WebSphere MQ Everyplace V2.0.2

Índice

Capítulo 1. Como Configurar Objetos MQe

Visão geral da configuração de filas, gerenciadores de filas e redes MQe

Esta parte do centro de informações fornece as informações básicas necessárias para configurar gerenciadores de filas e redes MQe. Também foi projetada para ajudá-lo a customizar uma configuração que corresponda a seus requisitos de negócios específicos. Descreve como componentes MQe individuais podem ser criados e administrados e como os componentes podem ser utilizados juntos em várias topologias.

Introdução

Este manual fornece as informações básicas necessárias para configurar gerenciadores de filas e redes MQe. Também foi projetado para permitir que um usuário customize uma configuração que corresponda a seus requisitos de negócios específicos. Descreve como componentes MQe individuais podem ser criados e administrados e como os componentes podem ser utilizados juntos em várias topologias.

O conteúdo inclui informações sobre:

- Criar e iniciar gerenciadores de filas
- Definir a conectividade entre gerenciadores de filas
- Estabelecer as rotas seguidas por mensagens através de uma rede MQe
- Exercer controle sobre os protocolos utilizados
- Determinar onde as mensagens são produzidas, se apropriado
- Configurar a segurança no nível de fila
- Avaliar as vantagens e desvantagens das opções de configuração do MQe disponíveis

Esta introdução fornece um mapa de várias rotas pelo resto do guia, dependendo do tipo de configuração que o usuário espera obter. Como essas rotas são descritas em termos de configurações do gerenciador de filas, segue uma descrição resumida do gerenciador de filas MQe e dos componentes associados.

Na tabela a seguir, as etapas necessárias para configurar cada tipo de gerenciador de filas são relacionadas, juntamente com os capítulos correspondentes deste manual. A configuração do Gerenciador de Filas Básico é um pré-requisito de todas as outras configurações; ou seja, qualquer gerenciador de filas deve ser configurado primeiro como Gerenciador de Filas Básico. Em seguida, outros tipos de funcionalidade podem ser incluídos, conforme necessário.

Portanto:

Para configurar um Cliente
Siga as etapas 1, 2, 3, 4 e 5

Para configurar um Servidor
Siga as etapas 1, 2, 6 e 7

Para configurar um gerenciador de filas com as funcionalidades de Servidor e Cliente
Siga as etapas de 1 a 7, inclusive

Tabela 1. Configurando Clientes, Servidores e Gerenciadores de Filas

Etapas de Requisito	Tópicos
Gerenciador de Filas Básico	
1. Crie e inicie o gerenciador de filas	“Configurando com Mensagens” na página 14

Tabela 1. Configurando Clientes, Servidores e Gerenciadores de Filas (continuação)

Etapas de Requisito	Tópicos
2. Crie uma fila local	"Configurando Gerenciadores de Filas" na página 41 "Configurando Filas Locais" na página 50
Gerenciador de filas do cliente	
3. Crie uma definição de conexão para um servidor	"Configurando Definições de Conexão" na página 76
4. Crie uma definição de fila remota	"Configurando Filas Remotas" na página 61
5. Crie uma fila de servidor home para transmissão acionada (requerido para filas assíncronas remotas)	"Configurando Filas de Servidor Home" na página 67
Gerenciador de filas do servidor	
6. Crie um listener	"Configurando um Listener" na página 82
7. Crie uma fila de armazenamento e redirecionamento (opcional)	"Configurando Filas de Armazenamento e Redirecionamento" na página 70
8. Inclua a funcionalidade de ponte	"Configurando Recursos de Ponte/Gateway" na página 84

Visão Geral de Objetos MQe

Gerenciador de Filas

Um gerenciador de filas possui e controla mensagens, filas e conexões MQe (consulte a seguir). Ele permite que aplicativos acessem mensagens e filas. Cada gerenciador de filas possui um nome exclusivo que o distingue de qualquer outro gerenciador de filas MQe. Dependendo das necessidades de um aplicativo, os gerenciadores de filas podem ser diferentes quanto a coletas de filas, mensagens, conexões e outros objetos e também quanto à função que desempenham em uma configuração.

O MQe identifica três funções distintas para os gerenciadores de filas, além da funcionalidade do gerenciador de filas básicas:

- **Cliente**: Um gerenciador de filas que fornece mensagens para, ou recebe mensagens de um servidor
- **Servidor**: Um gerenciador de filas que fornece serviços para vários gerenciadores de filas do cliente conectados
- **Gateway**: Um gerenciador de filas de servidor que também possui o recurso de troca de mensagens com os gerenciadores de filas do sistema de mensagens base MQ

Fila

Uma fila pode ser utilizada para armazenar, processar ou mover mensagens. Cada fila pertence a um gerenciador de filas e os aplicativos podem acessar as filas por meio do gerenciador de filas. Cada fila possui um nome exclusivo que a distingue de qualquer outra fila nesse mesmo gerenciador de filas. As filas locais não são rigorosamente obrigatórias, entretanto, não há muito o que fazer sem elas.

Mensagem

Uma mensagem é uma coleta de dados que podem ser armazenados em uma fila ou movidos através de uma rede MQe.

Conexão

Uma conexão fornece seu gerenciador de filas locais com as informações que ele precisa para estabelecer links de comunicação com um gerenciador de filas remotas. O nome de uma conexão é o nome desse gerenciador de filas remotas. Apenas uma definição de conexão pode existir em um gerenciador de filas

locais para cada nome do gerenciador de filas remotas.

Canal

Um canal é uma entidade que permite que um gerenciador de filas mova mensagens para um gerenciador de filas remotas.

Registro

O registro é o armazenamento primário para informações relacionadas ao gerenciador de filas. Cada gerenciador de filas possui seu próprio registro. Cada gerenciador de filas utiliza o registro para armazenar detalhes de suas propriedades e objetos.

Gerenciadores de Filas

Não importa qual função um gerenciador de filas desempenha, existe uma quantidade básica de configuração requerida. Essa configuração básica resulta no que chamamos aqui de *Gerenciador de Filas Básico*. Dependendo do tipo de função desejada para o gerenciador de filas, esse Gerenciador de Filas Básico é estendido, resultando em um Gerenciador de Filas do Cliente, um Gerenciador de Filas do Servidor ou um Gerenciador de Filas de Gateway. O diagrama a seguir tenta resumir essas configurações:

Tabela 2. Configuração do Gerenciador de Filas

Gerenciador de Filas Básico	+	Definição de Conexão e Definição de Fila Remota	=	Gerenciador de filas do cliente
Gerenciador de Filas Básico	+	Listener	=	Gerenciador de filas do servidor
Gerenciador de Filas Básico	+	Funcionalidade de Ponte	=	Gerenciador de Filas de Gateway
Gerenciador de Filas Básico	+	Configuração de Segurança, etc.		

O ciclo de vida completo de gerenciamento pode ser controlado, para a maioria dos recursos gerenciados, com mensagens de administração. Isso significa que o recurso gerenciado pode ser trazido, gerenciado e, em seguida, excluído com mensagens de administração. Este não é o caso dos gerenciadores de filas. Antes de um gerenciador de filas ser gerenciado, ele deve ser criado e iniciado.

O gerenciador de filas possui poucas características próprias, mas controla outros recursos MQe. Ao consultar um gerenciador de filas, você pode obter uma lista de conexões com outros gerenciadores de filas e uma lista de filas com as quais o gerenciador de filas pode trabalhar. Cada item da lista é o nome de uma conexão ou fila. Uma vez conhecido o nome de um recurso, você pode utilizar a mensagem apropriada para gerenciá-lo. Por exemplo, utilize MQeConnectionAdminMessage para gerenciar conexões.

Conexões

As conexões definem como conectar um gerenciador de filas a outro gerenciador de filas. Assim que uma conexão é definida, um gerenciador de filas pode enviar mensagens para filas no gerenciador de filas remotas. O diagrama a seguir mostra as partes constituintes que são necessárias para uma fila remota em um gerenciador de filas se comunicar com uma fila em um gerenciador de filas diferente:

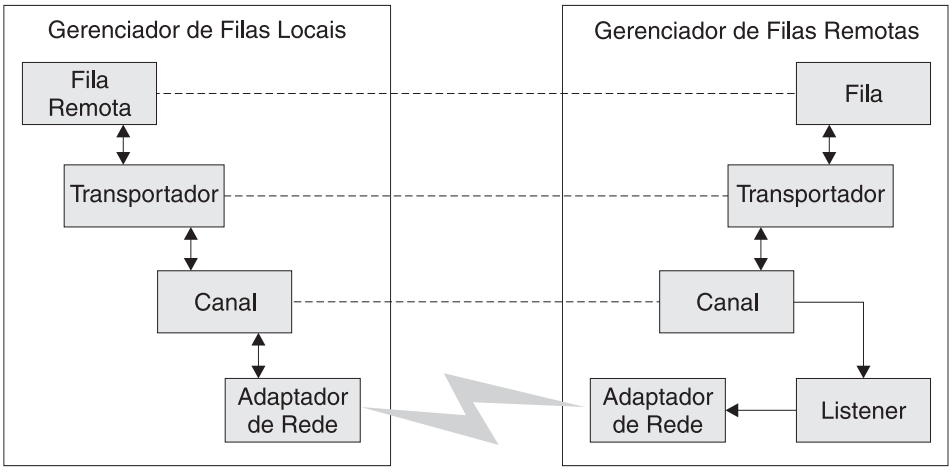


Figura 1. Conexões do Gerenciador de Filas

A comunicação ocorre em níveis diferentes:

Transportador:
Conexão lógica entre duas filas

Canal: Conexão lógica entre dois sistemas

Adaptador:
Comunicação específica do protocolo

O canal e o adaptador são especificados como parte de uma definição de conexão. O transportador é especificado como parte de uma definição de fila remota. O código de exemplo a seguir mostra um método que instancia e prepara uma MQeConnectionAdminMsg para criar uma conexão:

```
/**
* Configurar uma mensagem admin para criar uma definição de conexão
 */
public MQeConnectionAdminMsg addConnection( remoteQMgr
     adapter,
                  parms,
                  options,
                  channel,
                  description ) throws Exception
{
  String remoteQMgr = "ServerQM";
  /*
   * Criar um campo de parâmetros e de mensagens admin do gerenciador de filas vazias
   */
  MQeConnectionAdminMsg msg = new MQeConnectionAdminMsg();

  /*
   * Preparar mensagem com o destinatário para resposta e um identificador exclusivo
   */
  MQeFields msgTest = primeAdminMsg( msg );

  /*
   * Configurar o nome do gerenciador de filas para o qual incluir rotas
   */
  msg.setName( remoteQMgr );

  /*
   * Configurar a ação admin para criar uma nova fila
   * A conexão está configurada para utilizar um canal padrão. Este é um alias
   * que precisa ser configurado no gerenciador de filas para que a conexão
   * funcione.
   */
```

```
  msg.create( adapter,
              parms,
              options,
              channel,
              description );

  return msg;
}
```

Utilize MQeConnectionAdminMsg para configurar a parte do cliente de uma conexão. O tipo de canal é `com.ibm.mqe.MQeChannel`. Normalmente um alias de `DefaultChannel` está configurado para o MQeChannel. O fragmento de código a seguir mostra como configurar uma conexão em um cliente para comunicar-se com um servidor utilizando o protocolo HTTP.

```
/**
 * Criar uma mensagem admin de conexão que cria uma definição de
 * conexão para um gerenciador de filas remotas utilizando o protocolo HTTP. Depois
 * enviar a mensagem para o gerenciador de filas de cliente.
 */
public addClientConnection( MQeQueueManager myQM,
   String targetQMgr ) throws Exception
{
  String remoteQMgr  = "ServerQM";
  String adapter     = "Network:127.0.0.1:80";
// Presume-se que um alias denominado Network tenha sido configurado para
// o adaptador de rede com.ibm.mqe.adapters.MQeTcpipHttpAdapter
  String parameters  = null;
  String options     = null;
  String channel     = "DefaultChannel";
  String description = "client connection to ServerQM";

  /*
   * Configurar a mensagem admin
   */
  MQeConnectionAdminMsg msg = addConnection( remoteQMgr,
                                             adapter,
                                             parameters,
                                             options,
                                             channel,
                                             description );

  /*
   *  Enviar a mensagem admin para a fila admin (não utilizando fluxos assegurados)
   */
  myQM.putMessage(targetQMgr,
   MQe.Admin_Queue_Name,
   msg,
   null,
   0 );

}
```

Rotas e Aliases

Conexões de Rotas

É possível configurar uma conexão para que um gerenciador de filas roteie mensagens através de um gerenciador de filas intermediárias. Isso requer duas conexões:

1. Uma conexão com o gerenciador de filas intermediárias
2. Uma conexão com o gerenciador de filas de destino

A primeira conexão é criada pelos métodos descritos anteriormente nesta seção, como uma conexão de cliente ou período. Para a segunda conexão, o nome do gerenciador de filas intermediárias é especificado no lugar do nome do adaptador de rede. Com essa configuração, um aplicativo pode enviar mensagens

no gerenciador de filas de destino mas roteá-las por meio de um ou mais gerenciadores de filas intermediárias.

Aliases

É possível designar vários nomes ou aliases a uma conexão. Quando um aplicativo chama métodos na classe MQeQueueManager que requerem a especificação de um nome de gerenciador de filas, ele também pode utilizar um alias.

Você pode designar um alias para gerenciadores de filas locais e remotas. Para designar um alias a um gerenciador de filas locais, é necessário primeiro estabelecer uma definição de conexão com o mesmo nome que o gerenciador de filas locais. Essa é uma conexão lógica que pode ter todos os parâmetros configurados como nulos.

Para incluir e remover aliases, utilize as ações Action_AddAlias e Action_RemoveAlias da classe MQeConnectionAdminMsg. É possível incluir ou remover vários aliases em uma mensagem. Coloque os aliases a serem manipulados diretamente na mensagem, configurando o campo de matriz ASCII *Con_Aliases*. Alternativamente, você pode utilizar os dois métodos: addAlias() ou removeAlias(). Cada um desses métodos obtém um único nome de alias, mas o método pode ser chamado repetidamente para incluir vários aliases em uma mensagem.

O seguinte snippet de código mostra como incluir aliases de conexão em uma mensagem:

```
/**
 * Configurar uma mensagem admin para incluir aliases
    em um gerenciador de filas (conexão)
 */
public MQeConnectionAdminMsg addAliases(  String queueManagerName
                                  String aliases[] )
                                  throws Exception
{
  /*
   * Criar uma mensagem admin de conexão vazia
   */
  MQeConnectionAdminMsg msg = new MQeConnectionAdminMsg();

  /*
   * Preparar mensagem com o destinatário
     para resposta e um identificador exclusivo  */
  MQeFields msgTest = primeAdminMsg( msg );

  /*
   * Configurar o nome da conexão para incluir aliases
   */
  msg.setName( queueManagerName );

  /*
   * Utilizar o método addAlias para incluir aliases na mensagem.
   */
  for ( int i=0; i<aliases.length; i++ )
  {
    msg.addAlias( aliases[i] );
  }

  return msg;
}
```

Filas

A mais simples das filas é uma fila local implementada na classe MQeQueue e é gerenciada pela classe MQeQueueAdminMsg. Todos os outros tipos de fila são herdados de MQeQueue. Para cada tipo de fila,

existe uma mensagem de administração correspondente que é herdada de MQeQueueAdminMsg. As seções a seguir descrevem a administração dos vários tipos de filas.

Fila Local

Você pode criar, atualizar, excluir e consultar filas locais e seus descendentes utilizando as ações de administração fornecidas no MQe. O mecanismo de administração básica é herdado de MQeAdminMsg.

O nome de uma fila é formado pelo nome do gerenciador de filas de destino (para uma fila local, este é o nome do gerenciador de filas que possui a fila) e por um nome exclusivo para a fila nesse gerenciador de filas. Dois campos na mensagem de administração são utilizados para identificar exclusivamente a fila; eles são os campos ASCII *Admin_Name* e *Queue_QMgrName*. Você pode utilizar o método setName(queueManagerName, queueName) para configurar esses dois campos na mensagem de administração.

O diagrama a seguir mostra um exemplo de gerenciador de filas configurado com uma fila local. O gerenciador de filas `qm1` possui uma fila local denominada `invQ`. A característica de nome do gerenciador de filas da fila é qm1, que corresponde ao nome do gerenciador de filas. O seguinte diagrama mostra uma fila local:

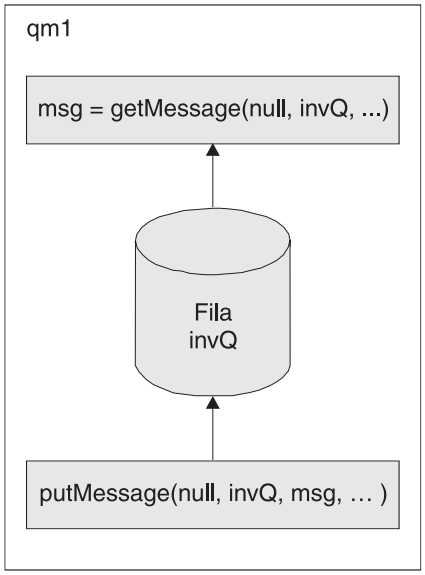


Figura 2. Fila local

Armazenamento de Mensagem:

As filas locais requerem um armazenamento de mensagem para armazenar suas mensagens. Cada fila pode especificar o tipo de armazenamento a ser utilizado e onde ele está localizado. Utilize a característica da fila *Queue_FileDesc* para especificar o tipo de armazenamento de mensagem e para fornecer seus parâmetros. O tipo de campo é `ascii` e o valor deve ser um descritor de arquivo no formato:

classe do adaptador:parâmetros do adaptador
ou
alias do adaptador:parâmetros do adaptador

Por exemplo:

```
MsgLog:d:\QueueManager\ServerQM12\Queues
```

O MQe Versão 2.1 fornece dois adaptadores, um para gravar mensagens em disco e outro para armazená-las na memória. Criando um adaptador apropriado, as mensagens podem ser armazenadas em qualquer local ou meio apropriado (como banco de dados DB2 ou CDs graváveis).

A opção do adaptador determina a persistência e a resiliência das mensagens. Por exemplo, se um adaptador de memória for utilizado, as mensagens serão apenas tão resilientes quanto a memória. A memória pode ser um meio mais rápido que o disco, mas é altamente volátil em comparação com o disco. Portanto, a opção do adaptador é algo importante.

Se você não fornecer informações de armazenamento de mensagem ao criar uma fila, será assumido o armazenamento de mensagem que foi especificado durante a criação do gerenciador de filas.

Considere o seguinte ao configurar o campo *Queue_FileDesc*:

- Assegure-se de que a sintaxe correta seja utilizada para o sistema no qual a fila reside. Por exemplo, em um sistema Windows, utilize "\" como um separador de arquivo. Em sistemas UNIX, utilize "/" como um separador de arquivo. Em alguns casos, é possível utilizar ambos, mas isso depende do suporte fornecido pela JVM (Java Virtual Machine) na qual o gerenciador de filas é executado. Além das diferenças de separador de arquivo, alguns sistemas utilizam letras de unidade (como Windows NT) enquanto outros (como UNIX) não as utilizam.
- Em alguns sistemas, é possível especificar diretórios relativos (" .\"), enquanto em outros isso não é possível. Mesmo naqueles em que diretórios relativos podem ser especificados, eles devem ser utilizados com muito cuidado pois o diretório atual pode ser alterado durante a existência da JVM. Essa alteração causa problemas ao interagir com filas que utilizam diretórios relativos.

Criando uma Fila Local:

O seguinte fragmento de código demonstra como criar uma fila local:

```
/**
 * Criar uma nova fila local
 */
protected void createQueue(MQeQueueManager localQM,
                           String          qMgrName,
                           String          queueName,
                           String          description,
                           String          queueStore
   ) throws Exception
{
  /*
   * Criar um campo de mensagem admin e parâmetros de fila vazia
   */
  MQeQueueAdminMsg msg = new MQeQueueAdminMsg();
  MQeFields parms = new MQeFields();

  /*
   * Preparar mensagem com o destinatário para resposta e um identificador exclusivo
   */
  MQeFields msgTest = primeAdminMsg( msg );

  /*
   * Definir nome da fila para gerenciar
   */
  msg.setName( qMgrName, queueName );

  /*
   * Incluir qualquer característica da fila aqui, caso contrário,
   * as características serão deixadas com os valores padrão.
   /
  if ( description != null )  // configurar a descrição ?
    parms.putUnicode( MQeQueueAdminMsg.Queue_Description,
                      description);
```

```
  if ( queueStore != null )   // Configurar o armazenamento de fila ?
    // Se o armazenamento de fila incluir informações sobre diretório e arquivo,
    // ele deverá ser configurado para o estilo correto do sistema no qual a
    // fila residirá; por exemplo \ ou /
    parms.putAscii(MQeQueueAdminMsg.Queue_FileDesc,
                   queueStore );
  /*
   * Outras características da fila, como profundidade da fila e expiração da mensagem,
   * podem ser configuradas aqui ...
   */

  /*
   * Configurar a ação admin para criar uma nova fila
   */
  msg.create( parms );

  /*
   *  Enviar a mensagem admin para a fila admin (entrega não assegurada)
   */
  localQM.putMessage( qMgrName,
                      MQe.Admin_Queue_Name,
                      msg,
                      null,
                      0);
}
```

Segurança da Fila:

O acesso e a segurança são de propriedade da fila e podem ser concedidos para ser utilizados por um gerenciador de filas remotas (quando conectado a uma rede), permitindo que outros gerenciadores de filas na rede enviem mensagens para a fila ou recebam mensagens dela. As seguintes características são utilizadas ao configurar a segurança da fila:

- *Queue_Cryptor*
- *Queue_Authenticator*
- *Queue_Compressor*
- *Queue_TargetRegistry*
- *Queue_AttrRule*

Se um criptografador ou autenticador tiver sido especificado em uma fila, o gerenciador de filas precisará ter um registro privado definido. Qualquer outro gerenciador de filas que tenha filas remotas direcionadas a uma fila com segurança também deverá ter um registro privado. A única exceção a esse requisito é ao utilizar as filas síncronas remotas.

Outras Características da Fila:

Você pode configurar as filas com várias outras características, como o número máximo de mensagens permitidas na fila. Para obter uma descrição delas, consulte a seção *MQeQueueAdminMsg* do Java API Programming Reference.

Aliases:

Os nomes das filas podem ter aliases semelhantes àqueles descritos para conexões em “Rotas e Aliases” na página 5. O fragmento de código no exemplo de alias da seção de conexões mostra como configurar aliases em uma conexão. A configuração de aliases em uma fila é igual, exceto pelo fato de que uma MQeQueueAdminMsg é utilizada no lugar de uma MQeConnectionAdminMsg.

Restrições da Ação:

Algumas ações administrativas podem ser desempenhadas apenas quando a fila está em um estado predefinido, conforme a seguir:

Action_Update

- Se a fila estiver em uso, as características da fila não poderão ser alteradas
- As características de segurança da fila não poderão ser alteradas se houver mensagens na fila
- O armazenamento de mensagem de fila não poderá ser alterado depois de ser configurado

Action_Delete

A fila não poderá ser excluída se ela estiver em uso ou se houver mensagens na fila

Se o pedido exigir que a fila não esteja em uso ou que não tenha nenhuma mensagem, o pedido de administração poderá ser tentado novamente, quando o gerenciador de filas for reiniciado ou em intervalos regulares de tempo. Consulte "A Mensagem de Administração Básica" na página 17 para obter detalhes sobre como configurar uma nova tentativa de pedido de administração.

Fila de Servidor Home

As filas de servidor home são implementadas pela classe MQeHomeServerQueue. Elas são gerenciadas com a classe MQeHomeServerQueueAdminMsg, que é uma subclasse da MQeRemoteQueueAdminMsg. A única inclusão na subclasse é a característica *Queue_QTimerInterval*. Esse campo é do tipo int e é configurado para um intervalo do cronômetro em milissegundos. Se você configurar esse campo para um valor maior que zero, a fila do servidor home verificará o servidor home a cada `n` milissegundos para saber se há mensagens esperando pela coleta. Qualquer mensagem em espera é entregue à fila de destino. Um valor igual a `0` para esse campo significa que o servidor home é controlado apenas quando o método MQeQueueManager.triggertransmission é chamado.

Nota: Se ocorrer uma falha na conexão de uma fila de servidor home com sua fila de armazenamento e redirecionamento (por exemplo, se a fila de armazenamento e redirecionamento não estiver disponível quando a fila de servidor home for iniciada), a tentativa será cessada até que uma chamada de transmissão de acionamento seja efetuada.

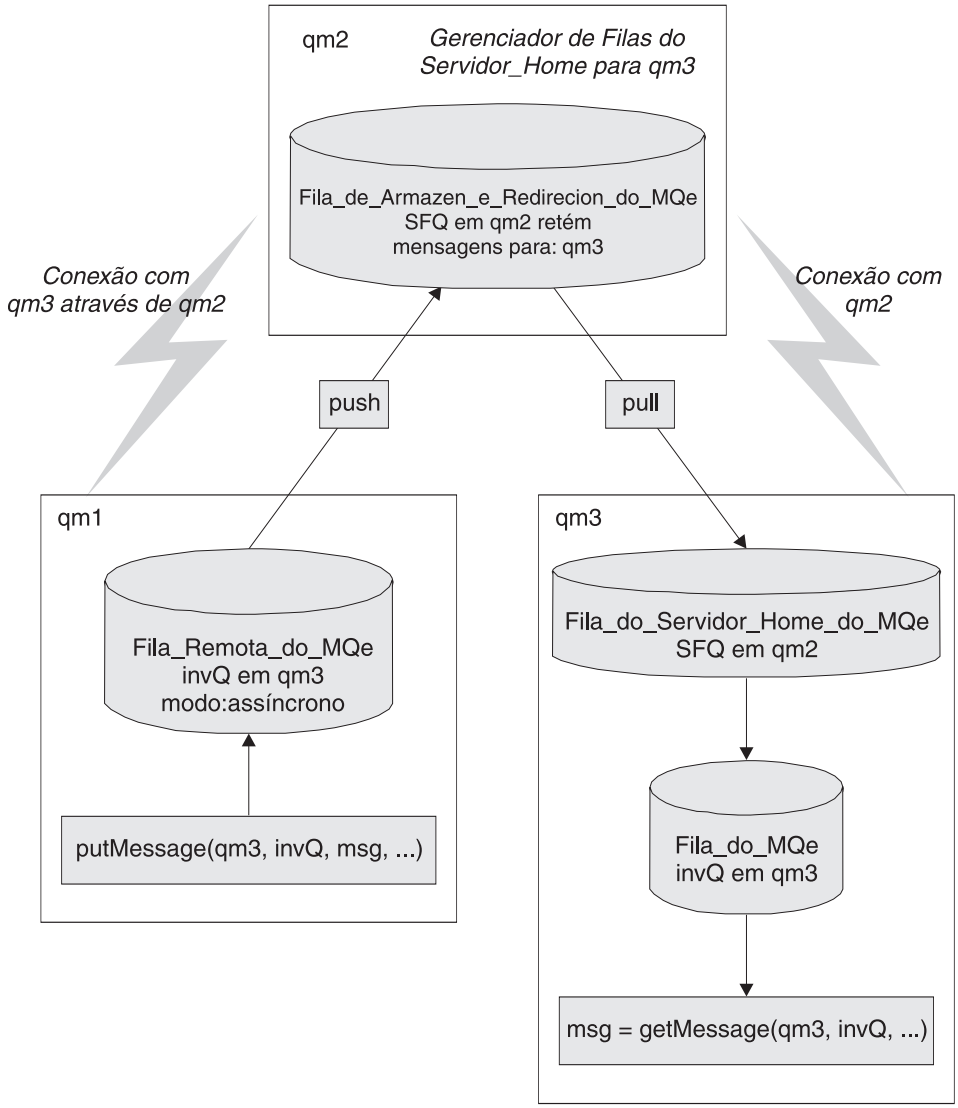


Figura 3. Fila de Servidor Home

O nome da fila de servidor home é configurado da seguinte forma:

- O nome da fila deve corresponder ao nome da fila de armazenamento e redirecionamento
- O atributo de gerenciador de filas do nome da fila deve ser o nome do gerenciador de filas de servidor home

O gerenciador de filas no qual a fila de servidor home reside deve ter uma conexão configurada com o gerenciador de filas do servidor home.

A Figura 3 mostra o exemplo de um gerenciador de filas `qm3` que possui uma fila de servidor home `SFQ` configurada para coletar mensagens do seu gerenciador de filas de servidor home `qm2`.

A configuração consiste em:

- Um gerenciador de filas de servidor home `qm2`
- Uma fila de armazenamento e redirecionamento `SFQ` no gerenciador de filas `qm2` que mantém as mensagens do gerenciador de filas `qm3`
- Um gerenciador de filas `qm3` que normalmente é executado desconectado e não pode aceitar conexões do gerenciador de filas `qm2`

- O gerenciador de filas qm3 possui uma conexão configurada com o qm2
- Uma fila de servidor home SFQ que utiliza o gerenciador de filas qm2 como seu servidor home

Qualquer mensagem direcionada ao gerenciador de filas qm3 por meio do qm2 é armazenada na fila de armazenamento e redirecionamento SFQ no qm2 até que seja coletada pela fila de servidor home no qm3.

Fila de Ponte do MQ

Uma fila de ponte MQ é uma definição de fila remota que refere-se a uma fila que reside em um gerenciador de filas MQ. A fila que mantém as mensagens reside no gerenciador de filas MQ, não no gerenciador de filas locais.

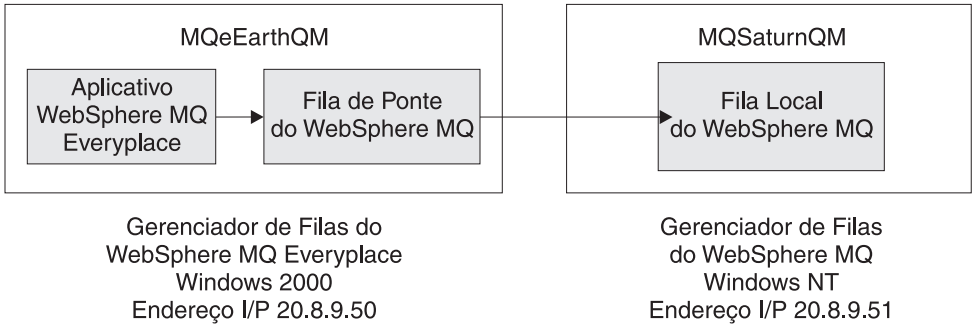


Figura 4. Fila de Ponte MQ

- O gerenciador de filas do MQ MQSaturnQM possui uma fila local MQSaturnQ definida.
- O MQeEarthQM deve ter uma fila de ponte do MQ definida, denominada MQSaturnQ, no gerenciador de filas MQSaturnQM.
- Os aplicativos conectados ao gerenciador de filas MQeEarthQM enviam mensagens para a fila de ponte do MQ MQSaturnQ e a fila de ponte entrega as mensagens para a MQSaturnQ no gerenciador de filas MQSaturnQM.

A definição da fila de ponte requer a especificação dos nomes da ponte, do proxy do gerenciador de filas MQ e da conexão do cliente, a fim de identificar exclusivamente um objeto de conexão do cliente na hierarquia de objetos de ponte. Consulte a Figura 17 na página 87 para obter informações adicionais. Essas informações identificam como a ponte MQ acessa o gerenciador de filas MQ para manipular uma fila MQ.

A fila de ponte MQ fornece o recurso para enviar uma fila para um gerenciador de filas que não esteja conectado diretamente à ponte MQ. Isso permite que uma mensagem seja enviada para um gerenciador de filas MQ (o destino) roteado através de um outro gerenciador de filas MQ. A fila de ponte MQ obtém o nome do gerenciador de filas de destino e o gerenciador de filas intermediárias é nomeado pelo proxy do gerenciador de filas MQ.

Para obter uma lista completa das características utilizadas pela fila de ponte MQ, consulte *MQeMQBridgeQueueAdminMsg* na seção *com.ibm.mqe.bridge* do Java Programming Reference.

A tabela a seguir detalha a lista de operações suportadas pela fila de ponte MQ, assim que ela tiver sido configurada:

Tabela 3. Operações de Mensagens Suportadas pela Fila de Ponte do MQ

Tipo de Operação	**Suportada pela Fila de Ponte MQ**
getMessage()	Sim*
putMessage()	Sim

Tabela 3. Operações de Mensagens Suportadas pela Fila de Ponte do MQ (continuação)

Tipo de Operação	Suportada pela Fila de Ponte MQ
browseMessage()	Sim*
browseAndLockMessage	Não
Nota: * essas funções possuem restrições de uso.	

Se um aplicativo tentar utilizar uma das operações não suportadas, uma MQeException de `Except_NotSupported` será retornada.

Quando um aplicativo envia uma mensagem para a fila de ponte, uma conexão lógica com o gerenciador de filas MQ é estabelecida pela fila de ponte a partir do conjunto de conexões mantido pelo objeto de conexão do cliente da ponte. A conexão lógica com o MQ é fornecida pelas classes de Ligações MQ Java ou pelas Classes MQ para Java. As opções de classes dependem do valor do campo *`hostname`* nas configurações de proxy do gerenciador de filas MQ. Depois de estabelecer uma conexão com o gerenciador de filas MQ, a fila de ponte MQ tenta enviar a mensagem para a fila MQ.

Uma fila de ponte MQ sempre deve ter um modo de acesso síncrono e não pode ser configurada como uma fila assíncrona. Isso significa que, se a operação de entrada estiver manipulando diretamente uma fila de ponte MQ e retornar êxito, sua mensagem foi transmitida ao sistema MQ enquanto o processo estava aguardando a conclusão dessa operação.

Se você não desejar utilizar operações síncronas com a fila de ponte MQ, poderá configurar uma definição de fila remota assíncrona que faça referência à fila de ponte MQ. Como alternativa, é possível configurar uma fila de armazenamento e redirecionamento e uma fila de servidor home. Essas duas configurações alternativas fornecem ao aplicativo uma fila assíncrona na qual pode enviar mensagens. Com essas configurações, quando o método `putMessage()` retorna, não significa necessariamente que a mensagem tenha sido transmitida ao gerenciador de filas MQ.

Um exemplo de uso da fila de ponte MQ é descrito em “Exemplo de Configuração de Ponte” na página 97.

Fila de Administração

A fila de administração é implementada na classe MQeAdminQueue e é uma subclasse de MQeQueue, portanto possui os mesmos recursos que uma fila local. Ela é gerenciada utilizando a classe de administração MQeAdminQueueAdminMsg.

Se uma mensagem falhar porque o recurso a ser administrado está em uso, é possível solicitar que a mensagem seja tentada novamente. “A Mensagem de Administração Básica” na página 17 fornece detalhes sobre como configurar a contagem do número máximo de tentativas. Se a mensagem falhar porque o recurso gerenciado não está disponível e o número máximo de tentativas não tiver sido atingido, a mensagem será deixada na fila para ser processada em uma data posterior. Se o número máximo de tentativas tiver sido atingido, o pedido falhará com uma MQeException. Por padrão, a mensagem será tentada novamente na próxima vez em que o gerenciador de filas for iniciado. Alternativamente, é possível configurar um cronômetro na fila que processa mensagens da fila em intervalos especificados. O intervalo do cronômetro é especificado configurando o campo longo *`Queue_QTimerInterval`* na mensagem de administração. O valor do intervalo é especificado em milissegundos.

Segurança e Administração

Por padrão, qualquer aplicativo MQe pode administrar recursos gerenciados. O aplicativo pode ser executado como um aplicativo local no gerenciador de filas que está sendo gerenciado ou pode ser executado em um gerenciador de filas diferente. É importante que as ações de administração sejam

seguras, caso contrário, é provável que o sistema seja utilizado incorretamente. O MQe fornece os recursos básicos para proteger a administração utilizando a segurança baseada em filas, conforme descrito neste centro de informações.

Se você utilizar uma segurança síncrona, poderá proteger a fila de administração configurando as características de segurança sobre a fila. Por exemplo, você pode configurar um autenticador para exigir que o usuário seja autenticado no sistema operacional (Windows NT ou UNIX) antes de desempenhar ações administrativas. Isso pode ser estendido de modo que apenas um usuário específico possa desempenhar a administração.

A fila de administração não permite aos aplicativos o acesso direto a mensagens da fila, as mensagens são processadas internamente. Isso significa que as mensagens enviadas para a fila que foi protegida com a segurança no nível de mensagem não podem ser desagrupadas utilizando o mecanismo normal de fornecimento de um atributo em um pedido de recebimento ou procura. Entretanto, uma classe de regra de fila pode ser aplicada à fila de administração para desagrupar quaisquer mensagens protegidas, de modo que possam ser processadas pela fila de administração. A regra de fila `browseMessage()` deve ser codificada para desempenhar esse desagrupamento e para permitir que ocorra a administração.

Configurando com Mensagens

Este tópico explica como você pode administrar recursos MQe, local ou remotamente, utilizando mensagens de administração.

Visão Geral de Configuração Utilizando Mensagens

Você pode administrar recursos MQe utilizando mensagens especializadas, denominadas mensagens de administração (mensagens admin). A utilização dessas mensagens permite administrar recursos local ou remotamente. O código-base nativo, se configurado com uma fila de administração (fila admin), responde a mensagens admin. Entretanto, ele não fornece funções auxiliares para criar mensagens admin. Para obter informações adicionais sobre isso, consulte “Configurando com a API do Administrador C” na página 31. O Java é administrado por mensagens admin. C pode ser, mas possui uma interface de administração para administração local.

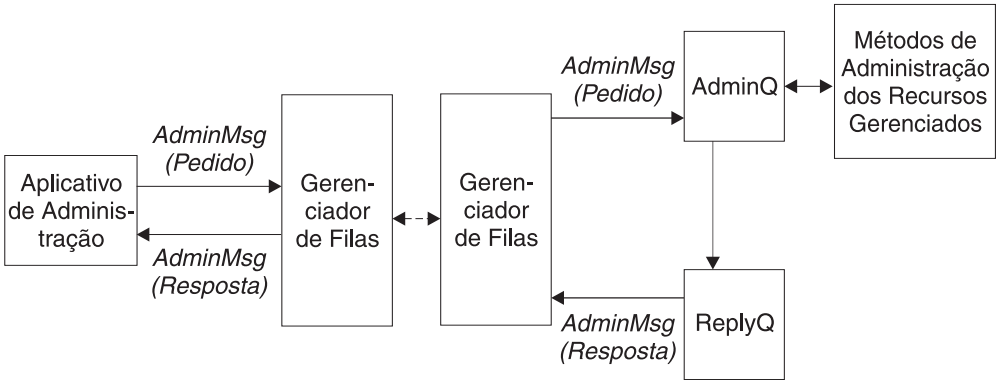


Figura 5. Administração do MQe Utilizando Mensagens de Administração

Estas são as etapas a serem seguidas ao utilizar mensagens de administração para administrar um recurso:

1. Crie uma fila admin no recurso que está desempenhando a administração ou certifique-se de que exista alguma.

2. Crie uma mensagem admin apropriada para o recurso que está sendo gerenciado.
3. Configure os campos requeridos na mensagem.
4. Envie a mensagem admin na fila admin apropriada.
5. Espere uma mensagem de resposta admin na fila de resposta admin apropriada, se uma resposta tiver sido solicitada na mensagem admin.
6. Analise os dados na mensagem de resposta admin.

A Fila de Administração

Antes de administrar um gerenciador de filas (ou seus recursos) utilizando mensagens admin, você deve iniciar o gerenciador de filas e configurar nele uma fila admin. A função da fila admin é processar mensagens admin na seqüência em que elas chegam na fila. Apenas um pedido é processado por vez.

Java

Em Java, a fila pode ser criada utilizando o método `defineDefaultAdminQueue()` da classe MQeQueueManagerConfigure. O nome da fila é `AdminQ` e os aplicativos podem referir-se a ele utilizando a constante `MQe.Admin_Queue_Name`.

C

No código-base nativo, uma fila admin é criada utilizando a seguinte API:

```
MQeAdminQParms params = ADMIN_Q_INIT_VAL;
rc = mqeAdministrator_AdminQueue_create(hAdmin, // manipular para o MQeAdministrator
         pExceptBlock, // manipular para um bloco de exceções
         hQueueName, // o nome da fila a ser criada
         hQueueQMgrName, // o nome do gerenciador de filas que
               //possui a fila
         &params); // ponteiro para estrutura
              // para configurar a
              // fila do tipo MQeAdminQParms,
```

Em particular, o manipulador de cadeia constante `MQE_ADMIN_QUEUE_NAME` pode ser utilizado como o nome da fila admin. Esse é o equivalente da constante `MQe.Admin_Queue_Name` no código-base Java.

A estrutura `params` pode ser inicializada para conter valores padrão para todas as propriedades da fila admin. A estrutura também contém um elemento de máscara de bit opFlags que deve ser utilizado para indicar quais propriedades foram configuradas para um valor diferente do valor padrão. O exemplo anterior aceita todos os valores padrão, conforme especificados utilizando a constante `ADMIN_Q_INIT_VAL`.

A Fila de Resposta de Administração

Este tópico descreve o uso de filas de resposta de administração em Java e C.

Java

Em Java, um aplicativo de administração típico instancia uma subclasse de `MQeAdminMsg`, configura-a com o pedido de administração necessário e a transmite para a `AdminQ` no gerenciador de filas de destino. Se o aplicativo precisar saber o resultado da ação, uma resposta poderá ser solicitada. Quando o pedido tiver sido processado, o resultado do pedido será retornado em uma mensagem para a fila de resposta e o gerenciador de filas especificado na mensagem de pedido.

A resposta pode ser enviada para qualquer gerenciador de filas ou fila, mas você pode configurar uma fila de resposta padrão que seja utilizada unicamente para mensagens de resposta de administração. Essa fila padrão é criada utilizando o método `defineDefaultAdminReplyQueue()` da classe

MQeQueueManagerConfigure. O nome da fila é `AdminReplyQ`, e os aplicativos podem referir-se a ela utilizando a constante `MQe.Admin_Reply_Queue_Name`.

C

No código-base nativo, como no código-base Java, qualquer fila pode ser especificada como a fila de resposta admin. Entretanto, é recomendável que o nome da fila de resposta admin padrão, `MQE_ADMIN_REPLY_QUEUE_NAME`, seja utilizado para nomear uma fila dedicada à função de fila de resposta admin. Esse nome corresponde a `MQe.Admin_Reply_Queue_Name` no código-base Java.

Na prática, é mais provável que o cliente nativo receba do que envie mensagens admin. Neste caso, o cliente precisa de uma definição de fila assíncrona remota da fila de resposta admin no servidor, bem como uma fila de servidor home que corresponda a uma fila de armazenamento e redirecionamento no servidor, para permitir que as mensagens admin e de resposta admin sejam transferidas.

Criar a Mensagem de Administração Apropriada

A fila de administração não sabe como desempenhar a administração de recursos individuais. As informações são encapsuladas em cada recurso e em sua mensagem correspondente.

Java

Em Java, existe uma hierarquia de tipos de mensagens de administração. Para determinadas operações, o tipo exato de mensagem de administração é necessário. Por exemplo, para criar uma 'fila' de Servidor Home, é necessária uma mensagem de administração de Fila de Servidor Home. Para outras operações, uma mensagem de administração mais geral é apropriada. Por exemplo, para consultar uma fila de servidor home, você pode utilizar uma mensagem de administração de fila ou uma mensagem de administração de fila remota. No caso de dúvida, utilize o tipo exato de mensagem de administração.

As seguintes mensagens são fornecidas para a administração de recursos MQe:

Tabela 4. Mensagens de Administração

Nome da Mensagem	Propósito
MQeAdminMsg	Uma classe abstrata que age como a classe-base para todas as mensagens de administração
MQeAdminQueueAdminMsg	Fornece suporte para administrar a fila de administração
MQeConnectionAdminMsg	Fornece suporte para administrar conexões entre os gerenciadores de filas
MQeHomeServerQueueAdminMsg	Fornece suporte para administrar filas de servidor home
MQeQueueAdminMsg	Fornece suporte para administrar filas locais
MQeQueueMangerAdminMsg	Fornece suporte para administrar gerenciadores de filas
MQeRemoteQueueAdminMsg	Fornece suporte para administrar filas remotas
MQeStoreAndForwardQueueAdminMsg	Fornece suporte para administrar filas de armazenamento e redirecionamento
MQeCommunicationsListenerAdminMsg	Fornece suporte para administrar listeners de comunicação

Essas mensagens de administração base são fornecidas no pacote com.ibm.mqe.administration. É possível gerenciar outros tipos de recursos subclassificando MQeAdminMsg ou uma das mensagens de

administração existentes. Por exemplo, uma mensagem de administração adicional para gerenciar a ponte MQ é fornecida no pacote com.ibm.mqe.mqbridge.

C

No código-base C, todas as mensagens são instâncias MQeFields. Isso aplica-se a mensagens admin. Os tipos de mensagens admin são diferenciados por um campo especial inserido no objeto fields. É necessário criar uma mensagem admin totalmente nova, do tipo apropriado, inserindo todos os campos requeridos. Alternativamente, para administração local, utilize a API de administração nativa. O código-base nativo pode responder corretamente a todas as mensagens de administração, mas a API de administração nativa é geralmente utilizada para administração local.

Configurar os Campos Necessários em uma Mensagem - Java

As mensagens de administração comunicam a ação de administração requerida por uma combinação de campos de dados armazenados na mensagem. Esses campos possuem nomes, tipos e valores bem definidos e você pode configurar a mensagem de administração utilizando a API de campos de nível inferior. Em Java, existem diversos métodos auxiliares para facilitar essa tarefa.

As seções seguintes descrevem os campos constituintes de mensagens admin e mensagens de resposta admin.

A Mensagem de Administração Básica

Todo pedido para administrar um recurso MQe assume a mesma forma básica. A tabela a seguir mostra a estrutura básica para todas as mensagens de pedido de administração:

Um pedido é constituído de:

1. Campos de administração base, que são comuns a todos os pedidos de administração.
2. Campos de administração, que são específicos para o recurso sendo gerenciado.
3. Campos opcionais para ajudar no processamento de mensagens de administração.

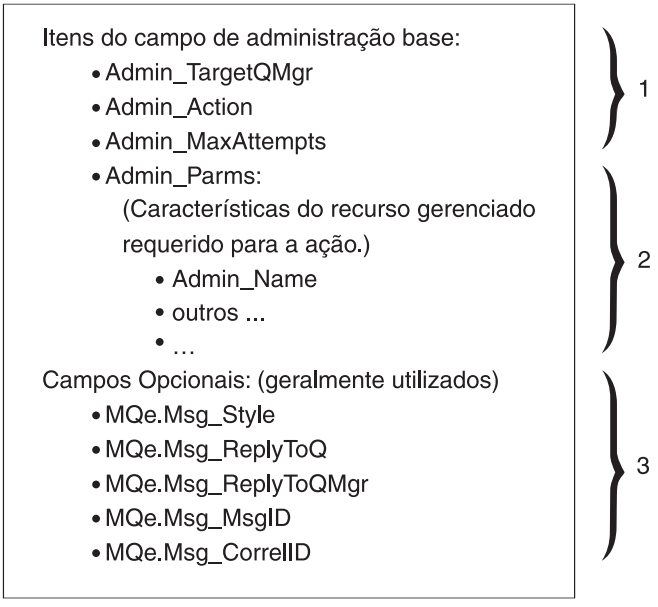


Figura 6. Mensagem de Pedido de Administração

Campos de Administração Básicos

Os campos de administração base, que são comuns a todas as mensagens de administração, são:

Admin_Target_QMgr

Esse campo fornece o nome do gerenciador de filas no qual a ação solicitada ocorrerá (gerenciador de filas de destino). O gerenciador de filas de destino pode ser um gerenciador de filas locais ou remotas. Como apenas um único gerenciador de filas pode estar ativo por vez em uma Java Virtual Machine, o gerenciador de filas de destino e aquele para o qual a mensagem é enviada são iguais.

Admin_Action

Esse campo contém a ação de administração que será desempenhada. Cada recurso gerenciado fornece um conjunto de ações administrativas que ele pode desempenhar. Uma única mensagem de administração pode solicitar apenas o desempenho de uma única ação. As seguintes ações comuns são definidas:

Tabela 5. Ações de Administração

Ação de Administração	**Propósito**
Action_Create	Criar uma nova instância de um recurso gerenciado.
Action_Delete	Excluir um recurso gerenciado existente
Action_Inquire	Consultar uma ou mais características de um recurso gerenciado
Action_InquireAll	Consultar todas as características de um recurso gerenciado
Action_Update	Atualizar uma ou mais características de um recurso gerenciado

Nem todos os recursos implementam necessariamente essas ações. Por exemplo, não é possível criar um gerenciador de filas utilizando uma mensagem de administração. Mensagens de administração específicas podem estender o conjunto base para fornecer ações adicionais específicas para um recurso.

Cada ação comum fornece um método que configura o campo *Admin_Action*:

Tabela 6. Configurando o Campo de Ação de Administração

Ação de Administração	**Método de Configuração**
Action_Create	create (MQeFields parms)
Action_Delete	delete (MQeFields parms)
Action_Inquire	inquire (MQeFields parms)
Action_InquireAll	inquireAll (MQeFields parms)
Action_Update	update(MQeFields parms)

Admin_MaxAttempts

Esse campo determina quantas vezes é possível tentar novamente uma ação se a ação inicial falhar. A nova tentativa ocorrerá no próximo reinício do gerenciador de filas ou no próximo intervalo configurado na fila de administração.

Outros Campos

Para a maioria das falhas, informações adicionais estão disponíveis na mensagem de resposta. É responsabilidade do aplicativo solicitante ler e manipular as informações de falha. Consulte “A Mensagem de Resposta de Administração Básica” na página 24 para obter detalhes adicionais sobre como utilizar os dados de resposta.

Um conjunto de métodos está disponível para configurar alguns dos campos de pedido:

Tabela 7. Configurando Campos do Pedido de Administração

Ação de Administração	Tipo de Campo	Métodos de Configuração e Recebimento
Admin_Parms	MQeFields	MQeFields getInputFields()
Admin_Action	int	setAction (int action)
Admin_TargetQMgr	ASCII	setTargetQMgr(String qmgr)
Admin_MaxAttempts	int	setMaxAttempts(int attempts)

Campos Específicos do Recurso Gerenciado

Admin_Parms

Esse campo contém as características do recurso que são requeridas para a ação.

Todo recurso possui um conjunto de características exclusivas. Cada característica possui um nome, tipo e valor e o nome de cada uma é definido por uma constante na mensagem de administração. O nome do recurso é uma característica comum a todos os recursos gerenciados. O nome do recurso está contido no Admin_Name e possui um tipo ASCII.

É possível determinar o conjunto completo de características de um recurso utilizando o método characteristics() com uma instância de uma mensagem de administração. Esse método retorna um objeto MQeFields que contém um campo para cada característica. Os métodos MQeFields podem ser utilizados para enumerar a respeito do conjunto de características para obter o nome, o tipo e o valor padrão de cada característica.

A ação solicitada determina o conjunto de características que podem ser transmitidas para a ação. Em todos os casos, pelo menos o nome do recurso, *Admin_Name*, deve ser transmitido. No caso de Action_InquireAll, esse é o único parâmetro requerido.

O seguinte código poderia ser utilizado para configurar o nome do recurso a ser gerenciado em uma mensagem de administração:

```
SetResourceName( MQeAdminMsg msg, String name )
{
  MQeFields parms;
  if ( msg.contains( Admin_Parms ) )
    parms = msg.getFields( Admin_Parms );
  else
    parms = new MQeFields();

  parms.putAscii( Admin_Name, name );
  msg.putFields( Admin_Parms, parms );
}
```

Alternativamente, o código pode ser simplificado utilizando o método getInputFields() para retornar o campo *Admin_Parms* da mensagem, ou setName() para configurar o campo *Admin_Name* na mensagem. Isso é mostrado no seguinte código:

```
SetResourceName( MQeAdminMsg msg, String name )
{
  msg.SetName( name );
}
```

Outros Campos Úteis

Por padrão, nenhuma resposta é gerada quando um pedido de administração é processado. Se uma resposta for requerida, a mensagem de pedido deverá ser configurada para solicitar uma mensagem de resposta. Os campos a seguir são definidos na classe MQe e são utilizados para solicitar uma resposta.

Msg_Style

Um campo do tipo int que pode ter um dos três valores:

Msg_Style_Datagram
Um comando que não solicita uma resposta

Msg_Style_Request
Um pedido que deseja uma resposta

Msg_Style_Reply
Uma resposta a um pedido

Se Msg_Style for configurado como Msg_Style_Request (uma resposta é requerida), o local para envio da resposta deverá ser configurado na mensagem de pedido. Os dois campos utilizados para configurar o local são:

Msg_ReplyToQ
Um campo ASCII utilizado para conter o nome da fila para a resposta

Msg_ReplyToQMgr
Um campo ASCII utilizado para conter o nome do gerenciador de filas para a resposta

Se o gerenciador de filas de resposta não for aquele que processa o pedido, então o gerenciador de filas que processa o pedido deverá ter uma conexão definida para o gerenciador de filas de resposta.

Para que uma mensagem de pedido de administração seja correlacionada com sua mensagem de resposta, a mensagem de pedido precisa conter campos que identifiquem exclusivamente o pedido e que possam ser, então, copiados na mensagem de resposta. O MQe fornece dois campos que podem ser utilizados para esse propósito:

Msg_MsgID
Uma matriz de byte contendo o ID de mensagem

Msg_CorrelID
Uma matriz de byte contendo o ID de Correlação da mensagem

Qualquer outro campo pode ser utilizado, mas esses dois têm a vantagem adicional de serem utilizados pelo gerenciador de filas para otimizar a procura de filas e a recuperação de mensagens. O fragmento de código a seguir fornece um exemplo de como preparar uma mensagem de pedido.

