivos ausentes, se necessário. Para usar o FTP gerenciado, deve-se ter o software de gerenciamento instalado em ambas as extremidades da conexão.

O sistema de mensagens confiável de serviços da web (WSRM) assume uma abordagem diferente para solucionar o problema de conexões não confiáveis. Seu objetivo é transferir as solicitações e respostas de serviço da web de forma confiável, sem que ambas as extremidades da conexão tenham que usar o mesmo software. Qualquer software, implementando o protocolo de sistema de mensagens confiável de serviços da web, pode trocar mensagens de forma confiável com outro software.

Quando uma conexão falhar, um emissor e um receptor deverão preservar o contexto da transferência de mensagem WSRM, usando o URI gerado como uma chave. O emissor e receptor tentam estabelecer uma nova conexão. Se uma nova conexão for estabelecida com sucesso, a transferência será concluída. A especificação WSRM não especifica como o contexto é preservado ou quando uma nova conexão é tentada.

Você pode decidir que apenas indisponibilidades de curta duração são de interesse. Para mais indisponibilidades, você pode estar preparado para descartar as transferências que não puderam ser reiniciadas após uma hora. Da mesma forma, você pode estar preparado para descartar as transferências, se o cliente ou serviço falhar. Deixar o usuário responsável por garantir as transferências, coloca menos demandas no gerenciamento de coordenação de cliente e serviço.

Se as indisponibilidades de rede forem de longa duração, mais de 30 minutos mais ou menos, ou se o cliente ou o servidor falhar, haverá uma probabilidade maior de que algumas conexões nunca serão restabelecidas. Não é mais possível contar com o WSRM restaurando a transferência de mensagem automaticamente em um modo não gerenciado. É necessário considerar o gerenciando das conexões WSRM com falha, que significa o desenvolvimento de software para gerenciar a rede de clientes e serviços.

O uso do WSRM para superar indisponibilidades curtas pode reduzir significativamente como lidar com mensagens perdidas em uma rede remota. Se você não tiver para garantir a entrega de mensagens, os benefícios de reduzir a perda de mensagens poderá justificar o custo adicional de desenvolvimento de uma implementação do WSRM.

O SOAP sobre JMS fornece entrega de mensagem assegurada e lida com indisponibilidades de duração mais longa do cliente, do servidor e da rede. Se você estiver buscando uma qualidade de serviço confiável para SOAP do que o HTTP, qual solução você escolhe: IBM MQ Transport for SOAP ou WSRM? A resposta depende de vários fatores. Alguns dos fatores a serem considerados são listados:

1. Se as incertezas for devido a uma falha de conexão.
2. Quanto tempo as falhas de conexão são desejadas.
3. Se for possível gerenciar ambos, o cliente e o lado do servidor da conexão.
4. Se o usuário ou um administrador for responsável basicamente para a entrega de mensagens.

Conceitos relacionados

Integração do SOAP e IBM MQ

A implementação do WebSphere Transport for SOAP no .NET Framework 1, .NET 2 e Axis 1.4

Você pode desejar gravar seu próprio emissor e listener do IBM MQ SOAP. Use a implementação do IBM MQ Transport for SOAP no .NET Framework 1, .NET Framework 2 e Axis 1.4 como um guia.

Instalando e verificando os serviços da web do IBM MQ

Use as instruções nestes tópicos para instalar e verificar o IBM MQ Transport for SOAP.

Instalando o transporte da web do IBM MQ para SOAP

Use estas instruções para instalar o transporte da web do IBM MQ para SOAP. A instalação cria ferramentas para executar serviços ou clientes de serviço da web usando o IBM MQ como o transporte SOAP. As ferramentas são usadas nos ambientes .NET Framework 1, .NET 2, Axis 1.4 ou Axis2 SOAP.

Antes de começar

Verifique os produtos com pré-requisitos em Requisitos do sistema para IBM MQ. O processo de instalação não verifica a presença e a disponibilidade do software obrigatório. Deve-se verificar se os pré-requisitos estão instalados.

O IBM MQ fornece uma cópia do tempo de execução do Axis 1.4. Use essa versão com o IBM MQ em vez de quaisquer outros que você possa ter instalado. A IBM não fornece suporte técnico para o Apache Axis. Entre em contato com a Apache Software Foundation se tiver problemas técnicos com ele.

Para executar serviços da web no ambiente .NET Framework 3 SOAP, o IBM MQ usa o Windows Communication Foundation. O canal customizado do IBM MQ para o Windows Communication Foundation executa clientes de serviço da web usando o IBM MQ como um transporte para mensagens SOAP.

Sobre esta tarefa

É possível instalar o transporte da web do IBM MQ para SOAP como um aplicativo IBM MQ MQI client ou Server. Se já tiver instalado o IBM MQ como um cliente ou um servidor em seu computador, verifique se instalou os componentes listados.

O *MQ_INSTALLATION_PATH* representa o diretório de alto nível no qual o IBM MQ está instalado.

Execute as etapas de instalação a seguir.

Procedimento

1. Selecione o componente `Java and .Net Messaging and web services` para instalação.
2. No Solaris e no HP-UX, selecione o componente `Java Runtime environment`.
3. Selecione o kit de ferramentas de desenvolvimento para instalação.
4. Instale e verifique o IBM MQ conforme descrito no manual Iniciação rápida para sua plataforma.
5. Copie o tempo de execução do Apache Axis 1.4, `axis.jar`, do diretório `prereqs/axis` na mídia de instalação do IBM MQ. Copie-o para o diretório de instalação descrito em Tabela 187 na página 1318, Tabela 188 na página 1318 ou Tabela 189 na página 1318.

   Windows

   ```
   Copy D:\PreReqs\axis\axis.jar MQ_INSTALLATION_PATH\java\lib\soap
   ```

   AIX

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

Diretórios de instalação do HP-UX, do Solaris e do Linux (todas as plataformas)

```
cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
```

6. No Windows 2003, execute **Aspnet_regiis.exe** para atualizar os mapas de script para apontar para a versão do Common Language Runtime que está usando.

   Procure o utilitário **Aspnet_regiis.exe** em %SystemRoot%Microsoft.NET/Framework/*version-number*.
7. Configure a variável de ambiente, WMQSOAP_HOME, para apontar para o diretório de instalação do IBM MQ.

Resultados

Tabela 187. Diretórios de instalação do Windows	
Local	**Conteúdos**
MQ_INSTALLATION_PATH \programs\bin	Arquivos binários, de comando, DLL e executáveis
MQ_INSTALLATION_PATH \programs\java\lib	.jar files
MQ_INSTALLATION_PATH \programs\java\lib\soap	SOAP .jar files
MQ_INSTALLATION_PATH \programs\soap\samples	Amostras e IVT

Tabela 188. Diretórios de instalação do AIX	
Local	**Conteúdos**
MQ_INSTALLATION_PATH/bin	Shell scripts
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	Arquivos axis.jar e outros .jar de JAX-RPC
MQ_INSTALLATION_PATH/samp/soap	Amostras e IVT

Tabela 189. Diretórios de instalação do HP-UX, do Solaris e do Linux (todas as plataformas)	
Local	**Conteúdos**
MQ_INSTALLATION_PATH/bin	Shell scripts
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	Arquivos axis.jar e outros .jar de JAX-RPC
MQ_INSTALLATION_PATH/samp/soap	Amostras e IVT

Como proceder a seguir

1. Somente para o .NET, deve-se registrar os arquivos transporte do IBM MQ para SOAP com o Global Assembly Cache. Se o .NET já estiver instalado ao instalar o IBM MQ, o registro será executado automaticamente na instalação. Se você instalar o .NET após o IBM MQ, o registro será executado automaticamente quando o IVT for executado pela primeira vez.

   É possível executar **amqiregisterdotnet.cmd** para realizar o registro dos conjuntos do .NET. Também é possível executar **amqiregisterdotnet.cmd** para forçar o novo registro em qualquer

estágio. Uma vez realizado, esse registro sobrevive a reinicializações do sistema e o novo registro subsequente normalmente não é necessário.

2. Execute o Teste de verificação de instalação, conforme descrito em "Verificando o IBM MQ Transport for SOAP" na página 1319.
3. Se pretende desenvolver o cliente Axis2, deve-se fazer download do Axis2 1.4.1 a partir da Apache;, consulte "Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse" na página 1337.

Verificando o IBM MQ Transport for SOAP

Verifique se o IBM MQ Transport for SOAP está usando o comando **runivt**. O comando executa vários aplicativos de demonstração e assegura que o ambiente está corretamente configurado após a instalação. Observe que a parte de serviços da web do transporte para SOAP é descontinuado e se você for um novo usuário, não será necessário usar este.

Antes de começar

Antes de executar o comando **runivt**, assegure-se de que você tenha os seguintes ambientes de tempo de execução:

- Para executar em Axis somente: deve-se ter um Java SDK (dentro de SOE) disponível em seu sistema. Também deve-se incluir o local dos comandos `java.exe` e `javac.exe` na variável de ambiente **PATH** dos sistemas.
- Para executar um teste em .NET apenas (suportado apenas no Windows): deve-se ter ambos um Java SDK e os compiladores e as ferramentas do .NET em seu sistema. Para isso, acesse um prompt de comandos do Visual Studio ou o prompt de comandos do SDK do Microsoft Windows e, em seguida, inclua o local dos arquivos `java.exe` e `javac.exe` na variável de ambiente **PATH**.
- Para executar todos os testes disponíveis: para plataformas do Windows, o ambiente deve ser configurado conforme descrito na execução de teste do .NET. Em plataformas do UNIX and Linux, o ambiente deve ser configurado conforme descrito na execução de teste para apenas Axis.

Sobre esta tarefa

Em vez de executar o teste de verificação em .NET e Axis, você pode desejar executar o teste apenas no Axis ou no .NET.

Se você tiver problemas com os testes e desejar iniciar novamente:

1. Pare o gerenciador de filas `WMQSOAP.DEMO.QM` usando a opção `immediate`.
2. Pare o listener que foi iniciado em uma janela diferente.
3. Exclua o gerenciador de filas.
4. Exclua o diretório de amostras temporário que você criou e inicie novamente.

Em plataformas do UNIX and Linux, deve-se executar o comando usando uma sessão do sistema X Windows.

O comando **runivt** muda o conteúdo do diretório `soap/samples`. Para manter a imagem de instalação inalterado, copie o diretório de amostras para um local temporário e execute o teste de verificação do local temporário.

É possível executar a verificação da instalação, quantas vezes desejar.

Execute as seguintes etapas para verificar a instalação do IBM MQ Transport for SOAP no .NET Framework 1, .NET Framework 2 e Axis 1.4:

Procedimento

1. Copie a árvore de diretórios `./tools/soap/samples` para um local temporário.
2. Inicie uma janela de comandos com o diretório temporário como o diretório atual.

3. Use o comando **runivt** para iniciar o teste de instalação. O script `runivt` compila uma classe de teste, o cliente de amostra e serviços antes de implementá-los e executá-los. Para obter a classe de teste, o cliente de amostra e os serviços para execução, conclua as etapas de instalação descritas em Instalando o WebSphere(r) MQ Web transport for SOAP e assegure-se de que o prompt de comandos usado para executar o comando `runivt` tenha o conjunto de ambiente do tempo de execução necessário. Use qualquer um dos seguintes métodos para executar o comando **runivt**:

   - Execute um teste apenas no Axis: `runivt Axis`.
   - Execute um teste apenas no .NET (suportado apenas no Windows): `runivt DotNet`.
   - Execute todos os testes disponíveis: `runivt`.

   Para obter mais informações sobre os parâmetros e sintaxe do comando `runivt`, consulte **runivt**: teste de verificação de instalação do IBM MQ Transport for SOAP. Os testes que são possíveis executar são listados no arquivo `ivttests.txt` no Windows e `ivttests_unix.txt` em plataformas do UNIX and Linux.

Informações relacionadas

runivt: teste de verificação de instalação do IBM MQ Transport for SOAP

Desenvolvendo serviços da web para transporte do IBM MQ para SOAP

Use o seu ambiente de desenvolvimento de serviço da web normal para desenvolver serviços para uso com o transporte do IBM MQ para SOAP.

Antes de começar

1. Se você estiver planejando usar as ferramentas de linha de comandos fornecidas com o transporte do IBM MQ para SOAP:
   a. Crie um diretório de implementação para o serviço.
   b. Inicie uma janela de comandos no diretório.
   c. Para o .NET, csc.exe e wsdl.exe deve estar no caminho e ser da mesma versão do .NET Framework.
   d. Para o Java,
      i) Execute o comando **amqwsetcp** para configurar o caminho de classe.
      ii) Um IBM JRE e um JDK no mesmo nível de versão devem estar no caminho atual. O nível de versão deve ser pelo menos 5.0.
      iii) Customize o caminho de classe para incluir as localizações de quaisquer bibliotecas adicionais `.jar` e diretórios que contêm pacotes .java, inclusive para o serviço que você está desenvolvendo. Coloque o diretório atual "`.`" no caminho de classe..
      iv) Crie um diretório, relativo ao diretório atual da janela de comandos, correspondente ao nome do pacote do serviço que você está desenvolvendo.
2. Como alternativa, use as ferramentas do ambiente de trabalho que suportam o desenvolvimento de serviços da web. As tarefas de desenvolvimento de exemplo usam Microsoft Visual Studio 2008, Eclipse IDE para Desenvolvedores Java EE e WebSphere Application Server Community Edition.

Sobre esta tarefa

Os serviços da web existentes não precisam de modificação para funcionar com o WebSphere Transport for SOAP. As ferramentas fornecidas como o transporte do IBM MQ para SOAP implementam um serviço da web e executam-no usando um listener SOAP do IBM MQ. As ferramentas também geram WSDL, stubs do cliente .NET e classes proxy `.java` para desenvolver clientes de transporte do IBM MQ para SOAP.

Siga estas etapas para criar um serviço e prepará-lo para a implementação e a geração de clientes. Siga as etapas nas tarefas relacionadas ao criar um serviço usando o Eclipse ou o Microsoft Visual Studio 2008.

Procedimento

1. Desenvolva o serviço usando o seu ambiente de desenvolvimento normal.
2. Teste o serviço usando um cliente de serviços da web HTTP
3. Siga estas etapas para preparar o diretório de implementação:
   - Para Java
     a. Copie o arquivo `.java` definindo a interface de serviço no diretório de implementação.
     b. Copie quaisquer arquivos `.class` para o serviço no diretório correspondente ao nome do pacote.
     c. Verifique se o caminho de classe pode localizar todas as classes que são necessárias: compile o serviço `.java` usando **`javac`**.
   - Para .NET
     a. Copie o arquivo `.asmx` definindo o serviço no diretório de implementação e
     b. Se estiver usando o modelo code-behind, copie quaisquer arquivos `.dll` em um diretório *`deployment directory`*`\bin` .

Desenvolvendo um serviço do .NET 1 ou 2 para o transporte de IBM MQ para SOAP usando o Microsoft Visual Studio 2012

Desenvolva o serviço da web SampleStockQuote para .NET 1 ou .NET 2 usando o Microsoft Visual Studio 2012.

Sobre esta tarefa

Crie o serviço StockQuote com uma implementação protegida por código usando o Microsoft Visual Studio 2012.

