MQSeries®

Utilización de Java™

SC10-3345-01

MQSeries®

Utilización de Java™

SC10-3345-01

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información general del "Apéndice F. Avisos" en la página 383.

Segunda edición (Enero de 2001)

Este manual es la traducción del original inglés *MQSeries Using Java*, SC34-5456-06.

Esta edición se aplica a IBM® MQSeries Classes for Java Versión 5.2.0 y Servicio de mensajes de MQSeries Classes for Java Versión 5.2 y a todos los releases y modificaciones posteriores mientras no se indique lo contrario en nuevas ediciones.

Contenido

Figuras

Tablas

Acerca de este manual

En este manual se describe:

- MQSeries Classes for Java, que se puede utilizar para acceder a sistemas MQSeries
- Servicio de mensajes de MQSeries Classes for Java, que se puede utilizar para acceder tanto al servicio de mensajes de Java™ (JMS) como a aplicaciones MQSeries

Nota: Esta documentación sólo está disponible en copia software (PDF y HTML) como parte del producto y en el sitio web de la familia MQSeries, en la dirección siguiente:

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

No se puede solicitar como publicación impresa.

Abreviaturas utilizadas en este manual

En este manual se utilizan las abreviaturas siguientes:

MQ Java MQSeries Classes for Java y Servicio de mensajes de MQSeries Classes for Java combinados

Java base de MQ
MQSeries Classes for Java

MQ JMS Servicio de mensajes de MQSeries Classes for Java

A quién va dirigido este manual

Esta información está escrita para programadores que estén familiarizados con la interfaz de programación de aplicaciones orientadas a procedimientos de MQSeries tal como se describe en la publicación *MQSeries® Application Programming Guide*, y muestra cómo usar estos conocimientos para utilizar las interfaces de programación MQ Java de forma productiva.

Conocimientos necesarios para comprender este manual

Se requiere:

- Conocer el lenguaje de programación Java™
- Comprender la finalidad de MQI (Interfaz de Colas de Mensajes), tal como se describe en el capítulo sobre la interfaz de colas de mensajes en la publicación *MQSeries Application Programming Guide* y en el capítulo sobre descripciones de invocaciones en la publicación *MQSeries Application Programming Reference*.
- Tener experiencia en programas MQSeries en general o estar familiarizado con el contenido de las demás publicaciones de MQSeries

Los usuarios que tengan previsto utilizar Java base de MQ con CICS® Transaction Server para OS/390®, también deben estar familiarizados con:

- Los conceptos de CICS (sistema de control de la información del cliente)
- La utilización de la API (interfaz de programas de aplicación) de CICS® de Java™

- La ejecución de programas Java™ desde CICS®

Los usuarios que tengan previsto utilizar VisualAge® para Java™ para desarrollar aplicaciones OS/390® UNIX® System Services High Performance Java™ (HPJ) deben estar familiarizados con Enterprise Toolkit para OS/390® (suministrado con VisualAge® para Java™ Enterprise Edition para OS/390®, Versión 2).

Cómo utilizar este manual

En la Parte 1 de este manual se describe cómo utilizar Java base de MQ y MQ JMS, en la Parte 2 se proporciona la ayuda necesaria para los programadores que desean utilizar Java base de MQ, y en la Parte 3 se facilita la información necesaria para los programadores que desean utilizar MQ JMS.

Primero, lea los capítulos de la Parte 1, en la que se presentan Java base de MQ y MQ JMS y, a continuación, utilice la guía de programación de la Parte 2 ó 3 para comprender cómo se deben utilizar las clases para enviar y recibir mensajes MQSeries en el entorno que desea utilizar.

Al final de la publicación puede encontrar un glosario y la bibliografía.

Resumen de los cambios

En este apartado se describen los cambios que se han realizado en esta edición de *MQSeries Utilización de Java*. Los cambios realizados desde la edición anterior de la publicación aparecen marcados mediante líneas verticales, situadas en el margen izquierdo.

Cambios realizados en esta edición (SC10-3345-01)

Esta edición incluye actualizaciones para la nueva función de MQ Java™ V5.2. Incluye lo siguiente:

- Actualizaciones en los procedimientos de instalación. Consulte el "Capítulo 2. Procedimientos de instalación" en la página 9.
- Soporte para agrupación de conexiones, que puede mejorar el rendimiento para las aplicaciones y el middleware que utilizan varias conexiones a gestores de colas MQSeries. Consulte los apartados:
  - "Agrupación de conexiones" en la página 71
  - "MQEnvironment" en la página 97
  - "MQPoolServices" en la página 133
  - "MQPoolServicesEvent" en la página 134
  - "MQPoolToken" en la página 136
  - "MQQueueManager" en la página 151
  - "MQSimpleConnectionManager" en la página 161
  - "MQConnectionManager" en la página 165
  - "MQPoolServicesEventListener" en la página 164
  - "ManagedConnection" en la página 172
  - "ManagedConnectionFactory" en la página 175
  - "ManagedConnectionMetaData" en la página 177
- Nuevas opciones de configuración de colas de suscriptores para ofrecer una cola múltiple y una propuesta de colas compartidas para aplicaciones de publicación/suscripción. Consulte los apartados siguientes:
  - "Propiedades" en la página 44
  - "Configuración de la cola de suscriptores base" en la página 199
  - "Topic" en la página 336
  - "TopicConnectionFactory" en la página 340
- Un nuevo programa de utilidad de limpieza de suscriptor, para evitar los problemas que se producen al cerrar de modo incorrecto los objetos de suscriptor. Consulte el apartado "Programa de utilidad de limpieza de suscriptores" en la página 202.
- Soporte para Recursos de servidor de aplicaciones, es decir, el proceso simultáneo de mensajes. Consulte lo siguiente:
  - "Capítulo 13. Recursos de servidor de aplicaciones MQ JMS" en la página 225
  - "ConnectionConsumer" en la página 261
  - "QueueConnection" en la página 307
  - "Session" en la página 320
  - "TopicConnection" en la página 338

- Actualizaciones en la información de configuración del servidor LDAP. Consulte el "Apéndice C. Configuración del servidor LDAP para objetos Java™" en la página 373.
- Soporte para transacciones distribuidas utilizando el protocolo XA de X/Open. Es decir, MQ JMS incluye clases XA que permiten que MQ JMS pueda participar en una confirmación en dos fases coordinada por un gestor de transacciones adecuado. Consulte los apartados siguientes:
  - "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377
  - "XAConnection" en la página 354
  - "XAConnectionFactory" en la página 355
  - "XAQueueConnection" en la página 356
  - "XAQueueConnectionFactory" en la página 357
  - "XAQueueSession" en la página 359
  - "XASession" en la página 360
  - "XATopicConnection" en la página 362
  - "XATopicConnectionFactory" en la página 364
  - "XATopicSession" en la página 366

Cambios con respecto a la sexta edición (de la publicación en inglés, SC34-5456-05)

Se ha incluido el soporte para Linux.

Cambios con respecto a la quinta edición (de la publicación en inglés, SC34-5456-04)

Soporte para WebSphere™ y MQSeries® Integrator V2

Java™ base de MQ versión 5.1.2 ahora está disponible como una extensión del producto. Ofrece la posibilidad de:

- Conectar a MQSeries® Integrator para Windows NT®, Versión 2.0, para dar soporte para publicación/suscripción (Publish/Subscribe). Para obtener información más detallada, consulte el "Apéndice D. Conexión a MQSeries® Integrator V2" en la página 375.
- Utilizar el suministrador de servicio JNDI CosNaming de WebSphere™. Para obtener información más detallada, consulte el apartado "Configuración" en la página 36.

Parte 1. Instrucciones para los usuarios

Capítulo 1. Cómo empezar

En este capítulo se proporciona una visión general de MQSeries Classes for Java, el Servicio de mensajes de MQSeries Classes for Java sus utilizaciones.

¿Qué es MQSeries Classes for Java?

MQSeries Classes for Java (Java base de MQ) permite que un programa creado en el lenguaje de programación de Java™ pueda:

- Conectarse a MQSeries como un cliente MQSeries
- Conectarse directamente a un servidor MQSeries

Permite que los servlets, las aplicaciones y las applets de Java™ emitan invocaciones y consultas para MQSeries. De este modo, se proporciona el acceso a las aplicaciones heredadas y de sistema principal, por lo general, a través de Internet, sin que sea necesario tener otro código MQSeries en la máquina cliente. Con Java base de MQ, el usuario de un terminal Internet puede convertirse en un participante real en las transacciones, en lugar de ser sólo un emisor y receptor de información.

¿Qué es el Servicio de mensajes de MQSeries Classes for Java?

El Servicio de mensajes de MQSeries Classes for Java (MQ JMS) es un conjunto de clases Java™ que implementan las interfaces JMS Java™ de Sun para permitir que los programas JMS accedan a sistemas MQSeries. Se ofrece soporte para ambos modelos de JMS, de punto a punto y de publicación y suscripción.

La utilización de MQ JMS como API para crear aplicaciones MQSeries ofrece numerosas ventajas. Algunas derivan de que JMS es un estándar abierto con varias implementaciones. Otras ventajas son resultado de las características adicionales que se encuentran en MQ JMS, pero no en Java base de MQ.

Las ventajas de utilizar un estándar abierto incluyen:
- Protección de la inversión, tanto en conocimientos específicos como en código de aplicación
- Disponibilidad del personal formado para la programación de aplicaciones JMS
- Posibilidad de conectar en implementaciones JMS diferentes para ajustarse a requisitos distintos

Puede obtener más información acerca de las ventajas de la API de JMS en el sitio web de Sun, en la dirección siguiente: `http://java.sun.com`.

Las funciones adicionales que se facilitan a través de Java base de MQ incluyen:
- Entrega de mensajes asíncrona
- Selectores de mensaje
- Soporte para envío de mensajes de publicación/suscripción
- Clases de mensajes estructuradas

¿Quién debe utilizar MQ Java?

Si su empresa se ajusta a alguno de los ejemplos siguientes, puede obtener numerosas ventajas utilizando MQSeries Classes for Java y el Servicio de mensajes de MQSeries Classes for Java:

Quién debe utilizar MQ Java

- Una empresa mediana o grande que se está iniciando en soluciones de cliente/servidor basadas en intranets. Aquí, la tecnología de Internet proporciona fácil acceso a las comunicaciones globales a un bajo coste, mientras que la conectividad de MQSeries proporciona una alta integridad con entrega asegurada e independencia horaria.
- Una empresa mediana o grande con necesidad de comunicaciones fiables de negocio a negocio con empresas asociadas. Aquí de nuevo, Internet proporciona fácil acceso a las comunicaciones globales a un bajo coste, mientras que la conectividad de MQSeries proporciona alta integridad con entrega asegurada e independencia horaria.
- Una empresa mediana o grande que desea proporcionar acceso desde la Internet pública a algunas de sus aplicaciones de empresa. Aquí, Internet proporciona el alcance global a un bajo coste, mientras que la conectividad de MQSeries proporciona alta integridad mediante el paradigma de gestión de colas. Además del bajo coste, la empresa puede obtener una mayor satisfacción del cliente mediante disponibilidad de 24 horas al día, respuesta rápida y una mayor exactitud.
- Un suministrador de servicios de Internet u otro suministrador de red de valor añadido. Estas compañías pueden aprovechar el bajo coste y la facilidad de las comunicaciones que proporciona Internet, además de añadir el valor de la alta integridad que proporciona la conectividad de MQSeries. Un suministrador de servicio de Internet que aprovecha MQSeries puede acusar inmediatamente el recibo de datos de entrada de un navegador de la web, garantizar la entrega y proporcionar un modo fácil para que el usuario del navegador de la web supervise el estado del mensaje.

MQSeries y el Servicio de mensajes de MQSeries Classes for Java proporcionan una infraestructura excelente para acceder a las aplicaciones de empresa y para desarrollar aplicaciones complejas de la web. Una petición de servicio de un navegador de la web puede ponerse en cola y procesarse cuando sea posible, permitiendo de este modo enviar una respuesta a tiempo al usuario final, independientemente de la carga del sistema. Mediante la colocación de esta cola 'cerca' del usuario final en términos de red, la puntualidad de la respuesta no queda afectada por la carga de la red. Además, la naturaleza transaccional del envío de mensajes de MQSeries significa que una petición simple del navegador puede expandirse de forma segura en una secuencia de procesos de fondo individuales de un modo transaccional.

Quién debe utilizar MQ Java

MQSeries Classes for Java también permite que los desarrolladores de aplicaciones aprovechen la potencia del lenguaje de programación Java™ para crear applets y aplicaciones que se puedan ejecutar en cualquier plataforma que ofrezca soporte para el entorno de ejecución de Java™. Estos factores se combinan para reducir considerablemente el tiempo de desarrollo de las aplicaciones MQSeries para varias plataformas. Además, si se realizan mejoras en las applets en el futuro, los usuarios finales pueden recogerlas de modo automático al bajar el código de applet.

Opciones de conexión

Las opciones programables permiten conectar MQ Java con MQSeries utilizando cualquiera de los procedimientos siguientes:

- Como un cliente MQSeries, utilizando TCP/IP (Transmission Control Protocol/Internet Protocol)
- En modalidad de enlaces, conectándose directamente a MQSeries

También se puede conectar Java base de MQ en Windows NT® utilizando VisiBroker para Java™. La Tabla 1 muestra las modalidades de conexión que se pueden utilizar para cada plataforma.

Tabla 1. Plataformas y modalidades de conexión

Plataforma servidor	**Modalidad de conexión**		
	Cliente		**Enlaces**
	Estándar	**VisiBroker**	
Windows NT®	sí	sí	sí
Windows® 2000	sí	no	sí
AIX®	sí	no	sí
Sun OS (v4.1.4 y anterior)	sí	no	no
Sun Solaris (v2.6, v2.8, V7 o SunOS v5.6, v5.7)	sí	no	sí
OS/2®	sí	no	sí
OS/400®	sí	no	sí
HP-UX	sí	no	sí
AT&T GIS UNIX®	sí	no	no
SINIX y DC/OSx	sí	no	no
OS/390®	no	no	sí
Linux	sí	no	no

Notas:

1. El soporte de enlaces Java™ HP-UX sólo está disponible para los sistemas que ejecuten la versión POSIX draft10 pthreaded de MQSeries. También necesita HP-UX Developer's Kit para Java™ 1.1.7 (JDK™), Release C.01.17.01 o superior.
2. En HP-UXv10.20, Linux, Windows® 95 y Windows® 98, sólo se ofrece soporte para la conectividad de cliente TCP/IP.

En los apartados siguientes se describen estas opciones con más detalle.

Conexiones de cliente

Para utilizar MQ Java como un cliente MQSeries, puede instalarlo en la máquina servidor de MQSeries, que también puede contener un servidor de la web, o en una otra máquina. Si instala MQ Java en la misma máquina que el servidor de la web, una de las ventajas es que se pueden bajar y ejecutar aplicaciones cliente de MQSeries en máquinas que no tienen instalado MQ Java localmente.

Dondequiera que elija instalar el cliente, puede ejecutarlo en tres modalidades diferentes:

Desde cualquier navegador de web habilitado para Java™
En esta modalidad, puede que las ubicaciones de los gestores de colas de MQSeries a las que se puede acceder estén limitadas por las restricciones de seguridad del navegador que se está utilizando.

Utilizando un visor de applets
Para utilizar este método, deberá tener instalado JDK (Java™ Developer's Kit) o JRE (Java™ Runtime Environment) en la máquina cliente.

Como un programa Java™ autónomo o en un servidor de aplicaciones web
Para utilizar este método, deberá tener instalado JDK (Java™ Developer's Kit) o JRE (Java™ Runtime Environment) en la máquina cliente.

Utilización de VisiBroker para Java™

En la plataforma Windows®, la conexión mediante VisiBroker se proporciona como alternativa a la utilización de los protocolos estándar de cliente MQSeries. Este soporte lo proporciona VisiBroker para Java™ junto con Netscape Navigator, y necesita VisiBroker para Java™ y un servidor de objetos MQSeries en la máquina servidor MQSeries. Con Java base de MQ se proporciona un servidor de objetos adecuado.

Conexión de enlaces

Cuando se utiliza la modalidad de enlaces, MQ Java utiliza JNI (Java™ Native Interface) para invocar directamente en la API del gestor de colas existente, en lugar de comunicarse a través de una red. De este modo, se proporciona mejor rendimiento para las aplicaciones MQSeries que si se utilizan conexiones de red. A diferencia de la modalidad de cliente, las aplicaciones que se crean utilizando la modalidad de enlaces no se pueden bajar como applets.

Para utilizar la conexión de enlaces, se debe haber instalado MQ Java en el servidor MQSeries.

Requisitos previos

Para ejecutar Java base de MQ, necesita el software siguiente:

- MQSeries para la plataforma servidor que desee utilizar.
- JDK (Java™ Developers Kit) para la plataforma servidor.
- Java™ Developers Kit o JRE (Java™ Runtime Environment), o el navegador de la web habilitado para Java para plataformas cliente. (Consulte el apartado "Conexiones de cliente").

  Nota: Para ejecutar applets Java base de MQ (por ejemplo, el programa de verificación de la instalación) en un navegador de la web, necesita un navegador que pueda ejecutar applets Java™ 1.1.6. HotJava™ de Sun

System, Netscape Navigator 4 e Internet Explorer 4 de Microsoft® son ejemplos de navegadores que satisfacen este requisito.

- VisiBroker para Java™ (sólo si se ejecuta en Windows® con una conexión de VisiBroker).
- Para OS/390®, OS/390® Versión 2 Release 5 con UNIX® System Services.
- Para OS/400®, AS/400® Developer Kit para Java™, 5769-JV1 y el intérprete Qshell, OS/400® (5769-SS1) Opción 30.

Para utilizar la herramienta de administración MQ JMS (consulte el "Capítulo 5. Utilización de la herramienta de administración de MQ JMS" en la página 35), necesita el software adicional siguiente:

- Como mínimo, uno de los paquetes de suministrador de servicio siguientes:
  - LDAP (Lightweight Directory Access Protocol) - `ldap.jar`, `providerutil.jar`.
  - Sistema de archivos - `fscontext.jar`, `providerutil.jar`.
- Un suministrador de servicio JNDI (Java™ Naming and Directory Service). Es el recurso que almacena representaciones físicas de los objetos administrados. Es probable que los usuarios de MQ JMS utilicen un servidor LDAP para esta finalidad, pero la herramienta también ofrece soporte para la utilización del suministrador de servicio de contexto del sistema de archivos. Si se utiliza un servidor LDAP, se debe haber configurado para almacenar objetos JMS. Para obtener información que le puede ayudar para la configuración, consulte el "Apéndice C. Configuración del servidor LDAP para objetos Java™" en la página 373.

Para utilizar los recursos XA de X/Open de MQ JMS, necesita MQSeries V5.2.

Capítulo 2. Procedimientos de instalación

En este capítulo se describe cómo instalar MQSeries Classes for Java y el producto Servicio de mensajes de MQSeries Classes for Java.

Instalación de MQSeries Classes for Java y del Servicio de mensajes de MQSeries Classes for Java

Este producto está disponible para las plataformas AIX®, AS/400®, HP-UX, Linux, Sun Solaris y Windows®. Contiene:

- MQSeries Classes for Java (Java base de MQ) Versión 5.2.0
- Servicio de mensajes de MQSeries Classes for Java (MQ JMS) Versión 5.2 (no AS/400®)

Para la conectividad disponible en cada plataforma específica, consulte el apartado "Opciones de conexión" en la página 5.

El producto se proporciona en archivos de formato comprimido, disponibles en el sitio web de MQSeries, `http://www.ibm.com/software/mqseries/`.

Nota: Para OS/390®, Java base de MQ se proporciona como un SupportPac™ de MQSeries que se puede bajar de `http://www.ibm.com/software/mqseries/`.

Para las versiones más recientes de las clases de Java base de MQ, puede instalar sólo Java base de MQ Versión 5.2.0. Para utilizar aplicaciones de MQ JMS, debe instalar Java base de MQ y MQ JMS (que, juntos, se conocen como MQ Java).

Java base de MQ está contenido en los archivos `.jar` de Java™ siguientes:

com.ibm.mq.jar	Este código incluye soporte para todas las opciones de conexión.
com.ibm.mq.iiop.jar	Este código sólo ofrece soporte para la conexión de VisiBroker. Sólo se proporciona en la plataforma Windows®.
com.ibm.mqbind.jar	Este código sólo ofrece soporte para la conexión de enlaces y no se suministra ni se ofrece soporte para el mismo en todas las plataformas. Se aconseja no utilizarlo en ninguna aplicación nueva.

MQ JMS está contenido en el archivo `.jar` de Java™ siguiente:

com.ibm.mqjms.jar

Las siguientes bibliotecas Java™ de Sun Microsystems se redistribuyen con el producto MQ JMS:

connector.jar	Versión 1.0 Public Draft
fscontext.jar	Early Access 4 Release
jms.jar	Versión 1.0.2
jndi.jar	Versión 1.1.2

ldap.jar	Versión 1.0.3
providerutil.jar	Versión 1.0

Nota: En OS/390®, sólo se proporciona el archivo **com.ibm.mq.jar**. Este archivo ofrece soporte para la conexión de enlaces a MQSeries desde UNIX® System Services y CICS® Transaction Server para OS/390®.

Para las instrucciones de instalación, consulte el apartado correspondiente a la plataforma que necesita:

AIX®, HP-UX y Sun Solaris	"Instalación en UNIX®"
AS/400®	"Instalación en AS/400®" en la página 11
Linux	"Instalación en Linux" en la página 11
Windows®	"Instalación en Windows®" en la página 12

Cuando haya finalizado la instalación, los archivos y los ejemplos estarán instalados en las ubicaciones que se muestran en el apartado "Directorios de instalación" en la página 12.

Después de la instalación, debe actualizar las variables de entorno, tal como se muestra en el apartado "Variables de entorno" en la página 12.

Nota: Preste atención si instala el producto y, a continuación, instala o vuelve a instalar MQSeries básico. Asegúrese de que no instala Java base de MQ versión 5.1, puesto que el soporte de MQSeries Java™ revertirá un nivel hacia atrás.

Instalación en UNIX®

En este apartado se describe cómo instalar MQ Java en AIX®, HP-UX y Sun Solaris. Para obtener información sobre el modo de instalar Java base de MQ en Linux, consulte el apartado "Instalación en Linux" en la página 11.

Nota: Si es una instalación sólo de cliente (es decir, NO se ha instalado un servidor MQSeries), debe establecer el mqm de ID de usuario y de grupo. Si desea obtener más información, consulte la publicación MQSeries Comienzo rápido que corresponda a su plataforma.

1. Inicie la sesión como root.
2. Copie el archivo `ma88_xxx.tar.Z` en formato binario y almacénelo en el directorio /tmp, donde*xxx* es el identificador correspondiente a la plataforma:
   - `aix` AIX®
   - `hp10` HP-UXv10
   - `hp11` HP-UXv11
   - `sol` Sun Solaris
3. Entre los mandatos siguientes (donde *xxx* es el identificador correspondiente a la plataforma):

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

   Estos mandatos crean los directorios y los archivos adecuados.
4. Utilice la herramienta de instalación adecuada para cada plataforma:
   - Para AIX®, utilice `smitty` y:

a. Desinstale todos los componentes que empiecen por mqm.java.
b. Instale los componentes desde el directorio /tmp.

- Para HP-UX, utilice sam e instale desde los archivos ma88_hp10 o ma88_hp11, según proceda.

  Nota: Java™ no ofrece soporte para la página de códigos 1051 (que es el valor por omisión para HP-UX). Para ejecutar el intermediario de publicación/suscripción en HP-UX, quizá necesite cambiar el CCSID del gestor de colas del intermediario por un valor alternativo como, por ejemplo, 819.
- Para Sun Solaris, entre el mandato siguiente y seleccione las opciones que necesita:

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

  A continuación, entre el mandato siguiente:

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

Instalación en AS/400®

En este apartado se describe cómo instalar Java base de MQ en AS/400®.

1. Copie el archivo ma88_400.zip en un directorio de su PC.
2. Descomprima el archivo utilizando el recurso Unzip de InfoZip.

   De este modo se crea el archivo ma88_400.sav.
3. Cree un archivo denominado MA88 en una biblioteca adecuada en AS/400® como, por ejemplo, la biblioteca QGPL:

   ```
   CRTSAVF FILE(QGPL/MA88)
   ```
4. Transfiera ma88_400.sav a este archivo como una imagen binaria. Si utiliza FTP para hacerlo, el mandato put debe ser similar alsiguiente:

   ```
   PUT C:\TEMP\MA88_400.SAV QGPL/MA88
   ```
5. Instale MQSeries Classes for Java, ID de producto 5648C60, utilizando RSTLICPGM:

   ```
   RSTLICPGM LICPGM(5648C60) DEV(*SAVF) SAVF(QGPL/MA88)
   ```
6. Suprima el archivo que creó en el Paso 3:

   ```
   DLTF FILE(QGPL/MA88)
   ```

Instalación en Linux

En este apartado se describe cómo instalar MQ Java en Linux.

Para Linux, hay dos archivos de instalación disponibles: ma88_linux.tgz y MQSeriesJava™-5.2.0-1.noarch.rpm. Cada archivo proporciona una instalación idéntica.

Si dispone de acceso root al sistema de destino, o si utiliza una base de datos RPM (Red Hat Package Manager) para instalar paquetes, utilice MQSeriesJava™-5.2.0-1.noarch.rpm.

Si no dispone de acceso root al sistema de destino, o si el sistema de destino no tiene RPM instalado, utilice ma88_linux.tgz.

Para realizar la instalación utilizando ma88_linux.tgz:

1. Seleccione un directorio de instalación para el producto (por ejemplo, /opt).

Si este directorio no está situado en el directorio inicial, es posible que deba iniciar la sesión como root.

2. Copie el archivo `ma88_linux.tgz` en el directorio inicial.
3. Vaya al directorio de instalación que ha seleccionado, por ejemplo:
   ```
   cd /opt
   ```
4. Entre el mandato siguiente:
   ```
   tar -xpzf ~/ma88_linux.tgz
   ```

   De este modo se crea y rellena un directorio denominado mqm en el directorio actual (por ejemplo, /opt).

Para realizar la instalación utilizando Java™-5.2.0-1.noarch.rpm:

1. Inicie la sesión como root.
2. Copie MQSeriesJava™-5.2.0-1.noarch.rpm en un directorio de trabajo.
3. Entre el mandato siguiente:
   ```
   rpm -i MQSeriesJava-5.2.0-1.noarch.rpm
   ```

   De este modo, se instala el producto en /opt/mqm/. Se puede instalar en otra vía de acceso (consulte la documentación de RPM para obtener información más detallada).

Instalación en Windows®

En este apartado se describe cómo instalar MQ Java en Windows®.

1. Cree un directorio vacío denominado tmp y conviértalo en el directorio actual.
2. Copie el archivo `ma88_win.zip` en este directorio.
3. Descomprima `ma88_win.zip` utilizando el recurso Unzip de InfoZip.
4. Ejecute `setup.exe` desde este directorio y siga las instrucciones que se indican en las ventanas que aparecen.

   Nota: Si sólo desea instalar Java base de MQ seleccione las opciones adecuadas en esta fase.

Directorios de instalación

Los archivos de MQ Java V5.2 se instalan en los directorios que se muestran en la Tabla 2.

Tabla 2. Directorios de instalación del producto

Plataforma	Directorio
AIX®	usr/mqm/java/
AS/400®	/QIBM/ProdData/mqm/java/
HP-UX y Sun Solaris	opt/mqm/java/
Linux	*dir_instalación*/mqm/java/
Windows® 95, 98, 2000 y NT	*dir_instalación*\
Nota: *dir_instalación* es el directorio en el que ha instalado el producto. En Linux, es probable que sea /opt.	

Variables de entorno

Después de la instalación, debe actualizar la variable de entorno CLASSPATH para incluir los directorios de ejemplo y el código de Java base de MQ. En la Tabla 3 en la página 13

la página 13 se muestran los valores de CLASSPATH habituales para las distintas plataformas.

Tabla 3. Sentencias CLASSPATH de ejemplo para el producto

Plataforma	CLASSPATH de ejemplo
AIX®	CLASSPATH=*dir_jdk*/lib/classes.zip: /usr/mqm/java/lib/com.ibm.mq.jar: /usr/mqm/java/lib/connector.jar: /usr/mqm/java/lib: /usr/mqm/java/samples/base:
HP-UX y Sun Solaris	CLASSPATH=*dir_jdk*/lib/classes.zip: /opt/mqm/java/lib/com.ibm.mq.jar: /opt/mqm/java/lib/connector.jar: /opt/mqm/java/lib: /opt/mqm/java/samples/base:
Windows® 95, 98, 2000 y NT	CLASSPATH=C:*dir_jdk*\lib\classes.zip; *dir_instalación*\lib\com.ibm.mq.jar; *dir_instalación*\lib\com.ibm.mq.iiop.jar; *dir_instalación*\lib\connector.jar; *dir_instalación*\lib\; *dir_instalación*\samples\base\;
AS/400®	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/lib/connector.jar: /QIBM/ProdData/mqm/java/lib: /QIBM/ProdData/mqm/java/samples/base:
Linux	CLASSPATH=*dir_jdk*/lib/classes.zip: *dir_instalación*/mqm/java/lib/com.ibm.mq.jar: *dir_instalación*/mqm/java/lib/connector.jar: *dir_instalación*/mqm/java/lib: *dir_instalación*/mqm/java/samples/base:
Notas: 1. *dir_jdk* es el directorio en el que se ha instalado JDK™. 2. *dir_instalación* es el directorio en el que se ha instalado el producto.	

Para utilizar MQ JMS, debe incluir archivos jar adicionales a la classpath. Se enumeran en el apartado “Configuración después de la instalación” en la página 23.

Si hay aplicaciones existentes con una dependencia en el paquete de enlaces `com.ibm.mqbind` ignorado, debe añadir el archivo `com.ibm.mqbind.jar` a la classpath.

En algunas plataformas, debe actualizar variables de entorno adicionales, tal como se muestra en la Tabla 4.

Tabla 4. Variables de entorno para el producto

Plataforma	Variable de entorno
AIX®	LD_LIBRARY_PATH=/usr/mqm/java/lib
HP_UX	SHLIB_PATH=/opt/mqm/java/lib
Sun Solaris	LD_LIBRARY_PATH=/opt/mqm/java/lib

Tabla 4. Variables de entorno para el producto (continuación)

Plataforma	Variable de entorno
Windows® 95, 98, 2000 y NT	PATH=*dir_instalación*\lib
Nota: *dir_instalación* es el directorio de instalación para el producto	

Notas:

1. Para utilizar MQSeries Enlaces para Java en OS/400®, asegúrese de que la biblioteca QMQMJAVA está en la lista de bibliotecas.
2. Asegúrese de que añade las variables de MQSeries y que no escribe encima de ninguna de las variables de entorno existentes del sistema. Si escribe encima de variables de entorno existentes del sistema, la aplicación puede no responder durante la compilación o la ejecución.

Configuración del servidor web

Si instala MQSeries Java™ en un servidor web, puede bajar y ejecutar aplicaciones Java™ de MQSeries en máquinas que no tengan Java™ de MQSeries instalado localmente. Para que el servidor web pueda acceder a los archivos Java™ de MQSeries, debe definir la configuración del servidor web de forma que indique al directorio donde está instalado el cliente. En la documentación del servidor web encontrará información detallada sobre cómo efectuar esta configuración.

Nota: En OS/390®, las clases instaladas no ofrecen soporte para la conexión de cliente y no resultan útiles si se bajan a los clientes. Sin embargo, se pueden transferir los archivos jar de otras plataformas a OS/390® y utilizarlos para atender a los clientes.

Capítulo 3. Utilización de MQSeries Classes for Java (Java base de MQ)

En este capítulo se describe:

- Cómo configurar el sistema para ejecutar la applet de ejemplo y programas de aplicación para verificar la instalación de Java base de MQ
- Cómo modificar los procedimientos para ejecutar sus propios programas

Los procedimientos dependen de la opción de conexión que desee utilizar. Siga las instrucciones de este apartado que se ajusten a sus necesidades.

Utilización de la applet de ejemplo para verificar el cliente TCP/IP

Java base de MQ incluye una applet de verificación de la instalación, `mqjavac.html`. La puede utilizar para verificar la modalidad de cliente conectado TCP/IP de Java base de MQ. (Consulte también el apartado "Verificación con la aplicación de ejemplo" en la página 18).

La applet se conecta a un gestor de colas determinado, efectúa todas las invocaciones MQSeries y emite mensajes de diagnóstico en caso de que se produzca alguna anomalía.

Puede ejecutar la applet desde el visor de applets que se proporciona con JDK™. El visor de applets puede acceder a un gestor de colas de cualquier sistema principal.

En todos los casos, si la applet no finaliza correctamente, siga las indicaciones que se proporcionan en los mensajes de diagnóstico e intente ejecutar la applet otra vez.

Utilización de la applet de ejemplo en AS/400®

El sistema operativo OS/400® no tiene una interfaz gráfica de usuario (GUI) nativa. Para ejecutar la applet de ejemplo, debe utilizar AWT (Remote Abstract Window Toolkit for Java™) o CBJ (Class Broker for Java™) en hardware con posibilidades de gráficos. También puede verificar el cliente desde la línea de mandatos (consulte el apartado "Verificación con la aplicación de ejemplo" en la página 18).

Configuración del gestor de colas para que acepte conexiones de cliente

Utilice los procedimientos siguientes para configurar el gestor de colas para que acepte las peticiones de conexión entrantes de los clientes.

Cliente TCP/IP

1. Defina un canal de conexión de servidor utilizando los procedimientos siguientes:

   Para plataformas que no sean AS/400®:

   a. Inicie el gestor de colas utilizando el mandato strmqm.
   b. Escriba el mandato siguiente para iniciar el programa runmqsc:

      ```
      runmqsc
      ```

   c. Defina un canal de ejemplo denominado JAVA.CHANNEL, escribiendo:

      ```
      DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Sample channel for MQSeries Client for Java')
      ```

   Para la plataforma AS/400®:

   a. Inicie un gestor de colas utilizando el mandato STRMQM.
   b. Defina un canal de ejemplo denominado JAVA.CHANNEL escribiendo:

      ```
      CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
      MCAUSERID(SOMEUSERID) TEXT('Sample channel for MQSeries Client for Java')
      ```

      donde `QMGRNAME` es el nombre del gestor de colas y `SOMEUSERID` es un ID de usuario de AS/400® con la autorización necesaria para los recursos de MQSeries.

2. Inicie un programa de escucha con los mandatos siguientes:

   Para los sistemas operativosOS/2® y NT:

   Emita el mandato:

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

   Nota: Si utiliza el gestor de colas por omisión, puede omitir la opción -m.

   Utilización de VisiBroker para Java™ en el sistema operativo Windows NT®:

   Inicie el servidor IIOP (Internet Inter-ORB Protocol) con el mandato siguiente:

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

   Nota: Para detener el servidor IIOP, emita el mandato siguiente:

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

   Para los sistemas operativos UNIX®:

   Configure el daemon inetd, para que inetd inicie los canales MQSeries. Consulte la publicación *MQSeries® Clientes* para obtener instrucciones sobre cómo efectuar dicha operación.

   Para el sistema operativo OS/400®:

   Emita el mandato:

   ```
   STRMQMLSR MQMNAME(QMGRNAME)
   ```

   donde `QMGRNAME` es el nombre del gestor de colas.

Ejecución desde el visor de applets

Para utilizar este método, debe haber instalado JDK (Java™ Developer's Kit) en la máquina.

Procedimiento para la instalación local

1. Vaya al directorio de ejemplos de su idioma.
2. Escriba:

   ```
   appletviewer mqjavac.html
   ```

Procedimiento de instalación del servidor web:

Entre el mandato:

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

Notas:

1. En algunas plataformas, el mandato es 'applet', y no 'appletviewer'.
2. En algunas plataformas, puede que necesite seleccionar 'Properties' en el menú 'Applet' en la parte superior izquierda de la pantalla y luego establecer 'Network Access' en 'Unrestricted'.

La utilización de esta técnica le permitirá conectarse a cualquier gestor de colas que se ejecute en cualquier sistema principal al que tenga acceso TCP/IP.

Personalización de la applet de verificación

El archivo `mqjavac.html` incluye algunos parámetros opcionales. Estos parámetros le permiten modificar la applet para adaptarla a sus necesidades. Cada parámetro se define en una línea de HTML, cuyo aspecto es el siguiente:

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

Para especificar un valor del parámetro, elimine el símbolo de admiración inicial y edite el valor como desee. Puede especificar los parámetros siguientes:

hostname Valor que se visualiza inicialmente en el cuadro de edición del nombre del sistema principal.

port Valor que se visualiza inicialmente en el cuadro de edición del número de puerto.

channel Valor que se visualiza inicialmente en el cuadro de edición del canal.

queueManager Valor que se visualiza inicialmente en el cuadro de edición del gestor de colas.

userID Utiliza el ID de usuario especificado al conectarse al gestor de colas.

password Utiliza la contraseña especificada al conectarse al gestor de colas.

trace Hace que Java base de MQ escriba anotaciones de rastreo. Utilice esta opción sólo si se lo indica el servicio técnico de IBM®.

Verificación con la aplicación de ejemplo

Con Java base de MQ se proporciona un programa de verificación de la instalación, MQIVP. Puede utilizar esta aplicación para probar todas las modalidades de conexión de Java base de MQ. El programa le solicita que seleccione diversos elementos y otros datos para determinar la modalidad de conexión que desea verificar. Utilice el procedimiento siguiente para verificar la instalación:

1. Para probar una conexión de cliente:
   a. Configure el gestor de colas, tal como se describe en el apartado "Configuración del gestor de colas para que acepte conexiones de cliente" en la página 16.
   b. Lleve a cabo el resto del procedimiento en la máquina cliente.

   Para probar una conexión de enlaces, lleve a cabo el resto del procedimiento en la máquina servidor MQSeries.
2. Vaya al directorio de ejemplos.
3. Escriba:
   ```
   java MQIVP
   ```
   El programa intenta:
   a. Conectarse y desconectarse del gestor de colas indicado.
   b. Abrir, transferir, obtener y cerrar la cola local por omisión del sistema.
   c. Devolver un mensaje si las operaciones son satisfactorias.
4. En el mensaje de solicitud (1), deje 'MQSeries' por omisión.
5. En el mensaje de solicitud (2):
   - Para utilizar una conexión TCP/IP, entre un nombre de sistema principal servidor MQSeries.
   - Para utilizar la conexión nativa (modalidad de enlaces), deje el campo en blanco. (No entre ningún nombre).

A continuación se muestra un ejemplo de los mensajes de solicitud y las respuestas que puede que vea. Los mensajes de solicitud reales y las respuestas dependerán de la red MQSeries.

```
Entre el tipo de conexión (MQSeries)                 :  (MQSeries)(1)
  Entre la dirección IP del servidor MQSeries        :  myhost(2)
  Entre el puerto al que debe conectarse             :   (1414) (3)
  Entre el nombre de canal de conexión del servidor  : JAVA.CHANNEL(3)
  Entre el nombre del gestor de colas                :
  Correcto: Se ha conectado al gestor de colas.
  Correcto: Se ha abierto SYSTEM.DEFAULT.LOCAL.QUEUE
  Correcto: Se ha transferido un mensaje a
SYSTEM.DEFAULT.LOCAL.QUEUE
  Correcto: Se ha obtenido un mensaje de SYSTEM.DEFAULT.LOCAL.QUEUE
  Correcto: Se ha cerrado SYSTEM.DEFAULT.LOCAL.QUEUE
  Correcto: Se ha desconectado del gestor de colas

  Pruebas finalizadas -
  CORRECTO: Esta transmisión está funcionando correctamente.
  Pulse Intro para continuar...
```

Notas:
1. Si elige la conexión de servidor, no verá los mensajes de solicitud marcados(3).
2. En OS/390®, no se muestran los mensajes de solicitud (1), (2) y (3).

3. En OS/400®, sólo puede ejecutar el mandato `java MQIVP` desde la interfaz interactiva Qshell (Qshell es la opción 30 de OS/400®, 5769-SS1). De forma alternativa, puede ejecutar la aplicación utilizando el mandato CL `RUNJVA CLASS(MQIVP)`.
4. Para utilizar los enlaces MQSeries para Java™ en OS/400®, debe asegurarse de que la biblioteca QMQMJAVA está incluida en la lista de bibliotecas.

Utilización de la conectividad de VisiBroker

Si utiliza VisiBroker, los procedimientos que se describen en el apartado "Configuración del gestor de colas para que acepte conexiones de cliente" en la página 16 no son necesarios.

Para probar una instalación utilizando VisiBroker, siga los procedimientos que se describen en el apartado "Verificación con la aplicación de ejemplo" en la página 18, pero en el mensaje de solicitud (1), escriba `VisiBroker`, respetando exactamente las mayúsculas y minúsculas.

Utilización de CICS® Transaction Server para OS/390®

1. Defina el programa de aplicación de ejemplo para CICS®.
2. Defina una transacción para ejecutar la aplicación de ejemplo.
3. Transfiera el nombre del gestor de colas al archivo que se utiliza para la entrada estándar.
4. Ejecute la transacción.

La salida del programa se sitúa en los archivos que se utilizan para la salida estándar y de errores.

Consulte la documentación de CICS® para obtener más información sobre cómo ejecutar programas Java™ y establecer los archivos de entrada y salida.

Ejecución de los programas Java base de MQ propios

Para ejecutar sus propias aplicaciones o applets Java™, utilice los procedimientos que se describen para los programas de verificación, sustituyendo 'mqjavac.html' o 'MQIVP' por el nombre de la aplicación.

Para obtener más información sobre cómo crear applets y aplicaciones Java base de MQ, consulte la "Parte 2. Programación con Java base de MQ" en la página 51.

Solución de problemas de Java base de MQ

Si un programa no finaliza correctamente, ejecute la applet o el programa de verificación de la instalación y siga los consejos que se proporcionan en los mensajes de diagnóstico. Ambos se describen en el "Capítulo 3. Utilización de MQSeries Classes for Java (Java base de MQ)" en la página 15.

Si sigue teniendo problemas y necesita ponerse en contacto con el equipo de servicio técnico de IBM, puede que se le solicite que active el recurso de rastreo. El método para realizar esta operación depende de si está ejecutando en modalidad de cliente o de enlaces. Consulte en los apartados siguientes los procedimientos adecuados para su sistema.

Rastreo de la applet de ejemplo

Para ejecutar el rastreo para la applet de ejemplo, edite el archivo `mqjavac.html`. Busque la línea siguiente:

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

Elimine el símbolo de admiración y cambie el valor 1 por un número del 1 al 5 en función del nivel de detalle necesario. (Cuanto mayor sea el número, más información se reunirá). Entonces la línea tendrá el siguiente aspecto:

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

donde 'n' es un número entre 1 y 5.

La salida de rastreo aparece en la consola de Java o en el archivo de anotaciones Java del navegador de la web.

Rastreo de la aplicación de ejemplo

Para rastrear el programa `MQIVP`, entre lo siguiente:

```
java MQIVP -trace n
```

donde 'n' es un número entre 1 y 5, en función del nivel de detalle necesario. (Cuanto mayor sea el número, más información se reunirá).

Para obtener más información sobre cómo utilizar el rastreo, consulte el apartado "Rastreo de programas Java base de MQ" en la página 78.

Rastreo con CICS Transaction Server para OS/390

Cuando se utiliza CICS Transaction Server para OS/390, no se pueden proporcionar argumentos de línea de mandatos directamente al programa. Debe crear un pequeño programa ajustador que invoque a MQIVP.main() con los argumentos adecuados.

Mensajes de error

He aquí algunos de los mensajes de error más frecuentes que puede que vea:

No se ha podido identificar la dirección IP del sistema principal local
El servidor no está conectado a la red.

Acción recomendada: Conecte el servidor a la red y vuélvalo a intentar.

No se ha podido cargar el archivo gatekeeper.ior
Esta anomalía puede producirse en un servidor web que pone en funcionamiento applets VisiBroker, cuando el archivo `gatekeeper.ior` no está ubicado en el lugar correcto.

Acción recomendada: Reinicie el archivo gatekeeper de VisiBroker desde el directorio en el que ha difundido la applet. El archivo `gatekeeper` se graba en este directorio.

Anomalía: Falta software. Puede ser MQSeries o la variable VBROKER_ADM
Esta anomalía se produce en el programa de ejemplo MQIVP si el entorno de software Java está incompleto.

Acción recomendada: En el cliente, asegúrese de que la variable de entorno VBROKER_ADM está establecida para el directorio VisiBroker para administración Java (`adm`) y vuélvalo a intentar.

En el servidor, asegúrese de que está instalada la versión más reciente de Java base de MQ y vuélvalo a intentar.

NO_IMPLEMENT
Existe un problema de comunicaciones que implica VisiBroker Smart Agents.

Acción recomendada: Consulte la documentación de VisiBroker.

COMM_FAILURE
Existe un problema de comunicaciones que implica VisiBroker Smart Agents.

Acción recomendada: Utilice el mismo número de puerto para todos los VisiBroker Smart Agents y vuélvalo a intentar. Consulte la documentación de VisiBroker.

MQRC_ADAPTER_NOT_AVAILABLE
Si obtiene este error cuando está intentando utilizar VisiBroker, es probable que no se pueda encontrar la clase JAVA org.omg.CORBA.ORB en CLASSPATH.

Acción recomendada: Asegúrese de que la sentencia CLASSPATH incluye la vía de acceso a los archivos `vbjorb.jar` y `vbjapp.jar` de VisiBroker.

MQRC_ADAPTER_CONN_LOAD_ERROR
Cuando obtiene este error cuando está intentando ejecutar en OS/390, asegúrese de que los conjuntos de datos SCSQANLE y SCSQAUTH de MQSeries están en la sentencia STEPLIB.

Mensajes de error

Capítulo 4. Utilización del Servicio de mensajes de MQSeries Classes for Java (MQ JMS)

En este capítulo se describen las tareas siguientes:

- Cómo configurar el sistema para utilizar los programas de prueba y de ejemplo.
- Cómo ejecutar el programa IVT (Prueba de verificación de la instalación) para verificar la instalación del Servicio de mensajes de MQSeries Classes for Java
- Cómo ejecutar el programa PSIVT (Prueba de verificación de la instalación de Publish/Subscribe) para verificar la instalación de Publish/Subscribe
- Cómo ejecutar sus programas

Configuración después de la instalación

Para que todos los recursos necesarios estén disponibles para los programas MQ JMS, debe actualizar las variables del sistema siguientes:

Classpath

Una operación satisfactoria de los programas JMS requiere que determinados paquetes Java™ estén disponibles para JVM. Los debe especificar en la classpath después de obtener e instalar los paquetes necesarios.

Añada los archivos .jar a la classpath:

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

Variables de entorno

El subdirectorio `bin` de la instalación de MQ JMS contiene algunos scripts. Su finalidad es utilizarlos como métodos abreviados para varias acciones comunes. Muchos de estos scripts dan por supuesto que se ha definido la variable de entorno `MQ_JAVA_INSTALL_PATH` y que señala al directorio en que se ha instalado MQ JMS. No es obligatorio establecer esta variable pero, si no lo hace, debe editar los scripts del directorio `bin` según proceda.

En Windows NT®, puede establecer la classpath en una nueva variable de entorno utilizando el separador **Entorno** de **Propiedades del sistema**. En UNIX®, normalmente se establecen desde los scripts de inicio de sesión de cada usuario. En cualquier plataforma, puede elegir si desea utilizar scripts para mantener classpaths diferentes y otras variables de entorno para distintos proyectos.

Configuración adicional para la modalidad Publish/Subscribe

Antes de utilizar la implementación MQ JMS de JMS Publish/Subscribe, se debe realizar alguna configuración adicional:

Asegúrese de que el intermediario está ejecutándose

Para verificar que el intermediario de MQSeries Publish/Subscribe está instalado y en ejecución, utilice el mandato siguiente:

```
dspmqbrk -m MI.GESTOR.COLAS
```

donde `MI.GESTOR.COLAS` es el nombre del gestor de colas en el que está ejecutándose el intermediario. Si el intermediario está ejecutándose, se muestra un mensaje similar al siguiente:

```
El intermediario de mensajes MQSeries para el gestor
de colas  MI.GESTOR.COLAS está en ejecución.
```

Si el sistema operativo informa que no puede ejecutar el mandato `dspmqbrk`, asegúrese de que el intermediario de MQSeries Publish/Subscribe está correctamente instalado.

Si el sistema operativo informa que el intermediario no está activo, inícielo utilizando el mandato:

```
strmqbrk -m MI.GESTOR.COLAS
```

Cree las colas del sistema MQ JMS

Para que la implementación de MQ JMS Publish/Subscribe funcione correctamente, se deben crear varias colas del sistema. En el subdirectorio `bin` de la instalación de MQ JMS se proporciona un script que le puede ayudar a realizar esta tarea. Para utilizar el script, entre el mandato siguiente:

```
runmqsc MI.GESTOR.COLAS < MQJMS_PSQ.mqsc
```

En caso de que se produzca algún error, compruebe si ha escrito correctamente el nombre del gestor de colas y si el gestor de colas está en ejecución.

Colas para las que los usuarios sin privilegios necesitan autorización

Los usuarios sin privilegios necesitan que se les otorgue autorización para acceder a las colas que utiliza JMS. Para obtener información más detallada sobre el control de acceso en MQSeries, consulte el capítulo referente a la protección de objetos MQSeries en la publicación *MQSeries® Administración del sistema*.

En la modalidad punto a punto de JMS, las cuestiones relacionadas con el control de acceso son similares a las de MQSeries Classes for Java:

- Las colas que utiliza QueueSender necesitan autorización de transferencia (put).
- Las colas que utilizan QueueReceivers y QueueBrowsers necesitan autorizaciones de obtención (get), consulta (inq) y examen (browse).
- El método QueueSession.createTemporaryQueue necesita acceso a la cola modelo que se define en el campo temporaryModel de QueueConnectionFactory (por omisión, es SYSTEM.DEFAULT.MODEL.QUEUE).

Para la modalidad JMS Publish/Subscribe, se utilizan las colas del sistema siguientes:

SYSTEM.JMS.ADMIN.QUEUE

SYSTEM.JMS.REPORT.QUEUE
SYSTEM.JMS.MODEL.QUEUE
SYSTEM.JMS.PS.STATUS.QUEUE
SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
SYSTEM.JMS.D.SUBSCRIBER.QUEUE
SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
SYSTEM.BROKER.CONTROL.QUEUE

Además, cualquier aplicación que publique mensajes necesita acceso a la cola STREAM que se especifica en la fábrica de conexión de temas que se está utilizando. El valor por omisión es:

SYSTEM.BROKER.DEFAULT.STREAM

Ejecución de IVT punto a punto

En este apartado se describe IVT (programa de Prueba de verificación de la instalación) punto a punto que se proporciona con MQ JMS.

IVT intenta verificar la instalación al conectar al gestor de colas por omisión de la máquina local utilizando MQ JMS en la modalidad de enlaces. A continuación, envía un mensaje a la cola `SYSTEM.DEFAULT.LOCAL.QUEUE` y lo vuelve a leer.

Puede ejecutar el programa utilizando una de las dos modalidades posibles siguientes:

Con búsqueda JNDI de objetos administrados
La modalidad JNDI fuerza al programa para obtener sus objetos administrados de un espacio de nombres JNDI, que es la operación que se espera de las aplicaciones cliente JMS. (En el apartado "Administración de objetos JMS" en la página 40 se proporciona una descripción de los objetos administrados). Este método de invocación tiene los mismos requisitos previos que la herramienta de administración (consulte el "Capítulo 5. Utilización de la herramienta de administración de MQ JMS" en la página 35).

Sin búsqueda JNDI de objetos administrados
Si no desea utilizar JNDI, los objetos administrados se pueden crear durante la ejecución al ejecutar IVT en modalidad que no sea JNDI. Puesto que la configuración de un depósito basado en JNDI es relativamente compleja, se aconseja ejecutar primero IVT sin JNDI.

Verificación punto a punto sin JNDI

Se proporciona un script, denominado `IVTRun` en UNIX®, o `IVTRun.bat` en Windows NT®, para ejecutar IVT. Este archivo está instalado en el subdirectorio `bin` de la instalación.

Para ejecutar la prueba sin JNDI, emita el mandato siguiente:

```
IVTRun -nojndi
```

Para ejecutar la prueba en modalidad cliente, sin JNDI, emita el mandato siguiente:

```
IVTRun -nojndi -client -m <grcl> -host <nombreSistemaPrincipal> [-port <puerto>]
            [-channel <canal>]
```

donde:

grcl es el nombre del gestor de colas al que se desea conectar

nombreSistemaPrincipal
es el sistema principal en el que está ejecutándose el gestor de colas

puerto es el puerto TCP/IP en el que está ejecutándose el escucha del gestor de colas (el valor por omisión es 1414)

canal es el canal de la conexión de cliente (el valor por omisión es SYSTEM.DEF.SVRCONN)

Si la prueba finaliza correctamente, debe visualizar una salida similar a la siguiente:

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

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message: Message Class:   jms_text          JMSType:          null
JMSDeliveryMode: 2                              JMSExpiration:    0
JMSPriority:     4                              JMSMessageID:     ID:414d5120716
d312020202020202020203000c43713400000
JMSTimestamp:    935592657000                   JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:      null
JMSRedelivered:  false
JMS_IBM_Format:MQSTR            JMS_IBM_PutApplType:11
JMSXGroupSeq:1                  JMSXDeliveryCount:0
JMS_IBM_MsgType:8               JMSXUserID:kingdon
JMSXApplID:D:\jdk1.1.8\bin\java.exe
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Verificación punto a punto con JNDI

Para ejecutar IVT con JNDI, el servidor LDAP debe estar en ejecución y configurado para aceptar objetos Java™. Si se visualiza un mensaje similar al siguiente, indica que existe una conexión con el servidor LDAP, pero el servidor no está correctamente configurado:

```
No se puede enlazar el objeto
```

Este mensaje significa que el servidor no está almacenando objetos Java™ o que los permisos de los objetos o el sufijo no son correctos. Consulte el apartado "Comprobación de la configuración del servidor LDAP" en la página 373.

Además, se deben poder recuperar los objetos administrados siguientes de un espacio de nombres JNDI:

- MQQueueConnectionFactory
- MQQueue

Se proporciona un script, denominado `IVTSetup` en UNIX®, e `IVTSetup.bat` en Windows NT®, para crear estos objetos automáticamente. Entre el mandato:

```
IVTSetup
```

El script invoca la herramienta de administración MQ JMS (consulte el "Capítulo 5. Utilización de la herramienta de administración de MQ JMS" en la página 35) y crea los objetos en un espacio de nombres JNDI.

IVT punto a punto

MQQueueConnectionFactory se enlaza con el nombre `ivtQCF` (para LDAP, `cn=ivtQCF`). Todas las propiedades son valores por omisión:

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

MQQueue se enlaza con el nombre `ivtQ` (`cn=ivtQ`). El valor de la propiedad `QUEUE` se convierte en `QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)`. Todas las demás propiedades tienen valores por omisión:

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

Después de crear los objetos administrados en el espacio de nombres JNDI, ejecute el script `IVTRun` (`IVTRun.bat` en Windows NT®) utilizando el mandato siguiente:

```
IVTRun [ -t ] [ -url <"URLsuministrador"> [ -icf <fábCtxInic> ] ]
```

donde:

-t significa activar el rastreo (por omisión, el rastreo está desactivado)

URLsuministrador
es la ubicación JNDI de los objetos administrados. Si se está utilizando la fábrica de contextos iniciales, es un URL LDAP con el formato siguiente:

```
ldap://nombreSistemaPrincipal.empresa.com/nombreContexto
```

Si se utiliza un suministrador de servicio del sistema de archivos, (consulte `initCtxFact` más abajo), el URL tiene el formato:

```
archivo://directorioEsp
```

Nota: Incluya la serie de caracteres *URLsuministrador* entre comillas (″).

fábCtxInic es el nombre de clase de la fábrica de contextos iniciales. El valor por omisión es un suministrador de servicio LDAP, y tiene el valor siguiente:

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

Si utiliza un suministrador de servicio del sistema de archivos, establezca el parámetro en:

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

Si la prueba finaliza correctamente, la salida es similar a la salida que no es JNDI, excepto que las líneas 'create' `QueueConnectionFactory` y `Queue` indican la recuperación del objeto de JNDI. En el fragmento de código siguiente se muestra un ejemplo.

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

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
 ...
 ...
```

Aunque no es estrictamente necesario, se aconseja eliminar los objetos que crea el script IVTSetup del espacio de nombres JNDI. Se proporciona un script denominado `IVTTidy` (`IVTTidy.bat` en Windows NT®) para esta finalidad.

Recuperación de errores de IVT

Las indicaciones siguientes pueden resultarle útiles si el resultado de la prueba no es satisfactorio.

- Para obtener ayuda con cualquier mensaje de error relacionado con la classpath, compruebe si la classpath está establecida correctamente, tal como se describe en el apartado "Configuración después de la instalación" en la página 23.
- IVT puede no ejecutarse correctamente y mostrar un mensaje de 'anomalía al crear MQQueueManager', con un mensaje adicional que incluya el número 2059. Indica que MQSeries no se ha podido conectar al gestor de colas local por omisión de la máquina en la que se ha ejecutado IVT. Compruebe si el gestor de colas está en ejecución y aparece marcado como gestor de colas por omisión.
- Un mensaje de 'anomalía al abrir la cola MQ' indica que MQSeries se ha conectado al gestor de colas por omisión, pero no ha podido abrir 'SYSTEM.DEFAULT.LOCAL.QUEUE'. Puede significar que la cola no existe en el gestor de colas por omisión o que la cola no está habilitada para PUT y GET. Añada o habilite la cola mientras dure la prueba.

En la Tabla 5 se enumeran las clases que prueba IVT y el paquete del que proceden:

Tabla 5. Clases que prueba IVT

Clase	**Archivo Jar**
Clases MQSeries® JMS	com.ibm.mqjms.jar
com.ibm.mq.MQMessage	com.ibm.mq.jar
javax.jms.Message	jms.jar
javax.naming.InitialContext	jndi.jar
javax.resource.cci.Connection	connector.jar
javax.transaction.xa.XAException	jta.jar
com/sun/jndi/toolkit/ComponentDirContext	providerutil.jar
com.sun.jndi.ldap.LdapCtxFactory	ldap.jar

Prueba de verificación de la instalación de Publish/Subscribe

El programa PSIVT (Prueba de verificación de la instalación de Publish/Subscribe) sólo se proporciona en formato compilado. Se encuentra en el paquete `com.ibm.mq.jms`.

PSIVT intenta:

1. Crear un publicador, `p`, que publique sobre el tema `MQJMS/PSIVT/Information`
2. Crear un suscriptor, `s`, suscrito al tema `MQJMS/PSIVT/Information`
3. Utilizar `p` para publicar un mensaje de texto simple
4. Utilizar `s` para recibir un mensaje en espera en la cola de entrada

Cuando se ejecuta PSIVT, el publicador publica el mensaje y el suscriptor lo recibe y lo visualiza. El publicador publica en la corriente de datos por omisión del intermediario. El suscriptor es no duradero, no lleva a cabo ninguna selección de mensaje y acepta los mensajes de las conexiones locales. Realiza una recepción síncrona, y espera 5 segundos, como máximo, la llegada de un mensaje.

Puede ejecutar PSIVT, del mismo modo que IVT, en la modalidad JNDI o en modalidad autónoma. La modalidad JNDI utiliza JNDI para recuperar una `TopicConnectionFactory` y un `Topic` de un espacio de nombres JNDI. Si no se utiliza JNDI, estos objetos se crean durante la ejecución.

Verificación de Publish/Subscribe sin JNDI

Se proporciona un script 'PSIVTRun' denominado `PSIVTRun` (`PSIVTRun.bat` en Windows NT®) para ejecutar PSIVT. El archivo se encuentra en el subdirectorio `bin` de la instalación.

Para ejecutar la prueba sin JNDI, emita el mandato siguiente:

```
PSIVTRun -nojndi [-m <grcl>] [-t]
```

Para ejecutar la prueba en modalidad de cliente, sin JNDI, emita el mandato siguiente:

```
PSIVTRun -nojndi -client -m <grcl> -host <nombreSistemaPrincipal> [-port <puerto>]
               [-channel <canal>] [-t]
```

donde:

-nojndi significa que no se realiza la búsqueda JNDI de los objetos administrados

grcl es el nombre del gestor de colas al que se desea conectar

nombreSistemaPrincipal
es el sistema principal en el que está ejecutándose el gestor de colas

puerto es el puerto TCP/IP en el que está ejecutándose el escucha del gestor de colas (el valor por omisión es 1414)

canal es el canal de la conexión de cliente (el valor por omisión es SYSTEM.DEF.SVRCONN)

-t significa activar el rastreo (por omisión, el rastreo está desactivado)

Si la prueba finaliza correctamente, la salida es similar a la siguiente:

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service
Publish/Subscribe Installation Verification Test
Creating a TopicConnectionFactory
Creating a Topic
Creating a Connection
Creating a Session
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding Text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive...

Got message:

JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d2e504f4c415249532e4254b7dc3753700000
  JMSTimestamp:    937232048000
  JMSCorrelationID:ID:414d5158000000000000000000000000000000000000000000
  JMSDestination:  topic
  ://MQJMS/PSIVT/Information
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_Format:MQSTR
  UNIQUE_CONNECTION_ID:937232047753
  JMS_IBM_PutApplType:26
  JMSXGroupSeq:1
  JMSXDeliveryCount:0
  JMS_IBM_MsgType:8
  JMSXUserID:hollingl
  JMSXApplID:QM.POLARIS.BROKER
A simple text message from the MQJMSPSIVT program

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

Verificación de Publish/Subscribe con JNDI

Para ejecutar PSIVT en la modalidad de JNDI, se deben poder recuperar dos objetos administrados de un espacio de nombres JNDI:

- Un TopicConnectionFactory enlazado con el nombre `ivtTCF`
- Un Topic enlazado con el nombre `ivtT`

Puede definir estos objetos utilizando la herramienta de administración MQ JMS (consulte el “Capítulo 5. Utilización de la herramienta de administración de MQ JMS” en la página 35) y los mandatos siguientes:

```
DEFINE TCF(ivtTCF)
```

Este mandato define la TopicConnectionFactory.

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

Este mandato define el Topic.

IVT de Publish/Subscribe

En estas definiciones se da por supuesto que hay un gestor de colas por omisión disponible, en el que está ejecutándose el intermediario. Para obtener información más detallada sobre la configuración de estos objetos para utilizar un gestor de colas que no sea el valor por omisión, consulte el apartado "Administración de objetos JMS" en la página 40. Estos objetos deben residir en un contexto al que señale el parámetro de línea de mandatos `-url` que se describe más abajo.

Para ejecutar la prueba en la modalidad JNDI, entre el mandato siguiente:

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

donde:

-t significa activar el rastreo (por omisión, el rastreo está desactivado)

-url <purl> es el URL de la ubicación JNDI en la que residen los objetos administrados

-icf <fcinic> es la initialContextFactory para JNDI [`com.sun.jndi.ldap.LdapCtxFactory`]

Si la prueba finaliza correctamente, la salida es similar a la salida que no es JNDI, excepto que las líneas 'create' `QueueConnectionFactory` y `Queue` indican la recuperación del objeto de JNDI.

Recuperación de errores de PSIVT

Las indicaciones siguientes pueden resultarle útiles si el resultado de la prueba no es satisfactorio.

- Si visualiza un mensaje similar al siguiente:

  ```
  *** ¡El intermediario no está en funcionamiento! Arránquelo utilizando 'strmqbrk' ***
  ```

  indica que el intermediario está instalado en el gestor de colas de destino, pero su cola de control contiene algunos mensajes pendientes, lo que significa que el intermediario no está en ejecución. Para iniciarlo, utilice el mandato `strmqbrk`. (Consulte el apartado "Configuración adicional para la modalidad Publish/Subscribe" en la página 24).
- Si se muestra un mensaje similar al siguiente:

  ```
  No se puede conectar con el gestor de colas: <valor por omisión>
  ```

  asegúrese de que el sistema MQSeries ha configurado un gestor de cola por omisión.
- Si se muestra un mensaje similar al siguiente:

  ```
  No se puede conectar con el gestor de colas: ...
  ```

  asegúrese de que la TopicConnectionFactory administrada que utiliza PSIVT está configurada con un nombre de gestor de colas válido. De forma alternativa, si utiliza la opción `-nojndi`, asegúrese de que indica un gestor de colas válido (utilice la opción `-m`).
- Si se muestra un mensaje similar al siguiente:

  ```
  No se puede acceder a la cola de control del intermediario en el gestor de colas: ...
  Asegúrese de que el intermediario está instalado en este gestor de colas
  ```

  asegúrese de que la TopicConnectionFactory administrada que utiliza PSIVT está configurada con el nombre del gestor de colas en el que está instalado el intermediario. Si utiliza la opción `-nojndi`, asegúrese de que indica el nombre de un gestor d e colas (utilice la opción `-m`).

Ejecución de programas MQ JMS propios

Para obtener información sobre cómo crear sus propios programas MQ JMS, consulte el “Capítulo 10. Creación de programas MQ JMS” en la página 181.

MQ JMS incluye un archivo de programa de utilidad, `runjms`(`runjms.bat` en Windows NT®), para ayudarle a ejecutar los programas que se suministran y los programas que ha creado.

El programa de utilidad facilita ubicaciones por omisión para los archivos de anotaciones y de rastreo, y le permite añadir los parámetros de ejecución de aplicación que necesite su aplicación. En el script que se proporciona se da por supuesto que la variable de entorno MQ_JAVA_INSTALL_PATH está establecida en el directorio en el que ha instalado MQ JMS. En el script también se da por supuesto que los subdirectorios `trace` y `log` de dicho directorio se utilizan para la salida de rastreo y anotaciones, respectivamente. Son sólo las ubicaciones sugeridas. Puede editar el script para utilizar el directorio que desee.

Para ejecutar la aplicación, utilice el mandato siguiente:

```
runjms <nombre de clase de aplicación> [argumentos específicos de la aplicación]
```

Para obtener más información sobre la creación de applets y aplicaciones MQ JMS, consulte la “Parte 3. Programación con MQ JMS” en la página 179.

Solución de problemas

Si un programa no finaliza correctamente, ejecute el programa de verificación de la instalación, que se describe en el “Capítulo 4. Utilización del Servicio de mensajes de MQSeries Classes for Java (MQ JMS)” en la página 23 y siga los consejos que se proporcionan en los mensajes de diagnóstico.

Rastreo de programas

El recurso de rastreo de MQ JMS se proporciona con el fin de ayudar al personal de IBM a diagnosticar problemas de los clientes.

Por omisión, el rastreo está inhabilitado, puesto que la salida adquiere rápidamente grandes dimensiones y es poco probable que resulte útil en circunstancias normales.

Si desea que se proporcione salida de rastreo, puede habilitarla estableciendo la propiedad de Java MQJMS_TRACE_LEVEL en uno de los valores siguientes:

on sólo rastrea las invocaciones de MQ JMS

base rastrea las invocaciones de MQ JMS y las invocaciones de Java base de MQ subyacente

Por ejemplo:

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

Para inhabilitar el rastreo, establezca MQJMS_TRACE_LEVEL en **off**.

Por omisión, la salida del rastreo se sitúa en un archivo denominado `mqjms.trc` en el directorio de trabajo actual. Puede redirigirla a otro directorio utilizando la propiedad de Java `MQJMS_TRACE_DIR`.

Ejecución del rastreo de MQ JMS

Por ejemplo:

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

El script del programa de utilidad `runjms` establece estas propiedades utilizando las variables de entorno `MQJMS_TRACE_LEVEL` y `MQ_JAVA_INSTALL_PATH`, tal como se indica a continuación:

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

Sólo es una sugerencia y la puede modificar según proceda.

Anotaciones cronológicas

El recurso de anotaciones de MQ JMS se proporciona para informar de problemas graves, en especial, los que pueden indicar errores de configuración, más que errores de programación. Por omisión, la salida de las anotaciones se envía a la corriente de datos `System.err` que, normalmente, aparece en `stderr` de la consola en la que se ejecuta JVM.

Puede redirigir la salida a un archivo utilizando una propiedad de Java que especifique la nueva ubicación como, por ejemplo:

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

El script de programa de utilidad `runjms`, del directorio bin de la instalación de MQ JMS, establece la propiedad en:

```
<VÍA_ACCESO_INSTALACIÓN_MQ_JAVA>/log
```

donde `VÍA_ACCESO_INSTALACIÓN_MQ_JAVA` es la vía de acceso de la instalación de MQ JMS. Es una sugerencia y la puede modificar según proceda.

Cuando se redirigen las anotaciones a un archivo, la salida tiene un formato binario. Para visualizar las anotaciones, se proporciona el programa de utilidad `formatLog` (`formatLog.bat` en Windows NT), que convierte el archivo en texto sin formato. El programa de utilidad está almacenado en el directorio `bin` de la instalación de MQ JMS. Ejecute la conversión tal como se indica a continuación:

```
formatLog <archivoEntrada> <archivoSalida>
```

Capítulo 5. Utilización de la herramienta de administración de MQ JMS

La herramienta de administración permite que los administradores puedan definir las propiedades de ocho tipos de objetos MQ JMS y los almacenen en un espacio de nombres JNDI. Después, los clientes JMS pueden recuperar los objetos administrados del espacio de nombres con JNDI y utilizarlos.

Los objetos JMS que se pueden administrar utilizando la herramienta son los siguientes:
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

Para obtener información más detallada sobre estos objetos, consulte el apartado "Administración de objetos JMS" en la página 40.

Nota: JMSWrapXAQueueConnectionFactory y JMSWrapXATopicConnectionFactory son clases específicas de WebSphere. Están contenidas en el paquete **com.ibm.ejs.jms.mq**.

La herramienta también permite que los administradores manipulen subcontextos de espacio de nombres de directorio en JNDI. Consulte el apartado "Manipulación de subcontextos" en la página 40.

Invocación de la herramienta de administración

La herramienta de administración tiene una interfaz de línea de mandatos. La puede utilizar de modo interactivo o para iniciar un proceso por lotes. La modalidad interactiva proporciona un indicador de mandatos en el que puede entrar mandatos de administración. En la modalidad de proceso por lotes, el mandato para iniciar la herramienta incluye el nombre de un archivo que contiene un script de mandatos de administración.

Para iniciar la herramienta en la modalidad interactiva, entre el mandato:

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

donde:

-t	Habilita el rastreo (el valor por omisión es que esté desactivado)
-v	Produce una salida detallada (el valor por omisión es la salida concisa)
-cfg nombreArchivo_config	Nombre de un archivo de configuración alternativo (consulte el apartado "Configuración" en la página 36)

Invocación de la herramienta de administración

Se visualiza un indicador de mandatos, que indica que la herramienta está preparada para aceptar mandatos de administración. Inicialmente, este indicador aparece como:

```
InitCtx>
```

indica que el contexto actual (es decir, el contexto JNDI al que hacen referencia actualmente todas las operaciones de directorio y denominación) es el contexto inicial definido en el parámetro de configuración PROVIDER_URL (consulte el apartado "Configuración").

A medida que avanza en el espacio de nombres de directorio, el indicador cambia para reflejarlo, de modo que siempre muestra el contexto actual.

Para iniciar la herramienta en la modalidad de proceso por lotes, entre el mandato:

```
JMSAdmin <test.scp
```

donde *test.scp* es un archivo de script que contiene mandatos de administración (consulte el apartado "Mandatos de administración" en la página 39). El último mandato del archivo debe ser el mandato `END`.

Configuración

Debe configurar la herramienta de administración con valores para los tres parámetros siguientes:

INITIAL_CONTEXT_FACTORY
Indica el suministrador de servicio que utiliza la herramienta. Actualmente, hay tres valores para lo que se ofrece soporte para esta propiedad:
- com.sun.jndi.ldap.LdapCtxFactory (para LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (para el contexto de sistema de archivos)
- com.ibm.ejs.ns.jndi.CNInitialContextFactory (para trabajar con el depósito CosNaming de WebSphere)

PROVIDER_URL
Indica el URL del contexto inicial de la sesión, la raíz de todas las operaciones JNDI que realiza la herramienta. Actualmente, se ofrece soporte para tres formatos de esta propiedad:
- ldap://nombreSistemaPrincipal/nombreContexto (para LDAP)
- archivo:[unidad:]/nombreVíaAcceso (para el contexto de sistema de archivos)
- iiop://nombreSistemaPrincipal[:puerto] /[?ContextoDestino=ctx] (para acceder al espacio de nombres CosNaming "base" de WebSphere)

SECURITY_AUTHENTICATION
Indica si JNDI omite las credenciales de seguridad al suministrador de servicio. Este parámetro sólo se utiliza cuando se usa un suministrador de servicio LDAP. Actualmente, esta propiedad puede tomar uno de los tres valores siguientes:
- none (autentificación anonymous (anónima))
- simple (autentificación simple)
- CRAM-MD5 (mecanismo de autentificación CRAM-MD5)

Si no se proporciona un valor válido, la propiedad toma como valor por omisión none. Para obtener información más detallada sobre la seguridad con la herramienta de administración, consulte el apartado "Seguridad" en la página 38.

Estos parámetros se establecen en un archivo de configuración. Al invocar la herramienta, puede especificar esta configuración utilizando el parámetro de línea de mandatos `-cfg`, como se describe en el apartado "Invocación de la herramienta de administración" en la página 35. Si no especifica ningún nombre de archivo de configuración, la herramienta intenta cargar el archivo de configuración por omisión (`JMSAdmin.config`). En primer lugar, busca el archivo en el directorio actual y, a continuación, en el directorio `<VÍA_ACCESO_INSTALACIÓN_MQ_JAVA>/bin`. (Donde <VÍA_ACCESO_INSTALACIÓN_MQ_JAVA> es la vía de acceso de la instalación de MQ JMS).

El archivo de configuración es un archivo de texto sin formato que consta de un conjunto de pares de clave-valor, separados mediante un signo de igualdad '=', tal como se muestra en el ejemplo siguiente:

```
#Establecer el suministrador de servicio
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Establecer el contexto inicial
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Establecer el tipo de autentificación
    SECURITY_AUTHENTICATION=none
```

(El símbolo '#' en la primera columna de la línea indica un comentario o una línea que no se utiliza).

La instalación incluye un archivo de configuración de ejemplo que se denomina `JMSAdmin.config` y se encuentra en el directorio `<VÍA_ACCESO_INSTALACIÓN_MQ_JAVA>/bin`. Edítelo para adaptar la configuración a su sistema.

Configuración para WebSphere

Para utilizar la herramienta de administración (o cualquier aplicación cliente que deba realizar búsquedas posteriormente) para trabajar con el depósito CosNaming de WebSphere, necesita la configuración siguiente:

- CLASSPATH debe incluir archivos jar relacionados con JNDI de WebSphere:
  - Para WebSphere V3.5:

    `<WSAppserver>\lib\ejs.jar`
- PATH para WebSphere V.3.5 debe incluir:

  `<WSAppserver>\jdk\jre\bin`

  donde <WSAppserver> es la vía de acceso de instalación para WebSphere.

Seguridad

Los administradores deben conocer el efecto de la propiedad `SECURITY_AUTHENTICATION` descrita en el apartado “Configuración” en la página 36.

- Si se establece este parámetro en *none*, JNDI no pasa ninguna credencial de seguridad al suministrador de servicio, y se realiza una ″autentificación anónima″.
- Si se establece el parámetro en `simple` o `CRAM-MD5`, las credenciales de seguridad se pasan a través de JNDI al suministrador de servicio subyacente. Estas credenciales de seguridad tienen el formato de un nombre de usuario distinguido (DN de usuario) y una contraseña.

Si se necesitan las credenciales de seguridad, se le solicitan al usuario al inicializar la herramienta.

Nota: El texto que se escribe se visualiza en la pantalla, y también incluye la contraseña. Por este motivo, tenga cuidado de que las contraseñas no queden expuestas a usuarios no autorizados.

La herramienta no realiza ninguna autentificación; delega la tarea al servidor LDAP. Es responsabilidad del administrador del servidor LDAP configurar y mantener los privilegios de acceso a las distintas partes del directorio. Si la autentificación no se ejecuta correctamente, la herramienta muestra un mensaje de error adecuado y concluye.

Puede obtener información más detallada sobre la seguridad y JNDI en la documentación que se proporciona en el sitio web Java™ de Sun (`http://java.sun.com`).

Mandatos de administración

Cuando se visualiza el indicador de mandatos, la herramienta está preparada para aceptar mandatos. Los mandatos de administración tienen, generalmente, el formato siguiente:

```
verbo [param]*
```

donde *verbo* es uno de los verbos de administración que se enumeran en la Tabla 6. Todos los mandatos válidos constan, como mínimo, de un verbo (y sólo uno), que aparece al principio del mandato en la forma abreviada o estándar.

Los parámetros que puede tomar un verbo dependen de éste ultimo. Por ejemplo, el verbo `END` no puede tomar ningún parámetro, pero el verbo `DEFINE` puede tomar de 1 a 20 parámetros. En apartados posteriores de este capítulo se detallan los verbos que pueden tomar como mínimo un parámetro.

Tabla 6. Verbos de administración

Verbo		**Descripción**
Estándar	**Forma abrev.**	
ALTER	ALT	Cambia como mínimo una de las propiedades de un objeto administrado determinado
DEFINE	DEF	Crea y almacena un objeto administrado o crea un nuevo subcontexto
DISPLAY	DIS	Muestra las propiedades de uno o más objetos administrados o el contenido del contexto actual
DELETE	DEL	Elimina uno o más objetos administrados del espacio de nombres o elimina un subcontexto vacío
CHANGE	CHG	Modifica el contexto actual, permitiendo que el usuario se desplace a cualquier lugar del espacio de nombres de directorio bajo el contexto inicial (dependiente del permiso de seguridad)
COPY	CP	Realiza una copia de un objeto administrado almacenado, guardándolo con otro nombre
MOVE	MV	Modifica el nombre bajo el que se ha almacenado un objeto administrado
END		Cierra la herramienta de administración

Los nombres de los verbos no son sensibles a las mayúsculas y minúsculas.

Por lo general, para terminar los mandatos, debe pulsar la tecla de retorno de carro. Sin embargo, lo puede alterar temporalmente escribiendo el símbolo '+' directamente antes del retorno de carro. De este modo, puede entrar mandatos de varias líneas, tal como se muestra en el ejemplo siguiente:

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

Las líneas que empieza con un carácter *, # o / se tratan como comentarios o líneas que se ignoran.

Manipulación de subcontextos

Puede utilizar los verbos `CHANGE`, `DEFINE`, `DISPLAY` y `DELETE` para manipular subcontextos de espacio de nombres de directorio. Su utilización se describe en la Tabla 7.

Tabla 7. Sintaxis y descripción de los mandatos que se utilizan para manipular subcontextos

Sintaxis de los mandatos	Descripción
DEFINE CTX(ctxName)	Intenta crear un nuevo subcontexto dependiente del contexto actual, con el nombre ctxName. No se ejecuta correctamente si se produce una violación de la seguridad, si el subcontexto ya existe o si el nombre proporcionado no es válido.
DISPLAY CTX	Muestra el contenido del contexto actual. Los objetos administrados se anotan con `'a'` y los subcontextos con `'[D]'`. También se muestra el tipo Java de cada objeto.
DELETE CTX(ctxName)	Intenta suprimir el contexto dependiente con el nombre ctxName del contexto actual. No se ejecuta correctamente si no se encuentra el contexto, si no está vacío o si se produce una violación de la seguridad.
CHANGE CTX(ctxName)	Modifica el contexto actual, de modo que haga referencia al contexto dependiente denominado ctxName. Se puede proporcionar uno de los dos valores especiales de ctxName siguientes: **=UP** mueve al elemento superior del contexto actual **=INIT** mueve directamente al contexto inicial No se ejecuta correctamente si el contexto especificado no existe o si se produce una violación de la seguridad.

Administración de objetos JMS

En este apartado se describen los ocho tipos de objetos que puede manejar la herramienta de administración. Incluye detalles sobre cada una de sus propiedades configurables y los verbos que pueden manipularlos.

Tipos de objetos

La Tabla 8 muestra los ocho tipos de objetos administrados. La columna Palabra clave muestra las series de caracteres que puede utilizar para sustituir *TIPO* en los mandatos de la Tabla 9 en la página 42.

Tabla 8. Tipos de objetos JMS que maneja la herramienta de administración

Tipo de objeto		Descripción
Java	**Palabra clave**	
MQQueueConnectionFactory	QCF	Implementación de MQSeries® de la interfaz QueueConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en el dominio punto a punto de JMS.

Tabla 8. Tipos de objetos JMS que maneja la herramienta de administración (continuación)

Tipo de objeto		Descripción
Java	**Palabra clave**	
MQTopicConnectionFactory	TCF	Implementación de MQSeries® de la interfaz TopicConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en el dominio de publicación/suscripción de JMS.
MQQueue	Q	Implementación de MQSeries de la interfaz de cola (Queue) de JMS. Representa un destino para los mensajes en el dominio punto a punto de JMS.
MQTopic	T	Implementación de MQSeries de la interfaz de tema (Topic) de JMS. Representa un destino para mensajes en el dominio de publicación/suscripción de JMS.
MQXAQueueConnectionFactory[1]	XAQCF	Implementación de MQSeries® de la interfaz XAQueueConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en el dominio punto a punto de JMS que utiliza las versiones XA de las clases JMS.
MQXATopicConnectionFactory[1]	XATCF	Implementación de MQSeries® de la interfaz XATopicConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en el dominio de publicación/suscripción de JMS que utiliza las versiones XA de las clases JMS.
JMSWrapXAQueueConnectionFactory[2]	WSQCF	Implementación de MQSeries® de la interfaz QueueConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en el dominio punto a punto de JMS que utiliza las versiones XA de las clases JMS con WebSphere.
JMSWrapXATopicConnectionFactory[2]	WSTCF	Implementación de MQSeries® de la interfaz TopicConnectionFactory de JMS. Representa un objeto de fábrica para crear conexiones en el dominio de publicación/suscripción de JMS que utiliza las versiones XA de las clases JMS con WebSphere.

1. Estas clases se proporcionan para que las utilicen los proveedores de servidores de aplicaciones. Es improbable que los programadores de aplicaciones puedan utilizarlas directamente.
2. Utilice este estilo de ConnectionFactory si desea que las sesiones JMS participen en las transacciones globales que coordina WebSphere™.

Verbos que se utilizan con objetos JMS

Puede utilizar los verbos `ALTER`, `DEFINE`, `DISPLAY`, `DELETE`, `COPY` y `MOVE` para manipular objetos administrados en el espacio de nombres de directorio. En la Tabla 9 se resume su utilización. Sustituya *`TIPO`* por la palabra clave que representa el objeto administrado necesario, tal como se indica en la Tabla 8 en la página 40.

Tabla 9. Sintaxis y descripción de los mandatos que se utilizan para manipular objetos administrados

Sintaxis de los mandatos	Descripción
ALTER *TIPO*(nombre) [propiedad]*	Intenta actualizar las propiedades del objeto administrado especificado con las que se proporcionan. No se ejecuta correctamente si se produce una violación de la seguridad, si no se ha podido encontrar el objeto especificado o si las nuevas propiedades suministradas no son válidas.
DEFINE *TIPO*(nombre) [propiedad]*	Intenta crear un objeto administrado de tipo *`TIPO`* con las propiedades suministradas y trata de almacenarlo con el nombre `nombre` en el contexto actual. No se ejecuta correctamente si se produce una violación de la seguridad, si el nombre proporcionado no es válido o ya existe, o si las propiedades suministradas no son válidas.
DISPLAY *TIPO*(nombre)	Muestra las propiedades del objeto administrado de tipo *`TIPO`*, enlazado bajo el nombre `nombre` en el contexto actual. No se ejecuta correctamente si el objeto no existe o si se produce una violación de la seguridad.
DELETE *TIPO*(nombre)	Intenta eliminar el objeto administrado de tipo *`TIPO`*, con el nombre `nombre`, del contexto actual. No se ejecuta correctamente si el objeto no existe o si se produce una violación de la seguridad.
COPY *TIPO*(nombreA) *TIPO*(nombreB)	Hace una copia del objeto administrado de tipo *`TIPO`*, con el nombre `nombreA`, denominando la copia `nombreB`. Todo esto se lleva a cabo en el ámbito del contexto actual. No se ejecuta correctamente si el objeto copiado no existe, si el objeto con el nombreB ya existe o si se produce una violación de la seguridad.
MOVE *TIPO*(nombreA) *TIPO*(nombreB)	Mueve (redenomina) el objeto administrado de tipo *`TIPO`* con el nombre `nombreA`, por el `nombreB`. Todo esto se lleva a cabo en el ámbito del contexto actual. No se ejecuta correctamente si el objeto que se va a mover no existe, si ya existe un objeto con el nombreB o si se produce una violación de la seguridad.

Creación de objetos

Los objetos se crean y se almacenan en un espacio de nombres JNDI utilizando la sintaxis de mandatos siguiente:

```
DEFINE TIPO(nombre) [propiedad]*
```

Es decir, el verbo `DEFINE` seguido de una referencia de objeto administrado de *`TIPO`*`(nombre)` y después cero o más *propiedades* (consulte el apartado "Propiedades" en la página 44).

Consideraciones de denominación de LDAP

Para almacenar objetos en un entorno LDAP, sus nombres deben cumplir convenios determinados. Uno de ellos es que los nombres de subcontexto y de objetos deben incluir un prefijo como, por ejemplo, `cn=` (nombre común) o `ou=` (unidad de organización).

La herramienta de administración simplifica la utilización de suministradores de servicio LDAP al permitir que se puedan consultar los nombres de subcontexto y de objeto sin un prefijo. Si no se proporciona ningún prefijo, la herramienta añade automáticamente un prefijo por omisión (actualmente, `cn=`) al nombre que se indica.

Se muestra en el ejemplo siguiente.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

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

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

Tenga en cuenta que, aunque el nombre de objeto proporcionado (`testQueue`) no tiene ningún prefijo, la herramienta añade uno automáticamente para garantizar su conformidad con las normas del convenio de denominación de LDAP. Del mismo modo, al emitir el mandato `DISPLAY Q(testQueue)` también se añade este prefijo.

Es posible que deba configurar el servidor LDAP para almacenar objetos Java. En el "Apéndice C. Configuración del servidor LDAP para objetos Java™" en la página 373 puede obtener información que le puede ayudar para esta configuración.

Propiedades

Una propiedad consta de un par de nombre-valor con el formato:

```
NOMBRE_PROPIEDAD(valor_propiedad)
```

Los nombres de las propiedades no son sensibles a las mayúsculas y minúsculas y están restringidos al conjunto de nombres reconocidos que se muestra en la Tabla 10. En esta tabla también se muestran los valores de propiedad válidos para cada propiedad.

Tabla 10. Nombres de propiedades y valores válidos

Propiedad		**Valores válidos (los valores por omisión se muestran en negrita)**
Estándar	**Forma abrev.**	
DESCRIPTION	DESC	Cualquier serie de caracteres
TRANSPORT	TRAN	• **BIND** - Las conexiones utilizan enlaces MQSeries. • CLIENT - Se utiliza la conexión de cliente.
CLIENTID	CID	Cualquier serie de caracteres
QMANAGER	QMGR	Cualquier serie de caracteres
HOSTNAME	HOST	Cualquier serie de caracteres
PORT		Cualquier entero positivo
CHANNEL	CHAN	Cualquier serie de caracteres
CCSID	CCS	Cualquier entero positivo
RECEXIT	RCX	Cualquier serie de caracteres
RECEXITINIT	RCXI	Cualquier serie de caracteres
SECEXIT	SCX	Cualquier serie de caracteres
SECEXITINIT	SCXI	Cualquier serie de caracteres
SENDEXIT	SDX	Cualquier serie de caracteres
SENDXITINIT	SDXI	Cualquier serie de caracteres
TEMPMODEL	TM	Cualquier serie de caracteres

Tabla 10. Nombres de propiedades y valores válidos (continuación)

Propiedad		**Valores válidos (los valores por omisión se muestran en negrita)**
Estándar	**Forma abrev.**	
MSGRETENTION	MRET	• **Yes** - Los mensajes no deseados permanecen en la cola de entrada. • No - Los mensajes no deseados se tratan conforme a sus opciones de disposición.
BROKERVER	BVER	**V1** - El único valor que se permite actualmente.
BROKERPUBQ	BPUB	Cualquier serie de caracteres (el valor por omisión es **SYSTEM.BROKER.DEFAULT.STREAM**).
BROKERSUBQ	BSUB	Cualquier serie de caracteres (el valor por omisión es **SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE**).
BROKERDURSUBQ	BDSUB	Cualquier serie de caracteres (el valor por omisión es **SYSTEM.JMS.D.SUBSCRIPTION.QUEUE**).
BROKERCCSUBQ	CCSUB	Cualquier serie de caracteres (el valor por omisión es **SYSTEM.JMS.ND.CC.SUBSCRIPTION.QUEUE**).
BROKERCCDSUBQ	CCDSUB	Cualquier serie de caracteres (el valor por omisión es **SYSTEM.JMS.D.CC.SUBSCRIPTION.QUEUE**).
BROKERQMGR	BQM	Cualquier serie de caracteres
BROKERCONQ	BCON	Cualquier serie de caracteres
EXPIRY	EXP	• **APP** - La aplicación JMS puede definir la caducidad. • UNLIM - No existe caducidad. • Cualquier entero positivo que represente la caducidad en milisegundos.
PRIORITY	PRI	• **APP** - La aplicación JMS puede definir la prioridad. • QDEF - La prioridad toma el valor del valor por omisión de la cola. • Cualquier entero del 0 al 9.
PERSISTENCE	PER	• **APP** - La aplicación JMS puede definir la permanencia. • QDEF - La permanencia toma el valor del valor por omisión de la cola. • PERS - Los mensajes son permanentes. • NON - Los mensajes no son permanentes.
TARGCLIENT	TC	• **JMS** - El destino del mensaje es una aplicación JMS. • MQ - El destino del mensaje es una aplicación MQSeries tradicional, no JMS.
ENCODING	ENC	Vea también “Propiedad ENCODING” en la página 49
QUEUE	QU	Cualquier serie de caracteres
TOPIC	TOP	Cualquier serie de caracteres

Muchas propiedades sólo son adecuadas para un subconjunto específico de tipos de objeto. En la Tabla 11 en la página 46 se muestran las combinaciones de tipos de objeto-propiedad válidas y se proporciona una breve descripción de cada propiedad.

Administración de objetos JMS

Tabla 11. Combinaciones válidas de tipo de objeto y propiedad

Propiedad	Tipos de objeto válidos						Descripción
	QCF	TCF	Q	T	WSQCF XAQCF	WSTCF XATCF	
DESCRIPTION	S	S	S	S	S	S	Descripción del objeto almacenado
TRANSPORT	S	S			S[1]	S[1]	Si las conexiones van a utilizar Enlaces MQ o una conexión de cliente
CLIENTID	S	S			S	S	Identificador de serie de caracteres para el cliente
QMANAGER	S	S	S		S	S	Nombre del gestor de colas al que conectar
PORT	S	S					Puerto en el que escucha el gestor de colas
HOSTNAME	S	S					Nombre del sistema principal en el que reside el gestor de colas
CHANNEL	S	S					Nombre del canal de conexión con el cliente que se utiliza
CCSID	S	S	S	S			ID de juego de caracteres codificados que se va a utilizar en las conexiones
RECEXIT	S	S					Nombre de clase completo de la salida de recepción que se utiliza
RECEXITINIT	S	S					Serie de caracteres de inicialización de la rutina de salida de recepción
SECEXIT	S	S					Nombre de clase completo de la rutina de salida de seguridad que se utiliza
SECEXITINIT	S	S					Serie de caracteres de inicialización de la rutina de salida de seguridad
SENDEXIT	S	S					Nombre de clase completo de la rutina de salida de emisión que se utiliza
SENDEXITINIT	S	S					Serie de caracteres de inicialización de la rutina de salida de emisión
TEMPMODEL	S				S		Nombre de la cola modelo de la que se crean colas temporales
MSGRETENTION	S				S		Si el consumidor de conexiones mantiene los mensajes no deseados en la cola de entrada
BROKERVER		S				S	Versión del intermediario que se utiliza
BROKERPUBQ		S				S	Nombre de la cola de entrada del intermediario (cola de corriente de datos)
BROKERSUBQ		S				S	Nombre de la cola de la que se recuperan los mensajes de suscripciones no duraderas
BROKERDURSUBQ				S			Nombre de la cola de la que se recuperan los mensajes de suscripciones duraderas
BROKERCCSUBQ		S				S	Nombre de la cola de la que se recuperan los mensajes de las suscripciones no duraderas para un ConnectionConsumer

Tabla 11. Combinaciones válidas de tipo de objeto y propiedad (continuación)

Propiedad	Tipos de objeto válidos						Descripción
	QCF	TCF	Q	T	WSQCF XAQCF	WSTCF XATCF	
BROKERCCDSUBQ				S			Nombre de la cola en la que se recuperan los mensajes de las suscripciones duraderas para un ConnectionConsumer
BROKERQMGR		S				S	Gestor de colas en el que se ejecuta el intermediario
BROKERCONQ		S				S	Nombre de la cola de control del intermediario
EXPIRY			S	S			Período tras el cual caducan los mensajes de un destino
PRIORITY			S	S			Prioridad para los mensajes enviados a un destino
PERSISTENCE			S	S			Permanencia de los mensajes enviados a un destino
TARGCLIENT			S	S			Campo que indica si se utiliza el formato RFH2 de MQSeries para intercambiar información con las aplicaciones destino
ENCODING			S	S			Esquema de codificación que se utiliza para este destino
QUEUE			S				Nombre subyacente de la cola que representa este destino
TOPIC				S			Nombre subyacente del tema que representa este destino

Notas:

1. Para los objetos WSTCF, WSQCF, XATCF y XAQCF, sólo se permite el tipo de transporte BIND.
2. En el "Apéndice A. Correlación entre las propiedades de la herramienta de administración y las propiedades programables" en la página 369 se muestra la relación entre las propiedades que establece la herramienta y las propiedades programables.
3. La propiedad TARGCLIENT indica si se utiliza el formato RFH2 de MQSeries para intercambiar información con aplicaciones destino.

   La constante MQJMS_CLIENT_JMS_COMPLIANT indica que se utiliza el formato RFH2 para enviar información. Las aplicaciones que utilizan MQ JMS comprenden el formato RFH2. Debe establecer la constante MQJMS_CLIENT_JMS_COMPLIANT cuando intercambie información con una aplicación MQ JMS destino.

   La constante MQJMS_CLIENT_NONJMS_MQ indica que no se utiliza el formato RFH2 para enviar información. Generalmente, este valor se utiliza para una aplicación MQSeries existente (es decir, una aplicación que no maneje RFH2).

Dependencias de las propiedades

Algunas propiedades tienen dependencias entre sí, lo que puede significar que carecen de sentido para proporcionar una propiedad, a menos que se haya establecido otra propiedad en un valor determinado. Los dos grupos de propiedades específicos en los que puede ocurrir son las propiedades de cliente (Client) y las series de caracteres de inicialización de rutina de salida (Exit).

Propiedades de cliente

Si la propiedad `TRANSPORT(CLIENT)` no se ha establecido explícitamente en una fábrica de conexiones, el transporte que se utiliza en las conexiones que proporciona la fábrica es Enlaces MQ. Por este motivo, no se puede configurar ninguna propiedad de cliente en esta fábrica de conexiones. Son las siguientes:

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

Si intenta establecer estas propiedades sin establecer la propiedad `TRANSPORT` en `CLIENT`, se producirá un error.

Series de caracteres de inicialización de rutina de salida

No se puede establecer ninguna de las series de caracteres de inicialización de rutina de salida a menos que se indique el nombre de la rutina de salida correspondiente. Las propiedades de inicialización de la rutina de salida son las siguientes:

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

Por ejemplo, si se especifica `RECEXITINIT(miSerie)` sin especificar `RECEXIT(nombre.clase.alguna.rutina.salida)` se produce un error.

Propiedad ENCODING

Los valores válidos que puede tomar la propiedad ENCODING son más complejos que el resto de las propiedades. La propiedad de codificación (encoding) se construye a partir de tres subpropiedades:

codificación de entero normal o invertido

codificación decimal normal o invertido

codificación de coma flotante IEEE normal, IEEE invertido o System/390®

ENCODING se expresa como una serie de tres caracteres con la sintaxis siguiente:
{N|R}{N|R}{N|R|3}

En esta serie de caracteres:

- N indica normal
- R indica invertido
- 3 indica System/390®
- el primer carácter representa *codificación de entero*
- el segundo carácter representa *codificación decimal*
- el tercer carácter representa *codificación de coma flotante*

Proporciona un conjunto de doce valores posibles para la propiedad ENCODING.

Existe otro valor, la serie de caracteres NATIVE, que establece los valores de codificación adecuados para la plataforma Java.

Los valores siguientes muestran las combinaciones válidas para ENCODING:

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

Condiciones de error de ejemplo

En este apartado se proporcionan ejemplos de condiciones de error que pueden surgir durante la creación de un objeto.

Propiedad desconocida

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  No se ha podido crear un objeto válido, compruebe los parámetros suministrados
  Propiedad desconocida: PIZZA
```

Propiedad no válida para el objeto

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  No se ha podido crear un objeto válido, compruebe los parámetros suministrados
  Propiedad no válida para un QCF: PRI
```

Tipo de valor de propiedad no válido

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  No se ha podido crear un objeto válido, compruebe los parámetros suministrados
  Valor no válido para la propiedad CCS: English
```

Valor de propiedad fuera del rango válido

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  No se ha podido crear un objeto válido, compruebe los parámetros suministrados
  Valor no válido para la propiedad PRI: 12
```

Conflicto de propiedades - enlaces/cliente

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  No se ha podido crear un objeto válido, compruebe los parámetros suministrados
  Propiedad no válida en este contexto: Conflicto de atributos enlaces-cliente
```

Conflicto de propiedades - Inicialización de rutina de salida

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  No se ha podido crear un objeto válido, compruebe los parámetros suministrados
  Propiedad no válida en este contexto: Se ha suministrado la serie
  de caracteres ExitInit sin la serie de caracteres Exit
```

Parte 2. Programación con Java base de MQ

Capítulo 6. Introducción para programadores

Este capítulo contiene información general para programadores. Para obtener información más detallada sobre cómo crear programas, consulte el "Capítulo 7. Creación de programas Java base de MQ" en la página 57.

¿Por qué debo utilizar la interfaz Java™?

La interfaz de programación de MQSeries Classes for Java pone las numerosas ventajas de Java™ a su disposición como desarrollador de aplicaciones MQSeries®:

- El lenguaje de programación Java™ es **fácil de utilizar**.

  No son necesarios archivos de cabecera, punteros, estructuras, uniones ni sobrecargar de trabajo al operador. Los programas escritos en Java™ son más fáciles de desarrollar y depurar que sus equivalentes en C y C++.
- Java™ está **orientado a objetos**.

  Las características orientadas a objetos de Java™ son comparables a las de C++, pero la herencia múltiple no existe. En su lugar, Java™ utiliza el concepto de la interfaz.
- Java™ se **distribuye** de modo inherente.

  Las bibliotecas de clases Java™ contienen una biblioteca de rutinas para copiar con protocolos TCP/IP como HTTP y FTP. Los programas Java™ pueden acceder a los URL tan fácilmente como se accede a un sistema de archivos.
- Java™ es **sólido**.

  Java™ da mucha importancia a la comprobación temprana de posibles problemas, la comprobación (ejecución) dinámica y la eliminación de situaciones que son propensas al error. Java™ utiliza un concepto de referencias que elimina la posibilidad de grabar encima de la memoria y de corromper datos.
- Java™ es **seguro**.

  Java™ está destinado a ejecutarse en entornos distribuidos o de red y se ha hecho mucho hincapié en la seguridad. Los programas Java™ no pueden desbordar la pila de ejecución y no pueden corromper memoria fuera de su espacio de proceso. Los programas Java™ que se bajan de Internet no pueden leer o escribir archivos locales.
- Los programas Java™ son **portátiles**.

  En la especificación Java™ no existen aspectos "dependientes de la implementación". El compilador Java™ genera un formato de archivo de objeto neutro respecto a la arquitectura. El código compilado se puede ejecutar en muchos procesadores, siempre y cuando esté presente el sistema de tiempo de ejecución de Java™.

Si crea una aplicación utilizando MQSeries Classes for Java, los usuarios pueden bajar los códigos de bytes Java (denominados *applets*) para el programa desde Internet, y después pueden ejecutar las applets en sus propias máquinas. Esto significa que los usuarios con acceso al servidor web pueden cargar y ejecutar la aplicación sin ninguna instalación previa necesaria en sus máquinas.

Cuando se necesita actualizar el programa, se actualiza la copia del servidor web. La próxima vez que los usuarios accedan a la applet, recibirán automáticamente la última versión. Esto puede reducir significativamente los costes implicados en la

instalación y actualización de aplicaciones cliente tradicionales donde están implicados un gran número de sistemas de sobremesa.

Si coloca la applet en un servidor web que sea accesible fuera del cortafuegos corporativo, cualquier persona puede bajar y utilizar la aplicación de Internet. Esto significa que puede obtener en el sistema MQSeries mensajes procedentes de cualquier parte de Internet, lo que permite la creación de un conjunto totalmente nuevo de aplicaciones de servicio, soporte y comercio electrónico accesible a través de Internet.

La interfaz MQSeries Classes for Java

La interfaz de programación de aplicaciones de MQSeries orientada a procedimientos se basa en los verbos siguientes:

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

Todos estos verbos toman, como parámetro, un manejador para el objeto MQSeries en el que deben operar. Como Java™ está orientada a objetos, la interfaz de programación Java™ invierte este procedimiento. El programa consta de un conjunto de objetos MQSeries, sobre los que se actúa invocando métodos para dichos objetos, como en el ejemplo siguiente.

Cuando se utiliza la interfaz orientada a procedimientos, se desconecta del gestor de colas utilizando la invocación MQDISC(Hconn, CompCode, Reason), donde *Hconn* es un manejador para el gestor de colas.

En la interfaz Java™, el gestor de colas se representa mediante un objeto de clase MQQueueManager. Puede desconectarse del gestor de colas invocando al método disconnect() para dicha clase.

```
// declarar un objeto de tipo gestor de colas
MQQueueManager queueManager=new MQQueueManager();
...
// hacer algo...
...
// desconectar del gestor de colas
queueManager.disconnect();
```

Java™ Development Kit

Antes de compilar las applets o las aplicaciones que cree, debe tener acceso a JDK (Java™ Development Kit) para la plataforma de desarrollo. JDK™ contiene todas las clases Java™, las variables, los constructores y las interfaces estándar de los que dependen las clases MQSeries Classes for Java. También contiene las herramientas necesarias para compilar y ejecutar las applets y los programas en cada plataforma para la que se ofrece soporte.

Puede bajar JDK™ de IBM® Software Download Catalog, que está disponible en la World Wide Web en la dirección siguiente:

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

También puede desarrollar aplicaciones utilizando el JDK™ que se incluye con el entorno de desarrollo integrado de IBM® Visual Age para Java™.

Para poder compilar aplicaciones Java™ en la plataforma AS/400®, primero debe instalar:

- AS/400® Developer Kit para Java™, 5769-JV1
- El intérprete Qshell, OS/400® (5769-SS1) Opción 30

Biblioteca de clases MQSeries Classes for Java

MQSeries Classes for Java es un conjunto de clases Java™ que permiten que las applets y las aplicaciones Java™ interactúen con MQSeries.

Se proporcionan las clases siguientes:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

Se proporcionan las interfaces Java™ siguientes:
- MQC
- MQPoolServicesEventListener
- MQReceiveExit
- MQSecurityExit
- MQSendExit

También se proporciona la implementación de las interfaces Java™ siguientes. Sin embargo, estas interfaces no son para que las utilicen las aplicaciones directamente:
- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

En Java™, un *paquete* es un mecanismo para agrupar conjuntos de clases relacionadas. Las interfaces y las clases MQSeries se envían con un paquete Java™ denominado `com.ibm.mq`. Para incluir el paquete de MQSeries Classes for Java en su programa, añada la línea siguiente en la parte superior del archivo fuente:

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

biblioteca de clases Java base de MQ

Capítulo 7. Creación de programas Java base de MQ

Para utilizar MQSeries Classes for Java para acceder a colas MQSeries®, debe crear programas Java™ que contengan invocaciones que transfieran y obtengan mensajes de las colas MQSeries. Los programas pueden tomar la forma de *applets* Java™, *servlets* Java™ o *aplicaciones* Java™.

En este capítulo se proporciona información para ayudarle a crear applets, servlets y aplicaciones Java™ que interactúen con sistemas MQSeries. Encontrará información detallada sobre las clases individuales en el "Capítulo 9. Clases e interfaces Java base de MQ" en la página 87.

¿Debo crear applets o aplicaciones?

Debe crear applets, servlets o aplicaciones según la conexión que desee utilizar y el lugar desde el cual desee ejecutar los programas.

Las principales diferencias entre las applets y las aplicaciones son las siguientes:

- Las applets se ejecutan con un visor de applets o en un navegador de la web, los servlets se ejecutan en un servidor de aplicaciones web y las aplicaciones se ejecutan de forma autónoma.
- Las applets pueden bajarse de un servidor web a una máquina de navegador de la web, pero las aplicaciones y los servlets no.

Se aplican las normas generales siguientes:

- Si desea ejecutar los programas desde máquinas que no tienen MQSeries Classes for Java instalado localmente, debe crear applets.
- La modalidad de enlaces nativa de MQSeries Classes for Java no ofrece soporte para applets. Por consiguiente, si desea utilizar los programas en todas las modalidades de conexión, incluida la modalidad de enlaces nativa, debe crear servlets o aplicaciones.

Diferencias de conexión

El modo de programar para MQSeries Classes for Java tiene algunas dependencias según las modalidades de conexión que se deseen utilizar.

Conexiones de cliente

Cuando se utiliza MQSeries Classes for Java como cliente, es similar al cliente C de MQSeries, pero con las diferencias siguientes:

- Sólo ofrece soporte para TCP/IP.
- No ofrece soporte para tablas de conexiones.
- No lee ninguna variable de entorno de MQSeries en el arranque.
- La información que se almacenaría en una definición de canal y en variables de entorno se almacena en una clase denominada MQEnvironment. De forma alternativa, esta información se puede pasar como parámetros al efectuar la conexión.
- Las condiciones de error y excepción se escriben en unas anotaciones cronológicas especificadas en la clase MQException. El destino por omisión de los errores es la consola Java™.

Diferencias de conexión

Los clientes MQSeries Classes for Java no ofrecen soporte para el verbo MQBEGIN o los enlaces rápidos.

Para obtener información general sobre los clientes MQSeries, consulte la publicación *MQSeries® Clientes*.

Nota: Cuando se utiliza la conexión VisiBroker, los valores de contraseña e ID de usuario de MQEnvironment no se reenvían al servidor MQSeries. El ID de usuario efectivo es el que se aplica al servidor IIOP.

Modalidad de enlaces

La modalidad de enlaces de MQSeries Classes for Java difiere de las modalidades de cliente en lo siguiente:

- La mayoría de los parámetros que proporciona la clase MQEnvironment se ignoran
- Los enlaces ofrecen soporte para el verbo MQBEGIN y para enlaces rápidos en el gestor de colas MQSeries

Nota: MQSeries para AS/400® no ofrece soporte para la utilización de MQBEGIN para iniciar las unidades de trabajo globales que coordina el gestor de colas.

Definición de la conexión que se va a utilizar

La conexión la determina el valor de las variables de la clase MQEnvironment.

MQEnvironment.properties
Puede contener los pares clave/valor siguientes:

- Para conexiones de cliente y enlaces:

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

- Para conexiones VisiBroker:

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

MQEnvironment.hostname
Establezca el valor de esta variable del modo siguiente:

- Para conexiones de cliente, establézcalo en el nombre de sistema principal del servidor MQSeries al que desea conectarse
- Para la modalidad de enlaces, establézcalo en nulo

Fragmentos de código de ejemplo

En este apartado se incluyen dos fragmentos de código de ejemplo: la Figura 1 en la página 59 y la Figura 2 en la página 62. Cada uno utiliza una conexión determinada e incluye notas para describir los cambios que se deben realizar para utilizar otras conexiones.

Código de applet de ejemplo

El fragmento de código siguiente muestra una applet que utiliza una conexión TCP/IP para:

1. Conectarse a un gestor de colas
2. Transferir un mensaje a SYSTEM.DEFAULT.LOCAL.QUEUE
3. Obtener de nuevo el mensaje

```
// ============================================================================
//
// Materiales bajo licencia - Propiedad de IBM
//
// 5639-C34
//
// (c) Copyright IBM Corp. 1995, 1999
//
// ============================================================================
// Applet de ejemplo de MQSeries Cliente para Java
//
// Este ejemplo se ejecuta como una applet con el visor de applets y el archivo HTML,
// utilizando el mandato :-
//            appletviewer MQSample.html
// La salida es en la línea de mandatos, NO en la ventana del visor de applets.
//
// Nota. Si recibe el error 2 razón 2059 de MQSeries y está
seguro de que la
// configuración de MQSeries y TCP/IP es correcta,
// deberá pulsar en la selección "Applet" de la ventana del visor de applets,
// seleccionar propiedades y cambiar el "Network access" a unrestricted.
import com.ibm.mq.*;            // Incluir el paquete de MQSeries Classes for Java

public class MQSample extends java.applet.Applet
{

  private String hostname = "your_hostname";      // definir el nombre del
                                                  // sist. princ. al que se debe conectar.
  private String channel  = "server_channel";     // definir el nombre de canal
                                                  // que debe utilizar el cliente.
                                                  // Nota. Se supone que el servidor MQSeries
                                                  // está a la escucha en el puerto
                                                  // TCP/IP por omisión 1414.
  private String qManager = "your_Q_manager";     // definir el nombre
                                                  // de gestor de colas al que
                                                  // se debe conectar.

  private MQQueueManager qMgr;                    // definir un objeto de gestor de colas

  // Cuando se invoca a esta clase, se realiza primero esta inicialización.

  public void init()
  {
     // Configurar el entorno de MQSeries
     MQEnvironment.hostname = hostname;           // Se podría poner la
                                                  // serie de canal & de
     MQEnvironment.channel  = channel;            // sistema principal directamente aquí.

     MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY,//Establecer conexión de servidor o
                                  MQC.TRANSPORT_MQSERIES);//TCP/IP

  } // fin de init
```

Figura 1. Applet de ejemplo de MQSeries Classes for Java (Pieza 1 de 3)

Código de ejemplo

```
    public void start()
  {

    try {
      // Crear una conexión con el gestor de colas
      qMgr = new MQQueueManager(qManager);

      // Configurar las opciones en la cola que se va a abrir...
      // Nota. Todas las opciones de MQSeries tienen el prefijo MQC en Java.
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;
      // Ahora especificar la cola que se va a abrir y las opciones de apertura...

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

      // Definir un mensaje MQSeries simple y escribir texto en formato UTF...

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");

      // especificar las opciones de mensaje...

      MQPutMessageOptions pmo = new MQPutMessageOptions();  // aceptar los valores por omisión,
                                                            // igual que
                                                            // constante
 // MQPMO_DEFAULT
      // transferir el mensaje a la cola

      system_default_local_queue.put(hello_world,pmo);

      // obtener de nuevo el mensaje...
      // Definir primero un almacenamiento intermedio de mensajes MQSeries para recibir el mensaje.

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

      // Establecer las opciones de obtención de mensajes.

      MQGetMessageOptions gmo = new MQGetMessageOptions();  // aceptar los valores por omisión
                                                            // igual que
                                                            //  MQGMO_DEFAULT
      // extraer el mensaje de la cola.

      system_default_local_queue.get(retrievedMessage, gmo);

      // Y probar que tenemos el mensaje visualizando el texto de mensaje UTF

      String msgText = retrievedMessage.readUTF();
      System.out.println("El mensaje es: " + msgText);

      // Cerrar la cola

      system_default_local_queue.close();

      // Desconectar del gestor de colas

      qMgr.disconnect();

    }

    // Si se ha producido un error más arriba, intentar identificar lo que ha ido mal.
   // ¿Era un error de MQSeries?
```

Figura 1. Applet de ejemplo de MQSeries Classes for Java (Pieza 2 de 3)

```
        catch (MQException ex)
    {
      System.out.println("Se ha producido un error de MQSeries : Código de terminación " +
                         ex.completionCode +
                         " Código de razón " + ex.reasonCode);
    }
    // ¿Era un error de espacio de almacenamiento intermedio de Java?
    catch (java.io.IOException ex)
    {
      System.out.println("Se ha producido un error al grabar en el
      almacenamiento intermedio de mensajes: " + ex);
    }

  } // fin de inicio

} // fin de ejemplo
```

Figura 1. Applet de ejemplo de MQSeries Classes for Java (Pieza 3 de 3)

Cambio de la conexión para utilizar VisiBroker para Java™

Cambie la línea:

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

por:

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

y añada las líneas siguientes para inicializar ORB (Object Request Broker):

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

También debe añadir la sentencia de importación siguiente al principio del archivo:

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

No es necesario especificar el número de puerto o canal si se utiliza VisiBroker.

Código de aplicación de ejemplo

El fragmento de código siguiente muestra una aplicación simple que utiliza la modalidad de enlaces para:

1. Conectarse a un gestor de colas
2. Transferir un mensaje a SYSTEM.DEFAULT.LOCAL.QUEUE
3. Obtener de nuevo el mensaje

```
// ============================================================================
// Materiales bajo licencia - Propiedad de IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 1999
// ============================================================================
// Aplicación de ejemplo de MQSeries classes for Java
//
// Este ejemplo se ejecuta en una aplicación Java utilizando el mandato :- java MQSample

import com.ibm.mq.*;            // Incluir el paquete de MQSeries classes for Java

public class MQSample
{
  private String qManager = "your_Q_manager";  // definir el nombre del gestor
                                               // de colas al que se debe conectar.
  private MQQueueManager qMgr;                 // definir un objeto de gestor
                                               // de colas
  public static void main(String args[]) {
     new MQSample();
  }

  public MQSample() {
   try {

      // Crear una conexión con el gestor de colas

      qMgr = new MQQueueManager(qManager);

      // Configurar las opciones en la cola que se va a abrir...
      // Nota. Todas las opciones de MQSeries tienen el prefijo MQC en Java.

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;

      // Ahora, especificar la cola que se va a abrir
      // y las opciones de apertura...

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

      // Definir un mensaje MQSeries simple y escribir texto en formato UTF...

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");

      // especificar las opciones de mensaje...

      MQPutMessageOptions pmo = new MQPutMessageOptions(); // aceptar los valores por omisión,
                                                           // iguales que MQPMO_DEFAULT
```

Figura 2. Aplicación de ejemplo de MQSeries Classes for Java (Pieza 1 de 2)

```
     // transferir el mensaje a la cola

      system_default_local_queue.put(hello_world,pmo);

      // obtener de nuevo el mensaje...
      // Definir primero un almacenamiento intermedio de mensajes MQSeries para recibir el mensaje...

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

      // Establecer las opciones de obtención de mensajes...

      MQGetMessageOptions gmo = new MQGetMessageOptions();  // aceptar los valores por omisión
                                                            // iguales que MQGMO_DEFAULT
      // extraer el mensaje de la cola...

      system_default_local_queue.get(retrievedMessage, gmo);

      // Y probar que tenemos el mensaje visualizando el texto de mensaje UTF

      String msgText = retrievedMessage.readUTF();
      System.out.println("El mensaje es: " + msgText);
      // Cerrar la cola...
      system_default_local_queue.close();
      // Desconectar del gestor de colas

      qMgr.disconnect();
    }
   // Si se ha producido un error más arriba, intentar identificar lo que ha ido mal
      // ¿Era un error de MQSeries?
    catch (MQException ex)
    {
      System.out.println("Se ha producido un error de MQSeries : Código de terminación " +
                         ex.completionCode + " Código de razón " + ex.reasonCode);
    }
    // ¿Era un error de espacio de almacenamiento intermedio de Java?
    catch (java.io.IOException ex)
    {
      System.out.println("Se ha producido un error al grabar mensaje en almac. interm. de mensajes: " + ex);
    }
  }
} // fin de ejemplo
```

Figura 2. Aplicación de ejemplo de MQSeries Classes for Java (Pieza 2 de 2)

Operaciones en gestores de colas

En este apartado se describe cómo conectar y desconectar de un gestor de colas utilizando MQSeries Classes for Java.

Configuración del entorno MQSeries

Nota: Este paso no es necesario cuando se utiliza MQSeries Classes for Java en la modalidad de enlaces. En este caso, puede ir directamente al apartado "Conexión a un gestor de colas". Antes de utilizar la conexión de cliente para conectarse a un gestor de colas, debe acordarse de configurar MQEnvironment.

Los clientes MQSeries basados en ″C″ dependen de las variables de entorno para controlar la presentación de la invocación MQCONN. Puesto que las applets Java™ no tienen acceso a las variables de entorno, la interfaz de programación Java™ incluye una clase MQEnvironment, que le permite especificar los detalles siguientes que se van a utilizar durante el intento de conexión:

- Nombre de canal
- Nombre del sistema principal
- Número de puerto
- ID de usuario
- Contraseña

Para especificar el nombre de canal y el nombre del sistema principal, utilice el código siguiente:

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

Equivale a un valor de variable de entorno MQSERVER de:

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

Por omisión, los clientes Java™ intentan conectarse a un escucha MQSeries en el puerto 1414. Para especificar otro puerto, utilice el código:

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

El ID de usuario y la contraseña toman por omisión espacios en blanco. Para especificar una contraseña o un ID de usuario diferentes a un espacio en blanco, utilice el código:

```
MQEnvironment.userID   = "uid";  // equivalente a var ent MQ_USER_ID
MQEnvironment.password = "pwd";  // equivalente a var ent MQ_PASSWORD
```

Nota: Si está configurando una conexión utilizando VisiBroker para Java™, consulte el apartado "Cambio de la conexión para utilizar VisiBroker para Java™" en la página 61.

Conexión a un gestor de colas

Ahora está preparado para conectarse a un gestor de colas creando una nueva instancia de la clase MQQueueManager:

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

Para desconectarse de un gestor de colas, llame al método disconnect() en el gestor de colas:

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

Si invoca al método de desconexión, se cierran todas las colas y los procesos abiertos a los que ha accedido mediante dicho gestor de colas. Sin embargo, una práctica de programación recomendada consiste en cerrar los recursos explícitamente al terminar de utilizarlos. Para hacerlo, utilice el método close().

En un gestor de colas, los métodos commit() y backout() sustituyen a las invocaciones MQCMIT y MQBACK que se utilizan con la interfaz de procedimientos.

Acceso a colas y procesos

Para acceder a colas y procesos, debe utilizar la clase MQQueueManager. MQOD (estructura de descriptor de objeto) se ha contraído en los parámetros de estos métodos. Por ejemplo, para abrir una cola de un gestor de colas "queueManager", utilice el código siguiente:

```
MQQueue queue = queueManager.accessQueue("qName",
                            MQC.MQOO_OUTPUT,
                          "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

El parámetro *options* es igual que el parámetro Options de invocación MQOPEN.

El método accessQueue devuelve un nuevo objeto de la clase MQQueue.

Cuando haya terminado de utilizar la cola, utilice el método close() para cerrarla, tal como se muestra en el ejemplo siguiente:

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

Con MQSeries Classes for Java, también puede crear una cola utilizando el constructor MQQueue. Los parámetros son exactamente los mismos que para el método accessQueue, con la adición de un parámetro de gestor de colas. Por ejemplo:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                          "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

La creación de un objeto de cola utilizando este método le permite escribir sus propias subclases de MQQueue.

Para acceder a un proceso, utilice el método accessProcess en lugar de accessQueue. Este método no tiene un parámetro *dynamic queue name*, puesto que no se aplica a los procesos.

El método accessProcess devuelve un nuevo objeto de la clase MQProcess.

Cuando haya terminado de utilizar el objeto de proceso, utilice el método close() para cerrarlo, tal como se muestra en el ejemplo siguiente:

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

Con MQSeries Classes for Java, también puede crear un proceso utilizando el constructor MQProcess. Los parámetros son exactamente los mismos que para el

método accessProcess, con la adición de un parámetro de gestor de colas. La creación de un objeto de proceso utilizando este método le permite crear sus propias subclases de MQProcess.

Manejo de mensajes

Los mensajes se transfieren a las colas utilizando el método put() de la clase MQQueue y se obtienen de las colas utilizando el método get() de la clase MQQueue. A diferencia de la interfaz de procedimientos, donde MQPUT y MQGET transfieren y obtienen matrices de bytes, el lenguaje de programación Java™ transfiere y obtiene instancias de la clase MQMessage. La clase MQMessage encapsula el almacenamiento intermedio de datos que contiene los datos reales de los mensajes, junto con todos los parámetros MQMD (descriptor de mensajes) que describen dicho mensaje.

Para crear un nuevo mensaje, cree una nueva instancia de la clase MQMessage y utilice los métodos writeXXX para transferir datos al almacenamiento intermedio de mensajes.

Cuando se crea una nueva instancia de mensaje, todos los parámetros de MQMD se establecen automáticamente en sus valores por omisión, tal como se define en la publicación *MQSeries® Application Programming Reference*. El método put() de MQQueue también toma una instancia de la clase MQPutMessageOptions como parámetro. Esta clase representa la estructura MQPMO. En el ejemplo siguiente se crea un mensaje y se coloca en una cola:

```
// Crear un nuevo mensaje que contenga mi edad seguida de mi nombre
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

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

// Utilizar las opciones de transferir mensaje por omisión...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// ¡transferir el mensaje!
queue.put(myMessage,pmo);
```

El método get() de MQQueue devuelve una instancia nueva de MQMessage, que representa el mensaje recién tomado de la cola. También toma una instancia de la clase MQGetMessageOptions como parámetro. Esta clase representa la estructura MQGMO.

No es necesario especificar un tamaño máximo de mensaje, puesto que el método get() ajusta automáticamente el tamaño de su almacenamiento intermedio interno para adaptarlo al mensaje entrante. Utilice los métodos readXXX de la clase MQMessage para acceder a los datos del mensaje devuelto.

En el ejemplo siguiente se muestra cómo obtener un mensaje de una cola:

```
// Obtener un mensaje de la cola
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // tiene valores por omisión

// Extraer los datos del mensaje
int age = theMessage.readInt();
```

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

Puede modificar el formato numérico que utilizan los métodos de lectura y escritura estableciendo la variable de miembro *encoding*.

Puede modificar el juego de caracteres a utilizar para leer y escribir series de caracteres estableciendo la variable de miembro *characterSet*.

Consulte el apartado "MQMessage" en la página 112 para obtener información más detallada.

Nota: El método writeUTF() de MQMessage codifica automáticamente la longitud de la serie así como los bytes Unicode que contiene. Cuando otro programa Java™ va a leer el mensaje (utilizando readUTF()), este procedimiento es el más sencillo para enviar información sobre la serie de caracteres.

Manejo de errores

Los métodos de la interfaz Java™ no devuelven ningún código de terminación ni código de razón. En lugar de ello, emiten una excepción siempre que el código de terminación y el código de razón resultantes de una invocación MQSeries no sean ambos cero. De este modo se simplifica la lógica del programa y el usuario no tiene que comprobar los códigos de retorno después de cada invocación a MQSeries. Puede decidir en qué puntos del programa desea tratar la posibilidad de anomalía. En los puntos indicados, puede rodear el código con bloques 'try' y 'catch', tal como se muestra en el ejemplo siguiente:

```
try {
myQueue.put(messageA,putMessageOptionsA);
myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
// Este bloque de código sólo se ejecuta si uno de
// los dos métodos de transferencia ha emitido un
// código de terminación o código de razón distinto de cero.
System.out.println("Se ha producido un error durante la operación de transferir:" +
                           "CT = " + ex.completionCode +
                           "CR = " + ex.reasonCode);
}
```

Obtención y establecimiento de valores de atributo

Para muchos de los atributos comunes, las clases MQManagedObject, MQQueue, MQProcess y MQQueueManager contienen métodos getXXX() y setXXX(). Estos métodos permiten obtener y establecer sus valores de atributo. Tenga en cuenta que para MQQueue, los métodos sólo funcionan si se especifican los distintivos 'inquire' y 'set' apropiados al abrir la cola.

Para atributos menos comunes, las clases MQQueueManager, MQQueue y MQProcess herendan de una clase denominada MQManagedObject. Esta clase define las interfaces inquire() y set().

Cuando se crea un nuevo objeto de gestor de colas utilizando el operador *new*, éste se abre automáticamente para 'inquiry'. Cuando se utiliza el método accessProcess() para acceder a un objeto de proceso, dicho objeto se abre automáticamente para 'inquiry'. Cuando se utiliza el método accessQueue() para

acceder a un objeto de cola, dicho objeto *no* se abre de modo automático para las operaciones 'inquire' o 'set', puesto que la adición de estas opciones puede causar problemas automáticamente con algunos tipos de colas remotas. Para utilizar los métodos inquire, set, getXXX y setXXX en una cola, debe especificar los distintivos 'inquire' y 'set' adecuados en el parámetro openOptions del método accessQueue().

Los métodos inquire y set toman tres parámetros:
- matriz selectors
- matriz intAttrs
- matriz charAttrs

No necesita los parámetros SelectorCount, IntAttrCount y CharAttrLength que se encuentran en MQINQ, puesto que en Java™ siempre se conoce la longitud de una matriz. En el ejemplo siguiente se muestra cómo efectuar una consulta sobre una cola:

```
// consultar sobre una cola
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Prioridad por omisión = " + intAttrs[0]);
System.out.println("Descripción : " + new String(charAttrs,0));
```

Programas de múltiples hebras

Es difícil evitar los programas de múltiples hebras en Java™. Examine un programa simple que se conecta a un gestor de colas y abre una cola en el arranque. El programa visualiza un solo botón en la pantalla y, cuando el usuario lo pulsa, el programa busca un mensaje en la cola y lo carga.

De modo inherente, el entorno de ejecución de Java™ es de múltiples hebras. Por este motivo, la inicialización de la aplicación tiene lugar en una hebra y el código que se ejecuta en respuesta a la pulsación del botón se ejecuta en otra hebra independiente (la hebra de interfaz de usuario).

Con el cliente MQSeries basado en ″C″ puede causar un problema, puesto que los manejadores no se pueden compartir entre múltiples hebras. MQSeries Classes for Java suaviza la restricción, al permitir que se pueda compartir un objeto del gestor de colas (y sus objetos de proceso y colas asociadas) entre múltiples hebras.

La implementación de MQSeries Classes for Java asegura que, para una conexión determinada (instancia de objeto MQQueueManager), se sincronicen todos los accesos al gestor de colas MQSeries de destino. Por consiguiente, una hebra que desee emitir una invocación a un gestor de colas queda bloqueada hasta que finalizan todas las demás invocaciones en curso para dicha conexión. Si necesita acceso simultáneo al mismo gestor de colas desde dentro del programa, cree un nuevo objeto MQQueueManager para cada hebra que necesite acceso concurrente. (Equivale a emitir una invocación MQCONN independiente para cada hebra).

Nota: En el entorno CICS® Transaction Server for OS/390®, sólo la primera hebra (principal) puede emitir invocaciones MQSeries o CICS®. Por este motivo, no se pueden compartir objetos MQQueueManager o MQQueue entre hebras en este entorno, ni se puede crear un nuevo MQQueueManager en una hebra dependiente.

Creación de rutinas de salida de usuario

MQSeries Classes for Java le permite facilitar sus propias rutinas de salida de seguridad, recepción o emisión.

Para implementar una rutina de salida, defina una nueva clase Java™ que implemente la interfaz adecuada. En el paquete de MQSeries hay tres interfaces de rutina de salida definidas:
- MQSendExit
- MQReceiveExit
- MQSecurityExit

En el ejemplo siguiente se define una clase que implementa las tres:

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

  // Este método proviene de la rutina de salida de emisión
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // rellene el cuerpo de la rutina de salida de emisión aquí
  }


  // Este método proviene de la rutina de salida de recepción
  public byte[] receiveExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // rellene el cuerpo de la rutina de salida de recepción aquí
  }


  // Este método proviene de la rutina de salida de seguridad
  public byte[] securityExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // rellene el cuerpo de la rutina de salida de seguridad aquí
  }

}
```

A cada rutina de salida se le pasan dos instancias de objeto: MQChannelExit y MQChannelDefinition. Estos objetos representan las estructuras MQCXP y MQCD definidas en la interfaz de procedimientos.

Para una rutina de salida de emisión, el parámetro *agentBuffer* contiene los datos que se van a enviar. Para una rutina de salida de recepción, el parámetro *agentBuffer* contiene los datos que se acaban de recibir. No es necesario ningún parámetro de longitud, puesto que la expresión agentBuffer.length indica la longitud de la matriz.

Para las rutinas de salida de emisión y de seguridad, el código de salida debe devolver la matriz de bytes que se desea enviar al servidor. Para una rutina de salida de recepción, el código de salida debe devolver los datos modificados que se desea que interprete MQSeries Classes for Java.

El cuerpo de rutina de salida más simple posible es:

```
{
  return agentBuffer;
}
```

Si el programa se va a ejecutar como una applet Java™ bajada, las restricciones de seguridad que se aplican significan que no se puede leer o escribir ningún archivo local. Si la rutina de salida necesita un archivo de configuración, puede colocar el archivo en la web y utilizar la clase java.net.URL para bajarlo y examinar su contenido.

Agrupación de conexiones

MQSeries Classes for Java Versión 5.2 proporciona soporte adicional para aplicaciones que tratan con varias conexiones a gestores de colas MQSeries. Cuando ya no se necesita una conexión, en lugar de destruirla, se puede agrupar y volver a utilizar más tarde. De este modo, se ofrece una mejora considerable del rendimiento para las aplicaciones y el middleware que se conectan en serie a gestores de cola arbitrarios.

MQSeries proporciona una agrupación de conexiones por omisión. Las aplicaciones pueden activar o desactivar la agrupación de conexiones registrando o anulando el registro de señales a través de la clase MQEnvironment. Si la agrupación está activa, cuando Java base de MQ construye un objeto MQQueueManager busca la agrupación por omisión y reutiliza cualquier conexión adecuada. Cuando se produce una invocación MQQueueManager.disconnect(), se devuelve la conexión subyacente a la agrupación.

De forma alternativa, las aplicaciones pueden construir una agrupación de conexiones MQSimpleConnectionManager para una finalidad específica. Después, la aplicación puede especificarla durante la construcción de un objeto MQQueueManager o pasarla a MQEnvironment para utilizarla como agrupación de conexiones por omisión.

Además, Java base de MQ proporciona una implementación parcial de Java™ 2 Platform Enterprise Edition (J2EE) Connector Architecture. Las aplicaciones que se ejecutan en Java™ 2 v1.3 JVM con JAAS 1.0 (Java Authentication and Authorization Service) pueden proporcionar su propia agrupación de conexiones al implementar la interfaz **javax.resource.spi.ConnectionManager**. En este caso, también se puede especificar la interfaz en el constructor MQQueueManager o especificar como agrupación de conexiones por omisión.

Control de la agrupación de conexiones por omisión

En el caso de la aplicación de ejemplo siguiente, MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

Agrupación de conexiones

MQApp1 toma una lista de gestores de cola locales de la línea de mandatos, se conecta a cada uno, uno después de otro, y realiza alguna operación. Sin embargo, cuando la línea de mandatos enumera el mismo gestor de colas varias veces, resulta más eficaz conectarse sólo una vez y reutilizar dicha conexión varias veces.

Java base de MQ proporciona una agrupación de conexiones por omisión que se puede utilizar para esta finalidad. Para habilitar la agrupación, utilice uno de los métodos MQEnvironment.addConnectionPoolToken(), y para inhabilitarla, utilice MQEnvironment.removeConnectionPoolToken().

Funcionalmente, la aplicación de ejemplo siguiente, MQApp2, es idéntica a MQApp1, pero sólo se conecta una vez a cada gestor de colas.

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

La primera línea en negrita activa la agrupación de conexiones por omisión, al registrar un objeto MQPoolToken con MQEnvironment.

El constructor MQQueueManager ahora busca en la agrupación una conexión adecuada y sólo crea una conexión para el gestor de colas si no encuentra ninguna existente. La invocación de qmgr.disconnect() devuelve la conexión a la agrupación para volver a utilizarla más tarde. Las invocaciones de API son las mismas que las de la aplicación de ejemplo MQApp1.

La segunda línea resaltada desactiva la agrupación de conexiones por omisión, lo cual destruye todas las conexiones del gestor de colas almacenadas en la agrupación. Es importante ya que, de lo contrario, la aplicación puede terminar con algunas conexiones de gestor de colas activas en la agrupación. Esta situación podrá producir errores que aparecerían en las anotaciones del gestor de colas.

La agrupación de conexiones por omisión almacena diez conexiones no utilizadas como máximo y mantiene activas las conexiones no utilizadas durante un máximo de cinco minutos. La aplicación puede modificar esto (para obtener información más detallada, consulte el apartado “Suministro de otra agrupación de conexiones” en la página 75).

En vez de utilizar MQEnvironment para suministrar un MQPoolToken, la aplicación puede construir el suyo propio:

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

Algunos proveedores de aplicaciones o middleware pueden facilitar subclases de MQPoolToken para pasar información a una agrupación de conexiones personalizada. De este modo, se pueden construir y pasar a addConnectionPoolToken(), lo que permite que se pueda pasar información adicional a la agrupación de conexiones.

Agrupación de conexiones por omisión y varios componentes

MQEnvironment mantiene un conjunto de objetos MQPoolToken registrados. Para añadir o eliminar MQPoolTokens del conjunto, utilice los métodos siguientes:

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

Una aplicación puede constar de varios componentes que existan de modo independiente y trabajen utilizando un gestor de colas. En este tipo de aplicación, cada componente debe añadir un MQPoolToken al MQEnvironment establecido para toda su duración.

Por ejemplo, la aplicación de ejemplo MQApp3 crea diez hebras e inicia cada una de ellas. Cada hebra registra su MQPoolToken específico, espera un tiempo determinado y, a continuación, se conecta al gestor de colas. Después la hebra se desconecta y elimina su MQPoolToken.

La agrupación de conexiones por omisión permanece activa mientras hay, como mínimo, una señal en el conjunto de MQPoolTokens, por lo que permanece activa mientras dura la aplicación. La aplicación no necesita mantener un objeto maestro para el control global de las hebras.

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

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

Suministro de otra agrupación de conexiones

En este apartado se describe cómo utilizar la clase **com.ibm.mq.MQSimpleConnectionManager** para suministrar otra agrupación de conexiones. Esta clase proporciona los recursos básicos para la agrupación de conexiones, y las aplicaciones la pueden utilizar para personalizar la presentación de la agrupación.

Después de convertirlo en instancia, un MQSimpleConnectionManager se puede especificar en el constructor MQQueueManager. A continuación, MQSimpleConnectionManager gestiona la conexión que subyace al MQQueueManager construido. Si MQSimpleConnectionManager contiene una conexión agrupada adecuada, se vuelve a utilizar la conexión y se devuelve a MQSimpleConnectionManager después de una invocación a MQQueueManager.disconnect().

En el fragmento de código siguiente se muestra esta presentación:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

La conexión que se crea durante el primer constructor MQQueueManager se almacena en myConnMan después de la invocación a qmgr.disconnect(). La conexión se vuelve a utilizar después durante la segunda invocación al constructor MQQueueManager.

La segunda línea habilita MQSimpleConnectionManager. La última línea inhabilita MQSimpleConnectionManager, destruyendo todas las conexiones mantenidas en la agrupación. MQSimpleConnectionManager está, por omisión, en MODE_AUTO, que se describe más adelante en este apartado.

MQSimpleConnectionManager asigna conexiones según la utilización más reciente y destruye las conexiones conforme a la utilización menos reciente. Por omisión, una conexión se destruye si no se ha utilizado durante cinco minutos, o si hay más de diez conexiones no utilizadas en la agrupación. Puede modificar estos valores utilizando:

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

También puede establecer un MQSimpleConnectionManager para utilizarlo como agrupación de conexiones por omisión, cuando no se proporciona ningún gestor de conexiones en el constructor MQQueueManager.

Se muestra en la aplicación siguiente:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String[] args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setHighThreshold(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

Las líneas en negrita establecen un MQSimpleConnectionManager. Se establece para:

- destruir conexiones que no se han utilizado durante una hora
- limitar a 50 las conexiones no utilizadas que se mantienen en la agrupación
- MODE_AUTO (en realidad, el valor por omisión). Significa que la agrupación sólo está activa si es el gestor de conexiones por omisión y el conjunto de MQPoolTokens que mantiene MQEnvironment tiene, como mínimo, una señal.

Así, el nuevo MQSimpleConnectionManager se establece como gestor de conexiones por omisión.

En la última línea, la aplicación invoca a MQApp3.main(), que ejecuta varias hebras, y cada una de ellas utiliza MQSeries de modo independiente. Estas hebras ahora utilizan myConnMan cuando crean conexiones.

Suministro del ConnectionManager personalizado

En Java™ 2 v1.3, con JAAS 1.0 instalado, los proveedores de aplicaciones y middleware pueden proporcionar implementaciones de agrupaciones de conexiones alternativas. Java base de MQ ofrece una implementación parcial de J2EE Connector Architecture. Las implementaciones de **javax.resource.spi.ConnectionManager** se pueden utilizar como gestor de conexiones por omisión, o se pueden especificar en el constructor MQQueueManager.

Java base de MQ cumple con el contrato Connection Management (gestión de conexiones) de J2EE Connector Architecture. Lea este apartado junto con el contrato Connection Management de J2EE Connector Architecture (consulte el sitio web de Sun, en la dirección siguiente: `http://java.sun.com`).

La interfaz ConnectionManager sólo define un método:

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

El constructor MQQueueManager invoca a allocateConnection en el ConnectionManager adecuado. Pasa las implementaciones adecuadas de ManagedConnectionFactory y ConnectionRequestInfo como parámetros para describir la conexión necesaria.

ConnectionManager busca su agrupación para un objeto javax.resource.spi.ManagedConnection que se ha creado con objetos

ManagedConnectionFactory y ConnectionRequestInfo idénticos. Si ConnectionManager encuentra algún objeto ManagedConnection adecuado, crea un java.util.Set que contiene las ManagedConnections candidatas. A continuación, ConnectionManager invoca lo siguiente:

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

La implementación de MQSeries de ManagedConnectionFactory ignora el parámetro subject. Este método selecciona y devuelve una ManagedConnection adecuada del conjunto o devuelve un valor nulo si no encuentra ninguna ManagedConnection apropiada. Si no hay ninguna ManagedConnection adecuada en la agrupación, ConnectionManager puede crear una utilizando:

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

En este caso, también se ignora el parámetro. Este método se conecta a un gestor de colas MQSeries y devuelve una implementación de javax.resource.spi.ManagedConnection que representa la conexión que se acaba de crear. Cuando ConnectionManager obtiene una ManagedConnection (de la agrupación o recién creada), crea un manejador de conexión utilizando:

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

Este manejador de conexión se puede devolver de allocateConnection().

ConnectionManager debe registrar un interés en ManagedConnection a través de:

```
mc.addConnectionEventListener()
```

Se notifica a ConnectionEventListener si se produce un error grave en la conexión o cuando se invoca a MQQueueManager.disconnect(). Cuando se invoca a MQQueueManager.disconnect(), ConnectionEventListener puede realizar cualquiera de las acciones siguientes:

- restablecer la ManagedConnection utilizando la invocación mc.cleanup() y, a continuación, devolver la ManagedConnection a la agrupación
- destruir la ManagedConnection utilizando la invocación mc.destroy()

Si ConnectionManager va a ser el ConnectionManager por omisión, también puede registrar un interés en el estado del conjunto gestionado MQEnvironment de MQPoolTokens. Para hacerlo, en primer lugar construya un objeto MQPoolServices y, a continuación, registre un objeto MQPoolServicesEventListener con el objeto MQPoolServices:

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

Se notifica al escucha cuando se añade o elimina un MQPoolToken del conjunto o cuando se produce algún cambio en el ConnectionManager por omisión. El objeto MQPoolServices también facilita un procedimiento para consultar el tamaño actual del conjunto de MQPoolTokens.

Compilación y prueba de programas Java base de MQ

Antes de compilar programas Java base de MQ, debe asegurarse de que el directorio de instalación de MQSeries Classes for Java se encuentra en la variable de entorno CLASSPATH, tal como se describe en el “Capítulo 2. Procedimientos de instalación” en la página 9.

Para compilar una clase ″MyClass.java″, utilice el mandato:

```
javac MyClass.java
```

Ejecución de applets Java base de MQ

Si crea una applet (subclase de java.applet.Applet), debe crear un archivo HTML que haga referencia a la clase para poderla ejecutar. Un archivo HTML de ejemplo puede tener el aspecto siguiente:

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

Ejecute la applet cargando el archivo HTML en un navegador de la web habilitado para Java™ o utilizando el visor de applets que se incluye con JDK (Java™ Development Kit).

Para utilizar el visor de applets, entre el mandato:

```
appletviewer myclass.html
```

Ejecución de aplicaciones Java base de MQ

Se crea una aplicación (una clase que contiene un método main()), utilizando la modalidad de enlaces o de cliente, ejecute el programa utilizando el intérprete Java™. Utilice el mandato:

```
java MyClass
```

Nota: Se omite la extensión '.class' en el nombre de clase.

Ejecución de aplicaciones Java base de MQ con CICS Transaction Server para OS/390

Para ejecutar una aplicación Java™ como una transacción con CICS®, debe:

1. Definir la aplicación y la transacción para CICS® utilizando la transacción CEDA que se proporciona.
2. Asegúrese de que el adaptador MQSeries CICS® está instalado en el sistema CICS®. (Consulte la publicación *MQSeries® for OS/390® System Management Guide* para obtener información detallada).
3. Asegúrese de que el entorno JVM especificado en el parámetro DHFJVM de JCL (Lenguaje de Control de Trabajos) de arranque de CICS® incluye las entradas CLASSPATH y LIBPATH adecuadas.
4. Inicie la transacción utilizando cualquiera de los procesos habituales.

Para obtener más información sobre cómo ejecutar transacciones CICS® Java™, consulte la documentación del sistema CICS®.

Rastreo de programas Java base de MQ

Java base de MQ incluye un recurso de rastreo, que puede utilizar para producir mensajes de diagnóstico si cree que puede haber algún problema con el código. (Normalmente, sólo necesita utilizar este recurso cuando se lo solicite el servicio técnico de IBM®).

Los métodos enableTracing y disableTracing de la clase MQEnvironment controlan el rastreo. Por ejemplo:

```
MQEnvironment.enableTracing(2);   // rastrear en el nivel 2
 ...                             // se rastrean estos mandatos
MQEnvironment.disableTracing();   // desactivar de nuevo el rastreo
```

El rastreo se escribe en la consola Java™ (`System.err`).

Si el programa es una aplicación o si lo ejecuta desde el disco local utilizando el mandato appletviewer, también puede redireccionar la salida de rastreo al archivo que desee. En el fragmento de código siguiente se muestra cómo efectuar la redirección a un archivo denominado `myapp.trc`:

```
import java.io.*;

try {
  FileOutputStream
  traceFile = new FileOutputStream("myapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
catch (IOException ex) {
  // no se ha podido abrir el archivo,
  // en su lugar se rastrea en System.err
  MQEnvironment.enableTracing(2);
}
```

Existen cinco niveles diferentes de rastreo:

1. Proporciona rastreo de entrada, salida y excepción
2. Proporciona información de parámetros además de la proporcionada en el nivel 1
3. Proporciona cabeceras y bloques de datos MQSeries transmitidos y recibidos además de lo que proporciona el nivel 2
4. Proporciona datos de mensajes de usuario transmitidos y recibidos además de lo que proporciona el nivel 3
5. Proporciona rastreo de métodos en la máquina virtual Java™ además de lo que proporciona el nivel 4

Para rastreo de métodos en la máquina virtual Java™ con el nivel de rastreo 5:

- Para una aplicación, ejecútelo emitiendo el mandato `java_g` (en lugar de `java`)
- Para una applet, ejecútelo emitiendo el mandato `appletviewer_g` (en lugar de `appletviewer`)

Notas:

1. No se ofrece soporte para `java_g` para aplicaciones HPJ (High Performance Java™) en OS/390®.
2. No se ofrece soporte para `java_g` en OS/400®, pero se proporciona una función similar utilizando `OPTION(*VERBOSE)` en el mandato RUNJVA.

Rastreo de programas Java base de MQ

Capítulo 8. Presentación dependiente del entorno

En este capítulo se describe la presentación de las clases Java™ en los distintos entornos en los que se pueden utilizar. Las clases de MQSeries Classes for Java le permiten crear aplicaciones que se pueden utilizar en los entornos siguientes:

1. MQSeries Cliente para Java conectado a un servidor MQSeries V2.x en plataformas UNIX o Windows
2. MQSeries Cliente para Java conectado a un servidor MQSeries V5 en plataformas UNIX o Windows
3. MQSeries Enlaces para Java en ejecución en un servidor MQSeries V5 en plataformas UNIX o Windows
4. MQSeries Enlaces para Java en ejecución en un servidor MQSeries para MVS/ESA™
5. MQSeries Enlaces para Java en ejecución en un servidor MQSeries para MVS/ESA™ con CICS® Transaction Server para OS/390® Versión 1.3

En todos los casos, el código de MQSeries Classes for Java utiliza servicios que proporciona el servidor MQSeries subyacente. Existen diferencias en el nivel de función (por ejemplo, MQSeries V5 proporciona un superconjunto de la función de la V2). También hay diferencias en la presentación de algunas opciones e invocaciones de API. La mayor parte de las diferencias de presentación no son muy importantes, y muchas de ellas se producen entre los servidores OS/390® (MQSeries para MVS/ESA) y los servidores de otras plataformas.

MQSeries Classes for Java proporciona un 'núcleo' de clases, que facilita la presentación y el funcionamiento coherente en todos los entornos. También proporciona 'extensiones V5', que se han diseñado para utilizarlas sólo en entornos 2 y 3. En los apartados siguientes se describe el núcleo y las extensiones.

Detalles del núcleo

MQSeries Classes for Java contiene el núcleo de clases siguiente, que se puede utilizar en todos los entornos con las únicas variaciones mínimas que se enumeran en el apartado "Limitaciones y variaciones para las clases del núcleo" en la página 82.

- MQEnvironment
- MQException
- MQGetMessageOptions

  Excepto:
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  Excepto:
  - inquire()
  - set()
- MQMessage

  Excepto:
  - groupId
  - messageFlags

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

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

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

Notas:

1. Algunas constantes no se incluyen en el núcleo (consulte el apartado "Limitaciones y variaciones para las clases del núcleo" para obtener información más detallada) y no se deberán utilizar en programas completamente portátiles.
2. Algunas plataformas no ofrecen soporte para todas las modalidades de conexión. En dichas plataformas sólo se pueden utilizar las clases del núcleo y las opciones relacionadas con las modalidades para las que se ofrece soporte. (Consulte el apartado Tabla 1 en la página 5).

Limitaciones y variaciones para las clases del núcleo

Aunque, generalmente, las clases del núcleo se presentan de modo coherente en todos los entornos, existen algunas limitaciones y variaciones que se documentan en la Tabla 12.

Aparte de estas variaciones documentadas, las clases del núcleo tienen una presentación coherente en todos los entornos, incluso si las clases MQSeries equivalentes tienen normalmente diferencias de entorno. En general, la presentación será la misma en los entornos 2 y 3.

Tabla 12. Limitaciones y variaciones de las clases del núcleo

Clase o elemento	Limitaciones y variaciones
MQGMO_LOCK MQGMO_UNLOCK MQGMO_BROWSE_MSG_UNDER_CURSOR	Producen MQRC_OPTIONS_ERROR cuando se utilizan en los entornos 4 ó 5.
MQPMO_NEW_MSG_ID MQPMO_NEW_CORREL_ID MQPMO_LOGICAL_ORDER	Emiten errores excepto en los entornos 2 y 3. (Consulte las extensiones de la V5).
MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MESSAGE MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE	Emiten errores excepto en los entornos 2 y 3. (Consulte las extensiones de la V5).

Tabla 12. Limitaciones y variaciones de las clases del núcleo (continuación)

Clase o elemento	Limitaciones y variaciones
MQGMO_SYNCPOINT_IF_PERSISTENT	Emite errores en el entorno 1. (Consulte las extensiones de la V5).
MQGMO_MARK_SKIP_BACKOUT	Produce MQRC_OPTIONS_ERROR excepto en entornos 4 y 5.
MQCNO_FASTPATH_BINDING	Sólo se ofrece soporte en el entorno 3. (Consulte las extensiones de la V5).
Campos MQPMRF_*	Sólo se ofrece soporte en entornos 2 y 3.
Transferencia de un mensaje con MQQueue.priority > MaxPriority	Se rechazan con MQCC_FAILED y MQRC_PRIORITY_ERROR en entornos 4 y 5. Otros entornos los aceptan con los avisos MQCC_WARNING yMQRC_PRIORITY_EXCEEDS_MAXIMUM y tratan el mensaje como si hubiera establecido con MaxPriority.
BackoutCount	Los entornos 4 y 5 devuelven una cuenta de restituciones máxima de 255, incluso si el mensaje se ha restituido más de 255 veces.
Nombre de cola dinámica por omisión	CSQ.* para entornos 4 y 5. AMQ.* para otros sistemas.
Opciones de MQMessage.report: MQRO_EXCEPTION_WITH_FULL_DATA MQRO_EXPIRATION_WITH_FULL_DATA MQRO_COA_WITH_FULL_DATA MQRO_COD_WITH_FULL_DATA MQRO_DISCARD_MSG	No se ofrece soporte si un gestor de colas OS/390® genera un mensaje de informe, aunque se pueden establecer en todos los entornos. Este tema afecta a todos los entornos de Java™, porque el gestor de colas de OS/390® podría estar alejado de la aplicación Java™. Evite depender de cualquiera de estas opciones si existe una posibilidad de que pueda estar implicado un gestor de colas OS/390®.
MQQueueManager.commit() y MQQueueManager.backout()	En el entorno 5, estos métodos devuelven MQRC_ENVIRONMENT_ERROR. En este entorno, las aplicaciones deben utilizar los métodos de sincronización de tareas de JCICS: com.ibm.cics.server.Task.commit() y com.ibm.cics.server.Task.rollback().
Constructor MQQueueManager	En los entornos 4 y 5, si las opciones presentes en MQEnvironment (y el argumento de propiedades opcionales) implican una conexión de cliente, el constructor no se ejecuta correctamente y produce MQRC_ENVIRONMENT_ERROR. En los entornos 4 y 5, el constructor también puede devolver MQRC_CHAR_CONVERSION_ERROR. Asegúrese de que esté instalado el componente National Language Resources de OS/390® Language Environment®. En especial, asegúrese de que estén disponibles todas las conversiones entre las páginas de códigos IBM-1047 e ISO8859-1. En los entornos 4 y 5, el constructor también puede devolver MQRC_UCS2_CONVERSION_ERROR. Las clases MQSeries para Java™ intentan convertir de Unicode a la página de códigos del gestor de colas, y toman como valor por omisión IBM-500 si una página de códigos específica no está disponible. Asegúrese de que dispone de las tablas de conversión adecuadas para Unicode, que se deben instalar como parte de la característica opcional OS/390® C/C++, y también debe asegurarse de que Language Environment® puede localizar las tablas. Consulte la publicación *OS/390® C/C++ Programming Guide*, SC09-2362, para obtener más información sobre la habilitación de conversiones UCS-2.

Extensiones de la versión 5 que operan en otros entornos

MQSeries Classes for Java contiene las funciones siguientes, diseñadas específicamente para utilizar las extensiones de API incorporadas en MQSeries V5. Estas funciones sólo operan como se han diseñado en los entornos 2 y 3. En este tema se describe la presentación que pueden tener en otros entornos.

Opciones del constructor MQQueueManager

El constructor MQQueueManager incluye un argumento de entero opcional. Éste se correlaciona en el campo MQCNO.options de MQI y se utiliza para conmutar entre la conexión normal y la de vía de acceso rápida. Esta forma ampliada del constructor se acepta en todos los entornos, a condición de que las únicas opciones utilizadas sean MQCNO_STANDARD_BINDING o MQCNO_FASTPATH_BINDING. Cualquier otra opción hace que el constructor no se ejecute correctamente y produzca MQRC_OPTIONS_ERROR. La opción de vía de acceso rápida MQC.MQCNO_FASTPATH_BINDING sólo se acepta cuando se utiliza en los enlaces MQSeries V5 (entorno 3). Si se utiliza en cualquier otro entorno, se ignora.

Método MQQueueManager.begin()

Sólo se puede utilizar en el entorno 3. En cualquier otro entorno, no se ejecuta correctamente y produce MQRC_ENVIRONMENT_ERROR. MQSeries para AS/400® no ofrece soporte para la utilización del método `begin()` para iniciar unidades de trabajo globales que coordina el gestor de colas.

Opciones de MQPutMessageOptions

Los distintivos siguientes se pueden establecer en los campos de opciones de MQPutMessageOptions en cualquier entorno. Sin embargo, si se utilizan con un MQQueue.put() posterior en cualquier entorno distinto del 2 o el 3, put() no se ejecuta correctamente y produce MQRC_OPTIONS_ERROR.

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

Opciones de MQGetMessageOptions

Los distintivos siguientes se pueden establecer en los campos de opciones de MQGetMessageOptions en cualquier entorno. Sin embargo, si se utilizan con un MQQueue.get() posterior en cualquier entorno distinto del 2 o el 3, get() no se ejecuta correctamente y produce MQRC_OPTIONS_ERROR.

- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

El distintivo siguiente se puede establecer en los campos de opciones de MQGetMessageOptions en cualquier entorno. Sin embargo, si se utiliza con un MQQueue.get() posterior en un entorno 1, get() no se ejecuta correctamente y produce MQRC_OPTIONS_ERROR.

- MQGMO_SYNCPOINT_IF_PERSISTENT

Campos de MQGetMessageOptions

Se pueden establecer valores en los campos siguientes, sin tener en cuenta el entorno. Sin embargo, si la MQGetMessageOptions utilizada en un MQQueue.get() posterior contiene valores distintos a aquellos por omisión al ejecutarse en cualquier entorno distinto del 2 o el 3, get() no se ejecuta correctamente y produce MQRC_GMO_ERROR. Esto significa que en entornos distintos del 2 o el 3, estos campos se establecen siempre en sus valores iniciales después de cada get() satisfactorio.

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

Listas de distribución

Se utilizan las clases siguientes para crear Listas de distribución:

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

Puede crear y rellenar MQDistributionList y MQDistributionListItems en cualquier entorno, pero sólo puede crear y abrir MQDistributionList correctamente en los entornos 2 y 3. Cualquier intento de crear y abrir uno de ellos en otro entorno se rechaza con MQRC_OD_ERROR.

Campos de MQPutMessageOptions

Se presentan cuatro campos de MQPMO como las variables de miembro siguientes de la clase MQPutMessageOptions:

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

Aunque, principalmente, están destinados a utilizarse con listas de distribución, el servidor MQSeries V5 también rellena los campos DestCount después de una MQPUT a una cola individual. Por ejemplo, si la cola se convierte en una cola local, knownDestCount se establece en 1 y los otros dos campos en 0. En los entornos 2 y 3, los valores que establece el servidor V5 se devuelven en la clase MQPutMessageOptions. En los demás entornos, los valores de retorno se simulan del modo siguiente:

- Si put() se ejecuta correctamente, unknownDestCount se establece en 1 y los demás se establecen en 0.
- Si put() no se ejecuta correctamente, invalidDestCount se establece en 1 y los demás se establecen en 0.

recordFields se utiliza con las listas de distribución. Se puede escribir un valor en recordFields en cualquier momento, independientemente del entorno. Sin embargo, se ignora si se utilizan las opciones de MQPutMessage en un MQQueue.put() posterior, en lugar de MQDistributionList.put().

Campos de MQMD

Los campos de MQMD siguientes están principalmente relacionados con la segmentación de mensajes:

- GroupId
- MsgSeqNumber
- Offset MsgFlags
- OriginalLength

V5, extensiones

Si una aplicación establece cualquiera de estos campos de MQMD en valores que no sean los valores por omisión y luego efectúa un put() o get() en un entorno distinto del 2 o el 3, put() o get() activan la excepción (MQRC_MD_ERROR). Un put() o get() satisfactorio en un entorno que no sea el 2 o el 3, siempre deja los nuevos campos de MQMD establecidos en sus valores por omisión. Normalmente no se debe enviar un mensaje agrupado o segmentado a una aplicación Java™ que se ejecute para un gestor de colas que no sea MQSeries Versión 5 o superior. Si una aplicación de este tipo emite un get y el mensaje físico que se debe recuperar forma parte de un mensaje agrupado o segmentado (tiene valores que no son los valores por omisión para los campos de MQMD), éste se recupera sin error. Sin embargo, los campos de MQMD de MQMessage no se actualizan. La propiedad de formato de MQMessage se establece en MQFMT_MD_EXTENSION, y los verdaderos datos del mensaje toman como prefijo una estructura MQMDE que contiene los valores para los nuevos campos.

Capítulo 9. Clases e interfaces Java base de MQ

En este capítulo se describen todas las clases e interfaces MQSeries Classes for Java. Se incluyen detalles de las variables, los constructores y los métodos de cada clase e interfaz.

Se describen las clases siguientes:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

Se describen las interfaces siguientes:
- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData

MQChannelDefinition

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

clase pública **MQChannelDefinition**
expande **Object**

La clase MQChannelDefinition se utiliza para pasar información relacionada con la conexión al gestor de colas a las rutinas de salida de emisión, recepción y seguridad.

Nota: Esta clase no se aplica cuando se conecta directamente a MQSeries en la modalidad de enlaces.

Variables

channelName
`public String channelName`

Nombre del canal a través del cual se establece la conexión.

queueManagerName
`public String queueManagerName`

Nombre del gestor de colas con el que se efectúa la conexión.

maxMessageLength
`public int maxMessageLength`

Longitud máxima del mensaje que se puede enviar al gestor de colas.

securityUserData
`public String securityUserData`

Área de almacenamiento para que la utilice la rutina de salida de seguridad. La información que se coloca aquí se conserva durante las invocaciones de la rutina de salida de seguridad y también está disponible para las rutinas de salida de emisión y recepción.

sendUserData
`public String sendUserData`

Área de almacenamiento para que la utilice la rutina de salida de emisión. La información que se coloca aquí se conserva durante las invocaciones de la rutina de salida de emisión y también está disponible para las rutinas de salida de seguridad y recepción.

receiveUserData
`public String receiveUserData`

Área de almacenamiento para que la utilice la rutina de salida de recepción. La información que se coloca aquí se conserva durante las invocaciones de la rutina de salida de recepción y también está disponible para las rutinas de salida de emisión y seguridad.

connectionName
`public String connectionName`

Nombre de sistema principal TCP/IP de la máquina en la que reside el gestor de colas.

remoteUserId

```
public String remoteUserId
```

ID de usuario utilizado para establecer la conexión.

remotePassword

```
public String remotePassword
```

Contraseña utilizada para establecer la conexión.

Constructores

MQChannelDefinition

```
public MQChannelDefinition()
```

MQChannelExit

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

clase pública **MQChannelExit**
expande **Object**

Esta clase define información de contexto pasada a las rutinas de salida de emisión, recepción y seguridad cuando éstas se invocan. La rutina de salida debe establecer la variable de miembro exitResponse para indicar qué acción debe realizar MQSeries Cliente para Java a continuación.

Nota: Esta clase no es aplicable a MQSeries en la modalidad de enlaces para conexiones directas.

Variables

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

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

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

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

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

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

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

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

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

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

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

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

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

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

exitID `public int exitID`

Tipo de rutina de salida que se ha invocado. Para MQSecurityExit es siempre MQXT_CHANNEL_SEC_EXIT. Para MQSendExit es siempre MQXT_CHANNEL_SEND_EXIT y para MQReceiveExit es siempre MQXT_CHANNEL_RCV_EXIT.

exitReason

`public int exitReason`

Razón para invocar la rutina de salida. Los valores posibles son:

MQXR_INIT
Inicialización de rutina de salida; se invoca después de haber negociado las condiciones de conexión de canal pero antes de que se hayan enviado flujos de comunicaciones de seguridad.

MQXR_TERM
Terminación de rutina de salida; se invoca después de haber mandado los flujos de comunicaciones de desconexión, pero antes de que se destruya la conexión de socket.

MQXR_XMIT
Para una rutina de salida de emisión, indica que los datos deben transmitirse al gestor de colas.

Para una rutina de salida de recepción, indica que se han recibido datos del gestor de colas.

MQXR_SEC_MSG
Indica a la rutina de salida de seguridad que se ha recibido un mensaje de seguridad del gestor de colas.

MQXR_INIT_SEC
Indica que la rutina de salida debe iniciar el diálogo de seguridad con el gestor de colas.

exitResponse

`public int exitResponse`

Lo establece la rutina de salida para indicar la acción que debe realizar MQSeries Classes for Java a continuación. Los valores válidos son:

MQXCC_OK
Lo establece la rutina de salida de seguridad para indicar que se han completado los intercambios de seguridad.

Lo establece la rutina de salida de emisión para indicar que los datos devueltos deben transmitirse al gestor de colas.

Lo establece la rutina de salida de recepción para indicar que los datos devueltos están disponibles para que los procese MQSeries Cliente para Java.

MQXCC_SUPPRESS_FUNCTION
Lo establece la rutina de salida de seguridad para indicar que las comunicaciones con el gestor de colas deben concluirse.

MQXCC_SEND_AND_REQUEST_SEC_MSG
Lo establece la rutina de salida de seguridad para indicar que los datos devueltos deben transmitirse al gestor de colas y que se espera una respuesta del gestor de colas.

MQXCC_SEND_SEC_MSG
Lo establece la rutina de salida de seguridad para indicar que los datos devueltos deben transmitirse al gestor de colas y que no se espera ninguna respuesta.

MQXCC_SUPPRESS_EXIT
Lo establece cualquier rutina de salida para indicar que ya no se la deberá invocar más.

MQXCC_CLOSE_CHANNEL
Lo establece cualquier rutina de salida para indicar que debe cerrarse la conexión al gestor de colas.

maxSegmentLength
```
public int maxSegmentLength
```

Longitud máxima para cualquier transmisión a un gestor de colas.

Si la rutina de salida devuelve datos que se deben enviar al gestor de colas, la longitud de los datos devueltos no deberá exceder este valor.

exitUserArea
```
public byte exitUserArea[]
```

Área de almacenamiento disponible para que la utilice la rutina de salida.

MQSeries Cliente para Java conserva todos los datos situados en exitUserArea a través de las invocaciones de rutina de salida con el mismo exitID. (Es decir, las rutinas de salida de emisión, recepción y seguridad tienen cada una sus propias áreas de usuario independientes).

capabilityFlags
```
public static final int capabilityFlags
```

Indica la posibilidad del gestor de colas.

Sólo se soporta el distintivo MQC.MQCF_DIST_LISTS.

fapLevel
```
public static final int fapLevel
```

Nivel de FAP (Formato y Protocolo) negociado.

Constructores

MQChannelExit
```
public MQChannelExit()
```

MQDistributionList

```
java.lang.Object
   |
   └ com.ibm.mq.MQManagedObject          |
        └ com.ibm.mq.MQDistributionList
```

clase pública **MQDistributionList**
expande **MQManagedObject** (Consulte la página 109).

Nota: Sólo puede utilizar esta clase cuando esté conectado a un gestor de colas MQSeries Versión 5 (o superior).

MQDistributionList se crea utilizando el constructor MQDistributionList o utilizando el método accessDistributionList para MQQueueManager.

Una lista de distribución representa un conjunto de colas abiertas a las que se pueden enviar mensajes utilizando una sola invocación al método put(). (Consulte el apartado ″Distribution lists″ de la publicación *MQSeries® Application Programming Guide*).

Constructores

MQDistributionList

```
public MQDistributionList(MQQueueManager qMgr,
               MQDistributionListItem[] litems,
               int openOptions,
               String alternateUserId)
                          throws MQException
```

`qMgr` es el gestor de colas donde debe abrirse la lista.

`litems` son los elementos que se deben incluir en la lista de distribución.

En el apartado "accessDistributionList" en la página 159 se proporciona información detallada sobre los demás parámetros.

Métodos

put

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions)
                            throws MQException
```

Transfiere un mensaje a las colas de la lista de distribución.

Parámetros

message
Parámetro de entrada/salida que contiene la información del descriptor de mensaje y los datos del mensaje devuelto.

putMessageOptions
Opciones que controlan la acción de MQPUT. (Para obtener información detallada, consulte el apartado "MQPutMessageOptions" en la página 139).

Emite MQException si la transferencia no se ejecuta correctamente.

MQDistributionList

getFirstDistributionListItem

```
public MQDistributionListItem getFirstDistributionListItem()
```

Devuelve el primer elemento de la lista de distribución o *nulo* si la lista está vacía.

getValidDestinationCount

```
public int getValidDestinationCount()
```

Devuelve el número de elementos de la lista de distribución que se han abierto correctamente.

getInvalidDestinationCount

```
public int getInvalidDestinationCount()
```

Devuelve el número de elementos de la lista de distribución que no se han podido abrir correctamente.

MQDistributionListItem

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
          |
          +-- com.ibm.mq.MQDistributionListItem
```

clase pública **MQDistributionListItem**
expande **MQMessageTracker** (Consulte la página 131).

Nota: Sólo puede utilizar esta clase cuando esté conectado a un gestor de colas MQSeries Versión 5 (o superior).

MQDistributionListItem representa un solo elemento (cola) dentro de una lista de distribución.

Variables

completionCode
`public int completionCode`

Código de terminación resultante de la última operación efectuada en este elemento. Si esta operación ha sido la construcción de una MQDistributionList, el código de terminación está relacionado con la apertura de la cola. Si se trataba de una operación de transferir, el código de terminación está relacionado con el intento de transferencia de un mensaje a esta cola.

El valor inicial es ″0″.

queueName
`public String queueName`

Nombre de una cola que se va a utilizar con una lista de distribución. Éste no puede ser el nombre de una cola modelo.

El valor inicial es ″″.

queueManagerName
`public String queueManagerName`

Nombre del gestor de colas en el que se define la cola.

El valor inicial es ″″.

reasonCode
`public int reasonCode`

Código de razón resultante de la última operación efectuada en este elemento. Si esta operación ha sido la construcción de una MQDistributionList, el código de razón está relacionado con la apertura de la cola. Si ha sido una operación de transferir, el código de razón está relacionado con el intento de transferencia de un mensaje a esta cola.

El valor inicial es ″0″.

Constructores

MQDistributionListItem
`public MQDistributionListItem()`

MQDistributionListItem

Construye un objeto MQDistributionListItem nuevo.

MQEnvironment

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

clase pública **MQEnvironment**
expande **Object**

Nota: Todos los métodos y los atributos de esta clase se aplican a las conexiones de cliente de MQSeries Classes for Java, pero sólo enableTracing, disableTracing, properties y version_notice se aplican a las conexiones de enlaces.

MQEnvironment contiene variables de miembro estáticas que controlan en entorno en el que se construye un objeto MQQueueManager (y su conexión correspondiente con MQSeries).

Los valores establecidos en la clase MQEnvironment entran en vigor cuando se invoca al constructor MQQueueManager, de modo que debe establecer los valores en la clase MQEnvironment antes de construir una instancia de MQQueueManager.

Variables

Nota: Las variables marcadas con un asterisco (*) no se aplican cuando se conecta directamente a MQSeries en la modalidad de enlaces.

version_notice

```
public final static String version_notice
```

La versión actual de MQSeries Classes for Java.

securityExit*

```
public static MQSecurityExit securityExit
```

Rutina de salida de seguridad que le permite personalizar los flujos de comunicaciones de seguridad que se producen cuando se efectúa un intento de conexión a un gestor de colas.

Para proporcionar su propia rutina de salida de seguridad, defina una clase que implemente la interfaz MQSecurityExit y asigne securityExit a una instancia de dicha clase. De lo contrario, puede dejar securityExit establecida en nulo, en cuyo caso no se invoca ninguna rutina de salida de seguridad.

Vea también “MQSecurityExit” en la página 168.

sendExit*

```
public static MQSendExit sendExit
```

Una rutina de salida de emisión permite examinar y, posiblemente, modificar los datos enviados a un gestor de colas. Por lo general, se utiliza junto con una rutina de salida de recepción correspondiente en el gestor de colas.

Para proporcionar su propia rutina de salida de emisión, defina una clase que implemente la interfaz MQSendExit y asigne sendExit a una instancia de dicha clase. De lo contrario, puede dejar sendExit establecida en nulo, en cuyo caso no se invoca ninguna rutina de salida de emisión.

Vea también “MQSendExit” en la página 170.

receiveExit*

```
public static MQReceiveExit receiveExit
```

Una rutina de salida de recepción permite examinar y, posiblemente, modificar datos recibidos de un gestor de colas. Por lo general, se utiliza junto con una rutina de salida de emisión correspondiente en el gestor de colas.

Para proporcionar su propia rutina de salida de recepción, defina una clase que implemente la interfaz MQReceiveExit y asígnele receiveExit a una instancia de dicha clase. De lo contrario, puede dejar receiveExit establecida en nulo, en cuyo caso no se invoca ninguna rutina de salida de recepción.

Vea también “MQReceiveExit” en la página 166.

hostname*

```
public static String hostname
```

Nombre de sistema principal TCP/IP de la máquina en la que reside el servidor MQSeries. Si no se establece el nombre de sistema principal y no se establecen propiedades de alteración temporal, se utiliza la modalidad de enlaces para conectarse al gestor de colas local.

port* `public static int port`

Puerto al que se debe conectar. Es el puerto en el que el servidor MQSeries está a la escucha de las peticiones de conexión entrantes. El valor por omisión es 1414.

channel*

```
public static String channel
```

Nombre del canal con el que se debe conectar en el gestor de colas de destino. *Deberá* establecer esta variable de miembro, o la propiedad correspondiente, antes de construir una instancia de MQQueueManager para utilizarla en modalidad cliente.

userID*

```
public static String userID
```

Equivalente a la variable de entorno de MQSeries MQ_USER_ID.

Si no se define una rutina de salida de seguridad para este cliente, el valor de userID se transmite al servidor y estará disponible para la rutina de salida de seguridad del servidor cuando ésta se invoque. Se puede utilizar el valor para verificar la identidad del cliente MQSeries.

El valor por omisión es `""`.

password*

```
public static String password
```

Equivalente a la variable de entorno de MQSeries MQ_PASSWORD.

Si no se define una rutina de salida de seguridad para este cliente, el valor de password se transmite al servidor y está disponible para la rutina de salida de seguridad del servidor cuando ésta se invoca. Se puede utilizar el valor para verificar la identidad del cliente MQSeries.

El valor por omisión es ″″.

properties

```
public static java.util.Hashtable properties
```

Conjunto de pares clave/valor que definen el entorno de MQSeries.

Esta tabla de totales de control permite establecer las propiedades de entorno como pares clave/valor en lugar de establecerlas como variables individuales.

Las propiedades también se pueden pasar como una tabla de totales de control en un parámetro del constructor MQQueueManager. Las propiedades pasadas al constructor tienen prioridad sobre los valores establecidos con esta variable properties, pero en todo lo demás son intercambiables. El orden de prioridad de búsqueda de propiedades es:

1. Parámetro `properties` en el constructor MQQueueManager
2. MQEnvironment.properties
3. Otras variables de MQEnvironment
4. Valores por omisión constantes

En la tabla siguiente se muestran los pares clave/valor posibles:

Clave	Valor
MQC.CCSID_PROPERTY	Entero (altera temporalmente MQEnvironment.CCSID)
MQC.CHANNEL_PROPERTY	Serie (altera temporalmente MQEnvironment.channel)
MQC.CONNECT_OPTIONS_PROPERTY	Entero, toma por omisión MQC.MQCNO_NONE
MQC.HOST_NAME_PROPERTY	Serie (altera temporalmente MQEnvironment.hostname)
MQC.ORB_PROPERTY	org.omg.CORBA.ORB (opcional)
MQC.PASSWORD_PROPERTY	Serie (altera temporalmente MQEnvironment.password)
MQC.PORT_PROPERTY	Entero (altera temporalmente MQEnvironment.port)
MQC.RECEIVE_EXIT_PROPERTY	MQReceiveExit (altera temporalmente MQEnvironment.receiveExit)
MQC.SECURITY_EXIT_PROPERTY	MQSecurityExit (altera temporalmente MQEnvironment.securityExit)
MQC.SEND_EXIT_PROPERTY	MQSendExit (altera temporalmente MQEnvironment.sendExit).
MQC.TRANSPORT_PROPERTY	MQC.TRANSPORT_MQSERIES_BINDINGS o MQC.TRANSPORT_MQSERIES_CLIENT o MQC.TRANSPORT_VISIBROKER o MQC.TRANSPORT_MQSERIES (el valor por omisión, que selecciona enlaces o clientes, dependiendo del valor de ″hostname″).

Clave	Valor
MQC.USER_ID_PROPERTY	Serie (altera temporalmente MQEnvironment.userID).

CCSID*

```
public static int CCSID
```

CCSID utilizado por el cliente.

La modificación de este valor afecta al modo en que el gestor de colas con el que se efectúa la conexión convierte la información de las cabeceras MQSeries. Todos los datos de las cabeceras MQSeries se extraen de la parte invariable del conjunto de códigos ASCII, excepto los datos de los campos applicationIdData y putApplicationName de la clase MQMessage. (Consulte el apartado "MQMessage" en la página 112).

Si evita utilizar caracteres de la parte variable del conjunto de códigos ASCII para estos dos campos, podrá cambiar de forma segura el CCSID 819 por cualquier otro conjunto de códigos ASCII.

Si cambia el CCSID del cliente para que sea el mismo que el del gestor de colas al que se está conectando, conseguirá una mejora del rendimiento en el gestor de colas, porque éste no intentará convertir las cabeceras de mensaje.

El valor por omisión es 819.

Constructores

MQEnvironment

```
public MQEnvironment()
```

Métodos

disableTracing

```
public static void disableTracing()
```

Desactiva el recurso de rastreo de MQSeries Cliente para Java.

enableTracing

```
public static void enableTracing(int level)
```

Activa el recurso de rastreo de MQSeries Cliente para Java.

Parámetros

level Nivel de rastreo necesario, de 1 a 5 (donde 5 es el más detallado).

enableTracing

```
public static void enableTracing(int level,
                                 OutputStream stream)
```

Activa el recurso de rastreo de MQSeries Cliente para Java.

Parámetros:

level Nivel de rastreo necesario, de 1 a 5 (donde 5 es el más detallado).

stream Corriente de datos en la que se escribe el rastreo.

setDefaultConnectionManager

```
public static void setDefaultConnectionManager(MQConnectionManager cxManager)
```

Establece el MQConnectionManager suministrado como ConnectionManager por omisión. El ConnectionManager por omisión se utiliza cuando no se ha especificado ningún ConnectionManager en el constructor MQQueueManager. Este método también vacía el conjunto de MQPoolTokens.

Parámetros:

cxManager
MQConnectionManager establecido como ConnectionManager por omisión.

setDefaultConnectionManager

```
public static void setDefaultConnectionManager
                    (javax.resource.spi.ConnectionManager cxManager)
```

Establece el ConnectionManager por omisión y vacía el conjunto de MQPoolTokens. El ConnectionManager por omisión se utiliza cuando no se ha especificado ningún ConnectionManager en el constructor MQQueueManager.

Este método necesita JVM con Java 2 v1.3 o posterior, con JAAS 1.0 o posterior instalado.

Parámetros:

cxManager
ConnectionManager por omisión (que implementa la interfaz javax.resource.spi.ConnectionManager).

getDefaultConnectionManager

```
public static javax.resource.spi.ConnectionManager
                                  getDefaultConnectionManager()
```

Devuelve el ConnectionManager por omisión. Si el ConnectionManager por omisión es en realidad un MQConnectionManager, devuelve un valor nulo.

addConnectionPoolToken

```
public static void addConnectionPoolToken(MQPoolToken token)
```

Añade el MQPoolToken suministrado al conjunto de señales. Un ConnectionManager por omisión lo puede utilizar como indicación. Generalmente sólo se habilitan cuando hay, como mínimo, una señal en el conjunto.

Parámetros:

token MQPoolToken para añadir al conjunto de señales.

addConnectionPoolToken

```
public static MQPoolToken addConnectionPoolToken()
```

Construye un MQPoolToken y los añade al conjunto de señales. El MQPoolToken se devuelve a la aplicación que se va a pasar posteriormente en removeConnectionPoolToken().

removeConnectionPoolToken

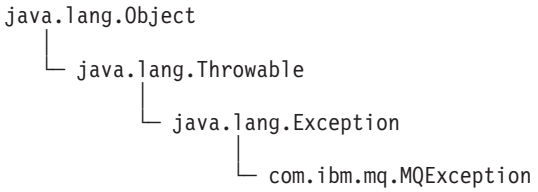
```
public static void removeConnectionPoolToken(MQPoolToken token)
```

Elimina el MQPoolToken especificado del conjunto de señales. Si el MQPoolToken no está en el conjunto, no se lleva a cabo ninguna acción.

Parámetros:

token MQPoolToken que se va a eliminar del conjunto de señales.

MQException



clase pública **MQException**
expande **Exception**

Siempre que se produce un error de MQSeries se emite una MQException. Puede cambiar la corriente de datos de salida para las excepciones que se anotan cronológicamente estableciendo el valor de MQException.log. El valor por omisión es System.err. Esta clase contiene definiciones de constantes de código de error y código de terminación. Las constantes que empiezan por MQCC_ son códigos de terminación de MQSeries y las constantes que empiezan por MQRC_ son códigos de razón de MQSeries. La publicación *MQSeries® Application Programming Reference* contiene una descripción completa de estos errores y sus causas probables.

Variables

log `public static java.io.outputStreamWriter log`

Corriente de datos en la que se anotan cronológicamente las excepciones. (El valor por omisión es System.err). Si se establece en nulo, no se realizan anotaciones cronológicas.

completionCode
`public int completionCode`

Código de terminación de MQSeries que da lugar al error. Los valores posibles son:
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

reasonCode
`public int reasonCode`

Código de razón de MQSeries que describe el error. Si desea ver una explicación completa de los códigos de razón, consulte el manual *MQSeries® Application Programming Reference*.

exceptionSource
`public Object exceptionSource`

Instancia del objeto que ha emitido la excepción. Puede utilizarla como parte del diagnóstico al determinar la causa de un error.

Constructores

MQException

```
public MQException(int completionCode,
                   int reasonCode,
                   Object source)
```

Construir un objeto MQException nuevo.

Parámetros

completionCode
Código de terminación de MQSeries.

reasonCode
Código de razón de MQSeries.

source Objeto en el que se ha producido el error.

MQGetMessageOptions

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

clase pública **MQGetMessageOptions**
expande **Object**

Esta clase contiene opciones que controlan la presentación de MQQueue.get().

Nota: La presentación de algunas de las opciones disponibles en esta clase depende del entorno en el que se utilicen. Estos elementos están marcados con el símbolo *. Consulte el "Capítulo 8. Presentación dependiente del entorno" en la página 81 para obtener información más detallada.

Variables

options

```
public int options
```

Opciones que controlan la acción de MQQueue.get. Se puede especificar cualquiera o ninguno de los valores siguientes. Si se necesita más de una opción, los valores pueden añadirse juntos o combinarse utilizando el operador OR que tiene en cuenta los bits.

MQC.MQGMO_NONE

MQC.MQGMO_WAIT
Esperar a que llegue un mensaje.

MQC.MQGMO_NO_WAIT
Volver inmediatamente si no hay ningún mensaje adecuado.

MQC.MQGMO_SYNCPOINT
Obtener el mensaje bajo el control de punto de sincronismo; el mensaje se marca como no disponible para otras aplicaciones, pero sólo se suprime de la cola cuando se confirma la unidad de trabajo. El mensaje está disponible otra vez si se restituye la unidad de trabajo.

MQC.MQGMO_NO_SYNCPOINT
Obtener el mensaje sin control de punto de sincronismo.

MQC.MQGMO_BROWSE_FIRST
Examinar desde el inicio de la cola.

MQC.MQGMO_BROWSE_NEXT
Examinar desde la posición actual de la cola.

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
Examinar el mensaje bajo el cursor de examinar.

MQC.MQGMO_MSG_UNDER_CURSOR
Obtener el mensaje que hay bajo el cursor de examinar.

MQC.MQGMO_LOCK*
Bloquear el mensaje que se está examinando.

MQC.MQGMO_UNLOCK*
Desbloquear un mensaje bloqueado anteriormente.

MQC.MQGMO_ACCEPT_TRUNCATED_MSG
Permite el truncamiento de datos de mensaje.

MQC.MQGMO_FAIL_IF_QUIESCING
Error si el gestor de colas se está inmovilizando.

MQC.MQGMO_CONVERT
Solicitar los datos de aplicación que se deben convertir, para que se ajusten a los atributos characterSet y de codificación de MQMessage, antes de que se copien los datos en el almacenamiento intermedio de mensajes. Dado que la conversión de datos también se aplica mientras se recuperan los datos del almacenamiento intermedio de mensajes, normalmente las aplicaciones no establecen esta opción.

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
Obtener mensaje con control de punto de sincronismo si el mensaje es permanente.

MQC.MQGMO_MARK_SKIP_BACKOUT*
Permitir que se restituya una unidad de trabajo sin volver a colocar el mensaje en la cola.

Segmentación y agrupación Los mensajes de MQSeries se pueden enviar o recibir como una única entidad, se pueden subdividir en varios segmentos para su envío o recepción y también se pueden enlazar con otros mensajes en un grupo.

Cada fragmento de datos que se envía se conoce como mensaje *físico*, que puede ser un mensaje *lógico* completo o un segmento de un mensaje lógico más largo.

Cada mensaje físico tiene, generalmente, un `MsgId` diferente. Todos los segmentos de un solo mensaje lógico tienen el mismo valor de `groupId` y de `MsgSeqNumber`, pero el valor de `Offset` es diferente para cada segmento. El campo `Offset` proporciona el desplazamiento de los datos en el mensaje físico desde el inicio del mensaje lógico. Los segmentos tienen normalmente valores de `MsgId` diferentes, puesto que son mensajes físicos individuales.

Los mensajes lógicos que forman parte de un grupo tienen el mismo valor de `groupId`, pero cada mensaje del grupo tiene un valor de `MsgSeqNumber` diferente. Los mensajes de un grupo también se pueden segmentar.

Se pueden utilizar las opciones siguientes para tratar con mensajes segmentados o agrupados:

MQC.MQGMO_LOGICAL_ORDER*
Devolver mensajes en grupos y segmentos de mensajes lógicos en orden lógico.

MQC.MQGMO_COMPLETE_MSG*
Recuperar sólo mensajes lógicos completos.

MQC.MQGMO_ALL_MSGS_AVAILABLE*
Recuperar mensajes de un grupo solamente cuando todos los mensajes del grupo estén disponibles.

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
Recuperar los segmentos de un mensaje lógico sólo cuando todos los segmentos del grupo estén disponibles.

waitInterval
`public int waitInterval`

Tiempo máximo (en milisegundos) que una invocación MQQueue.get espera a que llegue un mensaje adecuado (se utiliza junto con MQC.MQGMO_WAIT). Un valor de MQC.MQWI_UNLIMITED indica que se necesita una espera ilimitada.

resolvedQueueName
`public String resolvedQueueName`

Éste es un campo de salida que el gestor de colas establece en el nombre local de la cola de la que se ha recuperado el mensaje. Será diferente del nombre utilizado para abrir la cola si se ha abierto una cola alias o cola modelo.

matchOptions*
`public int matchOptions`

Criterios de selección que determinan qué mensaje se recupera. Se pueden establecer las opciones siguientes de comparación:

MQC.MQMO_MATCH_MSG_ID
ID de mensaje a comparar.

MQC.MQMO_MATCH_CORREL_ID
ID de correlación a comparar.

MQC.MQMO_MATCH_GROUP_ID
ID de grupo a comparar.

MQC.MQMO_MATCH_MSG_SEQ_NUMBER
Número de secuencia de mensaje de comparación.

MQC.MQMO_NONE
No es necesaria ninguna comparación.

groupStatus*
`public char groupStatus`

Éste es un campo de salida que indica si el mensaje recuperado está en un grupo y, si lo está, si es el último del grupo. Los valores posibles son:

MQC.MQGS_NOT_IN_GROUP
El mensaje no está en un grupo.

MQC.MQGS_MSG_IN_GROUP
El mensaje está en un grupo, pero no es el último del grupo.

MQC.MQGS_LAST_MSG_IN_GROUP
El mensaje es el último del grupo. También es el valor devuelto si el grupo consta solamente de un mensaje.

segmentStatus*
`public char segmentStatus`

Se trata de un campo de salida que indica si el mensaje recuperado es un segmento de un mensaje lógico. Si el mensaje es un segmento, el distintivo especifica si es el último segmento o no. Los valores posibles son:

MQC.MQSS_NOT_A_SEGMENT
El mensaje no es un segmento.

MQC.MQSS_SEGMENT
El mensaje es un segmento, pero no es el último segmento del mensaje lógico.

MQC.MQSS_LAST_SEGMENT
El mensaje es el último segmento del mensaje lógico. Es también el valor devuelto si el mensaje lógico consta solamente de un segmento.

segmentation*
```
public char segmentation
```

Se trata de un campo de salida que indica si se permite o no la segmentación porque el mensaje recuperado es un segmento de un mensaje lógico. Los valores posibles son:

MQC.MQSEG_INHIBITED
Segmentación no permitida.

MQC.MQSEG_ALLOWED
Segmentación permitida.

Constructores

MQGetMessageOptions
```
public MQGetMessageOptions()
```

Construir un objeto MQGetMessageOptions nuevo con opciones establecidas en MQC.MQGMO_NO_WAIT, un intervalo de espera de cero y un nombre de cola resuelto en blanco.

MQManagedObject

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

clase pública **MQManagedObject**
expande **Object**

MQManagedObject es una superclase para MQQueueManager, MQQueue y MQProcess. Proporciona la posibilidad de consultar y establecer atributos de estos recursos.

Variables

alternateUserId
`public String alternateUserId`

ID de usuario alternativo especificado (si existía) al abrir este recurso. El establecimiento de este atributo no tiene ningún efecto.

name `public String name`

Nombre de este recurso (el nombre proporcionado en el método de acceso o el nombre asignado por el gestor de colas para una cola dinámica). El establecimiento de este atributo no tiene ningún efecto.

openOptions
`public int openOptions`

Opciones especificadas al abrir este recurso. El establecimiento de este atributo no tiene ningún efecto.

isOpen
`public boolean isOpen`

Indica si este recurso está abierto actualmente. Este atributo se *ignora* y su establecimiento no tiene ningún efecto.

connectionReference
`public MQQueueManager connectionReference`

Gestor de colas al que pertenece este recurso. El establecimiento de este atributo no tiene ningún efecto.

closeOptions
`public int closeOptions`

Establezca este atributo para controlar el modo en que se cierra el recurso. El valor por omisión es MQC.MQCO_NONE y éste es el único valor permitido para todos los recursos distintos de las colas dinámicas permanentes, y las colas dinámicas temporales a las que acceden los objetos que las han creado. Para estas colas, se permiten los valores adicionales siguientes:

MQC.MQCO_DELETE
Suprime la cola si no hay mensajes.

MQC.MQCO_DELETE_PURGE
Suprime la cola, eliminando los mensajes que contiene.

Constructores

MQManagedObject

```
protected MQManagedObject()
```

Método constructor.

Métodos

getDescription

```
public String getDescription()
```

Emite MQException.

Devuelve la descripción de este recurso tal como está en el gestor de colas.

Si se invoca este método después de que se haya cerrado el recurso, se emite una MQException.

inquire

```
public void inquire(int selectors[],
                    int intAttrs[],
                    byte charAttrs[])
```

Emite MQException.

Devuelve una matriz de enteros y un conjunto de series de caracteres que contiene los atributos de un objeto (cola, proceso o gestor de colas).

Los atributos a consultar se especifican en la matriz de selectores. Consulte la publicación *MQSeries® Application Programming Reference* para obtener información detallada sobre los selectores permitidos y sus valores de entero correspondientes.

Tenga en cuenta que muchos de los atributos más comunes se pueden consultar utilizando los métodos getXXX() definidos en MQManagedObject, MQQueue, MQQueueManager y MQProcess.

Parámetros

selectors
Matriz de enteros que identifica a los atributos con valores acerca de los cuales se efectúan consultas.

intAttrs
Matriz en la que se devuelven los valores de atributos de enteros. Los valores de atributos de enteros se devuelven en el mismo orden que los selectores de atributos de enteros en la matriz de selectores.

charAttrs
Almacenamiento intermedio en el que se devuelven los atributos de carácter concatenados. Los atributos de carácter se devuelven en el mismo orden que los selectores de atributos de carácter en la matriz de selectores. La longitud de cada serie de atributos es fija para cada atributo.

Emite MQException si la consulta no se ejecuta correctamente.

isOpen

```
public boolean isOpen()
```

Devuelve el valor de la variable `isOpen`.

set

```
public synchronized void set(int selectors[],
                             int intAttrs[],
                             byte charAttrs[])
```

Emite MQException.

Establece los atributos definidos en el vector del selector.

Los atributos que se deben establecer se especifican en la matriz de selectores. Consulte la publicación *MQSeries® Application Programming Reference* para obtener información detallada sobre los selectores permitidos y sus valores de entero correspondientes.

Tenga en cuenta que algunos atributos de cola pueden establecerse utilizando los métodos setXXX() definidos en MQQueue.

Parámetros

selectors
: Matriz de enteros que identifica los atributos con valores que se deben establecer.

intAttrs
: Matriz de valores de atributos de enteros que se deben establecer. Estos valores deben estar en el mismo orden que los selectores de atributos de enteros en la matriz de selectores.

charAttrs
: Almacenamiento intermedio en el que se concatenan los atributos de carácter que se deben establecer. Estos valores deben estar en el mismo orden que los selectores de atributos de carácter en la matriz de selectores. La longitud de cada atributo de carácter es fija.

Emite MQException si el establecimiento no se ejecuta correctamente.

close

```
public synchronized void close()
```

Emite MQException.

Cierra el objeto. No se permiten operaciones adicionales en este recurso después de invocarse este método. La presentación del método close puede modificarse estableciendo el atributo closeOptions.

Emite MQException si la invocación de MQSeries no se ejecuta correctamente.

MQMessage

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

clase pública **MQMessage**
implementa **DataInput**, **DataOutput**

MQMessage representa el descriptor de mensaje y los datos para un mensaje MQSeries. Hay un grupo de métodos readXXX para leer datos de un mensaje y un grupo de métodos writeXXX para grabar datos en un mensaje. El formato de los números y las series de caracteres utilizado por estos métodos de lectura y grabación puede controlarse mediante la codificación y las variables de miembro characterSet. Las variables de miembro restantes contienen información de control que acompaña a los datos de mensaje de la aplicación cuando un mensaje viaja entre aplicaciones emisoras y receptoras. La aplicación puede establecer valores en la variable de miembro antes de transferir un mensaje a una cola y puede leer valores después de recuperar un mensaje de una cola.

Variables

report `public int report`

Un informe es un mensaje acerca de otro mensaje. Esta variable de miembro permite a la aplicación que envía el mensaje original especificar qué mensajes de informe son necesarios, si deben incluirse en ellos los datos de mensaje de la aplicación y también cómo deben establecerse los identificadores de mensaje y correlación en el informe o la respuesta. Se puede solicitar uno, varios, ninguno o la totalidad de los tipos de informe siguientes:

- Excepción
- Caducidad
- Confirmación de llegada
- Confirmación de entrega

Para cada tipo, sólo se deberá especificar uno de los tres valores correspondientes indicados más abajo, en función de si se deben incluir o no en el mensaje de informe los datos de mensaje de aplicación.

Nota: Los gestores de colas MVS no soportan los valores marcados con ** en la lista siguiente y dichos valores no deberán utilizarse si es probable que la aplicación vaya a acceder a un gestor de colas MVS, independientemente de la plataforma en la que se ejecuta la aplicación.

Los valores válidos son:

- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**
- MQC.MQRO_COA
- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**

- MQC.MQRO_COD
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

Puede especificar uno de los siguientes valores para controlar cómo se genera el ID de mensaje para el informe o mensaje de respuesta:
- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

Puede especificar uno de los siguientes valores para controlar cómo debe establecerse el ID de correlación del informe o mensaje de respuesta:
- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

Puede especificar uno de los siguientes valores para controlar la disposición del mensaje original cuando no puede entregarse a la cola de destino:
- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **

Si no se ha especificado ninguna opción de informe, el valor por omisión es el siguiente:

```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

Puede especificar uno de los siguientes valores o ambos para solicitar que la aplicación receptora envíe un mensaje de informe de acción positiva o acción negativa.
- MQRO_PAN
- MQRO_NAN

messageType

```
public int messageType
```

Indica el tipo de mensaje. El sistema define actualmente los valores siguientes:
- MQC.MQMT_DATAGRAM
- MQC.MQMT_REQUEST
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT

También se pueden utilizar valores definidos por la aplicación. Éstos deberán estar en el rango MQC.MQMT_APPL_FIRST a MQC.MQMT_APPL_LAST.

El valor por omisión de este campo es MQC.MQMT_DATAGRAM.

expiry `public int expiry`

Tiempo de caducidad expresado en décimas de segundo, establecido por la aplicación que transfiere el mensaje. Cuando ha transcurrido el tiempo de caducidad de un mensaje, el gestor de colas podrá seleccionarlo para eliminarlo. Si el mensaje especificaba uno de los distintivos MQC.MQRO_EXPIRATION, se generará un informe cuando se elimine el mensaje.

El valor por omisión es MQC.MQEI_UNLIMITED, lo que significa que el mensaje no caduca nunca.

feedback

```
public int feedback
```

Se utiliza con un mensaje de tipo MQC.MQMT_REPORT para indicar la naturaleza del informe. El sistema define los códigos de información de retorno siguientes:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

También se pueden utilizar los valores de información de retorno definidos por aplicación en el rango MQC.MQFB_APPL_FIRST a MQC.MQFB_APPL_LAST.

El valor por omisión de este campo es MQC.MQFB_NONE, que indica que no se proporciona ninguna información de retorno.

encoding

```
public int encoding
```

Esta variable de miembro especifica la representación utilizada para los valores numéricos en los datos de mensajes de aplicación; esto se aplica a datos binarios, decimales empaquetados y datos de coma flotante. La presentación de los métodos de lectura y grabación para estos formatos numéricos se modifica como corresponde.

Para enteros binarios se definen las codificaciones siguientes:

MQC.MQENC_INTEGER_NORMAL
Enteros big endian, como en Java™

MQC.MQENC_INTEGER_REVERSED
Enteros little endian, como los que utilizan los PC.

Para enteros decimales empaquetados se definen las codificaciones siguientes:

MQC.MQENC_DECIMAL_NORMAL
Decimales empaquetados big endian, como los que utiliza System/390®.

MQC.MQENC_DECIMAL_REVERSED
Decimales empaquetados little endian.

Para números de coma flotante se definen las codificaciones siguientes:

MQC.MQENC_FLOAT_IEEE_NORMAL
Números de coma flotante IEEE big endian, como en Java™.

MQC.MQENC_FLOAT_IEEE_REVERSED
Números de coma flotante IEEE little endian, como los que utilizan los PC.

MQC.MQENC_FLOAT_S390
Números de coma flotante con formato de System/390®.

Un valor para el campo de codificación debe construirse añadiendo un valor de cada una de estas tres secciones (o utilizando el operador OR que tiene en cuenta los bits). El valor por omisión es el siguiente:

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

Para mayor comodidad, MQC.MQENC_NATIVE también representa este valor. Este valor hace que writeInt() escriba un entero big endian y que readInt() lea un entero big endian. Si en su lugar se hubiera establecido el distintivo MQC.MQENC_INTEGER_REVERSED, writeInt() escribiría un entero little endian y readInt() leería un entero little endian.

Tenga en cuenta que se puede producir una pérdida de precisión al convertir de comas flotantes de formato IEEE a comas flotantes de formato System/390®.

characterSet

```
public int characterSet
```

Especifica el identificador de juego de caracteres codificado de datos de tipo carácter en los datos de mensaje de la aplicación. La presentación de los métodos readString, readLine y writeString se modifica como corresponde.

El valor por omisión para este campo es MQC.MQCCSI_Q_MGR, que especifica que los datos de tipo carácter de los datos de mensaje de la aplicación están en el juego de caracteres del gestor de colas. Se soportan los valores de juego de caracteres adicionales mostrados en la Tabla 13.

Tabla 13. Identificadores de juego de caracteres

characterSet	Descripción
819	iso-8859-1 / latin1 / ibm819
912	iso-8859-2 / latin2 / ibm912
913	iso-8859-3 / latin3 / ibm913
914	iso-8859-4 / latin4 / ibm914
915	iso-8859-5 / cirílico / ibm915
1089	iso-8859-6 / árabe / ibm1089
813	iso-8859-7 / griego / ibm813
916	iso-8859-8 / hebreo / ibm916
920	iso-8859-9 / latin5 / ibm920
37	ibm037
273	ibm273
277	ibm277
278	ibm278
280	ibm280
284	ibm284
285	ibm285
297	ibm297
420	ibm420
424	ibm424
437	ibm437 / PC Original
500	ibm500
737	ibm737 / PC Griego

Tabla 13. Identificadores de juego de caracteres (continuación)

characterSet	Descripción
775	ibm775 / PC Báltico
838	ibm838
850	ibm850 / PC Latin 1
852	ibm852 / PC Latin 2
855	ibm855 / PC Cirílico
856	ibm856
857	ibm857 / PC Turco
860	ibm860 / PC Portugués
861	ibm861 / PC Islandés
862	ibm862 / PC Hebreo
863	ibm863 / PC Francés canadiense
864	ibm864 / PC Árabe
865	ibm865 / PC Nórdico
866	ibm866 / PC Ruso
868	ibm868
869	ibm869 / PC Griego moderno
870	ibm870
871	ibm871
874	ibm874
875	ibm875
918	ibm918
921	ibm921
922	ibm922
930	ibm930
933	ibm933
935	ibm935
937	ibm937
939	ibm939
942	ibm942
948	ibm948
949	ibm949
950	ibm950 / Big 5 Chino tradicional
964	ibm964 / CNS 11643 Chino tradicional
970	ibm970
1006	ibm1006
1025	ibm1025
1026	ibm1026
1097	ibm1097
1098	ibm1098
1112	ibm1112
1122	ibm1122
1123	ibm1123
1124	ibm1124
1381	ibm1381
1383	ibm1383
2022	JIS
932	PC Japonés
954	EUCJIS
1250	Windows® Latin 2
1251	Windows® Cirílico
1252	Windows® Latin 1
1253	Windows® Griego
1254	Windows® Turco

Tabla 13. Identificadores de juego de caracteres (continuación)

characterSet	Descripción
1255	Windows® Hebreo
1256	Windows® Árabe
1257	Windows® Báltico
1258	Windows® Vietnamita
33722	ibm33722
5601	ksc-5601 Coreano
1200	Unicode
1208	UTF-8

format

```
public String format
```

Nombre de formato que utiliza el emisor del mensaje para indicar la naturaleza de los datos del mensaje al receptor. Puede utilizar sus propios nombres de formato, pero los nombres que empiezan con las letras ″MQ″ tienen significados definidos por el gestor de colas. Los formatos incorporados al gestor de colas son:

MQC.MQFMT_NONE
Ningún nombre de formato.

MQC.MQFMT_ADMIN
Mensaje de petición/respuesta del servidor de mandatos.

MQC.MQFMT_COMMAND_1
Mensaje de respuesta de mandato de tipo 1.

MQC.MQFMT_COMMAND_2
Mensaje de respuesta de mandato de tipo 2.

MQC.MQFMT_DEAD_LETTER_HEADER
Cabecera de mensaje no entregado.

MQC.MQFMT_EVENT
Mensaje de suceso.

MQC.MQFMT_PCF
Mensaje definido por el usuario en formato de mandato programable.

MQC.MQFMT_STRING
Mensaje que consta enteramente de caracteres.

MQC.MQFMT_TRIGGER
Mensaje de activación.

MQC.MQFMT_XMIT_Q_HEADER
Cabecera de la cola de transmisión.

El valor por omisión es MQC.MQFMT_NONE.

priority

```
public int priority
```

Prioridad del mensaje. También se puede establecer el valor especial MQC.MQPRI_PRIORITY_AS_Q_DEF en los mensajes salientes, en cuyo caso la prioridad para el mensaje se toma del atributo de prioridad por omisión de la cola de destino.

El valor por omisión es MQC.MQPRI_PRIORITY_AS_Q_DEF.

persistence

```
public int persistence
```

Permanencia del mensaje. Se definen los valores siguientes:
- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

El valor por omisión es MQC.MQPER_PERSISTENCE_AS_Q_DEF, que indica que la permanencia para el mensaje debe tomarse del atributo de permanencia por omisión de la cola de destino.

messageId

```
public byte messageId[]
```

Para una invocación MQQueue.get(), este campo especifica el identificador del mensaje que se debe recuperar. Normalmente, el gestor de colas devuelve el primer mensaje con un identificador de mensaje y un identificador de correlación que coinciden con los especificados. El valor especial MQC.MQMI_NONE permite que coincida *cualquier* identificador de mensaje.

Para una invocación MQQueue.put(), especifica el identificador de mensaje que se debe utilizar. Si se especifica MQC.MQMI_NONE, el gestor de colas genera un identificador de mensaje exclusivo cuando se transfiere el mensaje. El valor de esta variable de miembro se actualiza después de la operación de transferencia para indicar el identificador de mensaje que se ha utilizado.

El valor por omisión es MQC.MQMI_NONE.

correlationId

```
public byte correlationId[]
```

Para una invocación MQQueue.get(), este campo especifica el identificador de correlación de mensaje que se debe recuperar. Normalmente, el gestor de colas devuelve el primer mensaje con un identificador de mensaje y un identificador de correlación que coinciden con los especificados. El valor especial MQC.MQCI_NONE permite que coincida *cualquier* identificador de correlación.

Para una invocación MQQueue.put(), especifica el identificador de correlación que se debe utilizar.

El valor por omisión es MQC.MQCI_NONE.

backoutCount

```
public int backoutCount
```

Recuento del número de veces que el mensaje ha sido devuelto anteriormente por una invocación MQQueue.get() como parte de una unidad de trabajo y, posteriormente, se ha restituido.

El valor por omisión es cero.

replyToQueueName

```
public String replyToQueueName
```

Nombre de la cola de mensajes a la que debe enviar mensajes MQC.MQMT_REPLY y MQC.MQMT_REPORT la aplicación que ha emitido la petición de obtención para el mensaje.

El valor por omisión es "".

replyToQueueManagerName
`public String replyToQueueManagerName`

Nombre del gestor de colas al que se deben enviar mensajes de respuesta o de informe.

El valor por omisión es "".

Si el valor es "" en una invocación MQQueue.put(), QueueManager rellena el valor.

userId `public String userId`

Parte del contexto de identidad del mensaje; identifica al usuario que ha originado el mensaje.

El valor por omisión es "".

accountingToken
`public byte accountingToken[]`

Parte del contexto de identidad del mensaje; permite a una aplicación cargar apropiadamente el trabajo efectuado como resultado del mensaje.

El valor por omisión es "MQC.MQACT_NONE".

applicationIdData
`public String applicationIdData`

Parte del contexto de identidad del mensaje; es información que está definida por la serie de aplicaciones y que se puede utilizar para proporcionar información adicional acerca del mensaje o de su emisor.

El valor por omisión es "".

putApplicationType
`public int putApplicationType`

Tipo de aplicación que ha transferido el mensaje. Puede ser un valor definido por el usuario o por el sistema. El sistema define los valores siguientes:
- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

El valor por omisión es el valor especial MQC.MQAT_NO_CONTEXT, que indica que no existe información de contexto en el mensaje.

putApplicationName
`public String putApplicationName`

Nombre de la aplicación que ha transferido el mensaje. El valor por omisión es "".

putDateTime
`public GregorianCalendar putDateTime`

Hora y fecha en la que se ha transferido el mensaje.

applicationOriginData
```
public String applicationOriginData
```

Información definida por la aplicación que se puede utilizar para proporcionar información adicional acerca del origen del mensaje.

El valor por omisión es "".

groupId
```
public byte[] groupId
```

Serie de bytes que identifica el grupo de mensajes al que pertenece el mensaje físico.

El valor por omisión es "MQC.MQGI_NONE".

messageSequenceNumber
```
public int messageSequenceNumber
```

Número de secuencia de un mensaje lógico dentro de un grupo.

orfset `public int offset`

En un mensaje segmentado, desplazamiento de los datos de un mensaje físico desde el inicio de un mensaje lógico.

messageFlags
```
public int messageFlags
```

Distintivos que controlan la segmentación y el estado de un mensaje.

originalLength
```
public int originalLength
```

Longitud original de un mensaje segmentado.

Constructores

MQMessage
```
public MQMessage()
```

Crea un nuevo mensaje con información del descriptor del mensaje por omisión y un almacenamiento intermedio de mensajes vacío.

Métodos

getTotalMessageLength
```
public int getTotalMessageLength()
```

Número total de bytes del mensaje tal como se ha almacenado en la cola de mensajes de la que se ha recuperado (o se ha intentado recuperar). Cuando un método MQQueue.get() no se ejecuta correctamente y produce un código de error de mensaje truncado, este método indica el tamaño total del mensaje en la cola.

Vea también "MQQueue.get" en la página 143.

getMessageLength
```
public int getMessageLength
```

Emite IOException.

Número de bytes de datos de mensaje de este objeto MQMessage.

getDataLength

```
public int getDataLength()
```

Emite MQException.

Número de bytes de datos de mensaje que quedan por leer.

seek

```
public void seek(int pos)
```

Emite IOException.

Coloca el cursor en la posición absoluta del almacenamiento intermedio de mensajes que proporciona *pos*. Las lecturas y grabaciones subsiguientes actuarán en esta posición del almacenamiento intermedio.

Emite EOFException si pos está fuera de la longitud de datos del mensaje.

setDataOffset

```
public void setDataOffset(int offset)
```

Emite IOException.

Coloca el cursor en la posición absoluta del almacenamiento intermedio de mensajes. Este método es un sinónimo de seek() y se proporciona para compatibilidad de los lenguajes cruzados con las demás API de MQSeries.

getDataOffset

```
public int getDataOffset()
```

Emite IOException.

Devuelve la posición de cursor actual dentro de los datos de mensaje (el punto en el que entrarán en vigor las operaciones de lectura y grabación).

clearMessage

```
public void clearMessage()
```

Emite IOException.

Elimina cualquier dato del almacenamiento intermedio de mensajes y establece el desplazamiento de datos otra vez en cero.

getVersion

```
public int getVersion()
```

Devuelve la versión de la estructura en uso.

resizeBuffer

```
public void resizeBuffer(int size)
```

Emite IOException.

Indicación para el objeto MQMessage acerca del tamaño del almacenamiento intermedio que se puede necesitar para operaciones de obtención subsiguientes. Si actualmente el mensaje contiene datos de mensaje y el tamaño nuevo es menor que el tamaño actual, se truncan los datos del mensaje.

readBoolean

```
public boolean readBoolean()
```

Emite IOException.

Lee un byte (con signo) desde la posición actual del almacenamiento intermedio de mensajes.

readChar

```
public char readChar()
```

Emite IOException, EOFException.

Lee un carácter Unicode desde la posición actual del almacenamiento intermedio de mensajes.

readDouble

```
public double readDouble()
```

Emite IOException, EOFException.

Lee un número de dos bytes desde la posición actual del almacenamiento intermedio de mensajes. El valor de la variable de miembro de codificación determina la presentación de este método.

Los valores MQC.MQENC_FLOAT_IEEE_NORMAL y MQC.MQENC_FLOAT_IEEE_REVERSED leen números de dos bytes conformes a las normas IEEE en formatos big endian y little endian respectivamente.

Un valor MQC.MQENC_FLOAT_S390 lee un número de coma flotante de formato System/390®.

readFloat

```
public float readFloat()
```

Emite IOException, EOFException.

Lee un número de coma flotante desde la posición actual en el almacenamiento intermedio de mensajes. El valor de la variable de miembro de codificación determina la presentación de este método.

Los valores MQC.MQENC_FLOAT_IEEE_NORMAL y MQC.MQENC_FLOAT_IEEE_REVERSED leen números de coma flotante conformes a las normas IEEE en formatos big endian y little endian respectivamente.

Un valor MQC.MQENC_FLOAT_S390 lee un número de coma flotante de formato System/390®.

readFully

```
public void readFully(byte b[])
```

Emite Exception, EOFException.

Rellena la matriz de bytes `b` con datos del almacenamiento intermedio de mensajes.

readFully

```
public void readFully(byte b[],
                      int off,
                      int len)
```

Emite IOException, EOFException.

Rellena los elementos *len* de la matriz de bytes b con datos del almacenamiento intermedio de mensajes, empezando en el desplazamiento *off*.

readInt

```
public int readInt()
```

Emite IOException, EOFException.

Lee un entero desde la posición actual del almacenamiento intermedio de mensajes. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_INTEGER_NORMAL lee un entero big endian, un valor MQC.MQENC_INTEGER_REVERSED lee un entero little endian.

readInt4

```
public int readInt4()
```

Emite IOException, EOFException.

Sinónimo de readInt(), proporcionado para compatibilidad de las API de MQSeries de lenguajes cruzados.

readLine

```
public String readLine()
```

Emite IOException.

Convierte a Unicode el conjunto de códigos identificado en la variable de miembro characterSet y luego lee una línea que se ha terminado con \n, \r, \r\n o EOF.

readLong

```
public long readLong()
```

Emite IOException, EOFException.

Lee un número largo desde la posición actual del almacenamiento intermedio de mensajes. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_INTEGER_NORMAL lee un número largo big endian, un valor MQC.MQENC_INTEGER_REVERSED lee un número largo little endian.

readInt8

```
public long readInt8()
```

Emite IOException, EOFException.

Sinónimo de readLong(), proporcionado para compatibilidad de las API de MQSeries de lenguajes cruzados.

readObject

```
public Object readObject()
```

Emite OptionalDataException, ClassNotFoundException e IOException.

Lee un objeto del almacenamiento intermedio de mensajes. Se leen la clase del objeto, la firma de la clase y el valor de los campos no transitorios y no estáticos de la clase.

readShort

```
public short readShort()
```

Emite IOException, EOFException.

readInt2

```
public short readInt2()
```

Emite IOException, EOFException.

Sinónimo de readShort(), proporcionado para compatibilidad de las API de MQSeries de lenguajes cruzados.

readUTF

```
public String readUTF()
```

Emite IOException.

Lee una serie de caracteres UTF, con prefijo de un campo de 2 bytes de longitud, desde la posición actual del almacenamiento intermedio de mensajes.

readUnsignedByte

```
public int readUnsignedByte()
```

Emite IOException, EOFException.

Lee un byte sin signo desde la posición actual del almacenamiento intermedio de mensajes.

readUnsignedShort

```
public int readUnsignedShort()
```

Emite IOException, EOFException.

Lee un número corto sin signo desde la posición actual del almacenamiento intermedio de mensajes. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_INTEGER_NORMAL lee un número corto big endian sin signo, un valor MQC.MQENC_INTEGER_REVERSED lee un número corto little endian sin signo.

readUInt2

```
public int readUInt2()
```

Emite IOException, EOFException.

Sinónimo de readUnsignedShort(), proporcionado para compatibilidad las API de MQSeries de lenguajes cruzados.

readString

```
public String readString(int length)
```

Emite IOException, EOFException.

Lee una serie de caracteres en el conjunto de códigos que identifica la variable de miembro characterSet y convertirla a Unicode.

Parámetros:

length Número de caracteres que se van a leer (que puede ser diferente del número de bytes según el conjunto de códigos, porque algunos conjuntos de códigos utilizan más de un byte por carácter).

readDecimal2

```
public short readDecimal2()
```

Emite IOException, EOFException.

Lee un número decimal de 2 bytes empaquetado (-999..999). El valor de la variable de miembro de codificación controla la presentación de este método. Un valor MQC.MQENC_DECIMAL_NORMAL lee un número decimal big endian empaquetado y un valor MQC.MQENC_DECIMAL_REVERSED lee un número decimal little endian empaquetado.

readDecimal4

```
public int readDecimal4()
```

Emite IOException, EOFException.

Lee un número decimal de 4 bytes empaquetado (-9999999..9999999). El valor de la variable de miembro de codificación controla la presentación de este método. Un valor MQC.MQENC_DECIMAL_NORMAL lee un número decimal big endian empaquetado y un valor MQC.MQENC_DECIMAL_REVERSED lee un número decimal little endian empaquetado.

readDecimal8

```
public long readDecimal8()
```

Emite IOException, EOFException.

Lee un número decimal de 8 bytes empaquetado (-999999999999999 a 999999999999999). La variable de miembro de codificación controla la presentación de este método. Un valor MQC.MQENC_DECIMAL_NORMAL lee un número decimal big endian empaquetado y MQC.MQENC_DECIMAL_REVERSED lee un número decimal little endian empaquetado.

setVersion

```
public void setVersion(int version)
```

Especifica qué versión de la estructura se debe utilizar. Los valores posibles son:
- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

Normalmente no necesita invocar este método a no ser que desee forzar al cliente a utilizar una estructura de la versión 1 cuando esté conectado a un gestor de colas que pueda manejar estructuras de la versión 2. En todas las demás situaciones, el cliente determina la versión correcta de la estructura que se va a utilizar consultando las posibilidades del gestor de colas.

skipBytes

```
public int skipBytes(int n)
```

Emite IOException, EOFException.

Omite los próximos n bytes del almacenamiento intermedio de mensajes.

Este método se bloquea hasta que sucede algo de lo siguiente:
- Se omiten todos los bytes
- Se detecta el final del almacenamiento intermedio de mensajes
- Se emite una excepción

Devuelve el número de bytes omitidos, que siempre es n.

write

```
public void write(int b)
```

Emite IOException.

Graba un byte en el almacenamiento intermedio de mensajes en la posición actual.

write

```
public void write(byte b[])
```

Emite IOException.

Graba una matriz de bytes en el almacenamiento intermedio de mensajes en la posición actual.

write

```
public void write(byte b[],
                  int off,
                  int len)
```

Emite IOException.

Graba una serie de bytes en el almacenamiento intermedio de mensajes en la posición actual. Se graban los bytes *len*, tomados del desplazamiento *off* de la matriz b.

writeBoolean

```
public void writeBoolean(boolean v)
```

Emite IOException.

Graba una expresión booleana en el almacenamiento intermedio de mensajes en la posición actual.

writeByte

```
public void writeByte(int v)
```

Emite IOException.

Graba un byte en el almacenamiento intermedio de mensajes en la posición actual.

writeBytes

```
public void writeBytes(String s)
```

Emite IOException.

Graba la serie de caracteres en el almacenamiento intermedio de mensajes como una secuencia de bytes. Cada carácter de la serie se graba en secuencia descartando los ocho bits más significativos.

writeChar

```
public void writeChar(int v)
```

Emite IOException.

Graba un carácter Unicode en el almacenamiento intermedio de mensajes en la posición actual.

writeChars

```
public void writeChars(String s)
```

Emite IOException.

Graba una serie de caracteres como una secuencia de caracteres Unicode en el almacenamiento intermedio de mensajes en la posición actual.

writeDouble

```
public void writeDouble(double v)
```

Emite IOException.

Grabar un número de dos bytes en el almacenamiento intermedio de mensajes en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Los valores MQC.MQENC_FLOAT_IEEE_NORMAL y MQC.MQENC_FLOAT_IEEE_REVERSED graban números de coma flotante IEEE conformes a las normas en formatos big endian y little endian respectivamente.

Un valor MQC.MQENC_FLOAT_S390 graba un número de coma flotante de formato System/390®. Tenga en cuenta que el rango de números de dos bytes IEEE es mayor que el rango de números de números de coma flotante de precisión doble de S/390® y, por consiguiente, no se pueden convertir números muy grandes.

writeFloat

```
public void writeFloat(float v)
```

Emite IOException.

Graba un número de coma flotante en el almacenamiento intermedio en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Los valores de MQC.MQENC_FLOAT_IEEE_NORMAL y MQC.MQENC_FLOAT_IEEE_REVERSED graban números de coma flotante IEEE estándares en formatos big endian y little endian respectivamente.

Un valor MQC.MQENC_FLOAT_S390 graba un número de coma flotante de formato System/390®.

writeInt

```
public void writeInt(int v)
```

Emite IOException.

Graba un entero en el almacenamiento intermedio de mensajes en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_INTEGER_NORMAL graba un entero big endian, un valor MQC.MQENC_INTEGER_REVERSED graba un entero little endian.

writeInt4

```
public void writeInt4(int v)
```

Emite IOException.

Sinónimo de writeInt(), proporcionado para compatibilidad las API de MQSeries de lenguajes cruzados.

writeLong

```
public void writeLong(long v)
```

Emite IOException.

Graba un número largo en el almacenamiento intermedio de mensajes en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_INTEGER_NORMAL graba un número largo big endian, un valor MQC.MQENC_INTEGER_REVERSED graba un número largo little endian.

writeInt8

```
public void writeInt8(long v)
```

Emite IOException.

Sinónimo de writeLong(), proporcionado para compatibilidad de las API de MQSeries de lenguajes cruzados.

writeObject

```
public void writeObject(Object obj)
```

Emite IOException.

Graba el objeto especificado en el almacenamiento intermedio de mensajes. Se graban la clase del objeto, la firma de la clase y los valores de los campos no transitorios y no estáticos de la clase y de todos sus supertipos.

writeShort

```
public void writeShort(int v)
```

Emite IOException.

Graba un número corto en el almacenamiento intermedio de mensajes en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_INTEGER_NORMAL graba un número corto big endian, un valor MQC.MQENC_INTEGER_REVERSED graba un número corto little endian.

writeInt2

```
public void writeInt2(int v)
```

Emite IOException.

Sinónimo de writeShort(), proporcionado por compatibilidad de API de MQSeries de lenguajes cruzados.

writeDecimal2

```
public void writeDecimal2(short v)
```

Emite IOException.

Graba un número de 2 bytes de formato decimal empaquetado en el almacenamiento intermedio en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_DECIMAL_NORMAL graba un número decimal big endian empaquetado, un valor MQC.MQENC_DECIMAL_REVERSED graba un número decimal little endian empaquetado.

Parámetros

v puede estar en el rango de -999 a 999.

writeDecimal4

```
public void writeDecimal4(int v)
```

Emite IOException.

Graba un número de 4 bytes de formato decimal empaquetado en el almacenamiento intermedio de mensajes en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_DECIMAL_NORMAL graba un número decimal big endian empaquetado, un valor MQC.MQENC_DECIMAL_REVERSED graba un número decimal little endian empaquetado.

Parámetros

v puede estar en el rango de -9999999 a 9999999.

writeDecimal8

```
public void writeDecimal8(long v)
```

Emite IOException.

Graba un número de 8 bytes de formato decimal empaquetado en el almacenamiento intermedio de mensajes en la posición actual. El valor de la variable de miembro de codificación determina la presentación de este método.

Un valor MQC.MQENC_DECIMAL_NORMAL graba un número decimal big endian empaquetado, un valor MQC.MQENC_DECIMAL_REVERSED graba un número decimal little endian empaquetado.

Parámetros:

v puede estar en el rango de -999999999999999 a 999999999999999.

writeUTF

```
public void writeUTF(String str)
```

Emite IOException.

Graba una serie de caracteres UTF, con el prefijo de un campo de 2 bytes de longitud, en la posición actual en el almacenamiento intermedio de mensajes.

writeString

```
public void writeString(String str)
```

Emite IOException.

Graba una serie en el almacenamiento intermedio de mensajes en la posición actual, convirtiéndolo al conjunto de códigos identificado por la variable de miembro characterSet.

MQMessageTracker

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

clase pública abstracta **MQMessageTracker**
expande **Object**

Nota: Sólo puede utilizar esta clase cuando esté conectado a un gestor de colas MQSeries Versión 5 (o superior).

MQDistributionListItem (en la página 95) hereda esta clase cuando se utiliza para adaptar parámetros de mensaje para un destino específico de una lista de distribución.

Variables

feedback

```
public int feedback
```

Se utiliza con un mensaje de tipo MQC.MQMT_REPORT para indicar la naturaleza del informe. El sistema define los códigos de información de retorno siguientes:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

También se pueden utilizar valores de información de retorno definidos por aplicación en el rango de MQC.MQFB_APPL_FIRST a MQC.MQFB_APPL_LAST.

El valor por omisión de este campo es MQC.MQFB_NONE, que indica que no se proporciona ninguna información de retorno.

messageId

```
public byte messageId[]
```

Especifica el identificador de mensaje que se debe utilizar cuando se transfiere el mensaje. Si se especifica MQC.MQMI_NONE, el gestor de colas genera un identificador de mensaje exclusivo cuando se transfiere el mensaje. El valor de esta variable de miembro se actualiza después de la operación de transferencia para indicar el identificador de mensaje que se ha utilizado.

El valor por omisión es MQC.MQMI_NONE.

correlationId

```
public byte correlationId[]
```

Especifica el identificador de correlación que se debe utilizar cuando se transfiere el mensaje.

El valor por omisión es MQC.MQCI_NONE.

accountingToken

```
public byte accountingToken[]
```

Forma parte del contexto de identidad del mensaje. Permite a una aplicación hacer que el trabajo efectuado como resultado del mensaje se cargue apropiadamente.

El valor por omisión es MQC.MQACT_NONE.

groupId

```
public byte[] groupId
```

Serie de bytes que identifica el grupo de mensajes al que pertenece el mensaje físico.

El valor por omisión es MQC.MQGI_NONE.

MQPoolServices

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

clase pública **MQPoolServices**
expande **Object**

Nota: Normalmente, las aplicaciones no utilizan esta clase.

Pueden utilizar la clase MQPoolServices las implementaciones de ConnectionManager que se van a utilizar como ConnectionManager por omisión para las conexiones MQSeries.

Un ConnectionManager puede construir un objeto MQPoolServices y, a través de éste, registrar un escucha. El escucha recibe sucesos relacionados con el conjunto de MQPoolTokens que gestiona MQEnvironment. ConnectionManager puede utilizar esta información para realizar los trabajos de limpieza o arranque necesarios.

Vea también los apartados “MQPoolServicesEvent” en la página 134 y“MQPoolServicesEventListener” en la página 164.

Constructores

MQPoolServices

```
public MQPoolServices()
```

Construye un nuevo objeto MQPoolServices.

Métodos

addMQPoolServicesEventListener

```
public void addMQPoolServicesEventListener
                              (MQPoolServicesEventListener listener)
```

Añade un MQPoolServicesEventListener. El escucha recibe un suceso cada vez que se añade o elimina una señal del conjunto de MQPoolTokens que controla MQEnvironment, o cada vez que cambia el ConnectionManager por omisión.

removeMQPoolServicesEventListener

```
public void removeMQPoolServicesEventListener
                                 (MQPoolServicesEventListener listener)
```

Elimina un MQPoolServicesEventListener.

getTokenCount

```
public int getTokenCount()
```

Devuelve el número de MQPoolTokens que están registrados actualmente con MQEnvironment.

MQPoolServicesEvent

```
java.lang.Object
   |
   +- java.util.EventObject
         |
         +- com.ibm.mq.MQPoolServicesEvent
```

Nota: Normalmente, las aplicaciones no utilizan esta clase.

Se genera un MQPoolServicesEvent cada vez que se añade o se elimina un MQPoolToken del conjunto de señales que controla MQEnvironment. También se genera un suceso cuando se modifica el ConnectionManager por omisión.

Vea también los apartados “MQPoolServices” en la página 133 y“MQPoolServicesEventListener” en la página 164.

Variables

TOKEN_ADDED
`public static final int TOKEN_ADDED`

ID de suceso que se utiliza cuando se añade un MQPoolToken al conjunto.

TOKEN_REMOVED
`public static final int TOKEN_REMOVED`

ID de suceso que se utiliza cuando se elimina un MQPoolToken del conjunto.

DEFAULT_POOL_CHANGED
`public static final int DEFAULT_POOL_CHANGED`

ID de suceso que se utiliza cuando se modifica el ConnectionManager por omisión.

ID `protected int ID`

ID del suceso. Los valores válidos son:

TOKEN_ADDED
TOKEN_REMOVED
DEFAULT_POOL_CHANGED

token `protected MQPoolToken token`

La señal. Cuando el ID de suceso es DEFAULT_POOL_CHANGED, es nulo.

Constructores

MQPoolServicesEvent
`public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)`

Construye un MQPoolServicesEvent basado en el ID de suceso y la señal.

MQPoolServicesEvent
`public MQPoolServicesEvent(Object source, int eid)`

Construye un MQPoolServicesEvent basado en el ID de suceso.

Métodos

getId `public int getId()`

Obtiene el ID de suceso.

Devuelve

El ID de suceso, con uno de los valores siguientes:

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

getToken

`public MQPoolToken getToken()`

Devuelve la señal que se ha añadido o eliminado del conjunto. Si el ID de suceso es DEFAULT_POOL_CHANGED, es nulo.

MQPoolToken

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

clase pública **MQPoolToken**
expande **Object**

Un MQPoolTokenSe se puede utilizar para habilitar la agrupación de conexiones por omisión. Los MQPoolTokens se registran con la clase MQEnvironment antes de que un componente de aplicación se conecte a MQSeries. Después, cuando el componente deja de utilizar MQSeries, se anula el registro. Normalmente, el ConnectionManager por omisión está activo mientras el conjunto de MQPoolTokens registrados no está vacío.

MQPoolToken no proporciona métodos o variables. Los suministradores de ConnectionManager pueden elegir si desean expandir MQPoolToken de modo que se puedan pasar indicaciones a ConnectionManager.

Consulte los apartados “MQEnvironment.addConnectionPoolToken” en la página 101 y “MQEnvironment.removeConnectionPoolToken” en la página 102.

Constructores

MQPoolToken

```
public MQPoolToken()
```

Construye un nuevo objeto MQPoolToken.

MQProcess

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
            |
            +-- com.ibm.mq.MQProcess
```

clase pública **MQProcess**
expande **MQManagedObject**. (Consulte la página 109).

MQProcess proporciona operaciones de consulta para los procesos MQSeries.

Constructores

MQProcess

```
public MQProcess(MQQueueManager qMgr,
                                 String processName,
                                 int openOptions,
                                 String queueManagerName,
                                 String alternateUserId)
                 throws MQException
```

Accede a un proceso del gestor de colas `qMgr`. Consulte `accessProcess` en "MQQueueManager" en la página 151 para encontrar información detallada acerca de los parámetros restantes.

Métodos

getApplicationId

```
public String getApplicationId()
```

Serie de caracteres que identifica la aplicación que se debe iniciar. Esta información es para que la utilice una aplicación supervisora de activación que procese mensajes en la cola de inicio; la información se envía a la cola de inicio como parte del mensaje de activación.

Este método emite MQException si se invoca después de haber cerrado el proceso.

getApplicationType

```
public int getApplicationType()
```

Emite MQException (consulte la página 103).

Identifica la naturaleza del programa que se va a iniciar en respuesta a la recepción de un mensaje de activación. El tipo de aplicación puede tomar cualquier valor pero se recomiendan los valores siguientes para los tipos estándares:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT

- MQC.MWQAT_USER_FIRST (valor más bajo para tipo de aplicación definido por el usuario)
- MQC.MQAT_USER_LAST (valor más alto para tipo de aplicación definido por el usuario)

getEnvironmentData

```
public String getEnvironmentData()
```

Emite MQException.

Serie de caracteres que contiene información relacionada con el entorno relativa a la aplicación que se debe iniciar.

getUserData

```
public String getUserData()
```

Emite MQException.

Serie que contiene información de usuario referente a la aplicación que se debe iniciar.

close

```
public synchronized void close()
```

Emite MQException.

Alteración temporal de “MQManagedObject.close” en la página 111.

MQPutMessageOptions

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

clase pública **MQPutMessageOptions**
expande **Object**

Esta clase contiene opciones que controlan la presentación de MQQueue.put().

Nota: La presentación de algunas de las opciones disponibles en esta clase depende del entorno en el que se utilicen. Estos elementos están marcados con el símbolo *. Para obtener información más detallada, consulte el apartado “Extensiones de la versión 5 que operan en otros entornos” en la página 84.

Variables

options

```
public int options
```

Opciones que controlan la acción de MQQueue.put. Se puede especificar cualquiera o ninguno de los valores siguientes. Si se necesita más de una opción, los valores pueden añadirse juntos o combinarse utilizando el operador OR que tiene en cuenta los bits.

MQC.MQPMO_SYNCPOINT
Transfiere un mensaje con control de punto de sincronismo. El mensaje no está visible fuera de la unidad de trabajo hasta que ésta se confirma. Si se restituye la unidad de trabajo, se suprime el mensaje.

MQC.MQPMO_NO_SYNCPOINT
Transfiere un mensaje sin punto de sincronismo. Tenga en cuenta que, si no se especifica la opción de control de punto de sincronismo, se da por supuesto el valor por omisión ‘sin punto de sincronismo’, lo que se aplica a todas las plataformas para las que se ofrece soporte, incluida OS/390.

MQC.MQPMO_NO_CONTEXT
No se asocia ningún contexto con el mensaje.

MQC.MQPMO_DEFAULT_CONTEXT
Se asocia contexto por omisión con el mensaje.

MQC.MQPMO_SET_IDENTITY_CONTEXT
Establece contexto de identidad desde la aplicación.

MQC.MQPMO_SET_ALL_CONTEXT
Establece todo el contexto desde la aplicación.

MQC.MQPMO_FAIL_IF_QUIESCING
Error si el gestor de colas se está inmovilizando.

MQC.MQPMO_NEW_MSG_ID*
Genera un nuevo ID de mensaje para cada mensaje enviado.

MQC.MQPMO_NEW_CORREL_ID*
Genera un nuevo ID de correlación para cada mensaje enviado.

MQC.MQPMO_LOGICAL_ORDER*
Transfiere los segmentos y los mensajes lógicos de los grupos de mensajes en orden lógico.

MQC.MQPMO_NONE
No se ha especificado ninguna opción. No se debe utilizar junto con otras opciones.

MQC.MQPMO_PASS_IDENTITY_CONTEXT
Pasa el contexto de identidad desde un manejador de colas de entrada.

MQC.MQPMO_PASS_ALL_CONTEXT
Pasa todo el contexto desde un manejador de colas de entrada.

contextReference
`public MQQueue ContextReference`

Se trata de un campo de entrada que indica el origen de la información de contexto.

Si el campo `options` incluye MQC.MQPMO_PASS_IDENTITY_CONTEXT o MQC.MQPMO_PASS_ALL_CONTEXT, establezca este campo para que haga referencia a la MQQueue de la que debe tomarse la información de contexto.

El valor inicial de este campo es nulo.

recordFields *
`public int recordFields`

Distintivos que especifican qué campos deben personalizarse en cada cola cuando se coloca un mensaje en una lista de distribución. Se puede especificar uno o varios de los distintivos siguientes:

MQC.MQPMRF_MSG_ID
Utiliza el atributo messageId en MQDistributionListItem.

MQC.MQPMRF_CORREL_ID
Utiliza el atributo correlationId en MQDistributionListItem.

MQC.MQPMRF_GROUP_ID
Utiliza el atributo groupId en MQDistributionListItem.

MQC.MQPMRF_FEEDBACK
Utiliza el atributo feedback en MQDistributionListItem.

MQC.MQPMRF_ACCOUNTING_TOKEN
Utiliza el atributo accountingToken en MQDistributionListItem.

El valor especial MQC.MQPMRF_NONE indica que no debe personalizarse ningún campo.

resolvedQueueName
`public String resolvedQueueName`

Se trata de un campo de salida al que el gestor de colas asigna el nombre de la cola en la que se coloca el mensaje. Éste puede ser diferente del nombre utilizado para abrir la cola si la cola abierta era una cola alias o modelo.

resolvedQueueManagerName
`public String resolvedQueueManagerName`

Se trata de un campo de salida al que el gestor de colas asigna el nombre del gestor de colas propietario de la cola especificada por el nombre de cola remota. Éste puede ser diferente del nombre del gestor de colas desde el que se ha accedido a la cola si la cola es una cola remota.

knownDestCount *

```
public int knownDestCount
```

Se trata de un campo de salida que el gestor de colas establece en el número de mensajes que la invocación actual ha enviado satisfactoriamente a las colas que se convierten en colas locales. Este campo también se establece al abrir una cola individual que no forma parte de una lista de distribución.

unknownDestCount *

```
public int unknownDestCount
```

Se trata de un campo de salida que el gestor de colas establece en el número de mensajes que la invocación actual ha enviado satisfactoriamente a las colas que se convierten en colas remotas. Este campo también se establece al abrir una cola individual que no forma parte de una lista de distribución.

invalidDestCount *

```
public int invalidDestCount
```

Se trata de un campo de salida que el gestor de colas establece en el número de mensajes que no se han podido enviar a las colas de una lista de distribución. El recuento incluye las colas que no se han podido abrir así como las colas que se han abierto satisfactoriamente, pero para las que ha fallado la operación de transferir. Este campo también se establece al abrir una cola individual que no forma parte de una lista de distribución.

Constructores

MQPutMessageOptions

```
public MQPutMessageOptions()
```

Construye un nuevo objeto MQPutMessageOptions sin opciones establecidas y resolvedQueueName y resolvedQueueManagerName en blanco.

MQQueue

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueue
```

clase pública **MQQueue**
expande **MQManagedObject**. (Consulte la página 109).

MQQueue proporciona operaciones de consultar, establecer transferir y obtener para colas MQSeries. Las posibilidades de consulta y establecimiento se heredan de MQ.MQManagedObject.

Vea también “MQQueueManager.accessQueue” en la página 156.

Constructores

MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
                                 String alternateUserId)
               throws MQException
```

Accede a una cola en el gestor de colas `qMgr`.

En el apartado “MQQueueManager.accessQueue” en la página 156 se proporciona información detallada sobre los demás parámetros.

Métodos

get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions,
                             int MaxMsgSize)
```

Emite MQException.

Recupera un mensaje de la cola, hasta el tamaño máximo de mensaje especificado.

Este método toma un objeto MQMessage como parámetro. Utiliza algunos campos del objeto como parámetros de entrada, en especial, messageId y correlationId, por lo que es importante que se hayan establecido como obligatorios. (Consulte el apartado “Message” en la página 280).

Si la operación de obtener no se ejecuta correctamente, el objeto MQMessage no se modifica. Si la operación de obtener se ejecuta correctamente, el descriptor de mensaje (variables de miembro) y las partes de datos de mensaje de MQMessage se sustituyen completamente por el descriptor de mensaje y los datos del mensaje entrante.

Tenga en cuenta que todas las invocaciones a MQSeries desde un MQQueueManager determinado son síncronas. Por consiguiente, si efectúa una operación de obtener con espera, todas las demás hebras que utilizan el mismo MQQueueManager quedan bloqueadas para realizar invocaciones

MQSeries adicionales hasta que se completa la operación de obtener. Si necesita que varias hebras accedan a MQSeries simultáneamente, cada una de ellas debe crear se propio objeto MQQueueManager.

Parámetros

message
Parámetro de entrada/salida que contiene la información del descriptor de mensaje y los datos de los mensajes devueltos.

getMessageOptions
Opciones que controlan la acción de la operación de obtener. (Consulte el apartado "MQGetMessageOptions" en la página 105).

MaxMsgSize
El mensaje más largo que podrá recibir esta invocación. Si el mensaje de la cola tiene una longitud mayor que este tamaño, puede ocurrir una de estas dos cosas:

1. Si el distintivo MQC.MQGMO_ACCEPT_TRUNCATED_MSG se establece en la variable de miembro de opciones del objeto MQGetMessageOptions, el mensaje se rellena con tantos datos de mensaje como quepan en el tamaño de almacenamiento intermedio especificado y se emite una excepción con el código de terminación MQException.MQCC_WARNING y el código de razón MQException.MQRC_TRUNCATED_MSG_ACCEPTED.
2. Si no se establece el distintivo MQC.MQGMO_ACCEPT_TRUNCATED_MSG se deja el mensaje en la cola y se presenta una MQException con el código de terminación MQException.MQCC_WARNING y el código de razón MQException.MQRC_TRUNCATED_MSG_FAILED.

Emite MQException si la operación de obtener no se ejecuta correctamente.

get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions)
```

Emite MQException.

Recupera un mensaje de la cola, independientemente del tamaño del mensaje. Para mensajes grandes, puede que el método de obtención tenga que emitir dos invocaciones a MQSeries en nombre del usuario, una para establecer el tamaño de almacenamiento intermedio necesario y otra para obtener los datos del mensaje en sí.

Este método toma un objeto MQMessage como parámetro. Utiliza algunos campos del objeto como parámetros de entrada, en especial, messageId y correlationId, por lo que es importante que se hayan establecido como obligatorios. (Consulte el apartado "Message" en la página 280).

Si la operación de obtener no se ejecuta correctamente, el objeto MQMessage no se modifica. Si la operación de obtener se ejecuta correctamente, las partes correspondientes al descriptor de mensaje (variables de miembro) y a los datos de mensaje del MQMessage se sustituyen completamente por el descriptor de mensaje y los datos de mensaje del mensaje entrante.

Tenga en cuenta que todas las invocaciones a MQSeries desde un MQQueueManager determinado son síncronas. Por consiguiente, si efectúa una operación de obtener con espera, todas las demás hebras que utilizan el mismo MQQueueManager quedan bloqueadas para realizar invocaciones MQSeries adicionales hasta que se completa la operación de obtener. Si necesita que varias hebras accedan a MQSeries simultáneamente, cada una de ellas debe crear su propio objeto MQQueueManager.

Parámetros

message
Parámetro de entrada/salida que contiene la información del descriptor de mensaje y los datos de los mensajes devueltos.

getMessageOptions
Opciones que controlan la acción de la operación de obtener. (En el apartado "MQGetMessageOptions" en la página 105 se proporciona información detallada).

Emite MQException si la operación de obtener no se ejecuta correctamente.

get

```
public synchronized void get(MQMessage message)
```

Se trata de una versión simplificada del método de obtención descrito anteriormente.

Parámetros

MQMessage
Parámetro de entrada/salida que contiene la información del descriptor de mensaje y los datos de los mensajes devueltos.

Este método utiliza una instancia por omisión de MQGetMessageOptions para efectuar la operación de obtener. La opción de mensaje utilizada es MQGMO_NOWAIT.

put

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions)
```

Emite MQException.

Coloca un mensaje en la cola.

Este método toma un objeto MQMessage como parámetro. Las propiedades del descriptor de mensaje de este objeto pueden modificarse como resultado de este método. Los valores que tienen inmediatamente después de haber completado este método son los valores que se transfirieron a la cola de MQSeries.

Las modificaciones efectuadas en el objeto MQMessage después de finalizar la transferencia no afectan al mensaje real de la cola de MQSeries.

Una operación de transferir actualiza el messageId y el correlationId, por lo que debe tenerse en consideración al efectuar invocaciones adicionales para realizar operaciones de transferir/obtener utilizando el mismo objeto

MQMessage. Además, la invocación a una operación de transferir no borra los datos del mensaje y, por consiguiente:

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

transfiere dos mensajes. El primero contiene ″a″ y el segundo ″ab″.

Parámetros

message
Almacenamiento intermedio de mensajes que contiene los datos del descriptor de mensaje y el mensaje que se debe enviar.

putMessageOptions
Opciones que controlan la acción de la operación de transferir. (Consulte el apartado "MQPutMessageOptions" en la página 139)

Emite MQException si no se ejecuta correctamente la transferencia.

put

```
public synchronized void put(MQMessage message)
```

Se trata de una versión simplificada del método de transferencia descrito anteriormente.

Parámetros

MQMessage
Almacenamiento intermedio de mensajes que contiene los datos del descriptor de mensaje y el mensaje que se debe enviar.

Este método utiliza una instancia por omisión de MQPutMessageOptions para efectuar la operación de transferir.

Nota: Todos los métodos siguientes emiten MQException si se invoca el método después de cerrar la cola.

getCreationDateTime

```
public GregorianCalendar getCreationDateTime()
```

Emite MQException.

Fecha y hora en que se ha creado esta cola.

getQueueType

```
public int getQueueType()
```

Emite MQException

Devuelve
El tipo de esta cola con uno de los valores siguientes:
- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

getCurrentDepth

```
public int getCurrentDepth()
```

Emite MQException.

Obtiene el número de mensajes que hay actualmente en la cola. Este valor se incrementa durante una invocación de transferencia y durante la restitución de una invocación de obtención. Disminuye durante una operación de obtener sin examinar y durante la restitución de una invocación de transferencia.

getDefinitionType

```
public int getDefinitionType()
```

Emite MQException.

Indica cómo se ha definido la cola.

Devuelve

Uno de los siguientes:
- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

getMaximumDepth

```
public int getMaximumDepth()
```

Emite MQException.

Número máximo de mensajes que pueden existir en la cola en cualquier momento. Un intento de transferencia de un mensaje a una cola que ya contiene todos estos mensajes falla con el código de razón MQException.MQRC_Q_FULL.

getMaximumMessageLength

```
public int getMaximumMessageLength()
```

Emite MQException.

Se trata de la longitud máxima de los datos de aplicación que pueden existir en cada mensaje de esta cola. Un intento de transferencia de un mensaje de mayor longitud que este valor fallará con el código de razón MQException.MQRC_MSG_TOO_BIG_FOR_Q.

getOpenInputCount

```
public int getOpenInputCount()
```

Emite MQException.

Número de manejadores actualmente válidos para eliminar mensajes de la cola. Es el número *total* de los manejadores que conoce el gestor de colas local, no sólo los que ha creado MQSeries Classes for Java (utilizando accessQueue).

getOpenOutputCount

```
public int getOpenOutputCount()
```

Emite MQException.

Número de manejadores actualmente válidos para añadir mensajes a la cola. Es el número *total* de los manejadores que conoce el gestor de colas local, no sólo los que ha creado MQSeries Classes for Java (utilizando accessQueue).

getShareability

```
public int getShareability()
```

Emite MQException.

Indica si se puede abrir la cola para entrada múltiples veces.

Devuelve

Uno de los siguientes:
- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

getInhibitPut

```
public int getInhibitPut()
```

Emite MQException.

Indica si se permiten o no operaciones de transferir para esta cola.

Devuelve

Uno de los siguientes:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

setInhibitPut

```
public void setInhibitPut(int inhibit)
```

Emite MQException.

Controla si se permiten o no operaciones de transferir para esta cola. Los valores permitidos son:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

getInhibitGet

```
public int getInhibitGet()
```

Emite MQException.

Indica si se permiten o no operaciones de obtener para esta cola.

Devuelve

Los valores posibles son:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

setInhibitGet

```
public void setInhibitGet(int inhibit)
```

Emite MQException.

Controla si se permiten o no operaciones de obtener para esta cola. Los valores permitidos son:
- MQC.MQQA_GET_INHIBITED

- MQC.MQQA_GET_ALLOWED

getTriggerControl

```
public int getTriggerControl()
```

Emite MQException.

Indica si los mensajes de activación se graban o no en una cola de inicio, para hacer que se inicie una aplicación con el fin de dar servicio a la cola.

Devuelve

Los valores posibles son:
- MQC.MQTC_OFF
- MQC.MQTC_ON

setTriggerControl

```
public void setTriggerControl(int trigger)
```

Emite MQException.

Controla si los mensajes de activación se graban o no en una cola de inicio, para hacer que se inicie una aplicación con el fin de dar servicio a la cola. Los valores permitidos son:
- MQC.MQTC_OFF
- MQC.MQTC_ON

getTriggerData

```
public String getTriggerData()
```

Emite MQException.

Datos de formato libre que el gestor de colas inserta en el mensaje de activación cuando un mensaje que llega a esta cola hace que se grabe un mensaje de activación en la cola de inicio.

setTriggerData

```
public void setTriggerData(String data)
```

Emite MQException.

Establece los datos de formato libre que el gestor de colas inserta en el mensaje de activación cuando un mensaje que llega a esta cola hace que se grabe un mensaje de activación en la cola de inicio. La longitud máxima permitida de la serie de caracteres la proporciona MQC.MQ_TRIGGER_DATA_LENGTH.

getTriggerDepth

```
public int getTriggerDepth()
```

Emite MQException.

Número de mensajes que tienen que estar en la cola antes de que se grabe un mensaje de activación cuando el tipo activación se ha establecido en MQC.MQTT_DEPTH.

setTriggerDepth

```
public void setTriggerDepth(int depth)
```

Emite MQException.

Establece el número de mensajes que tienen que estar en la cola antes de que se grabe un mensaje de activación cuando el tipo de activación se ha establecido en MQC.MQTT_DEPTH.

getTriggerMessagePriority

```
public int getTriggerMessagePriority()
```

Emite MQException.

Ésta es la prioridad del mensaje por debajo de la cual los mensajes no contribuyen a la generación de mensajes de activación (es decir, el gestor de colas ignora estos mensajes al decidir si se debe generar una activación). Un valor de cero hace que todos los mensajes contribuyan a la generación de mensajes de activación.

setTriggerMessagePriority

```
public void setTriggerMessagePriority(int priority)
```

Emite MQException.

Establece la prioridad del mensaje por debajo de la cual los mensajes no contribuyen a la generación de mensajes de activación (es decir, el gestor de colas ignora estos mensajes al decidir si se debe generar una activación). Un valor de cero hace que todos los mensajes contribuyan a la generación de mensajes de activación.

getTriggerType

```
public int getTriggerType()
```

Emite MQException.

Condiciones bajo las cuales se graban mensajes de activación como resultado de la llegada de mensajes a esta cola.

Devuelve

Los valores posibles son:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

setTriggerType

```
public void setTriggerType(int type)
```

Emite MQException.

Establece las condiciones bajo las cuales se graban mensajes de activación como resultado de la llegada de mensajes a esta cola. Los valores posibles son:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

close

```
public synchronized void close()
```

Emite MQException.

Alteración temporal de “MQManagedObject.close” en la página 111.

MQQueueManager

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueueManager
```

clase pública **MQQueueManager**
expande **MQManagedObject**. (Consulte la página 109).

Nota: La presentación de algunas de las opciones disponibles en esta clase depende del entorno en el que se utilicen. Estos elementos están marcados con el símbolo *. Para obtener información más detallada, consulte el "Capítulo 8. Presentación dependiente del entorno" en la página 81.

Variables

isConnected

```
public boolean isConnected
```

Verdadero si la conexión con el gestor de colas aún está abierta.

Constructores

MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

Emite MQException.

Crea una conexión con el gestor de colas mencionado.

Nota: Cuando se utiliza MQSeries Classes for Java, el nombre del sistema principal, el nombre del canal y el puerto que se deben utilizar durante la petición de conexión se especifican en la clase MQEnvironment. Debe hacerse *antes* de invocar este constructor.

En el ejemplo siguiente se muestra una conexión a un gestor de colas "MYQM", que se ejecuta en una máquina con el nombre de sistema principal fred.mq.com.

```
MQEnvironment.hostname = "fred.mq.com";  // sistema principal al que conectarse
MQEnvironment.port     = 1414;           // puerto al que conectarse
                                         // Si no establezco esto,
                                         // toma por omisión 1414
                                         // (el puerto de MQSeries por omisión)
MQEnvironment.channel  = "channel.name"; // nombre SENSIBLE A LAS MAYÚSCULAS
                                         // Y MINÚSCULAS del
                                         // canal SVR CONN
                                         // del gestor de colas
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

Si el nombre del gestor de colas se deja en blanco (nulo o ""), se realiza una conexión al gestor de colas por omisión.

Vea también "MQEnvironment" en la página 97.

MQQueueManager

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager cxManager)
```

Emite MQException.

Este constructor conecta al gestor de colas especificado, utilizando las propiedades de MQEnvironment. El MQConnectionManager especificado gestiona la conexión.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager cxManager)
```

Emite MQException.

Este constructor conecta al gestor de colas especificado, utilizando las propiedades de MQEnvironment. El ConnectionManager especificado gestiona la conexión.

Este método necesita JVM con Java™ 2 v1.3 o posterior, con JAAS 1.0 o posterior instalado.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options)
```

Emite MQException.

Esta versión del constructor está destinada a utilizarse solamente en modalidad de enlaces y utiliza la API de conexión ampliada (MQCONNX) para conectarse al gestor de colas. El parámetro *options* permite elegir enlaces rápidos o normales. Los valores posibles son los siguientes:

- MQC.MQCNO_FASTPATH_BINDING para enlaces rápidos *.
- MQC.MQCNO_STANDARD_BINDING para enlaces normales.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      MQConnectionManager cxManager)
```

Emite MQException.

Este constructor realiza una MQCONNX, pasándole las opciones suministradas. El MQConnectionManager especificado gestiona la conexión.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

Emite MQException.

Este constructor realiza una MQCONNX, pasándole las opciones suministradas. El ConnectionManager especificado gestiona la conexión.

Este método necesita Java™ 2 v1.3 o posterior, con JAAS 1.0 o posterior instalado.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      java.util.Hashtable properties)
```

El parámetro properties toma una serie de pares clave/valor que describen el entorno de MQSeries para este gestor de colas en concreto. Estas propiedades, si se especifican, alteran temporalmente los valores establecidos por la clase MQEnvironment y permiten establecer las propiedades individuales en cada uno de los gestores de colas. Consulte el apartado ""MQEnvironment.properties"" en la página 99.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      MQConnectionManager cxManager)
```

Emite MQException.

Este constructor conecta con el gestor de colas especificado, utilizando la tabla de propiedades de totales de control proporcionada para alterar temporalmente las de MQEnvironment. El MQConnectionManager especificado gestiona la conexión.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      ConnectionManager cxManager)
```

Emite MQException.

Este constructor conecta con el gestor de colas especificado, utilizando la tabla de propiedades de totales de control proporcionada para alterar temporalmente las de MQEnvironment. El ConnectionManager especificado gestiona la conexión.

Este método necesita JVM con Java™ 2 v1.3 o posterior, con JAAS 1.0 o posterior instalado.

Métodos

getCharacterSet

```
public int getCharacterSet()
```

Emite MQException.

Devuelve el CCSID (Identificador de juego de caracteres codificado) del juego de códigos del gestor de colas. Define el juego de caracteres utilizado por el gestor de colas para todos los campos de series de caracteres de la interfaz de programación de aplicaciones.

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

getMaximumMessageLength

```
public int getMaximumMessageLength()
```

Emite MQException.

Devuelve la longitud máxima de un mensaje (en bytes) que puede manejar el gestor de colas. No se puede definir ninguna cola con una longitud máxima de mensajes mayor que este valor.

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

getCommandLevel

```
public int getCommandLevel()
```

Emite MQException.

Indica el nivel de mandatos de control del sistema soportado por el gestor de colas. El conjunto de mandatos de control del sistema que corresponde a un nivel de mandato determinado varía de acuerdo con la arquitectura de la plataforma en la que se está ejecutando el gestor de colas. Para obtener información más detallada, consulte la documentación de MQSeries de la plataforma correspondiente.

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

Devuelve

Una de las constantes MQC.MQCMDL_LEVEL_xxx

getCommandInputQueueName

```
public String getCommandInputQueueName()
```

Emite MQException.

Devuelve el nombre de la cola de entrada de mandatos definida en el gestor de colas. Ésta es una cola a la que las aplicaciones pueden enviar mandatos, si están autorizadas a hacerlo.

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

getMaximumPriority

```
public int getMaximumPriority()
```

Emite MQException.

Devuelve la prioridad máxima de mensajes soportada por el gestor de colas. Las prioridades están en un rango de cero (más baja) a este valor.

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

getSyncpointAvailability

```
public int getSyncpointAvailability()
```

Emite MQException.

Indica si el gestor de colas soporta unidades de trabajo y comprobación por puntos de sincronismo con los métodos MQQueue.get y MQQueue.put.

Devuelve
- MQC.MQSP_AVAILABLE si está disponible la comprobación por puntos de sincronismo.
- MQC.MQSP_NOT_AVAILABLE si no está disponible la comprobación por puntos de sincronismo.

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

getDistributionListCapable

```
public boolean getDistributionListCapable()
```

Indica si el gestor de colas ofrece soporte para listas de distribución.

disconnect

```
public synchronized void disconnect()
```

Emite MQException.

Termina la conexión con el gestor de colas. Todas las colas abiertas y los procesos a los que accede este gestor de colas se cierran y, por tanto, quedan inutilizables. Cuando se haya desconectado de un gestor de colas, el único modo de volverse a conectar es creando un objeto MQQueueManager nuevo.

Normalmente, se confirman todos los trabajos realizados como parte de una unidad de trabajo. Sin embargo, si un ConnectionManager gestiona esta conexión, en lugar de un MQConnectionManager, la unidad de trabajo se puede restituir.

commit

```
public synchronized void commit()
```

Emite MQException.

La invocación de este método indica al gestor de colas que la aplicación ha alcanzado un punto de sincronismo y que la totalidad de las operaciones de obtener y transferir del mensaje que se han producido desde el último punto de sincronismo deben hacerse permanentes. Los mensajes transferidos como parte de una unidad de trabajo (con el distintivo MQC.MQPMO_SYNCPOINT establecido en el campo de opciones de MQPutMessageOptions) quedan disponibles para otras aplicaciones. Los mensajes recuperados como parte de una unidad de trabajo (con el distintivo MQC.MQGMO_SYNCPOINT establecido en el campo de opciones de MQGetMessageOptions) se suprimen.

Vea también la descripción siguiente de ″backout″.

backout

```
public synchronized void backout()
```

Emite MQException.

La invocación de este método indica al gestor de colas que todas las operaciones de obtener y transferir del mensaje que se han producido desde el último punto de sincronismo deben restituirse. Los mensajes transferidos como parte de una unidad de trabajo (con el distintivo

MQC.MQPMO_SYNCPOINT establecido en el campo de opciones de MQPutMessageOptions) se suprimen; los mensajes recuperados como parte de una unidad de trabajo (con el distintivo MQC.MQGMO_SYNCPOINT establecido en el campo de opciones de MQGetMessageOptions) se vuelven a colocar en la cola.

Vea también la descripción anterior de ″commit″.

accessQueue

```
public synchronized MQQueue accessQueue
                          (
                          String queueName, int openOptions,
                          String queueManagerName,
                          String dynamicQueueName,
                          String alternateUserId
                          )
```

Emite MQException.

Establece el acceso a una cola de MQSeries en este gestor de colas para obtener o examinar mensajes, transferir mensajes, realizar consultas acerca de los atributos de la cola o establecer los atributos de la cola.

Si la cola nombrada es una cola modelo, se creará una cola local dinámica. El nombre de la cola creada puede determinarse inspeccionando el atributo `name` del objeto MQQueue devuelto.

Parámetros

queueName
Nombre de la cola que se debe abrir.

openOptions
Opciones que controlan la apertura de la cola. Las opciones válidas son:

MQC.MQOO_BROWSE
Abrir para examinar mensaje.

MQC.MQOO_INPUT_AS_Q_DEF
Abrir para obtener mensajes utilizando el valor por omisión definido por la cola.

MQC.MQOO_INPUT_SHARED
Abrir para obtener mensajes con acceso compartido.

MQC.MQOO_INPUT_EXCLUSIVE
Abrir para obtener mensajes con acceso exclusivo.

MQC.MQOO_OUTPUT
Abrir para transferir mensajes.

MQC.MQOO_INQUIRE
Abrir para consulta - necesario si desea consultar las propiedades.

MQC.MQOO_SET
Abrir para establecer atributos.

MQC.MQOO_SAVE_ALL_CONTEXT
Guardar contexto al recuperar mensaje*.

MQC.MQOO_SET_IDENTITY_CONTEXT
Permite establecer contexto de identidad.

MQC.MQOO_SET_ALL_CONTEXT
Permite establecer todo el contexto.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validar con el identificador de usuario especificado.

MQC.MQOO_FAIL_IF_QUIESCING
Error si el gestor de colas se está inmovilizando.

MQC.MQOO_BIND_AS_QDEF
Utiliza el enlace por omisión para la cola.

MQC.MQOO_BIND_ON_OPEN
Enlaza un manejador al destino cuando se abre la cola.

MQC.MQOO_BIND_NOT_FIXED
No enlazar a ningún destino específico.

MQC.MQOO_PASS_ALL_CONTEXT
Permitir que se pueda pasar todo el contexto.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Permitir que se pueda pasar el contexto de identidad.

Si se necesita más de una opción, los valores pueden añadirse juntos o combinarse utilizando el operador OR que tiene en cuenta los bits. Consulte la publicación MQSeries *MQSeries® Application Programming Reference* para obtener una descripción más completa de estas opciones.

queueManagerName
Nombre del gestor de colas en el que se define la cola. Un nombre totalmente en blanco o nulo indica el gestor de colas al que se ha conectado este objeto MQQueueManager.

dynamicQueueName
Este parámetro se ignora a no ser que queueName especifique el nombre de una cola modelo. Si lo especifica, este parámetro especifica el nombre de la cola dinámica que se debe crear. No es válido un nombre en blanco o nulo si queueName especifica el nombre de una cola modelo. Si el último carácter que no sea un blanco del nombre es un asterisco (*), el gestor de colas sustituye el asterisco por una serie de caracteres que garantiza que el nombre generado para la cola es exclusivo de este gestor de colas.

alternateUserId
Si se especifica MQOO_ALTERNATE_USER_AUTHORITY en el parámetro openOptions, este parámetro especifica el identificador de usuario alternativo que se debe utilizar para comprobar la autorización de la apertura. Si no se especifica MQOO_ALTERNATE_USER_AUTHORITY, este parámetro puede dejarse en blanco (o especificarse como nulo).

Devuelve
MQQueue que se ha abierto correctamente.

Emite MQException si no se ejecuta correctamente la apertura.

Vea también “"accessProcess"” en la página 158.

accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName,
                        int openOptions
                        )
```

Emite MQException si se invoca este método después de desconectarse del gestor de colas.

Parámetros

queueName
Nombre de la cola que se debe abrir

openOptions
Opciones que controlan la apertura de la cola

En "MQQueueManager.accessQueue" en la página 156 encontrará información detallada de los parámetros.

queueManagerName, *dynamicQueueName* y *alternateUserId* están establecidos en "".

accessProcess

```
public synchronized MQProcess accessProcess
                       (
                       String processName,
                       int openOptions,
                       String queueManagerName,
                       String alternateUserId
                       )
```

Emite MQException.

Establece acceso a un proceso de MQSeries en este gestor de colas para realizar consultas acerca de los atributos de proceso.

Parámetros

processName
Nombre del proceso que se debe abrir.

openOptions
Opciones que controlan la apertura del proceso. La consulta se añade automáticamente a las opciones especificadas, de modo que no es necesario especificarla explícitamente.

Las opciones válidas son las siguientes:

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validar con el ID de usuario especificado

MQC.MQOO_FAIL_IF_QUIESCING
Error si el gestor de colas se está inmovilizando

Si se necesita más de una opción, los valores pueden añadirse juntos o combinarse utilizando el operador OR que tiene en cuenta los bits. Consulte la publicación *MQSeries® Application Programming Reference* para obtener una descripción más completa de estas opciones.

queueManagerName
Nombre del gestor de colas en el que se define el proceso. Las aplicaciones deben dejar este parámetro en blanco o especificarlo como nulo.

alternateUserId
Si se especifica MQOO_ALTERNATE_USER_AUTHORITY en el parámetro openOptions, este parámetro especifica el identificador de usuario alternativo que se debe utilizar para comprobar la autorización de la apertura. Si no se especifica MQOO_ALTERNATE_USER_AUTHORITY, este parámetro puede dejarse en blanco (o especificarse como nulo).

Devuelve
MQProcess que se ha abierto satisfactoriamente.

Emite MQException si no se ejecuta correctamente la apertura.

Vea también "MQQueueManager.accessQueue" en la página 156.

accessProcess
Se trata de una versión simplificada del método AccessProcess descrito anteriormente.

```
public synchronized MQProcess accessProcess
                      (
                     String processName,
                     int openOptions
                     )
```

Se trata de una versión simplificada del método AccessQueue descrito anteriormente.

Parámetros

processName
Nombre del proceso que se debe abrir.

openOptions
Opciones que controlan la apertura del proceso.

Vea ""accessProcess"" en la página 158 para encontrar información detallada de las opciones.

queueManagerName y *alternateUserId* están establecidos en "".

accessDistributionList

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems, int openOptions,
           String alternateUserId
          )
```

Emite MQException.

Parámetros

litems Elementos que se deben incluir en la lista de distribución.

openOptions
Opciones que controlan la apertura de la lista de distribución.

alternateUserId
Si se especifica MQOO_ALTERNATE_USER_AUTHORITY en el parámetro openOptions, este parámetro especifica el identificador de usuario alternativo que se debe utilizar para comprobar la autorización de la apertura. Si no se especifica MQOO_ALTERNATE_USER_AUTHORITY, este parámetro puede dejarse en blanco (o especificarse como nulo).

Devuelve
Una MQDistributionList recién creada que está abierta y preparada para operaciones de transferir.

Emite MQException si no se ejecuta correctamente la apertura.

Vea también "MQQueueManager.accessQueue" en la página 156.

accessDistributionList
Se trata de una versión simplificada del método AccessDistributionList descrito anteriormente.

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems,
           int openOptions,
          )
```

Parámetros

litems Elementos que se deben incluir en la lista de distribución.

openOptions
Opciones que controlan la apertura de la lista de distribución.

En "accessDistributionList" en la página 159 encontrará información detallada de los parámetros.

alternateUserId está establecido en "".

begin* (sólo conexión de enlaces)

```
public synchronized void begin()
```

Emite MQException.

Sólo ofrece soporten para este método MQSeries Classes for Java en la modalidad de enlaces y señala al gestor de colas que está iniciando una nueva unidad de trabajo.

Este método no se debe utilizar para las aplicaciones que utilizan transacciones locales de una fase.

isConnected

```
public boolean isConnected()
```

Devuelve el valor de la variable isConnected.

MQSimpleConnectionManager

```
java.lang.Object      com.ibm.mq.MQConnectionManager
   |                          |
   +- com.ibm.mq.MQSimpleConnectionManager
```

clase pública **MQSimpleConnectionManager**
implementa **MQConnectionManager** (Consulte la página 165).

MQSimpleConnectionManager proporciona la función básica de agrupación de conexiones. Puede utilizar MQSimpleConnectionManager como gestor de conexiones por omisión o como parámetro para un constructor MQQueueManager. Cuando se construye un MQQueueManager, se utiliza la conexión de la agrupación que se ha utilizado más recientemente.

Las conexiones se destruyen (mediante una hebra separada) cuando no se han utilizando durante un período de tiempo especificado o cuando el número de conexiones no utilizadas de la agrupación excede el número especificado. Puede especificar el período de tiempo de espera y el número máximo de conexiones no utilizadas.

Variables

MODE_AUTO
`public static final int MODE_AUTO`. Consulte "setActive".

MODE_ACTIVE
`public static final int MODE_ACTIVE`. Consulte "setActive".

MODE_INACTIVE
`public static final int MODE_INACTIVE`. Consulte "setActive".

Constructores

MQSimpleConnectionManager
`public MQSimpleConnectionManager()`

Construye un MQSimpleConnectionManager.

Métodos

setActive
`public void setActive(int mode)`

Establece la modalidad activa de la agrupación de conexiones.

Parámetros

modalidad
La modalidad activa necesaria de la agrupación de conexiones. Los valores válidos son:

MODE_AUTO
La agrupación de conexiones está activa mientras el gestor de conexiones es el gestor de conexiones por omisión y MQEnvironment mantiene, como mínimo, una señal del conjunto de MQPoolTokens. Ésta es la modalidad por omisión.

MODE_ACTIVE
La agrupación de conexiones siempre está activa. Cuando se invoca a MQQueueManager.disconnect(), se agrupa la conexión subyacente y es posible que se vuelva a utilizar la próxima vez que se construya un objeto MQQueueManager. Una hebra separada destruye las conexiones si no se utilizan después del período de tiempo de espera o si el tamaño de la agrupación excede HighThreshold.

MODE_INACTIVE
La agrupación de conexiones siempre está inactiva. Cuando se especifica esta modalidad, se borra la agrupación de conexiones de MQSeries. Cuando se invoca a MQQueueManager.disconnect(), se destruye la conexión que subyace a cualquier objeto MQQueueManager activo.

getActive

```
public int getActive()
```

Obtiene la modalidad de la agrupación de conexiones.

Devuelve

La modalidad activa actual de la agrupación de conexiones, con uno de los valores siguientes (consulte "setActive" en la página 161):

MODE_AUTO

MODE_ACTIVE

MODE_INACTIVE

setTimeout

```
public void setTimeout(long timeout)
```

Establece el valor de tiempo de espera, cuando una hebra separada destruye las conexiones que no se han utilizado durante ese tiempo.

Parámetros

timeout
Valor de tiempo de espera en milisegundos.

getTimeout

```
public long getTimeout()
```

Devuelve el valor de tiempo de espera.

setHighThreshold

```
public void setHighThreshold(int threshold)
```

Establece el HighThreshold. Si el número de conexiones no utilizadas de la agrupación excede este valor, se destruye la conexión más antigua no utilizada de la agrupación.

Parámetros

threshold
Número máximo de conexiones no utilizadas de la agrupación.

getHighThreshold

```
public int getHighThreshold ()
```

Devuelve el valor de HighThreshold.

MQC

interfaz pública **MQC**
expande **Object**

La interfaz MQC define todas las constantes que utiliza la interfaz de programación MQ Java (excepto para las constantes de código de error y código de terminación). Para consultar una de estas constantes desde dentro de los programas, ponga el prefijo ″MQC″ delante del nombre de la constante. Por ejemplo, puede establecer las opciones de cierre para una cola del modo indicado a continuación:

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // suprimir la
                                      // cola cuando
                                      // esté cerrada
 ...
```

En la publicación *MQSeries® Application Programming Reference* se proporciona una descripción completa de estas constantes.

En la clase MQException se definen las constantes de código de error y código de terminación. Consulte el apartado “MQException” en la página 103.

MQPoolServicesEventListener

interfaz pública **MQPoolServicesEventListener**
expande **Object**

Nota: Normalmente, las aplicaciones no utilizan esta interfaz.

MQPoolServicesEventListener se utiliza cuando los suministradores implementan ConnectionManagers por omisión. Cuando se registra un MQPoolServicesEventListener con un objeto MQPoolServices, el escucha de sucesos recibe un suceso cada vez que se añade o elimina un MQPoolToken del conjunto de MQPoolTokens que gestiona MQEnvironment. También recibe un suceso cada vez que se modifica el ConnectionManager por omisión.

Vea también los apartados “MQPoolServices” en la página 133 y “MQPoolServicesEvent” en la página 134.

Métodos

tokenAdded

```
public void tokenAdded(MQPoolServicesEvent event)
```

Se invoca cuando se añade un MQPoolToken al conjunto.

tokenRemoved

```
public void tokenRemoved(MQPoolServicesEvent event)
```

Se invoca cuando se elimina un MQPoolToken del conjunto.

defaultConnectionManagerChanged

```
public void defaultConnectionManagerChanged(MQPoolServicesEvent event)
```

Se invoca cuando se establece el ConnectionManager por omisión. Se borra el conjunto de MQPoolTokens.

MQConnectionManager

Interfaz privada que no pueden implementar las aplicaciones. MQSeries Classes for Java proporciona una implementación de esta interfaz (MQSimpleConnectionManager), que se puede especificar en el constructor MQQueueManager o a través de MQEnvironment.setDefaultConnectionManager.

Consulte el apartado “MQSimpleConnectionManager” en la página 161.

Las aplicaciones o el middleware que deseen facilitar un ConnectionManager propio deben implementar javax.resource.spi.ConnectionManager. Para ello se debe haber instalado Java™ 2 v1.3 con JAAS 1.0.

MQReceiveExit

interfaz pública **MQReceiveExit**
expande **Object**

La interfaz de rutina de salida de recepción permite examinar y, posiblemente, modificar los datos que MQSeries Classes for Java recibe del gestor de colas.

Nota: Esta interfaz no es aplicable para conexiones directas a MQSeries en modalidad de enlaces.

Para proporcionar su propia rutina de salida de recepción, defina una clase que implemente esta interfaz. Cree una nueva instancia de la clase y asígnele la variable MQEnvironment.receiveExit antes de construir el objeto MQQueueManager. Por ejemplo:

```
// en la clase MyReceiveExit.java
class MyReceiveExit implements MQReceiveExit {
  // debe proporcionar una implementación
  // del método receiveExit
  public byte[] receiveExit(
   MQChannelExit       channelExitParms,
   MQChannelDefinition channelDefinition,
   byte[]              agentBuffer)
  {
    // el código de rutina de salida va aquí...
  }
}
// en el programa principal...
MQEnvironment.receiveExit = new MyReceiveExit();
 ...    // otra inicialización
MQQueueManager qMgr        = new MQQueueManager("");
```

Métodos

receiveExit

```
public abstract byte[] receiveExit(MQChannelExit channelExitParms,
                                   MQChannelDefinition channelDefinition,
                                    byte agentBuffer[])
```

Método de rutina de salida de recepción que debe proporcionar la clase. Este método se invoca cada vez que MQSeries Classes for Java recibe datos del gestor de colas.

Parámetros

channelExitParms
Contiene información referente al contexto en el que se está invocando la rutina de salida. La variable de miembro exitResponse es un parámetro de salida que se utiliza para indicar a MQSeries Classes for Java la acción que se debe realizar a continuación. Para obtener información más detallada, consulte "MQChannelExit" en la página 90.

channelDefinition
Contiene detalles del canal a través del cual tendrán lugar todas las comunicaciones con el gestor de colas.

agentBuffer
Si channelExitParms.exitReason es MQChannelExit.MQXR_XMIT, agentBuffer contiene los datos recibidos del gestor de colas; de lo contrario agentBuffer es nulo.

Devuelve

Si el código de respuesta de rutina de salida (en channelExitParms) se establece de modo que MQSeries Classes for Java pueda procesar los datos (MQXCC_OK), el método de rutina de salida de recepción debe devolver los datos que se deben procesar. Por consiguiente, la rutina de salida de recepción más simple consta de la única línea ″return agentBuffer;″.

Vea también:

- "MQC" en la página 163
- "MQChannelDefinition" en la página 88

MQSecurityExit

interfaz pública **MQSecurityExit**
expande **Object**

La interfaz de rutina de salida de seguridad le permite personalizar los flujos de seguridad que se producen cuando se efectúa un intento de conexión a un gestor de colas.

Nota: Esta interfaz no es aplicable para conexiones directas a MQSeries en modalidad de enlaces.

Para proporcionar su propia rutina de salida de seguridad, defina una clase que implemente esta interfaz. Cree una instancia nueva de la clase y asígnele la variable MQEnvironment.securityExit antes de construir el objeto MQQueueManager. Por ejemplo:

```
// en la clase MySecurityExit.java
class MySecurityExit implements MQSecurityExit {
  // debe proporcionar una implementación
  // del método securityExit
  public byte[] securityExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // el código de rutina de salida va aquí...
  }
}
// en el programa principal...
MQEnvironment.securityExit = new MySecurityExit();
 ...    // otra inicialización
MQQueueManager qMgr        = new MQQueueManager("");
```

Métodos

securityExit

```
public abstract byte[] securityExit(MQChannelExit channelExitParms,
                                    MQChannelDefinition channelDefinition,
                                   byte agentBuffer[])
```

Método de rutina de salida de seguridad que debe proporcionar la clase.

Parámetros

channelExitParms
Contiene información referente al contexto en el que se está invocando la rutina de salida. La variable de miembro exitResponse es un parámetro de salida que se utiliza para indicar a MQSeries Cliente para Java la acción que se debe realizar a continuación. Para obtener información más detallada, consulte el apartado "MQChannelExit" en la página 90.

channelDefinition
Contiene detalles del canal a través del cual tienen lugar todas las comunicaciones con el gestor de colas.

agentBuffer
Si channelExitParms.exitReason es

MQChannelExit.MQXR_SEC_MSG, agentBuffer contiene el mensajes de seguridad recibido del gestor de colas; de lo contrario agentBuffer es nulo.

Devuelve

Si el código de respuesta de rutina de salida (en channelExitParms) se establece para que se deba transmitir un mensaje al gestor de colas, el método de rutina de salida de seguridad debe devolver los datos que se deben transmitir.

Vea también :
- "MQC" en la página 163
- "MQChannelDefinition" en la página 88

MQSendExit

interfaz pública **MQSendExit**
expande **Object**

La interfaz de rutina de salida de emisión le permite examinar y, posiblemente, modificar los datos que envía MQSeries Cliente para Java al gestor de colas.

Nota: Esta interfaz no es aplicable para conexiones directas a MQSeries en modalidad de enlaces.

Para proporcionar su propia rutina de salida de emisión, defina una clase que implemente esta interfaz. Cree una instancia nueva de la clase y asigne la variable MQEnvironment.sendExit antes de construir el objeto MQQueueManager. Por ejemplo:

```
// en la clase MySendExit.java
class MySendExit implements MQSendExit {
  // debe proporcionar una implementación del método sendExit
  public byte[] sendExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // el código de rutina de salida va aquí...
  }
}
// en el programa principal...
MQEnvironment.sendExit = new MySendExit();
 ...    // otra inicialización
MQQueueManager qMgr        = new MQQueueManager("");
```

Métodos

sendExit

```
public abstract byte[] sendExit(MQChannelExit channelExitParms,
                                MQChannelDefinition channelDefinition,
                                 byte agentBuffer[])
```

El método de rutina de salida de emisión que debe proporcionar la clase. Este método se invoca cada vez que MQSeries Classes for Java desea transmitir datos al gestor de colas.

Parámetros

channelExitParms
Contiene información referente al contexto en el que se está invocando la rutina de salida. La variable de miembro exitResponse es un parámetro de salida que se utiliza para indicar a MQSeries Classes for Java la acción que se debe realizar a continuación. Para obtener información más detallada, consulte el apartado “MQChannelExit” en la página 90.

channelDefinition
Contiene detalles del canal a través del cual tienen lugar todas las comunicaciones con el gestor de colas.

agentBuffer
Si channelExitParms.exitReason es MQChannelExit.MQXR_XMIT,

agentBuffer contiene datos que se deben transmitir al gestor de colas; de lo contrario agentBuffer es nulo.

Devuelve

Si el código de respuesta de rutina de salida (en channelExitParms) se establece para que se deba transmitir un mensaje al gestor de colas (MQXCC_OK), el método de rutina de salida de emisión debe devolver los datos que se deben transmitir. Por consiguiente, la rutina de salida de emisión más simple consta de la única línea ″return agentBuffer;″.

Vea también:

- "MQC" en la página 163
- "MQChannelDefinition" en la página 88

ManagedConnection

interfaz pública **javax.resource.spi.ManagedConnection**

Nota: Normalmente, las aplicaciones no utilizan esta clase. Su finalidad es que la utilicen las implementaciones de ConnectionManager.

MQSeries Classes for Java proporciona una implementación de ManagedConnection que se devuelve de ManagedConnectionFactory.createManagedConnection. Este objeto representa una conexión a un gestor de colas MQSeries.

Métodos

getConnection

```
public Object getConnection(javax.security.auth.Subject subject,
                            ConnectionRequestInfo cxRequestInfo)
```

Emite ResourceException.

Crea un nuevo manejador de conexión para la conexión física representada por el objeto ManagedConnection. Para MQSeries Classes for Java, devuelve un objeto MQQueueManager. Generalmente, ConnectionManager devuelve este objeto desde allocateConnection.

Se ignora el parámetro subject. Si el parámetro cxRequestInfo no es adecuado, se emite una ResourceException. Se pueden utilizar varios manejadores de conexión simultáneamente para cada ManagedConnection.

destroy

```
public void destroy()
```

Emite ResourceException.

Destruye la conexión física a un gestor de colas MQSeries. Se confirman todas las transacciones locales pendientes. Para obtener información más detallada, consulte el apartado “getLocalTransaction” en la página 173.

cleanup

```
public void cleanup()
```

Emite ResourceException.

Cierra todos los manejadores de conexión abiertos y restablece la conexión física a un estado inicial preparado para la agrupación. Se restituyen todas las transacciones locales pendientes. Para obtener información más detallada, consulte el apartado “getLocalTransaction” en la página 173.

associateConnection

```
public void associateConnection(Object connection)
```

Emite ResourceException.

Actualmente, MQSeries Classes for Java no ofrece soporte para este método. Se emite javax.resource.NotSupportedException.

addConnectionEventListener

```
public void addConnectionEventListener(ConnectionEventListener listener)
```

Añade un ConnectionEventListener a la instancia ManagedConnection.

Se notifica al escucha si se produce un error grave en ManagedConnection, al invocar a MQQueueManager.disconnect() en un manejador de conexión que está asociado a esta ManagedConnection. No se notifica al escucha sobre los sucesos de las transacciones locales (consulte el apartado "getLocalTransaction").

removeConnectionEventListener

```
public void removeConnectionEventListener(ConnectionEventListener listener)
```

Elimina un ConnectionEventListener registrado.

getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

Emite ResourceException.

Actualmente, MQSeries Classes for Java no ofrece soporte para este método. Se emite javax.resource.NotSupportedException.

getLocalTransaction

```
public LocalTransaction getLocalTransaction()
```

Actualmente, MQSeries Classes for Java no ofrece soporte para este método. Se emite javax.resource.NotSupportedException.

Actualmente, un ConnectionManager no puede gestionar la transacción local de MQSeries y no se informa a los ConnectionEventListeners registrados sobre los sucesos relacionados con la transacción local. Cuando se produce un cleanup(), se restituyen todas las unidades de trabajo en curso. Cuando se produce un destroy(), se confirman todas las unidades de trabajo en curso.

La presentación de las API existentes consiste en que una unidad de trabajo en curso se confirma en MQQueueManager.disconnect(). Esta presentación existente sólo se mantiene cuando un MQConnectionManager (en lugar de un ConnectionManager) gestiona la conexión.

getMetaData

```
public ManagedConnectionMetaData getMetaData()
```

Emite ResourceException.

Obtiene la información de metadatos para el gestor de colas subyacente. Consulte el apartado "ManagedConnectionMetaData" en la página 177.

ManagedConnection

setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

Emite ResourceException.

Establece el transcriptor de anotaciones para esta ManagedConnection. Cuando se crea una ManagedConnection, hereda el transcriptor de anotaciones de su ManagedConnectionFactory.

Actualmente, MQSeries Classes for Java no utiliza el transcriptor de anotaciones. Consulte el apartado “MQException.log” en la página 103 para obtener más información sobre las anotaciones.

getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

Emite ResourceException.

Devuelve el transcriptor de anotaciones para esta ManagedConnection.

Actualmente, MQSeries Classes for Java no utiliza el transcriptor de anotaciones. Consulte el apartado “MQException.log” en la página 103 para obtener más información sobre las anotaciones.

ManagedConnectionFactory

interfaz pública **javax.resource.spi.ManagedConnectionFactory**

Nota: Normalmente, las aplicaciones no utilizan esta clase.

MQSeries Classes for Java proporciona una implementación de esta interfaz para ConnectionManagers. ManagedConnectionFactory se utiliza para construir ManagedConnections, y para seleccionar ManagedConnections adecuadas de un conjunto de candidatos. Para obtener información más detallada sobre esta interfaz, consulte la especificación J2EE Connector Architecture (consulte el sitio web de Sun en la dirección `http://java.sun.com`).

Métodos

createConnectionFactory

```
public Object createConnectionFactory()
```

Emite ResourceException.

MQSeries Classes for Java actualmente no ofrece soporte para los métodos createConnectionFactory. Este método emite javax.resource.NotSupportedException.

createConnectionFactory

```
public Object createConnectionFactory(ConnectionManager cxManager)
```

Emite ResourceException.

MQSeries Classes for Java actualmente no ofrece soporte para los métodos createConnectionFactory. Este método emite javax.resource.NotSupportedException.

createManagedConnection

```
public ManagedConnection createManagedConnection
                                (javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

Emite ResourceException.

Crea una nueva conexión física con un gestor de colas MQSeries y devuelve un objeto ManagedConnection que representa dicha conexión. MQSeries ignora el parámetro subject.

matchManagedConnection

```
public ManagedConnection matchManagedConnection
                                (java.util.Set connectionSet,
                                 javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

Emite ResourceException.

Busca el conjunto suministrado de ManagedConnections candidatas para una ManagedConnection adecuada. Devuelve un valor nulo o una ManagedConnection adecuada del conjunto que cumple con los criterios para la conexión.

ManagedConnectionFactory

setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

Emite ResourceException.

Establece el transcriptor de anotaciones para esta ManagedConnectionFactory. Cuando se crea una ManagedConnection, hereda el transcriptor de anotaciones de su ManagedConnectionFactory.

Actualmente, MQSeries Classes for Java no utiliza el transcriptor de anotaciones. Consulte el apartado “MQException.log” en la página 103 para obtener más información sobre las anotaciones.

getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

Emite ResourceException.

Devuelve el transcriptor de anotaciones para esta ManagedConnectionFactory.

Actualmente, MQSeries Classes for Java no utiliza el transcriptor de anotaciones. Consulte el apartado “MQException.log” en la página 103 para obtener más información sobre las anotaciones.

hashCode

```
public int hashCode()
```

Devuelve el hashCode para esta ManagedConnectionFactory.

equals

```
public boolean equals(Object other)
```

Comprueba si esta ManagedConnectionFactory es igual a otra ManagedConnectionFactory. Devuelve verdadero si ambas ManagedConnectionFactories describen el mismo gestor de colas de destino.

ManagedConnectionMetaData

interfaz pública **javax.resource.spi.ManagedConnectionMetaData**

Nota: Normalmente, las aplicaciones no utilizan esta clase. Su finalidad es que la utilicen las implementaciones de ConnectionManager.

ConnectionManager puede utilizar esta clase para recuperar metadatos relacionados con una conexión física subyacente. Se devuelve una implementación de esta clase desde ManagedConnection.getMetaData().

Métodos

getEISProductName

```
public String getEISProductName()
```

Emite ResourceException.

Devuelve “IBM MQSeries”.

getProductVersion

```
public String getProductVersion()
```

Emite ResourceException.

Devuelve una serie de caracteres que describe el nivel de mandatos del gestor de colas MQSeries al que está conectada la ManagedConnection.

getMaxConnections

```
public int getMaxConnections()
```

Emite ResourceException.

Devuelve 0.

getUserName

```
public String getUserName()
```

Emite ResourceException.

Si ManagedConnection representa una conexión de cliente a un gestor de colas, devuelve el ID de usuario utilizado para la conexión. Si no es así, devuelve una serie vacía.

ManagedConnectionMetaData

Parte 3. Programación con MQ JMS

Capítulo 10. Creación de programas MQ JMS

En este capítulo se proporciona la información necesaria para ayudarle a crear aplicaciones MQ JMS. Incluye una breve introducción del modelo JMS e información detallada acerca de la programación de algunas tareas comunes que es probable que deban realizar los programas de aplicación.

El modelo JMS

JMS define una vista genérica de un servicio de envío de mensajes. Es importante que comprenda esta vista y cómo se correlaciona en el transporte MQSeries® subyacente.

El modelo JMS genérico se basa en torno a las interfaces siguientes, definidas en el paquete `javax.jms` de Sun:

Connection
Proporciona acceso al transporte subyacente y se utiliza para crear **Sessions**.

Session
Proporciona un contexto para producir y consumir mensajes, incluidos los métodos que se utilizan para crear **MessageProducers** y **MessageConsumers**.

MessageProducer
Se utiliza para enviar mensajes.

MessageConsumer
Se utiliza para recibir mensajes.

Tenga en cuenta que una **Connection** es de hebra segura, pero **Sessions**, **MessageProducers** y **MessageConsumers** no. La estrategia que se recomienda consiste en utilizar una sesión (Session) por hebra de aplicación.

En términos de MQSeries:

Connection
Proporciona un ámbito para colas temporales. Además, facilita un lugar para mantener los parámetros que controlan cómo conectar a MQSeries. Algunos ejemplos de estos parámetros son el nombre del gestor de colas y el nombre del sistema principal remoto, si se utiliza la conectividad de cliente MQSeries Java™.

Session
Contiene un `HCONN` y, por consiguiente, define un ámbito transaccional.

MessageProducer y MessageConsumer
Contienen un `HOBJ` que define una cola determinada en la que escribir y de la que leer.

Tenga en cuenta que se aplican las normas habituales de MQSeries:

- Sólo puede haber una operación en curso por HCONN al mismo tiempo. Por este motivo, no se puede invocar simultáneamente a MessageProducers o MessageConsumers asociados a una Session, lo que es coherente con la restricción de JMS de una sola hebra por Session.

- Las operaciones de transferencia (PUT) pueden utilizar colas remotas, pero sólo se pueden aplicar operaciones de obtener (GET) a las colas del gestor de colas local.

Las interfaces JMS genéricas se subclasifican en versiones más específicas para las presentaciones "Punto a punto" y "Publicación/suscripción".

Las versiones Punto a punto son las siguientes:
- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

Una idea clave en JMS es que es posible, y se aconseja encarecidamente, crear programas de aplicación que sólo utilicen referencias a las interfaces de `javax.jms`. Toda la información específica del proveedor se encapsula en implementaciones de:
- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

Se denominan "objetos administrados", es decir, objetos que se pueden crear utilizando la herramienta de administración que suministra el proveedor y se pueden almacenar en un espacio de nombres JNDI. Una aplicación JMS puede recuperar los objetos del espacio de nombres y utilizarlos sin necesidad de conocer el proveedor que ha proporcionado la implementación.

Creación de una conexión

Las conexiones no se crean directamente, sino utilizando una fábrica de conexiones. Los objetos de fábrica se pueden almacenar en un espacio de nombres JNDI, aislando así la aplicación JMS de la información específica del suministrador. En el “Capítulo 5. Utilización de la herramienta de administración de MQ JMS” en la página 35 se proporciona información detallada acerca de cómo crear y almacenar objetos de fábrica.

Si no dispone de un espacio de nombres JNDI, consulte el apartado “Creación de fábricas durante la ejecución” en la página 184.

Recuperación de la fábrica de JNDI

Para recuperar un objeto de un espacio de nombres JNDI, se debe establecer un contexto inicial, tal como se muestra en el fragmento siguiente, extraído del archivo de ejemplo de IVTRun:

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

donde:

icf define una clase de fábrica para el contexto inicial

url define un URL específico del contexto

Para obtener información más detallada sobre la utilización de JNDI, consulte la documentación de JNDI de Sun.

Nota: Algunas combinaciones de paquetes JNDI y suministradores de servicio LDAP pueden producir un error LDAP 84. Para resolver el problema, inserte la línea siguiente antes de la invocación a InitialDirContext.

```
environment.put(Context.REFERRAL, "throw");
```

Después de obtener un contexto inicial, los objetos se pueden recuperar del espacio de nombres utilizando el método lookup(). El código siguiente recupera una QueueConnectionFactory denominada ivtQCF de un espacio de nombres basado en LDAP:

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

Utilización de la fábrica para crear una conexión

El método `createQueueConnection()` del objeto de fábrica se utiliza para crear una conexión ('Connection'), tal como se muestra en el código siguiente:

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

Creación de fábricas durante la ejecución

Si no hay ningún espacio de nombres JNDI disponible, puede crear objetos de fábrica durante la ejecución. No obstante, la utilización de este método reduce la portabilidad de la aplicación JMS, puesto que requiere referencias a clases específicas de MQSeries.

El código siguiente crea una QueueConnectionFactory con todos los valores por omisión:

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

(Puede omitir el prefijo `com.ibm.mq.jms.` si importa el paquete `com.ibm.mq.jms`).

Una conexión creada de la fábrica anterior utiliza los enlaces Java™ para conectar con el gestor de colas por omisión de la máquina local. Se pueden utilizar los métodos `set` que se muestran en la Tabla 14 para personalizar la fábrica con información específica de MQSeries.

Inicio de la conexión

La especificación JMS define que las conexiones se deben crear en el estado "detenido". Hasta que se inicia la conexión, los MessageConsumers asociados a la conexión no pueden recibir ningún mensaje. Para iniciar la conexión, emita el mandato siguiente:

```
connection.start();
```

Tabla 14. Métodos de establecimiento (set) de MQQueueConnectionFactory

Método	Descripción
setCCSID(int)	Se utiliza para establecer la propiedad `MQEnvironment.CCSID`
setChannel(String)	Nombre del canal para una conexión de cliente
setHostName(String)	Nombre del sistema principal para una conexión de cliente
setPort(int)	Puerto para una conexión de cliente
setQueueManager(String)	Nombre del gestor de colas
setTemporaryModel(String)	Nombre de la cola modelo que se utiliza para generar un destino temporal como resultado de una invocación a `QueueSession.createTemporaryQueue()`. Se aconseja que sea el nombre de la cola dinámica temporal, en lugar de una cola dinámica permanente.
setTransportType(int)	Especifica el modo de conectar a MQSeries. Las opciones disponibles actualmente son las siguientes: • JMSC.MQJMS_TP_BINDINGS_MQ (el valor por omisión) • JMSC.MQJMS_TP_CLIENT_MQ_TCPIP. `JMSC` se encuentra en el paquete `com.ibm.mq.jms`

Tabla 14. Métodos de establecimiento (set) de MQQueueConnectionFactory (continuación)

Método	Descripción
setReceiveExit(String) setSecurityExit(String) setSendExit(String) setReceiveExitInit(String) setSecutityExitInit(String) setSendExitInit(String)	La finalidad de estos métodos es permitir que se puedan utilizar las rutina de salida de emisión, recepción y seguridad que proporciona MQSeries Classes for Java™. Los métodos `set*Exit` toman el nombre de una clase que implementa los métodos de rutina de salida adecuados. (Para obtener información más detallada, consulte la documentación del producto MQSeries 5.1). Además, la clase debe implementar un constructor con un solo parámetro `String`, que proporciona todos los datos de inicialización que puede necesitar la rutina de salida, y se establece en el valor que proporciona el método `set*ExitInit` correspondiente.

Elección del transporte de enlaces o de cliente

MQ JMS se puede comunicar con MQSeries utilizando los transportes de enlaces o de cliente. Si utiliza los enlaces Java™, la aplicación JMS y el gestor de colas MQSeries deben estar situados en la misma máquina. Si utiliza el cliente, el gestor de colas puede estar en una máquina diferente a la de la aplicación.

El contenido del objeto de fábrica de conexiones determina el transporte que se debe utilizar. En el “Capítulo 5. Utilización de la herramienta de administración de MQ JMS” en la página 35 se describe cómo definir un objeto de fábrica para utilizarlo con el transporte de enlaces o de cliente.

El fragmento de código siguiente muestra cómo definir el transporte en una aplicación:

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

Obtención de una sesión

Después de realizar una conexión, utilice el método `createQueueSession` de QueueConnection para obtener una sesión.

El método toma dos parámetros:

1. Un booleano que determina si se trata de una sesión ″ejecutada″ o ″no ejecutada″.
2. Un parámetro que determina la modalidad de ″reconocimiento″.

El caso más sencillo es el de la sesión ″no ejecutada″ con `AUTO_ACKNOWLEDGE`, tal como se muestra en el fragmento de código siguiente:

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

Nota: Una conexión es de hebra segura, pero las sesiones (y los objetos que se crean a partir de éstas) no. La práctica que se aconseja para las aplicaciones de múltiples hebras consiste en utilizar una sesión separada para cada hebra.

Envío de un mensaje

Los mensajes se envían utilizando un MessageProducer. En el caso de punto a punto, es un QueueSender que se crea utilizando el método createSender de QueueSession. Normalmente, un QueueSender se crea para una cola específica, de modo que todos los mensajes que se envían utilizando dicho emisor se envían al mismo destino. El destino se especifica utilizando un objeto de cola (Queue). Los objetos de cola se pueden crear durante la ejecución, o se pueden crear y almacenar en un espacio de nombres JNDI.

Los objetos de cola se recuperan de JNDI utilizando el procedimiento siguiente:

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

MQ JMS proporciona una implementación de Queue en com.ibm.mq.jms.MQQueue. Contiene propiedades que controlan los detalles de la presentación específica de MQSeries pero, en muchos casos, se pueden utilizar valores por omisión. JMS define un procedimiento estándar para especificar el destino que minimiza el código específico de MQSeries en la aplicación. Este mecanismo utiliza el método QueueSession.createQueue, que toma un parámetro string que describe el destino. En sí misma, la serie de caracteres (string) sigue estando en el formato específico de un proveedor, pero es una propuesta mucho más flexible que si se hace referencia directamente a las clases de proveedor.

MQ JMS acepta dos formatos para el parámetro string de createQueue().

- El primero es el nombre de la cola MQSeries, tal como se muestra en el fragmento siguiente extraído del programa IVTRun del directorio samples:

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- El segundo formato, más eficaz, se basa en URI (identificadores de recursos uniformes). Este formato le permite especificar colas remotas (colas que están en un gestor de colas distinto del gestor de colas al que está conectado). También le permite establecer otras propiedades contenidas en un objeto com.ibm.mq.jms.MQQueue.

  Para una cola, URI empieza por la secuencia queue://, seguida del nombre del gestor de colas en el que reside la cola. A continuación, se incluye otra barra inclinada '/', el nombre de la cola y, si se desea, una lista de pares de nombre-valor que establecen las propiedades restantes de Queue. Por ejemplo, el equivalente URI del ejemplo anterior es:

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

  Tenga en cuenta que se omite el nombre del gestor de colas. Se interpreta como el gestor de colas al que está conectada la QueueConnection propietaria en el momento de utilizar el objeto Queue.

En el ejemplo siguiente se conecta a la cola 'Q1' del gestor de colas 'HOST1.QM1', lo que provoca que se envíen todos los mensajes como no permanentes y con la prioridad 5:

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

En la Tabla 15 en la página 188 se incluye la lista de los nombres que se pueden utilizar en la parte de nombre-valor de URI. Un inconveniente de este formato es que no ofrece soporte para nombres simbólicos para los valores por lo que, cuando

procede, la tabla también indica valores ″especiales″, que pueden estar sujetos a cambios. (En el apartado “Establecimiento de propiedades con el método ‘set’” se proporciona un método alternativo para el establecimiento de propiedades).

Tabla 15. Nombres de propiedades para URI de cola

Propiedad	Descripción	Valores
expiry	Duración del mensaje en milisegundos	0 para ilimitada, enteros positivos para tiempo de espera (ms)
priority	Prioridad del mensaje	De 0 a 9, -1=QDEF, -2=APP
persistence	Si se hace una ″copia en disco″ del mensaje	1=no permanente, 2=permanente, -1=QDEF, -2=APP
CCSID	Juego de caracteres del destino	enteros - los valores válidos contenidos en la lista de la documentación MQSeries básico
targetClient	Indica si la aplicación receptora es compatible con JMS o no	0=JMS, 1=MQ
encoding	El modo de representar campos numéricos	Un valor de entero tal como se describe en la documentación de MQSeries básico
QDEF - valor especial que significa que la configuración de la cola MQSeries debe determinar la propiedad. **APP** - valor especial que significa que la aplicación JMS puede controlar esta propiedad.		

Después de obtener el objeto Queue (utilizando `createQueue` tal como se indica más arriba o desde JNDI), se debe pasar al método `createSender` para crear un QueueSender:

```
QueueSender queueSender = session.createSender(ioQueue);
```

El objeto `queueSender` resultante se puede utilizar para enviar mensajes utilizando el método `send`:

```
queueSender.send(outMessage);
```

Establecimiento de propiedades con el método ‘set’

Puede establecer propiedades Queue creando antes una instancia de `com.ibm.mq.jms.MQQueue` utilizando el constructor por omisión. Después puede rellenar los valores adecuados utilizando métodos `set` públicos. Este método significa que puede utilizar nombres simbólicos para los valores de propiedad. Sin embargo, puesto que estos valores son específicos del proveedor, y están incluidos en el código, las aplicaciones resultan menos portátiles.

En el fragmento de código siguiente se muestra el valor de una propiedad de cola con un método `set`.

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

La Tabla 16 en la página 189 muestra los valores de propiedad simbólicos que se proporcionan con MQ JMS para utilizarlos con los métodos `set`.

Tabla 16. Valores simbólicos para propiedades de cola

Propiedad	Palabra clave herramienta admin.	Valores
expiry	UNLIM APP	JMSC.MQJMS_EXP_UNLIMITED JMSC.MQJMS_EXP_APP
priority	APP QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
persistence	APP QDEF PERS NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON
targetClient	JMS MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
encoding	Integer(N) Integer(R) Decimal(N) Decimal(R) Float(N) Float(R) Native	JMSC.MQJMS_ENCODING_INTEGER_NORMAL JMSC.MQJMS_ENCODING_INTEGER_REVERSED JMSC.MQJMS_ENCODING_DECIMAL_NORMAL JMSC.MQJMS_ENCODING_DECIMAL_REVERSED JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED JMSC.MQJMS_ENCODING_NATIVE

Para obtener información más detallada sobre la codificación, consulte el apartado “Propiedad ENCODING” en la página 49.

Tipos de mensaje

JMS proporciona varios tipos de mensaje y, cada uno de ellos, incorpora alguna información de su contenido. Para evitar que se haga referencia a los nombres de clase específicos del proveedor para los tipos de mensaje, en el objeto Session se facilitan métodos para la creación de mensaje.

En el programa de ejemplo, se crea un mensaje de texto del modo siguiente:

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

Los tipos de mensaje que se pueden utilizar son los siguientes:
- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Consulte el “Capítulo 14. Interfaces y clases JMS” en la página 245 para obtener información detallada sobre estos tipos.

Recepción de un mensaje

Los mensajes se reciben utilizando un QueueReceiver. Se crea desde una sesión (Session) utilizando el método `createReceiver()`. Este método toma un parámetro Queue que define de dónde se reciben los mensajes. Consulte el apartado “Envío de un mensaje” en la página 187 para obtener información detallada sobre la creación de un objeto Queue.

Recepción de un mensaje

En el programa de ejemplo se crea un receptor y se lee el mensaje de prueba con el código siguiente:

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

El parámetro de la invocación de recepción es un tiempo de espera en milisegundos. Este parámetro define el tiempo que debe esperar el método si no hay ningún mensaje disponible inmediatamente. Puede omitir este parámetro, en cuyo caso, la invocación se bloquea de modo indefinido. Si no desea que haya retardo, utilice el método `receiveNoWait()`.

Los métodos de recepción devuelven un mensaje del tipo adecuado. Por ejemplo, si se transfiera un `TextMessage` a una cola, cuando se recibe el mensaje, el objeto que se devuelve es una instancia de `TextMessage`.

Para extraer el contenido del cuerpo del mensaje, se debe enviar de la clase Message genérica (que es el tipo de retorno declarado de los mensajes de recepción) a la subclase más específica como, por ejemplo, `TextMessage`. Si el mensaje recibido es de tipo desconocido, puede utilizar el operador 'instanceof' para determinarlo. Por lo general, suele ser conveniente probar la clase del mensaje antes de enviarlo ya que, de este modo, los errores inesperados pueden manejarse correctamente.

En el código siguiente se muestra la utilización de 'instanceof' y la extracción de contenido de un TextMessage:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Imprimir mensaje de error si el mensaje (Message) no es un TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Selectores de mensaje

JMS proporciona un mecanismo para seleccionar un subconjunto de mensajes de una cola, permitiendo que una invocación de recepción devuelva dicho subconjunto. Cuando se crea un QueueReceiver, se puede proporcionar una serie de caracteres que contenga una expresión SQL (Lenguaje de consulta estructurado) para determinar los mensajes que se deben recuperar. El selector pueden hacer referencia tanto a campos de la cabecera del mensaje JMS como a campos de las propiedades del mensaje (son campos de cabecera que define la aplicación). En el "Capítulo 12. Mensajes JMS" en la página 205 se proporcionan detalles de los nombres de campos de cabecera, así como la sintaxis para el selector SQL.

En el ejemplo siguiente se muestra cómo seleccionar una propiedad definida por el usuario denominada `myProp`:

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

Nota: La especificación JMS no permite cambiar el selector asociado a un receptor. Después de crear un receptor, se fija el selector para toda la duración de éste. Significa que, si necesita otros selectores, debe crear nuevos receptores.

Entrega asíncrona

Una alternativa a la emisión de invocaciones a `QueueReceiver.receive()` consiste en registrar un método al que se invoque automáticamente cuando haya un mensaje adecuado disponible. En el fragmento siguiente se muestra este mecanismo:

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // El método al que va a invocar JMS cuando hay
  // un mensaje disponible.
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // aquí proceso específico de la aplicación
    .
    .
    .
  }
}

 .
 .
 .
  // En programa principal (posiblemente de alguna otra clase)
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // el programa principal puede continuar ahora con otra presentación
  // específica de la aplicación.
```

Nota: La utilización de la entrega asíncrona con un QueueReceiver marca toda la Session como asíncrona. No se puede realizar una invocación explícita a los métodos `receive` de un QueueReceiver que esté asociado a una Session que utiliza entrega asíncrona.

Cierre

La recopilación de basura sola no puede liberar todos los recursos de MQSeries en el tiempo oportuno. Esto se aplica especialmente si la aplicación necesita crear varios objetos JMS de corta duración en el nivel Session o inferior. Por este motivo, es importante invocar métodos `close()` de las distintas clases (QueueConnection, QueueSession, QueueSender y QueueReceiver) cuando ya no se necesitan los recursos.

Java Virtual Machine se cuelga al cerrar

Si una aplicación MQ JMS finaliza sin invocar a `Connection.close()`, se pueden colgar algunas JVM. Si se produce este problema, edite la aplicación para incluir una invocación a `Connection.close()`, o termine JVM utilizando las teclas `Control-C`.

Manejo de errores

Las excepciones informan de todos los errores de ejecución de una aplicación JMS. En JMS, la mayor parte de los métodos emiten JMSExceptions para indicar errores. Una práctica de programación recomendada consiste en enviar estas excepciones y visualizarlas en una salida conveniente.

A diferencia de las excepciones normales de Java™, una JMSException puede contener otra excepción incluida. Para JMS, puede constituir una forma muy adecuada de pasar detalles importantes del transporte subyacente. En el caso de MQ JMS, cuando MQSeries emite una MQException, generalmente, ésta se muestra como una excepción incluida en una JMSException.

La implementación de JMSException no incorpora la excepción incluida en la salida de su método `toString()`. Por consiguiente, se debe comprobar de forma explícita si hay una excepción incluida e imprimirla, tal como se muestra en el fragmento siguiente:

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

Escucha de excepciones

Para la entrega de mensajes asíncrona, el código de la aplicación no puede captar las excepciones que se emiten al producirse anomalías en la recepción de mensajes. Se debe a que el código de la aplicación no realiza invocaciones explícitas a métodos `receive()`. Para hacer frente a esta situación, se puede registrar un `ExceptionListener`, que es una instancia de una clase que implementa el método `onException()`. Cuando se produce un problema grave, se invoca este método, en el que se pasa la JMSException como único parámetro. Puede obtener información más detallada en la documentación de JMS de Sun.

Capítulo 11. Programación de aplicaciones Publish/Subscribe

En este apartado se presenta el modelo de programación que se utiliza para crear aplicaciones Publish/Subscribe que utilicen el Servicio de mensajes de MQSeries® Classes for Java™.

Creación de una aplicación Publish/Subscribe simple

En este apartado se proporciona una 'guía' para una aplicación MQ JMS simple.

Importación de los paquetes necesarios

Una aplicación del Servicio de mensajes de MQSeries Classes for Java empieza con varias sentencias de importación (import), que deben incluir, como mínimo, lo siguiente:

```
import javax.jms.*;                // Interfaces JMS
import javax.naming.*;             // Se utiliza para la búsqueda JNDI de
import javax.naming.directory.*;   //    objetos administrados
```

Obtención o creación de objetos JMS

El paso siguiente consiste en obtener o crear varios objetos JMS:

1. Obtención de una TopicConnectionFactory
2. Creación de una TopicConnection
3. Creación de una TopicSession
4. Obtención de un Topic de JNDI
5. Creación de TopicPublishers y TopicSubscribers

Muchos de estos procesos son similares a los que se utilizan para punto a punto, tal como se muestra a continuación:

Obtención de una TopicConnectionFactory

El modo preferido de hacerlo consiste en utilizar la búsqueda JNDI, puesto que se mantiene la portabilidad del código de aplicación. El código siguiente inicializa un contexto JNDI:

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://server.company.com/o=company_us,c=us";

Java.util.Hashtable env = new java.util.Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, CTX_FACTORY );
env.put( Context.PROVIDER_URL,            INIT_URL );
env.put( Context.REFERRAL,                "throw" );

Context ctx = null;
try {
   ctx = new InitialDirContext( env );
} catch( NamingException nx ) {
   // Añadir código para manejar la imposibilidad de conectar al contexto JNDI
}
```

Nota: Debe personalizar las variables `CTX_FACTORY` y `INIT_URL` para ajustarlas a su instalación y al suministrador de servicio JNDI.

Las propiedades que necesita la inicialización JNDI están en una tabla de totales de control que se pasa al constructor InitialDirContext. Si la

conexión no se ejecuta correctamente, se emite una excepción para indicar que los objetos administrados que se necesitan más adelante en la aplicación no están disponibles.

A continuación, obtenga una TopicConnectionFactory utilizando una clave de búsqueda que ha definido el administrador:

```
TopicConnectionFactory factory;
factory = (TopicConnectionFactory)lookup("cn=sample.tcf");
```

Si no hay ningún espacio de nombres JNDI disponible, puede crear una TopicConnectionFactory durante la ejecución. El procedimiento para crear una nueva com.ibm.mq.jms.MQTopicConnectionFactory es similar al método que se describe para QueueConnectionFactory en el apartado "Creación de fábricas durante la ejecución" en la página 184.

Creación de una TopicConnection

Se crea desde el objeto TopicConnectionFactory. Las conexiones siempre se inicializan en un estado detenido (`stop`) y se deben iniciar con el código siguiente:

```
TopicConnection conn;
conn = factory.createTopicConnection();
conn.start();
```

Creación de una TopicSession

Se crea utilizando una TopicConnection. Este método toma dos parámetros; uno para indicar si la sesión es transaccional y otro para especificar la modalidad de reconocimiento:

```
TopicSession session = conn.createTopicSession( false,
                                                Session.AUTO_ACKNOWLEDGE );
```

Obtención de un Topic

Este objeto se puede obtener de JNDI para utilizarlo con los TopicPublishers y TopicSubscribers que se crean posteriormente. El código siguiente recupera un Topic:

```
Topic topic = null;
 try {
    topic = (Topic)ctx.lookup( "cn=sample.topic" );
 } catch( NamingException nx ) {
    // Añadir código para manejar la imposibilidad de recuperar temas (Topic) de JNDI
 }
```

Si no hay ningún espacio de nombres JNDI disponible, puede crear un tema (Topic) durante la ejecución, tal como se describe en el apartado "Creación de temas durante la ejecución" en la página 197.

Creación de clientes y productores de publicaciones

Dependiendo de la naturaleza de la aplicación cliente JMS que escriba, debe crear un suscriptor, un publicador o ambos. Utilice los métodos `createPublisher` y `createSubscriber` tal como se indica a continuación:

```
// Crear un publicador, que publique sobre el tema especificado
 TopicPublisher pub = session.createPublisher( topic );
// Crear un suscriptor, suscrito al tema especificado
 TopicSubscriber sub = session.createSubscriber( topic );
```

Publicación de mensajes

El objeto TopicPublisher, pub, se utiliza para publicar mensajes, de forma parecida a cuando se utiliza un QueueSender en el dominio punto a punto. El fragmento siguiente crea un TextMessage utilizando la sesión y, a continuación, publica el mensaje:

```
// Crear el TextMessage y poner algunos datos en el mismo
TextMessage outMsg = session.createTextMessage();
outMsg.setText( "This is a short test string!" );

// Utilizar el publicador para publicar el mensaje
pub.publish( outMsg );
```

Recepción de suscripciones

Los suscriptores deben poder leer las suscripciones que se les entregan, tal como se muestra en el código siguiente:

```
// Recuperar la siguiente suscripción en espera
TextMessage inMsg = (TextMessage)sub.receive();

// Obtener el contenido del mensaje
String payload = inMsg.getText();
```

Este fragmento de código realiza una operación de ″obtener con espera″, lo que significa que la invocación de recepción se bloquea hasta que hay un mensaje disponible. Hay versiones alternativas de la invocación de recepción (como, por ejemplo, 'receiveNoWait'). Para obtener información más detallada, consulte el apartado “TopicSubscriber” en la página 353.

Cierre de los recursos no deseados

Es importante liberar todos los recursos que utiliza la aplicación Publish/Subscribe cuando finaliza. Utilice el método close() en los objetos que se pueden cerrar (publicadores, suscriptores, sesiones y conexiones):

```
// Cerrar los publicadores y los suscriptores
pub.close();
sub.close();

// Cerrar las sesiones y las conexiones
session.close();
conn.close();
```

Utilización de temas

En este apartado se explica cómo utilizar objetos de temas JMS (JMS Topic) en aplicaciones de Servicio de mensajes de MQSeries Classes for Java.

Nombres de temas

En este apartado se describe cómo utilizar nombres de temas en el Servicio de mensajes de MQSeries Classes for Java.

Nota: La especificación JMS no especifica detalles exactos sobre la utilización y el mantenimiento de jerarquías de temas. Por este motivo, este área puede variar de un suministrador a otro.

En MQ JMS, los nombres de temas se ordenan en una jerarquía similar a un árbol, como se muestra en el ejemplo de la Figura 3 en la página 196.

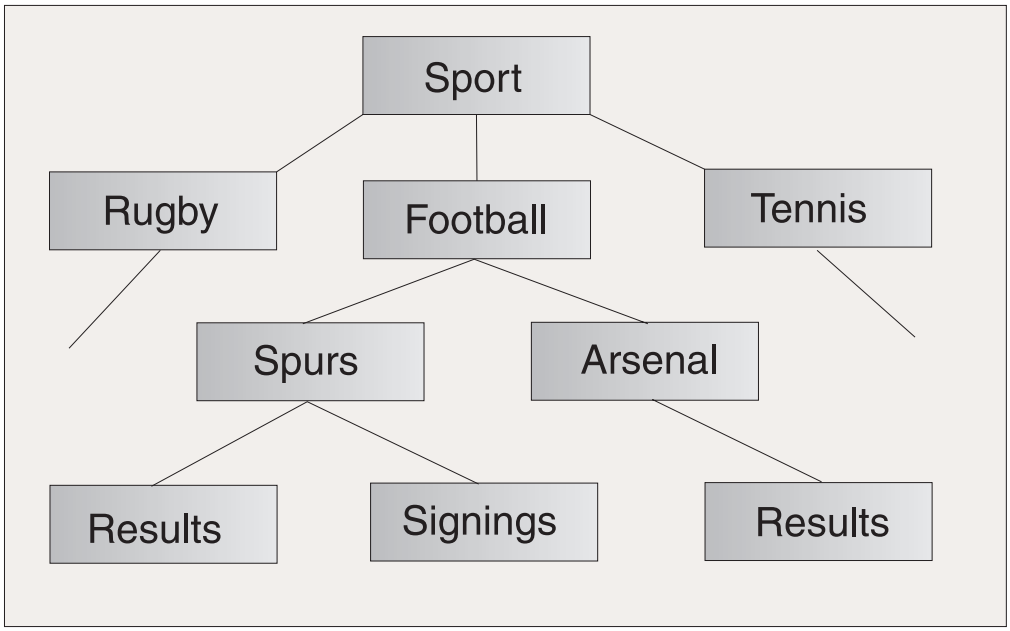


Figura 3. Jerarquía de nombres de temas

En un nombre de tema, los niveles del árbol están separados mediante el carácter '/', lo que significa que el nombre de tema representa el nodo 'Signings':

```
Sport/Football/Spurs/Signings
```

Un característica eficaz del sistema de temas del Servicio de mensajes de MQSeries Classes for Java consiste en utilizar caracteres comodín. Permiten que los suscriptores puedan suscribirse a más de un tema al mismo tiempo. Los caracteres comodín, '*', coinciden con cero o más caracteres, mientras que el comodín de símbolo de interrogación, '?', sólo coincide con un carácter.

Si un suscriptor se suscribe al tema que representa el nombre de tema siguiente:

```
Sport/Football/*/Results
```

recibe publicaciones sobre temas, tales como:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

Si el tema de la suscripción es:

```
Sport/Football/Spurs/*
```

recibe publicaciones sobre temas, tales como:
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

No es necesario administrar las jerarquías de temas que se utilizan en el área del intermediario del sistema explícitamente. Cuando se forma el primer publicador o suscriptor sobre un tema determinado, el intermediario crea de modo automático el estado de los temas que se están publicando o suscribiendo actualmente.

Nota: Un publicador no puede publicar sobre un tema cuyo nombre contiene caracteres comodín.

Creación de temas durante la ejecución

Existen cuatro procedimientos para crear objetos de temas (Topic) durante la ejecución:

1. Construir un tema utilizando el constructor `MQTopic` de un argumento.
2. Construir un tema utilizando el constructor `MQTopic` por omisión y, a continuación, invocar el método `setBaseTopicName(..)`.
3. Utilizar el método `createTopic(..)` de la sesión.
4. Utilizar el método `createTemporaryTopic()` de la sesión.

Método 1: Utilizar MQTopic(..)

Este método necesita una referencia a la implementación MQSeries de la interfaz JMS Topic y, por consiguiente, convierte el código en no portátil.

El constructor toma un argumento, que debe ser URI (identificador de recursos uniforme). Para los temas de Servicio de mensajes de MQSeries Classes for Java, debe tener el formato:

```
topic://NombreTema[?propiedad=valor[&propiedad=valor]*]
```

Para obtener información más detallada sobre los URI y los pares de nombre-valor permitidos, consulte el apartado “Envío de un mensaje” en la página 187.

El código siguiente crea un tema para mensajes de prioridad 5 no permanentes:

```
// Crear un tema utilizando el constructor MQTopic de un argumento
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

Método 2: Utilizar MQTopic() y después setBaseTopicName(..)

Este método utiliza el constructor `MQTopic` por omisión y, por consiguiente, convierte el código en no portátil.

Después de crear el objeto, establezca la propiedad `baseTopicName` utilizando el método `setBaseTopicName`, pasando el nombre de tema necesario.

Nota: El nombre de tema que se utiliza aquí no está en formato URI y no puede incluir pares de nombre-valor. Estos debe establecerlos utilizando los métodos ‘set’, tal como se describe en el apartado “Establecimiento de propiedades con el método ‘set’” en la página 188. El código siguiente utiliza este método para crear un tema:

```
// Crear un tema utilizando el constructor MQTopic por omisión
Topic rtTopic = new MQTopic();

// Establecer las propiedades de objeto utilizando los métodos
de establecimiento (setter)
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

Método 3: Utilizar session.createTopic(..)

También se puede crear un objeto de tema utilizando el método `createTopic` de TopicSession, que toma un URI de tema, tal como se muestra a continuación:

```
// Crear un tema utilizando el método de la fábrica de sesiones
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

Método 4: Utilizar session.createTemporaryTopic()

Un TemporaryTopic es un tema que sólo pueden consumir los suscriptores que crea la misma TopicConnection. Un TemporaryTopic se crea del modo siguiente:

```
// Crear un TemporaryTopic utilizando el método de la fábrica de sesiones
Topic rtTopic = session.createTemporaryTopic();
```

Opciones de suscriptor

Hay varios modos de utilizar los suscriptores JMS. En este apartado se describen algunos ejemplos de su utilización.

JMS proporciona dos tipos de suscriptores:

Suscriptores no duraderos

Estos suscriptores reciben mensajes sobre el tema que han elegido, sólo si se publican mientras el suscriptor está activo.

Suscriptores duraderos

Estos suscriptores reciben todos los mensajes que se publican sobre un tema, incluidos los que se publican mientras el suscriptor está inactivo.

Creación de suscriptores no duraderos

El suscriptor que se crea en el apartado "Creación de clientes y productores de publicaciones" en la página 194 no es duradero y se crea con el código siguiente:

```
// Crear un suscriptor, suscrito al tema especificado
 TopicSubscriber sub = session.createSubscriber( topic );
```

Creación de suscriptores duraderos

La creación de un suscriptor duradero es muy similar a la creación de un suscriptor no duradero, pero en este caso también se debe proporcionar un nombre que lo identifique de modo exclusivo:

```
// Crear un suscriptor duradero, suministrando un nombre de identificador exclusivo
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

Los suscriptores no duraderos revocan automáticamente su suscripción al invocar al método `close()` (o cuando abandonan el ámbito). Sin embargo, si desea terminar una suscripción duradera, debe notificarlo explícitamente al sistema. Para hacerlo, utilice el método `unsubscribe()` de la sesión y pase el nombre exclusivo que ha creado el suscriptor:

```
// Anular la suscripción del suscriptor duradero creado más arriba
session.unsubscribe( "D_SUB_000001" );
```

Un suscriptor duradero se crea en el gestor de colas especificado en el parámetro de gestor de colas MQTopicConnectionFactory. Si, posteriormente, se intenta crear un suscriptor duradero con el mismo nombre en otro gestor de colas, se devuelve un suscriptor duradero nuevo y totalmente independiente.

Utilización de selectores de mensaje

Puede utilizar selectores de mensaje para filtrar los mensajes que no cumplan los criterios especificados. Para obtener información más detallada sobre los selectores de mensaje, consulte el apartado "Selectores de mensaje" en la página 190. Los selectores de mensaje se asocian a un suscriptor tal como se indica a continuación:

```
// Asociar un selector de mensaje a un suscriptor no duradero
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

Supresión de publicaciones locales

Se puede crear un suscriptor que ignore publicaciones que se publican en la conexión particular del suscriptor. Establezca el tercer parámetro de la invocación de `createSubscriber` en verdadero (true), tal como se indica a continuación:

```
// Crear un suscriptor no duradero con la opción noLocal establecida
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

Combinación de opciones de suscriptor

Puede combinar las variaciones de suscriptor, lo que le permite crear un suscriptor duradero que aplique un selector e ignore las publicaciones locales, si así lo desea. En el fragmento de código siguiente se muestra la utilización de opciones combinadas:

```
// Crear un suscriptor duradero, no local al que se aplique un selector
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                       selector, true );
```

Configuración de la cola de suscriptores base

Con MQ JMS V5.2, existen dos procedimientos en los que puede configurar suscriptores:

- Propuesta de cola múltiple

  Cada suscriptor tiene una cola exclusiva asignada, de la que recupera todos sus mensajes. JMS crea una nueva cola para cada suscriptor. Ésta es la única propuesta disponible con MQ JMS V1.1.

- Propuesta de cola compartida

  Un suscriptor utiliza una cola compartida, de la que recuperan mensajes tanto él como otros suscriptores. Esta propuesta sólo requiere una cola para prestar servicio a varios suscriptores. Ésta es la propuesta por omisión que se utiliza con MQ JMS V5.2.

En MQ JMS V5.2, puede elegir la propuesta que desea utilizar, y configurar las colas que se van a usar.

Por lo general, la propuesta de la cola compartida ofrece una ligera mejora de rendimiento. Para sistemas con una gran productividad, también existen grandes ventajas administrativas y arquitectónicas, debido a la importante reducción del número de colas que se necesitan.

En algunas situaciones, sigue habiendo buenas razones para utilizar la propuesta de la cola múltiple:

- En teoría, la capacidad física para almacenamiento de mensajes es mayor.

  Una cola MQSeries no puede mantener más de 640000 mensajes y, en la propuesta de la cola compartida, se debe dividir entre todos los suscriptores que comparten la cola. Esta cuestión es más importante para los suscriptores duraderos, puesto que su duración suele ser muy superior a la de los suscriptores no duraderos. Por este motivo, se pueden acumular más mensajes de un suscriptor duradero.

- La administración externa de las colas de suscripciones es más sencilla.

  Para determinados tipos de aplicaciones, es posible que los administradores deseen supervisar el estado y la capacidad de colas de suscriptores determinadas. Esta tarea es mucho más sencilla cuando existe la correlación de uno a uno entre un suscriptor y una cola.

Configuración por omisión

La configuración por omisión utiliza las colas de suscripciones compartidas siguientes:

- SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE para suscripciones no duraderas
- SYSTEM.JMS.D.SUBSCRIPTION.QUEUE para suscripciones duraderas

Se crean automáticamente cuando se ejecuta el script MQJMS_PSQ.MQSC.

Si fuera necesario, puede especificar colas físicas alternativas. También puede cambiar la configuración para utilizar la propuesta de cola múltiple.

Configuración de suscriptores no duraderos

Puede establecer la propiedad del nombre de cola de suscriptores no duraderos utilizando cualquiera de los procedimientos siguientes:

- Utilizando la herramienta de administración MQ JMS (para objetos recuperados JNDI) para establecer la propiedad BROKERSUBQ
- Utilizando el método setBrokerSubQueue() en el programa

Para suscripciones no duraderas, el nombre de cola que proporciona debe empezar por los caracteres siguientes:

SYSTEM.JMS.ND.

Para seleccionar una propuesta de cola compartida, especifique un nombre de cola explícito, donde la cola nombrada sea la que se va a utilizar para la cola compartida. La cola que especifica ya debe existir físicamente antes de crear la suscripción.

Para seleccionar la propuesta de cola múltiple, especifique un nombre de cola que finalice con el carácter *. Posteriormente, todos los suscriptores que se creen con este nombre de cola crean una cola dinámica adecuada, para que la utilice exclusivamente el suscriptor indicado. MQ JMS utiliza una cola modelo interna propia para crear dichas colas. Por consiguiente, con la propuesta de cola múltiple, todas las colas necesarias se crean de modo dinámico.

Cuando utiliza la propuesta de cola múltiple, no puede especificar un nombre de cola explícito, pero puede especificar el prefijo de la cola, lo que le permite crear dominios de cola de suscriptores diferentes. Por ejemplo, puede utilizar:

SYSTEM.JMS.ND.MYDOMAIN.*

Los caracteres que preceden al carácter * se utilizan como prefijo, de modo que todas las colas dinámicas que están asociadas a esta suscripción tendrán nombres de cola que empiecen por SYSTEM.JMS.ND.MYDOMAIN.

Configuración de suscriptores duraderos

Tal como se ha indicado anteriormente, sigue habiendo buenas razones para utilizar la propuesta de cola múltiple para suscripciones duraderas. Es probable que las suscripciones duraderas tengan una mayor duración, por lo que existe la posibilidad de que se acumule un mayor número de mensajes no recuperados en la cola.

Por este motivo, la propiedad del nombre de la cola de suscriptores duraderos se establece en el objeto Topic (es decir, a un nivel más gestionable que el de TopicConnectionFactory), lo que permite que se puedan especificar varios nombres de cola de suscriptores diferentes, sin que sea necesario volver a crear varios objetos que se inicien desde TopicConnectionFactory.

Puede establecer el nombre de la cola de suscriptores duraderos utilizando cualquiera de los procedimientos siguientes:

- Utilizando la herramienta de administración MQ JMS (para objetos recuperados JNDI) para establecer la propiedad BROKERDURSUBQ
- Utilizando el método setBrokerDurSubQueue() en el programa:

```
// Establecer el nombre de la cola de suscriptores duraderos
MQTopic utilizando
// la propuesta de cola múltiple
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

  Después de inicializar el objeto Topic, se pasa en el método createDurableSubscriber() de TopicSession para crear la suscripción especificada:

```
// Crear un suscriptor duradero utilizando el tema (Topic) anterior
TopicSubscriber sub = new session.createDurableSubscriber
                                          (sportsTopic, "D_SUB_SPORT_001");
```

Para suscripciones duraderas, el nombre de cola que proporcione debe empezar por los caracteres siguientes:

SYSTEM.JMS.D.

Para seleccionar una propuesta de cola compartida, especifique un nombre de cola explícito, donde la cola nombrada sea la que se va a utilizar para la cola compartida. La cola que especifica ya debe existir físicamente antes de crear la suscripción.

Para seleccionar la propuesta de cola múltiple, especifique un nombre de cola que finalice con el carácter *. Posteriormente, todos los suscriptores que se creen con este nombre de cola crean una cola dinámica adecuada, para que la utilice exclusivamente el suscriptor indicado. MQ JMS utiliza una cola modelo interna propia para crear dichas colas. Por consiguiente, con la propuesta de cola múltiple, todas las colas necesarias se crean de modo dinámico.

Cuando utiliza la propuesta de cola múltiple, no puede especificar un nombre de cola explícito, pero puede especificar el prefijo de la cola, lo que le permite crear dominios de cola de suscriptores diferentes. Por ejemplo, puede utilizar:

SYSTEM.JMS.D.MYDOMAIN.*

Los caracteres que preceden al carácter * se utilizan como prefijo, de modo que todas las colas dinámicas que están asociadas a esta suscripción tendrán nombres de cola que empiecen por SYSTEM.JMS.D.MYDOMAIN.

Cuestiones relacionadas con la nueva creación y la migración para suscriptores duraderos

Para un suscriptor duradero, no se debe intentar volver a configurar el nombre de la cola de suscriptores antes de suprimir el suscriptor. Es decir, debe realizar una unsubscribe() y, a continuación, volver a crear la cola desde el principio (recuerde que se suprimen todos los mensajes del suscriptor anterior).

Sin embargo, si ha creado un suscriptor utilizando MQ JMS V1.1, se le reconoce al migrar al nivel actual. No es necesario suprimir la suscripción, puesto que sigue operando con la propuesta de cola múltiple.

Solución de problemas de Publish/Subscribe

En este apartado se describen algunos de los problemas que se pueden producir al desarrollar aplicaciones cliente JMS que utilizan el dominio de publicación/suscripción. Tenga en cuenta que en este apartado no se tratan los problemas específicos del dominio de publicación/suscripción. En los apartados “Manejo de errores” en la página 191 y “Solución de problemas” en la página 33 se proporcionan instrucciones para la solución de problemas generales.

Cierre incompleto de Publish/Subscribe

Es importante que las aplicaciones cliente JMS entreguen todos los recursos externos cuando terminen. Para hacerlo, invoque el método `close()` en todos los objetos que se puedan cerrar cuando ya no se necesiten. Para el dominio de publicación/suscripción, dichos objetos son los siguientes:

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

La implementación del Servicio de mensajes de MQSeries Classes for Java simplifica la tarea al utilizar un ″cierre en cascada″. Con este proceso, el resultado de una invocación a ’close’ en una TopicConnection son invocaciones a ’close’ en todas las TopicSessions que ha creado, lo que, a su vez, provoca invocaciones a ’close’ en todos los TopicSubscribers y TopicPublishers creados.

Por consiguiente, para asegurar que se liberen adecuadamente los recursos externos, es importante invocar a `connection.close()` para cada una de las conexiones que crea una aplicación.

Existen algunas circunstancias en las que el procedimiento ’close’ puede no finalizar. Incluyen las siguientes:

- Pérdida de una conexión cliente MQSeries
- Terminación inesperada de una aplicación

En estas circunstancias, no se invoca a `close()` y los recursos externos permanecen abiertos en nombre de la aplicación terminada. Las principales consecuencias son:

Incoherencia del estado del intermediario
: El intermediario de mensajes MQSeries puede contener información de registro de suscriptores y publicadores que ya no existan, lo que significa que el intermediario puede seguir enviando mensajes a suscriptores que no los recibirán nunca.

Permanencia de colas y mensajes de suscriptores
: Parte del procedimiento de anulación de registros de suscriptores consiste en la eliminación de mensajes de suscriptor. Si procede, también se elimina la cola MQSeries subyacente que se ha utilizado para recibir suscripciones. Si no se lleva a cabo el cierre normal, las colas y los mensajes permanecen. Si existe incoherencia del estado del intermediario, las colas se siguen rellenando con mensajes que no se van a leer nunca.

Programa de utilidad de limpieza de suscriptores

Para evitar los problemas asociados al cierre incorrecto de objetos de suscriptor, MQ JMS incluye un programa de utilidad de limpieza de suscriptores. Este programa de utilidad se ejecuta en un gestor de colas al inicializar la primera TopicConnection para utilizar el gestor de colas físico. Si se cierran todas las

TopicConnections del gestor de colas especificado, al inicializar la TopicConnection siguiente para el gestor de colas, el programa de utilidad se vuelve a ejecutar.

El programa de utilidad de limpieza trata de detectar cualquier posible problema de publicación/suscripción de MQ JMS anterior que se ha podido producir desde otras aplicaciones. Si detecta problemas, limpia los recursos asociados:

- anulando el registro para el intermediario de mensajes MQSeries
- limpiando todos los mensajes no recuperados y las colas asociadas a la suscripción

El programa de utilidad de limpieza se ejecuta de modo transparente en segundo plano y su duración es breve. No debe afectar a las demás operaciones de MQ JMS. Si se detectan muchos problemas para un gestor de colas determinado, se puede producir un breve retardo en el momento de la inicialización mientras se limpian los recursos.

Nota: Se recomienda encarecidamente que, siempre que sea posible, se cierren correctamente todos los objetos de suscriptor para evitar que se produzcan problemas con los suscriptores.

Manejo de informes del intermediario

La implementación MQ JMS utiliza mensajes de informe del intermediario para confirmar mandatos de registro y anulación de registro. Normalmente, la implementación del Servicio de mensajes de MQSeries Classes for Java consume dichos registros, pero bajo determinadas condiciones de error, pueden permanecer en la cola. Estos mensajes se envían a la cola `SYSTEM.JMS.REPORT.QUEUE` del gestor de colas local.

Con Servicio de mensajes de MQSeries Classes for Java se proporciona una aplicación Java™, `PSReportDump`, que vuelca el contenido de esta cola en texto sin formato. Después, el usuario o el personal de soporte de IBM® pueden analizar la información. También puede utilizar la aplicación para borrar la cola de mensajes después de diagnosticar o arreglar un problema.

El formato compilado de la herramienta se instala en el directorio `<VÍA_ACCESO_INSTALACIÓN_MQ_JAVA>/bin`. Para invocar la herramienta, cambie a este directorio y utilice el mandato siguiente:

```
java PSReportDump [-m gestorColas] [-clear]
```

donde:

-m gestorColas
= especifica el nombre del gestor de colas que se va a utilizar

-clear = borrar la cola de mensajes después de volcar su contenido

La salida se envía a la pantalla, o se puede redirigir a un archivo.

Capítulo 12. Mensajes JMS

Los mensajes JMS constan de las partes siguientes:

Cabecera Todos los mensajes ofrecen soporte para el mismo conjunto de campos de cabecera. Los campos de cabecera contienen valores que utilizan tanto los clientes como los suministradores para identificar y direccionar mensajes.

Propiedades Cada mensaje contiene un recurso incorporado para ofrecer soporte para los valores de propiedad que define la aplicación. Las propiedades facilitan un mecanismo eficaz para filtrar los mensajes que define la aplicación.

Cuerpo JMS define varios tipos de cuerpos de mensaje que abarcan la mayoría de los estilos de envío de mensajes que se utilizan actualmente.

JMS define cinco tipos de cuerpos de mensaje:

Corriente de datos (Stream)
corriente de datos de valores de primitivos de Java™. Se rellena y lee de modo secuencial.

Correlación (Map)
conjunto de pares de nombre-valor, donde los nombres son series de caracteres y los valores son tipos de primitivos de Java™. Se puede acceder a las entradas de modo secuencial o aleatorio, por nombre. El orden de las entradas no está definido.

Texto (Text) mensaje que contiene una java.util.String.

Objeto (Object)
mensaje que contiene un objeto Java Serializable

Bytes corriente de datos de bytes no interpretados. Este tipo de mensaje sirve para codificar literalmente un cuerpo para que coincida con un formato de mensaje existente.

El campo de cabecera JMSCorrelationID se utiliza para enlazar un mensaje con otro. Generalmente, enlaza un mensaje de respuesta con su mensaje de solicitud. JMSCorrelationID puede mantener el ID de mensaje específico de un suministrador, una serie de caracteres específica de la aplicación o un valor de byte[] nativo del suministrador.

Selectores de mensaje

Un mensaje contiene un recurso incorporado para ofrecer soporte para los valores de propiedad que define la aplicación. De hecho, proporciona un mecanismo que permite añadir campos de cabecera específicos de la aplicación a un mensaje. Las propiedades permiten que una aplicación tenga, a través de los selectores de mensaje, un suministrador de JMS que seleccione o filtre mensajes en su nombre, utilizando criterios específicos de la aplicación. Las propiedades que define la aplicación deben cumplir las normas siguientes:

- Los nombres de propiedad deben cumplir las normas de un identificador de selector de mensajes.
- Los valores de propiedad pueden ser boolean, byte, short, int, long, float, double y string.
- Los prefijos de nombre siguientes están reservados: JMSX y JMS_.

Los valores de las propiedades se establecen antes de enviar un mensaje. Cuando un cliente recibe un mensaje, las propiedades del mensaje son de sólo lectura. Si un cliente intenta establecer propiedades en este punto, se emite una MessageNotWriteableException. Si se invoca clearProperties, las propiedades pueden ser de lectura y escritura.

Un valor de propiedad puede duplicar un valor del cuerpo de un mensaje, o puede no hacerlo. JMS no define ninguna política que determine qué debe o no convertirse. Sin embargo, los desarrolladores de aplicaciones deben tener en cuenta que es probable que los suministradores de JMS manejen más eficazmente los datos del cuerpo del mensaje que los datos de las propiedades del mensaje. Para mejorar el rendimiento, las aplicaciones sólo deben utilizar propiedades de mensaje cuando deban personalizar la cabecera de un mensaje. La razón fundamental para ello es ofrecer soporte para la selección personalizada de mensajes.

Un selector de mensajes JMS permite que un cliente especifique los mensajes en los que está interesado, utilizando la cabecera de mensaje. Sólo se entregan los mensajes cuyas cabeceras coinciden con el selector.

Los selectores de mensaje no pueden hacer referencia a valores de cuerpo de mensaje.

Un selector de mensajes coincide con un mensaje cuando el selector seevalúa como verdadero (true) cuando se sustituyen los valores de propiedad y el campo de cabecera del mensaje por sus identificadores correspondientes en el selector.

Un selector de mensajes es una serie de caracteres (String), cuya sintaxis se basa en un subconjunto de la sintaxis de la expresión condicional SQL92. El orden en que se evalúa un selector de mensajes va de izquierda a derecha en el nivel de prioridad. Se pueden utilizar paréntesis para cambiar este orden. Los literales de selector y los nombres de operador predefinidos se escriben aquí en mayúsculas, aunque no son sensibles a las mayúsculas y minúsculas.

Un selector puede contener:

- Literales
  - Un literal de serie de caracteres se incluye entre comillas simples. Una comilla simple doble representa una comilla simple, por ejemplo, 'literal' y 'literal''s'. Como en el caso de los literales de serie de caracteres Java™, se utiliza la codificación de caracteres Unicode.
  - Un literal numérico exacto es un valor numérico sin coma decimal como, por ejemplo, 57, -957, +62. Se ofrece soporte para los números del rango de número largo (long) Java™.
  - Un literal numérico aproximado es un valor numérico en notación científica como, por ejemplo, 7E3 o -57.9E2, o un valor numérico con un decimal, como, por ejemplo, 7, -95,7 o +6,2. Se ofrece soporte para los números del rango doble (double) Java™.
  - Los literales booleanos TRUE y FALSE.
- Identificadores:

  - Un identificador es una secuencia de longitud ilimitada de letras Java™ y dígitos Java™, en la que en primer lugar debe haber una letra Java™. Una letra es cualquier carácter para el que el método Character.isJavaLetter devuelve verdadero (true). Incluye el carácter '_' y el símbolo '$'. Una letra o un dígito es cualquier carácter para el que el método Character.isJavaLetterOrDigit devuelve verdadero.
  - Los nombres NULL, TRUE o FALSE no pueden ser identificadores.
  - NOT, AND, OR, BETWEEN, LIKE, IN e IS no pueden ser identificadores.
  - Los identificadores son referencias de campo de cabecera o referencias de propiedades.
  - Los identificadores son sensibles a las mayúsculas y minúsculas.
  - Las referencias de los campos de cabecera de mensaje están restringidas a:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    Los valores JMSMessageID, JMSTimestamp, JMSCorrelationID y JMSType pueden ser nulos y, en este caso, se tratan como un valor NULL.
  - Todos los nombres que empiecen por 'JMSX' son nombres de propiedades definidos por JMS.
  - Todos los nombres que empiecen por 'JMS_' son nombres de propiedades específicos del suministrador.
  - Todos los nombres que no empiecen por 'JMS' son nombres de propiedades específicos de la aplicación. Si hay alguna referencia a una propiedad que no exista en un mensaje, su valor es NULL. Si existe, su valor es el de la propiedad correspondiente.
- El espacio en blanco equivale a lo que se ha definido para Java™: espacio, separador horizontal, salto de página y terminador de línea.
- Expresiones:
  - Un selector es una expresión condicional. Un selector que se evalúa como verdadero (true) coincide, y cuando lo hace como falso (false) o desconocido (unknown) no coincide.
  - Las expresiones aritméticas se componen de sí mismas, operaciones aritméticas, identificadores (cuyo valor se trata como un literal numérico) y literales numéricos.
  - Las expresiones condicionales se componen de sí mismas, operaciones de comparación y operaciones lógicas.
- Se ofrece soporte para corchetes estándar () para establecer el orden en que se evalúan las expresiones.
- Operadores lógicos por orden de prioridad: NOT, AND y OR.
- Operadores de comparación: =, >, >=, <, <=, <> (no igual).
  - Sólo se pueden comparar valores del mismo tipo, con la excepción de que se pueden comparar valores numéricos exactos y valores numéricos aproximados. (Las normas de promoción numérica de Java™ definen el tipo de conversión necesario). Si se intentan comparar tipos diferentes, el selector siempre es falso (false).

  - La comparación de serie de caracteres (String) y booleana está restringida a = y <>. Dos series de caracteres son iguales si, y sólo si, contienen la misma secuencia de caracteres.
- Operadores aritméticos por orden de prioridad:
  - +, - unario.
  - *, /, multiplicación y división.
  - +, -, suma y resta.
  - No se ofrece soporte para operaciones aritméticas en un valor NULL. Si se intentan, el selector completo siempre es falso.
  - Las operaciones aritméticas deben utilizar promoción numérica Java™.
- Operador de comparación expr-aritm1 [NOT] BETWEEN expr-aritm2 y expr-aritm3:
  - edad BETWEEN 15 y 19 es equivalente a edad >= 15 AND edad <= 19.
  - edad NOT BETWEEN 15 y 19 es equivalente a edad < 15 OR edad > 19.
  - Si alguna de las expresiones de una operación BETWEEN es NULL, el valor de la operación es falso (false). Si alguna de las expresiones de una operación NOT BETWEEN es NULL, el valor de la operación es verdadero (true).
- Operador de comparación identificador [NOT] IN (literal-serie1, literal-serie2,...), donde el identificador tiene un valor de serie (String) o NULL.
  - País IN (' ReinoUnido', 'EEUU', 'Francia') es verdadero para 'ReinoUnido' y falso para 'Perú'. Equivale a la expresión (País = ' ReinoUnido') OR (País = ' EEUU') OR (País = ' Francia').
  - País NOT IN (' ReinoUnido', 'EEUU', 'Francia') es falso para 'EEUU' y verdadero para 'Perú'. Equivale a la expresión NOT ((País = ' ReinoUnido') OR (País = ' EEUU') OR (País = ' Francia')).
  - Si el identificador de una operación IN o NOT IN es NULL, el valor de la operación es desconocido (unknown).
- Operador de comparación identificador [NOT] valor de patrón LIKE, carácter de escape [ESCAPE], en el que el identificador tiene un valor de serie de caracteres (String), el valor-patrón es un literal de serie de caracteres, donde '_' representa cualquier carácter y '%' representa cualquier secuencia de caracteres (incluida la secuencia vacía). Todos los demás caracteres se representan a sí mismos. El carácter de escape opcional es un literal de serie de un solo carácter, cuyo carácter se utiliza para alejarse del significado especial de '_' y '%' en el valor-patrón.
  - teléfono LIKE '12%3' es verdadero para '123' '12993' y falso para '1234'.
  - palabra LIKE 'l_se' es verdadero para 'lose' y falso para 'loose'.
  - subrayado LIKE '_%' ESCAPE '\' es verdadero para '_foo' y falso para 'bar'.
  - teléfono NOT LIKE '12%3' es falso para '123' '12993' y verdadero para '1234'.
  - Si el identificador de una operación LIKE o NOT LIKE es NULL, el valor de la operación es desconocido.
- El operador de comparación identificador IS NULL comprueba un valor de campo de cabecera nulo o un valor de propiedad no encontrado.
  - nombre_prop IS NULL.
- El operador de comparación identificador IS NOT NULL comprueba la existencia de un valor de campo de cabecera no nulo o un valor de propiedad.
  - nombre_prop IS NOT NULL.

El selector de mensajes siguiente selecciona mensajes con un tipo de mensaje de vehículo, color azul y de peso superior a 2500 libras:

```
"JMSType = 'vehículo' AND color = 'azul' AND peso > 2500"
```

Tal como se ha indicado anteriormente, los valores de propiedad pueden ser NULL. La semántica SQL 92 NULL define la evaluación de expresiones de selector que contengan valores NULL. A continuación, se describe brevemente dicha semántica:

- SQL trata un valor NULL como desconocido (unknown).
- El resultado de una comparación o aritmética con un valor desconocido siempre es un valor desconocido.
- Los operadores IS NULL e IS NOT NULL convierten un valor desconocido en los valores TRUE y FALSE respectivos.

Aunque SQL ofrece soporte para la aritmética y la comparación de decimales fijos, los selectores de mensaje JMS no. Éste es el motivo por el que se restringen los literales numéricos exactos a los que no tienen ningún decimal, y también es el motivo por el que hay numerales con un decimal como representación alternativa para un valor numérico aproximado.

No se ofrece soporte para comentarios SQL.

Correlación de mensajes JMS en mensajes MQSeries

En este apartado se describe cómo se correlaciona la estructura de mensajes JMS que se trata en la primera parte de este capítulo en un mensaje MQSeries®. Puede resultar interesante para los programadores que deseen transmitir mensajes entre aplicaciones MQSeries tradicionales y JMS. También les puede interesar a los usuarios que deseen manipular mensajes transmitidos entre dos aplicaciones JMS, por ejemplo, en una implementación de intermediario de mensajes.

Los mensajes MQSeries constan de tres componentes:
- MQMD (Descriptor de mensaje MQSeries)
- Una cabecera MQRFH2 MQSeries
- El cuerpo del mensaje.

MQRFH2 es opcional y su inclusión en un mensaje saliente se rige por un distintivo en la clase JMS Destination. Puede establecer este distintivo utilizando la herramienta de administración MQSeries JMS. Puesto que MQRFH2 contiene información específica de JMS, siempre se incluye en el mensaje cuando el emisor sabe que el destino receptor es una aplicación JMS. Por lo general, se puede omitir MQRFH2 cuando se envía un mensaje directamente a una aplicación que no es JMS (MQSeries Native application), puesto que dicha aplicación no espera un MQRFH2 en el mensaje MQSeries. En la Figura 4 se muestra la transformación de las estructuras:

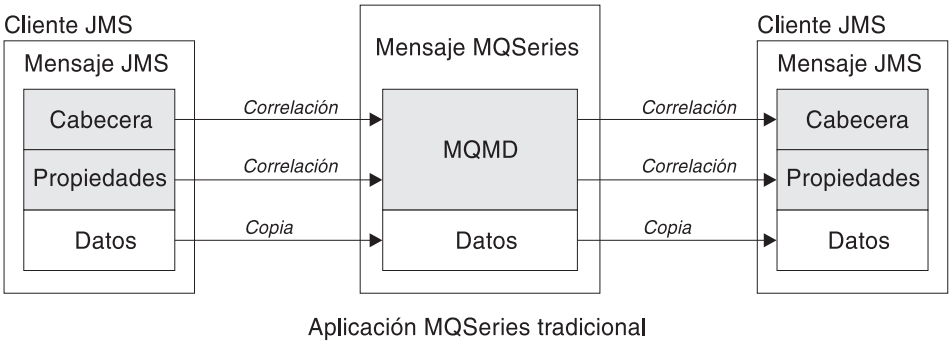


Figura 4. Modelo de correlación de JMS con MQSeries

Las estructuras se transforman de dos modos:

Correlación
Donde MQMD incluye un campo que equivale al campo JMS, el campo JMS se correlaciona con el campo MQMD. Los campos de MQMD adicionales se muestran como propiedades de JMS, puesto que es posible que una aplicación JMS necesite obtener o establecer dichos campos cuando se comunique con una aplicación que no sea JMS.

Copia Donde no existe equivalente MQMD, se pasa una propiedad o campo de cabecera JMS y, posiblemente, se transforma como campo dentro de MQRFH2.

Cabecera MQRFH2

En este apartado se describe la cabecera MQRFH Versión 2, que contiene datos específicos de JMS asociados al contenido del mensaje. MQRFH2 Versión 2 es una cabecera ampliable y también puede contener información adicional que no esté directamente asociada a JMS. No obstante, en este apartado sólo se trata su utilización por parte de JMS.

La cabecera consta de dos partes, una parte fija y otra variable.

Parte fija

La parte fija se modela en el patrón de cabecera MQSeries 'estándar' y consta de los campos siguientes:

StrucId (MQCHAR4)

Identificador de estructura.

Debe ser MQRFH_STRUC_ID (valor: ″RFH ″) (valor inicial)

MQRFH_STRUC_ID_ARRAY (valor: 'R','F','H',' ') también se define utilizando el procedimiento habitual.

Versión (MQLONG)

Número de versión de la estructura.

Debe ser MQRFH_VERSION_2 (valor: 2) (valor inicial)

StrucLength (MQLONG)

Longitud total de MQRFH2, incluidos los campos NameValueData.

El valor establecido en StrucLength debe ser un múltiplo de 4 (para ello, los datos de los campos NameValueData se pueden rellenar con caracteres de espacio).

Encoding (MQLONG)

Codificación de datos.

Codificación de todos los datos numéricos de la parte del mensaje que sigue de MQRFH2 (la cabecera siguiente o los datos de mensaje situados después de esta cabecera).

CodedCharSetId (MQLONG)

Identificador de juego de caracteres codificado.

Representación de todos los datos de carácter de la parte del mensaje que sigue a MQRFH2 (la cabecera siguiente o los datos de mensaje situados después de esta cabecera).

Format (MQCHAR8)

Nombre del formato.

Nombre del formato de la parte del mensaje situada después de MQRFH2.

Flags (MQLONG)

Distintivos.

MQRFH_NO_FLAGS =0. No se ha establecido ningún distintivo.

NameValueCCSID (MQLONG)

CCSID (identificador de juego de caracteres codificado) para las series de caracteres NameValueData contenidas en esta cabecera. NameValueData se puede codificar en un juego de caracteres que difiera de las demás series de caracteres contenidas en la cabecera (StrucID y Format).

Si NameValueCCSID es un CCSID Unicode de 2 bytes (1200, 13488 ó 17584), el orden de bytes de Unicode es el mismo que la disposición de bytes en los campos numéricos de MQRFH2. (Por ejemplo, Version, StrucLength y NameValueCCSID).

NameValueCCSID sólo puede tomar valores de la lista siguiente:

1200	UCS2 extremo abierto
1208	UTF8
13488	Subconjunto UCS2 2.0
17584	Subconjunto UCS2 2.1 (incluye el símbolo del Euro)

Parte variable

La parte variable sigue a la parte fija. Esta parte contiene un número variable de carpetas MQRFH2. Cada carpeta contiene un número variable de elementos o propiedades. Las carpetas agrupan propiedades relacionadas. Las cabeceras MQRFH2 que crea JMS puede contener hasta tres carpetas:

Carpeta <mcd>

Contiene propiedades que describen la 'forma' o el 'formato' del mensaje. Por ejemplo, la propiedad `msd` identifica el mensaje como texto (Text), bytes (Bytes), corriente de datos (Stream), correlación (Map), objeto (Object) o 'Null'. Esta carpeta siempre está presente en JMS MQRFH2.

Carpeta <jms>

Se utiliza para transportar campos de cabecera JMS y propiedades JMSX que no se pueden expresar completamente en MQMD. Esta carpeta siempre está presente en JMS MQRFH2.

Carpeta <usr>

Se utiliza para transportar todas las propiedades definidas por la aplicación asociadas al mensaje. Esta carpeta sólo está presente si la aplicación establece algunas propiedades definidas por aplicación.

En la Tabla 17 se muestra una lista completa de los nombres de propiedades.

Tabla 17. Propiedades y carpetas MQRFH2 que utiliza JMS

Campos JMS		**Campos MQRFH2**		
Nombre	**Tipo Java™**	**Nombre carpeta**	**Nombre propiedad**	**Valores/tipo**
JMSDestination	Destination	jms	Dst	serie caracteres
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	Cid	serie caracteres
JMSReplyTo	Destination	jms	Rto	serie caracteres
JMSType	String	mcd	Type	serie caracteres
JMSXGroupID	String	jms	Gid	serie caracteres
JMSXGroupSeq	int	jms	Seq	i4
xxx (Definido usuario)	Cualquiera	usr	xxx	cualquiera

Tabla 17. Propiedades y carpetas MQRFH2 que utiliza JMS (continuación)

Campos JMS		Campos MQRFH2		
Nombre	**Tipo Java™**	**Nombre carpeta**	**Nombre propiedad**	**Valores/tipo**
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

La sintaxis que se utiliza para expresar las propiedades de la parte variable es la siguiente:

NameValueLength (MQLONG)
Longitud en bytes de la serie de caracteres NameValueData que sigue inmediatamente a este campo de longitud (no incluye su longitud propia). El valor establecido en NameValueLength siempre es un múltiplo de 4 (para ello, el campo NameValueData se rellena con caracteres de espacio).

NameValueData (MQCHARn)
Serie de un solo carácter, para la que el campo NameValueLength anterior establece la longitud en bytes. Contiene una 'carpeta' que retiene una secuencia de 'propiedades'. Cada propiedad es un trío de 'nombre/tipo/valor' contenido en un elemento XML cuyo nombre es el nombre de la carpeta, tal como se muestra a continuación:

```
<nombrecarpeta> trío1 trío2 .....   tríon </nombrecarpeta>
```

El código `</nombrecarpeta>` de cierre puede ir seguido de espacios como caracteres de relleno. Cada trío se codifica utilizando una sintaxis similar a la de XML:

```
<nombre dt='tipodatos'>valor</nombre>
```

El elemento `dt='tipodatos'` es opcional y se omite para varias propiedades, puesto que tienen un tipo de datos predefinido. Si se incluye, se deben añadir uno o más caracteres de espacio antes del código `dt=`.

`nombre` es el nombre de la propiedad. Consulte la Tabla 17 en la página 212.

`tipodatos` debe coincidir, después de guardarlo en la carpeta, con uno de los valores de literal que se indican en la Tabla 18.

`valor` es una representación de serie de caracteres del valor que se va a transmitir, tal como se muestra en la Tabla 18.

Un valor nulo se codifica utilizando la sintaxis siguiente:

```
<nombre/>
```

Tabla 18. Tipos de datos y valores de propiedad

TipoDatos	Valor
serie	Cualquier secuencia de caracteres, excepto `<` y `&`
booleano	El carácter `0` ó `1` (1 = "true")

Tabla 18. Tipos de datos y valores de propiedad (continuación)

TipoDatos	Valor
bin.hex	Dígitos hexadecimales que representan octetos
i1	Un número, expresado utilizando los dígitos `0..9`, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -128 a 127 inclusive
i2	Un número, expresado utilizando los dígitos `0..9`, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -32768 a 32767 inclusive
i4	Un número, expresado utilizando los dígitos `0..9`, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -2147483648 a 2147483647 inclusive
i8	Un número, expresado utilizando los dígitos `0..9`, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del rango de -9223372036854775808 a 9223372036854775807 inclusive
int	Un número, expresado utilizando los dígitos 0..9, con símbolo opcional (ni fracciones ni exponentes). Debe situarse dentro del mismo rango que 'i8'. Se puede utilizar en lugar de los tipos 'i*' si el emisor no desea asociar una precisión específica a una propiedad
r4	Número de coma flotante, tamaño <= 3.40282347E+38, >= 1.175E-37 expresado utilizando dígitos de 0..9, símbolo opcional, dígitos de fracción opcionales y exponente opcional
r8	Número de coma flotante, tamaño <= 1.7976931348623E+308, >= 2.225E-307 expresado utilizando dígitos de 0..9, símbolo opcional, dígitos de fracción opcionales y exponente opcional

Un valor de serie de caracteres puede contener espacios. En un valor de serie de caracteres, debe utilizar las secuencias de escapada siguientes:

`&` para el carácter &
`<` para el carácter <

Puede utilizar las secuencias de escapada siguientes, pero no son obligatorias:

`>` para el carácter >
`&apos;` para el carácter '
`&quot;` para el carácter "

Campos y propiedades JMS con los campos MQMD correspondientes

En la Tabla 19 se enumeran las propiedades que se correlacionan directamente con los campos MQMD.

Tabla 19. Correlación de propiedades JMS con campos MQMD

Campo JMS		Campo MQMD	
Cabecera	**Tipo Java™**	**Campo**	**Tipo C**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiry	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	String	MessageID	MQBYTE24

Tabla 19. Correlación de propiedades JMS con campos MQMD (continuación)

Campo JMS		**Campo MQMD**	
Cabecera	**Tipo Java™**	**Campo**	**Tipo C**
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	String	CorrelId	MQBYTE24
Propiedades			
JMSXUserID	String	UserIdentifier	MQCHAR12
JMSXAppID	String	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	String	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG
Específico del suministrador			
JMS_IBM_Report_Exception	int	Report	MQLONG
JMS_IBM_Report_Expiration	int	Report	MQLONG
JMS_IBM_Report_COA	int	Report	MQLONG
JMS_IBM_Report_COD	int	Report	MQLONG
JMS_IBM_Report_PAN	int	Report	MQLONG
JMS_IBM_Report_NAN	int	Report	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Report	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Report	MQLONG
JMS_IBM_Report_Discard_Msg	int	Report	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	String	Format	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	String	CodedCharacterSetId	MQLONG

Correlación de campos JMS con campos MQSeries (mensajes salientes)

En la Tabla 20 en la página 216 se muestra cómo se correlacionan los campos de propiedad/cabecera con los campos MQMD/RFH2 en el momento del envío (send()) o la publicación (publish()).

Para los campos marcados con 'Establecido por Objeto de mensaje', el valor que se transmite es el valor que se mantiene en el mensaje JMS inmediatamente antes de send/publish(). send/publish() no modifica el valor del mensaje JMS.

Para los campos marcados con 'Establecido por Método Send', se asigna un valor cuando se ejecuta send/publish() (se ignoran todos los valores mantenidos en el mensaje JMS). Se actualiza el valor del mensaje JMS para mostrar el valor utilizado.

Los campos marcados con 'Sólo recepción' no se transmiten y send() o publish() no los modifican en el mensaje.

Tabla 20. Correlación de los campos de mensajes salientes

Campos JMS	**Transmitido en**		**Establecido por**
Nombre	**Campo MQMD**	**Cabecera**	
JMSDestination		MQRFH2	Método Send
JMSDeliveryMode	Persistence	MQRFH2	Método Send
JMSExpiration	Expiry	MQRFH2	Método Send
JMSPriority	Priority	MQRFH2	Método Send
JMSMessageID	MessageID		Método Send
JMSTimestamp	PutDate/PutTime		Método Send
JMSCorrelationID	CorrelId	MQRFH2	Objeto de mensaje
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Objeto de mensaje
JMSType		MQRFH2	Objeto de mensaje
JMSRedelivered			Sólo recepción
Propiedades			
JMSXUserID	UserIdentifier		Método Send
JMSXAppID	PutApplName		Método Send
JMSXDeliveryCount			Sólo recepción
JMSXGroupID	GroupId	MQRFH2	Objeto de mensaje
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Objeto de mensaje
Específico del suministrador			
JMS_IBM_Report_Exception	Report		Objeto de mensaje
JMS_IBM_Report_Expiration	Report		Objeto de mensaje
JMS_IBM_Report_COA/COD	Report		Objeto de mensaje
JMS_IBM_Report_NAN/PAN	Report		Objeto de mensaje
JMS_IBM_Report_Pass_Msg_ID	Report		Objeto de mensaje
JMS_IBM_Report_Pass_Correl_ID	Report		Objeto de mensaje
JMS_IBM_Report_Discard_Msg	Report		Objeto de mensaje
JMS_IBM_MsgType	MsgType		Objeto de mensaje
JMS_IBM_Feedback	Feedback		Objeto de mensaje
JMS_IBM_Format	Format		Objeto de mensaje
JMS_IBM_PutApplType	PutApplType		Método Send
JMS_IBM_Encoding	Encoding		Objeto de mensaje
JMS_IBM_Character_Set	CodedCharacterSetId		Objeto de mensaje

Correlación de los campos de cabecera durante el envío/publicación (send()/publish())

Las notas siguientes están relacionadas con la correlación de los campos JMS durante el envío/publicación (send()/publish()):

- **JMS Destination a MQRFH2:** Se almacena como una serie de caracteres que serializa las características más destacadas del objeto de destino, de modo que un JMS receptor pueda reconstituir un objeto destino equivalente. El campo

MQRFH2 se codifica como URI (para obtener información más detallada sobre la notación URI, consulte el apartado “identificadores de recursos uniformes” en la página 187).

- **JMSReplyTo a MQMD ReplyToQ, ReplyToQMgr, MQRFH2:** El nombre de cola (Queue) y del gestor de colas (QueueManager) se copian en los campos MQMD ReplyToQ y ReplyToQMgr respectivamente. La información de la extensión de destino se copia en el campo MQRFH2 (otros detalles ″útiles″ se mantienen en el objeto de destino). El campo MQRFH2 se codifica como URI (para obtener información más detallada sobre la notación URI, consulte el apartado “identificadores de recursos uniformes” en la página 187).
- **JMSDeliveryMode a MQMD Persistence:** MessageProducer o el método send/publish() establecen el valor JMSDeliveryMode, a menos que lo altere temporalmente el objeto de destino. El valor JMSDeliveryMode se correlaciona con el campo MQMD Persistence, tal como se indica a continuación:
  - El valor JMS PERSISTENT es equivalente a MQPER_PERSISTENT
  - El valor JMS NON_PERSISTENT es equivalente a MQPER_NOT_PERSISTENT

  Si se establece JMSDeliveryMode es un valor que no sea el valor por omisión, el valor de la modalidad de entrega también se codifica en MQRFH2.
- **JMSExpiration de/a MQMD Expiry, MQRFH2:** JMSExpiration almacena el tiempo de caducidad (la suma de la hora actual y el tiempo de vida), mientras que MQMD almacena el tiempo de vida. Además, JMSExpiration está en milisegundos, pero MQMD.expiry está en centisegundos.
  - Si el método send() establece un tiempo de vida ilimitado, MQMD Expiry se establece en MQEI_UNLIMITED y no se codifica ninguna JMSExpiration en MQRFH2.
  - Si el método send() establece un tiempo de vida inferior a 214748364.7 segundos (7 años, aproximadamente), el tiempo de vida se almacena en MQMD. La caducidad y la hora de caducidad (en milisegundos) se codifica como un valor i8 en MQRFH2.
  - Si el método send() establece un tiempo de vida superior a 214748364.7 segundos, MQMD.Expiry se establece en MQEI_UNLIMITED. La hora de caducidad real en milisegundos se codifica como un valor i8 en MQRFH2.
- **JMSPriority a MQMD Priority:** Correlaciona directamente el valor JMSPriority (0-9) con el valor de prioridad MQMD (0-9). Si se establece JMSPriority en un valor que no sea el valor por omisión, el nivel de prioridad también se codifica en MQRFH2.
- **JMSMessageID de MQMD MessageID:** Todos los mensajes enviados desde JMS tienen identificadores de mensaje exclusivos asignados por MQSeries. El valor asignado se devuelve en el campo MQMD messageId después de la invocación de MQPUT y se vuelve a pasar a la aplicación en el campo JMSMessageID. MQSeries messageId es un valor binario de 24 bytes, mientras que JMSMessageID es una serie de caracteres (String). JMSMessageID consta de valor messageId binario convertido en una secuencia de 48 caracteres hexadecimales con los caracteres ’ID:’ como prefijo. JMS proporciona una indicación que se puede establecer para inhabilitar la producción de identificadores de mensaje. Dicha indicación se ignora y se asigna un identificado exclusivo en todos los casos. Todos los valores que se establecen en el campo JMSMessageId antes de realizar un envío (send()), se sobrescriben.
- **JMSTimestamp de MQMD PutDate, PutTime:** Después de un envío, el campo JMSTimestamp se establece igual al valor de fecha/hora que proporcionan los campos MQMD PutDate y PutTime. Todos los valores que se establecen en el campo JMSMessageId antes de realizar un envío (send()), se sobrescriben.

- **JMSType a MQRFH2:** Esta serie de caracteres se establece en MQRFH2.
- **JMSCorrelationID a MQMD CorrelId, MQRFH2:** JMSCorrelationID puede mantener uno de los siguientes:
  - **Un ID de mensaje específico del suministrador:** Éste es un identificador de un mensaje enviado o recibido previamente, por lo que debe ser una serie de caracteres de 48 dígitos hexadecimales con el prefijo 'ID:'. El prefijo se elimina, los caracteres restantes se convierten en binarios y, a continuación, se establecen en el campo MQMD CorrelId. En MQRFH2 no se codifica ningún valor correlid.
  - **Un valor byte[] nativo del suministrador:** El valor se copia en el campo MQMD CorrelId y se rellena con valores de nulo o se trunca en 24 bytes, según proceda. En MQRFH2 no se codifica ningún valor correlid.
  - **Una serie de caracteres específica de la aplicación:** El valor se copia en MQRFH2. Se escriben los 24 primeros bytes de la serie de caracteres, en formato UTF8, en MQMD CorrelID.

Correlación de campos de propiedades JMS

Estas notas hacen referencia a la correlación de los campos de propiedades JMS en los mensajes MQSeries:

- **JMSXUserID de MQMD UserIdentifier:** JMSXUserID se establece al volver de la invocación de envío.
- **JMSXAppID de MQMD PutApplName:** JSMXAppID se establece al volver de la invocación de envío.
- **JMSXGroupID a MQRFH2 (punto a punto):** Para mensajes punto a punto, JMSXGroupID se copia en el campo MQMD GroupID. Si JMSXGroupID empieza con el prefijo 'ID:', se convierte a binario. Si no es así, se codifica como una serie de caracteres UTF8. Si es necesario, se rellena o se trunca el valor en la longitud de 24 bytes. Se establece el distintivo MQF_MSG_IN_GROUP.
- **JMSXGroupID a MQRFH2 (publicación/suscripción):** Para mensajes de publicación/suscripción, se copia el JMSXGroupID en MQRFH2 como una serie de caracteres.
- **JMSXGroupSeq MQMD MsgSeqNumber (punto a punto):** Para los mensajes punto a punto, se copia JMSXGroupSeq en el campo MQMD MsqSeqNumber. Se establece el distintivo MQF_MSG_IN_GROUP.
- **JMSXGroupSeq MQMD MsgSeqNumber (publicación/suscripción):** Para los mensajes de publicación/suscripción, se copia JMSXGroupSeq en MQRFH2 como i4.

Correlación de campos específicos del suministrador de JMS

Las notas siguientes hacen referencia a la correlación de los campos específicos del suministrador de JMS con los mensajes MQSeries:

- **JMS_IBM_Report_<nombre> a MQMD Report:** Una aplicación JMS puede establecer las opciones MQMD Report utilizando las propiedades JMS_IBM_Report_XXX siguientes. El MQMD único se puede correlacionar con varias propiedades JMS_IBM_Report_XXX. La aplicación debe establecer el valor para dichas propiedades en las constantes MQSeries MQRO_ estándar (incluidas en com.ibm.mq.MQC). Así, por ejemplo, para solicitar COD con los datos completos, la aplicación debe establecer JMS_IBM_Report_COD en el valor MQC.MQRO_COD_WITH_FULL_DATA.

  JMS_IBM_Report_Exception

MQRO_EXCEPTION o
MQRO_EXCEPTION_WITH_DATA o
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION o
MQRO_EXPIRATION_WITH_DATA o
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA o
MQRO_COA_WITH_DATA o
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD o
MQRO_COD_WITH_DATA o
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN
MQRO_PAN

JMS_IBM_Report_NAN
MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

- **JMS_IBM_MsgType a MQMD MsgType:** El valor se correlaciona directamente con MQMD MsgType. Si la aplicación no ha establecido explícitamente un valor de JMS_IBM_MsgType, se utiliza el valor por omisión, que se determina tal como se indica a continuación:
  - Si se establece JMSReplyTo en un destino de cola MQSeries, MSGType se establece en el valor MQMT_REQUEST
  - Si no se establece JMSReplyTo, o si se establece en algo distinto a un destino de cola MQSeries, MsgType se establece en el valor MQMT_DATAGRAM
- **JMS_IBM_Feedback a MQMD Feedback:** El valor se correlaciona directamente en MQMD Feedback.
- **JMS_IBM_Format a MQMD Format:** El valor se correlaciona directamente en MQMD Format.
- **JMS_IBM_Encoding a MQMD Encoding:** Si se establece, esta propiedad altera temporalmente la codificación numérica de la cola de destino (Destination Queue) o del tema (Topic).
- **JMS_IBM_Character_Set a MQMD CodedCharacterSetId:** Si se establece, esta propiedad altera temporalmente la propiedad del juego de caracteres codificado de la cola de destino (Destination Queue) o del tema (Topic).

Correlación de campos MQSeries con campos JMS (mensajes entrantes)

En la Tabla 21 se muestran cómo se correlacionan los campos de propiedad/cabecera con los campos de MQMD/MQRFH2 en el momento del envío (send()) o la publicación (publish()).

Tabla 21. Correlación de los campos de mensajes entrantes

Campos JMS	**Recuperado de**	
Nombre	**Campo MQMD**	**MQRFH2**
Cabeceras JMS		
JMSDestination		jms.Dst
JMSDeliveryMode	Persistence	
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MessageID	
JMSTimestamp	PutDate PutTime	
JMSCorrelationID	CorrelId	jms.Cid
JMSReplyTo	ReplyToQ ReplyToQMgr	jms.Rto
JMSType		mcd.Type
JMSRedelivered	BackoutCount	
Propiedades JMS		
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId	jms.Gid
JMSXGroupSeq	MsgSeqNumber	jms.Seq
Específico del suministrador de JMS		
JMS_IBM_Report_Exception	Report	
JMS_IBM_Report_Expiration	Report	
JMS_IBM_Report_COA	Report	
JMS_IBM_Report_COD	Report	
JMS_IBM_Report_PAN	Report	
JMS_IBM_Report_NAN	Report	
JMS_IBM_Report_ Pass_Msg_ID	Report	
JMS_IBM_Report_Pass_Correl_ID	Report	
JMS_IBM_Report_Discard_Msg	Report	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding[1]	Encoding	
JMS_IBM_Character_Set[1]	CodedCharacterSetId	

1. Sólo se establece si el mensaje entrante es un mensaje de bytes.

Correlación de JMS con una aplicación MQSeries nativa

En este apartado se describe lo que ocurre si se envía un mensaje desde una aplicación cliente JMS a una aplicación MQSeries tradicional que desconozca las cabeceras MQRFH2. En la Figura 5 se muestra un diagrama de la correlación.

El administrador indica que el cliente JMS se está comunicando con este tipo de aplicación estableciendo el valor TargetClient de MQSeries Destination en JMSC.MQJMS_CLIENT_NONJMS_MQ. Indica que no se va a producir ningún campo MQRFH2.

La correlación de JMS a MQMD destinado a una aplicación MQSeries nativa es la misma que de JMS a MQMD destinado a un cliente JMS real. Si JMS recibe un mensaje MQSeries con el campo MQMD Format establecido en un valor que no sea MQFMT_RFH2, indica que los datos se reciben de una aplicación que no es JMS. Si el formato (Format) es MQFMT_STRING, el mensaje se recibe como un mensaje de texto JMS. Si no es así, se recibe como un mensaje de bytes JMS. Puesto que no existe MQRFH2, sólo se pueden restaurar las propiedades JMS que se transmiten en MQMD.

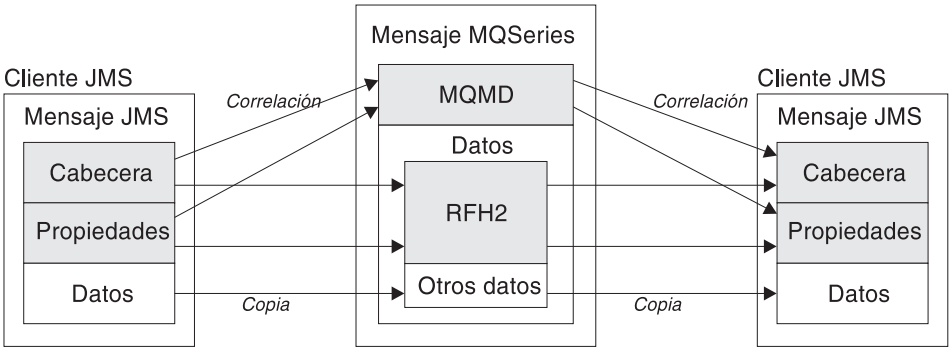


Figura 5. Modelo de correlación de JMS a MQSeries

Cuerpo del mensaje

En este apartado se trata la codificación del cuerpo del mensaje en sí. La codificación varía según el tipo de mensaje JMS:

ObjectMessage
es un objeto serializado por la Ejecución Java™ utilizando el procedimiento normal.

TextMessage
es una serie de caracteres codificada. Para un mensaje saliente, la serie de caracteres se codifica en el juego de caracteres que proporciona el objeto de destino (Destination). Toma como valor por omisión la codificación UTF8 (la codificación UTF8 empieza con el primer carácter del mensaje. No existe campo de longitud al principio). No obstante, se puede especificar cualquier juego de caracteres para el que MQ Java ofrezca soporte. Estos juegos de caracteres se utilizan principalmente cuando se envía un mensaje a una aplicación que no es JMS.

Si el juego de caracteres es un juego de doble byte (incluido UTF16), la especificación de codificación de enteros del objeto Destination determina el orden de los bytes.

Un mensaje entrante se interpreta utilizando el juego de caracteres y la codificación que se han especificado en el mensaje. Estas especificaciones se encuentran en la cabecera MQSeries situada más a la derecha (o MQMD, si no hay cabeceras). Para los mensajes JMS, la cabecera situada más a la derecha es, generalmente, MQRFH2.

BytesMessage
por omisión, es una secuencia de bytes tal como la define la especificación JMS 1.0.2 y la documentación de Java™.

Para un mensaje saliente que haya ensamblado la aplicación en sí, se puede utilizar la propiedad de codificación del objeto Destination para alterar temporalmente las codificaciones de los campos de coma flotante y entero contenidos en el mensaje. Por ejemplo, puede solicitar que los valores de coma flotante se almacenen en S/390®, en lugar de en formato IEEE).

Un mensaje entrante se interpreta utilizando la codificación numérica especificada en el mensaje. Esta especificación es la que se muestra en la cabecera MQSeries situada más a la derecha (o MQMDm si no hay cabeceras). Para los mensajes JMS, la cabecera situada más a la derecha es, generalmente, MQRFH2.

Si se recibe un BytesMessage y se vuelve a enviar sin realizar ninguna modificación, su cuerpo se transmite byte a byte, tal como se ha recibido. La propiedad de codificación del objeto Destination no tiene ningún efecto en el cuerpo. La única entidad similar a una serie de caracteres que se pueden enviar explícitamente en un BytesMessage es una serie de caracteres UTF8. Se codifica en formato UTF8 Java™ y empieza por un campo de 2 bytes de longitud. La propiedad del juego de caracteres del objeto Destination no tiene ningún efecto en la codificación de un BytesMessage saliente. El valor del juego de caracteres de un mensaje MQSeries entrante no tiene ningún efecto en la interpretación del mensaje como un BytesMessage de JMS.

Es poco probable que las aplicaciones que no son Java reconozcan la codificación UTF8 Java™. Por consiguiente, para que una aplicación JMS envíe un BytesMessage que contenga datos de texto, la aplicación en sí debe convertir sus series de caracteres en matrices de bytes y escribirlas en el BytesMessage.

MapMessage
es una serie de caracteres que contiene un conjunto de tríos nombre/tipo/valor XML, codificados como:

```
<map><nombreElemento1 dt='tipodatos'>valor</nombreElemento1>
<nombreElemento2 dt='tipodatos'>valor</nombreElemento2>.....
</map>
```

donde:
`tipodatos` puede tomar uno de los valores que se describen en la Tabla 18 en la página 213.
`string` es el tipo de datos por omisión, por lo que se omite `dt='string'`.

El juego de caracteres que se utiliza para codificar o interpretar la serie de caracteres XML que integra el cuerpo de MapMessage se determina siguiendo las normas que se aplican a un TextMessage.

StreamMessage
es similar a una correlación, pero sin nombres de elementos:

```
<stream><elt dt='tipodatos'>valor</elt>
<elt dt='tipodatos'>valor</elt>.....</stream>
```

Todos los elementos se envían utilizando el mismo nombre de código (elt). El tipo por omisión es serie de caracteres (string), por lo que se omite `dt='string'` para los elementos de serie de caracteres.

El juego de caracteres que se utiliza para codificar o interpretar la serie de caracteres XML que integra el cuerpo del StreamMessage se determina siguiendo las normas que se aplican a un TextMessage.

El campo MQRFH2.format se establece tal como se indica a continuación:

MQFMT_NONE
para ObjectMessage, BytesMessage o mensajes sin cuerpo.

MQFMT_STRING
para TextMessage, StreamMessage o MapMessage.

Correlación de mensajes JMS

Capítulo 13. Recursos de servidor de aplicaciones MQ JMS

MQ JMS V5.2 ofrece soporte para ASF (Recursos de servidor de aplicaciones) que se especifican en la especificación Java™ Message Service 1.0.2 (consulte el sitio web Java™ de Sun, que se encuentra en la dirección siguiente: `http://java.sun.com`). Esta especificación identifica tres funciones dentro de este modelo de programación:

- **El suministrador de JMS**, que proporciona ConnectionConsumer y funciones Session avanzadas.
- **El servidor de aplicaciones**, que proporciona las funciones ServerSessionPool y ServerSession.
- **La aplicación cliente**, que utiliza las funciones que proporcionan el suministrador de JMS y el servidor de aplicaciones.

Los apartados siguientes contienen información detallada acerca del modo en que MQ JMS implementa ASF:

- En "Clases y funciones de ASF", se describe cómo MQ JMS implementa la clase ConnectionConsumer y funciones avanzadas en la clase Session.
- En "Código de ejemplo del servidor de aplicaciones" en la página 232, se describe el código de ServerSessionPool y ServerSession de ejemplo que se proporciona con MQ JMS.
- En "Ejemplos de la utilización de ASF" en la página 236, se describen los ejemplos de ASF proporcionados y los ejemplos de la utilización de ASF desde la perspectiva de una aplicación cliente.

Nota: La especificación Java™ Message Service 1.0.2 para ASF también describe el soporte de JMS para transacciones distribuidas utilizando el protocolo XA de X/Open. Para obtener información más detallada sobre el soporte XA que proporciona MQ JMS, consulte "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377.

Clases y funciones de ASF

MQ JMS implementa la clase ConnectionConsumer y funciones avanzadas en la clase Session. Para obtener información más detallada, consulte los apartados siguientes:

- "MQPoolServices" en la página 133
- "MQPoolServicesEvent" en la página 134
- "MQPoolToken" en la página 136
- "MQPoolServicesEventListener" en la página 164
- "ConnectionConsumer" en la página 261
- "QueueConnection" en la página 307
- "Session" en la página 320
- "TopicConnection" en la página 338

ConnectionConsumer

La especificación JMS permite que un servidor de aplicaciones se integre estrechamente con una implementación de JMS utilizando la interfaz `ConnectionConsumer`. Esta característica proporciona proceso simultáneo de

mensajes. Generalmente, un servidor de aplicaciones crea una agrupación de hebras y la implementación de JMS hace que los mensajes estén disponibles para dichas hebras. Un servidor de aplicaciones que tenga constancia de JMS, puede utilizar esta característica para ofrecer funciones de envío de mensajes de alto nivel como, por ejemplo, beans de proceso de mensajes.

Las aplicaciones normales no utilizan ConnectionConsumer, pero los clientes JMS especializados pueden utilizarlo. Para este tipo de clientes, ConnectionConsumer proporciona un método de alto rendimiento para entregar mensajes simultáneamente a una agrupación de hebras. Cuando un mensaje llega a una cola o un tema, JMS selecciona una hebra de la agrupación y le entrega un lote de mensajes. Para hacerlo, JMS ejecuta un método `onMessage()` del MessageListener asociado.

Puede obtener el mismo resultado construyendo varios objetos Session y MessageConsumer, teniendo cada uno de ellos un MessageListener registrado. Sin embargo, ConnectionConsumer ofrece mejor rendimiento, menor utilización de recursos, mayor flexibilidad y, en especial, se requieren menos objetos Session.

Para ayudarle a desarrollar aplicaciones que utilicen ConnectionConsumers, MQ JMS proporciona una implementación de ejemplo con todas las funciones de una agrupación. La puede utilizar sin realizar ningún cambio o adaptarla de modo que se ajuste a las necesidades específicas de la aplicación.

Planificación de una aplicación

Principios generales para el envío de mensajes punto a punto

Cuando una aplicación crea un ConnectionConsumer desde un objeto QueueConnection, especifica un objeto JMS Queue y una serie de caracteres de selector. A continuación, ConnectionConsumer empieza a recibir mensajes (o, más concretamente, a proporcionar mensajes a las sesiones (Sessions) de la ServerSessionPool asociada). Los mensajes llegan a la cola y, si coinciden con el selector, se entregan a las sesiones de la ServerSessionPool asociada.

En términos de MQSeries, el objeto Queue hace referencia a un QLOCAL o QALIAS del gestor de colas local. Si tiene un QALIAS, dicho QALIAS debe hacer referencia a un QLOCAL. El MQSeries QLOCAL totalmente resuelto se denomina *QLOCAL subyacente*. Un ConnectionConsumer está *activo* si no está cerrado y el QueueConnection del elemento superior se ha iniciado.

Es posible que varios ConnectionConsumers, cada uno de ellos con selectores diferentes, se ejecuten para el mismo QLOCAL subyacente. Con el fin de mantener el rendimiento, los mensajes no deseados no se deben acumular en la cola. Los mensajes no deseados son aquellos para los que no hay ningún ConnectionConsumer activo que tenga un selector coincidente. Puede establecer QueueConnectionFactory de modo que los mensajes no deseados se eliminen de la cola (para obtener información más detallada, consulte el apartado “Eliminación de mensajes de la cola” en la página 229). Para establecer esta presentación, siga uno de los dos procedimientos que se indican a continuación:

- Utilice la herramienta de administración de JMS para establecer QueueConnectionFactory en MRET(NO).
- En el programa, utilice:

  ```
  MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
  ```

Si no cambia este valor, el valor por omisión es retener los mensajes no deseados en la cola.

Es posible que se puedan crear ConnectionConsumers cuyo destino sea el mismo QLOCAL subyacente a partir de varios objetos QueueConnection. Sin embargo, por motivos de rendimiento, se recomienda que varias JVM no creen múltiples ConnectionConsumer para el mismo QLOCAL subyacente.

Cuando establezca el gestor de colas MQSeries, tenga en cuenta los puntos siguientes:

- El QLOCAL subyacente debe estar habilitado para entrada compartida. Para hacerlo, utilice el mandato MQSC siguiente:

  ```
  ALTER QLOCAL(nombre.qlocal) SHARE GET(ENABLED)
  ```

- El gestor de colas debe tener una cola de mensajes no entregados habilitada. Si un ConnectionConsumer experimenta algún problema al colocar un mensaje en la cola de mensajes no entregados, la entrega de mensajes del QLOCAL subyacente se detiene. Para definir una cola de mensajes no entregados, utilice:

  ```
  ALTER QMGR DEADQ(nom.cola.mens.no.entregados)
  ```

- El usuario que ejecuta ConnectionConsumer debe tener autorización para realizar MQOPEN con MQOO_SAVE_ALL_CONTEXT y MQOO_PASS_ALL_CONTEXT. Para obtener información más detallada, consulte la documentación de MQSeries para su plataforma específica.
- Cuando los mensajes no deseados se dejan en la cola, reducen el rendimiento del sistema. Por este motivo, planifique los selectores de mensaje de modo que, entre ellos, haya ConnectionConsumers que eliminen todos los mensajes de la cola.

Para obtener información más detallada sobre los mandatos MQSC, consulte la publicación *MQSeries® Consulta de mandatos MQSC*.

Principios generales para el envío de mensajes de publicación/suscripción

Cuando una aplicación crea un ConnectionConsumer desde un objeto TopicConnection, especifica un objeto Topic y una serie de caracteres de selector. A continuación, ConnectionConsumer empieza a recibir los mensajes sobre dicho tema (Topic) que coinciden con el selector.

De forma alternativa, una aplicación puede crear un ConnectionConsumer duradero que esté asociado a un nombre específico. ConnectionConsumer recibe los mensajes que se han publicado sobre el tema desde la última vez que ha estado activo el ConnectionConsumer duradero. Recibe todos estos mensajes sobre el tema que coinciden con el selector.

En el caso de las suscripciones no duraderas, se utiliza una cola separada para las suscripciones de ConnectionConsumer. La opción configurable CCSUB de TopicConnectionFactory especifica la cola que se va a utilizar. Normalmente, CCSUB debe especificar una sola cola para que la utilicen todos los ConnectionConsumers que usan la misma TopicConnectionFactory. Sin embargo, se puede hacer que cada ConnectionConsumer genere una cola temporal especificando un prefijo de nombre de cola seguido del símbolo '*'.

Para las suscripciones duraderas, la propiedad CCDSUB del tema especifica la cola que se debe utilizar. También en este caso, puede ser una cola que ya exista o un prefijo de nombre de cola seguido del símbolo '*'. Si especifica una cola que ya existe, todos los ConnectionConsumers duraderos que se suscriben al tema utilizan esta cola. Si especifica un prefijo de nombre de cola seguido de un símbolo '*', la cola se genera la primera vez que se crea un ConnectionConsumer duradero con

un nombre determinado, y se vuelve a utilizar posteriormente, cuando se crea un ConnectionConsumer duradero con el mismo nombre.

Cuando establezca el gestor de colas MQSeries, tenga en cuenta los puntos siguientes:

- El gestor de colas debe tener una cola de mensajes no entregados habilitada. Si un ConnectionConsumer experimenta algún problema al colocar un mensaje en la cola de mensajes no entregados, la entrega de mensajes del QLOCAL subyacente se detiene. Para definir una cola de mensajes no entregados, utilice:

  ```
  ALTER QMGR DEADQ(su.nom.cola.mens.no.entregados)
  ```

- El usuario que ejecuta ConnectionConsumer debe tener autorización para realizar MQOPEN con MQOO_SAVE_ALL_CONTEXT y MQOO_PASS_ALL_CONTEXT. Para obtener información más detallada, consulte la documentación de MQSeries para su plataforma específica.
- Puede optimizar el rendimiento de un ConnectionConsumer individual creándole una cola dedicada separada, aunque puede producir una mayor utilización de los recursos.

Manejo de mensajes dañados

A veces llega un mensaje formateado de modo incorrecto a una cola. Este tipo de mensaje puede hacer que la aplicación receptora no se ejecute correctamente y restituya la recepción del mensaje. En este caso, puede que se reciba y se devuelva el mensaje a la cola repetidas veces. Estos mensajes se denominan *mensajes dañados*. ConnectionConsumer debe poder detectar los mensajes dañados y volverlos a direccionar a otro destino.

Cuando una aplicación utiliza ConnectionConsumers, las circunstancias en las se restituye un mensaje dependen de la sesión (Session) que proporciona el servidor de aplicaciones:

- Cuando la sesión no se ha ejecutado con AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE, el mensaje sólo se restituye después de un error del sistema o si la aplicación termina de modo inesperado.
- Cuando la sesión no se ha ejecutado con CLIENT_ACKNOWLEDGE, el servidor de aplicaciones puede restituir los mensajes no reconocidos invocando a `Session.recover()`.

  Por lo general, la implementación de cliente de MessageListener o el servidor de aplicaciones invocan a `Message.acknowledge()`. `Message.acknowledge()` reconoce todos los mensajes que se han entregado hasta el momento en la sesión.
- Cuando la sesión se ha ejecutado, normalmente el servidor de aplicaciones la confirma. Si detecta un error, puede elegir restituir uno o más mensajes.
- Si el servidor de aplicaciones suministra una XASession, los mensajes se confirman o restituyen en una transacción distribuida. El servidor de aplicaciones se responsabiliza de finalizar la transacción.

El gestor de colas MQSeries mantiene un registro del número de veces que se ha restituido cada mensaje. Cuando el número alcanza un umbral configurable, ConnectionConsumer reposiciona el mensaje en una cola de restitución especificada. Si, por cualquier motivo, la reposición en cola no se ejecuta correctamente, el mensaje se elimina de la cola y se reposiciona en la cola de mensajes no entregados o se descarta. Para obtener información más detallada, consulte el apartado “Eliminación de mensajes de la cola” en la página 229.

En la mayor parte de las plataformas, la cola de reposición y de umbral son propiedades de MQSeries QLOCAL. Para el envío de mensajes punto a punto,

debe ser la cola QLOCAL subyacente. Para el envío de mensajes de publicación/suscripción, es la cola CCSUB definida en TopicConnectionFactory o la cola CCDSUB definida en el tema (Topic). Para establecer las propiedades de la cola de reposición y de umbral, emita el mandato MQSC siguiente:

```
ALTER QLOCAL(nombre.cola) BOTHRESH(umbral) BOQUEUE(nombre.cola.reposición.cola)
```

Para el envío de mensajes de publicación/suscripción, si el sistema crea una cola dinámica para cada suscripción, los valores se obtienen de la cola modelo MQ JMS. Para modificar los valores, puede utilizar lo siguiente:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(umbral) BOQUEUE(nombre.cola.reposición.cola)
```

Si el umbral es cero, el manejo de mensajes dañados se inhabilita y estos mensajes permanecen en la cola de entrada. De lo contrario, cuando el contador de restituciones alcanza el umbral, el mensaje se envía a la cola de reposición especificada. Si el contador de restituciones alcanza el umbral, pero el mensaje no puede pasar a la cola de reposición, se envía a la cola de mensajes no entregados o se descarta. Esta situación se produce si no se ha definido la cola de reposición o si ConnectionConsumer no puede enviar el mensaje a la cola de reposición. En algunas plataformas, no se pueden especificar las propiedades de la cola de reposición y de umbral, por lo que los mensajes se envían a la cola de mensajes no entregados o se descartan cuando el contador de restituciones alcanza el número de 20. Para obtener información más detallada, consulte “Eliminación de mensajes de la cola”.

Eliminación de mensajes de la cola

Cuando una aplicación utiliza ConnectionConsumers, es posible que JMS necesite eliminar mensajes de la cola en algunas situaciones:

Mensaje formateado de modo incorrecto
Puede llegar un mensaje que JMS no pueda analizar.

Mensaje dañado
Un mensaje puede alcanzar el umbral de restitución, pero ConnectionConsumer quizá no pueda reposicionarlo en la cola de restitución.

ConnectionConsumer no interesado
Para el envío de mensajes punto a punto, cuando se establece QueueConnectionFactory de modo que no retenga los mensajes no deseados, puede llegar un mensaje que ningún ConnectionConsumer desee.

En estas situaciones, ConnectionConsumer intenta eliminar el mensaje de la cola. Las opciones de disposición del campo de informe MQMD del mensaje establecen la presentación exacta. Estas opciones son las siguientes:

MQRO_DEAD_LETTER_Q
El mensaje se reposiciona en la cola de mensajes no entregados del gestor de colas. Éste es el valor por omisión.

MQRO_DISCARD_MSG
El mensaje se descarta.

ConnectionConsumer también genera un mensaje de informe, que depende asimismo del campo de informe MQMD del mensaje. Este mensaje se envía a la cola de respuestas (ReplyToQ) del gestor de colas (ReplyToQmgr) del mensaje. Si se produce algún error durante el envío del mensaje de informe, el mensaje se envía a la cola de mensajes no entregados. Las opciones de informe de excepción

del campo de informe MQMD del mensaje establecen los detalles del mensaje de informe. Estas opciones son las siguientes:

MQRO_EXCEPTION
Se genera un mensaje de informe que contiene MQMD del mensaje original. No contiene ningún dato de cuerpo del mensaje.

MQRO_EXCEPTION_WITH_DATA
Se genera un mensaje de informe que contiene MQMD, todas las cabeceras MQ y 100 bytes de datos del cuerpo.

MQRO_EXCEPTION_WITH_FULL_DATA
Se genera un mensaje de informe que contiene todos los datos del mensaje original.

default
No se genera ningún mensaje de informe.

Cuando se generan mensajes de informe, se aceptan las opciones siguientes:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Si un ConnectionConsumer no puede cumplir las opciones de disposición, o las opciones de informe de excepción de MQMD del mensaje, su acción depende de la permanencia del mensaje. Si el mensaje no es permanente, se descarta y no se genera ningún mensaje de informe. Si el mensaje es permanente, se detiene la entrega de todos los mensajes de QLOCAL.

Por consiguiente, es importante definir una cola de mensajes no entregados y comprobarla con regularidad para verificar que no se ha producido ningún problema. En especial, debe asegurarse de que la cola de mensajes no entregados no alcanza la capacidad máxima y que el tamaño máximo de mensajes es suficientemente grande para todos los mensajes.

Cuando se reposiciona un mensaje en la cola de mensajes no entregados, va precedido de una cabecera de mensaje no entregado MQSeries (MQDLH). Para obtener información más detallada sobre el formato de MQDLH, consulte la publicación *MQSeries® Application Programming Reference*. Los campos siguientes le permiten identificar los mensajes que ha colocado un ConnectionConsumer en la cola de mensajes no entregados o los mensajes de informe que ha generado:

- PutApplType is `MQAT_JAVA (0x1C)`
- PutApplName is "`MQ JMS ConnectionConsumer`"

Estos campos se encuentran en el descriptor de mensajes (MQDLH) de la cola de mensajes no entregados y en MQMD de mensajes de informe. El campo de información de retorno de MQMD y el campo Razón (Reason) de MQDLH, contienen un código que describe el error. Para obtener información más detallada sobre estos códigos, consulte el apartado "Manejo de errores" en la página 231. En la publicación *MQSeries® Application Programming Reference* puede obtener información sobre otros campos.

Manejo de errores

Recuperación de condiciones de error

Si un ConnectionConsumer experimenta un problema grave, se detiene la entrega de mensajes a todos los ConnectionConsumers interesados en el mismo QLOCAL. Generalmente, ocurre si ConnectionConsumer no puede reposicionar un mensaje en la cola de mensajes no entregados o si se produce algún error al leer mensajes de QLOCAL.

Cuando se produce, se notifica a la aplicación y al servidor de aplicaciones tal como se indica a continuación:

- Se notifica a todos los ExceptionListener registrados a la conexión (Connection) afectada.

Puede utilizarlos para identificar la causa del problema. En algunos casos, el administrador del sistema debe intervenir para resolver el problema.

Para que la aplicación se recupere de estas condiciones de error, existen dos procedimientos:

- Invocar a `close()` en todos los ConnectionConsumers afectados. La aplicación puede crear nuevos ConnectionConsumers sólo después de que se hayan cerrado todos los ConnectionConsumers afectados y se hayan resuelto todos los problemas del sistema.
- Invocar a `stop()` en todas las conexiones (Connections) afectadas. Después de detener todas las conexiones y de resolver todos los problemas del sistema, la aplicación debe poder iniciar (`start()`) todas las conexiones correctamente.

Códigos de información de retorno y de razón

Para determinar la causa de un error, puede utilizar lo siguiente:

- El código de información de retorno de todos los mensajes de informe
- El código de razón de MQDLH de todos los mensajes de la cola de mensajes no entregados

ConnectionConsumer genera los códigos de razón siguientes:

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Causa El mensaje alcanza al umbral de restitución definido en QLOCAL, pero no se ha definido ninguna cola de restitución.

En las plataformas en las que no puede se definir la cola de restitución, el mensaje alcanza el umbral de restitución de 20 que define JMS.

Acción Para evitar esta situación, asegúrese de que los ConnectionConsumers que utilizan la cola proporcionen un conjunto de selectores que trate todos los mensajes, o establezca QueueConnectionFactory para retener los mensajes.

De forma alternativa, determine el origen del mensaje.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Causa En el envío de mensajes punto a punto, hay un mensaje que no coincide con ningún selector para la supervisión de

la cola de ConnectionConsumers. Para mantener el rendimiento, el mensaje se reposiciona en la cola de mensajes no entregados.

Acción Para evitar esta situación, asegúrese de que los ConnectionConsumers que utilizan la cola proporcionen un conjunto de selectores que traten todos los mensajes, o establezca QueueConnectionFactory para retener los mensajes.

De forma alternativa, determine el origen del mensaje.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Causa JMS no puede interpretar el mensaje de la cola.

Acción Determine el origen del mensaje. Por lo general, JMS entrega los mensajes con formato inesperado como BytesMessage o TextMessage pero, en ocasiones, puede dar error si el mensaje está muy mal formateado.

Los demás códigos que pueden aparecer en estos campos pueden deberse a que se haya intentado reposicionar el mensaje en la cola de restitución y no se haya podido. En este caso, el código describe la razón por la que la reposición en cola no se ha ejecutado correctamente. Para diagnosticar la causa de estos errores, consulte la publicación *MQSeries® Application Programming Reference*.

Si el mensaje de informe no se puede colocar en la cola de respuestas (ReplyToQ), se sitúa en la cola de mensajes no entregados. En este caso, el campo de información de retorno de MQMD se rellena tal como se ha descrito más arriba. El campo de razón de MQDLH explica el motivo por el que no se ha podido colocar el mensaje de informe en la cola de respuestas (ReplyToQ).

Código de ejemplo del servidor de aplicaciones

La Figura 6 en la página 233 resume los principios de las funciones de ServerSessionPool y ServerSession.

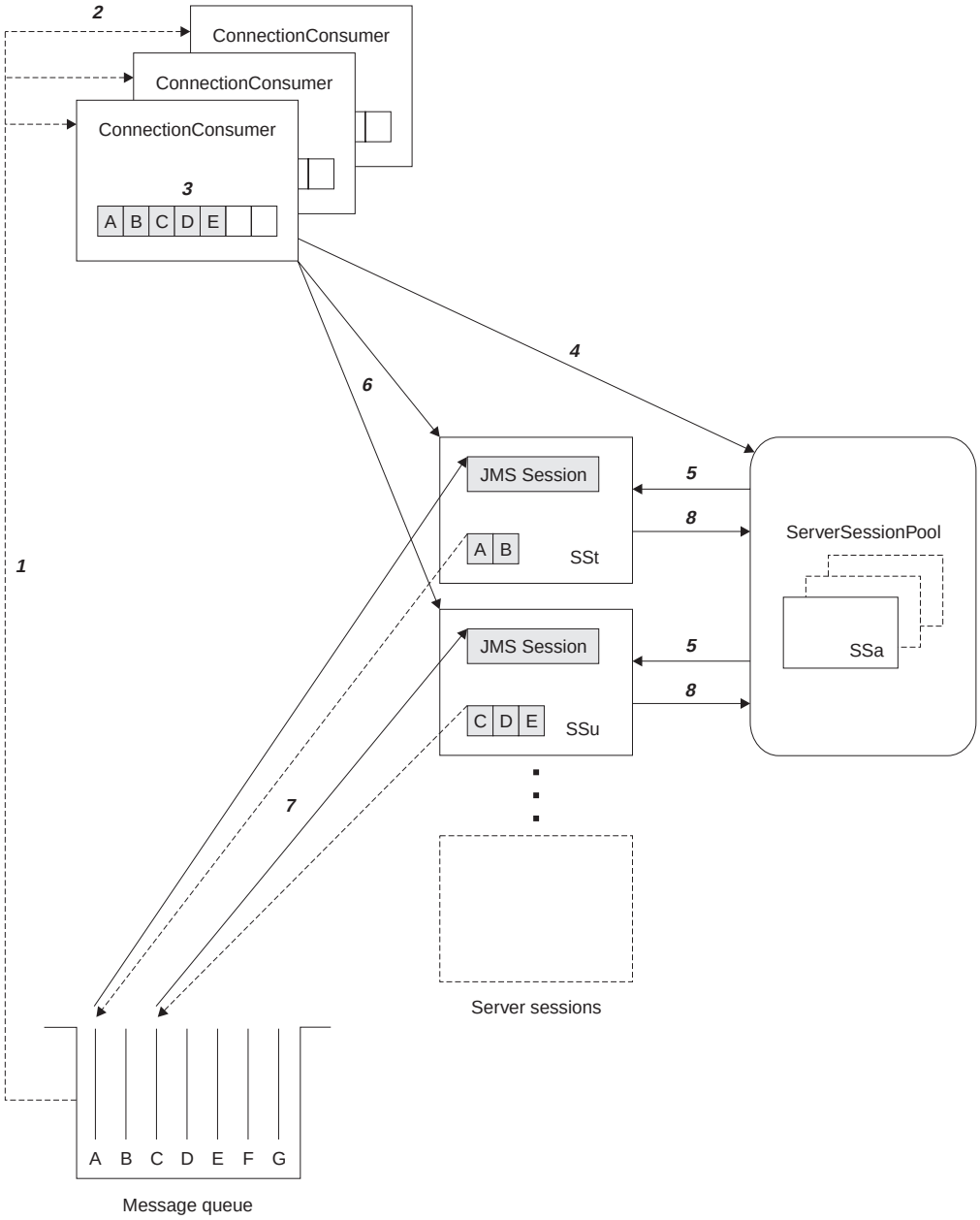


Figura 6. Funciones de ServerSessionPool y ServerSession

1. Los ConnectionConsumer obtienen las referencias de mensajes de la cola.
2. Cada ConnectionConsumer selecciona referencias de mensajes específicos.
3. El almacenamiento intermedio de ConnectionConsumer mantiene las referencias de los mensajes seleccionados.
4. ConnectionConsumer solicita una o más ServerSessions de ServerSessionPool.
5. Las ServerSessions se asignan de ServerSessionPool.
6. ConnectionConsumer asigna referencias de mensaje a ServerSessions e inicia las hebras de ServerSession en ejecución.
7. Cada ServerSession recupera sus mensajes de referencia de la cola, y los pasa al método `onMessage` del MessageListener que está asociado a la sesión JMS.
8. Cuando finaliza el proceso, se devuelve ServerSession a la agrupación.

Código de ejemplo del servidor de aplicaciones

Normalmente, el servidor de aplicaciones proporciona las funciones de ServerSessionPool y ServerSession. Sin embargo, se suministra a MQ JMS una implementación simple de estas interfaces, con la fuente del programa. Estos ejemplos están situados en el directorio siguiente, donde <dir_instalación> es el directorio de instalación de MQ JMS:

```
<dir_instalación>/samples/jms/asf
```

Estos ejemplos le permiten utilizar ASF de MQ JMS en un entorno autónomo (es decir, no necesita un servidor de aplicaciones específico). Además, proporcionan ejemplos sobre el modo de implementar las interfaces y beneficiarse de ASF de MQ JMS. La finalidad de estos ejemplos es ayudar tanto a los usuarios de MQ JMS como a los proveedores de otros servidores de aplicaciones.

MyServerSession.java

Esta clase implementa la interfaz `javax.jms.ServerSession`. Su función básica consiste en asociar una hebra con una sesión JMS. Un ServerSessionPool agrupa las instancias de esta clase (consulte el apartado "MyServerSessionPool.java"). Puesto que se trata de una ServerSession, debe implementar los dos métodos siguientes:

- `getSession()`, que devuelve la sesión JMS asociada a esta ServerSession
- `start()`, que inicia la hebra de ServerSession, cuyo resultado es la invocación del método `run()` de la sesión JMS

MyServerSession también implementa la interfaz `Runnable`. Por consiguiente, la creación de una hebra de ServerSession se puede basar en esta clase y no necesita una clase separada.

La clase utiliza un mecanismo `wait()-notify()` basado en los valores de dos distintivos booleanos, ready y quit. Este mecanismo significa que ServerSession crea e inicia su hebra asociada durante su construcción. Sin embargo, no ejecuta automáticamente el cuerpo del método `run()`. El cuerpo del método `run()` sólo se ejecuta cuando se establece el distintivo de preparado (ready) en verdadero (true) utilizando el método `start()`. ASF invoca al método `start()` cuando se necesita entregar mensajes a la sesión JMS asociada.

Para la entrega, se invoca al método `run()` de la sesión JMS. ASF de MQ JMS ya ha cargado previamente el método `run()` con mensajes.

Después de finalizar la entrega, el distintivo de preparado se restablece en falso (false) y se notifica a la ServerSessionPool propietaria que la entrega ha terminado. A continuación, ServerSession permanece en estado de espera hasta que se vuelve a invocar al método `start()`, o se invoca el método `close()`, que finaliza esta hebra de ServerSession.

MyServerSessionPool.java

Esta clase implementa la interfaz `javax.jms.ServerSessionPool` y existe para crear y controlar el acceso a una agrupación de ServerSessions.

En esta implementación simple, la agrupación consta de una matriz estática de objetos ServerSession que se crean durante la construcción de la agrupación. Se pasan los cuatro parámetros siguientes al constructor:

- `javax.jms.Connection` *`conexión`*

  La conexión que se va a utilizar para crear sesiones JMS.
- `int` *`capacidad`*

El tamaño de la matriz de objetos MyServerSession.

- `int ModRec`

  Modalidad de reconocimiento necesaria de las sesiones JMS.

- `MessageListenerFactory mlf`

  MesssageListenerFactory que crea el escucha de mensajes que se proporciona para las sesiones JMS. Consulte el apartado "MessageListenerFactory.java".

El constructor de la agrupación utiliza estos parámetros para crear una matriz de objetos MyServerSession. La conexión proporcionada se utiliza para crear sesiones JMS de la modalidad de reconocimiento especificada y el dominio correcto (QueueSessions para punto a punto y TopicSessions para publicación/suscripción). Las sesiones disponen de un escucha de mensajes. Por último, se crean los objetos ServerSession basados en las sesiones JMS.

Esta implementación de ejemplo es un modelo estático. Es decir, todas las ServerSessions de la agrupación se crean cuando se crea la agrupación, tras lo cual la agrupación no puede ni agrandarse ni reducirse. La finalidad de este enfoque es la simplicidad. Si fuera necesario, una ServerSessionPool puede utilizar un algoritmo complejo para crear ServerSessions de modo dinámico.

MyServerSessionPool mantiene un registro de las ServerSessions que se están utilizando actualmente, al mantener una matriz de valores booleanos denominada inUse. Todos estos valores booleanos se inicializan como falso (false). Cuando se invoca el método `getServerSession` y éste solicita una ServerSession de la agrupación, se busca el primer valor falso (false) en la matriz inUse. Cuando se encuentra, el valor booleano se establece en verdadero (true) y se devuelve la ServerSession correspondiente. Si no hay ningún valor falso en la matriz inUse, el método `getServerSession` debe esperar (`wait()`) hasta que se produzca la notificación.

La notificación se produce en cualquiera de las circunstancias siguientes:

- Se invoca el método `close()` de la agrupación, indicando que se debe concluir la agrupación.
- Una ServerSession que está actualmente en uso finaliza su carga de trabajo e invoca el método `serverSessionFinished`. El método `serverSessionFinished` devuelve la ServerSession a la agrupación y establece el distintivo inUse correspondiente en falso. La ServerSession queda entonces disponible para ser reutilizada.

MessageListenerFactory.java

En este ejemplo, se ha asociado un objeto de fábrica de escucha de mensajes a cada instancia de ServerSessionPool. La clase MessageListenerFactory representa una interfaz muy sencilla que se utiliza para obtener una instancia de una clase que implementa la interfaz `javax.jms.MessageListener`. La clase contiene un único método:

```
javax.jms.MessageListener createMessageListener();
```

Cuando se construye la ServerSessionPool, se proporciona una implementación de esta interfaz. Este objeto se utiliza para crear escuchas de mensajes para las sesiones JMS individuales que hacen una copia de seguridad de las ServerSessions de la agrupación. Esta arquitectura significa que cada implementación separada de la interfaz `MessageListenerFactory` debe tener su propia ServerSessionPool.

Código de ejemplo del servidor de aplicaciones

MQ JMS incluye una implementación de MessageListenerFactory de ejemplo, que se trata en el apartado “CountingMessageListenerFactory.java” en la página 237.

Ejemplos de la utilización de ASF

El directorio `<dir_instalación>/samples/jms/asf` (donde <dir_instalación> es el directorio de instalación de MQ JMS) contiene un conjunto de clases, con su fuente. Estas clases utilizan los recursos de servidor de aplicaciones MQ JMS que se describen en el apartado “Clases y funciones de ASF” en la página 225, en el entorno de servidor de aplicaciones autónomo que se describe en el apartado “Código de ejemplo del servidor de aplicaciones” en la página 232.

Se proporcionan cuatro ejemplos de la utilización de ASF desde la perspectiva de una aplicación cliente:

- Un ejemplo punto a punto simple que utiliza:
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- Un ejemplo punto a punto más complejo que utiliza:
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- Un ejemplo de publicación/suscripción simple que utiliza:
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- Un ejemplo de publicación/suscripción más complejo que utiliza:
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java

En los apartados siguientes se describe cada clase.

Load1.java

Esta clase es una aplicación JMS genérica simple que carga un número de mensajes determinado en una cola específica y, a continuación, termina. Puede recuperar los objetos administrados necesarios de un espacio de nombres JNDI o crearlos explícitamente utilizando las clases MQ JMS que implementan dichas interfaces. Los objetos administrados que se necesitan son QueueConnectionFactory y Queue. Puede utilizar las opciones de línea de mandatos para establecer los mensajes con los que cargar la cola y el tiempo de latencia entre transferencias de mensajes individuales.

Esta aplicación tiene dos versiones de sintaxis de línea de mandatos.

Para utilizarla con JNDI, la sintaxis es la siguiente:

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

Para utilizarla sin JNDI, la sintaxis es la siguiente:

```
java Load1 -nojndi [-qm qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

En la Tabla 22 se describen los parámetros y se indican sus valores por omisión.

Tabla 22. Parámetros de Load1 y valores por omisión

Parámetro	Significado	Valor por omisión
jndiICF	Clase de fábrica de contextos iniciales que se utiliza para JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	URL del suministrador que se utiliza para JNDI	ldap://localhost/o=ibm,c=us
qcfLookup	Clave de búsqueda JNDI que se utiliza para QueueConnectionFactory	cn=qcf
qLookup	Clave de búsqueda JNDI que se utiliza para Queue	cn=q
qMgrName	Nombre del gestor de colas al que conectar	"" (utilizar el gestor de colas por omisión)
qName	Nombre de la cola que se va a cargar	SYSTEM.DEFAULT.LOCAL.QUEUE
sleepTime	Tiempo (en milisegundos) de espera entre transferencias de mensajes	0 (sin tiempo de espera)
numMsgs	Número de mensajes a transferir	1000

Si se produce algún error, se visualiza un mensaje de error y la aplicación termina.

Puede utilizar esta aplicación para simular la carga de mensajes en una cola MQSeries. A su vez, esta carga de mensajes puede activar las aplicaciones habilitadas de ASF que se describen en los apartados siguientes. Los mensajes transferidos a la cola son objetos TextMessage JMS simples, que no contienen propiedades de mensajes definidas por el usuario, por lo que pueden resultar útiles para utilizar escuchas de mensajes diferentes. Se proporciona el código fuente, que le permite modificar esta aplicación de carga en caso de que sea necesario.

CountingMessageListenerFactory.java

Este archivo contiene definiciones para dos clases:

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener es una implementación muy simple de la interfaz `javax.jms.MessageListener`. Mantiene un registro de las veces que se ha invocado el método `onMessage`, pero no realiza ninguna acción con los mensajes que se le pasan.

CountingMessageListenerFactory es la clase de fábrica para CountingMessageListener. Es una implementación de la interfaz `MessageListenerFactory` que se describe en el apartado "MessageListenerFactory.java" en la página 235. Esta fábrica mantiene un registro

de todos los escuchas de mensajes que produce. También incluye un método, `printStats()`, que muestra estadísticas de utilización de cada uno de los escuchas.

ASFClient1.java

Esta aplicación actúa como un cliente de ASF MQ JMS. Establece un único ConnectionConsumer para consumir los mensajes de una sola cola MQSeries. Muestra estadísticas de productividad de cada escucha de mensajes utilizado y termina después de un minuto.

La aplicación puede recuperar los objetos administrados necesarios de un espacio de nombres JNDI o crearlos explícitamente utilizando las clases MQ JMS que implementan dichas interfaces. Los objetos administrados que se necesitan son QueueConnectionFactory y Queue.

Esta aplicación tiene dos versiones de sintaxis de línea de mandatos:

Para utilizarla con JNDI, la sintaxis es la siguiente:

```
java ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

Para utilizarla sin JNDI, la sintaxis es la siguiente:

```
java ASFClient1 -nojndi [-qm qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

En la Tabla 23 se describen los parámetros y se indican sus valores por omisión.

Tabla 23. Parámetros de ASFClient1 y valores por omisión

Parámetro	Significado	Valor por omisión
jndiICF	Clase de fábrica de contextos iniciales que se utiliza para JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	URL del suministrador que se utiliza para JNDI	ldap://localhost/o=ibm,c=us
qcfLookup	Clave de búsqueda JNDI que se utiliza para QueueConnectionFactory	cn=qcf
qLookup	Clave de búsqueda JNDI que se utiliza para Queue	cn=q
qMgrName	Nombre del gestor de colas al que conectar	"" (utilizar el gestor de colas por omisión)
qName	Nombre de la cola de la que se va a consumir	SYSTEM.DEFAULT.LOCAL.QUEUE
poolSize	Número de ServerSessions creadas en ServerSessionPool que se utilizan	5
batchSize	Número máximo de mensajes que se pueden asignar a una ServerSession al mismo tiempo	10

La aplicación obtiene una QueueConnection de QueueConnectionFactory.

ServerSessionPool, con el formato MyServerSessionPool, se construye utilizando:

- la QueueConnection que se ha creado previamente

- el poolSize necesario
- una modalidad de reconocimiento, AUTO_ACKNOWLEDGE
- una instancia de CountingMessageListenerFactory, tal como se describe en el apartado “CountingMessageListenerFactory.java” en la página 237

A continuación, se invoca al método `createConnectionConsumer` de la conexión pasando:

- la cola (Queue) que se ha obtenido anteriormente
- un selector de mensajes nulo (que indica que se deben aceptar todos los mensajes)
- la ServerSessionPool que se acaba de crear
- el batchSize necesario

El consumo de mensajes se inicia entonces a través de la invocación del método `start()` de la conexión.

Cada 10 segundos, la aplicación cliente muestra estadísticas de productividad de cada escucha de mensajes utilizado. Después de un minuto, la conexión se cierra, la agrupación de sesiones de servidor se detiene y la aplicación termina.

Load2.java

Esta clase es una aplicación JMS que carga un número de mensajes determinado en una cola específica y, a continuación, termina de un modo similar a Load1.java. La sintaxis de línea de mandatos también es similar a la de Load1.java (sustituyendo Load1 por Load2 en la sintaxis). Para obtener información más detallada, consulte el apartado “Load1.java” en la página 236.

La diferencia consiste en que cada mensaje contiene una propiedad de usuario denominada value, que toma un valor de entero seleccionado aleatoriamente entre 0 y 100. Esta propiedad significa que pueden aplicarse selectores de mensaje a los mensajes. Por este motivo, se pueden compartir los mensajes entre los dos consumidores que se han creado en la aplicación cliente descrita en el apartado “ASFClient2.java” en la página 240.

LoggingMessageListenerFactory.java

Este archivo contiene definiciones para dos clases:

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener es una implementación de la interfaz `javax.jms.MessageListener`. Toma los mensajes que se pasan y escribe una entrada en el archivo de anotaciones. El archivo de anotaciones por omisión es `./ASFClient2.log`. Puede consultar este archivo y comprobar los mensajes que se han enviado al consumidor de conexiones que utiliza este escucha de mensajes.

LoggingMessageListenerFactory es la clase de fábrica para LoggingMessageListener. Es una implementación de la interfaz `MessageListenerFactory` que se describe el apartado “MessageListenerFactory.java” en la página 235.

ASFClient2.java

ASFClient2.java es una aplicación cliente un poco más complicada que ASFClient1.java. Crea dos ConnectionConsumers que alimentan la misma cola, pero que aplican selectores de mensaje diferentes. La aplicación utiliza una CountingMessageListenerFactory para un consumidor y un LoggingMessageListenerFactory para el otro. La utilización de dos fábricas de escucha de mensajes diferentes significa que cada consumidor debe tener su propia agrupación de sesiones de servidor.

La aplicación muestra estadísticas relacionadas con un ConnectionConsumer en la pantalla y escribe las estadísticas relacionadas con el otro en un archivo de anotaciones.

La sintaxis de línea de mandatos es similar a la de "ASFClient1.java" en la página 238 (sustituyendo ASFClient1 por ASFClient2 en la sintaxis). Cada una de las dos agrupaciones de sesión de servidor contiene el número de ServerSessions que ha establecido el parámetro poolSize.

Debe haber una distribución desigual de mensajes. Los mensajes que carga Load2 en la cola origen contienen una propiedad de usuario en la que el valor debe situarse entre 0 y 100, distribuido de forma aleatoria y equitativa. El selector de mensajes `value>75` se aplica a highConnectionConsumer, y el selector de mensajes `value≤75` se aplica a normalConnectionConsumer. Los mensajes de highConnectionConsumer (aproximadamente el 25% de la carga total) se envían a un LoggingMessageListener. Los mensajes de normalConnectionConsumer (aproximadamente el 75% de la carga total) se envían a un CountingMessageListener.

Al ejecutar la aplicación cliente, las estadísticas relacionadas con normalConnectionConsumer y sus CountingMessageListenerFactories asociadas, se imprimen en la pantalla cada 10 segundos. Las estadísticas que hacen referencia a highConnectionConsumer y sus LoggingMessageListenerFactories asociadas, se escriben en el archivo de anotaciones.

Puede examinar la pantalla y el archivo de anotaciones para ver el destino real de los mensajes. Sume los totales de cada uno de los CountingMessageListeners. Siempre que la aplicación cliente no termine antes de que se hayan consumido todos los mensajes, debe dar cuenta del 75% de la carga, aproximadamente. El número de entradas del archivo de anotaciones debe dar cuenta del resto de la carga. (Si la aplicación cliente termina antes de que se hayan consumido todos los mensajes, puede aumentar el tiempo de espera de la aplicación).

TopicLoad.java

Esta clase es una aplicación JMS que es una versión de publicación/suscripción del cargador de cola Load2 que se describe en el apartado "Load2.java" en la página 239. Publica el número de mensajes necesarios del tema especificado y, a continuación, termina. Cada mensaje contiene una propiedad de usuario denominada value, que toma un valor de entero seleccionado aleatoriamente entre 0 y 100.

Para utilizar esta aplicación, asegúrese de que el intermediario está ejecutándose y que se ha llevado a cabo la configuración necesaria. Para obtener información más detallada, consulte el apartado "Configuración adicional para la modalidad Publish/Subscribe" en la página 24.

Esta aplicación tiene dos versiones de sintaxis de línea de mandatos.

Para utilizarla con JNDI, la sintaxis es la siguiente:

```
java TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

Para utilizarla sin JNDI, la sintaxis es la siguiente:

```
java TopicLoad -nojndi [-qm qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

En la Tabla 24 se describen los parámetros y se indican sus valores por omisión.

Tabla 24. Parámetros de TopicLoad y valores por omisión

Parámetro	Significado	Valor por omisión
jndiICF	Clase de fábrica de contextos iniciales que se utiliza para JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	URL del suministrador que se utiliza para JNDI	ldap://localhost/o=ibm,c=us
tcfLookup	Clave de búsqueda JNDI que se utiliza para TopicConnectionFactory	cn=tcf
tLookup	Clave de búsqueda JNDI que se utiliza para Topic	cn=t
qMgrName	Nombre del gestor de colas al que conectar y gestor de colas del intermediario en el que publicar mensajes	"" (utilizar el gestor de colas por omisión)
tName	Nombre del tema que se va a publicar	MQJMS/ASF/TopicLoad
sleepTime	Tiempo (en milisegundos) de espera entre transferencias de mensajes	0 (sin tiempo de espera)
numMsgs	Número de mensajes a transferir	200

Si se produce algún error, se visualiza un mensaje de error y la aplicación termina.

ASFClient3.java

ASFClient3.java es una aplicación cliente que es una versión de publicación/suscripción de "ASFClient1.java" en la página 238. Establece un único ConnectionConsumer para consumir los mensajes que se publican sobre un solo tema (Topic). Muestra estadísticas de productividad para cada escucha de mensajes utilizado y termina después de un minuto.

Esta aplicación tiene dos versiones de sintaxis de línea de mandatos.

Para utilizarla con JNDI, la sintaxis es la siguiente:

```
java ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize poolSize] [-batchsize batchSize]
```

Para utilizarla sin JNDI, la sintaxis es la siguiente:

```
java ASFClient3 -nojndi [-qm qMgrName] [-t tName]
                        [-poolsize poolSize] [-batchsize batchSize]
```

Ejemplos de la utilización de ASF

En la Tabla 25 se describen los parámetros y se indican sus valores por omisión.

Tabla 25. Parámetros de ASFClient3 y valores por omisión

Parámetro	Significado	Valor por omisión
jndiICF	Clase de fábrica de contextos iniciales que se utiliza para JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	URL del suministrador que se utiliza para JNDI	ldap://localhost/o=ibm,c=us
tcfLookup	Clave de búsqueda JNDI que se utiliza para TopicConnectionFactory	cn=tcf
tLookup	Clave de búsqueda JNDI que se utiliza para Topic	cn=t
qMgrName	Nombre del gestor de colas al que conectar y gestor de colas del intermediario en el que publicar mensajes	"" (utilizar el gestor de colas por omisión)
tName	Nombre del tema del que se va a consumir	MQJMS/ASF/TopicLoad
poolSize	Número de ServerSessions creadas en ServerSessionPool que se utilizan	5
batchSize	Número máximo de mensajes que se pueden asignar a una ServerSession al mismo tiempo	10

Como ASFClient1, cada 10 segundo la aplicación cliente muestra estadísticas de productividad de cada escucha de mensajes utilizado. Después de un minuto, la conexión se cierra, la agrupación de sesiones de servidor se detiene y la aplicación termina.

ASFClient4.java

ASFClient4.java es una aplicación cliente de publicación/suscripción más compleja. Crea tres ConnectionConsumers y todos ellos alimentan el mismo tema, pero cada uno aplica selectores de mensaje diferentes.

Los dos primeros consumidores utilizan selectores de mensaje 'high' y 'normal', igual que la aplicación "ASFClient2.java" en la página 240. El tercer consumidor no utiliza ningún selector de mensajes. La aplicación utiliza dos CountingMessageListenerFactories para los dos consumidores basados en el selector y una LoggingMessageListenerFactory para el tercer consumidor. Puesto que la aplicación utiliza fábricas de escucha de mensajes diferentes, cada consumidor debe tener su propia agrupación de sesiones de servidor.

La aplicación muestra estadísticas en la pantalla relacionadas con los dos consumidores basados en selector, y escribe las estadísticas referentes al tercer ConnectionConsumer en un archivo de anotaciones.

La sintaxis de línea de mandatos es similar a la de "ASFClient3.java" en la página 241 (sustituyendo ASFClient3 por ASFClient4). Cada una de las tres agrupaciones de sesión de servidor contiene el número de ServerSessions que ha establecido el parámetro poolSize.

Al ejecutar la aplicación cliente, las estadísticas que hacen referencia a normalConnectionConsumer y highConnectionConsumer y sus CountingMessageListenerFactories asociadas, se imprimen en la pantalla cada 10 segundos. Las estadísticas relacionadas con el tercer ConnectionConsumer y sus LoggingMessageListenerFactories asociadas, se escriben en el archivo de anotaciones.

Puede examinar la pantalla y el archivo de anotaciones para ver el destino real de los mensajes. Sume los totales de cada CountingMessageListeners y consulte el número de entradas del archivo de anotaciones.

La distribución de mensajes debe ser diferente de la distribución que obtiene una versión punto a punto de la misma aplicación (ASFClient2.java), puesto que, en el dominio de publicación/suscripción, cada consumidor de un tema obtiene su propia copia de cada mensaje publicado sobre dicho tema. En esta aplicación, para la carga de un tema determinado, los consumidores 'high' y 'normal' reciben aproximadamente el 25 y el 75 por ciento de la carga respectivamente. El tercer consumidor sigue recibiendo el cien por cien de la carga. Por consiguiente, el número total de los mensajes recibidos es superior al cien por cien de la carga publicada originalmente sobre el tema.

Capítulo 14. Interfaces y clases JMS

El Servicio de mensajes de MQSeries Classes for Java consta de varias interfaces y clases Java que se basan en el paquete de clases e interfaces javax.jms de Sun. Los clientes se deben crear utilizando las clases y las interfaces de Sun que se enumeran más abajo, y que se describen con detalle en los apartados siguientes. Los nombres de los objetos MQSeries® que implementan las clases y las interfaces de Sun tienen el prefijo 'MQ' (a menos que se indique de otro modo en la descripción del objeto). Las descripciones incluyen detalles sobre las desviaciones de los objetos MQSeries de las definiciones JMS estándar. Dichas desviaciones se indican con el símbolo '*'.

Interfaces y clases del Servicio de mensajes Java™ de Sun

Las tablas siguientes contienen la lista de los objetos JMS contenidos en el paquete **javax.jms**. Las aplicaciones implementan las interfaces marcadas con el símbolo '*'. Los servidores de aplicaciones implementan las interfaces marcadas con el símbolo '**'.

Tabla 26. Resumen de las interfaces

Interfaz	Descripción
BytesMessage	BytesMessage se utiliza para enviar un mensaje que contiene una corriente de bytes no interpretados.
Connection	JMS Connection es una conexión activa del cliente a su suministrador de JMS.
ConnectionConsumer	Para los servidores de aplicaciones, Connections proporciona un recurso especial para crear un ConnectionConsumer.
ConnectionFactory	ConnectionFactory encapsula un conjunto de parámetros de configuración de conexión que ha definido un administrador.
ConnectionMetaData	ConnectionMetaData proporciona información que describe la conexión.
DeliveryMode	Modalidades de entrega para las que ofrece soporte JMS.
Destination	La interfaz superior para Queue y Topic.
ExceptionListener*	Un escucha de excepciones se utiliza para recibir las excepciones que emiten las hebras de entrega asíncrona de conexiones (Connections).
MapMessage	MapMessage se utiliza para enviar un conjunto de pares de nombre-valor, donde los nombres son Series de caracteres (Strings) y los valores son tipos de primitivos de Java™.
Message	La interfaz Message es la interfaz raíz de todos los mensajes JMS.
MessageConsumer	La interfaz superior para todos los consumidores de mensajes.
MessageListener*	MessageListener se utiliza para recibir los mensajes entregados asíncronamente.

Tabla 26. Resumen de las interfaces (continuación)

Interfaz	Descripción
MessageProducer	Un cliente utiliza un MessageProducer para enviar mensajes a un destino (Destination).
ObjectMessage	ObjectMessage se utiliza para enviar un mensaje que contiene un objeto Java™ serializable.
Queue	Un objeto Queue encapsula el nombre de cola específico de un suministrador.
QueueBrowser	Un cliente utiliza QueueBrowser para ver los mensajes sin quitarlos de la cola.
QueueConnection	QueueConnection es una conexión activa con un suministrador punto a punto de JMS.
QueueConnectionFactory	Un cliente utiliza QueueConnectionFactory para crear QueueConnections con un suministrador punto a punto de JMS.
QueueReceiver	Un cliente utiliza un QueueReceiver para recibir mensajes que se han entregado en una cola.
QueueSender	Un cliente utiliza QueueSender para enviar mensajes a una cola.
QueueSession	QueueSession proporciona métodos para crear QueueReceivers, QueueSenders, QueueBrowsers y TemporaryQueues.
ServerSession **	ServerSession es un objeto implementado por un servidor de aplicaciones.
ServerSessionPool **	ServerSessionPool es un objeto implementado por un servidor de aplicaciones para proporcionar una agrupación de sesiones de servidor (ServerSessions) para procesar los mensajes de un consumidor de conexiones (ConnectionConsumer).
Session	Una sesión (Session) de JMS es un contexto de una sola hebra para producir y consumir mensajes.
StreamMessage	StreamMessage se utiliza para enviar una corriente de datos de primitivos de Java™.
TemporaryQueue	TemporaryQueue es un objeto Queue exclusivo que se crea para la duración de una QueueConnection.
TemporaryTopic	TemporaryTopic es un objeto Topic exclusivo que se crea para la duración de una TopicConnection.
TextMessage	TextMessage se utiliza para enviar un mensaje que contiene una `java.lang.String`.
Topic	Un objeto Topic encapsula el nombre de tema específico de un suministrador.
TopicConnection	TopicConnection es una conexión activa con un suministrador Pub/Sub JMS.
TopicConnectionFactory	Un cliente utiliza TopicConnectionFactory para crear TopicConnections con un suministrador de JMS Publish/Subscribe.
TopicPublisher	Un cliente utiliza TopicPublisher para publicar mensajes sobre un tema.
TopicSession	TopicSession proporciona métodos para crear TopicPublishers, TopicSubscribers y TemporaryTopics.

Tabla 26. Resumen de las interfaces (continuación)

Interfaz	Descripción
TopicSubscriber	Un cliente utiliza un TopicSubscriber para recibir mensajes que se han publicado de un tema.
XAConnection	XAConnection expande las posibilidades de Connection al proporcionar una XASession.
XAConnectionFactory	Algunos servidores de aplicaciones ofrecen soporte para agrupar el uso de recursos con posibilidades JTS (Java™ Transaction Service) en una transacción distribuida.
XAQueueConnection	XAQueueConnection proporciona las mismas opciones de creación que QueueConnection.
XAQueueConnectionFactory	XAQueueConnectionFactory proporciona las mismas opciones de creación que QueueConnectionFactory.
XAQueueSession	XAQueueSession proporciona una sesión QueueSession normal que se puede utilizar para crear QueueReceivers, QueueSenders y QueueBrowsers.
XASession	XASession expande las posibilidades de Session al añadir acceso al soporte de un suministrador de JMS para JTA (Java™ Transaction API).
XATopicConnection	XATopicConnection proporciona las mismas opciones de creación que TopicConnection.
XATopicConnectionFactory	XATopicConnectionFactory proporciona las mismas opciones de creación que TopicConnectionFactory.
XATopicSession	XATopicSession proporciona una TopicSession normal, que se puede utilizar para crear TopicSubscribers y TopicPublishers.

Tabla 27. Resumen de las clases

Clase	Descripción
QueueRequestor	JMS proporciona una clase de ayuda QueueRequestor para simplificar la emisión de peticiones de servicio.
TopicRequestor	JMS proporciona una clase TopicRequestor para simplificar la emisión de peticiones de servicio.

Clases JMS MQSeries

Las tablas siguientes contienen la lista de los paquetes **com.ibm.mq.jms** y **com.ibm.jms** que contienen las clases MQSeries que implementan las interfaces de Sun.

Tabla 28. Resumen de las clases del paquete 'com.ibm.mq.jms'

Clase	Implementa
MQConnection	Connection
MQConnectionConsumer	ConnectionConsumer
MQConnectionFactory	ConnectionFactory
MQConnectionMetaData	ConnectionMetaData
MQDestination	Destination
MQMessageConsumer	MessageConsumer
MQMessageProducer	MessageProducer
MQQueue	Queue
MQQueueBrowser	QueueBrowser
MQQueueConnection	QueueConnection
MQQueueConnectionFactory	QueueConnectionFactory
MQQueueEnumeration	java.util.Enumeration desde QueueBrowser
MQQueueReceiver	QueueReceiver
MQQueueSender	QueueSender
MQQueueSession	QueueSession
MQSession	Session
MQTemporaryQueue	TemporaryQueue
MQTemporaryTopic	TemporaryTopic
MQTopic	Topic
MQTopicConnection	TopicConnection
MQTopicConnectionFactory	TopicConnectionFactory
MQTopicPublisher	TopicPublisher
MQTopicSession	TopicSession
MQTopicSubscriber	TopicSubscriber
MQXAConnection	XAConnection
MQXAConnectionFactory	XAConnectionFactory
MQXAQueueConnection	XAQueueConnection
MQXAQueueConnectionFactory	XAQueueConnectionFactory
MQXAQueueSession	XAQueueSession
MQXASession	XASession
MQXATopicConnection	XATopicConnection
MQXATopicConnectionFactory	XATopicConnectionFactory
MQXATopicSession	XATopicSession

Tabla 29. Resumen de las clases del paquete 'com.ibm.jms'

Clase	Implementa
JMSBytesMessage	BytesMessage
JMSMapMessage	MapMessage
JMSMessage	Message
JMSObjectMessage	ObjectMessage
JMSStreamMessage	StreamMessage
JMSTextMessage	TextMessage

En este release de Servicio de mensajes de MQSeries Classes for Java se proporciona una implementación de ejemplo de las interfaces JMS siguientes:
- ServerSession
- ServerSessionPool

Consulte el apartado "Código de ejemplo del servidor de aplicaciones" en la página 232.

BytesMessage

interfaz pública **BytesMessage**
expande **Message**

Clase MQSeries: **JMSBytesMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

BytesMessage se utiliza para enviar un mensaje que contiene una corriente de bytes no interpretados. Hereda **Message** y añade un cuerpo de mensaje de bytes. El receptor del mensaje proporciona la interpretación de los bytes.

Nota: Este tipo de mensaje es para la codificación de cliente de formatos de mensaje existentes. Si es posible, se debe utilizar en su lugar otro de los tipos de mensaje autodefinidos.

Vea también: **MapMessage**, **Message**, **ObjectMessage**, **StreamMessage** y **TextMessage**.

Métodos

readBoolean

```
public boolean readBoolean() throws JMSException
```

Lee un `booleano` del mensaje de bytes.

Devuelve:
el valor `booleano` leído.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si es el final de los bytes del mensaje.

readByte

```
public byte readByte() throws JMSException
```

Lee un valor de 8 bits con signo del mensaje de bytes.

Devuelve:
el byte siguiente del mensaje de bytes como `byte` de 8 bits con signo.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readUnsignedByte

```
public int readUnsignedByte() throws JMSException
```

Lee un número de 8 bits sin signo del mensaje de bytes.

Devuelve:
el byte siguiente del mensaje de bytes, interpretado como un número de 8 bits sin signo.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readShort

```
public short readShort() throws JMSException
```

Lee un número de 16 bits con signo del mensaje de bytes.

Devuelve:
los dos bytes siguientes del mensaje de bytes, interpretados como un número de 16 bits con signo.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readUnsignedShort

```
public int readUnsignedShort() throws JMSException
```

Lee un número de 16 bits sin signo del mensaje de bytes.

Devuelve:
los dos bytes siguientes del mensaje de bytes, interpretados como un entero de 16 bits sin signo.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readChar

```
public char readChar() throws JMSException
```

Lee un valor de carácter Unicode del mensaje de bytes.

Devuelve:
los dos bytes siguientes del mensaje de bytes como un carácter Unicode.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readInt

```
public int readInt() throws JMSException
```

Lee un entero de 32 bits con signo del mensaje de bytes.

Devuelve:

los cuatro bytes siguientes del mensaje de bytes, interpretados como `int`.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readLong

```
public long readLong() throws JMSException
```

Lee un entero de 64 bits con signo del mensaje de bytes.

Devuelve:

los ocho bytes siguientes del mensaje de bytes, interpretados como `long`.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readFloat

```
public float readFloat() throws JMSException
```

Lee un valor `flotante` del mensaje de bytes.

Devuelve:

los cuatro bytes siguientes del mensaje de bytes, interpretados como valor `flotante`.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readDouble

```
public double readDouble() throws JMSException
```

Lee un `double` del mensaje de bytes.

Devuelve:
los ocho bytes siguientes del mensaje de bytes, interpretados como `double`.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readUTF

```
public java.lang.String readUTF() throws JMSException
```

Lee en el mensaje de bytes una serie de caracteres que se ha codificado utilizando un formato UTF-8 modificado. Los dos primeros caracteres se interpretan como un campo de 2 bytes de longitud.

Devuelve:
una serie de caracteres Unicode del mensaje de bytes.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- MessageEOFException - si es el final de los bytes del mensaje.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readBytes

```
public int readBytes(byte[] value) throws JMSException
```

Lee una matriz de bytes del mensaje de bytes. Si en la corriente de datos aún quedan suficientes bytes, se rellena todo el almacenamiento intermedio pero, si no es así, se rellena parcialmente.

Parámetros:
`value` - almacenamiento intermedio en el que se leen los datos.

Devuelve:
el número total de bytes leídos en el almacenamiento intermedio, o -1 si no hay más datos debido a que se ha llegado al final de los bytes.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

readBytes

```
public int readBytes(byte[] value, int length)
                                        throws JMSException
```

Lee una parte del mensaje de bytes.

Parámetros:

- `value` - almacenamiento intermedio en el que se leen los datos.
- `length` - número de bytes a leer.

Devuelve:

el número total de bytes leídos en el almacenamiento intermedio, o -1 si no hay más datos debido a que se ha llegado al final de los bytes.

Emite:

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.
- IndexOutOfBoundsException - si `length` es negativo, o es menor que la longitud de la matriz `value`
- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

Escribe un `booleano` en el mensaje de bytes como un valor de 1 byte. El valor `true` se escribe como valor `(byte)1` y el valor `false` se escribe como valor `(byte)0`.

Parámetros:

`value` - valor `booleano` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeByte

```
public void writeByte(byte value) throws JMSException
```

Escribe `byte` en el mensaje de bytes como un valor de 1 byte.

Parámetros:

`value` - valor `byte` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeShort

```
public void writeShort(short value) throws JMSException
```

Escribe un `short` en el mensaje de bytes como dos bytes.

Parámetros:

`value` - valor `short` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeChar

```
public void writeChar(char value) throws JMSException
```

Escribe un `char` en el mensaje de bytes como un valor de 2 bytes, el byte más significativo en primer lugar.

Parámetros:

`value` - valor `char` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeInt

```
public void writeInt(int value) throws JMSException
```

Escribe un `int` en el mensaje de bytes como cuatro bytes.

Parámetros:

`value` - valor `int` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeLong

```
public void writeLong(long value) throws JMSException
```

Escribe un `long` en el mensaje de bytes como ocho bytes.

Parámetros:

`value` - valor `long` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeFloat

```
public void writeFloat(float value) throws JMSException
```

Convierte el argumento del valor flotante en `int` utilizando el método `floatToIntBits` en la clase `Float` y, a continuación, escribe el valor `int` en el mensaje de bytes como una cantidad de 4 bytes.

Parámetros:

`value` - valor `flotante` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeDouble

```
public void writeDouble(double value) throws JMSException
```

Convierte el argumento double en `long` utilizando el método `doubleToLongBits` en la clase `Double` y, a continuación, escribe el valor `long` en el mensaje de bytes como una cantidad de 8 bytes.

Parámetros:

`value` - valor `double` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeUTF

```
public void writeUTF(java.lang.String value)
                              throws JMSException
```

Escribe una serie de caracteres en el mensaje de bytes utilizando codificación UTF-8 de un modo independiente de la máquina. La serie de caracteres UTF-8 que se escribe en el almacenamiento intermedio empieza con un campo de 2 bytes de longitud.

Parámetros:

`value` - valor `String` que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

Escribe una matriz de bytes en el mensaje de bytes.

Parámetros:

`value` - matriz de bytes que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeBytes

```
public void writeBytes(byte[] value,
                       int length) throws JMSException
```

Escribe una parte de una matriz de bytes en un mensaje de bytes.

Parámetros:

- `value` - valor de la matriz de bytes que se va escribir.
- `offset` - desplazamiento inicial en la matriz de bytes.
- `length` - número de bytes que se van a utilizar.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

writeObject

```
public void writeObject(java.lang.Object value)
                                        throws JMSException
```

Escribe un objeto Java™ en el mensaje de bytes.

Nota: Este método sólo funciona para los tipos de objeto de primitivos (como, por ejemplo, Integer, Double y Long), Series de caracteres (Strings) y matrices de bytes.

Parámetros:

`value` - objeto Java™ que se va a escribir.

Emite:

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- MessageFormatException - si el tipo de objeto no es válido.
- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

reset

```
public void reset() throws JMSException
```

Pone el cuerpo del mensaje en modalidad de sólo lectura y vuelve a situar los bytes de bytes al principio.

Emite:

- JMSException - si JMS no puede restablecer el mensaje debido a un error interno de JMS.
- MessageFormatException - si el formato del mensaje no es válido

Connection

interfaz pública **Connection**
Subinterfaces: **QueueConnection**, **TopicConnection**, **XAQueueConnection** y **XATopicConnection**

Clase MQSeries: **MQConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

JMS Connection es una conexión activa del cliente a su suministrador de JMS.

Vea también: **QueueConnection**, **TopicConnection**, **XAQueueConnection** y **XATopicConnection**

Métodos

getClientID

```
public java.lang.String getClientID()
                                throws JMSException
```

Obtiene el identificador de cliente para esta conexión. El administrador puede configurar previamente el identificador de cliente en un ConnectionFactory, o se puede asignar invocando a setClientId.

Devuelve:
el identificador de cliente exclusivo.

Emite: JMSException - si la implementación de JMS no devuelve el ID de cliente para esta conexión debido a un error interno.

setClientID

```
public void setClientID(java.lang.String clientID)
                                          throws JMSException
```

Establece el identificador de cliente para esta conexión.

Nota: Se ignora el identificador de cliente para las conexiones punto a punto.

Parámetros:
clientID - identificador de cliente exclusivo.

Emite:

- JMSException - si la implementación de JMS no puede establecer el ID de cliente para esta conexión debido a un error interno.
- InvalidClientIDException - si el cliente JMS especifica un ID de cliente duplicado o no válido.
- IllegalStateException - si se intenta establecer un identificador de cliente de la conexión en un momento no adecuado o si se ha configurado de modo administrativo.

getMetaData

```
public ConnectionMetaData getMetaData() throws JMSException
```

Obtiene los metadatos para esta conexión.

Devuelve:
los metadatos de la conexión.

Emite: JMSException - excepción general si la implementación de JMS no puede obtener los metadatos de la conexión para esta conexión.

Vea también:
ConnectionMetaData

getExceptionListener

```
public ExceptionListener getExceptionListener()
                                                throws JMSException
```

Obtiene ExceptionListener para esta conexión.

Devuelve:
ExceptionListener para esta conexión

Emite: JMSException - excepción general si la implementación de JMS no puede obtener el escucha de excepciones para esta conexión.

setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
                                                              throws JMSException
```

Establece un escucha de excepciones para esta conexión.

Parámetros:
handler - escucha de excepciones.

Emite: JMSException - excepción general si la implementación de JMS no puede establecer el escucha de excepciones para esta conexión.

start

```
public void start() throws JMSException
```

Inicia (o reinicia) la entrega de mensajes entrantes de una conexión. Se ignora el inicio de una sesión iniciada.

Emite: JMSException - si la implementación de JMS no inicia la entrega de mensajes debido a un error interno.

Vea también:
stop

stop

```
public void stop() throws JMSException
```

Se utiliza para detener temporalmente la entrega de mensajes entrantes de una conexión. Se puede reiniciar utilizando su método `start`. Cuando se detiene, se inhibe la entrega a todos los consumidores de mensajes de la conexión. Las recepciones síncronas se bloquean y no se entregan los mensajes a los escuchas de mensajes.

La detención de una sesión no afecta a su posibilidad de enviar mensajes. Se ignora la detención de una sesión detenida.

Emite: JMSException - si la implementación de JMS no detiene la entrega de mensajes debido a un error interno.

Vea también:
start

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de una conexión, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez. No se necesario cerrar las sesiones, los productores y los consumidores de una conexión cerrada.

El cierre de una conexión hace que se restituyan todas las transacciones en proceso de sus sesiones. En caso de que un gestor de transacciones externo coordine el trabajo de una sesión, cuando se utiliza XASession, no se utilizan los métodos de confirmación y restitución de la sesión y un gestor de transacciones determina posteriormente el resultado del trabajo de una sesión cerrada. Cerrar una conexión NO fuerza un reconocimiento de las sesiones reconocidas del cliente.

MQ JMS mantiene una agrupación MQSeries hConns disponible para que la utilicen las sesiones. Bajo determinadas circunstancias, Connection.close() borra esta agrupación. Si una aplicación utiliza varias conexiones de modo secuencial, se puede forzar la agrupación para que permanezca activa entre conexiones JMS. Para hacerlo, registre un MQPoolToken con com.ibm.mq.MQEnvironment para la duración de la aplicación JMS. Para obtener más información, consulte el apartado “Agrupación de conexiones” en la página 71 y “MQEnvironment” en la página 97.

Emite: JMSException - si la implementación de JMS no cierra la conexión debido a un error interno. Algunos ejemplos cuando se produce una anomalía al liberar recursos o cerrar una conexión de socket.

ConnectionConsumer

interfaz pública **ConnectionConsumer**

Clase MQSeries: **MQConnectionConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionConsumer
```

Para los servidores de aplicaciones, Connections proporciona un recurso especial para crear un ConnectionConsumer. Un Destino (Destination) y un Selector de propiedades (Property Selector) especifican los mensajes que va a consumir. Además, un ConnectionConsumer debe disponer de un ServerSessionPool para utilizarlo para procesar sus mensajes.

Vea también: **QueueConnection** y **TopicConnection**.

Métodos

close()

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un ConnectionConsumer, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Emite: JMSException - si una implementación de JMS no puede liberar recursos en nombre de ConnectionConsumer, o si no puede cerrar el consumidor de conexión.

getServerSessionPool()

```
public ServerSessionPool getServerSessionPool()
                                          throws JMSException
```

Obtiene una sesión de servidor asociada a este consumidor de conexión.

Devuelve:
la agrupación de sesiones de servidor que utiliza este consumidor de conexión.

Emite: JMSException - si una implementación de JMS no puede obtener la agrupación de sesiones de servidor asociada a este consumidor de conexión debido a un error interno.

ConnectionFactory

interfaz pública **ConnectionFactory**
Subinterfaces: **QueueConnectionFactory**, **TopicConnectionFactory**, **XAQueueConnectionFactory** y **XATopicConnectionFactory**

Clase MQSeries: **MQConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

ConnectionFactory encapsula un conjunto de parámetros de configuración de conexión que ha definido un administrador. Un cliente lo utiliza para crear una conexión con un suministrador de JMS.

Vea también: **QueueConnectionFactory**, **TopicConnectionFactory**, **XAQueueConnectionFactory** y **XATopicConnectionFactory**

Constructor de MQSeries

MQConnectionFactory

```
public MQConnectionFactory()
```

Métodos

setDescription *

```
public void setDescription(String x)
```

Breve descripción del objeto.

getDescription *

```
public String getDescription()
```

Recupera la descripción del objeto.

setTransportType *

```
public void setTransportType(int x) throws JMSException
```

Establece el tipo de transporte que se va a utilizar. Puede ser JMSC.MQJMS_TP_BINDINGS_MQ o JMSC.MQJMS_TP_CLIENT_MQ_TCPIP.

getTransportType *

```
public int getTransportType()
```

Recupera el tipo de transporte.

setClientId *

```
public void setClientId(String x)
```

Establece el identificador de cliente que se va a utilizar para todas las conexiones que se crean utilizando esta conexión.

getClientId *

```
public String getClientId()
```

Obtiene el identificador de cliente que se va a utilizar para todas las conexiones que se crean utilizando esta ConnectionFactory.

setQueueManager *

```
public void setQueueManager(String x) throws JMSException
```

Establece el nombre del gestor de colas al que se va a conectar.

getQueueManager *

```
public String getQueueManager()
```

Obtiene el nombre del gestor de colas.

setHostName *

```
public void setHostName(String hostname)
```

Sólo para clientes, el nombre del sistema principal al que conectar.

getHostName *

```
public String getHostName()
```

Recupera el nombre del sistema principal.

setPort *

```
public void setPort(int port) throws JMSException
```

Establece el puerto para un conexión de cliente.

Parámetros:
port - nuevo valor a utilizar.

Emite: JMSException si el puerto es negativo.

getPort *

```
public int getPort()
```

Sólo para las conexiones de cliente, obtiene el número de puerto.

setChannel *

```
public void setChannel(String x) throws JMSException
```

Sólo para clientes, establece el canal a utilizar.

getChannel *

```
public String getChannel()
```

Sólo para clientes, obtiene el canal que se ha utilizado.

setCCSID *

```
public void setCCSID(int x) throws JMSException
```

Establece el juego de caracteres que se va a utilizar al conectar al gestor de colas. Consulte la Tabla 13 en la página 115, donde se proporciona una lista de los valores permitidos. En la mayor parte de la situaciones, se aconseja utilizar el valor por omisión (819).

getCCSID *

```
public int getCCSID()
```

Obtiene el juego de caracteres del gestor de colas.

setReceiveExit *

```
public void setReceiveExit(String receiveExit)
```

Nombre de una clase que implementa una salida de recepción.

getReceiveExit *

```
public String getReceiveExit()
```

Obtiene el nombre de la clase de salida de recepción.

setReceiveExitInit *

```
public void setReceiveExitInit(String x)
```

Serie de caracteres de inicialización que se pasa al constructor de la clase de salida de recepción.

getReceiveExitInit *

```
public String getReceiveExitInit()
```

Obtiene la serie de caracteres de inicialización que se ha pasado a la clase de salida de recepción.

setSecurityExit *

```
public void setSecurityExit(String securityExit)
```

Nombre de una clase que implementa una rutina de salida de seguridad.

getSecurityExit *

```
public String getSecurityExit()
```

Obtiene el nombre de la clase de rutina de salida de seguridad.

setSecurityExitInit *

```
public void setSecurityExitInit(String x)
```

Serie de caracteres de inicialización que se pasa al constructor de la rutina de salida de seguridad.

getSecurityExitInit *

```
public String getSecurityExitInit()
```

Obtiene la serie de caracteres de inicialización de la rutina de salida de seguridad.

setSendExit *

```
public void setSendExit(String sendExit)
```

Nombre de la clase que implementa una salida de emisión.

getSendExit *

```
public String getSendExit()
```

Obtiene el nombre de la clase de salida de emisión.

setSendExitInit *

```
public void setSendExitInit(String x)
```

Serie de caracteres de inicialización que se pasa al constructor de la rutina de salida de emisión.

getSendExitInit *

```
public String getSendExitInit()
```

Obtiene la serie de caracteres de inicialización de la rutina de salida de emisión.

ConnectionMetaData

interfaz pública **ConnectionMetaData**

Clase MQSeries: **MQConnectionMetaData**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

ConnectionMetaData proporciona información que describe la conexión.

Constructor de MQSeries

MQConnectionMetaData

```
public MQConnectionMetaData()
```

Métodos

getJMSVersion

```
public java.lang.String getJMSVersion() throws JMSException
```

Obtiene la versión de JMS.

Devuelve:
la versión de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

getJMSMajorVersion

```
public int getJMSMajorVersion() throws JMSException
```

Obtiene el número de versión principal de JMS.

Devuelve:
el número de versión principal de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

getJMSMinorVersion

```
public int getJMSMinorVersion() throws JMSException
```

Obtiene el número de versión secundario de JMS.

Devuelve:
el número de versión secundario de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

getJMSXPropertyNames

```
public java.util.Enumeration getJMSXPropertyNames()
                                                throws JMSException
```

Obtiene una enumeración de los nombres de las Propiedades de JMSX para los que ofrece soporte esta conexión.

Devuelve:
una enumeración de JMSX PropertyNames.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los nombres de propiedades.

getJMSProviderName

```
public java.lang.String getJMSProviderName()
                                        throws JMSException
```

Obtiene el nombre del suministrador de JMS.

Devuelve:
el nombre del suministrador de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

getProviderVersion

```
public java.lang.String getProviderVersion()
                                         throws JMSException
```

Obtiene la versión del suministrador de JMS.

Devuelve:
la versión del suministrador de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

getProviderMajorVersion

```
public int getProviderMajorVersion() throws JMSException
```

Obtiene el número de versión principal del suministrador de JMS.

Devuelve:
el número de versión principal del suministrador de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

getProviderMinorVersion

```
public int getProviderMinorVersion() throws JMSException
```

Obtiene el número de versión secundario del suministrador de JMS.

Devuelve:
el número de versión secundario del suministrador de JMS.

Emite: JMSException - si se produce un error interno en la implementación de JMS durante la recuperación de los metadatos.

toString *

```
public String toString()
```

Altera temporalmente:
toString en la clase Object.

DeliveryMode

interfaz pública **DeliveryMode**

Modalidades de entrega para las que ofrece soporte JMS.

Campos

NON_PERSISTENT

```
public static final int NON_PERSISTENT
```

Modalidad de entrega de menor actividad general, puesto que no necesita que se haya anotado cronológicamente el mensaje en almacenamiento estable.

PERSISTENT

```
public static final int PERSISTENT
```

Esta modalidad indica al suministrador de JMS que anote cronológicamente el mensaje en almacenamiento estable como parte de la operación de envío del cliente.

Destination

interfaz pública **Destination**
Subinterfaces: **Queue**, **TemporaryQueue**, **TemporaryTopic** y **Topic**

Clase MQSeries: **MQDestination**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

El objeto Destination encapsula las direcciones específicas del suministrador.

Vea también: **Queue**, **TemporaryQueue**, **TemporaryTopic** y **Topic**

Constructores de MQSeries

MQDestination

```
public MQDestination()
```

Métodos

setDescription *

```
public void setDescription(String x)
```

Breve descripción del objeto.

getDescription *

```
public String getDescription()
```

Obtiene la descripción del objeto.

setPriority *

```
public void setPriority(int priority) throws JMSException
```

Se utiliza para alterar temporalmente la prioridad de todos los mensajes enviados a este destino.

getPriority *

```
public int getPriority()
```

Obtiene el valor de prioridad alterado temporalmente.

setExpiry *

```
public void setExpiry(int expiry) throws JMSException
```

Se utiliza para alterar temporalmente la caducidad de todos los mensajes enviados a este destino.

getExpiry *

```
public int getExpiry()
```

Obtiene el valor de la caducidad para este destino.

setPersistence *

```
public void setPersistence(int persistence)
                                    throws JMSException
```

Se utiliza para alterar temporalmente la permanencia de todos los mensajes enviados a este destino.

getPersistence *

```
public int getPersistence()
```

Obtiene el valor de la permanencia para este destino.

setTargetClient *

```
public void setTargetClient(int targetClient)
                                    throws JMSException
```

Distintivo que especifica si la aplicación remota es compatible con JMS.

getTargetClient *

```
public int getTargetClient()
```

Obtiene el distintivo de conforme a las normas de JMS.

setCCSID *

```
public void setCCSID(int x) throws JMSException
```

Juego de caracteres que se va a utilizar para codificar series de caracteres de texto en los mensajes que se envían a este destino. Consulte la Tabla 13 en la página 115, donde se proporciona una lista de los valores permitidos. El valor por omisión es 1208 (UTF8).

getCCSID *

```
public int getCCSID()
```

Obtiene el nombre del juego de caracteres que utiliza este destino.

setEncoding *

```
public void setEncoding(int x) throws JMSException
```

Especifica la codificación que se va a utilizar para los campos numéricos en los mensajes que se envían a este destino. Consulte la Tabla 13 en la página 115, donde se proporciona una lista de los valores permitidos.

getEncoding *

```
public int getEncoding()
```

Obtiene la codificación que se va a utilizar para este destino.

ExceptionListener

interfaz pública **ExceptionListener**

Si un suministrador de JMS detecta un problema grave con una conexión, informa al ExceptionListener de la conexión, si tiene uno registrado. Para hacerlo, invoca al método onException() del escucha y le pasa una JMSException que describe el problema.

Así, se puede notificar un problema al cliente de modo asíncrono. Algunas conexiones sólo consumen mensajes, por lo que no tienen otro modo de enterarse de que la conexión no se ha ejecutado correctamente.

Se entregan excepciones cuando:
- Existe una anomalía en la recepción de un mensaje asíncrono
- Un mensaje emite una excepción de ejecución

Métodos

onException

```
public void onException(JMSException exception)
```

Notifica al usuario una excepción de JMS.

Parámetros:

exception - excepción de JMS. Estas excepciones son el resultado de la entrega de mensajes asíncronos. Por lo general, indican que se ha producido un problema al recibir un mensaje del gestor de colas o, posiblemente, un error interno en la implementación de JMS.

MapMessage

interfaz pública **MapMessage**
expande **Message**

Clase MQSeries: **JMSMapMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

MapMessage se utiliza para enviar un conjunto de pares de nombre-valor, donde los nombres son Series de caracteres (Strings) y los valores son tipos de primitivos de Java™. Se puede acceder a las entradas de modo secuencial o aleatoria, por nombre. El orden de las entradas no está definido.

Vea también: **BytesMessage**, **Message**, **ObjectMessage**, **StreamMessage** y **TextMessage**

Métodos

getBoolean

```
public boolean getBoolean(java.lang.String name)
                                                throws JMSException
```

Devuelve el valor booleano con el nombre especificado.

Parámetros:
name - nombre del valor booleano

Devuelve:
el valor booleano con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getByte

```
public byte getByte(java.lang.String name)
                                       throws JMSException
```

Devuelve el valor del byte con el nombre especificado.

Parámetros:
name - nombre del byte.

Devuelve:
el valor del byte con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getShort

```
public short getShort(java.lang.String name) throws JMSException
```

Devuelve el valor corto con el nombre especificado.

Parámetros:
name - nombre del valor corto.

Devuelve:
el valor corto con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getChar

```
public char getChar(java.lang.String name)
                                        throws JMSException
```

Devuelve el valor del carácter Unicode con el nombre especificado.

Parámetros:
name - nombre del carácter Unicode.

Devuelve:
el valor del carácter Unicode con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getInt

```
public int getInt(java.lang.String name)
                                              throws JMSException
```

Devuelve el valor de entero con el nombre especificado.

Parámetros:
name - nombre del entero.

Devuelve:
el valor de entero con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getLong

```
public long getLong(java.lang.String name)
                                            throws JMSException
```

Devuelve el valor largo con el nombre especificado.

Parámetros:
name - nombre del valor largo.

Devuelve:
el valor largo con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getFloat

```
public float getFloat(java.lang.String name) throws JMSException
```

Devuelve el valor flotante con el nombre especificado.

Parámetros:
name - nombre del valor flotante.

Devuelve:
el valor flotante con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getDouble

```
public double getDouble(java.lang.String name) throws JMSException
```

Devuelve el valor doble con el nombre especificado.

Parámetros:
name - nombre del valor doble.

Devuelve:
el valor doble con el nombre especificado.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getString

```
public java.lang.String getString(java.lang.String name)
                                                  throws JMSException
```

Devuelve el valor de la Serie (String) con el nombre especificado.

Parámetros:
name - nombre de la Serie.

Devuelve:
el valor de la Serie con el nombre especificado. Si no existe ningún elemento con este nombre, se devuelve un valor nulo.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

- MessageFormatException - si esta conversión de tipos no es válida.

getBytes

```
public byte[] getBytes(java.lang.String name) throws JMSException
```

Devuelve el valor de la matriz de bytes con el nombre especificado.

Parámetros:
name - nombre de la matriz de bytes.

Devuelve:
una copia del valor de la matriz de bytes con el nombre especificado. Si no existe ningún elemento con este nombre, se devuelve un valor nulo.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getObject

```
public java.lang.Object getObject(java.lang.String name)
                                  throws JMSException
```

Devuelve el valor de objeto Java™ con el nombre especificado. Este método devuelve, en formato de objeto, un valor que se ha almacenado en la correlación utilizando la invocación al método setObject o el método set primitivo equivalente.

Parámetros:
name - nombre del objeto Java™.

Devuelve:
una copia del valor del objeto Java™ con el nombre especificado, en formato de objeto (si se ha establecido como int, se devuelve un entero). Si no existe ningún elemento con este nombre, se devuelve un valor nulo.

Emite: JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

getMapNames

```
public java.util.Enumeration getMapNames() throws JMSException
```

Devuelve una enumeración de todos los nombres del mensaje de correlación (Map).

Devuelve:
una enumeración de todos los nombres del mensaje de correlación.

Emite: JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.

setBoolean

```
public void setBoolean(java.lang.String name,
                       boolean value) throws JMSException
```

Establece un valor booleano con el nombre especificado en la correlación.

Parámetros:

- name - nombre del valor booleano.
- value - valor booleano que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setByte

```
public void setByte(java.lang.String name,
                    byte value) throws JMSException
```

Establece un valor de byte con el nombre especificado en la correlación.

Parámetros:

- name - nombre del byte.
- value - valor booleano que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setShort

```
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

Establece un valor corto con el nombre especificado en la correlación.

Parámetros:

- name - nombre del valor corto.
- value - valor corto que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setChar

```
public void setChar(java.lang.String name,
                    char value) throws JMSException
```

Establece un valor de carácter Unicode con el nombre especificado en la correlación.

Parámetros:

- name - nombre del carácter Unicode.
- value - valor de carácter Unicode que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setInt

```
public void setInt(java.lang.String name,
                   int value) throws JMSException
```

Establece un valor de entero con el nombre especificado en la correlación.

Parámetros:

- name - nombre del entero.
- value - valor de entero que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setLong

```
public void setLong(java.lang.String name,
                    long value) throws JMSException
```

Establece un valor largo con el nombre especificado en la correlación.

Parámetros:

- name - nombre del valor largo.
- value - valor largo que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setFloat

```
public void setFloat(java.lang.String name,
                     float value) throws JMSException
```

Establece un valor flotante con el nombre especificado en la correlación.

Parámetros:

- name - nombre del valor flotante.
- value - valor flotante que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setDouble

```
public void setDouble(java.lang.String name,
                      double value) throws JMSException
```

Establece un valor doble con el nombre especificado en la correlación.

Parámetros:

- name - nombre del valor doble.
- value - valor doble que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setString

```
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

Establece un valor de Serie (String) con el nombre especificado en la correlación.

Parámetros:

- name - nombre de la Serie.
- value - valor de la serie que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value) throws JMSException
```

Establece un valor de matriz de bytes con el nombre especificado en la correlación.

Parámetros:

- name - nombre de la matriz de bytes.
- value - valor de la matriz de bytes que se va a establecer en la correlación.

  Se copia la matriz, por lo que las modificaciones que se realizan posteriormente en la matriz no modifican el valor de la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value,
                     int offset,
                     int length) throws JMSException
```

Establece una parte del valor de la matriz de bytes con el nombre especificado en la correlación.

Se copia la matriz, por lo que las modificaciones que se realizan posteriormente en la matriz no modifican el valor de la correlación.

Parámetros:

- name - nombre de la matriz de bytes.
- value - valor de la matriz de bytes que se va a establecer en la correlación.
- offset - desplazamiento inicial en la matriz de bytes.
- length - número de bytes a copiar.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

setObject

```
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

Establece un valor de objeto Java™ con el nombre especificado en la correlación. Este método sólo funciona para los tipos de primitivos de objeto (como, por ejemplo, Integer, Double, Long), series de caracteres (Strings) y matrices de bytes.

Parámetros:

- name - nombre del objeto Java™.
- value - valor de objeto Java™ que se va a establecer en la correlación.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a algún error interno de JMS.
- MessageFormatException - si el objeto no es válido.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

itemExists

```
public boolean itemExists(java.lang.String name)
                                      throws JMSException
```

Comprueba si existe un elemento en este MapMessage.

Parámetros:
name - nombre del elemento a comprobar.

Devuelve:
verdadero (true) si el elemento existe.

Emite: JMSException - si se produce un error de JMS.

Message

interfaz pública **Message**
Subinterfaces: **BytesMessage**, **MapMessage**, **ObjectMessage**, **StreamMessage** y **TextMessage**

Clase MQSeries: **JMSMessage**

```
java.lang.Object
  |
  +----com.ibm.jms.MQJMSMessage
```

La interfaz Message es la interfaz raíz de todos los mensajes JMS. Define la cabecera JMS y el método de reconocimiento que se utiliza para todos los mensajes.

Campos

DEFAULT_DELIVERY_MODE

```
public static final int DEFAULT_DELIVERY_MODE
```

Valor de la modalidad de entrega por omisión.

DEFAULT_PRIORITY

```
public static final int DEFAULT_PRIORITY
```

Valor de la prioridad por omisión.

DEFAULT_TIME_TO_LIVE

```
public static final long DEFAULT_TIME_TO_LIVE
```

Valor del tiempo de vida por omisión.

Métodos

getJMSMessageID

```
public java.lang.String getJMSMessageID()
                                    throws JMSException
```

Obtiene el ID del mensaje.

Devuelve:
el ID del mensaje.

Emite: JMSException - si JMS no puede obtener el ID del mensaje debido a un error interno de JMS.

Vea también:
setJMSMessageID()

setJMSMessageID

```
public void setJMSMessageID(java.lang.String id)
                                          throws JMSException
```

Establece el ID del mensaje.

Todos los valores que se establecen utilizando este método se ignoran cuando se envía el mensaje, pero este método se puede utilizar para cambiar el valor en un mensaje recibido.

Parámetros:
id - ID del mensaje.

Emite: JMSException - si JMS no puede establecer el ID del mensaje debido a un error interno de JMS.

Vea también:
getJMSMessageID()

getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException
```

Obtiene la indicación de la hora del mensaje.

Devuelve:
la indicación de la hora del mensaje.

Emite: JMSException - si JMS no puede obtener la indicación de la hora debido a un error interno de JMS.

Vea también:
setJMSTimestamp()

setJMSTimestamp

```
public void setJMSTimestamp(long timestamp)
                                        throws JMSException
```

Establece la indicación de la hora del mensaje.

Todos los valores que se establecen utilizando este método se ignoran cuando se envía el mensaje, pero este método se puede utilizar para cambiar el valor de un mensaje recibido.

Parámetros:
timestamp - indicación de la hora de este mensaje.

Emite: JMSException - si JMS no puede establecer la indicación de la hora debido a un error interno de JMS.

Vea también:
getJMSTimestamp()

getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes()
                                                    throws JMSException
```

Obtiene el ID de correlación como una matriz de bytes para el mensaje.

Devuelve:
el ID de correlación de un mensaje como una matriz de bytes.

Emite: JMSException - si JMS no puede obtener el ID de correlación debido a un error interno de JMS.

Vea también:
setJMSCorrelationID(), getJMSCorrelationID(), setJMSCorrelationIDAsBytes()

setJMSCorrelationIDAsBytes

```
public void setJMSCorrelationIDAsBytes(byte[]
                                       correlationID)
                                              throws JMSException
```

Establece el ID de correlación como una matriz de bytes para el mensaje. Un cliente puede utilizar esta invocación para establecer el correlationID igual a un messageID de un mensaje anterior o a una serie específica de la aplicación. Las series específicas de aplicación no deben empezar con los ID de caracteres:

Parámetros:
correlationID - ID de correlación como una serie, o el ID de mensaje de un mensaje al que se hace referencia.

Emite: JMSException - si JMS no puede establecer el ID de correlación debido a un error interno de JMS.

Vea también:
setJMSCorrelationID(), getJMSCorrelationID(), getJMSCorrelationIDAsBytes()

getJMSCorrelationID

```
public java.lang.String getJMSCorrelationID()
                                          throws JMSException
```

Obtiene el ID de correlación para el mensaje.

Devuelve:
el ID de correlación de un mensaje como una serie.

Emite: JMSException - si JMS no puede obtener el ID de correlación debido a un error interno de JMS.

Vea también:
setJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()

setJMSCorrelationID

```
public void setJMSCorrelationID
                 (java.lang.String correlationID)
                                        throws JMSException
```

Establece el ID de correlación para el mensaje.

Un cliente puede utilizar el campo de cabecera JMSCorrelationID para enlazar un mensaje con otro. Una utilización habitual consiste en enlazar un mensaje de respuesta con un mensaje de solicitud.

Nota: El uso de un valor de byte[] para JMSCorrelationID no es portátil.

Parámetros:
correlationID - ID de mensaje de un mensaje al que se hace referencia.

Emite: JMSException - si JMS no puede establecer el ID de correlación debido a un error interno de JMS.

Vea también:
getJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()

getJMSReplyTo

```
public Destination getJMSReplyTo() throws JMSException
```

Obtiene si se debe enviar una respuesta a este mensaje.

Devuelve:
si se debe enviar una respuesta a este mensaje

Emite: JMSException - si JMS no puede obtener el destino ReplyTo debido a un error interno de JMS.

Vea también:
setJMSReplyTo()

setJMSReplyTo

```
public void setJMSReplyTo(Destination replyTo)
                                          throws JMSException
```

Establece si se debe enviar una respuesta a este mensaje.

Parámetros:
replyTo - si se debe enviar una respuesta a este mensaje. Un valor nulo indica que no se espera ninguna respuesta.

Emite: JMSException - si JMS no puede establecer un destino ReplyTo debido a un error interno de JMS.

Vea también:
getJMSReplyTo()

getJMSDestination

```
public Destination getJMSDestination() throws JMSException
```

Obtiene el destino de este mensaje.

Devuelve:
el destino de este mensaje.

Emite: JMSException - si JMS no puede obtener el destino de JMS debido a un error interno de JMS.

Vea también:
setJMSDestination()

setJMSDestination

```
public void setJMSDestination(Destination destination)
                                                 throws JMSexception
```

Establece el destino de este mensaje.

Todos los valores que se establecen utilizando este método se ignoran cuando se envía el mensaje, pero este método se puede utilizar para cambiar el valor de un mensaje recibido.

Parámetros:
destination - destino de este mensaje.

Emite: JMSException - si JMS no puede establecer el destino de JMS debido a un error interno de JMS.

Vea también:
getJMSDestination()

getJMSDeliveryMode

```
public int getJMSDeliveryMode() throws JMSException
```

Obtiene la modalidad de entrega de este mensaje.

Devuelve:
la modalidad de entrega de este mensaje.

Emite: JMSException - si JMS no puede obtener la DeliveryMode de JMS debido a un error interno de JMS.

Vea también:
setJMSDeliveryMode(), DeliveryMode

setJMSDeliveryMode

```
public void setJMSDeliveryMode(int deliveryMode)
                                          throws JMSException
```

Establece la modalidad de entrega de este mensaje.

Todos los valores que se establecen utilizando este método se ignoran cuando se envía el mensaje, pero este método se puede utilizar para cambiar el valor en un mensaje recibido.

Para modificar la modalidad de entrega cuando se envía el mensaje, utilice el método setDeliveryMode en QueueSender o TopicPublisher (este método se hereda de MessageProducer).

Parámetros:
deliveryMode - modalidad de entrega de este mensaje.

Emite: JMSException - si JMS no puede establecer la DeliveryMode de JMS debido a un error interno de JMS.

Vea también:
getJMSDeliveryMode(), DeliveryMode

getJMSRedelivered

```
public boolean getJMSRedelivered() throws JMSException
```

Obtiene una indicación sobre si se está volviendo a entregar este mensaje.

Si un cliente recibe un mensaje con el indicador de nueva entrega establecido, es probable, aunque no se garantiza, que este mensaje se haya entregado anteriormente al cliente, pero que éste no haya acusado su recibo.

Devuelve:
establecido en verdadero si se está efectuando una nueva entrega de este mensaje.

Emite: JMSException - si JMS no puede obtener el distintivo de nueva entrega de JMS debido a un error interno de JMS.

Vea también:
setJMSRedelivered()

setJMSRedelivered

```
public void setJMSRedelivered(boolean redelivered)
                                            throws JMSException
```

Se establece para indicar si se está volviendo a entregar este mensaje.

Todos los valores que se establecen utilizando este método se ignoran cuando se envía el mensaje, pero este método se puede utilizar para cambiar el valor en un mensaje recibido.

Parámetros:
redelivered - indicación de si se está volviendo a entregar este mensaje.

Emite: JMSException - si JMS no puede establecer el distintivo JMSRedelivered debido a un error interno de JMS.

Vea también:
getJMSRedelivered()

getJMSType

```
public java.lang.String getJMSType() throws JMSException
```

Obtiene el tipo de mensaje.

Devuelve:
el tipo de mensaje.

Emite: JMSException - si JMS no puede obtener el tipo de mensaje JMS debido a un error interno de JMS.

Vea también:
setJMSType()

setJMSType

```
public void setJMSType(java.lang.String type)
                                        throws JMSException
```

Establece el tipo de mensaje.

Los clientes JMS deben asignar un valor para el tipo tanto si lo utiliza las aplicación como si no. De este modo, se asegura que esté correctamente establecido para los suministradores que lo necesitan.

Parámetros:
type - clase de mensaje.

Emite: JMSException - si JMS no puede establecer el tipo de mensaje JMS debido a un error interno de JMS.

Vea también:
getJMSType()

getJMSExpiration

```
public long getJMSExpiration() throws JMSException
```

Obtiene el valor de caducidad del mensaje.

Devuelve:
la hora a la que caduca el mensaje. Es la suma del valor del tiempo de vida que ha especificado el cliente y la hora de GMT en el momento del envío.

Emite: JMSException - si JMS no puede obtener la caducidad del mensaje JMS debido a un error interno de JMS.

Vea también:
setJMSExpiration()

setJMSExpiration

```
public void setJMSExpiration(long expiration)
                                          throws JMSException
```

Establece el valor de caducidad del mensaje.

Todos los valores que se establecen utilizando este método se ignoran cuando se envía el mensaje, pero este método se puede utilizar para cambiar el valor de un mensaje recibido.

Parámetros:
expiration - hora de caducidad del mensaje.

Emite: JMSException - si JMS no puede establecer la caducidad del mensaje JMS debido a un error interno de JMS.

Vea también:
getJMSExpiration()

getJMSPriority

```
public int getJMSPriority() throws JMSException
```

Obtiene la prioridad del mensaje.

Devuelve:
la prioridad del mensaje.

Emite: JMSException - si JMS no puede obtener la prioridad del mensaje JMS debido a un error interno de JMS.

Vea también:
setJMSPriority() para los niveles de prioridad

setJMSPriority

```
public void setJMSPriority(int priority)
                                       throws JMSException
```

Establece la prioridad para este mensaje.

JMS define un valor de prioridad de diez niveles, donde 0 es la prioridad más baja y 9 la más alta. Además, los clientes deben considerar las prioridades de 0 a 4 como gradaciones de la prioridad normal, y las prioridades de 5 a 9 como gradaciones de la prioridad urgente.

Parámetros:
priority - prioridad de este mensaje.

Emite: JMSException - si JMS no puede establecer la prioridad del mensaje JMS debido a un error interno de JMS.

Vea también:
getJMSPriority()

clearProperties

```
public void clearProperties() throws JMSException
```

Borra las propiedades de un mensaje. Los campos de la cabecera y el cuerpo del mensaje no se borran.

Emite: JMSException - si JMS no puede borrar las propiedades del mensaje JMS debido a un error interno de JMS.

propertyExists

```
public boolean propertyExists(java.lang.String name)
                                                throws JMSException
```

Comprueba si existe un valor de propiedad.

Parámetros:
name - nombre de la propiedad a comprobar.

Devuelve:
verdadero (true) si la propiedad existe.

Emite: JMSException - si JMS no puede comprobar si existe la propiedad debido a un error interno de JMS.

getBooleanProperty

```
public boolean getBooleanProperty(java.lang.String name)
                                                    throws JMSException
```

Devuelve el valor de propiedad de booleano con el nombre especificado.

Parámetros:
name - nombre de la propiedad de booleano.

Devuelve:
el valor de propiedad de booleano con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida

getByteProperty

```
public byte getByteProperty(java.lang.String name)
                                                throws JMSException
```

Devuelve el valor de propiedad de byte con el nombre especificado.

Parámetros:
name - nombre de la propiedad de byte.

Devuelve:
el valor de propiedad de byte con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getShortProperty

```
public short getShortProperty(java.lang.String name)
                                                    throws JMSException
```

Devuelve el valor de propiedad de corto con el nombre especificado.

Parámetros:
name - nombre de la propiedad de corto.

Devuelve:
el valor de propiedad de corto con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getIntProperty

```
public int getIntProperty(java.lang.String name)
                                                  throws JMSException
```

Devuelve el valor de propiedad de entero con el nombre especificado.

Parámetros:
name - nombre de la propiedad de entero.

Devuelve:
el valor de propiedad de entero con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getLongProperty

```
public long getLongProperty(java.lang.String name)
                                                   throws JMSException
```

Devuelve el valor de propiedad de largo con el nombre especificado.

Parámetros:
name - nombre de la propiedad de largo.

Devuelve:
el valor de propiedad de largo con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getFloatProperty

```
public float getFloatProperty(java.lang.String name)
                                                    throws JMSException
```

Devuelve el valor de propiedad de flotante con el nombre especificado.

Parámetros:
name - nombre de la propiedad de flotante.

Devuelve:
el valor de propiedad de flotante con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getDoubleProperty

```
public double getDoubleProperty(java.lang.String name)
                                                 throws JMSException
```

Devuelve el valor de propiedad de doble con el nombre especificado.

Parámetros:
name - nombre de la propiedad de doble.

Devuelve:
el valor de propiedad de doble con el nombre especificado.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getStringProperty

```
public java.lang.String getStringProperty (java.lang.String name)
                                                       throws JMSException
```

Devuelve el valor de propiedad de serie.

Parámetros:
name - nombre de la propiedad de serie

Devuelve:
el valor de propiedad de serie con el nombre especificado. Si no existe ninguna propiedad con este nombre, se devuelve un valor nulo.

Emite:

- JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.
- MessageFormatException - si esta conversión de tipos no es válida.

getObjectProperty

```
public java.lang.Object getObjectProperty (java.lang.String name)
                                                       throws JMSException
```

Devuelve el valor de propiedad de objeto Java™ con el nombre especificado.

Parámetros:
name - nombre de la propiedad de objeto Java™.

Devuelve:
el valor de propiedad de objeto Java™ con el nombre especificado, en formato de objeto (por ejemplo, si se ha establecido como int, se devuelve un entero). Si no existe ninguna propiedad con este nombre, se devuelve un valor nulo.

Emite: JMSException - si JMS no puede obtener el valor de la propiedad debido a un error interno de JMS.

getPropertyNames

```
public java.util.Enumeration getPropertyNames()
                                              throws JMSException
```

Devuelve una enumeración de todos los nombres de propiedades.

Devuelve:
una enumeración de todos los nombres de los valores de propiedad.

Emite: JMSException - si JMS no puede obtener los nombres de propiedades debido a un error interno de JMS.

setBooleanProperty

```
public void setBooleanProperty(java.lang.String name,
                               boolean value) throws JMSException
```

Establece un valor de propiedad de booleano con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de booleano.
- value - valor de propiedad de booleano que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setByteProperty

```
public void setByteProperty(java.lang.String name,
                            byte value) throws JMSException
```

Establece un valor de propiedad de byte con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de byte.
- value - valor de propiedad de byte que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setShortProperty

```
public void setShortProperty(java.lang.String name,
                             short value) throws JMSException
```

Establece un valor de propiedad de corto con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de corto.
- value - valor de propiedad de corto que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setIntProperty

```
public void setIntProperty(java.lang.String name,
                           int value) throws JMSException
```

Establece un valor de propiedad de entero con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de entero.
- value - valor de propiedad de entero que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setLongProperty

```
public void setLongProperty(java.lang.String name,
                            long value) throws JMSException
```

Establece un valor de propiedad de largo con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de largo.
- value - valor de propiedad de largo que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setFloatProperty

```
public void setFloatProperty(java.lang.String name,
                             float value) throws JMSException
```

Establece un valor de propiedad de flotante con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de flotante.
- value - valor de la propiedad de flotante que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setDoubleProperty

```
public void setDoubleProperty(java.lang.String name,
                              double value) throws JMSException
```

Establece un valor de propiedad de doble con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de doble.
- value - valor de propiedad de doble que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setStringProperty

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value) throws JMSException
```

Establece un valor de propiedad de serie con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de serie.
- value - valor de propiedad de serie que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

setObjectProperty

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value) throws JMSException
```

Establece un valor de propiedad con el nombre especificado en el mensaje.

Parámetros:

- name - nombre de la propiedad de objeto Java™.

- value - valor de propiedad de objeto Java™ que se va a establecer en el mensaje.

Emite:

- JMSException - si JMS no puede establecer la propiedad debido a un error interno de JMS.
- MessageFormatException - si el objeto no es válido.
- MessageNotWriteableException - si las propiedades son de sólo lectura.

acknowledge

```
public void acknowledge() throws JMSException
```

Reconoce éste y todos los mensajes que ha recibido la sesión anteriormente.

Emite: JMSException - si JMS no puede efectuar el reconocimiento debido a un error interno de JMS.

clearBody

```
public void clearBody() throws JMSException
```

Borra el cuerpo del mensaje. Todas las demás partesdel mensaje permanecen intactas.

Emite: JMSException - si JMS no puede hacerlo debido a un error interno de JMS.

MessageConsumer

interfaz pública **MessageConsumer**
Subinterfaces: **QueueReceiver** y **TopicSubscriber**

Clase MQSeries: **MQMessageConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

Interfaz superior para todos los consumidores de mensajes. Un cliente utiliza un consumidor de mensajes para recibir mensajes de un destino.

Métodos

getMessageSelector

```
public java.lang.String getMessageSelector()
                                  throws JMSException
```

Obtiene la expresión del selector de mensajes de este consumidor de mensajes.

Devuelve:
el selector de mensajes de este consumidor de mensajes.

Emite: JMSException - si JMS no puede obtener el selector de mensajes debido a un error de JMS.

getMessageListener

```
public MessageListener getMessageListener()
                                       throws JMSException
```

Obtiene el MessageListener del consumidor de mensajes.

Devuelve:
el escucha para el consumidor de mensajes, o nulo si no se ha establecido ningún escucha.

Emite: JMSException - si JMS no puede obtener el escucha de mensajes debido a un error de JMS.

Vea también:
setMessageListener

setMessageListener

```
public void setMessageListener(MessageListener listener)
                                              throws JMSException
```

Establece el MessageListener del consumidor de mensajes.

Parámetros:
messageListener - los mensajes se entregan a este escucha.

Emite: JMSException - si JMS no puede establecer el escucha de mensajes debido a un error de JMS.

Vea también:
getMessageListener

receive

```
public Message receive() throws JMSException
```

Recibe el mensaje siguiente que se ha creado para este consumidor de mensajes.

Devuelve:
el mensaje siguiente que se ha creado para este consumidor de mensajes.

Emite: JMSException - si JMS no puede recibir el mensaje siguiente debido a un error.

receive

```
public Message receive(long timeOut) throws JMSException
```

Recibe el mensaje siguiente que llega en el intervalo del tiempo de espera especificado. Un valor de tiempo de espera de cero hace que la invocación espere de modo indefinido hasta que llegue un mensaje.

Parámetros:
timeout - valor de tiempo de espera (en milisegundos).

Devuelve:
el mensaje siguiente que se ha creado para este consumidor de mensajes, o nulo si no hay ninguno disponible.

Emite: JMSException - si JMS no puede recibir el mensaje siguiente debido a un error.

receiveNoWait

```
public Message receiveNoWait() throws JMSException
```

Recibe el mensaje siguiente si hay alguno inmediatamente disponible.

Devuelve:
el mensaje siguiente que se ha creado para este consumidor de mensajes, o nulo si no hay ninguno disponible.

Emite: JMSException - si JMS no puede recibir el mensaje siguiente debido a un error.

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un MessageConsumer, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Esta invocación se bloquea hasta que finaliza un escucha de mensajes o una recepción en curso.

Emite: JMSException - si JMS no puede cerrar el consumidor debido a un error.

MessageListener

interfaz pública **MessageListener**

MessageListener se utiliza para recibir los mensajes entregados asíncronamente.

Métodos

onMessage

```
public void onMessage(Message message)
```

Pasa un mensaje al escucha.

Parámetros:
message - mensaje que se pasa al escucha.

Vea también
Session.setMessageListener

MessageProducer

interfaz pública **MessageProducer**
Subinterfaces: **QueueSender** y **TopicPublisher**

Clase MQSeries: **MQMessageProducer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

Un cliente utiliza un productor de mensajes para enviar mensajes a un destino.

Constructores de MQSeries

MQMessageProducer

```
public MQMessageProducer()
```

Métodos

setDisableMessageID

```
public void setDisableMessageID(boolean value)
                                          throws JMSException
```

Establece si se han inhabilitado los ID de mensaje.

Por omisión, los ID de mensaje están habilitados.

Nota: Este método se ignora en la implementación del Servicio de mensajes de MQSeries Classes for Java.

Parámetros:
value - indica si se han inhabilitado los ID de mensaje.

Emite: JMSException - si JMS no puede establecer el ID de mensaje inhabilitado debido a un error interno.

getDisableMessageID

```
public boolean getDisableMessageID() throws JMSException
```

Obtiene una indicación de si se han inhabilitado los ID de mensaje.

Devuelve:
una indicación de si se han inhabilitado los ID de mensaje.

Emite: JMSException - si JMS no puede obtener el ID de mensaje inhabilitado debido a un error interno.

setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value)
                                                 throws JMSException
```

Establece si se han inhabilitado las indicaciones de hora de los mensajes.

Por omisión, las indicaciones de la hora de los mensajes están habilitadas.

Nota: Este método se ignora en la implementación del Servicio de mensajes de MQSeries Classes for Java.

Parámetros:
value - indica si se han inhabilitado las indicaciones de la hora de los mensajes.

Emite: JMSException - si JMS no puede establecer la indicación de la hora de mensaje inhabilitada debido a un error interno.

getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp()
                                        throws JMSException
```

Obtiene una indicación de si han inhabilitado las indicaciones de la hora de los mensajes.

Devuelve:
una indicación de si se han inhabilitado los ID de mensaje.

Emite: JMSException - si JMS no puede obtener la indicación de la hora de mensaje inhabilitada debido a un error interno.

setDeliveryMode

```
public void setDeliveryMode(int deliveryMode)
                                         throws JMSException
```

Establece la modalidad de entrega por omisión del productor.

Por omisión, la modalidad de entrega se establece en PERSISTENT.

Parámetros:
deliveryMode - modalidad de entrega de mensaje para este productor de mensajes.

Emite: JMSException - si JMS no puede establecer la modalidad de entrega debido a un error interno.

Vea también:
getDeliveryMode

getDeliveryMode

```
public int getDeliveryMode() throws JMSException
```

Obtiene la modalidad de modalidad de entrega por omisión del productor.

Devuelve:
la modalidad de entrega de mensaje para este productor de mensajes.

Emite: JMSException - si JMS no puede obtener la modalidad de entrega debido a un error interno.

Vea también:
setDeliveryMode

setPriority

```
public void setPriority(int priority) throws JMSException
```

Establece la prioridad por omisión del productor.

Por omisión, la prioridad se establece en 4.

Parámetros:
priority - prioridad de mensaje para este productor de mensajes.

Emite: JMSException - si JMS no puede establecer la prioridad debido a un error interno.

Vea también:
getPriority

getPriority

```
public int getPriority() throws JMSException
```

Obtiene la prioridad por omisión del productor.

Devuelve:
la prioridad de mensaje para este productor de mensajes.

Emite: JMSException - si JMS no puede obtener la prioridad debido a un error interno.

Vea también:
setPriority

setTimeToLive

```
public void setTimeToLive(long timeToLive)
                                        throws JMSException
```

Establece el tiempo, en milisegundos, a partir del momento del envío, que el sistema de mensajes debe retener un mensaje producido.

Por omisión, el tiempo de vida se establece en cero.

Parámetros:
timeToLive - tiempo de vida del mensaje en milisegundos. Cero es ilimitado.

Emite: JMSException - si JMS no puede establecer el tiempo de vida debido a un error interno.

Vea también:
getTimeToLive

getTimeToLive

```
public long getTimeToLive() throws JMSException
```

Obtiene el tiempo, en milisegundos, a partir del momento del envío, que el sistema de mensajes debe retener un mensaje producido.

Devuelve:
el tiempo de vida del mensaje en milisegundos. Cero es ilimitado.

Emite: JMSException - si JMS no puede obtener el tiempo de vida debido a un error interno.

Vea también:
setTimeToLive

MessageProducer

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un MessageProducer, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Emite: JMSException - si JMS no puede cerrar el productor debido a un error.

MQQueueEnumeration *

clase pública **MQQueueEnumeration**
expande **Object**
implementa **Enumeration**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

Enumeración de los mensajes de una cola. Esta clase no está definida en la especificación JMS, se crea invocando el método getEnumeration de MQQueueBrowser. La clase contiene una instancia de MQQueue base para retener el cursor de examinar. La cola se cierra después de que el cursor se ha retirado del final de la cola.

No existe ningún procedimiento para restablecer una instancia de esta clase, puesto que actúa como un mecanismo de 'una sola acción'.

Vea también: **MQQueueBrowser**

Métodos

hasMoreElements

```
public boolean hasMoreElements()
```

Indica si se puede devolver otro mensaje.

nextElement

```
public Object nextElement() throws NoSuchElementException
```

Devuelve el mensaje actual.

Si `hasMoreElements()` devuelve 'true', `nextElement()` siempre devuelve un mensaje. El mensaje devuelto puede cumplir su fecha de caducidad entre las invocaciones a `hasMoreElements()` y `nextElement`.

ObjectMessage

interfaz pública **ObjectMessage**
expande **Message**

Clase MQSeries: **JMSObjectMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

ObjectMessage se utiliza para enviar un mensaje que contiene un objeto Java™ serializable. Hereda de Message y añade un cuerpo que contiene una única referencia Java™. Sólo se pueden utilizar objetos Java™ serializables.

Vea también: **BytesMessage**, **MapMessage**, **Message**, **StreamMessage** y **TextMessage**

Métodos

setObject

```
public void setObject(java.io.Serializable object)
                                        throws JMSException
```

Establece el objeto serializable que contiene los datos del mensaje. ObjectMessage contiene una instantánea del objeto al invocar a setObject(). Los posteriores modificaciones que se realizan en el objeto no afectan al cuerpo de ObjectMessage.

Parámetros:
object - los datos del mensaje.

Emite:

- JMSException - si JMS no puede establecer el objeto debido a un error interno de JMS.
- MessageFormatException - si la serialización de objetos no se ha ejecutado correctamente.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

getObject

```
public java.io.Serializable getObject()
                               throws JMSException
```

Obtiene el objeto serializable que contiene los datos del mensaje. El valor por omisión es nulo.

Devuelve:
el objeto serializable que contiene los datos del mensaje.

Emite:

- JMSException - si JMS no puede obtener el objeto debido a un error interno de JMS.
- MessageFormatException - si la deserialización de objetos no se ha ejecutado correctamente.

Queue

interfaz pública **Queue**
expande **Destination**
Subinterfaces: **TemporaryQueue**

Clase MQSeries: **MQQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

Un objeto Queue encapsula el nombre de cola específico de un suministrador. De este modo, un cliente especifica la identidad de una cola para los métodos JMS.

Constructores de MQSeries

MQQueue *

```
public MQQueue()
```

Constructor por omisión para que lo utilice la herramienta de administración.

MQQueue *

```
public MQQueue(String URIqueue)
```

Crea una nueva instancia de MQQueue. La serie toma un formato URI, tal como se describe en la página 187.

MQQueue *

```
public MQQueue(String queueManagerName,
               String queueName)
```

Métodos

getQueueName

```
public java.lang.String getQueueName()
                                    throws JMSException
```

Obtiene el nombre de esta cola.

Los clientes que dependen del nombre no son portátiles.

Devuelve:
el nombre de cola

Emite: JMSException - si la implementación de JMS para Queue no puede devolver el nombre de cola debido a un error interno.

toString

```
public java.lang.String toString()
```

Devuelve una versión impresa del nombre de cola.

Devuelve:
los valores de identidad específicos del suministrador para esta cola.

Altera temporalmente:
toString en la clase java.lang.Object

getReference *

```
public Reference getReference() throws NamingException
```

Crea una referencia para esta cola.

Devuelve:
una referencia para este objeto.

Emite: NamingException

setBaseQueueName *

```
public void setBaseQueueName(String x) throws JMSException
```

Establece el valor del nombre de cola MQSeries.

Nota: Sólo la herramienta de administración debe utilizar este método. No realiza ningún intento de decodificación de las series con formato cola:gestorcolas:cola.

getBaseQueueName *

```
public String getBaseQueueName()
```

Devuelve:
el valor del nombre de cola MQSeries.

setBaseQueueManagerName *

```
public void setBaseQueueManagerName(String x) throws JMSException
```

Establece el valor del nombre del gestor de colas MQSeries.

Nota: Sólo la herramienta de administración debe utilizar este método.

getBaseQueueManagerName *

```
public String getBaseQueueManagerName()
```

Devuelve:
el valor del nombre del gestor de colas MQSeries.

QueueBrowser

interfaz pública **QueueBrowser**

Clase MQSeries: **MQQueueBrowser**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

Un cliente utiliza QueueBrowser para ver los mensajes sin quitarlos de la cola.

Nota: La clase MQSeries **MQQueueEnumeration** se utiliza para mantener el cursor de examinar.

Vea también: **QueueReceiver**

Métodos

getQueue

```
public Queue getQueue() throws JMSException
```

Obtiene la cola asociada a este examinador de cola.

Devuelve:
la cola.

Emite: JMSException - si JMS no puede obtener la cola asociada a este examinador debido a un error de JMS.

getMessageSelector

```
public java.lang.String getMessageSelector() throws JMSException
```

Obtiene la expresión del selector de mensajes del examinador de cola.

Devuelve:
el selector de mensajes de este examinador de cola.

Emite: JMSException - si JMS no puede obtener el selector de mensajes para este examinador debido a un error de JMS.

getEnumeration

```
public java.util.Enumeration getEnumeration() throws JMSException
```

Obtiene una enumeración para examinar los mensajes de la cola actual en el orden en que se han recibido.

Devuelve:
una enumeración para examinar los mensajes.

Emite: JMSException - si JMS no puede obtener la enumeración para este examinador debido a un error de JMS.

Nota: Si se ha creado el examinador para una cola que no existe, no se detecta hasta que se efectúa la primera invocación a getEnumeration.

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un QueueBrowser, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Emite: JMSException - si JMS no puede cerrar este examinador debido a un error de JMS.

QueueConnection

interfaz pública **QueueConnection**
expande **Connection**
Subinterfaces: **XAQueueConnection**

Clase MQSeries: **MQQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
```

QueueConnection es una conexión activa con un suministrador punto a punto de JMS. Un cliente utiliza QueueConnection para crear una o más QueueSessions para producir y consumir mensajes.

Vea también: **Connection**, **QueueConnectionFactory** y **XAQueueConnection**

Métodos

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

Crea una QueueSession.

Parámetros:

- transacted - si es verdadero, la sesión se ha ejecutado.
- acknowledgeMode - indica si el consumidor o el cliente van a reconocer los mensajes que reciben. Los valores posibles son:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Este parámetro se ignora si la sesión es transaccional.

Devuelve:
Una sesión de colas recién creada.

Emite: JMSException - si la conexión de JMS no puede crear una sesión debido a un error interno, o si no hay el soporte adecuado para la transacción específica y la modalidad de reconocimiento.

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                                    (Queue queue,
                                     java.lang.String messageSelector,
                                     ServerSessionPool sessionPool,
                                     int maxMessages)
                                                throws JMSException
```

Crea un consumidor de conexiones para esta conexión. Éste es un recurso especializado que los clientes JMS habituales no utilizan.

Parámetros:

- queue - cola a la que se va a acceder.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes.
- sessionPool - agrupación de sesiones de servidor que se va a asociar a este consumidor de conexiones.
- maxMessages - número máximo de mensajes que se pueden asignar a una sesión de servidor al mismo tiempo.

Devuelve:

el consumidor de conexiones.

Emite:

- JMSException - si la conexión de JMS no puede crear un consumidor de conexiones debido a un error interno, o si los argumentos para sessionPool y messageSelector no son válidos.
- InvalidSelectorException - si el selector de mensajes no es válido.

Vea también:

ConnectionConsumer

close *

```
public void close() throws JMSException
```

Altera temporalmente:

close en la clase MQConnection.

QueueConnectionFactory

interfaz pública **QueueConnectionFactory**
expande **ConnectionFactory**
Subinterfaces: **XAQueueConnectionFactory**

Clase MQSeries: **MQQueueConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
```

Un cliente utiliza QueueConnectionFactory para crear QueueConnections con un suministrador punto a punto de JMS.

Vea también: **ConnectionFactory** y**XAQueueConnectionFactory**

Constructor de MQSeries

MQQueueConnectionFactory

```
public MQQueueConnectionFactory()
```

Métodos

createQueueConnection

```
public QueueConnection createQueueConnection()
                                          throws JMSException
```

Crea una conexión de colas con una identidad de usuario por omisión. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Devuelve:
una conexión de colas recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear la conexión de colas debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

createQueueConnection

```
public QueueConnection createQueueConnection
                                  (java.lang.String userName,
                                   java.lang.String password)
                                                  throws JMSException
```

Crea una conexión de colas con la identidad de usuario especificada.

Nota: Este método sólo se puede utilizar con el tipo de transporte JMSC.MQJMS_TP_CLIENT_MQ_TCPIP (consulte el apartado ConnectionFactory). La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Parámetros:

- userName - nombre de usuario del canal de llamada.
- password - contraseña del canal de llamada.

Devuelve:

una conexión de colas recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear la conexión de colas debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

setTemporaryModel *

```
public void setTemporaryModel(String x) throws JMSException
```

getTemporaryModel *

```
public String getTemporaryModel()
```

getReference *

```
public Reference getReference() throws NamingException
```

Crea una referencia para esta fábrica de conexión de colas.

Devuelve:

una referencia para este objeto.

Emite: NamingException.

setMessage Retention*

```
public void setMessageRetention(int x) throws JMSException
```

Establece el método para el atributo messageRetention.

Parámetros:

Los valores válidos son:

- JMSC.MQJMS_MRET_YES - los mensajes no deseados permanecen en la cola de entrada.
- JMSC.MQJMS_MRET_NO - se tratan los mensajes no deseados conforme a sus opciones de disposición.

getMessage Retention*

```
public void getMessageRetention()
```

Obtiene el método para el atributo messageRetention.

Devuelve:

- JMSC.MQJMS_MRET_YES - los mensajes no deseados permanecen en la cola de entrada.
- JMSC.MQJMS_MRET_NO - se tratan los mensajes no deseados conforme a sus opciones de disposición.

QueueReceiver

interfaz pública **QueueReceiver**
expande **MessageConsumer**

Clase MQSeries: **MQQueueReceiver**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

Un cliente utiliza QueueReceiver para recibir mensajes que se han entregado en una cola.

Vea también: **MessageConsumer**

Esta clase hereda los métodos siguientes de **MQMessageConsumer**.
- receive
- receiveNoWait
- close
- getMessageListener
- setMessageListener

Métodos

getQueue

```
public Queue getQueue() throws JMSException
```

Obtiene la cola asociada a este receptor de colas.

Devuelve:
la cola.

Emite: JMSException - si JMS no puede obtener la cola de este receptor de colas debido a un error interno.

QueueRequestor

clase pública **QueueRequestor**
expande **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.QueueRequestor
```

JMS proporciona esta clase de ayuda de QueueRequestor para simplificar la emisión de peticiones de servicio. Se proporciona al constructor QueueRequestor una cola de destino y una QueueSession no transaccional. Crea una TemporaryQueue para las respuestas y proporciona un método request() que envía el mensaje de solicitud y espera su respuesta. Los usuarios pueden crear versiones más complejas si lo desean.

Vea también: **TopicRequestor**

Constructores

QueueRequestor

```
public QueueRequestor(QueueSession session,
                      Queue queue)
                              throws JMSException
```

Esta implementación da por supuesto que el parámetro de sesión es no transaccional, y ya sea AUTO_ACKNOWLEDGE o bien DUPS_OK_ACKNOWLEDGE.

Parámetros:

- session - sesión de colas a la que pertenece la cola.
- queue - cola en la que se va a realizar la invocación de petición/respuesta.

Emite: JMSException - si se produce un error de JMS.

Métodos

request

```
public Message request(Message message)
                                  throws JMSException
```

Envía una petición y espera una respuesta. La cola temporal se utiliza para replyTo y sólo se espera una respuesta por petición.

Parámetros:

message - mensaje que se va a enviar.

Devuelve:

el mensaje de respuesta.

Emite: JMSException - si se produce un error de JMS.

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un QueueRequestor, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Nota: Este método cierra el objeto Session que se ha pasado al constructor QueueRequestor.

Emite: JMSException - si se produce un error de JMS.

QueueSender

interfaz pública **QueueSender**
expande **MessageProducer**

Clase MQSeries: **MQQueueSender**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

Un cliente utiliza QueueSender para enviar mensajes a una cola.

Normalmente, un QueueSender se asocia a una cola determinada. Sin embargo, se puede crear un QueueSender no identificado que no esté asociado a ninguna cola específica.

Vea también: **MessageProducer**

Métodos

getQueue

```
public Queue getQueue() throws JMSException
```

Obtiene la cola asociada a este emisor de colas.

Devuelve:
la cola.

Emite: JMSException - si JMS no puede obtener la cola para este emisor de colas debido a un error interno.

send

```
public void send(Message message) throws JMSException
```

Envía un mensaje a la cola. Utiliza la prioridad, el tiempo de vida y la modalidad de entrega por omisión de QueueSender.

Parámetros:
message - mensaje que se va a enviar.

Emite:

- JMSException - si JMS no puede enviar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con un emisor de colas con una cola no válida.

send

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

Envía un mensaje a la cola especificando la modalidad de entrega, la prioridad y el tiempo de vida.

Parámetros:

- message - mensaje que se va a enviar.
- deliveryMode - modalidad de entrega que se va a utilizar.
- priority - prioridad de este mensaje.
- timeToLive - tiempo de vida del mensaje (en milisegundos).

Emite:

- JMSException - si JMS no puede enviar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con un emisor de colas con una cola no válida.

send

```
public void send(Queue queue,
                 Message message) throws JMSException
```

Envía un mensaje a la cola especificada con la prioridad, el tiempo de vida y la modalidad de entrega por omisión de QueueSender.

Nota: Este método sólo se puede utilizar con QueueSenders no identificados.

Parámetros:

- queue - cola a la que se debe enviar este mensaje.
- message - mensaje que se va a enviar.

Emite:

- JMSException - si JMS no puede enviar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con una cola no válida.

send

```
public void send(Queue queue,
                 Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

Envía un mensaje a la cola especificada con la modalidad de entrega, la prioridad y el tiempo de vida.

Nota: Este método sólo se puede utilizar con QueueSenders no identificados.

Parámetros:

- queue - cola a la que se debe enviar este mensaje.
- message - mensaje que se va a enviar.
- deliveryMode - modalidad de entrega que se va a utilizar.
- priority - prioridad de este mensaje.

- timeToLive - tiempo de vida del mensaje (en milisegundos).

Emite:

- JMSException - si JMS no puede enviar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con una cola no válida.

close *

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos de JVM en nombre de un QueueSender, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Emite: JMSException si JMS no puede cerrar el productor debido a algún error.

Altera temporalmente:
close en la clase MQMessageProducer.

QueueSession

interfaz pública **QueueSession**
expande **Session**

Clase MQSeries: **MQQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

QueueSession proporciona métodos para crear QueueReceivers, QueueSenders, QueueBrowsers y TemporaryQueues.

Vea también: **Session**

Los métodos siguientes Se heredan de **MQSession**:
- close
- commit
- rollback
- recover

Métodos

createQueue

```
public Queue createQueue(java.lang.String queueName)
                                              throws JMSException
```

Crea una cola con un nombre de cola determinado, lo que permite que se pueda crear una cola con un nombre de suministrador específico. La serie toma un formato URI, tal como se describe la página 187.

Nota: Los clientes que dependen de esta posibilidad no son portátiles.

Parámetros:
queueName - nombre de esta cola.

Devuelve:
una cola con el nombre especificado.

Emite: JMSException - si una sesión no puede crear una cola debido a un error de JMS.

createReceiver

```
public QueueReceiver createReceiver(Queue queue)
                                              throws JMSException
```

Crea un QueueReceiver para recibir mensajes de la cola especificada.

Parámetros:
queue - cola a la que se va a acceder.

Emite:
- JMSException - si una sesión no puede crear un receptor debido a un error de JMS.
- InvalidDestinationException - si la cola especificada no es válida.

QueueSession

createReceiver

```
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
                                    throws JMSException
```

Crea un QueueReceiver para recibir mensajes de la cola especificada.

Parámetros:

- queue - cola a la que se va a acceder.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes.

Emite:

- JMSException - si una sesión no puede crear un receptor debido a un error de JMS.
- InvalidDestinationException - si la cola especificada no es válida.
- InvalidSelectorException - si el selector de mensajes no es válido.

createSender

```
public QueueSender createSender(Queue queue)
                                        throws JMSException
```

Crea un QueueSender para enviar mensajes a la cola especificada.

Parámetros:

queue - cola a la que se va a acceder, o nulo si se va a tratar de un productor no identificado.

Emite:

- JMSException - si una sesión no puede crear un emisor debido a un error de JMS.
- InvalidDestinationException - si la cola especificada no es válida.

createBrowser

```
public QueueBrowser createBrowser(Queue queue)
                                      throws JMSException
```

Crea un QueueBrowser para mirar los mensajes de la cola especificada.

Parámetros:

queue - cola a la que se va a acceder.

Emite:

- JMSException - si una sesión no puede crear un examinador debido a un error de JMS.
- InvalidDestinationException - si la cola especificada no es válida.

createBrowser

```
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
                                  throws JMSException
```

Crea un QueueBrowser para mirar los mensajes de la cola especificada.

Parámetros:

- queue - cola a la que se va a acceder.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes.

Emite:

- JMSException - si una sesión no puede crear un examinador debido a un error de JMS.
- InvalidDestinationException - si la cola especificada no es válida.
- InvalidSelectorException - si el selector de mensajes no es válido.

createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                        throws JMSException
```

Crea una cola temporal. Su duración es la de QueueConnection, a menos que se haya suprimido antes.

Devuelve:
una cola temporal.

Emite: JMSException - si una sesión no puede crear una cola temporal debido a un error de JMS.

Session

interfaz pública **Session**
expande **java.lang.Runnable**
Subinterfaces: **QueueSession**, **TopicSession**, **XAQueueSession**, **XASession** y **XATopicSession**

Clase MQSeries: **MQSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

Una sesión (Session) de JMS es un contexto de una sola hebra para producir y consumir mensajes.

Vea también: **QueueSession**, **TopicSession**, **XAQueueSession**, **XASession** y **XATopicSession**

Campos

AUTO_ACKNOWLEDGE

```
public static final int AUTO_ACKNOWLEDGE
```

Con esta modalidad de reconocimiento, la sesión reconoce automáticamente un mensaje cuando se devuelve de modo satisfactorio de una invocación de recepción o cuando lo devuelve correctamente el escucha de mensajes al que ha invocado para procesarlo.

CLIENT_ACKNOWLEDGE

```
public static final int CLIENT_ACKNOWLEDGE
```

Con esta modalidad de reconocimiento, el cliente reconoce un mensaje invocando un método de reconocimiento del mensaje.

DUPS_OK_ACKNOWLEDGE

```
public static final int DUPS_OK_ACKNOWLEDGE
```

Esta modalidad de reconocimiento indica a la sesión que reconozca la entrega de mensajes de modo diferido.

Métodos

createBytesMessage

```
public BytesMessage createBytesMessage()
                                throws JMSException
```

Crea un BytesMessage. BytesMessage se utiliza para enviar un mensaje que contiene una corriente bytes no interpretados.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

createMapMessage

```
public MapMessage createMapMessage() throws JMSException
```

Crea un MapMessage. MapMessage se utiliza para enviar un conjunto de pares de nombre-valor autodefinidos, donde los nombres son Series de caracteres (Strings) y los valores son tipos de primitivos de Java™.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

createMessage

```
public Message createMessage() throws JMSException
```

Crea un mensaje. La interfaz Message es la interfaz raíz de todos los mensajes JMS. Mantiene toda la información de cabecera de mensaje estándar. Se puede enviar cuando es suficiente con un mensaje que contenga sólo la información de cabecera.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

createObjectMessage

```
public ObjectMessage createObjectMessage()
                             throws JMSException
```

Crea un ObjectMessage. ObjectMessage se utiliza para enviar un mensaje que contiene un objeto Java™ serializable.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

createObjectMessage

```
public ObjectMessage createObjectMessage
                            (java.io.Serializable object)
                             throws JMSException
```

Crea un ObjectMessage inicializado. ObjectMessage se utiliza para enviar un mensaje que contiene un objeto Java™ serializable.

Parámetros:
object - objeto que se va a utilizar para inicializar este mensaje.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

createStreamMessage

```
public StreamMessage createStreamMessage()
                              throws JMSException
```

Crea un StreamMessage. StreamMessage se utiliza para enviar una corriente de datos autodefinida de primitivos de Java™.

Emite: JMSException si JMS no puede crear el mensaje debido a un error interno.

Session

createTextMessage

```
public TextMessage createTextMessage() throws JMSException
```

Crea un TextMessage. TextMessage se utiliza para enviar un mensaje que contenga una serie.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

createTextMessage

```
public TextMessage createTextMessage
                      (java.lang.String string)
                      throws JMSException
```

Crea un TextMessage inicializado. TextMessage se utiliza para enviar un mensaje que contenga una serie.

Parámetros:

string - serie de caracteres que se va a utilizar para inicializar este mensaje.

Emite: JMSException - si JMS no puede crear el mensaje debido a un error interno.

getTransacted

```
public boolean getTransacted() throws JMSException
```

¿Está la sesión en modalidad transaccional?

Devuelve:

verdadero (true) si la sesión está en modalidad transaccional.

Emite: JMSException - si JMS no puede devolver la modalidad de transacción debido a un error interno del suministrador de JMS.

commit

```
public void commit() throws JMSException
```

Confirma todos los mensajes realizados en esta transacción y libera todos los bloqueos mantenidos actualmente.

Emite:

- JMSException - si la implementación de JMS no puede confirmar la transacción debido a un error interno.
- TransactionRolledBackException - si se restituye la transacción debido a un error interno durante la confirmación.

rollback

```
public void rollback() throws JMSException
```

Restituye todos los mensajes realizados en esta transacción y libera todos los bloqueos mantenidos actualmente.

Emite: JMSException - si la implementación de JMS no puede restituir la transacción debido a un error interno.

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de una sesión, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Cuando se cierra una sesión transaccional se restituyen todas las transacciones en curso. Al cerrar una sesión, se cierran automáticamente sus productores y consumidores de mensajes, por lo que no es necesario cerrarlos de forma individual.

Emite: JMSException - si la implementación de JMS no puede cerrar una sesión debido a un error interno.

recover

```
public void recover() throws JMSException
```

Detiene la entrega de mensajes en esta sesión y reinicia el envío de mensajes con el mensaje no reconocido más antiguo.

Emite: JMSException - si la implementación de JMS no puede detener la entrega y reiniciar el envío de mensajes debido a un error interno.

getMessageListener

```
public MessageListener getMessageListener()
                                  throws JMSException
```

Devuelve el escucha de mensajes distintivo de la sesión.

Devuelve:
el escucha de mensajes asociado a esta sesión.

Emite: JMSException - si JMS no puede obtener el escucha de mensajes debido a un error interno en el suministrador de JMS.

Vea también:
setMessageListener

setMessageListener

```
public void setMessageListener(MessageListener listener)
                                             throws JMSException
```

Establece el escucha de mensajes distintivo de la sesión. Cuando se establece, no se puede utilizar en la sesión ningún otro formato de recepción de mensajes. Sin embargo, se sigue ofreciendo soporte para todos los formatos de envío de mensajes.

Éste es un recurso especializado que no utilizan los clientes JMS habituales.

Parámetros:
listener - escucha de mensajes que se va a asociar a esta sesión.

Emite: JMSException - si JMS no puede establecer el escucha de mensajes debido a un error interno en el suministrador de JMS.

Vea también:
getMessageListener, ServerSessionPool, ServerSession

run

```
public void run()
```

Este método es para que lo utilicen sólo los servidores de aplicaciones.

Especificado por:
`run` en la interfaz java.lang.Runnable

Vea también:
ServerSession

StreamMessage

interfaz pública **StreamMessage**
expande **Message**

Clase MQSeries: **JMSStreamMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

StreamMessage se utiliza para enviar una corriente de datos de primitivos de Java™.

Vea también: **BytesMessage**, **MapMessage**, **Message**, **ObjectMessage** y **TextMessage**

Métodos

readBoolean

```
public boolean readBoolean() throws JMSException
```

Lee un valor booleano del mensaje de corriente de datos.

Devuelve:
el valor booleano leído.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException - si esta conversión de tipos no es válida.
- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.

readByte

```
public byte readByte() throws JMSException
```

Lee un valor de byte del mensaje de corriente de datos.

Devuelve:
el byte siguiente del mensaje de corriente de datos como un byte de 8 bits.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException - si esta conversión de tipos no es válida.

- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.

readShort

```
public short readShort() throws JMSException
```

Lee un número de 16 bits del mensaje de corriente de datos.

Devuelve:
un número de 16 bits del mensaje de corriente de datos.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException - si esta conversión de tipos no es válida.
- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.

readChar

```
public char readChar() throws JMSException
```

Lee un valor de carácter Unicode del mensaje de corriente de datos.

Devuelve:
un carácter Unicode del mensaje de corriente de datos.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException si el tipo de conversión no es válido.
- MessageNotReadableException si el mensaje está en modalidad de sólo escritura.

readInt

```
public int readInt() throws JMSException
```

Lee un entero de 32 bits del mensaje de corriente de datos.

Devuelve:
un valor de entero de 32 bits del mensaje de corriente de datos, interpretado como int.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException si el tipo de conversión no es válido.
- MessageNotReadableException si el mensaje está en modalidad de sólo escritura.

readLong

```
public long readLong() throws JMSException
```

Lee un entero de 64 bits del mensaje de corriente de datos.

Devuelve:
un valor de entero de 64 bits del mensaje de corriente de datos, interpretado como un número largo.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - en caso de una corriente de datos de fin de mensaje
- MessageFormatException si el tipo de conversión no es válido.
- MessageNotReadableException si el mensaje está en modalidad de sólo escritura.

readFloat

```
public float readFloat() throws JMSException
```

Lee un valor flotante del mensaje de corriente de datos.

Devuelve:
un valor flotante del mensaje de corriente de datos.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - en caso de una corriente de datos de fin de mensaje
- MessageFormatException si el tipo de conversión no es válido.
- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.

readDouble

```
public double readDouble() throws JMSException
```

Lee un valor doble del mensaje de corriente de datos.

Devuelve:
un valor doble del mensaje de corriente de datos.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException - si esta conversión de tipos no es válida.
- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.

StreamMessage

readString

```
public java.lang.String readString() throws JMSException
```

Lee un valor de serie de caracteres del mensaje de corriente de datos.

Devuelve:
una serie de caracteres Unicode del mensaje de corriente de datos.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException - si esta conversión de tipos no es válida.
- MessageNotReadableException - si el mensaje está en modalidad de sólo lectura

readBytes

```
public int readBytes(byte[] value)
             throws JMSExceptioneam message.
```

Lee un campo de matriz de bytes del mensaje de corriente de datos en el objeto byte[] especificado (el almacenamiento intermedio de lectura). Si el tamaño del almacenamiento intermedio es menor o igual que el tamaño de los datos del campo del mensaje, una aplicación debe llevar a cabo otras invocaciones a este método para recuperar el resto de los datos. Después de realizar la primera invocación de readBytes en el valor de un campo de byte[], para que sea válido para leer el campo siguiente, antes se debe leer el valor completo del campo. Si se intenta leer el campo siguiente antes de hacerlo, se emite una MessageFormatException.

Parámetros:
value - almacenamiento intermedio en el que se leen los datos.

Devuelve:
el número total de bytes leídos en el almacenamiento intermedio, o -1 si no hay más datos debido a que se ha llegado al final del campo de byte.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- MessageFormatException - si esta conversión de tipos no es válida.
- MessageNotReadableException - si el mensaje está en modalidad de sólo escritura.

readObject

```
public java.lang.Object readObject() throws JMSException
```

Lee un objeto Java™ del mensaje de corriente de datos.

Devuelve:
un objeto Java™ del mensaje de corriente de datos en formato de objeto (por ejemplo, si se ha establecido como int, se devuelve un entero).

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageEOFException - si se recibe una corriente de datos de fin de mensaje.
- NotReadableException - si el mensaje está en modalidad de sólo escritura.

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

Escribe un valor booleano en el mensaje de corriente de datos.

Parámetros:
value - valor booleano que se va a escribir.

Emite:

- JMSException - si JMS no puede leer el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeByte

```
public void writeByte(byte value) throws JMSException
```

Escribe un byte en el mensaje de corriente de datos.

Parámetros:
value - valor de byte que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeShort

```
public void writeShort(short value) throws JMSException
```

Escribe un número corto en el mensaje de corriente de datos.

Parámetros:
value - número corto que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeChar

```
public void writeChar(char value) throws JMSException
```

Escribe un valor char en el mensaje de corriente de datos.

Parámetros:
value - valor char que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeInt

```
public void writeInt(int value) throws JMSException
```

Escribe un valor int en el mensaje de corriente de datos.

Parámetros:
value - valor int que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeLong

```
public void writeLong(long value) throws JMSException
```

Escribe un número largo en el mensaje de corriente de datos.

Parámetros:
value - número largo que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeFloat

```
public void writeFloat(float value) throws JMSException
```

Escribe un valor flotante en el mensaje de corriente de datos.

Parámetros:
value - valor flotante que se va escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeDouble

```
public void writeDouble(double value) throws JMSException
```

Escribe un valor doble en el mensaje de corriente de datos.

Parámetros:

value - valor doble que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeString

```
public void writeString(java.lang.String value)
                                          throws JMSException
```

Escribe una serie de caracteres en el mensaje de corriente de datos.

Parámetros:

value - valor de la serie de caracteres que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

Escribe una matriz de bytes en el mensaje de corriente de datos.

Parámetros:

value - matriz de bytes que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeBytes

```
public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException
```

Escribe una parte de una matriz de bytes en el mensaje de corriente de datos.

Parámetros:

- value - valor de la matriz de bytes que se va a escribir.
- offset - desplazamiento inicial en la matriz de bytes.
- length - número de bytes que se van a utilizar.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.

- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

writeObject

```
public void writeObject(java.lang.Object value)
                                        throws JMSException
```

Escribe un objeto Java™ en el mensaje de corriente de datos. Este método sólo funciona para tipos de primitivos de objeto (como, por ejemplo, Integer, Double, Long), series de caracteres (Strings) y matrices de bytes.

Parámetros:

value - objeto Java™ que se va a escribir.

Emite:

- JMSException - si JMS no puede escribir el mensaje debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.
- MessageFormatException - si el objeto no es válido.

reset

```
public void reset() throws JMSException
```

Pone el mensaje en modalidad de sólo lectura y vuelve a situar corriente de datos al principio.

Emite:

- JMSException - si JMS no puede restablecer el mensaje debido a un error interno de JMS.
- MessageFormatException - si el formato del mensaje no es válido.

TemporaryQueue

interfaz pública **TemporaryQueue**
expande **Queue**

Clase MQSeries: **MQTemporaryQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

TemporaryQueue es un objeto de cola exclusivo que se crea para la duración de una QueueConnection.

Métodos

delete

```
public void delete() throws JMSException
```

Suprime esta cola temporal. Si aún hay emisores o receptores que la utilizan, se emite una JMSException.

Emite: JMSException - si la implementación de JMS no puede suprimir una TemporaryQueue debido a un error interno.

TemporaryTopic

interfaz pública **TemporaryTopic**
expande **Topic**

Clase MQSeries: **MQTemporaryTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

TemporaryTopic es un objeto de tema exclusivo que se crea para la duración de una TopicConnection y que sólo pueden consumir los consumidores de dicha conexión.

Constructor de MQSeries

MQTemporaryTopic

```
MQTemporaryTopic() throws JMSException
```

Métodos

delete

```
public void delete() throws JMSException
```

Suprime este tema temporal. Si aún hay publicadores o suscriptores que lo utilizan, se emita una JMSException.

Emite: JMSException - si la implementación de JMS no puede suprimir un TemporaryTopic debido a un error interno.

TextMessage

interfaz pública **TextMessage**
expande **Message**

Clase MQSeries: **JMSTextMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

TextMessage se utiliza para enviar un mensaje que contiene una java.lang.String. Hereda de Message y añade un cuerpo de mensaje de texto.

Vea también: **BytesMessage**, **MapMessage**, **Message**, **ObjectMessage** y **StreamMessage**

Métodos

setText

```
public void setText(java.lang.String string)
                                        throws JMSException
```

Establece la serie de caracteres que contiene los datos de este mensaje.

Parámetros:
string - serie de caracteres que contiene los datos del mensaje.

Emite:

- JMSException - si JMS no puede establecer texto debido a un error interno de JMS.
- MessageNotWriteableException - si el mensaje está en modalidad de sólo lectura.

getText

```
public java.lang.String getText() throws JMSException
```

Obtiene la serie de caracteres que contiene los datos de este mensaje. El valor por omisión es nulo.

Devuelve:
la serie de caracteres que contiene los datos del mensaje.

Emite: JMSException - si JMS no puede obtener el texto debido a un error interno de JMS.

Topic

interfaz pública **Topic**
expande **Destination**
Subinterfaces: **TemporaryTopic**

Clase MQSeries: **MQTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

Un objeto Topic encapsula el nombre de tema específico de un suministrador. De este modo, un cliente especifica la identidad de un tema para los métodos JMS.

Vea también: **Destination**

Constructor de MQSeries

MQTopic

```
public MQTopic()
public MQTopic(string URItopic)
```

Vea **TopicSession.createTopic**.

Métodos

getTopicName

```
public java.lang.String getTopicName() throws JMSException
```

Obtiene el nombre de este tema en formato URI. (En el apartado “Creación de temas durante la ejecución” en la página 197 se describe el formato URI).

Nota: Los clientes que dependen del nombre no son portátiles.

Devuelve:
el nombre del tema.

Emite: JMSException - si la implementación de JMS para Topic no puede devolver el nombre del tema debido a un error interno.

toString

```
public String toString()
```

Devuelve una versión impresa del nombre del tema.

Devuelve:
los valores de la identidad específica del suministrador para este tema.

Altera temporalmente:
toString en la clase Object.

getReference *

```
public Reference getReference()
```

Crea una referencia para este tema.

Devuelve:
una referencia para este objeto.

Emite: NamingException.

setBaseTopicName *

```
public void setBaseTopicName(String x)
```

Establece el método para el nombre del tema MQSeries subyacente.

getBaseTopicName *

```
public String getBaseTopicName()
```

Obtiene el método para el nombre del tema MQSeries subyacente.

setBrokerDurSubQueue *

```
public void setBrokerDurSubQueue(String x) throws JMSException
```

Establece el método para el atributo brokerDurSubQueue subyacente.

Parámetros:
brokerDurSubQueue - nombre de la cola de suscripciones duraderas que se va a utilizar.

getBrokerDurSubQueue *

```
public String getBrokerDurSubQueue()
```

Obtiene el método para el atributo brokerDurSubQueue.

Devuelve:
el nombre de la cola de suscripciones duraderas (brokerDurSubQueue) que se va a utilizar.

setBrokerCCDurSubQueue *

```
public void setBrokerCCDurSubQueue(String x) throws JMSException
```

Establece el método para el atributo brokerCCDurSubQueue.

Parámetros:
brokerCCDurSubQueue - nombre de la cola de suscripciones duraderas que se va a utilizar para ConnectionConsumer.

getBrokerCCDurSubQueue *

```
public String getBrokerCCDurSubQueue()
```

Obtiene el método para el atributo brokerCCDurSubQueue.

Devuelve:
el nombre de la cola de suscripciones duraderas (brokerCCDurSubQueue) que se va a utilizar para ConnectionConsumer.

TopicConnection

interfaz pública **TopicConnection**
expande **Connection**
Subinterfaces: **XATopicConnection**

Clase MQSeries: **MQTopicConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
```

TopicConnection es una conexión activa con un suministrador de JMS Publish/Subscribe.

Vea también: **Connection**, **TopicConnectionFactory** y **XATopicConnection**

Métodos

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                       throws JMSException
```

Crea una TopicSession.

Parámetros:

- transacted - si es verdadero, la sesión es transaccional.
- acknowledgeMode - uno de los siguientes:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Indica si el consumidor o el cliente van a reconocer los mensajes que reciben. Este parámetro se ignora si la sesión es transaccional.

Devuelve:
una sesión de temas recién creada.

Emite: JMSException - si la conexión de JMS no puede crear una sesión debido a un error interno, o si no hay soporte adecuado para la transacción específica y la modalidad de reconocimiento.

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                          (Topic topic,
                           java.lang.String messageSelector,
                           ServerSessionPool sessionPool,
                           int maxMessages)
                                    throws JMSException
```

Crea un consumidor de conexiones para esta conexión. Éste es un recurso especializado que no utilizan los clientes JMS habituales.

Parámetros:

- topic - tema al que se va a acceder.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes.
- sessionPool - agrupación de sesiones de servidor que se va a asociar a este consumidor de conexiones.
- maxMessages - número máximo de mensajes que se pueden asignar a una sesión de servidor al mismo tiempo.

Devuelve:

el consumidor de conexiones.

Emite:

- JMSException - si la conexión de JMS no puede crear un consumidor de conexiones debido a un error interno, o si los argumentos para sessionPool no son válidos.
- InvalidSelectorException - si el selector de mensajes no es válido.

Vea también:

ConnectionConsumer

createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer
                              (Topic topic,
                               java.lang.String subscriptionName
                               java.lang.String messageSelector,
                               ServerSessionPool sessionPool,
                               int maxMessages)
                                        throws JMSException
```

Crea un conexión de conexiones duraderas para esta conexión. Éste es un recurso especializado que no utilizan los clientes JMS habituales.

Parámetros:

- topic - tema al que se va a acceder.
- subscriptionName - nombre de la suscripción duradera.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes.
- sessionPool - agrupación de sesiones de servidor que se va a asociar a este consumidor de conexiones duraderas.
- maxMessages - número máximo de mensajes que se pueden asignar a una sesión de servidor al mismo tiempo.

Devuelve:

el consumidor de conexiones duraderas.

Emite:

- JMSException - si la conexión de JMS no puede crear un consumidor de conexiones debido a un error interno, o si los argumentos para sessionPool y messageSelector no son válidos.
- InvalidSelectorException - si el selector de mensajes no es válido.

Vea también:

ConnectionConsumer

TopicConnectionFactory

interfaz pública **TopicConnectionFactory**
expande **ConnectionFactory**
Subinterfaces: **XATopicConnectionFactory**

Clase MQSeries: **MQTopicConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
```

Un cliente utiliza TopicConnectionFactory para crear TopicConnections con un suministrador de JMS Publish/Subscribe.

Vea también: **ConnectionFactory** y **XATopicConnectionFactory**

Constructor de MQSeries

MQTopicConnectionFactory

```
public MQTopicConnectionFactory()
```

Métodos

createTopicConnection

```
public TopicConnection createTopicConnection()
                                        throws JMSException
```

Crea una conexión de temas con la identidad de usuario por omisión. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Devuelve:
una conexión de temas recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear una conexión de temas debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

createTopicConnection

```
public TopicConnection createTopicConnection
                                        (java.lang.String userName,
                                        java.lang.String password)
                                        throws JMSException
```

Crea una conexión de temas con la identidad de usuario especificada. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Nota: Este método sólo es válido para el tipo de transporte IBM_JMS_TP_CLIENT_MQ_TCPIP. Vea ConnectionFactory.

Parámetros:

- userName - nombre de usuario del canal de llamada.
- password - contraseña del canal de llamada.

Devuelve:

una conexión de temas recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear una conexión de temas debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

setBrokerControlQueue *

```
public void setBrokerControlQueue(String x) throws JMSException
```

Establece el método para el atributo brokerControlQueue.

Parámetros:

brokerControlQueue - nombre de la cola de control del intermediario.

getBrokerControlQueue *

```
public String getBrokerControlQueue()
```

Obtiene el método para el atributo brokerControlQueue.

Devuelve:

el nombre de la cola de control del intermediario

setBrokerQueueManager *

```
public void setBrokerQueueManager(String x) throws JMSException
```

Establece el método para el atributo brokerQueueManager.

Parámetros:

brokerQueueManager - nombre del gestor de colas del intermediario.

getBrokerQueueManager *

```
public String getBrokerQueueManager()
```

Obtiene el método para el atributo brokerQueueManager.

Devuelve:

el nombre del gestor de colas del intermediario.

setBrokerPubQueue *

```
public void setBrokerPubQueue(String x) throws JMSException
```

Establece el método para el atributo brokerPubQueue.

Parámetros:

brokerPubQueue - nombre de la cola de publicación del intermediario.

getBrokerPubQueue *

```
public String getBrokerPubQueue()
```

Obtiene el método para el atributo brokerPubQueue.

Devuelve:
el nombre de la cola de publicación del intermediario.

setBrokerSubQueue *

```
public void setBrokerSubQueue(String x) throws JMSException
```

Establece el método para el atributo brokerSubQueue.

Parámetros:
brokerSubQueue - nombre de la cola de suscripciones no duraderas que se va a utilizar.

getBrokerSubQueue *

```
public String getBrokerSubQueue()
```

Obtiene el método para el atributo brokerSubQueue.

Devuelve:
el nombre de la cola de suscripciones no duraderas que se va a utilizar.

setBrokerCCSubQueue *

```
public void setBrokerCCSubQueue(String x) throws JMSException
```

Establece el método para el atributo brokerCCSubQueue.

Parámetros:
brokerSubQueue - nombre de la cola de suscripciones no duraderas que se va a utilizar para ConnectionConsumer.

getBrokerCCSubQueue *

```
public String getBrokerCCSubQueue()
```

Obtiene el método para el atributo brokerCCSubQueue.

Devuelve:
el nombre de la cola de suscripciones no duraderas que se va a utilizar para ConnectionConsumer.

setBrokerVersion *

```
public void setBrokerVersion(int x) throws JMSException
```

Estable el método para el atributo brokerVersion.

Parámetros:
brokerVersion - número de versión del intermediario.

getBrokerVersion *

```
public int getBrokerVersion()
```

Obtiene el método para el atributo brokerVersion.

Devuelve:
el número de versión del intermediario.

getReference *

```
public Reference getReference()
```

Devuelve una referencia para esta fábrica de conexión de temas.

Devuelve:
una referencia para esta fábrica de conexión de temas.

Emite: NamingException.

TopicPublisher

interfaz pública **TopicPublisher**
expande **MessageProducer**

Clase MQSeries: **MQTopicPublisher**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

Un cliente utiliza TopicPublisher para publicar mensajes sobre un tema. TopicPublisher es la variante Pub/Sub de un productor de mensajes JMS.

Métodos

getTopic

```
public Topic getTopic() throws JMSException
```

Obtiene el tema asociado a este publicador.

Devuelve:
el tema de este publicador.

Emite: JMSException - si JMS no puede obtener el tema de este publicador de temas debido a un error interno.

publish

```
public void publish(Message message) throws JMSException
```

Publica un mensaje del tema. Utiliza la prioridad, el tiempo de vida y la modalidad de entrega por omisión del tema.

Parámetros:
message - mensaje que se va a publicar

Emite:

- JMSException - si JMS no puede publicar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con un tema no válido de un publicador de temas.

publish

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

Publica un mensaje sobre el tema especificando la modalidad de entrega, la prioridad y el tiempo de vida del tema.

Parámetros:

- message - mensaje que se va a publicar.
- deliveryMode - modalidad de entrega que se va a utilizar.
- priority - prioridad de este mensaje.
- timeToLive - tiempo de vida del mensaje (en milisegundos).

Emite:

- JMSException - si JMS no puede publicar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con un publicador de temas con un tema no válido.

publish

```
public void publish(Topic topic,
                    Message message) throws JMSException
```

Publica un mensaje de un tema para un productor de mensajes no identificado. Utiliza la prioridad, el tiempo de vida y la modalidad de entrega por omisión del tema.

Parámetros:

- topic - tema de la publicación de este mensaje.
- message - mensaje que se va a enviar.

Emite:

- JMSException - si JMS no puede publicar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con un tema no válido.

publish

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

Publica un mensaje de un tema para un productor de mensajes no identificado, especificando la modalidad de entrega, la prioridad y el tiempo de vida.

Parámetros:

- topic - tema de la publicación de este mensaje.
- message - mensaje que se va a enviar.
- deliveryMode - modalidad de entrega que se va a utilizar.
- priority - prioridad de este mensaje.
- timeToLive - tiempo de vida del mensaje (en milisegundos).

Emite:

- JMSException - si JMS no puede publicar el mensaje debido a un error interno.
- MessageFormatException - si el mensaje especificado no es válido.
- InvalidDestinationException - si un cliente utiliza este método con un tema no válido.

close *

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un TopicPublisher, los clientes deben cerrarlos cuando no los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Emite: JMSException si JMS no puede cerrar el productor debido a un error.

Altera temporalmente:
close en la clase MQMessageProducer.

TopicRequestor

clase pública **TopicRequestor**
expande **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.TopicRequestor
```

JMS proporciona esta clase TopicRequestor como ayuda para la realización de peticiones de servicio.

Se proporciona al constructor TopicRequestor un tema de destino y una TopicSession no transaccional. Crea un TemporaryTopic para las respuestas y proporciona un método request() que envía el mensaje de solicitud y espera su respuesta. Los usuarios pueden crear versiones más complejas si lo desean.

Constructores

TopicRequestor

```
public TopicRequestor(TopicSession session,
                      Topic topic) throws JMSException
```

Constructor para la clase TopicRequestor. Esta implementación da por supuesto que el parámetro de sesión es no transaaccional y AUTO_ACKNOWLEDGE o DUPS_OK_ACKNOWLEDGE.

Parámetros:

- session - sesión de temas a la que pertenece el tema.
- topic - tema en el que se va a realizar la invocación de invocación/respuesta.

Emite: JMSException - si se produce un error de JMS.

Métodos

request

```
public Message request(Message message) throws JMSException
```

Envía una petición y espera una respuesta.

Parámetros:
message - mensaje que se va a enviar.

Devuelve:
el mensaje de respuesta.

Emite: JMSException - si se produce un error de JMS.

close

```
public void close() throws JMSException
```

Puesto que un suministrador puede asignar algunos recursos fuera de JVM en nombre de un TopicRequestor, los clientes deben cerrarlos cuando no

los necesiten. Finalmente, no puede contar con la recopilación de basura para reclamar estos recursos, puesto que puede no llevarse a cabo con la suficiente rapidez.

Nota: Este método cierra el objeto Session que se ha pasado al constructor TopicRequestor.

Emite: JMSException - si se produce un error de JMS.

TopicSession

interfaz pública **TopicSession**
expande **Session**

Clase MQSeries: **MQTopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

TopicSession proporciona métodos para crear TopicPublishers, TopicSubscribers y TemporaryTopics.

Vea también: **Session**

Constructor de MQSeries

MQTopicSession

```
public MQTopicSession(boolean transacted,
                      int acknowledgeMode) throws JMSException
```

Vea también **TopicConnection.createTopicSession**.

Métodos

createTopic

```
public Topic createTopic(java.lang.String topicName)
                                                 throws JMSException
```

Crea un tema con un nombre especificado en formato URI. (En el apartado “Creación de temas durante la ejecución” en la página 197 se describe el formato URI). De este modo, se puede crear un tema con un nombre de suministrador específico.

Nota: Los clientes que dependen de esta posibilidad no son portátiles.

Parámetros:
topicName - nombre de este tema.

Devuelve:
un tema con el nombre especificado.

Emite: JMSException - si una sesión no puede crear un tema debido a un error de JMS.

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic)
                                        throws JMSException
```

Crea un suscriptor no duradero para el tema especificado.

Parámetros:
topic - tema al que se va a suscribir

Emite:

- JMSException - si una sesión no puede crear un suscriptor debido a un error de JMS.
- InvalidDestinationException - si el tema especificado no es válido.

createSubscriber

```
public TopicSubscriber createSubscriber
                            (Topic topic,
                             java.lang.String messageSelector,
                             boolean noLocal) throws JMSException
```

Crea un suscriptor no duradero para el tema especificado.

Parámetros:

- topic - tema al que se va a suscribir.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes. Este valor puede ser nulo.
- noLocal - si se establece, inhibe la entrega de los mensajes que ha publicado su propia conexión.

Emite:

- JMSException - si una sesión no puede crear un suscriptor debido a un error de JMS o un selector no válido.
- InvalidDestinationException - si el tema especificado no es válido.
- InvalidSelectorException - si el selector de mensajes no es válido.

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name) throws JMSException
```

Crea un suscriptor duradero para el tema especificado. Un cliente puede cambiar una suscripción duradera existente creando un suscriptor duradero con el mismo nombre y un nuevo tema y/o selector de mensajes.

Parámetros:

- topic - tema al que se va a suscribir.
- name - nombre que se va a utilizar para identificar esta suscripción.

Emite:

- JMSException - si una sesión no puede crear un suscriptor debido a un error de JMS.
- InvalidDestinationException - si el tema especificado no es válido.

Vea también **TopicSession.unsubscribe**

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name,
                                     java.lang.String messageSelector,
                                     boolean noLocal)  throws JMSException
```

Crea un suscriptor duradero para el tema especificado.

Parámetros:

- topic - tema al que se va a suscribir.
- name - nombre que se va a utilizar para identificar esta suscripción.
- messageSelector - sólo se entregan los mensajes cuyas propiedades coinciden con la expresión del selector de mensajes. Este valor puede ser nulo.
- noLocal - si se establece, inhibe la entrega de los mensajes que ha publicado su propia conexión.

Emite:

- JMSException - si una sesión no puede crear un suscriptor debido a un error de JMS o un selector no válido.
- InvalidDestinationException - si el tema especificado no es válido.
- InvalidSelectorException - si el selector de mensajes no es válido.

createPublisher

```
public TopicPublisher createPublisher(Topic topic)
                                      throws JMSException
```

Crea un publicador para el tema especificado.

Parámetros:

topic - tema con el que se va a publicar o nulo si se va a tratar de un productor no identificado.

Emite:

- JMSException - si una sesión no puede crear un publicador debido a un error de JMS.
- InvalidDestinationException - si el tema especificado no es válido.

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                       throws JMSException
```

Crea un tema temporal. Su duración es la de TopicConnection, a menos que se haya suprimido antes.

Devuelve:

un tema temporal.

Emite: JMSException - si una sesión no puede crear un tema temporal debido a un error de JMS.

unsubscribe

```
public void unsubscribe(java.lang.String name)
                                        throws JMSException
```

Anula una suscripción duradera creada por un cliente.

Nota: Este método no se debe utilizar mientras exista una suscripción activa. Antes se debe cerrar (close()) el suscriptor.

Parámetros:

name - nombre que se va a utilizar para identificar esta suscripción.

Emite:

- JMSException - si JMS no puede anular la suscripción duradera debido a un error de JMS error.
- InvalidDestinationException - si el tema especificado no es válido.

TopicSubscriber

interfaz pública **TopicSubscriber**
expande **MessageConsumer**

Clase MQSeries: **MQTopicSubscriber**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

Un cliente utiliza TopicSubscriber para recibir mensajes que se han publicado de un tema. TopicSubscriber es la variante Pub/Sub de un consumidor de mensajes JMS.

Vea también: **MessageConsumer** y **TopicSession.createSubscriber**

MQTopicSubscriber hereda los métodos siguientes de MQMessageConsumer:
close
getMessageListener
receive
receiveNoWait
setMessageListener

Métodos

getTopic

```
public Topic getTopic() throws JMSException
```

Obtiene el tema asociado a este suscriptor.

Devuelve:
el tema de este suscriptor.

Emite: JMSException - si JMS no puede obtener el tema para este suscriptor de temas debido a un error interno.

getNoLocal

```
public boolean getNoLocal() throws JMSException
```

Obtiene el atributo NoLocal para este TopicSubscriber. El valor por omisión para este atributo es falso (false).

Devuelve:
establecido en verdadero si se están inhibiendo los mensajes publicados localmente.

Emite: JMSException - si JMS no puede obtener el atributo NoLocal para este suscriptor de temas debido a un error interno.

XAConnection

interfaz pública **XAConnection**
Subinterfaces: **XAQueueConnection** y **XATopicConnection**

Clase MQSeries: **MQXAConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnection
```

XAConnection expande las posibilidades de Connection al proporcionar una XASession. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **XAQueueConnection** y **XATopicConnection**

XAConnectionFactory

interfaz pública **XAConnectionFactory**
Subinterfaces: **XAQueueConnectionFactory** y **XATopicConnectionFactory**

Clase MQSeries: **MQXAConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnectionFactory
```

Algunos servidores de aplicaciones ofrecen soporte para agrupar el uso de recursos con posibilidades JTS en una transacción distribuida. Para incluir transacciones JMS en una transacción JTS, un servidor de aplicaciones necesita un suministrador de JMS que tenga constancia de JTS. Un suministrador de JMS muestra su soporte de JTS utilizando una XAConnectionFactory de JMS, y el servidor de aplicaciones la utiliza para crear XASessions. XAConnectionFactories son objetos administrados por JMS, del mismo modo que las ConnectionFactories. Los servidores de aplicaciones deben utilizar JNDI para buscarlos.

Consulte el “Apéndice E. Interfaz JMS JTA/XA con WebSphere™” en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **XAQueueConnectionFactory** y **XATopicConnectionFactory**

XAQueueConnection

interfaz pública **XAQueueConnection**
expande **QueueConnection** y **XAConnection**

Clase MQSeries: **MQXAQueueConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQQueueConnection
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnection
```

XAQueueConnection proporciona las mismas opciones de creación que QueueConnection. La única diferencia es que, por definición, una XAConnection es transaccional. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **XAConnection** and **QueueConnection**

Métodos

createXAQueueSession

```
public XAQueueSession createXAQueueSession()
```

Crea una XAQueueSession.

Emite: JMSException - si la conexión de JMS no puede crear una sesión de colas XA debido a un error interno.

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                              throws JMSException
```

Crea una QueueSession.

Parámetros:

- transacted - si es verdadero, la sesión es transaccional.
- acknowledgeMode - indica si el consumidor o el cliente van a reconocer los mensajes que reciben. Los valores posibles son:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Este parámetro se ignora si la sesión es transaccional.

Devuelve:
un sesión de colas recién creada (tenga en cuenta que no es una sesión de colas XA).

Emite: JMSException - si la conexión de JMS no puede crear una sesión de colas debido a un error interno.

XAQueueConnectionFactory

interfaz pública **XAQueueConnectionFactory**
expande **QueueConnectionFactory** y **XAConnectionFactory**

Clase MQSeries: **MQXAQueueConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQQueueConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

XAQueueConnectionFactory proporciona las mismas opciones de creación que QueueConnectionFactory. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **QueueConnectionFactory** and **XAConnectionFactory**

Métodos

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection()
                                          throws JMSException
```

Crea una XAQueueConnection utilizando la identidad de usuario por omisión. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Devuelve:
una conexión de colas XA recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear una conexión de colas XA debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection
                                  (java.lang.String userName,
                                   java.lang.String password)
                                                 throws JMSException
```

Crea una conexión de colas XA utilizando una identidad de usuario específica. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Parámetros:

- userName - nombre de usuario del canal de llamada.
- password - contraseña del canal de llamada.

Devuelve:

una conexión de colas XA recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear una conexión de colas XA debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

XAQueueSession

interfaz pública **XAQueueSession**
expande **XASession**

Clase MQSeries: **MQXAQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXAQueueSession
```

XAQueueSession proporciona una sesión QueueSession normal que se puede utilizar para crear QueueReceivers, QueueSenders y QueueBrowsers. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Se puede obtener el XAResource que corresponde a la QueueSession invocando al método getXAResource, que se hereda de XASession.

Vea también: **XASession**

Métodos

getQueueSession

```
public QueueSession  getQueueSession()
                                 throws JMSException
```

Obtiene la sesión de colas asociada a esta XAQueueSession.

Devuelve:
el objeto de la sesión de colas.

Emite: JMSException - si se produce un error de JMS.

XASession

interfaz pública **XASession**
expande **Session**
Subinterfaces: **XAQueueSession** y **XATopicSession**

Clase MQSeries: **MQXASession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
```

XASession expande las posibilidades de Session al añadir acceso al soporte de un suministrador de JMS para JTA. Este soporte toma la forma de un objeto javax.transaction.xa.XAResource. Las funciones de este objeto son muy similares a las que define la interfaz de recursos XA de X/Open estándar.

Un servidor de aplicaciones controla la asignación transaccional de una XASession obteniendo su XAResource. Utiliza XAResource para asignar la sesión a una transacción, preparar y confirmar el trabajo de la transacción y así sucesivamente.

XAResource proporciona algunos recursos bastante complejos como, por ejemplo, intercalar el trabajo en varias transacciones y recuperar una lista de las transacciones en curso.

Un suministrador de JMS que tenga constancia de JTA debe implementar completamente esta función. Para hacerlo, el suministrador de JMS puede utilizar los servicios de una base de datos que ofrezca soporte para XA, o puede implementar esta función desde el principio.

Un cliente del servidor de aplicaciones dispone de lo que parece ser una sesión de JMS normal. Detrás de los bastidores, el servidor de aplicaciones controla la gestión de las transacciones de la XASession subyacente.

Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **XAQueueSession** y**XATopicSession**

Métodos

getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

Devuelve un recurso XA al canal de llamada.

Devuelve:
un recurso XA al canal de llamada.

getTransacted

```
public boolean getTransacted()
                       throws JMSException
```

Siempre devuelve verdadero (true).

Especificado por:
getTransacted en la interfaz Session.

Devuelve:
true - si la sesión está en modalidad transaccional.

Emite: JMSException - si JMS no puede devolver la modalidad de transacción debido a un error interno del suministrador de JMS.

commit

```
public void commit()
            throws JMSException
```

Este método no se debe invocar para un objeto XASession. Si se invoca, emite una TransactionInProgressException.

Especificado por:
commit en la interfaz Session.

Emite: TransactionInProgressException - si se invoca este método en una XASession.

rollback

```
public void rollback()
              throws JMSException
```

Este método no se debe invocar para un objeto XASession. Si se invoca, emite una TransactionInProgressException.

Especificado por:
rollback en la interfaz Session.

Emite: TransactionInProgressException - si se invoca este método en una XASession.

XATopicConnection

interfaz pública **XATopicConnection**
expande **TopicConnection** y**XAConnection**

Clase MQSeries: **MQXATopicConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQTopicConnection
                    |
                    +----com.ibm.mq.jms.MQXATopicConnection
```

XATopicConnection proporciona las mismas opciones de creación que TopicConnection. La única diferencia es que, por definición, una XAConnection es transaccional. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **TopicConnection** y **XAConnection**

Métodos

createXATopicSession

```
public XATopicSession createXATopicSession()
                                      throws JMSException
```

Crea una XATopicSession.

Emite: JMSException - si la conexión de JMS no puede crear una sesión de temas XA debido a un error interno.

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                throws JMSException
```

Crea una TopicSession.

Especificado por:
createTopicSession en la interfaz TopicConnection.

Parámetros:

- transacted - si es verdadero, la sesión es transaccional.
- acknowledgeMode - uno de los siguientes:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Indica si el consumidor o el cliente van a reconocer los mensajes que reciben. Este parámetro se ignora si la sesión es transaccional.

Devuelve:
una sesión de temas recién creada (tenga en cuenta que no es una sesión de temas XA).

Emite: JMSException - si la conexión de JMS no puede crear una sesión de temas debido a un error interno.

XATopicConnectionFactory

interfaz pública **XATopicConnectionFactory**
expande **TopicConnectionFactory** y
XAConnectionFactory

Clase MQSeries: **MQXATopicConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQTopicConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

XATopicConnectionFactory proporciona las mismas opciones de creación que TopicConnectionFactory. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Vea también: **TopicConnectionFactory** y**XAConnectionFactory**

Métodos

createXATopicConnection

```
public XATopicConnection createXATopicConnection()
                                        throws JMSException
```

Crea una conexión de temas XA utilizando la identidad de usuario por omisión. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Devuelve:

una conexión de temas XA recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear una conexión de temas XA debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

createXATopicConnection

```
public XATopicConnection createXATopicConnection(java.lang.String userName,
                                                 java.lang.String password)
                                        throws JMSException
```

Crea una conexión de temas XA utilizando la identidad de usuario especificada. La conexión se crea en modalidad detenida. No se entrega ningún mensaje hasta que se invoca explícitamente al método Connection.start.

Parámetros:

- userName - nombre de usuario del canal de llamada
- password - contraseña del canal de llamada

Devuelve:

una conexión de temas XA recién creada.

Emite:

- JMSException - si el suministrador de JMS no puede crear una conexión de temas XA debido a un error interno.
- JMSSecurityException - si la autentificación de cliente no se ejecuta correctamente debido a una contraseña o un nombre de usuario no válidos.

XATopicSession

interfaz pública **XATopicSession**
expande **XASession**

Clase MQSeries: **MQXATopicSession**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQXASession
            |
            +----com.ibm.mq.jms.MQXATopicSession
```

XATopicSession proporciona una TopicSession, que se puede utilizar para crear TopicSubscribers y TopicPublishers. Consulte el "Apéndice E. Interfaz JMS JTA/XA con WebSphere™" en la página 377 para obtener información más detallada sobre el modo en que MQ JMS utiliza las clases XA.

Se puede obtener el XAResource que corresponde a la TopicSession invocando al método getXAResource, que se hereda de XASession.

Vea también: **TopicSession** y**XASession**

Métodos

getTopicSession

```
public TopicSession getTopicSession()
                            throws JMSException
```

Obtiene la sesión de temas asociada a esta XATopicSession.

Devuelve:
el objeto de la sesión de temas.

Emite:

- JMSException - si se produce un error de JMS.

Parte 4. Apéndices

Apéndice A. Correlación entre las propiedades de la herramienta de administración y las propiedades programables

El Servicio de mensajes de MQSeries® Classes for Java™ proporciona recursos para establecer y consultar las propiedades de los objetos administrados utilizando la herramienta de administración de MQ JMS o un programa de aplicación. La Tabla 30 muestra la correlación entre cada nombre de propiedad utilizado con la herramienta de administración y la variable de miembro correspondiente a la que hace referencia. También muestra la correlación entre los valores de propiedad simbólicos que se utilizan en la herramienta y sus equivalentes programables.

Tabla 30. Comparación de las representaciones de las propiedades entre la herramienta de administración y los equivalentes programables.

Propiedad	**Nombre de variable de miembro**	**Correlación del valor de propiedad**	
		Herramienta	**Programa**
DESCRIPTION	description		
TRANSPORT	transportType	• BIND • CLIENT	JMSC.MQJMS_TP_BINDINGS_MQ JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
CLIENTID	clientId		
QMANAGER	queueManager*		
HOSTNAME	hostName		
PORT	port		
CHANNEL	channel		
CCSID	CCSID		
RECEXIT	receiveExit		
RECEXITINIT	receiveExitInit		
SECEXIT	securityExit		
SECEXITINIT	securityExitInit		
SENDEXIT	sendExit		
SENDEXITINIT	sendExitInit		
TEMPMODEL	temporaryModel		
MSGRETENTION	messageRetention	• YES • NO	JMSC.MQJMS_MRET_YES JMSC.MQJMS_MRET_NO
BROKERVER	brokerVersion	• V1	JMSC.MQJMS_BROKER_V1
BROKERPUBQ	brokerPubQueue		
BROKERSUBQ	brokerSubQueue		
BROKERDURSUBQ	brokerDurSubQueue		
BROKERCCSUBQ	brokerCCSubQueue		
BROKERCCDSUBQ	brokerCCDurSubQueue		
BROKERQMGR	brokerQueueManager		
BROKERCONQ	brokerControlQueue		

Propiedades

Tabla 30. Comparación de las representaciones de las propiedades entre la herramienta de administración y los equivalentes programables. (continuación)

Propiedad	Nombre de variable de miembro	Correlación del valor de propiedad	
		Herramienta	**Programa**
EXPIRY	expiry	• APP • UNLIM	JMSC.MQJMS_EXP_APP JMSC.MQJMS_EXP_UNLIMITED
PRIORITY	priority	• APP • QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
PERSISTENCE	persistence	• APP • QDEF • PERS • NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON
TARGCLIENT	targetClient	• JMS • MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
ENCODING	encoding		
QUEUE	baseQueueName		
TOPIC	baseTopicName		
Nota: * para un objeto MQQueue, el nombre de la variable de miembro es baseQueueManagerName			

Apéndice B. Scripts que se proporcionan con Servicio de mensajes de MQSeries Classes for Java

Los archivos siguientes se proporcionan en el directorio `bin` de la instalación de MQ JMS. La finalidad de estos scripts es ayudarle a realizar las tareas comunes que se deben llevar a cabo al instalar o utilizar MQ JMS. En la Tabla 31 se incluye la lista de los scripts y sus utilizaciones.

Tabla 31. Programas de utilidad que se proporcionan con Servicio de mensajes de MQSeries Classes for Java

Programa de utilidad	Utilización
IVTRun.bat IVTTidy.bat IVTSetup.bat	Se utilizan para ejecutar el programa de prueba de verificación de la instalación punto a punto. Se describe en el apartado "Ejecución de IVT punto a punto" en la página 25.
PSIVTRun.bat	Se utiliza para ejecutar el programa de prueba de verificación de la instalación de Pub/Sub. Se describe en el apartado "Prueba de verificación de la instalación de Publish/Subscribe" en la página 30.
formatLog.bat	Se utiliza para convertir archivos de anotaciones binarios en archivos de texto sin formato. Se describe en el apartado "Anotaciones cronológicas" en la página 34.
JMSAdmin.bat	Se utiliza para ejecutar la herramienta de administración. Se describe en el "Capítulo 5. Utilización de la herramienta de administración de MQ JMS" en la página 35.
JMSAdmin.config	Archivo de configuración para la herramienta de administración. Se describe en el apartado "Configuración" en la página 36.
runjms.bat	Script de programa de utilidad para ayudarle a ejecutar las aplicaciones JMS. Se describe en el apartado "Ejecución de programas MQ JMS propios" en la página 33.
PSReportDump.class	Se utiliza para visualizar los mensajes de informe del intermediario. Se describe en el apartado "Manejo de informes del intermediario" en la página 203.
Nota: En los sistemas UNIX®, la extensión '.bat' se omite en los nombres de archivo.	

Scripts

Apéndice C. Configuración del servidor LDAP para objetos Java™

Si utiliza JNDI para almacenar los objetos que administra MQ JMS y utiliza un servidor LDAP como suministrador de servicio JNDI, el servidor debe ser LDAP v3 (por ejemplo, SecureWay® eNetwork Directory v3.1), y se debe haber configurado para almacenar objetos Java™.

Comprobación de la configuración del servidor LDAP

Para comprobar si el servidor LDAP ya está configurado para aceptar objetos Java™, ejecute la herramienta de administración MQ JMS en la modalidad LDAP (consulte el apartado "Invocación de la herramienta de administración" en la página 35).

Intente crear y visualizar un objeto de prueba utilizando los mandatos siguientes:

```
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

Si no se produce ninguna excepción, significa que el servidor está correctamente configurado y puede empezar a almacenar objetos JMS.

Si se devuelve una 'SchemaViolationException' o si se muestra el mensaje 'No se puede enlazar el objeto', significa que el servidor no está correctamente configurado. El servidor no está configurado para almacenar objetos Java™ o los permisos de los objetos o el sufijo no son correctos. Los procedimientos siguientes pueden ayudarle a realizar la tarea de configuración.

Procedimientos de configuración

Muchos servidores LDAP proporcionan herramientas que le permiten administrar el servidor. Consulte la documentación del servidor para obtener información más detallada acerca de la utilización de dichas herramientas. Las herramientas deben permitirle visualizar y actualizar el esquema, que contiene definiciones de 'attribute' (atributos) y 'objectclass' (clases de objeto).

Asegúrese de que el esquema contiene las definiciones de objectclass siguientes y añádalas si fuera necesario:

```
( 1.3.6.1.4.1.42.2.27.4.2.1
  NAME 'javaContainer'
  DESC 'Container for a Java object'
  SUP top
  STRUCTURAL
  MUST ( cn )
)
( 1.3.6.1.4.1.42.2.27.4.2.4
  NAME 'javaObject'
  DESC 'Java object representation'
  SUP top
  ABSTRACT
  MUST ( javaClassName )
  MAY ( javaClassNames $
```

```
        javaCodebase $
        javaDoc $
        description )
)
( 1.3.6.1.4.1.42.2.27.4.2.5
  NAME 'javaSerializedObject'
  DESC 'Java serialized object'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)
( 1.3.6.1.4.1.42.2.27.4.2.7
  NAME 'javaNamingReference'
  DESC 'JNDI reference'
  SUP javaObject
  AUXILIARY
  MAY ( javaReferenceAddress $
        javaFactory )
)
```

Asegúrese también de que el esquema contiene las definiciones de atributo siguientes y actualice el esquema si fuera necesario:

```
( 1.3.6.1.4.1.42.2.27.4.1.11
        NAME 'javaReferenceAddress'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.10
        NAME 'javaFactory'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.7
        NAME 'javaCodebase'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Cuando haya terminado de realizar las actualizaciones, detenga y reinicie el servidor LDAP y, a continuación, repita el procedimiento de comprobación de la configuración que se describe en el apartado “Comprobación de la configuración del servidor LDAP” en la página 373.

Apéndice D. Conexión a MQSeries® Integrator V2

Puede utilizar MQSeries Integrator V2:

- como intermediario de publicación/suscripción para MQ JMS
- para direccionar o transformar mensajes creados por una aplicación cliente JMS y enviar o publicar mensajes en un cliente JMS

Publicación/suscripción

Puede utilizar MQSeries Integrator V2 como intermediario de publicación/suscripción para MQ JMS. Para hacerlo, debe realizar las actividades de configuración siguientes:

- MQSeries básico

  En primer lugar, debe crear una cola de publicación del intermediario. Se trata de una cola MQSeries en el gestor de colas del intermediario y se utiliza para someter publicaciones al intermediario. Puede elegir el nombre que desea para esta cola, pero debe coincidir con el nombre de cola de la propiedad BROKERPUBQ de TopicConnectionFactory. Por omisión, la propiedad BROKERPUBQ de TopicConnectionFactory se establece en el valor SYSTEM.BROKER.DEFAULT.STREAM por lo que, a menos que desee configurar otro nombre en TopicConnectionFactory, debe denominar la cola SYSTEM.BROKER.DEFAULT.STREAM.

- MQSeries Integrator V2

  El paso siguiente consiste en configurar un *flujo de mensajes* en un grupo de ejecución para el intermediario. La finalidad de este flujo de mensajes es leer los mensajes de la cola de publicación del intermediario. (Si lo desea, puede configurar varias colas de publicación, pero cada una de ellas necesita una TopicConnectionFactory y un flujo de mensajes propios).

  El flujo de mensajes básico consta de un nodo MQInput (configurado para leer de la cola SYSTEM.BROKER.DEFAULT.STREAM) cuya salida esté conectada a la entrada de un nodo Publication (o MQOutput).

  El diagrama del flujo de mensajes tiene, por consiguiente, un aspecto similar al que se muestra a continuación:

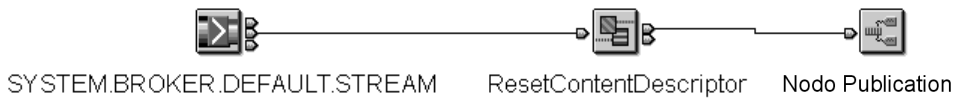


Figura 7. Flujo de mensajes de MQSeries Integrator

Al difundir el flujo de mensajes e iniciar el intermediario, desde la perspectiva de la aplicación JMS, el intermediario de MQSeries Integrator V2 se presenta como un intermediario de MQSeries Publish/Subscribe. El estado actual de las suscripciones se puede visualizar utilizando el Centro de control de MQSeries Integrator.

Notas:

1. No es necesario realizar ninguna modificación en el Servicio de mensajes de MQSeries Classes for Java™.

2. Los intermediarios de MQSeries Publish/Subscribe y MQSeries Integrator V2 no pueden coexistir en el mismo gestor de colas.
3. En la publicación *MQSeries Integrator para Windows NT® Versión 2.0 Guía de instalación* se proporciona información detallada sobre los procedimientos de instalación y configuración de MQSeries Integrator V2.

Transformación y direccionamiento

Puede utilizar MQSeries Integrator V2 para direccionar o transformar los mensajes creados por una aplicación cliente JMS y enviar o publicar mensajes en un cliente JMS.

La implementación MQSeries JMS utiliza la carpeta mcd de MQRFH2 para transferir información sobre los mensajes, tal como se describe en el apartado "Cabecera MQRFH2" en la página 211. Por omisión, la propiedad Msd (Dominio del mensaje) se utiliza para identificar si se trata de un mensaje de texto, de bytes, de corriente de datos, de correlación o de objeto.

Cuando una aplicación JMS crea un mensaje de bytes o de texto, la aplicación puede alterar temporalmente esta propiedad Msd y establecer otros campos de la carpeta mcd. Lo hace estableciendo una propiedad de tipo (Type) de JMS con un formato URI especial, por ejemplo:

`mcd://dominio/conjuntot/tipo[?formato=fmt]`

Los valores de los campos *dominio*, *conjunto*, *tipo* y *fmt* (donde *fmt* es opcional) se copian en el MQRFH2 de salida. Significa que la aplicación puede establecer estos campos en valores que reconozca un flujo de mensajes MQSeries Integrator V2.

Apéndice E. Interfaz JMS JTA/XA con WebSphere™

El Servicio de mensajes de MQSeries Classes for Java incluye las interfaces JMS XA. Permiten que MQ JMS participe en una confirmación en dos fases que coordina un gestor de transacciones que cumple con JTA (Java™ Transaction API).

En este apartado se describe cómo utilizar estas características con WebSphere™ Application Server, Advanced Edition, con el fin de que WebSphere pueda coordinar las operaciones de envío y recepción de JMS y las actualizaciones de base de datos, en una transacción global.

Antes de utilizar MQ JMS y las clases XA con WebSphere™, quizá deba realizar pasos de configuración o instalación adicionales. Consulte el archivo `Readme.txt` de la página web de MQSeries Utilización de Java SupportPac para obtener la información más actualizada (`www.ibm.com/software/ts/mqseries/txppacs/ma88.html`).

Utilización de la interfaz JMS con WebSphere™

En este apartado se proporcionan las instrucciones necesarias para utilizar la interfaz JMS con WebSphere™ Application Server, Advanced Edition.

Previamente, debe comprender los conceptos básicos de los programas JMS, MQSeries® y los beans EJB. Puede obtener información detallada en la especificación JMS, la especificación EJB V2 (ambas disponibles a través de Sun), esta publicación, los ejemplos que se proporcionan con MQ JMS y otras publicaciones para MQSeries® y WebSphere™.

Objetos administrados

JMS utiliza objetos administrados para encapsular información específica del proveedor. De este modo, se minimiza el impacto de los detalles específicos del proveedor en las aplicaciones de usuario final. Los objetos administrados se almacenan en un espacio de nombres JNDI y se pueden recuperar y utilizar de un modo portátil sin que seanecesario conocer el contenido específico del proveedor.

Para la utilización autónoma, MQ JMS proporciona las clases siguientes:

- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere™ facilita un par adicional de objetos administrados, que permite que MQ JMS se pueda integrar con WebSphere:

- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

Puede utilizar estos objetos exactamente del mismo modo que MQQueueConnectionFactory y MQTopicConnectionFactory. Sin embargo, utilizan las versiones XA de las clases JMS e incluyen MQ XAResource en la transacción de WebSphere™.

Transacciones gestionadas por contenedor respecto a transacciones gestionadas por bean

Las transacciones gestionadas por contenedor son las transacciones de beans EJB que demarca automáticamente el contenedor de EJB. Las transacciones gestionadas por bean son las transacciones de beans EJB que demarca el programa (a través de la interfaz UserTransaction).

Confirmación en dos fases respecto a optimización de una fase

El coordinador de WebSphere™ sólo invoca una confirmación en dos fases verdadera si se utiliza más de un XAResource en una transacción determinada. Las transacciones que implica un solo recurso se confirman utilizando una optimizaciones una fase. De este modo se evita en gran medida la necesidad de utilizar ConnectionFactories diferentes para transacciones distribuidas y no distribuidas.

Definición de objetos administrados

Puede utilizar la herramienta de administración MQ JMS para definir las fábricas de conexiones específicas de WebSphere y almacenarlas en un espacio de nombres JNDI. El archivo admin.config, situado en *dir_instal_MQ*/bin debe contener las líneas siguientes:

```
 INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
PROVIDER_URL=iiop://nombreSistemaPrincipal/
```

dir_instal_MQ es el directorio de instalación de MQ JMS y *nombreSistemaPrincipal* es el nombre o la dirección IP de la máquina que ejecuta WebSphere™.

Para acceder a com.ibm.ejs.ns.jndi.CNInitialContextFactory, debe añadir el archivo ejs.jar del directorio lib de WebSphere™ en la CLASSPATH.

Para crear las nuevas fábricas, utilice el verbo de definición (define) con los dos nuevos tipos que se indican a continuación:

```
def WSQCF(nombre) [properties]
def WSTCF(nombre) [properties]
```

Estos nuevos tipos utilizan las mismas propiedades que los tipos QCF o TCF equivalentes, excepto que sólo se permite el tipo de transporte BIND (y, por consiguiente, no se pueden configurar propiedades de cliente). Para obtener información más detallada, consulte el apartado “Administración de objetos JMS” en la página 40.

Recuperación de objetos de administración

En un bean EJB, puede recuperar los objetos que administra JMS utilizando el método InitialContext.lookup(), por ejemplo:

```
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

Los objetos se pueden enviar y utilizar como interfaces JMS genéricas. Normalmente, no suele ser necesario programar las clases específicas de MQSeries® en el código de aplicación.

Ejemplos

Hay tres ejemplos que ilustran los conceptos básicos de la utilización de MQ JMS con WebSphere™ Application Server Advanced Edition. Están situados en subdirectorios de *dir_instal_MQ*/samples/ws, donde *dir_instal_MQ* es el directorio de instalación de MQ JMS.

- El Ejemplo 1 (Sample1) muestra una transferencia y obtención simple de un mensaje de una cola utilizando transacciones gestionadas por contenedor.
- El Ejemplo 2 (Sample2) muestra una transferencia y obtención simple de un mensaje de una cola utilizando transacciones gestionadas por bean.
- El Ejemplo 3 (Sample3) muestra la utilización de la API de publicación/suscripción.

Para obtener información más detallada sobre la creación y la difusión de beans EJB, consulte la documentación de WebSphere™ Application Server.

El archivos readme.txt de cada directorio de ejemplo incluyen salida de ejemplo de cada bean EJB. En los scripts que se proporcionan se da por supuesto que hay un gestor de colas disponible en la máquina local. Si su instalación varía del valor por omisión, puede editar estos scripts según sea necesario.

Ejemplo 1

Sample1EJB.java, situado en el directorio sample1, define dos métodos que utilizan JMS:

- putMessage() envía un TextMessage a una cola y devuelve el MessageID del mensaje enviado
- getMessage() lee el mensaje con el MessageID especificado de la cola

Antes de ejecutar el ejemplo, debe almacenar dos objetos administrados en el espacio de nombres JNDI de WebSphere™:

QCF1 una fábrica de conexiones de cola específica de WebSphere

Q1 una cola

Ambos objetos deben enlazarse en el subcontexto jms/Samples.

Para configurar los objetos administrados, puede utilizar la herramienta de administración MQ JMS y configurarlos manualmente, o puede utilizar el script que se proporciona.

La herramienta de administración MQ JMS se debe haber configurado para acceder al espacio de nombres WebSphere™. Para obtener información más detallada sobre el modo de configurar la herramienta de administración, consulte el apartado “Configuración para WebSphere” en la página 37.

Para configurar los objetos administrados con los valores por omisión habituales, puede entrar el mandato siguiente para ejecutar el script `admin.scp`:

```
JMSAdmin < admin.scp
```

El bean se debe difundir con los métodos getMessage y putMessage marcados como TX_REQUIRED. De este modo, se asegura que el contenedor inicie una transacción antes de especificar cada método y la confirme cuando finalice el método. En los métodos, no necesita ningún código de aplicación que se relacione

con el estado transaccional. Sin embargo, recuerde que el mensaje enviado desde putMessage se produce bajo el punto de sincronismo y no va a estar disponible hasta que se confirme la transacción.

En el directorio sample1 hay un programa cliente simple, Sample1Client.java, para invocar al bean EJB. También se ha incluido un script, runClient, con el fin de simplificar la ejecución de este programa.

El programa cliente (o script) toma un único parámetro, que se utiliza como cuerpo de un TextMessage que envía el método putMessage del bean EJB. A continuación, se invoca getMessage para leer el mensaje que vuelve de la cola y devolver el cuerpo al cliente para su visualización. El bean EJB envía mensajes de progreso a la salida estándar (stdout) del servidor de aplicaciones, lo que le permite supervisar la salida durante la ejecución.

Si el servidor de aplicaciones está en una máquina remota respecto al cliente, es posible que deba editar Sample1Client.java. Si no utiliza los valores por omisión, quizá deba editar el script runClient para que coincida con la vía de acceso de la instalación local y el nombre del archivo jar difundido.

Ejemplo 2

Sample2EJB.java, situado en el directorio sample2, realiza la misma tarea que sample1, y necesita los mismos objetos administrados. A diferencia de sample1, sample2 utiliza transacciones gestionadas por bean para controlar los límites transaccionales.

Si aún no ha ejecutado el ejemplo 1, asegúrese de que configura los objetos administrados QCF1 y Q1, tal como se describe en el “Ejemplo 1” en la página 379.

Los métodos putMessage y getMessage empiezan por obtener una instancia de UserTransaction, y la utilizan para crear una transacción a través del método UserTransaction.begin(). Después, el cuerpo principal del código es el mismo que el de sample1 hasta que finaliza cada método. Al final de cada método, la transacción termina con la invocación UserTransaction.commit().

En el directorio sample2 hay un programa cliente simple, Sample2Client.java, para invocar al bean EJB. También se ha incluido un script, runClient, con el fin de simplificar la ejecución de este programa. Los puede utilizar siguiendo el procedimiento que se ha descrito en el “Ejemplo 1” en la página 379.

Ejemplo 3

Sample3EJB.java, situado en el directorio sample3, muestra la utilización de la API de publicación/suscripción con WebSphere™. La publicación de un mensaje es muy similar al caso de punto a punto. Sin embargo, existen diferencias cuando se reciben mensajes a través de TopicSubscriber.

Normalmente, los programas de publicación/suscripción utilizan suscriptores no duraderos, que sólo existen mientras duran sus sesiones (o menos, si se cierra el suscriptor de forma explícita). Además, sólo pueden recibir mensajes del intermediario durante dicha duración.

Para convertir sample1 para publicación/suscripción, puede sustituir el QueueSender de putMessage por un TopicPublisher, y el QueueReceiver de getMessage por un TopicSubscriber no duradero. Sin embargo, se producirá una

anomalía, puesto que cuando se envíe el mensaje, el intermediario no conocerá ninguno de los suscriptores del tema y, por consiguiente, se descartará el mensaje.

La solución consiste en crear un suscriptor duradero antes de publicar el mensaje. Los suscriptores duraderos permanecen como un punto final de entrega después de la duración de la sesión. Por este motivo, el mensaje está disponible para su recuperación durante la invocación de getMessage().

El bean EJB incluye dos métodos adicionales:

- createSubscription crea una suscripción duradera
- destroySubscription suprime una suscripción duradera

Estos métodos (junto con putMessage y getMessage) se deben difundir con el atributo TX_REQUIRED.

Interfaz JMS JTA/XA con WebSphere

Antes de ejecutar sample3, debe almacenar dos objetos administrados en el espacio de nombres JNDI de WebSphere™:

TCF1

T1

Ambos objetos deben enlazarse en el subcontexto jms/Samples.

Para configurar los objetos administrados, puede utilizar la herramienta de administración MQ JMS y configurarlos manualmente, o puede utilizar un script. En el directorio sample3 se proporciona el script `admin.scp`.

La herramienta de administración MQ JMS se debe haber configurado para acceder al espacio de nombres WebSphere™. Para obtener información más detallada sobre el modo de configurar la herramienta de administración, consulte el apartado “Configuración para WebSphere” en la página 37.

Para configurar los objetos administrados con los valores por omisión habituales, puede entrar el mandato siguiente para ejecutar el script `admin.scp`:

```
JMSAdmin < admin.scp
```

Si ya ha ejecutado admin.scp para configurar objetos para sample1 o sample2, al ejecutar admin.scp para sample3 se visualizan mensajes de error. (Se producen al intentar crear los subcontextos de jms y Samples). Puede ignorar estos mensajes de error sin problema.

Además, antes de ejecutar sample3, debe asegurarse de que el intermediario de publicación/suscripción MQSeries® (SupportPac MA0C) está instalado y en ejecución.

En el directorio sample3 hay un programa cliente simple, Sample3Client.java, para invocar al bean EJB. También se ha incluido un script, runClient, con el fin de simplificar la ejecución de este programa. Los puede utilizar siguiendo el procedimiento que se ha descrito en el apartado “Ejemplo 1” en la página 379.

Apéndice F. Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en EE.UU. Puede que en otros países IBM no ofrezca los productos, servicios o características que se describen en esta información. Consulte con el representante local de IBM para obtener información sobre los productos y servicios disponibles actualmente en su área. Las referencias a programas, productos o servicios de IBM no pretenden establecer ni implicar que sólo puedan utilizarse los productos, programas o servicios de IBM. En su lugar se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no infrinja ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, la evaluación y verificación del funcionamiento de cualquier producto, programa o servicio que no sea de IBM son responsabilidad del usuario.

IBM puede tener patentes o solicitudes de patentes pendientes que cubran el tema principal descrito en esta información. La entrega de esta información no le otorga ninguna licencia sobre dichas patentes. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
EE.UU.

Para consultas sobre licencias relacionadas con información de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM de su país o envíe sus consultas, por escrito, a:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokio 106, Japón

El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país donde estas disposiciones sean incompatibles con la legislación vigente:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN NINGÚN TIPO DE GARANTÍA, NI EXPLÍCITA NI IMPLÍCITA, INCLUIDAS, PERO NO LIMITÁNDOSE A, LAS GARANTÍAS IMPLÍCITAS DE NO VULNERACIÓN, COMERCIALIZACIÓN O DE ADECUACIÓN A UN PROPÓSITO DETERMINADO. Algunos países no permiten la renuncia a garantías explícitas o implícitas en determinadas transacciones y, por lo tanto, esta declaración puede que no se aplique a su caso.

Esta información puede incluir imprecisiones técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; estos cambios se incorporarán en nuevas ediciones de la información. IBM puede realizar en cualquier momento mejoras y/o cambios en el (los) producto(s) y/o programa(s) descrito(s) en esta información sin previo aviso.

Las referencias contenidas en esta información a sitios web que no sean de IBM sólo se proporcionan por comodidad y en ningún modo constituyen una aprobación de dichos sitios web. Los materiales de dichos sitios web no forman parte de los materiales para este producto de IBM y el uso de dichos sitios web corre a cuenta y riesgo del usuario.

IBM puede utilizar o distribuir la información que se le proporciona en la forma que considere adecuada, sin incurrir por ello en ninguna obligación para con el remitente.

Los titulares de licencias de este programa que deseen información sobre el mismo con el fin de permitir: (i) el intercambio de información entre programas creados independientemente y otros programas (incluido éste) y (ii) la utilización mutua de la información intercambiada, deben ponerse en contacto con

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Dicha información puede estar disponible, sujeta a los términos y condiciones adecuados, incluyendo, en algunos casos, el pago de unos derechos.

El programa bajo licencia descrito en esta información y todo el material bajo licencia disponible para el mismo los proporciona IBM bajo los términos del Acuerdo de cliente de IBM, Acuerdo internacional de licencia de programación de IBM o cualquier acuerdo equivalente entre IBM y el cliente.

La información relacionada con productos que no son de IBM se ha obtenido de los proveedores de dichos productos, sus anuncios publicados o de otras fuentes disponibles públicamente. IBM no ha probado estos productos y no puede confirmar la precisión de su rendimiento, compatibilidad o cualquier otra declaración relacionada con los productos que no son de su propiedad. Las preguntas relacionadas con las posibilidades de los productos que no son de IBM se deben dirigir a los proveedores de dichos productos.

Marcas registradas

Los términos siguientes son marcas registradas de International Business Machines Corporation en los Estados Unidos y/o en otros países:

AIX	AS/400	BookManager
CICS	IBM	IBMLink
Language Environment	MQSeries	MVS/ESA
OS/2	OS/390	OS/400
SecureWay	SupportPac	System/390
S/390	VisualAge	VSE/ESA
WebSphere		

Java, HotJava, JDK y todas las marcas registradas y los logotipos basados en Java son marcas registradas o marcas comerciales registradas de Sun Microsystems, Inc. en los Estados Unidos y/o en otros países.

Microsoft, Windows y Windows NT son marcas registradas de Microsoft Corporation en los Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en los Estados Unidos y en otros países.

Otros nombres de empresas, productos y servicios pueden ser marcas registradas o marcas de servicio de otras compañías.

Glosario de términos y abreviaturas

Este glosario describe términos utilizados en este manual y palabras utilizadas con un significado distinto del habitual. En algunos casos, una definición puede no ser la única aplicable a un término, pero proporciona un sentido determinado con el que se utiliza la palabra en este manual.

Si no encuentra el término que está buscando, consulte el índice o la publicación *IBM® Dictionary of Computing*, New York: McGraw-Hill, 1994.

Abstract Window Toolkit para Java™ (AWT). Colección de componentes de interfaz gráfica de usuario (GUI) que se implementan utilizando versiones de plataforma nativa de los componentes.

API. Interfaz de programas de aplicación

applet. Programa Java™ que está diseñado para ejecutarse sólo en una página web.

AWT. Abstract Window Toolkit para Java™.

canal. Véase canal MQI.

canal MQI. Un canal MQI conecta un cliente MQSeries a un gestor de colas de un sistema servidor y transfiere las invocaciones y respuestas MQI de un modo bidireccional.

clase. Una clase es una colección encapsulada de datos y métodos para operar en los datos. Se puede convertir en instancia para producir un objeto que sea una instancia de la clase.

cliente. En MQSeries®, un cliente es un componente de ejecución que proporciona acceso a los servicios de gestión de colas de un servidor para aplicaciones de usuario locales.

cola. Una cola es un objeto MQSeries. Las aplicaciones de gestión de colas de mensajes pueden transferir mensajes a una cola y obtener mensajes de la misma.

cola de mensajes. Véase cola.

conversión. Término que se utiliza en Java™ para describir la conversión explícita del valor de un tipo de primitivo u objeto en otro tipo.

Descriptor de mensajes de MQSeries (MQMD). Información de control que describe las propiedades y el formato del mensaje, que se lleva a cabo como parte del mensaje de MQSeries.

EJB. Enterprise JavaBeans.

encapsulación. La encapsulación es una técnica de programación orientada a objetos que hace que los datos de un objeto sean privados o estén protegidos y sólo permite a los programadores acceder a los datos y manipularlos mediante invocaciones de método.

Enterprise JavaBeans (EJB). Arquitectura de componentes del área del servidor que distribuye Sun Microsystems para crear aplicaciones portátiles de empresa y lógica de negocio reutilizable. Los componentes de enterprise JavaBeans se crean enteramente en Java y se ejecutan en cualquier servidor compatible con EJB.

gestión de colas de mensajes. Técnica de programación en la que cada programa de una aplicación se comunica con los demás programas poniendo mensajes en colas.

gestor de colas. Un gestor de colas es un programa del sistema que proporciona servicios de gestión de colas de mensajes a las aplicaciones.

HTML. Hypertext Markup Language (Lenguaje de marcación de hipertexto)

Hypertext Markup Language (Lenguaje de marcación de hipertexto)(HTML). Lenguaje que se utiliza para definir información que debe visualizarse en la World Wide Web.

IEEE. Institute of Electrical and Electronics Engineers.

IIOP. Internet Inter-ORB Protocol.

instancia. Una instancia es un objeto. Cuando una clase se convierte en instancia para producir un objeto, decimos que el objeto es una instancia de la clase.

interfaz. Una interfaz es una clase que sólo contiene métodos abstractos y ninguna variable de instancia. Una interfaz proporciona un conjunto común de métodos que pueden ser implementados por subclases de varias clases diferentes.

Interfaz de programas de aplicación (API). Una interfaz de programas de aplicación consta de las funciones y variables que pueden utilizar los programadores en sus aplicaciones.

Internet. Internet es una red cooperativa pública de información compartida. Físicamente, Internet utiliza un subconjunto de los recursos totales de todas las redes de telecomunicaciones públicas existentes actualmente. Técnicamente, lo que distingue a Internet como red cooperativa pública es su uso de un conjunto

de protocolos denominados TCP/IP (Transmission Control Protocol/Internet Protocol).

Internet Inter-ORB Protocol (IIOP). Estándar para las comunicaciones TCP/IP entre ORB de proveedores diferentes.

JAAS. Java Authentication and Authorization Service.

Java Authentication and Authorization Service (JAAS). Servicio de Java que proporciona control de acceso y autentificación de entidad.

Java Developers Kit (JDK). Paquete de software distribuido por Sun Microsystems para los programadores de Java. Incluye el intérprete Java, las clases Java y las herramientas de desarrollo de Java: compilador, depurador, desensamblador, visor de applets, generador de archivos de apéndice y generador de documentación.

Java Naming and Directory Service (JNDI). API especificada en el lenguaje de programación de Java. Proporciona funciones de directorio y denominación para las aplicaciones creadas en el lenguaje de programación de Java.

Java 2 Platform, Enterprise Edition (J2EE). Conjunto de servicios, API y protocolos que proporciona las funciones necesarias para desarrollar aplicaciones basadas en web de varias capas.

Java Runtime Environment (JRE). Subconjunto de JDK (Java Development Kit) que contiene los archivos y los ejecutables centrales que constituyen la plataforma Java estándar. JRE incluye Java Virtual Machine, clases centrales y archivos de soporte.

Java Transaction API (JTA). API que permite que las aplicaciones y los servidores J2EE accedan a las transacciones.

Java Transaction Service (JTS). Gestor de transacciones que ofrece soporte para JTA e implementa la correlación Java de la especificación OMG (Object Transaction Service) 1.1 por debajo del nivel de la API.

Java Virtual Machine (JVM). Implementación de software de una unidad central de proceso (CPU) que ejecuta código Java compilado (applets y aplicaciones).

JDK. Java Developers Kit.

J2EE. Java 2 Platform, Enterprise Edition.

JMS. Servicio de mensajes de Java.

JNDI. Java Naming and Directory Service.

JRE. Java Runtime Environment.

JTA. Java Transaction API.

JTS. Java Transaction Service.

JVM. Java Virtual Machine.

LDAP. Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP). Protocolo cliente/servidor para acceder al servicio de directorios.

Localizador de recursos uniforme (URL). Secuencia de caracteres que representa recursos de información en un sistema o en una red como, por ejemplo, Internet.

Mandatos MQSeries (MQSC). Mandatos legibles, uniformes a través de todas las plataformas, que se utilizan para manipular objetos de MQSeries.

mensaje. En aplicaciones de gestión de colas de mensajes, un mensaje es una comunicación enviada entre programas.

método. Método es el término de programación orientado a objetos para una función o un procedimiento.

MQDLH. Cabecera de mensajes no entregados de MQSeries. Consulte *MQSeries® Application Programming Reference*.

MQMD. MQSeries Message Descriptor (descriptor de mensajes de MQSeries).

MQSC. Mandatos MQSeries.

MQSeries. MQSeries es una familia de programas bajo licencia de IBM que proporcionan servicios de gestión de colas de mensajes.

navegador de la web. Programa que formatea y visualiza información que se distribuye en la World Wide Web.

Object Management Group (OMG). Consorcio que establece los estándares en la programación orientada al objeto.

Object Request Broker (ORB). Infraestructura de aplicaciones que proporciona interoperatividad entre objetos, creados en lenguajes diferentes, que se ejecutan en máquinas diferentes, en entornos distribuidos heterogéneos.

objeto. (1) En Java, un objeto es una instancia de una clase. Una clase forma un grupo de elementos; un objeto es un miembro determinado de dicho grupo. (2) En MQSeries, un objeto es un gestor de colas, una cola o un canal.

OMG. Object Management Group.

ORB. Object Request Broker.

paquete. En Java un paquete es un modo de proporcionar a una parte del código Java acceso a un conjunto específico de clases. El código Java que forma

parte de un paquete determinado tiene acceso a todas las clases del paquete y a todos los métodos y campos no privados de las clases.

privado. Un campo privado no es visible fuera de su propia clase.

protegido. Un campo protegido sólo es visible dentro de su propia clase, dentro de una subclase o dentro de paquetes de los que la clase forma parte.

público. Una clase o interfaz pública (public class) es visible en todas partes. Un método público o variable es visible en todas partes en las que es visible su clase.

Red Hat Package Manager (RPM). Sistema de empaquetado de software para utilizarlo en plataformas Red Hat Linux y en otra plataformas Linux y UNIX.

RPM. Red Hat Package Manager.

Servicio de mensajes de Java (JMS). API de Sun Microsystem para acceder a sistemas de envío de mensajes de empresa desde programas Java.

servidor. (1) Un servidor MQSeries es un gestor de colas que proporciona servicios de gestión de colas de mensajes a aplicaciones cliente que se ejecutan en una estación de trabajo remota. (2) De forma más general, un servidor es un programa que responde a peticiones de información en el modelo determinado de cliente/servidor de flujo de información entre dos programas. (3) Sistema en el que se ejecuta el programa servidor.

servlet. Programa Java que se ha diseñado para ejecutarse sólo en un servidor web.

sobrecarga. Situación en la que un identificador hace referencia a varios elementos del mismo ámbito. En Java, se puede sobrecargar los métodos, pero no las variables o los operadores.

subclase. Una subclase es una clase que expande otra. La subclase adopta los métodos públicos y protegidos y las variables de su superclase.

superclase. Una superclase es una clase que se expande mediante otra clase. Los métodos públicos y protegidos y las variables de la superclase están disponibles para la subclase.

TCP/IP. Transmission Control Protocol/Internet Protocol (Protocolo de control de transmisión/Protocolo Internet).

Transmission Control Protocol/Internet Protocol (TCP/IP). Conjunto de protocolos de comunicaciones que soportan funciones de conectividad de igual a igual para redes locales y redes de área amplia.

URL. Uniform Resource Locator (Localizador de recursos uniforme).

VisiBroker para Java. Un ORB (Object Request Broker) escrito en Java.

Web. Véase World Wide Web.

World Wide Web (Web). La World Wide Web es un servicio de Internet, basado en un conjunto común de protocolos, que permite a un sistema servidor especialmente configurado distribuir documentos en Internet de un modo estándar.

Glosario

Bibliografía

En este apartado se describe la documentación disponible para todos los productos MQSeries® actuales.

Publicaciones de MQSeries para varias plataformas

La mayor parte de estas publicaciones, a las que a veces de hace referencia como manuales de la "familia" MQSeries, se aplican a todos los productos MQSeries de Nivel 2. Los productos MQSeries de Nivel 2 más recientes son los siguientes:

- MQSeries para AIX, V5.2
- MQSeries para AS/400, V5.2
- MQSeries para AT&T GIS UNIX, V2.2
- MQSeries para Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries para Compaq Tru64 UNIX, V5.1
- MQSeries para HP-UX, V5.2
- MQSeries para Linux, V5.2
- MQSeries para OS/2 Warp, V5.1
- MQSeries para OS/390, V5.2
- MQSeries para SINIX y DC/OSx, V2.2
- MQSeries para Sun Solaris, V5.2
- MQSeries para Sun Solaris, Intel Platform Edition, V5.1
- MQSeries para Tandem NonStop Kernel, V2.2.0.1
- MQSeries para VSE/ESA, V2.1.1
- MQSeries para Windows, V2.0
- MQSeries para Windows, V2.1
- MQSeries para Windows NT y Windows 2000, V5.2

Las publicaciones de MQSeries® para varias plataformas son las siguientes:

- *MQSeries® Brochure*, G511-1908
- *Introducción a mensajes y colas*, GC10-9570
- *MQSeries® Intercommunication*, SC33-1872
- *MQSeries® Queue Manager Clusters*, SC34-5349
- *MQSeries® Clientes*, GC10-9654
- *MQSeries® Administración del sistema*, SC10-3081
- *MQSeries® Consulta de mandatos MQSC*, SC10-9438
- *MQSeries® Event Monitoring*, SC34-5760
- *MQSeries® Programmable System Management*, SC33-1482
- *MQSeries® Interfaz de administración. Guía de programación y consulta*, SC10-3342
- *MQSeries® Mensajes*, GC10-3078
- *MQSeries® Application Programming Guide*, SC33-0807
- *MQSeries® Application Programming Reference*, SC33-1673
- *MQSeries® Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries® Using C++*, SC33-1877
- *MQSeries® Utilización de Java™*, SC10-3345
- *MQSeries® Application Messaging Interface*, SC34-5604

Publicaciones de MQSeries específicas de las plataformas

Cada producto MQSeries® tiene, como mínimo, una publicación específica para la plataforma, además de los manuales de la familia MQSeries®.

MQSeries para AIX, V5.2

MQSeries para AIX Guía rápida de iniciación, GC10-3073

MQSeries para AS/400, V5.2

MQSeries® for AS/400® Guía rápida de iniciación, GC10-3410

MQSeries® para AS/400® Administración del sistema Versión 5.1, SC10-3411

MQSeries® para AS/400® Consulta de programación de aplicaciones (ILE RPG), SC34-5559

MQSeries para AT&T GIS UNIX, V2.2

MQSeries® for AT&T GIS UNIX® System Management Guide, SC33-1642

MQSeries para Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries® para Compaq (DIGITAL) OpenVMS Guía de gestión del sistema, GC10-3089

MQSeries para Compaq Tru64 UNIX, V5.1

MQSeries para Compaq Tru64 UNIX Guía rápida de iniciación, GC10-3468

MQSeries para HP-UX, V5.2

MQSeries para HP-UX Guía rápida de iniciación, GC10-3075

MQSeries para Linux, V5.2

MQSeries para Linux Guía rápida de iniciación, GC10-3591

MQSeries para OS/2 Warp, V5.1

MQSeries para OS/2 Warp Guía rápida de iniciación, GC10-3074

MQSeries para OS/390, V5.2

MQSeries® for OS/390® Concepts and Planning Guide, GC34-5650

MQSeries® for OS/390® System Setup Guide, SC34-5651

MQSeries® for OS/390® System Administration Guide, SC34-5652

MQSeries® for OS/390® Problem Determination Guide, GC34-5892

MQSeries® for OS/390® Messages and Codes, GC34-5891

MQSeries® for OS/390® Licensed Program Specifications, GC34-5893

MQSeries® for OS/390® Program Directory

MQSeries® link para R/3, Versión 1.2

MQSeries® link for R/3 Guía del usuario, GC10-3257

MQSeries para SINIX y DC/OSx, V2.2

MQSeries® for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries para Sun Solaris, V5.2

MQSeries para Sun Solaris Guía rápida de iniciación, GC10-3076

MQSeries para Sun Solaris, Intel Platform Edition, V5.1

MQSeries para Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries para Tandem NonStop Kernel, V2.2.0.1

MQSeries® para Tandem NonStop Kernel Guía de gestión del sistema, GC10-3147

MQSeries para VSE/ESA, V2.1.1

MQSeries® for VSE/ESA™ Licensed Program Specifications, GC34-5365

MQSeries® for VSE/ESA® System Management Guide, GC34-5364

MQSeries para Windows, V2.0

MQSeries® para Windows® Guía del usuario

MQSeries para Windows, V2.1

MQSeries® para Windows® Guía del usuario

MQSeries para Windows NT y Windows 2000, V5.2

MQSeries para Windows NT Guía rápida de iniciación, GC10-3341

MQSeries® para Windows NT® Utilización de la interfaz de modelo de objetos de componentes, SC10-3344

MQSeries® LotusScript Extension, SC10-3343

Publicaciones en copia software

La mayoría de los manuales de MQSeries se proporcionan en formato de copia impresa y en formato de copia software.

Formato HTML

La documentación de MQSeries® más importante se proporciona en formato HTML con los productos MQSeries® siguientes:

- MQSeries para AIX, V5.2
- MQSeries para AS/400, V5.2
- MQSeries para Compaq Tru64 UNIX, V5.1
- MQSeries para HP-UX, V5.2
- MQSeries para Linux, V5.2
- MQSeries para OS/2 Warp, V5.1
- MQSeries para OS/390, V5.2
- MQSeries para Sun Solaris, V5.2
- MQSeries para Sun Solaris, Intel Platform Edition, V5.1
- MQSeries para Windows NT y Windows 2000, V5.2 (HTML compilado)
- MQSeries® link for R/3, V1.2

Las publicaciones de MQSeries® también se pueden obtener en formato HTML en el sitio web de la familia de productos MQSeries®, en la dirección siguiente:

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

PDF (Portable Document Format)

Los archivos PDF se pueden ver e imprimir utilizando Adobe Acrobat Reader.

Si necesita obtener Adobe Acrobat Reader, o si desea información actualizada sobre las plataformas que disponen de soporte para Acrobat Reader, visite el sitio web de Adobe Systems Inc., en la dirección siguiente:

```
http://www.adobe.com/
```

Se proporcionan las versiones en formato PDF de las publicaciones de MQSeries® más importantes con los productos MQSeries® siguientes:

- MQSeries para AIX, V5.2
- MQSeries para AS/400, V5.2
- MQSeries para Compaq Tru64 UNIX, V5.1
- MQSeries para HP-UX, V5.2
- MQSeries para Linux, V5.2
- MQSeries para OS/2 Warp, V5.1
- MQSeries para OS/390, V5.2
- MQSeries para Sun Solaris, V5.2
- MQSeries para Sun Solaris, Intel Platform Edition, V5.1
- MQSeries para Windows NT y Windows 2000, V5.2
- MQSeries® link for R/3, V1.2

Las versiones en formato PDF de todas las publicaciones de MQSeries® actuales también se pueden obtener en el sitio web de la familia de productos MQSeries®, en la dirección siguiente:

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

Formato BookManager®

La biblioteca MQSeries® se proporciona en formato IBM® BookManager® en varios kits de colecciones de bibliotecas en línea, incluido el kit de la colección *Transaction Processing and Data*, SK2T-0730. Puede visualizar las publicaciones en copia software en formato IBM® BookManager® utilizando los programas bajo licencia IBM® siguientes:

BookManager® READ/2
BookManager® READ/6000
BookManager® READ/DOS
BookManager® READ/MVS
BookManager® READ/VM
BookManager® READ for Windows®

Formato PostScript

La biblioteca de MQSeries® se proporciona en formato PostScript (.PS) con muchos productos MQSeries® Versión 2. Los manuales en formato PostScript pueden imprimirse en una impresora PostScript o visualizarse con un visor adecuado.

Formato Windows®

La publicación *MQSeries® para Windows® Guía del usuario* se proporciona en formato de ayuda de Windows® con MQSeries® para Windows®, Versión 2.0 y MQSeries® paraWindows®, Versión 2.1.

Información MQSeries disponible en Internet

El sitio web de la familia de productos MQSeries® está en la dirección siguiente:

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

Siguiendo los enlaces de este sitio web, puede:

- Obtener la información más actualizada acerca de la familia de productos MQSeries®.
- Acceder a las publicaciones de MQSeries® en los formatos HTML y PDF.
- Bajar un MQSeries® SupportPac™.

Índice

CH

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Envío de comentarios a IBM

Si ha habido algún aspecto de este manual que le ha agradado o desagradado especialmente, utilice uno de los métodos que se indican a continuación para enviar sus comentarios a IBM®.

No dude en comentar todo aquello que considere errores u omisiones específicos, así como lo que afecte a la precisión, organización, tema tratado o integridad del manual.

Limite sus comentarios a la información contenida en esta publicación y a la forma en que está presentada.

Si desea hacer comentarios sobre las funciones de productos o sistemas de IBM®, póngase en contacto con su representante de IBM® o con su concesionario autorizado de IBM®.

Al enviar comentarios a IBM®, otorga a IBM® un derecho no exclusivo para utilizar o distribuir sus comentarios en la forma que considere adecuada, sin incurrir por ello en ninguna obligación para con el remitente.

Puede enviar sus comentarios a IBM® utilizando cualquiera de los procedimientos siguientes:

- Por correo, a la dirección:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- Por fax:
  - Fuera del Reino Unido, después del código de acceso internacional, debe marcar 44–1962–870229
  - En el Reino Unido, utilice el número 01962–870229
- Electrónicamente, utilizando el ID de red adecuado:
  - IBM Mail Exchange: GBIBM2Q9 en IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Cualquiera que sea el método que utilice, asegúrese de que incluye:

- El título de la publicación y el número de pedido
- El tema al que se aplica su comentario
- Su nombre y dirección/número de teléfono/número de fax/ID de red.

Printed in Denmark by IBM Danmark A/S.

SC10-3345-01