Exemplos de Mensagem de Administração em Java 1

Como este é um processo desempenhado com freqüência, este exemplo de código combina cada etapa no método primeAdminMsg(), que pode ser chamado em outras seções desta documentação (supondo que o método tenha sido definido para a classe em questão).

```
public class LocalQueueAdmin extends MQe
{
  public String   targetQMgr = "ExampleQM";
// gerenciador de filas de destino

public MQeFields primeAdminMsg(MQeAdminMsg msg) throws Exception
{
  /*
   * Configurar o gerenciador de filas de destino que processará essa mensagem
   */
  msg.setTargetQMgr( targetQMgr );

  /*
   * Solicitar o envio de uma mensagem de resposta para o
   * gerenciador de filas que processa o pedido admin
   */
  msg.putInt  (MQe.Msg_Style,       MQe.Msg_Style_Request);
  msg.putAscii(MQe.Msg_ReplyToQ,    MQe.Admin_Reply_Queue_Name);
  msg.putAscii(MQe.Msg_ReplyToQMgr, targetQMgr);

  /*
```

```
   * Configurar o ID de correlação para que seja possível corresponder a resposta com o pedido.
   * - Utilize um valor exclusivo para esse gerenciador de filas.
   */
  byte[] correlID =
    Long.toHexString( (MQe.uniqueValue()).getBytes() );
  msg.putArrayOfByte( MQe.Msg_CorrelID, correlID );

  /*
   * Assegurar que a mensagem de resposta correspondente seja recuperada
   * - configurar um objeto fields que pode ser utilizado como um parâmetro de correspondência
   *   ao procurar e recuperar mensagens.
   */
  MQeFields msgTest = new MQeFields();
  msgTest.putArrayOfByte( MQe.Msg_CorrelID, new Byte{1, 2, 3, 4} );

  /*
   * Retornar o filtro exclusivo para essa mensagem
   */
  return msgTest;
}
```

Dependendo de como a fila de administração de destino é definida, a entrega da mensagem pode ser síncrona ou assíncrona.

O próximo exemplo é utilizado para fazer uma 'consulta de todos' em um gerenciador de filas. Este método desempenha as etapas requeridas para endereçar a mensagem admin, solicitar uma resposta e incluir um marcador exclusivo na mensagem.

```
/* Este método desempenha o processamento padrão */
/* que prepara uma mensagem de administração para que */
/* seja possível manipulá-lo de um modo padrão */
/* Esse método configura o gerenciador de filas de destino */
/* (o gerenciador de filas sob o qual a ação de */
/* admin ocorre. */
/* Solicita o envio de uma mensagem de resposta para a */
/* fila de resposta admin no *gerenciador de filas de destino. */
/* Incorpora uma chave exclusiva na mensagem que */
/* pode ser utilizada para recuperar a resposta para essa mensagem.*/
/* A chave exclusiva é retornada como uma cadeia a ser */
/* utilizada pela rotina que extrai a resposta. */

public static final String decorateAdminMsg(MQeAdminMsg msg,
                           String targetQMName)throws Exception {
    //configurar o gerenciador de filas de destino
    msg.setTargetQMgr(targetQMName);
    //indicar que uma mensagem de resposta é necessária
    msg.putInt(MQe.Msg_Style,MQe.Msg_Style_Request);
    //utilizar fila de resposta padrão no gerenciador de filas de destino.
   msg.putAscii(MQe.Msg_ReplyToQ,    MQe.Admin_Reply_Queue_Name);
    msg.putAscii(MQe.Msg_ReplyToQMgr,targetQMName);
    //criar uma tag exclusiva que possa identificar a resposta com
    String match ="Msg"+System.currentTimeMillis();
    msg.putArrayOfByte(MQe.Msg_CorrelID,match.getBytes());
    return match;
}
```

Enviar a Mensagem para a Fila de Destino: A ação definida na mensagem admin será desempenhada apenas quando a mensagem chegar à fila admin no gerenciador de filas de destino. O gerenciador de filas de destino precisará ter uma fila admin.

Para receber a mensagem de um gerenciador de filas de destino remoto, você precisará ter toda a conectividade apropriada no lugar.

Se a administração for feita no gerenciador de filas locais, nenhuma conectividade será necessária. A entrega de mensagens é realizada por uma chamada simples de envio de mensagem. Simplesmente utilize a chamada `putMessage()` da API MQeQueueManager, especificando o gerenciador de filas de destino e o nome da fila admin padrão.

É possível ignorar o atributo e os parâmetros confirmados no exemplo, embora estejam disponíveis para acesso mais controlado à fila admin.

```
//enviar a mensagem para a fila admin correta
LocalQueueManager.putMessage(targetQueueManagerName, MQe.Admin_Queue_Name,
                             msg,null,0L);
```

Esperar por uma Mensagem de Resposta de Administração: Como a administração é desempenhada de maneira assíncrona, você precisará esperar a resposta à mensagem admin para determinar se a ação foi bem-sucedida. O processamento de mensagens MQe padrão é utilizado para esperar uma resposta ou a notificação de uma resposta. No código-base Java, por exemplo, a chamada `waitForMessage()` da API do gerenciador de filas pode ser utilizada para esse propósito.

Existe um retardo de tempo entre o envio do pedido e o recebimento da mensagem de resposta. O retardo de tempo poderá ser curto se o pedido estiver sendo processado localmente ou poderá ser longo se as mensagens de pedido e de resposta forem entregues de maneira assíncrona. O fragmento de código Java a seguir poderia ser utilizado para enviar uma mensagem de pedido e esperar por uma resposta:

```
public class LocalQueueAdmin extends MQe
{
  public String   targetQMgr = "ExampleQM";
  // gerenciador de fila de destino
  public int      waitFor    =   10000;
  // milissegundos de espera pela resposta

/*
 * Enviar uma mensagem admin concluída.
 * Utiliza o método simples putMessage, que não será assegurado se
 * a fila for definida para operação síncrona.
 */
public void sendRequest( MQeAdminMsg msg ) throws Exception
{
  myQM.putMessage( targetQMgr,
                   MQe.Admin_Queue_Name,
                   msg,
                   null,
                   0L );
}
/*
 * Esperar dez segundos por uma mensagem de resposta. Esse método aguardará durante
 * um tempo limitado em uma resposta local ou remota à fila.
 *
 *
 */
public MQeAdminMsg waitForReply(MQeFields msgTest) throws Exception {
  int secondsElapsed = 0;
  MQeAdminMsg msg = null;
  try {
    msg = (MQeAdminMsg)myQM.getMessage(
                   targetQMgr,
                   MQe.Admin_Reply_Queue_Name,
                   msgTest, null, 0L);
  } catch (MQeException e) {
    if (e.code() != MQe.Except_Q_NoMatchingMsg) {
      // se a exceção for 'sem mensagem
      // correspondente', ignorá-la.  Isso
      // resultará em um valor de retorno nulo.
      //Emita novamente todas as outras exceções
      throw e;
    }
```

```
  }
  while (null == msg && secondsElapsed < 10) {
    Thread.sleep(1000);
    secondsElapsed++;
    try {
      msg = (MQeAdminMsg)myQM.getMessage(
                  targetQMgr,
                    MQe.Admin_Reply_Queue_Name,
                    msgTest, null, 0L);
    } catch (MQeException e) {
      if (e.code() != MQe.Except_Q_NoMatchingMsg) {
        // se a exceção for 'sem mensagem correspondente', ignorá-la. Isso
        // resultará em um valor de retorno nulo. Emitir novamente toda as as outras exceções
        throw e;
      }
    }
  }
  return msg;
}
```

Este método é um wrapper simples para a chamada de API `waitForMessage()` do `MQeQueueManager`, que configura um filtro para selecionar a resposta admin requerida e difunde qualquer mensagem obtida para uma mensagem admin.

```
/**
*Aguardar mensagem - aguarda a chegada de uma mensagem na fila de resposta admin
*do gerenciador de filas de destino especificado. Aguardará apenas mensagens com a
*mensagem de retorno de tag exclusiva especificada ou retornará nulo se o tempo
limite estiver esgotado  */

public static final MQeAdminMsg waitForRemoteAdminReply(
                                     MQeQueueManager localQueueManager,
                                     String remoteQueueManagerName,
                                     String match)throws Exception {
   //construir um filtro para assegurar que apenas a resposta correspondente seja recebida
    MQeFields filter =new MQeFields();
    filter.putArrayOfByte(MQe.Msg_CorrelID,match.getBytes());
    //aguardar agora a mensagem de resposta
    MQeMsgObject reply =localQueueManager.waitForMessage(
                                            remoteQueueManagerName,
                                            MQe.Admin_Reply_Queue_Name,
                                            filter,
                                            null,
                                            0L,
                                            10000);//aguardar 10 segundos
    return (MQeAdminMsg)reply;
}
```

Configurar os Campos Necessários em uma Mensagem - C

Esta seção aplica-se apenas ao código-base C.

Como a administração é desempenhada de maneira assíncrona, você precisa esperar a resposta à mensagem de administração para determinar se a ação foi bem-sucedida. Portanto, é necessário solicitar uma resposta (o padrão é não enviar resposta) e especificar para onde enviar a mensagem de resposta. O destino para a mensagem de resposta deve ser uma fila local conveniente.

Lembre-se de que o código de administração precisa enviar a mensagem de resposta para o destino especificado e, portanto, pode precisar de definições de conexão e listeners configurados. É mais fácil receber a mensagem de resposta de administração enviada para a fila de resposta de administração na máquina em que a administração é desempenhada. A conectividade utilizada para entregar a mensagem de administração para o gerenciador de filas de destino pode ser utilizada, por conseguinte, para recuperar a mensagem de resposta de administração do gerenciador de filas de destino. Essa é a técnica utilizada nos exemplos seguintes.

Uma outra tarefa útil que pode ser desempenhada neste estágio é incluir um campo de identificação na mensagem de pedido de administração, para que você possa identificar facilmente a resposta correspondente. Isso pode ser feito incluindo um campo de matriz de byte denominado `MQe.Msg_CorrelID` na mensagem. O código de administração assegura que esse campo seja copiado na mensagem de resposta. Se você desejar, poderá utilizar isso para correlacionar a ação de administração com a resposta de administração.

Analisando os Dados na Mensagem de Resposta

As mensagens de resposta de administração contêm informações sobre o sucesso ou falha da tentativa de desempenhar o pedido de administração. Existem três níveis de sucesso:

1. **Sucesso total** - a ação ocorreu conforme solicitada. Para pedidos de consulta, as mensagens contêm os dados solicitados.
2. **Falha total** - a ação falhou. A mensagem contém um motivo pelo qual a ação falhou.
3. **Falha parcial** - alguma parte de um pedido composto falhou. Por exemplo, uma tentativa de atualizar cinco campos poderá ser bem-sucedida para três, mas malsucedida para dois. Os campos que falharam e o motivo da falha estão contidos na mensagem.

Sucesso total

Se a ação de administração for bem-sucedida, a mensagem de retorno conterá um campo de byte chamado `MQeAdminMsg.Admin_RC` com um valor igual a `MQeAdminMsg.RC_Success`.

Falha total

Se a ação de administração falhar completamente, a mensagem de retorno conterá um campo de byte chamado MQeAdminMsg.Admin_RC com um valor igual a MQeAdminMsg.RC_Fail. Também contém um campo de Cadeia chamado `MQeAdminMsg#Admin_Reason`, que possui uma descrição da falha.

Falha parcial

Se a ação de administração falhar parcialmente, a mensagem de retorno conterá um campo de byte chamado `MQeAdminMsg.Admin_RC` com um valor igual a `MQeAdminMsg.RC_Mixed`. O campo de Cadeia chamado `MQeAdminMsg.Admin_Reason` contém apenas uma explicação geral 'ocorreram erros'. Para obter detalhes adicionais, acesse o campo chamado `MQeAdminMsg.Admin_Errors`. O objeto MQeFields contém quaisquer erros relacionados a subproblemas que ocorrem quando um pedido falha com um código de retorno `RC_Fail ou RC_Mixed`. Para cada atributo em erro, existe um campo correspondente nesse objeto MQeFields. Se o campo processado era uma matriz, o campo de erro correspondente será do tipo matriz ASCII. Se o campo processado não era uma matriz, o campo de erro correspondente será do tipo ASCII.

Por exemplo, se fosse feito um pedido de atualização para alterar 4 atributos de um recurso e 2 das atualizações fossem bem-sucedidas e 2 falhassem, esse campo conteria informações detalhando o motivo das 2 falhas.

Cada erro é geralmente uma representação `toString()` da exceção que causou a falha. Se a exceção for do tipo `com.ibm.mqe.MQeException`, a cadeia incluirá o código `MQeException` no início da cadeia como ″Code=nnn″.

A Mensagem de Resposta de Administração Básica

Assim que um pedido de administração é processado, uma resposta, se solicitada, é enviada para a fila do gerenciador de filas de resposta. A mensagem de resposta possui o mesmo formato básico que a mensagem de pedido, com alguns campos adicionais.

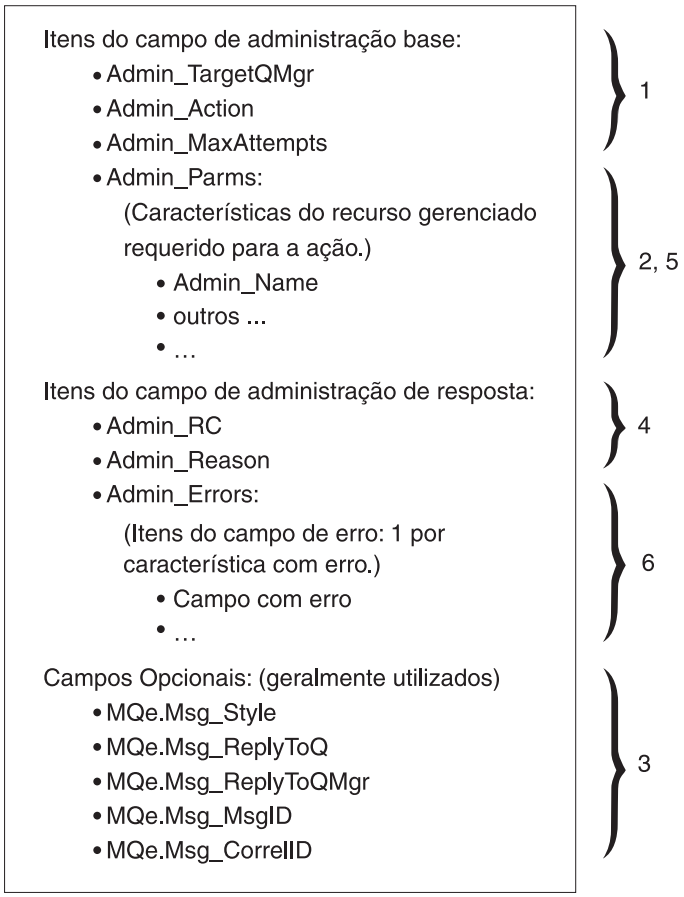


Figura 7. Mensagem de Resposta de Administração

Uma resposta é constituída de:

1. Campos de administração base. Eles são copiados da mensagem de pedido.
2. Campos de administração que são específicos para o recurso sendo gerenciado.
3. Campos opcionais para ajudar no processamento de mensagens de administração. Eles são copiados da mensagem de pedido.
4. Campos de administração que detalham o resultado do pedido.
5. Campos de administração que fornecem resultados detalhados do pedido, os quais são específicos do recurso sendo gerenciado.
6. Campos de administração detalhando erros que são específicos para o recurso sendo gerenciado.

Os três primeiros itens são descritos em "A Mensagem de Administração Básica" na página 17. Os campos específicos de resposta são descritos nas seções seguintes.

Resultado de Campos do Pedido

Campo Admin_RC

Esse campo de byte contém o resultado geral do pedido. Consiste em um campo do tipo `int` que é configurado para um destes:

MQeAdminMsg.RC_Success
A ação foi concluída com êxito.

MQeAdminMsg.RC_Failed
O pedido falhou completamente.

MQeAdminMsg.RC_Mixed
O pedido foi parcialmente bem-sucedido. O resultado poderá ser um código de retorno misto se for feito um pedido para atualizar quatro atributos de uma fila e três obtiverem êxito e um falhar.

Admin_Reason
Um campo Unicode contendo o motivo geral para a falha nos casos de Mixed e Failed.

Admin_Parms
Um objeto MQeFields que contém um campo para cada característica do recurso gerenciado.

Admin_Errors
Um objeto MQeFields que contém um campo para cada atualização com falha. Cada entrada contida no campo *Admin_Errors* é do tipo ASCII ou asciiArray.

Os seguintes métodos estão disponíveis para obter alguns dos campos de resposta:

Tabela 8. Obtendo Campos de Resposta de Administração

Campo de Administração	Tipo de Campo	Método de Recebimento
Admin_RC	int	int getAction()
Admin_Reason	Unicode	String getReason()
Admin_Parms	MQeFields	MQeFields getOutputFields()
Admin_Errors	MQeFields	MQeFields getErrorFields()

Dependendo da ação desempenhada, os únicos campos de interesse podem ser o código de retorno e a razão. Este é o caso de delete. Para outras ações, como inquire, detalhes adicionais podem ser requeridos na mensagem de resposta. Por exemplo, se for feito um pedido inquire para os campos *Queue_Description* e *Queue_FileDesc*, o objeto MQeFields resultante conterá os valores da fila real nesses dois campos.

A tabela a seguir mostra os campos *Admin_Parms* de uma mensagem de pedido e uma mensagem de resposta para uma consulta em vários parâmetros de uma fila:

Tabela 9. Consultando Parâmetros de Fila

Nome do Campo Admin_Parms	Mensagem de Pedido		Mensagem de Resposta	
	Tipo	Valor	Tipo	Valor
Admin_Name	ASCII	″TestQ″	ASCII	″TestQ″
Queue_QMgrName	ASCII	″ExampleQM″	ASCII	″ExampleQM″
Queue_Description	Unicode	nulo	Unicode	″Uma fila de teste″
Queue_FileDesc	ASCII	nulo	ASCII	″c:\queues\″

Para ações em que não são esperados dados adicionais na resposta, o campo *Admin_Parms* na resposta corresponde àquele da mensagem de pedido. Esse é o caso das ações create e update.

Algumas ações, como create e update, podem solicitar que várias características de um recurso gerenciado sejam configuradas ou atualizadas. Nesse caso, é possível que um código de retorno igual a RC_Mixed seja recebido. Detalhes adicionais indicando o motivo da falha de cada atualização estão disponíveis no campo *Admin_Errors*. A tabela a seguir mostra um exemplo do campo *Admin_Parms* de um pedido para atualizar uma fila e o campo *Admin_Errors* resultante:

Tabela 10. Mensagem de Pedido e de Resposta para Atualizar uma Fila

Nome do Campo	Mensagem de Pedido		Mensagem de Resposta	
	Tipo	**Valor**	**Tipo**	**Valor**
Campo Admin_Parms				
Admin_Name	ASCII	″TestQ″	ASCII	″TestQ″
Queue_QMgrName	ASCII	″ExampleQM″	ASCII	″ExampleQM″
Queue_Description	Unicode	nulo	Unicode	″ExampleQM″ ″Uma nova descrição″
Queue_FileDesc	ASCII	nulo	Unicode	″D:\queues″
Campo Admin_Errors				
Queue_FileDesc	n/d	n/d	ASCII	″Código=4;com.ibm.mqe.MQeException: tipo de campo incorreto″

Para campos em que a atualização ou configuração é bem-sucedida, não há nenhuma entrada no campo *Admin_Errors*.

Uma descrição detalhada de cada erro é retornada em uma cadeia ASCII. O valor da cadeia de erro é a exceção que ocorreu durante a tentativa de configuração ou atualização. Se a exceção foi uma MQeException, o código de exceção real será retornado juntamente com a representação *toString* da exceção. Portanto, para uma MQeException, o formato do valor é:

```
"Código=nnnn;representação toString da exceção"
```

Exemplos de Mensagem de Administração em Java - 2

Este método mostra como é possível analisar uma mensagem de resposta e retornar um booleano para indicar se a ação foi bem-sucedida ou não. As mensagens de erro são impressas no console.

```
/**
*Responder true se a mensagem de resposta admin
*fornecida representar uma ação admin
*bem-sucedida. Caso contrário, retornar false.
*Uma mensagem indicando sucesso
*ou falha será impressa no console.
*Se a ação admin não foi bem-sucedida, o motivo será impresso
*no console
*/
public static final boolean isSuccess(MQeAdminMsg reply)
          throws Exception {
    boolean success =false;
    final int returnCode =reply.getRC();
    switch (returnCode){
      case MQeAdminMsg.RC_Success:
        System.out.println("Administração bem-sucedida");
        success =true;
        break;
      case MQeAdminMsg.RC_Fail:
    /* tudo em uma única linha */
       System.out.println("Falha na administração, motivo:"+
                reply.getReason());
        break;
      case MQeAdminMsg.RC_Mixed:
        System.out.println("Administração parcialmente bem-sucedida:\n"
                +reply.getErrorFields());
```

```
        break;
    }
    return success;
}
```

Decorando o Gerenciador de Filas

Este método é implementado na classe examples.config.BasicAdministration. Endereça a mensagem de administração, solicita uma resposta e inclui um marcador exclusivo na mensagem.

```
    /**
       * Este método desempenha o processamento padrão que
       * decora uma mensagem de administração
       * para que seja possível manipulá-la de um modo padrão.
       * <p>Este método:
       * <p> Configura o gerenciador de filas de destino
       * (o gerenciador de filas sob o qual
       * a ação de administração ocorre).
       * <p> Solicita o envio de uma mensagem de resposta
    * para a fila de resposta de administração no
       * gerenciador de filas de destino.
       * <p> Incorpora uma chave exclusiva na mensagem
    * que pode ser utilizada para recuperar
       * a resposta para essa mensagem.
    * A chave exclusiva é retornada como uma cadeia para ser
      * utilizada pela rotina que extrai a resposta.
     */
    public static final String decorateAdminMsg(MQeAdminMsg msg,
                               String targetQMName)throws Exception {

        // configurar o gerenciador de filas de destino
        msg.setTargetQMgr(targetQMName);

        // esperar a mensagem de resposta
        msg.putInt(MQe.Msg_Style, MQe.Msg_Style_Request);

       //utilizar fila de resposta padrão no gerenciador de filas de destino.
        msg.putAscii(MQe.Msg_ReplyToQ, MQe.administration_Reply_Queue_Name);
        msg.putAscii(MQe.Msg_ReplyToQMgr, targetQMName);

       //criar uma tag exclusiva que possa identificar a resposta com
        String match ="Msg"+System.currentTimeMillis();
        msg.putArrayOfByte(MQe.Msg_CorrelID,match.getBytes());

        return match;
    }
```

Enviando a Mensagem de Administração

Utilize a chamada `putMessage()` da API MQeQueueManager, especificando o gerenciador de filas de destino e o nome da fila de administração padrão. É possível ignorar o atributo e os parâmetros confirmados no exemplo, embora estejam disponíveis para acesso mais controlado à fila de administração.

```
        // enviar a mensagem para a fila de administração correta
        localQueueManager.putMessage(targetQueueManagerName,
                                     MQe.Admin_Queue_Name,
                                     msg, null, 0L);
```

Aguardando a Resposta da Administração

Este método é implementado na classe `examples.config.BasicAdministration`. Consiste em um wrapper simples para a chamada `waitForMessage()` da API MQeQueueManager, que configura um filtro para selecionar a resposta de administração requerida e difunde qualquer mensagem obtida para uma mensagem de administração.

```
/**
   * Aguardar mensagem - aguarda a chegada de uma mensagem
* na fila de resposta de administração
   * do gerenciador de filas de destino especificado.
* Aguardará apenas as mensagens com a
   * mensagem de retorno de tag exclusiva
   * especificada ou nula, se o tempo limite estiver esgotado
   */
public static final MQeAdminMsg waitForRemoteAdminReply(
                MQeQueueManager localQueueManager,
                String remoteQueueManagerName,
                String match) throws Exception {
    // construir um filtro para assegurar que apenas a resposta correspondente seja recebida
    MQeFields filter = new MQeFields();
    filter.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());

    // esperar a mensagem de resposta
    MQeMsgObject reply = localQueueManager.waitForMessage(
        remoteQueueManagerName,
        MQe.Admin_Reply_Queue_Name,
        filter,
        null,
        0L,
        10000);          // aguardar 10 segundos
    return (MQeAdminMsg)reply;
}
```

Analisando a Mensagem de Resposta

Este método é implementado na classe `examples.config.BasicAdministration`. Ele mostra como é possível analisar uma mensagem de resposta e retornar uma resposta que indica se a ação foi bem-sucedida ou não. Qualquer mensagem de erro é impressa no console.

```
/**
   *  Responder true se a mensagem de resposta de
 * administração fornecida representar uma ação de
   * administração bem-sucedida.  Caso contrário, retornar false.
 * Uma mensagem indicando êxito
   * ou falha será impressa no console.
   * Se a ação de administração não foi bem-sucedida,
 * o motivo será impresso
   * no console
 */
public static final boolean isSuccess(MQeAdminMsg reply)
                            throws Exception {
    boolean success = false;
    final int returnCode = reply.getRC();
    switch (returnCode) {
        case MQeAdminMsg.RC_Success:
            System.out.println("Administração bem-sucedida");
            success = true;
            break;
        case MQeAdminMsg.RC_Fail:
            System.out.println("Falha na administração, motivo:
                        "+ reply.getReason());
            break;
        case MQeAdminMsg.RC_Mixed:
            System.out.println("Administração parcialmente bem-sucedida:\n"
                        +reply.getErrorFields());
            break;
    }
    return success;
}
```

Atualizando uma Descrição do Gerenciador de Filas

Este método é implementado na classe `examples.config.QueueManagerAdmin`. Ele mostra como utilizar as primitivas na classe BasicAdministration para atualizar uma descrição do gerenciador de filas e para relatar o sucesso da ação.

```
/**
   * Atualizar o campo de descrição do
* gerenciador de filas especificado para a cadeia
  * especificada.  Utilize a queueManager
* fornecida como o acesso à
   * rede MQe.
   *
   * @param queueManager (MQeQueueManager): access point to the MQe network
   * @param queueManagerName (String): name of queue manager to modify
   * @param (String): new description for queue manager
   */
public static final boolean updateQueueManagerDescription(
                                     MQeQueueManager queueManager,
                                     String targetQueueManagerName,
                                     String description)
                                             throws Exception {
     // criar mensagem de administração
    MQeQueueManagerAdminMsg msg = new MQeQueueManagerAdminMsg();

     // solicitar uma atualização
    msg.setAction(MQeAdminMsg.Action_Update);

      // configurar o novo valor do parâmetro
  //nos campos de entrada na mensagem
      // o nome do campo é o nome do atributo
  // e o valor do campo é o novo
      // valor do atributo.  O tipo é especificado
  // pela mensagem de administração.
      // Neste caso, o nome do campo é 'descrição',
  // o valor é a nova
      // descrição e o tipo é Unicode.
    msg.getInputFields().putAscii(
                        MQeQueueManagerAdminMsg.QMgr_Description,
                        description);

     // configurar para resposta etc
    String uniqueTag = BasicAdministration.decorateAdminMsg(
                              msg, targetQueueManagerName);

    // enviar a mensagem para a fila de administração correta
    queueManager.putMessage(targetQueueManagerName,
                            MQe.Admin_Queue_Name,
                            msg, null, 0L);

    // esperar a mensagem de resposta
    MQeAdminMsg reply = BasicAdministration.waitForRemoteAdminReply(
                          queueManager,
                          targetQueueManagerName,
                          uniqueTag);

    return BasicAdministration.isSuccess(reply);
}
```

Configurando com a API do Administrador C

Para criar e administrar Gerenciadores de Filas e seus objetos associados (filas, etc.), a API Java utiliza a classe MQeQueueManagerConfigure e as mensagens admin. Na API C, as atividades de administração são desempenhadas utilizando uma API de Administrador. O código-base nativo responde corretamente às mensagens admin, mas nenhuma provisão é fornecida para criá-las. Portanto, a API de Administrador é o método recomendado para a administração local.

Para a documentação completa sobre a API de Administrador e todas as opções disponíveis, consulte o Referência de Programação C.

Criando um Identificador de Administrador

Antes que qualquer administração possa ocorrer, um identificador de administrador deve ser criado utilizando a chamada de API `mqeAdministrator_new`. O protótipo para a chamada é:

```
MQERETURN mqeAdministrator_new(MQeExceptBlock*   pExceptBlock,
                               MQeAdministratorHndl* phAdmin,
                               MQeQueueManagerHndl   hQueueMgr)
```

O primeiro parâmetro é um ponteiro para um bloco de exceções válidas. O segundo parâmetro é um ponteiro para um identificador de administrador, que é preenchido com um identificador válido após o retorno bem-sucedido da função. O terceiro parâmetro é um identificador de gerenciador de filas opcional. Se o gerenciador de filas a ser administrado já existir, o identificador deverá ser criado por meio da função `mqeQueueManager_new`, e o identificador de gerenciador de filas retornado deverá ser transmitido à chamada `mqeAdministrator_new`.

Para criar um gerenciador de filas, `NULL` deve ser transmitido como o terceiro parâmetro para a chamada `mqeAdministrator_new`. Se `NULL` for utilizado, transmita a chamada `mqeAdministrator_free` ou `mqeAdministrator_QueueManager_create`. Após a execução da chamada `mqeAdministrator_QueueManager_create`, o identificador de administrador pode ser utilizado normalmente.

Utilizando o Identificador de Administrador

Após a criação de um Identificador de Administrador, qualquer uma das chamadas `mqeAdministrator` poderá ser utilizada. As chamadas estão no seguinte formato:

```
MQERETURN mqeAdministrator_Object_action(
                            MQeAdministratorHndl hAdministrator,
                            MQeExceptBlock*   pExceptBlock,
                            ...)
```

Em que:

- `object` é o tipo de objeto a ser administrado; por exemplo, um gerenciador de filas, uma fila local ou uma fila remota síncrona
- `action` é a operação a ser desempenhada; por exemplo, criar, excluir, consultar ou atualizar.

Nota: Algumas ações estão disponíveis apenas para alguns tipos de objetos.

Exemplos de chamadas:

Se `NULL` for utilizado para criar um `MQeAdministratirHndl`, a próxima chamada de API de administração poderá ser apenas MQeAdministrator_free ou MQeAdministrator_create_QueueManager. Após a criação do gerenciador de filas, todas as APIs de administração estão disponíveis para serem utilizadas.

```
 mqeAdministrator_LocalQueue_create
/* criar uma fila local */

 mqeAdministrator_AdminQueue_inquire
/* consultar uma fila local */
```

Muitas APIs, principalmente as chamadas de consulta e atualização, possuem argumentos que são estruturas que contêm vários elementos, alguns dos quais podem estar preenchidos. Para acomodar essa funcionalidade, essas estruturas contêm um elemento chamado "opFlags", um conjunto de bits para indicar quais elementos da estrutura estão configurados. Também são fornecidas macros que inicializam essas estruturas opFlag com valores apropriados e macros para cada bit que pode ser configurado.

Por exemplo, se você desejar consultar uma fila local, mas estiver interessado apenas nos campos de descrição e de Tamanho Máximo da Mensagem, execute o seguinte:

```
MQeLocalQParms lqParms = LOCAL_Q_INIT_VAL;
lqParms.opFlags |= QUEUE_DESC_OP;
lqParms.opFlags |= QUEUE_MAX_MSG_SIZE_OP;
/* Observe que a função | está sendo utilizada */

/* chamar função de consulta */
```

De modo semelhante, se você desejar testar quais elementos são preenchidos quando essa estrutura é retornada de uma função, faça o seguinte:

```
if(lqParms.opFlags & QUEUE_DESC_OP)
{ /* a descrição está configurada*/
}
if(lqParms.opFlags & QUEUE_MAX_MSG_SIZE_OP)
{ /* o tamanho máximo da msg está configurado*/
}
```

Liberando o Identificador de Administrador

Depois que o aplicativo conclui o identificador de administrador, este deve ser destruído utilizando a chamada `mqeAdministrator_free`. Isso permite que o sistema libere quaisquer recursos em uso pelo administrador. Depois que um identificador de administrador é liberado, ele não deve ser utilizado em nenhuma chamada de API `mqeAdministrator_*` - se o identificador for utilizado, o comportamento será indeterminado, mas provavelmente causará uma violação de acesso. Se for necessário desempenhar ações de administração adicionais, o identificador poderá ser recriado com a chamada `mqeAdministrator_new`.

```
rc = mqeAdministrator_new(&exceptBlock,
                          &hAdministrator,
                          NULL);
if(MQERETURN_OK == rc)
{  /* mqeAdministrator_QueueManager_create  */
   /* chamadas mqeAdministrator adicionais */
   /* ... */
   rc = mqeAdministrator_free(hAdministrator,
                              &exceptBlock);
} hAdministrator = NULL;
```

Figura 8. Criando um Identificador de Administrador para um Novo Gerenciador de Filas

Após a liberação de um identificador, configure-o como `NULL`. Se depois esse identificador for reutilizado por engano, a API retornará um erro.

```
/* mqeQueueManager_new(...,&hQueueManager,...) */
/* ... */
rc = mqeAdministrator_new(&exceptBlock,
                          &hAdministrator,
                          hQueueManager);
if(MQERETURN_OK == rc)
{
    /* chamadas mqeAdministrator adicionais */
   /* ... */
   rc = mqeAdministrator_free(hAdministrator,
                              &exceptBlock);
}
```

Figura 9. Criando um Identificador de Administrador para um Gerenciador de Filas Existente

Tabela 11. Códigos de Razão e de Retorno Comuns

Códigos de Retorno	Códigos de Razão	Notas
MQERETURN_ADMINISTRATION_ERROR	MQEREASON_INVALID_QMGR_NAME	O nome possui um caractere inválido ou é NULL
	MQEREASON_INVALID_QUEUE_NAME	O nome possui um caractere inválido ou é NULL
MQERETURN_INVALID_ARGUMENT	MQEREASON_API_NULL_POINTER	O ponteiro é NULL
	MQEREASON_WRONG_TYPE	Um identificador de tipo incorreto foi transmitido; por exemplo, QueueManager hndl em vez de MQeFields
MQERETURN_QUEUE_ERROR	MQEREASON_QMGR_QUEUE_EXISTS	A fila já existe
	MQEREASON_QMGR_QUEUE_NOT_EMPTY	A fila não está vazia
MQERETURN_QUEUE_MANAGER_ERROR	MQEREASON_UNKOWN_QUEUE	A fila não existe
	MQEREASON_UNKOWN_QUEUE_MANAGER	O gerenciador de filas não existe
MQERETURN_NOTHING_TO_DO	MQEREASON_DUPLICATE	Nome já em uso
	MQEREASON_NO_SUCH_QUEUE_ALIAS	O alias da fila especificado não existe

Configurando a partir da Linha de Comandos

O MQe inclui algumas ferramentas que permitem a administração de objetos MQe a partir da linha de comandos, utilizando scripts simples. As seguintes ferramentas são fornecidas:

QueueManagerUpdater
Cria um gerenciador de filas de dispositivo a partir de um arquivo ini e envia uma mensagem de administração para atualizar as características de um gerenciador de filas.

IniFileCreator
Cria um arquivo ini com o conteúdo necessário para um gerenciador de filas de cliente.

LocalQueueCreator
Abre um gerenciador de filas de cliente, inclui uma definição de fila local e fecha o gerenciador de filas.

HomeServerCreator
Abre um gerenciador de filas de servidor, inclui uma fila de servidor home e fecha o gerenciador de filas.

ConnectionCreator
Permite que uma conexão seja incluída em um gerenciador de filas MQe sem programar nada em Java.

RemoteQueueCreator
Abre um gerenciador de filas de dispositivo para ser utilizado, envia a ele uma mensagem de administração para que uma definição de fila remota seja criada, em seguida, fecha o gerenciador de filas.

MQBridgeCreator
Cria uma ponte MQ em um gerenciador de filas MQe.

MQQMgrProxyCreator
Cria um proxy do gerenciador de filas MQ para uma ponte.

MQConnectionCreator
Cria uma definição de conexão para um sistema MQ em um objeto de proxy.

MQListenerCreator
Cria um listener de fila de transmissão MQ para receber mensagens do MQ.

MQBridgeQueueCreator
Cria uma fila MQe que pode fazer referência a mensagens em uma fila MQ.

StoreAndForwardQueueCreator
Cria uma fila de armazenamento e redirecionamento.

StoreAndForwardQueueQMgrAdder
Inclui um nome de gerenciador de filas na lista de gerenciadores de filas para os quais a fila de armazenamento e redirecionamento aceita mensagens.

Os seguintes arquivos também são fornecidos:

Arquivos de Script de Exemplo
Dois arquivos .bat de exemplo e um script runmqsc para demonstrar como definir uma configuração de rede fictícia, envolvendo uma ramificação, um gateway e um sistema MQ.

Exemplo de Java Agregado
Um exemplo de como um arquivo em batch pode ser agregado a um arquivo Java para independência de linguagem do batch.

Exemplo de Uso das Ferramentas de Linha de Comandos

Você pode utilizar as ferramentas de linha de comandos para criar uma configuração inicial do gerenciador de filas utilizando um script, sem precisar saber como programar em Java.

O exemplo a seguir demonstra como utilizar essas ferramentas para configurar a topologia de rede mostrada na figura a seguir:

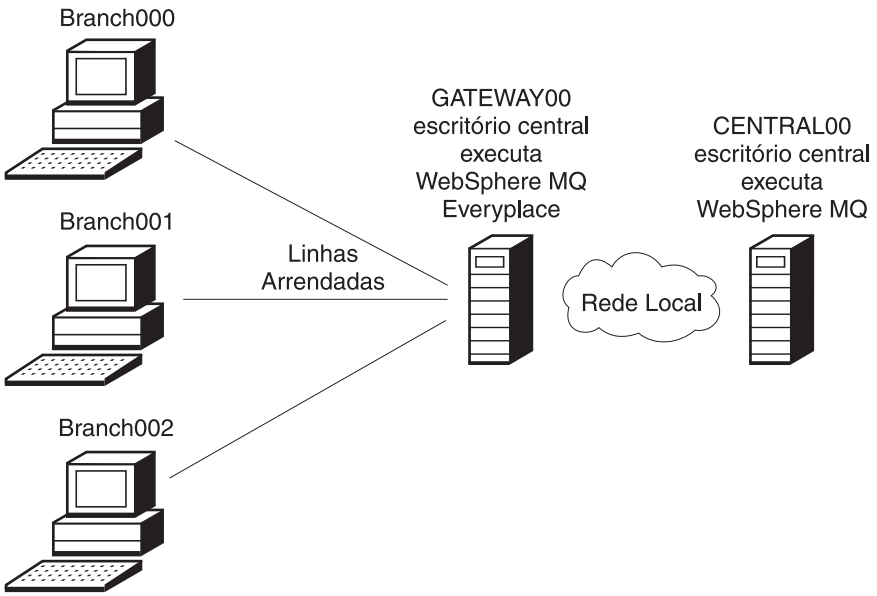


Figura 10. Cenário de Administração do MQe

Nesse cenário:

- Os escritórios de ramificação precisam enviar informações de vendas ao site central para serem processadas por aplicativos no servidor MQ
- Cada ramificação possui uma única máquina com nomes DNS `BRANCH000`, `BRANCH001` e `BRANCH002`, respectivamente. Todas essas máquinas executam o MQe e cada uma delas possui um único gerenciador de filas denominado `BRANCH000QM`, `BRANCH001QM` e `BRANCH002QM`, respectivamente.
- A máquina do escritório central `GATEWAY00` executa um único gerenciador de filas de gateway `GATEWAY00QM`
- A máquina do escritório central `CENTRAL00` executa o MQ com um único gerenciador de filas denominado `CENTRAL00QM`
- Quando ocorre uma venda, uma mensagem é enviada ao gerenciador de filas do MQ denominado `CENTRAL00QM`, para uma fila denominada `BRANCH.SALES.QUEUE`.
- As mensagens são codificadas em uma matriz de byte na ramificação e enviadas dentro de um MQeMQMsgObject.
- O sistema MQ deve estar apto a retornar as mensagens para cada gerenciador de filas de ramificação.
- A topologia também precisa estar apta a lidar com a inclusão de um Firewall posteriormente entre as ramificações e o gateway.
- O tráfego de filas de limite do MQ deve utilizar o criptografador DES de 56 bits.

Arquivos de Script Requeridos

Os seguintes scripts são necessários para configurar esta topologia de rede:

Central.tst
: Utilizado com o script runmqsc para criar objetos relevantes no `CENTRAL00QM`

CentralQMDetails.bat
: Utilizado para descrever o `CENTRAL00QM` para outros scripts

GatewayQMDetails.bat
: Utilizado para descrever o `GATEWAY00QM` para outros scripts

CreateGatewayQM.bat
: Utilizado para criar o gerenciador de filas de gateway

CreateBranchQM.bat
Utilizado para criar um gerenciador de filas de ramificação

Todos esses arquivos .bat podem ser localizados no produto instalado, em MQe\Java\Demo\Windows.

Nota: Embora os scritps de exemplo fornecidos estejam no formato de arquivo .bat do Windows, eles podem ser convertidos e funcionar bem em qualquer linguagem de script disponível no sistema.

Objetos MQe e MQ Definidos pelos Scripts

Os seguintes objetos são criados pelos scripts para estabelecer a rota ramificação para central:

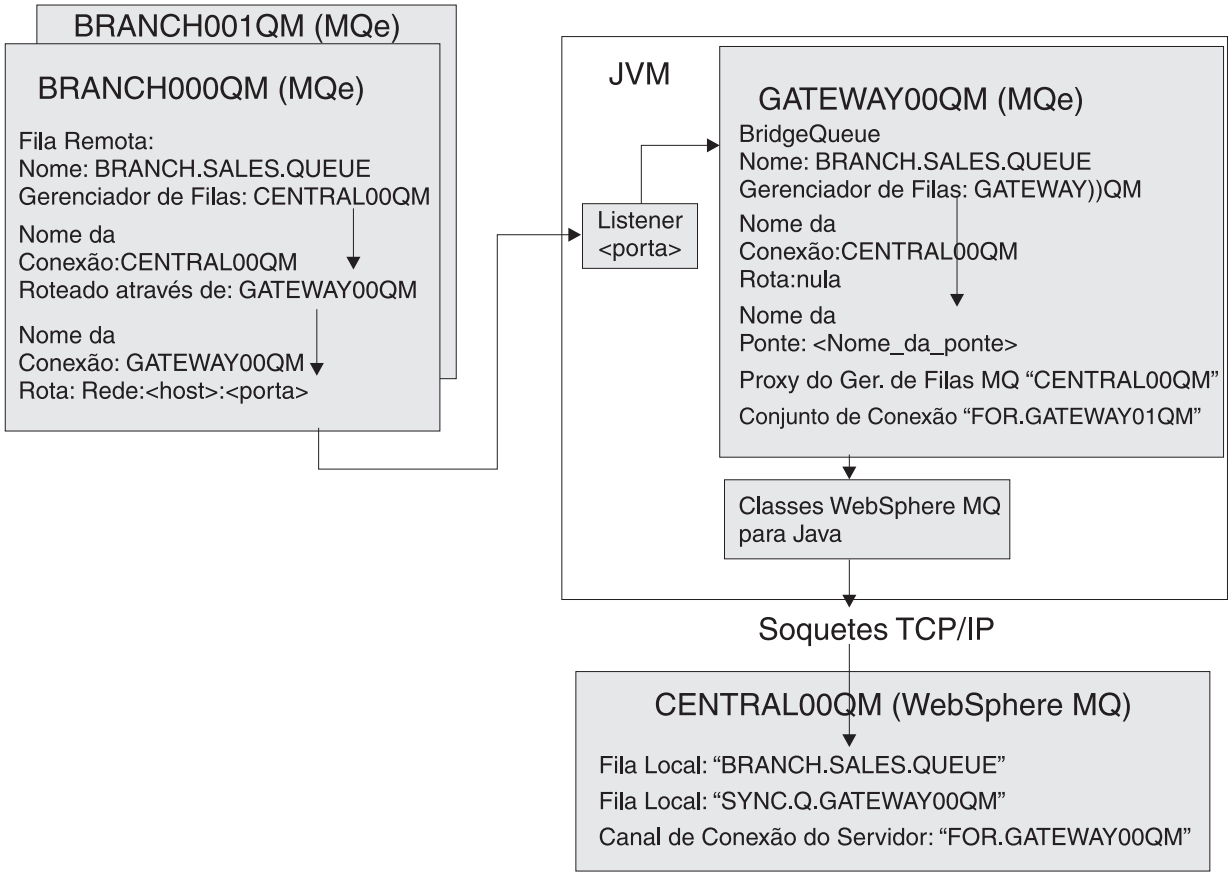


Figura 11. Rota Ramificação para Central

Os seguintes objetos são criados pelos scripts para estabelecer a rota de central para ramificação:

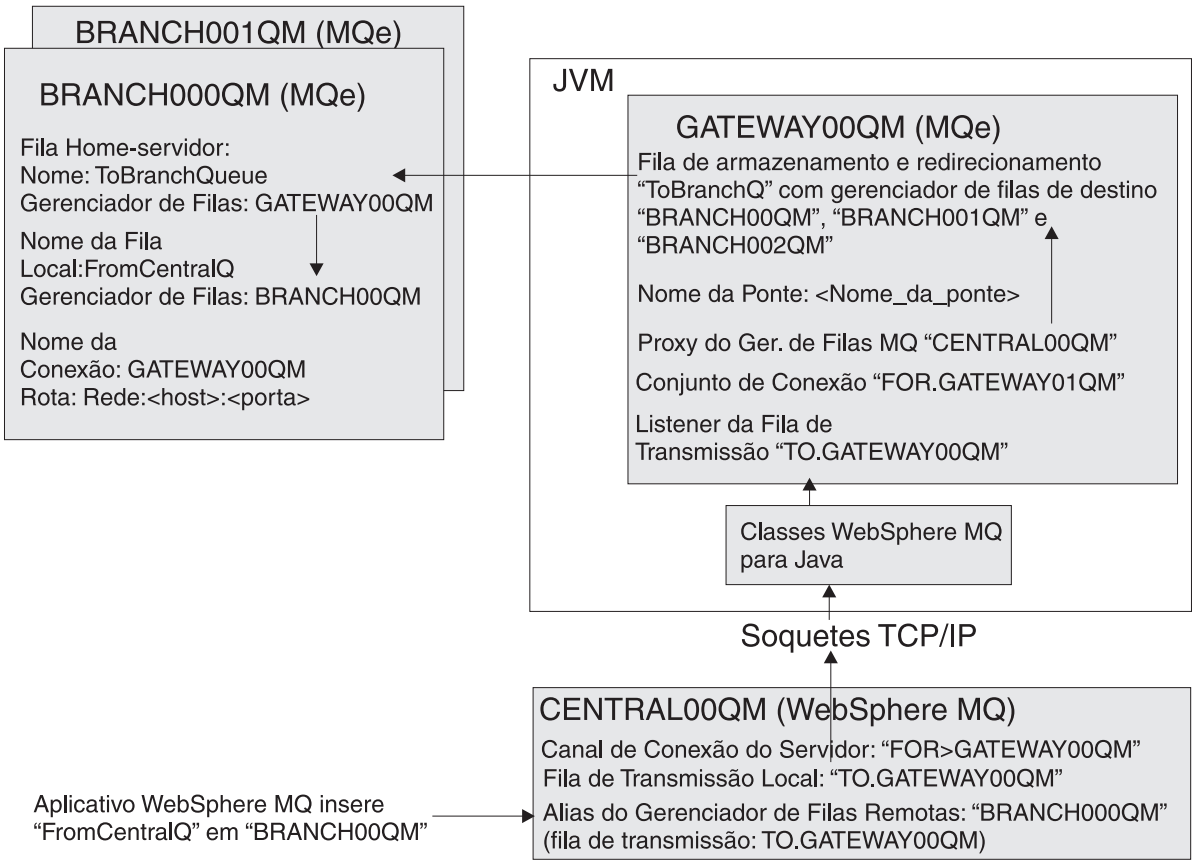


Figura 12. Rota Central para Ramificação

Como Utilizar os Arquivos de Script

Siga estes procedimentos para criar os objetos requeridos e operar o cenário de exemplo, utilizando os arquivos de script fornecidos:

1. **Edite o JavaEnv.bat**

   Certifique-se de que você tenha editado o arquivo JavaEnv.bat para configurar seu ambiente de trabalho requerido.

2. **Crie uma sessão de linha de comandos**

   Crie uma sessão de linha de comandos e chame o JavaEnv.bat para tornar as configurações disponíveis no ambiente atual.

3. **Reúna o hardware necessário**

   Localize todo o hardware em que você instalará a topologia de rede. Reúna os nomes das máquinas disponíveis e anote-os. Se houver apenas uma máquina disponível, ainda assim você poderá utilizar os scripts para implementar a topologia de rede de exemplo, uma vez que pode especificar o mesmo nome do host para cada gerenciador de filas.

4. **Crie um gerenciador de filas do MQ**

   Por padrão, os scripts assumem que ele seja chamado `CENTRAL00QM` e atenda na porta 1414 para conexões de canal do cliente.

5. **Descreva o gerenciador de filas do MQ**

   Edite e revise o arquivo CentralQMDetails.bat para certificar-se de que seus detalhes correspondam àqueles do gerenciador de filas MQ recém-criado. Todos os valores, exceto o nome da máquina em que o gerenciador de filas MQ está situado, assumem o padrão no arquivo de script.

6. **Descreva o gerenciador de filas de gateway**

   Edite e revise o arquivo GatewayQMDetails.bat para certificar-se de que os detalhes do gerenciador de filas de gateway tenham sido determinados e estejam disponíveis para serem utilizados por outros arquivos .bat. O nome padrão do gerenciador de filas de gateway criado pelos scripts é `GATEWAY00QM`. Você precisará configurar o nome da máquina e o número da porta de atendimento. Essa porta deverá estar disponível para ser utilizada. ***Dica***: em máquinas Windows, utilize o comando netstat -a para obter uma lista de portas atualmente em uso.

7. **Revise o arquivo central.tst**

   Leia o arquivo central.tst e certifique-se de que ele não criará objetos MQ que você não deseja em seu gerenciador de filas MQ.

8. **Distribua todos os scripts para todas as máquinas**

   Copie todos os scripts em todas as máquinas em que executará os gerenciadores de filas MQe. Esta etapa propaga conhecimento para todas as máquinas na rede, como nomes dos hosts, números das portas e nomes dos gerenciadores de filas que você decidiu utilizar. Se algum desses arquivos for alterado, exclua todos os gerenciadores de filas MQe e reinicie a partir deste ponto nas instruções.

9. **Execute o script central.tst no novo gerenciador de filas do MQ**

   O script central.tst está em um formato utilizado pelo programa de amostra runmqsc fornecido com o MQ. Canalize o arquivo central.tst no runmqsc para configurar o gerenciador de filas MQ. Por exemplo:

   ```
   runmqsc CENTRAL00QM < Central.tst
   ```

   Utilize o MQ Explorer para visualizar os objetos MQ resultantes criados. **Marco:** agora seu sistema MQ está configurado.

10. **Execute o script CreateGatewayQM**

    O script CreateGatewayQM utiliza os detalhes dos scripts CentralQMDetails e GatewayQMDetails para criar um gerenciador de filas de gateway. O script não precisa de parâmetros.

11. **Verifique a existência da mensagem de teste**

    O script que cria o gerenciador de filas envia uma mensagem de teste para o sistema MQ. Utilize a ferramenta MQ Explorer para consultar a fila de destino (`BRANCH.SALES.QUEUE` por padrão) e certificar-se de que uma mensagem de teste tenha chegado. O corpo da mensagem de teste contém a cadeia `ABCD`. **Marco:** agora o gerenciador de filas de gateway MQe está configurado.

12. **Mantenha o gerenciador de filas de gateway em execução**

    Durante a execução do script CreateGatewayQM, um programa do servidor de exemplo é chamado para iniciar o gerenciador de filas de gateway e mantê-lo em execução. Um aplicativo AWT é executado, exibindo uma janela na tela. ***Não feche essa janela***. Todo o tempo em que essa janela estiver ativa, o gerenciador de filas de gateway MQe que ela representa também estará ativo. Fechar a janela também fecha o gerenciador de filas de gateway MQe e interrompe o caminho dos gerenciadores de filas de ramificação para o gerenciador de filas MQ.

13. **Crie um gerenciador de filas de ramificação**

    Se o gerenciador de filas de ramificação precisar ser executado em uma máquina diferente, talvez seja necessário editar o arquivo JavaEnv.bat para configurar seu ambiente local. Crie uma sessão de linha de comandos e chame JavaEnv.bat como antes para configurar o ambiente. Utilize o script CreateBranchQM para criar um gerenciador de filas de ramificação. A sintaxe do comando é:

    ```
    CreateBranchQM.bat branchNumber portListeningOn
    ```

    Em que:

    branchNumber
    : Consiste em um número de 3 dígitos, preenchido com zeros à esquerda, indicando para qual ramificação o gerenciador de filas está sendo criado. Por exemplo, 000, 001, 002...

    portListeningOn
    : Consiste em uma porta na qual o gerenciador de filas de ramificação do dispositivo atende aos pedidos de administração. Por exemplo, 8082, 8083...

Nota: A porta ainda não deve estar em uso
Sugestão: Em máquinas Windows, utilize o comando netstat -a para visualizar a lista de portas em uso. Durante o script, uma mensagem de teste é enviada ao sistema MQ. Utilize o MQ Explorer para certificar-se de que a mensagem de teste tenha chegado com êxito. O corpo da mensagem de teste contém a cadeia `ABCD`.

No final do script, um programa de exemplo é utilizado para iniciar o gerenciador de filas MQe. Um aplicativo AWT é executado, exibindo uma janela na tela. Tal como ocorre com o gerenciador de filas de gateway, ***não feche essa janela*** até que deseje fechar o gerenciador de filas.