Procedimento

1. Crie um modelo para o serviço e verifique se ele é executado em HTTP.
   a) Inicie o Visual Studio 2012 > **Arquivo** > **Novo** > **Projeto...**. Selecione Tipo de projeto **C#**, **,NET Framework 2** e **ASP.NET Web Service Application**. Digite o **Nome:** e o **Nome da solução:** `StockQuoteDotNet` > **OK**
   b) Clique com o botão direito em **Service1.asmx** no **Solution Explorer** > **Renomear** > `StockQuote.asmx`.
   c) Mude o fragmento do código `public class Service1` para `public class StockQuote`.
   d) Clique com o botão direito em **StockQuote.asmx** no **Explorador de Soluções** > **Abrir com ...** > **Editor XML**. Mude `Class="StockQuoteDotNet.Service1"` para `Class="StockQuoteDotNet.StockQuote"`
   e) Mude o fragmento do código `[WebService(Namespace = "http://tempuri.org/")]` para `[WebService(Namespace = "http://stock.samples/")]`.
   f) Remova a linha de código `[ToolboxItem(false)]`.
   g) Verifique se tudo está correto até agora: **Depurar** > **Iniciar depuração (F5)**. Verifique a saída no Explorer.
2. Inclua os métodos da amostra `SQDNNonInline.asmx.cs` e teste o serviço em HTTP.
   a) Abra o *`MQ_INSTALLATION_PATH`* `\tools\soap\samples\dotnet\SQDNNonInline.asmx.cs` e substitua o método `HelloWorld` com os quatro métodos `Quote`; consulte Figura 167 na página 1323. *`MQ_INSTALLATION_PATH`* representa o diretório no qual o IBM MQ está instalado.
   b) **Construir** > **Reconstruir** a solução > Clique com o botão direito em um dos **Encadeamentos** com erro > **Resolver** > Usando **System.Threading**.
   c) Pressione F5 para iniciar a depuração.

O serviço não tem conformante com o WS-I Basic Profile v1.1. Você tem a opção de mudar a anotação WebMethod de `[SoapRpcMethod]` para `[SoapDocumentMethod]` ou remover a anotação `[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]`.

d) Pressione F5 para verificar a implementação usando HTTP.

3. Gere WSDL, clientes e execute o serviço usando o IBM MQ Transport for SOAP.
   a) Abra uma janela de comando na árvore de diretórios do projeto, em que o `StockQuote.asmx` é armazenado.
   b) (Opcional) Use amqwdeployWMQService para gerar artefatos. O gerenciador de filas deve ser iniciado:

```
amqswdeployWMQService -f StockQuote.asmx
-u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   Todos os artefatos são criados na árvore de diretórios `./generated`.

   c) (Opcional) Gere apenas o WSDL para chamar o serviço usando o IBM MQ Transport for SOAP.

```
amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   a) Execute o listener do .NET. Use `.\generated\server\startWMQNListener.cmd` ou digite o comando:

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```

4. Teste o serviço usando um cliente gerado a partir do WSDL ou usando os clientes gerados pelo **amqwdeployWMQService**.

Código de Amostra

O serviço da web de amostra .NET , `StockQuoteDotNet`, é instalado no *MQ_INSTALLATION_PATH* `\tools\soap\samples\dotnet` *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado. A ligação de serviço da web das amostras publicadas difere um pouco da ligação usada na tarefa. A tarefa usa os padrões usados no Microsoft Visual Studio 2012.

Há dois exemplos de serviços da web .NET Framework 1 e .NET Framework 2. `StockQuoteDotNet.asmx`, é um serviço sequencial.. `SQDNNoninline.asmx`, é um serviço da web code-behind implementado pelo `SQDNNoninline.asmx.cs`..

`StockQuoteDotNet` tem quatro métodos:

1. `float getQuote(String symbol)`
2. `void getQuoteOneWay(String symbol)`.
3. `int asyncQuote(int delay)`
4. `float getQuoteDOC(String symbol)`

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
     [WebMethod] [ SoapRpcMethod(OneWay=true) ]
     public void getQuoteOneWay(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
          System.Console.WriteLine("getQuoteOneWay was invoked.");
     }
     [WebMethod] [SoapRpcMethod]
     public float getQuote(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
          return 88.88F;
     }
     [WebMethod] [SoapRpcMethod]
     public int asyncQuote(int delay) {
          Thread.Sleep(delay);
          return delay;
     }
     [WebMethod]
     public float getQuoteDOC(String symbol) {
          return 77.77F;
     }
}
```

Figura 167. Serviço sequencial: `StockQuoteDotNet.asmx`

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

Figura 168. Code-behind: design `SQDNNonInline.asmx`

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }

    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

Figura 169. Code-behind: implementação: SQDNNonInline.asmx.cs

Tarefas relacionadas

Desenvolvendo um serviço da web JAX-WS EJB para W3C SOAP sobre JMS

Um serviço da web ligado à recomendação candidata do W3C para SOAP sobre JMS deve ser executado no contêiner EJB de um servidor de aplicativos JEE. Esta tarefa é a etapa 2 de conexão de um cliente de serviço da web Axis2 e um serviço da web implementados no WebSphere Application Server usando o protocolo W3C SOAP sobre JMS.

Desenvolvendo um serviço da web JAX-WS EJB para W3C SOAP sobre JMS

Um serviço da web ligado à recomendação candidata do W3C para SOAP sobre JMS deve ser executado no contêiner EJB de um servidor de aplicativos JEE. Esta tarefa é a etapa 2 de conexão de um cliente de serviço da web Axis2 e um serviço da web implementados no WebSphere Application Server usando o protocolo W3C SOAP sobre JMS.

Antes de começar

Use o Rational Application Developer para criar o serviço da web EJB. O assistente de serviço da web no Rational Application Developer tem uma opção para criar um serviço da web usando a recomendação candidata do W3C para a ligação de SOAP sobre JMS. O Rational Application Developer 7.54 é necessário. O exercício usou o Rational Application Developer incluído no Rational Software Architect for WebSphere Software v7.5.5.1,

O EJB é implementado no WebSphere Application Server a partir do Rational Application Developer como parte dessa tarefa.

Para criar o WSDL efetivamente usado na tarefa, deve-se primeiro configurar o perfil Liberty. Em seguida, é possível importar o WSDL do projeto Dinâmico da web na área de trabalho do Eclipse Galileo ou do serviço da web HTTP em execução implementado no perfil Liberty.

O WebSphere Application Server ainda podem estar em execução. Se não estiver, é possível iniciá-lo a partir da visualização Servidores no RAD.

Sobre esta tarefa

Nesta tarefa, você reimplementa o serviço do `StockQuoteAxis` da execução como um serviço JAX-RPC Axis executado pelo **`SimpleJavaListener`** usando o transporte do IBM MQ para SOAP para ser um serviço JAX-WS em execução no WebSphere Application Server usando o protocolo W3C SOAP sobre JMS.

Há duas partes para migrar o serviço do **`SimpleJavaListener`** para o WebSphere Application Server:

1. Gerar a interface de serviço da web a partir do WSDL para o serviço que está usando o assistente de serviço da web EJB descendente no Rational Application Developer.
2. Implementando o serviço importando a amostra SOAP do IBM MQ `StockQuoteAxis.java`.

Uma abordagem alternativa teria sido gerar o serviço de forma ascendente, a partir do `StockQuoteAxis.java`. No entanto, para ter certeza de que a interface para o serviço migrado seja idêntica, a abordagem descendente é melhor, pois usa o mesmo WSDL.

O serviço da web é desenvolvido para o contêiner EJB e não para o contêiner da web porque o suporte do JMS faz parte do contêiner EJB.

Procedimento

1. Inicie o Rational Application Developer e verifique se o WebSphere Application Server está em execução.
   a) Inicie o Rational Application Developer em uma nova área de trabalho.
   b) Abra a perspectiva Java EE.
   c) Abra a guia **Servidores** e verifique se o WebSphere Application Server está em execução.
      - Se não houver WebSphere Application Server 7.0 na visualização, clique com o botão direito na visualização> **Novo** > **Servidor**. Siga as opções no assistente para criar uma instância do WebSphere Application Server 7.0
      - Se o servidor estiver presente, mas não iniciado, clique na ponta da seta para iniciá-lo.
      - Para verificar as propriedades e obter acesso rápido aos registros do servidor, clique com o botão direito em **WebSphere Application Server 7.0 no host local** > **Propriedades** > **WebSphere Application Server**.
      - Para administrar o servidor, use um navegador externo e abra a URL, `http://localhost:9061/ibm/console/unsecureLogon.jsp`ou clique com o botão direito em **WebSphere Application Server 7.0 no host local** > **Executar console administrativo**.
      - A configuração padrão é publicar automaticamente. Muitas pessoas preferem implementar atualizações no servidor manualmente. Clique duas vezes em **WebSphere Application Server 7.0 no host local**e expanda a seção **Publicação** na janela **Visão Geral** . Clique em **Nunca publicar automaticamente**.
      - Outro padrão que você pode desejar alterar é limpar a caixa de seleção **Finalizar o servidor no encerramento do ambiente de trabalho** na janela **Visão geral**.
2. Criar os projetos JEE

   Deve-se criar um Projeto do aplicativo corporativo (EAR) e um projeto Enterprise Java Bean (EJB).

   a) **Arquivo** > **Novo** > **Projeto do aplicativo corporativo**. Dê ao projeto o nome `W3CJMSEAR` > **Concluir**.

   Os padrões devem identificar WebSphere Application Server 7.0 como o tempo de execução de destino e a versão EAR 5.0. A configuração padrão deve ser selecionada.

b) **Arquivo** > **Novo** > **Projeto EJB**. Denomine o projeto `W3CJMSEJB`. Selecione `W3CEARJMS` como o **Nome do projeto EAR** > **Avançar**.

A versão do módulo EJB padrão é 3.0 e a configuração padrão será usada novamente.

c) Limpe a caixa de seleção **Criar um módulo JAR do cliente EJB** > **Concluir**.

3. Gere e implemente o serviço da web EJB por meio do WSDL `StockQuoteAxis`.

a) **Executar** > **Ativar o Web Services Explorer**.

b) Selecione a página WSDL usando os ícones na janela **Web Services Explorer** > clique em **WSDL principal** no Navegador.

c) Na janela **Ações**, digite ou procure a URL do WSDL para `StockQuoteAxis.wsdl`.

Se você tiver o Liberty em execução com `StockQuoteAxis` implementado como um serviço HTTP, a URL será:

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

Se tiver o WSDL no sistema de arquivos, a URL poderá ser:

```
File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl
```

d) Clique na linha que contém a URL importada na árvore do Navegador.

É a linha imediatamente após **WSDL principal**, se for o primeiro WSDL importado para o Web Services Explorer.

e) Na janela **Ações**, clique em **Ativar assistente de serviço da web** > **Estrutura básica de serviço da web** > **Ir**

f) No assistente de serviço da web, selecione **Serviço da web EJB descendente**

Selecione ou verifique a Configuração usando informações da Tabela 190 na página 1326. Marque **Sobrescrever arquivos sem aviso** > **Avançar**.

Tabela 190. Configuração de serviço da web EJB descendente

Campo	**Value**
Servidor	`WebSphere Application Server 7.0`
Tempo de execução de serviço da web	`IBM WebSphere JAX-WS`
Projeto de serviço	`W3CJMSEJB`
Projeto de EAR de serviço	`W3CJMSEAR`
Configuração:	`No client generation`

g) Na página com o subtítulo **Especificar opções para criar um serviço da web descendente WebSphere JAX-WS EJB**, marque a caixa **Alternar para a ligação JMS**. Marque também **Ativar o estilo wrapper**, **Copiar WSDL para o projeto** e **Gerar descritor de implementação de serviço da web** > **Avançar**.

h) Na página com o título **Configuração de ligação do WebSphere JAX-WS JMS**, marque **Usar protocolo de interoperabilidade SOAP/JMS** e forneça valores da Tabela 191 na página 1326, deixando outros campos em branco > **Avançar**.

Tabela 191. Configuração de ligação de JMS JAX-WS WebSphere

Campo	**Value**
Destino de JMS	`queue`
Nome JNDI de destino:	`requestaxis`
Connection factory JMS	`qm1`

Tabela 191. Configuração de ligação de JMS JAX-WS WebSphere (continuação)	
Campo	**Value**
Nome para resposta	W3CJMSEAR
Configuração:	replyaxis

a) Na página com o título **Configuração do projeto de roteador do WebSphere JAX-WS**, digite qm1as no campo **Nome JNDI de ActivationSpec** > **Avançar**.

O RAD leva cerca de 30 segundos a um minuto para gerar e implementar o projeto.

b) Ignore as opções na página **Publicação de serviço da web** > **Concluir**.

4. Verifique o WSDL gerado.

Você solicitou que o WSDL específico do serviço fosse gerado e salvo no projeto.

a) No navegador Enterprise Explorer, abra a pasta **W3CJMSEJB** >> **ejbmodule** > **META-INF** > **wsdl**. Clique duas vezes em StockQuoteAxis.wsdl para abri-lo no editor WSDL.

Inspecione a ligação; você verá a url de JMS:

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qm1&targetService=StockQuoteAxis
```

5. Etapa opcional: ligue o EJB a SOAP sobre HTTP usando JAX-WS.

Fornecer duas ligações para o EJB fornece aos clientes a opção de ligações SOAP para chamar o serviço da web. Também fornece aos clientes o meio para consultar o servidor da web para obter seu WSDL, usando HTTP.

As etapas para ligar um EJB a SOAP sobre HTTP não estão incluídas como parte da tarefa.

6. Implemente e reimplemente StockQuoteAxis usando a amostra StockQuoteAxis.java

a) No navegador Enterprise Explorer, abra a pasta **W3CJMSEJB** > **Serviços** Clique duas vezes em StockQuoteAxisService para abrir a classe de implementação em um editor Java .

b) Abra o programa de amostra StockQuoteAxis.java na pasta *WebSphere MQ Installation directory*\tools\soap\samples\java\server > Selecione todos os métodos, mas não o nome da classe> **Copiar**

c) Em StockQuoteAxisSoapBindingImpl.java, selecione todos os métodos, mas não o nome da classe, e cole os métodos a partir de StockQuoteAxis.java.

d) Inclua uma instrução de impressão na saída no console do WebSphere Application Server quando o serviço for chamado.
Mude o método getQuote(String symbol):

```
public float getQuote(String symbol) {
System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
+ symbol);
return ((float) 55.25);
```

e) Corrija as importações: **Origem** > **Organizar importações** > **Salvar**.

f) Corrija os três erros devido à implementação não corresponder à interface.

Os erros se devem a três dos métodos em StockQuoteAxis.java lançarem exceções e o WSDL para o serviço não conter quaisquer mensagens de falha. O problema é diagnosticado como sendo uma incompatibilidade entre as assinaturas de método e as anotações de serviço da web do método.

Anote os métodos com @WebFault e gere o WSDL novamente ou mantenha a interface inalterada e remova as exceções.

Para manter a interface igual, remova os três throws exception das assinaturas de método > **Salvar**.

Como proceder a seguir

"Implementando em um cliente do Axis2 usando o W3C SOAP sobre JMS" na página 1367

Tarefas relacionadas

Desenvolvendo um serviço do .NET 1 ou 2 para o transporte de IBM MQ para SOAP usando o Microsoft Visual Studio 2012

Desenvolva o serviço da web SampleStockQuote para .NET 1 ou .NET 2 usando o Microsoft Visual Studio 2012.

Desenvolvendo os clientes de serviço da web do IBM MQ para o IBM MQ Transport for SOAP

Use seu ambiente de desenvolvimento normal para desenvolver os clientes de serviços da web para uso com o IBM MQ Transport for SOAP.

Antes de começar

Crie o serviço. É possível usar um dos exemplos em "Desenvolvendo serviços da web para transporte do IBM MQ para SOAP" na página 1320.

Faça escolhas sobre como você vai desenvolver, implementar e usar os clientes e o local para obter o WSDL para a geração do cliente.

Decida sobre sua abordagem de desenvolvimento de clientes e serviços para o IBM MQ Transport for SOAP.

Há duas abordagens.

1. Use as ferramentas de desenvolvimento padrão, desenvolva um serviço HTTP e o cliente e, em seguida, use a URL para o WebSphere MQ Transport for SOAP.
2. Use as ferramentas e as amostras fornecidas com o IBM MQ Transport for SOAP.

Se tomar a rota HTTP, será possível executar o serviço em um servidor HTTP e também executá-lo usando o IBM MQ Transport for SOAP. Para executá-lo usando o IBM MQ Transport for SOAP, configure o listener apropriado do IBM MQ for SOAP e configure os caminhos e descritores de implementação para executar o serviço. As ferramentas fornecidas pelo IBM MQ Transport for SOAP executam a configuração para você. Como alternativa, é possível configurar o ambiente para executar os listeners.

As ferramentas fornecidas com o IBM MQ Transport for SOAP são úteis para introduzir e aprender como implementar o transporte. Para o trabalho de produção, há benefícios em usar as ferramentas padrão e implementar o mesmo serviço acessível para diferentes transportes SOAP.

Decida sobre o tipo de cliente para desenvolver

Deve-se decidir qual tipo de cliente de serviço da web desenvolver. A opção depende se você conhece a interface de serviço e o endereço do serviço.

Se a interface for conhecida, use o Axis ou as ferramentas do .NET para gerar as classes do cliente proxy a partir da interface de serviço. As classes do cliente proxy tornam mais fácil gravar um cliente para chamar o serviço. Se o local do serviço for conhecido ao desenvolver o cliente, então use a interface de proxy estático. Se o local do serviço for mudado, por exemplo, se o serviço for reimplementado em um servidor de produção, use a interface de proxy dinâmico.

Se a interface de serviço não for conhecida no momento em que você desenvolver um cliente no Axis, será possível criar um cliente Dynamic Invocation Interface (DII) para o Axis 1.4. Um cliente DII usa uma interface genérica para chamar qualquer serviço. Para transmitir os parâmetros para um serviço específico corretamente, é necessário construir a interface com o serviço específico programaticamente. Construa a interface programaticamente no cliente ou ao carregar o WSDL para o serviço no cliente. No Axis2, é possível criar um cliente Dispatch. O cliente Dispatch usa um modelo de documento para descrever a solicitação do cliente, enquanto que um cliente DII usa um modelo de chamada. Ambos trabalham na construção da solicitação dinamicamente.

Obtenha o WSDL para o serviço

Exceto para o caso da interface de serviço que está sendo construída programaticamente, deve-se primeiro obter o WSDL de serviço para criar um cliente de serviço da web. O WSDL do serviço é obtido a partir de três origens diferentes:

1. Diretamente da implementação de serviço da web usando uma ferramenta como **java2wsdl** (Axis) ou **disco** (.NET).
2. Consultando o serviço da web usando a URL: *Web service http url* ?wsdl.
3. A partir de um arquivo, em um sistema de arquivos ou de um registro, como UDDI ou WebSphere Service Registry and Repository.

Nota: Se o serviço não estiver acessível usando HTTP, então a consulta WSDL não funcionará. O próprio serviço pode apenas estar disponível usando o IBM MQ Transport for SOAP.

O WSDL gerado pelo **amqwdeployWMQService** não é o mesmo que WSDL gerado usando **java2wsdl** ou **disco**. O WSDL gerado também é diferente para quaisquer WSDL você pode ter iniciado para criar o serviço "Top Down". No Axis, o descritor de implementação server-config.wsdd mapeia a mensagem SOAP produzida por um cliente para uma operação e serviço. **amqwdeployWMQService** gera um descritor de implementação diferente do Eclipse.

O WSDL usado para construir clientes depende de como o serviço é implementado:

Implementados usando o amqwdeployWMQService
Use o WSDL gerado pelo **amqwdeployWMQService**. Especifique o sinalizador -w e selecione o WSDL rpcLiteral. Para compatibilidade, é possível selecionar o WSDL rpcEncoded. o WSDL rpcEncoded funciona apenas com o .NET e clientes Axis 1.4.

Implementação manual usando SimpleJavaListener
Use um dos seguintes arquivos WSDL:

1. WSDL usado para definir o serviço ou armazenado em um repositório.
2. WSDL gerado a partir do serviço por **java2wsdl**.
3. WSDL consultado usando a URL *Web service http url* ?wsdl, se disponível a partir de um servidor HTTP. Você pode executar uma ferramenta como o Web Services Explorer para importar a definição de serviço diretamente no Eclipse.

Pode ser necessário mudar o URI para o serviço. Mude-o a partir do endereço de serviço HTTP para o URI para o IBM MQ Transport for SOAP.

Implementação manual usando amqSOAPNETListener.
Use um dos seguintes arquivos WSDL:

1. WSDL usado para definir o serviço ou armazenado em um repositório.
2. WSDL obtido a partir da classe de serviço do .NET (.asmx). usando o **disco**.
3. WSDL consultado usando a URL *web services http url* ?wsdl, se disponível. Você pode executar uma ferramenta como o Web Services Explorer para importar a definição de serviço diretamente no Eclipse.
4. WSDL obtido executando **amqswsdl** com relação à classe de serviço do .NET (.asmx).

Pode ser necessário mudar o URI para o serviço. Mude-o a partir do endereço de serviço HTTP para o URI para o IBM MQ Transport for SOAP.

Implementado para Windows Communication Foundation
Obtenha o WSDL de serviço usando a URL *Web service http url* ?wsdl O serviço deve ser definido com a configuração de comportamento serviceMetaData como parte da definição de serviço.

Implementação para uma plataforma de servidor diferente.
Siga a orientação fornecida com a plataforma sobre como obter o serviço correto WSDL.

Sobre esta tarefa

Desenvolva clientes usando ferramentas de desenvolvimento padrão. As tarefas a seguir ilustram como construir clientes para .NET 1 e 2, Axis 1.4 (JAX-RPC) e Axis2 (JAX-WS). Para Windows Communication Foundation, consulte os links de tarefas relacionadas.

Desenvolvendo um cliente JAX-RPC para transporte do WebSphere para SOAP usando o Eclipse

Desenvolva um cliente de serviço da web Axis 1.4 para execução usando o transporte do IBM MQ para SOAP.

Antes de começar

Deve-se ter o serviço disponível. É necessário ter um servidor de aplicativos em execução no Eclipse que suporte serviços da web do Axis 1.4. Nesta tarefa, usamos o perfil Liberty disponível gratuitamente. Você também pode usar o Tomcat 6, que é um servidor de aplicativos de software livre menor.

Sobre esta tarefa

A tarefa mostra o desenvolvimento de três tipos de cliente para o serviço StockQuoteAxis de amostra usando o Eclipse em execução no Windows. Os clientes são um cliente estático e um dinâmico desenvolvidos usando o proxy de cliente e um cliente DII.

Duas abordagens alternativas para gerar os proxies de cliente a partir do WSDL são ilustradas:

1. Gerar proxies de cliente usando **amqwdeployWMQService**.
2. Importar WSDL para o Eclipse e usar o assistente de serviço da web para gerar os proxies de cliente.

Procedimento

1. Inicie o Eclipse IDE for Java EE Developers.
2. Crie um projeto Java chamado `StockQuoteAxisClient`:
   a) Alterne para a perspectiva Java > **Arquivo** > **Novo** > **Projeto Java**. No campo **Project name** do tipo **Criar um Projeto Java** , `StockQuoteAxisEclipseClient`. Certifique-se de que o ambiente de execução seja **J2SE1-1.4** ou **J2SE-1.5** > **Avançar**.
   b) Na página **Configurações Java**, selecione a guia **Bibliotecas** > **Incluir JARs externos...**
   c) Procure *MQ_INSTALLATION_PATH*`/java/lib` e selecione todos os arquivos `.jar` > **Abrir**.
   MQ_INSTALLATION_PATH é o diretório no qual IBM MQ está instalado.
   d) Procure *MQ_INSTALLATION_PATH*`/java/lib/soap` e selecione todos os arquivos `.jar` > **Abrir**. Deve-se ter instalado o `axis.jar` a partir da mídia de instalação do IBM MQ neste diretório.
   MQ_INSTALLATION_PATH é o diretório no qual IBM MQ está instalado.
   e) A guia **Biblioteca** agora faz referência a todos os arquivos `.jar` necessários para construir o cliente > **Concluir**.
3. Siga uma destas duas abordagens para criar proxies no Eclipse para o serviço da web StockQuoteAxis de amostra:
   - Gere os proxies de cliente usando **amqwdeployWMQService**.
     a. Crie um gerenciador de filas. Para a tarefa, crie `QM1` como o gerenciador de filas padrão.
     b. Crie um diretório de trabalho, `samples`. Copie o programa de amostra `StockQuoteAxis.java` para `samples/soap/server`.
     c. Modifique `amqwsetcp.cmd` em *MQ_INSTALLATION_PATH*`/bin` para incluir o diretório atual no caminho de classe. *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado.
     d. Abra uma janela de comando em `samples` e execute o comando **amqwsetcp** modificado
     e. Crie WSDL para o serviço StockQuoteAxis executando o comando,

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWsdl
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

Não se esqueça: Use "/"em vez de "." ou "\" ao usar comandos Java .

Sugestão: Em vez de importar os proxies gerados no Eclipse, é possível pode importar o WSDL gerado a partir de `.samples/generated`. As proxies resultantes diferem de duas maneiras:

i) Os nomes de pacotes são diferentes, o que é possível refatorar.

ii) Os proxies gerados pelo Eclipse incluem uma classe auxiliar adicional, `StockQuoteAxisProxy.java`

f. Crie os proxies de cliente para o serviço StockQuoteAxis executando o comando:

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

g. Importe os proxies de cliente para StockQuoteAxisClient:

i) Clique com o botão direito em **StockQuoteAxisClient\src** > Selecionar **Sistema de Arquivos** > **Avançar** > **Procurar ...** > localizar a pasta `.\samples\generated\client\remote\soap\server` > **OK**.

ii) Marque **servidor** na página **Importar** > **Concluir**.

h. Refatore o nome do pacote para `soap.server`.

i) Clique com o botão direito no pacote que contém os proxies de cliente > **Refatorar** > **Renomear**. Digite o **New name**: `soap.server` > deixe os padrões selecionados para as outras opções > **OK**. Todos os erros são corrigidos.

- Gere os proxies de cliente usando o Eclipse.

Você tem uma opção de maneiras de obter o WSDL para o serviço. Neste exemplo, o serviço foi implementado no perfil Liberty e você obtém o WSDL a partir do servidor da web.

a. No Eclipse, alterne para a perspectiva da web e verifique se o perfil Liberty está em execução e StockQuoteAxis está implementado e sincronizado.

b. Importe o WSDL para o Web Services Explorer:

i) Clique no ícone **Web Services Explorer** na barra de ação ou clique em **Executar** > **Ativar o Web Services Explorer**.

ii) Clique no ícone de página WSDL no Web Services Explorer para alternar para a página WSDL.

iii) Clique em **WSDL Principal** na janela do Navegador do Web Services Explorer.

iv) Digite a URL do serviço da web, seguida por ?WSDL. A URL para StockQuoteAxis, implementada no perfil Liberty, é:

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

c. Gere os proxies de cliente:

i) No navegador Web Services Explorer, clique em **http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl**.

ii) Na janela **Ações**, clique em **Ativar assistente de serviço da web** > deixe **Cliente de serviço da web** selecionado > **Ir**.

iii) Na primeira página do assistente, clique no link do projeto **Cliente** na configuração > Selecione o projeto do cliente **StockQuoteAxisClient** > **OK**.

Sugestão: A janela do assistente pode perder o foco. É necessário trazê-la de volta ao foco manualmente.

iv) O tempo de execução do serviço da web deve ser o Apache Axis para gerar um cliente JAX-RPC.

v) Clique em **Concluir**.

vi) Mude a URL estática do serviço para apontar para o endereço do transporte do IBM MQ para SOAP para o serviço StockQuoteAxis. Você pode optar por ignorar essa etapa, até que você tenha testado o cliente com um servidor HTTP.

a) Abra `StockQuoteAxisServiceLocator.java` e localize a declaração para `StockQuoteAxis_address`.

b) Mude a URL para

```
"jms:/queue?destination=REQUESTAXIS
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
&amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

Sugestão: O Eclipse transforma & automaticamente em `&`, e o inverso, quando você copia e cola sequências para o código `.java`.

d. Crie três classes do cliente Java, cada uma com um método principal:

i) Crie um pacote. Clique com o botão direito em **StockQuoteAxisClient/src** > **Novo pacote**. Dê a ele o nome `soap.client` > **Concluir**.

ii) Selecione **soap.client** > **Novo** > **Classe**. Dê à classe o nome `SQAStaticClient` > Marque **public static void main(string [] args)** > Concluir

iii) Repita o procedimento para criar `SQADynamicClient.java` e `SQADIIClient.java`

e. Escreva o código do cliente.

Figura 173 na página 1336 a Figura 177 na página 1337 fornecem exemplos dos três estilos de código do cliente. Os exemplos usam uma URL HTTP para testar o cliente usando o serviço StockQuoteAxis implementado em um servidor HTTP. Para executar os clientes com relação ao serviço StockQuoteAxis implementado usando o transporte do IBM MQ para SOAP, mude a URL para:

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.Nojndi
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

– Figura 173 na página 1336 e Figura 175 na página 1336 usam o proxy gerado pelo Eclipse, que tem a classe auxiliar StockQuoteAxisproxy adicional que facilita um pouco a codificação.

– Figura 174 na página 1336 e Figura 176 na página 1337 usam o proxy gerado por **amqwdeployWMQService**.

– Figura 177 na página 1337 não usa nenhuma classe de proxy.

Cada um dos clientes chama `com.ibm.mq.soap.Register.extension()` para vincular ao transporte do IBM MQ para SOAP. A extensão é registrada no descritor de implementação do cliente. A implementação do cliente no Axis 1.4 é descrita em "Implementando um cliente de serviço da web para o Axis 1.4 para usar o transporte do IBM MQ para SOAP" na página 1362.

f. Execute os clientes, enviando a solicitação SOAP para StockQuoteAxis hospedado pelo servidor WebSphere Application Server Community Edition configurado na área de trabalho.

i) Verifique se o servidor está em execução, StockQuoteAxis está implementado e sincronizado.

ii) Selecione ou abra o cliente que deseja testar > Clique em **Executar** na barra de ação. Como alternativa, clique no ícone Executar verde ou clique no cliente no navegador > **Executar como** > **Executar configurações...**. Configure os parâmetros que você precisa para executar o cliente.

g. Execute o cliente usando o transporte do IBM MQ para SOAP.

O procedimento usa **amqwdeployWMQService** para implementar o serviço e funciona somente com o cliente que usa o WSDL ou proxies construídos por **amqwdeployWMQService**.
Para executar o cliente usando o WSDL original ou proxies construídos pelo Eclipse, implemente o serviço com seu descritor de implementação construído pelo Eclipse. Inicie **SimpleJavaListener** manualmente usando o nome de ligação da porta de serviço como `targetServiceName`.

i) Siga as instruções em "Implementando um Serviço no Axis 1.4 para usar para transporte do WebSphere para SOAP usando amqwdeployWMQService" na página 1356 para implementar o serviço no listener IBM MQ Simple Java SOAP. A implementação de serviço funciona somente para o cliente que está usando os proxies de WSDL ou de cliente construídos por **amqwdeployWMQService**.

ii) Em uma janela de comando, execute **amqwclientconfig** para criar o arquivo do descritor de implementação do cliente, `client-deploy.wsdd`.

iii) Importe `client-deploy.wsdd` para a raiz do projeto Java que você deseja testar usando o transporte do IBM MQ para SOAP.

   a) Clique com o botão direito no projeto Java **StockQuoteAxisEclipseClient** > **Importar** > **Sistema de arquivos** > **Avançar** > **Procurar...**

   b) Procure o diretório que contém `client-deploy.wsdd` > **Abrir** > Selecione o diretório na página do assistente **Importar** > marque `client-deploy.wsdd`.

   c) Verifique se **Na pasta:** tem `StockQuoteAxisEclipseClient` inserido > **Concluir**.

iv) Confirme se o diretório ativo para executar um aplicativo Java neste projeto é o diretório `StockQuoteAxisEclipseClient`:

   Clique com o botão direito do mouse no Java projeto **StockQuoteAxisEclipseClient** > **Executar como ....** > **Configurações de Execução ...** > Selecione a guia **(x) = Argumentos** > Verifique se no Diretório Ativo o botão de opções **Padrão** está marcado e se o caminho é `StockQuoteAxisEclipseClient` Como alternativa, faça uma das escolhas a seguir para selecionar um local diferente ou arquivo que contém a configuração do cliente:

   – Marque **Outro:** > digite um caminho de diretório de sua preferência.

   – Na janela **Argumentos da VM** , digite `-Daxis.ClientConfigFile=` *`full path to client deployment descriptor file`*

v) Certifique-se de que a URL esteja configurada para apontar para o serviço implementado usando o transporte do IBM MQ para SOAP. Execute o cliente conforme descrito na etapa ii.

Sugestão: Geralmente, você pode encontrar um desses erros:

i) `Exception: No client transport named 'jms' found!.`

ii) Um erro de conexão do JMS.

iii) `Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java`

iv) `Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis`

Explicações:

i) `client-config.wsdd` não foi localizado ou não inclui a linha `<transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/>` em `client-config.wsdd`

ii) Possivelmente um problema de caminho de construção, sem incluir arquivos .jar em *`MQ_INSTALLATION_PATH`*`/java/lib`. *`MQ_INSTALLATION_PATH`* é o diretório no qual IBM MQ está instalado.

iii) Problema de implementação de serviço, com `server-config-wsdd` ou com parâmetros passados para **SimpleSoapListener**.

iv) Incompatibilidade entre o descritor de implementação e a implementação do serviço.

Se você estiver tendo dificuldade para executar o cliente no Eclipse, tente usar uma janela de comando:

i) Alterne para o diretório `StockQuoteAxisEclipseClient\bin` na árvore de diretórios da área de trabalho.

ii) Execute **amqwsetcp** e **amqwclientconfig**

iii) Execute `java soap/client/SQAStaticClient`.

Clientes de serviço da web JAX-RPC de amostra

Os clientes de serviço da web Java de amostra enviados com o IBM MQ são instalados no *MQ_INSTALLATION_PATH* `\tools\soap\samples\java\clients`. *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado.

SQAxis2Axis.java
SQAxis2Axis.java, Figura 170 na página 1334, é um cliente proxy dinâmico para chamar o serviço do StockQuoteAxis. É possível substituir a URL do serviço, compilado no proxy dinâmico, fornecendo uma URL na linha de comandos.

SQAxis2DotNet.java
SQAxis2DotNet.java, Figura 171 na página 1335, é um cliente proxy dinâmico para chamar o serviço do StockQuoteDotNet. É possível substituir a URL do serviço, compilado no proxy dinâmico, fornecendo uma URL na linha de comandos.

WsdlClient.java
WsdlClient.java, Figura 172 na página 1335, é um cliente de chamada dinâmica para chamar o serviço StockQuoteDotNet ou StockQuoteAxis. O cliente chama o serviço StockQuoteAxis por padrão. Inclua a opção da linha de comandos -D, chame o serviço StockQuoteDotNet e -w para fornecer uma porta diferente para aquela em `.\generated\StockQuoteDotNet_Wmq.wsdl`

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
   public static void main(String[] args) {
      com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
         StockQuoteAxis service = null;
         if (args.length == 0)
            service = locator.getSoapServerStockQuoteAxis_Wmq();
         else
            service = locator.getSoapServerStockQuoteAxis_Wmq(
                new java.net.URL(args[0]));
         System.out.println("Response: " + service.getQuote("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
  }
```