14. **Explore o gerenciador de filas de ramificação**

    O gerenciador de filas de ramificação é configurado com um gerenciador de canais e um listener, na porta que você especificou quando o criou, e a conexão de Rede Primária é HttpTcpipAdapter. Como resultado, você pode utilizar o MQe_Explorer para visualizar os gerenciadores de filas. Consulte "Como Utilizar o MQe_Explorer para Visualizar a Configuração". **Marco:** agora você tem um gerenciador de filas de ramificação configurado.

    Nota: Um gerenciador de filas MQe deve ter um nome exclusivo. Nunca crie dois gerenciadores de filas com o mesmo nome.

    Agora você pode utilizar o MQe_Explorer para visualizar a configuração.

Como Utilizar o MQe_Explorer para Visualizar a Configuração

Para utilizar o MQe_Explorer para visualizar sua configuração:

1. Inicie o programa MQe_Explorer.exe.
2. Pare um dos gerenciadores de filas de ramificação, por exemplo, `BRANCH002QM`.
3. Abra o arquivo BRANCH002QM.ini e navegue a partir dele.

Capítulo 2. Configurando Objetos do MQe

Configurando Gerenciadores de Filas

Introdução à Configuração de Gerenciadores de Filas

O gerenciador de filas é o componente central do MQe.

Fornece a principal interface de programação para programas aplicativos e também possui subsistemas de filas, comunicações e pontes do MQ.

Java e C diferem significativamente na área de criar e excluir gerenciadores de filas:

- Em Java, a configuração geral do gerenciador de filas é desempenhada utilizando mensagens de administração, mas a criação e a exclusão são desempenhadas utilizando a classe `MQeQueueManagerConfigure`.
- Em C, toda a administração é desempenhada utilizando a API de administrador.

Java

Os gerenciadores de filas são criados e excluídos utilizando a classe `MQeQueueManagerConfigure`. A administração geral do gerenciador de filas é desempenhada utilizando a classe `MQeQueueManagerAdminMsg` que é herdada de `MQeAdminMsg`.

As seguintes ações são aplicáveis aos gerenciadores de filas:

- `MQeAdminMsg.Action_Inquire`
- `MQeAdminMsg.Action_InquireAll`
- `MQeAdminMsg.Action_Update`

O campo `MQeAdminMsg.Admin_Name` na mensagem de administração é utilizado para identificar o gerenciador de filas. O método `setName(String)` pode ser utilizado para configurar esse campo na mensagem de administração.

Nota: Para todas as mensagens de administração, as informações relacionadas ao gerenciador de filas de destino, fila de resposta e outros, devem ser configuradas. Isso é mencionado nos exemplos a seguir ao preparar a mensagem de administração.

Os exemplos mostram como criar a mensagem de administração para alcançar o resultado necessário. A mensagem precisa ser enviada e as mensagens de resposta de administração precisam ser verificadas, conforme necessário.

C

Toda a administração é feita por meio da API de administração. Essas APIs têm o formato:

```
MQERETURN MQEPUBLISHED mqeAdministrator_QueueManager_action();
```

Em que *action* é um dos seguintes:

create Cria um Gerenciador de Filas

delete Exclui um Gerenciador de Filas

update
Atualiza as Propriedades de um Gerenciador de Filas

inquire
Consulta as Propriedades de um Gerenciador de Filas

addAlias
Inclui um Alias do Gerenciador de Filas

removeAlias
Remove um Alias do Gerenciador de Filas

listAliasNames
Lista todos os Aliases Existentes para esse qmgr.

isAlias
Determina se um Nome qmgr é um Alias ou um qmgr Real.

Para as chamadas de criação, atualização e consulta, uma estrutura é transmitida para vários parâmetros.

Atributos do Gerenciador de Filas

Os Gerenciadores de Filas possuem vários atributos, que são listados a seguir. As informações sobre esses atributos são transmitidas por meio de parâmetros de API ou estruturas de configuração ou objetos MQeField.

A primeira lista mostra todos os atributos possíveis do gerenciador de filas e indica quais estão disponíveis nos códigos base.

Tabela 12. Atributos do Gerenciador de Filas

Atributo	Descrição	Java	C Nativo	Leitura/Gravação
Apto à Ponte	Determina se o gerenciador de filas possui a funcionalidade MQBridge	Sim	Sim (mas sempre falso)	Leitura
Regra de Atributo do Canal	A regra de atributo a ser utilizada pelos canais desse gerenciador de filas	Sim	Não	Leitura/Gravação
Tempo Limite do Canal	O tempo limite a ser utilizado pelos canais de saída desse gerenciador de filas	Sim	Sim	Leitura/Gravação
Communications Listeners	A lista de listeners definidos nesse gerenciador de filas	Sim	Não	Leitura
Conexões	A lista de conexões conhecidas por esse gerenciador de filas	Sim	Sim	Leitura
Descrição	Uma descrição textual de formato livre desse gerenciador de filas.	Sim	Sim	Leitura/Gravação
Máximo de Encadeamentos de Transmissão	O número máximo de encadeamentos de transmissão de segundo plano suportados por esse gerenciador de filas.	Sim	Não	Leitura/Gravação
Filas	A lista de filas pertencentes a esse gerenciador de filas	Sim	Sim	Leitura

Tabela 12. Atributos do Gerenciador de Filas (continuação)

Atributo	Descrição	Java	C Nativo	Leitura/Gravação
Armazenamento de Filas	O local no qual esse gerenciador de filas armazenará suas filas	Sim	Sim	Leitura/Gravação
Regras Qmgr	A classe de regras que será utilizada por esse gerenciador de filas	Sim	Sim	Leitura/Gravação

Java

Os parâmetros em Java são transmitidos utilizando objetos MQeFields. Os valores são transmitidos utilizando elementos de campos de tipos específicos.

Os nomes dos campos são mostrados a seguir. Todos os nomes simbólicos são cadeias public static final na classe MQeQueueManagerAdminMsg.

Tabela 13. Parâmetros Java Transmitidos Utilizando MQeFields

Tipo de Elemento	Constantes de Nomes de Campos	
	Simbólicas	Valor
booleano	QMgr_BridgeCapable	bridge_capable
ascii	QMgr_ChnlAttrRules	chnlattrrules
long	QMgr_ChnlTimeout	chnltimeout
fields array	QMgr_CommsListeners	commsls
fields array	QMgr_Connections	conns
unicode	QMgr_Description	desc
int	QMgr_MaximumTransmissionThreads	maximumTransmissionThreads
O elemento fields arrayEach possui um objeto fields contendo {QMgr_QueueName, QMgr_QueueQMgrName, QMgr_QueueType}	QMgr_Queues	filas
ascii	QMgr_QueueStore	queueStore
ascii	QMgr_Rules	regras

C

Todos os parâmetros C são transmitidos utilizando uma estrutura de parâmetros. Essa estrutura precisa ser inicializada antes de ser utilizada - configure-a como `QMGR_INIT_VAL`.

Tabela 14. Estruturas de Parâmetros para C

Tipo de Elemento	Nome do Elemento	Notas
MQEINT32	opFlags	Sinalizadores para indicar quais partes dessa estrutura foram configuradas/solicitadas
MQeStringHndl	hDescription	
MQeStringHndl	hQueueManagerRules	
MQEINT64	channelTimeOut	
MQeStringHndl	hQueueStore	

Tabela 14. Estruturas de Parâmetros para C (continuação)

Tipo de Elemento	Nome do Elemento	Notas
MQeVectorHndl	hQueues	
MQeVectorHndl	hConnections	
MQEBOOL	bridgeCapable	Valores válidos {MQE_TRUE, MQE_FALSE}

Criar um Gerenciador de Filas

Java

```
MQeFields parms = new MQeFields();
MQeFields queueManagerParameters = new MQeFields();
queueManagerParameters.putAscii(MQeQueueManager.Name, "MyQmgrName");
parms.putFields(MQeQueueManager.QueueManager, queueManagerParameters);

MQeFields registryParameters = new MQeFields();
registryParameters.putAscii(MQeRegistry.DirName, "c:\MyRegLocation");
parms.putFields(MQeQueueManager.Registry, registryParameters);

String queueStore = "MsgLog:" + java.io.File.separator + "filas";
MQeQueueManagerConfigure qmConfig = new MQeQueueManagerConfigure(parms, queueStore);

qmConfig.defineQueueManager();
qmConfig.defineDefaultSystemQueue();
qmConfig.defineDefaultDeadLetterQueue();
qmConfig.defineDefaultAdminReplyQueue();
qmConfig.defineDefaultAdminQueue();
qmConfig.close();
```

C

As informações para a fila são transmitidas por meio de uma estrutura para a API. Há dois pontos importantes:

- A estrutura é inicializada utilizando `QMGR_INIT_VAL`
- As propriedades configuradas são indicadas utilizando os elementos `opFlags` da estrutura. Cada propriedade possui uma máscara de bit correspondente – elas precisam ser ligadas com OU bit a bit.

```
MQeQueueManagerParms qmParams  = QMGR_INIT_VAL;
MQeRegistryParms  regParams = REGISTRY_INIT_VAL;

/* Parâmetros de cadeia para o local do armazenamento de mensagem */
qmParams.hQueueStore = hQueueStore;

/* Indicar quais partes da estrutura foram configuradas */
qmParams.opFlags = QMGR_Q_STORE_OP;

/* ... criar os parâmetros de registro - o mínimo requerido */
regParams.hBaseLocationName  =  hRegistryDir;

rc = mqeAdministrator_QueueManager_create(hAdministrator,
                                          &exceptBlk,
                                          &hQueueManager,
                                          hLocalQMName,
                                          &qmParams,
                                          &regParams);
```

Excluir um Gerenciador de Filas

Java

```
MQeFields parms = new MQeFields();
MQeFields queueManagerParameters = new MQeFields();
queueManagerParameters.putAscii(MQeQueueManager.Name, "MyQmgrName");
parms.putFields(MQeQueueManager.QueueManager, queueManagerParameters);

MQeFields registryParameters = new MQeFields();
registryParameters.putAscii(MQeRegistry.DirName, "c:\MyRegLocation");
parms.putFields(MQeQueueManager.Registry, registryParameters);

String queueStore = "MsgLog:" + java.io.File.separator + "filas";
MQeQueueManagerConfigure qmConfig =
      new MQeQueueManagerConfigure(parms, queueStore);

qmConfig.deleteDefaultAdminReplyQueue();
qmConfig.deleteDefaultAdminQueue();
qmConfig.deleteDefaultDeadLetterQueue();
qmConfig.deleteDefaultSystemQueue();
qmConfig.deleteQueueManager();
qmConfig.close();
```

C

Para excluir um gerenciador de filas:

- O gerenciador de filas deve ser parado
- Todas as filas devem ser excluídas
- Todas as definições de conexão devem ser excluídas

Observe que não existe estrutura de parâmetros aqui – apenas um identificador do Gerenciador de Filas.

```
rc =  mqeAdministrator_QueueManager_delete(hAdministrator,
                                           pExceptBlock);
if ( EC(&exceptBlk)  == MQERETURN_QUEUE_MANAGER_ERROR )
   {
      if(ERC(&exceptBlk) == MQEREASON_QMGR_ACTIVATED)
         {
         /* qmgr não foi parado - execute as ações apropriadas */
         }
      else if(ERC(&exceptBlk) == MQEREASON_QMGR_QUEUE_EXISTS)
         {
         /* filas existentes - execute as ações apropriadas */
         }
      else if(ERC(&exceptBlk) == MQEREASON_CONNECTION_DEFINITION_EXISTS)
         {
         /* defs de conexão existentes - execute as ações apropriadas */
         }
      else
         {
         /* erro desconhecido */
         }
   }
```

Consultar e Consultar Tudo

Em geral, ao consultar objetos no MQe, você pode:

- solicitar parâmetros específicos que sejam de interesse utilizando `inquire`
- solicitar todas as informações utilizando `inquireAll`.

Java
Consultar

```
//consultar

//Solicitar o valor de descrição
try {
    //Preparar a mensagem admin com o nome de targetQM, fila de resposta e assim por diante
    MQeAdminMsg msg = (MQeAdminMsg) new MQeQueueManagerAdminMsg();

    parms = new MQeFields();
    parms.putUnicode(MQeQueueManagerAdminMsg.QMgr_Description, null);

    //configurar o nome da fila que será consultada
    msg.setName("ExampleQM");

    //Configurar a ação requerida e seus parâmetros para a mensagem
    msg.inquire(parms);

//Enviar mensagem para a fila admin de destino (código não mostrado)

    } catch (Exception e) {
    System.err.println("Falha ! " + e.toString());
}
```

Consultar Tudo

```
//consultar tudo
try {
    MQeAdminMsg msg = (MQeAdminMsg) new MQeQueueManagerAdminMsg();

    //configurar o nome da fila que será consultada
    msg.setName("ExampleQM");

    //Configurar a ação requerida e seus parâmetros para a mensagem
    msg.inquireAll(new MQeFields());
    } catch (Exception e) {
    System.err.println("Falha ! " + e.toString());
 }
```

C

O exemplo a seguir mostra como consultar a lista de filas. Esta é a consulta mais complexa que pode ser desempenhada quando um vetor de estruturas é retornado. Todas essas estruturas devem ser liberadas conforme mostrado a seguir.

Essa estrutura de informações da fila contém três cadeias e um MQeQueueType:

- Cadeia: Nome do QueueQueueManager. Deve ser liberado
- Cadeia: QueueName. Deve ser liberado
- Cadeia de constante: O Nome da Classe Java - não precisa ser liberado
- Primitiva: MQeQueueType.

A estrutura de Informações da Fila deve ser liberada utilizando a função mqeMemory_free. Consulte Referência de Programação C para obter informações adicionais sobre a função mqeMemory.

Assim como informações sobre filas, um vetor de definições de conexão pode ser retornado. Isso também deverá ser liberado após o processamento.

```
MQeQueueManagerParms qmParms = QMGR_INIT_VAL;
qmParms.opFlags |= QMGR_QUEUES_OP;
rc = mqeAdministrator_QueueManager_inquire(hAdministrator,
                                           &exceptBlk,
                                           &qmParms);
if (MQERETURN_OK == rc) {
   /* Isso retornou um Vetor de blocos de */
```

```
      /* informações sobre as filas */
      MQeVectorHndl hListQueues = qmParms.hQueues;
      MQEINT32 numberQueues;

      rc = mqeVector_size(hListQueues,&exceptBlk,&numberQueues);
      if (MQERETURN_OK == rc) {
         MQEINT32 count;
         /* O loop circunda a matriz para obter as informações */
         /* sobre as filas */
         for (count=0;count<numberQueues;count++) {
            MQeQMgrQParms *pQueueInfo;
            rc = mqeVector_removeAt(hListQueues,
                                    &exceptBlk,
                                    &pQueueInfo,
                                     count);
            if (MQERETURN_OK == rc) {
               /* QueueManager da Fila - LIBERAR ESTA CADEIA quando concluída */
               MQeStringHndl hQMgrName = pQueueInfo->hOwnerQMgrName;
               /* QueueName - LIBERAR ESTA CADEIA*/
               MQeStringHndl hQueueName = pQueueInfo->hQMgrQName;
               /* Uma Cadeia de Constante correspondente ao Nome da Classe Java */
               /* para uma destas filas
               * MQE_QUEUE_LOCAL
               * MQE_QUEUE_REMOTE
               * MQE_QUEUE_ADMIN
               * MQE_QUEUE_HOME_SERVER
               */
               MQeStringHndl hQueueClassName = pQueueInfo->hQueueType;

               /* Será configurado a partir do MQeQueueType */
               MQeQueueType queueType = pQueueInfo->queueExactType;

               (void)mqeMemory_free(&exceptBlk,pQueueInfo);

            }
         }
      }

      /* o vetor também precisa ser liberado */
      mqeVector_free(hListQueues,&exceptBlk);
   }
```

Atualizar

Java

```
//Configurar o nome do recurso a ser gerenciado
try {
    MQeAdminMsg msg = (MQeAdminMsg) new MQeQueueManagerAdminMsg();

    msg.setName("ExampleQM");

    //Alterar o valor da descrição
    parms = new MQeFields();
    Parms.putUnicode(MQeQueueManagerAdminMsg.QMgr_Description,
                    "Alterar descrição ...");

    //Configurar a ação requerida e seus parâmetros para a mensagem
    msg.update(parms);
    } catch (Exception e) {
    System.err.println("Falha ! " + e.toString());
}
```

C

Isso mostra como atualizar a descrição. Observe que as filas, e assim por diante, não podem ser atualizadas por meio dessa API - elas devem ser concluídas por meio dos métodos de atualização específicos da Fila.

As atualizações de Description, ChannelTimeout e QueueStore são permitidas. As alterações de QueueStore terão efeito apenas para as novas filas criadas.

```
MQeQueueManagerParms qmParms = QMGR_INIT_VAL;
qmParms.opFlags | = QMGR_DESC_OP;
qmParms.hDescription = hNewDescription;
rc = mqeAdministrator_QueueManager_update(hAdministrator,
                                          &exceptBlk,
                                          &qmParms);
```

Incluir Alias

Nota: Observe que não é possível encadear os aliases juntos. Portanto, QM1 não pode ser um alias para QM2, o qual é um alias para QM3.

Java

Em Java, aliases de gerenciador de filas são manipulados utilizando o `MQeConnectionAdminMsg`.

Consulte a seção Configurando uma Conexão para obter informações adicionais.

C

O nome real do gerenciador de filas é `hRealTargetQMname` e o alias para isso é `hAliasName`.

Observe que essas cadeias serão duplicadas internamente, portanto poderiam ser liberadas se não requeridas em outra parte.

```
rc =  mqeAdministrator_QueueManager_addAlias(hAdministrator,
                                          &exceptBlk,
                                          hAliasName,
                                          hRealTargetQMName);
```

Remover Alias

Java

Em Java, aliases de gerenciador de filas são manipulados utilizando o `MQeConnectionAdminMsg`.

Consulte a seção Configurando uma Conexão para obter informações adicionais.

C

Remove o Alias `hAliasName`. Um erro será retornado se ele não existir.

```
rc = mqeAdministrator_QueueManager_removeAlias(hAdministrator,
                                            &exceptBlk,
                                            hAliasName);
```

Listar Nomes de Alias

Java

Em Java, aliases de gerenciador de filas são manipulados utilizando o `MQeConnectionAdminMsg`.

Consulte a seção Configurando uma Conexão para obter informações adicionais.

C

Lista todos os aliases em um novo `MQeVector`. Estes são os nomes de alias.

Observe que quando o vetor é liberado, seu conteúdo também será liberado automaticamente.

```
MQeVectorHndl hAliasList;

rc = mqeAdministrator_QueueManager_listAliasNames(hAdministrator,
                                                  &exceptBlk,
                                                  &hAliasList);
if (MQERETURN_OK == rc) {
   /* executar o processamento */
   rc = mqeVector_free(hAliasList,&exceptBlk);
}
```

IsAlias

Java

Em Java, aliases de gerenciador de filas são manipulados utilizando o `MQeConnectionAdminMsg`.

Consulte a seção Configurando uma Conexão para obter informações adicionais.

C

```
MQEBOOL isAlias;

rc =  mqeAdministrator_QueueManager_isAlias(hAdministrator,
                                            &exceptBlk,
                                            hName,
                                            &isAlias);

if (isAlias==MQE_TRUE) {
   /* o nome é alias */
}
```

Configurando um Gerenciador de Filas Utilizando Apenas a Memória

Este tópico aplica-se apenas ao código-base Java.

Algumas vezes, é necessário que os aplicativos tenham um gerenciador de filas que exista apenas na memória. O MQe Versão 2.0 fornece a capacidade para configurar e utilizar um gerenciador de filas utilizando apenas recursos da memória, sem que qualquer informação precise persistir em disco.

Normalmente, um gerenciador de filas MQe utiliza dois mecanismos para armazenar dados:

- As informações de configuração são armazenadas por meio de um registro em um adaptador.
- As mensagens são armazenadas por meio de um armazenamento de mensagem, que por sua vez utiliza um adaptador para armazenar dados.

O padrão é o MQeDiskFieldsAdapter, que persiste informações em disco.

Utilizar o MQeMemoryFieldsAdapter em vez do MQeDiskFieldsAdapter para ambas as tarefas permite que o gerenciador de filas seja definido, utilizado para transmitir e armazenar mensagens e totalmente excluído sem acessar um disco.

Os gerenciadores de filas MQe em memória possuem as seguintes características:

- Funcionalmente, eles podem fazer tudo o que os outros gerenciadores de filas MQe podem fazer.
- Nada é armazenado em disco.
- As mensagens e a configuração armazenadas nos registros ou filas não são persistentes. Elas serão perdidas se todas as instâncias do MQeMemoryFieldsAdapter forem submetidas à coleta de lixo ou no caso de encerramento da JVM.

- As mesmas etapas são requeridas para configurar o gerenciador de filas em memória, exceto que são requeridas toda vez que a JVM é iniciada.
- Os gerenciadores de filas transientes que são criados, utilizados e destruídos podem ser mais fáceis de serem implementados, sem problemas de limpeza se a JVM terminar anormalmente.

As soluções que consideram útil essa configuração específica de um gerenciador de filas MQe possuem as seguintes propriedades:

- O espaço em disco não está disponível ou não existe, por exemplo, em applets Java.
- O tráfego de mensagens é síncrono apenas para gerenciadores de filas remotas.
- O aplicativo não requer armazenamento de mensagem local, o qual não pode ser recuperado de outra parte se a JVM for terminada.
- O desempenho mais alto é requerido. As operações de memória são mais rápidas que as operações de disco; portanto, configurar um gerenciador de filas utilizando simplesmente recursos da memória geralmente aumenta o desempenho de configurações do gerenciador de filas que, caso contrário, armazenam informações em disco. O uso de muita memória pode resultar em esgotamento e as filas remotas síncronas geralmente são executadas na mesma velocidade em um gerenciador de filas hospedado em memória ou hospedado em disco.
- A criação e o envio de mensagens para as quais nenhuma resposta é requerida; embora os gerenciadores de filas de memória possam obter respostas, normalmente você deixa as respostas em gerenciadores de filas persistentes e as procura ou as recebe utilizando uma fila remota síncrona.

Um exemplo da técnica de configuração pode ser visto na classe `examples.queuemanager.MQeMemoryQM`. Observe que o MQeMemoryFieldsAdapter é instanciado explicitamente no início e uma referência é mantida até o ponto em que o gerenciador de filas e as mensagens que ele contém não sejam mais necessários.

Observe também que ainda é importante que os gerenciadores de filas de memória tenham nomes exclusivos na rede do sistema de mensagens.

Configurando Filas Locais

Introdução

As filas locais, como o próprio nome sugere, são locais em relação ao gerenciador de filas que as possui.

O nome de uma fila é formado pelo nome do gerenciador de filas de destino (para uma fila local, este é o nome do gerenciador de filas que possui a fila) e por um nome exclusivo para a fila nesse gerenciador de filas. Esses dois componentes de um nome da fila possuem valores ASCII.

O método `setName(String, String)` pode ser utilizado para configurar o `QueueName` e o `QueueManagerName` que o possui na mensagem de administração.

Java

O tipo mais simples de fila é uma fila local, gerenciada pela classe `MQeQueueAdminMsg`.

Para outros tipos de filas, existe uma mensagem de administração correspondente que é herdada de `MQeQueueAdminMsg`.

A `MQeQueueAdminMsg` é herdada da `MQeAdminMsg`.

As seguintes ações são aplicáveis em filas:

- `MQeAdminMsg.Action_Create`
- `MQeAdminMsg.Action_Delete`
- `MQeAdminMsg.Action_Inquire`

- `MQeAdminMsg.Action_InquireAll`
- `MQeAdminMsg.Action_Update`
- `MQeQueueAdminMsg.Action_AddAlias`
- `MQeQueueAdminMsg.Action_RemoveAlias`

Nota: Para todas as mensagens de administração, as informações relacionadas ao gerenciador de filas de destino devem ser configuradas. Isso é mencionado nos exemplos a seguir ao preparar a mensagem de administração. Os exemplos mostram como criar a mensagem de administração para alcançar o resultado necessário. As mensagens precisam, então, ser enviadas e as mensagens de resposta de administração precisam ser verificadas, conforme necessário.

C

Toda a administração é feita por meio das APIs de administração, que estão no formato:

```
MQERETURN MQEPUBLISHED mqeAdministrator_queuetype_action();
```

Em que *action* pode ser um dos seguintes:

create Criar uma Fila

delete Excluir uma Fila

update
Atualizar as propriedades de uma fila

inquire
Consultar as propriedades de uma fila

listAliasName
Lista todos os Aliases de Fila

addAlias
Incluir um Alias de Fila

removeAlias
Remover um Alias de Fila

O QueueType pode ser um dos seguintes:

- LocalQueue
- SyncRemoteQueue
- AsyncRemoteQueue
- AdminQueue
- HomeServerQueue

Para as chamadas create, update e inquire, uma estrutura é transmitida como um parâmetro. Existe uma estrutura geral para elementos que são aplicáveis a todas as filas. Para formas mais especializadas de filas, como HomeServer, existem estruturas que são compostas de uma referência à estrutura geral, além de informações adicionais. Para obter informações adicionais, consulte "Configurando com a API do Administrador C" na página 31.

Propriedades de Fila Local

As filas possuem várias propriedades, que são listadas a seguir. As informações sobre estas propriedades são transmitidas por meio de objetos distintos dos parâmetros de API ou das estruturas de configuração (`MQeFields`).

A primeira lista mostra todas as propriedades possíveis de fila e indica quais estão disponíveis nos códigos base. Todas as outras filas também terão essas propriedades.

Tabela 15. Propriedades de Fila Disponíveis em cada Código-base

Propriedade	Descrição	Java	Nativo	Leitura/Gravação
Nome da fila	Identifica o nome da fila local	Sim	Sim	Leitura (gravação na criação)
qMgr Local	O nome do gerenciador de filas locais que possui a fila	Sim	Sim	Leitura (gravação na criação)
Adaptador	A classe (ou alias) de um adaptador de armazenamento que fornece acesso ao meio de armazenamento de mensagem (consulte Adaptadores de Armazenamento na página 116)	Sim	Não – apenas um adaptador no código-base	Leitura
Alias	Os nomes de alias são nomes alternativos opcionais para a fila (consulte a seguir)	Sim	Sim	Leitura/Gravação
Regra do Atributo	A classe (ou alias) de atributo associada aos atributos de segurança da fila (para obter detalhes adicionais, consulte posteriormente neste capítulo)	Sim	Não	Leitura/Gravação
Autenticador	A classe (ou alias) de autenticador associada à fila (para obter detalhes adicionais, consulte posteriormente neste capítulo)	Sim	Não	Leitura/Gravação
Classe	A classe (ou alias) utilizada para realizar a fila local	Sim	Não	Leitura
Compressor	A classe (ou alias) de compactador associada à fila (para obter detalhes adicionais, consulte posteriormente neste capítulo)	Sim	Não	Leitura/Gravação
Criptografador	A classe (ou alias) de criptografador associada à fila (para obter detalhes adicionais, consulte posteriormente neste capítulo)	Sim	Não	Leitura/Gravação

Tabela 15. Propriedades de Fila Disponíveis em cada Código-base (continuação)

Propriedade	Descrição	Java	Nativo	Leitura/Gravação
Descrição	Uma cadeia arbitrária que descreve a fila	Sim	Sim	Leitura/Gravação
Expiração	O tempo após o qual as mensagens enviadas para a fila expiram	Sim	Sim	Leitura/Gravação
Profundidade Máxima	O número máximo de mensagens que podem ser colocadas na fila	Sim	Sim	Leitura/Gravação
Comprimento Máximo da Mensagem	O comprimento máximo de uma mensagem que pode ser colocada na fila	Sim	Sim	Leitura/Gravação
Armazenamento de Mensagem	A classe (ou alias) que determina como as mensagens na fila local são armazenadas	Sim	Não – apenas um armazenamento de mensagem disponível	Leitura (gravação na criação)
Caminho	O local do armazenamento de fila	Sim	Sim	Leitura
Prioridade	A prioridade padrão associada a mensagens na fila	Sim	Sim	Leitura/Gravação
Regra	A classe (ou alias) da regra associada à fila; determina o comportamento quando existe uma alteração no estado para a fila	Sim	Não – regras manipuladas no nível global	Leitura/Gravação
Registro de Destino	O registro de destino a ser utilizado com a classe de autenticador (ou seja, Nenhum, Fila ou Gerenciador de Filas)	Sim	Não	Leitura/Gravação

Java

Os parâmetros em Java são transmitidos utilizando os objetos `MQeFields`. Os valores são transmitidos utilizando elementos de campos de tipos específicos.

Os nomes dos campos são mostrados a seguir. Todos os nomes simbólicos são Cadeias estáticas public static final na classe `MQeQueueAdminMsg`.

Tabela 16. Propriedades de Fila Disponíveis em Java

Tipo de Elemento	Constantes de Nomes de Campos		Notas
	Simbólicas	Valor	
Unicode	Queue_CreationDate	qcd	
Int	Queue_CurrentSize	qcs	

Tabela 16. Propriedades de Fila Disponíveis em Java (continuação)

Tipo de Elemento	Constantes de Nomes de Campos		Notas
	Simbólicas	Valor	
Unicode	Queue_Description	qd	
Long	Queue_Expiry	qe	
Ascii	Queue_FileDesc	qfd	
Int	Queue_MaxMsgSize	qms	Se não houver limite, utilize Queue_NoLimit (que é -1)
Int	Queue_MaxQSize	qmqs	Se não houver limite, utilize Queue_NoLimit (que é -1)
Ascii	Queue_Mode	qm	Os valores possíveis são fornecidos pelas constantes: Queue_Asynchronous Queue_Synchronous
Byte	Queue_Priority	qp	Entre 0 e 9, inclusive
Matriz Ascii	Queue_QAliasNameList	qanl	
Ascii	Queue_QMgrName	qqmn	
Ascii	Queue_AttrRule	qar	
Ascii	Queue_Authenticator	qau	
Ascii	Queue_Compressor	qco	
Ascii	Queue_Cryptor	qcr	
Byte	Queue_TargetRegistry	qtr	Os valores possíveis são fornecidos pelas constantes: Queue_RegistryNone Queue_RegistryQMgr Queue_RegistryQueue
Ascii	Queue_Rule	qr	

C

Todos os parâmetros C são transmitidos utilizando uma estrutura de parâmetros. Essa estrutura precisa ser inicializada antes de ser utilizada, configurando-a como `LOCAL_Q_INIT_VAL`.

Tabela 17. Propriedades de Fila Disponíveis em C

Tipo de Elemento	Nome de Elemento	Descrição
MQEINT32	opFlags	Sinalizadores para indicar quais partes dessa estrutura foram configuradas/solicitadas
MQeStringHndl	hDescription	Descrição da fila
MQeStringHndl	hFileDesc	Descrição do Arquivo para o Armazenamento de Mensagem (Leitura/Criação/Gravação)
MQeVectorHndl	hQAliasNameList	Lista de Alias
MQEINT64	queueExpiry	Expiração da Fila
MQEINT64	queueCreationDate	Data de Criação da Fila
MQEINT32	queueMaxMsgSize	Tamanho Máximo da Mensagem da Fila
MQEINT32	queueMaxQSize	Número Máximo de mensagens na fila

Tabela 17. Propriedades de Fila Disponíveis em C (continuação)

Tipo de Elemento	Nome de Elemento	Descrição
MQEINT32	queueCurrentSize	Tamanho atual da Fila (todos os estados da mensagem)
MQEBOOL	queueActive	Indicação do estado da Fila
MQEBYTE	queuePriority	Prioridade de mensagens na fila

Criar uma Fila Local

Ao criar uma fila, é possível especificar vários parâmetros. Neste exemplo, uma fila é criada com um tamanho máximo de 200 mensagens, tempo de expiração de 20.000 ms e uma descrição.

Java

Primeiramente, crie o objeto `MQeQueueAdminMsg`. Isso precisa ser preparado para configurar a resposta de administração do gerenciador de filas de origem.

```
/* Criar um campo de parâmetros e de mensagens admin de fila vazia */
MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

MQeFields parms = new MQeFields();

/** Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

/* Configurar o nome da fila a ser gerenciada */
msg.setName( qMgrName, queueName );

/* Incluir qualquer característica da fila aqui, caso contrário */
/* as características serão deixadas com seus valores padrão. */
parms.putUnicode( MQeQueueAdminMsg.Queue_Description,  description);

parms.putInt32(MQeQueueAdminMsg.Queue_MaxQSize,200);
parms.putInt32(MQeQueueAdminMsg. Queue_Expiry, 20000);_

/* Configurar a ação admin para criar uma nova fila */
msg.create( parms );
```

Depois que a mensagem admin é criada, ela deve ser enviada para a fila de administração local.

C

As informações para a fila são transmitidas por meio de uma estrutura para a API. Há dois pontos importantes:

- A estrutura é inicializada utilizando `LOCAL_Q_INIT_VAL`
- As propriedades configuradas são indicadas utilizando os elementos `opFlags` da estrutura. Cada propriedade possui uma máscara de bit correspondente, que precisa ser ligada com OU. A omissão do `QUEUE_DESC_OP` significa que a fila não possui seu conjunto de descrições, mesmo que um valor esteja presente na estrutura.

```
MQeLocalQParms localQParms = LOCAL_Q_INIT_VAL;

localQParms.queueMaxQSize  = 200;
localQParms.queueExpiry    = 20000;
localQParms.queueDescription = hDescription;
//este é um MQeStringHndl

localQParms.opFlags = QUEUE_MAX_Q_SIZE_OP | QUEUE_EXPIRY_OP | QUEUE_DESC_OP;

rc = mqeAdministrator_LocalQueue_create(hAdministrator,
                                        &exceptBlk,
                                        hLocalQueueName,
                                        hLocalQMName,
                                        &localQParms);
```

Excluir uma Fila Local

Antes de excluir uma fila, ela deve estar vazia. Crie uma nova mensagem de administração e configure a ação de exclusão.

Java

```
/* Criar um campo de parâmetros e de mensagens admin de fila vazia */
MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

MQeFields parms = new MQeFields();

/** Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

/* Configurar o nome da fila a ser gerenciada */
msg.setName( qMgrName, queueName );

/* Configurar a ação admin para criar uma nova fila */
msg.delete( parms );
```

C

A exclusão de uma fila requer que a fila não contenha mensagens.

Observe que não existe nenhuma estrutura de parâmetros aqui – apenas o QueueName e o nome do QueueManager.

```
rc = mqeAdministrator_LocalQueue_delete(hAdministrator,
                                        &exceptBlk,
                                        hLocalQueueName,
                                        hLocalQMName);

if ( EC(&exceptBlk) == MQERETURN_QUEUE_ERROR
     && ERC(&exceptBlk) == MQEREASON_QMGR_QUEUE_NOT_EMPTY)
   {
      /* fila não vazia - execute as ações apropriadas */
   }
```

Incluir Alias

As filas podem ser conhecidas por vários nomes ou aliases. Se você tentar incluir um alias já existente, receberá um erro.

Java

Para incluir um nome de alias em uma fila, utilize o método `addAlias` no `MQeQueueAdminMsg`.

Com mensagens admin, várias operações de inclusão e de remoção de alias podem ser executadas em uma mensagem admin.

```
/* Criar um campo de parâmetros e de mensagens admin de fila vazia */
MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

/*  Preparar mensagem com o destinatário para resposta e um identificador exclusivo
 *  e configurar o nome do QueueManager e da Fila
 */

/* Incluir um nome que será o alias dessa fila */
msg.addAlias( "Fred" );

/* Definir a ação admin para atualizar a fila */
msg.update( parms );
```

Figura 13. Incluindo um Alias em uma Fila em Java

C

Utilize o método `addAlias()` para incluir um nome de alias.

Observe que os aliases precisam ser incluídos um por vez.

Para outros tipos de filas, como Filas Remotas, o formato da API permanece o mesmo, simplesmente altere `LocalQueue` para, por exemplo, `SyncRemoteQueue`.

```
rc = mqeAdministrator_LocalQueue_addAlias(hAdministrator,
                                          &exceptBlk,
                                          hLocalQueueName,
                                          hLocalQMName,
                                          hAliasName);

if ( EC(&exceptBlk) == MQERETURN_NOTHING_TO_DO
     && ERC(&exceptBlk) ==MQEREASON_DUPLICATE )
     {
        /* já possui alias */
     }
```

Listar Aliases

Utilize o método `listAlias()` para listar os aliases em uso.

Java

Para obter uma lista de Nomes de Alias Utilizando Mensagens de Administração, utilize a ação inquire e especifique um campo de `Queue_QAliasNameList` no objeto Fields dos parâmetros.

C

Uma lista de aliases pode ser obtida a partir da API C, utilizando a seguinte API. Observe que o Vetor deve ser liberado após o uso.

```
if (MQERETURN_OK == rc)
   {
    MQeVectorHndl hVectorAliases;
    rc = mqeAdministrator_LocalQueue_listAliasNames(hAdministrator,
                                                    &exceptBlk,
                                                    hLocalQueueName,
                                                    hLocalQMName,
                                                    &hVectorAliases);

    /* processar o vetor de aliases aqui */

    rc = mqeVector_free(hVectorAliases,&exceptBlk);

   }
```

Remover Alias

Observe que a remoção de um alias pode, provavelmente, alterar a rota das mensagens. Portanto, essa operação deve ser tratada com cautela.

Java

```
/* Criar um campo de parâmetros e de mensagens admin de fila vazia */
MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

/* Preparar a mensagem com o destinatário para resposta e um identificador exclusivo
/* e configurar o nome do QueueManager e da Fila  */

/* Especificar o alias da fila a ser removida */
msg.removeAlias( "Fred" );

/* Definir a ação admin para atualizar a fila */
msg.update( parms );
```

C

```
rc = mqeAdministrator_LocalQueue_removeAlias(hAdministrator,
                                             &exceptBlk,
                                             hLocalQueueName,
                                             hLocalQMName,
                                             hAliasName);

if ( EC(&exceptBlk) == MQERETURN_NOTHING_TO_DO
     && ERC(&exceptBlk) == MQEREASON_NO_SUCH_QUEUE_ALIAS )
       {
             /* o alias não existe */
       }
```

Atualizar

Algumas das propriedades de uma fila podem ser atualizadas.

Estas são apenas as propriedades marcadas como graváveis na tabela de propriedades.

Uma técnica semelhante é utilizada para atualizar e consultar outros tipos de filas, como filas remotas e de servidor home.

Java

O objeto de campo de parâmetro precisa ser configurado com elementos de campos que precisam ser atualizados.

```
/* Criar um campo de parâmetros e de mensagens admin de fila vazia */
MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

/*  Preparar a mensagem com o destinatário para resposta e um identificador exclusivo
*  e configurar o nome do QueueManager e da Fila
 */
MQeFields params = new MQeFields();

/* Incluir uma nova descrição para a fila */
msg.putAscii(MQeQueueAdminMsg.Queue_Descrpition,"Nova Descrição");

/* Definir a ação admin para atualizar a fila */
msg.update( parms );
```

C

De um modo semelhante à criação da Fila, a estrutura de parâmetros precisa ser configurada com os detalhes para atualização.

Por exemplo, para atualizar a descrição da fila:

```
MQeLocalQParms localQParms = LOCAL_Q_INIT_VAL;

localQParms.queueDescription = hDescription; //MQeStringHndl

localQParms.opFlags |= QUEUE_DESC_OP;

rc = mqeAdministrator_LocalQueue_update(hAdministrator,
                                        &exceptBlk,
                                        hLocalQueueName,
                                        hLocalQMName,
                                        &localQParms);
```

Consultar e Consultar Tudo

É possível consultar as propriedades da fila utilizando a ação `inquire`.

Os detalhes requeridos estão configurados.

Ao utilizar a mensagem de administração Java, a mensagem de resposta de administração contém um objeto fields com as informações requeridas.

Ao utilizar a API C, uma estrutura será preenchida com as informações solicitadas.

Java

Existem duas maneiras de consultar uma fila: `inquire` e `inquireAll`.

`InquireAll` retornará um objeto Fields na mensagem de resposta admin.

```
/* Criar um campo de parâmetros e de mensagens admin de fila vazia */
 MQeQueueAdminMsg msg = new MQeQueueAdminMsg();

/* Preparar mensagem com o destinatário para resposta e um identificador exclusivo
 * Configurar a ação admin para obter todas as características do gerenciador de filas.
 */
msg.inquireAll(new MQeFields());

/* receber de volta a mensagem da fila de resposta admin para corresponder */
/* e recuperar os resultados da mensagem de resposta */
```

O objeto fields que é retornado na mensagem de resposta de administração é preenchido com todas as propriedades da fila. Para ter acesso a um valor específico, utilize as etiquetas dos campos como na tabela de propriedades acima. Por exemplo, para obter a descrição da fila, supondo que `respMsg` seja a mensagem de resposta de administração:

```
// tudo em uma única linha:
String description = respMsg.getOutputFields().
    getAscii(com.ibm.mqe.administration.Queue_Description)
```

Em vez de solicitar todas as propriedades de uma fila, é possível solicitar e retornar as específicas. Por exemplo, se apenas a descrição for requerida, o seguinte poderá ser utilizado:

```
MQeFields requestedProperties = new MQeFields();
requestedProperties.putAscii(Queue_Description);
msg.inquire(requestedProperties)

/* Recuperar a mensagem de resposta */
/* de administração da fila relevante */
/* Em seguida, recuperar o objeto MQeFields */
/* retornado dessa mensagem */
MQeFields  outputFields = respMsg.getOutputFields();
```

Agora, `outputFields` contém apenas o campo `Queue_Description`.

C

A API utiliza a mesma estrutura de parâmetros que as outras APIs (como `create`).

Para especificar os elementos de seu interesse, configure o `opFlags` adequadamente.

Para obter, por exemplo, o comprimento máximo da fila, a expiração e a descrição, configure `opflags` conforme a seguir:

```
MQeLocalQParms params = LOCAL_Q_INIT_VAL;

params.opflags = QUEUE_MAX_Q_SIZE_OP | QUEUE_EXPIRY_OP | QUEUE_DESC_OP;

rc = mqeAdministrator_LocalQueue_inquire(hAdministrator,
                                         &exceptBlk,
                                         hQueueName,
                                         hQueueMgrName,
                                         &params);
if (MQERETURN_OK == rc) {
```

```
   MQEINT64 queueExpiry = params.queueExpiry;
   MQEINT32 queueMaxSize = params.queueMaxQSize;
   MQeStringHndl queueDescription = params. hDescription;
}
```

Adaptador de Armazenamento de Mensagem

Uma fila local utiliza um adaptador de armazenamento de fila para manipular sua comunicação com o dispositivo de armazenamento. Os adaptadores são interfaces entre o MQe e os dispositivos de hardware, como discos ou redes, ou o software, como bancos de dados. Os adaptadores foram projetados para serem componentes que podem ser conectados, permitindo a alteração fácil do armazenamento de fila.

Todos os tipos de filas que não sejam remotos e síncronos requerem um armazenamento de mensagem para armazenar suas mensagens. Cada fila pode especificar o tipo de armazenamento a ser utilizado e onde ele está localizado. A característica da fila `Queue_FileDesc` é utilizada para especificar o tipo de armazenamento de mensagem e para fornecer seus parâmetros. O descritor de arquivo tem o formato:

- `adapterClass:adapterParameters` or
- `adapterAlias:adapterParameters`

Por exemplo, supondo que `MsgLog` tenha sido definido como um alias MQe:

```
MsgLog:d:\QueueManager\ServerQM12\Queues
```

Vários adaptadores de armazenamento são fornecidos e incluem:

- MQeDiskFieldsAdapter para armazenar mensagens em um sistema de arquivo
- MQeMemoryFieldsAdapter para armazenar mensagens na memória
- Outros adaptadores de armazenamento podem ser localizados no pacote com.ibm.mqe.adapters

A opção do adaptador determina a persistência e a resiliência das mensagens. Por exemplo, se um adaptador de memória for utilizado, as mensagens serão apenas tão resilientes quanto a memória. A memória pode ser um meio mais rápido que o disco, mas é altamente volátil em comparação com o disco. Portanto, a opção do adaptador é algo importante.

Se um armazenamento de mensagem não for definido durante a criação de uma fila, o padrão será utilizar o armazenamento de mensagem que foi especificado durante a criação do gerenciador de filas.

Observe que, sob o código-base C, é fornecido apenas um armazenamento de mensagem e um único adaptador; portanto, o formato do `QueueStore` é fixo (o `MsgLog` é deixado como um sinalizador de substituição para expansão futura).

Exemplos de uso dessa opção são:

- Quando você desejar utilizar o MemoryFieldsAdapter para armazenar dados na memória e não em disco
- Armazenamentos de Mensagem Alternativos são fornecidos, como o armazenamento de mensagem ShortFilename para 4690

Considere o seguinte ao configurar o campo *`Queue_FileDesc`*:

- Assegure-se de que a sintaxe correta seja utilizada para o sistema no qual a fila reside. Por exemplo, em um sistema Windows, utilize "\" como um separador de arquivo e, em sistemas UNIX, utilize "/". Em alguns casos, é possível utilizar ambos, mas isso depende do suporte fornecido pela JVM (Java Virtual Machine) na qual o gerenciador de filas é executado. Além das diferenças do separador de arquivo, alguns sistemas como o Windows utilizam letras de unidade, mas outros como o UNIX não as utilizam.
- Em alguns sistemas, é possível especificar diretórios relativos (" .\"); em outros, isso não é possível. Mesmo naqueles em que diretórios relativos podem ser especificados, eles devem ser utilizados com

muito cuidado pois o diretório atual pode ser alterado durante a existência da JVM. Essa alteração causa problemas ao interagir com filas que utilizam diretórios relativos.

Configurando Filas Remotas

Introdução

Considere dois QueueManagers, QM_A e QM_B:

- Existe uma fila no QM_**B** denominada Queue_One – que é uma fila local no QM_B. Inicialmente, ela é acessível apenas para o QM_B, o QM_A não tem acesso a ela.
- Para ter acesso à Queue_One, o QM_**A** precisa de uma *Definição de Fila Remota* (geralmente abreviada como RemoteQueue).
- Ao fazer referência à Definição de Fila Remota, o termo *QueueQueueManager* é utilizado para referir-se ao QM_**B**, ou seja, o QueueQueueManager é o QueueManager sob o qual a LocalQueue referenciada pela Definição de Fila Remota reside.

Em resumo, as filas remotas são referências a filas que residem em um gerenciador de filas que é remoto em relação ao local da definição. A fila remota possui o mesmo nome que a fila de destino, mas a definição de fila remota também identifica o gerenciador de filas de destino ou proprietário da fila real.

A definição remota da fila deve, na maioria dos casos, corresponder àquela da fila real. Se esse não for o caso, resultados diferentes poderão ser vistos ao interagir com a fila. Por exemplo:

Para **filas assíncronas**, se o *tamanho máximo da mensagem* na definição remota for maior que na fila real, a mensagem será aceita para armazenamento na fila remota, mas poderá ser rejeitada quando movida para a fila real. A mensagem não é perdida, ela permanece na fila remota mas não pode ser entregue.

Se as características de segurança para uma **fila síncrona** não corresponderem, o MQe negociará com a fila real para decidir quais características de segurança devem ser utilizadas. Em alguns casos, a mensagem enviada é bem-sucedida; em outros, uma exceção de incompatibilidade de atributos é retornada.

Estruturas

As constantes fornecidas para configurar o parâmetro Transporte e Transportador XOR são fornecidas para retrocompatibilidade. A estrutura das Filas Remotas Assíncronas é a mesma, exceto o nome.

```
typedef struct MQeRemoteAsyncQParms
{
    /**< Estrutura de Parâmetros da Fila - para parâmetros gerais */

    MQeQueueParms      baseParms;

    /**< Classe de Transporte (Leitura/Gravação) */

    MQeStringHndl      hQTransporterClass;

} MQeRemoteAsyncQParms;
```

Síncrona e Assíncrona

A diferença entre os dois tipos de definição de fila remota é que, na definição síncrona, uma mensagem enviada para uma definição de fila remota é enviada através da rede em tempo real e enviada para a fila do gerenciador de filas remotas, enquanto na definição assíncrona, a mensagem é enviada para um armazenamento temporário e transmitida quando uma conexão de rede torna-se disponível.

Síncronas

As filas remotas síncronas são filas que podem ser acessadas somente quando conectadas a uma rede que tenha um caminho de comunicações para o gerenciador de filas proprietário (ou

próximo salto). Se a rede não for estabelecida, operações como enviar, receber e procurar causarão o surgimento de uma exceção. A fila proprietária controla as permissões de acesso e os requisitos de segurança necessários para acessar a fila. É de responsabilidade do aplicativo manipular quaisquer erros ou novas tentativas ao enviar ou receber mensagens pois, nesse caso, o MQe não é mais responsável pela entrega garantida uma única vez.

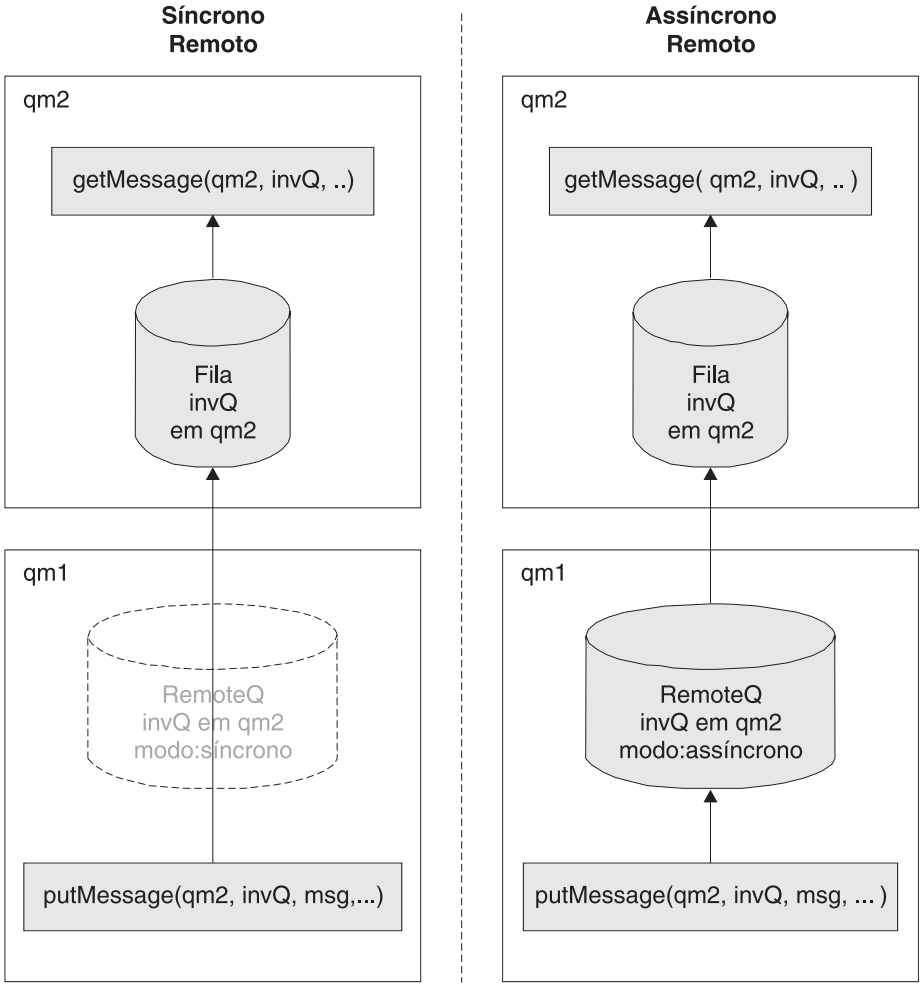
Assíncronas

As filas remotas assíncronas são filas que movem mensagens para filas remotas, mas não podem recuperar mensagens remotamente. Quando a mensagem é enviada para a fila remota, as mensagens são armazenadas localmente de modo temporário. Quando existir conectividade de rede, a transmissão tiver sido acionada e as regras permitirem, será feita uma tentativa para mover as mensagens para a fila de destino. A entrega de mensagens será no modo de entrega garantida em uma única vez.

Isso permite que os aplicativos operem na fila quando o dispositivo estiver off-line. Conseqüentemente, as filas assíncronas requerem um armazenamento de mensagem para que as mensagens possam ser armazenadas temporariamente no gerenciador de filas de envio enquanto aguardam transmissão.

Nota: No código-base Java, o *modo* de uma instância da classe MQeRemoteQueue é configurado como Queue_Synchronous ou Queue_Asynchronous para indicar se a fila é síncrona ou assíncrona. No código-base nativo, dois conjuntos distintos de APIs são utilizados para criar e administrar filas remotas síncronas e assíncronas.

Este diagrama mostra um exemplo de uma fila remota configurada para operação síncrona e uma fila remota configurada para operação assíncrona.



Nos exemplos síncrono e assíncrono, o gerenciador de filas qm2 possui uma fila local invQ.

No exemplo síncrono, o gerenciador de filas qm1 possui uma definição de fila remota da fila invQ. O invQ reside no gerenciador de filas qm2. O modo de operação é configurado como síncrono.

Um aplicativo utilizando o gerenciador de filas qm1 e enviando mensagens para a fila qm2.invQ estabelece uma conexão de rede para o gerenciador de filas qm2 (se ainda não existir) e a mensagem é enviada imediatamente na fila real. Se a conexão de rede não puder ser estabelecida, o aplicativo receberá uma exceção que ele deve manipular.

No exemplo assíncrono, o gerenciador de filas qm1 possui uma definição de fila remota da fila invQ. O invQ reside no gerenciador de filas qm2. O modo de operação é configurado como assíncrono.

Um aplicativo utilizando o gerenciador de filas qm1 e enviando mensagens para a fila qm2.invQ armazena mensagens temporariamente na fila remota no qm1. Quando as regras de transmissão permitem, a mensagem é movida para a fila real no gerenciador de filas qm2. A mensagem permanece na fila remota até que a transmissão seja bem-sucedida.