Figura 170. SQAxis2Axis.java

```
public class SQAxis2DotNet {
public static void main(String[] args) {
   com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteDotNet locator = new StockQuoteDotNetLocator();
         StockQuoteDotNetSoap_PortType service = null;
         if (args.length == 0)
            service = locator.getStockQuoteDotNetSoap();
         else
            service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                  args[0]));
         System.out.println("Response: " + service.getQuoteDOC("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Figura 171. SQAxis2DotNet.java

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdlClient {
public static void main(String[] args) {
      String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
      try {
         Register.extension();
         Options opts = new Options(args);
         if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
         } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
         }
         if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
         System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
               + " resolving uri to ...");
         QName servQN = new QName(namespace, wsdlService);
         QName portQN = new QName(namespace, wsdlPort);
         Service service = ServiceFactory.newInstance().createService(
               new URL(wsdlSource), servQN);
         Call call = (Call) service.createCall(portQN, "getQuote");
         wsdlTargetURI = call.getTargetEndpointAddress().toString();
         System.out.println("   '" + wsdlTargetURI + "'");
         Object ret = call.invoke(new Object[] { "XXX" });
         System.out.println("Response: " + ret);
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
         e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Figura 172. WsdlClient.java

Os clientes de exemplo usados nesta tarefa:

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
      System.out.println("Static client synchronous result is:"
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 173. Cliente estático usando proxy gerado pelo Eclipse

```
package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
      System.out.println("Static client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 174. Cliente estático usando o proxy gerado por amqwdeployWMQService

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 175. Cliente dinâmico usando proxy gerado pelo Eclipse

```
package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL sqaURL = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq(sqaURL);
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 176. Cliente dinâmico usando o proxy gerado por amqwdeployWMQService

```
package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQADIIClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL wsdl = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
      Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
          new QName("http://server.soap", "StockQuoteAxisService"));
      Call SQACall = SQAService.createCall(new QName("http://server.soap",
          "StockQuoteAxis"), "getQuote");
      System.out.println("DII client synchronous result is "
          + SQACall.invoke(new Object[] { "ibm" }));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figura 177. Cliente DII (nenhum proxy)

Tarefas relacionadas

Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse
Desenvolva um cliente de serviço da Web do Axis2 para executar usando o transporte de IBM MQ para SOAP. Os clientes Axis2 de amostra fornecidos com o transporte de IBM MQ para SOAP são listados e o comando **wsimport** usado para gerar proxies.

Desenvolvendo um cliente .NET 1 ou 2 para o transporte do WebSphere para SOAP usando o Microsoft Visual Studio 2012
Desenvolva um cliente do serviço da web .NET 1 ou 2 para executar usando o transporte de IBM MQ para SOAP.

Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse

Desenvolva um cliente de serviço da Web do Axis2 para executar usando o transporte de IBM MQ para SOAP. Os clientes Axis2 de amostra fornecidos com o transporte de IBM MQ para SOAP são listados e o comando **wsimport** usado para gerar proxies.

Antes de começar

Obtenha as bibliotecas Axis2 e configure um ambiente de desenvolvimento e teste para executar o cliente.

Nota: A nomenclatura de versões e liberações usada pelo Axis causa confusão. Normalmente o Axis 1.4 refere-se à implementação JAX-RPC, e o Axis2 à implementação JAX-WS.

Axis 1.4 é um nível de versão. Se você procurar por Axis 1.4 na Internet, será levado para http://ws.apache.org/axis/. A página contém uma lista de versões precedentes do Axis (1.2, 1.3) e a liberação final de 22 de abril de 2006 do Axis 1.4. Há liberações mais recentes do Axis 1.4 que corrigem erros, mas todas elas são conhecidas como Axis 1.4. Essa é dessas liberações para correção que é enviada com o IBM MQ. Para o Axis 1.4, use a versão do `axis.jar` que é enviada com o IBM MQ em vez daquela que pode ser obtida a partir de http://ws.apache.org/axis/.

O Web site do Axis também encaminha para o Axis 1.1, para encaminhar para todas as versões daquele que, normalmente, é chamado de Axis 1.4. O Axis 1.2 é usado para encaminhar para aquele que normalmente é chamado de Axis2.

Axis 1.5 não é uma liberação mais recente do Axis 1.4, é uma liberação do Axis2. Se procurar o Axis 1.5, você será direcionado para http://ws.apache.org/axis2/.https://ws.apache.org/axis2/download.cgi contém uma lista de versões de liberação de Axis2, rotuladas como 0,9 1.5.1 (e incluindo, de maneira confusa, a version 1.4). A versão de liberação do Axis2 para usar com o transporte do IBM MQ para SOAP é a 1.4.1. Faça o download do Axis2 1.4.1 em http://ws.apache.org/axis2/download/1_4_1/download.cgi.

É possível optar por gerar proxies para os clientes de serviço da Web para o transporte de IBM MQ para SOAP usando **`wsimport`** ou o conjunto de ferramentas fornecido com um IDE. O Eclipse IDE para Java EE Developer 3.5 SR1 usa **`wsdl2java`**. **`wsimport`** é fornecido com Java 6. É possível usar o Java 5 para executar proxies do cliente gerados com **`wsimport`** ou **`wsdl2java`**

Os clientes Axis2 do serviço da Web de amostra fornecidos com o transporte do IBM MQ para SOAP foram desenvolvidos usando **`wsimport`**; consulte “Clientes Axis2 de amostra” na página 1343.

A tarefa a seguir demonstra como gerar e usar os proxies produzidos pelo assistente de serviços da Web que é empacotado com o Eclipse IDE para Java EE Developers. Os clientes de amostra mostram como usar os proxies produzidos pelo **`wsimport`**.

Para usar o assistente de serviços da web, deve-se incluir um servidor de aplicativos que suporte o Axis2 no ambiente de trabalho. As etapas mostram como configurar o perfil Liberty para suportar o Axis2 usando o ambiente de trabalho.

1. Configure o servidor de aplicativos usado no Eclipse IDE para Java EE Developers para suportar o Axis2. Neste exemplo, configure o perfil Liberty.
   a. Abra as preferências da área de trabalho para configurar o servidor: abra **Janela** > **Preferências**.
   b. Verifique se o JRE instalado é o `Java50`: Clique em **JREs instalados**.
   c. Inclua o perfil Liberty como o servidor:
   d. Inclua Axis2: clique em **Serviços da web** > **Preferências de Axis2**. Na guia **Axis2 Tempo de execução** > **Procurar ...** Abra o diretório que contém muitos arquivos jar `Axis2` > **Aplicar**.
   e. Associe o Liberty ao Axis2: clique em **Serviços da web** > **Servidor e tempo de execução**. Em **`Server`** selecione **IBM Liberty Server**e, em **`Web service runtime`**, selecione **Apache Axis2** > **Aplicar** > **OK**
   f. Inicie o servidor: Abra a perspectiva da Web e abra a visualização Servidores. Clique com o botão direito na visualização Servidores > **Novo** > **Servidor**. **Servidor Liberty da IBM** é selecionado e configurado > **Concluir**. Inicialize o servidor.
2. Verifique se você implementou o serviço StockQuoteAxis no Liberty para executar o assistente de serviço da web.
3. Para testar o serviço com o transporte de IBM MQ para o serviço SOAP, implemente o serviço em um transporte do IBM MQ para listener SOAP para o Axis 1.4; consulte o perfil Liberty.

Sobre esta tarefa

O Eclipse IDE para Java EE Developers usa Java50 e o assistente de serviços da Web para gerar as classes de proxy para o serviço. As classes de proxy são diferentes das classes criadas pela ferramenta **wsimport** fornecida com Java 6. Uma abordagem alternativa é gerar as classes de proxy usando **wsimport** e importar os pacotes que ele cria em seu Eclipse Java EE IDE for Web Developers.

O assistente de serviços da Web no Eclipse IDE para Java EE Developers constrói um cliente de serviço da Web em um projeto da Web. É possível executar o cliente como um aplicativo simples do Java; ele não requer um servidor de aplicativos. É possível também transferir o código para um projeto do Java e configurar o caminho de construção para incluir os arquivos JAR do Axis2.

Procedimento

1. Crie um projeto da Web em novo projeto Corporativo:
   a) Com nada selecionado no Explorador de Projetos > Clique com o botão direito no espaço em branco > **Novo** > **Projeto do aplicativo corporativo** > Dê a ele o nome `StockQuoteAxis2EAR` > **Concluir**. Responda No para a janela fornecendo a opção de abrir a perspectiva Java EE.

      Os padrões são configurados para usar o Liberty.

   b) Clique com o botão direito em `StockQuoteAxis2EAR` > **Novo** > **Projeto dinâmico da web**. Dê ao projeto o nome `StockQuoteAxis2WebClient` > Marque a caixa de associação EAR para incluir o projeto em **StockQuoteAxis2EAR**. O Liberty é selecionado como o tempo de execução de destino.

   c) Na seção Configuração da página **Novo Projeto Dinâmico da Web** > **Modificar ...** > Verificar o aspecto do projeto de serviços da web Axis2 ... **Módulo da Web Dinâmico 2.5**, **Java 6.0**e **Liberty** já estão verificados > **OK** > **Concluir** Responda No para a janela fornecendo a opção de abrir a perspectiva Java EE.
2. Importe o WSDL para o serviço na área de trabalho e gere o proxy de cliente:

   Neste exemplo, o documento WSDL contém a ligação de serviços HTTP e se torna o destino para o proxy de cliente da Web estático. É possível modificar a URL na ligação de serviço da Web para apontar para o transporte de IBM MQ para a URL SOAP antes de gerar o proxy de cliente. O proxy de cliente da Web estático é, então, o serviço que é implementado no transporte de IBM MQ para SOAP.

   a) Ative o Web Services Explorer: use o ícone na barra de ação ou **Executar** > **Ativar o Web Services Explorer**.

   b) Selecione o explorador WSDL clicando no ícone WSDL na janela **Web Services Explorer** > Clique em **WSDL principal** na janela Navegador > Digite a URL do arquivo WSDL StockQuoteAxis > **Ir**. Neste exemplo, obtenha o WSDL diretamente do serviço HTTP: `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`

   c) No Navegador, clique na linha com a URL do serviço da web. Na janela **Ações**, clique em **Importar WSDL para o ambiente de trabalho** > Selecione um **StockQuoteAxis2WebClient** como o **Projeto de ambiente de trabalho** > Digite o **Nome do arquivo WSDL**, `StockQuoteAxisHTTP.wsdl` > **Ir**.

   d) Clique com o botão direito em **StockQuoteAxisHTTP.wsdl** > **Serviços da web** > **Gerar cliente**. Verifique se as informações de configuração sobre a página de serviços da Web do assistente são como segue: Servidor: IBM Servidor Liberty, tempo de execução de serviço da Web: Apache Axis2, projeto do cliente: StockQuoteAxis2WebClient, Projeto EAR do cliente: StockQuoteAxisEAR. Para corrigir a configuração, clique nas linhas que estão erradas.

   e) Clique em **Avançar** > verifique as configurações de geração de código > **Concluir**.

      Observe que um novo pacote, `soap.server`, é criado e ele contém os proxies requeridos.
3. Configure o projeto para executar o transporte de IBM MQ para SOAP como o transporte de JMS.

   O transporte de IBM MQ para SOAP fornece um `transportSender`, mas nenhum `transportReceiver`. Em outras palavras, o transporte de IBM MQ para SOAP suporta clientes do Axis2. Atualmente, ele não suporta serviços do Axis2.

   a) No projeto **StockQuoteAxis2WebClient** , clique com o botão direito em `WebContent\WEB-INF\conf\axis2.xml` > **Abrir com ...** > **Editor XML**.

b) Procure o último `transportSender` (no final do arquivo) e localize a linha comentada JMS `transportSender` > Clique com o botão direito na linha> **Incluir antes ...** > **transportSender**.

c) Clique com o botão direito em **transportSender** > **Incluir atributo** > **Nome** > Clique com o botão direito em **transportSender** > **Incluir atributo** > **Classe**.

d) Clique com o botão direito em **Nome** > **Editar atributo** > Digite o **Valor:** `jms`

e) Clique com o botão direito em **Classe** > **Editar Atributo** > Digite o **Valor:** `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender`.> Salvar.

f) Inclua `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender` no caminho de construção: Clique com o botão direito em **StockQuoteAxis2WebClient** > **Caminho de Construção** > **Configurar Caminho de Construção ...** > Clique na guia **Bibliotecas** > **Incluir JARs Externos ...**. Selecione todos os JARs em *MQ_INSTALLATION_PATH*`\java\lib` > **OK**.

MQ_INSTALLATION_PATH é o diretório no qual IBM MQ está instalado.

4. Crie um cliente estático assíncrono, teste-o usando HTTP e, em seguida, converta o proxy para executar o cliente estático usando o transporte de IBM MQ para SOAP.

a) Clique com o botão direito em **Recursos Java: src** > **Novo** > **Pacote** > Dê ao pacote o nome `soap.client` > Concluir

b) Clique com o botão direito em **soap.client** > **Nova** > **Classe** > Dê à classe o nome `SQA2StaticClient` > **Concluir**.

c) Substitua a classe pelo código a seguir, em seguida, clique em **Salvar**.

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("Response is: "
          + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

5. Teste o cliente com o serviço StockQuoteAxis implementado para o Liberty e com o transporte de IBM MQ para SOAP.

a) No Explorador de Projetos, clique com o botão direito em **SQA2StaticClient** >> **Executar como ...** > **Java Aplicativo**.

O resultado, `Response is 55.25` , aparece na visualização de Console Também é possível selecionar a janela do console Liberty na visualização Console e ver a saída no servidor Liberty, `StockQuoteAxis called with parameter: ibm`.

b) O proxy foi construído com o endereço de serviço, `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis`, e, portanto, o cliente estático chama o serviço em execução no HTTP. É possível mudar o cliente estático para chamar o serviço usando o IBM MQ transporte para SOAP. As instruções a seguir mudam o endereço de serviço em `StockQuoteAxisServiceStub.java` sem reconstruir o proxy e configuram os parâmetros de tempo de execução `SQA2StaticClient` para carregar `axis2.xml`. Configure `axis2.xml`, configura o Axis2 para usar o IBM MQ transporte para SOAP.

c) Abra `StockQuoteAxisServiceStub.java` >
Substitua as duas ocorrências de `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis` por,

```
jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
```

```
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

d) Se você executar SQA2StaticClient agora, ele lançará uma exceção porque não localizou um transportSender configurado para JMS
A exceção é:

```
Exception: null java.lang.NullPointerException at
soap.server.StockQuoteAxisServiceStub.getQuote(StockQuoteAxisServiceStub.java:547)
at soap.client.SQA2StaticClient.main(SQA2StaticClient.java:11)
```

e) No Explorador de Projetos, clique com o botão direito em **SQA2StaticClient** > **Executar como ...** > **Configurações de Execução ...**. Alterne para a guia **(x)= Argumentos** e, na área de entrada **Argumentos da VM**, digite o caminho para o arquivo axis2.conf > **Aplicar** > **Executar**.
O argumento da VM é: -Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml. Ou é possível fornecer um caminho padrão para o arquivo de configuração do Axis2.

f) Execute SQA2StaticClient novamente. Nesta execução, você está usando o transporte de IBM MQ para SOAP. Confirme-o marcando que não há nenhuma saída nova no console do Liberty. Abra o console ou a janela de comandos que está associada ao ListenerJavaSimples e a saída lá é StockQuoteAxis called with parameter: ibm

6. Crie um cliente dinâmico para HTTP e o transporte de IBM MQ para SOAP e teste-o.

a) Clique com o botão direito em **soap.client** > **Nova** > **Classe** > Dê à classe o nome SQA2DynamicClient > **Concluir**.

b) Substitua a classe pelo código a seguir, em seguida, clique em **Salvar**.

Figura 178. SQA2DynamicClient.java

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
       System.out.println("JMS sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
     } catch (Exception e) {
       System.out.println("Exception: " + e.getMessage());
       e.printStackTrace();
     }
   }
 }
```

c) Crie uma configuração Executar para SQA2DynamicClient.java e inclua o caminho em axis2.xml:
-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml

d) Execute SQA2DynamicClient. Verifique a saída do console para o SQA2DynamicClient, Liberty e **SimpleJavaListener**.

7. Crie um cliente assíncrono e acesse o resultado em um manipulador de retorno de chamada e no encadeamento do programa principal.

Os proxies de clientes assíncronos criados pelo assistente de serviço da Web para Eclipse Java EE IDE para Web Developers diferem dos proxies criados por **wsimport**. Os tipos genéricos **wsimport** proxies usam `Future`, `Responsee` `AsyncHandler` .

O assistente de serviço da Web para Eclipse Java EE IDE para Web Developers cria uma classe abstrata `StockQuoteAxisServiceCallbackHandler`. Deve-se estender `StockQuoteAxisServiceCallbackHandler` e criar um manipulador de retorno de chamada.

a) Clique com o botão direito em **soap.client** > **Nova** > **Classe** > Dê à classe o nome `SQA2CallbackHandler` > **Concluir**.

b) Substitua a classe pelo seguinte código.

```
package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
  private boolean complete = false;
  SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
  }
  public void receiveResultgetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
  }
  public boolean isComplete() {
    return complete;
  }
}
```

c) Clique com o botão direito em **soap.client** > **Nova** > **Classe** > Dê à classe o nome `SQA2AsyncClient` > **Concluir**.

d) Substitua a classe pelo seguinte código.

Figura 179. `SQA2AsyncClient.java`

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      SQA2CallbackHandler callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      do {
        System.out.println("Waiting for HTTP callback");
        Thread.sleep(2000);
      } while (!callback.isComplete());
      System.out.println("HTTP poll: "
          + ((GetQuoteResponse) (callback.getClientData()))
                .getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
      System.out.println("JMS Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      while (!callback.isComplete()) {
```

```
        System.out.println("Waiting for JMS callback");
        Thread.sleep(2000);
      }
      System.out.println("JMS poll: "
        + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
     }
   }
 }
```

A saída do console é a seguinte:

```
HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
```

Clientes Axis2 de amostra

Os proxies de amostra são gerados usando a ferramenta **wsimport** que é empacotada com Java 6. Seis amostras são fornecidas:

1. DynamicProxyClientSync.java
2. DynamicProxyClientAsyncPolling.java
3. DynamicProxyClientAsyncCallback.java
4. DispatchClientSync.java
5. DispatchClientAsyncPolling.java
6. DispatchClientAsyncCallback.java

As amostras do cliente são geradas para o servidor StockQuoteAxis de amostra. Gere o WSDL com o comando **amqwdepoyWMQServer**, especificando o comutador -w para selecionar o estilo `rpcLiteral`. Use o comando a seguir para gerar os proxies para as amostras:

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

Figura 180. DynamicProxyClientSync.java

```
package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientSync");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      service.getQuoteOneWay("48");
      System.out.println(" > getQuoteOneWay has returned");

      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      float result = service.getQuote("48");
```

```
      System.out.println(" > getQuote has returned result of " + result);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figura 181. DynamicProxyClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncPolling");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously by
polling...");
      Response<Float> response = service.getQuoteAsync("49");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** Retrieve the result **/
      try {
        Float result = response.get();
        System.out.println(" > getQuoteAsync call has returned result of " + result);
      }
      catch (CancellationException ce) {
        // processing was cancelled via response.cancel()
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
```

```
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figura 182. DynamicProxyClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncCallback");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = service.getQuoteAsync("50", handler);
      System.out.println(" > Invoke call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<Float> response) {
    try {
      Float result = response.get();
      System.out.println(" > Async Handler has received a result of " + result);
```

```
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

Figura 183. DispatchClientSync.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientSync");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service **/
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /************************************
       * Create OneWay SOAPMessage request.
       ************************************/
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a OneWay SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
```

```
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      dispatch.invokeOneWay(request);
      System.out.println(" > getQuoteOneWay call has returned");

      /****************************************
       * Create Request Reply SOAPMessage request.
       ****************************************/
      mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a Request Reply SOAP Message");
      request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      part = request.getSOAPPart();
      env = part.getEnvelope();
      header = env.getHeader();
      body = env.getBody();

      /** Construct the message payload **/
      operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
      value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      SOAPMessage ans = dispatch.invoke(request);
      System.out.println(" > getQuote call has returned");

      /** Retrieve the result **/
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      System.out.println("Parsing SOAP response...");
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
```

```
    } // end of catch block
  }
}
```

Figura 184. DispatchClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncPolling");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation asynchronously by
polling...");
      Response<SOAPMessage> response = dispatch.invokeAsync(request);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
```

```
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** retrieve the result **/
      SOAPMessage ans = response.get();
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");

    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figura 185. DispatchClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncCallback");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
```

```
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload. **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint. **/
      DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

      System.out
          .println("Invoking getQuote Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = dispatch.invokeAsync(request, handler);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<SOAPMessage> response) {
    try {
      // retrieve the result
      SOAPMessage ans = response.get();
      SOAPPart part = ans.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPBody body = env.getBody();
```

```
      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String result = responseElement.getValue();

      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Tarefas relacionadas

Desenvolvendo um cliente JAX-RPC para transporte do WebSphere para SOAP usando o Eclipse
Desenvolva um cliente de serviço da web Axis 1.4 para execução usando o transporte do IBM MQ para SOAP.

Desenvolvendo um cliente .NET 1 ou 2 para o transporte do WebSphere para SOAP usando o Microsoft Visual Studio 2012
Desenvolva um cliente do serviço da web .NET 1 ou 2 para executar usando o transporte de IBM MQ para SOAP.

Desenvolvendo um cliente .NET 1 ou 2 para o transporte do WebSphere para SOAP usando o Microsoft Visual Studio 2012

Desenvolva um cliente do serviço da web .NET 1 ou 2 para executar usando o transporte de IBM MQ para SOAP.

Antes de começar

Importante: .NET Framework 2.0 é um pré-requisito para esta tarefa, portanto, certifique-se de que ele tenha sido instalado antes de começar.

É possível iniciar o desenvolvimento de um cliente .NET 1 ou 2 do de várias maneiras diferentes:

1. Use **amqwdeployWMQService** para gerar stubs de cliente por meio de um serviço da web e importe-os para o Microsoft Visual Studio.
2. Use **java2wsdl** para gerar WSDL a partir de uma implementação Java de um serviço da web e, em seguida, use `wsdl.exe`, que é enviado com .NET, para gerar stubs do cliente
3. Gere o WSDL a partir de uma implementação do .NET `.asmx` do serviço usando **amqswsdl** e, em seguida, use `wsdl.exe`.
4. Se você tiver desenvolvido e implementado o serviço para HTTP, use o **Incluir referência da web ...** em Microsoft Visual Studio para configurar o cliente para acessar o serviço HTTP.. Altere a referência da URL para o serviço implementado para o transporte do IBM MQ para SOAP.

A tarefa usa o serviço desenvolvido em"Desenvolvendo um serviço do .NET 1 ou 2 para o transporte de IBM MQ para SOAP usando o Microsoft Visual Studio 2012" na página 1321.

Sobre esta tarefa

Siga estas etapas para criar um .NET 1 ou 2 Client para HTTP e transporte do IBM MQ para SOAP.

Procedimento

1. Crie o aplicativo de console do cliente e modifique-o para chamar o serviço da web StockQuote HTTP.
   a) Clique com o botão direito em **Solução 'StockQuoteDotNet'** no **Solution Explorer** > Incluir ...> Novo Projeto. Selecione o Tipo de Projeto **C#**, **.NET Framework 2.0** e **Application Console**. Dê ao projeto o nome `StockQuoteClientDotNet` > **OK**
   b) Clique com o botão direito em **Solução 'StockQuoteDotNet'** no **Solution Explorer** > Incluir ...> Novo Projeto. Selecione o Tipo de Projeto **C#**, **.NET Framework 2.0** e **Application Console**. Dê ao projeto o nome `StockQuoteClientDotNet` > **OK**
   c) Clique com o botão direito em **StockQuoteClientDotNet** > **Configurar como projeto de inicialização**.
   d) Clique com o botão direito em **StockQuoteClientDotNet** > **Incluir Referência da Web ...** > Procurar Serviços da Web nesta solução> Selecionar **StockQuote** > **Incluir Referência**. Observe que você incluiu uma referência da web para host local e um novo arquivo de configuração `app.config`.
   e) No Solution Explorer, mude o nome do aplicativo de console de `Program.cs` para `StockQuoteClientDotNet.cs` > Clique em **OK** para mudar todos os usos de `Program.cs` para `StockQuoteClientDotNet.cs`.
   f) Substitua o conteúdo de `StockQuoteClientDotNet.cs` pelo código em Figura 186 na página 1352.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Figura 186. Programa HTTP StockQuoteClientDotNet

   g) Ative StockQuoteClientDotNet para testar no serviço `StockQuote.asmx`:
      i) Pressione **F5**, clique na seta verde na barra de ação ou **Depurar** > **Iniciar depuração (F5)**.

      Se o projeto StockQuoteDOtNet estiver na mesma solução, ele será iniciado automaticamente. Caso contrário, será necessário iniciar o serviço primeiro.

      A janela de comandos com os resultados é aberta atrás da área de trabalho. A instrução `Console.ReadLine();` impede que feche até **Enter** ser pressionado.

   Sugestão: Certifique-se de que `StockQuote.asmx` seja a página Iniciar no projeto `StockQuoteDotNet`.
2. Modifique StockQuoteClientDotNet para chamar o serviço `StockQuote.asmx` usando o transporte do IBM MQ para SOAP.
   a) Inclua as linhas mostradas em negrito no cliente.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        IBM.WMQSOAP.Register.Extension();
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
        stockobj.Url = "jms:/queue?"
          + "initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
          + "&targetService=StockQuote.asmx";
        Console.WriteLine("jms reply is: "
          + stockobj.getNonInlineQuote("jms request"));
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Figura 187. Programa StockQuoteClientDotNet modificado

Como alternativa, modifique a URL padrão. Abra **StockQuoteClientDotNet** > **Propriedades** > **Settings.settings** e mude o valor da propriedade `StockQuoteClientDotNet_localhost_StockQuote` para a URL do transporte do IBM MQ para SOAP.

b) Inclua uma referência em `amqsoap.dll`

i) No projeto **StockQuoteClientDotNet** no **Explorador de Soluções**, clique com o botão direito em **Referências** > **Incluir Referência ...** > Clique na guia **Navegar** > navegue até *MQ_INSTALLATION_PATH*\bin > Selecionar **amqsoap.dll** > **OK**. *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado.

3. Teste o cliente com o serviço do `StockQuote.asmx` usando o transporte do IBM MQ para SOAP.

a) Abra uma janela de comando no diretório do projeto StockQuoteDotNet: `.\StockQuoteDotNet\StockQuoteDotNet` > Verifique se o `.bin\StockQuoteDotNet.dll` existe. Caso contrário, reconstrua a solução.

b) Digite o comando **amqwRegisterdotNet**.

Você precisa apenas executar **amqwRegisterdotNet** uma vez por instalação.

c) Se você tiver executado o **amqwdeployWMQServer** com o `genAsmxWMQBits`, execute o Listener SOAP do .NET:
`generated\server\startWMQNListener`

d) Como alternativa, execute o listener diretamente:

```
amqwSOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. No Visual Studio 2012, pressione **F5** para executar StockQuoteClientDotNet.

Clientes de serviço da web .NET Framework 1 e .NET Framework 2

Os clientes .NET de amostra fornecidos com o transporte do IBM MQ para SOAP usam stubs gerados para chamar os serviços Axis e .NET de amostra.