Configurando o Modo de Operação

- Para configurar uma fila para operação síncrona, configure o campo `Queue_Mode` como `Queue_Synchronous`.
- Para configurar uma fila para operação assíncrona, configure o campo `Queue_Mode` como `Queue_Asynchronous`.

As filas assíncronas requerem um armazenamento de mensagem para armazenar temporariamente as mensagens. A definição desse armazenamento de mensagem é a mesma que para as filas locais.

Criando uma Fila Remota

Os fragmentos de código a seguir mostram como configurar uma mensagem de administração para criar uma fila remota.

Para operação síncrona, as características da fila para inclusão na definição de fila remota podem ser obtidas utilizando a *descoberta de fila*.

Java

O fragmento de código a seguir mostra como configurar uma mensagem de administração para criar uma fila remota.

```
/**
 * Criar uma fila remota
 */
protected void createQueue(MQeQueueManager localQM,
                           String     targetQMgr,
                           String     qMgrName,
                           String     queueName,
                           String     description,
                           String     queueStore,
                           byte       queueMode)
   throws Exception
{
  /*
   * Criar campo de parâmetros e de mensagens admin
   * de fila vazia
   */
   MQeRemoteQueueAdminMsg msg = new MQeRemoteQueueAdminMsg();
   MQeFields parms = new MQeFields();

  /*
   * Preparar mensagem com o destinatário para resposta
   * e um identificador exclusivo
   */
   MQeFields msgTest = primeAdminMsg( msg );

  /*
   * Definir nome da fila para gerenciar
   */
   msg.setName( qMgrName, queueName );

  /*
   * Incluir qualquer característica da fila aqui, caso contrário,
   *  as características serão deixadas com seus valores padrão.
   */
   if ( description != null )  // configurar a descrição ?
      parms.putUnicode( MQeQueueAdminMsg.Queue_Description,
                        description);

  /*
   * configurar o modo de acesso de fila, se o modo for válido
   */
   if ( queueStore != MQeQueueAdminMsg.Queue_Asynchronous &&
        queueStore != MQeQueueAdminMsg.Queue_Synchronous )
      throw new Exception ("Armazenamento de fila inválido");

      parms.putByte( MQeQueueAdminMsg.Queue_Mode,
                     queueMode);

   if ( queueStore != null )   // Configurar o armazenamento de fila ?
      /*
       * Se o armazenamento de fila incluir informações sobre diretório e arquivo,
```

```
       * ele deverá ser configurado para o estilo correto do sistema no qual a
       * fila residirá; por exemplo \ ou /
       */
      parms.putAscii( MQeQueueAdminMsg.Queue_FileDesc,
                      queueStore );
  /*
   * Outras características da fila, como profundidade da fila e expiração da mensagem,
   * podem ser configuradas aqui ...
   */

  /*
   * Configurar a ação admin para criar uma nova fila
   */
   msg.create( parms );

  /*
   *  Enviar a mensagem admin para a fila
   *  admin (entrega não assegurada)
   *  no gerenciador de filas de destino
   */
   localQM.putMessage( targetQMgr,
                       MQe.Admin_Queue_Name,
                       msg,
                       null,
                       0);
}
```

C

A estrutura de parâmetros da fila remota síncrona contém dois elementos:

- O primeiro é uma estrutura de parâmetros do mesmo tipo que aquele utilizado para filas locais: `MQeQueueParms`.
- O segundo é o *transportador* para ser utilizado com essa fila.

A fila remota compartilha as propriedades da fila local e, portanto, o motivo para a estrutura da fila local.

Observe que o parâmetro `opFlags`, para especificar quais elementos da estrutura foram configurados, está na estrutura `MQeQueueParms`.

```
typedef struct MQeRemoteSyncQParms
{
   /*< Estrutura de Parâmetros da Fila para parâmetros gerais */

   MQeQueueParms     baseParms;

   /*< Classe de Transportador (Leitura/Gravação) */

   MQeStringHndl     hQTransporterClass;

} MQeRemoteSyncQParms;
```

Criar Síncrona

Java

Primeiro crie a mensagem de administração de fila remota.

```
MQeRemoteQueueAdminMsg msg = new AdminMsg();
MQeFields params = new MQeFields();
```

Em seguida, prepare a mensagem de administração, conforme explicado no Capítulo 1, “Como Configurar Objetos MQe”, na página 1.

Em seguida, configure o nome do gerenciador de filas da fila.

```
msg.setName(queueQMgrName, queueName);

params.putUnicode(descriptiorn);

/* configurar esta para ser uma fila síncrona */
params.putByte(MQeQueueAdminMsg.Queue_Mode,
               MQeQueueAdminMsg.Queue_Synchronous);
```

Agora, configure a ação de administração para criar a fila.

```
msg.create(params);

/* enviar a mensagem */
```

C

Esta é a API C para criar uma fila síncrona. É muito semelhante à criação da Fila Local. As opções para descrição, tamanho máximo e etc podem ser configuradas exatamente como as da fila local.

```
MQeRemoteSyncQParms remoteSyncQParms = REMOTE_SYNC_Q_INIT_VAL;

rc = mqeAdministrator_SyncRemoteQueue_create(hAdministrator,
                                             &exceptBlk,
                                             hQueueName,
                                             hServerName,
                                             &remoteSyncQParms);
```

Criar Assíncrona

Java

```
MQeRemoteQueueAdminMsg msg = new MQeRemoteQueueAdminMsg();
MQeFields params = new MQeFields();

/* Preparar a mensagem admin */

msg.setName(queueQMgrName, queueName);

params.putUnicode(description);

/* configurar isto para ser uma fila assíncrona */
params.putByte(MQeQueueAdminMsg.Queue_Mode,
               MQeQueueAdminMsg.Queue_Asynchronous);

/*
 * Supondo que MsgLog seja um Alias estabelecido,
 * configurar o local QueueStore
 */
params.putAscci(MQeQueueAdminMsg.Queue_FileDesc,
                "MsgLog:c:\queuestore");

/* Configurar a ação de administração para criar a fila */
msg.create(params);

/* enviar a mensagem */
```

C

Esta é a API C para criar uma fila assíncrona. É muito semelhante à criação da Fila Local. As opções para descrição, tamanho máximo e etc podem ser configuradas exatamente como as da fila local.

```
MQeRemoteAsyncQParms remoteAsyncQParms = REMOTE_ASYNC_Q_INIT_VAL;

rc = mqeAdministrator_AsyncRemoteQueue_create(hAdministrator,
                                              &exceptBlk,BROKERTRADE_Q_NAME,
                                              SERVER_QM_NAME, &remoteAsyncQParms);
```

Transportador

Um dos parâmetros de Definição de Fila Remota é o transporte em uso. É possível modificá-lo, se necessário.

Geralmente, ele é configurado como `DefaultTransporter`, `com.ibm.mqe.MQeTransporter`.

Observe que não é possível modificá-lo após a criação da Fila.

Aliases da Fila

A administração de aliases é a mesma que para as LocalQueues, porque a `MQeRemoteQueueAdminMsg` é uma subclasse da `MQeQueueAdminMsg`.

Em C, utilize as APIs a seguir da mesma maneira que para uma fila local.

```
mqeAdministrator_SyncRemoteQueue_addAlias
mqeAdministrator_SyncRemoteQueue_removeAlias

mqeAdministrator_AsyncRemoteQueue_addAlias
mqeAdministrator_AsyncRemoteQueue_removeAlias
```

Configurando Filas de Servidor Home

Introdução

Uma definição de fila de servidor home identifica uma fila de armazenamento e redirecionamento em um gerenciador de filas remotas. A fila do servidor home passa as mensagens que são destinadas ao gerenciador de filas local da fila do servidor home para a fila de armazenamento e redirecionamento. Várias definições de fila de servidor home podem ser definidas em um único gerenciador de filas, em que cada uma está associada a um gerenciador de filas remotas diferente.

As filas de servidor home normalmente residem em um dispositivo e são geralmente configuradas para receber mensagens de um servidor sempre que o dispositivo é conectado à rede. Quando uma mensagem é recebida do servidor, a mensagem é enviada para a fila local de destino correta. Se a fila de destino não existir, será chamada uma regra que permite que a mensagem seja colocada em uma fila de devoluções.

O nome da fila de servidor home é configurado da seguinte forma:

- O nome da fila deve corresponder ao nome da fila de armazenamento e redirecionamento
- O atributo de gerenciador de filas do nome da fila deve ser o nome do gerenciador de filas de servidor home
- O gerenciador de filas no qual a fila de servidor home reside deve ter uma conexão configurada para o gerenciador de filas de servidor home no qual a fila de armazenamento redirecionamento reside.

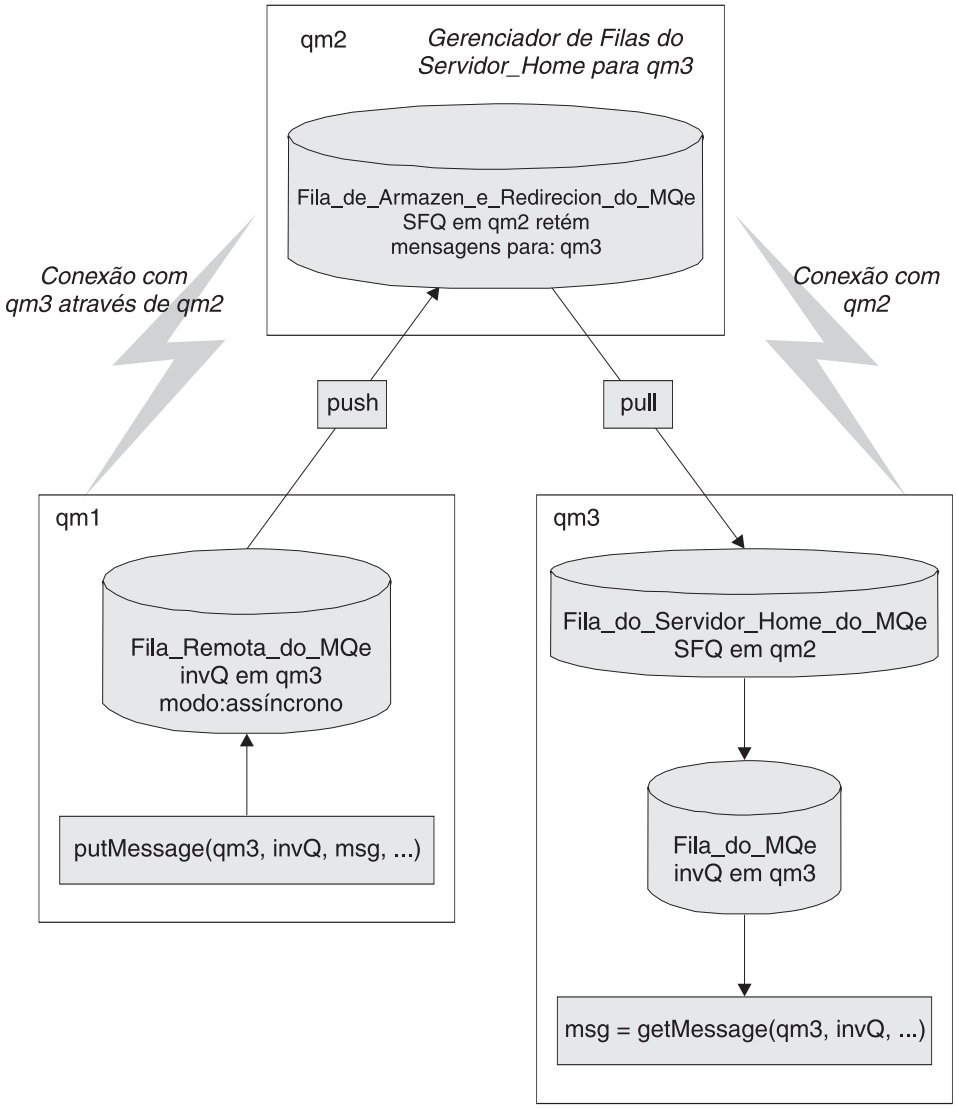


Figura 14. Fila de Servidor Home

O diagrama acima mostra um exemplo de gerenciador de filas qm3 que possui uma fila de servidor home SFQ configurada para coletar mensagens de seu gerenciador de filas de servidor home qm2. A configuração consiste de:

- Um gerenciador de filas de servidor home qm2
- Uma fila de armazenamento e redirecionamento SFQ no gerenciador de filas qm2 que contém as mensagens do gerenciador de filas qm3
- Um gerenciador de filas qm3 que normalmente é executado desconectado e não pode aceitar conexões do gerenciador de filas qm2
- O gerenciador de filas qm3 possui uma conexão configurada para o qm2
- Uma fila de servidor home SFQ que utiliza o gerenciador de filas qm2 como seu servidor home

Qualquer mensagem direcionada ao gerenciador de filas qm3 através do qm2 é armazenada na fila de armazenamento e redirecionamento SFQ no qm2 até que seja coletada pela fila de servidor home no qm3.

Mensagens de Configuração

A classe Java estende a `MQeRemoteQueueAdminMsg`, que fornece a maior parte dos recursos de administração `MQeHomeServerQueueAdminMsg` para filas remotas. Essa classe inclui ações e constantes adicionais para gerenciar filas de servidor home.

As filas de servidor home são implementadas pela classe `MQeHomeServerQueue`. Elas são gerenciadas com a classe `MQeHomeServerQueueAdminMsg`, que é uma subclasse de `MQeRemoteQueueAdminMsg`. A única inclusão na subclasse é a característica *`Queue_QTimerInterval`*. Esse campo é do tipo int e é configurado para um intervalo do cronômetro em milissegundos. Se você configurar esse campo para um valor maior que zero, a fila do servidor home verificará o servidor home a cada `n` milissegundos para saber se há mensagens esperando pela coleta. Qualquer mensagem em espera é entregue à fila de destino. Um valor igual a `0` para esse campo significa que o servidor home é controlado apenas quando o método MQeQueueManager.triggertransmission é chamado.

Nota: Se ocorrer uma falha na conexão de uma fila de servidor home com sua fila de armazenamento e redirecionamento (por exemplo, se a fila de armazenamento e redirecionamento não estiver disponível quando a fila de servidor home for iniciada), a tentativa será cessada até que uma chamada de transmissão de acionamento seja efetuada.

Transmissão de Mensagem

Java

Uma fila de servidor home pode ser solicitada para verificar mensagens pendentes:

- Configurando um intervalo de poll no campo `Queue_QTimerInterval`, uma verificação regular de mensagens é feita no servidor enquanto a conectividade está disponível. Quando a conectividade de rede não estiver disponível ou ocorrer uma interrupção de rede, o polling será parado e não será reiniciado até que a fila seja acionada utilizando o método `MQeQueueManager.triggerTransmission()`.
- Quando o método `MQeQueueManager.triggerTransmission()` é chamado.

As filas de servidor home possuem uma função importante na ativação de dispositivos para receber mensagens através de canais de cliente e servidor, particularmente em ambientes nos quais não é possível um servidor estabelecer uma conexão com um dispositivo.

C

O código-base C não possui encadeamentos de segundo plano.

Portanto, o HomeServerQueue apenas receberá mensagens de uma Fila de Armazenamento e Redirecionamento quando `mqeQueueManager_triggerTransmission` for chamado.

O método de transmissão de acionamento será retornado apenas quando for feita uma tentativa de transmitir todas as mensagens.

Criando uma Fila de Servidor Home

Java

A fila de servidor home é criada de um modo semelhante a outras filas. Geralmente, recomenda-se não utilizar um intervalo de tempo, mas controlar a transmissão utilizando `triggerTransmission`.

C

```
if (MQERETURN_OK == rc) {
   MQeHomeServerQParms homeServerQParms = HOME_SERVER_Q_INIT_VAL;

   rc = mqeAdministrator_HomeServerQueue_create(hAdministrator,
                                                &exceptBlk,
```

```
                                        hQueueName,
                                        hServerName,
                                        &homeServerQParms);
```

A administração é desempenhada utilizando as APIs a seguir.

`mqeAdministration_HomeServerQueue_action()`

A estrutura `MQeHomeServerQParms` é utilizada para transmitir parâmetros. Observe que o primeiro elemento é a estrutura `MQeRemoteSyncQParms`. Esta mapeia para a `MQeHomeServerQueueAdminMsg` herdando a função da `MQeRemoteQueueAdminMsg`.

```
typedef struct MQeHomeServerQParms
{

   /**<Parâmetros de Fila Remota a serem preenchidos  */

   MQeRemoteSyncQParms     remoteQParms;

   /**<Intervalo de Tempo - apenas para compatibilidade Java*/
   MQEINT64              qTimerInterval;

} MQeHomeServerQParms;
```

Configurando Filas de Armazenamento e Redirecionamento

Introdução

Nota: Como não existe nenhum conceito de uma fila de armazenamento e redirecionamento em C, todas as informações a seguir estão relacionadas ao código-base Java. A fila de armazenamento e redirecionamento é gerenciada pela classe `MQeStoreAndForwardQueueAdminMsg`, que é herdada de `MQeQueueAdminMsg`.

Uma fila de armazenamento e redirecionamento é normalmente definida em um servidor e pode ser configurada das seguintes formas:

- Redirecionar mensagens para o gerenciador de filas de destino ou para um outro gerenciador de filas entre os gerenciadores de filas de envio e de destino. Neste caso, a fila de armazenamento e redirecionamento envia mensagens para o próximo salto ou para o gerenciador de filas de destino
- Manter as mensagens até que o gerenciador de filas de destino possa coletá-las da fila de armazenamento e redirecionamento. Isso pode ser realizado utilizando uma fila de *servidor home*, conforme descrito em Configurando Filas de Servidor Home - Introdução. Utilizando essa abordagem, as mensagens são *recebidas* da fila de armazenamento e redirecionamento.

As filas de armazenamento e redirecionamento são implementadas pela classe MQeStoreAndForwardQueue. Elas são gerenciadas com a classe MQeStoreAndForwardQueueAdminMsg, que é uma subclasse de MQeRemoteQueueAdminMsg. A inclusão principal na subclasse é a capacidade para incluir e remover os nomes dos gerenciadores de filas nos quais a fila de armazenamento e redirecionamento pode manter as mensagens.

Além das características compartilhadas por todas as filas remotas, um objeto de fila de armazenamento e redirecionamento também possui uma propriedade que identifica seu conjunto de gerenciadores de filas de destino. O campo de cadeia `Queue_QMgrNameList`, com o valor `"qqmnl"`, identifica o campo em uma mensagem de administração que representa o conjunto de gerenciadores de filas de destino. O valor desse campo é configurado ou recuperado utilizando os métodos `putAsciiArray()` e `getAsciiArray()`.

Cada fila de armazenamento e redirecionamento precisa ser configurada para manipular mensagens para qualquer gerenciador de filas no qual ela pode manter mensagens. Utilize a ação Action_AddQueueManager, descrita anteriormente nesta seção, para incluir as informações do gerenciador de filas em cada fila:

- Se você desejar que a fila de armazenamento e redirecionamento envie mensagens para o próximo gerenciador de filas, o atributo de nome do gerenciador de filas da fila de armazenamento e redirecionamento deverá ser o nome do próximo gerenciador de filas. Uma conexão com o mesmo nome que o próximo gerenciador de filas também deve ser configurada. A fila de armazenamento e redirecionamento utiliza essa conexão como o mecanismo de transporte para enviar mensagens para o próximo salto.
- Se você desejar que a fila de armazenamento e redirecionamento espere a coleta e a recebimento de mensagens, o atributo de nome do gerenciador de filas dessa fila não terá significado , mas mesmo assim deverá ser configurado. A única restrição no atributo de gerenciador de filas em relação ao nome da fila é que não deverá existir uma conexão com o mesmo nome. Se essa conexão existir, a fila tentará utilizar a conexão para redirecionar mensagens.

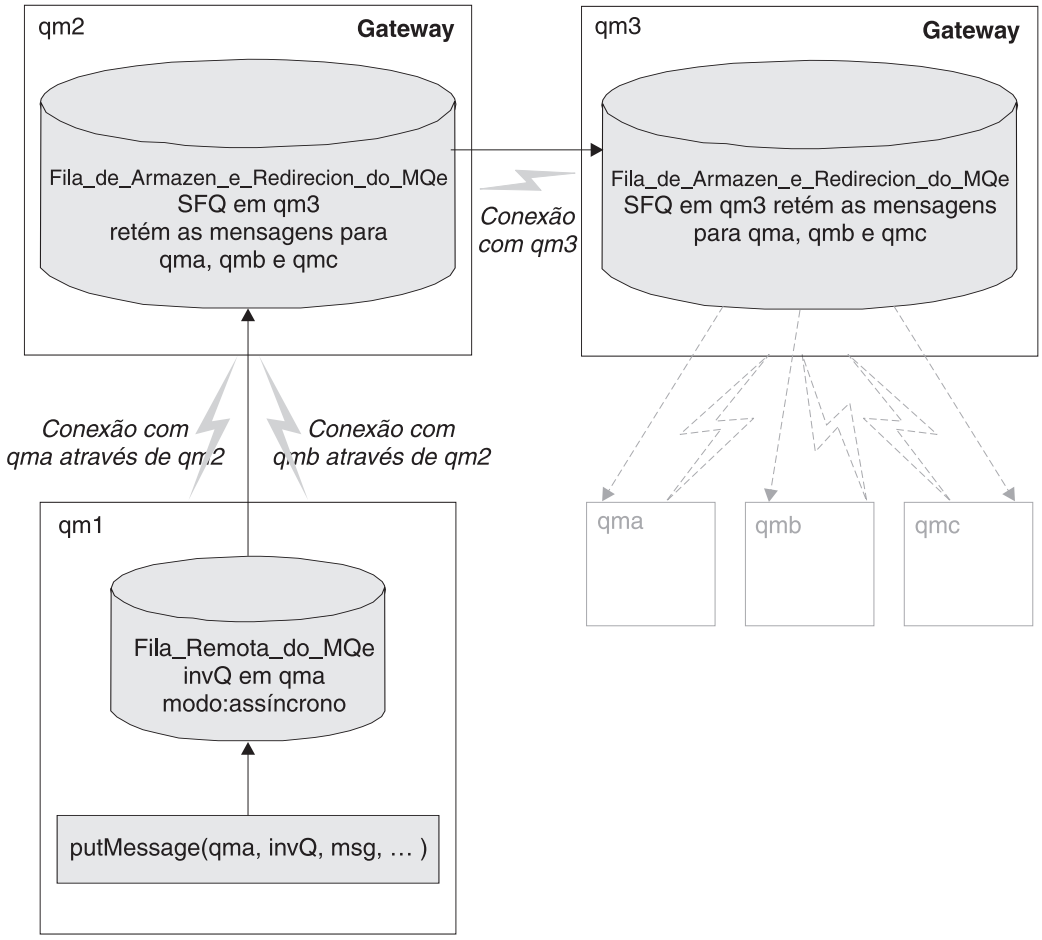


Figura 15. Fila de armazenamento e redirecionamento

O diagrama mostra um exemplo de duas filas de armazenamento e redirecionamento em gerenciadores de filas diferentes; uma configuração para enviar mensagens para o próximo gerenciador de filas e a outra para esperar a coleta de mensagens:

- O gerenciador de filas `qm2` possui uma conexão configurada para o gerenciador de filas `qm3`
- O gerenciador de filas `qm2` possui uma configuração de fila de armazenamento e redirecionamento que envia mensagens utilizando a conexão `qm3` para o gerenciador de filas `qm3`. Observe que a parte do

nome do gerenciador de filas da fila de armazenamento e redirecionamento é qm3, que corresponde ao nome da conexão. A fila de armazenamento e redirecionamento qm3.SFQ no qm2 > mantém temporariamente as mensagens em nome de qma, qmb e qmc, (mas não de qm3).

- O gerenciador de filas qm3 possui uma fila de armazenamento e redirecionamento qm3.SFQ. A parte de nome do gerenciador de filas do nome da fila qm3 não possui uma conexão correspondente chamada qm3, portanto, todas as mensagens são armazenadas na fila até que sejam coletadas.
- A fila de armazenamento e redirecionamento qm3.SFQ no qm3 mantém as mensagens em nome dos gerenciadores de filas qma, qmb e qmc. As mensagens ficam armazenadas até que sejam coletadas ou até que expirem.

Se um gerenciador de filas desejar enviar uma mensagem para outro gerenciador de filas utilizando uma fila de armazenamento e redirecionamento em um gerenciador de filas intermediárias, o gerenciador de filas iniciador deverá ter:

- Uma conexão configurada para o gerenciador de filas intermediárias
- Uma conexão configurada para o gerenciador de filas de destino, roteado através do gerenciador de filas intermediárias
- Uma definição de fila remota para a fila de destino

Quando essas condições forem satisfeitas, um aplicativo poderá enviar uma mensagem para a fila de destino, no gerenciador de filas de destino, sem ter qualquer conhecimento do layout da rede do gerenciador de filas. Isso significa que as alterações na rede do gerenciador de filas subjacente não afetam os programas aplicativos.

No diagrama, o gerenciador de filas qm1 foi configurado para permitir que as mensagens sejam enviadas para a fila invQ no gerenciador de filas qma. A configuração consiste em:

- Uma conexão para o gerenciador de filas intermediárias qm2
- Uma conexão para o gerenciador de filas de destino qma
- Uma fila assíncrona remota invQ no qma

Se um programa aplicativo utilizar o gerenciador de filas qm1 para enviar uma mensagem para a fila invQ no gerenciador de filas qma, a mensagem circulará da seguinte forma:

1. O aplicativo envia a mensagem para a fila assíncrona qma.invQ. A mensagem é armazenada localmente no qm1 para o qual ela é transmitida.
2. Quando as regras de transmissão permitem, a mensagem é movida. Com base na definição de conexão para qma, a mensagem é roteada para o gerenciador de filas qm2.
3. A única fila configurada para manipular mensagens para a fila invQ no gerenciador de filas qma é a fila de armazenamento e redirecionamento qm3.SFQ no qm2. A mensagem é armazenada temporariamente nessa fila
4. A fila de armazenamento e redirecionamento possui uma conexão que permite que ela envie mensagens para seu próximo salto, que é o gerenciador de filas qm3
5. O gerenciador de filas qm3 possui uma fila de armazenamento e redirecionamento qm3.SFQ que pode manter as mensagens destinadas ao gerenciador de filas qma, para que a mensagem seja armazenada nessa fila.
6. As mensagens para qma permanecem na fila de armazenamento e redirecionamento até que sejam coletadas pelo gerenciador de filas qma. Consulte Configurando Filas de Servidor Home - Introdução para saber como configurar isso.

Atributos da Fila de Armazenamento e Redirecionamento

As filas de armazenamento e redirecionamento possuem vários atributos além daqueles das filas remotas – eles são listados a seguir. As informações sobre esses atributos são transmitidas por meio de parâmetros de API ou estruturas de configuração/objetos MQeFields.

No Java, a lista de nomes de gerenciadores de filas identifica o campo na mensagem que representa um conjunto de gerenciadores de filas de destino. Isso não ocorre no código-base nativo.

Java

Os parâmetros em Java são transmitidos utilizando objetos MQeFields. Os valores são transmitidos utilizando elementos de campo de tipos específicos. Os nomes dos campos são os seguintes:

Tabela 18. Parâmetros Java

Tipo de Elemento	Etiqueta do Campo	Valor Textual da Etiqueta do Campo
public static final java.lang.String	Queue_QMgrNameList	″qqmnl″

Criar Fila de Armazenamento e Redirecionamento

Não existem parâmetros além daqueles utilizados na criação de uma fila remota que possam ser especificados para criar uma fila de armazenamento e redirecionamento. Neste exemplo, uma fila com uma descrição é criada.

Java

Tal como ocorre em todas as filas, a primeira ação é criar o objeto de mensagem admin apropriado. Em seguida, é necessário preparar a mensagem utilizando o código apresentado em “Configurando com Mensagens” na página 14.

```
/* Criar um campo de parâmetros e de mensagens admin de uma fila de armazenamento
e redirecionamento vazia */

MQeStoreAndForwardQueueAdminMsg msg = new MQeStoreAndForwardQueueAdminMsg();

MQeFields parms = new MQeFields();

/* Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

primeAdminMsg( msg );

/* Configurar o nome da fila a ser gerenciada */

msg.setName( qMgrName, queueName );

/* Incluir aqui qualquer característica da fila, caso contrário */
/* as características serão deixadas com seus valores padrão. */

parms.putUnicode( MQeQueueAdminMsg.Queue_Description, description);

/* Configurar a ação admin para criar uma nova fila */

msg.create( parms );
```

Após a criação da mensagem de administração, é necessário enviá-la para a fila de administração local.

Excluir Fila de Armazenamento e Redirecionamento

Neste exemplo, o construtor é utilizado para configurar o `QueueName` e o nome do `QueueManager`. Essa é uma alternativa ao uso do método `setName()` na mensagem admin.

Java

Como ocorre com todas as filas, a exclusão requer que a fila não contenha mensagens. Observe que não existe nenhuma estrutura de parâmetros aqui – apenas o QueueName e o nome do QueueManager.

```
/*  Criar uma mensagem admin da fila de armazenamento e direcionamento vazia */

MQeStoreAndForwardQueueAdminMsg msg =
   new MQeStoreAndForwardQueueAdminMsg (qMgrName, queueName);
```

```
/* Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

primeAdminMsg( msg );

/* Configurar a ação admin para excluir uma fila */

  msg.delete(new MQeFields() );
```

Incluir Gerenciador de Filas

É possível incluir e excluir nomes de gerenciadores de filas com as seguintes ações:

- `Action_AddQueueManager`
- `Action_RemoveQueueManager`

É possível incluir ou remover vários nomes de gerenciadores de filas com uma mensagem de administração.

É possível enviar nomes diretamente para a mensagem, configurando o campo de matriz ASCII `Queue_QMgrNameList`.

Alternativamente, é possível utilizar estes métodos:

- `addQueueManager()`
- `removeQueueManager()`

Cada um desses métodos obtém um único nome de gerenciador de filas, mas você pode chamar o método repetidamente para incluir vários gerenciadores de filas em uma mensagem.

Essa ação é específica para armazenar e redirecionar filas. No exemplo a seguir, vários nomes de gerenciadores de filas são incluídos em uma matriz de Cadeia (`queueManagerNames`) e configurados no objeto fields. A ação e o objeto fields são incluídos na mensagem.

Java

```
/* Criar um campo de parâmetros e de mensagem admin de uma fila de armazenamento
e redirecionamento vazia */

MQeStoreAndForwardQueueAdminMsg msg =
   new MQeStoreAndForwardQueueAdminMsg (qMgrName, queueName);

MQeFields parms = new MQeFields();

/* Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

primeAdminMsg(msg);

/*
 * Incluir qualquer característica da fila aqui, caso contrário,
 *  as características serão deixadas com seus valores padrão.
 */
parms.putAsciiArray(MQeStoreAndForwardQueueAdminMsg.Queue_QMgrNameList,
                    queueManagerNames);

/* Configurar a ação admin para incluir um gerenciador de filas em uma fila */

msg.putInt(MQeAdminMsg.Admin_Action,
           MQeStoreAndForwardQueueAdminMsg.Action_AddQueueManager);

/* Enviar o objeto fields na mensagem */

msg.putFields(MQeAdminMsg.Admin_Parms, parms);
```

Remover Gerenciador de Filas

Essa ação é específica para armazenar e redirecionar filas. Neste exemplo, o método auxiliar `removeQueueManager()` é utilizado para remover um único gerenciador de filas.

Java

```
/* Criar uma mensagem admin da fila de armazenamento e redicionamento vazia */

MQeStoreAndForwardQueueAdminMsg msg =
   new MQeStoreAndForwardQueueAdminMsg (qMgrName, queueName);

/* Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

primeAdminMsg(msg);

/* Configurar a ação admin para remover um gerenciador de filas */

msg.removeQueueManager(queueManagerName);
```

Atualizar

Neste exemplo, a descrição de uma fila de armazenamento e redirecionamento e o número máximo de mensagens permitidas na fila são atualizados.

Java

```
/* Criar um campo de parâmetros e de mensagem admin de uma fila de armazenamento
e redirecionamento vazia */

MQeStoreAndForwardQueueAdminMsg msg =
   new MQeStoreAndForwardQueueAdminMsg ();

MQeFields parms = new MQeFields();

/* Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

primeAdminMsg(msg);

/* Configurar o nome da fila a ser gerenciada */

msg.setName(qMgrName, queueName);

/*
 * Incluir qualquer característica da fila aqui, caso contrário,
 * as características serão deixadas com seus valores padrão
 */

parms.putUnicode(MQeQueueAdminMsg.Queue_Description, description);
parms.putInt(MQeQueueAdminMsg.Queue_MaxQSize,10);

/* Configurar a ação admin para atualizar */

msg.update(parms);
```

Consultar

Neste exemplo, a lista de nomes de gerenciadores de filas de uma fila de armazenamento e redirecionamento é consultada.

Java

```
/* Criar um campo de parâmetros e de mensagem admin de uma fila de armazenamento
e redirecionamento vazia */

MQeStoreAndForwardQueueAdminMsg msg = new MQeStoreAndForwardQueueAdminMsg ();

MQeFields parms = new MQeFields();
```

```
/** Preparar mensagem com o destinatário para resposta e um identificador exclusivo */

primeAdminMsg(msg);

/* Configurar o nome da fila a ser gerenciada */

msg.setName(qMgrName, queueName);

/* Incluir aqui qualquer característica da fila que você deseja consultar.*/

parms.putAsciiArray(MQeStoreAndForwardQueueAdminMsg.Queue_QMgrNameList,
                    new String[0]);

/* Configurar a ação admin a ser consultada */

msg.inquire(parms);
```

Configurando Definições de Conexão

Introdução

As definições de conexão fornecem ao MQe as informações sobre como localizar e comunicar-se com gerenciadores de filas remotas. O nome de uma definição de conexão é aquele do gerenciador de filas remotas para o qual ele descreve uma rota, portanto é provável que haja apenas uma definição de conexão direta para um gerenciador de filas remotas. Como as definições de conexão definem a rede MQe, elas são mantidas em armazenamento permanente no registro e, portanto, persistem entre as instâncias do gerenciador de filas.

A rota criada com uma definição de conexão utiliza um objeto interno, denominado canal, como o mecanismo de transporte para enviar dados entre dois gerenciadores de filas. Os canais podem não ser acessados diretamente por um usuário, mas as decisões de configuração tomadas para um gerenciador de filas afetam o comportamento de um canal.

No nível mais baixo das camadas de comunicação está o adaptador de comunicações. Torna-se necessário mencioná-lo aqui porque é impreterível que a definição de conexão especifique a mesma classe de adaptador de comunicação que a classe de adaptador utilizada pelo listener no gerenciador de filas de atendimento. Se os adaptadores de comunicação não forem idênticos, a conexão não será bem-sucedida.

Para que a definição de conexão crie uma conexão bem-sucedida para um gerenciador de filas remotas, é necessário especificar corretamente o adaptador de comunicação, o endereço de rede do gerenciador de filas de atendimento e o local de atendimento. Se alguma dessas informações estiver incorreta, não será possível estabelecer uma conexão com o gerenciador de filas remotas.

Nota:

Conforme será mostrado nos exemplos, existe muito código repetitivo envolvido na criação e na verificação da resposta de uma mensagem de administração. Portanto, é aconselhável inserir esse código em uma classe comum que possa ser utilizada por todas as classes que criam e verificam as respostas de mensagens de administração.

O código completo para atualizar uma definição de conexão e para excluir uma definição de conexão pode ser localizado nos exemplos fornecidos com o produto MQe.

Definição de Conexão Direta

Uma definição de conexão direta fornece informações para permitir que o gerenciador de filas locais crie um canal para um gerenciador de filas remotas na rede MQe. Essas são as informações reais da rede para o gerenciador de filas remotas e não envolvem qualquer rota através de outros gerenciadores de filas.

Existem duas variantes de uma conexão direta:

Definição de Conexão de Alias

Uma definição de conexão de alias fornece apenas uma parte das informações, o nome de uma definição de conexão real ou um outro alias. Esses aliases podem ser considerados como aliases de gerenciadores de filas; eles permitem que um administrador configure uma definição de conexão para um gerenciador de filas específico que, por conseguinte, pode ser referido por um outro nome.

Definição de Conexão do MQ

Esta é uma conexão especializada que identifica um gerenciador de filas remotas como um gerenciador de filas do MQ contraposto a um gerenciador de filas do MQe. Para obter informações adicionais sobre a funcionalidade de Ponte do MQe, consulte "Configurando Recursos de Ponte/Gateway" na página 84.

Definição de Conexão Indireta

Você também pode ter uma definição de conexão indireta:

Definição de Conexão de Caminho

Uma definição de conexão de caminho fornece informações para permitir que o gerenciador de filas locais crie um canal para um gerenciador de filas remotas utilizando uma rota por meio de um gerenciador de filas intermediárias. Os gerenciadores de filas intermediárias devem ser configurados de modo que tenham as definições de conexão para o próximo gerenciador de filas na rota ou para o gerenciador de filas de destino final. É de responsabilidade do administrador assegurar que todas as definições de conexão necessárias sejam configuradas na rota.

Configurando Definições de Conexão em Java

Criando uma Definição de Conexão

Para criar uma definição de conexão, uma mensagem de administração deve ser criada e enviada para a fila de administração. É necessário receber uma resposta indicando a criação bem-sucedida de uma definição de conexão antes de qualquer tentativa de utilizar a conexão; um comportamento imprevisto poderá ocorrer se for feita uma tentativa de utilizar uma conexão antes do recebimento da resposta.

Para mostrar como é possível criar uma definição de conexão, utilizamos o exemplo `examples.config.CreateConnectionDefinition`. Uma mensagem de administração de definição de conexão possui vários métodos para ajudar a criar a mensagem corretamente. Primeiramente, precisamos criar uma MQeConnectionAdminMsg:

```
MQeConnectionAdminMsg connectionMessage = new MQeConnectionAdminMsg();
```

Após a criação da mensagem de administração de conexão, precisamos configurar o nome do recurso com o qual desejamos trabalhar:

```
connectionMessage.setName("RemoteQM");
```

Agora precisamos configurar as informações na mensagem de administração que configurarão a ação a ser criada e forneceremos as informações da rota para o gerenciador de filas remotas:

```
connectionMessage.create("com.ibm.mqe.adapters.MQeTcpipHistoryAdapter:
                         127.0.0.1:8082",
                         null,
                         null,
                         "Canal Padrão",
                         "Exemplo de conexão");
```

Há várias observações a serem feitas sobre as informações transmitidas ao método create.

O primeiro parâmetro é uma cadeia delimitada por dois-pontos e afeta muito o tipo de definição de conexão que será criado. A cadeia utilizada no exemplo anterior criará uma conexão para um gerenciador de filas denominado RemoteQM utilizando o adaptador de comunicação MQeTcpipHistoryAdapter em execução na máquina local que atende na porta 8082. Se simplesmente um nome de gerenciador de filas tivesse sido especificado, por exemplo, ″ServerQM″, então uma definição de conexão de caminho teria sido criada e seria necessário já ter uma definição de conexão para o ServerQM ou criar uma antes da tentativa de utilizar a definição de conexão de caminho.

O segundo parâmetro é realmente útil apenas para adaptadores HTTP que podem executar um servlet no servidor. Nele, você definiria o nome do servlet, que seria transmitido no cabeçalho HTTP.

O terceiro parâmetro permite que a opção persistente seja configurada ou desconfigurada, embora, na realidade, isso deva ser feito com muito cuidado porque os valores padrão para persistência estão configurados nos adaptadores de comunicação para que fiquem consistentes com o protocolo sendo utilizado. Por exemplo, o MQeTcpipLengthAdapter e o MQeTcpipHistoryAdapter utilizam a persistência, ou seja, o soquete é mantido aberto; por outro lado, o MQeTcpipHttpAdapter utiliza um novo soquete para cada conversação.

O quarto parâmetro define o canal, ele deve ser sempre configurado como ″Canal Padrão″.

O quinto parâmetro fornece texto descritivo para a definição de conexão.

Agora precisamos incluir informações para a mensagem de administração que determinará qual gerenciador de filas receberá a mensagem de administração.

```
connectionMessage.setTargetQMgr("LocalQM");
```

Especifique que você deseja receber uma resposta; se utilizar o `Msg_Style_Datagram`, indique que nenhuma resposta é requerida. A resposta indica sucesso ou falha da ação administrativa.

```
connectionMessage.putInt(MQe.Msg_Style, MQe.Msg_Style_Request);
```

A fila e o gerenciador de filas que receberão a resposta; pode não ser necessariamente o gerenciador de filas que criou e enviou a mensagem de administração. Utilizar a fila de resposta de administração padrão permite utilizar a definição da Cadeia fornecida na classe MQe. Além disso, a resposta deve chegar à fila local.

```
connectionMessage.putAscii(MQe.Msg_ReplyToQ, MQe.Admin_Reply_Queue_Name);
connectionMessage.putAscii(MQe.MSG_ReplyToQMgr, "LocalQM");
```

Um identificador exclusivo deve ser incluído na mensagem antes de enviá-la para a fila de administração. Isso permite identificar a mensagem de resposta apropriada. Utilize o horário do sistema para fazer isso.

```
String match = "Msg" + System.currentTimeMillis();
connectionMessage.putArrayOfByte(MQe.Msg_CorrelID, match.getByte());
```

Agora você pode enviar a mensagem de administração para a fila de administração padrão; o quarto parâmetro permite que um MQeAttribute seja especificado com o quinto parâmetro que suporta um identificador que possibilita desfazer um envio. Como nenhum deles é obrigatório, especifique nulo e zero, respectivamente.

```
queueManager.putMessage("LocalQM",
                        MQe.Admin_Queue_Name,
                        connectionMessage, null, 0);
```

Antes de utilizar seguramente a definição de conexão, precisamos assegurar que ela tenha sido criada corretamente e, portanto, devemos esperar uma resposta. Especificamos se a resposta deve ser enviada ao gerenciador de filas LocalQM na fila de resposta de administração padrão. Criamos um filtro utilizando o ID de correlação, portanto recebemos a resposta correta:

```
MQeFields filter = new MQeFields();
filter.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());
```

Agora, utilizando o filtro criado, esperamos por uma mensagem de resposta na fila de respostas de administração padrão. O retorno do método waitForMessage fornece um MQeMsgObject, portanto nós o difundimos para um MQeAdminMsg. O quarto parâmetro configurado como nulo pode ser utilizado para um MQeAttribute. Isso é configurado como nulo porque não utilizamos a segurança durante esse exemplo. O quinto parâmetro transmitido como zero é para um ID de confirmação que pode ser utilizado em uma operação desfazer, mas novamente não o utilizamos. O último parâmetro define o tempo de espera em milissegundos; estamos esperando há três segundos.

```
// tudo em uma única linha
MQeAdminMsg response = (MQeAdminMsg)
                       queueManager.waitForMessage(queueManagerName,
                       MQe.Admin_Reply_Queue_Name,
                       filter,
                       null, 0, 3000);
```

Depois de recebermos a resposta, verificamos se temos um código de retorno bem-sucedido; existe uma verificação adicional no exemplo, para os propósitos deste manual, verificamos apenas o retorno bem-sucedido. Como pode ser observado, existe um método útil na mensagem de administração que nos retornará um código de retorno para facilitar a verificação.

```
switch (response.getRC()) {
   case MQeAdminMsg.RC_Success :
      System.out.println("conexão criada");
      break;
```

Criamos com sucesso uma definição de conexão para um gerenciador de filas remotas.

Alterando e Excluindo Definições de Conexão

As definições de conexão definem a rede para o MQe e, portanto, deve-se tomar muito cuidado ao alterá-las ou excluí-las. É altamente recomendável que, ao alterar ou excluir uma definição de conexão, alguém se assegure de que não exista atividade na rede que possa estar utilizando a definição de conexão.

Assim como ao criar uma definição de conexão, para alterar ou excluir uma definição de conexão, é necessário utilizar uma mensagem de administração. A abordagem é a mesma que para criar uma definição de conexão, com uma ação diferente sendo utilizada para a mensagem de administração. Por exemplo, para atualizar uma definição de conexão, o seguinte método deve ser utilizado:

```
updateMessage.update(
     "com.ibm.mqe.adapters.MQeTcpipHttpAdapter:127.0.0.1:8083",
      null, null, "DefaultChannel", "Exemplo de Conexão Alterada");
```

Para excluir uma definição de conexão, é necessário apenas o nome do recurso e ação relevante que está sendo configurada, portanto o seguinte método é utilizado:

```
deleteMessage.setAction(MQeAdminMsg.Action_Delete);
```

Configurando Definições de Conexão em C

Existe uma diferença importante entre a administração disponível em C e aquela em Java. O produto Java dispõe unicamente da mensagem de administração, o C fornece uma API de administração para o usuário administrar localmente o MQe. Informações adicionais sobre a API de administração podem ser localizadas em “Configurando com a API do Administrador C” na página 31; este capítulo presume que você já tenha lido o capítulo sobre administração e saiba como criar um identificador de administração e um bloco de exceções utilizados nas chamadas à API de administração. Este exemplo está em transport.c no broker.dll para C.

Antes de examinar as funções individuais que fornecem a API para administrar a definição de conexão, vale a pena examinar a estrutura que contém as informações sobre a definição de conexão que são transmitidas para todas as funções que requerem informações, ou seja, todas exceto a função para excluir a definição de conexão. A estrutura MQeConnectionDefinitionParms é a seguinte:

```
MQEVERSION         version;
MQEINT32           opFlags;
MQeStringHndl      hDescription;
MQeStringHndl      hAdapterClass;
MQeStringHndl *    phAdapterParms;
MQEINT32           destParmLen;
MQeStringHndl      hAdapterCommand;
MQeStringHndl      hChannelClass;
MQeStringHndl      hViaQMName;
```

Versão
Este é um campo apenas para uso interno e não deve ser configurado pelo usuário.

opFlags
Na entrada de uma função, esse campo fornece sinalizadores de bits que indicam as áreas do recurso a serem administradas. Na saída de uma função, se a ação tiver obtido êxito, os sinalizadores indicarão as operações desempenhadas; se a ação tiver falhado, os sinalizadores indicarão o componente com falha.

hDescription
A descrição para essa definição de conexão.

hAdapterClass
A classe de adaptador de comunicações que será utilizada por essa definição de conexão; atualmente, existe apenas um adaptador de comunicações para C. No arquivo de cabeçalho MQe_Adapter_Constants.h existe uma constante para definir a classe – MQE_HTTP_ADAPTER.

phAdapterParams
Uma matriz que contém as informações de rede requeridas para conectar-se ao gerenciador de filas remotas. Em uma rede IP, isso conterá o endereço de rede e a porta IP. O endereço IP é assumido como o primeiro elemento e o número da porta é assumido como o segundo elemento na matriz.

destParmLen
O comprimento da matriz phAdapterParams.

hAdapterCommand
Esse campo pode conter um nome de servlet a ser incluído em um cabeçalho HTTP.

hChannelClass
A classe de canal a ser utilizada; deve ser configurada como MQE_CHANNEL_CLASS, definido em MQe_Connection_Constants.h

hViaQMName
Se essa definição de conexão especificar uma conexão de caminho, todos os outros parâmetros deverão ser nulos com esse parâmetro contendo o nome do gerenciador de filas do caminho.

Uma constante em MQe_Connection_Constant.h - CONNDEF_INIT_VAL configurará os valores dessa estrutura para os valores iniciais, que podem ser alterados conforme necessário.

Criando uma Definição de Conexão

Para criar uma definição de conexão, precisaremos chamar a função:

```
mqeAdministrator_Connection_create(MQeAdministratorHndl, hAdmin,
                                   MQeExceptBlock* pExceptBlock,
                                   MQeStringHndl hConnectionName,
                                   MQeConnectionDefinitionParms* pParams);
```

O terceiro parâmetro definirá o nome da definição de conexão. Conforme mencionado, esse deve ser o nome do gerenciador de filas remotas no qual essa definição de conexão mantém a rota.

O quarto parâmetro é uma estrutura que mantém as informações necessárias para configurar as informações de definição de conexão. É necessário configurar o campo hViaQMName ou o

hAdapterClass, phAdapterParams, destParmLen, hAdapterCommand e hChannelClass, para criar uma definição de conexão. Por exemplo, para criar uma definição de conexão, primeiro crie e configure uma estrutura de parâmetros MQeConnectionDefinition:

```
/* Criar a estrutura e configurá-la para os valores iniciais */
MQeConnectionDefinitionParms parms = CONNDEF_INIT_VAL;
```

Crie um MQeString para manter o nome do gerenciador de filas remotas, que torna-se o nome da definição de conexão:

```
rc = OSAMQESTRING_NEW(&error, "ServerQM", SB_STR, &hQueueMgrName);
```

Configure os nomes das classes de adaptador e de canal. Eles devem ser configurados para estes nomes, pois são as únicas classes suportadas atualmente:

```
parms.hAdapterClass = MQE_HTTP_ADAPTER;
parms.hChannelClass = MQE_CHANNEL_CLASS;
```

Para configurar uma matriz, precisamos alocar um pouco de memória, em seguida, configurar as informações de rede. Este exemplo mostra a utilização do endereço de auto-retorno com o listener supostamente na porta 8080:

```
OSAMEMORY_ALLOC(&error, (MQEVOID**) &parms.phAdapterParms,
                (sizeof(MQEHANDLE) * 2), "teste comms");
rc = OSAMQESTRING_NEW(&error, "127.0.0.1", SB_STR,
                      &parms.phAdapterParms[0]);
rc = OSAMQESTRING_NEW(&error, "8080", SB_STR,
                      &parms.phAdapterParms[1]);
```

Agora, configuramos o número de elemento na matriz:

```
parms.destParmLen = 2;
```

E, por último, configure os sinalizadores para indicar à função de administração receptora quais informações devem ser procuradas na estrutura:

```
parms.opFlags = CONNDEF_ADAPTER_CLASS_OP |
                CONNDEF_ADAPTER_PARMS_OP |
                CONNDEF_CHANNEL_CLASS_OP;
```

Agora, depois de configurar tudo, podemos chamar a função de administração para criar a definição de conexão. Nota: é aconselhável verificar o código de retorno para determinar se a chamada foi bem-sucedida

```
rc = mqeAdministrator_Connection_create( hAdministrator, &error,
                                         hQueueMgrName, &parms);
if (MQERETURN_OK == rc) {
   fprintf(pOutput,
           "definição de conexão para ServerQM em 127.0.0.1:8081 incluída com êxito\n");
   }
```

Esse trecho cria uma definição de conexão direta; para criar uma definição de conexão de caminho, seria necessário configurar a estrutura de parâmetros para os valores padrão e o nome do gerenciador de filas remotas normalmente:

```
MQeConnectionDefinitionParms parms = CONNDEF_INIT_VAL;
rc = OSAMQESTRING_NEW(&error, "ServerQM", SB_STR, &hQueueMgrName);

Agora precisamos configurar o nome do gerenciador de filas que, por conseguinte, roteará
as mensagens para o gerenciador de filas remotas.

rc = OSAMQESTRING_NEW(&error,
                      "RoutingQM",
                      SB_STR,
                      &parms.hViaQMName);
```

Agora só falta configurar corretamente os sinalizadores, que indicarão à função de administração o que deve ser procurado na estrutura:

```
parms.opFlags  = CONNDEF_VIAQM_OP;
```

Chamamos então a função, como com a definição de conexão direta:

```
rc = mqeAdministrator_Connection_create(hAdministrator,
                                        &error,
                                        hQueueMgrName,
                                        &parms);
```

Alterando e Excluindo Definições de Conexão
Alterando uma Definição de Conexão

Conforme já mencionado, é altamente recomendável assegurar que uma conexão não esteja sendo utilizada quando uma definição de conexão é atualizada. Os sinalizadores são utilizados para determinar quais partes das informações na definição de conexão devem ser atualizadas. Portanto, mesmo se um valor for fornecido na estrutura, se o sinalizador correto não estiver configurado, esse valor não será utilizado:

```
MQeConnectionDefinitionParms parms = CONNDEF_INIT_VAL;
```

Criaremos uma nova descrição:

```
rc = OSAMQESTRING_NEW(&error, "descrição de substituição", SB_STR,
                      &parms.hDescription);
```

Se configurarmos o campo `opFlags` conforme a seguir, a descrição não será atualizada; em vez disso, a função de administração tentará atualizar o valor para o nome do gerenciador de filas do caminho:

```
parms.opFlags  = CONNDEF_VIAQM_OP;
```

Precisamos configurar o campo `opFlags` conforme a seguir para obter o comportamento desejado:

```
Parms.opFlags = CONNDEF_DESC_OP;
```

A função para atualizar a definição de conexão é então chamada, conforme a seguir:

```
rc = mqeAdministration_Connection_update(hAdministrator , &error,
                                         hQueueMgrName, &parms);
```

Excluindo Definições de Conexão

Uma conexão pode ser excluída conforme a seguir. Se a conexão não existir, o código de retorno `MQERETURN_COMMS_MANAGER_WARNING` será fornecido com o código de razão `MQEREASON_CONDEF_DOES_NOT_EXIST`.

```
rc = mqeAdministrator_Connection_delete(hAdministrator,
                                        &error, hQueueMgrName);
```

Configurando um Listener

Para que um gerenciador de filas receba pedidos de outros gerenciadores de filas, é necessário que um `MQeListener` esteja instanciado e em execução.

Nota: Essa funcionalidade está disponível apenas em Java.

Um listener utiliza um adaptador de comunicações para atender em um local nomeado; em uma rede IP, isso é uma porta nomeada.

Para que um cliente faça uma conexão bem-sucedida, o endereço de rede do gerenciador de filas de atendimento, o local nomeado e a classe de adaptador de comunicações devem ser informados ao cliente.

Um erro em uma dessas definições de conexão no cliente resultará em um erro de conexão.

Java

Para criar um listener, é necessário utilizar uma mensagem de administração. O trecho a seguir baseia-se no exemplo `example.config.ConfigListener`, a mensagem de administração é instanciada da seguinte forma:

```
MQeCommunicationsListenerAdminMsg createMessage =
   new MQeCommunicationsListenerAdminMsg();
```

Agora precisamos fornecer um nome para o listener:

```
createMessage.setName("Listener1");
```

Também é necessário o nome do gerenciador de filas para o qual a mensagem de administração está destinada:

```
createMessage.setTargetQMgr(queueManagerName);
```

O próximo passo é configurar a ação para a mensagem de administração, bem como fornecer as informações necessárias para o funcionamento do listener.

```
createMessage.create(com.ibm.mqe.adapters.MQeTcpipHistoryAdapter,
                     8087, 36000000, 10);
```

O primeiro parâmetro fornece o nome do adaptador de comunicações que desejamos utilizar; nesta instância, determinamos o MQeTcpipHistoryAdapter; é possível utilizar um alias no lugar. O tipo de adaptador de comunicações utilizado pelo listener precisa ser informado aos clientes que desejam conectar-se ao gerenciador de filas utilizando o listener.

O segundo parâmetro define o local nomeado que o listener utiliza; nesta instância, um número de porta IP de 8087; novamente os clientes precisarão ser informados disso para contactar esse listener.

O terceiro parâmetro especifica o valor de tempo limite do canal. Esse valor é utilizado para determinar quando um canal de entrada deve ser fechado. O MQe controla os canais; se um canal estiver inativo por mais tempo que o valor de tempo limite, ele será fechado.

O último parâmetro determina o número máximo de canais que o listener terá em execução de uma única vez. Se um cliente tentar conectar depois que esse valor tiver sido atingido, a conexão será negada.

Depois de configurarmos a ação correta e fornecermos as informações relevantes, podemos configurar o tipo de mensagem; nesta instância, estamos utilizando um estilo da mensagem de pedido que indica que gostaríamos de uma resposta para indicar sucesso ou falha. Entretanto, talvez não faça diferença saber se uma descrição foi alterada com êxito ou não. Neste caso, utilize um estilo da mensagem de datagrama que indica que nenhuma resposta é necessária.

```
createMessage.putInt(MQe.Msg_Style, MQe.Msg_Style_Request);
```

Ao solicitar uma resposta, forneça os nomes da fila e do gerenciador de filas proprietário para os quais a resposta deve ser enviada. Este exemplo utiliza a fila de resposta de administração padrão.

```
createMessage.putAscii(MQe.Msg_ReplyToQ, MQe.Admin_Reply_Queue_Name);
createMessage.putAscii(MQe.Msg_ReplyToQMgr, queueManagerName);
```

Para receber a mensagem de resposta correta que corresponda à mensagem de administração, utilize um ID de correlação. Isto é copiado da mensagem de administração para a resposta, para que possamos receber a mensagem correta. Para gerar um ID que seja relativamente seguro em termos de exclusividade, utilize a hora do sistema:

```
String match = "Msg" + System.currentTimeMillis();
createMessage.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());
```

Agora vamos enviar a mensagem de administração para a fila de administração do gerenciador de filas de destino. Os últimos dois parâmetros fornecem a capacidade para utilizar um atributo e um ID para permitir que o método desfazer seja chamado, com os quais não devemos nos preocupar agora.

```
queueManager.putMessage(queueManagerName, MQe.Admin_Queue_Name,
                        createMessage, null, 0);
```

Depois de enviada a mensagem para a fila, agora precisamos esperar uma resposta. Como pudemos acompanhar, usamos o identificador de correlação que utilizamos para enviar a mensagem a fim de receber a resposta e existe um método seguro que nos fornece o código de razão para indicar sucesso ou falha.

```
MQeFields filter = new MQeFields();
filter.putArrayOfByte(MQe.Msg_CorrelID, match.getBytes());

// esperar agora uma resposta
MQeAdminMsg response = (MQeAdminMsg)
                        queueManager.waitForMessage(queueManagerName,
                        MQe.Admin_Reply_Queue_Name,
                        filter,
                        null, 0, 3000);

// a mensagem de administração possui um método que
// extrairá o código de retorno:
switch (response.getRC()) {
   case MQeAdminMsg.RC_Success :
      break;
```

Depois de termos criado nosso listener com êxito, precisamos iniciá-lo; o listener será iniciado automaticamente apenas no próximo reinício do gerenciador de filas. Novamente, uma mensagem de administração é necessária para iniciar ou parar um listener; nós podemos utilizar a abordagem anterior, utilizando os métodos a seguir na classe MQeCommunicationsListenerAdminMsg. Para iniciar o listener:

```
MQeCommunicationsListenerAdminMsg startMessage =
   new MQeCommunicationsListenerAdminMsg();
.
.
.
startMessage.start();
```

Para parar o listener:

```
MQeCommunicationsListenerAdminMsg startMessage =
   new MQeCommunicationsListenerAdminMsg();
.
.
.
startMessage.stop();
```

Para excluir um listener, precisamos configurar a ação da mensagem de administração para exclusão, da seguinte forma:

```
deleteMessage.setAction(MQeAdminMsg.Action_Delete);
```

Se você tentar excluir um listener em execução, receberá uma exceção; portanto, certifique-se de que o listener seja parado com êxito antes de tentar excluí-lo.

Configurando Recursos de Ponte/Gateway

Introdução à Ponte do MQ

Esta seção descreve como o MQe pode ser preparado para interagir com o MQ utilizando um `gateway`.

- Um `gateway` é um gerenciador de filas do MQe configurado com uma `ponte` que permite que ele interaja com o MQ.

- A ponte é um objeto do MQe (no mesmo sentido que filas, conexões e etc. são objetos).
- O gerenciador de filas do MQ não requer configuração especial. Ele é referido no MQe como o gerenciador de filas da fila.
- O gateway é executado em uma máquina que age como um servidor que pode conectar-se a uma outra máquina que está executando o MQ. O gateway não pode ser executado em um dispositivo.
- O gateway e o MQ podem, ambos, ser executados na mesma máquina, se necessário.
- Em uma configuração completa, um gateway pode ter várias pontes configuradas (isso é muito incomum).

O Que Torna um Gerenciador de Filas Ativado para Ponte

Alguns gerenciadores de filas do MQe são capazes de trocar mensagens com o MQ e outros não.

Aqueles que possuem essa capacidade são chamados de ativados para ponte ou aptos à ponte. Simplificando, um gerenciador de filas ativado para ponte é aquele executado em um ambiente capaz de suportar as classes do Java MQ, e quando o software de ponte do MQ está disponível para ser carregado pela JVM.

Ao ser ativado, um gerenciador de filas do MQe tenta carregar o componente de software de ponte do MQ. Se todas as classes do MQe e o software dependente forem carregáveis, o gerenciador de filas poderá relatar posteriormente que é apto à ponte. Se as classes Java requeridas não forem carregáveis, as informações de erro serão rastreadas nesse ponto, mas o gerenciador de filas continuará ativado, resultando em um gerenciador de filas que relata que não é apto à ponte.

Descobrindo se um Gerenciador de Filas Está Ativado para Ponte

Se você aplicar uma operação inquireAll a um gerenciador de filas, uma propriedade bridge-capable será retornada. Esse campo é booleano. Um valor true indica que as classes requeridas para suportar a função de ponte estão presentes no caminho de classe. Um valor false indica que as classes requeridas estão ausentes no caminho de classe.

- Se o gerenciador de filas relatar que é apto à ponte, os recursos de ponte poderão ser configurados e manipulados nesse gerenciador de filas.
- Se o gerenciador de filas relatar que não é apto à ponte, qualquer tentativa de administrar os recursos de ponte falharão. Essas situações geralmente indicam que as classes Java MQ requeridas, ou partes do software de ponte do MQ, não estão disponíveis no caminho de classe.

  A alteração do caminho de classe para referenciar as classes Java MQ e as classes de ponte do MQ e o reinício da JVM na qual o gerenciador de filas do MQe é executado deverão resultar no gerenciador de filas relatando que é apto à ponte. O código em examples.mbridge.administration.commandline.IsQueueManagerBridgeCapable fornece um exemplo de como codificar essa consulta.

Classes para Ativar um Gerenciador de Filas para Ponte

Para utilizar a ponte do MQ, você deve ter estas duas disposições:

1. MQ Classes para Java versão 5.1 ou posterior, instalado em seu sistema MQe e disponível no caminho de classe para ser utilizado pelas JVMs. O MQ Classes para Java está disponível para download gratuito na Web como SupportPac MA88. O download é gratuito. O MQ Classes para Java também é fornecido com o software MQ, mas talvez não seja instalado dependendo das opções selecionadas quando o MQ foi instalado. Um exemplo de script a seguir demonstra o que pode ser necessário para configurar o ambiente correto em um sistema Windows. Este exemplo foi extraído da pasta Java\Demo\Windows. Um exemplo de bsh semelhante do UNIX pode ser localizado no diretório Java\Demo\Unix.

   ```
   @Rem  Configurar o nome do diretório do MQ Series.
   @Rem  Isso deve ser modificado de acordo com sua instalação.
   set MQDIR=C:\Program Files\IBM\MQSeries
   ```

```
@Rem   Se você desejar utilizar a ponte do MQ, o CLASSPATH também
@Rem   precisará saber como chegar ao MQSeries Java Client.
if Exist "%MQDIR%\java\lib"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib;
if Exist "%MQDIR%\java\lib\com.ibm.mq.jar"^
set CLASSPATH=%CLASSPATH%;  %MQDIR%\java\lib\com.ibm.mq.jar
if Exist "%MQDIR%\java\lib\com.ibm.mqbind.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\com.ibm.mqbind.jar
if Exist "%MQDIR%\java\lib\com.ibm.mq.iiop.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\com.ibm.mq.iiop.jar
if Exist "%MQDIR%\java\lib\jta.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\jta.jar
if Exist "%MQDIR%\java\lib\jndi.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\jndi.jar
if Exist "%MQDIR%\java\lib\jms.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\jms.jar
if Exist "%MQDIR%\java\lib\com.ibm.mqjms.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\com.ibm.mqjms.jar
if Exist "%MQDIR%\java\lib\connector.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\connector.jar
if Exist "%MQDIR%\java\lib\fscontext.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\fscontext.jar
if Exist "%MQDIR%\java\lib\ldap.jar"^
set CLASSPATH=%CLASSPATH%;%MQDIR%\java\lib\ldap.jar

@Rem    A Ponte do MQSeries também requer acesso aos Executáveis do MQSeries
@Rem    para que as DLLs nativas possam ser localizadas.
if Exist "%MQDIR%\java\lib" set PATH=%PATH%;%MQDIR%\java\lib
if Exist "%MQDIR%\bin"      set PATH=%PATH%;%MQDIR%\bin;
```

2. Classes do MQe, das quais um exemplo de classes de superconjunto pode ser localizado no arquivo Java/Jars/MQeGateway.jar. A implementação desse arquivo e sua inclusão no caminho de classe fornecerá ao gerenciador de filas todas as classes que são necessárias para utilizar a função de ponte. Por exemplo,

   ```
   set CLASSPATH=%CLASSPATH%;%MQeDIR%\Java\Jars\MQeGateway.jar
   ```

Visão Geral de Configuração da Ponte

A configuração da ponte do MQ requer que você desempenhe algumas ações no gerenciador de filas do MQ e outras no gerenciador de filas do MQe. A ponte pode ser dividida em duas partes:

- Configuração de recursos requeridos para rotear uma mensagem do MQe para o MQ
- Configuração de recursos requeridos para rotear uma mensagem do MQ para o MQe

A configuração de ambos os tipos de rotas é discutida nas seções seguintes.

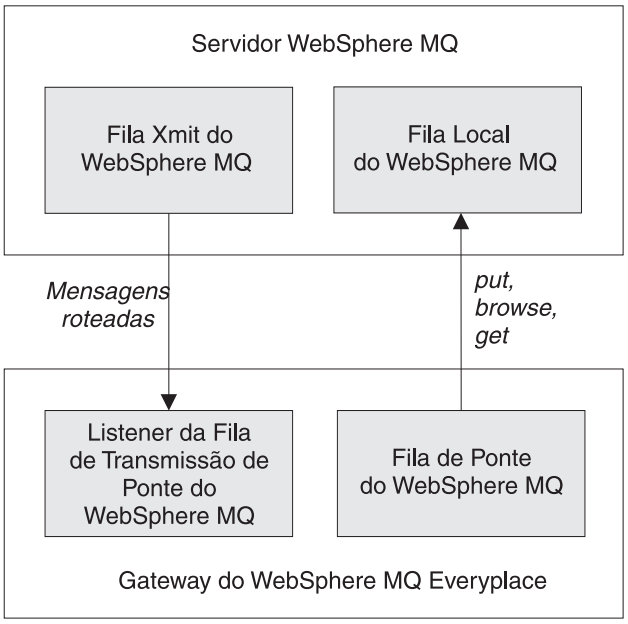


Figura 16. Configuração de Ponte

Os objetos de ponte são definidos em uma hierarquia, conforme mostrado no diagrama a seguir:

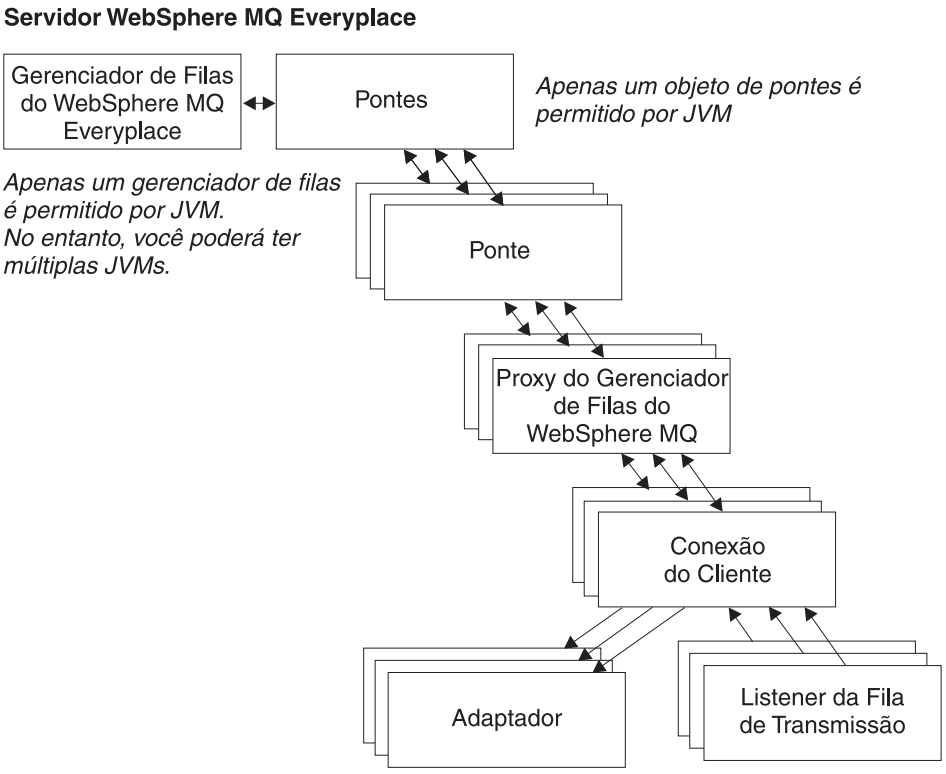


Figura 17. Hierarquia de Objetos de Ponte

As regras a seguir determinam o relacionamento entre os diversos objetos nesse diagrama:

- Um objeto de `ponte` do MQe é associado a um único gerenciador de filas do MQe.
- Um único objeto de `pontes` pode ter mais de um objeto de `ponte` associado a ele. É provável que você deseje configurar várias instâncias de pontes do MQ com rotas diferentes.

- Cada ponte pode ter várias definições de proxy do gerenciador de filas do MQ.
- Cada definição de proxy do gerenciador de filas do MQ pode ter várias conexões do cliente que permitam a comunicação com o MQe.
- Cada conexão do cliente é estabelecida com um único gerenciador de filas do MQ. Cada conexão pode utilizar uma *conexão do servidor* diferente no gerenciador de filas do MQ ou um conjunto diferente de segurança, saídas de envio e recebimento, portas ou outros parâmetros.
- Uma conexão do cliente de ponte do MQ pode ter vários listeners de fila de transmissão que utilizam esse serviço de ponte para a conexão com o gerenciador de filas do MQ.
- Um listener utiliza apenas uma conexão do cliente para estabelecer sua conexão.
- Cada listener é conectado a uma única fila de transmissão no sistema MQ.
- Cada listener move mensagens de uma única fila de transmissão MQ para qualquer lugar na rede do MQe (sua ponte é associada por meio do gerenciador de filas do MQe). Portanto, uma ponte do MQ pode convergir várias origens das mensagens do MQ por meio de um gerenciador de filas do MQe para a rede do MQe.
- Ao mover mensagens do MQe para a rede do MQ, o gerenciador de filas do MQe cria vários objetos de *adaptador*. Cada objeto de adaptador pode conectar-se a qualquer gerenciador de filas do MQ (desde que esteja configurado) e pode enviar suas mensagens para qualquer fila. Portanto, uma ponte do MQ pode distribuir mensagens do MQe, roteadas por meio de um único gerenciador de filas do MQe, para qualquer gerenciador de filas do MQ.

A opção de configuração de ponte permite que um gerenciador de filas do MQe se comunique com o host e os gerenciadores de filas distribuídos do MQ por meio de canais de clientes. O componente da ponte gerencia um conjunto de canais clientes que podem ser conectados a um ou mais hosts ou gerenciadores de filas distribuídos. Você pode configurar vários gerenciadores de filas do MQe ativados para ponte em uma única rede.

Um gateway pode ter vários listeners de fila de transmissão que utilizam esse gateway para conectar-se ao gerenciador de filas do MQ e recuperar uma mensagem do MQ para o MQe. Um listener utiliza apenas um serviço para estabelecer sua conexão, com cada listener conectando-se a uma única fila de transmissão no gerenciador de filas do MQ. Cada listener move mensagens de uma única fila de transmissão do MQ para qualquer lugar na rede do MQe, por meio de seu gerenciador de filas de gateway pai. Portanto, um único gerenciador de filas de gateway pode convergir várias origens das mensagens do MQ para a rede do MQe.

Ao mover mensagens na outra direção, do MQe para o MQ, o gerenciador de filas de gateway configura uma ou mais `filas de pontes`. Cada fila de ponte pode conectar-se a qualquer gerenciador de filas diretamente e enviar suas mensagens para a fila de destino. Dessa forma, um gateway pode distribuir mensagens do MQe roteadas por meio de um único gerenciador de filas do MQe para qualquer gerenciador de filas do MQ, direta ou indiretamente.

Objetos e Hierarquia da Ponte

Recurso de Pontes

O recurso de pontes é responsável por manter uma lista de recursos de pontes. Ele fornece um único recurso que pode ser iniciado e parado, em que iniciar e parar um recurso de pontes pode iniciar e parar todos os recursos sob ele na hierarquia de recursos. Ele pertence ao gerenciador de filas do MQe. Se o gerenciador de filas do MQe estiver ativado para ponte, um recurso de pontes será criado automaticamente e estará presente. Esse recurso não possui informações persistentes associadas a ele. Ele possui as seguintes propriedades:

Tabela 19. Propriedades de Pontes

Propriedade	Explicação
Nome da Ponte	Lista de Nomes de Pontes

Tabela 19. Propriedades de Pontes (continuação)

Propriedade	Explicação
Estado de Execução	Status: em execução ou parado

As pontes e outros recursos de pontes podem ser iniciados e parados independentemente do gerenciador de filas do MQe. Se esse recurso de ponte for iniciado (ou parado), a ação também se aplicará a todos os seus filhos, que são todas as pontes, os proxies de gerenciadores de filas, as conexões dos clientes e os listeners de fila de transmissão).

Detalhes adicionais sobre essas propriedades podem ser localizados no Java Programming Reference na classe de administração `com.ibm.mqe.mqbridge.MQeMQBridgesAdminMsg`. O recurso de ponte suporta as operações Inquire e InquireAll, start e stop. As ações create, delete e update não são apropriadas para serem utilizadas com esse recurso.

Recurso de Ponte

O recurso de ponte é responsável por manter vários valores de propriedades persistentes e uma lista de recursos de proxy do gerenciador de filas do MQ. Se iniciado ou parado, ele pode agir como um único ponto de controle para iniciar e parar todos os recursos sob ele na hierarquia de pontes. Cada objeto de ponte suporta o intervalo completo das operações create, inquire, inquire-all, update, start, stop e delete. Exemplos dessas operações podem ser localizados na classe java `examples.mqbridge.administration.programming.AdminHelperBridge`. O recurso de ponte possui as seguintes propriedades:

Tabela 20. Propriedades da Ponte

Propriedade	Explicação
Classe	Classe da ponte
Transformador Padrão	A classe padrão, classe de regra, a ser utilizada para transformar uma mensagem do MQe em MQ, ou vice-versa, se nenhuma outra classe de transformador tiver sido associada à fila de destino
Intervalo de Pulsação	A unidade básica de cronometragem a ser utilizada para desempenhar ações para as pontes
Nome	Nome da ponte
Estado de Execução	Status: em execução ou parado
Classe de regras de inicialização	Classe de regra utilizada quando a ponte é iniciada
Filhos do Proxy do Gerenciador de Filas do MQ	Lista de todos os Proxies de Gerenciadores de Filas de propriedade desta ponte

Detalhes adicionais de cada propriedade podem ser localizados no Java Programming Reference, na classe de administração `com.ibm.mqe.mqbridge.MQeMQBridgeAdminMsg`.

Em casos simples, um transformador padrão (regra) pode ser utilizado para manipular todas as conversões de mensagens. Além disso, um transformador pode ser configurado por listener (para mensagens do MQ para o MQe) que substitua esse padrão. Para um controle mais específico, as regras de transformação podem ser configuradas em uma base de fila de destino utilizando definições de filas de pontes no MQe Java Programming Reference. Isso se aplica às filas de destino do MQe e do MQ.

Proxy do Gerenciador de Filas do MQ

O proxy do gerenciador de filas do MQ mantém as propriedades específicas para um único gerenciador de filas do MQ. As propriedades do proxy são mostradas na tabela a seguir:

Tabela 21. Propriedades do Proxy do Gerenciador de Filas do MQ

Propriedade	Explicação
Classe	Classe do proxy do gerenciador de filas MQ
Nome do Host do MQ	O nome do host de IP utilizado para criar conexões para o gerenciador de filas do MQ por meio de classes do cliente Java. Se não especificado, assume-se que o gerenciador de filas do MQ esteja na mesma máquina que a ponte e as ligações Java serão utilizadas
Nome do Proxy do Gerenciador de Filas do MQ	O nome do gerenciador de filas MQ
Nome da Ponte Proprietária	Nome da ponte que possui esse proxy do gerenciador de filas do MQ
Estado de Execução	Status: em execução ou parado
Classe de regras de inicialização	Classe de regra utilizada quando o gerenciador de filas do MQ é iniciado
Filhos da Conexão do Cliente	Lista de todas as conexões do cliente de propriedade desse proxy

Detalhes adicionais de cada propriedade podem ser localizados no Java Programming Reference, na classe de administração `com.ibm.mqe.mqbridge.MQeMQQMgrProxyAdminMsg`.

Cada objeto de proxy suporta o intervalo completo das operações create, inquire, inquire-all, update, start, stop e delete. Exemplos dessas operações podem ser localizados na classe java `examples.mqbridge.administration.programming.AdminHelperMQQMgrProxy`.

Recurso de Conexão do Cliente

A definição de conexão do cliente mantém as informações detalhadas necessárias para estabelecer uma conexão com um gerenciador de filas do MQ. As propriedades da conexão são mostradas na tabela a seguir:

Tabela 22. Propriedades do Serviço de Conexão Cliente

Propriedade	Explicação
Classe do Adaptador	Classe a ser utilizada como a placa do gateway
CCSID*	O valor CCSID inteiro do MQ a ser utilizado
Classe	Classe do serviço de conexão cliente da ponte
Tempo máximo da conexão ociosa	O tempo máximo que uma conexão tem permissão para ficar ociosa antes de ser encerrada
Senha do MQ*	Senha a ser utilizada pelo cliente Java
Porta do MQ*	Número da porta de IP utilizado para criar conexões para o gerenciador de filas do MQ por meio das classes de cliente Java. Se não especificado, assume-se que o gerenciador de filas do MQ esteja na mesma máquina que a ponte e as ligações Java serão utilizadas
Classe de Saída de Recepção do MQ*	Utilizada para corresponder a saída de recepção utilizada na outra extremidade do canal cliente; a saída possui uma cadeia associada para permitir que os dados sejam passados para o código de saída
Classe de Saída de Segurança do MQ*	Utilizada para corresponder a saída de segurança utilizada na outra extremidade do canal cliente; a saída possui uma cadeia associada para permitir que os dados sejam passados para o código de saída
Classe de Saída de Envio do MQ*	Utilizada para corresponder a saída de envio utilizada na outra extremidade do canal cliente; a saída possui uma cadeia associada para permitir que os dados sejam passados para o código de saída
ID do Usuário do MQ*	ID do usuário a ser utilizado pelo cliente Java
Nome do serviço de conexão cliente	Nome do canal de conexão do servidor na máquina do MQ

Tabela 22. Propriedades do Serviço de Conexão Cliente (continuação)

Propriedade	Explicação
Nome do proxy do gerenciador de filas proprietário	O nome do proxy do gerenciador de filas proprietário
Classe de regras de inicialização	Classe de regra utilizada quando o serviço de conexão do cliente de ponte é iniciado
Nome da fila de sincronização	O nome da fila MQ que é utilizado pela ponte para propósitos de sincronização
Classe de regras de remoção da fila de sincronização	A classe de regras a ser utilizada quando uma mensagem é localizada na fila síncrona
Estado de Execução	Status: em execução ou parado
Nome da Ponte proprietária	O nome da ponte que possui essa conexão do cliente
Descendentes do Receptor MQ XmitQ	Lista de todos os listeners que utilizam essa conexão do cliente

A `classe do adaptador` é utilizada para enviar mensagens do MQe para o MQ e a `fila de sincronização` é utilizada para acompanhar o status desse processo. Seu conteúdo é utilizado na recuperação de situações para garantir as mensagens; depois de um encerramento normal a fila fica vazia. Ela pode ser compartilhada através de várias conexões clientes e através de várias definições de pontes, contanto que as saídas de recepção, envio e segurança sejam as mesmas. Essa fila também pode ser utilizada para armazenar o estado sobre mensagens que são movidas do MQ para o MQe, dependendo das propriedades do listener em uso. A `classe de regras de remoção da fila síncrona` é utilizada quando uma mensagem é localizada na fila síncrona, indicando um defeito do MQe para confirmar uma mensagem.

O tempo máximo de conexão ociosa é utilizado para controlar o conjunto de conexões do cliente Java mantido pelo serviço de conexão do cliente de ponte em seu sistema MQ. Quando uma conexão do MQ se tornar ociosa pela falta de uso, um cronômetro será iniciado e a conexão ociosa será descartada se ele expirar antes da conexão ser reutilizada. A criação de conexões do MQ é uma operação cara e esse processo assegura que elas sejam utilizadas com eficiência sem consumir recursos excessivos. Um valor zero indica que não se deve utilizar um conjunto de conexões.

Detalhes adicionais de cada propriedade podem ser localizados no Java Programming Reference, na classe de administração `com.ibm.mqe.mqbridge.MQeClientConnectionAdminMsg`.

Cada objeto de conexão do cliente suporta o intervalo completo de operações create, inquire, inquire-all, update, start, stop e delete. Exemplos dessas operações podem ser localizados na classe Java `examples.mqbridge.administration.programming.AdminHelperMQClientConnection`.

Recurso de Listener de Fila de Transmissão

O listener move mensagens do MQ para o MQe.

Tabela 23. Propriedades do Listener

Propriedade	Explicação
Classe	Classe do listener
Nome da Fila de Devoluções	Fila utilizada para manter mensagens do MQ para o MQe que não podem ser entregues
Placa de armazenamento de estado do receptor	Nome da classe da placa utilizada para armazenar informações sobre o estado
Nome do Listener	Nome da fila XMIT do MQ que fornece mensagens
Nome do serviço de conexão cliente proprietário	Nome do serviço de conexão cliente

Tabela 23. Propriedades do Listener (continuação)

Propriedade	Explicação
Estado de execução	Status: em execução ou parado
Classe de regras de inicialização	Classe de regra utilizada quando o listener é iniciado
Classe do transformador	Classe de regra utilizada para determinar a conversão de uma mensagem do MQ para o MQe
Classe de regras de mensagem não entregue	Classe de regra utilizada para determinar ação quando as mensagens do MQ para o MQe não podem ser entregues
Segundos de espera pela mensagem	Uma opção avançada que pode ser utilizada para controlar o desempenho do receptor em circunstâncias excepcionais

Detalhes adicionais de cada propriedade podem ser localizados no Java Programming Reference, na classe de administração `com.ibm.mqe.mqbridge.MQeListenerAdminMsg`.

Cada objeto de listener de fila de transmissão suporta o intervalo completo das operações create, inquire, inquire-all, update, start, stop e delete. Exemplos dessas operações podem ser localizados na classe Java `examples.mqbridge.administration.programming.AdminHelperMQTransmitQueueListener`

A `classe de regra de mensagem não entregue` determina qual ação é tomada quando uma mensagem do MQ para o MQe não pode ser entregue. Normalmente, ela é colocada na `fila de devoluções` do sistema MQ.

Para fornecer entrega garantida de mensagens, a classe de listener utiliza o `adaptador de armazenamento de estado do listener` para armazenar informações de estado no sistema MQe ou na fila síncrona do sistema MQ.

O listener de fila de transmissão permite que filas remotas do MQ se refiram às filas locais do MQe. Você também pode criar filas remotas do MQe que se referem às filas locais do MQ. Essas definições de filas remotas do MQe são chamadas de `filas de pontes do MQ` e podem ser utilizadas para receber, enviar e procurar mensagens em uma fila do MQ.

Fila de Ponte

Uma definição de fila de ponte do MQ pode conter os atributos a seguir.

Tabela 24. Propriedades de Filas de Ponte do MQ

Propriedade	Explicação
Nomes de alias	Nomes alternativos para a fila
Autenticador	Deve ser nulo
Classe	Classe de objeto
Conexão do Cliente	Nome do serviço de conexão cliente a ser utilizado
Compressor	Deve ser nulo
Criptografador	Deve ser nulo
Expiração	Passado para o transformador
Tamanho máximo da mensagem	Passado para a classe de regras
Modo	Deve ser síncrono
Proxy do gerenciador de filas do MQ	Nome do gerenciador de filas do MQ para o qual a mensagem deve ser enviada primeiro
Ponte do MQ	Nome da ponte para transportar a mensagem para o MQ
Nome	Nome pelo qual a fila remota do MQ é conhecida pelo MQe

Tabela 24. Propriedades de Filas de Ponte do MQ (continuação)

Propriedade	Explicação
Gerenciador de filas proprietário	Gerenciador de filas que possui a definição
Prioridade	Prioridade a ser utilizada para as mensagens, a não ser que substituídas por um valor de mensagem
Nome da Fila Remota do MQ	Nome da fila remota do MQ
Regra	Classe de regras utilizada para operações de filas
Gerenciador de filas de destino	O gerenciador de filas do MQ que possui a fila
Transformador	Nome da classe de transformador que converte a mensagem do formato do MQe para o formato do MQ
Tipo	Fila de Ponte do MQ

Detalhes adicionais de cada propriedade podem ser localizados no Java Programming Reference, na classe de administração `com.ibm.mqe.mqbridge.MQeMQBridgeQueueAdminMsg`.

O código de exemplo que manipula uma fila de ponte pode ser localizado na classe Java `examples.mqbridge.administration.programmingAdminHelperBridgeQueue`.

Nota: As classes criptografador, autenticador e compactador definem um conjunto de atributos das filas que indicam o nível de segurança de qualquer mensagem transmitida a essa fila. No MQe, do momento em que a mensagem é enviada inicialmente até o momento em que a mensagem é transmitida à fila de ponte do MQ, a mensagem é protegida com, pelo menos, o nível de segurança da fila. Esses níveis de segurança *não* são aplicáveis quando a fila de ponte do MQ transmite a mensagem para o sistema MQ, as saídas de envio e de recebimento da segurança são utilizadas na conexão do cliente durante essa transferência. Não são feitas verificações para certificar-se de que o nível de segurança da fila foi mantido.

As filas de pontes do MQ são apenas síncronas. Portanto, os aplicativos assíncronos devem utilizar uma combinação de filas de armazenamento e redirecionamento e de servidor home do MQe ou definições de filas remotas assíncronas como uma etapa intermediária ao enviar mensagens para as filas de pontes do MQ.

Os aplicativos utilizam as filas de pontes do MQ como qualquer outra fila remota do MQe, utilizando os métodos `putMessage`, `browseMessages` e `getMessage` da classe MQeQueueManager. O parâmetro de nome da fila nessas chamadas é o nome da fila de ponte do MQ e o parâmetro de nome do gerenciador de filas é o nome do gerenciador de filas do MQ. No entanto, para que esse nome do gerenciador de filas seja aceito pelo servidor MQe local, uma definição de conexão com esse nome do gerenciador de filas do MQ deve existir com nulo para todos os parâmetros, incluindo o nome do canal.

Nota: Existem algumas restrições no uso de getMessage e browseMessages com as filas de pontes do MQ. Não é possível receber ou procurar mensagens a partir de filas de pontes do MQ que apontam para as definições de filas remotas do MQ. Também não é possível utilizar IDs de Confirmação diferentes de zero nos recebimentos de filas de pontes do MQ. Isso significa que a operação getMessage em filas de pontes do MQ não fornece entrega garantida. Se você precisar que uma operação de recebimento seja garantida, deverá utilizar os listeners de fila de transmissão para transferir mensagens do MQ.

A administração da ponte do MQ é manipulada da mesma forma que a administração de um gerenciador de filas normal do MQe, utilizando mensagens de administração. Novas classes de mensagens são definidas conforme apropriado para a fila.

Recomendações de Nomenclatura para Interoperabilidade com o MQ

Para criar uma rede do MQe que possa interoperar com uma rede do MQ e evitar problemas, adote as mesmas limitações de convenção de nomenclatura para ambos os sistemas. As seguintes diferenças são relevantes:

- Os nomes das filas e dos gerenciadores de filas do MQ podem ter um caractere de barra (/). Esse caractere não é válido em nomes de objetos do MQe. Não utilize esse caractere no nome de qualquer fila ou gerenciador de filas do MQ.
- Os nomes das filas e dos gerenciadores de filas do MQ possuem um limite de 48 caracteres, mas os nomes do MQe não possuem restrições de comprimento. Não defina filas ou gerenciadores de filas do MQe com nomes que contenham mais de 48 caracteres.
- Os nomes das filas do MQ podem ter um caractere de ponto à direita ou esquerda (.). Isso não é permitido no MQe. Não defina nenhuma fila ou gerenciador de filas do MQ com um nome que inicie ou termine com esse caractere.
- Forneça nomes exclusivos aos gerenciadores de filas, de modo que não exista um gerenciador de filas com o mesmo nome na rede do MQe e na rede do MQ.

Se você optar por não seguir essas orientações, poderá se deparar com problemas ao tentar endereçar uma fila do MQe a partir de um aplicativo do MQ.

Configurando uma Ponte Básica do MQ

Para configurar uma instalação bem básica da ponte do MQ, conclua as seguintes etapas:

1. Certifique-se de que um sistema MQ esteja instalado e que você conheça as convenções de rota local e como configurar o sistema.
2. Instale o MQe em um sistema (pode ser no mesmo sistema que o MQ)
3. Se o MQ Classes para Java ainda não estiver instalado, faça download dele na Web e instale-o no sistema MQ.
4. Configure o sistema MQe e verifique se ele está funcionando corretamente antes de tentar conectá-lo ao MQ.
5. Atualize o arquivo MQe_java\Classes\JavaEnv.bat de modo que aponte para as classes Java que fazem parte do MQ Classes para Java e para o caminho de classe de seu JRE (Java Runtime Environment). Assegure-se de que os arquivos .jar do SupportPac MA88 estejam no caminho de classe, e que os diretórios java\lib e \bin estejam em seu caminho. Isso é configurado pelo `MQE_VM_OPTIONS_LOCN` que deve ser configurado para apontar para o arquivo vm_options.txt durante a instalação.
6. Planeje a rota que você pretende implementar. É necessário decidir quais gerenciadores de filas do MQ trocarão informações com quais gerenciadores de filas do MQe.
7. Determine a convenção de nomenclatura de objetos do MQe e do MQ e documente-a para uso futuro.
8. Modifique o sistema MQe para definir uma ponte do MQ em seu servidor MQe escolhido. Consulte o Java Programming Reference para obter informações sobre como utilizar o examples.mqbridge.awt.AwtMQBridgeServer para definir uma ponte.
9. Conecte o gerenciador de filas do MQ escolhido com a ponte no servidor MQe, conforme a seguir:
   - No gerenciador de filas do MQ:
     - Defina uma ou mais conexões do servidor Java de modo que o MQe possa utilizar o MQ Classes paraJava para trocar informações com esse gerenciador de filas. Isso envolve as seguintes etapas:
       a. Defina as conexões do servidor
       b. Defina uma fila síncrona para ser utilizada pelo MQe a fim de fornecer entrega assegurada para o sistema MQ. É necessária uma dessas para cada conexão do servidor que possa ser utilizada pelo sistema MQe.

- No servidor MQe:
  a. Defina um objeto proxy do gerenciador de filas do MQ que mantenha informações sobre o gerenciador de filas do MQ. Isso envolve as seguintes etapas:
     1) Colete o Nome do Host do gerenciador de filas do MQ.
     2) Envie o nome para o objeto proxy do gerenciador de filas do MQ.
  b. Defina um objeto de Conexão do Cliente que mantenha informações sobre como utilizar o MQ Classes for Java para conectar-se à conexão do servidor no sistema MQ. Isso envolve as seguintes etapas:
     1) Colete o Número da Porta e todos os outros parâmetros de conexão do servidor.
     2) Utilize esses valores para definir o objeto de conexão do cliente de modo que correspondam à definição no gerenciador de filas do MQ.

10. Modifique a configuração no MQe e no MQ para permitir que as mensagens sejam transmitidas do MQ para o MQe.
    a. Determine o número de rotas do MQ para a rede do MQe. O número de rotas necessário depende da quantidade do tráfego (carregamento) de mensagens utilizado em cada rota. Se o carregamento de mensagens for alto, é provável que você deseje dividir o tráfego em várias rotas.
    b. Defina as rotas da seguinte forma:
       1) Para cada rota, defina uma fila de transmissão em seu sistema MQ. NÃO defina uma conexão para essas filas de transmissão.
       2) Para cada rota, crie um listener de fila de transmissão correspondente no sistema MQe.
       3) Defina várias definições de filas remotas (como aliases de gerenciadores de filas remotas e aliases de filas) para permitir que as mensagens do MQ sejam roteadas para as diversas filas de transmissão no limite do MQe que você definiu na etapa b. 1.
11. Modifique a configuração no MQe para permitir que as mensagens sejam transmitidas do MQe para o MQ:
    a. Publique detalhes sobre todos os gerenciadores de filas na rede do MQ para os quais você deseja enviar mensagens a partir da rede do MQe. Cada gerenciador de filas do MQ requer uma definição de conexão em seu servidor MQe. Todos os campos, exceto o Nome do Gerenciador de Filas, devem ser nulos para indicar que as conexões de comunicações normais do MQe não devem ser utilizadas para a troca de informações com esse gerenciador de filas.
    b. Publique detalhes sobre todas as filas na rede do MQ para as quais você deseja enviar mensagens a partir da rede do MQe. Cada fila do MQ requer uma definição de fila de ponte do MQ no servidor MQe. Isso é o equivalente no MQe a um `DEFINE QREMOTE` no MQ.
       - O nome da fila é o nome da fila do MQ para a qual a ponte deve enviar quaisquer mensagens que chegam a essa fila de ponte do MQ.
       - O nome do gerenciador de filas é o nome do gerenciador de filas do MQ no qual a fila está localizada.
       - O nome da ponte indica qual ponte deve ser utilizada para enviar mensagens para a rede do MQ.
       - O nome do proxy do gerenciador de filas do MQ é o nome do objeto proxy do gerenciador de filas do MQ, na configuração do MQe, que pode conectar-se a um gerenciador de filas do MQ.
       - O gerenciador de filas do MQ deve ter uma rota definida para permitir que as mensagens sejam divulgadas para o `Nome da Fila` em `Nome do Gerenciador de Filas` para entregar a mensagem em seu destino final.
12. Inicie os sistemas MQ e MQe para permitir a circulação de mensagens. O listener do canal de clientes do sistema MQ deve ser iniciado. Todos os objetos definidos no MQe devem ser iniciados. Esse objetos podem ser iniciados de uma das seguintes formas:
    - Explicitamente, utilizando a GUI de Administração descrita no MQe Configuration Guide.
    - Configurando a classe de regras, conforme descrito no MQe System Programming Guide, para indicar o estado de inicialização (em execução ou parado) e reiniciando o servidor MQe

- Uma mistura dos dois métodos anteriores

O modo mais simples de iniciar objetos manualmente é enviar um comando start para o objeto de ponte relevante. Esse comando deve indicar que todos os filhos da ponte, bem como os filhos deles, devem ser iniciados.

- Para permitir que as mensagens sejam transmitidas do MQe para o MQ, inicie os objetos de conexão do cliente no MQe.
- Para permitir que as mensagens sejam transmitidas do MQ para o MQe, inicie ambos os objetos de conexão do cliente e os listeners relevantes da fila de transmissão.

13. Crie classes de transformador e modifique a configuração do MQe para utilizá-las. Uma classe de transformador converte mensagens do formato da mensagem do MQ em um formato da mensagem do MQe e vice-versa. Esses conversores de formatos devem ser gravados em Java e configurados em vários locais na configuração de ponte do MQ.
    a. Crie classes de transformador
        - Determine os formatos das mensagens do MQ que precisam ser transmitidas através da ponte.
        - Grave uma classe de transformador ou um conjunto de classes de transformador para converter cada formato da mensagem do MQ em uma mensagem do MQe. Os transformadores não são suportados diretamente pelas ligações em C.
    b. É possível substituir a classe de transformador padrão. Utilize a GUI de administração para atualizar o parâmetro de classe de transformador padrão na configuração do objeto de ponte.
    c. É possível especificar um transformador não padrão para cada definição de fila de ponte do MQ. Utilize a GUI de administração para atualizar o campo *transformador* de cada fila de ponte do MQ na configuração.
    d. É possível especificar um transformador não padrão para cada listener de fila de transmissão do MQ. Utilize a GUI de administração para atualizar o campo de *transformador* de cada listener na configuração.
    e. Reinicie a ponte e os listeners.

Utilizando Mensagens de Administração do MQe e Mensagens PCF do MQ

As mensagens PCF são mensagens de administração utilizadas por gerenciadores de filas MQ. O SupportPac MS0B: ″MQSeries Java classes para PCF″ contém código Java, que fornece suporte a mensagens PCF. Esse código está disponível como um download gratuito.

Se você fizer download dele e instalá-lo e inserir o arquivo `com.ibm.mq.pcf.jar` em sua variável de ambiente ClassPath, terá acesso a classes Java, que podem manipular dinamicamente os recursos do MQ. Quando as mensagens do PCF são combinadas com mensagens de administração do MQe, a configuração programática completa dos recursos de ponte e os recursos correspondentes em um gerenciador de filas do MQe são possíveis. O código de exemplo contido na classe `examples.mqbridge.administration.programming.AdminHelperMQ`, utilizado em conjunto com `examples.mqbridge.administration.programming.MQAgent`, demonstra como fazer isso. Esse código de exemplo foi incluído no programa `examples.awt.AwtMQeServer`, de modo que clicar no menu **Visualizar->Conectar Gerenciador de Filas Locais Padrão do MQ**, será possível:

- Assegurar que exista um objeto de ponte, criando um quando necessário.
- Consultar propriedades do gerenciador de filas padrão do MQ.
- Tentar conectar esse gerenciador de filas ao gerenciador de filas do MQe atualmente em execução.
- Assegurar que exista um objeto proxy que represente o gerenciador de filas padrão do MQ, criando um quando necessário.
- Assegurar que exista uma conexão do cliente MQe e que também exista um canal de conexão do servidor MQ correspondente, criando esses recursos quando necessário.
- Assegurar que exista uma 'fila síncrona' no gerenciador de filas do MQ.

- Assegurar que exista uma fila de transmissão no MQ, criando uma quando necessário.
- Assegurar que exista um listener correspondente de fila de transmissão do MQ na configuração atual do gerenciador de filas do MQe, criando um quando necessário.
- Assegurar que todos os recursos de ponte sejam iniciados.
- Assegurar que exista uma fila de teste no gerenciador de filas do MQ, criando uma quando necessário.
- Assegurar que exista uma fila de ponte do MQe correspondente, que faça referência a essa fila de teste.
- Enviar um MQeMQMsgObject de teste para a fila de teste a fim de certificar-se de que a configuração esteja funcionando.
- Receber o MQeMQMsgObject da fila de teste para certificar-se de que a configuração esteja funcionando.

Exemplo de Configuração de Ponte

Esta seção descreve uma configuração de exemplo de 4 sistemas.

Requisito

O requisito para esse exemplo é que todas as máquinas estejam aptas a enviar uma mensagem para uma fila em qualquer uma de três outras máquinas.

Como premissa, todas as máquinas devem estar conectadas permanentemente à rede, exceto a máquina MQeMoonQM, que é conectada apenas ocasionalmente.

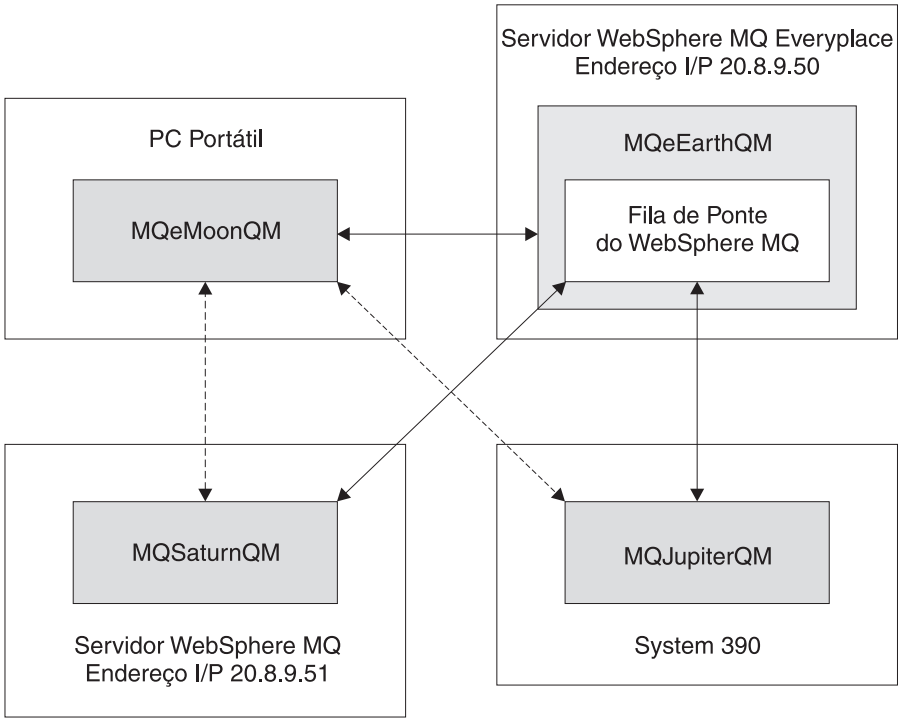


Figura 18. Exemplo de Configuração

Os quatro sistemas são:

MQeMoonQM
Este é um gerenciador de filas do cliente MQe, localizado em um computador de mão. O usuário conecta periodicamente o computador de mão à rede para se comunicar com o gateway MQeEarthQM do MQe.

MQeEarthQM
Está em uma máquina Windows 2000, com um endereço IP igual a 20.8.9.50. Esse é um gerenciador de filas do gateway (servidor) do MQe.

MQSaturnQM
Esse é um gerenciador de filas do MQ, instalado em uma plataforma Windows NT. O endereço IP é 20.8.9.51

MQJupiterQM
Esse é um gerenciador de filas do MQ, instalado em uma plataforma System/390.

Configuração Inicial

Para este exemplo, a premissa é que existam filas locais, para as quais as mensagens podem ser enviada, em todos os gerenciadores de filas. Essas filas são denominadas:

- `MQeMoonQ` em `MQeMoonQM`
- `MQeEarthQ` em `MQeEarthQM`
- `MQSaturnQ` em `MQSaturnQM`
- `MQJupiterQ` em `MQJupiterQM`

Agora qualquer aplicativo conectado a qualquer um dos gerenciadores de filas pode enviar uma mensagem para qualquer uma das filas, `MQeMoonQ`, `MQeEarthQ`, `MQSaturnQ` ou `MQJupiterQ`.

MQeMoonQM para/de MQeEarthQM

No MQeMoonQM:

1. Defina uma **conexão** com os seguintes parâmetros:

   Nome do gerenciador de filas de destino: `MQeEarthQM`
   Adaptador: `FastNetwork:20.8.9.50`

   Nota: Verifique se o adaptador especificado ao definir a conexão corresponde ao adaptador utilizado pelo Listener no gerenciador de filas MQeEarthQM.
   Os aplicativos agora podem conectar-se diretamente a qualquer fila definida no gerenciador de filas `MQeEarthQM`, quando o `MQeMoonQM` estiver conectado à rede. O requisito determina que os aplicativos no `MQeMoonQM` devem estar aptos a enviar mensagens para a `MQeEarthQ` de modo assíncrono. Isso requer uma definição de fila remota para configurar a ligação assíncrona para a fila `MQeEarthQ`.
2. Defina uma **fila remota** com os seguintes parâmetros:

   Nome da fila: `MQeEarthQ`
   Nome do gerenciador de filas: `MQeEarthQM`
   Modo de acesso: `Assíncrono`

   Os aplicativos no `MQeMoonQM` agora têm acesso à `MQeMoonQ` (uma fila local) de modo síncrono, e à `MQeEarthQ` de modo assíncrono.

MQeEarthQM para MQeMoonQM

Como o `MQeMoonQM` não está conectado à rede na maior parte do tempo, utilize uma fila de armazenamento e redirecionamento no `MQeEarthQM` para coletar mensagens destinadas ao gerenciador de filas `MQeMoonQM`.

No MQeEarthQM:

1. Defina uma **fila de armazenamento e redirecionamento** com os seguintes parâmetros:

   Nome da Fila: `TO.HANDHELDS`
   Nome do gerenciador de filas: `MQeEarthQM`

2. Inclua um **gerenciador de filas** na **fila de armazenamento e redirecionamento** utilizando os seguintes parâmetros:

   Nome da fila: `TO.HANDHELDS`
   Gerenciador de Filas: `MQeMoonQM`

Uma fila de servidor home (nomeada similarmente) é necessária no gerenciador de filas `MQeMoonQM`. Essa fila recebe as mensagens da fila de armazenamento e redirecionamento e as envia para uma fila no gerenciador de filas `MQeMoonQM`.

No MQeMoonQM:

1. Defina uma **fila de servidor home** com os seguintes parâmetros:

   Nome da Fila: `TO.HANDHELDS`
   Nome do gerenciador de filas: `MQeEarthQM`

Quaisquer mensagens que cheguem ao `MQeEarthQM`, destinadas ao `MQeMoonQM`, são armazenadas temporariamente na fila de armazenamento e redirecionamento `TO.HANDHELDS`. Na próxima conexão do `MQeMoonQM` com a rede, sua fila de servidor home `TO.HANDHELDS` recebe as mensagens atualmente na fila de armazenamento e redirecionamento e as entrega ao gerenciador de filas `MQeMoonQM` para armazenamento nas filas locais.

Agora os aplicativos no `MQeEarthQM` podem enviar mensagens para a `MQeMoonQ` de um modo assíncrono.

MQeEarthQM para MQSaturnQ

No MQeEarthQM:

1. Defina uma **ponte** com os seguintes parâmetros:

   Nome da ponte: `MQeEarthQMBridge`

2. Defina um **proxy do gerenciador de filas do MQ** com os seguintes parâmetros:

   Nome da Ponte: `MQeEarthQMBridge`
   Nome do Proxy do Gerenciador de Filas do MQ: `MQSaturnQM`
   Nome do Host: `20.8.9.51`

3. Defina uma **conexão do cliente** com os seguintes parâmetros:

   Nome da Ponte: `MQeEarthQMBridge`
   Nome do Proxy do Gerenciador de Filas do MQ: `MQSaturnQM`
   ClientConnectionName: `MQeEarth.CHANNEL`
   SyncQName: `MQeEarth.SYNC.QUEUE`

4. Defina uma **conexão** com os seguintes parâmetros:

   Nome da conexão: `MQSaturnQM`
   Canal: `null`
   Adaptador: `null`

5. Defina uma **fila de ponte do MQ** com os seguintes parâmetros:

   Nome da fila: `MQSaturnQ`
   Nome do gerenciador de filas do MQ: `MQSaturnQM`
   Nome da ponte: `MQeEarthQMBridge`
   Nome do proxy QMgr do MQ: `MQSaturnQM`
   Nome da conexão do cliente: `MQeEarth.CHANNEL`

No MQSaturnQM:

1. Defina um **canal de conexão do servidor** com os seguintes parâmetros:

Nome: MQeEarth.CHANNEL

2. Defina uma **fila síncrona local** com os seguintes parâmetros:

   Nome: MQeEarth.SYNC.QUEUE

   A fila síncrona é necessária para uma entrega assegurada.

Agora os aplicativos no MQeEarthQM podem enviar mensagens para a MQSaturnQ no MQSaturnQM.

MQeEarthQM para MQJupiterQ

No MQeEarthQM:

1. Defina uma **conexão** com os seguintes parâmetros:

   Nome da conexão: MQeJupiterQM
   Canal: null
   Adaptador: null

2. Defina uma **fila de ponte do MQ** com os seguintes parâmetros:

   Nome da fila: MQJupiterQ
   Nome do gerenciador de filas do MQ: MQJupiterQM
   Nome da ponte: MQeEarthQMBridge
   Nome do proxy QMgr do MQ: MQSaturnQM
   Nome da conexão do cliente: MQeEarth.CHANNEL

No MQSaturnQM:

1. Defina uma **definição de fila remota** com os seguintes parâmetros:

   Nome da Fila: MQJupiterQ
   Fila de Transmissão: MQJupiterQM.XMITQ

No MQSaturnQM e no MQJupiterQM:

1. Defina um **canal** para mover a mensagem do MQJupiterQM.XMITQ no MQSaturnQM para MQJupiterQM.

Agora os aplicativos no MQeEarthQM podem enviar uma mensagem para a MQJupiterQ no MQJupiterQM, por meio do MQSaturnQM.

MQeMoonQM para MQJupiterQ e MQSaturnQ

No MQeMoonQM:

1. Defina uma **conexão** com os seguintes parâmetros:

   Nome do gerenciador de filas de destino: MQSaturnQM
   Adaptador: MQeEarthQM

   A conexão indica que qualquer mensagem ligada ao gerenciador de filas MQSaturnQM deve passar pelo gerenciador de filas MQeEarthQM.

2. Defina uma **definição de fila remota** com os seguintes parâmetros:

   Nome da Fila: MQSaturnQ
   Nome do Gerenciador de Filas: MQSaturnQM
   Modo de Acesso: Assíncrono

3. Defina uma **conexão** com os seguintes parâmetros:

   Nome do gerenciador de filas de destino: MQJupiterQM
   Adaptador: MQeEarthQM

4. Defina uma **definição de fila remota** com os seguintes parâmetros:

   Nome da Fila: MQJupiterQ
   Nome do Gerenciador de Filas: MQJupiterQM
   Modo de Acesso: Assíncrono

Agora os aplicativos conectados ao MQeMoonQM podem emitir mensagens para a MQeMoonQ, MQeEarthQ, MQSaturnQ e MQJupiterQ, mesmo quando o computador de mão está desconectado da rede.

MQSaturnQM para MQeEarthQ

No MQSaturnQM:

1. Defina uma **fila local** com os seguintes parâmetros:

   Nome da fila: MQeEarth.XMITQ
   Tipo da fila: transmission queue
2. Defina um **alias do gerenciador de filas** (definição de fila remota) com os seguintes parâmetros:

   Nome da Fila: MQeEarthQM
   Nome do gerenciador de filas remotas: MQeEarthQM
   Fila de Transmissão: MQeEarth.XMITQ

No MQeEarthQM:

1. Defina um **listener de fila de transmissão** com os seguintes parâmetros:

   Nome da Ponte: MQeEarthQMBridge
   Nome do Proxy do Gerenciador de Filas do MQ: MQSaturnQM
   ClientConnectionName: MQeEarth.CHANNEL
   Nome do Listener: MQeEarth.XMITQ

Agora os aplicativos no MQSaturnQM podem enviar mensagens para a MQeEarthQ utilizando o alias do gerenciador de filas MQeEarthQM. Este roteia todas as mensagens para a MQeEarth.XMITQ, na qual o listener de fila de transmissão do MQe MQeEarth.XMITQ as recebe e as move para a rede do MQe.

MQSaturnQM para MQeMoonQ

No MQSaturnQM:

1. Defina um **alias do gerenciador de filas** (definição de fila remota) com os seguintes parâmetros:

   Nome da Fila: MQeMoonQM
   Nome do gerenciador de filas remotas: MQeMoonQM
   Fila de Transmissão: MQeEarth.XMITQ

Agora os aplicativos no MQSaturnQM podem enviar mensagens para a MQeMoonQ utilizando o alias do gerenciador de filas MQeMoonQM. Este roteia todas as mensagens para a MQeEarth.XMITQ, na qual o listener de fila de transmissão do MQe MQeEarth.XMITQ as recebe e as envia para a rede do MQe.

A fila de armazenamento e redirecionamento TO.HANDHELDS coleta a mensagem e, na próxima conexão do MQeMoonQM com a rede, a fila de servidor home recupera a mensagem da fila de armazenamento e redirecionamento e a entrega para a MQeMoonQ.

MQJupiterQM para MQeMoonQ

No MQJupiterQM:

Configure **aliases de gerenciador de filas remotas** para o MQeEarthQM e MQeMoonQM para rotear mensagens para o MQSaturnQM utilizando técnicas normais de rotas do MQ.

Administração da Ponte

Ações de Administração da Ponte

Estado de Execução: Cada objeto administrado possui um *estado de execução*. Esse estado pode ser *em execução* ou *parado*, indicando se o objeto administrado está ativo ou não.

Quando um objeto administrado está *parado*, ele não pode ser utilizado, mas seus parâmetros de configuração podem ser consultados ou atualizados.

Se a fila de ponte do MQ referir-se a um objeto administrado pela ponte que esteja *parado*, não será possível transportar uma mensagem do MQe para a rede do MQ até que os objetos de ponte, de proxy do gerenciador de filas do MQ e de conexão do cliente estejam todos *iniciados*.

O estado de execução de objetos administrados pode ser alterado utilizando as ações `start` e `stop` a partir das classes de mensagens de administração `MQeMQBridgeAdminMsg`, `MQeMQQMgrProxyAdminMsg`, `MQeClientConnectionAdminMsg` ou `MQeListenerAdminMsg`.

Ação Start: Um administrador pode enviar uma ação `start` para qualquer um dos objetos administrados.

O sinalizador booleano `affect children` afeta os resultados dessa ação:

- A ação `start` iniciará o objeto administrado e todos os seus filhos (e filhos dos filhos) se o campo booleano `affect children` estiver na mensagem e estiver configurado como `true`.
- Se o sinalizador não estiver na mensagem ou estiver configurado como `false`, apenas o objeto administrado que receber a ação start alterará seu estado de execução.

Por exemplo, o envio de `start` em um objeto de ponte com `affect children` como `true` faz com que todos os proxies, conexões do cliente e listeners ascendentes sejam iniciados. Se `affect children` não for especificado, apenas a ponte será iniciada. Não é possível iniciar um objeto a menos que seu objeto pai já tenha sido iniciado. O envio de um evento de início a um objeto administrado tenta iniciar todos os objetos superiores na hierarquia que ainda não estão em execução.

Ação Stop: É possível parar um objeto administrado enviando a ele uma ação `stop`. O objeto administrado receptor sempre certifica-se de que todos os objetos abaixo dele na hierarquia tenham sido parados antes de parar a si mesmo.

Ação Inquire: A ação `inquire` consulta valores de um objeto administrado.

Se o objeto administrado estiver em execução, os valores retornados na consulta serão aqueles atualmente em uso.

Os valores retornados de um objeto parado refletem quaisquer alterações recentes nos valores feitas por uma ação `update`.

Portanto, uma seqüência de:

```
start, update, inquire
```

retorna os valores configurados *antes* da atualização,

Uma seqüência de:

```
start, update, stop, inquire
```

retorna os valores configurados *após* a atualização.

Pode ser menos confuso se você parar qualquer objeto administrado antes de atualizar os valores de variáveis.

Ação Update: A ação `update` altera um ou mais valores de características de um objeto administrado. Os valores configurados por uma ação `update` serão atualizados somente na próxima vez em que o objeto administrado for parado. (Consulte "Ação Inquire" na página 102.)

Ação Delete: A ação `delete` remove permanentemente todas as informações atuais e persistentes sobre o objeto administrado. O sinalizador booleano `affect children` afeta o resultado dessa ação. Se o sinalizador `affect children` estiver presente e configurado como `true`, o objeto administrado que receber essa ação emitirá uma ação `stop` e, em seguida, uma ação `delete` para todos os objetos abaixo dele na hierarquia, removendo uma parte inteira da hierarquia com uma única ação. Se o sinalizador não estiver presente ou estiver configurado como `false` o objeto administrado excluirá a si próprio, mas essa ação não poderá ocorrer a menos que todos os objetos na hierarquia abaixo da atual já tenham sido excluídos.

Ação Create: A ação `create` cria um objeto administrado. O estado de execução do objeto administrado criado é configurado inicialmente como `stopped`.

Considerações sobre a Ponte ao Parar um Gerenciador de Filas do MQ

Antes de parar um gerenciador de filas do MQ, emita uma mensagem de administração `stop` para todos os objetos de ponte do proxy do gerenciador de filas do MQ. Com isso, a rede do MQe pára de tentar utilizar o gerenciador de filas do MQ e, possivelmente, de interferir no encerramento do gerenciador de filas do MQ. Isso também pode ser efetuado emitindo uma única mensagem de administração `stop` para o objeto MQeBridges.

Se você optar por não parar o objeto de ponte do proxy do gerenciador de filas do MQ antes de encerrar o gerenciador de filas do MQ, o comportamento do encerramento do MQ e da ponte do MQ dependerá do tipo de encerramento escolhido para o gerenciador de filas do MQ, encerramento imediato ou encerramento controlado.

Encerramento Imediato: Parar um gerenciador de filas do MQ utilizando o encerramento imediato interrompe quaisquer conexões que a ponte do MQ possui com o gerenciador de filas do MQ (isso aplica-se a conexões formadas utilizando o MQSeries Classes para Java no modo de ligações ou de cliente). O sistema MQ é encerrado normalmente.

Isso faz com que todos os listeners da fila de transmissão de ponte do MQ parem imediatamente, cada um deles avisando que foi encerrado devido à parada do gerenciador de filas do MQ.

Quaisquer filas de pontes do MQ ativas retêm uma conexão interrompida para o gerenciador de filas do MQ até que:

- A conexão tenha o tempo esgotado, depois de ficar inativa durante um período de tempo limite, conforme especificado no objeto de ponte de conexão do cliente, em cujo ponto a conexão interrompida é fechada.
- A fila de ponte do MQ seja informada para desempenhar alguma ação, como enviar uma mensagem para o MQ, que tenta utilizar a conexão interrompida. A operação `putMessage` falha e a conexão interrompida é fechada.

Quando uma fila de ponte do MQ não possuir conexão, a próxima operação nessa fila fará com que uma nova conexão seja obtida. Se o gerenciador de filas do MQ não estiver disponível, a operação na fila falhará sincronicamente. Se o gerenciador de filas do MQ tiver sido reiniciado após o encerramento e uma operação de fila, como `putMessage`, agir sobre a fila de ponte, uma nova conexão com o gerenciador de filas do MQ ativo será estabelecida e a operação será executada no modo esperado.

Encerramento Controlado: Parar um gerenciador de filas do MQ utilizando o encerramento controlado não interrompe nenhuma conexão imediatamente, mas espera até que todas as conexões sejam fechadas (isso aplica-se a conexões formadas utilizando o MQSeries Classes para Java no modo de ligações ou de cliente). Quaisquer listeners ativos da fila de transmissão de ponte do MQ observam que o sistema MQ está em quiesce e param com um aviso relevante.

Quaisquer filas de pontes do MQ ativas retêm uma conexão para o gerenciador de filas do MQ até que:

- A conexão tenha o tempo esgotado, depois de ficar inativa durante um período de tempo limite, conforme especificado no objeto de ponte de conexão do cliente, em cujo ponto a conexão interrompida é fechada e o encerramento controlado do gerenciador de filas do MQ é concluído.
- A fila de ponte do MQ seja informada para desempenhar alguma ação, como enviar uma mensagem para o MQ, que tenta utilizar a conexão interrompida. A operação `putMessage` falha, a conexão interrompida é fechada e o encerramento controlado do gerenciador de filas do MQ é concluído.

O objeto de conexão do cliente de ponte mantém um conjunto de conexões, em espera para serem utilizadas. Se não houver atividade de ponte, o conjunto reterá as conexões do canal do cliente MQ até que o tempo inativo de conexão exceda o período de tempo limite (conforme especificado na configuração do objeto de conexão do cliente), em cujo ponto os canais no conjunto são fechados.

Quando a última conexão do canal do cliente com o gerenciador de filas do MQ for fechada, o encerramento controlado do MQ será concluído.

Objetos Administrados e Suas Características

Esta seção descreve as características dos diferentes tipos de objetos administrados associados à ponte MQe MQ. As características são atributos do objeto que podem ser consultadas utilizando uma mensagem de administração `inquireAll()`. Os resultados podem ser lidos e utilizados pelo aplicativo ou podem ser enviados em uma mensagem de administração de atualização ou criação para configurar os valores das características. Algumas características também podem ser configuradas utilizando as mensagens de administração de criação e atualização. Cada característica possui uma etiqueta exclusiva associada a ela e essa etiqueta é utilizada para configurar e receber o valor da característica.

Características de Objetos de Pontes

Consulte o Java Programming Reference para obter informações sobre o `com.ibm.mqe.mqbridge.MQeMQBridgesAdminMsg`.

Características de Objetos de Ponte

Consulte o Java Programming Reference para obter informações sobre o `com.ibm.mqe.mqbridge.MQeMQBridgeAdminMsg`.

Características de Objetos de Proxy do Gerenciador de Filas do MQ

Consulte o Java Programming Reference para obter informações sobre o `com.ibm.mqe.mqbridge.MQeMQQMgrProxyAdminMsg`.

Características de Objetos de Conexão do Cliente

Consulte o Java Programming Reference para obter informações sobre o `com.ibm.mqe.mqbridge.MQeClientConnectionAdminMsg`.

Características de Objetos de Listener da Fila de Transmissão do MQ

Consulte o Java Programming Reference para obter informações sobre o `com.ibm.mqe.mqbridge.MQeListenerAdminMsg`.

Configurando uma Ponte para Rendimento de Processamento Ideal

Utilizando o MQe, é possível criar um gateway para permitir que as mensagens circulem até o MQ. Você precisará criar determinados objetos do MQe para configurar um gateway. Esses objetos incluem uma

ponte, o proxy do gerenciador de filas, a conexão do cliente, a fila de ponte e, opcionalmente, um listener de fila de transmissão, se as mensagens também precisarem ser retornadas do MQ para o MQe.

Em uma configuração padrão, uma única conexão do cliente é criada juntamente com uma única fila de ponte. Isso é suficiente para iniciar o envio de mensagens do MQe para o MQ. Se houver um grande número de clientes conectando-se ao gateway, o número de mensagens enviadas por segundo para o MQ será reduzido.

Configuração de Gateway

Conforme mostrado na Figura 19 na página 106, se houver vários clientes conectando ao gerenciador de filas de gateway por meio do listener de comunicação, poderá haver um gargalo criado pela fila de ponte. Para evitar esse problema, você pode criar várias filas de pontes, conexões do cliente, canais de conexão do servidor e filas síncronas.

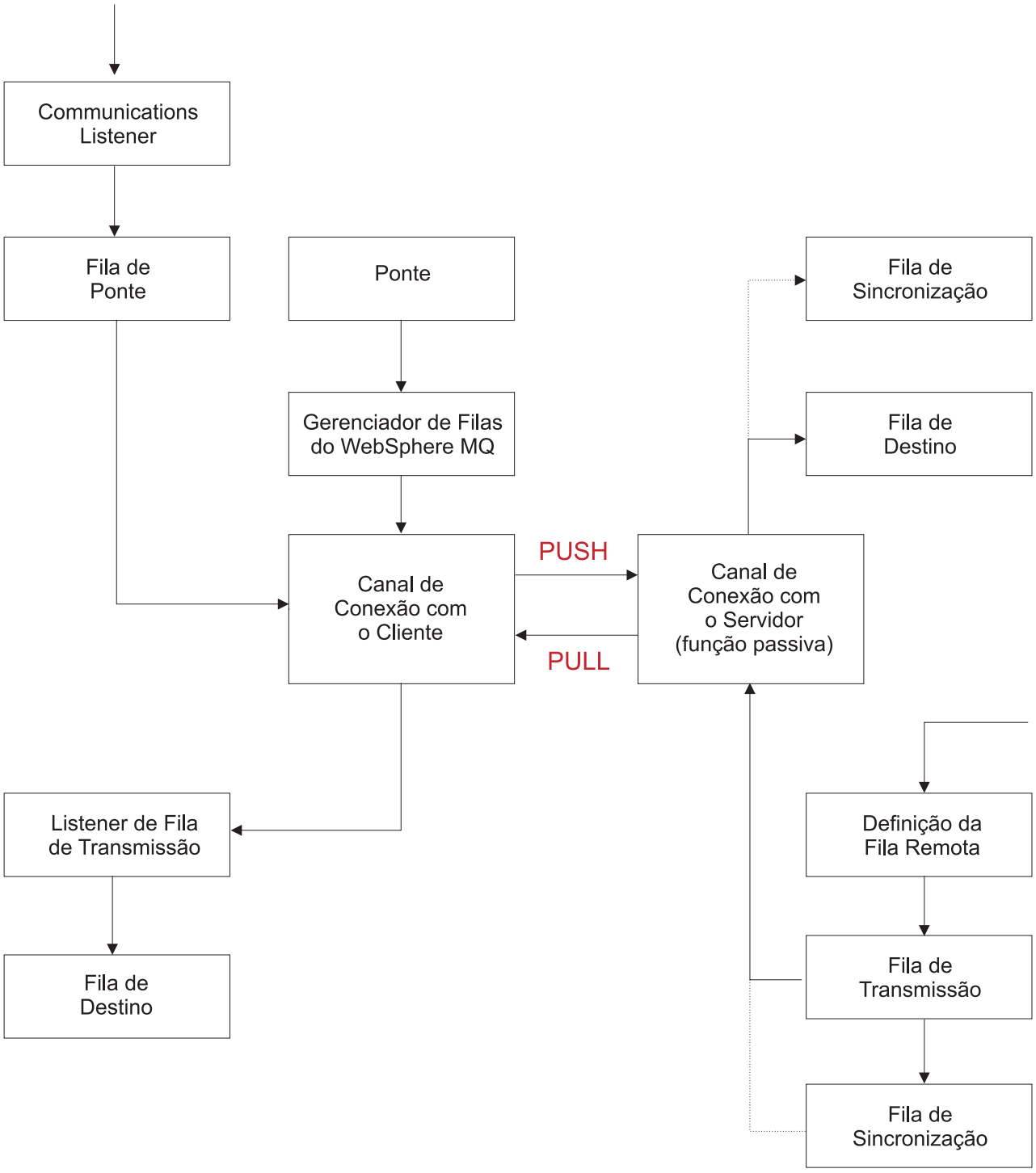


Figura 19. Configuração Padrão de um Gateway do MQe

É necessário ter cautela ao determinar o número de filas de pontes a serem criadas, pois existe um ponto em que o desempenho do gateway é reduzido com um número maior de filas de pontes.

Localizando e Criando a Configuração Ideal

Para encontrar a configuração ideal para sua configuração, é necessário criar os objetos do MQe e desempenhar testes para medir o rendimento do processamento das mensagens. Esse teste empírico deve ocorrer de maneira ideal em uma rede semelhante à rede de produção, utilizando dados da mensagem de tipo de produção.

Utilize o MQe_Script SupportPac para criar testes que são fáceis de serem configurados e podem ser repetidos. Esse SupportPac é uma ferramenta baseada em linha de comandos para criar e administrar recursos do MQe. O MQe_Script incorpora uma linguagem de script denominada TCL que você pode utilizar para gravar scripts inteligentes que incluem a lógica e o controle. É possível fazer download desse suporte no Web site do Business Integration SupportPacs: http://www.ibm.com/software/integration/support/supportpacs. Para acessar o download nesse site, selecione o produto WebSphere MQ Everyplace na lista de opções.

Utilizando o MQe_Script, você pode gravar scripts que criam todos os objetos necessários no gateway e, opcionalmente, nos clientes. Os parâmetros podem ser transmitidos em um script de modo que o script reconheça o número de filas de pontes requeridas. Como resultado, as filas de pontes e todos os outros objetos relevantes do MQe são criados.

Utilizando esse método para criar configurações do MQe, você pode testar de modo rápido e fácil com o número de filas de pontes requeridas para obter o rendimento do processamento desejado. Após a gravação do script, nenhuma programação extra é necessária.

O MQe_Script pode ainda ser utilizado para modificar quaisquer configurações existentes que possam existir.

Embora o MQe_Script não modifique objetos do MQ, eles podem ser criados utilizando comandos do MQSC. O MQe_Script não é fornecido com um script MQSC de exemplo mostrando como criar os objetos MQ necessários para a comunicação com um gateway do MQ Everyplace.

Configurando o Gateway

Siga estas etapas para configurar o gateway:

- O gateway requer uma única instância de um objeto de ponte e uma única instância de um objeto de proxy do gerenciador de filas do WebSphere MQ.
- Em seguida, você precisa criar objetos de conexão do cliente. A quantidade de objetos de conexão do cliente que precisa ser criada depende de quantas filas de pontes são necessárias.
- Por último, você precisa criar as filas de pontes, cada uma utilizando uma conexão do cliente diferente.

Configurando os Clientes

Cada cliente deve ter uma definição de fila remota para a fila MQ. Se várias filas de ponte estiverem sendo utilizadas no gateway, um alias será necessário na definição de fila remota para que qualquer aplicativo que envia mensagens utilizando um cliente obtenha o caminho correto para a fila do MQ. Isso também ajuda porque os aplicativos não precisam estar cientes dos clientes que enviam mensagens para filas diferentes.

Por exemplo, se existissem três clientes, cada um enviando para filas de ponte separadas *b1*, *b2*, e *b3*, todas as definições de filas remotas poderiam ter um alias igual a *b*. Qualquer aplicativo enviando para os clientes poderiam, então, simplesmente enviar para a fila b. Isso permite alterações na rede subjacente sem precisar fazer qualquer alteração no aplicativo.

Para utilizar todas as filas de ponte criadas no gateway e, portanto, obter rendimento do processamento de mensagem ideal, as filas de ponte devem ser divididas igualmente entre todos os clientes. Portanto, se houver 50 filas de ponte e 5.000 clientes, 100 clientes devem criar uma definição de fila remota para *bridgeq1*, outros 100 devem criar uma definição de fila remota para *bridgeq2* e assim por diante.

Criando o Gateway

Esta seção fornece as etapas básicas e os trechos de códigos para criar um gateway. Os comandos MQe_Script nesta seção são fornecidos como um script completo juntamente com quaisquer variáveis indefinidas em “Script de Amostra para Criar um Gateway” na página 111.

1. Crie um gerenciador de filas.

Para criar um gerenciador de filas básico, forneça um nome e um local de disco. O local de disco é onde o registro será salvo e quaisquer mensagens para as filas.

```
mqe_script_qm -create -qmname $GATEWAYQM -qmpath $PATH
```

2. Inicie o gerenciador de filas.

   O comando aqui utilizado não especifica um nome de gerenciador de filas a ser carregado. Isso ocorre porque, se utilizado diretamente após um comando create, os detalhes do gerenciador de filas são armazenados em cache. Para obter informações adicionais sobre como carregar um gerenciador de filas criado anteriormente, consulte a documentação que acompanha o MQe_Script ao fazer download do SupportPac.

   ```
   mqe_script_qm –load
   ```

3. Crie um listener.

   É necessário fornecer um nome e uma porta de atendimento para o listener. Opcionalmente, o tipo de adaptador poderá ser especificado se for diferente do padrão (TCPIP HTTP).

   ```
   mqe_script_listen -create -listenname $LISTENER -port $GATEWAYPORT
   ```

4. Inicie o listener.

   Por padrão, os listeners não são iniciados depois de serem criados. Entretanto, depois que o gerenciador de filas tiver sido parado e reiniciado, o listener será iniciado automaticamente.

   ```
   mqe_script_listen –start -listenname $LISTENER
   ```

5. Crie uma ponte.

   Um nome arbitrário é necessário para o objeto de ponte. Ele age como o pai para todos os outros objetos relacionados à ponte.

   ```
   mqe_script_bridge -create -bridgename $BRIDGE
   ```

6. Crie um proxy do gerenciador de filas.

   O proxy do gerenciador de filas deve ser associado ao objeto de ponte criado anteriormente. Também deve ser nomeado de acordo com o nome do gerenciador de filas do MQ com o qual a conexão ocorrerá. Por último, é necessário o endereço IP ou nome do host da máquina na qual o gerenciador de filas do MQ está definido.

   ```
   mqe_script_mqproxy -create -proxyname $PROXY -bridgename $BRIDGE -hostname $ADDRESS
   ```

7. Crie uma definição de conexão para o gerenciador de filas MQ.

   Uma conexão MQ especial deve ser criada para definir o gerenciador de filas MQ. Ela precisa ter o mesmo nome que o gerenciador de filas do MQ.

   ```
   mqe_script_condef -create -cdname $PROXY -type mq
   ```

8. Crie vários canais de conexão do cliente.

   Esses canais devem ser associados aos objetos de ponte e de proxy do gerenciador de filas criados anteriormente. Eles devem ter o mesmo nome que o canal de conexão do servidor no gerenciador de filas do MQ. Por último, é necessário definir a fila síncrona do MQ que será utilizada. Isso será diferente para cada canal de conexão do cliente. Se o gerenciador de filas do MQ não estiver atendendo na porta padrão 1414, o número da porta também deverá ser definido.

   ```
   mqe_script_mqconn -create -clientconnname $CC$j -proxyname $PROXY -bridgename $BRIDGE -syncqname
       $SYNCQ$j -port $PORT
   ```

9. Opcionalmente, para receber mensagens de volta do MQ, crie vários listeners de fila de transmissão.

   Se um listener de fila de transmissão for necessário, ele deverá ser associado aos objetos de canal de conexão do cliente, de proxy do gerenciador de filas e de ponte criados anteriormente. Também deve ter o mesmo que a fila de transmissão no gerenciador de filas do MQ.

   ```
   mqe_script_mqlisten -create -listenname $LISTEN -clientconnname $CC$j -proxyname
       $PROXY -bridgename $BRIDGE
   ```

10. Crie várias filas de pontes.

    É necessário fornecer um nome para a fila de ponte. Em circunstâncias normais, os nomes das filas de pontes refletem o nome da fila de destino do MQ. Um modo alternativo de vincular a fila de ponte à fila do MQ é utilizar um parâmetro adicional, o Nome da Fila do MQ, que permite que o

nome da fila de ponte seja algo diferente. É essa última abordagem que precisa ser considerada ao definir várias filas de pontes. Cada fila de ponte deve ser associada a uma conexão do cliente individual. Ou seja, existe um relacionamento de um-para-um. A fila de ponte também precisa ser associada à ponte e ao proxy do gerenciador de filas, vinculando a fila de ponte a um gerenciador de filas, uma fila e uma conexão específicos do MQ.

```
mqe_script_bridgeq -create -qname $BRIDGEQ$j -bridgename $BRIDGE -destination $PROXY
    -mqqname $REALBRIDGEQ -clientconnname $CC$j
```

11. Inicie a ponte.

    Iniciar a ponte, por padrão, também iniciará todos os objetos-filho. Se você criou listeners de fila de transmissão e o gerenciador de filas do MQ não puder ser contactado, isso poderá resultar na falha de início do listener de fila de transmissão. Deve-se tomar cuidado com isso, pois o comando pode voltar como bem-sucedido mesmo se todos os objetos-filho não tiverem sido iniciados. Por outro lado, um erro poderá ser emitido e talvez você deseje ignorá-lo e continuar com o script. Cada objeto-filho pode ser iniciado individualmente, se desejado.

    ```
    mqe_script_bridge -start -bridgename $BRIDGE
    ```

Etapas para Criar o Cliente

As etapas básicas para criar um cliente são definidas a seguir. Os comandos do MQe_Script para acompanhá-las são fornecidos juntamente com qualquer variável indefinida em "Script de Amostra para Criar um Cliente" na página 113.

1. Crie um gerenciador de filas.

   Forneça um nome e um local de disco.

   ```
   mqe_script_qm –create –qmname $CLIENT –qmpath $PATH
   ```

2. Inicie o gerenciador de filas.

   Consulte o exemplo de código a seguir.

   ```
   mqe_script_qm –load
   ```

3. Crie uma definição de conexão para o gerenciador de filas de gateway.

   A definição de conexão deve ter o mesmo nome que o gerenciador de filas de gateway. Defina a porta na qual o gateway está atendendo e, se o gateway estiver atendendo em um adaptador diferente do padrão, ele também deverá ser definido.

   ```
   mqe_script_condef –create –cdname $GATEWAYQM –port $PORT –address $ADDRESS
   ```

4. Crie uma definição de conexão de caminho para o gerenciador de filas do MQ.

   O cliente deve conhecer o gerenciador de filas do MQ para enviar mensagens para suas filas. Uma conexão de caminho pode ser criada, em que o nome da conexão é o nome do gerenciador de filas do MQ e o nome do caminho é o gerenciador de filas de gateway.

   ```
   mqe_script_condef –create –cdname $MQNAME –viaqmname $GATEWAYQM
   ```

5. Crie uma definição de fila remota para a fila de ponte e inclua um alias.

   O nome da definição de fila remota deve corresponder a uma das filas de pontes definidas no gateway. O nome do gerenciador de filas do MQ também deve ser definido.

   Para utilizar várias filas de pontes, em que o nome da fila de ponte não é igual ao nome da fila do MQ real, é necessário utilizar aliases. Um aplicativo não pode utilizar um nome da fila de ponte no gateway como o nome da fila do MQ porque essa fila não existe no gerenciador de filas do MQ.

   Além disso, a definição remota da fila não pode ser chamada com o nome da fila do MQ real, pois essa referência não existe no gerenciador de filas de gateway. Portanto, é útil incluir o nome real da fila do MQ como uma alias para a definição de fila remota, para que os aplicativos conheçam exatamente para onde a mensagem deve ser enviada.

   ```
   mqe_script_sproxyq –create –qname $BRIDGEQ$BRIDGEQNUM  –destination $MQNAME –alias $REALBRIDGENAME
   ```

   Ao utilizar o MQe_Script, o alias pode ser incluído no momento da criação ou como uma atualização em um estágio posterior se a definição de fila remota já existir.

6. Teste a conexão enviando uma mensagem do cliente para o gerenciador de filas do MQ.

Para que os aplicativos façam a mesma chamada, independentemente do cliente para os quais estão enviando mensagens, a mensagem pode ser enviada utilizando o alias da definição de fila remota.

```
mqe_script_msg –put –qname $REALBRIDGENAME –qmname $MQNAME
```

Dicas sobre Como Gravar um Script

Ao gravar um script, geralmente é mais fácil definir um conjunto de variáveis no início do script. Isso facilita o gerenciamento de quaisquer alterações nas convenções de nomenclatura ou no número de objetos do MQe.

Em TCL, as variáveis são definidas utilizando o comando set e referenciadas utilizando um $ na frente do nome da variável.

Também é útil definir variáveis em letras maiúsculas para que possam ser identificadas facilmente em um script.

```
set PATH "C:\\MQeScript\\gateway"
```

Ao chamar comandos do MQe_Script, eles podem ser definidos sozinhos ou estruturas de controle TCL podem ser incluídas para fornecer feedback sobre o sucesso do comando e, potencialmente, sair do script se ocorrerem erros.

Um método para verificar o sucesso de um comando é utilizar um bloco if / else com um comando catch.

```
if { [catch {mqe_script_qm -create -qmname $GATEWAYQM -qmpath
$PATH} error] } {
          puts "Ocorreu um erro ao criar o gerenciador de filas";
puts "O motivo foi: $error"
          exit
    } else {
      puts "Gerenciador de filas criado"
}
```

Conforme mostrado no trecho de código anterior, o comando puts é utilizado para imprimir texto na tela e o comando exit pára a execução do script. Se algum erro for emitido, o texto do erro será salvo na variável denominada “erro” e poderá, então, ser acessado.

Se um script estiver sendo gravado para criar vários objetos de ponte definidos por uma variável, a criação desses objetos poderá ser colocada dentro de um loop. O método mais fácil de criar vários objetos é ter um nome padrão para cada um dos objetos e, em seguida, incluir um número em cada um deles para que sejam exclusivos.

```
for {set j 1} {$j <= $QNUM} {incr j} {

    #criar todas as conexões do cliente

mqe_script_mqconn -create -clientconnname $CC$j -proxyname $PROXY -bridgename $BRIDGE
      -syncqname $SYNCQ$j -port $PORT

}
```

O snippet acima mostra como os nomes podem ser criados utilizando uma variável de loop. $CC já está definido como um prefixo de nome da conexão do cliente e $SYNCQ como um prefixo de nome da fila síncrona.

O trecho de código também utiliza # para definir um comentário.

Para fornecer o MQe_Script com um arquivo de script, ele deve ser salvo com uma extensão .tcl e o comando para fornecer um script é o comando source.

```
source {C:\MQeScript\gatewayscript.tcl}
```

Para obter informações adicionais sobre TCL e gravação de scripts, consulte a documentação que acompanha o MQe_Script.

Script de Amostra para Criar um Gateway

```
set PATH "C:\\MQeScript\\gateway"
set ADDRESS 127.0.0.1
set GATEWAYQM gatewayqm
set LISTENER listener
set GATEWAYPORT 1881
set PROXY QM_jane
set BRIDGE bridge
set REALBRIDGEQ mqlocalq
set CC FOR.GATEWAYQM.
set LISTEN togateway
set BRIDGEQ bridgeq
set SYNCQ SYNC.QUEUE.
set PORT 1414 set QNUM 50

#criar o gerenciador de filas de gateway

if { [catch {mqe_script_qm -create -qmname $GATEWAYQM -qmpath $PATH} error] } {
  puts "Ocorreu um erro ao criar o gerenciador de filas";
  puts "O motivo foi: $error"
  exit
    } else {
     puts "Gerenciador de filas criado"
}

#carregar o gerenciador de filas

if { [catch {mqe_script_qm -load} error] } {
    puts "Ocorreu um erro ao carregar o gerenciador de filas";
   puts "O motivo foi: $error"
   exit
} else {
     puts "Gerenciador de filas carregado"
}

#criar o listener

if { [catch {mqe_script_listen -create -listenname $LISTENER -port
$GATEWAYPORT} error] } {
   puts "Ocorreu um erro ao criar um listener";
   puts "O motivo foi: $error"
   exit
} else {
      puts "Listener criado" }

#criar uma ponte

if { [catch {mqe_script_bridge -create -bridgename $BRIDGE} error] }
{
            puts "Ocorreu um erro ao criar a ponte";
            puts "O motivo foi: $error"
            exit
} else {
      puts "Ponte criada"
}

#criar um proxy mq

if { [catch {mqe_script_mqproxy -create -proxyname $PROXY -bridgename $BRIDGE
            -hostname $ADDRESS} error] } {
            puts "ocorreu um erro ao criar o proxy";
            puts "O motivo foi: $error"
            exit
```

```
} else {
      puts "Proxy do gerenciador de filas MQ criado"
}

#criar uma conexão com o gerenciador de filas WebSphere MQ

if { [catch {mqe_script_condef -create -cdname $PROXY -type mq} error] } {
            puts "Ocorreu um erro ao criar a conexão para o gerenciador de filas MQ";
            puts "O motivo foi: $error"
            exit
} else {
      puts "Conexão com o gerenciador de filas MQ criada"
}

#criar as conexões do cliente, os listeners e as filas de ponte

for {set j 1} {$j <= $QNUM} {incr j} {
#criar todas as conexões do cliente

if { [catch {mqe_script_mqconn -create -clientconnname $CC$j -proxyname $PROXY
-bridgename $BRIDGE -syncqname $SYNCQ$j -port $PORT} error] } {
            puts "Ocorreu um erro ao criar a conexão do cliente $CC$j";
            puts "O motivo foi: $error"
            exit
} else {
            puts "conexão do cliente criada"
}

#criar todos os listeners nas novas conexões do cliente

if { [catch {mqe_script_mqlisten -create -listenname $LISTEN -clientconnname $CC$j
-proxyname $PROXY -bridgename $BRIDGE} error] } {
            puts "Ocorreu um erro ao criar o listener $LISTEN na conexão do cliente $CC$j";
            puts "O motivo foi: $error"
            exit
} else {
            puts "listener criado"
}

#criar todas as filas de ponte

if { [catch {mqe_script_bridgeq -create -qname $BRIDGEQ$j -bridgename $BRIDGE
-destination $PROXY -mqqname $REALBRIDGEQ -clientconnname $CC$j} error] } {
            puts "Ocorreu um erro ao criar a fila de ponte $BRIDGEQ ";
            puts "O motivo foi: $error"
            exit
            } else {
                  puts "fila de ponte criada"
            }
}

# O script foi concluído... vamos fechar o gerenciador de filas

if { [catch {mqe_script_qm -unload} error] } {
            puts "Falha ao parar o gerenciador de filas"
            puts "O motivo foi: $error"
            exit }
else {
            puts "Gerenciador de filas parado"
}
puts "CreateGatewayQM script completed successfully"
exit 0
```

Script de Amostra para Criar um Cliente

Este script mostra apenas os comandos básicos do MQe_Script para salvar a duplicação. Estruturas de controle poderiam ser colocadas em torno dos comandos ou eles poderiam ser executados no estado em que se encontram.

Para que este script seja executado com êxito, o gerenciador de filas criado pelo script de gateway deve estar em execução, assim como o gerenciador de filas MQ.

```
set CLIENT client1
set PATH "C:\\MQeScript\\client"
set ADDRESS 127.0.0.1
set GATEWAYQM gatewayqm
set PORT 1881
set BRIDGEQ bridgeq
set BRIDGEQNUM 1
set REALBRIDGENAME mqlocalq
set MQNAME QM_jane

mqe_script_qm –create –qmname $CLIENT –qmpath $PATH

mqe_script_qm –load

mqe_script_condef –create –cdname $GATEWAYQM –port $PORT –address $ADDRESS

mqe_script_condef –create –cdname $MQNAME –viaqmname $GATEWAYQM

mqe_script_sproxyq –create –qname $BRIDGEQ$BRIDGEQNUM –destination $MQNAME –alias $REALBRIDGENAME

mqe_script_msg –put –qname $REALBRIDGENAME –qmname $MQNAME
```

Aumento de Desempenho

Criando uma rede do MQe que utiliza várias filas de pontes, o número de mensagens por segundo que podem ser enviadas por meio do gateway pode aumentar muito.

O número de mensagens enviadas por meio do gateway não depende apenas da configuração de seu gateway, mas de outros problemas, como o tipo de mensagens enviadas, a configuração de rede e o hardware.

Em um cenário de teste, constatou-se que quando aproximadamente 1.350 clientes estavam se conectando a um gateway, o aumento no número de mensagens enviadas por segundo entre uma única fila de ponte e as 10 filas de ponte era de um pacote de 34.

Isso aumentou para um pacote de 86 ao comparar uma única fila de ponte com 50 filas de ponte, mas comparando uma única fila de ponte com 1.350 filas de ponte, o aumento caiu para um pacote de 34.

Observe que esse aumento está relacionado apenas às mensagens enviadas ao MQ e não às mensagens recebidas do MQ.

Conclusão

Utilizar várias filas de pontes e objetos de conexão do cliente pode aumentar muito o rendimento do processamento de mensagens do MQe para o MQ. Entretanto, não existem quantias concretas desses aumentos de desempenho porque isso não depende apenas da configuração do MQe.

O MQe_Script pode ser utilizado para criar essas configurações com a capacidade de fazer alterações simples para influenciar a quantidade de objetos criados, que é uma parte importante para alcançar o número ideal para sua configuração individual.

Manipulando Mensagens que Não Podem Ser Entregues

O listener de fila de transmissão da ponte do MQ age de um modo semelhante a um canal do MQ, recebendo mensagens de uma fila de transmissão do MQ e entregando-as para a rede MQe.

Ele segue a convenção do MQe, ou seja, se uma mensagem não puder ser entregue, uma *regra de mensagem não entregue* será consultada para determinar como o listener de fila de transmissão deve reagir.

Se a regra indicar as opções de relatório no cabeçalho da mensagem, e elas indicarem que a mensagem deve ser enviada para uma fila de devoluções, a mensagem será enviada para a fila do MQ, no gerenciador de filas de *envio*.

Suporte de Ponte a Idiomas Nacionais

Esta seção descreve como a ponte do MQ manipula mensagens que circulam entre os sistemas MQSeries que utilizam diferentes idiomas nacionais. O diagrama a seguir representa o fluxo de uma mensagem de um aplicativo cliente do MQe para um aplicativo do MQ.

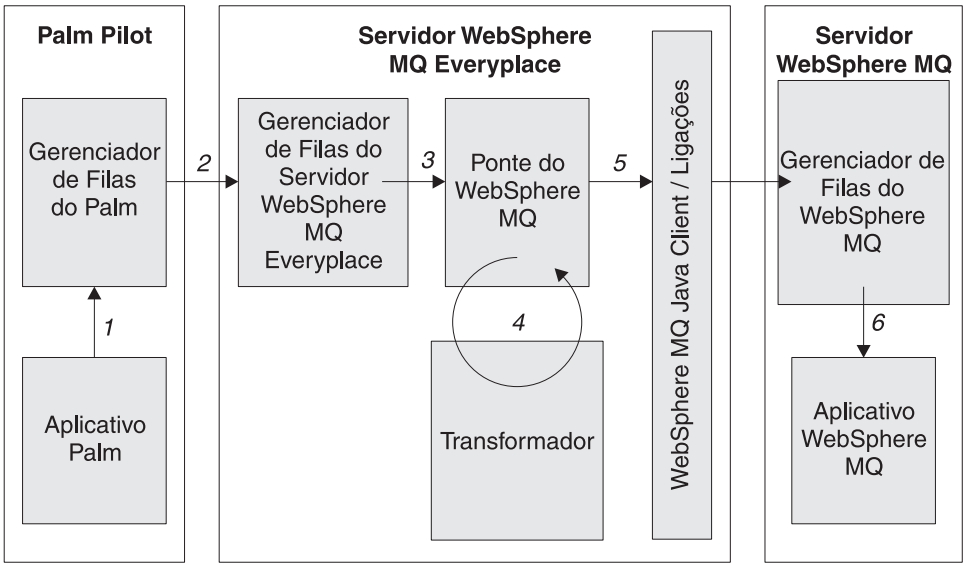


Figura 20. Fluxo de Mensagens do MQe para o MQ

1. **Aplicativo Cliente**
   a. O aplicativo cliente constrói um objeto de mensagem do MQe contendo os seguintes dados:

      Um Campo Unicode
      Essa cadeia é gerada utilizando as bibliotecas apropriadas disponíveis na máquina cliente (se C/C++ estiver sendo utilizado).

      Um Campo de Byte
      Esse campo nunca deve ser convertido

      Um Campo Ascii
      Essa cadeia possui um intervalo muito limitado de caracteres válidos, em conformidade com o padrão ASCII. Os únicos caracteres válidos são aqueles invariáveis em todas as páginas de códigos ASCII.

   b. A mensagem é enviada para o gerenciador de filas do Palm. Nenhuma conversão é feita durante esse envio.
2. **O gerenciador de filas de cliente envia para o gerenciador de filas de servidor**
   A mensagem não é convertida de modo algum nesta etapa.
3. **O servidor do MQe envia a mensagem para a fila de ponte do MQ**
   A mensagem não é convertida de modo algum nesta etapa.
4. **A ponte do MQ transmite a mensagem do MQe para o transformador gravado pelo usuário**

Nota: Os exemplos nesta seção estão em Java porque os transformadores podem ser gravados apenas em Java.

O transformador cria uma mensagem do MQ conforme a seguir:

- O campo Unicode na mensagem do MQe é recuperado utilizando:
  ```
  String value = MQemsg.GetUnicode(fieldname)
  ```
- O valor recuperado é copiado para a mensagem do MQ utilizando `MQmsg.writeChars( value )`
- O campo de byte na mensagem do MQe é recuperado utilizando:
  ```
  Byte value = MQemsg.getByte(fieldName)
  ```
- O valor recuperado é copiado para a mensagem MQ utilizando o `MQmsg.writeByte(value)`
- O campo ascii na mensagem do MQe é recuperado utilizando o `MQmsg.writeChars(value)` para criar um valor unicode ou `MQmsg.writeString( value)` para criar um valor dependente do conjunto de códigos na mensagem do MQ.

Se utilizar `writeString()`, o conjunto de caracteres da cadeia também poderá ser configurado. O transformador retorna a mensagem do MQ resultante para o código de ponte do MQ de chamada.

5. **A ponte do MQ transmite a mensagem para o MQ utilizando o MQ Classes para Java**

   Os valores Unicode na mensagem do MQ são convertidos de big-endian para little-endian e vice-versa, conforme necessário. Os valores de byte na mensagem do MQ são convertidos de big-endian para little-endian e vice-versa, conforme necessário. O campo que foi criado utilizando o `writeString()` é convertido conforme a mensagem é enviada para o MQ, utilizando rotinas de conversão no MQ Classes para Java. Os dados ASCII devem permanecer dados ASCII independentemente das conversões de conjunto de caracteres desempenhadas. As conversões feitas durante essa etapa dependem da página de códigos da mensagem, do CCSID da conexão do cliente MQ Classes para Java de envio e do CCSID da conexão do servidor MQ de recebimento.

6. **A mensagem é recebida por um aplicativo do MQ**

   Se a mensagem contiver uma cadeia unicode, o aplicativo deverá tratar essa cadeia como uma cadeia unicode ou então convertê-la em algum outro formato (UTF8, por exemplo). Se a mensagem contiver uma cadeia de bytes, o aplicativo poderá utilizar os bytes no estado em que se encontram (dados brutos). Se a mensagem contiver uma cadeia, ela será lida na mensagem e poderá ser convertida em um formato de dados diferente, conforme requerido pelo aplicativo. Essa conversão depende do valor do conjunto de código no campo de cabeçalho `characterSet`. As classes Java fornecem isso automaticamente.

Conclusão

Se você tiver um aplicativo do MQe e desejar transportar dados relacionados a caracteres do MQe para o MQ, sua opção de método será determinada amplamente pelos dados que deseja transportar:

- **Se os dados contiverem caracteres nos intervalos variantes das páginas de códigos de caracteres ASCII**, o caractere de um ponto de código será alterado na medida que você alternar entre as diversas páginas de códigos ASCII; em seguida, utilize o putUnicode, que nunca está sujeito à conversão entre as páginas de códigos, ou o putArrayOfByte, neste caso é necessário manipular a conversão entre a página de códigos do emissor e a página de códigos do receptor.

  Nota: ***NÃO UTILIZE*** putAscii(), pois os caracteres nas partes variantes das páginas de códigos ASCII estão sujeitos à conversão.
- **Se os dados contiverem apenas caracteres nos intervalos invariáveis das páginas de códigos de caracteres ASCII**, você poderá utilizar putUnicode (que nunca está sujeito à conversão entre as páginas de códigos) ou putAscii, que nunca está sujeito à conversão entre páginas de códigos, pois todos os dados permanecem no intervalo invariável das páginas de códigos ASCII.

Configurando Gerenciadores de Filas como Servlets

Introdução

Um gerenciador de filas do MQe pode ser executado em um servlet.

Nota: No MQe versão 2.0, o jar *reprovado* deve estar no caminho de classe para que os servlets funcionem.

Esta seção descreve um servlet de exemplo incluído no MQe e como configurá-lo utilizando o WAS (WebSphere Application Server) 4.0.

Uma Configuração de Servlet de Exemplo Utilizando WAS

Um gerenciador de filas do MQe pode ser executado em um servlet.

Nota: No MQe versão 2.0, o jar *reprovado* deve estar no caminho de classe para que os servlets funcionem.

Esta seção descreve um servlet de exemplo incluído no MQe e como configurá-lo utilizando o WAS (WebSphere Application Server) 4.0.

Um servlet de exemplo que recebe rastreio da rotina de tratamento de rastreio com.ibm.mqe.trace.MQeTraceToBinaryMidp está incluído nas classes de exemplo.

Esse servlet é o examples.trace.MQeTraceServlet.

Utilizando-o como um exemplo, as seguintes informações explicam como configurá-lo para que funcione com o WAS 4.0. Outros servidores de aplicativos exigirão etapas diferentes.

Iniciar a Application Assembly Tool

Primeiramente, o código do servlet deve ser compactado em um formato que se ajuste ao servidor de aplicativos. Este exemplo criará um módulo da Web para ser utilizado com o WAS 4.0.

No WebSphere Administrative Console, escolha o item de menu Application Assembly Tool no menu Tools. A Application Assembly Tool deverá aparecer.

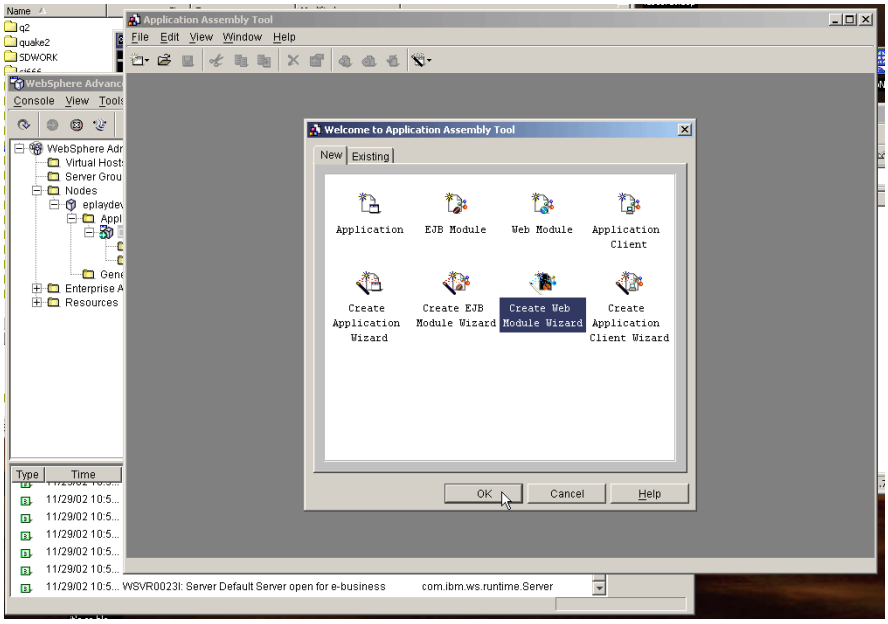


Figura 21. O WebSphere Administrative Console

Selecione "Create Web Module Wizard" e clique em OK.

Especificando Propriedades do Módulo da Web

Ao especificar as propriedades, digite o nome do arquivo e outras informações, se desejar.

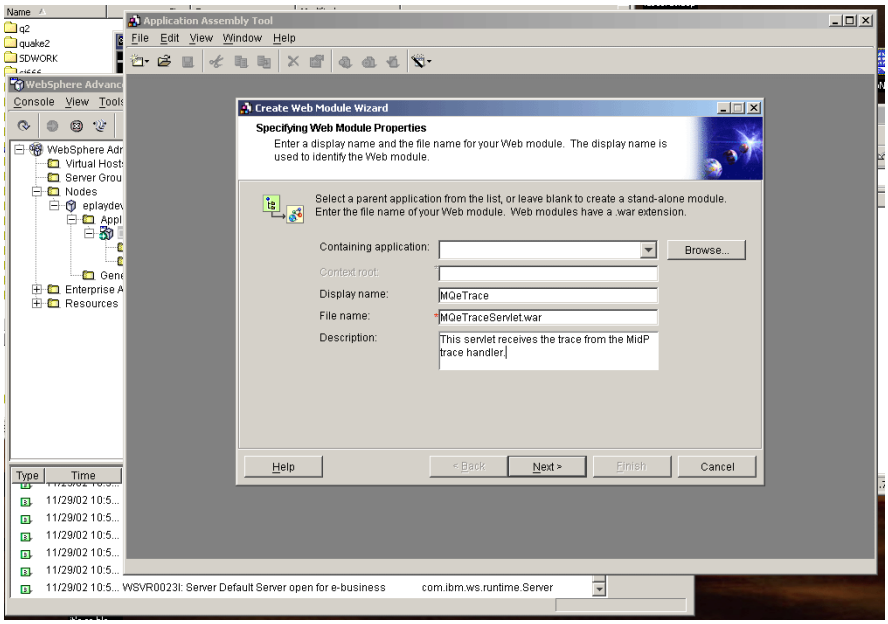


Figura 22. Especificando Propriedades do Módulo da Web

Incluindo Arquivos no Aplicativo

A próxima etapa é incluir arquivos no aplicativo. O examples.trace.MQeTraceServlet está no MQeExamples.jar e conta com as classes de MQeGateway.jar, MQeExamples.jar e MQeTraceDecode.jar.

Como todas as classes necessárias foram incluídas, o próximo painel que perguntar se você deseja criar um distribuível ou configurar um caminho de classe, pode ser deixado em branco, apenas clique em

Next. O próximo painel é para configurar qualquer ícone para esse aplicativo da Web. Se você não tiver nenhum, apenas clique em Next.

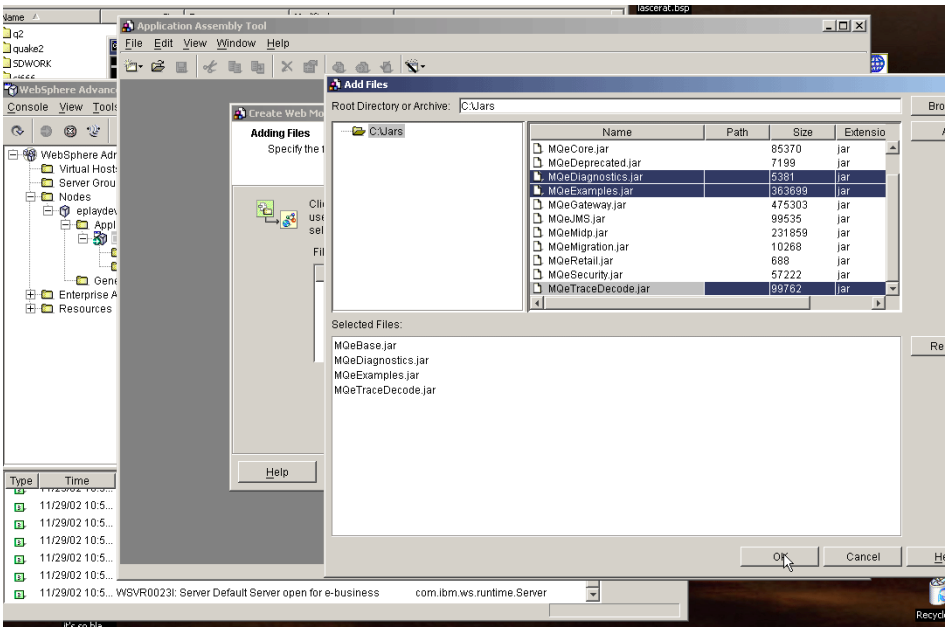


Figura 23. Incluindo Arquivos no Aplicativo

Incluindo Componentes da Web

Em seguida, você precisa especificar as propriedades dos componentes.

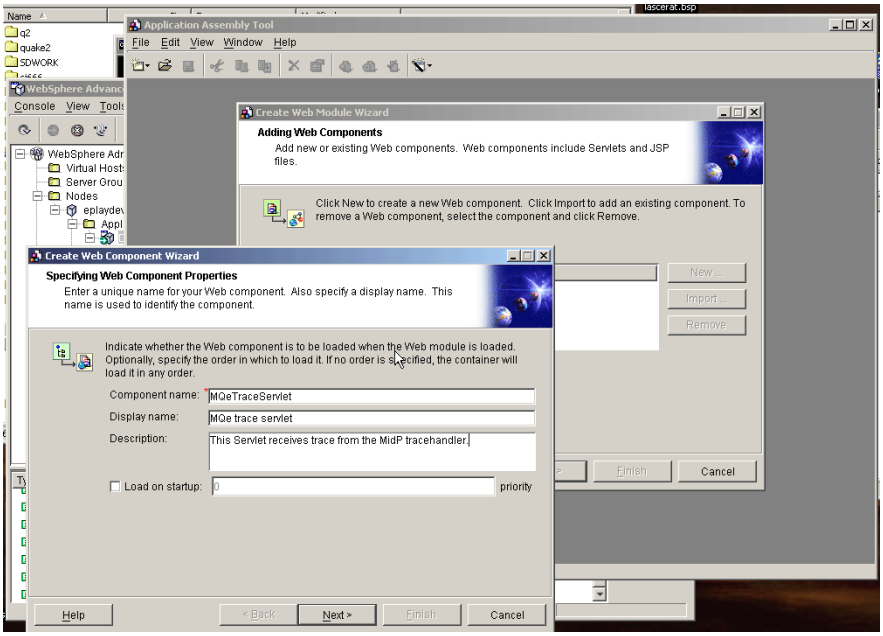


Figura 24. Incluindo Componentes da Web

Apenas o nome do componente é obrigatório, mas você pode querer incluir um nome de exibição e uma descrição.

Especificando o Tipo de Componente e o Nome da Classe

O próximo painel permite especificar em que classe o servlet deve ser executado.

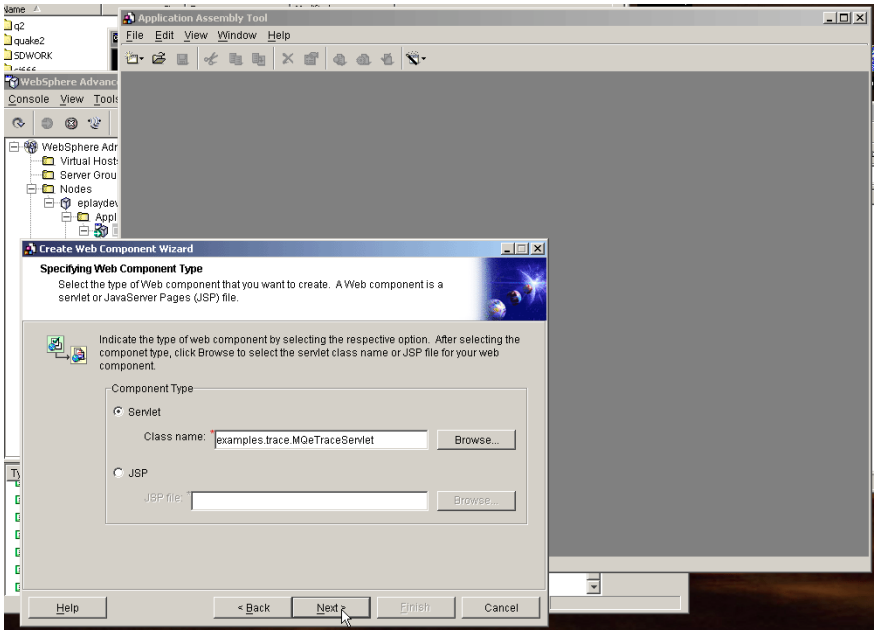


Figura 25. Especificando o Tipo de Componente e o Nome da Classe

Os próximos quatro painéis podem ser deixados seguramente em branco, eles especificam ícones, funções de segurança e parâmetros de inicialização.

Especificando uma URL a Ser Mapeada para o Servlet

Depois disso, você deve especificar qual URL será mapeada para o servlet. A URL final terá o formato http://hostname:port/specified_dir/specified_url_pattern

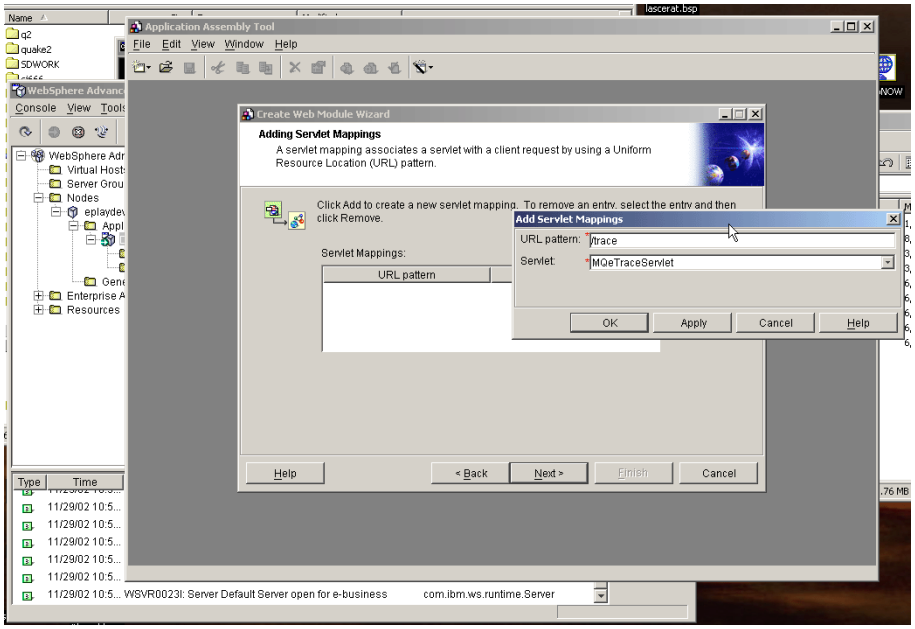


Figura 26. Especificando uma URL a Ser Mapeada para o Servlet

Todos os painéis subseqüentes podem ser deixados em branco. Eles são utilizados para incluir recursos, parâmetros de contexto, páginas de erros, mapeamentos MIME, bibliotecas de tags, arquivos de boas-vindas e referências a EJB.

Concluindo e Salvando o Arquivo

Clique em Finish e, em seguida, salve o arquivo. Se você salvar o arquivo em \AppServer\InstallableApps\ onde instalou o servidor de aplicativos WebSphere, ele aparecerá automaticamente na lista de servlets no painel de administração.

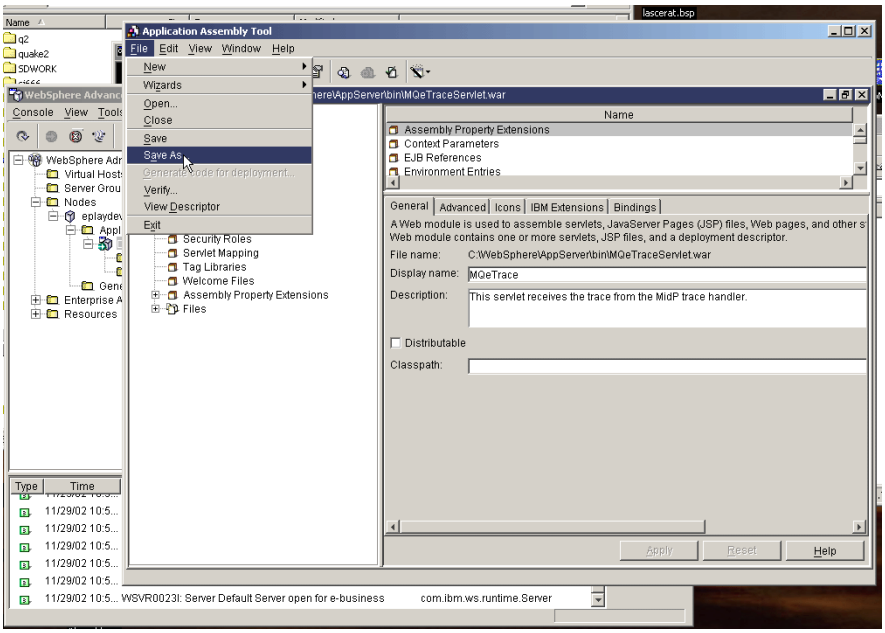


Figura 27. Salvando o Arquivo

Instalar o Aplicativo Corporativo

Em seguida, esse componente precisa ser importado e iniciado. No botão do assistente, selecione ″Install Enterprise Application″.

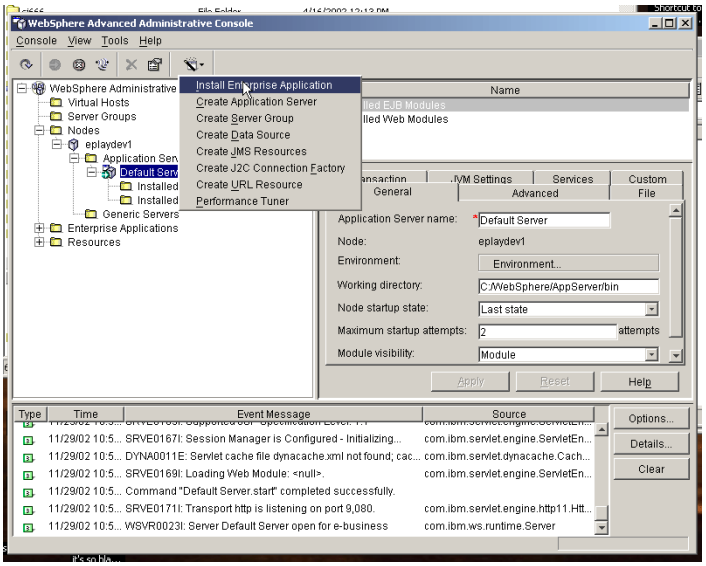


Figura 28. Instalar o Aplicativo Corporativo

Instalando o Componente como um Módulo Independente

Instale o componente como um módulo independente.

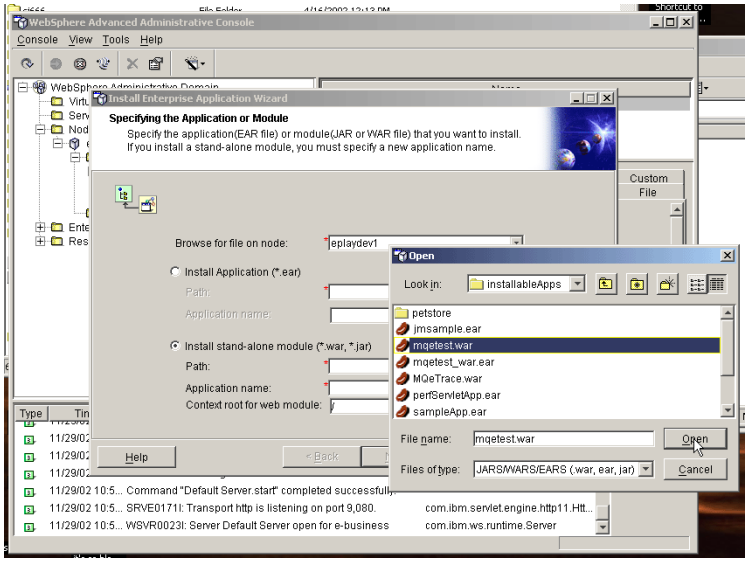


Figura 29. Instalando o Componente como um Módulo Independente

Especificando um Nome de Aplicativo

Especifique um nome de aplicativo e uma raiz para o módulo da Web. Esta é a parte da URL imediatamente após o http://hostname:portnumber/ e não deve ser deixada como /

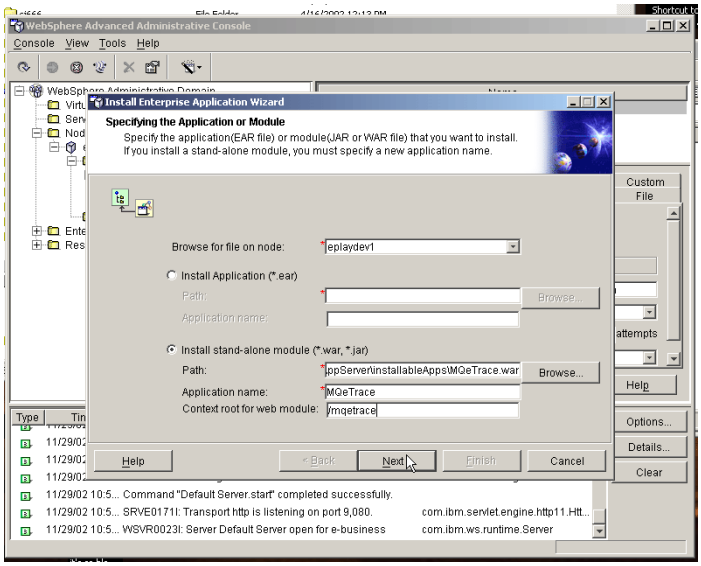


Figura 30. Especificando um Nome do Aplicativo

Todos os painéis subseqüentes podem ser deixados em branco; eles são utilizados para controlar os usuários, as funções do EJB, as ligações JNDI, os mapeamentos do EJB, as referências de recursos, as origens de dados para EJB, as origens de dados para CMP e os hosts virtuais.

Concluindo a Configuração

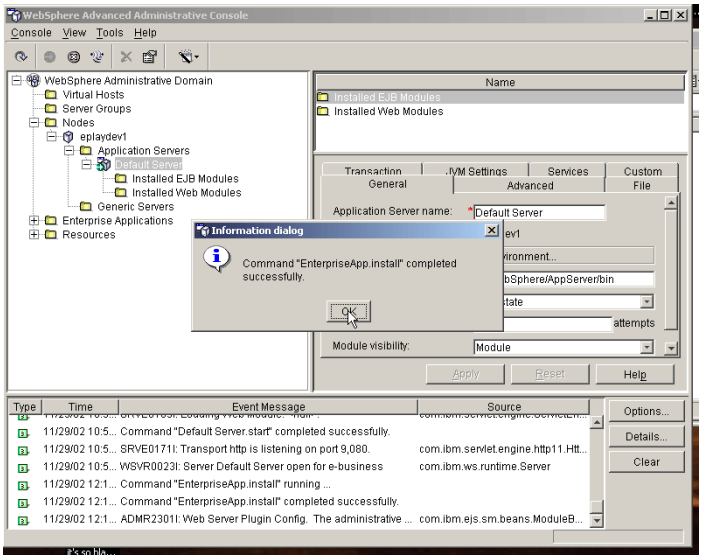


Figura 31. Diálogo de Informações

Iniciando o Módulo da Web

Em seguida, é necessário iniciar o módulo da Web. Selecione o servidor de aplicativos para o qual ele foi configurado. Ele deve aparecer sob Installed Web Modules.

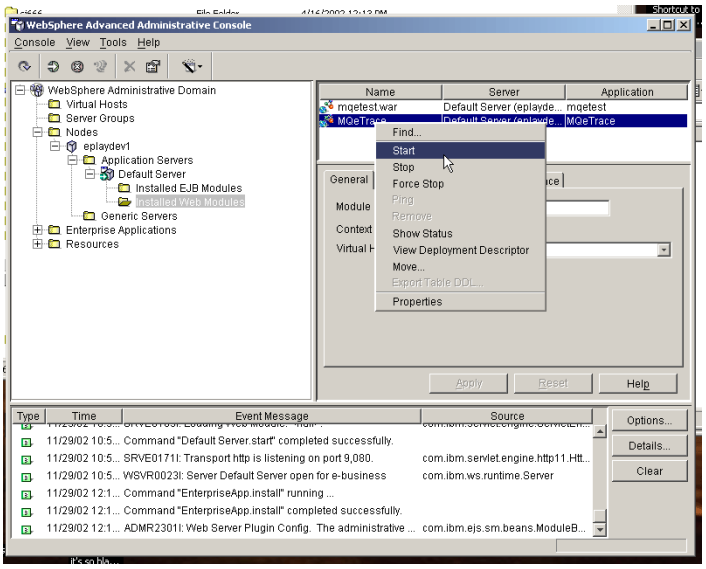


Figura 32. Iniciando o Módulo da Web

Inicialização Bem-sucedida

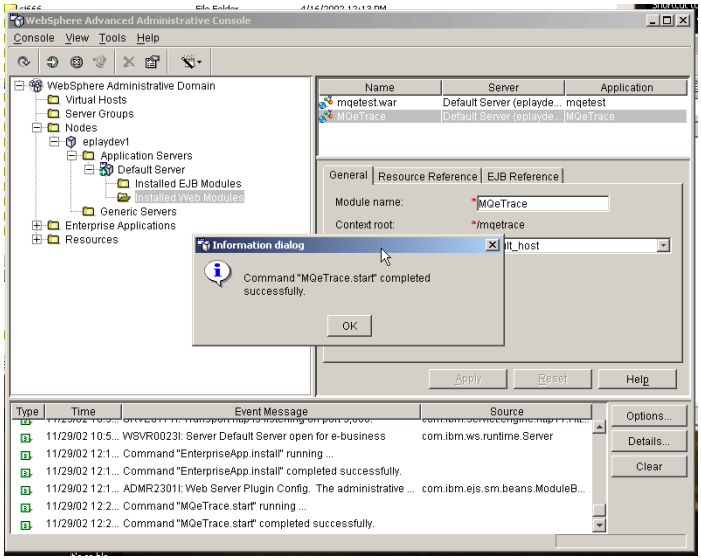


Figura 33. Mensagem de Sucesso do Diálogo de Informações

Utilizando o Servlet

Se tudo ocorreu corretamente, agora o servlet deverá estar disponível para ser utilizado a partir do com.ibm.mqe.trace.MQeTraceToBinaryMidp.

Como esse servlet não suporta recebimento, a visualização da URL com um navegador da Web resultará em um erro 405. Isto é normal.

Se o servidor de aplicativos for configurado com os padrões, a URL do servlet será http://localhost:9080/mqetrace/trace.

Configuração do JMS (Java Message Service)

Alterações de Nomenclatura de Objetos JMS da V2.0.1

As alterações de nomenclatura a seguir se aplicam a partir do MQe V2.0.1 (os nomes antigos ainda funcionarão por questões de retrocompatibilidade).

Nome Antigo	Novo Nome
QueueConnection	Conexão
MQeQueueConnection	MQeConnection
MQeQueueConnectionFactory	MQeConnectionFactory
QueueConnectionFactory	ConnectionFactory

Introdução ao JMS

Para que os aplicativos JMS sejam transportáveis, eles devem ser isolados da administração do fornecedor de sistema de mensagens subjacente. Isso é feito definindo *objetos administrados* pelo JMS, que encapsulam informações específicas do provedor. Os objetos administrados são criados e configurados utilizando recursos específicos do provedor, mas são utilizados pelos clientes por meio de interfaces JMS transportáveis.

Existem dois tipos de objetos administrados pelo JMS:

- Um `ConnectionFactory`, utilizado por um cliente para criar uma conexão com um provedor.
- Um `Destination`, utilizado por um cliente para especificar o destino das mensagens enviadas e a origem das mensagens recebidas.

No JMS do MQe, eles correspondem a duas classes:

- `MQeConnectionFactory` deve ser configurado para que possa obter uma referência a um gerenciador de filas do MQe.
- `MQeJMSQueue` pode ser configurado com detalhes de uma fila do MQe.

Nota:

Essas classes são normalmente colocadas em um espaço de nomes JNDI por um administrador.

Entretanto, como em pequenos dispositivos o acesso a um espaço de nomes JNDI pode ser impraticável ou pode representar um código extra desnecessário, essas classes não incluem os métodos necessários para permitir que sejam ligadas por JNDI.

Em vez disso, duas subclasses, `MQeJNDIConnectionFactory` e `MQeJMSJNDIQueue`, estendem essas classes para permitir que elas sejam armazenadas utilizando JNDI.

Configurando o MQeConnectionFactory

MQeConnectionFactory é a implementação MQe da interface javax.jms.ConnectionFactory. Ele é utilizado para gerar instâncias de classes Connection, que para o MQe devem ter uma referência a um gerenciador de filas ativo. A ConnectionFactory deve estar apta a criar uma referência a um gerenciador de filas ativo para transmiti-la para as classes Connection que ela gera. A classe MQeConnectionFactory pode ser configurada para obter uma referência a um gerenciador de filas das seguintes maneiras:

- Pode, ela própria, iniciar um gerenciador de filas de cliente.
- Pode procurar um gerenciador de filas já em execução na JVM.

Entretanto, se nenhuma dessas opções for apropriada, a classe MQeConnectionFactory poderá ser estendida para fornecer o comportamento esperado, consulte “Estendendo MQeConnectionFactory” na página 131.

Para configurar uma connection factory para ela própria iniciar um gerenciador de filas, é necessário fornecer a ela uma referência a um arquivo de inicialização (.ini) que contenha todas as informações necessárias para iniciar o gerenciador de filas. A connection factory é configurada utilizando seu método setIniFileName():

```
(MQeConnectionFactory(factory)).setIniFileName(filename);
```

em que ’filename’ é o nome do arquivo de inicialização. Quando a connection factory tiver sido configurada com o nome do arquivo de inicialização, ela poderá ser armazenada em um diretório JNDI para que possa ser consultada pelos programas aplicativos ou possa ser utilizada diretamente em um programa aplicativo. Quando a connection factory gera sua primeira Connection, ela inicia o gerenciador de filas de cliente utilizando o arquivo de inicialização e transmite uma referência ao gerenciador de filas ativo para a Connection. Se ela gerar classes Connection adicionais, transmitirá a elas uma referência ao mesmo gerenciador de filas ativo. Quando a última Connection é fechada, a connection factory fecha o gerenciador de filas.

Nota: Não utilize os métodos MQeQueueManager.close() para encerrar um gerenciador de filas iniciado por uma connection factory.

Para configurar uma connection factory para procurar um gerenciador de filas existente, o nome do arquivo de inicialização deverá ser configurado como nulo. Esse é o valor padrão quando a classe MQeConnectionFactory é criada e também pode ser configurado explicitamente utilizando o método setIniFileName():

```
(MQeConnectionFactory(factory)).setIniFileName(null);
```

Neste caso, quando a connection factory gera uma Connection, ela procura um gerenciador de filas já em execução na JVM e transmite à Connection uma referência a ele. Uma exceção será emitida se nenhum gerenciador de filas estiver em execução. Se ela gerar classes Connection adicionais, transmitirá a essas classes uma referência ao mesmo gerenciador de filas. Quando um gerenciador de filas externo é utilizado, a connection factory não fecha o gerenciador de filas quando a última Connection é fechada.

Nota: Uma JVM pode executar apenas um gerenciador de filas do MQe por vez. Portanto, se você utilizar uma connection factory para iniciar um gerenciador de filas, ela não deverá ser utilizada para iniciar o mesmo gerenciador de filas em uma JVM diferente, executando na mesma máquina, enquanto o primeiro ainda estiver ativo.

Configurando o MQeJMSQueue

O MQeJMSQueue é a implementação MQe da classe Queue. Ele é utilizado para representar filas do MQe em aplicativos JMS. Ele é configurado por seu construtor:

```
public MQeJMSQueue(String mqeQMgrName, String mqeQueueName) throws JMSException
```

em que:

- *mqeQMgrName* é o nome do gerenciador de filas MQe que possui a fila
- *mqeQueueName* é o nome da fila MQe

Se o nome do gerenciador de filas for nulo, o gerenciador de filas locais será utilizado (ou seja, o gerenciador de filas ao qual o JMS está conectado). Se o nome da fila for nulo, uma JMSException será emitida.

Quando a fila tiver sido configurada, ela poderá ser armazenada em um diretório JNDI para que possa ser procurada por programas aplicativos ou poderá ser utilizada diretamente em um programa aplicativo. Existe uma maneira alternativa de configurar uma fila em um aplicativo, utilizando o método QueueSession.createQueue(). Ele suporta um parâmetro, que é o nome da fila. Para o JMS do MQe, esse pode ser o nome do gerenciador de filas, seguido por um sinal de mais, seguido pelo nome da fila:

```
ioQueue =session.createQueue("myQM+myQueue");
```

ou apenas o nome da fila:

```
ioQueue =session.createQueue("myQueue");
```

Se o nome da fila for utilizado sozinho, assume-se o gerenciador de filas locais.

Nota: O JMS do MQe pode enviar mensagens apenas para uma fila local ou em uma fila remota assíncrona e pode receber mensagens apenas de uma fila local. Ele não pode enviar ou receber mensagens de uma fila remota síncrona.

Ferramenta de Administração do MQe para JMS

A ferramenta de administração fornece uma maneira simples para os administradores definirem e editarem as propriedades dos objetos administrados pelo JMS do MQe.

Essa ferramenta de administração baseia-se naquela fornecida com o JMS para MQ, diferenciando apenas nas propriedades que podem ser aplicadas aos objetos administrados pelo JMS.

A ferramenta de administração JMS está incluída em `MQeJMSAdmin.jar`.

Configurando a Ferramenta de Administração JMS

Você deve configurar a ferramenta de administração com valores para os três parâmetros a seguir:

INITIAL_CONTEXT_FACTORY

Indica o fornecedor de serviços que a ferramenta utiliza. Atualmente, dois valores são suportados para essa propriedade:

- com.sun.jndi.ldap.LdapCtxFactory (para LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (para contexto de sistema de arquivos)

PROVIDER_URL

Indica a URL do contexto inicial da sessão, a raiz de todas as operações JNDI executadas pela ferramenta. Atualmente, dois formatos dessa propriedade são suportados:

- ldap://hostname/contextname (para LDAP)
- file:[drive:]/pathname (para contexto de sistema de arquivos)

SECURITY_AUTHENTICATION

Indica se a JNDI passa pelas credenciais de segurança para seu fornecedor de serviços. Esse parâmetro é utilizado apenas quando um fornecedor de serviços LDAP é utilizado. Atualmente, essa propriedade pode assumir um dos três valores:

- none (autenticação anônima)
- simple (autenticação simples)
- CRAM-MD5 (mecanismo de autenticação CRAM-MD5)

Se não for fornecido um valor válido, a propriedade será assumida como none. Se o parâmetro for configurado como simple ou CRAM-MD5, as credenciais de segurança serão transmitidas por meio da JNDI para o fornecedor de serviços subjacente. Essas credenciais de segurança estão no formato de um nome distinto do usuário (DN do Usuário) e senha. Se as credenciais de segurança forem necessárias, elas serão solicitadas ao usuário quando a ferramenta for inicializada.

Nota: O texto digitado é repetido na tela e isso inclui a senha. Portanto, tome cuidado para que as senhas não sejam reveladas a usuários não autorizados.

Esses parâmetros estão configurados em um arquivo de configuração de texto simples que consiste em um conjunto de pares de chave-valor, separados por um ″=″. Isso é mostrado no seguinte exemplo:

```
#Configurar o fornecedor de serviços
INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Configurar o contexto inicial
PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Configurar o tipo de autenticação
SECURITY_AUTHENTICATION=none
```

(Um ″#″ na primeira coluna da linha indica um comentário ou uma linha que não é utilizada.)

Um arquivo de configuração de exemplo está incluído em `examples/jms/MQeJMSAdmin.config`.

Iniciando a Ferramenta de Administração JMS

Para iniciar a ferramenta no modo interativo, digite o comando:

```
java com.ibm.mqe.jms.admin.MQeJMSAdmin [-cfg config_filename]
```

em que a opção -cfg especifica o nome de um arquivo de configuração alternativo. Se nenhum arquivo de configuração for especificado, a ferramenta procurará um arquivo nomeado MQeJMSAdmin.config no diretório atual.

Após a autenticação, se necessário, a ferramenta exibe um prompt de comandos:

```
InitCtx>
```

indicando que a ferramenta está utilizando o contexto inicial definido no parâmetro de configuração PROVIDER_URL.

Para iniciar a ferramenta no modo de batch, digite o comando:

```
java com.ibm.mqe.jms.admin.MQeJMSAdmin < script.scp
```

em que script.scp é um arquivo de script que contém comandos de administração. O último comando nesse arquivo deve ser um comando END.

Comandos de Administração do JMS

Quando o prompt de comandos é exibido, a ferramenta está pronta para aceitar comandos. Os comandos de administração estão geralmente no seguinte formato:

```
verb [param ]*
```

em que verb é um dos verbos de administração listados na Tabela 25. Todos os comandos válidos consistem em pelo menos (e apenas um) verbo, que aparece no início do comando em seu formato padrão ou abreviado.

Os parâmetros que um verbo pode assumir dependem do verbo. Por exemplo, o verbo END não pode assumir parâmetros, mas o verbo DEFINE pode assumir entre 1 e 20 parâmetros. Os detalhes dos verbos que assumem pelo menos um parâmetro são discutidos posteriormente nesta seção.

Tabela 25. Verbos de Administração

Verbo	Formato Abreviado	Descrição
ALTER	ALT	Alterar pelo menos uma das propriedades de um determinado objeto administrado
DEFINE	DEF	Criar e armazenar um objeto administrado ou criar um novo subcontexto
DISPLAY	DIS	Exibir as propriedades de um ou mais objetos administrados armazenados ou o conteúdo do contexto atual
DELETE	DEL	Remover um ou mais objetos administrados do espaço de nomes ou remover um subcontexto vazio
CHANGE	CHG	Alterar o contexto atual, permitindo que o usuário atravesse o espaço de nomes de diretório em qualquer parte abaixo do contexto inicial (liberação de segurança pendente)
COPY	CP	Fazer uma cópia de um objeto administrado armazenado, armazenando-o com um nome alternativo
MOVE	MV	Alterar o nome com o qual um objeto administrado é armazenado
END		Fechar a ferramenta de administração

Os nomes dos verbos não fazem distinção entre maiúsculas e minúsculas.

Geralmente, para terminar os comandos, você pressiona a tecla de retorno de carro. Entretanto, você pode substituir isso digitando o símbolo ″+″ diretamente antes do retorno de carro. Isso permite digitar comandos em várias linhas, conforme mostrado no exemplo a seguir:

```
DEFINE Q(BookingsInputQueue)+
QMGR(ExampleQM)+
QUEUE(QUEUE.BOOKINGS)
```

As linhas que começam com um dos caracteres *, # ou / são tratadas como comentários.

Manipulando Subcontextos

Você pode utilizar os verbos CHANGE , DEFINE , DISPLAY e DELETE para manipular subcontextos de espaço de nomes de diretório. O uso deles é descrito na tabela a seguir

Tabela 26. Sintaxe e Descrição de Comandos Utilizados para Manipular Subcontextos

Sintaxe de Comando	Descrição
DEFINE CTX(ctxName)	Tenta criar um novo subcontexto filho do contexto atual, com o nome ctxName. Falhará se existir uma violação de segurança, se o subcontexto já existir ou se o nome fornecido for inválido.
DISPLAY CTX	Exibe o conteúdo do contexto atual. Os objetos administrados são anotados com um 'a', os subcontextos com '[D]'. O tipo Java de cada objeto também é exibido.
DELETE CTX(ctxName)	Tenta excluir o contexto filho do contexto atual que possui o nome ctxName. Falhará se o contexto não for localizado, não for vazio ou se houver uma violação de segurança.
CHANGE CTX(ctxName)	Altera o contexto atual, para que agora se refira ao contexto filho que possui o nome ctxName. Um dos dois valores especiais de ctxName podem ser fornecidos: **=UP** que move para o pai do contexto atual **=INIT** que move diretamente para o contexto inicial Falhará se o contexto especificado não existir ou se houver uma violação de segurança.

Administrando Objetos do JMS

Atualmente, dois tipos de objetos podem ser manipulados pela ferramenta de administração. Eles são listados na tabela a seguir:

Tabela 27. Objetos Administrados pelo JMS

Tipo de Objeto	Palavra-chave	Descrição
MQeJNDIQueueConnectionFactory	QCF	A implementação MQe da interface JMS ConnectionFactory. Representa um objeto de depósito de informações do provedor para criar conexões no domínio de mensagem Ponto-a-Ponto do JMS 1.02b.
MQeJNDIConnectionFactory	CF	A implementação MQe da interface JMS ConnectionFactory. Representa um objeto de depósito de informações do provedor para criar conexões no domínio de mensagem unificada do JMS 1.1.

Tabela 27. Objetos Administrados pelo JMS (continuação)

Tipo de Objeto	Palavra-chave	Descrição
MQeJMSJNDIQueue	Q	A implementação MQe da interface JMS Queue. Representa um Destino de mensagem.

Verbos Utilizados com Objetos do JMS

Você pode utilizar os verbos ALTER, DEFINE, DISPLAY, DELETE, COPY e MOVE para manipular objetos administrados no espaço de nomes de diretório. A tabela a seguir resume o uso desses verbos. Substitua TYPE pela palavra-chave que representa o objeto administrado requerido, conforme listado na tabela anterior em "Administrando Objetos do JMS" na página 128.

Tabela 28. Sintaxe e Descrição de Comandos Utilizados para Manipular Objetos Administrados

Sintaxe de Comando	Descrição
ALTER TYPE(name) [property]*	Tenta atualizar as propriedades do objeto administrado especificado com aquelas fornecidas. Falhará se existir uma violação de segurança, se o objeto especificado não puder ser localizado ou se as novas propriedades fornecidas forem inválidas.
DEFINE TYPE(name) [property]*	Tenta criar um objeto administrado do tipo TYPE com as propriedades fornecidas e tenta armazená-lo com o nome name no contexto atual. Falhará se existir uma violação de segurança, se o nome fornecido for inválido ou já existir ou se as propriedades fornecidas forem inválidas.
DISPLAY TYPE(name)	Exibe as propriedades do objeto administrado do tipo TYPE, ligadas sob o nome name no contexto atual. Falhará se o objeto não existir ou se houver uma violação de segurança.
DELETE TYPE(name)	Tenta remover o objeto administrado do tipo TYPE, que possui o nome name, do contexto atual. Falhará se o objeto não existir ou se houver uma violação de segurança.
COPY TYPE(nameA) TYPE(nameB)	Faz uma cópia do objeto administrado do tipo TYPE, que possui o nome nameA, nomeando a cópia nameB. Isso tudo ocorrerá no escopo do contexto atual. Falhará se o objeto a ser copiado não existir, se um objeto com o nome nameB já existir ou se houver uma violação de segurança.
MOVE TYPE(nameA) TYPE(nameB)	Move (renomeia) o objeto administrado do tipo TYPE, que possui o nome nameA, para nameB. Isso tudo ocorrerá no escopo do contexto atual. Falhará se o objeto a ser movido não existir, se um objeto com o nome nameB já existir ou se houver uma violação de segurança.

Criando Objetos do JMS

Os objetos são criados e armazenados em um espaço de nomes JNDI utilizando a seguinte sintaxe de comando:

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

Ou seja, o verbo DEFINE, seguido por uma referência ao objeto administrado TYPE (name), seguido por zero ou mais propriedades.

Nomenclatura LDAP de Objetos do JMS

Para armazenar seus objetos em um ambiente LDAP, os nomes devem estar em conformidade com determinadas convenções. Uma delas é que os nomes de objeto e de subcontexto devem incluir um prefixo, como cn=(common name) ou ou=(organizational unit). A ferramenta de administração simplifica o uso de fornecedores de serviços LDAP, permitindo que você se refira a nomes de objeto e de contexto sem um prefixo. Se você não for fornecer um prefixo, a ferramenta incluirá automaticamente um prefixo padrão (atualmente cn=) no nome fornecido.

Isso é mostrado no exemplo a seguir.

```
InitCtx>DEFINE Q(testQueue)
InitCtx>DISPLAY CTX
Contents of InitCtx

a cn=testQueue com.ibm.mqe.jms.MQeJMSJNDIQueue

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

Observe que, embora o nome do objeto fornecido não tenha um prefixo, a ferramenta inclui algum automaticamente para assegurar conformidade com a convenção de nomenclatura LDAP. Do mesmo modo, enviar o comando `DISPLAY Q(testQueue)` também resulta na inclusão desse prefixo.

Pode ser necessário configurar o servidor LDAP para armazenar objetos Java. As informações para auxiliar nessa configuração são fornecidas em "Definição do Esquema LDAP para Armazenamento de Objetos Java" na página 132.

Propriedades de Objetos do JMS

Uma propriedade consiste em um par nome-valor no formato:

```
PROPERTY_NAME(property_value)
```

Os nomes e valores não fazem distinção entre maiúsculas e minúsculas, mas estão restritos a um conjunto de nomes reconhecidos, mostrados na tabela a seguir:

Tabela 29. Nomes das Propriedades e Valores Válidos

Propriedade	Formato Abreviado	Valores Válidos
AUTHENTICATOR	AUTH	Qualquer Cadeia
CLIENTID	CID	Qualquer Cadeia
DESCRIPTION	DESC	Qualquer Cadeia
DUPSOKCOUNT	DOC	Qualquer inteiro positivo
INIFILE	INI	Qualquer Cadeia
ISMQNATIVE	ISMQ	″True″ ou ″False″
JMXENABLED	JMSX	″True″ ou ″False″
QUEUE	QU	Qualquer Cadeia
QMANAGER	QMGR	Qualquer Cadeia
SHUTDOWN	SHUT	Qualquer inteiro positivo

A maioria dessas propriedades aplica-se apenas a tipos de objetos específicos, mas observe que as propriedades ConnectionFactory também aplicam-se às propriedades QueueConnectionFactory. As propriedades e os tipos aos quais elas se aplicam são listados na tabela a seguir, juntamente com uma descrição abreviada.

Duas colunas indicam as propriedades que se aplicam a **QCF/CF** (QueueConnectionFactory ou ConnectionFactory) e **Q** (Queue).

Tabela 30. Nomes e Descrições das Propriedades

Propriedade	QCF/CF	Q	Descrição
AUTHENTICATOR	S		Nome completo da classe que implementa a interface com.ibm.mqe.jms.MQeJMSAuthenticator.
CLIENTID	S		Um identificador de cadeia para o cliente
DESCRIPTION	S	S	Uma descrição do objeto armazenado
DUPSOKCOUNT	S		O número de mensagens a serem recebidas antes da confirmação em uma Sessão DUPS_OK_ACKNOWLEDGE.
INIFILE	S		Um arquivo de inicialização (.ini) para um Gerenciador de Filas do MQe
ISMQNATIVE		S	O destino é uma fila do MQ, não-JMS.
JMSXENABLED	S		Ativar propriedades do JMSX.
QUEUE		S	O nome de uma fila do MQe
QMANAGER		S	O nome de um gerenciador de filas do MQe
SHUTDOWN	S		Retardo antes do encerramento da conexão, em milissegundos.

Estendendo MQeConnectionFactory

Por padrão, o MQeConnectionFactory procurará um gerenciador de filas já em execução na JVM ou iniciará o seu próprio utilizando um arquivo de inicialização (.ini).

Uma terceira opção é estender MQeConnectionFactory para fornecer o comportamento desejado. O modo preferido para fazer isso é substituir dois métodos internos, `startQueueManager()` e `stopQueueManager()`. O primeiro método é chamado para iniciar e configurar um gerenciador de filas MQe quando uma Conexão é criada pela primeira vez, enquanto o segundo o encerra normalmente quando a Conexão final é fechada. Ambos os métodos são públicos para facilitar suas substituições, mas normalmente não devem ser chamados por um aplicativo.

A classe a seguir mostra um modo simples de estender o MQeConnectionFactory para iniciar seu próprio gerenciador de filas, sem a necessidade de um arquivo de inicialização:

```
import javax.jms.*;
import examples.config.*;
import com.ibm.mqe.jms.MQeConnectionFactory;
import com.ibm.mqe.MQeQueueManager;
import java.io.File;

// digite em uma única linha:
public class MQeExtendedConnectionFactory
         extends MQeConnectionFactory {

  // Nome do Gerenciador de Filas -
  private static final String queueManagerName = "ExampleQM";
  // Local do registro -
  private static final String registryLocation = ".\\ExampleQM";
  // Armazenamento de fila -
  private static final String queueStore = "MsgLog:"
                                         + registryLocation
                                         + File.separator
                                         + "Filas";
  // o Gerenciador de Filas MQe -
  private static MQeQueueManager queueManager = null;
```

```
  public MQeQueueManager startQueueManager() throws JMSException {
    try {
      CreateQueueManager.createQueueManagerDefinition(
      queueManagerName, registryLocation, queueStore);
      queueManager=CreateQueueManager.startQueueManager(
      queueManagerName, registryLocation);
    }
    catch (Exception e) {
      JMSException je = new JMSException("QMgr start  failed");
                  je.setLinkedException(e);
                  throw je;
    }
  return queueManager;
  }

  public void stopQueueManager() throws Exception {
    CreateQueueManager.stopQueueManager(queueManager);
  }

}
```

Neste exemplo, a inicialização e o encerramento reais do gerenciador de filas foram delegados aos exemplos CreateQueueManager descritos em um capítulo anterior.

Definição do Esquema LDAP para Armazenamento de Objetos Java

Esta seção fornece detalhes das definições de esquema (definições de atributo e objectClass) necessárias em um diretório LDAP para que ele armazene objetos Java. Elas serão requeridas se você desejar utilizar um servidor LDAP como seu fornecedor de serviços JNDI para armazenar objetos administrados pelo JMS do MQe.

Alguns servidores podem já conter essas definições em seus esquemas. O procedimento exato para verificar se o servidor contém essas definições e incluí-las, caso não contenha, varia de um servidor para outro. Leia a documentação fornecida com o servidor LDAP e o fornecedor de serviços JNDI LDAP.

Muitos dos dados contidos nesta seção foram extraídos do *RFC 2713 Schema for Representing Java Objects in an LDAP Directory*, que pode ser localizado em http://www.faqs.org/rfcs/rfc2713.html.

Observe que, em alguns servidores LDAP, pode ser necessário desativar a verificação de esquemas, mesmo depois que essas definições tiverem sido incluídas.

Definições de Atributos

Tabela 31. Configurações de Atributos para javaCodebase

Atributo	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.1.7
Sintaxe	Cadeia IA5 (1.3.6.1.4.1.1466.115.121.1.26)
Comprimento máximo	2.048
Valor único/múltiplo	Multi-avaliado
Pode ser modificado pelo usuário?	Sim
Regras de correspondência	caseExactIA5match
Classe de acesso	Normal
Uso	userApplications
Descrição	URL(s) especificando o local da definição de classe

Tabela 32. Configurações de Atributos para javaClassName

Atributo	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.1.6
Sintaxe	Cadeia de Diretório (1.3.6.1.4.1.1466.115.121.1.15)
Comprimento máximo	2.048
Valor único/múltiplo	Valor único
Pode ser modificado pelo usuário?	Sim
Regras de correspondência	caseExactMatch
Classe de acesso	Normal
Uso	userApplications
Descrição	Nome completo da classe ou interface Java distinta

Tabela 33. Configurações de Atributos para javaClassNames

Atributo	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.1.13
Sintaxe	Cadeia de Diretório (1.3.6.1.4.1.1466.115.121.1.15)
Comprimento máximo	2.048
Valor único/múltiplo	Multi-avaliado
Pode ser modificado pelo usuário?	Sim
Regras de correspondência	caseExactMatch
Classe de acesso	Normal
Uso	userApplications
Descrição	Nome completo da classe ou interface Java

Tabela 34. Configurações de Atributos para javaFactory

Atributo	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.1.10
Sintaxe	Cadeia de Diretório (1.3.6.1.4.1.1466.115.121.1.15)
Comprimento máximo	2.048
Valor único/múltiplo	Valor único
Pode ser modificado pelo usuário?	Sim
Regras de correspondência	caseExactMatch
Classe de acesso	Normal
Uso	userApplications
Descrição	Nome completo da classe Java de um Depósito de Informações do Provedor de Objetos JNDI

Tabela 35. Configurações de Atributos para javaReferenceAddress

Atributo	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.1.11
Sintaxe	Cadeia de Diretório (1.3.6.1.4.1.1466.115.121.1.15)
Comprimento máximo	2.048
Valor único/múltiplo	Multi-avaliado

Tabela 35. Configurações de Atributos para javaReferenceAddress (continuação)

Atributo	**Valor**
Pode ser modificado pelo usuário?	Sim
Regras de correspondência	caseExactMatch
Classe de acesso	Normal
Uso	userApplications
Descrição	Endereços associados a uma Referência JNDI

Tabela 36. Configurações de Atributos para javaSerializedData

Atributo	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.1.8
Sintaxe	Cadeia de Octeto (1.3.6.1.4.1.1466.115.121.1.40)
Valor único/múltiplo	Valor único
Pode ser modificado pelo usuário?	Sim
Classe de acesso	Normal
Uso	userApplications
Descrição	Formato serializado de um objeto Java

Definições objectClass

Tabela 37. Definição objectClass para javaSerializedObject

Definição	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.2.5
Extensões/superior	javaObject
Tipo	AUXILIAR
Atributos requeridos	javaSerializedData

Tabela 38. Definição objectClass para javaObject

Definição	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.2.4
Extensões/superior	Início
Tipo	ABSTRATO
Atributos requeridos	javaClassName
Atributos opcionais	Descrição de javaClassNames, javaCodebase, javaDoc

Tabela 39. Definição objectClass para javaContainer

Definição	**Valor**
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.2.1
Extensões/superior	Início
Tipo	ESTRUTURAL
Atributos requeridos	cn

Tabela 40. Definição objectClass para javaNamingReference

Definição	Valor
OID (Object Identifier)	1.3.6.1.4.1.42.2.27.4.2.7
Extensões/superior	javaObject
Tipo	AUXILIAR
Atributos opcionais	attrs javaReferenceAddress javaFactory

Interface do JMX (Java Management Extensions)

Esta seção descreve a interface **MQe Java Management Extensions**. O nome é abreviado no texto como **MQe JMX**.

A interface do MQe JMX fornece um nível de instrumentação JMX para recursos do MQe (gerenciadores de filas, filas e etc). Esse nível de instrumentação JMX facilita a configuração e a administração locais e remotas de gerenciadores de filas do MQe e seus objetos associados, como filas, conexões, listeners e objetos de ponte. Para validar a operação da rede, mensagens de teste podem ser enviadas para filas na rede MQe.

Esse recurso facilita o gerenciamento de recursos do MQe através do JMX a partir de todas as plataformas suportadas para o JMX. Para obter informações adicionais sobre o suporte a plataformas para JMX, consulte a especificação JMX V1.2 em

`http://jcp.org/aboutJava/communityprocess/final/jsr003/index3.html`

.

(Observe que os gerenciadores de filas e seus recursos em plataformas que não são suportadas pelo JMX podem ser ainda administrados remotamente por meio do JMX.)

Para aqueles que estão familiarizados com a ferramenta MQe_Explorer (anteriormente empacotada no IBM SupportPac ES02) ou a ferramenta MQe_Script (anteriormente empacotada no IBM SupportPac ES04), a interface do MQe JMX facilita a funcionalidade configurativa e administrativa equivalente. Os objetos criados utilizando o MQe_Script, MQe_Explorer (versão 2 ou posterior) e a interface do MQe JMX podem interagir juntos. Agora esses SupportPacs são empacotados juntos como ES06 MQe Server Support SupportPac.

A interface do MQe JMX inclui suporte completo para a configuração e o gerenciamento de gerenciadores de filas de gateway, ou seja, aqueles gerenciadores de filas do MQe que podem servir de ponte para os gerenciadores de filas e as filas do MQ. A interface não suporta a configuração dos próprios gerenciadores de filas MQ, pois eles devem ser configurados utilizando os diversos protocolos e ferramentas de gerenciamento do MQ.

Introdução ao MQe JMX

O Java Management Extensions (também chamado de especificação JMX) define uma arquitetura, os padrões de design, as APIs e os serviços para gerenciamento de aplicativo e de rede na linguagem de programação Java. A especificação JMX oferece aos desenvolvedores de Java em todos os segmentos de mercado um meio para instrumentar o código Java, criar agentes Java inteligentes, implementar gerenciadores e middleware de gerenciamento distribuído, além de integrar facilmente essas soluções aos sistemas de gerenciamento existentes.

A arquitetura JMX fornece os seguintes benefícios:

- Permite que os aplicativos Java sejam gerenciados sem grandes investimentos

Um aplicativo Java precisa simplesmente incorporar um servidor de objetos gerenciados e disponibilizar sua funcionalidade como um ou vários Beans Gerenciáveis registrados no servidor de objetos; isso é tudo para se beneficiar da infra-estrutura de gerenciamento.

- Fornece uma arquitetura de gerenciamento escalável

  Cada serviço do agente JMX é um módulo independente que pode ser conectado ao agente de gerenciamento, dependendo dos requisitos. Essa abordagem baseada em componentes significa que as soluções JMX podem escalar de pequenos dispositivos de base a grandes comutadores de telecomunicações e superiores.

- Integra soluções de gerenciamento existentes

  Os agentes inteligentes JMX são capazes de serem gerenciados por meio de navegadores HTML ou por vários protocolos de gerenciamento, como SNMP e WBEM. As APIs do JMX são interfaces abertas que podem ser alavancadas por qualquer fornecedor de sistema de gerenciamento.

- Pode alavancar conceitos de gerenciamento futuro

  As APIs da especificação JMX podem implementar soluções de gerenciamento flexível e dinâmico por meio da linguagem de programação Java, que pode alavancar tecnologias emergentes.

  O objetivo da API do JMX para MQe é fornecer um nível de instrumentação JMX para recursos do MQe (gerenciadores de filas, filas e etc). A instrumentação está projetada para ter uma base pequena e para ser flexível, fácil de ser utilizada e em conformidade com o JMX.

- Base pequena

  A API do JMX para MQe e a implementação do JMX minimizam as demandas de recursos em termos de requisitos de tamanho e memória.

- Flexível

  A implementação é modular para que os desenvolvedores do MQe possam optar por utilizar a API quando a gerenciabilidade é desejável e o código extra da implementação do JMX está dentro das capacidades de suas plataformas de destino. Ou eles podem optar por deixá-la de lado sem incorrer em qualquer perda de memória ou desempenho.

- Fácil de utilizar

  A API é simples e fácil de ser utilizada: é possível ativar um aplicativo MQe existente para JMX com apenas algumas linhas de código.

- Conformidade com JMX

  A conformidade com o JMX é garantida virtualmente, utilizando a implementação de referência desenvolvida pela Sun Microsystems. Do JMX v.1.2 em diante, os MBeans abertos são uma parte obrigatória de qualquer implementação do JMX, de maneira que aderimos aos tipos de dados requeridos para a instrução MBean aberta. Portanto, todos os parâmetros e atributos da operação são dos seguintes tipos de dados:

  – Tipos de dados simples:
    - java.lang.Void
    - java.lang.Boolean
    - java.lang.Byte
    - java.lang.Character
    - java.lang.String
    - java.lang.Short
    - java.lang.Integer
    - java.lang.Long
    - java.lang.Float
    - java.lang.Double

  – Matrizes dos tipos anteriores:
    - javax.management.ObjectName
    - javax.management.openmbean.CompositeData
    - javax.management.openmbean.TabularData

Arquitetura JMX

A arquitetura do JMX possui várias camadas, conforme mostrado no diagrama a seguir:

<table>
<tr><td rowspan="2">Nível de Serviços Distribuídos</td><td>Navegador da Web</td><td>Outros</td><td>Aplicativos de gerenciamento em conformidade com o JMX</td><td>Aplicativos de gerenciamento proprietário</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td rowspan="4">Nível de Agente</td><td colspan="2" rowspan="2">Adaptadores de protocolos</td><td rowspan="2">Conectores...</td><td>Gerenciador do JMX</td></tr>
<tr><td>..</td></tr>
<tr><td colspan="3" rowspan="2">MBeanServer...</td><td>...</td></tr>
<tr><td>Serviços do Agente</td></tr>
<tr><td rowspan="2">Nível de Instrumentação</td><td colspan="4">Estratégia de instrumentação (implementação do MQe JMX)</td></tr>
<tr><td colspan="4">Recursos de Aplicativos</td></tr>
</table>

O escopo da implementação do MQe JMX está limitado ao *Nível de instrumentação* e à **Estratégia de instrumentação (implementação do MQe JMX)**.

A seguir estão explicações de alguns termos utilizados no diagrama:

Nível de Serviços Distribuídos
: O nível de serviços distribuídos da arquitetura JMX contém o middleware que conecta os agentes aos aplicativos de gerenciamento.

Nível de Agente
: O nível de agente da arquitetura JMX fornece um registro para manipular os recursos gerenciáveis, denominados MBeanServer, bem como vários serviços do agente que são os próprios MBeans.

Agente JMX
: Um agente JMX é uma combinação de uma instância do MBeanServer, seus MBeans registrados e quaisquer serviços do agente em uma única JVM.

MBeans (Beans Gerenciados)
: Recursos instrumentados de acordo com as regras da especificação JMX. Existem duas categorias principais de MBeans:
  - Os *MBeans Padrão* implementam suas próprias interfaces e são estáticos.
  - Os *MBeans Dinâmicos* (dos quais existem várias subcategorias) implementam uma interface do JMX chamada DynamicMBean. Essa interface contém métodos que permitem que a interface de gerenciamento do recurso gerenciado seja descoberta no tempo de execução.

Nível de Instrumentação
: O nível de instrumentação da arquitetura JMX é o nível no qual os recursos a serem gerenciados são instrumentados para gerenciamento do JMX. Para tornar isso possível, os recursos devem ser instrumentados como MBeans.

Recurso
: Qualquer entidade que precise ser monitorada ou controlada por um aplicativo de gerenciamento. No contexto dessa implementação, gerenciadores de filas do MQe, filas e etc são todos recursos.

Instrumentando os Recursos do MQe como MBeans JMX

Em seu aplicativo, carregue ou crie o gerenciador de filas e ative-o de modo normal. Agora, crie os MBeans JMX para o gerenciador de filas e seus recursos, conforme a seguir:

```
MQeQueueManagerJmx.createMQeMBeans(mbServer);
```

em que mbServer é a sua instância do MBeanServer.

Esse método cria e registra MBeans para todos os recursos do gerenciador de filas: filas, objetos de ponte, conexões e listeners.

Nota: Sempre que os recursos do MQe são criados ou removidos a partir desse ponto, os MBeans correspondentes também são registrados ou têm o registro removido da instância MBeanServer. É altamente recomendável não criar e registrar ou remover o registro de MBeans do MQe, independentemente de utilizar a interface do MQe, caso contrário, é provável que as representações de MBean do MQe fiquem inconsistentes e não funcionem conforme desejado. Por exemplo, você não deve utilizar os recursos registrar/remover registro oferecidos por vários adaptadores. Utilizar a interface do MQe JMX para criar e excluir métodos assegura que os MBeans sejam registrados e tenham o registro removido do modo aprovado. Entretanto, o AdminBean é uma exceção a essa regra – consulte "ObjectName" na página 140.

É necessário criar um ou mais conectores ou adaptadores para permitir que os clientes de gerenciamento do JMX se conectem e gerenciem seus aplicativos do MQe. As Implementações de Referência JMX da Sun e da Tivoli fornecem adaptadores que permitem gerenciar o aplicativo MQe por meio de um navegador da Web. Consulte as implementações de referência para obter a documentação e exemplos.

Além do HtmlAdaptorServer, o Sun JDMK fornece o HttpConnectorServer, HttpsConnectorServer, RmiConnectorServer e SnmpAdaptorServer. Eles permitem que os clientes de gerenciamento do JMX se conectem a recursos gerenciáveis do JMX e os gerenciem utilizando os protocolos HTTP, HTTPS, RMI e SNMP. Consulte o JDMK para obter a documentação e os exemplos.

Depois que tiver o(s) conector(es) ou adaptador(es), ou ambos, você está em condições de acessar os MBeans do MQe, conforme especificado na especificação JMX. É necessário ter todas as seguintes filas configuradas:

- Uma fila admin no gerenciador de filas locais para administração local. O padrão assume que essa fila esteja nomeada como AdminQ, mas você pode reconfigurá-la utilizando o MBean Admin.
- Uma fila de resposta admin chamada AdminReplyQ.
- As filas nomeadas AdminQ e AdminReplyQ em quaisquer gerenciadores de filas remotas que você deseja gerenciar por meio da interface do JMX. Se uma dessas filas não existir (ou as definições de conexão e listeners relevantes para comunicação bidirecional remota do admin-adminReply não existirem), você poderá ter problemas ao desempenhar a administração remota.

Quando você tiver fechado o gerenciador de filas do MQe no final do aplicativo, deverá chamar o seguinte método estático para assegurar que todos os recursos do MQe JMX tenham sido limpos:

```
MQeQueueManagerJmx.endMQeJMXSession()
```

É importante que esse método seja chamado após o fechamento do gerenciador de filas.

Convenções Tipográficas Nesta Documentação do JMX

O texto entre os sinais de maior e menor e em itálico; por exemplo, *<Nome_do_Gerenciador_de_Filas>*; representa um nome simbólico, o valor do qual deve ser substituído por um valor fornecido pelo usuário quando o comando é digitado.

O texto escrito utilizando uma fonte monoespaçada; por exemplo, `getMBeanInfo`; representa entrada do usuário, código em arquivos ou texto digitado em uma caixa de texto do navegador.

Configurando a Interface do MQe JMX

A interface do MQe JMX é executada como um aplicativo em execução em uma JVM (Java Virtual Machine). Tudo o que é requerido está em um gerenciador de filas ativo local. Dado isso, a interface pode, então, gerenciar o gerenciador de filas locais instrumentado e os recursos do gerenciador de filas.

Ela também pode gerenciar quaisquer gerenciadores de filas do MQe ativados remotamente (e seus recursos), para os quais o gerenciador de filas locais está apto a conectar-se diretamente à rede do MQe.

Nota: Antes de poder utilizar o JMX, você deve certificar-se de que determinados arquivos de propriedades existam no caminho de classe. Consulte "Tradução" na página 154, para obter detalhes.

MQe JMX

- Uma JVM versão 1.2 ou posterior
- Uma implementação de conformidade da especificação JMX.

Os arquivos jar fornecidos pela implementação da especificação JMX devem ser incluídos à CLASSPATH antes de tentar utilizar as APIs da interface do MQe JMX. A Implementação de Referência do JMX, fornecida pela Sun, está disponível e é redistribuível livremente (`http://java.sun.com/products/JavaManagement/`). Para instalá-la:

1. Faça download do código binário da Implementação de Referência, fornecido em um arquivo ZIP
2. Extraia o conteúdo em um diretório
3. Copie o lib/jmxri.jar e o lib/jmxtools.jar no diretório de extensão do Java Runtime Environment ou certifique-se de que estejam no caminho de classe.

A API do JMX para MQe foi desenvolvida em conformidade com a especificação JMX v.1.2.

A Implementação Tivoli da especificação JMX também está disponível livremente (`http://www.alphaworks.ibm.com/tech/TMX4J`). Para instalá-la:

1. Faça download do código binário, fornecido em um arquivo ZIP
2. Extraia o conteúdo em um diretório
3. Copie os jars relevantes para sua plataforma no diretório de extensão de seu Java Runtime Environment ou certifique-se de que estejam em seu caminho de classe.

Ativando Aplicativos do MQe para Gerenciamento do JMX

Para ativar os aplicativos do MQe para o gerenciamento do JMX:

1. Obtenha uma implementação de conformidade da especificação JMX e configure o ambiente de desenvolvimento Java para que as bibliotecas de classes JMX e MQe JMX fiquem acessíveis (conforme descrito em "Configurando a Interface do MQe JMX" na página 138).
2. Crie o aplicativo MQe do modo normal, assegurando que o gerenciador de filas seja carregado, em seguida, chame o seguinte método estático, transmitindo a ele sua instância MBeanServer, para criar MBeans para todos os recursos do gerenciador de filas:

   ```
   MQeQueueManagerJmx.createMQeMBeans(MBeanServer mbServer);
   ```

   Cada um dos MBeans é registrado com sua instância MBeanServer. Para obter detalhes adicionais, consulte "Instrumentando os Recursos do MQe como MBeans JMX" na página 137.
3. Crie um ou mais conectores ou adaptadores para permitir que os clientes de gerenciamento do JMX se conectem e gerenciem seus aplicativos do MQe. Para obter detalhes adicionais sobre como criar conectores e adaptadores, consulte "Instrumentando os Recursos do MQe como MBeans JMX" na página 137. Agora você está em condições de gerenciar todos os recursos do MQe por meio do JMX utilizando a interface correspondente ao conector ou adaptador escolhido. Também é muito importante chamar o seguinte método estático no final do aplicativo para assegurar que todos os recursos do MQe JMX estejam limpos:

   ```
   MQeQueueManagerJmx.endMQeJMXSession()
   ```

Acessando os MBeans do MQe por Meio do MBeanServer

A interface do MQe JMX fornece o nível de instrumentação da arquitetura JMX. Nós não estamos fornecendo uma implementação do nível de agente, que é constituída do MBeanServer e dos serviços do

agente JMX. Como a função do servidor MBean é agir como um registro para MBeans, a instância do usuário do servidor MBean precisa ser transmitida para o código de instrumentação por meio do método MQeQueueManagerJmx.createMQeMBeans() descrito nesta seção. Todos os MBeans do MQe instrumentados são, então, registrados com essa instância MBeanServer.

Uma instância de um MBeanServer é criada utilizando um dos dois métodos estáticos da classe MBeanServerFactory: `createMBeanServer()` ou `newMBeanServer()`. Após a criação dessa instância, os métodos MBeanServer podem ser utilizados para acessar e manipular os MBeans registrados com o MBeanServer. Em particular, os métodos a seguir permitem que o usuário recupere e configure os atributos MBean do MQe e chame as operações. (Os nomes dos métodos a seguir assumem que `mbeanServer` seja a instância de MBeanServer.)

```
mbeanServer.getAttribute(ObjectName objName, String attributeName);
mbeanServer.getAttributes(ObjectName objName, String[] attributeNames);
mbeanServer.setAttribute(ObjectName objName, Attribute attribute);
mbeanServer.setAttributes(ObjectName objName, AttributeList attributes);
mbeanServer.invoke( ObjectName name, String operationName,
Object params[], String signature[]  );
```

Para obter detalhes adicionais sobre os tipos de parâmetros *Atributo* e *AttributeList*, consulte “Informações Relacionadas no JMX” na página 155. O conceito de ObjectName de um MBean é central para a interface do MQe JMX e é discutido na seção seguinte.

ObjectName

Uma instância MBeanServer interage com os MBeans registrados com ela por meio de seus ObjectNames. Quando um MBean é registrado com um MBeanServer, a instância do objeto MBean e a instância do ObjectName MBean correspondente são transmitidas como parâmetros para o método de registro. Deste ponto em diante, o ObjectName é transmitido a todos os métodos MBeanServer pertencentes a esse MBean.

Os ObjectNames também são retornados de métodos de consulta na instância MBeanServer que são projetados para consultar os MBeans registrados com o MBeanServer. Uma forma de correspondência padrão pode ser utilizada nesses métodos. Portanto, a hierarquia de ObjectName correspondente aos recursos instrumentados do MQe foi projetada para facilitar as consultas sobre os tipos de recursos do MQe, como Fila do Aplicativo e Conexão Indireta.

Um nome de objeto consiste em uma cadeia formada de dois componentes: o nome do domínio e a lista de propriedades da chave. Ele tem o formato:

```
Domain-name:key1=value1[,key2=value2,...keyX=valueX]
```

Um nome de domínio corresponde a um prefixo de espaço de nomes que identifica um grupo de recursos do MQe.

A tabela a seguir fornece uma descrição completa das convenções de nomenclatura de objetos MQe JMX.

Tabela 41. Convenções de Nomenclatura de Objetos MQe JMX

Recurso do MQe	ObjectName
Gerenciador de Filas Locais	`com.ibm.MQe_LocalQueueManager:name = <Nome_do_Gerenciador_de_Filas>`
Gerenciador de Filas Remotas	`com.ibm.MQe_RemoteQueueManagers:name = <Nome_do_Gerenciador_de_Filas>`
Alias do Gerenciador de Filas Locais	`com.ibm.MQe_LocalQueueManager:name = <Nome_do_Alias_do_Gerenciador_de_Filas>, type = alias, resourceName = <Nome_do_Gerenciador_de_Filas>`

Tabela 41. Convenções de Nomenclatura de Objetos MQe JMX (continuação)

Recurso do MQe	ObjectName
Alias do Gerenciador de Filas Remotas	com.ibm.MQe_RemoteQueueManagers:name = *<Nome_do_Alias_do_Gerenciador_de_Filas>*, type = alias, resourceName = *<Nome_do_Gerenciador_de_Filas>*
Fila do Aplicativo	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ApplicationQueues:name = *<Nome_da_Fila>*
Alias da Fila do Aplicativo	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ApplicationQueues:name = *<Alias_da_Fila>*, type = alias, resourceName = *<Nome_da_Fila@Nome_do_Gerenciador_de_Filas_Proprietário>*
Fila Síncrona do Proxy	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_SyncProxyQueues:name = *<Nome_da_Fila>*, DestinationQMgr = *<Nome_do_Gerenciador_de_Filas_de_Destino>*
Alias da Fila Síncrona do Proxy	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ SyncProxyQueues:name = *<Alias_da_Fila>*, type = alias, resourceName = *<Nome_da_Fila@Nome_do_Gerenciador_de_Filas_Proprietário>*
Fila Assíncrona do Proxy	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ AsyncProxyQueues:name = *<Nome_da_Fila>*, DestinationQMgr = *<Nome_do_Gerenciador_de_Filas_de_Destino>*
Alias da Fila Assíncrona do Proxy	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ AsyncProxyQueues:name = *<Alias_da_Fila>*, type = alias, resourceName = *<Nome_da_Fila@Nome_do_Gerenciador_de_Filas_Proprietário>*
Fila de Administração	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_AdminQueues:name = *<Nome_da_Fila>*
Alias da Fila Admin	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ AdminQueues:name = *<Alias_da_Fila>*, type = alias, resourceName = *<Nome_da_Fila@Nome_do_Gerenciador_de_Filas_Proprietário>*
Fila de Servidor Home	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_HomeServerQueues:name = *<Nome_da_Fila>*, GetFromQMgr = *<Receber_do_Gerenciador_de_Filas>*
Fila de Armazenamento	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_StoreQueues:name = *<Nome_da_Fila>*
Fila de Redirecionamento	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_ForwardQueues:name = *<Nome_da_Fila>*, ForwardToQMgr = *<Nome_do_Gerenciador_de_Filas_de_Redirecionamento>*
Fila do MQeMQBridge	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_BridgeQueues:name = *<Nome_da_Fila>*, DestinationQMgr = *<Nome_do_Gerenciador_de_Filas_de_Destino>*
Alias da Fila do MQeMQBridge	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_BridgeQueues:name = *<Alias_da_Fila>*, type = alias, resourceName = *<Nome_da_Fila@Nome_do_Gerenciador_de_Filas_Proprietário>*
Conexão do Alias	com.ibm.MQe_*<Nome_do_Gerenciador_de_Filas_Proprietário>*_MQConnections:name = *<Nome_da_Conexão>*

Tabela 41. Convenções de Nomenclatura de Objetos MQe JMX (continuação)

Recurso do MQe	**ObjectName**
Alias de Conexão do Alias	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_MQConnections:name = <*Nome_do_Alias*>, type = alias, resourceName = <*Nome_da_Conexão*>
Conexão Direta	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_DirectConnections:name = <*Nome_da_Conexão*>
Alias de Conexão Direta	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_DirectConnections:name = <*Nome_do_Alias*>, type = alias, resourceName = <*Nome_da_Conexão*>
Conexão Indireta	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_IndirectConnections:name = <*Nome_da_Conexão*>
Alias de Conexão Indireta	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_IndirectConnections:name = <*Nome_do_Alias*>, type = alias, resourceName = <*Nome_da_Conexão*>
Conexão do MQ	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_MQConnections:name = <*Nome_da_Conexão*>
Alias de Conexão do MQ	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_MQConnections:name = <*Nome_do_Alias*>, type = alias, resourceName = <*Nome_da_Conexão*>
Listener de Comunicação	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_CommunicationsListeners:name = <*Nome_do_Listener*>
Ponte do MQ	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_Bridges:name = <*Nome_da_Ponte*>
QMgrProxy do MQ	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_MQQueueManagerProxies:name = <*Nome_do_Proxy*>, bridge = <*Nome_da_Ponte*>
Conexão do Cliente do MQ	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_MQClientConnections:name = <*Nome_da_Conexão_do_Cliente*>, bridge = <*Nome_da_Ponte*>, qmgrProxy = <*Nome_do_Proxy*>
Listener do MQ	com.ibm.MQe_<*Nome_do_Gerenciador_de_Filas_Proprietário*>_MQListeners:name = <*Nome_do_Listener*>, bridge = <*Nome_da_Ponte*>, qmgrProxy = <*Nome_do_Proxy*>, clientConnection = <*Nome_da_Conexão_do_Cliente*>
Bean Admin do MQe	com.ibm.MQe_Admin:name = AdminBean

No aplicativo, utilizando esse esquema, as consultas podem ser feitas nos MBeans utilizando os campos de nome ou tipo ou curingas na cadeia que precede os dois-pontos (essa cadeia é conhecida como Domínio). Assim, fica fácil procurar todas as filas do aplicativo, todas as filas do proxy, todos os aliases de conexão e etc.

Existem alguns pontos importantes a serem observados sobre o uso de ObjectNames para recursos do MQe:

- Não há espaços nesses nomes porque isso dificulta as consultas. Portanto, você deve assegurar que não sejam inseridos espaços por engano nos nomes dos objetos utilizados como parâmetros para os métodos; caso contrário, ocorrerão exceções devido ao recurso não localizado.
- Os nomes dos objetos fazem distinção entre maiúsculas e minúsculas.
- Quando uma fila possui um alias, a chave da propriedade resourceName no nome do objeto MBean do alias possui um valor que é composto de uma cadeia no formato queueName@queueManagerName.

A interface do MQe JMX fornece um método auxiliar para instrumentar os recursos de um gerenciador de filas como MBeans (o método createMQeMBeans() do MQeQueueManagerJmx). Quando esse método é utilizado, todos os MBeans possuem nomes de objetos após o padrão especificado acima. Entretanto, também seria possível para um aplicativo instanciar as instâncias de MBeans do MQe chamando o construtor apropriado e seria possível, então, registrar o MBean resultante utilizando o MBeanServer com um nome de objeto escolhido pelo aplicativo.

É altamente recomendável aderir às convenções de nomenclatura descritas acima e não registrar ou remover o registro de MBeans do MQe, independentemente da interface do MQe JMX. O método auxiliar MQeQueueManagerJmx.createMQeMBeans() da interface do MQe JMX deve ser sempre utilizado para instrumentar os recursos do MQe como MBeans.

O único caso em que a chamada do construtor para criar um MBean do MQe é suportada é para alterar os padrões do MBean Admin. Isso pode ser visto no código de exemplo fornecido no diretório `examples\jmx`. Por exemplo, os recursos registrar/remover registro oferecidos por diversos adaptadores não devem ser utilizados - o uso dos métodos de criação e exclusão da interface do MQe JMX assegura que os MBeans sejam registrados e tenham o registro removido de um modo aprovado (o AdminBean é uma exceção a essa regra).

Essas convenções são utilizadas no código do MQe e o processamento de MBeans pode não ser consistente se um padrão de nomenclatura diferente for utilizado. O uso do método auxiliar assegura uma consistência de comportamento, por exemplo, quando ocorre uma atualização em MBeans do gerenciador de filas remotas devido à inclusão ou remoção de um recurso por algum método que não seja através do JMX.

Os padrões de nomes de objetos descritos acima foram selecionados visando facilitar as consultas na instância MBeanServer para seus MBeans registrados. Essas consultas permitem a correspondência de padrões com base nos nomes dos objetos. (Consulte a documentação do recurso JMX listada no prefácio para obter descrições de como as consultas do MBeanServer funcionam). Por exemplo, para obter um subconjunto de MBeans registrados que correspondam às filas do aplicativo do gerenciador de filas locais, a seguinte consulta pode ser feita:

```
// configurar um filtro para recuperar MBeans da Fila do Aplicativo do MyLocalQM
ObjectName scope =
   new ObjectName("com.ibm.MQe_MyLocalQM_ApplicationQueues:*");

// utilizar a API MBeanServer do JMX para fazer a consulta
Set results = mbeanServer.queryNames(scope,null);

// iterar através dos resultados
Iterator iter = results.iterator();
ObjectName objName = null;
while(iter.hasNext()) {
objName = (ObjectName)iter.next();
  // processar cada resultado
    ...
}
```

O exemplo a seguir mostra como descobrir quais recursos realmente representam aliases da fila:

```
// configurar um filtro para recuperar todos os aliases das filas
ObjectName scope = new ObjectName("*Queues:*,type=alias");
```

```
// utilizar a API MBeanServer do JMX para fazer a consulta
Set results = mbeanServer.queryNames(scope,null);

// etc.
```

Métodos Úteis do MBeanServer

Depois de ter chamado o método auxiliar `MQeQueueManagerJmx.createMQeMBeans()`, que instrumenta todos os recursos do gerenciador de filas como MBeans, você está em condições de manipular esses recursos utilizando a API MBeanServer padrão.

Em termos gerais, essa manipulação envolve a configuração ou recebimento de atributos de recursos MBean ou a chamada de operações de recursos MBean.

Toda manipulação de atributo e operação nesse nível é feita por meio da API da camada de agente a seguir.

getMBeanInfo:

```
public MBeanInfo getMBeanInfo(ObjectName objName)
  throws InstanceNotFoundException,
         IntrospectionException,
         ReflectionException;
```

Para utilizar alguns dos outros métodos MBeanServer descritos nesta seção, como o método `invoke()`, você precisa de informações sobre os parâmetros de entrada relevantes. Por exemplo, pode ser necessário saber quais operações podem ser chamadas para um determinado recurso, quais parâmetros de entrada são requeridos para uma operação específica, qual o tipo de cada parâmetro e qual o tipo de valor de retorno.

Existem duas maneiras de obter essas informações. Primeiramente, você pode utilizar a lista de atributos e operações para cada recurso instrumentado pelo MQe JMX. Você pode consultar as informações necessárias e atribuir a elas o código permanente em um aplicativo, conforme necessário.

Alternativamente, você pode utilizar o método getMBeanInfo() MBeanServer que a camada de agente fornece para recuperar as informações do parâmetro de entrada. Esse método assume como seu único parâmetro a instância ObjectName que corresponde ao recurso equivalente do MQe. O método retorna uma estrutura complexa que contém informações sobre as seguintes propriedades de um MBean: nome da classe, descrição, atributos, construtores, operações e notificações.

As informações que o método getMBeanInfo() retorna sobre atributos, construtores, operações e notificações consistem em estruturas adicionais dos tipos MBeanAttributeInfo, MBeanConstructorInfo, MBeanOperationInfo e MBeanNotificationInfo. O método também pode recuperar uma instância MBeanParameterInfo que corresponda a cada instância MBeanOperationInfo e assim por diante.

A seguir está um exemplo de como utilizar o método getMBeanInfo(). Devido à complexidade do objeto MBeanInfo, também pode ser útil consultar as fontes de informações do JMX listadas na seção Material Relacionado.

Suponha que você saiba que um MBean instrumentado da fila do aplicativo do MQe tenha um método addAlias, mas deseja verificar o tipo de retorno desse método. Para fazer isso, você utilizaria o método getMBeanInfo() da seguinte forma:

```
/* chamar o método getMBeanInfo() em             */
/*  a instância MBeanServer para o MBean da fila  */
MBeanInfo beanInfo = mbeanServer.getMBeanInfo(queueObjName);

/* recuperar informações sobre operações para esse MBean */
MBeanOperationInfo[] beanOps = beanInfo.getOperations();

/* efetuar loop através das operações até localizar aquele desejado */
```

```
for(int i = 0; i < beanOps.length; i++) {
    if(beanOps[i].getName().equals("addAlias")) {
      /* obter o tipo de retorno para essa operação */
      String retval = beanOps[i].getReturnType();
      System.out.println(retval);
      break;
    }
```

Um aspecto muito útil dessas estruturas de informações é o parâmetro de descrição. Todas as instâncias de MBeanAttributeInfo, MBeanParameterInfo, MBeanConstructorInfo e MBeanOperationInfo possuem um método getDescription(), que você pode utilizar para retornar uma descrição de texto do item em questão.

getAttribute:

```
public Object getAttribute(ObjectName objName, String attrname)
  throws MBeanException,
         AttributeNotFoundException,
         InstanceNotFoundException,
         ReflectionException;
```

Essa API permite que a camada de agente recupere o valor de um atributo MBean. (Consulte a documentação do JMX na classe Attribute para obter detalhes adicionais.) Do ponto de vista da interface do MQe JMX, as propriedades mais importantes dessa classe são o nome e o valor. O método `getAttribute()` assume dois parâmetros: um `ObjectName` correspondente ao recurso em questão (uma fila instrumentada pelo JMX, por exemplo) e um parâmetro `String` correspondente ao nome do Atributo. O método retorna um Objeto que deve ser difundido para o tipo esperado do valor do Atributo.

Portanto, por exemplo, se um MBean da fila do MQe tiver um atributo nomeado Descrição do tipo java.lang.String, o valor para esse atributo será recuperado na camada de agente conforme a seguir (supondo que o ObjectName para a fila em questão tenha sido recuperado de uma consulta):

```
String queueDesc = (String)mbeanServer.getAttribute( queueObjName,
                                                      "Descrição");
```

Esse método emite exceções do tipo: `AttributeNotFoundException`, `MBeanException` ou `ReflectionException`. As Exceções do MQe são retornadas agrupadas em MBeanExceptions. Consulte “Identificando Erros” na página 151.

Nota: Por conveniência, os blocos try/catch necessários para capturar as exceções emitidas por esses métodos MBeanServer são omitidos nos exemplos destas seções. Consulte “Identificando Erros” na página 151.

Alguns adaptadores, como o Sun HtmlAdaptorServer, chamam os métodos getAttribute() e setAttribute() recursivamente ao receber ou configurar vários atributos em vez de chamar getAttributes() ou setAttributes(). Isso pode resultar em um elevado código extra. Nesse caso, é aconselhável aumentar o atributo `cacheInterval` no MBean Admin. O armazenamento em cache dos valores de atributos diminui a quantidade de trabalho feito pelo adaptador.

getAttributes:

```
public AttributeList getAttributes( ObjectName name,
                                    String[] attributes)
  throws InstanceNotFoundException,
          ReflectionException;
```

O parâmetro `attributes` consiste em uma matriz dos nomes de atributos a serem recuperados. O valor de retorno é do tipo `javax.management.AttributeList` que estende `java.util.ArrayList` e fornece métodos para incluir objetos de `Atributo` em uma `AttributeList`. Os atributos são recuperados de uma `AttributeList` utilizando uma instância de `Iterador` e os métodos de `Classe de Atributos` `getName()` e `getValue()`.

```
String[] attributeNames = {"Descrição","Expiração"};
AttributeList myAttrs =
  mbeanServer.getAttributes(queueObjName,attributeNames);
Iterator myIter = myAttrs.iterator();
while(myIter.hasNext()) {
  Attribute attribute = (Attribute)myIter.next();
  System.out.println("Nome do atributo: " + attribute.getName());
  System.out.println("Valor do atributo: " + attribute.getValue());
}
```

O método correspondente para getAttributes() no nível da instrumentação não pode emitir exceções do usuário. Isso limita a utilidade de getAttributes() na camada de agente; por exemplo, as exceções MQe não podem ser recuperadas. Em vez de utilizar getAttributes(), pode ser mais útil efetuar loop através da matriz String de nomes de atributos (attributeNames), chamando getAttribute() para cada um deles, embora isso aumente o código extra. O mesmo aplica-se a setAttributes().

setAttribute:

```
public void setAttribute( ObjectName name,
                          Attribute attribute)
  throws InstanceNotFoundException,
         AttributeNotFoundException,
         InvalidAttributeValueException,
         MBeanException,
         ReflectionException;
```

Esse método é utilizado para configurar o nome e o valor de um novo atributo ou para atualizar o valor atual de um atributo. O exemplo a seguir mostra como utilizar o MBean da fila do MQe instrumentado pelo JMX, conhecido pelo nome do objeto queueObjName para configurar o atributo Descrição no nível de agente:

```
Attribute descAttr =
  new Attribute("Descrição","Uma descrição para minha fila");
mbeanServer.setAttribute(queueObjName, descAttr);
```

Alguns adaptadores, como o Sun HtmlAdaptorServer, chamam os métodos getAttribute() e setAttribute() recursivamente ao receber ou configurar vários atributos em vez de chamar getAttributes() ou setAttributes(). Isso pode resultar em um elevado código extra. Nesse caso, é aconselhável aumentar o atributo cacheInterval no MBean Admin. O armazenamento em cache dos valores de atributos diminuirá a quantidade de trabalho feito pelo adaptador.

setAttributes:

```
public AttributeList setAttributes( ObjectName name,
                                    AttributeList attribute)
  throws InstanceNotFoundException,
         ReflectionException;
```

Esse método é utilizado para configurar ou atualizar vários Atributos de uma vez. O exemplo a seguir mostra como utilizar o MBean da fila do MQe instrumentado pelo JMX, conhecido pelo nome do objeto queueObjName, para configurar os atributos Descrição e Expiração no nível de agente:

```
/*criar os atributos para atualização */
Attribute descAttr =
  new Attribute("Descrição","Uma nova descrição para minha fila");
Attribute expiryAttr =
  new Attribute("Expiração", new Long(1000));

/*criar o parâmetro de entrada AttributeList   */
/* e incluir os Atributos na lista       */
AttributeList toUpdate = new AttributeList();
toUpdate.add(descAttr);
toUpdate.add(expiryAttr);
```

```
/* os resultados da chamada de setAttributes() e da verificação, se necessário, */
AttributeList updates = mbeanServer.setAttributes(queueObjName, toUpdate);
/* agora podem processar atualizações conforme mostrado em getAttributes() */
```

Nota: Aplicam-se as mesmas limitações à identificação de erros para setAttributes que aquelas descritas anteriormente para getAttributes().

invoke:

```
public Object invoke( ObjectName name,
                      String operationName,
                      Object[] params,
                      String[] signature)
  throws InstanceNotFoundException,
         MBeanException,
         ReflectionException;
```

Esse método é utilizado para chamar operações do MQe agrupadas pelo JMX em recursos instrumentados pelo MQe JMX.

Os parâmetros de entrada são:

name
o ObjectName correspondente ao recurso do MQe a ser administrado.

operationName
o nome da operação a ser chamada, por exemplo: `addAlias`.

params
uma matriz representando os parâmetros de entrada para a operação.

signature
uma matriz representando os tipos de dados correspondentes a cada parâmetro.

Nota: Os índices para as entradas em params e signature devem corresponder: a entrada no índice j em signature deve representar o tipo de dados da entrada no índice j em params.

Suponha que você queira chamar o método addAlias() em uma fila do MQe representada pelo nome do objeto queueObjName no qual exista um parâmetro de entrada do tipo String que represente o nome do alias. O exemplo a seguir mostra como fazer isso:

```
Object[] params = {new String("myAlias")};
String[] signature = {new String("java.lang.String")};

mbeanServer.invoke(queueObjName, "addAlias", params, signature);
```

Neste caso, não existe valor de retorno com o qual se preocupar. Entretanto, embora esse seja um exemplo relativamente simples, ele ilustra os princípios que se aplicam a todas as operações chamadas utilizando esse método.

Tipos de Dados

No exemplo para o método invoke() do MBeanServer na subseção anterior, a assinatura do parâmetro de entrada String[] representa os tipos de dados de todos os parâmetros de entrada para o método que está sendo chamado.

Para assegurar conformidade com o modelo OpenMBean, utilizamos apenas os tipos de dados aprovados para os métodos getter() e setter() do atributo e para os parâmetros da operação. Os tipos de dados especificados quando invoke() é chamado são, portanto, sempre limitados a um conjunto de tipos aprovados, conforme a seguir:

- Tipos de dados simples:
  - java.lang.Void

  - java.lang.Boolean
  - java.lang.Byte
  - java.lang.Character
  - java.lang.String
  - java.lang.Short
  - java.lang.Integer
  - java.lang.Long
  - java.lang.Float
  - java.lang.Double
- Matrizes dos tipos anteriores:
  - javax.management.ObjectName
  - javax.management.openmbean.CompositeData
  - javax.management.openmbean.TabularData

Os literais do nome da classe para cada tipo possuem um formato específico, conforme a seguir:

- Os tipos de dados simples listados acima são retornados conforme descritos, por exemplo, ″java.lang.Byte″
- Para matrizes desses tipos, a situação é mais complexa. Para o propósito dos tipos JMX do MQe, os únicos tipos de matrizes são java.lang.String e java.lang.Short. Esses tipos de matrizes são definidos conforme a seguir:

Tabela 42. Tipos de Dados e Cadeias de Literais do Nome da Classe

Tipo de Dados	Cadeia de Literal do Nome da Classe
String[]	″[Ljava.lang.String;″
Short[]	″[Ljava.lang.Short;″

Nota: Observe o ponto-e-vírgula como o final do nome da classe. Em todos os contextos em que um tipo de dados precisa ser especificado em todas as camadas de agente e instrumentação JMX, o formato de literal do nome da classe deve ser utilizado.

Divergência da Interface de Administração do MQe

Esta seção descreve os conceitos da interface do MQe JMX que diferem em sua implementação da interface administrativa do MQe.

Operações do Sistema de Mensagens

A interface do MQe JMX é projetada como um recurso administrativo para auxiliar na configuração e no gerenciamento de recursos do MQe por meio do JMX. Portanto, as operações do sistema de mensagens não pertencem a estas instruções e as operações fornecidas destinam-se apenas para propósitos de teste. As operações putMessage() e deleteMessage() são fornecidas para os tipos de filas permitidas. Essas operações de sistema de mensagens fornecem um escopo muito limitado para testar a conectividade e a configuração de uma rede. Como as operações do sistema de mensagens são mínimas, não é possível testar a operação de uma rede utilizando as filas de armazenamento ou redirecionamento.

O método `putMessage()` assume um único parâmetro java.lang.String que representa o texto de um corpo da mensagem. O usuário pode fornecer apenas essa única Cadeia – nenhuma customização adicional da mensagem de teste pode ocorrer. Essa mensagem é enviada para a fila representada pelo MBean sob o qual o método é chamado.

O MBean que representa a fila em questão também possui um atributo chamado Messages (para qualificar essa instrução, apenas MBeans que representam filas de um tipo no qual a procura é permitida possuem esse atributo). Esse atributo é do tipo [Ljava.lang.String; e pode ser recuperado utilizando `getAttribute()` ou `getAttributes()`. Cada item na matriz String representa o texto do corpo da mensagem enviado para a fila utilizando o método `putMessage()`. Se as mensagens forem enviadas para a

fila em questão por qualquer meio diferente do método putMessage() da interface do JMX, o corpo do texto não será legível pela interface do JMX e o valor retornado para Mensagens refletirá isso.

O índice de um corpo da mensagem na matriz Messages pode ser utilizado para excluir essa mensagem da fila utilizando o método deleteMessage(). O índice é transmitido como o único parâmetro para o método deleteMessage().

Observe que ambos os métodos, putMessage() e deleteMessage(), devem ser chamados apenas por meio do método MBeanServer invoke(). Isso é verdadeiro para todas as operações listadas nesta seção.

Filas de Armazenamento e Redirecionamento

No MQe, existe uma única classe de fila, MQeStoreAndForwardQueue, que inclui a funcionalidade de Filas de Armazenamento e Filas de Redirecionamento na interface do MQe JMX. Esse tipo de fila tem a capacidade para fazer o seguinte:

- Redirecionar mensagens para o gerenciador de filas de destino (que o MQe JMX chama de ForwardToQMgr) ou para outro gerenciador de filas entre os gerenciadores de filas de envio e de destino. Neste caso, a fila de armazenamento e redirecionamento envia mensagens para o próximo salto ou para o gerenciador de filas de destino.
- Manter as mensagens até que o gerenciador de filas de destino possa coletá-las da fila de armazenamento e redirecionamento. Isso pode ser feito utilizando uma fila de servidor home. Utilizando essa abordagem, as mensagens são recebidas da fila de armazenamento e redirecionamento. O gerenciador de filas de destino, neste caso, é incluído na DestinationQMgrList, tal como é chamada pelo MQe JMX.

As MQeStoreAndForwardQueues possuem uma propriedade que identifica seus conjuntos de gerenciadores de filas de destino (Queue_QMgrNameList).

No caso do MBean *Fila de Armazenamento* , não há qualquer ForwardToQMgr. O propósito exclusivo dessa fila é armazenar mensagens para as filas em sua DestinationQMgrList.

A instância de MBean *Fila de Redirecionamento* , por contraste, possui um ForwardToQMgr bem como uma DestinationQMgrList. Desse modo, possui ambos os recursos *redirecionar* e *armazenar* de MQeStoreAndForwardQueue enquanto que *Fila de Armazenamento* possui apenas o recurso *armazenar*.

Essa divisão de funcionalidade entre as representações MBean da fila tem como finalidade simplificar as funções das filas em questão. A Fila de Armazenamento é, na verdade, uma fila de "armazenamento" sem o recurso de "redirecionamento" da Fila de Redirecionamento.

Terminologia da Interface Programática versus Terminologia da Interface com o Usuário

Referências de Fila: Ao programar no MQe, você consulta os diferentes tipos de filas pelas referências utilizadas nesta documentação.

Entretanto, quando uma fila é exibida na interface com o usuário do JMX, ela é atribuída a uma referência diferente. A tabela a seguir mostra o relacionamento entre as referências da interface com o usuário e as referências programáticas.

Tabela 43. Mapeamento de Referências de Fila

Referência de Fila da Interface com o Usuário	Referência de Fila da Interface Programática	Classe MQeAdminMessage
Admin	Admin	MQeAdminQueueAdminMsg
Aplicativo	Local	MQeQueueAdminMsg

Tabela 43. Mapeamento de Referências de Fila (continuação)

Referência de Fila da Interface com o Usuário	Referência de Fila da Interface Programática	Classe MQeAdminMessage
Proxy Assíncrono	Remoto (em que o modo é assíncrono)	MQeRemoteQueueAdminMsg
Ponte	Ponte	MQeBridgeQueueAdminMsg
Redirecionar	Armazenamento e Redirecionamento (nome do gerenciador de filas proprietário diferente do gerenciador de filas locais)	MQeStoreAndForwardQueueAdminMsg
Servidor Home	Servidor Home	MQeHomeServerQueueAdminMsg
Proxy	Remoto (incorpora ambos os modos de fila)	MQeRemoteQueueAdminMsg
Armazenar	Armazenamento e Redirecionamento (mesmo nome do gerenciador de filas proprietário que o gerenciador de filas locais)	MQeStoreAndForwardQueueAdminMsg
Proxy Síncrono	Remoto (em que o modo é síncrono)	MQeRemoteQueueAdminMsg

Referências de Fila do Gerenciador de Filas: Ao programar no MQe, você utiliza o termo *queueqm* — a maioria das filas possui um gerenciador de filas associado e esse é o *queueqm*. Você consulta esses diferentes tipos de gerenciadores de filas pelas referências utilizadas nesta documentação.

No entanto, quando um *queueqm* é exibido na interface com o usuário do JMX, uma referência diferente é atribuída a ele.

A tabela a seguir mostra como o *queueqm* é referido para cada tipo de fila, utilizando as referências da interface com o usuário para ambos.

Tabela 44. Mapeamento de Referências do Queueqm

Referência de Fila da Interface com o Usuário	Referência de Queueqm da Interface com o Usuário	Descrição
Admin	nenhum	Igual ao nome do gerenciador de filas locais
Aplicativo	nenhum	Igual ao nome do gerenciador de filas locais
Proxy Assíncrono	DestinationQMgr	O nome do gerenciador de filas que mantém a fila do aplicativo correspondente
Ponte	DestinationQMgr	O nome do gerenciador de filas do MQ que mantém a fila do MQ correspondente
Redirecionar	ForwardToQMgr	O nome do gerenciador de filas para o qual as mensagens que chegam a essa fila serão redirecionadas
Servidor Home	GetFromQMgr	O nome do gerenciador de filas do qual as mensagens nas filas de armazenamento ou redirecionamento serão recebidas
Armazenar	nenhum	Igual ao nome do gerenciador de filas locais
Proxy Síncrono	DestinationQMgr	O nome do gerenciador de filas que mantém a fila do aplicativo correspondente

Identificando Erros

A classe MBeanException é definida na *Especificação Sun JMX (1.2)*, conforme a seguir:

Essa classe representa exceções ″definidas pelo usuário″ emitidas por métodos MBean no agente. Ela ″agrupa″ a exceção real ″definida pelo usuário″ emitida. Essa exceção será construída pelo MBeanServer quando uma chamada a um método MBean resultar em uma exceção desconhecida.

Existem métodos na classe MBeanException que retornarão a classe de exceção original e qualquer mensagem que estava dentro da exceção:

```
public Exception getTargetException();
public Throwable getCause();
```

Portanto, um aplicativo pode recuperar e manipular quaisquer exceções do MQe (ou outro).

Entretanto, pode ser o caso em que as exceções capturadas na camada de agente não são exibidas adequadamente por meio do adaptador ou conector utilizado. Por exemplo, o Sun RI HtmlAdaptorServer não recupera e exibe exceções agrupadas em MBeanExceptions.

Isso impacta o benefício de utilizar o HtmlAdaptorServer para receber de volta as MQeExceptions ao configurar os atributos. Por exemplo, o MQe emitirá uma exceção se você tentar configurar uma prioridade de fila fora do intervalo de 0 a 9. O HtmlAdaptorServer mostra apenas que o valor de atributo Priority não foi configurado. Essa é uma limitação sem propósito para esse adaptador específico.

Se um valor nulo for digitado para um parâmetro obrigatório em uma operação, uma NullPointerException estará dentro da MBeanException agrupada.

Qualquer exceção que retornar do MQe após uma operação ou uma tentativa de configurar um atributo será do tipo MQeException dentro da MBeanException.

Notificações

A especificação JMX fornece um mecanismo de notificação que foi implementado na interface JMX do MQe.

Essa interface implementa a classe JMX NotificationBroadcaster e, portanto, pode enviar notificações para quaisquer aplicativos que implementam a classe JMX NotificationListener correspondente.

Neste contexto, uma notificação é uma mensagem enviada por um difusor de notificação para um listener de notificação por meio da infra-estrutura do JMX.

As notificações de duas classes que subdividem a classe de Notificação do JMX são enviadas a partir da interface JMX do MQe. Essas classes são:

- com.ibm.mqe.jmx.MQeAliasNotification;
- javax.management.AttributeChangeNotification.

MQeAliasNotification

Esta classe estende javax.management.Notification e fornece os seguintes tipos de notificação:

- mqe.connection.alias.added: quando um alias é incluído em uma conexão.
- mqe.connection.alias.removed: quando um alias é removido de uma conexão.
- mqe.queue.alias.added: quando um alias é incluído em uma fila.
- mqe.queue.alias.removed: quando um alias é removido de uma fila.
- mqe.queuemanager.alias.added: quando um alias é incluído em um gerenciador de filas.
- mqe.queuemanager.alias.removed: quando um alias é removido de um gerenciador de filas.

AttributeChangeNotification

Essa classe é utilizada para notificar listeners do JMX interessados quando o valor de um atributo MBean é alterado. Ela fornece o seguinte tipo de notificação:

- jmx.attribute.changed: quando o valor de um atributo é alterado.

Se ocorrer uma exceção quando for feita uma tentativa de alterar um atributo, o texto da exceção será transmitido de volta para o usuário por meio da 'mensagem' de notificação. O método getMessage() pode ser utilizado, então, para recuperar o texto da exceção. A classe AttributeChangeNotification também fornece os métodos getOldValue() e getNewValue() para retornar o valor de atributo original e o valor para o qual ele está sendo alterado. No caso de um erro, getNewValue() não retornará o valor de atributo real (pois a tentativa de alterar o atributo não obteve êxito) – neste caso, getOldValue() retorna o valor de atributo real no ponto de notificação.

Utilizando Notificações

Para receber notificações, um usuário (na camada de Agente) implementa a interface JMX NotificationListener e, em seguida, chama o addNotificationListener na instância MBeanServer:

```
public void addNotificationListener( ObjectName objName,
                                     NotificationListener listener,
                                     NotificationFilter filter,
                                     Object handback);
```

em que

- `objectName` representa o MBean a partir do qual as notificações serão recebidas
- `listener` é a instância do usuário de NotificationListener
- `filter` é um filtro opcional utilizado se apenas um subconjunto de notificações possíveis for requerido (pode ser nulo)
- `handback` é um objeto que pode ser utilizado para conter dados privados que a rotina de tratamento da notificação recebida deseja acessar (pode ser nulo)

  Nota: Existe um método `addNotificationListener()` alternativo no MBeanServer que transmite o ObjectName para o listener em vez da instância NotificationListener real. Ele poderá ser utilizado se a própria instância NotificationListener for um MBean registrado.

Se você tiver quaisquer recursos com aliases e incluir listeners para notificações de um recurso e de seus aliases, você receberá várias notificações idênticas. É recomendável assegurar que os nomes dos objetos transmitidos como parâmetros para o método addNotificationListener() não contenham o par de chave-valor da propriedade "tipo=alias".

Depois de chamar essa API, o listener do usuário será incluído na tabela de listeners do difusor.

Para manipular notificações recebidas, o usuário também precisa implementar o seguinte método:

```
public void handleNotification ( Notification notification,
                                 Object handback);
```

em que:

- `notification` é a instância de notificação enviada pelo objeto NotificationBroadcaster
- `handback` é um objeto que pode ser utilizado para conter dados privados que a rotina de tratamento da notificação recebida deseja acessar (pode ser nulo)

Este método é onde as notificações recebidas são processadas. A classe de notificação fornece vários métodos úteis que podem ser utilizados para extrair informações sobre a notificação:

```
public String getType();     // retorna o tipo de notificação
public Object getSource();   // retorna a origem da notificação
public long   getSequence(); // retorna o número de seqüência da
                             //  notificação [1]
```

```
public String getMessage();  // retorna uma mensagem de texto associada
                             //  à notificação
public Object getUserData(); // retorna o objeto handback
```

Nota:

1. O número de seqüência fornece informações sobre a ocorrência da notificação mas não é configurado nessa implementação JMX do MQe, portanto sempre terá um valor igual a 0.

O exemplo a seguir mostra como o agente poderia configurar listeners para determinados MBeans do MQe. Neste exemplo, o usuário está interessado apenas em receber notificações de MBeans que representam filas do aplicativo pertencentes ao gerenciador de filas locais TestQueueManager:

```
 /* localizar todos os mbeans e configurar listeners para eles */
ObjectName scope = new
ObjectName("com.ibm.MQe_TestQueueManager_ApplicationQueues:*");
Set results = mbeanServer.queryNames(scope,null);
Iterator iter = results.iterator();
while(iter.hasNext()) {
  /* para cada bean, verificar se não é um MBean do alias –
    * esses beans possuem tipo=alias no ObjectName        */
   ObjectName objName = (ObjectName)iter.next();
   String type = objName.getKeyProperty("tipo");

   if(type == null || !type.equals("alias")) {
      /*  incluir um listener */
      mbeanServer.addNotificationListener(objName,this,null,null);
      }
 }
```

Outros Problemas

Configurando Atributos de Tipo de Matriz

É possível que os atributos de tipo de matriz (por exemplo, `[Ljava.lang.String;`) sejam gravados e lidos. Portanto, é possível, por exemplo, atualizar a matriz de aliases de uma fila utilizando a interface do MQe JMX.

Entretanto, existem limitações quanto ao modo em que alguns adaptadores permitem que o usuário faça essas atualizações. Por exemplo, o adaptador Sun RI HtmlAdaptorServer apenas fornecerá uma matriz para atualização que tenha as mesmas dimensões que a matriz atual. Desse modo, se uma fila não tiver aliases existentes, a matriz para atualização terá tamanho igual a zero e, por conseguinte, nenhum novo alias poderá ser incluído na fila utilizando o atributo de alias da fila. No entanto, neste caso, um alias poderá ser incluído utilizando a operação `addAlias()`.

Se uma fila tiver dois aliases existentes, a matriz fornecida para a atualização do atributo de alias terá tamanho igual a dois. Um ou os dois aliases poderão ser alterados utilizando a matriz gravável. Entretanto, alguns adaptadores não permitem que o usuário limpe o conteúdo das células da matriz e transmita de volta uma matriz contendo cadeia(s) vazia(s). Isso causará uma exceção. Portanto, esses adaptadores permitirão apenas uma atualização que mantenha constante o número de aliases existentes.

Como estas são limitações apenas de adaptadores específicos, decidimos permitir que os atributos de matriz sejam atualizados quando apropriado. A alternativa seria forçar os usuários a utilizarem as operações para incluir aliases em vez de utilizarem a possível atualização do atributo.

Esse exemplo específico também pode ser estendido a situações em que a capacidade do adaptador ou conector não corresponde à capacidade de uma interface programática. Não é possível prever essas limitações antecipadamente e, portanto, podem existir recursos de nossa implementação que não sejam, de maneira ideal, apropriados para alguns adaptadores e conectores. Decidimos por constringir a funcionalidade de nossa camada de instrumentação para corresponder às capacidades de adaptadores específicos.

MBeans de Alias

Determinados recursos do MQe – filas, gerenciadores de filas e conexões – podem ter aliases, outros nomes pelos quais eles são conhecidos. Para facilitar a administração, a interface do MQe JMX registra novamente os MBeans que possuem aliases sob um nome de objeto correspondente ao alias. Portanto, por exemplo, se a interface do JMX for utilizada para incluir um alias `myAlias` em uma fila do aplicativo `myQueue`, o MBean da fila será, na verdade, registrado duas vezes,

uma vez com o nome do objeto

`com.ibm.MQe_<Nome_do_Gerenciador_de_Filas_Proprietário>_ApplicationQueues:name=myQueue`

e uma vez com o nome do objeto

`com.ibm.MQe_<Nome_do_Gerenciador_de_Filas_Proprietário>_ApplicationQueues:name=myAlias,`
`type=alias,`
`resourceName=myQueue@<Nome_do_Gerenciador_de_Filas_Proprietário>.`

Isso significa que o administrador não precisa estar ciente do nome real do recurso para poder administrá-lo por meio do JMX.

Do mesmo modo, quando os aliases são removidos dos recursos, o registro do ObjectName correspondente é removido.

Um efeito secundário dessa prática é que se um usuário optar por criar e registrar alguns MBeans do MQe sem utilizar o método auxiliar `createMQeMBeans()`, isso poderá resultar em um quadro inconsistente em que alguns recursos são registrados com nomes de alias enquanto outros não. Isso aplica o argumento para utilizar o método auxiliar para criar e registrar todos os MBeans do MQe.

Tradução

Os atributos de descrição para recursos do MQe estão disponíveis em todos os diferentes idiomas suportados pelo MQe, em arquivos de propriedades traduzidos. O idioma utilizado para as descrições será selecionado de acordo com o código do idioma padrão de sua máquina, utilizando a convenção normal do Java, conforme descrito resumidamente a seguir.

Você deve fornecer os arquivos de propriedades adequados em seu caminho de classe para todos os idiomas, **inclusive inglês (EUA)**.

Os arquivos de propriedades podem ser localizados na pasta `Java/com/ibm/mqe/properties` abaixo da pasta na qual o MQe foi instalado. Esse diretório contém dois arquivos de propriedades requeridos em todos os idiomas suportados:

- `AdminDescBundle`
- `JMXDescBundle`

É possível incluir todos os arquivos de propriedades no caminho de classe ou apenas os arquivos requeridos para seu idioma.

Existem dois arquivos padrão:

- `AdminDescBundle.properties`
- `JMXDescBundle.properties`

que contêm as descrições em inglês. Esses arquivos serão utilizados se não houver um arquivo traduzido correspondente ao seu código do idioma.

Todos os outros arquivos possuem o código do país XX anexado à primeira parte do nome do arquivo para criar o `JMXDescBundle_XX.properties`, em que *XX* é um dos seguintes códigos do país:

Código	Idioma
de	Alemão
es	Espanhol
fr	Francês
it	Italiano
ja	Japonês
ko	Coreano
pt_BR	Português (Brasil)
zh	Chinês
zh_TW	Chinês tradicional

O sistema Java para selecionar o idioma é o seguinte:

1. Determine o código do idioma padrão do computador, por exemplo, `fr_FR`
2. Procure esse código do idioma nos arquivos fornecidos, verificando o código do idioma em cada um deles:
   - Se um arquivo de código do idioma completo, por exemplo, `file_fr_FR` **não** for localizado, o sistema utilizará o arquivo de código parcial, se existir — neste exemplo, `file_fr`
   - Quando um arquivo com um código completo **for** localizado, apenas o código do idioma completamente correspondente o selecionará, por exemplo, o código do idioma `pt_BR` utilizará o arquivo `file_pt_BR`, mas o código do idioma `pt_PT` não, e portanto o padrão será inglês.

Nota: O JTC recomenda que, quando existir um arquivo de código do idioma completo, o arquivo de código semiqualificado também deverá existir, mesmo se estiver vazio.

Para obter informações completas sobre isso, consulte os Web sites:

```
http://oss.software.ibm.com/icu/userguide/design.html
http://java.sun.com/products/jdk/1.2/docs/api/java/util/ResourceBundle.html
```

Informações Relacionadas no JMX

Esta seção lista várias fontes de informações que podem ser úteis.

Livros

- Perry, J. Steven, Java Management Extensions: Managing Java Applications with JMX (O'Reilly & Associates, Inc: 2002)
- Jasnowski, Mike. JMX Programming (Publicação Wiley, Inc: 2002)

Artigos

- http://java.sun.com/products/JavaManagement/ (Artigos, downloads e recursos da Sun para JMX)
- http://mx4j.sourceforge.net/ (JMX de Código Aberto http://mx4j.sourceforge.net/)
- http://www-106.ibm.com/developerworks/library/j-jmx1/
- http://www-106.ibm.com/developerworks/library/j-jmx2/
- http://www-106.ibm.com/developerworks/library/j-jmx3/ (Artigo: From black boxes to enterprises Parts 1-3)
- http://www.informit.com/content/index.asp?session_id={61F6D302-4F4F-4D68-96D8-AD545B00CA28}&product_id={583CB44A-AC47-4959-9C01-FA8DF0884EEE} (Artigo: Managing Complex systems with JMX)
- http://jcp.org/aboutJava/communityprocess/maintenance/jsr003/jmx1.2-change-log.txt (Alterações entre JMX 1.1 e 1.2)

Outros Recursos

- http://www.alphaworks.ibm.com/tech/TMX4J (IBM Tivoli - Implementação Tivoli da Especificação JMX)
- http://www.jguru.com/forums/JMX (Fórum JGuru JMX).

Índice Remissivo