Para cliente .NET Framework 1 e .NET Framework 2, o IBM MQ fornece acesso aos serviços da web usando os clientes .NET. O comando **amqwdeployWMQService** possui uma opção, genProxiestoDotNet, que gera stubs de cliente do .NET Framework 1 ou .NET Framework 2 para um serviço da web Também é possível usar stubs do cliente gerados pela ferramenta .NET **wsdl** ou pelo Microsoft Visual Studio 2012.

Os clientes de serviço da web .NET Framework 1 e .NET de amostra são instalados no *MQ_INSTALLATION_PATH* \tools\soap\samples\dotnet. *MQ_INSTALLATION_PATH* é o diretório no qual IBM MQ está instalado.

SQVB2Axis.vb
SQVB2Axis.vb,Figura 188 na página 1354, é o cliente Visual Basic para chamar o serviço **StockQuoteAxisService**.

SQVB2DotNet.vb
QVB2DotNet.vb,Figura 189 na página 1354, é o cliente Visual Basic para chamar o serviço **StockQuoteDotNet**.

SQCS2Axis.cs
SQCS2Axis.cs,Figura 190 na página 1355, é o cliente C# para chamar o serviço **StockQuoteAxisService**. É possível substituir a URL do serviço fornecendo uma URL na linha de comandos.

SQCS2DotNet.cs
SQCS2DotNet.cs,Figura 191 na página 1355, é o cliente C# para chamar o serviço **StockQuoteDotNet**. É possível substituir a URL do serviço fornecendo uma URL na linha de comandos.

```
Module SQVB2Axis
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj As New StockQuoteAxisService()
      Dim res As Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
   End Function
End Module
```

Figura 188. SQVB2Axis

```
Module SQVB2DotNet
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj as new StockQuoteDotNet()
      Dim res as Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
   End Function
End Module
```

Figura 189. SQVB2DotNet

```
using System;
class SQCS2Axis {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteAxisService stockobj = new StockQuoteAxisService();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("SQCS2Axis RPC reply is: " + res);
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Figura 190. SQCS2Axis

```
using System;
class SQCS2DotNet {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteDotNet stockobj = new StockQuoteDotNet();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("RPC reply is: " + res);
         if (args.GetLength(0) == 0) {
            res = stockobj.getQuoteDOC("XXX");
            Console.WriteLine("DOC reply is: " + res);
         }
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Figura 191. SQCS2DotNet

Tarefas relacionadas

Desenvolvendo um cliente JAX-RPC para transporte do WebSphere para SOAP usando o Eclipse
Desenvolva um cliente de serviço da web Axis 1.4 para execução usando o transporte do IBM MQ para SOAP.

Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse
Desenvolva um cliente de serviço da Web do Axis2 para executar usando o transporte de IBM MQ para SOAP. Os clientes Axis2 de amostra fornecidos com o transporte de IBM MQ para SOAP são listados e o comando **wsimport** usado para gerar proxies.

Implementando serviços da web usando o IBM MQ Transport for SOAP

Implemente um serviço da web a um dos vários ambientes de servidor diferentes e se conecte a ele usando o IBM MQ Transport for SOAP.

Antes de começar

Desenvolva um serviço da web e teste-o usando o SOAP sobre HTTP no ambiente de destino.

Sobre esta tarefa

É possível implementar um serviço da web para executar com o IBM MQ Transport for SOAP em vários diferentes ambientes de tempo de execução SOAP. É possível implementar um serviço para o Axis 1.4 usando apenas o software instalado com o IBM MQ. Para os ambientes de tempo de execução diferentes, deve-se instalar o software adicional.

Você não está restrito à execução do IBM MQ Transport for SOAP para os servidores aos quais existem instruções de implementação. Use as instruções para implementar um serviço em um dos ambientes listados.

Nota: Alguns ambientes integrados oferecem SOAP sobre JMS usando a ligação SOAP do JMS recomendado pelo W3C, assim como a ligação do IBM MQ Transport for SOAP. Liberações de IBM MQ, até e incluindo a 7.0.1.2, suportam somente o transporte do IBM MQ para a ligação SOAP. Na 7.0.1.3 em diante, é possível implementar clientes do Axis2 usando um URI que esteja de acordo com a recomendação do candidato do W3C para SOAP sobre JMS. Consulte o tutorial, Desenvolva um aplicativo de serviços da Web SOAP/JMS JAX-WS com o WebSphere Application Server V7 e o Rational Application Developer 7.5.

Implementando um Serviço no Axis 1.4 para usar para transporte do WebSphere para SOAP usando `amqwdeployWMQService`

Implemente um serviço do Axis 1.4 no transporte do IBM MQ para SOAP criando um diretório de implementação, executando o comando **amqwdeployWMQService** e iniciando o listener do Axis 1.4.

Antes de começar

1. Siga as instruções para instalar o IBM MQ Transport for SOAP
2. Verifique a instalação e seu ambiente usando o comando **runivt**.
3. Para reimplementar um serviço:
   a. Exclua o subdiretório `./generated` e todos os seus subdiretórios.
   b. Remova solicitações da fila de destino e as exclua.
   c. Continue com as instruções da etapa “2” na página 1356.

Sobre esta tarefa

Essas instruções são para implementar um serviço do Axis 1.4 pela primeira vez. Para reiniciar um serviço do Axis 1.4, execute novamente o listener de SOAP do Axis 1.4: etapa “11” na página 1357.

Use as instruções a seguir para implementar um novo serviço do Axis 1.4 para transporte do IBM MQ para SOAP:

Procedimento

1. Crie um diretório *deployDir* para manter os arquivos de implementação.
   O utilitário de implementação requer que cada serviço seja implementado a partir de um diretório separado.
2. Abra uma janela de comando no Windows ou um shell de comando usando X Window System em sistemas UNIX and Linux em *deployDir* para executar **amqwdeployWMQService**.
3. Execute **amqwsetcp** para configurar o caminho de classe.
   JRE e JDK devem estar no caminho de classe, na versão 5.0 ou posterior, e no mesmo nível de versão.
4. Copie a origem da classe, *className*`.java`, no *deployDir*
5. Copie todos os arquivos de origem Java no mesmo pacote como *className* em *deployDir*/*packageName*, em que *packageName* é uma árvore de diretórios correspondente ao nome do pacote.
6. Executar o **javac** *packageName*`.` *className*`.`

Pode ser necessário incluir um caminho para o diretório atual " . "ou para o diretório *packageName* para **javac** localizar as outras classes.

7. Crie o WSDL do Axis para o serviço:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdl
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. Crie os recursos do IBM MQ para o serviço:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

Sugestão:

Se desejar configurar um novo gerenciador de filas e os recursos de que necessita, para realizar desenvolvimento e teste, execute **setupWMQSOAP**.

Se você desejar configurar o novo gerenciador de filas como o padrão, faça uma cópia do **setupWMQSOAP** por meio do *WMQ installation directory*\tools\soap\samples e inclua o parâmetro -q na linha

```
call :try -q crtmqm %QMGR%
```

9. Crie o listener do Axis e implemente o serviço:

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. Se for necessário gerar o WSDL para o serviço, gere stubs do cliente ou proxies de cliente, execute **amqwdeployWMQService** com um dos parâmetros a seguir:

    - genAsmxWsdl
    - genAxisWsdl
    - genProxiesToDotNet
    - genProxiestoAxis

    Nota: Deve-se gerar o WSDL antes de gerar os proxies. A opção AllAxis falhará se o CLASSPATH não estiver configurado para localizar todas as classes que são importadas para compilar *className*.java. Se houver vários arquivos Java no pacote contendo *className*.java, você deverá compilá-los primeiro usando **javac**. **amqwdeployWMQService** -f *packageName*.*className*.java -c CompileJava compila apenas *className*.java.

11. Inicie o listener do Axis gerado.

```
.\generated\server\startWMQJListener.cmd
```

Tarefas relacionadas

Implementando um serviço ao serviço do .NET Framework 1 ou 2 para usar o IBM MQ Transport for SOAP
Implemente um serviço do .NET Framework 1 ou 2 para o IBM MQ Transport for SOAP. Crie um diretório de implementação, execute o comando **amqwdeployWMQService** e iniciar o listener do .NET.

Implementando um serviço para o CICS Transaction Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado ao suporte de serviços da web do CICS Transaction Server 4.1.

Implementando um serviço ao WebSphere Application Server para usar o WebSphere Transport for SOAP

O IBM MQ Transport for SOAP é integrado no barramento de integração de serviços no WebSphere Application Server.

Implementando um serviço ao terminal em serviço do WebSphere ESB e Process Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP não é diretamente suportado pelo WebSphere ESB e Process Server. Deve-se configurar um Export customizado.

Implementando um serviço ao serviço do .NET Framework 1 ou 2 para usar o IBM MQ Transport for SOAP

Implemente um serviço do .NET Framework 1 ou 2 para o IBM MQ Transport for SOAP. Crie um diretório de implementação, execute o comando **amqwdeployWMQService** e iniciar o listener do .NET.

Antes de começar

1. Siga as instruções para instalar o IBM MQ Transport for SOAP
2. Verifique a instalação e seu ambiente usando o comando **runivt**.
3. O caminho para os arquivos de estrutura .NET `wsdl.exe` e `csc.exe` deve ser configurado. As cópias do `wsdl.exe` e `csc.exe` identificadas pela variável PATH devem estar no mesmo nível da estrutura do .NET. Se você tiver várias estruturas do .NET instaladas ou estiver usando o Visual Studio, verifique a variável PATH cuidadosamente.
4. Para reimplementar um serviço:
   a. Exclua o subdiretório `./generated` e todos os seus subdiretórios
   b. Remova solicitações da fila de destino e as exclua.
   c. Continue com as instruções da etapa “2” na página 1358.

Sobre esta tarefa

Estas instruções são para implementar um serviço do .NET pela primeira vez. Para reiniciar um serviço do .NET, execute novamente o listener do .NET SOAP, etapa “9” na página 1359.

Use as instruções a seguir para implementar um novo serviço do .NET Framework 1 ou .NET Framework 2 para o IBM MQ Transport for SOAP:

Procedimento

1. Crie um diretório *deployDir* para manter os arquivos de implementação.
   O utilitário de implementação requer que cada serviço seja implementado a partir de um diretório separado.
2. Abra uma janela de comando no *deployDir* para executar o **amqwdeployWMQService**.

```
C:\IBM\ID\QuoteClient>
```

3. Execute **amqwsetcp** para configurar o caminho de classe.
   Um caminho de classe é necessário apenas para clientes do Axis.
4. Copie o serviço do .NET, *className*`.asmx`, em *deployDir*
5. Construa a implementação de serviço em uma biblioteca (`.dll`).
   A implementação do serviço sequencial está em *className*`.asmx`. A implementação do serviço de code-behind pode ser *className*`.asmx.cs`.
   Figura 192 na página 1359 mostra um exemplo de um comando para construir um serviço do .NET Framework V2 como uma biblioteca.

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

Figura 192. Construa o comando para o serviço do .NET Framework V2

6. Copie *className*.dll no *deployDir**bin*.
7. Configure os recursos do IBM MQ e crie o listener necessário para o serviço:

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```

8. Se for necessário gerar o WSDL para o serviço, gere stubs do cliente ou proxies de cliente, execute **amqwdeployWMQService** com um dos parâmetros a seguir:
   - genAsmxWsdl
   - genAxisWsdl
   - genProxiesToDotNet
   - genProxiestoAxis

   Nota: Deve-se gerar o WSDL antes de gerar os proxies.
9. Inicie o listener gerado do .NET.

```
.\generated\server\startWMQNListener.cmd
```

Tarefas relacionadas

Implementando um Serviço no Axis 1.4 para usar para transporte do WebSphere para SOAP usando amqwdeployWMQService
Implemente um serviço do Axis 1.4 no transporte do IBM MQ para SOAP criando um diretório de implementação, executando o comando **amqwdeployWMQService** e iniciando o listener do Axis 1.4.

Implementando um serviço para o CICS Transaction Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado ao suporte de serviços da web do CICS Transaction Server 4.1.

Implementando um serviço ao WebSphere Application Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado no barramento de integração de serviços no WebSphere Application Server.

Implementando um serviço ao terminal em serviço do WebSphere ESB e Process Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP não é diretamente suportado pelo WebSphere ESB e Process Server. Deve-se configurar um Export customizado.

Implementando um serviço para o CICS Transaction Server para usar o WebSphere Transport for SOAP

O IBM MQ Transport for SOAP é integrado ao suporte de serviços da web do CICS Transaction Server 4.1.

Antes de começar

Use as mesmas ferramentas para desenvolver para um cliente ou serviço para o IBM MQ, como você faria para desenvolver para HTTP. CICS tem ferramentas correspondentes ao **Java2wsdl** e **wsdl2Java**:

- **DFHWS2LS** toma uma descrição de serviço da web como um ponto de início. Ele usa as descrições das mensagens e os tipos de dados usados nessas mensagens para construir estruturas de dados de linguagem de alto nível. É possível usar nas estruturas em programas de aplicativos gravados em linguagens diferentes.
- **DFHLS2WS** tem uma estrutura de dados de linguagem de alto nível como um ponto de início. Ele usa a estrutura para construir uma descrição de serviços da web que contém descrições de mensagens. Ele também cria os esquemas para as mensagens da estrutura de dados de linguagem.

Siga as instruções Criando um serviço da web na documentação do produto CICS para criar um serviço da web.

Sobre esta tarefa

Siga as instruções, Configurando o CICS para usar o transporte do IBM MQ na documentação do produto CICS. Ao usar as instruções, será possível implementar o serviço da web no IBM MQ Transport for SOAP.

Tarefas relacionadas

Implementando um Serviço no Axis 1.4 para usar para transporte do WebSphere para SOAP usando amqwdeployWMQService
Implemente um serviço do Axis 1.4 no transporte do IBM MQ para SOAP criando um diretório de implementação, executando o comando **amqwdeployWMQService** e iniciando o listener do Axis 1.4.

Implementando um serviço ao serviço do .NET Framework 1 ou 2 para usar o IBM MQ Transport for SOAP
Implemente um serviço do .NET Framework 1 ou 2 para o IBM MQ Transport for SOAP. Crie um diretório de implementação, execute o comando **amqwdeployWMQService** e iniciar o listener do .NET.

Implementando um serviço ao WebSphere Application Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado no barramento de integração de serviços no WebSphere Application Server.

Implementando um serviço ao terminal em serviço do WebSphere ESB e Process Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP não é diretamente suportado pelo WebSphere ESB e Process Server. Deve-se configurar um Export customizado.

Implementando um serviço ao WebSphere Application Server para usar o WebSphere Transport for SOAP

O IBM MQ Transport for SOAP é integrado no barramento de integração de serviços no WebSphere Application Server.

Antes de começar

Use o Rational Application Developer, WebSphere Integration Developer ou um kit de serviços da web para desenvolver o serviço da web.

Sobre esta tarefa

Use as instruções a seguir para implementar um serviço usando o IBM MQ Transport for SOAP como um transporte SOAP no WebSphere Application Server.

Procedimento

1. Configure o IBM MQ como o provedor do sistema de mensagens do JMS para o barramento de integração de serviços no WebSphere Application Server.
2. Configure os recursos do IBM MQ requeridos pelo serviço.

3. Siga as instruções de Configurando recursos do JMS para o listener terminal do SOAP síncrono sobre JMS na documentação do produto WebSphere Application Server Network Deployment.

   Há instruções correspondentes para outras plataformas do WebSphere Application Server.
4. Modifique o URI de serviço para conformidade com o URI do transporte para SOAP do IBM MQ.
5. Implemente o serviço em WebSphere Application Server.

Como proceder a seguir

Implemente o serviço com HTTP como um transporte para que os clientes possam consultar o serviço e receber o WSDL na resposta.

Tarefas relacionadas

Implementando um Serviço no Axis 1.4 para usar para transporte do WebSphere para SOAP usando amqwdeployWMQService
Implemente um serviço do Axis 1.4 no transporte do IBM MQ para SOAP criando um diretório de implementação, executando o comando **amqwdeployWMQService** e iniciando o listener do Axis 1.4.

Implementando um serviço ao serviço do .NET Framework 1 ou 2 para usar o IBM MQ Transport for SOAP
Implemente um serviço do .NET Framework 1 ou 2 para o IBM MQ Transport for SOAP. Crie um diretório de implementação, execute o comando **amqwdeployWMQService** e iniciar o listener do .NET.

Implementando um serviço para o CICS Transaction Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado ao suporte de serviços da web do CICS Transaction Server 4.1.

Implementando um serviço ao terminal em serviço do WebSphere ESB e Process Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP não é diretamente suportado pelo WebSphere ESB e Process Server. Deve-se configurar um Export customizado.

Implementando um serviço ao terminal em serviço do WebSphere ESB e Process Server para usar o WebSphere Transport for SOAP

O IBM MQ Transport for SOAP não é diretamente suportado pelo WebSphere ESB e Process Server. Deve-se configurar um Export customizado.

Sobre esta tarefa

O WebSphere Integration Developer fornece uma transformação de dados SOAP que é possível ligar ao IBM MQ JMS Export para criar um Export IBM MQ JMS SOAP customizado.

Siga as instruções para criar um Export customizado para receber solicitações SOAP sobre o IBM MQ Transport for SOAP.

Procedimento

1. Leia "Visão geral de importações e exportações" e "Como conectar-se ao IBM MQ" na documentação do produto WebSphere Process Server for Multiplatforms.

   Consulte Visão geral de importações e exportações e Como conectar-se ao IBM MQ.
2. Siga a tarefa "Gerando uma ligação de exportação do MQ JMS" na documentação do produto WebSphere Integration Developer.

   Consulte Gerando uma ligação de exportação do MQ JMS. Use a ligação de dados SOAP predefinida descrita em Transformações de formato de dados JMS predefinidos para formatar a mensagem SOAP.

Tarefas relacionadas

Implementando um Serviço no Axis 1.4 para usar para transporte do WebSphere para SOAP usando amqwdeployWMQService
Implemente um serviço do Axis 1.4 no transporte do IBM MQ para SOAP criando um diretório de implementação, executando o comando **amqwdeployWMQService** e iniciando o listener do Axis 1.4.

Implementando um serviço ao serviço do .NET Framework 1 ou 2 para usar o IBM MQ Transport for SOAP

Implemente um serviço do .NET Framework 1 ou 2 para o IBM MQ Transport for SOAP. Crie um diretório de implementação, execute o comando **amqwdeployWMQService** e iniciar o listener do .NET.

Implementando um serviço para o CICS Transaction Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado ao suporte de serviços da web do CICS Transaction Server 4.1.

Implementando um serviço ao WebSphere Application Server para usar o WebSphere Transport for SOAP
O IBM MQ Transport for SOAP é integrado no barramento de integração de serviços no WebSphere Application Server.

Implementando os clientes de serviço da web para usar o IBM MQ Transport for SOAP

Implemente um cliente de serviço da web a um dos vários ambientes do cliente diferentes e se conecte a um serviço usando o IBM MQ Transport for SOAP.

Antes de começar

Desenvolva o serviço da web e implemente-o para usar o IBM MQ Transport for SOAP.

Sobre esta tarefa

É possível implementar um cliente de serviço da web para executar com o IBM MQ Transport for SOAP em vários ambientes de cliente diferentes. É possível implementar um cliente Java para o Axis 1.4 usando apenas o software instalado com o IBM MQ. Para os outros ambientes de cliente, deve-se instalar o software adicional.

Você não está restrito à execução do WebSphere Transport for SOAP nos ambientes do cliente para o qual existem instruções de implementação. Use as instruções para implementar um cliente a um dos ambientes suportados.

Nota: Alguns ambientes integrados oferecem SOAP sobre JMS usando a ligação SOAP do JMS recomendado pelo W3C, assim como a ligação do IBM MQ Transport for SOAP. Liberações de IBM MQ, até e incluindo a 7.0.1.2, suportam somente o transporte do IBM MQ para a ligação SOAP. Na 7.0.1.3 em diante, é possível implementar clientes do Axis2 usando um URI que esteja de acordo com a recomendação do candidato do W3C para SOAP sobre JMS. Consulte o tutorial, Desenvolva um aplicativo de serviços da Web SOAP/JMS JAX-WS com o WebSphere Application Server V7 e o Rational Application Developer 7.5.

Implementando um cliente de serviço da web para o Axis 1.4 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Antes de começar

Sugestão: Implemente o serviço para HTTP, desenvolva e teste o cliente para HTTP e, em seguida, modifique o cliente para o transporte do IBM MQ para SOAP:

1. Inclua a chamada Register.extension() no cliente.
2. Mude o endereço de serviço da web estático na classe do localizador de proxy de cliente para usar o URI para o transporte do IBM MQ para SOAP.

Sobre esta tarefa

Implementando um cliente Axis 1.4 para usar o transporte do IBM MQ para SOAP requer uma etapa de implementação adicional comparada com um cliente HTTP. Deve-se criar um descritor de implementação do cliente, `client-config.wsdd`, para mapear o transporte `jms:` para o emissor de classe `com.ibm.mq.soap.transport.jms.WMQSender`.

Se você usar o comando **amqwdeployWMQService** para gerar proxies de cliente, será possível implementar o cliente usando os diretórios que o comando gera.

Procedimento

1. Crie um diretório *deployDir* para manter os arquivos de implementação do cliente.
2. Abra uma janela de comandos em sistemas Windows ou um shell de comando usando X Window System em sistemas UNIX and Linux, em *deployDir*.
3. Execute o comando **amqwsetcp.cmd** para configurar o CLASSPATH
4. Execute o comando **amqwclientconfig.cmd** para criar um descritor de implementação do cliente Axis 1.4, `client-config.wsdd` em *deployDir*.
5. Certifique-se de que as classes no pacote do cliente, as classes de proxy cliente e as bibliotecas que o cliente usa estejam no CLASSPATH.

   amqwdeployWMQService coloca os proxies de cliente .NET em `./generated/server/soap/client/remote/dotnetService` e os proxies do Axis 1.4 em `./generated/server/soap/client/remote/`*`client package`*.

exemplo

O exemplo mostra a configuração e a saída, Figura 195 na página 1364, a partir de um cliente Java Axis 1.4. O cliente, Figura 194 na página 1363, chama um serviço da web que ecoa seu parâmetro de entrada. A definição de serviço, Figura 193 na página 1363, mostra o URI obtido do serviço WSDL.

```
<wsdl:service name="QuoteSOAPImplService">
    wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
                      name="org.example.www.QuoteSOAPImpl_Wmq">
       <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
           &connectionFactory=(connectQueueManager(QM1)binding(server))
           &initialContextFactory=com.ibm.mq.jms.Nojndi
           &targetService=org.example.www.QuoteSOAPImpl.java
           &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
</wsdl:service>
```

Figura 193. Definição de serviço

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                  + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

Figura 194. Cliente Axis 1.4 Java

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

Figura 195. Configuração do cliente e de saída

Como proceder a seguir

1. Se você estiver implementando o cliente como um cliente IBM MQ, configure o canal de conexão do cliente e do servidor.
2. Se você estiver implementando o cliente em um gerenciador de filas diferente no serviço, deve-se disponibilizar a fila de destino para o cliente. Configure a fila de destino no gerenciador de filas de serviço como uma fila de cluster ou no gerenciador de filas do cliente como uma definição de fila remota.

Tarefas relacionadas

Implementando um cliente de serviço da web para Axis2 para usar o transporte do IBM MQ para SOAP
Prepare um diretório de implementação e o arquivo de configuração do Axis2 para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando em um cliente do Axis2 usando o W3C SOAP sobre JMS
Um serviço da web ligado à recomendação do candidato W3C para SOAP sobre JMS deve ser executado no contêiner EJB de um servidor de aplicativos Java EE. Esta tarefa é a etapa 4 da conexão de um cliente de serviço da web do Axis2 e um serviço da web implementado no WebSphere Application Server usando o protocolo W3C SOAP sobre JMS. Modifique a URL no cliente do Axis2 desenvolvido para o IBM MQ Transport for SOAP para usar a recomendação do candidato do W3C para SOAP sobre JMS.

Implementando um cliente de serviço da web no .NET Framework 1 e 2 para usar o transporte do IBM MQ para SOAP
Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça o proxy de cliente e a classe do cliente. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando um cliente de serviço da web para Axis2 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e o arquivo de configuração do Axis2 para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Antes de começar

Sugestão: Implemente o serviço para HTTP. Desenvolva e teste o cliente para HTTP e, em seguida, modifique a URL para fazer referência ao serviço usando o transporte do IBM MQ para SOAP.

A tarefa mostra como implementar um cliente não gerenciado Axis2 para Java Standard Edition. Pode ser que deseje implementar um cliente do Axis2 em um contêiner da web. No “Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse” na página 1337, você desenvolveu um cliente em um contêiner da web e implementou-o no WebSphere Application Server Community Edition. Como parte da configuração do servidor, você ativou o aspecto do Axis2 e incluiu o aspecto na configuração do contêiner da web. Para configurar contêineres da web em

outros servidores de aplicativos, consulte a documentação do Axis2 , https://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_containerou a documentação fornecida com o servidor da web.

Nota: O Axis2 usa o termo, contêiner do Servlet. Um contêiner do Servlet é o mesmo que um contêiner da web.

Sobre esta tarefa

Implementando um cliente Axis2 para usar o transporte do IBM MQ para SOAP é como implementar um cliente Axis2 para usar HTTP. Etapas adicionais são necessárias para fornecer um caminho de classe para arquivos JAR do IBM MQ e para modificar o arquivo de configuração do Axis2. O arquivo de configuração do Axis2 requer uma entrada adicional para JMS. A entrada se refere ao transporte do IBM MQ para o arquivo JAR SOAP que implementa a JMS `transportSender`.

O Axis2 fornece um script, `axis2.bat` ou `axis2.sh`, que simplifica a implementação do cliente; consulte os exemplos em Figura 199 na página 1367 e Figura 200 na página 1367.

Nota:

1. `axis2.bat` tem um erro que deve ser corrigido. A sequência `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` deve ser mudada para `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`.
2. No `axis2.bat` e `axis2.sh`, `-Djava.ext.dirs` é usado como uma maneira rápida para fazer referência a todos os arquivos JAR Axis2, em vez de incluí-los separadamente no caminho de classe. Infelizmente esta abordagem é imperfeita e funciona apenas com alguns JREs. Ela não funciona com os IBM JREs.

   O parâmetro JVM, `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`, torna os arquivos JAR Axis disponíveis para a JVM. A JVM tenta instanciar alguns dos arquivos JAR Axis e leva a um erro, os detalhes dos quais dependem da JVM. Geralmente, você poderá ver uma das linhas a seguir no rastreio de pilha:

   ```
   org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)
   ```

   ou `org.apache.axis2.deployment.DeploymentException: java.security.NoSuchAlgorithmException: MD5 MessageDigest not available`

   A maneira correta para executar um cliente não gerenciado Axis2 é incluir os arquivos JAR do Axis2 no caminho de classe. O caminho de classe está disponível apenas para o aplicativo cliente e não para a JVM.

O procedimento descreve as etapas gerais para executar um cliente Axis2 não gerenciado sem usar o script axis2. Os exemplos em Figura 197 na página 1366 e Figura 198 na página 1367 são scripts para o Windows e Linux.

Procedimento

1. Faça download do Axis2 1.4.1 de https://ws.apache.org/axis2/download/1_4_1/download.cgi e descompacte em uma pasta, `Axis2-1.4.1`.
2. Atualize `axis2.xml` em `Axis2-1.4.1\conf`.
   a) Atualize `axis2.xml` em `Axis2-1.4.1\conf`. Inclua transporte do IBM MQ para SOAP como um `transportSender`:

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
   ```

   b) Se necessário, altere o tamanho do conjunto de conexões do padrão de 10.

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
   ```

```
<parameter name="ResourcePoolCapacity">20</parameter>
</transportSender>
```

`ResourcePoolCapacity` define quantas entradas de terminal de serviço são mantidas no cache. O valor deve ser pelo menos 1. Se o número de entradas do terminal em serviço exceder o tamanho do cache, as entradas serão excluídas para abrir espaço para novas entradas. O tamanho de uma entrada do terminal varia. Configure um número que seja grande o suficiente para evitar a sobrecarga de cache.

Consulte a etapa 3 em "Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse" na página 1337.

3. Crie um diretório *deployDir*. Nesse diretório, copie a estrutura de pasta contendo o cliente e os proxies de cliente. *deployDir* é equivalente à pasta *project*`\bin` em um projeto Eclipse Java .
4. Abra uma janela de comandos no Windows ou um shell de comando usando X Window System em sistemas UNIX and Linux, em *deployDir*.
5. Atualize o caminho de classe para incluir o diretório atual, arquivos JAR do Axis2, `com.ibm.mqjms.jar` e `com.ibm.mq.axis2.jar`.

   `com.ibm.mqjms.jar` faz referência a todos os outros arquivos JAR do IBM MQ que são necessários.
6. Use o comando **Java** para iniciar o programa cliente.

Examples

Quatro exemplos da execução de um cliente do Axis2 são listados em Figura 198 na página 1367 para Figura 200 na página 1367. Figura 196 na página 1366 mostra a saída da execução do cliente assíncrono listado em Figura 179 na página 1342.

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

Figura 196. Saída da execução SQA2AsyncClient

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
".;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

Figura 197. runpojo.bat: Windows, usando um caminho de classe

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
  AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME":"$JAVA_HOME/lib/tools.jar":"$AXIS2_CLASSPATH":"$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

Figura 198. runpojo.sh: Linux, usando um caminho de classe.

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

Figura 199. runaxis2.bat: Windows, usando axis2.bat

Nota

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

Figura 200. runaxis2.sh: Linux, usando axis2.sh

Nota

Tarefas relacionadas

Implementando um cliente de serviço da web para o Axis 1.4 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando em um cliente do Axis2 usando o W3C SOAP sobre JMS

Um serviço da web ligado à recomendação do candidato W3C para SOAP sobre JMS deve ser executado no contêiner EJB de um servidor de aplicativos Java EE. Esta tarefa é a etapa 4 da conexão de um cliente de serviço da web do Axis2 e um serviço da web implementado no WebSphere Application Server usando o protocolo W3C SOAP sobre JMS. Modifique a URL no cliente do Axis2 desenvolvido para o IBM MQ Transport for SOAP para usar a recomendação do candidato do W3C para SOAP sobre JMS.

Implementando um cliente de serviço da web no .NET Framework 1 e 2 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça o proxy de cliente e a classe do cliente. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando em um cliente do Axis2 usando o W3C SOAP sobre JMS

Um serviço da web ligado à recomendação do candidato W3C para SOAP sobre JMS deve ser executado no contêiner EJB de um servidor de aplicativos Java EE. Esta tarefa é a etapa 4 da conexão de um cliente de serviço da web do Axis2 e um serviço da web implementado no WebSphere Application Server usando

o protocolo W3C SOAP sobre JMS. Modifique a URL no cliente do Axis2 desenvolvido para o IBM MQ Transport for SOAP para usar a recomendação do candidato do W3C para SOAP sobre JMS.

Antes de começar

Deve-se primeiro concluir a tarefa, "Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse" na página 1337 para chamar **SimpleJavaListener** usando um cliente do Axis2 e o protocolo do transporte para SOAP do IBM MQ.

Também deve-se ter criado o serviço da web e configurado o IBM MQ e WebSphere Application Server.

Consulte Usando o IBM MQ e o WebSphere Application Server juntos.

Também deve-se ter concluído a seguinte tarefa, "Desenvolvendo um serviço da web JAX-WS EJB para W3C SOAP sobre JMS" na página 1324.

Na tarefa, o cliente é executado no Eclipse Galileo. Você pode executar o cliente a partir da linha de comandos modificando o arquivo `Axis2.bat` fornecido com o Axis2.

Sobre esta tarefa

A única mudança que deve-se fazer ao cliente estático `StockQuoteAxis` do Axis2 existente para chamar o serviço StockQuoteAxis hospedado por WebSphere Application Server é mudar a URL transmitida para o cliente. Como o WSDL não foi mudado, será possível usar as mesmas classes de proxy no pacote `soap.server`.

Você tem duas abordagens para definir a URL para transmitir para o cliente. Você pode usar a mesma URL como gerada em `StockQuoteAxis.wsdl`. Deve-se incluir os parâmetros `jndiInitialContextFactory` e o `jndiURL` para acessar o diretório JNDI do WebSphere Application Server. Outra abordagem é mudar a URL e fornecer o acesso direto do cliente para as filas `REQUESTAXIS` e `REPLYAXIS` em `QM1`, sem usar uma consulta JNDI.

Os parâmetros de conexão definidos na URL transmitida para o cliente do Axis2 são usados para se conectar ao gerenciador de filas e as filas do IBM MQ necessárias para enviar e receber mensagens SOAP. Os parâmetros de conexão transmitidos ao cliente Axis2 não são necessariamente usados pelo serviço. É possível usar os recursos de enfileiramento distribuídos de IBM MQ para desacoplar o cliente e o serviço de usar o mesmo gerenciador de filas ou o mesmo servidor de nome.

Procedimento

1. Salve a URL do `StockQuoteAxis.wsdl` gerado e feche o Rational Application Developer para ser salvo na memória.

   Se você não mudou a configuração do servidor, o fechamento do Rational Application Developer irá parar o servidor de aplicativos. Nesse caso, inicie o servidor com o comando:

   ```
   startserver server1
   ```

2. Abra o Eclipse Galileo na área de trabalho com o projeto do cliente do Axis2.
3. Abra `SQA2StaticClient.java`.

   Consulte SQA2StaticClient.java.
4. Chame o serviço usando a variante queue do URI.
   a) Modifique a URL.

      O novo URI é:

      ```
      jms:queue:REQUESTAXIS
                ?replyToName=REPLYAXIS
                &connectionFactory=connectQueueManager(QM1)Bind(Server)
                &targetService=StockQuoteAxis;
      ```

Compare a esta URL do `StockQuoteAxis.wsdl`:

```
jms:jndi:requestaxis
         ?jndiConnectionFactoryName=qm1
         &targetService=StockQuoteAxis
```

Figura 201. URL a partir do `StockQuoteAxis.wsdl`

- REQUESTAXIS está agora em maiúsculas já que ele é um nome de fila e não um nome de JNDI.
- A conexão com QM1 é direta.
- O URI não contém o nome da resposta para o destino. O cliente deve definir a fila a qual ele espera as respostas.

b) Execute `SQA2StaticClient.java` usando o mesmo **Executar como ...** como você fez na tarefa, "Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse" na página 1337.

5. Chame o serviço usando a variante `jndi` do URI, usando o WebSphere Application Server como o servidor de nomenclatura.

a) Use a URL a partir do `StockQuoteAxis.wsdl`, Figura 201 na página 1369, fornecendo os parâmetros ausentes para usar o serviço de nomenclatura no WebSphere Application Server.

Os parâmetros e valores ausentes que deve-se fornecer são:

Tabela 192. Parâmetros adicionais de JNDI

Parâmetro	Valor usado neste exemplo	Descrição
`&jndiURL`	`iiop://localhost:2810` ou `corbaname:iiop:localhost:2810`	URI do provedor de nomenclatura. Para o WebSphere Application Server, o valor é padronizado para 2809. Também é conhecido como o número da porta do conector RMI e a porta de autoinicialização. O valor é listado no `SystemOut.log` `00000000 NameServerImp A  NMSV0018I: Name server available on bootstrap port 2810`
`&jndiInitialContextFactory`	`com.ibm.websphere.naming.WsnInitialContextFactory`	O nome da factory de contexto inicial usado pelo WebSphere Application Server.
`&replyToName`	`replyaxis`	Nome JNDI da fila REPLYAXIS.

```
jms:jndi:requestaxis?
         &jndiURL=iiop//localhost:2810
         &jndiConnectionFactoryName=qm1
         &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
         &targetService=StockQuoteAxis
         &replyToName=replyaxis;
```

b) Inclua os arquivos JAR requeridos pela consulta JNDI.

Nessa configuração, os seguintes arquivos JAR foram incluídos no caminho de construção para executar a tarefa usando a variante `jndi` da URL do JMS:

- `com.ibm.jaxws.thinclient_7.0.0.jar` de *`Rational installation directory`*`\SDP\runtimes\base_v7\runtimes`.
- `com.ibm.ws.runtime.jar` from *`Rational installation directory`*`\SDP\runtimes\base_v7\plugins`

Para um provedor JNDI diferente, requeira arquivos JAR diferentes.

Os outros arquivos JAR no caminho de construção são:

i) Todos os arquivos JAR em *`WebSphere MQ Installation directory`*`\java\lib`.

ii) Todos os arquivos JAR em *`Axis2-1.5.1`*`\lib`.

iii) Java 6.0 JRE.

c) Execute `SQA2StaticClient.java` usando o mesmo **Executar como ...** como você fez na tarefa, "Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse" na página 1337.

Resultados

Em ambos os casos a resposta do serviço é exibida na visualização de console do cliente.

Tarefas relacionadas

Implementando um cliente de serviço da web para o Axis 1.4 para usar o transporte do IBM MQ para SOAP
Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando um cliente de serviço da web para Axis2 para usar o transporte do IBM MQ para SOAP
Prepare um diretório de implementação e o arquivo de configuração do Axis2 para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando um cliente de serviço da web no .NET Framework 1 e 2 para usar o transporte do IBM MQ para SOAP
Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça o proxy de cliente e a classe do cliente. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando um cliente de serviço da web no .NET Framework 1 e 2 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça o proxy de cliente e a classe do cliente. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Antes de começar

Sugestão: Desenvolva e teste o serviço e o cliente usando o Visual Studio. Em seguida, modifique o cliente para transporte do IBM MQ para SOAP.

1. Se você estiver implementando um serviço usando .NET Framework 1 ou 2, construa o serviço como uma biblioteca (`.dll`). Implementar usando o transporte IBM MQ para SOAP.
2. Inclua a chamada Register.Extension() no cliente.
3. Inclua uma referência em amqsoap.dll, que está localizado em *`MQ_Install`*`\bin`.
4. Mude a propriedade `Url` estática no construtor da classe de proxy do cliente no URI `jms:/`, para o transporte do IBM MQ para SOAP.

Sobre esta tarefa

Implementar um cliente de serviço da web para o .NET Framework 1 ou 2 para usar transporte do IBM MQ para SOAP requer uma etapa de implementação adicional. É necessário registrar `amqsoap.dll` com

o .NET Framework. `amqsoap.dll` é automaticamente registrado como parte da instalação do transporte do IBM MQ para SOAP, mas pode ser necessário registrá-lo novamente.

Se você usar o comando **amqwdeployWMQService** para gerar proxies de cliente, será possível implementar o cliente usando os diretórios que o comando gera.

Procedimento

1. Crie um diretório *deployDir* para manter os arquivos de implementação do cliente.
2. Abra uma janela de comando em *deployDir*.
3. Execute **amqwsetcp** para configurar o CLASSPATH se o serviço for ser executado no Axis 1.4.
4. Se necessário, execute **amqwRegisterDotNet** para registrar `amqsoap.dll` com o .NET Framework.

exemplo

O exemplo mostra a configuração e a saída, Figura 204 na página 1371, a partir de um cliente .NET Framework V2. O cliente, Figura 203 na página 1371, chama um serviço da web que ecoa seu parâmetro de entrada. A definição da Url estática, Figura 202 na página 1371, mostra o construtor para o proxy de cliente.

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
               &connectionFactory=(connectQueueManager(QM1)binding(server))
               &initialContextFactory=com.ibm.mq.jms.Nojndi
               &targetService=Quote.asmx
               &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

Figura 202. Construtor de proxy de cliente estático

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
```

Figura 203. Programa cliente

```
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM
```

Figura 204. Configuração e saída

Como proceder a seguir

1. Se você estiver implementando o cliente como um IBM MQ MQI client, configure o canal de conexão do cliente e do servidor.
2. Se você estiver implementando o cliente em um gerenciador de filas diferente no serviço, deve-se disponibilizar a fila de destino para o cliente. Configure a fila de destino no gerenciador de filas de serviço como uma fila de cluster ou no gerenciador de filas do cliente como uma definição de fila remota.

Tarefas relacionadas

Implementando um cliente de serviço da web para o Axis 1.4 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e um descritor de implementação para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando um cliente de serviço da web para Axis2 para usar o transporte do IBM MQ para SOAP

Prepare um diretório de implementação e o arquivo de configuração do Axis2 para o cliente. Forneça os proxies do cliente e a classe do cliente e configure CLASSPATH. Configure as filas e os canais do IBM MQ, inicie o serviço e teste o cliente.

Implementando em um cliente do Axis2 usando o W3C SOAP sobre JMS

Um serviço da web ligado à recomendação do candidato W3C para SOAP sobre JMS deve ser executado no contêiner EJB de um servidor de aplicativos Java EE. Esta tarefa é a etapa 4 da conexão de um cliente de serviço da web do Axis2 e um serviço da web implementado no WebSphere Application Server usando o protocolo W3C SOAP sobre JMS. Modifique a URL no cliente do Axis2 desenvolvido para o IBM MQ Transport for SOAP para usar a recomendação do candidato do W3C para SOAP sobre JMS.

Conecte um cliente do Axis2 para um serviço JAX-WS usando o W3C SOAP sobre JMS e WebSphere Application Server

Ao concluir essa tarefa, você terá que chamou um serviço da web JAX-WS em execução no WebSphere Application Server a partir de um cliente do Axis2. O cliente do Axis2 e o WebSphere Application Server usam a recomendação do candidato do W3C para o protocolo SOAP sobre JMS em execução no IBM MQ. Use Eclipse Galileo e Rational Application Developer para construir o cliente de serviço da web e o serviço da web, respectivamente.

Antes de começar

A tarefa requer a versão 7 do Rational Software Development Environment e WebSphere Application Server. A tarefa foi criada usando o Rational Application Developer empacotado com o Rational Software Architect for WebSphere Software 7.5.5.1e o WebSphere Application Server 7.0 Test Environment 7.0.0.9 Atualização 1. Você também requer IBM MQ 7.0.1.3ou posterior.

A tarefa é construída em duas outras tarefas, o perfil Liberty e “Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse” na página 1337. Para concluir essas tarefas, seu ambiente de desenvolvimento já tem o Eclipse Galileo, o perfil Liberty, o plug-in do Eclipse para Liberty e o Axis2 1.4.1 instalados.

Algumas das etapas são complexas. As etapas assumem alguma familiaridade com o desenvolvimento de aplicativos de serviço da Web para WebSphere Application Server usando o Rational Application Developer. As demandas do processador e da memória da tarefa são grandes. A tarefa foi executada em uma máquina virtual VMWare do Windows 7 SP1 alocada com 1,8 GB de memória.

Instale todo o software antes de iniciar a tarefa. O software leva cerca de um dia para fazer download e um dia para instalar, dependendo da largura de banda. A tarefa leva pelo menos meio dia.

Sobre esta tarefa

O cenário para essa tarefa é que você desenvolveu um serviço da web de cotação de ações, `StockQuoteAxis`, usando uma ferramenta de software livre, Eclipse Galileo. `StockQuoteAxis` é implementado usando SOAP sobre HTTP em execução em um servidor de software livre, o perfil Liberty.

Você deseja ligar os serviços da web implementados a um transporte de sistema de mensagens baseado em padrões, como SOAP sobre JMS ou ao sistema de mensagens confiável de serviços da web, assim como o SOAP sobre HTTP. Você deseja que o cliente e o serviço usem interfaces baseadas em padrões. Por essa razão, embora sua futura equipe de desenvolvimento de projetos tenha implementado uma solução usando o IBM MQ Transport for SOAP, você não entrou em produção.

O cliente do Axis2 removeu o problema que o cliente SOAP para o IBM MQ Transport for SOAP necessitava para uma mudança a partir do cliente HTTP. O problema ainda permaneceu que o serviço conectado pelo IBM MQ Transport for SOAP está hospedado por um listener especial fornecido pelo IBM MQ: `SimpleJavaListener`.

Com o W3C SOAP sobre JMS padrão no status de recomendação do candidato, alguns fornecedores estão oferecendo suporte para W3C SOAP sobre JMS. O suporte permite implementar um serviço da web em um servidor de aplicativos e conectar-se ao mesmo serviço usando uma variedade de protocolos de conectividade. O suporte fornecido pelo WebSphere Application Server v7 remove o problema de ter que hospedará o serviço da web separadamente para usar um transporte SOAP baseado em mensagem. O uso de uma interface com transporte de mensagens baseados em padrões, JMS, significa que é possível desenvolver soluções usando ferramentas de diferentes fornecedores. É possível que as ferramentas de serviços da web no Eclipse incluirão ligações do SOAP sobre JMS no futuro.

A maioria das etapas são executadas usando o Eclipse ou as ferramentas de gerenciamento fornecidas com os produtos WebSphere. As etapas são descritas para um ambiente do Windows. Com pequenas modificações para alguns comandos, é possível executar as etapas em outras plataformas.

As etapas preliminares para criar o serviço da web HTTP e conectar-se a ele usando o Axis2 são listadas. O cliente e WSDL, a partir destas etapas, são usados para criar a solução

Procedimento

1. Conecte-se ao serviço da web StockQuoteAxis usando um cliente do Axis2 e o IBM MQ Transport for SOAP
   a) Use o perfil Liberty
   b) “Desenvolvendo um cliente JAX-WS para o WebSphere Transport for SOAP usando o Eclipse” na página 1337
   c) “Implementando um cliente de serviço da web para Axis2 para usar o transporte do IBM MQ para SOAP” na página 1364
2. Conecte-se ao serviço da web StockQuoteAxis usando um cliente do Axis2 e a recomendação do candidato do W3C para SOAP sobre JMS.
   a) Configure recursos do IBM MQ.
   b) Configure recursos do WebSphere Application Server.
   c) “Desenvolvendo um serviço da web JAX-WS EJB para W3C SOAP sobre JMS” na página 1324
   d) “Implementando em um cliente do Axis2 usando o W3C SOAP sobre JMS” na página 1367

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 um IBM produto, programa ou serviço não se destina a estado ou significa que apenas esse produto IBM, programas ou serviços possam ser utilizados. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM poderá ser utilizado em substituição. 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:

Relações Comerciais e Industriais da IBM
Av. Pasteur, 138-146
Botafogo
Rio, RJ 10504-1785
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:

licença de propriedade intelectual
IBM World Trade Asia Corporation Licensing
IBM Japan, Ltd.
Minato-ku
Tóquio 103-8510, Japão

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. Periodicamente, são feitas nas informações aqui contidas; essas 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.

Referências nestas informações a websites não IBM são fornecidas apenas por conveniência e não representam de forma alguma 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.

Licenciados deste programa que desejam obter informações sobre este assunto com objetivo de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) a utilização mútua das informações trocadas, devem 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 interfaces de programação desejadas que permitem que o cliente grave programas para obter os serviços do WebSphere 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 comerciais 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 (http://www.eclipse.org/).

Java e todas as marcas comerciais e logotipos baseados em Java são marcas comerciais ou marcas registradas da Oracle e/ou de suas afiliadas.

Part Number:

(1P) P/N: