9.1

为 *IBM MQ* 开发应用程序

注

在使用本资料及其支持的产品之前，请阅读第 1167 页的『声明』中的信息。

本版本适用于 IBM® MQ V 9 发行版 1 以及所有后续发行版和修订版，直到在新版本中另有声明为止。

内容

开发应用程序.....5
应用程序开发概念.....7
应用程序可执行的操作.....7
应用程序、应用程序名称和应用程序实例.....9
使用 MQI 的应用程序.....10
使用客户机连接来连接到多个 IBM MQ 队列管理器.....10
面向对象的应用程序.....13
IBM MQ 消息.....15
准备和运行 Microsoft Transaction Server 应用程序.....40
IBM MQ 应用程序的设计注意事项.....40
以受支持的编程语言指定应用程序名称.....43
消息的设计方法.....48
应用程序设计和性能注意事项.....49
高级应用程序的设计方法.....50
IBM i 应用程序的设计和性能注意事项.....52
Linux on POWER Systems - Little Endian 应用程序的设计注意事项.....53
z/OS 应用程序的设计和性能注意事项.....53
IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序.....56
开发 JMS 和 Java 应用程序.....66
使用 IBM MQ classes for JMS.....67
使用 IBM MQ classes for Java.....281
使用 IBM MQ 资源适配器.....366
将 IBM MQ 与 WebSphere Application Server 结合使用.....413
使用 IBM MQ 头包.....427
使用 Java 和 JMS 在 IBM i 上设置 IBM MQ.....430
使用 Maven 存储库的 Java 应用程序开发.....436
开发 C++ 应用程序.....437
C++ 样本程序.....440
C++ 语言注意事项.....444
C++ 中的消息传递.....447
构建 IBM MQ C++ 程序.....453
开发 .NET 应用程序.....463
开始使用 IBM MQ classes for .NET.....464
编写和部署 IBM MQ .NET 程序.....480
开发 XMS .NET 应用程序.....511
XMS 支持的消息传递样式.....511
XMS 对象模型.....512
XMS 消息模型.....514
设置消息传递服务器环境.....514
使用 XMS 样本应用程序.....522
编写 XMS 应用程序.....524
编写 XMS .NET 应用程序.....540
使用 XMS .NET 受管对象.....544
防止应用程序使用较新的 XMS 版本.....551
确保 XMS 应用程序安全通信.....551
XMS 消息.....554
使用组件对象模型接口 (IBM MQ Automation Classes for ActiveX).....561
使用 IBM MQ Automation Classes for ActiveX 进行设计和编程.....562
IBM MQ Automation Classes for ActiveX 参考.....567
跟踪 IBM MQ Automation Classes for ActiveX.....632
到 MQAI 的 ActiveX 接口.....636
关于 IBM MQ Automation Classes for ActiveX 启动程序样本.....644

开发 AMQP 客户机应用程序.....648
MQ Light 和 AMQP（高级消息排队协议）.....649
AMQP 1.0 支持.....650
映射 AMQP 和 IBM MQ 消息字段.....651
具有 AMQP 的消息传递可靠性.....657
用于将 AMQP 客户机与 IBM MQ 配合使用的拓扑.....658
使用 IBM MQ 开发 REST 应用程序.....664
使用 REST API 进行消息传递.....665
使用 IBM MQ 开发 MQT 应用程序.....669
IBM MQ 数据定义文件.....670
编写用于排队的过程应用程序.....672
编写客户机过程应用程序.....834
用户出口、API 出口和 IBM MQ 可安装服务.....853
构建过程应用程序.....908
处理过程程序错误.....947
多点广播编程.....951
采用 C 进行编码.....956
在 Visual Basic 中编码.....959
采用 COBOL 进行编码.....959
采用 System/390 汇编语言（消息队列接口）进行编码.....960
采用 RPG 对 IBM MQ 程序进行编码（仅限 IBM i）.....962
采用 PL/I 编码（仅限 z/OS）.....963
使用 IBM MQ 样本过程程序.....963
为 Managed File Transfer 开发应用程序.....1104
指定要使用 MFT 运行的程序.....1104
将 Apache Ant 与 MFT 结合使用.....1106
使用用户出口定制 MFT.....1109
通过将消息放置在代理命令队列中来控制 MFT.....1121
为 MQ Telemetry 开发应用程序.....1122
IBM MQ Telemetry Transport 样本程序.....1122
MQTT 客户机编程概念.....1123
使用 IBM MQ 开发 Microsoft Windows Communication Foundation 应用程序.....1141
将适用于 WCF 的 IBM MQ 定制通道与 .NET 相结合简介.....1141
使用适用于 WCF 的 IBM MQ 定制通道.....1145
使用 WCF 样本.....1160

声明.....1167
编程接口信息.....1168
商标.....1168

为 IBM MQ 开发应用程序

您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

首次为 IBM MQ 开发应用程序?

要了解为 IBM MQ 开发应用程序的相关信息，请访问 IBM Developer：

- LearnMQ （学习基础知识、运行演示、编写应用程序代码、学习更多高级教程）
- MQ 开发者下载（包括免费的开发者版本和试用版本）

如果您熟悉以下部分中所描述的概念，那么可能会发现开发应用程序很容易：

- 第 7 页的『应用程序开发概念』
- 第 40 页的『IBM MQ 应用程序的设计注意事项』

支持面向对象的语言和框架

IBM MQ 为使用以下语言和框架开发的应用程序提供了核心支持：

- JMS
- Java
- C++
- .NET
- ActiveX（不推荐；使用 .NET）

另请参阅第 13 页的『面向对象的应用程序』。

.NET 支持使用多种语言开发的应用程序。 为解释如何使用 IBM MQ classes for .NET 来访问 IBM MQ 队列，MQ 产品文档分别为以下语言提供了相应的信息：

- C# 示例代码和样本应用程序
- C++ 样本应用程序
- Visual Basic 样本应用程序

请参阅 第 480 页的『编写和部署 IBM MQ .NET 程序』。

V 9.1.1 V 9.1.2 IBM MQ 支持 .NET Core for applications in Windows environment from IBM MQ 9.1.1 and for applications in Linux® environment from IBM MQ 9.1.2。 有关更多信息，请参阅第 465 页的『安装 IBM MQ classes for .NET Standard』。

ULW IBM MQ 还支持 MQ Light API，此 API 实现了 OASIS AMQP 1.0 协议。 其中包含对应于以下语言的消息传递 API：

- Node.js
- Ruby
- Java
- Python
- Maven（框架项目；使用 Java API）
- Gradle（框架项目；使用 Java API）

另请参阅第 648 页的『开发 AMQP 客户机应用程序』。

按原状提供以下语言绑定：

- Go 绑定

- 用于 Node.js 应用程序的 JavaScript API 实现

支持编程 REST API

IBM MQ 支持使用以下编程 REST API 来发送和接收消息：

- V 9.1.0 IBM MQ messaging REST API
- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM DataPower 网关

请参阅第 664 页的『使用 IBM MQ 开发 REST 应用程序』，以及 IBM Developer 的 IBM MQ 区域中的教程：Get started with the IBM MQ messaging REST API。 本教程包含以下语言示例（按现状提供），可配合 IBM MQ messaging REST API 一起使用：

- 使用 MQ 消息传递 REST API 的 Go 示例
- 使用 HTTPS 模块的 Node.js 示例
- 使用 Promise 模块的 Node.js 示例

支持过程化编程语言

IBM MQ 为使用以下过程化编程语言开发的应用程序提供支持：

- C
- Windows Visual Basic（仅限 Windows 系统）
- COBOL
- z/OS 汇编语言（仅限 IBM MQ for z/OS）
- IBM i RPG（仅限 IBM MQ for IBM i）
- z/OS PL/I（仅限 IBM MQ for z/OS）

这些语言使用消息队列接口 (MQI) 来访问消息排队服务。 请参阅第 669 页的『使用 IBM MQ 开发 MQT 应用程序』。 请注意，面向对象的语言和框架所使用的 IBM MQ 对象模型提供了一些其他功能，这些功能不可用于使用 MQI 的过程化语言。

指定应用程序名称

ULW V 9.1.2

在 IBM MQ 9.1.2 之前，您可以在 Java 或 JMS 客户机应用程序上指定应用程序名称。 从 IBM MQ 9.1.2 开始，您还可以在其他编程语言上指定应用程序名称。 有关更多信息，请参阅第 43 页的『以受支持的编程语言指定应用程序名称』。

相关任务

第 1122 页的『为 MQ Telemetry 开发应用程序』

第 1141 页的『使用 IBM MQ 开发 Microsoft Windows Communication Foundation 应用程序』
IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 定制通道在 WCF 客户机和服务之间发送和接收消息。

相关参考

第 1104 页的『为 Managed File Transfer 开发应用程序』
指定要随 Managed File Transfer 一起运行的程序，将 Apache Ant 用于 Managed File Transfer，使用用户出口定制 Managed File Transfer，并通过在代理命令队列上放置消息来控制 Managed File Transfer。

应用程序开发概念

您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

有关可为 IBM MQ 编写的应用程序类型的信息，请参阅第 5 页的『为 IBM MQ 开发应用程序』和第 7 页的『应用程序可执行的操作』。

相关概念

第 40 页的『IBM MQ 应用程序的设计注意事项』
当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

应用程序可执行的操作

您可以开发一个应用程序，用以发送和接收支持业务流程所需的消息。 您也可以开发用于管理队列管理器和相关资源的应用程序。

应用程序可以在 IBM MQ for Multiplatforms 上执行的操作

Multi

在 多平台 上，您可以编写执行以下操作的应用程序：

- 向同一操作系统下运行的其他应用程序发送消息。 这些应用程序可以位于同一系统，也可以位于其他系统。
- 向其他 IBM MQ 平台上运行的应用程序发送消息。
- 在以下系统的 CICS 中使用消息排队：
  - AIX TXSeries for AIX
  - IBM i IBM i
  - Solaris Solaris
  - Windows Windows
- 在以下系统的 Encina 中使用消息排队：
  - AIX AIX
  - Solaris Solaris
  - Windows Windows
- 在以下系统的 Tuxedo 中使用消息排队：
  - AIX AIX
  - AT&T
  - Solaris Solaris
  - Windows Windows
- 将 IBM MQ 用作事务管理器，协调外部资源管理器在 IBM MQ 工作单元内所做的更新。 以下外部资源管理器受支持并符合 X/OPEN XA 接口
  - Db2
  - Informix
  - Oracle
  - Sybase
- 将多个消息作为可以提交或取消的单个工作单元一起处理。

- 从完全 IBM MQ 环境中运行，或从 IBM MQ 客户机环境运行。

应用程序可以在 IBM MQ for z/OS 上执行的操作

z/OS

在 z/OS 上，您可以编写执行以下操作的应用程序：

- 在 CICS 或 IMS 中使用消息队列。
- 在批处理程序、CICS 和 IMS 应用程序之间发送消息，为每个功能选择最适合的环境。
- 向其他 IBM MQ 平台上运行的应用程序发送消息。
- 将多个消息作为可以提交或取消的单个工作单元一起处理。
- 通过 IMS 网桥向 IMS 应用程序发送消息并与之进行交互。
- 参与由 RRS 协调的工作单元。

z/OS 中的每个环境都有自己的特征、优点和缺点。 IBM MQ for z/OS 的优点是应用程序不依赖于任何一个环境，但可以分发以利用每个环境的优势。 例如，您可以使用 TSO 或 CICS 开发最终用户接口，可以在 z/OS 批处理中运行处理密集型模块，也可以在 IMS 或 CICS 中运行数据库应用程序。 在任何情况下，应用程序的各个部分都可以使用消息和队列进行通信。

IBM MQ 应用程序的设计人员必须了解这些环境的区别及其限制。 例如：

- IBM MQ 提供了可在队列管理器（称为分布式队列）之间相互通信的工具。
- 提交和取消更改的方式在批处理和 CICS 环境之间存在差别。
- IBM MQ for z/OS 在 IMS 环境中为在线消息处理程序 (MPP)、交互式快速路径程序 (IFP) 和成批消息处理程序 (BMP) 提供支持。 如果要编写批处理 DL/I 程序，请遵循 第 935 页的『构建 z/OS 批处理应用程序』和 第 682 页的『z/OS 批处理注意事项』 for z/OS 批处理程序之类的主题中提供的指导。
- 虽然 IBM MQ for z/OS 的多个实例可以存在于单个 z/OS 系统上，但一个 CICS 区域一次只能连接到一个队列管理器。 但是，多个 CICS 区域可以连接到同一队列管理器。 在 IMS 和 z/OS 批处理环境中，程序可以连接到多个队列管理器。
- IBM MQ for z/OS 允许一组队列管理器共享本地队列，提高了吞吐量和可用性。 此类队列称为共享队列，而这些队列管理器形成一个队列共享组，该组可以在相同的共享队列中处理消息。 通过指定队列共享组名称而非特定队列管理器名称，批处理应用程序可以连接到队列共享组中其中一个队列管理器。 这称为组批量连接，或简称为组连接。 请参阅共享队列和队列共享组。

z/OS 第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』中进一步说明了受支持环境之间的区别及各个环境的限制。

相关概念

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

第 40 页的『IBM MQ 应用程序的设计注意事项』
当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

第 672 页的『编写用于排队的过程应用程序』
使用此信息来了解有关编写排队应用程序、连接和断开队列管理器、发布/预订以及打开和关闭对象的信息。

第 834 页的『编写客户机过程应用程序』
使用过程语言在 IBM MQ 上编写客户机应用程序时需要知道的内容。

第 67 页的『使用 IBM MQ classes for JMS』
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

第 561 页的『使用组件对象模型接口 (IBM MQ Automation Classes for ActiveX)』
IBM MQ Automation Classes for ActiveX (MQAX) 是一些 ActiveX 组件，这些组件提供了一些类，您可以在应用程序中使用这些类来访问 IBM MQ。

第 281 页的『使用 IBM MQ classes for Java』

在 Java 环境中使用 IBM MQ 。 IBM MQ classes for Java 允许 Java 应用程序作为 IBM MQ 客户机连接到 IBM MQ，或直接连接到 IBM MQ 队列管理器。

第 463 页的『开发 .NET 应用程序』
IBM MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 IBM MQ MQI client 连接到 IBM MQ 或直接连接到 IBM MQ 服务器。

第 437 页的『开发 C++ 应用程序』
IBM MQ 提供等同于 IBM MQ 对象的 C++ 类，以及等同于数组数据类型的一些其他类。 它提供许多不能通过 MQI 使用的功能。

第 908 页的『构建过程应用程序』
您可以使用若干过程语言之一编写 IBM MQ 应用程序，并在几个不同平台上运行该应用程序。

相关任务

第 963 页的『使用 IBM MQ 样本过程程序』
这些样本程序采用过程语言编写，并演示了消息队列接口 (MQI) 的典型用法。 IBM MQ 在不同平台上编程。

第 1141 页的『使用 IBM MQ 开发 Microsoft Windows Communication Foundation 应用程序』
IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 定制通道在 WCF 客户机和服务之间发送和接收消息。

保护

Multi 应用程序、应用程序名称和应用程序实例

在开始设计和编写应用程序之前，请先熟悉应用程序、应用程序名称和应用程序实例的基本概念。

应用程序

Multi

如果到队列管理器的连接提供同一个应用程序名称，那么将这些连接视为来自同一个应用程序。 应用程序名称将显示为 DISPLAY CONN(*) TYPE CONN 命令的 APPLTAG 属性。

注意:

1. 对于使用版本低于 IBM MQ 9.1.2 的 IBM MQ client 的应用程序，应用程序名称由 IBM MQ client 自动设置。 其值取决于应用程序编程语言和用于运行应用程序的平台。 有关更多信息，请参阅 PutApplName。
2. V 9.1.2 如果 IBM MQ client 应用程序使用的 IBM MQ client 版本为 IBM MQ 9.1.2 或更高版本，那么可以将应用程序名称设置为特定值。 在大多数情况下，这无需更改应用程序代码，也无需重新编译应用程序。 请参阅在支持的编程语言中使用应用程序名称以获取更多信息。

应用程序实例

Multi

连接将进一步细分为应用程序实例。 应用程序的实例是一组紧密相关的连接，它们为该应用程序提供一个 "执行单元"。 通常，这是一个单一的操作系统进程，可以具有多个线程和关联的 IBM MQ 连接。

在 IBM MQ for Multiplatforms 上，应用程序实例与特定连接标记相关联。 队列管理器在发现新连接与现有应用程序实例存在关联时会自动将这两者相关联。

注意:

- 如果使用客户机连接，那么这些进程可能通过一个或多个正在运行的通道来连接到队列管理器。
- 在 JMS 应用程序中，应用程序实例会映射到特定 JMS 连接和所有关联的 JMS 会话。

在 IBM MQ for Multiplatforms 上，使用统一集群自动应用程序均衡时，应用程序实例尤其重要。 在 IBM MQ for Multiplatforms 平台上，您可以使用 DISPLAY APSTATUS 命令查看当前连接的应用程序实例。

在某些情况下，队列管理器无法正确地将连接与应用程序实例相关联，尤其是在以下情况下:

- 通过同一个进程在共享对话上使用不同应用程序名称建立了多个连接时。
- 正在使用较低级别的客户机库时。 例如，在 IBM MQ 9.1.2 和更低版本上安装 IBM MQ JMS 客户机。

在此类情况下，如果应用程序未将自身定义为可重新连接，那么将允许执行此操作，但某些应用程序实例分组可能不正确。 如果有任一连接声明为 MQCNO_RECONNECT，那么这会对应用程序均衡产生极大的负面影响；由此会导致 MQCONN 调用被拒绝并返回 MQCNO_RECONNECT_INCOMPATIBLE。

相关概念

第 43 页的『以受支持的编程语言指定应用程序名称』
在 IBM MQ 9.1.2 之前，您已经可以在 Java 或 JMS 客户机应用程序上指定应用程序名称。 从 IBM MQ 9.1.2 开始，将此功能扩展至 IBM MQ for Multiplatforms 上的其他编程语言。

使用 MQI 的应用程序

IBM MQ 应用程序需要特定对象才能成功运行。

第 10 页的图 1 显示某个应用程序从队列除去消息，并处理它们，然后将结果发送至同一队列管理器上的另一个队列。

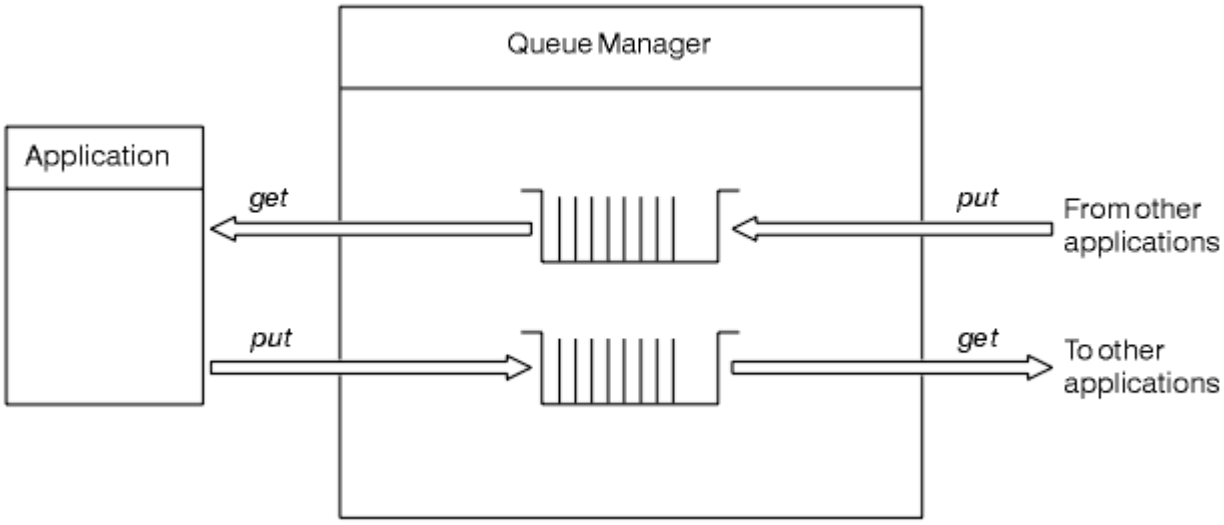


图 *1:* 队列、消息和应用程序

尽管应用程序可以将消息放入本地或远程队列（使用 MQPUT），但它们只能直接从本地队列获取消息（使用 MQGET）。

在此应用程序可以运行之前，必须满足以下条件：

- 队列管理器必须存在并正在运行。
- 必须定义将从其除去消息的第一个应用程序队列。
- 还必须定义应用程序放入消息所在的第二个队列。
- 应用程序必须能连接到队列管理器。 为此，应用程序必须链接到 IBM MQ。 请参阅第 908 页的『构建过程应用程序』。
- 将消息放入第一个队列的应用程序还必须连接到队列管理器。 如果这些应用程序是远程的，还必须为它们设置传输队列和通道。 第 10 页的图 1 中没有显示系统的这一部分。

使用客户机连接来连接到多个 IBM MQ 队列管理器

了解如何使用客户机通道定义表 (CCDT)、连接名称列表（CONNAME 列表）、负载均衡和代码存根来连接到多个队列管理器，并了解每个选项在满足特定需求方面的优缺点。

在下文中：

CCDT 多队列管理器

这是一个 CCDT 文件，其中包含多个具有相同组（即队列管理器名称客户机连接 (QMNAME CLNTCONN) 属性）的客户机连接 (CLNTCONN) 通道，其中不同的 CLNTCONN 条目会解析为不同的队列管理器。

这与包含多个 CLNTCONN 条目的 CCDT 文件不同，后者中的 CLNTCONN 条目只是同一多实例队列管理器（可以选择将此方法与代码存根结合使用）的不同 IP 地址或主机名。

如果您确实选择了 CCDT 多队列管理器方法，那么需要选择是划分条目优先级还是使用随机工作负载管理 (WLM)：

划分优先级
使用多个按字母顺序排序的条目以及 CLNTWGHT(1) 和 AFFINITY(PREFERRED) 属性来记住最后一个正常连接。

随机
使用 CLNTWGHT(1) 和 AFFINITY(NONE) 属性。 可通过调整 CLNTWGHT 来调整不同规模的 IBM MQ 服务器上的 WLM 权重

注: 您应尽量缩小各通道的 CLNTWGHT 差异。

负载均衡器
这是一个网络设备，它具有虚拟 IP 地址 (VIP)，并且配置了对多个 IBM MQ 队列管理器的 TCP/IP 侦听器进行端口监视。 在网络设备中配置 VIP 的方式取决于您所使用的网络设备。

以下选项仅与发送消息或启动同步请求和回复消息传递的应用程序相关。 在“将消息侦听器连接到队列”（待定）中详细讨论了可处理这些消息和请求的应用程序的注意事项（例如，侦听器完全分离）

连接到单个队列管理器的现有应用程序所需的代码更改规模

CONNAME 列表、CCDT 多队列管理器和负载均衡器
MQCONN("QMNAME") 到 MQCONN("*QMNAME")

队列管理器名称可能位于 Java Enterprise Edition (EE) 应用程序的 Java 命名和目录接口 (JNDI) 配置中。 否则，这需要进行一次单字符代码更改。

这可能会对代码产生正面影响。

代码存根
用代码存根替换现有的 JMS 或 MQI 连接逻辑。

这可能会对代码产生负面影响。

支持不同的 WLM 策略

CONNAME 列表
仅限“划分优先级”。

这可能会对代码产生负面影响。

CCDT 多队列管理器
“划分优先级”或“随机”。

这不太可能对代码有任何影响。

负载均衡器
任何（包括适用于所有消息的每一个连接）。

这可能会对代码产生正面影响。

代码存根
任何（包括适用于所有消息的每一条消息）。

这可能会对代码产生正面影响。

主队列管理器不可用时的性能开销

CONNAME 列表
始终先在列表中尝试。

这可能会对代码产生负面影响。

CCDT 多队列管理器
记住最后一个正常连接。

这可能会对代码产生正面影响。

负载均衡器
端口监视可防止使用错误的队列管理器。

这可能会对代码产生正面影响。

代码存根
可记住最后一个正常连接，然后智能地进行重试。

这可能会对代码产生正面影响。

XA 事务支持

CONNAME 列表、CCDT 多队列管理器和负载均衡器
事务管理器需要存储用于重新连接到相同队列管理器资源的恢复信息。

解析为不同队列管理器的 MQCONN 调用通常会使此信息无效。 例如，在 Java EE 中，使用 XA 时，单个连接工厂应解析为单个队列管理器。

这可能会对代码产生负面影响。

代码存根
代码存根可满足事务管理器的 XA 需求（例如，多个连接工厂）。

这可能会对代码产生正面影响。

支持不同的 WLM 策略

CONNAME 列表
仅限 DNS。

这可能会对代码产生负面影响。

CCDT 多队列管理器
DNS 和共享文件系统、共享文件系统或 CCDT 文件推送。

这不太可能对代码有任何影响。

负载均衡器
动态虚拟 IP 地址 (VIP)。

这可能会对代码产生正面影响。

代码存根
DNS 或单个队列管理器 CCDT 条目。

这不太可能对代码有任何影响。

避免计划维护期间出现中断

您还需要考虑并规划另一种情况，即如何在队列管理器的计划维护期间避免应用程序中断，例如，向最终用户显示的错误和超时。 避免中断的最佳方法是在停止队列管理器前移除其中的所有工作。

考虑一个请求/应答场景。 您希望所有未完成的请求都会完成，并且由应用程序处理应答，但是不希望将任何其他工作提交到系统。 仅停顿队列管理器并不能满足此需求，因为正确编码的应用程序在收到未完成请求的回复消息之前，会收到返回码 RC2161 MQRC_Q_MGR_QUIESCING 异常。

您可以在用于提交工作的请求队列上设置 PUT(DISABLED)，并在回复队列上保留 PUT(ENABLED) 和 GET(ENABLED)。 通过这种方式，您可以监视请求、传输和回复队列的深度。 在全部稳定（也就是，未完成的请求都已完成或超时）后，便可以停止队列管理器。

但是，必须在请求应用程序中进行正确编码才能处理 PUT(DISABLED) 请求队列，这将在尝试发送消息时产生返回码 RC2051 MQRC_PUT_INHIBITED 错误。

请注意，在建立到 IBM MQ 的连接或者打开请求队列时，不会出现该异常。 仅当尝试实际使用 MQPUT 调用来发送消息时才会出现此异常。

通过构建包含这种用于请求和应答场景的错误处理逻辑的代码存根，并让应用程序团队在以后使用此类代码存根，可以帮助开发具备一致行为的应用程序。

面向对象的应用程序

IBM MQ 提供了对 JMS， Java， C++， .NET 和 ActiveX 的支持。 这些语言和框架使用 IBM MQ 对象模型，它提供的类与 IBM MQ 调用和结构提供相同的功能。

使用 IBM MQ 对象模型的某些语言和框架提供其他函数，在将过程语言和消息队列接口 (MQI) 结合使用时，这些函数不可用。

有关此模型提供的类、方法和属性的详细信息，请参阅 第 13 页的『IBM MQ 对象模型』。

JMS

IBM MQ 提供用于实现 Java Message Service (JMS) 规范的类。 有关 IBM MQ classes for JMS 的详细信息，请参阅使用 IBM MQ classes for JMS。 有关 IBM MQ classes for Java 和 IBM MQ classes for JMS 之间的差异的信息（用于帮助决定使用哪种类），请参阅第 66 页的『开发 JMS 和 Java 应用程序』。

IBM Message Service Client for C/C++ 和 IBM Message Service Client for .NET 提供名为 XMS 的应用程序编程接口 (API)，其接口集与 Java Message Service (JMS) 相同 API。 有关更多信息，请参阅第 511 页的『开发 XMS .NET 应用程序』。

Java

请参阅 使用 IBM MQ classes for Java，以获取有关使用 Java 中的 IBM MQ 对象模型对程序进行编码的信息。 IBM 不会对 IBM MQ classes for Java 进行进一步增强，并且它们在功能上已稳定在 IBM MQ 8.0 中提供的级别。 有关 IBM MQ classes for Java 和 IBM MQ classes for JMS 之间差异的信息（有助于确定使用哪种），请参阅第 66 页的『开发 JMS 和 Java 应用程序』。

C++

IBM MQ 提供等同于 IBM MQ 对象的 C++ 类，以及等同于数组数据类型的一些其他类。 它提供许多不能通过 MQI 使用的功能。 有关使用 IBM MQ 对象模型编写 C++ 程序的信息，请参阅 使用 C++。C/C++ 和 .NET 的消息服务客户机提供了一个称为 XMS 的应用程序编程接口 (API)，此接口与 Java Message Service (JMS) API 具有相同的接口集。

.NET

请参阅 开发 .NET 应用程序，以获取有关使用 IBM MQ .NET 类对 .NET 程序进行编码的信息。 针对 C/C++ 和 .NET 的消息服务客户机提供与 Java Message Service (JMS) API 具有相同接口集的名为 XMS 的应用程序编程接口 (API)。

ActiveX

IBM MQ ActiveX 通常称为 MQAX。 MQAX 包含作为 IBM MQ for Windows 的一部分。 对 ActiveX 的支持已稳定在 IBM WebSphere MQ 6.0 级别。 有关使用 IBM MQ ActiveX 中的对象模型 使用组件对象模型接口 (WebSphere MQ Automation Classes for ActiveX)对程序进行编码的信息。

从 IBM MQ 9.0 开始，不推荐使用对 Microsoft Active X 的 IBM MQ 支持。 IBM MQ classes for .NET 是建议使用的替代技术。 有关更多信息，请参阅开发 .NET 应用程序。

相关概念

第 669 页的『使用 IBM MQ 开发 MQT 应用程序』
IBM MQ 提供了对 C、Visual Basic、COBOL、Assembler、RPG、pTAL 和 PL/I 的支持。 这些过程语言使用消息队列接口 (MQI) 来访问消息排队服务。

技术概述
第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

相关参考

应用程序开发参考

IBM MQ 对象模型

IBM MQ 对象模型由类、方法和属性组成。

IBM MQ 对象模型包括：

- 类 表示熟悉的 IBM MQ 概念，例如队列管理器，队列和消息。
- 与 MQI 调用对应的每个类上的方法。

- 对应于 IBM MQ 对象的属性的每个类上的 属性 。

使用 IBM MQ 对象模型创建 IBM MQ 应用程序时，将在应用程序中创建这些类的实例。 面向对象的编程中的类的实例称为对象。 创建对象后，可以通过检查或设置对象属性的值（相当于发出 MQINQ 或 MQSET 调用）以及通过针对对象进行方法调用（相当于发出其他 MQI 调用）来与该对象交互。

类

IBM MQ 对象模型提供类的以下基本集。

模型的实际实施在所支持的不同面向对象的环境之间略有变化。

MQQueueManager
MQQueueManager 类的对象表示与队列管理器的连接。 它具有 Connect()、Disconnect()、Commit() 和 Backout() 的方法（相当于 MQCONN 或 MQCONNX、MQDISC、MQCMIT 和 MQBACK）。 它具有与队列管理器的属性对应的属性。 访问队列管理器属性会隐式连接到队列管理器（如果还未连接）。 销毁 MQQueueManager 对象会隐式与队列管理器断开连接。

MQQueue
MQQueue 类的对象表示队列。 它具有从队列 Put() 和 Get() 消息的方法（相当于 MQPUT 和 MQGET）。 它具有与队列的属性对应的属性。 访问队列属性特性或者发出 Put() 或 Get() 方法调用会隐式打开队列（相当于 MQOPEN）。 销毁 MQQueue 对象会隐式关闭队列（相当于 MQCLOSE）。

MQTopic
MQTopic 类的对象表示主题。 它具有从主题 Put()（发布）和 Get()（接收或预订）消息的方法（相当于 MQPUT 和 MQGET）。 它具有与主题的属性对应的属性。 只能为发布或预订目的访问 MQTopic 对象，不能同时为两种目的进行访问。 当用于接收消息时，MQTopic 对象可以使用非受管或受管预订进行创建并创建为持久或非持久订户 - 对于这些不同方案提供了多个超负荷构造方法。

MQMessage
MQMessage 类的对象表示要放到队列或从队列获取的消息。 它包含缓冲区，并且封装应用程序数据和 MQMD。 它具有与 MQMD 字段对应的属性，以及使您能够向缓冲区写入和从中读取不同类型（例如、字符串、长整数、短整数、单字节）的用户数据的方法。

MQPutMessageOptions
MQPutMessageOptions 类的对象表示 MQPMO 结构。 它具有与 MQPMO 字段对应的属性。

MQGetMessageOptions
MQGetMessageOptions 类的对象表示 MQGMO 结构。 它具有与 MQGMO 字段对应的属性。

MQProcess
MQProcess 类的对象表示进程定义（通过触发进行使用）。 它具有表示进程定义的属性的属性。

MQDistributionList
MQDistributionList 类的对象表示分发列表（用于通过单个 MQPUT 发送多条消息）。 它包含 MQDistributionListItem 对象的列表。

MQDistributionListItem
MQDistributionListItem 类的对象表示单个分发列表目标。 它封装 MQOR、MQRR 和 MQPMR 结构，并且具有与这些结构的字段对应的属性。

对象引用

在使用 MQI 的 IBM MQ 程序中，IBM MQ 将连接句柄和对象句柄返回到该程序。

这些句柄必须在后续 IBM MQ 调用中作为参数进行传递。 通过 IBM MQ 对象模型，可以从应用程序隐藏这些句柄。 而从类创建对象会导致将对象引用返回到应用程序。 这是在针对对象进行方法调用和属性访问时所使用的对象引用。

返回码

发出方法调用或设置属性值会导致设置返回码。

这些返回码是完成代码或原因码，并且其本身是对象的属性。 完成代码和原因码的值与为 MQI 定义的值相同，其中一些额外的值特定于面向对象的环境。

IBM MQ 消息

IBM MQ 消息由消息属性和应用程序数据组成。 消息队列消息描述符 (MQMD) 包含消息在发送和接收应用程序之间传输时应用程序数据随附的控制信息。

消息的组成部分

IBM MQ 消息由两部分组成：

- 消息属性
- 应用程序数据

第 15 页的图 2 对消息进行了描述并显示了消息如何从逻辑上分为消息属性和应用程序数据。

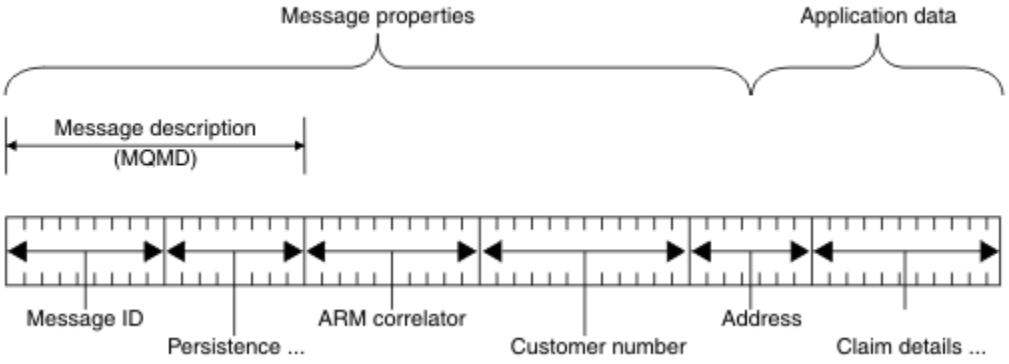


图 *2:* 消息说明

队列管理器不会更改 IBM MQ 消息中携带的应用程序数据，除非 IBM MQ 上实行数据转换。 此外，IBM MQ 对这些数据的内容没有任何限制。 每个消息中的数据长度不能超过队列和队列管理器的 **MaxMsgLength** 属性的值。

ULW 在 UNIX, Linux, and Windows 上，队列管理器和队列的 *MaxMsgLength* 属性缺省为 4 MB（4 194 304 字节），如果需要，此值最大可以更改为 100 MB（104 857 600 字节）。

IBM i 在 IBM i 上，队列管理器和队列的 *MaxMsgLength* 属性缺省为 4 MB（4 194 304 字节），如果需要，此值最大可以更改为 100 MB（104 857 600 字节）。 如果您打算在 IBM i 上使用大于 15 MB 的 IBM MQ 消息，请参阅 第 913 页的『在 IBM i 上构建过程化应用程序』。

z/OS 在 z/OS 上，队列管理器的 **MaxMsgLength** 属性固定为 100 MB，而队列的 **MaxMsgLength** 属性缺省为 4 MB（4 194 304 字节），此值最大可以更改为 100 MB（如果需要）。

某些情况下，请让消息稍短于 **MaxMsgLength** 属性的值。 有关更多信息，请参阅 第 705 页的『消息中的数据』。

在使用 MQI 或 MQPUT1 MQI 调用时创建一条消息。 作为这些调用的输入，您可以提供控制信息（如消息的优先级和应答队列的名称）和数据，然后调用会将消息放入队列。 请参阅 MQPUT 和 MQPUT1 以了解有关这些调用的更多信息。

消息描述符

您可以使用 MQMD 结构访问消息控制信息，该结构定义了消息描述符。

有关 MQMD 结构的完整描述，请参阅 MQMD - 消息描述符。

有关如何在包含原始消息相关信息的 MQMD 中使用这些字段的描述，请参阅第 39 页的『消息上下文』。

消息描述符有几种不同版本。 第二版的消息描述符（或 MQMDE）提供了分组和分段消息（请参阅第 36 页的『消息组』）的其他信息。 这与第一版消息描述符相同，但新增了额外字段。 MQMDE - 消息描述符扩展中对这些字段进行了描述。

消息类型

IBM MQ 定义了四种消息类型。

这四种消息为：

- 数据报
- 请求消息
- 应答消息
- 报告消息
  - 报告消息类型
  - 报告消息选项

应用程序可以使用前三种消息互相传递信息。 第四种类型（报告）供应用程序和队列管理器用于报告有关事件（如发生错误）的信息。

每种消息类型由 MQMT_* 值进行标识。 您也可以定义您自己的消息类型。 要获取您可以使用的值范围，请参阅 MsgType。

数据报

不要求接收消息（从队列取出消息）的应用程序进行应答时使用数据报。

可以使用数据报的应用程序示例：候机室用于显示航班信息的应用程序。 消息中可以包含用于整个航班信息屏幕的数据。 此类应用程序与请求消息确认不同，因为即使消息不发送也没什么关系。 应用程序很快会发送更新消息。

请求消息

如果您需要接收消息的应用程序进行应答，请使用请求消息。

可以使用请求消息的应用程序示例：用于显示支票帐户余额的应用程序。 请求消息中会包含帐号，而应答消息中将包含帐户余额。

如果您想将应答消息链接到请求消息，可以使用以下两个选项：

- 让处理请求消息的应用程序负责确保将信息放入与请求消息相关的应答消息内。
- 使用请求消息的消息描述符中的报告字段指定应答消息的 *MsgId* 和 *CorrelId* 字段内容。
  - 您可以请求将原始消息的 *MsgId* 或 *CorrelId* 复制到应答消息中的 *CorrelId* 字段（缺省操作为复制 *MsgId*）。
  - 您可以请求为应答消息生成新 *MsgId*，或将原始消息的 *MsgId* 复制到应答消息的 *MsgId* 字段（缺省操作为生成新消息标识）。

应答消息

应答另一条消息时，请使用应答消息。

在创建应答消息时，请保留要应答的消息的消息描述符内已设置的所有选项。 报告选项用于指定消息标识 (*MsgId*) 和相关标识 (*CorrelId*) 字段的内容。 这些字段能让接收应答的应用程序将应答关联到原始请求。

报告消息

报告消息用于将处理消息时发生错误之类的事件通知给应用程序。

此类消息可通过以下方式生成：

- 队列管理器，
- 消息通道代理程序（例如，如果消息通道代理程序无法发送消息）或
- 应用程序（例如，如果应用程序无法使用消息中的数据）。

报告消息随时都可能生成，也可能在应用程序未预料到的情况下到达队列。

报告消息类型

在队列上放入消息时，您可以选择接收以下项：

- 异常报告消息。 回应具有异常标志设置的消息时会发送此消息。 此消息由消息通道代理程序 (MCA) 或应用程序生成。
- 到期报告消息。 此消息表明应用程序尝试检索已达到其到期阈值的消息；此消息标记为废弃。 这种报告类型由队列管理器生成。
- 到达确认 *(COA)* 报告消息。 此消息表明消息已到达目标队列。 该消息由队列管理器生成。
- 发送确认 *(COD)* 报告消息。 此消息表明接收应用程序已检索到消息。 该消息由队列管理器生成。
- 肯定操作通知 *(PAN)* 报告消息。 此消息表明请求已成功处理（即，已成功执行消息中请求的操作）。 这种报告类型由应用程序生成。
- 否定操作通知 *(NAN)* 报告消息。 此消息表明请求未成功处理（即，没有成功执行消息中请求的操作）。 这种报告类型由应用程序生成。

注: 每种报告消息类型包含下列其中一项：

- 整条原始消息
- 原始消息中数据的前 100 个字节
- 没有来自原始消息的数据

在将消息放在队列上时，您可以请求多种报告消息类型。 当您选择发送确认报告消息和异常报告消息选项时，如果消息无法发送，您将收到异常报告消息。 但是，如果您只选择发送确认报告消息，但消息无法发送，您不会收到异常报告消息。

当满足生成特定消息的条件时，您请求的报告消息只是您接收的消息。

报告消息选项

您可以在出现异常后废弃消息。 如果您选择废弃选项并请求了异常报告消息，那么报告消息将转至 *ReplyToQ* 和 *ReplyToQMgr*，同时废弃原始消息。

注: 此选项的好处是您可以减少转至死信队列的消息数量。 但是，这并不表示您的应用程序必须处理返回的消息，除非它只发送数据报消息。 生成异常报告消息时，它会继承原始消息的持久性。

如果报告消息无法发送（例如，队列已满），报告消息将放到死信队列中。

如果您想接收报告消息，请在 *ReplyToQ* 字段中指定应答队列的名称；否则，原始消息的 MQPUT 或 MQPUT1 将失败，并附有 MQRC_MISSING_REPLY_TO_Q。

您可以使用消息的消息描述符 (MQMD) 中的其他报告选项指定为此消息创建的任何报告消息的 *MsgId* 和 *CorrelId* 字段的内容：

- 您可以请求将原始消息的 *MsgId* 或 *CorrelId* 复制到报告消息的 *CorrelId* 字段。 缺省操作为复制消息标识。 将使用 MQRO_COPY_MSG_ID_TO_CORRELID，因为它能使消息发送者将应答消息或报告消息关联到原始消息。 应答消息或报告消息的相关标识与原始消息的消息标识相同。
- 您可以请求为报告消息生成新 *MsgId*，或将原始消息的 *MsgId* 复制到报告消息的 *MsgId* 字段。 缺省操作为生成新消息标识。 将使用 MQRO_NEW_MSG_ID，因为它可以确保系统中的每个消息都有不同的消息标识，并可以明确地与系统中的其他所有消息区分开来。
- 专用的应用程序可能需要使用 MQRO_PASS_MSG_ID 或 MQRO_PASS_CORREL_ID。 但是，您需要设计从队列中读取消息的应用程序以确保其在某些情况下正常运作，例如，队列中包含多条带有相同消息标识的消息。

  服务器应用程序必须检查这些标志在请求消息中的设置，并相应地设置应答消息或报告消息中的 *MsgId* 和 *CorrelId* 字段。

  充当请求者应用程序和服务器应用程序之间中介的应用程序不需要检查这些标志的设置。 这是因为这些应用程序通常需要将消息转发到未更改过 *MsgId*、*CorrelId* 和 *Report* 字段的服务器应用程序。 这使得服务器应用程序能够将 *MsgId* 从原始消息复制到应答消息的 *CorrelId* 字段。

在生成关于消息的报告时，服务器应用程序必须进行测试以确认是否设置了以上任一选项。

有关如何使用报告消息的更多信息，请参阅报告。

为了体现报告的性质，队列管理器将使用一系列反馈代码。 队列管理器将这些代码放入报告消息的消息描述符中的 *Feedback* 字段。 队列管理器还可以将 MQI 原因码返回在 *Feedback* 字段中。 IBM MQ 定义了一系列反馈代码供应用程序使用。

有关反馈和原因码的更多信息，请参阅反馈。

可以使用反馈代码的程序示例：监控服务于队列的其他程序的工作负载的程序。 如果有多个程序实例服务于一个队列，且从该队列上收到的消息数量来看已无需如此，此程序将向其中一个服务程序发送一条报告消息（带有反馈代码 MQFB_QUIT），指示该程序应终止其活动。 （监控程序可以使用 MQINQ 调用了解有多少程序正在服务于此队列。）

Multi 报告和分段消息

IBM MQ for z/OS 上不支持。

如果消息已分段且您要求生成报告，那么相比于未分段消息，您可能会收到更多报告。

有关已分段消息的描述，请参阅第 733 页的『消息分段』。

对于 IBM MQ 生成的报告

如果您对消息进行分段或允许队列管理器对消息进行分段，那么只有一种情况可以接收整个消息的单一报告。 那就是当您只请求了 COD 报告，且已在获取应用程序上指定 MQGMO_COMPLETE_MSG 时。

其他情况下，您的应用程序必须准备处理多个报告，通常是每个分段一个报告。

注: 如果您对消息进行分段，并且只需要返回原始消息数据的前 100 个字节，请更改报告选项的设置，以请求不包含偏移量为 100 或以上的分段数据的报告。 如果不这样做，且保留此设置（这样每个分段将请求 100 字节的数据），并且使用单个 MQGET 检索报告消息（指定 MQGMO_COMPLETE_MSG），那么报告会组合为包含 100 字节读数据的大型消息，且每个报告采用相应的偏移量。 如果发生此情况，您需要大型缓冲区，或需要指定 MQGMO_ACCEPT_TRUNCATED_MSG。

应用程序生成的报告

如果您的应用程序生成报告，请将原始消息数据开头显示的 IBM MQ 头复制到报告消息数据。

然后向报告消息数据添加空内容、100 字节的原始消息数据或所有原始消息数据（或您通常包含的其他数据量）。

可以通过查看连续格式名称来识别必须复制的 IBM MQ 头，从 MQMD 开始直到显示的所有头。 以下 `Format` 名称指示这些 IBM MQ 头:

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* 表示以 MQH 字符开头的所有名称。

`Format` 名称出现在 MQDLH 和 MQXQH 的特定位置，但对于其他 IBM MQ 头，它出现在同一位置。 对于 MQMDE、MQIMS 和所有 MQH* 头，头的长度包含在还会在同一位置显示的字段内。

如果您使用的是第一版 MQMD，并且您要报告分段或报告组中的消息或报告允许分段的消息，那么报告数据必须以 MQMDE 开头。 将 *`OriginalLength`* 字段设置为原始消息数据的长度，不包括您查找的任何 IBM MQ 头的长度。

检索报告

如果要求获取 COA 或 COD 报告，您可以请求使用 MQGMO_COMPLETE_MSG 重新组合这两种报告。

以下情况满足带有 MQGMO_COMPLETE_MSG 的 MQGET：队列上有足够的报告消息（属于一种类型（例如 COA），并带有相同的 *GroupId*）来表示一条完整的原始消息时。 即使报告消息本身不包含完整原始消息，此情况依然成立；每条报告消息中的 *OriginalLength* 字段提供由此报告消息表示的原始数据的长度，即使数据本身不存在。

即使队列上存在多种不同的报告类型 (例如， COA 和 COD)，也可以使用此方法，因为具有 MQGMO_COMPLETE_MSG 的 MQGET 仅在消息具有相同的 *Feedback* 代码时才会重新组合报告消息。 但是，您通常无法将此技术用于异常报告，因为通常，这些报告具有不同的 *Feedback* 代码。

您可以使用此方法获取整个消息已到达的明确指示。 但是，在大多数情况中，您需要应对一部分分段到达，而另外一部分分段生成异常（或到期，如果您允许）的可能。 在这种情况下，无法使用 MQGMO_COMPLETE_MSG，因为通常，您可能会针对不同的段获得不同的 *Feedback* 代码，并且可能会针对一个段获得多个报告。 但您可以使用 MQGMO_ALL_SEGMENTS_AVAILABLE。

考虑到这点，您可能需要在收到报告时检索报告，然后在应用程序中分析原始消息所发生的状况。 您可以使用报告消息中的 *GroupId* 字段将报告与原始消息的 *GroupId* 相关联，并使用 *Feedback* 字段标识每条报告消息的类型。 执行此操作的方式取决于您的应用程序需求。

其中一种方式如下所示：

- 要求获取 COD 报告和异常报告。
- 经过特定时间后，使用 MQGMO_COMPLETE_MSG 检查是否已收到一组完整的 COD 报告。 如果已经收到，您的应用程序将知道整个消息已处理。
- 如果没有收到，将显示与此消息相关的异常报告，处理问题的方式与未分段消息一样，但会确保清除某些位置上的孤立分段。
- 如果某些分段没有任何类型的报告，原始分段（或报告）可能正在等待重新连接通道，或者是网络在某个时间点超载。 如果未收到任何异常报告（或者您认为收到的异常只是暂时的），您可能会决定让您的应用程序多等待一会儿。

  与之前一样，注意事项与处理未分段消息时类似，但必须同时考虑清除孤立分段的可能性。

如果原始消息不重要（例如，为查询或稍后可以重复的消息），请设置到期时间以确保移除孤立分段。

后备级别队列管理器

当报告由支持分段的队列管理器生成，但在不支持分段的队列管理器上接收时，除了消息中的零个字节、100 个字节或所有原始数据外，报告数据中将始终包含 MQMDE 结构（用于标识由报告表示的 *Offset* 和 *OriginalLength*）。

但是，如果消息分段通过不支持分段的队列管理器，并在其中生成了报告，原始消息中的 MQMDE 结构将纯粹地视为数据。 因此，如果请求零字节的原始数据，MQMDE 结构不会包含在报告数据中。 没有 MQMDE，报告消息可能不是很有用。

如果消息可能会通过后备级别队列管理器，请至少请求报告中 100 字节的数据。

消息控制信息和消息数据的格式

队列管理器只关注消息中控制信息的格式，然而，处理消息的应用程序关注于控制信息和数据的格式。

消息控制信息的格式

消息描述符的字符串字段中的控制信息必须在队列管理器使用的字符集中。

队列管理器对象的 **CodedCharSetId** 属性定义了此字符集。 控制信息必须在此字符集中，因为当应用程序将消息从一个队列管理器传递到另一个队列管理器时，传输这些消息的消息通道代理程序将使用此属性的值来确定执行哪些数据转换。

消息数据的格式

您可以指定以下任意项：

- 应用程序数据的格式

- 字符数据的字符集
- 数字数据的格式

要执行此操作，请使用以下字段：

格式
向消息接收者表明消息内应用程序数据的格式。

在队列管理器创建消息时，某些情况下会使用 *Format* 字段识别消息格式。 例如，当队列管理器无法发送消息时，会将消息放入死信（未发送消息）队列。 会在消息中添加一个头（包含更多控制信息），然后更改 *Format* 字段以显示此头。

队列管理器有很多名称以 MQ 开头的内置格式，例如，MQFMT_STRING。 如果这些格式不满足您的需求，您可以定义自己的格式（用户定义的格式），但不能为自己定义的格式使用以 MQ 开头的名称。

在创建和使用您自己的格式时，必须编写数据转换出口才能支持使用 MQGMO_CONVERT 获取消息的程序。

CodedCharSetId
此字段用于定义消息中字符数据的字符集。 如果您希望将字符集设置为队列管理器的字符集，可以将此字段设置为常量 MQCCSI_Q_MGR 或 MQCCSI_INHERIT。

从队列中获取消息时，将比较 *CodedCharSetId* 字段的值与应用程序需要的值。 如果两个值不同，您可能需要转换消息中的所有字符数据或使用数据转换消息出口（其中有可用出口）。

编码
此字段用于描述包含二进制整数、压缩十进制整数和浮点数字的数字消息数据的格式。 通常根据队列管理器运行所在的特定机器进行编码。

在队列上放入消息时，通常在 *Encoding* 字段中指定常量 MQENC_NATIVE。 这表示消息数据的编码与应用程序运行所在的机器的编码相同。

从队列中获取消息时，将比较消息描述符中 *Encoding* 字段的值与您机器上常量 MQENC_NATIVE 的值。 如果两个值不同，您可能需要转换消息中的所有数字数据或使用数据转换消息出口（其中有可用出口）。

应用程序数据转换
对于不同的平台来说，应用程序数据可能需要转换为另一个应用程序要求的字符集和编码。

数据可以在发送队列管理器上转换，也可以在接收队列管理器上转换。 如果内置的格式库不满足您的需要，您可以定义自己的格式。 转换类型取决于消息描述符 MQMD 的格式字段内指定的消息格式。

注: 不会转换指定了 MQFMT_NONE 的消息。

发送队列管理器上的转换

如果您需要让发送消息通道代理程序 (MCA) 转换应用程序数据，请将 CONVERT 通道属性设置为 YES。

将在发送队列管理器上为特定的内置格式和用户定义的格式（如果提供了合适的用户出口）执行转换。

内置格式

其中包括：

- 全部是字符的消息（使用格式名称 MQFMT_STRING）
- IBM MQ 定义的消息，例如可编程命令格式。

  IBM MQ 将可编程命令格式消息用于管理消息和事件（这种情况下使用的格式名称为 MQFMT_ADMIN）。 您可以为自己的消息使用相同的格式（使用格式名称 MQFMT_PCF），并充分利用内置数据转换。

队列管理器内置格式的名称全部以 MQFMT 开头。 格式中列出并描述了这些格式。

应用程序定义的格式

对于用户定义的格式，应用程序数据转换必须由数据转换出口程序执行（有关更多信息，请参阅第 893 页的『编写数据转换出口』）。 在客户机/服务器环境中，出口将加载到服务器并在服务器上执行转换。

接收队列管理器上的转换

接收队列管理器可以为内置和用户定义的格式转换应用程序消息数据。

如果您指定 MQGMO_CONVERT 选项，将在处理 MQGET 调用期间执行转换。 有关详细信息，请参阅选项

编码字符集

IBM MQ 产品支持底层操作系统提供的编码字符集。

在您创建队列管理器时，使用的队列管理器编码字符集标识 (CCSID) 基于底层环境的编码字符集标识。 如果是混合编码页，IBM MQ 将使用混合编码页的 SBCS 部分作为队列管理器 CCSID。

对于一般的数据转换，如果底层操作系统支持 DBCS 代码页，IBM MQ 可以使用 DBCS 代码页。

请参阅操作系统文档以获取有关系统支持的编码字符集的详细信息。

在编写跨多个平台的应用程序时，您需要考虑应用程序数据转换、格式名称和用户出口。 请参阅第 893 页的『编写数据转换出口』获取有关调用和编写数据转换出口的信息。

消息优先级

您可以将消息的优先级设置为数字值，或让消息采用队列的缺省优先级。

将消息放入队列时设置消息的优先级（在 MQMD 结构的 *Priority* 字段中）。 您可以设置优先级的数字值，或让消息采用队列的缺省优先级。

队列的 **MsgDeliverySequence** 属性决定了队列上的消息是按优先级序列内的 FIFO（先进先出）序列还是按 FIFO 进行存储。 如果此属性设置为 MQMDS_PRIORITY，消息将按照其消息描述符的 *Priority* 字段中指定的优先级排队；但如果设置为 MQMDS_FIFO，消息将按队列的缺省优先级排队。 优先级相同的消息按照到达顺序存储在队列上。

队列的 **DefPriority** 属性为要放入该队列的消息设置缺省优先级值。 该值在创建队列时设置，但之后可以更改。 别名队列和远程队列的本地定义可以与它们解析的基本队列有不同的缺省优先级。 如果解析路径中有多个队列定义（请参阅第 694 页的『名称解析』），那么缺省优先级取自开放命令中指定队列的 **DefPriority** 属性的值（在放入操作时）。

队列管理器的 **MaxPriority** 属性的值是您可以为该队列管理器处理的消息指定的最大优先级。 您无法更改此属性的值。 在 IBM MQ 中，该属性的值为 9；您可以创建优先级介于 0（最低）到 9（最高）之间的消息。

消息属性

使用消息属性可让应用程序选择要处理的消息，或在无需访问 MQMD 或 MQRFH2 头的情况下检索有关消息的信息。 这些属性也便于 IBM MQ 和 JMS 应用程序之间的通信。

消息属性是与消息关联的数据，由文本名称和特定类型的值组成。 消息选择器使用消息属性来过滤发布内容以获取主题或有选择性地从队列取出消息。 消息属性可用于包含业务数据或状态信息，同时无需存储在应用程序数据中。 应用程序不必访问 MQ 消息描述符 (MQMD) 或 MQRFH2 头中的数据，因为这些数据结构中的字段可以作为消息属性使用消息队列接口 (MQI) 函数调用来访问。

IBM MQ 中消息属性的使用方式模拟了 JMS 中的属性使用方法。 这意味着您可以在 JMS 应用程序中设置属性并在程序化 IBM MQ 应用程序中检索这些属性，反之亦然。 要使属性可用于 JMS 应用程序，请为其指定“usr”前缀；然后（不带前缀）便可以用作 JMS 消息用户属性。 例如， JMS 应用程序可以使用 JMS 调用 `message.getStringProperty('myproperty')` 来访问 IBM MQ 属性 *usr.myproperty* (字符串)。 请注意，如果属性包含两个或更多 U+002E（“.”）字符，那么 JMS 应用程序无法访问带有“usr”前缀的属性。 不带前缀且不包含 U+002E（“.”）字符的属性将视为与包含前缀“usr”一样。 相反，可以通过添加 "usr" 在 IBM MQ 应用程序中访问 JMS 应用程序中设置的用户属性。 在 MQINQMP 调用中查询的属性名的前缀。

消息属性和消息长度

使用队列管理器属性 *MaxPropertiesLength* 来控制可以随 IBM MQ 队列管理器中的任何消息流动的属性大小。

通常，在您使用 MQSETMP 设置属性时，属性的大小等于属性名称的长度（以字节计）加上传递到 MQSETMP 调用的属性值的长度（以字节计）。 因为字符集可以转换为 Unicode，所以在将消息传输到目标期间可以更改属性名称和属性值的字符集；这种情况下，属性的大小可能会更改。

在 MQPUT 或 MQPUT1 调用上，消息的属性不计入队列和队列管理器的消息的长度，不过它们计入被队列管理器认为是属性的长度中（无论是否使用消息属性 MQI 调用进行设置）。

如果属性的大小超过最大属性长度，那么将使用 MQRC_PROPERTIES_TOO_BIG 拒绝该消息。 因为属性大小取决于其表示法，因此应该设置总体上的最大属性长度。

如果缓冲区中包括属性，应用程序有可能会成功放入其缓冲区大于 *MaxMsgLength* 值的消息。 这是因为，即使表示为 MQRFH2 元素，消息属性也不会计入消息长度。 只有包含一个或多个文件夹且头中的每个文件夹都包含属性时，MQRFH2 头字段才会添加到属性长度中。 如果 MQRFH2 头中包含一个或多个文件夹并且所有文件夹都不包含属性，那么 MQRFH2 头字段会计入消息长度。

在 MQGET 调用上，就队列和队列管理器而言，消息属性不会计入消息长度。 但是，因为属性单独计算，所以 MQGET 调用返回的缓冲区有可能会大于 *MaxMsgLength* 属性的值。

调用 MQGET 之前，不要让您的应用程序查询 *MaxMsgLength* 的值并分配此大小的缓冲区；而是分配您认为足够大的缓冲区。 如果 MQGET 失败，请根据 *DataLength* 参数的大小分配缓冲区。

如果 MQGMO 结构中没有指定消息句柄，MQGET 调用的 *DataLength* 参数将返回应用程序数据和您提供的缓冲区中返回的任何属性的长度（以字节计）。

MQPUT 调用的 *Buffer* 参数包含要发送的应用程序消息数据和消息数据中存在的任何属性。

当消息流向 IBM WebSphere MQ 7.0 版本之前的队列管理器时，除了那些在消息描述符中的属性之外，消息属性将计入消息长度中。 因此，您应该根据需要提高到 IBM WebSphere MQ 7.0 之前的系统的通道的 *MaxMsgLength* 属性的值，以补偿可能针对每条消息发送更多数据的事实。 或者，您可以降低队列或队列管理器 *MaxMsgLength*，以便系统中要发送的数据的整体水平保持一致。

消息属性的长度限值是 100 MB，其中不包括每个消息的消息描述符或扩展。

消息内部表示的属性大小等于属性名称长度加上属性值大小，再加上属性的某些控制数据。 还有一些在一个属性添加到消息后对属性集的控制数据。

属性名称

属性名称是字符串。 其长度和可以使用的字符集存在一定限制。

属性名称是区分大小写的字符串，长度限制为 +4095 个字符，除非上下文另有限制。 该限制包含在 MQ_MAX_PROPERTY_NAME_LENGTH 常量中。

如果使用消息属性 MQI 调用时超出此最大长度，调用将失败并带有原因码 MQRC_PROPERTY_NAME_LENGTH_ERR。

因为 JMS 中没有最大属性名称长度，因此有可能 JMS 应用程序设置的有效 JMS 属性名称在存储到 MQRFH2 结构时不是有效的 IBM MQ 属性名称。

这种情况下，解析时只会使用属性名称的前 4095 个字符；后面的字符将截断。 因为多个属性可能会截断为同一名称，因此可能会导致使用选择器的应用程序无法匹配选择字符串，或在不希望匹配的情况下匹配字符串。 当属性名称截断时，WebSphereMQ 将发送错误日志消息。

除了允许 Unicode 字符 U+002E (.) 作为名称的一部分（但不能作为开头）外，所有属性名称都必须遵循针对 Java 标识的 Java 语言规范所定义的规则。 针对 Java 标识的规则等同于针对属性名称的 JMS 规范包含的规则。

禁止使用空格字符和比较运算符。 允许但不建议在属性名称中使用嵌入式 Null。 如果您使用嵌入式空值，当与 MQCHARV 结构一起用来指定变量长度字符串时将阻止使用 MQVS_NULL_TERMINATED 常量。

因为应用程序可以根据属性名称选择消息，并且名称和选择器的字符集之间的转换可能会导致选择意外失败，所以应保持属性名称尽量简单。

IBM MQ 属性名称使用字符 U+002E (.) 进行属性逻辑分组。 这可以分割属性的名称空间。 带有以下前缀的属性（大小写任意混合）将保留供产品使用：

- `mcd`
- `jms`

- usr
- mq
- sib
- wmq
- Root
- Body
- Properties

确保所有应用程序使用其互联网域名作为其消息属性前缀是一种避免名称冲突的好方法。 例如，如果您要开发使用域名 ourcompany.com 的应用程序，您可以以前缀 com.ourcompany 命名所有属性。 此命名约定还能让您更加简单的选择属性；例如，应用程序可以查询所有以 com.ourcompany.% 开头的消息属性。

请参阅属性名称限制以了解有关使用属性名称的更多信息。

属性名称限制
对属性命名时，必须遵循特定规则。

以下限制适用于属性名称：

1. 属性不能以下列字符串开头：
   - “JMS”- 保留供 IBM MQ classes for JMS 使用。
   - “usr.JMS”- 无效。

   唯一的例外是下列为 JMS 属性提供同义词的属性：

属性	同义词
JMSCorrelationID	Root .MQMD.CorrelId 或 jms.Cid
JMSDeliveryMode	Root .MQMD.Persistence 或 jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root .MQMD.Expiry 或 jms.Exp
JMSMessageID	Root .MQMD.MsgId
JMSPriority	Root .MQMD.Priority 或 jms.Pri
JMSRedelivered	Root .MQMD.BackoutCount
JMSReplyTo（以 URI 编码的字符串）	Root.MQMD.ReplyToQ、Root.MQMD.ReplyToQMgr 或 jms.Rto
JMSTimestamp	Root.MQMD.PutDate、Root.MQMD.PutTime 或 jms.Tms
JMSType	mcd.Type、mcd.Set 或 mcd.Fmt
JMSXAppID	Root.MQMD.PutApplName
JMSXDeliveryCount	Root .MQMD.BackoutCount
JMSXGroupID	Root.MQMD.GroupId 或 jms.Gid
JMSXGroupSeq	Root.MQMD.MsgSeqNumber 或 jms.Seq
JMSXUserID	Root.MQMD.UserIdentifier

这些同义词允许 MQI 应用程序以类似于 IBM MQ classes for JMS 客户机应用程序的方式访问 JMS 属性。 至于这些属性，只有 JMSCorrelationID、JMSReplyTo、JMSType、JMSXGroupID 以及 JMSXGroupSeq 能够使用 MQI 设置。

请注意，IBM MQ classes for JMS 中可用的 JMS_IBM_* 属性不能使用 MQI 访问。 MQI 应用程序可以通过其他方式访问 JMS_IBM_* 属性引用的字段。

2. 不能调用的属性（大小写任意混合）包括“NULL”、“TRUE”、“FALSE”、“NOT”、“AND”、“OR”、“BETWEEN”、“LIKE”、“IN”、“IS”和“ESCAPE”。 这些是选择字符串中使用的 SQL 关键字的名称。
3. 以“mq”（大小写任意混合）开头但不是“mq_usr”开头的属性名称只能包含一个“.” 字符 (U+002E)。 具有这些前缀的属性中 不允许使用多个“.”字符。
4. 两个“.” 字符之间必须包含其他字符；层次结构中不能有空白点。 同样，属性名称不能以“.” 字符结尾。
5. 如果应用程序设置了属性“a.b”，然后设置了属性“a.b.c”，那么层次结构“b”中包含值还是包含另一个逻辑组并不明确。 这种层次结构为“混合内容”，不受支持。 不允许设置会产生混合内容的属性。

如下所示，这些限制由验证机制强行执行：

- 如果创建消息句柄时要求验证，使用 MQSETMP-设置消息属性 调用设置属性时会对属性名称进行验证。 如果尝试验证属性时因为属性名称规范中的错误而失败，完成代码为 MQCC_FAILED，原因码如下：
  - MQRC_PROPERTY_NAME_ERROR 代表原因 1-4。
  - MQRC_MIXED_CONTENT_NOT_ALLOWED 代表原因 5。
- 直接指定为 MQRFH2 元素的属性名称不能保证会由 MQPUT 调用进行验证。

作为属性的消息描述符字段
大多数消息描述符字段可视为属性。 属性名可通过将前缀添加到消息描述符字段名来构建。

如果 MQI 应用程序想识别消息描述符字段内包含的消息属性（例如，在选择器字符串中或使用消息属性 API），请使用以下语法：

属性名	消息描述符字段
Root.MQMD.*Field*	字段

使用与 C 语言声明中的 MQMD 结构化字段相同的大小写指定 *Field*。 例如，属性名称 `Root.MQMD.AccountingToken` 访问消息描述符的 `AccountingToken` 字段。

消息描述符的 `StrucId` 和 `Version` 字段不能使用所显示的语法访问。

消息描述符字段从来不像其他属性一样以 MQRFH2 头的形式表示。

如果消息数据以队列管理器所拥有的 MQMDE 开头，那么 MQMDE 字段可以使用上述 `Root.MQMD.`*`Field`* 表示访问。 这种情况下，从属性角度来看，MQMDE 字段将从逻辑上视为 MQMD 的一部分。 请参阅 MQMDE 概述。

属性数据类型和值

属性可以是布尔值、字节串、字符串、浮点或整数。 属性可以存储数据类型范围内的任何有效值，除非上下文另有限制。

属性值的数据类型必须为以下值之一：

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

属性可以在无定义值的情况下存在；这种属性称为 Null 属性。 Null 属性不同于已定义但为空值的字节属性 (MQBYTE[ ]) 或字符串属性 (MQCHAR[ ])，空值即长度为零的值。

字节串在 JMS 或 XMS 中不是有效的属性数据类型。 建议您不要在 *usr* 文件夹中使用字节串属性。

从队列中选择消息

通过使用 MQGET 调用上的 MsgId 和 CorrelId 字段，或使用 MQOPEN 或 MQSUB 调用上的 SelectionString，可以从队列中选择消息。

选择器

消息选择器是长度可变的字符串，由应用程序用来仅注册某些消息，这些消息的属性满足选择字符串所表示的结构化查询语言 (SQL) 查询。

使用 MQSUB 和 MQOPEN 函数调用的选择

使用 MQCHARV 类型结构的 *SelectionString* 来通过 MQSUB 和 MQOPEN 调用进行选择。

SelectionString 结构用于将长度可变的选择字符串传递到队列管理器。

与选择器字符串关联的 CCSID 通过 MQCHARV 结构的 VSCCSID 字段设置。 使用的值必须是选择器字符串支持的 CCSID。 请参阅代码页转换以获取受支持的代码页列表。

指定没有 IBM MQ 支持的 Unicode 转换的 CCSID 会产生 MQRC_SOURCE_CCSID_ERROR 错误。 此错误在选择器提交到队列管理器时返回，即进行 MQSUB、MQOPEN 或 MQPUT1 调用时。

VSCCSID 字段的缺省值为 MQCCSI_APPL，表示选择字符串的 CCSID 等于队列管理器 CCSID，或等于客户机 CCSID（如果通过客户机连接）。 但是，编译之前，应用程序可通过重新定义来覆盖 MQCCSI_APPL 常量。

如果 MQCHARV 选择器表示 NULL 字符串，那么不会为该消息使用者进行任何选择，并且会像不使用选择器一样发送消息。

选择字符串的最大长度只受限于 MQCHARV 字段 *VSLength* 可以描述的内容。

如果您已提供缓冲区并且 VSBufSize 缓冲区长度为正，将使用 MQSO_RESUME 订阅选项在 MQSUB 调用的输出中返回 SelectionString。 如果您不提供缓冲区，那么 MQCHARV 的 VSLength 字段中只会返回选择字符串的长度。 如果提供的缓冲区小于返回字段所需的空间，那么只有 VSBufSize 字节会返回到提供的缓冲区。

如果不先关闭队列句柄（对于 MQOPEN）或预订句柄（对于 MQSUB），应用程序无法变更选择字符串。 然后会在后续 MQOPEN 或 MQSUB 调用上指定新选择字符串。

MQOPEN
使用 MQCLOSE 关闭打开的句柄，然后在后续 MQOPEN 调用上指定新的选择字符串。

MQSUB
使用 MQCLOSE 关闭返回的预订句柄 (hSub)，然后在后续 MQSUB 调用上指定新的选择字符串。

第 26 页的图 3 显示了使用 MQSUB 调用的选择过程。

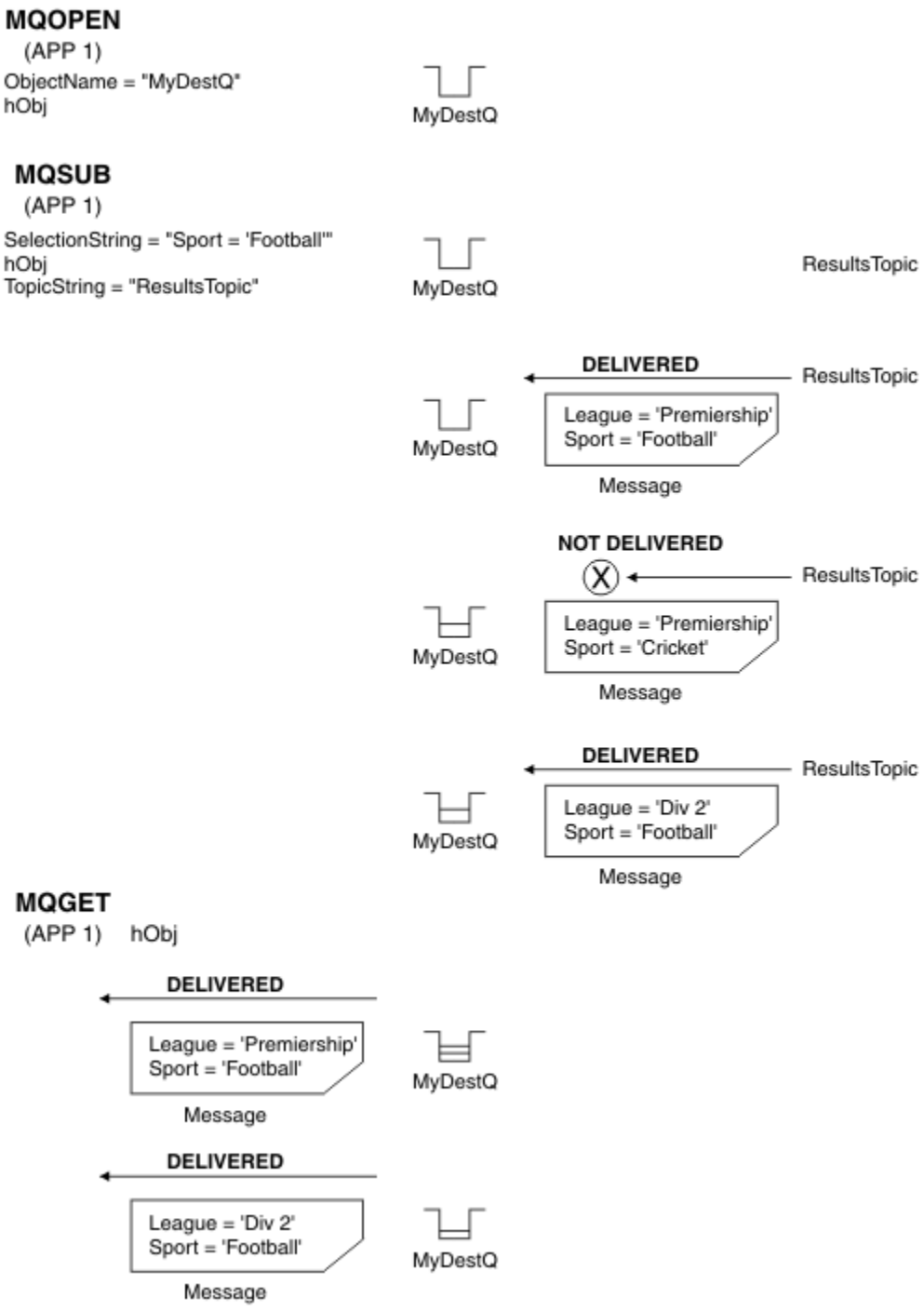


图 *3:* 使用 *MQSUB* 调用进行选择

通过使用 MQSD 结构中的 *SelectionString* 字段，选择器可以在 MQSUB 调用中进行传递。 在 MQSUB 上传递选择器的结果是只有那些发布到预订主题且与提供的选择字符串匹配的消息才能在目标队列上使用。

第 27 页的图 4 显示了使用 MQOPEN 调用的选择过程。

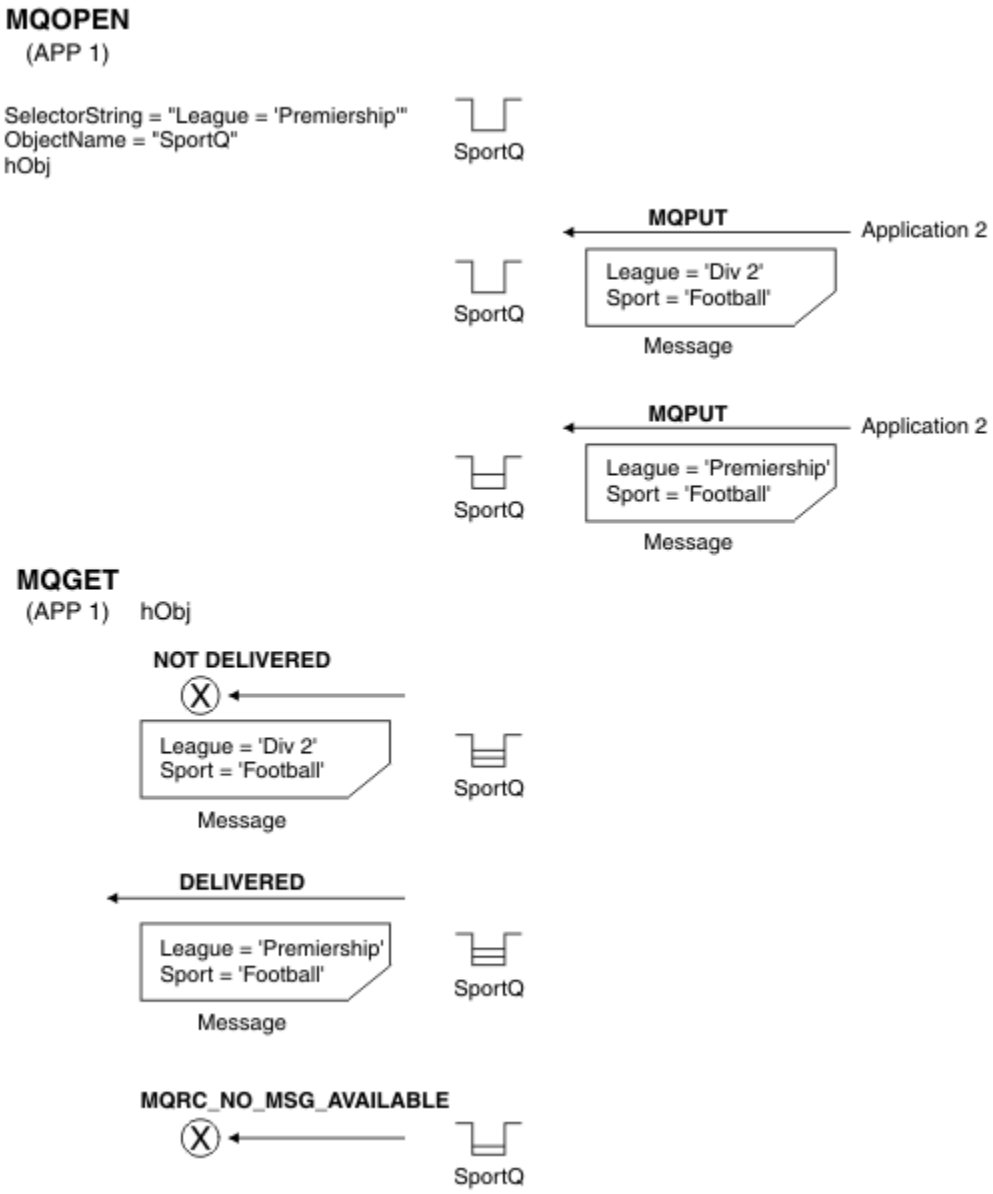


图 *4:* 使用 *MQOPEN* 调用进行选择

通过使用 MQSD 结构中的 *SelectionString* 字段，选择器可以在 MQOPEN 调用上传递。 在 MQOPEN 调用上传递选择器的结果是只有那些位于打开队列上且与选择器匹配的消息会发送给消息使用者。

选择器在 MQOPEN 调用上的主要用途是针对点到点情况，这种情况下，应用程序可以选择只接收那些与选择器匹配的队列上的消息。 上述示例展示了一个简单的场景，即两条消息放入一个由 MQOPEN 打开的队列，但获取这些消息的应用程序只接收到一条消息，因为这是唯一一个与选择器匹配的消息。

请注意，后续 MQGET 调用结果为 MQRC_NO_MSG_AVAILABLE，因为与给定选择器匹配的队列上不存在其他消息。

相关概念

第 33 页的『选择字符串规则和限制』

请熟悉关于如何理解选择字符串以及字符限制的相关规则，以避免使用选择器时出现问题。

选择行为

IBM MQ 选择行为概述。

如果 MQMD 符合以下条件，MQMDE 结构中的字段将视为对应消息描述符属性的消息属性：

- 格式为 MQFMT_MD_EXTENSION
- 后面紧跟有效的 MQMDE 结构
- 为第一版或只包含两个字段的缺省版本

出现任何匹配的消息属性之前，选择字符串有可能会解析为 TRUE 或 FALSE。 例如，如果将选择字符串设置为 “TRUE <> FALSE”，那么可能会出现这种情况。 只有选择字符串中没有消息属性引用的情况下才保证会出现此类预先评估。

如果在考虑到所有消息属性之前，选择字符串解析为 TRUE，那么将发送发布到使用者预订主题的所有消息。 如果考虑到所有消息属性之前，选择字符串解析为 FALSE，表示此选择器的函数调用上将返回原因码 MQRC_SELECTOR_ALWAYS_FALSE 和完成代码 MQCC_FAILED。

即使消息不包含消息属性（除头属性外），那么仍可以选择此消息。 如果选择字符串引用不存在的消息属性，将假设此属性的值为 NULL 或“Unknown”。

例如，消息仍满足 `'Color IS NULL'` 之类的选择字符串，其中 `'Color'` 不作为消息中的消息属性存在。

只能在与消息关联的属性上执行选择，而不是在消息本身执行选择，除非有扩展的消息选择提供程序。 如果提供了扩展的消息选择提供程序，那么只能在消息有效内容上执行选择。

每个消息属性都有一个关联类型。 在执行选择时，必须确保用于测试消息属性的表达式中所用的值是正确的类型。 如果出现类型不匹配，上述表达式将解析为 FALSE。

您有责任确保选择字符串和消息属性使用兼容的类型。

将代表不活动的持久订户继续应用选择条件，以便只保留与起初提供的选择字符串匹配的消息。

通过更改 (MQSO_ALTER) 恢复持久预订时，不能更改选择字符串。 如果持久订户恢复活动时提供了其他选择字符串，会向应用程序返回 MQRC_SELECTOR_NOT_ALTERABLE。

如果满足选择条件的队列上没有消息，应用程序将收到返回码 MQRC_NO_MSG_AVAILABLE。

如果应用程序已指定包含属性值的选择字符串，那么只有那些包含匹配属性的消息可供选择。 例如，订户指定了选择字符串“a = 3”，并且发布的消息不包含任何属性或不包含“a”不存在或不等于 3 的属性。 订户不会将该消息接收到其目标队列。

消息传递性能

从队列中选择消息将需要 IBM MQ 按照顺序检测队列上的每条消息。 对消息进行检查，直至找到符合选择标准的消息或没有其他消息可供检查。 因此，如果在较长的队列上使用消息选择，消息传递性能将受到损害。

当选择基于 JMSCorrelationID 或 JMSMessageID 时，为了优化较长队列上的消息选择，请使用以下格式的选择字符串：

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

其中：

- *correlation_id* 是包含标准 IBM MQ 相关标识的字符串。
- *message_id* 是包含标准 IBM MQ 消息标识的字符串。

注: 选择器应该仅引用其中一个属性。 使用具有其中一种格式的选择器可大幅提高基于 JMSCorrelationID 进行选择的性能，同时也略微提升了 JMSMessageID 的性能。 有关更多信息，请参阅第 112 页的『JMS 中的消息选择器』。

使用复合选择器

选择器可以包含很多组件，例如：

a 和 b， c 和 d， e 和 f， g 以及 h 或 i 和 j... 或 y 和 z

使用此类复合选择器对性能有严重影响并需要使用很多资源。 因此，IBM MQ 会变得无法处理导致系统资源短缺的过度复杂的选择器，从而保护系统。 当选择字符串包含 10 个以上的测试，或 IBM MQ 检测到即将达

到操作系统堆栈的大小限制时，会在选择字符串上实行保护。 您应该在适当的平台上通过多个组件彻底尝试和测试选择字符串的使用，以确保不会达到保护限制。

通过使用额外的括号合并组件可简化选择器，进而改进选择器的性能和复杂性。 例如：

(a 和 b 或 c 和 d) 或 (e 和 f 或 g 和 h) 或 (i 和 j) ...

相关概念

第 33 页的『选择字符串规则和限制』

请熟悉关于如何理解选择字符串以及字符限制的相关规则，以避免使用选择器时出现问题。

消息选择器语法

IBM MQ 消息选择器是配有语法的字符串，这些语法基于 SQL92 条件表达式语法的子集。

消息选择器的求值顺序为同一优先顺序级别内从左到右。 可使用括号来更改这一顺序。 预定义选择器字面值和运算符名称在此处以大写写入；但是，这些文本和运算符不区分大小写。

如果通过 API 提供选择器，那么 IBM MQ 会在提供消息选择器时验证其语法的正确性。 如果选择字符串的语法不正确或属性名无效，并且扩展消息选择提供程序不可用，那么 MQRC_SELECTION_NOT_AVAILABLE 将返回到应用程序。 如果恢复预订时选择字符串的语法不正确或属性名称无效，将向应用程序返回 MQRC_SELECTOR_SYNTAX_ERROR。 如果设置属性时禁用了属性名称验证（设置 MQCMHO_NONE，而不是 MQCMHO_VALIDATE），而应用程序随后放入了具有无效属性名称的消息，那么不会选择此消息。

如果 IBM MQ 确定以管理方式定义的预订选择器正在使用扩展消息语法 (如具有值 EXTENDED 的 **DISPLAY SUB** 参数 **SELTYPE** 所指示)，那么在提供选择器时不会返回任何错误。 在这种情况下，选择字符串的语法检查将延迟到发布时间（请参阅 MQRC_SELECTION_NOT_AVAILABLE）。

选择器可以包含：

- 字面值：
  - 字符串字面值以单引号括起。 两个连续的单引号表示一个单引号。 例如，'literal' 和 'literal''s'。 像 Java 字符串字面值一样，这些字面值使用 Unicode 字符编码。 不能使用双引号括起字符串字面值。 单引号之间可以使用任何字节序列。
  - 字节字符串是一对或多对十六进制字符，这些字符以双引号括起并带有前缀 0x。 示例为 “0x2F1C” 或 “0XD43A”。 字节字符串的长度必须至少一个字节。 如果选择器字节字符串与 MQTYPE_BYTE_STRING 类型的消息属性匹配，那么不需要对前导零或后导零采取特殊操作。 这些字节将视为另外的字符。 同时不考虑字节顺序。 选择器和属性字节字符串的长度必须相等，且字节的顺序必须相同。

    下面是匹配的字节字符串选择示例（假设 *myBytes* = 0AFC23）：

    - “myBytes = “0x0AFC23”” = TRUE

    以下字符串选择不匹配：

    - “myBytes = “0xAFC23”” = MQRC_SELECTOR_SYNTAX_ERROR（因为字节数量不是 2 的倍数）
    - “myBytes = “0x0AFC2300”” = FALSE（因为相较之下，后导零非常重要）
    - “myBytes = “0x000AFC23”” = FALSE（因为相较之下，前导零非常重要）
    - “myBytes = “0x23FC0A”” = FALSE（因为不考虑字节顺序）
  - 十六进制数字以 0 开头，后跟大写或小写 x。 文字的其余部分包含一个或多个有效十六进制字符。 例如，0xA、0xAF 和 0X2020。
  - 后跟一个或多个 0-7 范围内数字的前导零将始终认为是八进制数字的开头。 不能像以下示例这样表示以零为前缀的小数，例如，09 将返回语法错误，因为 9 不是有效的八进制数字。 八进制数字的示例有 0177、0713。
  - 准确的数字字面值为不带小数点的数字值，例如 57、-957 和 +62。 准确的数字字面值结尾可以有大写或小写的 L；这不会影响数字的存储方式和理解方式。 IBM MQ 支持 -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 范围内的准确数字。
  - 近似数字字面值是科学记数法格式的数字值，如 7E3 或 -57.9E2，或带有小数点的数字值，如 7.、-95.7 或 +6.2。 IBM MQ 支持 -1.797693134862315E+308 到 1.797693134862315E+308 范围内的数字。

有效数字应跟在可选符号字符（+ 或 -）后。 有效数字应为整数或小数。 有效数字的小数部分不需要前导数字。

大写或小写的 E 表示可选指数的开头。 指数有十进制基数，指数的数字部分可以加上可选符号字符前缀。

近似数字字面值可以以 F 或 D 字符（不区分大小写）结尾。 此语法的存在用于支持跨语言标记单精度或双精度数字的方式。 这些字符为可选字符，不会影响近似数字字面值的存储和处理方式。 始终会使用双精度存储和处理这些数字。

- 布尔文字 TRUE 和 FALSE。

注: 选择字符串中不支持无限 IEEE-754 表示法，如 NaN、+Infinity 和 -Infinity。 因此不能使用这些值作为表达式中的操作数。 对于数学运算而言，负零将视为与正零相同。

- 标识符:

  标识符是长度可变的字符顺序，必须以有效的标识符起始符开头，后跟零个或更多有效标识符组成字符。 标识符名称的规则与消息属性名称的规则相同，请参阅第 22 页的『属性名称』和第 23 页的『属性名称限制』以了解更多信息。

  注: 如果提供了扩展的消息选择提供程序，那么只能在消息有效内容上执行选择。

  标识为头字段引用或属性引用。 尽管会尽量执行数字提升，但消息选择器内属性值的类型必须与用于设置属性的类型对应。 如果出现类型不匹配，表达式的结果将为 FALSE。 如果引用了消息中不存在的属性，其值将为 NULL。

  当属性用在消息选择器表达式中时，适用于属性获取方法的类型转换不适用。 例如，如果您将属性设置为字符串值，然后使用选择器按照数字值查询此属性，表达式将返回 FALSE。

  映射到属性名或 MQMD 字段名称的 JMS 字段和属性名也是选择字符串中的有效标识。 IBM MQ 可将识别的 JMS 字段和属性名称映射到消息属性值。 请参阅 第 112 页的『JMS 中的消息选择器』，以获取更多信息。 例如，选择字符串 “JMSPriority >=” 将选择当前消息的 jms 文件夹中找到的 Pri 属性。

- 溢出/下溢:

  对于十进制和近似数字，未定义以下条件:

  - 指定不在定义范围内的数字
  - 指定将导致溢出或下溢的算术表达式

  不会对以上情况执行检查。

- 空格:

  定义为空格、换页符、换行符、回车符、横向制表符或纵向制表符。 以下 Unicode 字符将识别为空格:

  - \u0009 to \u000D
  - \u0020
  - \u001C
  - \u001D
  - \u001E
  - \u001F
  - \u1680
  - \u180E
  - \u2000 到 \u200A
  - \u2028
  - \u2029
  - \u202F
  - \u205F
  - \u3000

- 表达式:

  - 选择器是条件表达式。 求值结果为 true 的选择器匹配；结果为 false 或未知值的选择器不匹配。
  - 算术表达式由自身、算术运算、标识符（标识符值被视为数字字面值）和数字字面值组成。
  - 条件表达式由其自身、比较运算符以及逻辑运算符组成。
- 支持用于设置表达式求值顺序的标准括号 ()。
- 按照优先顺序排列的逻辑运算符：NOT、AND、OR。
- 比较运算符：=、>、>=、<、<=、<>（不等于）。
  - 两个字节字符串只有在长度相同且字节顺序相同的情况下才相等。
  - 只能比较相同类型的值。 以下情况例外：比较精确数字值和近似数字值有效（所需的类型转换由 Java 数字提升的规则定义）。 如果尝试比较不同的类型，那么选择器将始终为 false。
  - 字符串和布尔值比较限制为 = 和 <>。 两个字符串只有在包含相同顺序的字符时才相等。
- 按照优先顺序排列的算术运算符：
  - +、- 一元运算符。
  - * 乘法和 / 除法。
  - + 加法和 - 减法。
  - 不支持对 NULL 值使用算术运算。 如果尝试使用，那么整个选择器均始终为 false。
  - 算术运算符必须使用 Java 数字提升。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比较运算符：
  - `Age BETWEEN 15 and 19` 等价于 `age>= 15 AND age <= 19`。
  - `Age NOT BETWEEN 15 and 19` 等价于 `age < 15 OR age > 19`。
  - 如果任何 BETWEEN 运算的表达式为 NULL，那么运算值为 false。 如果任何 NOT BETWEEN 运算的表达式为 NULL，那么运算值为 true。
- 标识 [NOT] IN （string-literal1, string-literal2,...) 比较运算符，其中标识具有字符串或 NULL 值。
  - `Country IN ('UK', 'US', 'France')` 对于 'UK' 为 true，对于 'Peru' 为 false。 它相当于表达式 `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`。
  - `Country NOT IN ('UK', 'US', 'France')` 对于 'UK' 为 false，对于 'Peru' 为 true。 它相当于表达式 `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`。
  - 如果 IN 或 NOT IN 运算的标识符为 NULL，那么运算值未知。
- `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` 比较运算符，其中 identifier 具有字符串值。 *pattern-value* 是字符串字面值，其中 _ 代表任意单个字符，而 % 代表任何字符序列（包括空序列）。 所有其他字符均代表其自身。 可选 *escape-character* 是单个字符串字面值，用于转义 *pattern-value* 中 _ 和 % 的特殊含义。 LIKE 运算符只能用于比较两个字符串值。
  - `phone LIKE '12%3'` 对于 123 和 12993 为 true，对于 1234 为 false。
  - `word LIKE 'l_se'` 对于 lose 为 true，对于 loose 为 false。
  - `underscored LIKE '\_%' ESCAPE '\'` 对于 _foo 为 true，对于 bar 为 false。
  - `phone NOT LIKE '12%3'` 对于 123 和 12993 为 false，对于 1234 为 true。
  - 如果 LIKE 或 NOT LIKE 运算的标识符为 NULL，那么此运算的值为 unknown。

  注: LIKE 运算符必须用于比较两个字符串值。 `Root.MQMD.CorrelId` 的值为 24 字节的字节数组，而不是字符串。 解析器接受选择器字符串 `Root.MQMD.CorrelId LIKE 'ABC%'` 作为有效语法，但会求值为 false。 因此在比较字节数组和字符串时，不会使用 LIKE。
- `identifier IS NULL` 比较运算符测试是否存在 NULL 头字段值或是否缺失属性值。
- `identifier IS NOT NULL` 比较运算符测试是否存在非空头字段值或属性值。
- null 值

  总而言之，包含 NULL 值的选择器表达式的求值由 SQL 92 NULL 语义定义：
  - SQL 将 NULL 值视为 unknown。

– 带有 unknown 值的比较或算术始终会产生 unknown 值。
– IS NULL 和 IS NOT NULL 运算符可将 unknown 值转换为 TRUE 和 FALSE 值。

布尔运算符使用三值逻辑（T=TRUE、F=FALSE 和 U=UNKNOWN）

表 *1:* 逻辑为 *A AND B* 时的布尔运算符结果的值

运算符 A	**运算符 B**	**结果 (A AND B)**
T	F	F
T	U	U
T	T	T
F	T	F
F	U	F
F	F	F
U	T	U
U	U	U
U	F	F

表 *2:* 逻辑为 *A OR B* 时的布尔运算符结果的值

运算符 A	**运算符 B**	**结果 (A OR B)**
T	F	T
T	U	T
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

表 *3:* 逻辑为 *NOT A* 时的布尔运算符结果的值

运算符 A	**结果 (NOT A)**
T	F
F	T
U	U

以下消息选择器选择消息类型为汽车，颜色为蓝色且重量大于 2500 磅的消息：

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

尽管 SQL 支持固定的小数比较和算术，但消息选择器并不支持。 这是因为精确数字字面值限制为不带小数点的数字。 还因为某些带有小数点的数字作为近似数字值的替代表示。

不支持 SQL 注释。

相关概念

第 21 页的『消息属性』
使用消息属性可让应用程序选择要处理的消息，或在无需访问 MQMD 或 MQRFH2 头的情况下检索有关消息的信息。 这些属性也便于 IBM MQ 和 JMS 应用程序之间的通信。

相关参考

MsgHandle

MQBUFMH - 将缓冲区转换为消息句柄

选择字符串规则和限制

请熟悉关于如何理解选择字符串以及字符限制的相关规则，以避免使用选择器时出现问题。

- 发布/预订消息传递的消息选择在发布者发送消息时发生在消息上。 请参阅选择字符串。
- 使用单个等号字符测试等效性；例如，`a = b` 是正确的，而 `a == b` 是不正确的。
- 很多编程语言用于表示“不等于”的运算符为 `!=`。 这种表示不是 <> 的有效同义词；例如，`a <> b` 有效，但 `a != b` 无效。
- 只有在使用 ' (U+0027) 字符时，才会识别单引号。 同样，仅在用于包含字节字符串时才有效的双引号必须使用 " (U+0022) 字符。
- 符号 &、&&、| 和 || 不是逻辑合取/析取的同义词，例如 `a && b` 必须指定为 `a AND b`。
- 通配符 * 和 ? 不是 % 和 _ 的同义词。
- 包含 `20 < b < 30` 之类的复合表达式的选择器无效。 解析器将从左向右对具有同一优先顺序的运算符进行求值。 因此，该示例将变为 `(20 < b) < 30`（这没有任何意义）。 相反，此表达式必须编写为 `(b > 20) AND (b < 30)`。
- 字节串必须用双引号引起；如果使用单引号，字节串将被视为字符串字面值。 0x 后跟的字符数量（不是字符表示的数量）必须是二的倍数。
- 关键字 IS 不是等号的同义词。 因此选择字符串 `a IS 3` 和 `b IS 'red'` 无效。 IS 关键字的存在只是为了支持 `IS NULL` 和 `IS NOT NULL` 情况。

相关概念

选择字符串

第 27 页的『选择行为』
IBM MQ 选择行为概述。

使用消息选择器时的 *UTF-8* 和 *Unicode* 注意事项

组成选择字符串的保留关键字的字符（没有括在单引号中）必须以基本拉丁语 Unicode（字符范围为 U+0000 到 U+0007F）输入。 使用字母数字字符的其他代码点表示是无效的。 例如，数字 1 在 Unicode 中必须表示为 U+0031，使用相等的全形数字 U+FF11 或相等的阿拉伯数字 U+0661 是无效的。

消息属性名称可以使用任何有效的 Unicode 字符序列指定。 将对以 UTF-8 编码的选择字符串内包含的消息属性名称进行验证，即使其中包含多字节字符。 多字节 UTF-8 的验证非常严格，您必须确保为消息属性名称使用有效的 UTF-8 序列。 消息属性名称中不支持超出 Unicode 基本多文种平面的字符（超出 U+FFFF 的字符）、在 UTF-16 中通过超大字符集代码点（X'D800' 到 X'DFFF'）表示的字符或者 UTF-8 中四字节字符。

在比较等同性时不会对属性或值执行额外处理。 例如，这表示不会发生预组合/分解且不会对连字赋予任何特殊含义。 例如，预组合元音变音字符 U+00FC 不视为与 U+0075 + U+0308 相等，而字符序列 ff 不视为与 Unicode U+FB00 (LATIN SMALL LIGATURE FF) 相等

单引号内引起的属性数据可以由任意字节序列表示且不会对其进行验证。

选择消息内容

可以根据消息有效内容（也称为内容过滤）的选择进行预订，但是关于应将哪个消息发送到此预订的决定不能直接由 IBM MQ 执行；而是需要扩展消息选择提供程序（如 IBM Integration Bus）来处理这些消息。

当应用程序在主题字符串（其中一个或多个订户有选择消息内容的选择字符串）上发布时，IBM MQ 将请求该扩展消息选择提供程序解析发布内容并通知 IBM MQ 该发布内容是否符合每个订户通过内容过滤器指定的选择标准。

如果扩展消息选择提供程序确定发布内容与订户的选择字符串匹配，那么消息将继续传送给订户。

如果扩展消息选择提供程序确定发布内容不匹配，那么不会将消息传送给订户。 这可能会导致 MQPUT 或 MQPUT1 调用失败，同时带有原因码 MQRC_PUBLICATION_FAILURE。 如果扩展消息选择提供程序无法解析发布内容，那么将返回原因码 MQRC_CONTENT_ERROR，并且 MQPUT 或 MQPUT1 调用失败。

如果扩展消息选择提供程序不可用或者无法确定订户是否应该接收发布内容，那么将返回原因码 MQRC_SELECTION_NOT_AVAILABLE，并且 MQPUT 或 MQPUT1 调用失败。

在创建带有内容过滤器的预订时，如果扩展消息选择提供程序不可用，MQSUB 调用将失败并带有原因码 MQRC_SELECTION_NOT_AVAILABLE。 如果要恢复带有内容过滤器的预订，但扩展消息选择提供程序不可用，MQSUB 调用将返回警告 MQRC_SELECTION_NOT_AVAILABLE，但允许恢复预订。

相关概念

选择字符串

IBM MQ 消息的异步使用

“异步使用”使用一组消息队列接口 (MQI) 扩展（MQI 调用 MQCB 和 MQCTL），这些调用允许要编写的 MQI 应用程序使用来自一组队列的消息。 通过调用应用程序所标识的代码单元并传递消息或表示消息的标记，将消息传递到应用程序。

在最直接的应用程序环境中，代码单元由函数指针定义，但在其他环境中，代码单元可以由程序或模块名称定义。

在消息的异步使用中，将使用以下术语：

消息使用者
一种编程结构，允许您定义与应用程序需求相匹配的消息可用时要随消息一起调用的程序或函数。

事件处理程序
一种编程结构，允许您定义发生异步事件（例如，队列管理器停止）时要调用的程序或函数。

回调
一般术语，指的是消息使用者或事件处理程序例程。

异步使用可以简化新应用程序的设计和实施，尤其是那些处理多个输入队列或预订的应用程序。 但是，如果您使用多个输入队列并且按照优先级顺序处理消息，将单独遵循每个队列中的优先级顺序：即您可能会先得到来自一个队列的低优先级消息，后得到来自另一队列的高优先级消息。 不保证多个队列间的消息顺序。 另外请注意，如果您使用 API 出口，您可能需要更改这些出口来包含 MQCB 和 MQCTL 调用。

下图提供了如何使用此函数的示例。

第 35 页的图 5 显示使用来自两个队列的消息的多线程应用程序。 该示例显示了要发送到单个函数的所有消息。

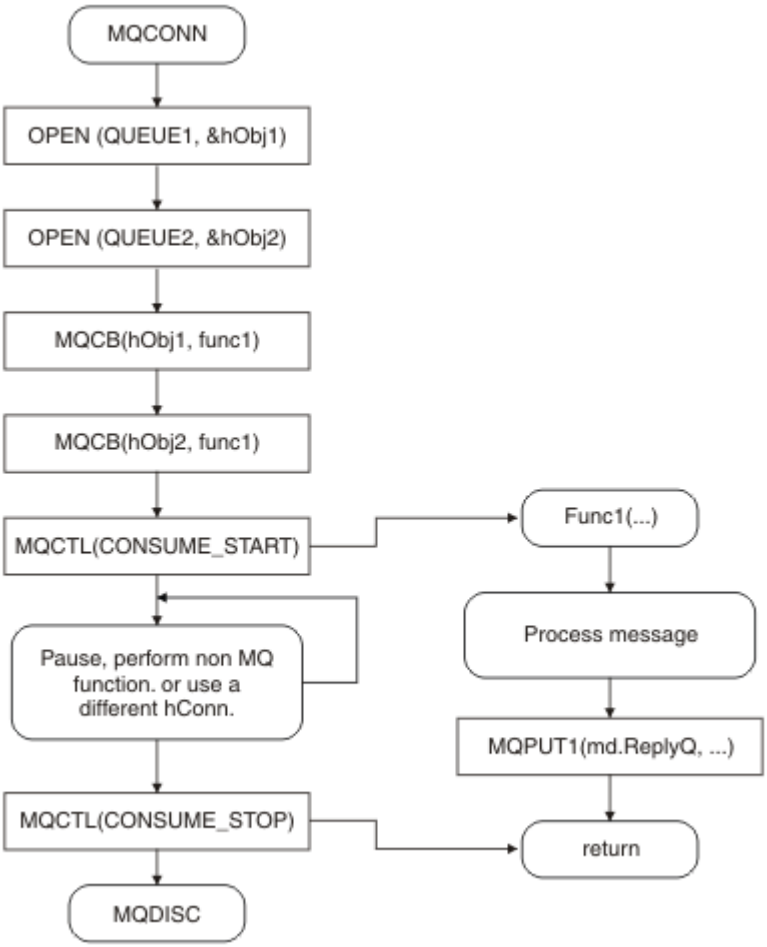


图 *5:* 使用来自两个队列的消息的标准消息驱动应用程序

z/OS 在 z/OS 上，主要控制线程必须在结束前发出 MQDISC 调用。 此操作能使任何回调线程结束并释放系统资源。

第 36 页的图 6 该样本流程显示使用来自两个队列的消息的单线程应用程序。 该示例显示了要发送到单个函数的所有消息。

与异步情况的差异在于该控制不会返回到 MQCTL 的发出者，直到所有使用者均已停用；即一个使用者已发出 MQCTL STOP 请求或队列管理器已停止。

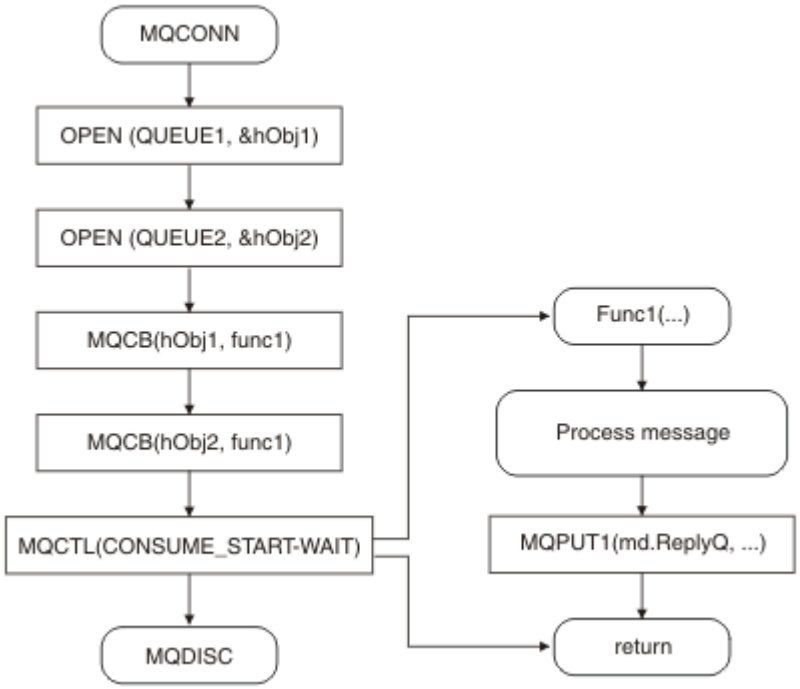


图 *6:* 使用来自两个队列的消息的单线程消息驱动应用程序

消息组

消息可以出现在组内以对消息排序。

消息组允许将多个消息标记为与另一个消息相关，并允许对组应用逻辑顺序（请参阅第 718 页的『逻辑与物理排序』）。 在 多平台 上，消息分段支持将大型消息分为较小的分段。 将消息放入主题时不能使用分组或分段消息。

组内的层次结构如下所示：

组

这是层次结构的最高级别，由 *GroupId* 标识。 它由一个或多个包含相同 *GroupId* 的消息组成。 这些消息可以存储在队列的任意位置。

注: 术语消息在这里用于表示队列上的一个项，例如将由未指定 MQGMO_COMPLETE_MSG 的单个 MQGET 返回的项。

第 36 页的图 7 显示了一组逻辑消息：

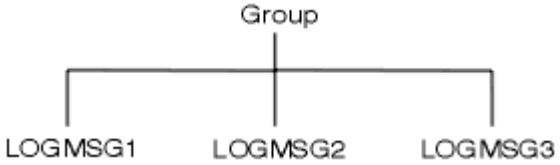


图 *7:* 逻辑消息组

通过打开队列并指定 MQOO_BIND_ON_GROUP，可以强制将发送到此队列的组中的所有消息发送到同一队列实例。 有关 BIND_ON_GROUP 选项的更多信息，请参阅处理消息亲缘关系。

逻辑消息

由 *GroupId* 和 *MsgSeqNumber* 字段标识的组内的逻辑消息。 对于组中的第一条消息，*MsgSeqNumber* 以 1 开头，如果消息不在组中，则该字段的值为 1。

使用组中的逻辑消息执行下列操作：

- 确保排序（如果传输消息时没有保证排序）。
- 允许应用程序对类似消息进行分组（例如，必须全部由同一服务器实例处理的消息）。

组中的每条消息由一个物理消息组成，除非该消息分为多个分段。 每条消息在逻辑上是单独的消息，只有 MQMD 中的 *GroupId* 和 *MsgSeqNumber* 字段需要拥有与组中其他消息的关系。 MQMD 中的其他字段是独立的；其中某些字段可能对于组中的所有消息都相同，但其他字段可能不同。 例如，组中的消息可以有不同的格式名称、CCSID 和编码。

分段

分段用于为输入或获取应用程序或者队列管理器（包括借助其传递消息的中间队列管理器）处理过大的消息。 有关更多信息，请参阅 第 733 页的『消息分段』。

单个消息拆分为名为分段的较小的消息。 消息段由 *GroupId*、*MsgSeqNumber* 和 *Offset* 字段标识。 对于消息中的第一个分段，*Offset* 字段以 0 开头。

每个分段由一个可能属于某个组的物理消息组成（第 37 页的图 8 显示了组中的消息示例）。 分段在逻辑上是单个消息的一部分，因此只有 MQMD 中的 *MsgId*、*Offset* 和 *MsgFlags* 字段可以区分同一消息的两个独立分段。 如果分段未能到达，那么将适当地返回原因码 MQRC_INCOMPLETE_GROUP 或 MQRC_INCOMPLETE_MSG。

第 37 页的图 8 显示了一组逻辑消息，其中某些消息已分段：

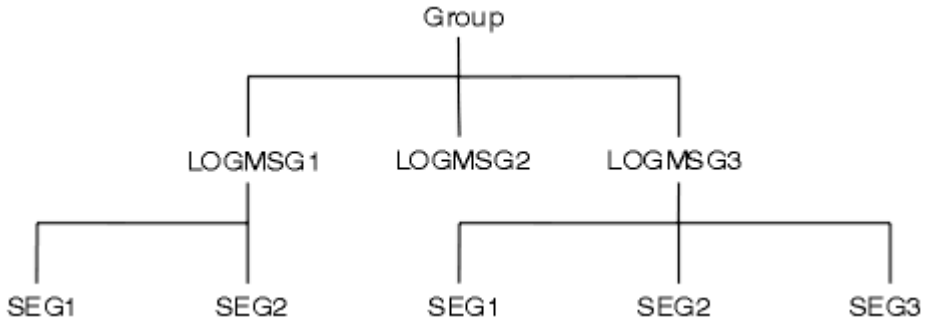


图 *8:* 分段消息

z/OS IBM MQ for z/OS 上不支持分段。

不能为发布/预订使用分段或组合消息。

相关概念

第 733 页的『消息分段』
使用此信息来了解消息分段。 在 IBM MQ for z/OS 中或使用 IBM MQ classes for JMS 的应用程序中不支持此功能。

相关参考

第 718 页的『逻辑与物理排序』
队列上的消息可以物理或逻辑顺序（在每个优先级内）显示。

MQMD - 消息描述符

消息持久性

持久消息将写入日志和队列数据文件。 如果队列管理器在发生故障后重新启动，会在必要时从已记录的数据中恢复这些持久消息。 如果队列管理器停止，将废弃非持久消息，无论该停止是因为运算符命令还是因为系统某部分发生故障。

z/OS 但对于 z/OS 上存储在耦合设施 (CF) 中的非持久性消息，此情况例外。 只要 CF 仍然可用，它们就会一直存在。

在创建消息时，如果使用缺省值初始化消息描述符 (MQMD)，那么消息的持久性取自 MQOPEN 命令中指定队列的 **DefPersistence** 属性。 或者，您可以使用 MQMD 结构的 *Persistence* 字段设置消息持久性以将消息定义为持久或非持久。

使用持久消息时，应用程序的性能将受到影响；影响的程度取决于机器 I/O 子系统的性能特征和您在各个平台上使用同步点选项的方式：

- 当前工作单元之外的持久消息在每次 PUT 和 GET 操作时写入磁盘。 请参阅 第 786 页的『落实和回退工作单元』。
- ULW z/OS 对于除 IBM i 以外的所有平台，只在提交工作单元并且工作单元可以包含很多队列操作时记录当前工作单元内的持久消息。

非持久消息可用于快速消息传递。 请参阅消息安全性以了解有关快速消息的更多信息。

注: 同时在工作单元内写持久消息和在工作单元外写持久消息可能会导致您的应用程序出现严重的性能问题。 为两种操作使用同一目标队列时更是如此。

发送失败的消息

当队列管理器无法将消息放入队列时，您可以使用多个选项。

您可以:

- 再次尝试将消息放入队列。
- 请求将消息返回给发送者。
- 将消息放入死信队列。

请参阅第 947 页的『处理过程程序错误』，了解更多信息。

回退的消息

处理受工作单元控制的队列中的消息时，工作单元可以由一个或多个消息组成。 如果发生回退，从该队列检索的消息将恢复到队列，并且可以重新在另一个工作单元内处理。 如果是因为特定消息的处理导致此问题，工作单元将再次回退。 这可能会导致处理循环。 之前放入队列的消息将从队列中移除。

通过测试 MQMD 的 *BackoutCount* 字段，应用程序可以检测陷入此类循环的消息。 应用程序可以纠正这种情况，或向操作员发出警告。

Multi 回退计数在重新启动队列管理器后始终存在。 对 **HardenGetBackout** 属性的任何更改将被忽略。

z/OS 对于共享队列，回退计数在重新启动队列管理器后始终存在。 对于 z/OS 上的所有其他配置，为了确保专用队列的回退计数在队列管理器重新启动后仍然存在，请将 *HardenGetBackout* 属性设置为 MQQA_BACKOUT_HARDENED；否则，如果队列管理器必须重新启动，将不会保留每个消息的精确回退计数。 使用这种方法设置属性会增加额外处理的成本。

有关提交和回退消息的更多信息，请参阅第 786 页的『落实和回退工作单元』。

应答队列和队列管理器

有些时候，您可能会收到对您所发送的消息进行响应的消息。

- 用于响应请求消息的应答消息
- 有关意外事件或到期的报告消息
- 有关 COA（确认到达）或 COD（发送确认）事件的报告消息
- 有关 PAN（肯定操作通知）或 NAN（否定操作通知）事件的报告消息

使用 MQMD 结构在 *ReplyToQ* 字段中指定要应答并报告消息已发送的队列的名称。 在 *ReplyToQMgr* 字段中指定拥有应答队列的队列管理器的名称。

如果将 *ReplyToQMgr* 字段留空，队列管理器会在该队列的消息描述符中设置以下字段的内容:

ReplyToQ
如果 *ReplyToQ* 是远程队列的本地定义，那么 *ReplyToQ* 字段将设置为远程队列的名称；如果不是，此字段不会更改。

ReplyToQMgr
如果 *ReplyToQ* 是远程队列的本地定义，那么 *ReplyToQMgr* 字段将设置为拥有此远程队列的队列管理器的名称；如果不是，*ReplyToQMgr* 字段将设置为应用程序所连接的队列管理器的名称。

注: 您可以请求队列管理器多次尝试传送消息，如果失败，您可以请求废弃此消息。 如果消息传送失败后不废弃此消息，远程队列管理器会将此消息放入死信（未发送消息）队列（请参阅第 950 页的『使用死信（未送达消息）队列』）。

消息上下文

消息上下文信息使得检索消息的应用程序能够了解有关消息发起方的信息。

检索应用程序可能希望执行以下操作:

- 检查发送应用程序是否拥有正确的权限级别
- 执行某些记帐功能，以便向发送应用程序就其必须执行的工作收取费用。
- 对其处理的所有消息进行审计跟踪

使用 MQPUT 或 MQPUT1 调用将消息放入队列时，您可以指定队列管理器向消息描述符中添加某些缺省上下文信息。 拥有相应权限级别的应用程序可以添加额外的上下文信息。 有关如何指定上下文信息的更多信息，请参阅第 706 页的『控制消息上下文信息』。

队列管理器在生成以下类型的报告消息时会使用用户上下文:

- 传送时确认
- 到期

生成这些报告消息时，在用户上下文中查找对报告目标的 +put 和 +passid 权限。 如果用户上下文没有足够的权限，报告消息将放入死信队列（如果定义了死信队列）。 如果没有死信队列，报告消息将被废弃。

所有上下文信息都存储在消息描述符的上下文字段中。 信息类型划分为身份信息、源信息和用户上下文信息。

身份上下文

身份上下文信息用于标识第一个在队列上放入消息的应用程序的用户。 经过适当授权的应用程序可以设置以下字段:

- 队列管理器使用用于标识用户的名称填充 *UserIdentifier* 字段；队列管理器执行此操作的方式取决于用于运行应用程序的环境。
- 队列管理器使用标记或数字填充 *AccountingToken* 字段，该标记或数字是根据放入消息的应用程序确定的。
- 应用程序可以使用 *ApplIdentityData* 字段以获取任何它们希望包括的有关用户的额外信息（例如，加密密码）。

在 IBM MQ for Windows 下创建消息时， Windows 系统安全标识 (SID) 存储在 *AccountingToken* 字段中。 SID 可以用来补充 *UserIdentifier* 字段和建立用户凭证。

有关队列管理器如何填充 *UserIdentifier* 和 *AccountingToken* 字段的信息，请参阅 UserIdentifier 和 AccountingToken 中有关这些字段的描述。

从一个队列管理器向另一个队列管理器传递消息的应用程序还应传递身份上下文信息，这样其他应用程序才能知道消息发起方的身份。

源上下文

源上下文信息描述将消息放入当前存储此消息的队列的应用程序。 消息描述符包含源上下文信息的以下字段:

- *PutApplType* 定义放入消息的应用程序的类型（例如，CICS 事务）。
- *PutApplName* 定义放入消息的应用程序的名称（例如，作业或事务的名称）。
- *PutDate* 定义消息放入队列的日期。

- *PutTime* 定义消息放入队列的时间。
- *ApplOriginData* 定义应用程序要包含的有关消息源的任何额外信息。 例如，该信息可以由经过适当授权的应用程序设置以指示身份数据是否可信。

源上下文信息通常由队列管理器提供。 格林威治标准时间 (GMT) 用于 *PutDate* 和 *PutTime* 字段。 请参阅 PutDate 和 PutTime 中有关这些字段的描述。

具有足够权限的应用程序可以提供自己的上下文。 当单个用户在处理他们所发出的消息的各个系统上具有不同的用户标识时，上下文能够保留帐户信息。

IBM MQ 对象

该信息提供有关 IBM MQ 对象的详细信息，其中包括：队列管理器、队列共享组、队列、管理主题对象、名称列表、流程定义、认证信息对象、通道、存储类、侦听器和服务。

队列管理器定义这些对象的属性（也称为特性）。 这些属性的值影响着 IBM MQ 处理这些对象的方式。 在您的应用程序中，使用消息队列接口 (MQI) 控制这些对象。 从程序中处理时，对象由对象描述符 (MQOD) 标识。

例如，当您使用 IBM MQ 命令定义、更改或删除对象时，队列管理器将检查您是否具有执行这些操作所需的权限级别。 同样，当应用程序使用 MQOPEN 调用打开对象时，队列管理器将检查应用程序是否具有所需的权限级别，然后才能访问此对象。 针对要打开的对象的名称执行检查。

相关概念

第 706 页的『控制消息上下文信息』
使用 MQPUT 或 MQPUT1 调用将消息放入队列时，您可以指定队列管理器向消息描述符中添加某些缺省上下文信息。 拥有相应权限级别的应用程序可以添加额外的上下文信息。 您可以在 MQPMO 结构中使用 options 字段来控制上下文信息。

相关参考

第 698 页的『用于关联消息上下文的 MQOPEN 选项』
如果要可以在将消息放置到队列上时将上下文信息与消息关联，必须在打开队列时使用其中一个消息上下文选项。

Windows 准备和运行 Microsoft Transaction Server 应用程序

要准备让 MTS 应用程序作为 IBM MQ MQI client 应用程序运行，请遵循适合您环境的指示说明。

有关如何开发访问 IBM MQ 资源的 Microsoft Transaction Server (MTS) 应用程序的常规信息，请参阅 IBM MQ 帮助中心中有关 MTS 的部分。

要准备让 MTS 应用程序作为 IBM MQ MQI client 应用程序运行，请对应用程序的各个组件执行下列操作：

- 如果组件为 MQI 使用 C 语言绑定，请遵循第 929 页的『在 Windows 中准备 C 程序』中的指示信息，但请将组件与库 mqicxa.lib（而不是 mqic.lib）链接在一起。
- 如果组件使用 IBM MQ C++ 类，请遵循第 459 页的『在 Windows 上构建 C++ 程序』中的指示信息，但请将组件与库 imqx23vn.lib（而不是 imqc23vn.lib）链接在一起。
- 如果组件将 Visual Basic 语言绑定用于 MQI，请遵循第 932 页的『在 Windows 中准备 Visual Basic 程序』中的指示说明，但在定义 Visual Basic 项目时，请在**条件编译自变量**字段中输入 MqType=3。
- 如果组件使用 IBM MQ Automation Classes for ActiveX (MQAX)，请定义一个值为 `mqic32xa.dll` 的环境变量 GMQ_MQ_LIB。

  您可以在您的应用程序中定义环境变量，或者将它定义为作用域是整个系统。 但是，将它的作用域定义为系统将导致任何现有的未在应用程序中定义环境变量的 MQAX 应用程序的行为不正常。

IBM MQ 应用程序的设计注意事项

当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

设计 IBM MQ 应用程序时，请考虑以下问题和选项：

应用程序类型

应用程序的用途是什么？ 请参阅以下链接以获取有关可以开发的不同类型的应用程序的信息：

- 服务器
- 客户机
- 发布／预订
- Web Service
- 用户出口、API 出口和可安装服务

此外，您还可以编写自己的应用程序来将 IBM MQ 的管理自动化。 有关更多信息，请参阅 IBM MQ 管理界面 (MQAI) 和自动完成管理任务。

编程语言

IBM MQ 支持使用多种不同的编程语言来编写应用程序。 有关更多信息，请参阅第 5 页的『为 IBM MQ 开发应用程序』。

用于多个平台的应用程序

您的应用程序是否将在多个平台上运行？ 您是否具有从如今使用的平台移至其他平台的策略？ 如果其中任一问题的回答为是，请确保对您的程序进行编码以实现平台独立性。

例如，如果使用的是 C，那么将使用 ANSI 标准 C 来编码。 使用标准 C 库函数（而不是特定于平台的同等函数），即使特定于平台的函数更快或更高效也是如此。 例外情况是在代码中的效率至关重要时，应使用 #ifdef 对两种情况均进行编码。 例如：

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

队列类型

您想要每次需要队列时都创建队列，还是想要使用已设置的队列？ 您想要在完成使用队列时将其删除，还是将要再次使用该队列？ 您是否想要使用别名队列以实现应用程序独立性？ 要查看支持的队列类型，请参阅队列。

z/OS 使用共享队列、队列共享组和队列共享组集群（仅限 IBM MQ for z/OS）

您可能想要利用在将共享队列与队列共享组结合使用时提高的可用性、可扩展性和工作负载均衡。 有关更多信息，请参阅共享队列和队列共享组 。

您可能还想要估算平均和峰值消息流，并且考虑使用队列共享组集群来分布工作负载。 有关更多信息，请参阅共享队列和队列共享组 。

使用队列管理器集群

您可能想要利用简化的系统管理，以及在使用集群时提高的可用性、可扩展性和工作负载均衡。

消息类型

您可能想要对简单消息使用数据报，但是对其他情况使用请求消息（对于此类消息您期望获取回复）。 您可能想要向某些消息分配不同的优先级。 有关设计消息的更多信息，请参阅第 48 页的『消息的设计方法』。

使用发布/预订或点到点消息传递

使用发布/预订消息传递，发送应用程序将其想要在 IBM MQ 消息中共享的信息发送到由 IBM MQ 发布/预订管理的标准目标，并且让 IBM MQ 处理该信息的分发。 目标应用程序不必了解有关其接收的信息源的任何信息，它只是表明对一个或多个主题的兴趣并在信息可用时接收该信息。 有关发布/预订消息传递的更多信息，请参阅发布/预订消息传递。

使用点到点消息传递，发送应用程序将消息发送到特定队列，它从中知道接收应用程序将对其进行检索。 接收应用程序从特定队列获取消息并处理其内容。 应用程序往往将同时充当发送方和接收方，将查询发送到其他应用程序并接收响应。

控制 IBM MQ 程序

您可能想要自动启动某些程序，或者使程序等待直至特定消息到达队列（使用 IBM MQ 触发功能，请参阅第 795 页的『使用触发器启动 IBM MQ 应用程序』）。 或者，您可能想要在队列上的消息处理不够快速时启动应用程序的其他实例（使用 IBM MQ 检测事件功能，如检测事件中所述）。

在 IBM MQ 客户机上运行应用程序

在客户机环境中支持完整 MQI，并且可以重新链接采用过程语言编写的几乎任何 IBM MQ 应用程序以在 IBM MQ MQI client 上运行。 请将 IBM MQ MQI client 上的应用程序链接到 MQIC 库而非 MQI 库。

z/OS 不支持 z/OS 上的 Get(signal)。

注: 在 IBM MQ 客户机上运行的应用程序可以并发连接到多个队列管理器，或者在 MQCONN 或 MQCONNX 调用中使用带有星号 (*) 的队列管理器名称。 由于此函数将不可用，因此如果要链接到队列管理器库而不是客户机库，请更改应用程序。

请参阅 第 841 页的『在 IBM MQ MQI client 环境中运行应用程序』 以获取更多信息。

应用程序性能

设计决策可以影响应用程序性能，有关增强 IBM MQ 应用程序性能的建议，请参阅第 49 页的『应用程序设计和性能注意事项』 IBM i 和 第 52 页的『IBM i 应用程序的设计和性能注意事项』。

高级 IBM MQ 方法

对于更高级的应用程序，您可能想要使用某些高级 IBM MQ 方法，例如关联回复以及生成和发送 IBM MQ 上下文信息。 有关更多信息，请参阅 第 50 页的『高级应用程序的设计方法』。

保护数据并维护其完整性

您可以使用随消息传递的上下文信息来测试是否已从可接受的源发送消息。 可以使用 IBM MQ 或操作系统提供的指向同步点设施来确保数据与其他资源保持一致（请参阅第 786 页的『落实和回退工作单元』以获取进一步详细信息）。 您可以使用 IBM MQ 消息的 持久性 功能来确保重要消息的传递。

测试 IBM MQ 应用程序

IBM MQ 程序的应用程序开发环境与任何其他应用程序并无不同，因此可以使用相同的开发工具以及 IBM MQ 跟踪功能。

z/OS 使用 IBM MQ for z/OS 测试 CICS 应用程序时，可以使用 CICS 执行诊断工具 (CEDF)。 CEDF 在每个 MQI 调用以及对所有 CICS 服务的调用的入口和出口设置陷阱。 此外，在 CICS 环境中，可以编写 API 交叉出口程序以在每个 MQI 调用前后提供诊断信息。 有关如何执行此操作的信息，请参阅 第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』。

IBM i 测试 IBM i 应用程序时，可以使用标准调试器。 要启动此调试器，请使用 STRDBG 命令。

处理异常和错误

您需要考虑如何处理无法传送的消息，以及如何解决队列管理器向您报告的错误情况。 对于某些报告，必须在 MQPUT 上设置报告选项。

相关概念

IBM MQ 技术概述

第 53 页的『z/OS 应用程序的设计和性能注意事项』
应用程序设计是影响性能的最重要因素之一。 使用本主题来了解性能中涉及的某些设计因素。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

第 672 页的『编写用于排队的过程应用程序』
使用此信息来了解有关编写排队应用程序、连接和断开队列管理器、发布/预订以及打开和关闭对象的信息。

第 834 页的『编写客户机过程应用程序』
使用过程语言在 IBM MQ 上编写客户机应用程序时需要知道的内容。

第 463 页的『开发 .NET 应用程序』

IBM MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 IBM MQ MQI client 连接到 IBM MQ 或直接连接到 IBM MQ 服务器。

第 437 页的『开发 C++ 应用程序』
IBM MQ 提供等同于 IBM MQ 对象的 C++ 类，以及等同于数组数据类型的一些其他类。 它提供许多不能通过 MQI 使用的功能。

第 67 页的『使用 IBM MQ classes for JMS』
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

第 281 页的『使用 IBM MQ classes for Java』
在 Java 环境中使用 IBM MQ 。 IBM MQ classes for Java 允许 Java 应用程序作为 IBM MQ 客户机连接到 IBM MQ，或直接连接到 IBM MQ 队列管理器。

第 561 页的『使用组件对象模型接口 (IBM MQ Automation Classes for ActiveX)』
IBM MQ Automation Classes for ActiveX (MQAX) 是一些 ActiveX 组件，这些组件提供了一些类，您可以在应用程序中使用这些类来访问 IBM MQ。

V 9.1.2 以受支持的编程语言指定应用程序名称

在 IBM MQ 9.1.2 之前，您已经可以在 Java 或 JMS 客户机应用程序上指定应用程序名称。 从 IBM MQ 9.1.2 开始，将此功能扩展至 IBM MQ for Multiplatforms 上的其他编程语言。

如何使用应用程序名称

通过以下方式输出应用程序名称：

- runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE(HANDLE) APPLTAG
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- 应用程序活动跟踪

配置应用程序活动跟踪时，也会使用应用程序名称。 非 Java 应用程序的缺省应用程序名称为可执行文件的截断名称，Windows 上除外。

Windows 在 Windows 上，缺省名称为标准可执行文件名称，截断至左起第 28 个字符。

对于 Java 应用程序，缺省名称为使用程序包名称作为前缀的类名，并截断至左起第 28 个字符。

有关更多信息，请参阅 PutApplName。

从 IBM MQ 9.1.2 开始，IBM MQ for Multiplatforms 上的应用程序能够以管理方式或者使用各种编程方法来设置应用程序名称。 在配置应用程序活动跟踪或者从各种 **runmqsc** 命令输出时，这支持应用程序提供更有意义的、独立于平台的名称。

IBM MQ 9.1.2 还添加了跨统一集群实现应用程序再平衡的功能。 有意义的应用程序名称可用于实现此功能。

支持的字符

请参阅第 44 页的『建议的应用程序名称字符』，获取有关如何指定应用程序的更多信息。

编程语言

请参阅第 45 页的『编程语言连接』，获取有关以 C 语言和其他编程语言解析为 IBM MQ 库的应用程序如何提供应用程序名称的更多信息。

受管 .NET 应用程序

请参阅第 46 页的『受管 .NET 应用程序』，获取有关受管 .NET 应用程序如何提供应用程序的信息。

XMS 应用程序

请参阅 第 47 页的『XMS 应用程序』，获取有关 XMS 应用程序如何提供应用程序名称的信息。

Java 和 JMS 绑定应用程序

ULW

请参阅 第 47 页的『Java 和 JMS 绑定应用程序』，获取有关 Java 和 JMS 应用程序如何提供应用程序名称的信息。

相关概念

应用程序活动跟踪

统一集群

相关参考

MQCNO

V 9.1.2 在支持的编程语言中使用应用程序名称

可使用此信息来了解如何在 IBM MQ 支持的各种语言中选择应用程序名称。

建议的应用程序名称字符

应用程序名称必须在队列管理器字段的 **CodedCharSetId** 属性给出的字符集中; 有关此属性的详细信息，请参阅 队列管理器的属性 。

但是，如果应用程序作为 IBM MQ MQI client 运行，那么该应用程序名称必须使用客户机的字符集和编码。

为确保在队列管理器之间平稳转换应用程序名称 V 9.1.5 ，以及为允许通过资源监视主题进行应用程序资源监视，应用程序名称应仅包含单字节可打印字符。

注: V 9.1.5 您还应该避免在应用程序名称中使用正斜杠与和 (&) 字符。

这样可将名称限制为:

- 字母数字字符: A-Z， a-z 和 0-9

  注: 在使用 EBCDIC 片假名的系统上，应用程序名称中不应使用小写字符 a-z。
- 空格字符
- V 9.1.5 EBCDIC 中不变的可打印字符: + < = > % * ' ( ) , _ - . : ; ?

如何设置字符

下表汇总了在 IBM MQ 支持的各种语言中选择应用程序名称的方式。 名称选择方式按优先顺序排列（优先级最高的方式位于最前面）。

	C 绑定和客户机	Java 绑定和客户机	JMS 绑定和客户机	受管 .NET 客户机	非受管 .NET 绑定和客户机	受管 XMS 客户机	非受管 .XMS 绑定和客户机
连接属性覆盖		Java 连接属性覆盖		.NET 连接属性覆盖	.NET 连接属性覆盖		

	C 绑定和客户机	Java 绑定和客户机	JMS 绑定和客户机	受管 .NET 客户机	非受管 .NET 绑定和客户机	受管 XMS 客户机	非受管 .XMS 绑定和客户机
被覆盖的属性		Java 被覆盖的属性		.NET 被覆盖的属性	.NET 被覆盖的属性		
MQEnvironment		Java MQ 环境		.NET MQEnvironment	.NET MQEnvironment		
连接工厂属性			连接工厂属性			连接工厂属性	连接工厂属性
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	连接选项						
环境变量	环境变量				环境变量		环境变量
mqclient.ini （仅适用于客户机连接）	客户机连接				客户机连接		客户机连接
Java 类名		Java 类名	Java 类名				
缺省名称	缺省名称			.NET 缺省名称	.NET 缺省名称	.NET 缺省名称	.NET 缺省名称

注: “C 绑定和客户机”列也适用于以下编程语言：

- COBOL
- 汇编程序
- Visual Basic

编程语言连接

对于解析为 C 语言和其他编程语言的 IBM MQ 库的应用程序，可以使用以下方式提供应用程序名称。

连接方法按优先顺序列出（从优先级最高的方法开始）。

Multi **连接选项**

ULW 已将两个新字段添加到 MQCNO，并且 **Version** 编号已增至 7。 有关更多信息，请参阅 MQCNO。

注: z/OS 仅限使用 IBM MQ classes for JMS 或 IBM MQ classes for Java 以及客户机方式连接的应用程序才能在连接到 IBM MQ for z/OS 队列管理器时设置应用程序名称。

ULW **环境变量**

如果您还未选择应用程序名称，那么可以使用以下环境变量 *MQAPPLNAME* 来标识到队列管理器的连接。 例如：

```
export MQAPPLNAME=ExampleApplName
```

请参阅 环境变量描述 以获取更多信息。

请注意，将仅使用前 28 个字符，并且这些字符不能全部为空。

注： 该属性仅适用于受支持的编程语言、非受管 .NET 连接和非受管 XMS 连接。

ULW 客户机配置文件

如果您尚未选择应用程序名称，并且连接是客户机连接，那么可以在客户机配置文件（例如 `mqclient.ini`）中指定以下内容来标识与队列管理器的连接。

```
Connection:
    ApplName=ExampleApplName
```

注意：

1. 仅使用前 28 个字符，并且这些字符不能全部为空。
2. 该属性仅适用于受支持编程语言的客户机连接、非受管 .NET 连接和非受管 XMS 连接。

请参阅使用配置文件配置客户机以获取配置文件示例。

缺省名称

如果您还未选择应用程序名称，那么将继续使用缺省名称，缺省名称包含操作系统所显示的路径和可执行文件名称。 有关更多信息，请参阅 PutApplName。

受管 .NET 应用程序

受管 .NET 应用程序使用以下方式提供应用程序名称。

连接方法按优先顺序列出（从优先级最高的方法开始）。

连接属性覆盖

您可以使用以下方式向应用程序提供连接详细信息覆盖文件：

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

`overrideConnectionDetailsFile` 所指定的文件包含以 `mqj` 为前缀的属性的列表。 应用程序必须定义 *mqj.APPNAME* 属性，而 *mqj.APPNAME* 属性值指定用于标识到队列管理器的连接的名称。

将仅使用名称的前 28 个字符。 例如：

```
mqj.APPNAME=ExampleApplName
```

被覆盖的属性

已使用值 *APPNAME* 定义了常量 **MQC.APPNAME_PROPERTY**。 您现在可以将此属性传递到 **MQQueueManager** 构造函数（仅使用名称的前 28 个字符）。 例如：

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

请参阅 第 541 页的『.NET 中的受管和非受管操作』 以获取更多信息。

MQEnvironment

AppName 属性将添加到 **MQEnvironment** 类，并且仅使用前 28 个字符。 例如：

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

缺省名称

如果您还未通过上文中提及的任何方式提供应用程序名称，那么应用程序名称将自动设置为可执行文件名称（以及适合的路径）。

XMS 应用程序

连接方法按优先顺序列出（从优先级最高的方法开始）。

连接工厂属性

与 JMS 类似，XMS 应用程序可以使用 *XMSC.WMQ_APPLICATIONNAME* 属性（“*XMSC_WMQ_APPNAME*”）在连接工厂上提供应用程序名称。 最多可以指定 28 个字符。

请参阅第 548 页的『XMS .NET 创建受管对象』和第 555 页的『XMS 消息的属性』，以获取更多信息。

JMSAdmin

在管理工具中，此属性简称为“**APPLICATIONNAME**”或“**APPNAME**”。

Java 和 JMS 绑定应用程序

连接方法按优先顺序列出（从优先级最高的方法开始）。

ULW Java 和 JMS 客户机应用程序已经可以指定应用程序名称，并且在 IBM MQ for Multiplatforms 上通过使用 MQCNO **ApplName** 字段已将此行为扩展到绑定应用程序。

连接属性覆盖

已将 **Application name** 属性添加到可覆盖的连接属性的列表中。 请参阅使用 IBM MQ 连接属性覆盖以获取更多信息。

注意: 连接属性以及使用“连接属性覆盖”文件的方式针对 IBM MQ classes for Java 和 .NET 是相同的。

被覆盖的属性

已使用值 *APPNAME* 定义了常量 **MQC.APPNAME_PROPERTY**。 您现在可以将此属性传递到 **MQQueueManager** 构造函数（仅使用名称的前 28 个字符）。 请参阅在 IBM MQ classes for Java 中使用连接属性覆盖，以获取更多信息。

MQEnvironment

AppName 属性将添加到 **MQEnvironment** 类，并且仅使用前 28 个字符。

请参阅 第 303 页的『为 IBM MQ classes for Java 设置 IBM MQ 环境』 以获取更多信息。

Java 类名

如果您还未通过上文中提及的任何方式提供应用程序名称，那么将从主类名派生应用程序名称。

请参阅 第 303 页的『为 IBM MQ classes for Java 设置 IBM MQ 环境』 以获取更多信息。

相关概念

第 303 页的『为 IBM MQ classes for Java 设置 IBM MQ 环境』
要让应用程序以客户机模式连接到队列管理器，该应用程序必须指定通道名称、主机名和端口号。

相关参考

MQCNO

消息的设计方法

帮助您设计消息的注意事项，包括有关选择器和消息属性的注意事项。

在设计阶段需要考虑的一些事项

您在使用 MQI 调用将消息放在队列上时创建消息。 作为调用的输入，您在消息描述符 (MQMD) 中提供控制消息，以及要发送到其他程序的数据。 但是在设计阶段，您需要考虑以下内容，因为它们会影响创建消息的方式：

要使用的消息类型

您是否在设计可以在其中发送消息，然后不采取进一步操作的简单应用程序？ 或者，您是否在请求问题的回复？ 如果您是在提问，那么可能会在消息描述符中包含要接收回复的队列的名称。

您是否希望请求和回复消息同步？ 这暗示您为回应请求的回复设置时间段，如果在该时期内没有收到回复，那么将其视为错误。

或者，您是否首选异步工作，以便您的进程不必依赖于特定事件（如普通计时信号）的发生？

另一个注意事项是您的所有消息是否都在一个工作单元内。

向消息分配不同优先级

您可以向各消息分配优先级值，并且定义队列，以便其按照消息的优先级来维护这些消息。 如果这样做，那么在其他程序从队列检索消息时，始终获取优先级最高的消息。 如果队列没有按优先级顺序维护其消息，那么从该队列检索消息的程序将按照消息添加到队列的顺序来检索这些消息。

程序也可以使用消息放到队列上时队列管理器分配的标识来选择消息。 或者，可以为各消息生成自己的标识。

重新启动队列管理器对消息的影响

队列管理器保留所有持久消息，从而在必要时从 IBM MQ 日志文件恢复这些消息（在其重新启动时）。 不会保留非持久消息和临时动态队列。 您不希望丢弃的任何消息必须在创建时定义为持久消息。 在 UNIX and Linux 系统上为 IBM MQ for Windows 或 IBM MQ 编写应用程序时，请确保您知道如何在日志文件分配方面设置系统，以降低设计将运行到日志文件限制的应用程序的风险。

z/OS 由于共享队列（仅在 IBM MQ for z/OS 上可用）上的消息存放在耦合设施 (CF) 中，因此只要 CF 保持可用，在队列管理器重新启动前后便会保留非持久消息。 如果 CF 发生故障，那么非持久消息会丢失。

向消息接收方提供有关您自身的信息

通常，队列管理器会设置用户标识，但是经过适当授权的应用程序也可以设置此字段，以便您能够包含自己的用户标识，以及接收程序可用于记帐或安全性用途的其他信息。

接收队列的数量

Multi 如果消息可能需要放在若干队列上，那么可以发布到主题或分发列表。

z/OS 如果消息可能需要放在若干队列上，那么可以发布到主题。

选择器和消息属性

消息可以与元数据以及主要消息有效内容关联。 这些消息属性在提供额外数据方面可能非常有用。

此额外数据有两个重要方面需要了解：

- 属性不受 Advanced Message Security (AMS) 保护。 如果您想使用 AMS 来保护数据，那么将它放入有效内容，而不是消息属性中。

- 可使用属性来执行消息的选择。

尤其值得注意的是，使用选择器会打破先进先出的标准消息约定。 由于已对此工作负载优化队列管理器，出于性能原因，不建议提供复杂的选择器。 队列管理器不储存消息属性的索引，因此搜索消息必须是线性搜索。 队列越深，选择器越复杂，与消息匹配的选择器负面影响性能的概率越低。

如果需要复杂选择，建议使用任何应用程序或处理引擎（如 IBM Integration Bus）将消息过滤到不同目标。 或者，使用主题层次结构可能也非常有用。

注: IBM MQ classes for Java 不支持使用选择器，如果您确实希望使用选择器，应通过 JMS API 来完成。

应用程序设计和性能注意事项

低劣的程序设计可以通过许多方式影响性能。 难以检测这些方式，因为程序可能本身执行良好，但会影响其他任务的性能。 本主题中说明了特定于进行 IBM MQ 调用的程序的若干问题。

以下是帮助设计高效应用程序的若干构想：

- 设计应用程序，以便处理与用户的思考时间并行进行：
  - 显示面板并允许用户在应用程序仍在初始化的同时开始输入。
  - 从不同服务器并行获取所需的数据。
- 如果将要复用连接和队列而不是重复打开和关闭、连接以及断开连接，请将其保持打开。
- 但是，仅放置一条消息的服务器应用程序应使用 MQPUT1。
- 针对大小介于 4 KB 和 100 KB 之间的消息优化队列管理器。 超大消息效率低下；发送 100 条 1 MB 的消息比发送单条 100 MB 的消息可能更好。 超小消息也效率低下。 队列管理器对于单字节消息和对于 4 KB 消息执行相同的工作量。
- 将消息保留在同一个工作单元内，以便可同时对其进行落实或退回。
- 对无需可恢复的消息使用非持久性选项。
- 如果需要将消息发送到许多目标队列，请考虑使用分发列表。

消息长度的影响

消息中的数据量可以影响处理该消息的应用程序的性能。 要使应用程序达到最佳性能，只发送消息中必不可少的数据。 例如，在记入银行帐户借方的请求中，可能需要从客户机传递到服务器应用程序的唯一信息是帐号和借记金额。

消息持久性的影响

通常会记录持久消息。 记录消息会降低应用程序的性能，因此应只对必不可少的数据使用持久消息。 如果消息中的数据可以在队列管理器停止或失败时废弃，请使用非持久消息。

z/OS 如果没有足够的恢复日志空间来记录操作，那么将阻止针对持久消息执行的 MQPUT 和 MQGET 操作。 在队列管理器作业日志中通过消息 CSQJ110E 和 CSQJ111A 指示此类条件。 确保已运行监视进程以便可管理和消除此类条件。

搜索特定消息

MQGET 调用通常检索来自队列的第一条消息。 如果使用消息描述符中的消息和相关标识（*MsgId* 和 *CorrelId*）来指定特定消息，那么队列管理器必须搜索队列，直至找到该消息为止。 以此方式使用 MQGET 调用会影响应用程序的性能。

包含不同长度的消息的队列

如果应用程序无法使用固定长度的消息，请动态增大和缩小缓冲区以适合典型消息大小。 如果应用程序发出 MQGET 调用，该调用由于缓冲区大小而失败，那么将返回消息数据的大小。 向应用程序添加代码，以便相应地调整缓冲区大小并重新发出 MQGET 调用。

注: 如果不显式设置 **MaxMsgLength** 属性，那么它缺省为 4 MB，如果这用于影响应用程序缓冲区大小，那么可能效率非常低下。

同步点的频率

在同步点内发出大量 MQPUT 或 MQGET 调用而没有将其落实的程序可能会导致性能问题。 受影响队列可能会充满当前不可访问的消息，而其他任务可能在等待获取这些消息。 这在存储以及在与尝试获取消息的任务捆绑的线程方面具有影响。

MQPUT1 调用的使用

仅在您要将单条消息放在队列上的情况下，使用 MQPUT1 调用。 如果要放置多条消息，请使用 MQOPEN 调用，后跟一系列 MQPUT 调用和单个 MQCLOSE 调用。

使用中的线程数

Windows 对于 IBM MQ for Windows，应用程序可能需要大量线程。 每个队列管理器进程分配有最大可允许的应用程序线程数。

应用程序可能使用过多线程。 请考虑应用程序是否将此可能性考虑在内，并采取操作停止或报告此类情况的发生。

将持久消息放在同步点下

应在同步点下放置和获取持久消息。 这是因为在同步点外获取持久消息时，如果获取失败，那么应用程序无法知道是否已从队列取出消息，以及是否在获取消息后，该消息也已丢失。 在同步点下获取持久消息时，如果任何操作失败，那么会回滚事务且持久消息不会丢失，因为它仍在队列上。

同样，在放置持久消息时，请将其放在同步点下。 在同步点下放置和获取持久消息的另一个原因是 IBM MQ 中的持久消息代码针对同步点进行了大量优化。 因此，在同步点下放置和获取持久消息比在同步点外放置和获取持久消息更快速。

如果应用程序确实将持久消息放在同步点之外，队列管理器将检查是否可以代表应用程序创建隐式同步点。 如果队列管理器可以执行此操作，那么可以包括放置在同步点中，并将其自动提交。 请参阅 第 792 页的『多平台上的隐式同步点』 以获取更详细的描述。

但是，在同步点外放置和获取非持久消息更快速，因为 IBM MQ 中的非持久性代码针对位于同步点外进行了优化。 放置和获取持久消息以磁盘速度进行，因为持久消息持久存储到磁盘。 但是，放置和获取非持久消息以 CPU 速度进行，因为没有涉及任何磁盘写操作，即使在使用同步点时也如此。

如果应用程序正在获取消息，并且事先不知道这些消息是否持久，那么可以使用 GMO 选项 MQGMO_SYNCPOINT_IF_PERSISTENT。

高级应用程序的设计方法

当设计更高级的应用程序时，有些方法可能要予以考虑，例如等待消息、关联回复、设置和使用上下文信息、自动启动应用程序、生成报告以及在使用集群时移除消息亲缘关系。

对于简单的 IBM MQ 应用程序，您需要决定要在应用程序中使用哪些 IBM MQ 对象，以及要使用哪些类型的消息。 对于更高级的应用程序，可能要使用以下部分中介绍的某些方法。

等待消息

为队列提供服务的程序可以通过以下方式等待消息:

- 等待直至消息到达，或者指定的时间间隔到期（请参阅第 737 页的『等待消息』）。
- z/OS 仅限在 IBM MQ for z/OS 上，设置信号，以便在消息到达时程序接获通知。 有关更多信息，请参阅 第 737 页的『发信号』。
- 建立要在消息到达时驱动的回调出口；请参阅 第 34 页的『IBM MQ 消息的异步使用』。
- 在队列上进行定期调用以查看消息是否已到达（轮询）。 通常不建议如此，因为它可能会影响性能。

关联回复

在 IBM MQ 应用程序中，当程序收到请求其执行工作的消息时，该程序通常会向请求者发送一条或多条回复消息。

要帮助请求者将这些回复与其原始请求相关联，应用程序可以在各消息的描述符中设置相关标识字段。 然后，程序将请求消息的消息标识复制到其回复消息的相关标识字段中。

设置和使用上下文信息

上下文信息用于将消息与生成这些消息的用户相关联，以及用于识别生成消息的应用程序。 此类信息对于安全性、记述、审计和问题确定有用。

创建消息时，您可以指定一个用于请求队列管理器将缺省上下文信息与您的消息相关联的选项。

有关使用和设置上下文信息的更多信息，请参阅第 39 页的『消息上下文』。

自动启动 IBM MQ 程序

使用 IBM MQ 触发以在消息到达队列时自动启动程序。

可以在队列上设置触发条件，以便程序开始处理该队列：

- 每次消息到达队列时
- 当第一条消息到达队列时
- 当队列上的消息数达到预定义数量时

有关触发的更多信息，请参阅第 795 页的『使用触发器启动 IBM MQ 应用程序』。 触发只是自动启动程序的一种方式。 例如，可以使用非 IBM MQ 工具在计时器上自动启动程序。

Multi 在 多平台 上，IBM MQ 可以将服务对象定义为在队列管理器启动时启动 IBM MQ 程序；请参阅服务对象。

生成 IBM MQ 报告

您可以在应用程序内请求以下报告：

- 异常报告
- 到期报告
- 确认到达 (COA) 报告
- 确认传送 (COD) 报告
- 肯定操作通知 (PAN) 报告
- 否定操作通知 (NAN) 报告

这些报告在第 16 页的『报告消息』中进行了描述。

集群和消息亲缘关系

在开始使用具有同一队列的多个定义的集群之前，请检查应用程序以查看是否有任何需要交换相关消息的应用程序。

在集群内，消息可以路由到托管相应队列的实例的任何队列管理器。 因此，具有消息亲缘关系的应用程序的逻辑可能被打乱。

例如，您可能具有两个依赖于在其之间以问答形式流动的一系列消息的应用程序。 所有问题都发送到同一队列管理器且所有答案都发回到其他队列管理器可能非常重要。 在此情况下，重要的是工作负载管理例程不要将消息发送到只是碰巧托管相应队列的实例的任何队列管理器。

如有可能，移除亲缘关系。 移除消息亲缘关系可提高应用程序的可用性和可扩展性。

有关更多信息，请参阅处理消息亲缘关系。

IBM i 应用程序的设计和性能注意事项

使用此信息来了解应用程序设计、线程和存储如何影响性能。

此信息分为两个部分:

- 第 52 页的『应用程序设计注意事项』
- 第 52 页的『特定性能问题』

应用程序设计注意事项

低劣的程序设计可以通过许多方式影响性能。 这些问题可能难以检测，因为程序可能看似执行良好，但同时会影响其他任务的性能。 以下部分中说明了特定于进行 IBM MQ for IBM i 调用的程序的若干问题。

有关应用程序设计的更多信息，请参阅 第 40 页的『IBM MQ 应用程序的设计注意事项』。

消息长度的影响
虽然 IBM MQ for IBM i 允许消息最多容纳 100 MB 数据，但是消息中的数据量会影响处理该消息的应用程序的性能。 要从应用程序获得最佳性能，请在消息中仅发送必不可少的数据；例如，在记入银行帐户借方的请求中，可能需要从客户机传递到服务器应用程序的唯一信息是帐号和借记金额。

消息持久性的影响
持久消息会记入日志。 对消息进行日志记录会降低应用程序的性能，因此应仅对必不可少的数据使用持久消息。 如果消息中的数据可以在队列管理器停止或失败时废弃，请使用非持久消息。

搜索特定消息
MQGET 调用通常检索来自队列的第一条消息。 如果在消息描述符中使用消息和相关标识（*MsgId* 和 *CorrelId*）来指定特定消息，那么队列管理器必须搜索该队列，直至找到该消息为止。 以此方式使用 MQGET 调用会影响应用程序的性能。

包含不同长度的消息的队列
如果队列上的消息长度不同，那么要确定消息的大小，应用程序可以使用 MQGET 调用并将 *BufferLength* 字段设置为零，这样即使调用失败，也会返回消息数据的大小。 然后，应用程序可以重复调用，指定其在第一个调用中测量的消息标识和正确大小的缓冲区。 但是，如果有其他应用程序为同一队列提供服务，那么您可能会发现应用程序的性能降低，因为其第二个 MQGET 调用花费时间搜索其他应用程序已在两个调用之间的时间内检索到的消息。

如果应用程序无法使用固定长度的消息，那么此问题的另一项解决方案是使用 MQINQ 调用来查找队列可以接受的消息的最大大小，然后在 MQGET 调用中使用该值。 队列的最大消息大小将存储在队列的 **MaxMsgLen** 属性中。 但是，此方法可能使用大量存储器，因为此队列属性的值可以是 IBM MQ for IBM i 允许的最大值, 该值可能大于 2 GB。

同步点的频率
在同步点内发出大量 MQPUT 调用而没有将其落实的程序可能会导致性能问题。 受影响队列可能会充满当前不可用的消息，而其他任务可能在等待获取这些消息。 此问题在存储以及在与尝试获取消息的任务捆绑的线程方面具有影响。

MQPUT1 调用的使用
仅在您要将单条消息放在队列上的情况下，使用 MQPUT1 调用。 如果要放置多条消息，请使用 MQOPEN 调用，后跟一系列 MQPUT 调用和单个 MQCLOSE 调用。

使用中的线程数
应用程序可能需要许多线程。 每个队列管理器进程分配有最大可允许的线程数。 如果某些应用程序有问题，那么可能是由于其设计使用过多线程。 请考虑应用程序是否将此可能性考虑在内，并采取操作停止或报告此类情况的发生。 IBM i 允许的最大线程数为 4,095。 但是，缺省值为 64。 IBM MQ 使最多 63 个线程可供其进程使用。

特定性能问题

此部分说明存储和低性能的问题。

存储问题
如果接收到系统消息 `CPF0907. Serious storage condition may exist`，那么可能是您正在填充与 IBM MQ for IBM i 队列管理器相关联的空间。

应用程序或 IBM MQ for IBM i 是否运行缓慢?

如果您的应用程序正在缓慢运行，那么可能指示其处于循环中，或者正在等待不可用的资源。 此缓慢运行也可能由性能问题导致。 可能是因为系统的运行接近其容量限制。 这类问题可能在系统负载的峰值时间最严重，通常在上午的中间时段和下午的中间时段。 （如果网络跨越了多个时区，那么峰值系统负载对您来说可能会在其他时间出现。）

如果您发现性能降低不是取决于系统负载，而是有时在系统轻负载时发生，那么可能问题在于设计低劣的应用程序。 此问题可能显现为仅当访问特定队列时才会发生的问题。

QTOTJOB 和 QADLTOTJ 是值得调查的系统值。

以下症状可能指示 IBM MQ for IBM i 正在缓慢运行：

- 如果系统缓慢响应 MQSC 命令。
- 如果队列深度重复显示，表明在为应用程序缓慢处理队列，对该应用程序您预计将要有大量的队列活动。
- IBM MQ 跟踪是否正在运行?

Linux Linux on POWER Systems - Little Endian 应用程序的设计注意事项

由于 Linux on POWER Systems - Little Endian 仅支持 64 位应用程序，IBM MQ 中不提供对 32 位应用程序的支持。

相关概念

第 40 页的『IBM MQ 应用程序的设计注意事项』

当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

z/OS z/OS 应用程序的设计和性能注意事项

应用程序设计是影响性能的最重要因素之一。 使用本主题来了解性能中涉及的某些设计因素。

低劣的程序设计可以通过许多方式影响性能。 这些问题可能难以检测，因为程序可能看似执行良好，但同时会影响其他任务的性能。 以下部分中说明了特定于进行 MQI 调用的程序的若干问题。

有关应用程序设计的更多信息，请参阅 第 40 页的『IBM MQ 应用程序的设计注意事项』。

消息长度的影响

虽然 IBM MQ for z/OS 允许消息最多容纳 100 MB 数据，但是消息中的数据量会影响处理该消息的应用程序的性能。 要使应用程序达到最佳性能，只发送消息中必不可少的数据。 例如，在记入银行帐户借方的请求中，可能需要从客户机传递到服务器应用程序的唯一信息是帐号和借记金额。

消息持久性的影响

系统会记录持久消息。 记录消息会降低应用程序的性能，因此应只对必不可少的数据使用持久消息。 如果消息中的数据可以在队列管理器停止或失败时废弃，请使用非持久消息。

持久消息的数据写入到日志缓冲区。 在以下情况下，这些缓冲区写入到日志数据集：

- 发生落实
- 在同步点外获取或放置消息
- WRTHRSH 缓冲区已填满

相比于针对每个工作单元处理一条消息或在同步点外处理消息，在一个工作单元中处理多条消息可能导致更少的输入/输出。

搜索特定消息

MQGET 调用通常检索来自队列的第一条消息。 如果您使用消息和相关标识（**MsgId** 和 **CorrelId**）在消息描述符中指定特定消息，队列管理器将搜索该队列直到找到该消息。 以此方式使用 MQGET 会影响应用程序的性能，因为要查找特定消息，IBM MQ 可能必须扫描整个队列。

您可以使用 **IndexType** 队列属性来指定您希望队列管理器维护索引，该索引可用于提高队列上 MQGET 操作的速度。 但是，维护索引会略微降低性能，因此仅当需要使用索引时才进行生成。 可以选择构建消息标识或相关标识的索引，也可以选择不为顺序检索消息的队列构建索引。 请尝试具有许多不同的键值，而不是许多键具有相同值。 例如 Balance1、Balance2 和 Balance3，而不是三个具有 Balance 的索引。 对于共享队列，必须具有正确的 **IndexType**。 有关 **IndexType** 队列属性的详细信息，请参阅 IndexType。

要通过使用索引队列避免影响队列管理器重新启动时间，请使用 CSQ6SYSP 宏中的 QINDXBLD(NOWAIT) 参数。 借此可以完成队列管理器重新启动，而不用等待队列索引构建完成。

有关 **IndexType** 属性和其他对象属性的完整描述，请参阅对象的属性。

包含不同长度的消息的队列

使用与消息的预期大小相匹配的缓冲区大小获取消息。 如果您收到指示消息太长的返回码，请获取更大的缓冲区。 当以此方式进行获取失败时，所返回的数据长度是未转换的消息数据的大小。 如果在 MQGET 调用中指定 MQGMO_CONVERT，并且数据在转换期间扩展，那么它可能仍不适合缓冲区大小，在此情况下需要进一步增大缓冲区的大小。

如果发出缓冲区长度为零的 MQGET，那么它将返回消息的大小，然后应用程序可以获取此大小的缓冲区并重新发出 get 调用。 如果有多个应用程序处理队列，那么在原始应用程序重新发出 get 调用时，其他应用程序可能已处理消息。 如果偶尔具有大型消息，那么可能需要仅为这些消息获取大缓冲区，并在处理消息后释放该缓冲区。 如果所有应用程序都具有大缓冲区，那么这应有助于减少虚拟存储器问题。

如果应用程序无法使用固定长度的消息，那么此问题的另一项解决方案是使用 MQINQ 调用来查找队列可以接受的消息的最大大小，然后在 MQGET 调用中使用该值。 队列的最大消息大小将存储在队列的 **MaxMsgL** 属性中。 但是，此方法可能使用大量存储空间，因为 **MaxMsgL** 的值可能高达 100 MB，这是 IBM MQ for z/OS 所允许的最大值。

注: 将大消息放入队列后，您可以降低 **MaxMsgL** 参数的值。 例如，您可以放置 100 MB 消息，然后将 **MaxMsgL** 设置为 50 个字节。 这意味着仍有可能获取比应用程序预期更大的消息。

同步点的频率

在同步点内发出许多 MQPUT 调用而没有将其落实的程序可能会导致性能问题。 受影响队列可能会充满当前不可用的消息，而其他任务可能在等待获取这些消息。 这在存储以及在与尝试获取消息的任务捆绑的线程方面具有影响。

通常，如果有多个应用程序处理队列，那么针对每个同步点在以下任一情况成立时通常会获取最佳性能

- 具有 100 条短消息（小于 1 KB），或者
- 针对更大的消息 (100 KB) 具有一条消息

。 如果仅有一个应用程序处理队列，那么针对每个工作单元必须具有更多消息。

您可以使用 **MAXUMSGS** 队列管理器属性来限制任务可以在单个恢复单元内获取或放入的消息数。 有关此属性的信息，请参阅 MQSC 命令中的 **ALTER QMGR** 命令。

MQPUT1 调用的优点

仅在您要将单条消息放在队列上的情况下，使用 MQPUT1 调用。 如果要放置多条消息，请使用 MQOPEN 调用，后跟一系列 MQPUT 调用和单个 MQCLOSE 调用。

队列管理器可以包含的消息数

本地队列

队列管理器可以容纳的本地消息数基本上是页集的大小。 您最多可以具有 100 个页集（虽然建议页集 0 和页集 1 用于系统相关对象和队列）。 可以使用具有扩展格式的页集并增大页集的容量。

共享队列

共享队列的容量取决于耦合设施 (CF) 的大小。 IBM MQ 使用基本存储单元是条目和元素的 CF 列表结构。 各消息作为包含关联 MQMD 及其他消息数据的 1 个条目和多个元素进行存储。 单条消息消耗的元素数取决于消息的大小以及在 MQPUT 时间生效的卸载规则（对于 CFLEVEL(5)）。 将消息数据卸载到 Db2 或 SMDS 时，需要的元素更少。 卸载消息后，消息数据访问更缓慢。 请参阅性能 Supportpac MP1H 以进一步比较性能和与消息卸载关联的 CPU 开销。

影响性能的因素

性能可能意味着消息的处理速度，并且还可能意味着每条消息所需的 CPU 用量。

影响消息处理速度的因素

对于持久消息，最大的影响是日志数据集的速度。 日志数据集的速度取决于它们所在的 DASD。 因此，应注意将日志数据集放在使用率低的卷上以减少争用。 当每个 I/O 写入多个页面时，对 MQ 日志进行条带分割可提高日志性能。 Z High Performance Fibre (zHPF) 连接在 I/O 子系统繁忙时也能显著改善 I/O 响应时间。

当存在获取和放置消息的请求时，在请求期间会锁定对队列的访问以保留队列的完整性。 出于规划目的，请考虑对整个请求进行锁定的队列。 因此，如果放置的时间为 100 微秒，并且每秒有超过 10,000 个请求，那么可能会遇到延迟。 在实践中可能会实现比这更好的性能，但它是理想的通用规则。 可以使用不同队列来提高性能。

此问题的可能原因可以是:

- 使用每个 CICS 事务使用的普通应答队列
- 每个 CICS 事务获取队列的唯一回复
- CICS 区域的队列应答和 CICS 区域中的所有事务都使用此队列。

应答取决于每秒的请求数和请求的响应时间。

如果必须从页集读取消息，那么与消息位于缓冲池中时相比，这些消息速度将更慢。 如果您具有的消息超过缓冲池大小，那么消息将溢出到磁盘。 因此，您需要确保缓冲池对于短期消息足够大。 如果您具有经过多个小时后处理的消息，那么它们可能会溢出到磁盘，因此应执行 get 调用，以使这些消息相比于其位于缓冲池中时速度更慢。

对于共享队列，消息的速度取决于耦合设施的速度。 物理处理器中的 CF 可能比外部 CF 速度更快。 CF 响应时间取决于 CF 的繁忙程度。 例如在 Hursley 系统上，当 CF 繁忙程度为 17% 时，响应时间为 14 微秒。 当 CF 繁忙程度为 95% 时，响应时间为 45 微秒。

如果 MQ 请求使用大量 CPU，那么这可能会影响消息的处理速度。 因为如果逻辑分区 (LPAR) 受 CPU 约束，那么应用程序将延迟，等待 CPU。

每条消息的 CPU 用量

一般而言，较大的消息使用较多 CPU，因此请尽量避免大型 (x MB) 消息。

从队列获取特定消息时，队列应建立索引，以使队列管理器可以直接转至消息（并因此避免可能对队列进行整体扫描）。 如果队列未建立索引，那么将从开始扫描队列来查找消息。 如果队列上有 1000 条消息，那么可能必须扫描全部 1000 条消息。 结果是许多不必要的 CPU 使用。

由于消息的加密，使用 TLS 的通道具有额外成本。

在 MQ V7 中，除了 **CORRELID** 或 **MSGID** 之外，还可以通过选择器字符串选择消息。 必须查看每条消息，因此如果队列上有许多消息，那么此操作的成本非常高昂。

应用程序执行 OPEN PUT PUT CLOSE 比执行 PUT1 PUT1 更高效。

在 CICS 中触发

当已触发队列的消息的消息到达率较低时，首先使用触发器比较高效。 当消息到达率为每秒 10 条以上消息时，触发第一个事务，然后由该事务处理消息并获取下一条消息（依此类推）更高效。 如果消息在短期（假设 0.1 和 1 秒之间）内未到达，那么事务结束。 在高吞吐量情况下，可能需要运行多个事务来处理消息并防止消息堆积。 对于产生的每条触发器消息，这要求放置和获取触发器消息，实际会使消息的成本加倍。

支持的连接数或并发用户数

每个连接在队列管理器内使用虚拟储存器，因此并发用户越多，所使用的存储器就越多。 如果需要超大缓冲池和大量用户，那么您可能会受制于虚拟存储器，并且可能需要减小缓冲池的大小。

如果使用安全性，那么队列管理器会将信息长期高速缓存在队列管理器中。 队列管理器内使用的虚拟存储量会受影响。

CHINIT 最多可以支持大约 10,000 个连接。 这受到虚拟存储器的限制。 如果连接使用更多存储空间（例如，通过 TLS 使用），则每个连接的存储空间会增加，这意味着 **CHINIT** 可支持的连接数会减少。 如果您要处理大消息，那么这些消息将需要更多存储空间以用于 **CHINIT** 中的缓冲区，因此 **CHINIT** 可支持的消息数会减少。

与远程队列管理器的连接比客户机连接更高效。 例如，每个 MQ 客户机请求需要两个网络流（一个用于请求，另一个用于响应）。 利用与远程队列管理器的通道，在返回响应之前，网络上可能有 50 次发送。 如果您考虑的是大型客户机网络，那么在分布式设备上使用集中器队列管理器并有一条出入于集中器的通道可能更高效。

影响性能的其他因素

日志数据集的大小应至少为 1000 个柱面。 如果日志小于此大小，那么检查点活动可能过于频繁。 在繁忙系统上，检查点通常应为每 15 分钟或更长时间，在超高吞吐量情况下，可能小于此时间。 当出现检查点时，将会扫描缓冲池并将“旧”消息和已更改页面写入到磁盘。 如果检查点过于频繁，那么这会影响性能。 LOGLOAD 的值也可影响检查点频率。 如果队列管理器异常结束，那么在重新启动时，可能必须回读至 3 个检查点。 最佳检查点时间间隔是在采用检查点时的活动与队列管理器重新启动时可能需要读取的日志数据量之间达到平衡。

启动通道时，将会招致重大开销。 通常最好启动通道并使其保持连接，而不是频繁启动和停止通道。

相关信息

MP1K：IBM MQ for z/OS 9.0 性能报告

z/OS IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序

此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

- 要在 IMS 应用程序中使用同步点和 MQI 调用，请参阅第 57 页的『使用 IBM MQ 编写 IMS 应用程序』。
- 要编写使用 IBM MQ - IMS 网桥的应用程序，请参阅第 60 页的『编写 IMS 桥接应用程序』。

使用以下链接了解有关 IMS 和 IBM MQ for z/OS 上的 IMS 桥接应用程序的更多信息：

- 第 57 页的『使用 IBM MQ 编写 IMS 应用程序』
- 第 60 页的『编写 IMS 桥接应用程序』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

使用 IBM MQ 编写 IMS 应用程序

在 IMS 应用程序中使用 IBM MQ 时还有其他注意事项。这些注意事项包括可以使用哪些 MQ API 调用以及用于同步点的机制。

使用以下链接可了解有关在 IBM MQ for z/OS 上编写 IMS 应用程序的更多信息:

- 第 57 页的『IMS 应用程序中的同步点』
- 第 58 页的『IMS 应用程序中的 MQI 调用』

限制

存在使用 IMS 适配器的应用程序可以使用 IBM MQ API 调用的限制。

以下 IBM MQ API 调用在使用 IMS 适配器的应用程序中不受支持:

- MQCB
- MQCB_FUNCTION
- MQCTL

相关概念

第 60 页的『编写 IMS 桥接应用程序』
本主题包含有关编写应用程序以使用 IBM MQ - IMS 网桥的应用程序的信息。

IMS 应用程序中的同步点

在 IMS 应用程序中，通过使用 IMS 调用（如 GU，获取唯一）IOPCB 和 CHKP（检查点）来建立同步点。

要回退从先前检查点开始的所有更改，可以使用 IMS ROLB（回滚）调用。 有关更多信息，请参阅 IMS 文档中的 ROLB 调用 。

队列管理器是两阶段落实协议的参与者；IMS 同步点管理器是协调者。

所有打开的句柄由 IMS 适配器在同步点时关闭（除了批处理或非消息驱动的 BMP 环境）。 这是因为执行 MQCONN、MQCONNX 和 MQOPEN 调用（而非 MQPUT 或 MQGET 调用）时不同用户可以启动下一个工作单元，并且执行 IBM MQ 安全性检查。

但在等待输入 (WFI) 或伪等待输入 (PWFI) 环境中，只有收到下一条消息或者将 QC 状态码返回至应用程序之后，IMS 才会通知 IBM MQ 关闭句柄。 如果应用程序在 IMS 区域内等待，并且所有句柄都属于已触发队列，那么由于队列处于打开状态，将不会发生触发。 因此，在为下一消息执行 GU 调用 IOPCB 之前，在 WFI 或 PWFI 环境中运行的应用程序应该以显式方式使用 MQCLOSE 关闭队列句柄。

如果 IMS 应用程序（BMP 或 MPP）发出 MQDISC 调用，那么将关闭打开的队列，但不会生成隐式同步点。 如果应用程序正常结束，所有打开的队列将关闭，并会发生隐式提交。 如果应用程序异常结束，所有打开的队列将关闭，并会发生隐式回退。

IMS 应用程序中的 *MQI* 调用

通过此信息了解 MQI 调用在服务器应用程序和查询应用程序上的使用。

本节介绍在以下 IMS 应用程序类型中使用 MQI 调用：

- 第 58 页的『服务器应用程序』
- 第 60 页的『查询应用程序』

服务器应用程序

此处为 MQI 服务器应用程序模型的概述：

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

样本程序 CSQ4ICB3 显示了在 C/370 中使用此型号实施 BMP。 程序首先与 IMS 建立通信，然后与 IBM MQ 建立通信：

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

Return
```

IMS 初始化确定是作为消息驱动的 BMP 调用程序还是作为面向批处理的 BMP 调用程序，同时相应地控制 IBM MQ 队列管理器连接和队列句柄：

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

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
```

```
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

IBM MQ 初始化连接到队列管理器并打开队列。 在消息驱动的 BMP 中，该调用发生在获取每个 IMS 同步点之后，在面向批处理的 BMP 中，仅在程序启动期间进行此调用。

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

Return to calling function
```

在 MPP 中实施服务器型号受到每次调用 MPP 只处理单个工作单元这一事实的影响。 这是因为在获取同步点 (GU) 时，连接和队列句柄会关闭并会发送下一条 IMS 消息。 以下方法可以在一定程度上克服此限制：

- **在一个工作单元内处理多条消息**

  这包括在循环中：

  – 正在读取消息
  – 处理所需的更新
  – 放置应答

  直到所有消息均处理完成，或者已处理最大消息数量，此时将获取同步点。

  该方法只适用于特定类型的应用程序（例如，简单的数据库更新或查询）。 尽管 MQI 应答消息可以通过正在处理的 MQI 消息的发起方权限放入，但任何 IMS 资源更新的安全含义都需要小心处理。

- **每个 MPP 调用处理一条消息并确保多个 MPP 调度处理所有可用消息**。

  使用 IBM MQ IMS 触发器监视器程序 (CSQQTRMN) 在 IBM MQ 队列上存在消息并且没有应用程序为其提供服务时调度 MPP 事务。

  如果触发器监视器启动 MPP，队列管理器名称和队列名称将传递到程序，如以下 COBOL 代码段所示：

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
```

```
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

尽管 BMP 不能使用 CSQQTRMN 触发，但此服务器型号（长时间运行的任务）最好在批处理区域内受支持。

查询应用程序

一般的 IBM MQ 应用程序按照以下过程启动查询或更新工作：

- 收集用户数据
- 放置一条或多条 IBM MQ 消息
- 获取应答消息（可能需要等待）
- 向用户提供响应

因为放入 IBM MQ 队列的消息在提交之前不可用于其他 IBM MQ 应用程序，所以这些消息必须放在同步点外，或者 IMS 应用程序必须分为两个事务。

如果查询涉及放置单条消息，您可以使用无同步点选项；但是，当查询更为复杂，或者涉及资源更新时，如果出现故障并且您不使用同步点，您可能会遇到一致性问题。

要解决此问题，您可以使用程序间消息开关通过 MQI 调用拆分 IMS MPP 事务; 请参阅 *IMS* 系统间通信 *(ISC)* 以获取有关此问题的信息。 这使得查询程序可以在 MPP 中实施：

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

编写 IMS 桥接应用程序

本主题包含有关编写应用程序以使用 IBM MQ - IMS 网桥的应用程序的信息。

有关 IBM MQ - IMS 网桥的信息，请参阅 IMS 网桥。

使用以下链接了解有关在 IBM MQ for z/OS 上编写 IMS 桥接应用程序的更多信息：

- 第 61 页的『IMS 网桥如何处理消息』
- 第 833 页的『通过 IBM MQ 编写 IMS 事务程序』

相关概念

第 57 页的『使用 IBM MQ 编写 IMS 应用程序』

在 IMS 应用程序中使用 IBM MQ 时还有其他注意事项。这些注意事项包括可以使用哪些 MQ API 调用以及用于同步点的机制。

IMS 网桥如何处理消息

使用 IBM MQ - IMS 网桥向 IMS 应用程序发送消息时，需要以特殊格式构造消息。

您还必须将消息放入定义了存储类的 IBM MQ 队列，存储类用于指定目标 IMS 系统的 XCF 组和成员名称。这些队列称为 MQ-IMS 网桥队列，或简称为**网桥**队列。

如果网桥队列定义了 QSGDISP(QMGR)，或者定义了 QSGDISP(SHARED) 和 NOSHARE 选项，IBM MQ-IMS 网桥需要对网桥队列具有独占输入访问权 (MQOO_INPUT_EXCLUSIVE)。

用户在向 IMS 应用程序发送消息之前不需要登录 IMS。 MQMD 结构的 *UserIdentifier* 字段中的用户标识用于安全性检查。 检查级别在 IBM MQ 连接到 IMS 时确定，并在 IMS 网桥的应用程序访问控制中进行了描述。 这使得伪登录可以实施。

IBM MQ - IMS 网桥接受以下消息类型：

- 包含 IMS 事务数据和 MQIIH 结构（MQIIH 中进行了描述）的消息：

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

  注：

  1. 方括号 [ ] 表示可选的多区段。
  2. 将 MQMD 结构的 *Format* 字段设置为 MQFMT_IMS 以使用 MQIIH 结构。

- 包含 IMS 事务数据，但不包含 MQIIH 结构的消息：

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

IBM MQ 将对消息数据进行验证以确保 LL 字节数加上 MQIIH（如果存在）的长度总和等于消息长度。

IBM MQ - IMS 网桥收到来自网桥队列的消息时，会按照以下过程处理这些消息：

- 如果消息中包含 MQIIH 结构，网桥将验证 MQIIH（请参阅 MQIIH），构建 OTMA 头并将此消息发送到 IMS。 事务代码在输入消息中指定。 如果事务代码为 LTERM，IMS 将应答 DFS1288E 消息。 如果事务代码表示命令，IMS 将执行此命令；否则，消息将在事务的 IMS 中排队。
- 如果消息包含 IMS 事务数据，但不包含 MQIIH 结构，IMS 网桥将进行以下假设：
  - 事务代码是 5 - 12 个字节的用户数据
  - 事务处于非会话模式
  - 会话使用提交模式 0（提交然后发送）
  - MQMD 中的 *Format* 用作 *MFSMapName*（输入时）
  - 安全模式为 MQISS_CHECK

  还会在不使用 MQIIH 结构的情况下构建应答消息，从 IMS 输出的 *MFSMapName* 中获取 MQMD 的 *Format* 。

IBM MQ - IMS 网桥为每个 IBM MQ 队列使用一个或两个 Tpipe：

- 同步的 Tpipe 用于使用落实方式 0 (COMMIT_THEN_SEND) 的所有消息 (这些显示在 IMS /DIS TMEMBER 客户机 TPIPE xxxx 命令的状态字段中带有 SYN)
- 非同步 Tpipe 用于使用提交模式 1 (SEND_THEN_COMMIT) 的所有消息

Tpipe 由 IBM MQ 在第一次使用时创建。 非同步 Tpipe 在 IMS 重新启动前一直存在。 同步 Tpipe 在 IMS 冷启动之前一直存在。 您自己无法删除这些 Tpipe。

请参阅以下主题以了解有关 IBM MQ - IMS 网桥如何处理消息的更多信息：

- 第 62 页的『将 IBM MQ 消息映射到 IMS 事务类型』
- 第 62 页的『如果消息无法放到 IMS 队列』
- 第 63 页的『IMS 网桥反馈代码』
- 第 63 页的『IMS 网桥消息中的 MQMD 字段』
- 第 64 页的『IMS 网桥消息中的 MQIIH 字段』
- 第 65 页的『来自 IMS 的回复消息』
- 第 65 页的『在 IMS 事务中使用备用响应 PCB』
- 第 65 页的『发送来自 IMS 的自发消息』
- 第 65 页的『消息分段』
- 第 66 页的『消息与 IMS 网桥之间的数据转换』

相关概念

第 833 页的『通过 IBM MQ 编写 IMS 事务程序』

通过 IBM MQ 处理 IMS 事务所需的编码取决于 IMS 事务所需的消息格式及其可返回的响应范围。 但是，在应用程序处理 IMS 屏幕格式化信息时，应考虑以下几点。

将 *IBM MQ* 消息映射到 *IMS* 事务类型

下表描述 IBM MQ 消息到 IMS 事务类型的映射。

表 *4:* 如何将 *IBM MQ* 消息映射到 *IMS* 事务类型

IBM MQ 消息类型	**落实后发送（方式 0）- 使用同步 IMS Tpipe**	**发送后落实（方式 1）- 使用非同步 IMS Tpipe**
持久 IBM MQ 消息	• 可恢复完全功能事务 • IMS 拒绝不可恢复事务	• 快速路径事务 • 对话式事务 • 完全功能事务
非持久 IBM MQ 消息	• 不可恢复完全功能事务 • IMS V8 和 APAR PQ61404 以及 IMS 的所有更高版本允许可恢复事务	• 快速路径事务 • 对话式事务 • 完全功能事务

注: IMS 命令不能将持久 IBM MQ 消息用于落实方式 0。 请参阅 落实方式 (commitMode) 以获取更多信息。

如果消息无法放到 *IMS* 队列

了解在消息无法放到 IMS 队列时要采取的操作。

如果消息无法放到 IMS 队列，那么 IBM MQ 将采取以下操作：

- 如果消息由于无效而无法放到 IMS，那么会将该消息放到死信队列，并向系统控制台发送消息。
- 如果消息有效但遭到 IMS 拒绝，那么 IBM MQ 会向系统控制台发送错误消息，该消息包含 IMS 检测代码，并且 IBM MQ 消息放到死信队列。 如果 IMS 检测代码为 001A，那么 IMS 会发送包含应答队列故障原因的 IBM MQ 消息。

  注: 在先前所列的情况下，如果 IBM MQ 由于任何原因无法将消息放到死信队列，那么该消息会返回到始发 IBM MQ 队列。 将向系统控制台发送错误消息，并且不会从该队列发送任何其他消息。

  要重新发送消息，请执行以下其中一**项**操作：

  – 在与队列对应的 IMS 中停止并重新启动 Tpipe
  – 将队列变更为 GET(DISABLED)，然后再次变更为 GET(ENABLED)
  – 停止并重新启动 IMS 或 OTMA
  – 停止并重新启动 IBM MQ 子系统

- 如果消息由于除消息错误以外的任何其他原因遭到 IMS 拒绝，那么 IBM MQ 消息会返回到始发队列，IBM MQ 停止处理队列，并向系统控制台发送错误消息。

  如果需要异常报告消息，那么网桥会使用发起方的权限将其放到应答队列。 如果消息无法放到该队列，那么会使用网桥的权限将报告消息放到死信队列。 如果它无法放到 DLQ，那么会将其丢弃。

IMS 网桥反馈代码
IMS 检测代码通常是诸如 CSQ2001I 之类的 IBM MQ 控制台消息中十六进制格式的输出（例如，检测代码 0x001F）。 放到死信队列的消息的死信消息头中所显示的 IBM MQ 反馈代码是十进制数字。

IMS 网桥反馈代码在范围 301 到 399 内，对于 NACK 检测代码 0x001A 则在范围 600 到 855 内。 它们按如下所述从 IMS-OTMA 检测代码进行映射：

1. IMS-OTMA 检测代码从十六进制数字转换为十进制数字。
2. 将 300 添加到 1 中的计算所产生的数字中，并给出 IBM MQ *Feedback* 代码。
3. IMS-OTMA 检测代码 0x001A，十进制 26 是特殊情况。 将生成范围 600-855 内的反馈代码。

   a. IMS-OTMA 原因码从十六进制数字转换为十进制数字。

   b. 将 a 中的计算产生的数字加上 600，得出 IBM MQ 反馈代码。

有关 IMS-OTMA 检测代码的信息，请参阅用于 NAK 消息的 OTMA 检测代码。

IMS 网桥消息中的 *MQMD* 字段
了解有关来自 IMS 网桥的消息中 MQMD 字段的信息。

始发消息的 MQMD 由 IMS 在 OTMA 头的“用户数据”部分中携带。 如果消息源于 IMS，那么该消息由 IMS 目标解析出口构建。 从 IMS 接收的消息的 MQMD 构建如下：

StrucID
“MD”

版本
MQMD_VERSION_1

报告
MQRO_NONE

MsgType
MQMT_REPLY

到期
如果在 MQIIH 的“标志”字段中设置了 MQIIH_PASS_EXPIRATION，那么此字段包含剩余到期时间，否则它设置为 MQEI_UNLIMITED

Feedback
MQFB_NONE

编码
MQENC.Native（z/OS 系统的编码）

CodedCharSetId
MQCCSI_Q_MGR（z/OS 系统的 CodedCharSetID）

格式
如果输入消息的格式为 MQFMT_IMS，那么值为 MQFMT_IMS，否则值为 IOPCB.MODNAME

优先级
输入消息的 MQMD.Priority

持久
取决于落实方式：如果方式为 CM-1，那么值为输入消息的 MQMD.Persistence；如果方式为 CM-0，那么持久性与 IMS 消息的可恢复性相匹配

MsgId
如果是 MQRO_PASS_MSG_ID，那么值为 MQMD.MsgId，否则值为新 MsgId（缺省值）

CorrelId
如果是 MQRO_PASS_CORREL_ID，那么值为来自输入消息的 MQMD.CorrelId，否则值为来自输入消息的 MQMD.MsgId（缺省值）

BackoutCount
0

ReplyToQ
空白

ReplyToQMgr
空白（在 MQPUT 期间由队列管理器设置为本地 qmgr 名称）

UserIdentifier
输入消息的 MQMD.UserIdentifier

AccountingToken
输入消息的 MQMD.AccountingToken

ApplIdentityData
输入消息的 MQMD.ApplIdentityData

PutApplType
如果没有错误，那么值为 MQAT_XCF，否则值为 MQAT_BRIDGE

PutApplName
如果没有错误，那么值为 <XCFgroupName><XCFmemberName>，否则值为 QMGR 名称

PutDate
消息的放置日期

PutTime
消息放置的时间

ApplOriginData
空白

IMS 网桥消息中的 *MQIIH* 字段
了解有关来自 IMS 网桥的消息中 MQIIH 字段的信息。

从 IMS 接收的消息的 MQIIH 构建如下：

StrucId
“IIH”

版本
1

StrucLength
84

编码
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

格式
如果 MQIIH.ReplyToFormat 不是空白，那么值为输入消息的 MQIIH.ReplyToFormat，否则值为 IOPCB.MODNAME

标志
0

LTermOverride
来自 OMTA 头的 LTERM 名称 (Tpipe)

MFSMapName
来自 OMTA 头的映射名称

ReplyToFormat
空白

Authenticator
如果是将回复消息放到 MQ-IMS 网桥队列，那么值为输入消息的 MQIIH.Authenticator；否则值为空白。

TranInstanceId
如果在对话中，那么值为来自 OTMA 头的对话标识/服务器令牌。 在 V14 之前的 IMS 版本中，如果不在对话中，那么此字段始终为空。 从 IMS V14 开始，可以通过 IMS 设置该字段，即使不在对话中。

TranState
如果在对话中，那么值为“C”，否则值为空白

CommitMode
来自 OTMA 头的落实方式（“0”或“1”）

SecurityScope
Blank

已保留
Blank

来自 *IMS* 的回复消息
当 IMS 事务 ISRT 到其 IOPCB 时，会将消息路由回原始 LTERM 或 TPIPE。

这些消息在 IBM MQ 中被视为回复消息。 来自 IMS 的回复消息放到原始消息中指定的应答队列上。 如果消息无法放到应答队列上，那么会使用网桥的权限将其放到死信队列上。 如果消息无法放到死信队列，那么会向 IMS 发送否定确认以表明无法接收消息。 于是，消息是否重发的责任又回到 IMS 手里。 如果使用的是落实方式 0，那么来自该 Tpipe 的消息不会发送到网桥，并且保留在 IMS 队列上；即，在重新启动之前不会发送任何其他消息。 如果使用的是落实方式 1，那么其他工作可以继续。

如果回复具有 MQIIH 结构，那么其格式类型为 MQFMT_IMS；如果没有此结构，那么其格式类型由插入消息时所使用的 IMS MOD 名称指定。

在 *IMS* 事务中使用备用响应 *PCB*
当 IMS 事务使用备用响应 PCB（ISRT 到 ALTPCB，或对可修改 PCB 发出 CHNG 调用）时，会调用预路由出口 (DFSYPRX0) 以确定是否应该重新路由该消息。

如果要重新路由消息，那么会调用目标解析出口 (DFSYDRU0) 来确认目标并准备头信息。请参阅在 IMS 中使用 OMTA 出口和预路由出口 DFSYPRX0，以获取有关这些出口程序的信息。

除非在出口中执行操作，否则从 IBM MQ 队列管理器 (无论是 IOPCB 还是 ALTPCB) 启动的 IMS 事务的所有输出都将返回到同一队列管理器。

发送来自 *IMS* 的自发消息
要将消息从 IMS 发送到 IBM MQ 队列，您需要调用要 ISRT 到 ALTPCB 的 IMS 事务。

您需要编写预路由出口和目标解析出口以路由来自 IMS 的自发消息并构建 OTMA 用户数据，以便可以正确构建消息的 MQMD。 请参阅预路由出口 DFSYPRX0 和目标解析用户出口以获取有关这些出口程序的信息。

注: IBM MQ - IMS 网桥不知道它接收的消息是回复消息还是自发消息。 它在各种情况下以相同方式处理消息，从而根据随消息到达的 OTMA UserData 构建回复的 MQMD 和 MQIIH

自发消息可以创建新 Tpipe。 例如，如果现有 IMS 事务切换到新 LTERM（例如 PRINT01），但是实施要求通过 OTMA 传递输出，那么将会创建新 Tpipe（在本例中名为 PRINT01）。 缺省情况下，这是非同步 Tpipe。 如果实施要求消息可恢复，请设置目标解析出口输出标志。 请参阅《*IMS* 定制指南》以获取更多信息。

消息分段
您可以将 IMS 事务定义为期望单段或多段输入。

始发 IBM MQ 应用程序必须将遵循 MQIIH 结构的用户输入构造为一个或多个 LLZZ 数据段。 IMS 消息的所有段都必须包含在通过单个 MQPUT 发送的单条 IBM MQ 消息中。

LLZZ 数据段的最大长度由 IMS/OTMA 定义（32767 字节）。 IBM MQ 消息总长度是 LL 字节的总和，加上 MQIIH 结构的长度。

回复的所有段都包含在单条 IBM MQ 消息中。

对于格式为 MQFMT_IMS_VAR_STRING 的消息的 32 KB 限制存在进一步制约。 当 ASCII 混合 CCSID 消息中的数据转换为 EBCDIC 混合 CCSID 消息时，每次 SBCS 和 DBCS 字符之间存在转换时，将会添加移入或移出字节。 32 KB 限制适用于消息的最大大小。 即，由于消息中的 LL 字段不能超过 32 KB，因此消息不得超过 32 KB（包括所有移入和移出字符）。 构建消息的应用程序必须将此考虑在内。

消息与 *IMS* 网桥之间的数据转换
数据转换由分布式排队工具（可以调用任何必要的出口）或由组内排队代理程序（不支持使用出口）在将消息放到具有为其存储类定义的 XCF 信息的目标队列时执行。 通过发布/预订将消息传递到队列时，不会发生数据转换。

所需的任何出口都必须在 CSQXLIB DD 语句所引用的数据集中可供分布式排队工具使用。 这意味着您可以使用 IBM MQ - IMS 网桥从任何 IBM MQ 平台将消息发送到 IMS 应用程序。

如果有转换错误，那么消息会未经转换放到队列；这最终导致它被 IBM MQ - IMS 网桥视为错误，因为网桥无法识别头格式。 如果发生转换错误，那么将会向 z/OS 控制台发送错误消息。

请参阅第 893 页的『编写数据转换出口』以获取有关一般数据转换的详细信息。

将消息发送到 IBM MQ - IMS 网桥

要确保正确执行转换，必须告知队列管理器消息的格式。

如果消息具有 MQIIH 结构，那么 MQMD 中的 *Format* 必须设置为内置格式 MQFMT_IMS，并且 MQIIH 中的 *Format* 必须设置为用于描述消息数据的格式的名称。 如果没有 MQIIH，请将 MQMD 中的 *Format* 设置为您的格式名称。

如果数据（除 LLZZ 以外）全部都是字符数据 (MQCHAR)，请将内置格式 MQFMT_IMS_VAR_STRING 用作格式名称（根据情况，在 MQIIH 或 MQMD 中）。 否则，使用自己的格式名称，在此情况下还必须为格式提供数据转换出口。 除数据本身以外，出口还必须处理消息中 LLZZ 的转换（但是不必处理消息开头的任何 MQIIH）。

如果应用程序使用 *MFSMapName*，那么可以改用具有 MQFMT_IMS 的消息，并在 MQIIH 的 MFSMapName 字段中定义传递到 IMS 事务的映射名称。

从 IBM MQ - IMS 网桥接收消息

如果在发送到 IMS 的原始消息上存在 MQIIH 结构，那么在回复消息上也存在该结构。

要确保正确转换回复，请执行以下操作：

- 如果在原始消息上具有 MQIIH 结构，请在原始消息的 MQIIH *ReplytoFormat* 字段中指定希望回复消息具有的格式。 该值放入回复消息的 MQIIH *Format* 字段中。 这在所有输出数据都为 LLZZ<character data> 形式时特别有用。
- 如果在原始消息上没有 MQIIH 结构，请在 IMS 应用程序的 ISRT 到 IOPCB 中指定希望回复消息具有的格式作为 MFS MOD 名称。

开发 JMS 和 Java 应用程序

IBM MQ 提供两个 Java 语言接口：IBM MQ classes for Java Message Service 和 IBM MQ classes for Java。

关于此任务

在 IBM MQ 中，Java 应用程序中可以使用以下两个备选 API。 Java 应用程序可以使用 IBM MQ classes for JMS 或 IBM MQ classes for Java 来访问 IBM MQ 资源。

IBM MQ classes for JMS
IBM MQ classes for Java Message Service (JMS) 是 IBM MQ 随附的 JMS 提供程序。 Java Platform, Enterprise Edition Connector Architecture (JCA) 提供一种标准方式将运行在 Java EE 环境中的应用程序连接到企业信息系统 (EIS)，如 IBM MQ 或 Db2。

如果您不熟悉 IBM MQ 或者已经具有 JMS 经验，那么可能会发现使用熟悉的 JMS API 来访问 IBM MQ 资源（通过使用 IBM MQ classes for JMS）更容易。 JMS 也是 Java Platform, Enterprise Edition (Java

EE) 平台的不可或缺的部分。 Java EE 应用程序可以使用消息驱动的 bean (MDB) 来异步处理消息。 JMS 也是 Java EE 与异步消息传递系统（例如 IBM MQ）交互的标准机制。 兼容 Java EE 的每个应用程序服务器都必须包含 JMS 提供程序，因此可以使用 JMS 在不同应用程序服务器之间进行通信，或者可以将应用程序从一个 JMS 提供程序移植到另一个提供程序而不对应用程序进行任何更改。

IBM MQ classes for Java

IBM MQ classes for Java 使您能够在 Java 环境中使用 IBM MQ 。 IBM MQ classes for Java 允许 Java 应用程序作为 IBM MQ 客户机连接到 IBM MQ，或直接连接到 IBM MQ 队列管理器。

IBM MQ classes for Java 封装消息队列接口 (MQI)（本机 IBM MQ API），并且与其他面向对象的接口使用同一对象模型，而 IBM MQ classes for Java Message Service 实施 Oracle 的 Java Message Service (JMS) 接口。

如果您熟悉 Java 以外的环境中的 IBM MQ ，那么可以使用过程语言或面向对象的语言，通过使用 IBM MQ classes for Java 将现有知识传输到 Java 环境。 您还可以利用 IBM MQ 的完整范围的功能，并非所有功能在 IBM MQ classes for JMS 中都可用。

注:

IBM 将不对 IBM MQ classes for Java 进行进一步的增强，功能稳定在 IBM MQ 8.0 随附的级别上。 使用 IBM MQ classes for Java 的现有应用程序继续完全受支持，但是将不会添加新功能部件，并且会拒绝针对增强功能的请求。 完全受支持意味着，在完成 IBM MQ 系统需求变化所需的任何更改的同时修复缺陷。

IBM MQ classes for Java 在 IMS 中不受支持。

IBM MQ classes for Java 在 WebSphere Liberty 中不受支持。 不得与 IBM MQ Liberty 消息传递功能或通用的 JCA 支持一起使用。 有关更多信息，请参阅在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口。

使用 IBM MQ classes for JMS

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

JMS 规范定义了一组可供应用程序用来执行消息传递操作的接口。 规范的最新版本为 JMS 2.0。 javax.jms 包定义 JMS 接口，并且 JMS 提供程序为特定消息传递产品实现这些接口。 IBM MQ classes for JMS 是用于实现 IBM MQ 的 JMS 接口的 JMS 提供程序。

JMS 规范认为 ConnectionFactory 和 Destination 对象都是受管对象。 管理员可在中央存储库中创建和维护受管对象，并且 JMS 应用程序可使用 Java Naming Directory Interface (JNDI) 来检索这些对象。IBM MQ classes for JMS 支持使用受管对象，并且管理员可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 来创建和维护受管对象。

IBM MQ classes for JMS 还提供两个 JMS API 扩展集。 这些扩展的主要目标是在运行时动态创建和配置连接工厂和目标，但是这些扩展还提供了与消息传递不直接相关的功能，例如问题确定功能。

IBM MQ JMS 扩展

IBM MQ classes for JMS 的前发行版包含诸如 MQConnectionFactory、MQQueue 和 MQTopic 之类的对象中实现的扩展。 这些对象包含特定于 IBM MQ 的属性和方法。 这些对象可以是受管对象，也可以由应用程序在运行时动态创建这些对象。 IBM MQ classes for JMS 的此发行版维护这些扩展，此类扩展现在称为 IBM MQ JMS 扩展。 您可以不经更改继续使用任何使用这些扩展的应用程序。

IBM JMS 扩展

IBM MQ classes for JMS 的此发行版提供更通用的 JMS API 扩展集，这些扩展并非像消息传递系统一样特定于 IBM MQ。 这些扩展称为 IBM JMS 扩展并具有以下广泛目标:

- 提高各个 IBM JMS 提供程序的一致程度
- 更容易编写两个 IBM 消息传递系统之间的桥接应用程序
- 更容易将应用程序从一个 IBM JMS 提供程序移植到另一个提供程序

扩展提供类似于 IBM Message Service Client for C/C++ 和 IBM Message Service Client for .NET 中所提供的功能。

从 IBM MQ 8.0 开始，IBM MQ classes for JMS 使用 Java 7 进行构建。

Java 7 运行时环境支持运行较早的类文件版本。

相关概念

IBM MQ Java 语言接口

第 109 页的『JMS 模型』
JMS 模型定义了一组接口， Java 应用程序可以使用这些接口来执行消息传递操作。 IBM MQ classes for JMS 作为 JMS 提供程序，定义 JMS 对象如何与 IBM MQ 概念相关。 JMS 规范认为某些 JMS 对象是受管对象。 JMS 2.0 引入了简化的 API，同时保留了 JMS 1.1 中的标准 API。

第 262 页的『使用 JMS 2.0 功能』
JMS 2.0 将多个新功能领域引入了 IBM MQ classes for JMS。

为什么应该使用 IBM MQ classes for JMS?

使用 IBM MQ classes for JMS 具有很多优点，包括：能够复用组织中的任何现有 JMS 技能、应用程序更独立于 JMS 提供程序和底层的 IBM MQ 配置。

IBM MQ classes for JMS 是两个备用 API 中的一个，Java 应用程序可以使用它来访问 IBM MQ 资源。 另一个 API 是 IBM MQ classes for Java。 虽然继续完全支持使用 IBM MQ classes for Java 的现有应用程序，新应用程序应使用 IBM MQ classes for JMS（请参阅 第 69 页的『如何选择 API』）。

使用 IBM MQ classes for JMS 的优点汇总

使用 IBM MQ classes for JMS，您可以复用现有 JMS 技能并提供应用程序独立性。

- 您可以复用 JMS 技能。

  IBM MQ classes for JMS 是 JMS 提供程序，用于将 IBM MQ 的 JMS 接口实现为消息传递系统。 如果您的组织不熟悉 IBM MQ，但是已经拥有 JMS 应用程序开发技能，您可能发现使用熟悉的 JMS API 访问 IBM MQ 资源比使用 IBM MQ 提供的其他某个 API 更加容易。

- JMS 是 Java Platform, Enterprise Edition (Java EE) 的不可缺少的部分。

  JMS 是用于 Java EE 平台消息传递的自然 API。 与 Java EE 相容的每一个应用程序服务器必须包括 JMS 提供者。 可以在应用程序客户机， Servlet ， Java Server Pages (JSP) ，企业 Java Bean (EJB) 和消息驱动的 Bean (MDB) 中使用 JMS 。 特别注意，Java EE 应用程序使用 MDB 异步处理消息，所有消息作为 JMS 消息传递到 MDB。

- 连接工厂和目标可以作为 JMS 受管对象存储在中央存储库，而不是硬编码到应用程序。

  管理员可以在中央存储库中创建和维护 JMS 受管对象，而 IBM MQ classes for JMS 应用程序可以使用 Java Naming Directory Interface (JNDI) 来检索这些对象。JMS 连接工厂和目标将封装特定于 IBM MQ 的信息，如队列管理器名称、通道名称、连接选项、队列名称和主题名称。 如果连接工厂和目标作为受管对象存储，此信息不会硬编码到应用程序。 因此，这种安排使应用程序在一定程度上独立于底层 IBM MQ 配置。

- JMS 是可以提供应用程序可移植性的行业标准 API。

  JMS 应用程序可以使用 JNDI 检索作为受管对象存储的连接工厂和目标，并且仅使用 javax.jms 包中定义的接口来执行消息传递操作。 然后应用程序完全独立于任何 JMS 提供程序，例如 IBM MQ classes for JMS，并且不需要更改应用程序就可以从一个 JMS 提供程序移植到另一个应用程序。 如果 JNDI 在特定应用程序环境中是不可用的，那么 IBM MQ classes for JMS 应用程序可以使用 JMS API 的扩展在运行时动态创建和配置连接工厂和目标。 然后，应用程序将完全自包含，但作为 JMS 提供程序与 IBM MQ classes for JMS 绑定。

- 使用 JMS 可更容易地编写桥接应用程序。

  桥接应用程序是从一个消息传递系统接收消息并将消息发送至其他消息传递系统的应用程序。 使用产品特定的 API 和消息格式可使桥接应用程序的编写更加复杂。 您可以改为使用两个 JMS 提供程序编写桥接应用程序，针对每个消息传递系统使用一个提供程序。 然后，应用程序仅使用一个 API，即 JMS API，并仅处理 JMS 消息。

可部署环境

为提供与 Java EE 应用程序服务器的集成，Java EE 标准需要消息传递提供程序来提供资源适配器。 根据 Java EE Connector Architecture (JCA) 规范，IBM MQ 提供的资源适配器将使用 JMS 在任何经过认证的 Java EE 环境中提供消息传递功能。

虽然可以使用 IBM MQ classes for Java 内部 Java EE，但此 API 未为此目的进行改造或优化。 请参阅 IBM 技术说明 在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口 ，以获取 Java EE 中 IBM MQ classes for Java 注意事项的详细信息。

在 Java EE 环境之外，提供 OSGi 和 JAR 文件可便于您仅获取 IBM MQ classes for JMS。 这些 JAR 文件现在可以更加轻松地独立部署或在软件管理框架（例如 Maven）中部署。有关更多信息，请参阅 IBM 技术说明 Obtaining the WebSphere MQ classes for JMS。

如何选择 API

新的应用程序应使用 IBM MQ classes for JMS，而非 IBM MQ classes for Java。

IBM MQ classes for JMS 使您能够访问 IBM MQ 的点到点消息传递功能和发布/预订消息传递功能。 除了发送可提供 JMS 标准消息传递模型支持的 JMS 消息之外，应用程序还可以发送和接收不带任何附加头的消息，因此可以与其他 IBM MQ 应用程序（例如 C MQI 应用程序）进行互操作。 提供了对 MQMD 和 MQ 消息有效内容的完全控制权。 还提供了其他 IBM MQ 功能，例如消息分流、异步放置和报告消息。 通过使用提供的 PCF 助手类，IBM MQ PCF 管理消息可以通过 JMS API 进行发送和接收，并可用于管理队列管理器。

最近添加到 IBM MQ 的功能（例如异步使用和自动重新连接）在 IBM MQ classes for Java 中不可用，但在 IBM MQ classes for JMS 中可用。 继续完全支持那些使用 IBM MQ classes for Java 的现有应用程序。

如果需要访问 IBM MQ 功能（无法通过 IBM MQ classes for JMS 来访问），那么可以提交改进请求 (RFE)。 然后 IBM 会建议是否可以在 IBM MQ classes for JMS 实现中进行实现，以及是否有可遵循的最佳实践。 由于 IBM 是开放式标准的贡献者之一，因此要获取其他消息传递功能，可以在 JCP 流程中提出这些功能。

相关任务

跟踪 IBM MQ classes for JMS 应用程序

Java 和 JMS 故障诊断

相关信息

IBM RFE 提交过程

JMS Java 规范审核过程

在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口

获取用于 JMS 的 WebSphere MQ 类

使用 JMS 发送 PCF 消息

IBM MQ classes for JMS 的先决条件

本主题说明您在使用 IBM MQ classes for JMS 之前需要了解的信息。 要开发和运行 IBM MQ classes for JMS 应用程序，需要将某些软件组件作为必备软件。

有关 IBM MQ classes for JMS 的先决条件的信息，请参阅 IBM MQ 的系统需求。

要开发 IBM MQ classes for JMS 应用程序，您需要 Java SE 软件开发包 (SDK)。 有关操作系统支持的 JDK 的详细信息，请参阅 IBM MQ 的系统需求。

要运行 IBM MQ classes for JMS 应用程序，您需要以下软件组件：

- IBM MQ 队列管理器。
- Java runtime environment (JRE)，针对运行应用程序的各系统。
- IBM i 对于 IBM i，Qshell（操作系统的选项 30）。
- z/OS 对于 z/OS，UNIX and Linux 系统服务 (USS)。

IBM JSSE 提供程序包含经过 FIPS 认证的加密提供程序，因此可以编程方式进行配置，从而符合 FIPS 140-2 以供立即使用。 因此，可以直接从 IBM MQ classes for Java 和 IBM MQ classes for JMS 支持 FIPS 140-2 合规性。

Oracle 的 JSSE 提供程序可具有经过 FIPS 认证的加密提供程序，该加密提供程序已配置到其中，但是还未能立即使用，并且不适用于程序配置。 因此，在此情况下，IBM MQ classes for Java 和 IBM MQ classes for JMS 无法直接启用 FIPS 140-2 合规性。 您可能可以手动启用此类合规性，但是 IBM 当前无法提供与此有关的指南。

如果 Java 虚拟机 (JVM) 和操作系统上的 TCP/IP 实现支持 IPv6 地址，那么可以在 IBM MQ classes for JMS 应用程序中使用 Internet Protocol V 6 (IPv6) 地址。 IBM MQ JMS 管理工具（请参阅使用管理工具配置 JMS 对象）也接受 IPv6 地址。

IBM MQ JMS 管理工具和 IBM MQ Explorer 使用 Java Naming Directory Interface (JNDI) 来访问用于存储受管对象的目录服务。 IBM MQ classes for JMS 应用程序还可以使用 JNDI 从目录服务检索受管对象。 服务提供者是通过将 JNDI 调用映射到目录服务来提供对该目录服务的访问的代码。 文件 `fscontext.jar` 和 `providerutil.jar` 中的文件系统服务提供程序随 IBM MQ classes for JMS 一起提供。 文件系统服务提供程序提供了对基于本地文件系统的目录服务的访问。

如果意图使用基于 LDAP 服务器的目录服务，那么必须安装并配置 LDAP 服务器，或者必须对现有 LDAP 服务器具有访问权。 特别是，必须将 LDAP 服务器配置为存储 Java 对象。 有关如何安装和配置 LDAP 服务器的信息，请参阅服务器随附的文档。

安装和配置 IBM MQ classes for JMS

本节描述在安装 IBM MQ classes for JMS 时创建的目录和文件，并且指示安装后如何配置 IBM MQ classes for JMS。

相关概念

第 366 页的『使用 IBM MQ 资源适配器』

通过资源适配器，在应用程序服务器中运行的应用程序可以访问 IBM MQ 资源。 它支持入站和出站通信。

为 *IBM MQ classes for JMS* 安装了什么

安装 IBM MQ classes for JMS 时，将会创建大量文件和目录。 在 Windows 上，通过自动设置环境变量在安装期间执行了某些配置。 在其他平台上的特定 Windows 环境中，必须先设置环境变量，然后才能运行 IBM MQ classes for JMS 应用程序。

对于大多数操作系统，IBM MQ classes for JMS 在您安装 IBM MQ 时安装为可选组件。

有关安装 IBM MQ 的更多信息，请参阅：

Multi 安装 IBM MQ

z/OS 安装 IBM MQ for z/OS

要点: 除第 72 页的『IBM MQ classes for JMS 可再定位的 JAR 文件』中描述的可再定位的 JAR 文件以外，不支持将 IBM MQ classes for JMS JAR 文件或本机库复制到其他机器，或者复制到已安装 IBM MQ classes for JMS 的机器上的其他位置。

安装目录

第 70 页的表 5 显示 IBM MQ classes for JMS 文件在各平台上的安装位置。

表 *5: IBM MQ classes for JMS* 安装目录

平台	目录
Linux UNIX UNIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java

表 5: *IBM MQ classes for JMS* 安装目录 (继续)	
平台	**目录**
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V9R1M0/java
	MQ_INSTALLATION_PATH/opt/mqm/java

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

安装目录包含以下内容:

- IBM MQ classes for JMS JAR 文件，包括位于 *MQ_INSTALLATION_PATH*\java\lib 目录中的可再定位的 JAR 文件。
- IBM MQ 本机库，供使用 Java 本机接口的应用程序使用。

  32 位本机库安装在 *MQ_INSTALLATION_PATH*\java\lib 目录中，而 64 位本机库安装在 *MQ_INSTALLATION_PATH*\java\lib64 目录中。

  有关 IBM MQ 本机库的更多信息，请参阅 第 75 页的『配置 Java 本机接口 (JNI) 库』。
- 第 97 页的『与 IBM MQ classes for JMS 一起提供的脚本』中描述的其他脚本。 这些脚本位于 *MQ_INSTALLATION_PATH*\java\bin 目录中。
- IBM MQ classes for JMS API 的规范。 Javadoc 工具已用于生成包含 API 规范的 HTML 页面。

  这些 HTML 页面位于 *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses 目录中:

  – ULW 在 UNIX, Linux, and Windows 上，该子目录包含个别 HTML 页面。

  – IBM i 在 IBM i 上，这些 HTML 页面位于名为 wmqjms_javadoc.jar 的文件中。

  – z/OS 在 z/OS 上，这些 HTML 页面位于名为 wmqjms_javadoc.jar 的文件中。
- 对 OGSi 的支持。 OSGi 捆绑软件安装在 java\lib\OSGi 目录中，并且在 第 98 页的『OSGi 支持』 中描述了这些捆绑软件。
- IBM MQ 资源适配器，可以部署到任何符合 Java Platform, Enterprise Edition 7 (Java EE 7) 的应用程序服务器中。

  IBM MQ 资源适配器位于 *MQ_INSTALLATION_PATH*\java\lib\jca 目录中；有关更多信息，请参阅 第 366 页的『使用 IBM MQ 资源适配器』
- Windows 在 Windows 上，可用于调试的符号安装在 *MQ_INSTALLATION_PATH*\java\lib\symbols 目录中。

安装目录还包含属于其他 IBM MQ 组件的一些文件。

样本应用程序

某些样本应用程序随附于 IBM MQ classes for JMS。 第 71 页的表 6 显示样本应用程序在各平台上的安装位置。

表 6: 样本目录	
平台	**目录**
AIX UNIX and Linux	*MQ_INSTALLATION_PATH*/samp/jms
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms
IBM i IBM i	/QIBM/ProdData/mqm/java/samples/jms
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/samples/jms

表 6: 样本目录 (继续)	
平台	**目录**
	MQ_INSTALLATION_PATH/opt/mqm/samp/jms

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

在安装后，可能需要执行一些配置任务以编译和运行应用程序。

第 73 页的『为 IBM MQ classes for JMS 设置环境变量』描述运行简单 IBM MQ classes for JMS 应用程序所需的类路径。 本主题还描述在特殊情况下需要参考的其他 JAR 文件，以及为运行 IBM MQ classes for JMS 随附的脚本而必须设置的环境变量。

要控制属性（例如应用程序的跟踪和日志记录），需要提供配置属性文件。 IBM MQ classes for JMS 配置属性文件在第 77 页的『IBM MQ classes for JMS 配置文件』中进行了描述。

相关概念

部署资源适配器时发生的问题

相关任务

第 94 页的『使用 IBM MQ classes for JMS 样本应用程序』
IBM MQ classes for JMS 样本应用程序概述了 JMS API 的常用功能。 使用这些样本应用程序，可以验证安装和消息传递服务器设置，并帮助您构建自己的应用程序。

IBM MQ classes for JMS 可再定位的 *JAR* 文件

可以将可再定位的 JAR 文件移至需要运行 IBM MQ classes for JMS 的系统。

要点:

- 除可再定位的 JAR 文件中描述的可再定位的 JAR 文件以外，不支持将 IBM MQ classes for JMS JAR 文件或本机库复制到其他机器，或者复制到已安装 IBM MQ classes for JMS 的机器上的其他位置。
- 请勿在部署到 Java EE 应用程序服务器（如 WebSphere Application Server 或 WebSphere Liberty）的应用程序中包含可再定位的 JAR 文件。 在这些环境中，应该改为部署和使用 IBM MQ 资源适配器。 请注意，WebSphere Application Server 将嵌入 IBM MQ 资源适配器，因此无需将其手动部署到此环境中。
- 为避免类装入器冲突，不建议将可再定位的 JAR 文件绑定在同一 Java 运行时内的多个应用程序中。 在此场景中，请让 IBM MQ 可再定位的 JAR 文件在 Java 运行时的类路径上可用。
- 如果要将可再定位的 JAR 文件绑定在应用程序中，请确保您包含所有必备的 JAR 文件，如可再定位的 JAR 文件中所述。 您还应确保具有相应的过程在应用程序维护期间更新绑定的 JAR 文件，以确保 IBM MQ classes for JMS 保持最新状态，并且重新解决已知问题。

可再定位的 JAR 文件

在企业内，以下文件可以移至需要运行 IBM MQ classes for JMS 的系统:

- `com.ibm.mq.allclient.jar`
- `com.ibm.mq.traceControl.jar`
- `jms.jar`
- `fscontext.jar`
- `providerutil.jar`
- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.1.0.9 `bcutil-jdk15on.jar`
- V 9.1.2 `org.json.jar`

`jms.jar` 文件是 `javax.jms.jar` 的接口文件，并且包含非 IBM JMS 类。

如果应用程序使用文件系统上下文执行 JNDI 查找，那么需要使用 `fscontext.jar` 和 `providerutil.jar` 文件。

需要 Bouncy Castle 安全提供程序和 CMS 支持 JAR 文件。 有关更多信息，请参阅支持使用 AMS 的非 IBM JRE。 需要以下 JAR 文件：

- bcpkix-jdk15on.jar
- bcprov-jdk15on.jar
- V 9.1.0.9 bcutil-jdk15on.jar

V 9.1.2 如果 IBM MQ classes for JMS 应用程序使用 JSON 格式的 CCDT，那么需要 org.json.jar 文件。

文件 com.ibm.mq.allclient.jar 包含 IBM MQ classes for JMS、IBM MQ classes for Java 以及 PCF 和 Headers 类。 如果您将此文件移动到新位置，请确保采取步骤使用新的 IBM MQ 修订包对此新位置进行维护。 此外，如果获取的是临时修订，请确保 IBM 支持人员知悉此文件的使用。

要确定 com.ibm.mq.allclient.jar 文件的版本，请使用以下命令：

```
java -jar com.ibm.mq.allclient.jar
```

以下示例显示此命令的某些样本输出：

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.1.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

com.ibm.mq.traceControl.jar 文件用于动态控制 IBM MQ classes for JMS 应用程序的跟踪。 有关更多信息，请参阅使用 IBM MQ classes for Java 和 IBM MQ classes for JMS 在运行中的流程中控制跟踪。

为 *IBM MQ classes for JMS* 设置环境变量

在编译和运行 IBM MQ classes for JMS 应用程序之前， CLASSPATH 环境变量的设置必须包含 IBM MQ classes for JMS Java 归档 (JAR) 文件。 根据需求，您可能需要向类路径中添加其他 JAR 文件。 要运行 IBM MQ classes for JMS 随附的脚本，必须设置其他环境变量。

关于此任务

要点: 不支持将 Java 选项 *-Xbootclasspath* 设置为包含 IBM MQ classes for JMS。

要编译和运行 IBM MQ classes for JMS 应用程序，请针对平台使用 CLASSPATH 设置，如第 74 页的表 7中所示。 设置包括样本目录，以便您可以编译和运行 IBM MQ classes for JMS 样本应用程序。 或者，可以在 **java** 命令中指定类路径，而不是使用环境变量。

表 7: 用于编译和运行 *IBM MQ classes for JMS* 应用程序（包括样本应用程序）的 *CLASSPATH* 设置	
平台	**CLASSPATH 设置**
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux Linux Solaris Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS z/OS	LTS CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/samples/jms/samples: CD CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R1M5/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V9R1M5/java/samples/jms/samples:

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

JAR 文件 `com.ibm.mqjms.jar` 的清单包含对 IBM MQ classes for JMS 应用程序所需的大部分其他 JAR 文件的引用，因此您不需要将这些 JAR 文件添加到类路径。 这些 JAR 文件包括使用 Java Naming Directory Interface (JNDI) 从目录服务检索受管对象的应用程序以及使用 Java 事务 API (JTA) 的应用程序所需的 JAR 文件。

但是，以下情况下必须在类路径中包含其他 JAR 文件：

- 如果要使用实现了 `com.ibm.mq` 包中定义的通道出口接口（而不是 `com.ibm.mq.exits` 包中定义的通道出口接口）的通道出口类，那么必须将 IBM MQ classes for Java JAR 文件 `com.ibm.mq.jar` 添加到类路径中。
- 如果应用程序使用 JNDI 从目录服务检索受管对象，那么还必须向类路径中添加以下 JAR 文件：
  - `fscontext.jar`
  - `providerutil.jar`
- 如果应用程序使用 JTA，那么还必须将 `jta.jar` 添加到类路径中。

注: 这些其他 JAR 文件仅对于编译应用程序是必需的，对于运行这些应用程序并非必需。

随 IBM MQ classes for JMS 提供的脚本使用以下环境变量：

MQ_JAVA_DATA_PATH
此环境变量指定日志和跟踪输出的目录。

MQ_JAVA_INSTALL_PATH
此环境变量指定 IBM MQ classes for JMS 安装所在的目录。

MQ_JAVA_LIB_PATH
此环境变量指定 IBM MQ classes for JMS 库存储所在的目录，如第 76 页的表 8 中所示。

过程

- Windows
  在 Windows 上，安装 IBM MQ 后，请运行命令 **setmqenv**。
  如果未首先运行此命令，那么在发出 **dspmqver** 命令时可能会出现以下错误消息：

  V 9.1.0
  ```
  AMQ8351: IBM MQ Java environment has not been configured
  correctly, or the IBM MQ JRE feature has not been installed.
  ```

  注: V 9.1.0 如果未安装 IBM MQ Java 运行时环境 (JRE)，那么预计将出现此消息（请参阅其他 Windows 功能先决条件检查）。
- 在任何其他平台上，请自行设置环境变量：
  - Linux UNIX 如果在 UNIX 或 Linux 系统上使用 32 位 JVM，要设置环境变量，可以使用脚本 setjmsenv。
  - Linux UNIX 如果在 UNIX 或 Linux 系统上运行的是 64 位 JVM，要设置环境变量，可以使用脚本 setjmsenv64。 这些脚本位于 *MQ_INSTALLATION_PATH*`/java/bin` 目录中，其中 *MQ_INSTALLATION_PATH* 表示用于安装 IBM MQ 的高级目录。

  您可以通过多种方式来使用 setjmsenv 或 setjmsenv64 脚本：您可以将其用作设置必需环境变量的基础（如表中所示），或者使用文本编辑器将其添加到 `.profile`。 如果您具有非典型安装，请在必要情况下编辑脚本内容。 或者，可以在要从中运行 JMS 启动脚本的每个会话中运行脚本。 如果选择此选项，那么需要在启动的每个 shell 窗口中运行脚本，在 JMS 验证过程中，通过输入 `. ./setjmsenv` 或 `. ./setjmsenv64`

  IBM i 在 IBM i 上，必须将环境变量 QIBM_MULTI_THREADED 设置为 Y。 随后可使用运行单线程应用程序的方法来运行多线程应用程序。 有关更多信息，请参阅使用 Java 和 JMS 设置 IBM MQ。

配置 *Java* 本机接口 *(JNI)* 库
IBM MQ classes for JMS 应用程序（使用绑定传输连接到队列管理器，或者使用客户机传输连接到队列管理器并使用以除 Java 以外的语言编写的通道出口程序）需要在允许访问 Java 本机接口 (JNI) 库的环境中运行。

开始之前

有关使用 WebSphere Application Server 环境的更多信息，请参阅使用本机库信息配置 IBM MQ 消息传递提供程序 。

关于此任务

要设置此环境，必须配置环境的库路径，以便 Java virtual machine (JVM) 能够在启动 IBM MQ classes for JMS 应用程序前装入 mqjbnd 库。

IBM MQ 提供两个 Java 本机接口 (JNI) 库：

mqjbnd
此库供使用绑定传输连接到队列管理器的应用程序使用。 它提供 IBM MQ classes for JMS 和队列管理器之间的接口。 随 IBM MQ 9.0 安装的 mqjbnd 库可用于连接到任何 IBM MQ 9.0（或更低版本）队列管理器。

mqjexitstub02
当应用程序使用客户机传输连接到队列管理器并使用以 Java 以外的语言编写的通道出口程序时， IBM MQ classes for JMS 将装入 mqjexitstub02 库。

在某些平台上，IBM MQ 安装这些 JNI 库的 32 位和 64 位版本。 表 1 中显示了每个平台的库的位置。

表 8: 各平台的 *IBM MQ classes for JMS* 库的位置	
平台	**包含 IBM MQ classes for JMS 库的目录**
AIX Linux （POWER、x86-64 和 zSeries s390x 平台） Solaris（x86-64 和 SPARC 平台）	*MQ_INSTALLATION_PATH*/java/lib（32 位库） *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
Windows	*MQ_INSTALLATION_PATH*\java\lib（32 位库） *MQ_INSTALLATION_PATH*\java\lib64（64 位库）
z/OS	*MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/lib （31 位和 64 位库）

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

注: z/OS 在 z/OS 上，您可以使用 31 位或 64 位 Java virtual machine (JVM)。 您不必指定要使用哪些 JNI 库；IBM MQ classes for JMS 可以自行确定要装入哪些 JNI 库。

过程

1. 配置 JVM 的 **java.library.path** 属性，可以按两种方式完成：
   - 通过指定以下示例中所示的 JVM 参数：

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux 例如，对于 Linux 上的 64 位 JVM，要进行缺省位置安装，请指定：

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - 通过配置 Shell 的环境，JVM 将设置自己的 `java.library.path`。 此路径因平台以及安装 IBM MQ 的位置而异。 例如，对于 64 位 JVM 及缺省 IBM MQ 安装位置，可使用以下设置：

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

     Solaris Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   未正确配置环境时您将看到的异常堆栈的示例如下：

```
Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Failed to load the WebSphere MQ native JNI library: 'mqjbnd'.
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    at java.security.AccessController.doPrivileged(AccessController.java:400)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
```

```
a:58)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    at
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 more
Caused by: java.lang.UnsatisfiedLinkError: mqjbnd (Not found in java.library.path)
    at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    at java.lang.System.loadLibrary(System.java:534)
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 more
```

2. 设置 32 位或 64 位环境后，使用以下命令启动 IBM MQ classes for JMS 应用程序：

   ```
   java application-name
   ```

   其中 *application-name* 是要运行的 IBM MQ classes for JMS 应用程序的名称。

   包含 IBM MQ 原因码 2495 (MQRC_MODULE_NOT_FOUND) 的异常由 IBM MQ classes for JMS 抛出，如果：

   - IBM MQ classes for JMS 应用程序在 32 位 Java runtime environment 中运行，并且已为 IBM MQ classes for JMS 设置 64 位环境，因为 32 位 Java runtime environment 无法装入 64 位 Java 本机库。
   - IBM MQ classes for JMS 应用程序在 64 位 Java runtime environment 中运行，并且已为 IBM MQ classes for JMS 设置 32 位环境，因为 64 位 Java runtime environment 无法装入 32 位 Java 本机库。

IBM MQ classes for JMS 配置文件

IBM MQ classes for JMS 配置文件指定用于配置 IBM MQ classes for JMS 的属性。

注: 配置文件中定义的属性也可设置为 JVM 系统属性。 如果某个属性既在配置文件中进行了设置，又设置为系统属性，那么系统属性优先。 因此，如果需要，可以通过在 **java** 命令中将配置文件中的任何属性设置为系统属性来将其覆盖。

IBM MQ classes for JMS 配置文件的格式是标准 Java 属性文件的格式。 IBM MQ classes for JMS 安装目录的 `bin` 子目录中提供了名为 `jms.config` 的样本配置文件。 该文件记录所有受支持的属性及其缺省值。

您可以选择 IBM MQ classes for JMS 配置文件的名称和位置。 启动应用程序时，请使用以下格式的 **java** 命令：

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

在该命令中，*config_file_url* 是统一资源定位符 (URL)，指定 IBM MQ classes for JMS 配置文件的名称和位置。 支持以下类型的 URL：http、file、ftp 和 jar。

以下是 **java** 命令的示例：

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

此命令将 IBM MQ classes for JMS 配置文件标识为本地 Windows 系统上的 `D:\mydir\mjms.config` 文件。

在应用程序启动时，IBM MQ classes for JMS 将读取配置文件的内容，并将指定的属性存储在内部属性库中。 如果 **java** 命令不识别配置文件，或者如果找不到配置文件，那么 IBM MQ classes for JMS 使用所有属性的缺省值。

IBM MQ classes for JMS 配置文件可以与应用程序和队列管理器或代理程序之间的任何受支持的传输一起使用。

覆盖 IBM MQ MQI client 配置文件中指定的属性

IBM MQ MQI client 配置文件还可以指定用于配置 IBM MQ classes for JMS 的属性。 但是，仅当应用程序以客户机方式连接到队列管理器时，IBM MQ MQI client 配置文件中指定的属性才适用。

如果需要，可以通过将 IBM MQ MQI client 配置文件中的任何属性指定为 IBM MQ classes for JMS 配置文件中的属性来将其覆盖。 要覆盖 IBM MQ MQI client 配置文件中的属性，请在 IBM MQ classes for JMS 配置文件中使用具有以下格式的条目：

```
com.ibm.mq.cfg. stanza. propName = propValue
```

该条目中的变量具有以下含义：

stanza
包含属性的 IBM MQ MQI client 配置文件中的节的名称

propName
在 IBM MQ MQI client 配置文件中指定的属性的名称

propValue
用于覆盖 IBM MQ MQI client 配置文件中指定属性值的特性的值

或者，您可以通过在 **java** 命令中将属性指定为系统属性来覆盖 IBM MQ MQI client 配置文件中的属性。 使用先前格式将属性指定为系统属性。

IBM MQ MQI client 配置文件中只有以下属性与 IBM MQ classes for JMS 相关。 如果您指定或覆盖其他属性，那么将无效。 具体来说，请注意，不会使用 客户机配置文件的 CHANNELS 节 中的 `ChannelDefinitionFile` 和 `ChannelDefinitionDirectory` 。 请参阅第 231 页的『将客户机通道定义表用于 IBM MQ classes for JMS』，以获取有关如何将 CCDT 与 IBM MQ classes for JMS 一起使用的详细信息。

表 *9:* 客户机配置文件的哪节包含哪个属性

节	属性
客户机配置文件的 CHANNELS 节	Put1DefaultAlwaysSync
客户机配置文件的 CHANNELS 节	DefRecon
客户机配置文件的 CHANNELS 节	ReconDelay
客户机配置文件的 CHANNELS 节	PasswordProtection
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath64
客户机配置文件的 ClientExitPath 节	JavaExitsClasspath
客户机配置文件的 JMQI 节	useMQCSPauthentication
客户机配置文件的 MessageBuffer 节	MaximumSize
客户机配置文件的 MessageBuffer 节	PurgeTime
客户机配置文件的 MessageBuffer 节	UpdatePercentage
客户机配置文件的 TCP 节	ClntRcvBufSize
客户机配置文件的 TCP 节	ClntSndBufSize
客户机配置文件的 TCP 节	Connect_Timeout
客户机配置文件的 TCP 节	KeepAlive

有关 IBM MQ MQI client 配置的进一步详细信息，请参阅使用配置文件配置客户机

Java 标准环境跟踪节
使用 Java 标准环境跟踪设置节配置 IBM MQ classes for JMS 跟踪功能。

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputName*
traceOutputName 是要将跟踪输出发送至的目录和文件名。

缺省情况下，会将跟踪信息写入应用程序当前工作目录中的跟踪文件。 跟踪文件的名称取决于应用程序的运行环境：

- 对于 IBM MQ classes for JMS for IBM MQ 9.0.0 Fix Pack 1 或更低版本，会将跟踪写入名为 `mqjms_%PID%.trc` 的文件。
- 从 IBM MQ 9.0.0 Fix Pack 2 起，如果应用程序已从 JAR 文件 `com.ibm.mqjms.jar` 装入 IBM MQ classes for JMS，那么会将跟踪写入名为 `mqjava_%PID%.trc` 的文件。
- 从 IBM MQ 9.0.0 Fix Pack 2 起，如果应用程序已从可重定位的 JAR 文件 `com.ibm.mq.allclient.jar` 装入 IBM MQ classes for JMS，那么会将跟踪写入名为 `mqjavaclient_%PID%.trc` 的文件。
- V 9.1.5 V 9.1.0.5 从 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5 开始，如果应用程序已从 JAR 文件 `com.ibm.mqjms.jar` 装入 IBM MQ classes for JMS，那么会将跟踪写入名为 `mqjava_%PID%.cl%u.trc` 的文件。
- V 9.1.5 V 9.1.0.5 从 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5 开始，如果应用程序已从可重新定位的 JAR 文件 `com.ibm.mq.allclient.jar` 装入 IBM MQ classes for JMS，那么会将跟踪写入名为 `mqjavaclient_%PID%.cl%u.trc` 的文件。

其中，*%PID%* 是正在跟踪的应用程序的进程标识，*%u* 是唯一编号，用于区分在不同 Java 类装入器下运行跟踪的线程之间的文件。

如果进程标识不可用，那么将生成一个随机数（以字母 `f` 为前缀）。 要将进程标识包含在您指定的文件名中，请使用字符串 `%PID%`。

如果指定备用目录，那么该目录必须存在并且您必须具有该目录的写许可权。 如果您没有写许可权，那么会将跟踪输出写入 `System.err`。

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList 是跟踪的程序包和类的列表，或者具有特殊值 `ALL` 或 `NONE`。

使用分号 `;` 分隔程序包或类名称。 *includeList* 缺省为 `ALL`，将跟踪 IBM MQ classes for JMS 中的所有程序包和类。

注: 您可以包含某个程序包，但在随后排除该程序包的子程序包。 例如，如果您包含程序包 `a.b` 并排除程序包 `a.b.x`，那么跟踪将包含 `a.b.y` 和 `a.b.z` 中的所有内容，但不包含 `a.b.x` 和 `a.b.x.1` 中的内容。

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList 是不跟踪的程序包和类的列表，或者具有特殊值 `ALL` 或 `NONE`。

使用分号 `;` 分隔程序包或类名称。 *excludeList* 缺省为 `NONE`，因此跟踪 IBM MQ classes for JMS 中的所有程序包和类。

注: 您可以排除某个程序包，但在随后包含该程序包的子程序包。 例如，如果您排除程序包 `a.b` 并包含程序包 `a.b.x`，那么跟踪将包含 `a.b.x` 和 `a.b.x.1` 中的所有内容，但不包含 `a.b.y` 和 `a.b.z`。

将包含在同一级别（包含和排除）指定的任何程序包或类。

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes 是从任何字节数组跟踪的最大字节数。

如果将 *maxArrayBytes* 设置为正整数，将限制字节数组中写出到跟踪文件的字节数。 将在写出 *maxArrayBytes* 之后截断字节数组。 设置 *maxArrayBytes* 可减小所生成跟踪文件的大小，降低跟踪对应用程序性能的影响。

该属性的值如果为 `0`，表示不会将任何字节数组的内容发送至跟踪文件。

缺省值为 `-1`，这将移除对字节数组中发送至跟踪文件的字节数的任何限制。

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes 是写入跟踪输出文件的最大字节数。

maxTraceBytes 将使用 *traceCycles*。 如果写入的跟踪字节数接近限制，将关闭文件，并启动新的跟踪输出文件。

值为 0 表示跟踪输出文件的长度为零。 缺省值为 -1，表示写入跟踪输出文件的数据量不受限制。

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles 是要循环的跟踪输出文件数。

如果当前的跟踪输出文件达到了 *maxTraceBytes* 指定的限制，那么将关闭该文件。 会按顺序将进一步的跟踪输出写入下一个跟踪输出文件。 通过将数字后缀附加到文件名来区分每个跟踪输出文件。 当前或最新的跟踪输出文件为 mqjms.trc.0，下一个最新的跟踪输出文件为 mqjms.trc.1。 较旧的跟踪文件将遵循相同的编号模式上限。

traceCycles 的缺省值为 1。 如果 *traceCycles* 为 1，那么在当前跟踪输出文件达到其最大大小时，将关闭并删除该文件。 将启动相同名称的新跟踪输出文件。 因此，同时只存在一个跟踪输出文件。

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters 将控制是否在跟踪中包含方法参数和返回值。

traceParameters 缺省为 TRUE。 如果将 *traceParameters* 设置为 FALSE，那么只会跟踪方法特征符。

com.ibm.msg.client.commonservices.trace.startup = *startup*
在分配资源期间存在 IBM MQ classes for JMS 的初始化阶段。 将在资源分配阶段期间对主要跟踪功能进行初始化。

如果将 *startup* 设置为 TRUE，将使用启动跟踪。 跟踪信息将立即生成并包含所有组件的设置，包括跟踪功能自身。 启动跟踪信息可用于诊断配置问题。 启动跟踪信息始终会写入 System.err。

startup 缺省为 FALSE。

将在初始化完成前检查 *startup*。 鉴于此，仅在命令行上将属性指定为 Java 系统属性。 请勿在 IBM MQ classes for JMS 配置文件中指定该属性。

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
将 *compressedTrace* 设置为 TRUE 以压缩跟踪输出。

compressedTrace 的缺省值为 FALSE。

如果将 *compressedTrace* 设置为 TRUE，那么将压缩跟踪输出。 缺省跟踪输出文件名具有扩展名 .trz。 如果将压缩设置为 FALSE（缺省值），那么该文件将具有扩展名 .trc 以指示它未压缩。 但是，如果在 *traceOutputName* 中指定了跟踪输出的文件名（将改用该名称）；将不会对该文件应用后缀。

压缩的跟踪输出小于未压缩的跟踪输出。 由于 I/O 较少，因此与未压缩的跟踪相比，写出速度更快。 与未压缩的跟踪相比，压缩的跟踪对 IBM MQ classes for JMS 的性能影响较小。

如果设置了 *maxTraceBytes* 和 *traceCycles*，将创建多个压缩跟踪文件以替代多个平面文件。

如果 IBM MQ classes for JMS 以不受控制的方式结束，那么压缩的跟踪文件可能无效。 鉴于此，只有当以受控方式关闭 IBM MQ classes for JMS 时，才能使用跟踪压缩。 只有当被调查的问题不会导致 JVM 自身意外停止时，才可以使用跟踪压缩。 如果诊断问题可能会导致 System.Halt() 关闭或出现异常、不受控制的 JVM 终止，请勿使用跟踪压缩。

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel 指定跟踪的过滤级别。 定义的跟踪级别如下所示：

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

每个跟踪级别包含所有较低级别。 例如，如果将跟踪级别设置为 TRACE_INFO，那么具有 TRACE_EXCEPTION、TRACE_WARNING 或 TRACE_INFO 定义级别的任何跟踪点都将写入跟踪。 所有其他跟踪点将被排除。

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace 控制是否在 WebSphere Application Server 环境中使用 IBM MQ JMS 客户机跟踪服务。

如果将 *standaloneTrace* 设置为 TRUE，那么 IBM MQ JMS 客户机跟踪属性将用于确定跟踪配置。

如果将 *standaloneTrace* 设置为 FALSE，并且 IBM MQ JMS 客户机正在 WebSphere Application Server 容器中运行，那么将使用 WebSphere Application Server 跟踪服务。 生成的跟踪信息取决于应用程序服务器的跟踪设置。

standaloneTrace 的缺省值为 FALSE。

Logging 节
使用 Logging 节配置 IBM MQ classes for JMS 日志设施。

以下属性可以包含在 Logging 节中:

com.ibm.msg.client.commonservices.log.outputName = path
IBM MQ classes for JMS 日志工具所使用的日志文件的名称。 缺省值为 `mqjms.log`，该值会写入到用于运行 IBM MQ classes for JMS 的 Java 运行时环境的当前工作目录。

属性可以具有下列其中一个值:

- 单路径名称
- 路径名的逗号分隔列表（将所有数据记录到所有文件）

每个路径名可以是绝对或相对路径名或:

“stderr”或“System.err”
表示标准错误流。

“stdout”或“System.out”
表示标准输出流。

com.ibm.msg.client.commonservices.log.maxBytes
通过对日志消息数据的任何调用记录的最大字节数。

正整数
写入数据直至达到每个日志调用的字节值。

0
不写入数据。

-1
无限写入数据（缺省值）。

com.ibm.msg.client.commonservices.log.limit
写入任何一个日志文件的最大字节数（缺省值为 262144）。

正整数
写入数据直至达到每个日志文件的字节值。

0
不写入数据。

-1
无限写入数据。

com.ibm.msg.client.commonservices.log.count
要循环的日志文件数。 当每个文件到达时，
`com.ibm.msg.client.commonservices.trace.limit` 跟踪将从下一个文件开始，缺省值为 3。

正整数
要循环的文件数。

0
单个文件。

Java SE Specifics 节
使用 Java SE 指定节来配置在 Java Standard Edition 环境中使用 IBM MQ classes for JMS 时使用的属性。

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE | FALSE
确定是否在 IBM MQ classes for JMS 生成 FDC 文件后立即写入 JavaCore 文件。 如果设置为 TRUE，那么将在运行 IBM MQ classes for JMS 的 Java 运行时环境的工作目录中生成 Java 核心文件。

TRUE
生成 JavaCore，取决于 Java 运行时环境能够执行此操作。

FALSE
不生成 JavaCore；这是缺省值。

IBM MQ Properties 节
使用 IBM MQ Properties 节设置影响 IBM MQ classes for JMS 与 IBM MQ 交互方式的属性。

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
当使用 IBM MQ classes for JMS 的应用程序使用 IBM MQ 消息传递提供程序迁移方式连接到 IBM MQ 队列管理器时，IBM MQ classes for JMS 将在接收消息时使用缺省缓冲区大小 4 KB。 如果应用程序尝试获取的消息大于 4 KB，那么 IBM MQ classes for JMS 会将缓冲区的大小调整的足够大以容纳消息。 然后，将在接收后续消息时使用较大的缓冲区大小。

该属性控制何时将缓冲区大小降低回到 4 KB。 缺省情况下，在收到小于较大缓冲区大小的十条连续消息时，缓冲区大小将降低回到 4 KB。 要在每次收到消息后将缓冲区大小重置回 4 KB，请将属性设置为值 0。

0

缓冲区将始终重置为缺省大小。

10

这是缺省值。 将在第十条消息后调整缓冲区大小。

com.ibm.msg.client.wmq.receiveConversionCCSID
当使用 IBM MQ classes for JMS 的应用程序要使用 IBM MQ 消息传递提供程序正常方式连接到 IBM MQ 队列管理器时，可以设置 `receiveConversionCCSID` 属性以覆盖用于从队列管理器接收消息的 MQMD 结构中的缺省 CCSID 值。 缺省情况下，MQMD 包含设置为 1208 的 CCSID 字段，但可以对此进行更改，例如在队列管理器无法将消息转换为该代码页的情况下。

有效值是任何有效的 CCSID 编号或以下值之一：

-1
使用平台缺省值。

1208
这是缺省值。

Client-mode specifics 节
使用 Client-mode specifics 节指定当 IBM MQ classes for JMS 连接到使用 CLIENT 传输的队列管理器时使用的属性。

com.ibm.mq.polling.RemoteRequestEntry
指定 IBM MQ classes for JMS 等待来自队列管理器的响应时用于检查中断连接的轮询时间间隔。

正整数
要在检查之前等待的毫秒数。 缺省值为 10000（即 10 秒）。 最小值是 3000，如果值更小，则仍会被视为 3000。

用于配置 *JMS* 客户机行为的属性
使用这些属性来配置 JMS 客户机的行为。

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE
JMS 2.0 规范介绍了对某些行为工作方式的更改。 IBM MQ 8.0 包括属性 `com.ibm.mq.jms.SupportMQExtensions`，该属性可以设置为 *TRUE*，以将这些更改后的行为还原

到先前的实现。 还原更改的行为可能对某些 JMS 2.0 应用程序很有必要，对某些使用 JMS 1.1 API 但是针对 IBM MQ 8.0 IBM MQ classes for JMS 运行的应用程序也必要。

TRUE
将 SupportMQExtensions 设置为 TRUE 将还原以下三个功能区：

消息优先级
可以将消息优先级指定为 *0* - *9*。 在 JMS 2.0 之前，消息还可以使用值 *-1*，指示使用队列的缺省优先级。 JMS 2.0 不允许将消息优先级设置为 *-1*。 启用 SupportMQExtensions 允许使用值 *-1*。

客户机标识
JMS 2.0 规范要求在非空客户机标识进行连接时检查其唯一性。 开启 SupportMQExtensions 意味着将忽略此要求并可以复用客户机标识。

NoLocal
JMS 2.0 规范要求，在开启该常量的情况下使用者无法接收由同一个客户机标识发布的消息。 在 JMS 2.0 之前，会在订户上设置此属性，以防止其接收自己的连接发布的消息。 启用 SupportMQExtensions 将此行为还原到其先前实现。

FALSE
保留行为的更改。

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
从 IBM MQ 8.0.0 Fix Pack 2 开始，在应用程序发送字节或流消息之后，IBM MQ classes for JMS 可以将刚发送的消息状态设置为只读或只写。

TRUE
对象在发送之后设置为只读。 设置此值会保持与 JMS 2.0 规范的兼容性。

FALSE
对象在发送之后设置为只写。 这是缺省值。

相关概念
第 265 页的『SupportMQExtensions 属性』
JMS 2.0 规范介绍了对某些行为工作方式的更改。 IBM MQ 8.0 及更高版本包含属性 com.ibm.mq.jms.SupportMQExtensions，可以将该属性设置为 *TRUE* 以将这些更改的行为还原回先前的实现。

z/OS 上 *IBM MQ classes for JMS* 的 *STEPLIB* 配置
在 z/OS 上，在运行时使用的 STEPLIB 必须包含 IBM MQ SCSQAUTH 和 SCSQANLE 库。 在启动 JCL 中或使用 .profile 文件指定这些库。

从 UNIX and Linux 系统服务中，可以使用 .profile 中的行来添加这些内容，如以下代码片段中所示，将 thlqual 替换为安装 IBM MQ 时选择的高级数据集限定符：

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

在其他环境中，通常需要编辑启动 JCL 以在 STEPLIB 并置上包含 SCSQAUTH 和 SCSQANLE：

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

IBM MQ classes for JMS 和软件管理工具
软件管理工具（如 Apache Maven）可以与 IBM MQ classes for JMS 一起使用。

许多大型开发组织都使用这些工具来集中管理第三方库的存储库。

IBM MQ classes for JMS 包含大量 JAR 文件。 在使用该 API 开发 Java 语言应用程序时，在开发应用程序的机器上需要安装 IBM MQ 服务器、IBM MQ 客户机或 IBM MQ 客户机 SupportPac。

如果要使用这样的工具，并将构成 IBM MQ classes for JMS 的 JAR 文件添加到集中管理的存储库，必须遵守以下要点：

- 存储库或容器必须仅对组织内部的开发人员可用。 禁止组织外部的任何分发。

- 存储库需要包含来自单个 IBM MQ 发行版或修订包的一组完整且一致的 JAR 文件。
- 您负责通过 IBM 支持人员提供的任何维护服务来更新存储库。

需要将以下 JAR 文件安装到存储库中:

- `com.ibm.mq.allclient.jar`.
- 如果使用 IBM MQ classes for JMS，那么需要 `jms.jar`。
- 如果使用 IBM MQ classes for JMS 并访问已存储在文件系统 JNDI 上下文中的 JMS 受管对象，那么需要 `fscontext.jar`。
- 如果使用 IBM MQ classes for JMS 并访问已存储在文件系统 JNDI 上下文中的 JMS 受管对象，那么需要 `providerutil.jar`。

从 IBM MQ 9.0 开始，需要使用 Bouncy Castle 安全提供程序和 CMS 支持 JAR 文件。 有关更多信息，请参阅第 70 页的『为 IBM MQ classes for JMS 安装了什么』和对非 IBM JRE 的支持。

在 *Java security manager* 下运行 *IBM MQ classes for JMS* 应用程序

IBM MQ classes for JMS 可以在 Java 安全管理器已启用的情况下运行。 要在 Java security manager 已启用的情况下成功运行应用程序，您必须使用合适的策略配置文件来配置 Java virtual machine (JVM)。

创建合适的策略定义文件的最简单方式是更改 Java runtime environment (JRE) 随附的策略配置文件。 在大多数系统上，此文件位于相对于您的 JRE 目录的 `lib/security/java.policy` 目录中。 您可以使用首选编辑器或者使用 JRE 提供的策略工具程序编辑策略配置文件。

要点:

在可能的情况下，术语 *allowlist*（白名单）已替换术语 *whitelist*（白名单）。 对于 IBM MQ 9.0 和更高发行版，这包括在本主题中提及的 Java 系统属性名称 (**com.ibm.mq.jms.***)。您无需更改任何现有配置。 先前系统属性名称也继续有效。

如果您的应用程序使用 Java security manager 机制，您必须授予以下权限:

- 您使用的任何允许列表文件上的 FilePermission，其中读许可权用于 ENFORCEMENT 方式，写许可权用于 DISCOVER 方式。
- **com.ibm.mq.jms.allowlist**、**com.ibm.mq.jms.allowlist.discover** 和 **com.ibm.mq.jms.allowlist.mode** 属性上的 PropertyPermission（读）。

有关更多信息，请参阅第 102 页的『允许列表概念』。

示例策略配置文件

以下是使 IBM MQ classes for JMS 能够在缺省安全管理器下成功运行的策略配置文件的示例。 此文件将需要定制，以指定某些文件和目录的位置：*MQ_INSTALLATION_PATH* 表示其中安装了 IBM MQ 的高级目录，*MQ_DATA_DIRECTORY* 表示 MQ 数据目录的位置，*QM_NAME* 是正在为其配置访问权的队列管理器的名称。

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
```

```
   permission javax.management.MBeanPermission "*","*";

   // We need to be able to read manifests etc from the jar files in the installation directory
   permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

   //Required if mqclient.ini/mqs.ini configuration files are used
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

   //For the client transport type.
   permission java.net.SocketPermission "*","connect,resolve";

   //For the bindings transport type.
   permission java.lang.RuntimePermission "loadLibrary.*";

   //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

   //For applications that use User Exits
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
   permission java.lang.RuntimePermission "createClassLoader";

   //Required for the z/OS platform
   permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

   // Used by the internal ConnectionFactory implementation
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

   // Used for controlled class loading
   permission java.lang.RuntimePermission "setContextClassLoader";

   // Used to default the Application name in Client mode connections
   permission java.util.PropertyPermission "sun.java.command","read";

   // Used by the IBM JSSE classes
   permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

   //Required to determine if an IBM Java Runtime is running in FIPS mode,
   //and to modify the property values status as required.
   permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
   permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
   //Required if an IBM FIPS provider is to be used for SSL communication.
   permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

   // Required for non-IBM Java Runtimes that establish secure client
   // transport mode connections using mutual TLS authentication
   permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
   permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

在示例中，`grant` 语句包含 IBM MQ classes for JMS 所需的许可权。 要在策略配置文件中使用这些 grant 语句，您可能需要根据 IBM MQ classes for JMS 的安装位置以及应用程序的存储位置来修改路径名。

IBM MQ classes for JMS 随附的样本应用程序以及用于运行这些应用程序的脚本不启用安全管理器。

IBM MQ classes for JMS 应用程序的后安装设置

此主题说明了 IBM MQ classes for JMS 应用程序需要哪些权限来访问队列管理器资源。 它还介绍了连接方式，并描述了如何配置队列管理器以使应用程序能够在客户机方式下进行连接。

请记住检查 IBM MQ 自述文件。 该文件可能包含可取代此主题中信息的信息。

需要对非特权用户进行授权的 *JMS* 使用的对象

非特权用户需要被授权，以访问 JMS 使用的队列。 每个 JMS 应用程序都需要对其运行所用的队列管理器进行授权。

有关 IBM MQ 中的访问控制的详细信息，请参阅设置安全性。

IBM MQ classes for JMS 应用程序需要针对队列管理器具有 `connect` 和 `inq` 权限。 可使用 **setmqaut** 控制命令来设置相应的权限，例如：

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

对于点到点域，需要以下权限：

- MessageProducer 对象使用的队列需要 put 权限。
- MessageConsumer 和 QueueBrowser 对象使用的队列需要 get、inq 和 browse 权限。
- QueueSession.createTemporaryQueue() 方法需要对 QueueConnectionFactory 对象的 TEMPMODEL 属性所指定的模型队列的访问权。 缺省情况下，此模型队列为 SYSTEM.TEMP.MODEL.QUEUE。

如果这些队列中的任意队列为别名队列，那么其目标队列需要 inquire 权限。 如果目标队列为集群队列，那么它还需要 browse 权限。

对于发布/预订域，如果 IBM MQ classes for JMS 正在以 IBM MQ 消息传递提供程序迁移方式连接到 IBM MQ 队列管理器，那么将使用以下队列：

- 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

有关 IBM MQ 消息传递提供程序迁移方式的更多信息，请参阅配置 JMS **PROVIDERVERSION** 属性

此外，如果 IBM MQ classes for JMS 正在以此方式连接到队列管理器，那么发布消息的任何应用程序都需要访问由 TopicConnectionFactory 或主题对象指定的流队列。 缺省情况下，此队列为 SYSTEM.BROKER.DEFAULT.STREAM。

如果使用 ConnectionConsumer、IBM MQ 资源适配器或 WebSphere Application Server IBM MQ 消息传递提供程序，那么可能需要其他权限。

由 ConnectionConsumer 读取的队列必须具有 get、inq 和 browse 权限。 系统死信队列和任何 ConnectionConsumer 使用的回退重排队列或报告队列必须有 put 和 passall 权限。

当应用程序使用 IBM MQ 消息传递提供程序正常方式来执行发布/预订消息传递时，它会利用队列管理器提供的集成发布/预订功能。 请参阅发布/预订安全性，以获取有关保护所使用的主题和队列的信息。

IBM MQ classes for JMS 的连接方式
IBM MQ classes for JMS 应用程序可以在客户机或绑定方式下连接到队列管理器。 IBM MQ classes for JMS 可在客户机方式下，通过 TCP/IP 连接到队列管理器。 IBM MQ classes for JMS 可在绑定方式下，使用 Java 本机接口 (JNI) 直接连接到队列管理器。

在 WebSphere Application Server on z/OS 中运行的应用程序可以以绑定或客户机方式连接到队列管理器，但在 z/OS 上的任何其他环境中运行的应用程序只能以绑定方式连接到队列管理器。 在任何其他平台上运行的应用程序均可在绑定或客户机方式下连接到队列管理器。

您可以将当前或任何早期受支持版本的 IBM MQ classes for JMS 与当前的队列管理器一起使用，并可将当前或早期受支持版本的队列管理器与当前版本的 IBM MQ classes for JMS 一起使用。 如果混用了不同的版本，那么功能将限于早期版本级别。

以下部分更详细地描述了每一种连接方式。

客户机方式

要在客户机方式下连接到队列管理器，IBM MQ classes for JMS 应用程序可以在运行队列管理器的同一系统或不同系统上运行。 在每种情况下，IBM MQ classes for JMS 均通过 TCP/IP 连接到队列管理器。

绑定方式

要在绑定方式下连接到队列管理器，IBM MQ classes for JMS 应用程序必须在运行队列管理器的同一系统上运行。

IBM MQ classes for JMS 使用 Java 本机接口 (JNI) 直接连接到队列管理器。 要使用绑定传输，IBM MQ classes for JMS 必须在能够访问 IBM MQ Java 本机接口库的环境中运行；请参阅第 75 页的『配置 Java 本机接口 (JNI) 库』以获取更多信息。

IBM MQ classes for JMS 针对 *ConnectOption* 支持以下值：

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

要更改 IBM MQ classes for JMS 所使用的连接选项，请修改“连接工厂”属性 CONNOPT。

有关连接选项的更多信息，请参阅第 687 页的『使用 MQCONNX 调用连接到队列管理器』

要使用绑定传输，所用 Java 运行时环境必须支持 IBM MQ classes for JMS 连接到的队列管理器的编码字符集标识 (CCSID)。

有关如何确定 Java 运行时环境支持的 CCSID 的详细信息可在 IBM MQ 使用 Java 的 IBM MQ V7 类或 JMS 的 IBM MQ V7 类时生成具有探测器标识 21 的 FDC 中找到。

配置队列管理器，以便 *IBM MQ classes for JMS* 应用程序能够在客户机方式下进行连接
要配置队列管理器，以便 IBM MQ classes for JMS 应用程序能够在客户机方式下进行连接，必须创建服务器连接通道定义并启动侦听器。

创建服务器连接通道定义

在所有平台上，均可使用 MQSC 命令 DEFINE CHANNEL，来创建服务器连接通道定义。 请参阅以下示例：

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i 在 IBM i 上，可改用 CL 命令 CRTMQMCHL，如以下示例中所示：

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

在此命令中，*QMGRNAME* 是队列管理器的名称。

Windows Linux 在 Linux 和 Windows 上，还可以使用 IBM MQ Explorer 创建服务器连接通道定义。

z/OS 在 z/OS 上，可使用操作和控制面板来创建服务器连接通道定义。

通道名称（以上示例中为 JAVA.CHANNEL）必须与应用程序用于连接到队列管理器的连接工厂的 CHANNEL 属性所指定的通道名称相同。 CHANNEL 属性的缺省值为 SYSTEM.DEF.SVRCONN。

启动侦听器

必须为队列管理器启动侦听器（如尚未启动）。

Multi 在 多平台 上，可以在使用 MQSC 命令 DEFINE LISTENER 初次创建侦听器对象后，使用 MQSC 命令 START LISTENER 启动侦听器，如以下示例中所示：

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS 在 z/OS 上，只能使用 START LISTENER 命令（如以下示例中所示），但请注意，在可启动侦听器之前，必须启动通道启动程序地址空间：

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i 在 IBM i 上，还可以使用 CL 命令 STRMQMLSR 来启动侦听器，如以下示例中所示：

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

在此命令中，*QMGRNAME* 是队列管理器的名称。

ULW 在 UNIX, Linux, and Windows 上，还可以使用控制命令 **runmqlsr** 来启动侦听器，如以下示例中所示：

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

在此命令中，*QMgrName* 是队列管理器的名称。

Windows Linux 在 Linux 和 Windows 上，还可以使用 IBM MQ Explorer 来启动侦听器。

z/OS 在 z/OS 上，还可以使用操作和控制面板来启动侦听器。

侦听器侦听的端口号必须与应用程序用于连接到队列管理器的连接工厂的 PORT 属性所指定的端口号相同。PORT 属性的缺省值为 1414。

针对 *IBM MQ classes for JMS* 进行点到点 *IVT*

与 IBM MQ classes for JMS 一起提供有点到点安装验证测试 (IVT) 程序。 该程序在绑定或客户机方式下连接到队列管理器，并向名为 SYSTEM.DEFAULT.LOCAL.QUEUE 的队列发送消息，然后从队列接收消息。 该程序可创建和配置在运行时动态需要的所有对象，或者可以使用 JNDI 来从目录服务检索受管对象。

在不先使用 JNDI 的情况下运行安装验证测试，因为该测试是自包含测试，无需使用目录服务。 要获取受管对象的描述，请参阅使用管理工具配置 JMS 对象。

在不使用 JNDI 的情况下进行点到点安装验证测试

在此测试中，IVT 程序会创建和配置在运行时动态需要的所有对象且不使用 JNDI。

提供了脚本来运行 IVT 程序。 此脚本在 UNIX and Linux 系统上称为 IVTRun，在 Windows 上称为 IVTRun.bat，位于 IBM MQ classes for JMS 安装目录的 bin 子目录中。

要在绑定方式下运行测试，请输入以下命令：

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

要以客户机方式运行测试，请首先设置队列管理器，如 第 972 页的『配置队列管理器以接受多平台上的客户机连接』中所述。 请注意，要使用的通道缺省为 **SYSTEM.DEF.SVRCONN**，要使用的队列为 **SYSTEM.DEFAULT.LOCAL.QUEUE**，然后输入以下命令：

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

z/OS 系统上未提供等效的脚本，但可使用以下命令，通过直接调用 Java 类，在绑定方式下运行 IVT：

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

类路径必须包含 com.ibm.mqjms.jar。

这些命令上的参数具有以下含义：

-m *qmgr*
IVT 程序连接到的队列管理器的名称。 如果在绑定方式下运行测试且省略此参数，那么 IVT 程序将连接到缺省队列管理器。

-host *hostname*
运行队列管理器的系统的主机名或 IP 地址。

-port *port*
队列管理器的侦听器正在侦听的端口的编号。 缺省值为 1414。

-channel *channel*
IVT 程序用于连接到队列管理器的 MQI 通道的名称。 缺省值为 SYSTEM.DEF.SVRCONN。

-v *providerVersion*
IVT 程序预期连接到的队列管理器的发行版级别。

此参数用于设置 MQQueueConnectionFactory 对象的 PROVIDERVERSION 属性，并具有与 PROVIDERVERSION 属性值相同的有效值。 有关此参数（包括其有效值）的更多信息，请参阅 JMS：对 PROVIDERVERSION 属性的更改，以及 IBM MQ classes for JMS 对象属性中 PROVIDERVERSION 属性的描述。

缺省值为 unspecified。

-ccsid *ccsid*
连接使用的编码字符集的标识 (CCSID) 或代码页。 缺省值为 819。

-t
已启用跟踪。 缺省情况下，已禁用跟踪。

测试成功后，将生成类似于以下样本输出的输出：

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

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

Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
```

```
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

使用 JNDI 进行点到点安装验证测试

在此测试中，IVT 程序使用 JNDI 从目录服务检索受管对象。

在可以运行测试前，必须配置基于轻量级目录访问协议 (LDAP) 服务器或本地文件系统的目录服务。 还必须配置 IBM MQ JMS 管理工具，以便其能够使用目录服务存储受管对象。 有关这些先决条件的更多信息，请参阅第 69 页的『IBM MQ classes for JMS 的先决条件』。 有关如何配置 IBM MQ JMS 管理工具的信息，请参阅配置 JMS 管理工具。

IVT 程序必须能够使用 JNDI 来从目录服务检索 MQQueueConnectionFactory 对象和 MQQueue 对象。 提供了脚本来为您创建这些受管对象。 此脚本在 UNIX and Linux 系统上称为 IVTSetup，在 Windows 上称为 IVTSetup.bat，并且位于 IBM MQ classes for JMS 安装目录的 bin 子目录中。 要运行此脚本，请输入以下命令：

```
IVTSetup
```

此脚本调用 IBM MQ JMS 管理工具来创建受管对象。

MQQueueConnectionFactory 对象与 ivtQCF 名称相关联，并使用其所有属性的缺省值来创建，这意味着 IVT 程序在绑定方式下运行，并连接到缺省队列管理器。 如果要使 IVT 程序在客户机方式下运行，或连接到除缺省队列管理器以外的队列管理器，那么必须使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 来更改 MQQueueConnectionFactory 对象的相应的属性。 有关如何使用 IBM MQ Explorer JMS 管理工具的信息，请参阅使用管理工具配置 JMS 对象。 有关如何使用 IBM MQ Explorer 的信息，请参阅 IBM MQ Explorer 的简介或 IBM MQ Explorer 随附的帮助。

MQQueue 对象与 ivtQ 名称相关联，并使用其所有属性的缺省值来创建，QUEUE 属性（其值为 SYSTEM.DEFAULT.LOCAL.QUEUE）除外。

创建受管对象时，可运行 IVT 程序。 要使用 JNDI 运行测试，请输入以下命令：

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

此命令上的参数具有以下含义：

-url "*providerURL*"
目录服务的统一资源定位符 (URL)。 URL 可以是以下某种格式：

- `ldap://hostname/contextName` （对于基于 LDAP 服务器的目录服务）
- `file:/directoryPath` （对于基于本地文件系统的目录服务）

请注意，必须使用引号 (") 将 URL 引起来。

-icf *initCtxFact*
初始上下文工厂的类名，其必须是以下值之一：

- `com.sun.jndi.ldap.LdapCtxFactory`（对于基于 LDAP 服务器的目录服务）。 这是缺省值。
- `com.sun.jndi.fscontext.RefFSContextFactory`（对于基于本地文件系统的目录服务）。

-t
已启用跟踪。 缺省情况下，已禁用跟踪。

测试成功后，将生成类似于不使用 JNDI 成功测试时的内容的输出。 主要的区别在于，此输出指示测试正在使用 JNDI 来检索 MQQueueConnectionFactory 对象和 MQQueue 对象。

虽然不是严格需要，但最好是在测试后通过删除由 IVTSetup 脚本创建的受管对象来进行整理。 为此，系统提供了脚本。 此脚本在 UNIX and Linux 系统上名为 IVTTidy，在 Windows 上名为 IVTTidy.bat，并且位于 IBM MQ classes for JMS 安装目录的 bin 子目录中。

确定点到点安装验证测试的问题

安装验证测试可能出于以下原因而失败：

- 如果 IVT 程序写入一条消息，指示其找不到类，请检查是否正确设置了类路径，如第 73 页的『为 IBM MQ classes for JMS 设置环境变量』中所述。
- 测试可能失败，显示以下消息：

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

及关联原因码 2059。 此消息中的变量具有以下含义：

QMGR
IVT 程序尝试连接到的队列管理器的名称。 如果 IVT 程序尝试在绑定方式下连接到缺省队列管理器，那么插入的此消息为空。

connMode
连接方式，可以是 `Bindings` 或 `Client`。

HOSTNAME
运行队列管理器的系统的主机名或 IP 地址。

此消息意味着 IVT 程序正在尝试连接到的队列管理器不可用。 请检查队列管理器是否正在运行，如果 IVT 程序正在尝试连接到缺省队列管理器，请确保将此队列管理器定义为系统的缺省队列管理器。

- 测试可能失败，显示以下消息：

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

此消息意味着 IVT 程序连接到的队列管理器上不存在队列 SYSTEM.DEFAULT.LOCAL.QUEUE。 或者，如果队列确实存在，那么表示 IVT 程序无法打开此队列，因为未启用它来放置和获取消息。 请检查队列是否存在，以及是否已启用它来放置和获取消息。

- 测试可能失败，显示以下消息：

```
Unable to bind to object
```

此消息意味着已连接到 LDAP 服务器，但未正确配置 LDAP 服务器。 未配置 LDAP 服务器来存储 Java 对象，或对象上的权限不正确或后缀不正确。 要在此情况下获取更多帮助，请参阅 LDAP 服务器文档。

- 测试可能失败，显示以下消息：

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

此消息意味着未正确设置队列管理器，以接受来自系统的客户机连接。 有关详细信息，请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』。

针对 *IBM MQ classes for JMS* 进行发布/预订 *IVT*

发布/预订安装验证测试 (IVT) 程序与 IBM MQ classes for JMS 一起提供。 该程序在绑定或客户机方式下连接到队列管理器，预订主题，发布有关主题的消息，然后接收刚刚发布的消息。 该程序可创建和配置在运行时动态需要的所有对象，或者可以使用 JNDI 来从目录服务检索受管对象。

在不先使用 JNDI 的情况下运行安装验证测试，因为该测试是自包含测试，无需使用目录服务。 要获取受管对象的描述，请参阅使用管理工具配置 JMS 对象。

在不使用 JNDI 的情况下进行发布/预订安装验证测试

在此测试中，IVT 程序会创建和配置在运行时动态需要的所有对象且不使用 JNDI。

提供了脚本来运行 IVT 程序。 此脚本在 UNIX and Linux 系统上称为 PSIVTRun，在 Windows 上称为 PSIVTRun.bat，并且位于 IBM MQ classes for JMS 安装目录的 `bin` 子目录中。

要在绑定方式下运行测试，请输入以下命令：

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

要在客户机方式下运行测试，请首先设置第 972 页的『配置队列管理器以接受多平台上的客户机连接』中所述的队列管理器（注意缺省情况下使用的通道为 SYSTEM.DEF.SVRCONN），然后输入以下命令：

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

这些命令上的参数具有以下含义：

-m *qmgr*
IVT 程序连接到的队列管理器的名称。 如果在绑定方式下运行测试且省略此参数，那么 IVT 程序将连接到缺省队列管理器。

-host *hostname*
运行队列管理器的系统的主机名或 IP 地址。

-port *port*
队列管理器的侦听器正在侦听的端口的编号。 缺省值为 1414。

-channel *channel*
IVT 程序用于连接到队列管理器的 MQI 通道的名称。 缺省值为 SYSTEM.DEF.SVRCONN。

-bqm *brokerQmgr*
运行代理程序的队列管理器的名称。 缺省值为 IVT 程序连接到的队列管理器的名称。

此参数与队列管理器版本号 v 7（或更高）无关。

-v *providerVersion*
IVT 程序预期连接到的队列管理器的发行版级别。

此参数用于设置 MQTopicConnectionFactory 对象的 PROVIDERVERSION 属性，并具有与 PROVIDERVERSION 属性值相同的有效值。 有关此参数（包括其有效值）的更多信息，请参阅 IBM MQ classes for JMS 对象属性中的 PROVIDERVERSION 属性描述。

缺省值为 unspecified。

-ccsid *ccsid*
连接使用的编码字符集的标识 (CCSID) 或代码页。 缺省值为 819。

-t
已启用跟踪。 缺省情况下，已禁用跟踪。

测试成功后，将生成类似于以下样本输出的输出：

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java(tm) Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
```

```
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

使用 JNDI 进行发布/预订安装验证测试

在此测试中，IVT 程序使用 JNDI 从目录服务检索受管对象。

在可以运行测试前，必须配置基于轻量级目录访问协议 (LDAP) 服务器或本地文件系统的目录服务。 还必须配置 IBM MQ JMS 管理工具，以便其能够使用目录服务存储受管对象。 有关这些先决条件的更多信息，请参阅第 69 页的『IBM MQ classes for JMS 的先决条件』。 有关如何配置 IBM MQ JMS 管理工具的信息，请参阅配置 JMS 管理工具。

IVT 程序必须能够使用 JNDI 来从目录服务检索 MQTopicConnectionFactory 对象和 MQTopic 对象。 提供了脚本来为您创建这些受管对象。 此脚本在 UNIX and Linux 系统上称为 IVTSetup，在 Windows 上称为 IVTSetup.bat，并且位于 IBM MQ classes for JMS 安装目录的 `bin` 子目录中。 要运行此脚本，请输入以下命令：

```
IVTSetup
```

此脚本调用 IBM MQ JMS 管理工具来创建受管对象。

MQTopicConnectionFactory 对象与 ivtTCF 名称相关联，并使用其所有属性的缺省值来创建，这意味着 IVT 程序在绑定方式下运行，连接到缺省队列管理器，并使用嵌入的发布/预订功能。 如果要使 IVT 程序在客户机方式下运行、连接到除缺省队列管理器以外的队列管理器或使用 IBM Integration Bus（而不是嵌入的发布/预订功能），那么必须使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 来更改 MQTopicConnectionFactory 对象的相应属性。 有关如何使用 IBM MQ JMS 管理工具的信息，请参阅使用管理工具配置 JMS 对象。 有关如何使用 IBM MQ Explorer 的信息，请参阅与 IBM MQ Explorer 一起提供的帮助。

MQTopic 对象与 ivtT 名称相关联，并使用其所有属性的缺省值来创建，TOPIC 属性（其值为 MQJMS/PSIVT/Information）除外。

创建受管对象时，可运行 IVT 程序。 要使用 JNDI 运行测试，请输入以下命令：

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

此命令上的参数具有以下含义：

-url "*providerURL*"
目录服务的统一资源定位符 (URL)。 URL 可以是以下某种格式：

- `ldap://`*`hostname`*`/`*`contextName`* （对于基于 LDAP 服务器的目录服务）
- `file:/`*`directoryPath`* （对于基于本地文件系统的目录服务）

请注意，必须使用引号 (") 将 URL 引起来。

-icf *initCtxFact*
初始上下文工厂的类名，其必须是以下值之一：

- `com.sun.jndi.ldap.LdapCtxFactory`（对于基于 LDAP 服务器的目录服务）。 这是缺省值。
- `com.sun.jndi.fscontext.RefFSContextFactory`（对于基于本地文件系统的目录服务）。

-t
已启用跟踪。 缺省情况下，已禁用跟踪。

测试成功后，将生成类似于不使用 JNDI 成功测试时的内容的输出。 主要的区别在于，此输出指示测试正在使用 JNDI 来检索 MQTopicConnectionFactory 对象和 MQTopic 对象。

虽然不是严格需要，但最好是在测试后通过删除由 IVTSetup 脚本创建的受管对象来进行整理。 为此，系统提供了脚本。 此脚本在 UNIX and Linux 系统上名为 IVTTidy，在 Windows 上名为 IVTTidy.bat，并且位于 IBM MQ classes for JMS 安装目录的 `bin` 子目录中。

确定发布/预订安装验证测试的问题

安装验证测试可能出于以下原因而失败：

- 如果 IVT 程序写入一条消息，指示其找不到类，请检查是否正确设置了类路径，如第 73 页的『为 IBM MQ classes for JMS 设置环境变量』中所述。
- 测试可能失败，显示以下消息：

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

及关联原因码 2059。 此消息中的变量具有以下含义：

QMGR
IVT 程序尝试连接到的队列管理器的名称。 如果 IVT 程序尝试在绑定方式下连接到缺省队列管理器，那么插入的此消息为空。

connMode
连接方式，可以是 `Bindings` 或 `Client`。

HOSTNAME
运行队列管理器的系统的主机名或 IP 地址。

此消息意味着 IVT 程序正在尝试连接到的队列管理器不可用。 请检查队列管理器是否正在运行，如果 IVT 程序正在尝试连接到缺省队列管理器，请确保将此队列管理器定义为系统的缺省队列管理器。

- 测试可能失败，显示以下消息：

```
Unable to bind to object
```

此消息意味着已连接到 LDAP 服务器，但未正确配置 LDAP 服务器。 未配置 LDAP 服务器来存储 Java 对象，或对象上的权限不正确或后缀不正确。 要在此情况下获取更多帮助，请参阅 LDAP 服务器文档。

- 测试可能失败，显示以下消息：

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

此消息表示未正确设置队列管理器以接受来自系统的客户机连接。 有关更多信息，请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』。

使用 *IBM MQ classes for JMS* 样本应用程序

IBM MQ classes for JMS 样本应用程序概述了 JMS API 的常用功能。 使用这些样本应用程序，可以验证安装和消息传递服务器设置，并帮助您构建自己的应用程序。

关于此任务

如果您在创建自己的应用程序时需要帮助，那么可以使用这些样本应用程序作为起点。 已提供了每个样本应用程序的源代码和已编译的版本。 请查看样本源代码，并确定关键步骤，以便为应用程序创建所有必需对象（ConnectionFactory、Connection、Session、Destination 以及 Producer 和/或 Consumer）并根据需要设置任何特定属性以指定应用程序的工作方式。 有关更多信息，请参阅第 109 页的『编写 IBM MQ classes for JMS 应用程序』。 在 IBM MQ 的未来发行版中可随时更改这些样本。

第 95 页的表 10 显示了 IBM MQ classes for JMS 样本应用程序在每个平台上的安装位置：

表 *10: IBM MQ classes for JMS* 样本应用程序的安装目录	
平台	**目录**
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*/samp/jms/samples
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms\samples
IBM i IBM i	/qibm/proddata/mqm/java/samples/jms/samples
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

在此目录中，有一些包含一个或多个样本应用程序（如第 95 页的表 11 中所示）的子目录。

表 *11: IBM MQ classes for JMS* 样本应用程序	
样本名称	**描述**
JmsBrowser.java	一个 JMS 队列浏览器应用程序，它会按照使用者应用程序接收消息的顺序来查看指定队列上的所有可用消息，但不移除这些消息。
JmsConsumer.java	一个 JMS 队列浏览器应用程序，它通过在初始上下文（此样本仅支持文件系统上下文）中查找连接工厂实例和目标实例，按照使用者应用程序接收消息的顺序来查看指定队列上的所有可用消息，但不移除这些消息。
JmsJndiConsumer.java	一个 JMS 使用者（接收方或订户）应用程序，它通过在初始上下文（此样本仅支持文件系统上下文）中查找连接工厂实例和目标实例，接收来自指定目标（队列或主题）的消息。
JmsJndiProducer.java	一个 JMS 生产者（发送方或发布者）应用程序，它通过在初始上下文（此样本仅支持文件系统上下文）中查找连接工厂实例和目标实例，向指定目标（队列或主题）发送简单消息。
JmsProducer.java	一个 JMS 生产者（发送方或发布者）应用程序，它会向指定目标（队列或主题）发送简单消息。
/interactive/	
SampleConsumerJava.java	用于接收来自主题/队列的消息。
SampleProducerJava.java	用于向主题/队列发送消息。
/interactive/helper/	
BaseOptions.java	一个抽象类，可扩展为提供用户选项功能。
IsValidType.java	有效性校验器类的抽象类。

表 *11: IBM MQ classes for JMS* 样本应用程序 *(继续)*	
样本名称	**描述**
JmsApp.java	一个抽象类，可扩展为提供使用者/生产者功能。
Keys.java	一组关键字，用于定义样本应用程序的选项。
Literals.java	一组常量字面值。
MyContext.java	用于提供选项的上下文。
Options.java	用于提供用户选项功能。
OptionsPresenter.java	用于提供当前选项的上下文。
/simple/	
SimpleAsyncPutPTP.java	一个简单应用程序，它用于点到点消息传递，并以异步方式发送消息（也称为即发即弃消息传递）。 它不接收任何消息。
SimpleDurableSub.java	一个简单应用程序，它会演示持久预订功能。
SimpleJNDILookup.java	一个极简的应用程序，它会演示如何使用初始上下文来查找 JMS 对象。 它不建立到队列管理器的连接，并且不发送或接收任何消息。
SimpleMQMDRead.java	一个简单应用程序，它会演示 JMS 应用程序如何使用 MQ 消息描述符 (MQMD) 字段作为 JMS 消息属性。 它不发送任何消息；假定正在使用的队列中已填充了一些消息。
SimpleMQMDWrite.java	一个简单应用程序，它会演示 JMS 应用程序如何编写 MQ 消息描述符 (MQMD) 字段。 它不接收任何消息。
SimplePTP.java	一个极简的应用程序，它用于点到点消息传递。
SimplePubSub.java	一个极简的应用程序，它用于发布/预订消息传递。
SimpleReadAheadPTP.java	一个简单应用程序，它用于点到点消息传递，并以流方式传递队列管理器中的消息（也称为预读功能）。 它不发送任何消息；假定正在使用的队列中已填充了一些消息。
SimpleRequestor.java	一个简单应用程序，它会使用请求程序来发送请求消息，然后等待，接收，再应答。 注：假定将由某个其他应用程序处理请求消息并发送应答消息。
SimpleResponder.java	一个简单应用程序，它会侦听目标以获取消息，然后将应答发送到消息的 replyTo 目标。 编写此应用程序是为了与 SimpleRequestor 样本配合使用。
SimpleRetainedPub.java	一个简单应用程序，它会演示一个保留发布。 它不接收任何消息。
SimpleWMQJMSPTP.java	一个极简的应用程序，它用于点到点消息传递。
SimpleWMQJMSPubSub.java	一个极简的应用程序，它用于发布/预订消息传递。

IBM MQ classes for JMS 提供了一个名为 `runjms` 的脚本，此脚本可用于运行样本应用程序。 该脚本会设置 IBM MQ 环境，以便您可以运行 IBM MQ classes for JMS 样本应用程序。

第 97 页的表 12 显示了该脚本在每个平台上的位置。

表 *12: runjms* 脚本的位置	
平台	**目录**
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*/java/bin/runjms
Windows Windows	*MQ_INSTALLATION_PATH*\java\bin\runjms.bat
IBM i IBM i	/qibm/proddata/mqm/java/bin/runjms 或者 /qibm/proddata/mqm/java/bin/runjms64
z/OS z/OS	*MQ_INSTALLATION_PATH*java/bin/runjms

要使用 runjms 脚本来调用样本应用程序，请完成以下步骤：

过程

1. 打开命令提示符，浏览至包含要运行的样本应用程序的目录。
2. 输入以下命令：

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   该样本应用程序会显示其需要的参数列表。
3. 输入以下命令以使用这些参数运行该样本：

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

示例

Linux 例如，要在 Linux 上运行 JmsBrowser 样本，请输入以下命令：

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

相关概念

第 70 页的『为 IBM MQ classes for JMS 安装了什么』
安装 IBM MQ classes for JMS 时，将会创建大量文件和目录。 在 Windows 上，通过自动设置环境变量在安装期间执行了某些配置。 在其他平台上的特定 Windows 环境中，必须先设置环境变量，然后才能运行 IBM MQ classes for JMS 应用程序。

与 *IBM MQ classes for JMS* 一起提供的脚本

系统提供多个脚本来帮助完成在使用 IBM MQ classes for JMS 时需要执行的常见任务。

第 97 页的表 13 列出所有脚本及其用途。 这些脚本位于 IBM MQ classes for JMS 安装目录的 bin 子目录中。

表 *13: 与 IBM MQ classes for JMS* 一起提供的脚本	
实用程序	**适用平台**
Cleanup [1]	此脚本为与先前发行版兼容而保留，但不未执行任何功能。 无需再手动清除预订信息
DefaultConfiguration	在除 Windows 外的平台上运行缺省配置应用程序。

表 13: 与 IBM MQ classes for JMS 一起提供的脚本 (继续)	
实用程序	**适用平台**
formatLog [1]	此脚本为与先前发行版兼容而保留，但不未执行任何功能。 现在以可读文本形式生成日志输出。
IVTRun [1] IVTSetup [1] IVTTidy [1]	用于点到点安装验证测试，如第 88 页的『针对 IBM MQ classes for JMS 进行点到点 IVT』中所述。
JMSAdmin [1]	运行 IBM MQ JMS 管理工具，如启动管理工具中所述。
JMSAdmin.config	IBM MQ JMS 管理工具配置文件，如配置 JMS 管理工具中所述。
PSIVTRun [1]	运行发布/预订安装验证测试程序，如第 91 页的『针对 IBM MQ classes for JMS 进行发布/预订 IVT』中所述。
PSReportDump.class	维护此类是为了与以前的发行版兼容，但不执行任何功能。
setjmsenv	设置用于在 UNIX and Linux 系统上的 32 位 Java 虚拟机 (JVM) 中运行 IBM MQ classes for JMS 应用程序的环境变量，如 第 73 页的『为 IBM MQ classes for JMS 设置环境变量』中所述。
setjmsenv64	设置用于在 UNIX and Linux 系统上的 64 位 JVM 中运行 IBM MQ classes for JMS 应用程序的环境变量，如 第 73 页的『为 IBM MQ classes for JMS 设置环境变量』中所述。

注:

1. 在 Windows 上，文件扩展名为 .bat。

OSGi 支持

OSGi 提供了一个框架，支持以捆绑软件形式部署应用程序。 在 IBM MQ classes for JMS 中提供了 9 个 OSGi 捆绑软件。

OSGi 提供了一个通用的安全受管 Java 框架，此框架支持部署以捆绑软件形式提供的应用程序。 与 OSGi 兼容的设备可以下载和安装捆绑软件，并在不再需要时移除这些捆绑软件。 此框架使用动态的可扩展方式管理捆绑软件的安装和更新过程。

IBM MQ classes for JMS 包含以下 OSGi 捆绑软件。

com.ibm.msg.client.osgi.jms*version_number*.jar
IBM MQ classes for JMS 中的通用代码层。 有关 IBM MQ classes for JMS 的分层体系结构的信息，请参阅 IBM MQ classes for JMS 体系结构。

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
通用层必备的 Java 归档 (JAR) 文件。

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Java Platform, Standard Edition (Java SE) 应用程序的通用服务。

com.ibm.msg.client.osgi.nls_*version_number*.jar
通用层的消息。

com.ibm.msg.client.osgi.wmq_*version_number*.jar
IBM MQ classes for JMS 中的 IBM MQ 消息传递提供程序。 有关 IBM MQ classes for JMS 的分层体系结构的信息，请参阅 IBM MQ classes for JMS 体系结构。

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
IBM MQ 消息传递提供程序必备的 JAR 文件。

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
IBM MQ 消息传递提供程序的消息。

com.ibm.mq.osgi.allclient_*version_number*.jar
此 JAR 文件允许应用程序同时使用 IBM MQ classes for JMS 和 IBM MQ classes for Java，并且还包含用于处理 PCF 消息的代码。

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
此 JAR 文件提供了 com.ibm.mq.osgi.allclient_*version_number*.jar 的先决条件，其中 *version_number* 是已安装的 IBM MQ 的版本号。

这些捆绑软件安装在 IBM MQ 安装的 java/lib/OSGi 子目录或 Windows 上的 java\lib\OSGi 文件夹中。

从 IBM MQ 8.0 起，对任何新应用程序都使用捆绑软件 com.ibm.mq.osgi.allclient_8.0.0.0.jar 和 com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar。 通过使用这些捆绑软件，可以消除无法在同一 OSGi 框架内同时运行 IBM MQ classes for JMS 和 IBM MQ classes for Java 的限制，但所有其他限制仍然适用。 对于 IBM MQ 8.0 之前的产品版本，使用 IBM MQ classes for JMS 或 IBM MQ classes for Java 的这一限制将适用。

捆绑软件 com.ibm.mq.osgi.java*version_number*.jar（也安装在 IBM MQ 安装的 java/lib/OSGi 子目录或 Windows 上的 java\lib\OSGi 文件夹中）是 IBM MQ classes for Java 的一部分。 不能将此捆绑软件装入到已装入 IBM MQ classes for JMS 的 OSGi 运行时环境中。

IBM MQ classes for JMS 的 OSGi 捆绑软件已写入 OSGi R4 规范中。 它们不能在 OSGi R3 环境下运行。

必须正确设置系统路径或库路径，以便 OSGi 运行时环境能够找到任何必需的 DLL 文件或共享库。

如果将 OSGi 捆绑软件用于 IBM MQ classes for JMS，那么临时主题将不适用。 此外，也不支持用 Java 编写的通道出口类，因为在像 OSGi 这样的多类装入器环境中装入类时存在固有的问题。 用户捆绑软件可以感知到 IBM MQ classes for JMS 捆绑软件，但 IBM MQ classes for JMS 捆绑软件无法感知到任何用户捆绑软件。 因此，IBM MQ classes for JMS 捆绑软件中使用的类装入器无法装入用户捆绑软件中的通道出口类。

有关 OSGi 的更多信息，请访问 OSGi Alliance Web 站点。

V 9.1.0 z/OS MQ Adv. VUE JMS 客户机与在 z/OS 上运行的批处理应用程序的连接

通过使用客户机连接， z/OS 上的 IBM MQ classes for JMS 应用程序可以连接到 z/OS 上具有 **ADVCAP**(ENABLED) 属性的队列管理器。 使用客户机连接可以简化 IBM MQ 拓扑。

值 **ADVCAP**（ENABLED）仅适用于 z/OS 队列管理器，许可为 IBM MQ Advanced for z/OS Value Unit Edition (请参阅 IBM MQ 产品标识和导出信息)，并在 **QMGRPROD** 设置为 ADVANCEDVUE 的情况下运行。

请参阅 DISPLAY QMGR 以获取有关 **ADVCAP** 的更多信息，并参阅 START QMGR 以获取有关 **QMGRPROD** 的更多信息。

请注意，批处理是唯一受支持的环境；不支持 JMS for CICS 或 JMS for IMS。

z/OS 上的 IBM MQ classes for JMS 应用程序无法使用客户机方式连接来连接到未在 z/OS 上运行的队列管理器或未设置 **ADVCAP**（ENABLED）选项的队列管理器。

如果 z/OS 上的 IBM MQ classes for JMS 应用程序尝试使用客户机方式进行连接，并且不允许这样做，那么将发出异常消息 JMSFMQ0005 。

Advanced Message Security (AMS) 支持

从 IBM MQ 9.1 开始， IBM MQ classes for JMS 客户机应用程序可以在连接到远程 z/OS 系统上的 IBM MQ Advanced for z/OS Value Unit Edition 队列管理器时使用 AMS 。

仅在 z/OS 上的 keystore.conf 中支持新的密钥库类型 jceracfks，其中:

- 属性名称前缀是 jceracfks，此名称前缀不区分大小写。
- 密钥库是 RACF 密钥环。
- 无需密码，因此将忽略密码。 原因是 RACF 密钥环不使用密码。
- 如果指定了提供程序，那么提供程序必须为 IBMJCE。

将 jceracfks 与 AMS 一起使用时，密钥库必须采用以下格式：safkeyring://*user*/*keyring*，其中：

- safkeyring 是字面值，此名称不区分大小写
- *user* 是拥有此密钥环的 RACF 用户标识
- *keyring* 是 RACF 密钥环的名称，密钥环名称区分大小写

以下示例将标准 AMS 密钥环用于用户 JOHNDOE:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

单独获取 IBM MQ classes for JMS

如果只想获取 IBM MQ classes for JMS JAR 文件，要部署到软件管理工具中，或者要与独立客户机应用程序配合使用，那么可以从 Fix Central 下载自解压 JAR 文件中提供 IBM MQ classes for JMS 。

开始之前

启动此任务之前，确保您已经在机器上安装 Java runtime environment (JRE) 并且 JRE 已经添加到系统路径。

此安装过程中使用的 Java 安装程序不需要以 root 用户或任何特定用户身份运行。 只要求运行它的用户具有您想要放入文件的目录的写入权限。

关于此任务

在 IBM MQ 8.0 之前，IBM WebSphere MQ classes for Java 或 IBM WebSphere MQ classes for JMS 不可用作单独的下载。 对于 IBM WebSphere MQ 7.5 或更早版本，如果您正在开发和运行使用 IBM WebSphere MQ classes for Java 或 IBM WebSphere MQ classes for JMS 的 Java 语言应用程序，那么需要通过执行完整服务器安装或者通过将其中一个客户机 SupportPacs 安装到正在开发应用程序的系统和将运行应用程序的系统上来安装这些应用程序。 此安装所安装的文件要比 IBM WebSphere MQ classes for Java 和 IBM WebSphere MQ classes for JMS 文件多。

但是，从 IBM MQ 8.0 开始，以下文件在自解压缩 JAR 文件中可用，这将最大限度地降低下载和安装的大小，并缩短执行安装所需要的时间：

- IBM MQ classes for JMS
- IBM MQ classes for Java
- IBM MQ 资源适配器
- IBM MQ OSGi 捆绑软件

V 9.1.0.8 从 IBM MQ 9.1.0 Fix Pack 8 开始， JMSAdmin 工具作为自解压 JAR 文件的一部分进行安装，该文件包含与 JMSAdmin 工具相关的以下额外文件：

- 用于在 Windows 上启动 JMSAdmin 工具的 JMSAdmin.bat 文件。
- JMSAdmin 脚本，用于在 Linux 和 UNIX 平台上启动此工具。
- JMSAdmin 工具的样本配置文件 (JMSAdmin.config)。

V 9.1.0.8 使用自解压 JAR 文件安装的客户机可以使用 JMSAdmin 工具在文件系统上下文（.bindings 文件）中创建 JMS 受管对象。 客户机还可以查找并使用这些管理对象。

V 9.1.0.8 以前，自解压 JAR 文件将 com.ibm.mq.allclient.jar 文件及其所有必备 JAR 文件安装到指定安装目录中的 wmq/JavaSE 目录中。 从 IBM MQ 9.1.0 Fix Pack 8 开始，这些文件将安装到 wmq/JavaSE/lib 目录中，与 JMSAdmin 工具相关的文件将安装到 wmq/JavaSE/bin 目录中。

在您运行可执行 JAR 文件时，它显示 IBM MQ 许可协议，必须接受该许可协议。 将要求指定安装 IBM MQ classes for Java、IBM MQ classes for JMS、资源适配器和 OSGi 捆绑软件的目录。 如果选定的安装目录不存在，将创建该目录并且将安装程序文件。 但是，如果该目录存在，将报告错误并且不会安装任何文件。

过程

1. 从 Fix Central 下载 IBM MQ Java / JMS 客户机 JAR 文件。
   a) 单击此链接：IBM MQ Java/JMS 客户机。
   b) 在显示的可用修订列表中查找您的 IBM MQ 版本的客户机。
      例如：

      ```
      release level: 9.1.4.0-IBM-MQ-Install-Java-All
      Continuous Delivery Release:9.1.4 IBM MQ JMS and Java 'All Client'
      ```

      然后，单击客户机文件名并执行下载过程。
2. 从您下载文件的目录启动安装。
   要开始安装，请采用以下格式输入命令：

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

   其中，*V.R.M.F* 是产品版本号，例如 9.1.4.0，*V.R.M.F*-IBM-MQ-Install-Java-All.jar 是从 Fix Central 下载的文件的名称。
   例如，要为 IBM MQ 9.1.4 发行版安装 Java / JMS 客户机，需要使用以下命令：

   ```
   java -jar 9.1.4.0-IBM-MQ-Install-Java-All.jar
   ```

   注: 要执行此安装，您必须在机器上安装 JRE 并将其添加到系统路径。

   在您输入命令时，将显示以下信息：

   ```
   使用、解压缩或安装 IBM MQ V9.1 之前，您必须接受下述协议中的条款：
   1 的条款。 IBM 国际评估许可协议
   程序 2。 IBM 国际程序许可协议和其他
   许可信息。 请仔细阅读以下许可协议。

   使用
   --viewLicenseInfo 选项可以单独查看许可协议。

   按“回车”以立即显示许可条款，或按“x”跳过。
   ```
3. 阅读并且接受许可条款：
   a) 要显示许可，按 Enter 键。
      或者，按 x 跳过显示许可。
      在显示许可之后，或者在您选择 x 之后，将立即显示以下消息：

      ```
      使用
      --viewLicenseInfo 选项可以单独查看其他许可信息。

      按“回车”以立即显示附加许可信息，或按“x”跳过。
      ```
   b) 要显示其他许可条款，请按 Enter 键。
      或者，按 x 跳过显示其他许可条款。
      在显示其他许可条款之后，或者在您选择 x 之后，将立即显示以下消息：

      ```
      通过选择下面的“我同意”选项，您同意
      许可协议条款和非 IBM 条款（如果适用）。 如果不
      选择“我不同意”。

      选择 [1] 我同意，或 [2] 我不同意：
      ```
   c) 要接受许可协议并且继续选择安装目录，请选择 1。
      或者，选择 2 立即结束安装。
      如果您选择 1，将显示以下消息：

      ```
      输入产品文件的目录或保留空白以接受缺省值。
      缺省目标目录为 H:\WMQ

      是否作为产品文件的目标目录？
      ```
4. 指定 Java/JMS 客户机的安装目录：
   - 如果您想要将产品文件安装在缺省位置，那么可在不指定值的情况下直接按 Enter 键。

- 如果想要将产品文件安装在与缺省位置不同的其他位置，请指定要安装产品文件的目录的名称，然后按 Enter 键以开始安装。

您指定的目录名称必须不存在，否则，在您开始安装时，将报告错误并且不会安装任何文件。

假如该目录尚不存在，将创建选定的安装目录，并且会将程序文件安装在此目录中。 在安装期间，会在您选择的安装目录中创建一个名为 wmq 的新目录。 将使用以下内容在 wmq 目录中创建三个子目录 JavaEE、JavaSE 和 OSGi：

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

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

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

其中 *V.R.M.F* 是版本、发行版、修订版和修订包编号。

安装完成后，将显示确认消息，如下示例所示：

```
正在将文件解压缩到 H:\WMQ\wmq
成功解压缩所有产品文件。
```

IBM MQ classes for JMS 中的允许列表

Java 对象序列化和取消序列化机制已经识别为潜在安全风险。 IBM MQ classes for JMS 中的允许列表将针对一些序列化风险提供防护。

Java 对象序列化和取消序列化机制已经识别为潜在安全风险，因为取消序列化将实例化任意 Java 对象，这些对象中可能存在可导致各种问题的恶意发送的数据。 一个值得注意的序列化应用程序位于 Java Message Service (JMS) ObjectMessages，它使用序列化封装和传输任意对象。

序列化允许列表可能会降低序列化造成的某些风险。 通过明确指定哪些类可以封装到 ObjectMessage 或可以从中抽取哪些类，允许列表可防范部分序列化风险。

IBM MQ classes for JMS 中的允许列表

请参阅：

- 第 102 页的『允许列表概念』，以获取允许列表的概述
- 第 105 页的『设置和使用 JMS 允许列表』，以获取有关如何设置允许列表的信息
- 第 107 页的『WebSphere Application Server 中的允许列表』，以获取有关如何在 WebSphere Application Server 中设置允许列表的信息。

相关概念

第 84 页的『在 Java security manager 下运行 IBM MQ classes for JMS 应用程序』
IBM MQ classes for JMS 可以在 Java 安全管理器已启用的情况下运行。 要在 Java security manager 已启用的情况下成功运行应用程序，您必须使用合适的策略配置文件来配置 Java virtual machine (JVM)。

允许列表概念

在 IBM MQ classes for JMS 中，实现 JMS ObjectMessage 接口时支持将类列入允许列表可能降低与 Java 对象序列化和取消序列化机制潜在相关的一些安全风险。

IBM MQ classes for JMS 中的允许列表

要点：

在可能的情况下，术语 *allowlist*（白名单）已替换术语 *whitelist*（白名单）。 对于 IBM MQ 9.0 和更高发行版，这包括在本主题中提及的 Java 系统属性名称 (**com.ibm.mq.jms.***)。您无需更改任何现有配置。 先前系统属性名称也继续有效。

IBM MQ classes for JMS 在 JMS ObjectMessage 接口的实现中支持将类列入允许列表。

允许列表定义哪些 Java 类可以使用 ObjectMessage.setObject() 序列化以及哪些可以使用 ObjectMessage.getObject() 取消序列化。

尝试使用 ObjectMessage 序列化或取消序列化不包含在允许列表中的类实例将导致抛出 javax.jms.MessageFormatException，其原因为 java.io.InvalidClassException。

生成允许列表

要点: IBM MQ classes for JMS 无法使用允许列表分发。 使用 ObjectMessages 传输哪些类是由应用程序设计进行选择，IBM MQ 无法抢占此功能。

鉴于此，列入允许列表机制允许两种操作模式：

发现
在此模式下，机制生成标准类名称的列表，报告观察到的在 ObjectMessages 中已序列化或取消序列化的所有类。

强制
在此模式下，机制强制实施允许列表功能，拒绝序列化或取消序列化不在允许列表中的类的尝试。

如果想使用此机制，您最初必须以“发现”模式运行，以收集目前序列化和取消序列化的类的列表，查看列表，并将它作为允许列表的基础。 甚至可以不经过更改使用列表，但是决定这样做之前必须首先查看列表。

控制允许列表机制

三个系统属性可用于控制允许列表机制：

com.ibm.mq.jms.allowlist
可以通过以下任一方法指定此属性：

- 包含允许列表的文件的路径名，其采用文件 URI 格式（即，以 `file:` 开头）。 在“发现”模式下，此文件由允许列表机制写入。 文件不得存在。 如果文件确实存在，机制将抛出异常，而不是覆盖它。 在“强制执行”模式下，此文件由允许列表机制读取。
- 构成允许列表的以逗号分隔的标准类名称。

如果未设置此属性，允许列表机制是非活动的。

如果您使用的是 Java security manager，您必须确保 IBM MQ classes for JMS JAR 文件具有此文件的读写访问权。

com.ibm.mq.jms.allowlist.discover

- 如果未设置此属性，或者已设置为 false，允许列表机制在“强制执行”模式下运行。
- 如果此属性被设成 true，并且允许列表已指定为文件 URI，允许列表机制以“发现”模式运行。
- 如果此属性被设成 true，并且允许列表已指定为类名列表，允许列表机制将抛出相应的异常。
- 如果属性设置为 true，并且允许列表没有使用 com.ibm.mq.jms.allowlist property 指定，那么允许列表机制处于不活动状态。
- 如果此属性设为 true 并且允许列表文件已存在，那么允许列表机制将抛出 java.io.InvalidClassException，并且不会将条目添加到文件。

com.ibm.mq.jms.allowlist.mode
可以采用以下三种方式之一指定此字符串属性：

- 如果将此属性设置为 SERIALIZE，那么 ENFORCEMENT 方式将仅在 ObjectMessage.setObject() 方法上执行允许列表验证。
- 如果将此属性设置为 DESERIALIZE，那么 ENFORCEMENT 方式将仅在 ObjectMessage.getObject() 方法上执行允许列表验证。

- 如果未设置此属性或者设置为任何其他值，那么 ENFORCEMENT 方式将在 ObjectMessage.getObject() 和 ObjectMessage.setObject() 方法上执行允许列表验证。

允许列表文件的格式

以下是允许列表文件格式的主要特性：

- 允许列表文件采用缺省平台文件编码（带有适用于平台的行结尾符）。

  注： 如果正在使用允许列表文件，那么将始终使用针对 JVM 的缺省文件编码来编写和读取此文件。

  如果采用以下任何方式生成允许列表文件，那么没问题：

  – z/OS 由运行在 z/OS 上的独立应用程序生成，并且由同样运行在 z/OS 上的其他独立应用程序使用。
  – 由在任何平台上的 WebSphere Application Server 内运行的应用程序生成，并且由另一个 WebSphere Application Server 实例使用。
  – Multi 由运行在 IBM MQ for Multiplatforms 上的独立应用程序生成，并且由运行在 IBM MQ for Multiplatforms 上的其他独立应用程序使用，或者由运行在任何平台上的 WebSphere Application Server 内的应用程序使用。

  但是，由于 WebSphere Application Server 使用 ASCII，而独立 JVM 使用 EBCDIC，因此如果采用以下任一方式生成允许列表文件，将出现文件编码问题：

  – 在 z/OS 上生成，然后由运行在 z/OS 以外的平台上的独立应用程序使用或者由 WebSphere Application Server 使用。
  – 由 WebSphere Application Server 生成或者由运行在 z/OS 以外的平台上的独立应用程序生成，然后由 z/OS 上的独立应用程序使用。
- 每一个非空行包含标准类名称。 空行将被忽略。
- 可以包含注释，将忽略“#”字符后一直到行尾的任何内容。
- 有非常基本的通配符机制：
  – “*”可以是类名的**最后**元素。
  – “*”匹配类名的**单个**元素，也就是类，但不匹配软件包的部分。

  因此，`com.ibm.mq.*` 将与 `com.ibm.mq.MQMessage` 匹配，但与 `com.ibm.mq.jmqi.remote.api.RemoteFAP` 不匹配。

  通配符不适用于缺省软件包中的类，缺省软件包是针对没有明确软件包名称的类，因此，拒绝类名“*”。
- 格式不正确的允许列表文件（例如，包含 `com.ibm.mq.*.Message` 之类的条目的文件，其中通配符不是最后一个元素）导致抛出了 java.lang.IllegalArgumentException。
- 空的允许列表文件具有完全禁用 ObjectMessage 的效果。

逗号分隔列表的允许列表的格式

同样的通配符机制可用于逗号分隔列表形式的允许列表。

- 如果在命令行或 shell 脚本或批处理文件中指定“*”，那么操作系统可以扩展 *，因此可能需要特殊处理。
- 仅在指定文件时，“#”注释字符适用。 如果允许列表指定为类名的逗号分隔列表，并假定操作系统或 shell 不进行处理，因为它是许多 UNIX 或 Linux shell 中的缺省注释字符，那么将它视为正常字符。

什么时候启动允许列表？

当应用程序首次运行 ObjectMessage setMessage() 或 getMessage() 方法时，启动允许列表。

对系统属性求值，允许列表文件已打开并处于“强制实施”模式，初始化机制时载入列入允许列表的类的列表时。 此时，条目写入应用程序的 IBM MQ JMS 日志文件。

在初始化机制时，可能不更改其参数。 初始化的时间不容易预测，因为这依赖于应用程序行为。 因此，从启动应用程序开始，应将系统属性设置和允许列表文件内容视为固定内容。 不要在应用程序运行时更改允许列表文件的属性或内容，因为结果不确定。

要考虑的要点

降低 Java 序列化机制内在风险的最佳方法是探索数据传输的替代方法，例如使用 JSON 代替 ObjectMessage。 使用 Advanced Message Security (AMS) 机制可以通过确保消息来自可信源，来进一步增加安全性。

如果您的应用程序使用 Java security manager 机制，您必须授予以下权限：

- 您使用的任何允许列表文件上的 FilePermission，其中读许可权用于 ENFORCEMENT 方式，写许可权用于 DISCOVER 方式。
- **com.ibm.mq.jms.allowlist**、**com.ibm.mq.jms.allowlist.discover** 和 **com.ibm.mq.jms.allowlist.mode** 属性上的 PropertyPermission（读）。

详细信息

请参阅第 105 页的『设置和使用 JMS 允许列表』和第 107 页的『WebSphere Application Server 中的允许列表』，以获取有关允许列表的更多信息。

相关概念

第 84 页的『在 Java security manager 下运行 IBM MQ classes for JMS 应用程序』

IBM MQ classes for JMS 可以在 Java 安全管理器已启用的情况下运行。 要在 Java security manager 已启用的情况下成功运行应用程序，您必须使用合适的策略配置文件来配置 Java virtual machine (JVM)。

设置和使用 *JMS* 允许列表

以下信息介绍了允许列表的工作方式，以及如何使用 IBM MQ classes for JMS 中提供的功能来生成允许列表文件（包含应用程序可处理的 ObjectMessage 类型的列表）以设置允许列表。

开始之前

要点:

在可能的情况下，术语 *allowlist*（白名单）已替换术语 *whitelist*（白名单）。 对于 IBM MQ 9.0 和更高发行版，这包括在本主题中提及的 Java 系统属性名称 (**com.ibm.mq.jms.***)。您无需更改任何现有配置。 先前系统属性名称也继续有效。

在开始此任务之前，请确保您已阅读并理解第 102 页的『允许列表概念』

关于此任务

在启用允许列表功能后，IBM MQ classes for JMS 可通过以下方式来使用此功能：

- 当应用程序想要发送 ObjectMessage 时，可通过以下两种方式之一来创建 ObjectMessage：
  - 调用 Session.createObjectMessage(Serializable) 方法，并传递要包含在消息中的对象。
  - 调用 Session.createObjectMessage() 方法来创建空的 ObjectMessage，然后调用 ObjectMessage.setObject(Serializable) 来存储要在 ObjectMessage 中发送的对象。

  在调用 Session.createObjectMessage(Serializable) 或 ObjectMessage.setObject(Serializable) 方法时，用于 JMS 的类将检查所传递的对象是否为允许列表中指定的类型。

  如果该对象是白名单中指定的类型，那么将序列化该对象并将其存储在 ObjectMessage 中。 但是，如果该对象不是允许列表中指定的类型，那么 IBM MQ classes for JMS 将抛出包含以下消息的 JMSException：

  ```
  JMSCC0052: 序列化对象时发生异常：
  'java.io.InvalidClassException: <object class>; The class may not be serialized
  or deserialized as it has not been included in the allowlist '<allowlist>'.
  ```

  返回到该应用程序。

要点: 如果是从 Session.createObjectMessage(Serializable) 方法中抛出异常，那么将不会创建 ObjectMessage。 同样，如果是从 ObjectMessage.setObject(Serializable) 方法中抛出 JMSException，那么不会将该对象添加到 ObjectMessage。

- 如果应用程序收到一个 ObjectMessage，那么它将调用 ObjectMessage.getObject() 方法来获取该 ObjectMessage 中包含的对象。 调用此方法时，IBM MQ classes for JMS 会检查 ObjectMessage 中所含对象的类型，以查看该对象是否为允许列表中指定的类型。

  如果该对象是白名单中指定的类型，那么将反序列化该对象并将其返回到该应用程序。 但是，如果该对象不是允许列表中指定的类型，那么 IBM MQ classes for JMS 将抛出包含以下消息的 JMSException：

  ```
  JMSCC0053: 反序列化消息时发生异常：
  'java.io.InvalidClassException: <object class>; The class may not be
  已序列化或反序列化，因为它尚未包含在
  allowlist '<allowlist>'.'.
  ```

  返回到该应用程序。

例如，假定您的应用程序包含以下代码，以用于发送包含类型为 java.net.URI 的对象的 ObjectMessage：

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

由于未启用允许列表功能，因此该应用程序可以成功地将消息放到所需的目标中。

如果您创建了名为 `C:\allowlist.txt` 且包含单个条目 `java.net.URL` 的文件，并通过设置该 Java 系统属性来重新启动该应用程序：

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

将启用允许列表功能。 应用程序仍然可以创建和发送包含类型为 java.net.URI 的对象的 ObjectMessage，因为在允许列表中指定了此类型。

但是，如果更改了 `allowlist.txt` 文件以使该文件包含单个条目 `java.util.Calendar`，由于允许列表功能仍处于启用状态，因此在该应用程序执行以下调用时：

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS 会检查允许列表并发现它不包含 java.net.URI 条目。

因此，将抛出包含 JMSCC0052 消息的 JMSException。

同样，假定您有另一个应用程序，并且此应用程序通过以下代码来接收 ObjectMessage：

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

如果未启用允许列表功能，那么该应用程序可以接收包含任何类型对象的 ObjectMessage。 然后，该应用程序在执行相应的处理之前会检查该对象是否为 java.net.URL 类型。

如果现在启动该应用程序且已设置有以下 Java 系统属性：

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

那么将开启允许列表功能。 在该应用程序执行以下调用时：

```
Object messageBody = objectMessage.getObject();
```

ObjectMessage.getObject() 方法将仅返回类型为 java.net.URL 的对象。

如果 ObjectMessage 中包含的对象不是此类型，那么 ObjectMessage.getObject() 方法将抛出包含 JMSCC0053 消息的 JMSException。 然后，该应用程序需要确定如何处理该消息；例如，可以将该消息移至此队列管理器的死信队列中。

仅当 ObjectMessage 中的对象为 java.net.URL 类型时，该应用程序才会正常返回。

过程

1. 在指定了以下 Java 系统属性后，运行可处理 ObjectMessage 的应用程序：

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   当该应用程序运行时，IBM MQ classes for JMS 会创建一个文件来包含该应用程序已处理的对象类型。
2. 在该应用程序处理了具有代表性的 ObjectMessage 样本一段时间后，停止该应用程序。

   允许列表文件现在包含一个列表，其中列出了该应用程序在运行期间处理过且包含在 ObjectMessage 中的所有对象类型。

   如果运行该应用程序的时间足够长，那么此列表将包含该应用程序可能会处理且包含在 ObjectMessage 中的所有可能的对象类型。
3. 在设置了以下系统属性后，重新启动该应用程序：

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   这样会启用允许列表功能，并且如果 IBM MQ classes for JMS 检测到某个 ObjectMessage 的类型不在允许列表中，那么将抛出包含 JMSCC0052 或 JMSCC0053 消息的 JMSException。

WebSphere Application Server 中的允许列表

如何在 WebSphere Application Server 中使用 IBM MQ classes for JMS 允许列表。

要点:

在可能的情况下，术语 *allowlist*（白名单）已替换术语 *whitelist*（白名单）。 对于 IBM MQ 9.0 和更高发行版，这包括在本主题中提及的 Java 系统属性名称 (**com.ibm.mq.jms.***)。您无需更改任何现有配置。 先前系统属性名称也继续有效。

您必须确保 WebSphere Application Server 安装包含支持允许列表的 IBM MQ 资源适配器版本。

请参阅第 413 页的『将 IBM MQ 与 WebSphere Application Server 结合使用』，以获取有关使用这两个产品的更多信息。

IBM MQ 9.0.0 Fix Pack 1 及更高版本均包含相应的功能。

更新应用程序服务器后，即可使用以下 Java 系统属性：

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

如第 105 页的『设置和使用 JMS 允许列表』中所述。

注: 您需要在用于运行应用程序服务器的 Java virtual machine 上将 Java 系统属性设置为通用 JVM 参数，然后重新启动应用程序服务器以使更改生效。

有关更多信息，请参阅 Java 虚拟机设置中的通用 *JVM* 参数部分。

要设置这些属性，请转至“进程定义”中的 Java virtual machine 窗口，并输入相应的参数。

以下设置：

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

会使应用程序服务器使用允许列表 *youruserId_MyObject*。 应用程序服务器将仅处理此类型的对象。

以下设置：

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

将应用程序服务器配置为使用发现方式，并将应用程序服务器处理的 JMS ObjectMessage 的详细信息记录到文件 `C:\allowlist.txt` 中

以下设置:

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

导致应用程序服务器装入文件 C:/allowlist.txt，并使用该文件中的信息来确定允许列表。

相关概念

第 84 页的『在 Java security manager 下运行 IBM MQ classes for JMS 应用程序』

IBM MQ classes for JMS 可以在 Java 安全管理器已启用的情况下运行。 要在 Java security manager 已启用的情况下成功运行应用程序，您必须使用合适的策略配置文件来配置 Java virtual machine (JVM)。

IBM MQ classes for JMS 中的字符串转换

IBM MQ classes for JMS 直接使用 CharsetEncoders 和 CharsetDecoders 进行字符串转换。 可以使用两个系统属性配置字符串转换的缺省行为。 包含无法映射字符的消息处理可以通过消息属性配置，消息属性用于设置 UnmappableCharacterAction 和替换字节

在 IBM MQ 8.0 之前， IBM MQ classes for JMS 中的字符串转换是通过调用 java.nio.charset.Charset.decode(ByteBuffer) 和 Charset.encode(CharBuffer) 方法完成的。

使用其中任一方法都会导致对格式不正确或不可转换的数据进行缺省替换 (REPLACE)。 该行为可能会隐藏应用程序中的错误，并且导致已转换数据中产生意外的字符，例如 ?。

在 IBM MQ 8.0 中，为了更早更有效地检测此类问题，IBM MQ classes for JMS 将直接使用 CharsetEncoder 和 CharsetDecoder 并明确配置对格式错误和不可转换数据的处理。 REPORT 的缺省行为是通过抛出适当的 MQException 来报告此类问题。

配置

从 UTF-16 (Java 中使用的字符表示) 到本机字符集 (例如 UTF-8) 的转换称为 encoding，而相反方向的转换称为 decoding。

当前，解码会采用 CharsetDecoders 的缺省行为，通过抛出异常来报告错误。

一个设置用于指定 java.nio.charset.CodingErrorAction，以控制编码和解码时的错误处理。 另一个设置用于在编码时控制一个或多个替换字节。 将在解码操作中使用缺省的 Java 替换字符串。

UnmappableCharacter IBM MQ Classes for JMS 中的操作和替换字节设置

从 IBM MQ 8.0 开始，还有两个属性可用于设置 UnmappableCharacter 操作和替换字节。 相应的常量定义位于 com.ibm.msg.client.wmq.WMQConstants 中

JMS_IBM_UNMAPPABLE_ACTION

设置或获取无法在编码或解码操作中映射字符时要应用的 CodingErrorAction。

您应将此项设置为 CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()，如下所示:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT

设置或获取无法在编码操作中映射字符时要应用的替换字节。

将在解码操作中使用缺省的 Java 替换字符串。

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

可以在目标或消息上设置 JMS_IBM_UNMAPPABLE_ACTION 和 JMS_IBM_UNMAPPABLE_REPLACEMENT 属性。 在消息上设置的值将覆盖在消息所发送至的目标上设置的值。

请注意，必须将 JMS_IBM_UNMAPPABLE_REPLACEMENT 设置为单字节。

用于设置系统缺省值的系统属性

从 IBM MQ 8.0 开始，提供了以下两个 Java 系统属性，用于配置关于字符串转换的缺省行为。

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
指定编码和解码时要对不可转换的数据采取的操作。 值可以是 REPORT、REPLACE 或 IGNORE。

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
设置或获取无法在解码操作中映射字符时要应用的替换字节。将在解码操作中使用缺省的 Java 替换字符串。

为避免 Java 字符与本机字节表示之间混淆，您应将 `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` 指定为表示本机字符集中替换字节的十进制数。

例如，如果本机字符集基于 ASCII（如 ISO-8859-1），那么本机字节 ? 的十进制值为 63，而如果本机字符集为 EBCDIC，它仍为 111。

注: 请注意，如果 MQMD 或 MQMessage 对象设置了 **unmappableAction** 或 **unMappableReplacement** 字段，那么这些字段的值将优先于 Java 系统属性。 这将允许覆盖由 Java 系统属性指定的每条消息的值（如果需要）。

相关概念
第 284 页的『IBM MQ classes for Java 中的字符串转换』
IBM MQ classes for Java 直接使用 CharsetEncoders 和 CharsetDecoders 进行字符串转换。 可以使用两个系统属性配置字符串转换的缺省行为。 可以通过 `com.ibm.mq.MQMD` 配置包含不可映射字符的消息的处理。

编写 IBM MQ classes for JMS 应用程序

继 JMS 模型简介后，此主题提供了有关如何编写 IBM MQ classes for JMS 应用程序的详细指南。

JMS 模型

JMS 模型定义了一组接口， Java 应用程序可以使用这些接口来执行消息传递操作。 IBM MQ classes for JMS 作为 JMS 提供程序，定义 JMS 对象如何与 IBM MQ 概念相关。 JMS 规范认为某些 JMS 对象是受管对象。 JMS 2.0 引入了简化的 API，同时保留了 JMS 1.1 中的标准 API。

JMS 规范和 javax.jms 包一起定义了一组可供 Java 应用程序用来执行消息传递操作的接口。

从 IBM MQ 8.0 开始，IBM MQ 支持 JMS 2.0 版本的 JMS 标准，该标准引入了简化的 API，同时保留了 JMS 1.1 中的标准 API。

简化的 API

JMS 2.0 引入了简化的 API，同时保留了 JMS 1.1 中特定于域的接口和独立于域的接口。 简化的 API 减少了发送和接收消息所需的对象数，由以下接口构成：

ConnectionFactory
ConnectionFactory 是供 JMS 客户机用来创建连接的受管对象。 此接口也用于标准 API。

JMS 上下文
此对象结合了标准 API 的 Connection 和 Session 对象。 可以通过复制底层连接来从其他 JMSContext 对象创建 JMSContext 对象。

JMS 生产者
JMSProducer 由 JMSContext 创建，并用于向队列或主题发送消息。 JMSProducer 对象会导致创建发送消息所需的对象。

JMS 使用者
JMSConsumer 由 JMSContext 创建，并用于从主题或队列接收消息。

简化的 API 具有以下影响：

- JMSContext 对象始终自动启动底层连接。

- JMSProducer 和 JMSConsumer 现在可以直接处理消息体，而无需使用消息的 getBody 方法来获取整个消息对象。
- 在发送“消息体”（即消息内容）之前，可以使用方法链在 JMSProducer 对象上设置消息属性。 JMSProducer 将负责创建发送消息所需的所有对象。 通过使用 JMS 2.0，可以按如下所示设置属性和发送消息：

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

JMS 2.0 还引入了可用于在多个使用者之间共享消息的共享预订。 所有 JMS 1.1 预订均视为非共享预订。

标准 API

以下列表汇总了标准 API 的主要 JMS 接口：

Destination
目标是应用程序发送消息的位置和/或应用程序从中接收消息的源。

ConnectionFactory
ConnectionFactory 对象封装了连接的一组配置属性。 应用程序使用连接工厂来创建连接。

Connection
Connection 对象将应用程序的活动连接封装到消息传递服务器中。 应用程序使用连接来创建会话。

Session
会话是用于发送和接收消息的单线程上下文。 应用程序使用会话来创建消息、消息生产者和消息使用者。 会话将进行事务处理或不进行事务处理。

消息
Message 对象封装了应用程序发送或接收的消息。

MessageProducer
应用程序使用消息生产者向目标发送消息。

MessageConsumer
应用程序使用消息使用者接收向目标发送的消息。

第 110 页的图 9 显示了这些对象及其关系。

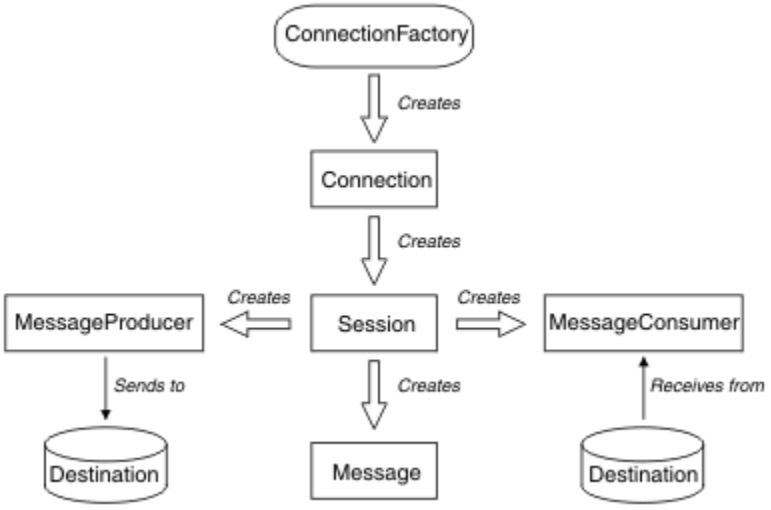


图 *9: JMS* 对象及其关系

Destination、ConnectionFactory 或 Connection 对象可以由多线程应用程序的不同线程并行使用，但 Session、MessageProducer 或 MessageConsumer 对象不能由不同线程并行使用。 确保 Session、

MessageProducer 或 MessageConsumer 对象不会被并行使用的最简单方法是为每个线程创建单独的 Session 对象。

JMS 支持以下两种类型的消息传递：

- 点到点消息传递
- 发布/预订消息传递

这两种类型的消息传递也称为消息传递域，您可以在应用程序中结合使用这两种类型的消息传递。 在点到点域中，目标是队列，在发布/预订域中，目标是主题。

对于 JMS 之前的 JMS 1.1 版本，点到点域的编程使用的是一组接口和方法，而发布/预订域的编程则使用的是另一组接口和方法。 二者相似，但相互独立。 从 JMS 1.1 开始，您可以使用一组同时支持这两种消息传递域的通用接口和方法。 通用接口为每个消息传递域提供独立于域的视图。 第 111 页的表 14 列出了独立于 JMS 域的接口及对应的域特定接口。

表 *14:* 独立于 *JMS* 域的接口和特定于域的接口

独立于域的接口	**点到点域的域特定接口**	**发布/预订域的域特定接口**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	队列	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 保留了所有特定于域的接口，因此现有应用程序仍可以使用这些接口。 但对于新的应用程序，请考虑使用 JMS 1.1 中独立于域的接口或 JMS 2.0 中的简化 API。

在 IBM MQ classes for JMS 中，JMS 对象在以下方面与 IBM MQ 概念相关：

- Connection 对象具有派生自连接工厂（用于创建连接）属性的属性。 这些属性控制应用程序如何连接到队列管理器。 这些属性的示例包括队列管理器名称以及运行队列管理器的系统的主机名或 IP 地址（针对以 CLIENT 方式连接到队列管理器的应用程序）。
- Session 对象封装了 IBM MQ 连接句柄，因此定义了会话的事务范围。
- MessageProducer 对象和 MessageConsumer 对象各自封装了一个 IBM MQ 对象句柄。

在使用 IBM MQ classes for JMS 时，IBM MQ 的所有标准规则都适用。 尤其需要注意，应用程序可以向远程队列发送消息，但只能从应用程序所连接到的队列管理器拥有的队列中接收消息。

JMS 规范认为 ConnectionFactory 和 Destination 对象都是受管对象。 管理员在中央存储库中创建和维护受管对象，而 JMS 应用程序可使用 Java 命名和目录接口 (JNDI) 来检索这些对象。

在 IBM MQ classes for JMS 中，Destination 接口的实现是 Queue 和 Topic 的抽象超类，因此 Destination 实例是 Queue 对象或 Topic 对象。 独立于域的接口将队列或主题视为目标。 MessageProducer 或 MessageConsumer 对象的消息传递域由目标是队列还是主题来确定。

因此，在 IBM MQ classes for JMS 中，以下类型的对象可以是受管对象：

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- 队列
- Topic
- XAConnectionFactory

- XAQueueConnectionFactory
- XATopicConnectionFactory

相关概念

IBM MQ Java 语言接口

第 262 页的『使用 JMS 2.0 功能』

JMS 2.0 将多个新功能领域引入了 IBM MQ classes for JMS。

JMS 消息

JMS 消息由头、属性以及主体组成。 JMS 定义了五类消息体。

JMS 消息包含以下部分：

头
所有消息均支持同一组头字段。 头字段包含可由客户和提供商识别和路由消息所用的值。

属性
每条消息均包含一项内置功能，来支持应用程序定义的属性值。 属性提供一种有效机制来过滤应用程序定义的消息。

正文
JMS 定义了五类消息体，涉及当前使用的大多数消息传递样式：

流
Java 原语值流。 会按顺序对其进行填充和读取。

映射
一组名称/值对，其中名称是字符串，值是 Java 原语类型。 可按名称顺序或随机地访问条目。 条目顺序未定义。

文本
包含 java.lang.String 的消息。

Object
包含可序列化的 Java 对象的消息

字节
未解释字节流。 此消息类型用于在字面上对主体进行编码，以与现有消息格式匹配。

JMSCorrelationID 头字段用于将一条消息与另一条消息相链接。 通常，它会将应答消息与其请求消息相链接。 JMSCorrelationID 可保留特定于提供者的消息标识、特定于应用程序的字符串或 provider-native byte[] 值。

JMS 中的消息选择器

消息可包含应用程序定义的属性值。 应用程序可使用消息选择器来让 JMS 提供程序过滤消息。

消息包含一项内置功能，来支持应用程序定义的属性值。 实际上，这提供了一种机制，来将特定于应用程序的头字段添加到消息中。 利用消息选择器及特定于应用程序的标准，属性支持应用程序以其名义让 JMS 提供程序选择或过滤消息。 应用程序定义的属性必须遵循以下规则：

- 属性名称必须遵循消息选择器标识规则。
- 属性值可以是 Boolean、byte、short、int、long、float、double 和 String。
- 将保留 JMSX 和 JMS_ name 前缀。

发送消息前，将设置属性值。 客户机接收消息时，消息属性为只读属性。 如果客户机尝试在此时设置属性，将抛出 MessageNotWriteableException。 如果调用 clearProperties，那么现在可以从其读取属性或向其写入属性。

属性值可在消息体中复制值。 JMS 未对可能构成属性的内容定义策略。 但是，应用程序开发人员必须清楚，与消息属性中的数据相比，JMS 提供程序也许能够更有效地处理消息体中的数据。 要获得最佳性能，应用程序必须只能在其需要定制消息头时使用消息属性。 这样做的主要原因是支持定制消息选择。

JMS 消息选择器允许客户机使用消息头指定其感兴趣的消息。 仅传送其头与选择器匹配的消息。

消息选择器无法参考消息体值。

在选择器中将消息头字段和属性值替换为其对应标识时，如果选择器求值结果为 true，那么表示消息选择器将与消息匹配。

消息选择器为字符串，其语法基于 SQL92 条件表达式语法的一个子集。 消息选择器的求值顺序为同一优先顺序级别内从左到右。 可使用括号来更改这一顺序。 预定义选择器字面值和运算符名称在此处以大写写入；但是，这些文本和运算符不区分大小写。

消息选择器的内容

消息选择器可以包含：

- 文字
  - 字符串文字引在引号中。 双引号表示引号。 例如，'literal' 和 'literal''s'。 像 Java 字符串字面值一样，这些字面值使用 Unicode 字符编码。
  - 精确的数字文字是一个不含小数点的数字值，如 57、-957 或 +62。 支持 Java long 范围内的数字。
  - 近似的数字文字是一个采用科学记数法的数字值（如 7E3 或 -57.9E2）或带小数的数字值（如 7.、-95.7 或 +6.2）。 支持 Java double 范围内的数字。
  - Boolean 文字 TRUE 和 FALSE。
- 标识符：
  - 标识是一系列长度无限的 Java 字母和 Java 数字，其中第一位必须是 Java 字母。 字母是 Character.isJavaLetter 方法针对其返回 true 的任何字符。 这包括 _ 和 $。字母或数字是方法 Character.isJavaLetterOrDigit 返回 true 的任何字符。
  - 标识不能是 NULL、TRUE 或 FALSE 等名称。
  - 标识不能是 NOT、AND、OR、BETWEEN、LIKE、IN 或 IS。
  - 标识为头字段引用或属性引用。
  - 标识区分大小写。
  - 消息头字段引用限于：
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、JMSCorrelationID 和 JMSType 值可以为空，此时会将其视为 NULL 值。
  - 以 JMSX 开头的任何名称均为 JMS 定义的属性名称。
  - 以 JMS_ 开头的任何名称均为特定于提供程序的属性名称。
  - 不以 JMS 开头的任何名称均为特定于应用程序的属性名称。 如果存在对消息中不存在的属性的引用，那么其值为 NULL。 如果确实存在，那么其值将为相应的属性值。
- 空格由于是为 Java 定义的，因此都相同：空格、水平制表符、换页以及行终止符。
- 表达式：
  - 选择器是条件表达式。 求值结果为 true 的选择器匹配；结果为 false 或未知值的选择器不匹配。
  - 算术表达式由其自身、算术运算符、标识（其值被视为数字文字）以及数字文字组成。
  - 条件表达式由其自身、比较运算符以及逻辑运算符组成。
- 支持用于设置表达式求值顺序的标准括号 ()。
- 采用优先顺序的逻辑运算符：NOT、AND 和 OR。
- 比较运算符：=、>、>=、<、<=、<>（不等于）。

  - 只能比较相同类型的值。 一种例外情况是，可比较精确的数字值与近似的数字值。 （所需的类型转换由 Java 数字提升规则来定义。） 如果尝试比较不同的类型，那么选择器将始终为 false。
  - 字符串和布尔值的比较仅限于 = 和 <>。两个字符串只有在包含相同字符序列时才相等。
- 按照优先顺序排列的算术运算符：
  - +、- 一元。
  - *、/，乘和除。
  - +、-，加和减。
  - 不支持对 NULL 值使用算术运算。 如果尝试使用，那么整个选择器均始终为 false。
  - 算术运算符必须使用 Java 数字提升。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 和 arithmetic-expr3 比较运算符：
  - Age BETWEEN 15 and 19 相当于 age >= 15 AND age <= 19。
  - Age NOT BETWEEN 15 and 19 相当于 age < 15 OR age> 19。
  - 如果任何 BETWEEN 操作的表达式为 NULL，那么运算值为 false。 如果任何 NOT BETWEEN 操作的表达式为 NULL，那么运算值为 true。
- identifier [NOT] IN (string-literal1, string-literal2,...) 是标识值为 String 或 NULL 的比较运算符。
  - Country IN ('UK', 'US', 'France') 对于“UK”为 true，对于“Preu”为 false。 它相当于表达式 (Country = 'UK') OR (Country = 'US') OR (Country = 'France')。
  - Country NOT IN ('UK', 'US', 'France') 对于“UK”为 false，对于“Preu”为 true。 它相当于表达式 NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))。
  - 如果 IN 或 NOT IN 运算标识为 NULL，那么运算值未知。
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] 是标识具有字符串值的比较运算符。 pattern-value 是字符串文字，其中 _ 代表任何单个字符，% 代表任何序列的字符（包括空序列）。 所有其他字符均代表其自身。 可选 escape-character 是单个字符串文字，具有用于对 pattern-value 中的 _ 和 % 的特殊含义进行转义的字符。
  - phone LIKE '12%3' 对于 123 和 12993 为 true，对于 1234 为 false。
  - word LIKE 'l_se' 对于“lose”为 true，对于“loose”为 false。
  - underscored LIKE '_%' ESCAPE '\' 对于“_foo”为 true，对于“bar”为 false。
  - phone NOT LIKE '12%3' 对于 123 和 12993 为 false，对于 1234 为 true。
  - 如果 LIKE 或 NOT LIKE 运算的标识为 NULL，那么该运算的值未知。
- 标识 IS NULL 比较运算符测试是否存在空头字段值或缺少属性值。
  - prop_name IS NULL。
- 标识 IS NOT NULL 比较运算符测试是否存在非空头字段值或属性值。
  - prop_name IS NOT NULL。

消息选择器的示例

以下消息选择器选择消息类型为汽车，颜色为蓝色且重量大于 2500 磅的消息：

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

NULL 属性值

如以上列表所示，属性值可以为 NULL。 包含 NULL 值的选择器表达式求值由 SQL 92 NULL 语义来定义。 以下列表给出了关于这些语义的简述：

- SQL 将 NULL 值视为“未知”。
- 带有 unknown 值的比较或算术始终会产生 unknown 值。

- IS NULL 运算符将未知值转换为 TRUE 值。
- IS NOT NULL 运算符将未知值转换为 FALSE 值。

JMSMessageID 和 JMSCorrelationID 的特殊行为

从基于 JMSMessageID 或 JMSCorrelationID 的队列选择消息时，JMS 的 IBM MQ 类包含优化。

如果应用程序指定以下格式的选择器：

JMSMessageID='ID:*message_id*'

其中，*message_id* 是包含标准 IBM MQ 消息标识的字符串，那么 JMS 的 IBM MQ 类使用 **MatchOption** MQMO_MATCH_MSG_ID 以获取具有指定的消息标识的消息。

例如，要从队列获取消息标识为 414D51207061756C745639314C545320C57C1A5F25ECE602 的消息，应用程序应使用以下消息选择器：

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

类似，如果应用程序指定具有以下格式的选择器：

JMSCorrelationID ='ID:*correlation_id*'

其中，*correlation_id* 是包含标准 IBM MQ 相关标识的字符串，那么 JMS 的 IBM MQ 类使用 **MatchOption** MQMO_MATCH_CORREL_ID 以从队列获取具有指定相关标识的消息。

在以下示例中，消息选择器用于获取具有相关标识 414D51207061756C745639314C545320846E5B5F25B1CC02 的消息：

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

如果消息选择器包含 *message_id* 或 *correlation_id* 全都为零的值，那么将与队列上的任何消息相匹配。 例如，如果应用程序使用选择器：

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

那么，队列上的任何消息都将被视为匹配项，并返回至应用程序。

有关 MQMO_MATCH_MSG_ID 和 MQMO_MATCH_CORREL_ID **MatchOptions** 的更多信息，请参阅 MatchOptions (MQLONG)。

限制

尽管 SQL 支持固定的小数比较和算术，但 JMS 消息选择器并不支持。 这是因为精确数字字面值限制为不带小数点的数字。 还因为某些带有小数点的数字作为近似数字值的替代表示。

不支持 SQL 注释。

将 *JMS* 消息映射到 *IBM MQ* 消息
IBM MQ 消息由消息描述符、可选 MQRFH2 头以及主体组成。 JMS 消息内容将部分映射，部分被复制到 IBM MQ 消息。

此主题描述了如何将本节第一部分中所述的 JMS 消息结构映射到 IBM MQ 消息。 那些希望在 JMS 与传统 IBM MQ 应用程序间传输消息的程序员对此深感兴趣。 那些希望控制在两个 JMS 应用程序间传输的消息（例如，在 IBM Integration Bus 实现中）的人员也对此兴趣盎然。

如果应用程序使用代理程序的实时连接，那么本部分内容不适用。 应用程序使用实时连接时，系统将直接通过 TCP/IP 执行所有通信；不涉及任何 IBM MQ 队列或消息。

IBM MQ 消息由三个组件组成：

- IBM MQ 消息描述符 (MQMD)
- IBM MQ MQRFH2 头
- 消息体。

MQRFH2 是可选的，其包含在外发消息中由 JMS Destination 类中的 TARGCLIENT 标志控制。 可使用 IBM MQ JMS 管理工具来设置此标志。 由于 MQRFH2 带有特定于 JMS 的信息，因此当发送方知道接收目标为 JMS 应用程序时，应始终在消息中包含 MQRFH2。 通常，在将消息直接发送至非 JMS 应用程序时，请省略 MQRFH2。 这是因为此类应用程序不期望其 IBM MQ 消息中有 MQRFH2。

如果入局消息无 MQRFH2 头，那么缺省情况下从消息的 JMSReplyTo 头字段衍生的“队列”或“主题”对象会设置此标志，从而使发送至队列或主题的应答消息同样不包含 MQRFH2 头。 只有在原始消息具有 MQRFH2 头时，才能切换应答消息中包含 MQRFH2 头的行为，方式是将连接工厂的 TARGCLIENTMATCHING 属性设置为 NO。

第 116 页的图 10 显示了如何将 JMS 消息结构转换为 IBM MQ 消息，然后再返回：

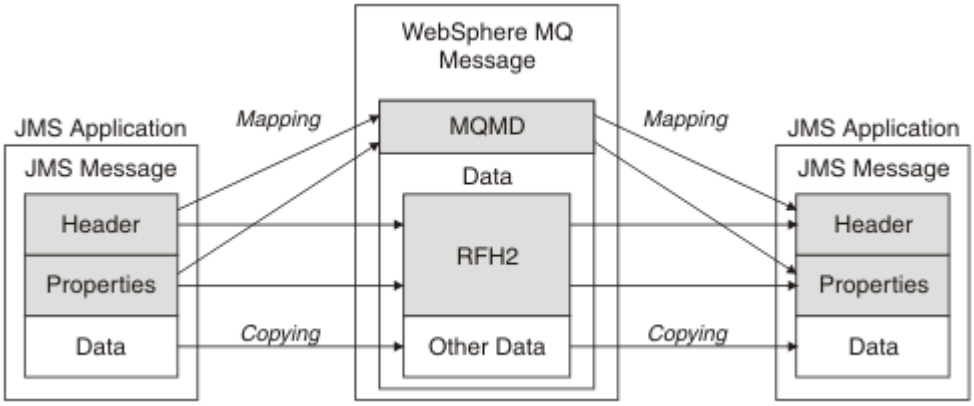


图 *10:* 如何使用 *MQRFH2* 头在 *JMS* 与 *IBM MQ* 之间转换消息

结构转换方式有两种：

映射
当 MQMD 包含相当于 JMS 字段的字段时，JMS 字段会映射到 MQMD 字段。 其他 MQMD 字段将公开为 JMS 属性，因为 JMS 应用程序在与非 JMS 应用程序通信时，可能需要获取或设置这些字段。

复制
如果无等效的 MQMD，那么会将 JMS 作为 MQRFH2 内部的一个字段进行传递（可能会进行转换）。

MQRFH2 头和 *JMS*
此主题集合描述 MQRFH V2 头，此头带有与消息内容关联的特定于 JMS 的数据。 MQRFH V2 头是一个可扩展头，也可以带有不与 JMS 直接关联的其他信息。 但是，本部分仅涉及 JMS 如何使用此头。要获取完整描述，请参阅 MQRFH2 - 规则及格式化头 2。

此头分两部分：固定部分和可变部分。

固定部分
固定部分在标准 IBM MQ 头模式上建模且包含以下字段：

StrucId (MQCHAR4)
结构标识。

必须为 MQRFH_STRUC_ID（值：“RFH ”）（初始值）。

同样定义了 MQRFH_STRUC_ID_ARRAY（值：“R”、“F”、“H”、“ ”）。

Version (MQLONG)
结构版本号。

必须为 MQRFH_VERSION_2（值：2）（初始值）。

StrucLength (MQLONG)
MQRFH2 总长度（包括 NameValueData 字段）。

设置到 StrucLength 的值必须是 4 的倍数（可使用空格字符填补 NameValueData 字段中的数据以达到此要求）。

Encoding (MQLONG)
数据编码。

对 MQRFH2 之后的消息部分中的任何数字数据（下一个头或此头之后的消息数据）进行编码。

CodedCharSetId (MQLONG)
编码字符集标识。

表示 MQRFH2 之后的消息部分中的任何字符数据（下一个头或此头之后的消息数据）。

Format (MQCHAR8)
格式名称。

MQRFH2 之后的消息部分的格式名称。

Flags (MQLONG)
标志。

MQRFH_NO_FLAGS = 0。 未设置标志。

NameValueCCSID (MQLONG)
此头中包含的 NameValueData 字符串的编码字符集标识 (CCSID)。 可以在与头（StrucID 和 Format）中包含的其他字符串不同的字符集中对 NameValueData 进行编码。

如果 NameValueCCSID 为双字节 Unicode CCSID（1200、13488 或 17584），那么 Unicode 的字节顺序与 MQRFH2 中的数字字段的字节定序相同。 （例如，Version、StrucLength 和 NameValueCCSID 本身。）

NameValueCCSID 从下表中提取值：

表 *15: NameValueCCSID* 字段的可能的值

CCSID	**含义**
1200	UTF-16，支持最新的 Unicode 版本
13488	UTF-16，Unicode V2.0 子集
17584	UTF-16，Unicode V3.0 子集（包含欧元符号 €）
1208	UTF-8，支持最新的 Unicode 版本

可变部分
可变部分在固定部分之后。 可变部分包含 MQRFH2 文件夹的可变编号。 每个文件夹均包含元素或属性的可变编号。 文件夹将相关属性分组到一起。 由 JMS 创建的 MQRFH2 头可包含以下任何文件夹：

mcd 文件夹

mcd 包含描述消息的格式的属性。 例如，消息服务域 Msd 属性将 JMS 消息标识为 JMSTextMessage、JMSBytesMessage、JMSStreamMessage、JMSMapMessage、JMSObjectMessage 或 null。

mcd 文件夹始终出现在包含 MQRFH2 的 JMS 消息中。

它始终显示在包含从 IBM Integration Bus 发送的 MQRFH2 的消息中。 它描述消息的域、格式、类型和消息集。

表 *16: mcd* 属性名称、同义词、数据类型和文件夹

属性同义词	**属性名**	**数据类型**	**文件夹**
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>

表 16: *mcd* 属性名称、同义词、数据类型和文件夹 *(继续)*			
属性同义词	**属性名**	**数据类型**	**文件夹**
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

不要在 mcd 文件夹中添加您自己的属性。

jms 文件夹

jms 包含 JMS 头字段，以及无法在 MQMD 中完全表达的 JMSX 属性。 jms 文件夹始终显示在 JMS MQRFH2 中。

usr 文件夹

usr 包含与消息关联的应用程序定义 JMS 属性。 仅当应用程序已设置应用程序定义的属性时，usr 文件夹才存在。

mqext 文件夹

mqext 包含以下类型的属性:

- 仅由 WebSphere Application Server 使用的属性。
- 与消息延迟传递相关的属性。

如果应用程序已设置至少一个 IBM 定义的属性或者已使用传递延迟，那么将存在该文件夹。

表 17: *mqext* 属性名称、同义词、数据类型和文件夹			
属性同义词	**属性名**	**数据类型**	**文件夹**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

不要在 mqext 文件夹中添加您自己的属性。

mqps 文件夹

mqps 包含仅由 IBM MQ 发布/预订使用的属性。 仅当应用程序已设置至少一个集成的发布/预订属性时，该文件夹才存在。

表 18: *mqps* 属性名称、同义词、数据类型和文件夹			
属性同义词	**属性名**	**数据类型**	**文件夹**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData...*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>

表 *18: mqps* 属性名称、同义词、数据类型和文件夹 *(*继续*)*			
属性同义词	**属性名**	**数据类型**	**文件夹**
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

不要在 mqps 文件夹中添加您自己的属性。

第 119 页的表 19 显示了属性名称的完整列表。

表 *19: JMS* 使用的 *MQRFH2* 文件夹和属性				
JMS 字段名称	**Java 类型**	**MQRFH2 文件夹名称**	**属性名**	**类型/值**
JMSDestination	Destination	jms	Dst	字符串
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	字符串	jms	Cid	字符串
JMSReplyTo	Destination	jms	Rto	字符串
JMSTimestamp	long	jms	Tms	i8
JMSType	字符串	mcd	Type、Set、Fmt	字符串
JMSXGroupID	字符串	jms	Gid	字符串
JMSXGroupSeq	int	jms	Seq	i4
xxx（用户定义）	任何	usr	xxx	任意
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength (MQLONG)
紧跟此长度字段（不包含其自身长度）的 NameValueData 字符串的长度（以字节为单位）。

NameValueData (MQCHARn)
单个字符串，其字节长度由前面的 NameValueLength 字段给出。 它包含具有一系列属性的文件夹。 每个属性均为名称/类型/值三元组，包含在其名称为文件夹名称的 XML 元素中，如下所示：

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

结束 </foldername> 标记后面可以后跟空格作为填充字符。 每个三元组均使用类似 XML 的语法进行编码：

```
<name dt='datatype'>value</name>
```

dt='datatype' 元素是可选的，许多属性都省略了该元素，因为数据类型是预定义的。 如果包含此项，那么 dt= 标记之前必须添加一个或多个空格字符。

name
是属性名称；请参阅第 119 页的表 19。

datatype
折叠后，必须匹配第 120 页的表 20 中列出的其中一个数据类型。

value
是要使用第 120 页的表 20 中的定义表示的值的字符串表示。

使用以下语法对空值进行编码：

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

不要使用 xsi:nil='false'。

表 *20:* 属性数据类型

数据类型	**定义**
字符串	任何字符序列（不包括 < 和 &）
布尔值	字符 0 或 1 (0 = false，1 = true)
bin.hex	表示八位元的十六进制数字
i1	使用数字 0..9 表示的数字，带有可选符号（无小数或指数）。 必须在 -128 到 127（含）之间
i2	使用数字 0..9 表示的数字，带有可选符号（无小数或指数）。 必须在 -32768 到 32767（含）之间
i4	使用数字 0..9 表示的数字，带有可选符号（无小数或指数）。 必须在 -2147483648 到 2147483647（含）之间
i8	使用数字 0..9 表示的数字，带有可选符号（无小数或指数）。 必须在 -9223372036854775808 到 9223372036854775807（含）之间
int	使用数字 0..9 表示的数字，带有可选符号（无小数或指数）。 必须在与 i8 相同的范围之间。 如果发送方不想将特定的精度与属性相关联，可使用它来代替其中一种 i* 类型
r4	浮点数，幅度 <= 3.40282347E+38 且 >= 1.175E-37，使用数字 0..9、可选符号、可选小数位数、可选指数表示
r8	浮点数，幅度 <= 1.7976931348623E+308 且 >= 2.225E-307，使用数字 0..9、可选符号、可选小数位数、可选指数表示

字符串值可以包含空格。 必须在字符串值中使用以下转义序列：

- & 表示 & 字符
- < 表示 < 字符

您可以使用以下转义序列，但它们不是必需的：

- > 表示 > 字符
- ' 表示 ' 字符
- " 表示 " 字符

具有相应的 *MQMD* 字段的 *JMS* 字段及属性
这些表显示了等效于 JMS 标题字段、JMS 属性以及特定于 JMS 提供程序的属性的 MQMD 字段。

第 121 页的表 21 列出 JMS 标题字段；第 121 页的表 22 列出直接映射到 MQMD 字段的 JMS 属性。 第 121 页的表 23 列出特定于提供程序的属性及其映射到的 MQMD 字段。

表 *21:* 映射到 *MQMD* 字段的 *JMS* 标题字段

JMS 标题字段	**Java 类型**	**MQMD 字段**	**C 类型**
JMSDeliveryMode	int	持久	MQLONG
JMSExpiration	long	到期	MQLONG
JMSPriority	int	优先级	MQLONG
JMSMessageID	字符串	MsgID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	字符串	CorrelId	MQBYTE24

表 *22:* 映射到 *MQMD* 字段的 *JMS* 属性

JMS 属性	**Java 类型**	**MQMD 字段**	**C 类型**
JMSXUserID	字符串	UserIdentifier	MQCHAR12
JMSXAppID	字符串	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	字符串	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

表 *23:* 映射到 *MQMD* 字段的特定于 *JMS* 提供程序的属性

特定于 JMS 提供程序的属性	**Java 类型**	**MQMD 字段**	**C 类型**
JMS_IBM_Report_Exception	int	报告	MQLONG
JMS_IBM_Report_Expiration	int	报告	MQLONG
JMS_IBM_Report_COA	int	报告	MQLONG
JMS_IBM_Report_COD	int	报告	MQLONG
JMS_IBM_Report_PAN	int	报告	MQLONG
JMS_IBM_Report_NAN	int	报告	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	报告	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	报告	MQLONG
JMS_IBM_Report_Discard_Msg	int	报告	MQLONG

表 *23:* 映射到 *MQMD* 字段的特定于 *JMS* 提供程序的属性 *(*继续*)*			
特定于 JMS 提供程序的属性	**Java 类型**	**MQMD 字段**	**C 类型**
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	字符串	Format 第 122 页的『1』	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	编码	MQLONG
JMS_IBM_Character_Set	字符串	CodedCharacterSetId 第 122 页的『2』	MQLONG
JMS_IBM_PutDate	字符串	PutDate	MQCHAR8
JMS_IBM_PutTime	字符串	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	布尔值	MsgFlags	MQLONG

注:

1. JMS_IBM_Format 表示消息体格式。 这可以由设置消息的 JMS_IBM_Format 属性的应用程序定义 (请注意，存在 8 字符限制)，也可以缺省为适合于 JMS 消息类型的消息体的 IBM MQ 格式。 只有在消息不包含 RFH 或 RFH2 部分时，JMS_IBM_Format 才能映射到“MQMD 格式”字段。 在典型消息中，它会映射到紧挨消息体之前的 RFH2 的“格式”字段。
2. JMS_IBM_Character_Set property 值是一个字符串值，包含对于数字 CodedCharacterSetId 值等效的 Java 字符集。 MQMD 字段 CodedCharacterSetId 是数字值，包含由 JMS_IBM_Character_Set 属性指定的 Java 字符集字符串的等效项。

将 *JMS* 字段映射到 *IBM MQ* 字段（外发消息）
这些表显示了如何在 send() 或 publish() 时间将 JMS 头和属性字段映射到 MQMD 和 MQRFH2 字段。

第 122 页的表 24 显示了如何在 send() 或 publish() 时间将 JMS 头字段映射到 MQMD/RFH2 字段。 第 123 页的表 25 显示了如何在 send() 或 publish() 时间将 JMS 属性映射到 MQMD/RFH2 字段。 第 123 页的表 26 显示了如何在 send() 或 publish() 时间将特定于 JMS 提供者的属性映射到 MQMD 字段。

对于由“消息对象”标记为“已设置”的字段，传输的值是 send() 或 publish() 操作前一刻的 JMS 消息中保留的值。 该操作会将 JMS 消息中的值保持不变。

对于由“发送方法”标记为“已设置”的字段，执行 send() 或 publish() 时会分配值（JMS 消息中保留的任何值都将被忽略）。 将更新 JMS 消息中的值以显示所用的值。

未传输标记为“仅接收”的字段，并且 send() 或 publish() 在消息中使其保持不变。

表 *24:* 外发消息字段映射			
JMS 头字段名称	**用于传输的 MQMD 字段**	**头**	**设置途径**
JMSDestination		MQRFH2	发送方法
JMSDeliveryMode	持久	MQRFH2	发送方法
JMSExpiration	到期	MQRFH2	发送方法
JMSPriority	优先级	MQRFH2	发送方法
JMSMessageID	MsgID		发送方法
JMSTimestamp	PutDate/PutTime		发送方法
JMSCorrelationID	CorrelId	MQRFH2	消息对象

表 *24:* 外发消息字段映射 *(*继续*)*			
JMS 头字段名称	**用于传输的 MQMD 字段**	**头**	**设置途径**
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	消息对象
JMSType		MQRFH2	消息对象
JMSRedelivered			仅接收

注:

1. MQMD 字段 CodedCharacterSetId 是数字值，包含由 JMS_IBM_Character_Set 属性指定的 Java 字符集字符串的等效项。

表 *25:* 外发消息 *JMS* 属性映射			
JMS 属性名称	**用于传输的 MQMD 字段**	**头**	**设置途径**
JMSXUserID	UserIdentifier		发送方法
JMSXAppID	PutApplName		发送方法
JMSXDeliveryCount			仅接收
JMSXGroupID	GroupId	MQRFH2	消息对象
JMSXGroupSeq	MsgSeqNumber	MQRFH2	消息对象

注:

这些属性根据 JMS 规范定义为只读，并且通过 JMS 提供程序进行设置（在一些情况下可选）。

在 IBM MQ classes for JMS 中，应用程序可能会覆盖其中两个属性。 为此，请确保通过设置以下属性来相应地配置目标:

1. 将属性 `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` 设置为 `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`。
2. 将属性 `WMQConstants.WMQ_MQMD_WRITE_ENABLED` 设置为 `true`。

应用程序可能会覆盖以下属性:

JMSXAppID
可以通过在消息上设置属性 `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` 来覆盖此属性 - 值应该是 Java 字符串。

JMSXGroupID
可以通过在消息上设置属性 `WMQConstants.JMS_IBM_MQMD_GROUPID` 来覆盖此属性 - 值应该是字节数组。

表 *26:* 外发消息特定于 *JMS* 提供者的属性映射			
特定于 JMS 提供者的属性名称	**用于传输的 MQMD 字段**	**头**	**设置途径**
JMS_IBM_Report_Exception	报告		消息对象
JMS_IBM_Report_Expiration	报告		消息对象
JMS_IBM_Report_COA/COD	报告		消息对象
JMS_IBM_Report_NAN/PAN	报告		消息对象
JMS_IBM_Report_Pass_Msg_ID	报告		消息对象
JMS_IBM_Report_Pass_Correl_ID	报告		消息对象
JMS_IBM_Report_Discard_Msg	报告		消息对象

表 *26:* 外发消息特定于 *JMS* 提供者的属性映射 *(*继续*)*			
特定于 JMS 提供者的属性名称	**用于传输的 MQMD 字段**	**头**	**设置途径**
JMS_IBM_MsgType	MsgType		消息对象
JMS_IBM_Feedback	Feedback		消息对象
JMS_IBM_Format	格式		消息对象
JMS_IBM_PutApplType	PutApplType		发送方法
JMS_IBM_Encoding	编码		消息对象
JMS_IBM_Character_Set	CodedCharacterSetId		消息对象
JMS_IBM_PutDate	PutDate		发送方法
JMS_IBM_PutTime	PutTime		发送方法
JMS_IBM_Last_Msg_In_Group	MsgFlags		消息对象

在执行 *send()* 或 *publish()* 时映射 *JMS* 头字段
这些注释是关于在执行 send() 或 publish() 时映射 JMS 字段的。

JMSDestination 到 MQRFH2
它存储为可序列化目标对象的显著特征，以便接收 JMS 能够重组等效的目标对象的字符串。 MQRFH2 字段编码为 URI（请参阅 第 175 页的『统一资源标识 (URI)』 以获取 URI 表示法的详细信息）。

JMSReplyTo 到 MQMD.ReplyToQ、ReplyToQMgr 和 MQRFH2
队列名称复制到 MQMD.ReplyToQ 字段，队列管理器名称复制到 ReplyToQMgr 字段。 目标扩展信息（保存在目标对象中的其他有用的详细信息）复制到 MQRFH2 字段。 MQRFH2 字段编码为 URI（请参阅第 175 页的『统一资源标识 (URI)』，以获取 URI 表示法的详细信息）。

JMSDeliveryMode 到 MQMD.Persistence
JMSDeliveryMode 值由 send() 或 publish() 方法或 MessageProducer 来设置，除非“目标对象”覆盖它。 JMSDeliveryMode 值映射到 MQMD.Persistence 字段，如下：

- JMS 值 PERSISTENT 相当于 MQPER_PERSISTENT
- JMS 值 NON_PERSISTENT 相当于 MQPER_NOT_PERSISTENT

如果未将 MQQueue 持久性属性设置为 WMQConstants.WMQ_PER_QDEF，那么还将在 MQRFH2 中对交付方式值进行编码。

JMSExpiration 到/从 MQMD.Expiry、MQRFH2
JMSExpiration 存储到期时间（当前时间和生存时间之和），而 MQMD 存储生存时间。 同样，JMSExpiration 单位为毫秒，但 MQMD.Expiry 单位为十分之一秒。

- 如果 send() 方法用于设置无限的生存时间，那么 MQMD.Expiry 将设置为 MQEI_UNLIMITED，并且 MQRFH2 中未对 JMSExpiration 进行编码。
- 如果 send() 方法用于设置小于 214748364.7 秒（约 7 年）的生存时间，那么生存时间将存储在 MQMD.Expiry 中，并且到期时间（毫秒）将在 MQRFH2 中编码为 i8 值。
- 如果 send() 方法用于设置大于 214748364.7 秒的生存时间，那么 MQMD.Expiry 将设置为 MQEI_UNLIMITED。 真实的到期时间（毫秒）在 MQRFH2 中编码为 i8 值。

JMSPriority 到 MQMD.Priority
将 JMSPriority 值 (0-9) 直接映射到 MQMD 优先级值 (0-9) 上。 如果将 JMSPriority 设置为非缺省值，那么还将在 MQRFH2 中对优先级进行编码。

从 MQMD.MessageID 到 JMSMessageID
所有从 JMS 发送的消息都由 IBM MQ 分配了唯一的消息标识。 调用 MQPUT 后，MQMD.MessageId 字段中将返回分配值，并传回到 JMSMessageID 字段中的应用程序。 IBM MQ messageId 是一个 24 字节二进制值，而 JMSMessageID 是字符串。 JMSMessageID 由转换为一连串 48 个十六进制字符且以字符标识开头的二进制 messageId 值组成： JMS 提供可设置来禁用生成消息标识的提示。 此提示会被忽

略，并且在所有情况下均会分配唯一的标识。 任何在 send() 前设置到 JMSMessageID 字段的值都会被覆盖。

如果确实需要指定 MQMD.MessageID，您可以使用 第 189 页的『在 IBM MQ classes for JMS 应用程序中读写消息描述符』中描述的其中一个 IBM MQ JMS 扩展来执行此操作。

JMSTimestamp 到 MQRFH2
send 期间，将根据 JVM 时钟设置 JMSTimestamp 字段。 此值将设置到 MQRFH2。 任何在 send() 前设置到 JMSTimestamp 字段的值都会被覆盖。 另请参阅 JMS_IBM_PutDate 和 JMS_IBM_PutTime 属性。

JMSType 到 MQRFH2
此字符串设置到 MQRFH2 mcd.Type 字段。 如果它采用 URI 格式，可能还会影响 mcd.Set 和 mcd.Fmt 字段。

JMSCorrelationID 到 MQMD.CorrelId、MQRFH2
JMSCorrelationID 可保留下列其中一项:

特定于提供程序的消息标识
这是来自先前发送或接收的消息的消息标识，此标识应该是包含 48 位小写十六进制数字的字符串（带有 `ID:` 前缀）。去掉前缀后，剩余字符将转换为二进制字符，然后将它们设置为 MQMD.CorrelId 字段。 未在 MQRFH2 中对 CorrelId 值编码。

provider-native byte[] 值
值将复制到 MQMD.CorrelId 字段 - 填补空，或在需要时截断到 24 个字节。 未在 MQRFH2 中对 CorrelId 值编码。

特定于应用程序的字符串
值将复制到 MQRFH2。 字符串的前 24 个字节将写入 MQMD.CorrelID（采用 UTF8 格式）。

映射 *JMS* 属性字段
这些说明引用了 IBM MQ 消息中 JMS 属性字段的映射。

从 MQMD UserIdentifier 到 JMSXUserID
返回时从发送调用设置 JMSXUserID。

从 MQMD PutApplName 到 JMSXAppID
返回时从发送调用设置 JSMXAppID。

从 JMSXGroupID 到 MQRFH2（点到点）
对于点到点消息，JMSXGroupID 将复制到 MQMD GroupID 字段。 如果 JMSXGroupID 以前缀 ID: 开头，它将转换为二进制形式。 否则，将编码为 UTF8 字符串。 如果要求长度为 24 个字节，那么会填补或截断值。 已设置 MQMF_MSG_IN_GROUP 标志。

从 JMSXGroupID 到 MQRFH2（发布/预订）
对于发布/预订消息，JMSXGroupID 将作为字符串复制到 MQRFH2。

JMSXGroupSeq MQMD MsgSeqNumber（点到点）
对于点到点消息，JMSXGroupSeq 将复制到 MQMD MsgSeqNumber 字段。 已设置 MQMF_MSG_IN_GROUP 标志。

JMSXGroupSeq MQMD MsgSeqNumber（发布/预订）
对于发布/预订消息，JMSXGroupSeq 将作为 i4 复制到 MQRFH2。

映射特定于 *JMS* 提供程序的字段
以下注释说明将特定于 JMS 提供程序的字段映射到 IBM MQ 消息。

JMS_IBM_Report_*XXX* 到 MQMD 报告
JMS 应用程序可使用以下 JMS_IBM_Report_*XXX* 属性来设置 MQMD 报告选项。 将单个 MQMD 映射至几个 JMS_IBM_Report_*XXX* 属性。

JMS_IBM_Report_*XXX* 常量位于 `com.ibm.msg.client.jakarta.wmq.WMQConstants` 或 `com.ibm.msg.client.wmq.WMQConstants` 中。

JMS_IBM_Report_Exception

MQRO_EXCEPTION 或
MQRO_EXCEPTION_WITH_DATA 或
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION 或
MQRO_EXPIRATION_WITH_DATA 或
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA 或
MQRO_COA_WITH_DATA 或
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD 或
MQRO_COD_WITH_DATA 或
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

MQRO 值位于 `com.ibm.mq.constants.CMQC` 中。

JMS_IBM_MsgType 到 MQMD MsgType
值直接映射到 MQMD MsgType 上。 如果应用程序未设置显式值 JMS_IBM_MsgType，那么将使用缺省值。 此缺省值可按如下所述来确定：

- 如果将 JMSReplyTo 设置为 IBM MQ 队列目标，那么 MSGType 将设置为 MQMT_REQUEST 值
- 如果未设置 JMSReplyTo，或设置为除 IBM MQ 队列目标以外的任何项，那么 MsgType 将设置为 MQMT_DATAGRAM 值

JMS_IBM_Feedback 到 MQMD Feedback
值直接映射到 MQMD Feedback 上。

JMS_IBM_Format 到 MQMD Format
值直接映射到 MQMD Format 上。

JMS_IBM_Encoding 到 MQMD Encoding
如果设置，那么此属性将覆盖“目标队列”或“主题”的数字编码。

JMS_IBM_Character_Set 到 MQMD CodedCharacterSetId
如果设置，那么此属性将覆盖“目标队列”或“主题”的编码字符集属性。

从 MQMD PutDate 到 JMS_IBM_PutDate
send 期间，将从 MQMD 中的 PutDate 字段直接设置此属性值。 任何在 send 前设置到 JMS_IBM_PutDate 属性的值都会被覆盖。 此字段是一个 8 字符字符串，采用 YYYYMMDD 的 IBM MQ 日期格式。 此属性可与 JMS_IBM_PutTime 属性一起使用，来确定根据队列管理器放入消息的时间。

从 MQMD PutTime 到 JMS_IBM_PutTime
send 期间，将从 MQMD 中的 PutTime 字段直接设置此属性值。 任何在 send 前设置到 JMS_IBM_PutTime 属性的值都会被覆盖。 此字段是一个 8 字符字符串，采用 HHMMSSTH 的 IBM MQ 时间格式。 此属性可与 JMS_IBM_PutDate 属性一起使用，来确定根据队列管理器放入消息的时间。

JMS_IBM_Last_Msg_In_Group 到 MQMD MsgFlags
对于点到点消息传递，此布尔值会映射到 MQMD MsgFlags 字段中的 MQMF_LAST_MSG_IN_GROUP 标志。 通常，它将与 JMSXGroupID 和 JMSXGroupSeq 属性一起使用，以向传统 IBM MQ 应用程序说明此消息为组中最后一条消息。 发布/预订消息传递忽略此属性。

将 *IBM MQ* 字段映射到 *JMS* 字段（入局消息）
这些表显示了如何在 get() 或 receive() 时间将 JMS 头和属性字段映射到 MQMD 和 MQRFH2 字段。

第 127 页的表 27 显示了如何在 get() 或 receive() 时间将 JMS 头字段映射到 MQMD/MQRFH2 字段。 第 127 页的表 28 显示了如何在 get() 或 receive() 时间将 JMS 属性字段映射到 MQMD/MQRFH2 字段。 第 128 页的表 29 显示了如何映射特定于 JMS 提供者的属性。

表 *27:* 入局消息 *JMS* 头字段映射

JMS 头字段名称	**MQMD 字段检索自**	**MQRFH2 字段检索自**
JMSDestination		jms.Dst 或 mqps.Top 第 127 页的『1』
JMSDeliveryMode	Persistence 第 127 页的『2』	jms.Dlv 第 127 页的『2』
JMSExpiration		jms.Exp
JMSPriority	优先级	
JMSMessageID	MsgID	
JMSTimestamp	PutDate 第 127 页的『2』 PutTime 第 127 页的『2』	jms.Tms 第 127 页的『2』
JMSCorrelationID	CorrelId 第 127 页的『2』	jms.Cid 第 127 页的『2』
JMSReplyTo	ReplyToQ 第 127 页的『2』 ReplyToQMgr 第 127 页的『2』	jms.Rto 第 127 页的『2』
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

注:

1. 如果设置了 jms.Dst 和 mqps.Top，将使用 jms.Dst 中的值。
2. 对于可从 MQRFH2 或 MQMD 检索值的属性，如果二者皆可用，将使用 MQRFH2 中的设置。
3. JMS_IBM_Character_Set property 值是一个字符串值，包含对于数字 CodedCharacterSetId 值等效的 Java 字符集。

表 *28:* 入局消息属性映射

JMS 属性名称	**MQMD 字段检索自**	**MQRFH2 字段检索自**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId 第 128 页的『1』	jms.Gid 第 128 页的『1』

表 *28:* 入局消息属性映射 *(继续)*		
JMS 属性名称	**MQMD 字段检索自**	**MQRFH2 字段检索自**
JMSXGroupSeq	MsgSeqNumber 第 128 页的『1』	jms.Seq 第 128 页的『1』

注:

1. 对于可从 MQRFH2 或 MQMD 检索值的属性，如果二者皆可用，将使用 MQRFH2 中的设置。 只有设置了 MQMF_MSG_IN_GROUP 或 MQMF_LAST_MSG_IN_GROUP 消息标志时，才会从 MQMD 值设置属性。

表 *29:* 入局消息特定于提供者的 *JMS* 属性映射		
JMS 属性名称	**MQMD 字段检索自**	**MQRFH2 字段检索自**
JMS_IBM_Report_Exception	报告	
JMS_IBM_Report_Expiration	报告	
JMS_IBM_Report_COA	报告	
JMS_IBM_Report_COD	报告	
JMS_IBM_Report_PAN	报告	
JMS_IBM_Report_NAN	报告	
JMS_IBM_Report_Pass_Msg_ID	报告	
JMS_IBM_Report_Pass_Correl_ID	报告	
JMS_IBM_Report_Discard_Msg	报告	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	格式	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding 第 128 页的『1』	编码	
JMS_IBM_Character_Set 第 128 页的『1』	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. 仅在入局消息为字节消息时设置。

在 *JMS* 应用程序与传统 *IBM MQ* 应用程序间交换消息

此主题描述了在 JMS 应用程序与无法处理 MQRFH2 头的传统 IBM MQ 应用程序交换消息时发生的情况。

第 129 页的图 11 显示映射。

管理员通过将目标的 TARGCLIENT 属性设置为 *MQ*，指示 JMS 应用程序正在与传统 IBM MQ 应用程序通信。 这表明将不会生成 MQRFH2 头。 如果不是这样，那么接收应用程序必须要能够处理 MQRFH2 头。

从 JMS 到针对传统 IBM MQ 应用程序的 MQMD 的映射与从 JMS 到针对 JMS 应用程序的 MQMD 的映射相同。 如果 IBM MQ classes for JMS 接收到 MQMD *Format* 字段设置为除 MQFMT_RFH2 以外的任何内容的 IBM MQ 消息，那么将从非 JMS 应用程序接收数据。 如果格式为 MQFMT_STRING，那么将以 JMS 文本消息形式接收消息。 否则，将以 JMS 字节消息形式接收消息。 由于无 MQRFH2，因此只会复原 MQMD 中传输的那些 JMS 属性。

如果 IBM MQ classes for JMS 收到无 MQRFH2 头的消息，那么缺省情况下从消息的 JMSReplyTo 头字段衍生的“队列”或“主题”对象的 TARGCLIENT 属性将设置为 MQ。 这意味着发送至队列或主题的应答消息同样不包含 MQRFH2 头。 只有在原始消息具有 MQRFH2 头时，才能切换应答消息中包含 MQRFH2 头的行为，方式是将连接工厂的 TARGCLIENTMATCHING 属性设置为 NO。

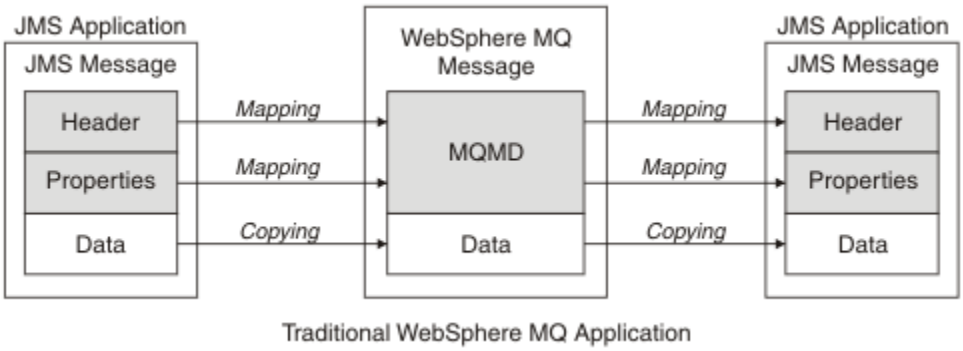


图 *11:* 如何将 *JMS* 消息转换为无 *MQRFH2* 头的 *IBM MQ* 消息

JMS 消息体
本主题包含有关消息体本身编码的信息。 编码取决于 JMS 消息的类型。

ObjectMessage
ObjectMessage 是 Java Runtime 以正常方式进行序列化的对象。

TextMessage
TextMessage 是编码字符串。 对于外发消息，字符串在由目标对象给定的字符集中进行编码。 缺省情况下使用 UTF8 编码（UTF8 编码从消息的第一个字符开始；开头处无长度字段）。 但是，可以指定 IBM MQ classes for JMS 支持的任何其他字符集。 此类字符集主要在将消息发送到非 JMS 应用程序时使用。

如果字符集是双字节集（包括 UTF16），那么目标对象的整数编码规范可确定字节顺序。

使用消息本身中指定的字符集及编码来解释入局消息。 这些规范在最后一个 IBM MQ 头中，如果没有头，那么在 MQMD 中。 对于 JMS 消息，最后一个头通常为 MQRFH2。

BytesMessage
缺省情况下，BytesMessage 是 JMS 1.0.2 规范及关联 Java 文档所定义的一系列字节。

对于由应用程序本身组合的外发消息，目标对象的编码属性可用于覆盖消息中所含的整数和浮点字段的编码。 例如，可以请求以 S/390 格式而非 IEEE 格式存储浮点值。

使用消息本身中指定的数字编码来解释入局消息。 此规范在最后一个 IBM MQ 头中，如果没有头，那么在 MQMD 中。 对于 JMS 消息，最后一个头通常为 MQRFH2。

如果收到 BytesMessage，并且在不进行修改的情况下重新发送，那么消息主体将按照其接收的方式逐字节进行传输。 目标对象的编码属性对主体无任何影响。 可以在 BytesMessage 中明确发送的唯一的类似字符串的实体是 UTF8 字符串。 它采用 Java UTF8 格式编码，并以双字节长度字段开头。 目标对象的字符集属性对外发 BytesMessage 编码无任何影响。 入局 IBM MQ 消息中的字符集值对于将此消息解释为 JMS BytesMessage 无任何影响。

非 Java 应用程序不太可能能够识别 Java UTF8 编码。 因此，对于要发送包含文本数据的 BytesMessage 的 JMS 应用程序，应用程序本身必须将其字符串转换为字节数组，并将这些字节数组写入 BytesMessage。

MapMessage
MapMessage 是包含编码如下的 XML 名称/类型/值三元组的字符串：

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

其中 datatype 是第 120 页的表 20 中列出的其中一个数据类型。 缺省数据类型为 string，因此会为字符串元素省略 dt="string" 属性。

遵循适用于文本消息的规则来确定用来编码或解释组成映射消息主体的 XML 字符串的字符集。

低于版本 5.3 的 IBM MQ classes for JMS 版本会对采用以下格式的映射消息主体进行编码：

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

5.3 和更高版本的 IBM MQ classes for JMS 可以解释任一格式，但低于 5.3 的 IBM MQ classes for JMS 版本无法解释当前格式。

如果某应用程序需要将映射消息发送到使用低于版本 5.3 的 IBM MQ classes for JMS 版本的其他应用程序，那么发送应用程序必须调用连接工厂方法 setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)，以指定以先前的格式发送映射消息。 缺省情况下，所有映射消息均以当前格式进行发送。

StreamMessage
StreamMessage 与映射消息类似，但无元素名称：

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

其中 datatype 是第 120 页的表 20 中列出的其中一个数据类型。 缺省数据类型为 string，因此会为字符串元素省略 dt="string" 属性。

用于编码或解释组成 StreamMessage 主体的 XML 字符串的字符集遵循适用于 TextMessage 的规则来确定。

MQRFH2.format 字段设置如下：

MQFMT_NONE
用于 ObjectMessage、BytesMessage 或不含主体的消息。

MQFMT_STRING
用于 TextMessage、StreamMessage 或 MapMessage。

JMS 消息转换
发送和接收消息时，将在 JMS 中执行消息数据转换。 IBM MQ 会自动执行大部分数据转换过程。 在 JMS 应用程序间传输消息时，它会转换文本和数字数据。 在 JMS 应用程序与 IBM MQ 应用程序之间交换 JMSTextMessage 时，将转换文本。

如果您计划完成更复杂的消息交换，那么您会对以下主题产生兴趣。 复杂消息交换包括：

- 在 IBM MQ 应用程序与 JMS 应用程序间传输非文本消息。
- 交换采用字节格式的文本数据。
- 在应用程序中转换文本。

JMS 消息数据

在应用程序间（即使是两个 JMS 应用程序间）交换文本和数字数据时，必须进行数据转换。 必须对文本和数字的内部表示进行编码，以便其能够在消息中传输。 编码过程会强制您决定如何表示数字和文本。 IBM MQ 负责管理对 JMS 消息中的文本和数字进行编码（JMSObjectMessage 除外），请参阅第 136 页的『JMSObjectMessage』。 它使用三个消息属性。 这三个属性是 CodedCharacterSetId、Encoding 和 Format。

这三个消息属性通常存储在 JMS 头、MQRFH2 以及 JMS 消息字段中。 如果消息类型为 MQ（而不是消息的 JMS 类型），那么属性将存储在消息描述符 MQMD 中。 属性用于转换 JMS 消息数据。 JMS 消息数据在 IBM MQ 消息的消息数据部分中传输。

JMS 消息属性

JMS 消息属性 (例如 JMS_IBM_CHARACTER_SET) 将在 JMS 消息的 MQRFH2 头部分中进行交换，除非已在没有 MQRFH2 的情况下发送消息。 只有 JMSTextMessage 和 JMSBytesMessage 能在无 MQRFH2 的情况下发送。 如果 JMS 属性作为 IBM MQ 消息属性存储在消息描述符 MQMD 中，那么会将其作为 MQMD 转换的一部分进行转换。 如果 JMS 属性存储在 MQRFH2 中，那么它将存储在由 MQRFH2.NameValueCCSID 指定的字符集中。 发送或接收消息时，消息属性会在 JVM 中转换为其内部表示，或从其内部表示进行转换。 将转换为消息描述符的字符集或 MQRFH2.NameValueCCSID，或从中进行转换。 数字数据转换为文本。

JMS 消息转换

以下主题包含当您计划交换需要转换的更复杂消息时将非常有用的示例及任务。

JMS 消息转换方法
许多数据转换方法对于 JMS 应用程序设计人员都是开放的。 这些方法没有排他性，某些应用程序可能会使用这些方法的组合。 如果应用程序正在仅交换文本，或正在仅与其他 JMS 应用程序交换消息，那么您通常不会考虑数据转换。 IBM MQ 将为您自动执行数据转换。

您可以提出有关如何解决消息转换的许多问题：

究竟是否需要考虑消息转换?
在某些情况（如 JMS 到 JMS 消息传输，以及与 IBM MQ 程序交换文本消息）下，IBM MQ 会为您自动执行必要的转换。 您可能出于性能原因希望控制数据转换，或可能正在交换具有预定义格式的复杂消息。 在这些情况下，必须理解消息转换，并阅读以下主题。

有哪些类型的转换?
有四种主要类型的转换，以下部分中分别做出说明：

1. 第 131 页的『JMS 客户机数据转换』
2. 第 132 页的『应用程序数据转换』
3. 第 132 页的『队列管理器数据转换』
4. 第 133 页的『消息通道数据转换』

应在何处执行转换?
第 133 页的『选择消息转换方法：接收方成功完成操作』部分描述了“接收方成功完成操作”的常规方法。 “接收方成功完成操作”还适用于 JMS 数据转换。

JMS 客户机数据转换

JMS 客户机 [1] 数据转换是将 Java 原语和对象转换为 JMS 消息中的字节，如同将其发送到目标一样，并在接收后转换回原样。 JMS 客户机数据转换使用 JMSMessage 类方法。 JMSMessage 类类型在第 134 页的表 30 中列出方法。

系统将针对读、获取、设置和写方法执行转换到数字和文本的内部 JVM 表示或从内部 JVM 表示进行转换。 发送消息以及在已接收的消息上调用任何读或获取方法时，将执行转换。

用于编写或设置消息内容的代码页和数字编码定义为目标属性。 可以管理方式更改目标代码页和数字编码。 应用程序还可以通过设置控制编写或设置消息内容的消息属性，来覆盖目标代码页和编码。

如果要在将 JMSBytesMessage 消息发送到未定义为 Native 编码的目标时转换数字编码，那么必须在发送消息前设置消息属性 JMS_IBM_ENCODING。 如果遵循“接收方成功完成操作”模式，或如果正在 JMS 应用程序间交换消息，那么应用程序无需设置 JMS_IBM_ENCODING。 在大多数情况下，可将 Encoding 属性保留为 Native。

[1] “JMS 客户机”是指实现 JMS 接口的 IBM MQ classes for JMS，它以客户机方式或绑定方式运行。

对于 JMSStreamMessage、JMSMapMessage 和 JMSTextMessage 消息，将使用目标的字符集标识属性。 发送时将忽略编码，因为数字将以文本格式写出。 如果目标字符集属性适用，那么 JMS 客户机应用程序在发送消息前无需设置 JMS_IBM_CHARACTER_SET。

要在消息中获取数据，应用程序会调用 JMS 消息读或获取方法。 这些方法参考前一个消息头中定义的，用于正确创建 Java 原语和对象的代码页和编码。

JMS 客户机数据转换符合在一个 JMS 客户机与另一个此客户机之间交换消息的大多数 JMS 应用程序的需要。 您未对任何明确的数据转换进行编码。 您未使用 java.nio.charset.Charset 类，在将文本写入文件时通常会使用它。 writeString 和 setString 方法会为您执行转换。

有关 JMS 客户机数据转换的更多详细信息，请参阅第 142 页的『JMS 客户机消息转换和编码』。

应用程序数据转换

JMS 客户机应用程序可以使用 java.nio.charset.Charset 类来执行显式字符数据转换；请参阅 第 135 页的图 14 和 第 135 页的图 15 中的示例。 字符串数据将使用 getBytes 方法转换为字节，并以字节形式发送。 将使用采用字节数组和 Charset 的 String 构造方法，将字节转换回文本。 使用 encode 和 decode Charset 方法转换字符数据。 通常以 JMSBytesMessage 形式发送或接收消息，因为 JMSBytesMessage 的消息部分不包含应用程序写入的数据以外的任何内容 [2] 也可以使用 JMSStreamMessage、JMSMapMessage 或 JMSObjectMessage 发送和接收字节。

无任何 Java 方法来对包含以不同编码格式表示的数字数据的字节进行编码和解码。 将使用数字 JMSMessage 读写方法，自动对数字数据进行编码和解码。 读写方法使用消息数据的 JMS_IBM_ENCODING 属性值。

应用程序数据转换的一个典型用例是 JMS 客户机使用应用程序数据转换从非 JMS 应用程序发送或接收格式化消息。 格式化消息包含按数据字段长度组织的文本、数字和字节数据。 除非非 JMS 应用程序已将消息格式指定为 “MQSTR”，否则会将消息构造为 JMSBytesMessage。 要在 JMSBytesMessage 中接收格式化消息数据，必须调用一系列方法。 必须以与将字段写入消息相同的顺序调用这些方法。 如果字段为数字字段，那么您必须知道数字数据的编码及长度。 如果任一字段均包含字节或文本数据，那么您必须知道消息中的任何字节数据的长度。 有两种易于使用的方法可用于将格式化消息转换为 Java 对象。

1. 构造对应于记录的 Java 类，以封装读写消息。 使用类的获取和设置方法来访问记录中的数据。
2. 通过扩展 com.ibm.mq.headers 类来构造对应于该记录的 Java 类。 使用表单的特定于类型的访问器 getStringValue(*fieldname*); 访问类中的数据

请参阅第 149 页的『与非 JMS 应用程序交换格式化记录』。

队列管理器数据转换

当 JMS 客户机程序获取消息时，队列管理器可以执行代码页转换。 此转换与为 C 程序执行的转换相同。 C 程序将 MQGMO_CONVERT 设置为 MQGET GetMsgOpts 参数选项；请参阅第 135 页的图 13。 队列管理器为正在接收消息的 JMS 客户机程序执行转换，如果将 WMQ_RECEIVE_CONVERSION 目标属性设置为 WMQ_RECEIVE_CONVERSION_QMGR，那么 JMS 客户机程序还可以设置目标属性；请参阅第 132 页的图 12。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

或者，

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

图 *12:* 启用队列管理器数据转换

[2] 一种例外情况：使用 writeUTF 编写的数据以 2 字节长度的字段开头。

在与非 JMS 应用程序交换消息时，将体现队列管理器转换的主要优势。 如果定义了消息中的 Format 字段，并且目标字符集或编码与消息不同，那么队列管理器将为目标应用程序执行数据转换（如应用程序请求转换）。 队列管理器会转换根据其中一种预定义 IBM MQ 消息类型 (例如 CICS bridge 头 (MQCIH)) 格式化的消息数据。 如果 Format 字段是用户定义的，那么队列管理器将查找具有 Format 字段中提供的名称的数据转换出口。

队列管理器数据转换用于通过“接收方成功完成操作”设计模式来获得最佳效果。 发送 JMS 客户机无需执行转换。 非 JMS 接收程序依靠转换出口来确保在所需代码页和编码中提供消息。 对于发送 JMS 客户机和非 JMS 接收方，此示例适用于 IBM MQ。

可使用数据转换出口实用程序 **crtmqcvx** 创建数据转换出口，以使队列管理器能够转换您自己的记录格式化数据。 您可以构建自己的记录格式，使用 com.ibm.mq.headers 访问该记录（作为 Java 类），并使用自己的转换出口转换该记录。 在 z/OS 上，实用程序称为 **CSQUCVX**；在 IBM i 上称为 **CVTMQMDTA**。 请参阅第 149 页的『与非 JMS 应用程序交换格式化记录』，

消息通道数据转换

IBM MQ 发送方、服务器、集群接收方和集群发送方通道均有一个消息转换选项 CONVERT。 发送消息时，可选择转换消息内容。 在通道发送端，将进行转换。 集群接收方定义用于自动定义相应的集群发送方通道。

如果无法使用其他形式的转换，那么通常将使用按消息通道转换数据。

选择消息转换方法：“接收方成功完成操作”

代码转换的 IBM MQ 应用程序设计中的常用方法是“接收方成功完成操作”。 “接收方成功完成操作”可减少消息转换数。 它还会避免在消息传输期间一些中间队列管理器上进行消息转换失败后，出现意外的通道错误的问题。 只有在存在一些接收方无法成功操作的原因的情况下，“接收方成功完成操作”规则才会被打破。 例如，接收平台可能没有正确的字符集。

“接收方成功完成操作”对于 JMS 客户机应用程序也是很好的常规指南。 但在特定情况下，转换到源上正确的字符集可能会更有效。 在发送包含文本或数字类型的消息时，必须从 JVM 内部表示进行转换。 如果接收方不是 JMS 客户机，那么转换到其所需的字符集可能使非 JMS 接收方无需执行转换。 如果接收方是 JMS 客户机，它无论如何都将重新转换以解码消息数据，并创建 Java 原语和对象。

JMS 客户机应用程序与采用诸如 C 等语言编写的应用程序之间的差异在于，Java 必须执行数据转换。 Java 应用程序必须将数字和文本从其内部表示转换为消息中使用的编码格式。

通过设置目标或消息属性，可设置 IBM MQ 用于对消息中的数字和文本编码的字符集和编码。 通常，将字符集保留为 1208；将编码保留为 Native。

IBM MQ 未转换字节数组。 要将字符串和字符数组编码为字节数组，请使用 java.nio.charset 软件包。 Charset 指定用于将字符串或字符数组转换为字节数组的字符集。 还可以使用 Charset，将字节数组解码为字符串或字符数组。 在编码字符串和字符数组时，最好不要依赖于 java.nio.charset.Charset.defaultCodePage。 通常，缺省的 Charset 为 windows-1252（在 Windows 上）；或 UTF-8（在 UNIX 上）。 windows-1252 是单字节字符集，UTF-8 是多字节字符集。

与其他 JMS 应用程序交换消息时，通常将目标字符集和编码属性保留为其缺省值 UTF-8 和 Native 。 如果您正在将包含数字或文本的消息与 JMS 应用程序交换，请选择适合于您的 JMSTextMessage、JMSStreamMessage、JMSMapMessage 或 JMSObjectMessage 消息类型中的某一个类型。 没有要完成的任何其他任务。

如果您正在与使用记录格式的非 JMS 应用程序交换消息，那么情况将更复杂。 除非整条记录包含文本，并且可以作为 JMSTextMessage 进行传输，否则必须在应用程序中对文本进行编码和解码。 将目标消息类型设置为 MQ，并使用 JMSBytesMessage 以避免 IBM MQ classes for JMS 将其他头和标记信息添加到消息数据中。 使用 JMSBytesMessage 方法来写入数字和字节，Charset 类会明确地将文本转换为字节数组。 影响选择字符集的因素可能有多个：

- 性能：是否能通过将文本转换为最大数目的服务器上使用的字符集来减少转换数?
- 统一性：在同一字符集中传输全部消息。
- 丰富性：哪些字符集具有应用程序必须使用的所有代码点?
- 简易性：与可变长度和多字节字符集相比，使用单字节字符集更简单。

请参阅第 149 页的『与非 JMS 应用程序交换格式化记录』。 以获取转换与非 JMS 应用程序交换的消息的示例。

示例

消息类型和转换类型表

表 *30:* 消息类型和转换类型

	转换类型			
消息类型	**文本**	**数值**	**其他**	**None**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT         /* wait for new messages            */
            | MQGMO_NO_SYNCPOINT /* no transaction                   */
            | MQGMO_CONVERT;     /* convert if necessary             */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle                */
           Hobj,                /* object handle                    */
           &md,                 /* message descriptor               */
           &gmo,                /* get message options              */
           buflen,              /* buffer length                    */
           buffer,              /* message buffer                   */
           &messlen,            /* message length                   */
           &CompCode,           /* completion code                  */
           &Reason);            /* reason code                      */
```

图 *13:* 来自 *amqsget0.c* 的代码片段

在 JMSBytesMessage 中发送和接收文本

第 135 页的图 14 中的代码在 BytesMessage 中发送字符串。 为简便起见，示例发送单个字符串，JMSTextMessage 对于其更为适用。 要在包含各种类型的字节消息中接收文本字符串，您必须知道字节中的字符串长度，在第 135 页的图 15 中称为 *TEXT_LENGTH*。 即使对于字符数固定的字符串，字节表示长度也可能会更长。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

图 *14:* 在 *JMSBytesMessage* 中发送 *String*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

图 *15:* 从 *JMSBytesMessage* 接收 *String*

相关概念

JMS 客户机消息转换和编码
以下列出了用于完成 JMS 客户机消息转换和编码的方法，其中包含每个类型的转换的代码示例。

队列管理器数据转换
系统会始终向从 JMS 客户机接收消息的非 JMS 应用程序提供队列管理器数据转换功能。 接收消息的 JMS 客户机还使用队列管理器数据转换，这是可选的。

相关任务

与非 JMS 应用程序交换格式化记录

遵循本任务中建议的步骤来设计和构建数据转换出口以及可以使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 可以在调用或不调用数据转换出口的情况下，与非 JMS 应用程序交换格式化消息。

相关参考

JMS 消息类型和转换
选择哪种消息类型会影响消息转换方法。 本部分为 JMS 消息类型 JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型交互内容。

JMS 消息类型和转换
选择哪种消息类型会影响消息转换方法。 本部分为 JMS 消息类型 JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型交互内容。

JMSObjectMessage

JMSObjectMessage 包含由 JVM 序列化到字节流的一个对象及其引用的所有对象。 文本会序列化到 UTF-8，并限制为不超过 65534 个字节的字符串或字符数组。 JMSObjectMessage 的一项优势是只要应用程序仅使用对象的方法和属性，就不会涉及任何数据转换问题。 JMSObjectMessage 在应用程序员不考虑如何对消息中的对象进行编码的情况下，为复杂对象提供数据转换。 使用 JMSObjectMessage 的缺点是它只能与其他 JMS 应用程序交换。 通过选择其他 JMS 消息类型中的某一类型，可以与非 JMS 应用程序交换 JMS 消息。

第 138 页的『发送和接收 JMSObjectMessage』显示消息中正在交换的 String 对象。

JMS 客户机应用程序只能在具有 JMS 样式主体的消息中接收 JMSObjectMessage 。 目标必须指定 JMS 样式主体。

JMSTextMessage

JMSTextMessage 包含单个文本字符串。 发送文本消息时，文本 Format 设置为 “MQSTR ”，WMQConstants.MQFMT_STRING。 文本的 CodedCharacterSetId 将设置为针对其目标定义的编码字符集标识。 文本由 IBM MQ 编码到 CodedCharacterSetId 中。 CodedCharacterSetId 和 Format 字段在消息描述符 MQMD 中设置，或在 MQRFH2 中的 JMS 字段中设置。 如果消息定义为具有 WMQ_MESSAGE_BODY_MQ 消息体样式，或未指定主体样式，但目标为 WMQ_TARGET_DEST_MQ，那么将设置消息描述符字段。 否则，消息将具有 JMS RFH2，并将在 MQRFH2 的固定部分中设置这些字段。

应用程序可覆盖为目标定义的编码字符集标识。 它必须将消息属性 JMS_IBM_CHARACTER_SET 设置为编码字符集标识；请参阅第 138 页的『发送和接收 JMSTextmessage』中的示例。

当 JMS 客户机调用 consumer.receive 方法时，队列管理器转换是可选功能。 通过将目标属性 WMQ_RECEIVE_CONVERSION 设置为 WMQ_RECEIVE_CONVERSION_QMGR，来启用队列管理器转换。 队列管理器会从为消息指定的 JMS_IBM_CHARACTER_SET 转换文本消息，然后将此消息传输到 JMS 客户机。 转换后消息的字符集为 1208, UTF-8，除非目标具有不同的 WMQ_RECEIVE_CCSID。 引用 JMSTextMessage 的消息中的 CodedCharacterSetId 将更新为目标字符集标识。 getText 方法会将文本从目标字符集解码为 Unicode；请参阅第 138 页的『发送和接收 JMSTextmessage』中的示例。

JMSTextMessage 可以在无 JMS MQRFH2 头的情况下，在 MQ 样式消息体中发送。 目标属性 WMQ_MESSAGE_BODY 和 WMQ_TARGET_DEST 的值可确定消息体样式，除非由应用程序覆盖。 应用程序可调用 destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) 或 destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)，来覆盖目标上设置的值。

如果通过将具有 MQ 样式体的 JMSTextMessage 发送到将 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_MQ 的目标来发送它，那么您将无法从同一目标接收它作为 JMSTextMessage。 所有从将 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_MQ 的目标接收的消息都接收为 JMSBytesMessage。 如果尝试接收消息为 JMSTextMessage，那么将导致发生异常 ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to javax.jms.TextMessage。

注: JMS 客户机未转换 JMSBytesMessage 中的文本。 该客户机只能作为字节数组接收消息中的文本。 如果启用队列管理器转换，那么队列管理器将转换文本，但 JMS 客户机必须仍在 JMSBytesMessage 中作为字节数组对其进行接收。

通常，最好使用 WMQ_TARGET_DEST 属性来控制是否使用 MQ 或 JMS 主体样式来发送 JMSTextMessage。 然后，可以从将 WMQ_TARGET_DEST 设置为 WMQ_TARGET_DEST_MQ 或 WMQ_TARGET_DEST_JMS 的目标接收消息。 WMQ_TARGET_DEST 对接收方无任何影响。

JMSMapMessage 和 JMSStreamMessage

这两个 JMS 消息类型非常类似。 可以使用基于 DataInputStream 和 DataOutputStream 接口的方法来读写原语类型到消息；请参阅第 141 页的『消息类型和转换类型表』。 第 142 页的『JMS 客户机消息转换和编码』中描述了详细信息。 每个原语均做了标记；请参阅第 129 页的『JMS 消息体』。

数字数据将读写到编码为 XML 文本的消息。 未引用目标属性 JMS_IBM_ENCODING。 将按与 JMSTextMessage 中的文本相同的方式处理文本数据。 如果要查看第 139 页的图 20 中的示例所创建的消息内容，那么所有消息数据都将采用 EBCDIC 格式，如同使用字符集值 37 发送一样。

可以在 JMSMapMessage 或 JMSStreamMessage 中发送多个项。

可以从 JMSMapMessage 按名称检索单个数据项，或从 JMSStreamMessage 按位置检索。 使用消息中存储的 CodedCharacterSetId 值调用获取或读取方法时，将对每一项进行解码。 如果用于检索项的方法返回的类型与发送的类型不同，那么表示已进行类型转换。 如果无法转换类型，将抛出异常。 请参阅类 JMSStreamMessage，以获取详细信息。 第 139 页的『在 JMSStreamMessage 和 JMSMapMessage 中发送数据』中的示例说明了类型转换以及如何不按顺序获取 JMSMapMessage 内容。

JMSMapMessage 和 JMSStreamMessage 的 MQRFH2.format 字段设置为 “MQSTR ”。 如果将目标属性 WMQ_RECEIVE_CONVERSION 设置为 WMQ_RECEIVE_CONVERSION_QMGR，那么队列管理器会转换消息数据，然后将其发送至 JMS 客户机。 消息的 MQRFH2.CodedCharacterSetId 是目标的 WMQ_RECEIVE_CCSID。 MQRFH2.Encoding 为 Native。 如果 WMQ_RECEIVE_CONVERSION 为 WMQ_RECEIVE_CONVERSION_CLIENT_MSG，那么 MQRFH2 的 CodedCharacterSetId 和 Encoding 为发送方设置的值。

JMS 客户机应用程序只能在具有 JMS 样式主体的消息中从未指定 MQ 样式主体的目标接收 JMSMapMessage 或 JMSStreamMessage 。

JMSBytesMessage

JMSBytesMessage 可包含多个原语类型。 可以使用基于 DataInputStream 和 DataOutputStream 接口的方法来读写原语类型到消息；请参阅第 141 页的『消息类型和转换类型表』。 第 136 页的『JMS 消息类型和转换』中描述了详细信息。

消息中数字数据的编码由在将数字数据写入 JMSBytesMessage 前设置的 JMS_IBM_ENCODING 值来控制。 应用程序可通过设置消息属性 JMS_IBM_ENCODING，来覆盖为 JMSBytesMessage 定义的缺省 Native 编码。

可使用 readUTF 和 writeUTF 以 UTF-8 格式读写文本数据，或使用 readChar 和 writeChar 方法以 Unicode 形式读写文本数据。 没有方法使用 CodedCharacterSetId。 或者，JMS 客户机可使用 Charset 类将文本编码和解码为字节。 它会在 IBM MQ classes for JMS 不执行任何转换的情况下，在 JVM 与消息间传输字节；请参阅第 139 页的『在 JMSBytesMessage 中发送和接收文本』。

发送到 MQ 应用程序的 JMSBytesMessage 通常在没有 JMS MQRFH2 头的 MQ 样式消息体中发送。 如果发送到 JMS 应用程序，那么消息体样式通常为 JMS。 目标属性 WMQ_MESSAGE_BODY 和 WMQ_TARGET_DEST 的值可确定消息体样式，除非由应用程序覆盖。 应用程序可调用 destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) 或 destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ)，来覆盖目标上设置的值。

如果发送具有 MQ 样式主体的 JMSBytesMessage，那么可从定义 MQ 或 JMS 消息体样式的目标接收消息。 如果发送具有 JMS 样式主体的 JMSBytesMessage，那么必须从定义 JMS 消息体样式的目标接收消息。 如果未接收，那么 MQRFH2 将视为用户消息数据的一部分，这可能不是您所期望的结果。

无论消息是否具有 MQ 或 JMS 主体样式，设置 WMQ_TARGET_DEST 都不会影响接收消息的方式。

如果为消息数据提供 Format，并且启用队列管理器数据转换，那么队列管理器可能会稍后转换消息。 请勿为除指定消息数据格式以外的其他任何操作使用格式字段，或将格式字段保留为空，MQConstants.MQFMT_NONE

可以在 JMSBytesMessage 中发送多个项。 使用为消息定义的编码发送消息时，将转换每一个数字项。

可以从 JMSBytesMessage 检索单个数据项。 以与创建消息时调用写方法相同的顺序调用读方法。 使用消息中存储的 Encoding 值调用消息时，将转换每一个数字项。

不同于 JMSMapMessage 和 JMSStreamMessage，JMSBytesMessage 仅包含应用程序写入的数据。 消息数据中未存储任何其他数据，如用于在 JMSMapMessage 和 JMSStreamMessage 中定义项的 XML 标记。 因此，请使用 JMSBytesMessage 来传输针对其他应用程序格式化的消息。

在某些应用程序中，在 JMSBytesMessage、DataInputStream 和 DataOutputStream 之间转换将非常有用。 基于示例第 140 页的『使用 DataInputStream 和 DataOutputStream 读写消息』的代码在将 com.ibm.mq.header 包与 JMS 一起使用时是必需的。

示例

发送和接收 JMSObjectMessage

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

图 *16:* 发送和接收 *JMSObjectMessage*

发送和接收 JMSTextmessage

文本消息无法包含不同的字符集中的文本。 该示例显示了位于不同的字符集中且在两条不同的消息中发送的文本。

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

图 *17:* 在目标定义的字符集中发送文本消息

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

图 *18:* 在 *ccsid 37* 中发送文本消息

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

图 *19:* 接收文本消息

在 JMSStreamMessage 和 JMSMapMessage 中发送数据

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                           " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                        " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

图 *20:* 在 *JMSStreamMessage* 和 *JMSMapMessage* 中发送数据

在 JMSBytesMessage 中发送和接收文本

第 139 页的图 21 中的代码在 BytesMessage 中发送字符串。 为简便起见，示例发送单个字符串，JMSTextMessage 对于其更为适用。 要在包含各种类型的字节消息中接收文本字符串，您必须知道字节中的字符串长度，在第 140 页的图 22 中称为 *TEXT_LENGTH*。 即使对于字符数固定的字符串，字节表示长度也可能会更长。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

图 *21:* 在 *JMSBytesMessage* 中发送 *String*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

图 *22:* 从 *JMSBytesMessage* 接收 *String*

使用 DataInputStream 和 DataOutputStream 读写消息

第 140 页的图 23 中的代码使用 DataOutputStream 创建 JMSBytesMessage。

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

图 *23:* 使用 *DataOutputStream* 发送 *JMSBytesMessage*

设置 JMS_IBM_ENCODING 属性的语句将注释掉。 如果直接写入 JMSBytesMessage，那么该语句有效，但在写入 DataOutputStream 时无任何作用。 写入 DataOutputStream 的数字将在 Native 编码中进行编码。 设置 JMS_IBM_ENCODING 无任何作用。

第 140 页的图 24 中的代码使用 DataInputStream 接收 JMSBytesMessage。

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

图 *24:* 使用 *DataInputStream* 接收 *JMSBytesMessage*

使用输入消息数据 JMS_IBM_CHARACTER_SET 的代码页属性打印代码页。 输入时，JMS_IBM_CHARACTER_SET 是 Java 代码页，不是数字编码字符集标识。

消息类型和转换类型表

表 *31:* 消息类型和转换类型

	转换类型			
消息类型	**文本**	**数值**	**其他**	**None**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

相关概念

JMS 消息转换方法

许多数据转换方法对于 JMS 应用程序设计人员都是开放的。 这些方法没有排他性，某些应用程序可能会使用这些方法的组合。 如果应用程序正在仅交换文本，或正在仅与其他 JMS 应用程序交换消息，那么您通常不会考虑数据转换。 IBM MQ 将为您自动执行数据转换。

JMS 客户机消息转换和编码

以下列出了用于完成 JMS 客户机消息转换和编码的方法，其中包含每个类型的转换的代码示例。

队列管理器数据转换
系统会始终向从 JMS 客户机接收消息的非 JMS 应用程序提供队列管理器数据转换功能。 接收消息的 JMS 客户机还使用队列管理器数据转换，这是可选的。

相关任务

与非 JMS 应用程序交换格式化记录
遵循本任务中建议的步骤来设计和构建数据转换出口以及可以使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 可以在调用或不调用数据转换出口的情况下，与非 JMS 应用程序交换格式化消息。

JMS 客户机消息转换和编码
以下列出了用于完成 JMS 客户机消息转换和编码的方法，其中包含每个类型的转换的代码示例。

当在 JMS 消息中读取或写入 Java 原语或对象时，将进行转换和编码。 此转换过程称为 JMS 客户机数据转换，以区别于队列管理器数据转换和应用程序数据转换。 转换仅在从 JMS 消息读取数据或将数据写入此消息时进行。 文本可与内部 16 位 Unicode 表示相互转换 [3] 为用于消息文本的字符集。 数字数据转换为 Java 原语数字类型及为消息定义的编码。 是否执行转换以及执行哪一类转换均取决于 JMS 消息类型及读写操作。

第 142 页的表 32 按执行的转换类型，针对不同的 JMS 消息类型对读写方法进行分类。 下表后的文本描述了转换类型。

表 *32:* 消息类型和转换类型

	转换类型			
消息类型	**文本**	**数值**	**其他**	**None**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

[3] 某些 Unicode 表示的长度必须超过 16 位。 请参阅 Java SE 参考。

表 32: 消息类型和转换类型 (继续)				
	转换类型			
消息类型	**文本**	**数值**	**其他**	**None**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

文本

目标的缺省 CodedCharacterSetId 是 1208，UTF-8。 缺省情况下，将从 Unicode 转换文本，并作为 UTF-8 文本字符串进行发送。 接收时，文本将从客户机接收的消息中的编码字符集转换为 Unicode。

setText 和 writeString 方法会将文本从 Unicode 转换为针对目标定义的字符集。 应用程序可通过设置消息属性 JMS_IBM_CHARACTER_SET，来覆盖目标字符集。 发送消息时，JMS_IBM_CHARACTER_SET 必须是数字编码的字符集标识。[4].

第 145 页的『发送和接收 JMSTextmessage』 中的代码片段可发送两条消息。 一条在为目标定义的字符集中发送，另一条在应用程序定义的字符集 37 中发送。

getText 和 readString 方法可将消息中的文本从消息中定义的字符集转换为 Unicode。 该方法使用消息属性 JMS_IBM_CHARACTER_SET 中定义的代码页。 除非消息为 MQ 型消息且不包含 MQRFH2，否则将从 MQRFH2.CodedCharacterSetId 映射代码页。 如果消息为 MQ 型消息且无 MQRFH2，那么将从 MQMD.CodedCharacterSetId 映射代码页。

第 146 页的图 29 中的代码片段可接收发送到目标的消息。 消息中的文本从代码页 IBM037 转换回 Unicode。

注: 检查是否已将文本转换为编码字符集 37 的一个简单的方法是使用 IBM MQ Explorer。 浏览队列并显示消息属性，然后进行检索。

对比第 145 页的图 28 中的代码片段与第 144 页的图 25 中不正确的代码片段。 在不正确的片段中，文本字符串会进行两次转换，一次由应用程序转换，另一次由 IBM MQ 转换。

[4] 接收消息时，JMS_IBM_CHARACTER_SET 是 Java Charset 代码页名称。

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

图 *25:* 不正确的代码页转换

writeUTF 方法可将文本从 Unicode 转换为 1208，UTF-8。 文本字符串以双字节长度开头。 文本字符串的最大长度是 65534 字节。 readUTF 方法可读取由 writeUTF 方法写入的消息中的项。 它会精确读取由 writeUTF 方法写入的字节数。

数值

目标的缺省数字编码为 Native。 Java 的 Native 编码常量具有值 273 x'00000111'，这对于所有平台都是相同的。 接收时，消息中的数字会正确地转换为数字 Java 原语。 转换过程使用消息中定义的编码以及由读取方法返回的类型。

发送方法可将由 set 和 write 添加到消息中的数字转换为针对目标定义的数字编码。 设置消息属性的应用程序可以为消息覆盖目标编码 JMS_IBM_ENCODING；例如：

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

get 和 read 数字方法可从消息中定义的数字编码转换消息中的数字。 它们会将数字转换为由 read 或 get 方法指定的类型；请参阅 ENCODING 属性。 这些方法使用 JMS_IBM_ENCODING 中定义的编码。 除非消息为 MQ 型消息且不包含 MQRFH2，否则将从 MQRFH2.Encoding 映射编码。 如果消息为 MQ 型消息且无 MQRFH2，那么这些方法将使用 MQMD.Encoding 中定义的编码。

第 146 页的图 30 中的示例显示了以目标格式对数字进行编码并在 JMSStreamMessage 中发送编码数字的应用程序。 将第 146 页的图 30 中的示例与第 146 页的图 31 中的示例进行比较。 差异在于必须在 JMSBytesMessage 中设置 JMS_IBM_ENCODING。

注: 检查是否对数字进行正确编码的一个简单的方法是使用 IBM MQ Explorer。 浏览队列并显示消息属性，然后进行使用。

其他

boolean 方法可以在 JMSByteMessage、JMSStreamMessage 和 JMSMapMessage 中，将 true 和 false 编码为 x'01' 和 x'00'。

UTF 方法可将 Unicode 编码和解码为 UTF-8 文本字符串。 字符串限于少于 65536 个字符，并以双字节长度字段开头。

Object 方法可将原语类型包装为对象。 系统会对数字和文本类型进行编码或转换，如同使用数字和文本方法读写原语类型一样。

None

readByte、readBytes、readUnsignedByte、writeByte 和 writeBytes 方法可在不进行转换的情况下，在应用程序与消息间获取或放置单个字节或字节数组。 readChar 和 writeChar 方法可在不进行转换的情况下，在应用程序与消息间获取和放置双字节 Unicode 字符。

应用程序可使用 readBytes 和 writeBytes 方法来执行其自身的代码点转换，如在第 146 页的『在 JMSBytesMessage 中发送和接收文本』中一样。

IBM MQ 未在客户机中执行任何代码页转换，因为消息是 JMSBytesMessage，并且使用了 readBytes 和 writeBytes 方法。 但是，如果字节表示文本，请确保应用程序使用的代码页与目标的编码字符集匹配。 消息可能重新由队列管理器转换出口进行转换。 另一种可能是，接收 JMS 客户机程序可能遵循使用消息中的 JMS_IBM_CHARACTER_SET 属性将表示消息文本的任何字节数组转换为字符串或字符的惯例。

在该示例中，客户机使用目标编码字符集进行转换：

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

或者，客户机可能已选择代码页，然后在消息的 JMS_IBM_CHARACTER_SET 属性中设置相应的编码字符集。 IBM MQ classes for Java 使用 JMS_IBM_CHARACTER_SET 来设置 MQRFH2 中的 JMS 属性中的 CodedCharacterSetId 字段，或在消息描述符中，设置 MQMD：

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(wMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

如果将字节数组写入 JMSStringMessage 或 JMSMapMessage，那么 IBM MQ classes for JMS 将不执行数据转换，因为字节将输入为十六进制数据，而不是 JMSStringMessage 和 JMSMapMessage 中的文本。

如果字节表示应用程序中的字符，那么您必须考虑要对消息读写哪些代码点。 第 145 页的图 26 中的代码遵循使用目标编码字符集的惯例。 如果使用缺省字符集为 JVM 创建字符串，那么字节内容将取决于平台。 Windows 上的 JVM 通常具有缺省 Charset windows-1252 和 UNIX， UTF-8。 在 Windows 与 UNIX 间进行交换，确实需要选择明确代码页来作为字节交换文本。

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

图 *26:* 在 *JMSStreamMessage* 中使用目标字符集写入表示字符串的字节

示例

发送和接收 JMSTextmessage

文本消息无法包含不同的字符集中的文本。 该示例显示了位于不同的字符集中且在两条不同的消息中发送的文本。

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

图 *27:* 在目标定义的字符集中发送文本消息

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

图 *28:* 在 *ccsid 37* 中发送文本消息

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

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

图 *29:* 接收文本消息

编码示例

这些事例显示在为目标定义的编码中发送的数字。 请注意，必须将 JMSBytesMessage 的 JMS_IBM_ENCODING 属性设置为针对目标指定的值。

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

图 *30:* 在 *JMSStreamMessage* 中使用目标编码发送数字

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

图 *31:* 在 *JMSBytesMessage* 中使用目标编码发送数字

在 JMSBytesMessage 中发送和接收文本

第 146 页的图 32 中的代码在 BytesMessage 中发送字符串。 为简便起见，示例发送单个字符串，JMSTextMessage 对于其更为适用。 要在包含各种类型的字节消息中接收文本字符串，您必须知道字节中的字符串长度，在第 147 页的图 33 中称为 *TEXT_LENGTH*。 即使对于字符数固定的字符串，字节表示长度也可能会更长。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

图 *32:* 在 *JMSBytesMessage* 中发送 *String*

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

图 *33:* 从 *JMSBytesMessage* 接收 *String*

相关概念

JMS 消息转换方法

许多数据转换方法对于 JMS 应用程序设计人员都是开放的。 这些方法没有排他性，某些应用程序可能会使用这些方法的组合。 如果应用程序正在仅交换文本，或正在仅与其他 JMS 应用程序交换消息，那么您通常不会考虑数据转换。 IBM MQ 将为您自动执行数据转换。

队列管理器数据转换

系统会始终向从 JMS 客户机接收消息的非 JMS 应用程序提供队列管理器数据转换功能。 接收消息的 JMS 客户机还使用队列管理器数据转换，这是可选的。

相关任务

与非 JMS 应用程序交换格式化记录

遵循本任务中建议的步骤来设计和构建数据转换出口以及可以使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 可以在调用或不调用数据转换出口的情况下，与非 JMS 应用程序交换格式化消息。

相关参考

JMS 消息类型和转换

选择哪种消息类型会影响消息转换方法。 本部分为 JMS 消息类型 JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型交互内容。

队列管理器数据转换

系统会始终向从 JMS 客户机接收消息的非 JMS 应用程序提供队列管理器数据转换功能。 接收消息的 JMS 客户机还使用队列管理器数据转换，这是可选的。

队列管理器可使用为消息数据设置的 CodedCharacterSetId、Encoding 和 Format 值，来转换消息数据中的字符和数字数据。 对于非 JMS 应用程序，可始终通过设置 GetMessageOption 和 GMO_CONVERT 来使用转换功能。

队列管理器能够转换发送到 JMS 客户机的消息。 通过将目标属性 WMQ_RECEIVE_CONVERSION 设置为 WMQ_RECEIVE_CONVERSION_QMGR 或 WMQ_RECEIVE_CONVERSION_CLIENT_MSG 来控制队列管理器转换。 应用程序可更改目标设置：

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

或者，

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

图 *34:* 启用队列管理器数据转换

JMS 客户机的队列管理器数据转换在客户机调用 consumer.receive 方法时进行。 缺省情况下，文本数据将转换为 UTF-8 (1208)。 后续读取和获取方法可对收到的 UTF-8 数据中的文本进行解码，并在其内部

Unicode 编码中创建 Java 文本原语。 UTF-8 不是源于队列管理器数据转换的唯一的目标字符集。 可通过设置 WMQ_RECEIVE_CCSID 目标属性来选择不同的 CCSID。

应用程序还可以更改目标设置，例如，将其设置为 437，DOS-US：

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

或者，

```
((MQDestination)destination).setReceiveCCSID(437);
```

图 *35:* 为队列管理器转换设置目标编码字符集

更改 WMQ_RECEIVE_CCSID 有专门的原因；所选 CCSID 与在 JVM 中创建的文本对象无任何差别。 但是，在一些平台上，某些 JVM 可能无法将消息中文本的 CCSID 转换为 Unicode。 该选项可使您为消息中向客户机提供的任何文本选择 CCSID。 一些 JMS 客户机平台在提供 UTF-8 形式的消息文本时遇到问题。

JMS 代码相当于第 148 页的图 36 中 C 代码中的粗体文本。

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

图 *36:* 来自 *amqsget0.c* 的代码片段

注:

仅对具有已知 IBM MQ 格式的消息数据执行队列管理器转换。 MQSTR 或 MQCIH 是预定义的已知格式的示例。 只要提供了数据转换出口，已知格式也可以是用户定义的格式。

构造为 JMSTextMessage、JMSMapMessage 和 JMSStreamMessage 的消息具有 MQSTR 格式，可由队列管理器进行转换。

相关概念

JMS 消息转换方法

许多数据转换方法对于 JMS 应用程序设计人员都是开放的。 这些方法没有排他性，某些应用程序可能会使用这些方法的组合。 如果应用程序正在仅交换文本，或正在仅与其他 JMS 应用程序交换消息，那么您通常不会考虑数据转换。 IBM MQ 将为您自动执行数据转换。

JMS 客户机消息转换和编码

以下列出了用于完成 JMS 客户机消息转换和编码的方法，其中包含每个类型的转换的代码示例。

第 894 页的『调用数据转换出口』

数据转换出口是用户编写的出口，用于在处理 MQGET 调用期间接收控制。

相关任务

与非 JMS 应用程序交换格式化记录
遵循本任务中建议的步骤来设计和构建数据转换出口以及可以使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 可以在调用或不调用数据转换出口的情况下，与非 JMS 应用程序交换格式化消息。

相关参考

JMS 消息类型和转换
选择哪种消息类型会影响消息转换方法。 本部分为 JMS 消息类型 JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型交互内容。

与非 *JMS* 应用程序交换格式化记录
遵循本任务中建议的步骤来设计和构建数据转换出口以及可以使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 可以在调用或不调用数据转换出口的情况下，与非 JMS 应用程序交换格式化消息。

开始之前

您可能可以设计更简单的解决方案，以使用 JMSTextMessage 与非 JMS 应用程序交换消息。 在遵循本任务中的步骤之前，要排除此可能性。

关于此任务

JMS 客户机能够更轻松地写入其是否未涉及对与其他 JMS 客户机交换的 JMS 消息进行格式化的详细信息。只要消息类型为 JMSTextMessage、JMSMapMessage、JMSStreamMessage 或 JMSObjectMessage，IBM MQ 便会考虑消息格式化的详细信息。 IBM MQ 可处理不同平台上代码页及数字编码之间的差异。

您可以使用这些消息类型来与非 JMS 应用程序交换消息。 为此，您必须了解 IBM MQ classes for JMS 如何构造这些消息。 您可能能够修改非 JMS 应用程序以解释消息；请参阅第 115 页的『将 JMS 消息映射到 IBM MQ 消息』。

使用这些消息类型中的某一个类型的一项优势是 JMS 客户机编程不依赖于其与之交换消息的应用程序类型。缺点是它可能需要对其他程序进行修改，并且您可能无法更改该程序。

一种备用方法是编写可处理现有消息格式的 JMS 客户机应用程序。 通常，现有消息采用固定格式，包含非格式化数据、文本和数字。 使用此任务中的步骤以及第 152 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』中的示例 JMS 客户机作为构建 JMS 客户机的起点，该客户机可以与非 JMS 应用程序交换格式化记录。

过程

1. 定义记录布局，或使用其中一个预定义的 IBM MQ 头类。

   要了解如何处理预定义的 IBM MQ 头，请参阅处理 IBM MQ 消息头。

   第 150 页的图 37 是一个用户定义的固定长度记录布局的示例，可由数据转换实用程序进行处理。
2. 创建数据转换出口。

   遵循编写数据转换出口程序中的指示信息，以编写数据转换出口。

   要尝试使用第 152 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』中的示例，请命名数据转换出口 MYRECORD。
3. 编写 Java 类以包含记录布局以及发送和接收记录。 您可以采用两种方法：

   - 编写类来读写包含记录的 JMSBytesMessage；请参阅第 152 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』。
   - 编写一个用于扩展 com.ibm.mq.header.Header 的类以定义记录的数据结构；请参阅为新头类型创建类。
4. 确定要在哪一个编码字符集中交换消息。

   请参阅选择消息转换方法：接收方成功完成操作。

5. 配置目标以交换 MQ 型消息（无 JMS MQRFH2 头）。

   必须配置发送和接收目标以交换 MQ 型消息。 可将同一目标同时用于发送和接收。

   应用程序可以覆盖目标消息体属性：

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   第 152 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』中的示例覆盖了目标消息体属性，确保已发送 MQ 样式消息。
6. 使用 JMS 和非 JMS 应用程序测试解决方案

   测试数据转换出口的有用的工具有：

   - amqsgetc0.c 样本程序对于以下测试很有用：测试是否正在接收由 JMS 客户机发送的消息。 请参阅建议的修改以使用 第 151 页的图 38 中的示例头 RECORD.h。 在修改后，amqsgetc0.c 将收到示例 JMS 客户机 TryMyRecord.java 所发送的消息；请参阅 第 152 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』。
   - 样本 IBM MQ 浏览程序 amqsbcg0.c 对于检查消息头的内容、JMS 头 MQRFH2 的内容以及消息内容很有用。
   - 先前在 SupportPac IH03 中提供的 **rfhutil** 程序允许捕获测试消息并将其存储在文件中，然后用于驱动消息流。 也可以读取输出消息并将其以各种格式显示。 格式包括两种类型的 XML 以及与 COBOL 副本匹配。 数据可以采用 EBCDIC 或 ASCII 格式。 可以在发送消息之前将 RFH2 头添加到消息。

   如果尝试使用已修改的 amqsgetc0.c 样本程序接收消息，并且收到原因码为 2080 的错误，请检查此消息是否具有 MQRFH2。 修改过程会假定已向未指定 MQRFH2 的目标发送消息。

示例

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                                MQLONG StructLength;
                                MQLONG Encoding;
                                MQLONG CodeCharSetId;
                                MQCHAR Format[8];
                                MQLONG Flags;
                                MQCHAR RecordData[32];
        };
```

图 *37: RECORD.h*

- 声明 RECORD.h 数据结构

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- 修改 MQGET 调用以使用 RECORD,

  1. 修改前:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. 修改后:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- 更改输出语句,

  1. 从:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. 到:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

图 *38:* 修改 *amqsget0.c*

相关概念

JMS 消息转换方法

许多数据转换方法对于 JMS 应用程序设计人员都是开放的。 这些方法没有排他性，某些应用程序可能会使用这些方法的组合。 如果应用程序正在仅交换文本，或正在仅与其他 JMS 应用程序交换消息，那么您通常不会考虑数据转换。 IBM MQ 将为您自动执行数据转换。

JMS 客户机消息转换和编码

以下列出了用于完成 JMS 客户机消息转换和编码的方法，其中包含每个类型的转换的代码示例。

队列管理器数据转换
系统会始终向从 JMS 客户机接收消息的非 JMS 应用程序提供队列管理器数据转换功能。 接收消息的 JMS 客户机还使用队列管理器数据转换，这是可选的。

相关参考

JMS 消息类型和转换
选择哪种消息类型会影响消息转换方法。 本部分为 JMS 消息类型 JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型交互内容。

创建转换出口代码的实用程序

编写用于在 *JMSBytesMessage* 中封装记录布局的类
例如，此任务的目的在于探索如何在 JMSBytesMessage 中组合使用数据转换的固定记录布局。 在此任务中，您将创建一些 Java 类以交换 JMSBytesMessage 中的示例记录结构。 您可以修改此示例以编写可交换其他记录结构的类。

JMSBytesMessage 是与非 JMS 程序交换混合数据类型记录的最佳 JMS 消息类型。 JMS 提供程序没有在这种类型的消息体中插入任何额外数据。 因此，如果 JMS 客户机程序要与现有 IBM MQ 程序互操作，最好使用此消息类型。 使用 JMSBytesMessage 的主要难题是匹配编码和其他程序所需的字符集。 解决方案是创建可封装记录的类。 为特定记录类型封装读取和编写 JMSBytesMessage 的类可以简化在 JMS 程序中发送和接收固定格式的记录。 通过捕获抽象类中接口的通用特征，解决方案的大部分可以重复用于其他记录格式。 其他记录格式可以在扩展了抽象通用类的类中实现。

另一种方法是扩展 com.ibm.mq.headers.Header 类。 Header 类含有诸如 addMQLONG 之类的方法，可以通过更声明式的方式构建记录格式。 使用 Header 类的缺点是获取和设置属性需要使用更复杂的解释性接口。 两种方式会产生相同数量的应用程序代码。

除非每条记录都使用相同的格式、编码字符集和编码，否则，除了 MQRFH2 之外，JMSBytesMessage 在一个消息中只能封装一种格式。 JMSBytesMessage 的格式、编码和字符集是 MQRFH2 后所有消息的属性。 该示例是在 JMSBytesMessage 只包含一条用户记录的前提下编写的。

开始之前

1. 技能等级：您必须熟悉 Java 编程和 JMS。 不提供有关设置 Java 开发环境的指示信息。 对于编写过交换 JMSTextMessage、JMSStreamMessage 或 JMSMapMessage 的开发人员有优势。 您可以看到使用 JMSBytesMessage 交换消息的区别。
2. 该示例需使用 IBM WebSphere MQ 7.0。
3. 该示例是使用 Eclipse 工作台的 Java 透视图创建的。 需要使用 JRE 6.0 或更高版本。 您可以使用 IBM MQ Explorer 中的 Java 透视图来开发和运行 Java 类。 或者，您也可以使用您自己的 Java 开发环境。
4. 相比于命令行实用程序，使用 IBM MQ Explorer 设置测试环境并进行调试更为简单。

关于此任务

系统将指导您完成 RECORD 和 MyRecord 这两个类的创建。 这两个类结合使用可以封装固定格式的记录。 它们包含用于获取和设置属性的方法。 GET 方法可从 JMSBytesMessage 读取记录，PUT 方法可将记录写入 JMSBytesMessage。

此任务的目的不是创建可以复用的生产质量类。 您可以选择使用此任务中的示例开始创建自己的类。 此任务的目的是为您提供指导说明，主要是关于使用 JMSBytesMessage 时要使用的字符集、格式和编码。 创建类的每个步骤都进行了解释，同时描述了有时会被忽略的 JMSBytesMessage 的用法。

RECORD 类是抽象类，定义一些用户记录的通用字段。 通用字段在具有醒目内容、版本和长度字段的标准 IBM MQ 头布局上建模。 将忽略很多 IBM MQ 头上找到的编码、字符集和格式字段。 其他头不能位于用户定义的格式后面。 扩展自 RECORD 的 MyRecord 类通过使用额外的用户字段按字面扩展记录来实现扩展。 类创建的 JMSBytesMessage 可以由队列管理器数据转换出口处理。

第 158 页的『用于运行示例的类』中包含 RECORD 和 MyRecord 的完整列表。 它还包括额外 “scaffolding” 类的列表以测试 RECORD 和 MyRecord。 额外的类包括：

TryMyRecord
用于测试 RECORD 和 MyRecord 的主要程序。

EndPoint
用于将 JMS 连接、目标和会话封装在一个类中的抽象类。 其接口只满足测试 RECORD 和 MyRecord 类的需要。 并不是为编写 JMS 应用程序而建立的设计模式。

注: 在创建目标后，Endpoint 类包含以下代码行：

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

在 V7.0 中，从 V7.0.1.5 开始，需要打开队列管理器转换。 缺省情况下，此选项处于禁用状态。 在 V7.0 中，直到 V7.0.1.4 为止，队列管理器转换在缺省情况下处于禁用状态，而此代码行会导致错误。

MyProducer 和 MyConsumer
可扩展 EndPoint 的类，用于创建 MessageConsumer 和 MessageProducer，已连接并准备好接受请求。

以上所有类一起组成一个完整的应用程序，您可以构建和试验此应用程序以理解如何在 JMSBytesMessage 中使用数据转换。

过程

1. 创建一个抽象类以通过缺省构造函数将标准字段封装在 IBM MQ 头中。 然后扩展此类以根据您的需求对头进行调整。

```
public abstract class RECORD implements Serializable {
   private static final long serialVersionUID = -1616617232750561712L;
   protected final static int UTF8 = 1208;
   protected final static int MQLONG_LENGTH = 4;
   protected final static int RECORD_STRUCT_ID_LENGTH = 4;
   protected final static int RECORD_VERSION_1 = 1;
   protected final String RECORD_STRUCT_ID = "BLNK";
   protected final String RECORD_TYPE = "BLANK   ";
   private String structID = RECORD_STRUCT_ID;
   private int version = RECORD_VERSION_1;
   private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
   private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
   private String headerCharset = "UTF-8";
   private String headerFormat = RECORD_TYPE;

   public RECORD() {
       super();
   }
```

注:

a. 属性（structID 到 nextFormat）按照它们在标准 IBM MQ 消息头中的布局顺序列出。

b. 属性 format、messageEncoding 和 messageCharset 描述头本身，不属于头的一部分。

c. 您必须决定是存储记录的编码字符集标识还是字符集。 Java 使用字符集，而 IBM MQ 消息使用编码字符集标识。 示例代码使用字符集。

d. int 由 IBM MQ 序列化为 MQLONG 。 MQLONG 是 4 个字节。

2. 为专有属性创建 getter 方法和 setter 方法。

a) 创建或生成 getter 方法：

```
    public String getHeaderFormat() { return headerFormat; }
   public int getHeaderEncoding() { return headerEncoding;    }
   public String getMessageCharset() { return headerCharset; }
   public int getMessageEncoding() { return headerEncoding; }
   public String getStructID() { return structID; }
   public int getStructLength() { return structLength; }
   public int getVersion() { return version; }
```

b) 创建或生成 setter 方法：

```
    public void setHeaderCharset(String charset) {
       this.headerCharset = charset; }
   public void setHeaderEncoding(int encoding) {
       this.headerEncoding = encoding; }
   public void setHeaderFormat(String headerFormat) {
       this.headerFormat = headerFormat; }
   public void setStructID(String structID) {
       this.structID = structID; }
   public void setStructLength(int structLength) {
       this.structLength = structLength; }
   public void setVersion(int version) {
       this.version = version; }
}
```

3. 创建构造函数以通过 JMSBytesMessage 创建 RECORD 实例。

```
public RECORD(BytesMessage message) throws JMSException, IOException,
       MQDataException {
   super();
   setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
   setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
   byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
   message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
   setStructID(new String(structID, getMessageCharset()));
   setVersion(message.readInt());
   setStructLength(message.readInt());
}
```

注:

a. messageCharset 和 messageEncoding 在覆盖目标的值设置时捕获自消息属性。 不会更新 format。 该示例不执行错误检查。 如果调用 Record(BytesMessage) 构造函数，将假设 JMSBytesMessage 为 RECORD 类型消息。 行 “setStructID(new String(structID, getMessageCharset()))” 将设置一些醒目内容。

b. 完成此方法的代码行按照在 RECORD 实例中更新缺省值的顺序反序列化消息中的字段。

4. 创建 PUT 方法以将头字段写入 JMSBytesMessage。

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
       JMSException, UnsupportedEncodingException {
   setHeaderEncoding(myProducer.getEncoding());
   setHeaderCharset(myProducer.getCharset());
   myProducer.setMQClient(true);
   BytesMessage bytes = myProducer.session.createBytesMessage();
   bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
   bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
   bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                        myProducer.getCCSID());
   bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                    + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                    .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
   bytes.writeInt(getVersion());
   bytes.writeInt(getStructLength());
   return bytes;
}
```

注:

a. MyProducer 将 JMS Connection、Destination、Session 和 MessageProducer 封装在一个类中。 稍后用到的 MyConsumer 将 JMS Connection、Destination、Session 和 MessageConsumer 封装在一个类中。

b. 对于 JMSBytesMessage，如果编码不是 Native，那么必须在消息中设置编码。 目标编码将复制到消息编码属性 JMS_IBM_CHARACTER_SET 中并保存为 RECORD 类的属性。

i) “setMessageEncoding(myProducer.getEncoding());” 调用 “(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));” 以获取目标编码。

ii) “Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());” 将设置消息编码。

c. 用于将文本转换为字节的字符集从目标中获取并保存为 RECORD 类的属性。 未在消息中设置此值，因为在编写 JMSBytesMessage 时 IBM MQ classes for JMS 不会使用此值。

“messageCharset = myProducer.getCharset();” calls

```
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());
}
```

它将从编码字符集标识中获取 Java 字符集。

“ CCSID.getCodepage(ccsid)” 位于包 com.ibm.mq.headers 中。 ccsid 从用于查询目标的 MyProducer 的另一个方法中获取：

```
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));
    }
```

d. “myProducer.setMQClient(true);” 将覆盖客户机类型的目标设置，并将其强制为 IBM MQ MQI client。 您可能会选择忽略此代码行，因为此代码行隐藏管理配置错误。

“myProducer.setMQClient(true);” 调用：

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

该代码具有副作用，即会将 IBM MQ 正文样式设置为不明确（如果必须覆盖 JMS 的设置）。

注：

IBM MQ classes for JMS 将消息的格式、编码和字符集标识写入消息描述符 MQMD 或写入 JMS 头 MQRFH2。 这取决于消息是否具有 IBM MQ 样式正文。 请不要手动设置 MQMD 字段。

存在手动设置消息描述符属性的方法。 该方法使用 JMS_IBM_MQMD_* 属性。 您必须设置目标属性 WMQ_MQMD_WRITE_ENABLED 才能设置 JMS_IBM_MQMD_* 属性：

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

您必须设置目标属性 WMQ_MQMD_READ_ENABLED 才能读取属性。

如果对整个消息有效内容有完全控制权，只需使用 JMS_IBM_MQMD_*。 与 JMS_IBM_* 属性不同，JMS_IBM_MQMD_* 属性不控制 IBM MQ classes for JMS 构造 JMS 消息的方式。 可以创建与 JMS 消息的属性冲突的消息描述符属性。

e. 完成此方法的代码行按照消息中的字段顺序序列化类中的属性。

字符串属性将以空白填补。 这些字符串将转换为使用为记录定义的字符集的字节，并截断为消息字段的长度。

5. 通过添加导入完成类。

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
```

```
import com.ibm.msg.client.wmq.WMQConstants;
```

6. 创建一个类来扩展 RECORD 类以包含其他字段。 包含缺省构造函数。

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

   public MyRecord() {
       super();
       super.setStructID(STRUCT_ID);
       super.setHeaderFormat(FORMAT);
       super.setStructLength(super.getStructLength() + MQLONG_LENGTH
               + DATA_LENGTH);
   }
```

注:

a. RECORD 子类 MyRecord 可定制头的醒目内容、格式和长度。

7. 创建或生成 getter 方法和 setter 方法。

a) 创建 getter 方法:

```
    public int getFlags() { return flags; }
   public String getRecordData() { return recordData; } .
```

b) 创建 setter 方法:

```
    public void setFlags(int flags) {
       this.flags = flags; }
   public void setRecordData(String recordData) {
       this.recordData = recordData; }
}
```

8. 创建构造函数以通过 JMSBytesMessage 创建 MyRecord 实例。

```
    public MyRecord(BytesMessage message) throws JMSException, IOException,
           MQDataException {
       super(message);
       setFlags(message.readInt());
       byte[] recordData = new byte[DATA_LENGTH];
       message.readBytes(recordData, DATA_LENGTH);
       setRecordData(new String(recordData, super.getMessageCharset()));
   }
```

注:

a. RECORD 类首先读取组成标准消息模板的字段。

b. recordData 文本将转换为使用消息的字符集属性的 String。

9. 创建静态方法以从使用者获取消息并创建新 MyRecord 实例。

```
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
           MQDataException, IOException {
       BytesMessage message = (BytesMessage) myConsumer.receive();
       return new MyRecord(message);
   }
```

注:

a. 在此示例中，为了简便起见，将从静态 GET 方法调用 MyRecord(BytesMessage) 构造函数。 通常，您可以将接收消息与新建 MyRecord 实例分开。

10. 创建 PUT 方法以将客户字段附加到包含消息头的 JMSBytesMessage。

```
 public BytesMessage put(MyProducer myProducer) throws JMSException,
        IOException {
    BytesMessage bytes = super.put(myProducer);
    bytes.writeInt(getFlags());
    bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                    + DATA_LENGTH + "s",getRecordData())
                    .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
    myProducer.send(bytes);
    return bytes;
}
```

注:

a. 代码中调用的方法按照消息中的字段顺序序列化 MyRecord 类中的属性。

- recordData String 将填充为空白，转换为使用为记录定义的字符集的字节，并截断为 RecordData 字段的长度。

11. 通过添加 include 语句完成类。

```
 package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

结果

结果:

- 运行 TryMyRecord 类的结果:
  - 以编码字符集 37 发送消息，并使用队列管理器转换出口:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - 以编码字符集 37 发送消息，不使用队列管理器转换出口:

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- 将 TryMyRecord 类修改为不接收消息，而是使用修改后的 amqsget0.c 样本接收消息的结果。 修改后的样本接受格式化记录；请参阅第 149 页的『与非 JMS 应用程序交换格式化记录』中的第 151 页的图 38。
  - 以编码字符集 37 发送消息，并使用队列管理器转换出口:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - 以编码字符集 37 发送消息，不使用队列管理器转换出口:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
```

```
no more messages
Sample AMQSGET0 end
```

试用示例并用其他代码页和数据转换出口进行试验。 创建 Java 类，配置 IBM MQ 并运行主程序 TryMyRecord；请参阅第 158 页的图 39。

1. 配置 IBM MQ 和 JMS 以运行示例。 这些指示信息用于在 Windows 上运行示例。

   a. 创建队列管理器

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. 创建队列

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. Create a JNDI directory

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. 切换到 JMS bin 目录

   JMS 管理程序必须从此处运行。 路径为 *MQ_INSTALLATION_PATH*\java\bin。

   e. 在名为 JMSQM1Q1.txt 的文件中创建以下 JMS 定义

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. 运行 JMSAdmin 程序以创建 JMS 资源

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. 您可以创建、变更和浏览已经使用 IBM MQ Explorer 创建的定义。
3. 运行 TryMyRecord。

用于运行示例的类

压缩文件中提供了图 第 158 页的图 39 到 第 162 页的图 44 中列出的类；下载 jm25529_.zip 或 jm25529_.tar.gz。

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

图 *39: TryMyRecord*

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

图 *40: RECORD*

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

图 *41: MyRecord*

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
             JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
              == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                  == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                  == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

图 *42: EndPoint*

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

图 *43: MyProducer*

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

图 *44: MyConsumer*

在 *IBM MQ classes for JMS* 应用程序中创建并配置连接工厂和目标

IBM MQ classes for JMS 应用程序可以通过从 Java 命名和目录接口 (JNDI) 名称空间中检索受管对象，使用 IBM JMS 扩展或使用 IBM MQ JMS 扩展来创建连接工厂和目标。 应用程序还可以使用 IBM JMS 扩展或 IBM MQ JMS 扩展来设置连接工厂和目标的属性。

连接工厂和目标是 JMS 应用程序逻辑流中的起点。 应用程序使用 ConnectionFactory 对象创建与消息传递服务器的连接，使用 Queue 或 Topic 对象作为消息发送到的目标或作为从中接收消息的源。 因此，应用程序至少需要创建一个连接工厂以及一个或多个目标。 创建连接工厂或目标后，应用程序可能需要通过设置对象的一个或多个属性来对其进行配置。

总之，应用程序可以通过以下方式创建和配置连接工厂和目标：

使用 JNDI 检索受管对象

管理员可以使用 IBM MQ JMS 管理工具 (如 使用 JMS 管理工具配置对象中所述) 或 IBM MQ Explorer (如 使用 IBM MQ Explorer 配置 JMS 对象中所述) 来创建连接工厂和目标并将其配置为 JNDI 名称空间中的受管对象。 然后，应用程序便可以从 JNDI 名称空间检索这些受管对象。 检索受管对象之后，如果需要，应用程序可以使用 IBM JMS 扩展或 IBM MQ JMS 扩展来设置或更改对象的一个或多个属性。

使用 IBM JMS 扩展

应用程序可以使用 IBM JMS 扩展在运行时动态创建连接工厂和目标。 应用程序首先创建 JmsFactoryFactory 对象，然后使用此对象的方法创建连接工厂和目标。 创建连接工厂或目标后，应用程序可以使用从 JmsPropertyContext 接口继承的方法设置此对象的属性。 或者，应用程序可以在创建目标时使用统一资源标识 (URI) 指定一个或多个目标属性。

使用 IBM MQ JMS 扩展

应用程序还可以使用 IBM MQ JMS 扩展在运行时动态创建连接工厂和目标。 此应用程序使用提供的构造函数创建连接工厂和目标。 创建连接工厂或目标后，应用程序可以使用该对象的方法设置其属性。 或者，应用程序可以在创建目标时使用 URI 指定一个或多个目标属性。

相关任务

配置 JMS 资源

使用 *JNDI* 检索 *JMS* 应用程序中的受管对象

要从 Java 命名和目录接口 (JNDI) 名称空间中检索受管对象，JMS 应用程序必须创建初始上下文，然后使用 lookup() 方法检索对象。

管理员必须先创建受管对象，然后应用程序才能从 JNDI 名称空间检索受管对象。 管理员可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 在 JNDI 名称空间中创建和维护受管对象。 有关更多信息，请参阅配置 JNDI 名称空间中的连接工厂和目标。

应用程序服务器通常为受管对象提供自己的存储库，并提供用于创建和维护这些对象的自有工具。

如以下示例所示，应用程序必须先创建初始上下文，然后才能从 JNDI 名称空间检索受管对象。

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

在以上代码中，字符串变量 `url` 和 `icf` 具有以下含义：

url

目录服务的统一资源定位符 (URL)。 URL 可以是以下某种格式：

- `ldap://`*`hostname`*`/`*`contextName`* （对于基于 LDAP 服务器的目录服务）
- `file:/`*`directoryPath`* （对于基于本地文件系统的目录服务）

icf

初始上下文工厂的类名，可以具有以下值之一：

- `com.sun.jndi.ldap.LdapCtxFactory`（对于基于 LDAP 服务器的目录服务）
- `com.sun.jndi.fscontext.RefFSContextFactory`（对于基于本地文件系统的目录服务）

请注意，JNDI 包和轻量级目录访问协议 (LDAP) 服务提供程序的某些组合可能会导致发生 LDAP 错误 84。 为了解决此问题，请在调用 InitialDirContext() 前插入以下代码行：

```
environment.put(Context.REFERRAL, "throw");
```

如以下示例所示，获取初始上下文之后，应用程序可以使用 lookup() 方法从 JNDI 名称空间检索受管对象。

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

以上代码可从基于 LDAP 的名称空间中检索以下对象：

- 与名称 myCF 绑定的 ConnectionFactory 对象
- 与名称 myQ 绑定的 Queue 对象
- 与名称 myT 绑定的 Topic 对象

有关使用 JNDI 的更多信息，请参阅 Oracle 公司提供的 JNDI 文档。

相关任务

使用 IBM MQ Explorer 配置 JMS 对象

使用管理工具配置 JMS 对象

在 WebSphere Application Server 中配置 JMS 资源

使用 *IBM JMS* 扩展

IBM MQ classes for JMS 中包含一组对 JMS API 的扩展，称为 IBM JMS 扩展。 应用程序可以使用这些扩展在运行时动态创建连接工厂和目标，并可以设置 IBM MQ classes for JMS 对象的属性。 这些扩展可与任何消息传递提供者配合使用。

IBM JMS 扩展是以下包中的一组接口和类：

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

可以在位于 *MQ_INSTALLATION_PATH*/java/lib 中的 com.ibm.mqjms.jar 中找到这些软件包。

这些扩展提供以下功能：

- 一种基于工厂的机制，用于在运行时动态创建连接工厂和目标，而不是作为受管对象从 Java 命名和目录接口 (JNDI) 名称空间中检索连接工厂和目标。
- 一组用于设置 IBM MQ classes for JMS 对象属性的方法
- 一组异常类，包含的方法可用于获取关于问题的详细信息
- 一组用于控制跟踪的方法
- 一组用于获取有关 IBM MQ classes for JMS 的版本信息的方法

对于在运行时动态创建连接工厂和目标以及设置和获取其属性，IBM JMS 扩展为 IBM MQ JMS 扩展提供了一组替代接口。 但是，虽然 IBM MQ JMS 扩展特定于 IBM MQ 消息传递提供程序，但 IBM JMS 扩展并不特定于 IBM MQ，并且可以与 JMS 体系结构的 IBM MQ 类中描述的分层体系结构中的任何消息传递提供程序配合使用。

接口 com.ibm.msg.client.wmq.WMQConstants 包含常量的定义，应用程序可以在使用 IBM JMS 扩展设置 IBM MQ classes for JMS 对象的属性时使用这些定义。 此接口包含用于 IBM MQ 消息传递提供程序的常量和独立于任何消息传递提供程序的 JMS 常量。

后面的示例代码假设已运行以下 import 语句：

```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

创建连接工厂和目标

应用程序必须先创建 JmsFactoryFactory 对象，然后才能使用 IBM JMS 扩展创建连接工厂和目标。 如以下示例所示，要创建 JmsFactoryFactory 对象，应用程序需调用 JmsFactoryFactory 类的 getInstance() 方法：

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

getInstance() 调用上的参数是一个常量，用于将 IBM MQ 消息传递提供程序识别为所选消息传递提供程序。 然后，应用程序便可以使用 JmsFactoryFactory 对象创建连接工厂和目标。

如以下示例所示，要创建连接工厂，应用程序需调用 JmsFactoryFactory 对象的 createConnectionFactory() 方法：

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

该语句可创建所有属性均为缺省值的 JmsConnectionFactory 对象，这表示应用程序将以绑定方式连接到缺省队列管理器。 如果您希望在客户机模式下连接应用程序或连接到缺省队列管理器之外的队列管理器，那么在创建连接之前，应用程序必须设置 JmsConnectionFactory 对象的正确属性。 有关如何执行此操作的信息，请参阅第 165 页的『设置 IBM MQ classes for JMS 对象的属性』。

JmsFactoryFactory 类还包含用于创建以下类型的连接工厂的方法：

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

如以下示例所示，要创建 Queue 对象，应用程序需调用 JmsFactoryFactory 对象的 createQueue() 方法：

```
JmsQueue q1 = ff.createQueue("Q1");
```

该语句可创建所有属性均为缺省值的 JmsQueue 对象。 该对象表示名为 Q1 的 IBM MQ 队列，此队列属于本地队列管理器。 此队列可以是本地队列、别名队列或远程队列定义。

createQueue() 方法还可以接受队列统一资源标识 (URI) 作为参数。 队列 URI 是一个字符串，用于指定 IBM MQ 队列的名称，（可选）拥有此队列的队列管理器的名称以及 JmsQueue 对象的一个或多个属性。 以下语句包含队列 URI 示例：

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

此语句创建的 JmsQueue 对象表示名为 Q2 的 IBM MQ 队列（由队列管理器 QM2 拥有），发送至此目标的所有消息都是持久消息，并且其优先级为 5。 有关队列 URI 的更多信息，请参阅 第 175 页的『统一资源标识 (URI)』。 有关设置 JmsQueue 对象属性的替代方法，请参阅第 165 页的『设置 IBM MQ classes for JMS 对象的属性』。

如以下示例所示，要创建 Topic 对象，应用程序可以使用 JmsFactoryFactory 对象的 createTopic() 方法：

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

该语句可创建所有属性均为缺省值的 JmsTopic 对象。 该对象表示名为 Sport/Football/Results 的主题。

createTopic() 方法还可以接受主题 URI 作为参数。 主题 URI 是一个字符串，用于指定主题的名称和（可选）JmsTopic 对象的一个或多个属性。 以下语句包含主题 URI 示例：

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

这些语句创建的 JmsTopic 对象表示名为 Sport/Tennis/Results 的主题，发送至此目标的所有消息都是非持久消息，并且其优先级为 0。 有关主题 URI 的更多信息，请参阅 第 175 页的『统一资源标识 (URI)』。 有关设置 JmsTopic 对象属性的替代方法，请参阅第 165 页的『设置 IBM MQ classes for JMS 对象的属性』。

应用程序创建连接工厂或目标后，该对象只能与所选消息传递提供者一起使用。

设置 IBM MQ classes for JMS 对象的属性

要使用 IBM JMS 扩展来设置 IBM MQ classes for JMS 对象的属性，应用程序将使用 com.ibm.msg.client.JmsPropertyContext 接口的方法。

对于每种 Java 数据类型，JmsPropertyContext 接口包含一个用于设置此数据类型的属性值的方法以及一个获取该数据类型的属性值的方法。 例如，应用程序调用 setIntProperty() 方法设置带有整数值的属性，调用 getIntProperty() 方法获取带有整数值的属性。

com.ibm.mq.jms 包中的类实例还继承了 JmsPropertyContext 接口的方法。 因此，应用程序可以使用这些方法来设置 MQConnectionFactory、MQQueue 和 MQTopic 对象的属性。

在应用程序创建 IBM MQ classes for JMS 对象时，将自动设置带有缺省值的属性。 当应用程序设置属性时，新值将替换属性之前具有的任何值。 属性一旦设置便无法删除，但是其值可以更改。

如果应用程序尝试将属性设置为对于该属性来说无效的值，IBM MQ classes for JMS 将抛出异常。 如果应用程序尝试获取尚未设置的属性，此行为将如 JMS 规范中所描述。 IBM MQ classes for JMS 对原始数据类型抛出 NumberFormatException 异常并对引用的数据类型返回 null。

除了 IBM MQ classes for JMS 对象的预定义属性外，应用程序可以设置自己的属性。 这些应用程序定义的属性将被 IBM MQ classes for JMS 忽略。

有关 IBM MQ classes for JMS 对象属性的更多信息，请参阅 IBM MQ classes for JMS 对象的属性。

以下代码是如何使用 IBM JMS 扩展设置属性的示例。 以下代码设置了连接工厂的五个属性。

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

设置这些属性的效果是应用程序将使用名为 QM1.SVR 的 MQI 通道以客户机模式连接到队列管理器 QM1。 队列管理器在主机名为 HOST1 的系统上运行，而队列管理器的侦听器在端口号 1415 上侦听。 此连接和与该连接下的会话关联的其他队列管理器连接具有与这些连接关联的应用程序名称“My Application”。

注: z/OS 平台上运行的队列管理器不支持设置应用程序名称，因此该设置将被忽略。

JmsPropertyContext 接口还包含 setObjectProperty() 方法，应用程序可以使用此方法设置属性。 该方法的第二个参数是用于封装属性值的对象。 例如，以下代码创建用于封装整数 1415 的 Integer 对象，然后调用 setObjectProperty() 将连接工厂的 PORT 属性设置为值 1415：

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

因此，此代码等同于以下语句：

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

反之，getObjectProperty() 方法返回用于封装属性值的对象。

属性值从一种数据类型到另一种数据类型的隐式转换

当应用程序使用 JmsPropertyContext 接口的方法设置或获取 IBM MQ classes for JMS 对象的属性时，属性值可以从一种数据类型隐式转换为另一个数据类型。

例如，以下语句设置 JmsQueue object q1 的 PRIORITY 属性：

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

PRIORITY 属性具有整数值，因此 setStringProperty() 调用隐式地将字符串 “5”（源值）转换为整数 5（目标值），这将成为 PRIORITY 属性的值。

反之，以下语句获取 JmsQueue object q1 的 PRIORITY 属性：

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

整数 5（源值）是 PRIORITY 属性的值，通过 getStringProperty() 调用隐式地转换为字符串 “5”（目标值）。

IBM MQ classes for JMS 支持的转换显示在 第 167 页的表 33 中。

表 *33:* 受支持的从一种数据类型到另一种数据类型的转换	
源数据类型	**受支持的目标数据类型**
布尔值	字符串
byte	整型、长整型、短整型、字符串
char	字符串
双精度值	字符串
浮点值	双精度值、字符串
int	长整型、字符串
long	字符串
短整型	整型、长整型、字符串
字符串	布尔值、字节、双精度值、浮点值、整型、长整型、短整型

用于管理受支持转换的一般规则如下:

- 数字值可以从一种数据类型转换为另一种数据类型，前提是转换过程中无任何数据丢失。 例如，数据类型为 `int` 的值可以转换为数据类型为 `long` 的值，但不能转换为数据类型为 `short` 的值。
- 任何数据类型的值都可以转换为字符串。
- 可以将字符串转换为任何其他数据类型的值 (`char` 除外) 前提是字符串采用正确的格式进行转换。 如果应用程序尝试转换格式不正确的字符串，IBM MQ classes for JMS 会抛出 umberFormatException 异常。
- 如果应用程序尝试执行不支持的转换，IBM MQ classes for JMS 会抛出 MessageFormatException 异常。

将值从一种数据类型转换为另一种数据类型的特定规则如下所示:

- 将布尔值转换为字符串时，会将值 `true` 转换为字符串 “true”，并将值 `false` 转换为字符串 “false”。
- 将字符串转换为布尔值时，会将字符串 “true”（不区分大小写）转换为 `true`，并将字符串 “false”（不区分大小写）转换为 `false`。 其他任何字符串将转换为 `false`。
- 将字符串转换为数据类型为 `byte`， `int`， `long` 或 `short` 的值时，该字符串必须具有以下格式:

  [*blanks*][*sign*] *digits*

  该字符串各个组成部分的含义如下:

  blanks
  可选前导空白字符

  sign
  可选加号 (+) 或减号 (-)。

  digits
  一组连续数字 (0-9)。 至少要存在一个数字。

  在该数字序列后，字符串可以包含其他非数字字符，但是转换到达第一个非数字字符时，转换将停止。 假设该字符串表示十进制整数。

  如果字符串的格式不正确，IBM MQ classes for JMS 会抛出 NumberFormatException 异常。
- 将字符串转换为数据类型为 `double` 或 `float` 的值时，该字符串必须具有以下格式:

  [*blanks*][*sign*] *digits* [*e_char* [*e_sign*] *e_digits*]

  该字符串各个组成部分的含义如下:

  blanks
  可选前导空白字符

  sign
  可选加号 (+) 或减号 (-)。

digits
一组连续数字 (0-9)。 至少要存在一个数字。

e_char
阶符，为 *E* 或 *e*。

e_sign
用于表示指数的加号 (+) 或减号 (-)（可选）。

e_digits
用于表示指数的一组连续数字 (0-9)。 如果字符串中包含阶符，则至少要存在一个数字。

在该数字序列后，或表示阶符的可选字符后，字符串可以包含其他非数字字符，但是转换到达第一个非数字字符时，转换将停止。 假设该字符串表示十进制浮点数，指数幂为 10。

如果字符串的格式不正确，IBM MQ classes for JMS 会抛出 NumberFormatException 异常。

- 转换数字值 (包括数据类型为 byte 的值) 时 将该值转换为该值的字符串表示为十进制数，而不是包含该值的 ASCII 字符的字符串。 例如，整数 65 将转换为字符串 “65”，而不是字符串 “A”。

在一个调用中设置多个属性

JmsPropertyContext 接口还包含 setBatchProperties() 方法，应用程序可以使用此方法在一个调用中设置多个属性。 该方法的参数是一个 Map 对象，可封装一组属性名称值对。

例如，以下代码可使用 setBatchProperties() 方法设置与 第 165 页的『设置 IBM MQ classes for JMS 对象的属性』 中所示相同的五个连接工厂属性。 该代码将创建一个 HashMap 类实例，用于实现 Map 接口。

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

请注意，Map.put() 方法的第二个参数必须是对象。 因此，如示例所示，带有原语数据类型的属性值必须封装在对象内或由字符串表示。

setBatchProperties() 方法用于验证各个属性。 如果 setBatchProperties() 方法由于某些原因无法设置属性（例如，值无效），那么不会设置任何指定的属性。

属性名称和值

如果应用程序使用 JmsPropertyContext 接口的方法设置或获取 IBM MQ classes for JMS 对象的属性，那么该应用程序可以通过以下任一方式指定属性的名称和值。 每个随附示例展示了如何设置 JmsQueue 对象 q1 的 PRIORITY 属性，以便发送到队列的消息具有 send() 调用上指定的优先级。

使用 com.ibm.msg.client.wmq.WMQConstants 接口中定义为常量的属性名称和值
以下语句是关于如何通过此方式指定属性名称和值的示例：

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

使用可以用在队列和主题统一资源标识 (URI) 中的属性名称和值
以下语句是关于如何通过此方式指定属性名称和值的示例：

```
q1.setIntProperty("priority", -2);
```

使用此方式只能指定目标的属性名称和值。

使用 IBM MQ JMS 管理工具识别的属性名称和值
以下语句是关于如何通过此方式指定属性名称和值的示例：

```
q1.setStringProperty("PRIORITY", "APP");
```

如以下示例所示，还可以接受简短形式的属性名称：

```
q1.setStringProperty("PRI", "APP");
```

当应用程序获取属性时，返回的值取决于应用程序指定属性名称的方式。 例如，如果应用程序指定常量 WMQConstants.WMQ_PRIORITY 作为属性名称，则返回的值为整数 -2：

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

如果应用程序指定字符串“priority”作为属性名称，将返回相同的值：

```
int n2 = getIntProperty("priority");
```

但是，如果应用程序指定字符串“PRIORITY”或“PRI”作为属性名称，返回的值为字符串“APP”：

```
String s1 = getStringProperty("PRI");
```

在内部，IBM MQ classes for JMS 将属性名称和值存储为 com.ibm.msg.client.wmq.WMQConstants 接口内定义的字面值。 这是已定义的用于属性名称和值的规范格式。 一般情况下，如果应用程序使用另外两种指定属性名称和值的方法中的任一方法来设置属性，那么 IBM MQ classes for JMS 必须将名称和值从指定输入格式转换为规范格式。 同样，如果应用程序使用另外两种指定属性名称和值的方法中的任一方法来获取属性，那么 IBM MQ classes for JMS 必须将名称从指定输入格式转换为规范格式，并将值从规范格式转换为所需的输出格式。 必须执行这些转换可能会对性能有影响。

通过跟踪文件或 IBM MQ classes for JMS 日志中的异常返回的属性名称和值始终采用规范格式。

使用 Map 接口

JmsPropertyContext 接口可扩展 java.util.Map 接口。 因此，应用程序可以使用 Map 接口的方法访问 IBM MQ classes for JMS 对象的属性。

例如，以下代码将打印出连接工厂所有属性的名称和值。 此代码只使用 Map 接口的方法获取属性的名称和值。

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

使用 Map 接口的方法不会绕过任何属性验证或转换。

使用 *IBM MQ JMS* 扩展

IBM MQ classes for JMS 中包含一组对 JMS API 的扩展，称为 IBM MQ JMS 扩展。 应用程序可以使用这些扩展在运行时动态创建连接工厂和目标，并可以设置连接工厂和目标的属性。

IBM MQ classes for JMS 在包 com.ibm.jms 和 com.ibm.mq.jms 中包含一组类。 这些类可实现 JMS 接口并包含 IBM MQ JMS 扩展。 后面的示例代码假设这些包已通过以下语句导入：

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

应用程序可以使用 IBM MQ JMS 扩展执行以下功能：

- 在运行时动态创建连接工厂和目标，而不是作为受管对象从 Java 命名和目录接口 (JNDI) 名称空间中检索连接工厂和目标。
- 设置连接工厂和目标的属性

创建连接工厂

如以下示例所示，要创建连接工厂，应用程序可以使用 MQConnectionFactory 构造函数：

```
MQConnectionFactory factory = new MQConnectionFactory();
```

该语句可创建所有属性均为缺省值的 MQConnectionFactory 对象，这表示应用程序将以绑定方式连接到缺省队列管理器。 如果您希望在客户机模式下连接应用程序或连接到缺省队列管理器之外的队列管理器，那么在创建连接之前，应用程序必须设置 MQConnectionFactory 对象的正确属性。 有关如何执行此操作的信息，请参阅第 170 页的『设置连接工厂的属性』。

应用程序可以通过类似方式创建以下类型的连接工厂：

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

设置连接工厂的属性

应用程序可以通过调用连接工厂的相应方法来设置连接工厂的属性。 连接工厂可以是受管对象或运行时动态创建的对象。

例如，请考虑使用以下代码：

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

此代码可创建 MQConnectionFactory 对象，然后设置对象的五个属性。 设置这些属性的效果是应用程序使用名为 QM1.SVR 的 MQI 通道连接到客户机模式下的队列管理器 QM1。 队列管理器在主机名为 HOST1 的系统上运行，而队列管理器的侦听器在端口号 1415 上侦听。

使用与代理程序实时连接的应用程序只能使用发布/预订式的消息传递。 不能使用点到点样式的消息传递。

只有特定的连接工厂属性组合有效。 有关哪些组合有效的信息，请参阅 IBM MQ classes for JMS 对象属性之间的依赖关系。

有关连接工厂的属性以及用于设置其属性的方法的更多信息，请参阅 IBM MQ classes for JMS 对象的属性。

创建目标

如以下示例所示，要创建 Queue 对象，应用程序可以使用 MQQueue 构造函数：

```
MQQueue q1 = new MQQueue("Q1");
```

该语句可创建所有属性均为缺省值的 MQQueue 对象。 该对象表示名为 Q1 的 IBM MQ 队列，此队列属于本地队列管理器。 此队列可以是本地队列、别名队列或远程队列定义。

如以下示例所示，另一种形式的 MQQueue 构造函数有两个参数：

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

该语句创建的 MQQueue 对象表示名为 Q2 的 IBM MQ 队列，此队列归队列管理器 QM2 所有。 以这种方式识别的队列管理器可以是本地队列管理器或远程队列管理器。 如果是远程队列管理器，那么必须配置 IBM MQ，以便在应用程序向此目标发送消息时，WebSphere MQ 可以将该消息从本地队列管理器路由到远程队列管理器。

MQQueue 构造函数还可以接受队列统一资源标识 (URI) 作为单个参数。 队列 URI 是一个字符串，用于指定 IBM MQ 队列的名称，（可选）拥有此队列的队列管理器的名称以及 MQQueue 对象的一个或多个属性。 以下语句包含队列 URI 示例：

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

此语句创建的 MQQueue 对象表示名为 Q3 的 IBM MQ 队列（由队列管理器 QM3 拥有），发送至此目标的所有消息都是持久消息，并且其优先级为 5。 有关队列 URI 的更多信息，请参阅 第 175 页的『统一资源标识 (URI)』。 有关设置 MQQueue 对象属性的替代方法，请参阅第 171 页的『设置目标的属性』。

如以下示例所示，要创建 Topic 对象，应用程序可以使用 MQTopic 构造函数：

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

该语句可创建所有属性均为缺省值的 MQTopic 对象。 该对象表示名为 Sport/Football/Results 的主题。

MQTopic 构造函数还可以接受主题 URI 作为参数。 主题 URI 是一个字符串，用于指定主题的名称和（可选）MQTopic 对象的一个或多个属性。 以下语句包含主题 URI 示例：

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

此语句创建的 MQTopic 对象表示名为 Sport/Tennis/Results 的主题，发送至此目标的所有消息都是非持久消息，并且其优先级为 0。 有关主题 URI 的更多信息，请参阅 第 175 页的『统一资源标识 (URI)』。 有关设置 MQTopic 对象属性的替代方法，请参阅第 171 页的『设置目标的属性』。

设置目标的属性

应用程序可以通过调用目标的相应方法来设置目标的属性。 目标可以是受管对象或运行时动态创建的对象。

例如，请考虑使用以下代码：

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

此代码可创建 MQQueue 对象，然后设置对象的两个属性。 设置这些属性的效果是发送到此目标的所有消息都是持久性的且具有优先级 5。

如以下示例所示，应用程序可以通过类似地方法设置 MQTopic 对象的属性：

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

此代码可创建 MQTopic 对象，然后设置对象的两个属性。 设置这些属性的效果是发送到此目标的所有消息都是非持久性的且具有优先级 0。

有关目标的属性以及用于设置其属性的方法的更多信息，请参阅 IBM MQ classes for JMS 对象的属性。

在 *JMS* 应用程序中构建连接

要构建连接，JMS 应用程序需使用 ConnectionFactory 对象创建 Connection 对象，然后启动连接。

如以下示例所示，要创建 Connection 对象，应用程序可以使用 ConnectionFactory 对象的 createConnection() 方法：

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

创建 JMS 连接时，IBM MQ classes for JMS 会创建连接句柄 (Hconn) 并启动与队列管理器的对话。

QueueConnectionFactory 接口和 TopicConnectionFactory 接口分别从 ConnectionFactory 接口继承了 createConnection() 方法。 因此，如以下示例所示，您可以使用 createConnection() 方法创建特定于域的对象：

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

这个代码段可创建 QueueConnection 对象。 应用程序现在可以在此对象上执行独立于域的操作，或执行仅适用于点到点域的操作。 但是，如果应用程序尝试执行仅适用于发布/预订域的操作，将抛出 IllegalStateException 异常并带有以下消息：

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

这是因为连接是从特定于域的连接工厂中创建的。

注: 请注意，应用程序进程标识用作要传递到队列管理器的缺省用户身份。 如果应用程序在客户机传输模式下运行，那么此进程标识必须存在于服务器上并具有相关权限。 如果要使用不同标识，那么请使用 createConnection(username, password) 方法。

JMS 规范规定了以 `stopped` 状态创建连接。 连接启动之前，与连接关联的消息使用者无法收到任何消息。 如以下示例所示，要启动连接，应用程序将使用 Connection 对象的 start() 方法：

```
connection.start();
```

在 *JMS* 应用程序中创建会话

要创建会话，JMS 应用程序需使用 Connection 对象的 createSession() 方法。

createSession() 方法有两个参数：

1. 一个参数用于指定会话为事务性还是非事务性。
2. 一个参数用于指定会话的确认模式

例如，以下代码可创建非事务性且确认模式为 AUTO_ACKNOWLEDGE 的会话：

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

创建 JMS 会话时，IBM MQ classes for JMS 会创建连接句柄 (Hconn) 并启动与队列管理器的对话。

Session 对象以及从 Session 对象创建的任何 MessageProducer 或 MessageConsumer 对象不能同时用于多线程应用程序的不同线程。 确保这些对象不会被并行使用的最简单方法是为每个线程创建单独的 Session 对象。

JMS 应用程序中的事务性会话

JMS 应用程序可通过先创建事务性会话来运行本地事务。 应用程序可以提交或回滚事务。

JMS 应用程序可以运行本地事务。 本地事务表示此事务只涉及对该应用程序所连接的对队列管理器的资源的更改。 要运行本地事务，应用程序必须先通过调用 Connection 对象的 createSession() 方法创建事务性会话，指定为该会话处理的参数。 然后，会话内发送和接收的所有消息将分组到事务序列中。 应用程序提交或回滚事务开始后所发送和接收的消息时，表明事务结束。

要提交事务，应用程序需调用 Session 对象的 commit() 方法。 提交事务时，事务中发送的所有消息可用于发送到其他应用程序，而事务内收到的所有消息将得到确认，这样一来消息传递服务器便不会尝试再次将这些消息发送到应用程序。 在点到点域中，消息传递服务器还会从队列中移除收到的消息。

要回滚事务，应用程序需调用 Session 对象的 rollback() 方法。 回滚事务时，消息传递服务器将废弃事务内发送的所有消息，而事务内收到的所有消息可用于再次发送。 在点到点域中，之前收到的消息将放回队列并可再次被其他应用程序看到。

应用程序创建事务性会话或调用 commit() 或 rollback() 方法时会自动启动新事务。 因此，事务性会话总是有活动事务。

当应用程序关闭事务性会话时，会发生隐式回滚。 当应用程序关闭连接时，所有连接的事务性会话将发生隐式回滚。

如果应用程序结束时没有关闭连接，所有连接的事务性会话也发生隐式回滚。

事务完全包含在事务性会话中。 事务不能跨越多个会话。 这表示一个应用程序不能在两个或多个事务性会话中发送和接收消息然后作为单个事务提交或回滚所有操作。

JMS 会话的确认模式

每个非事务性会话都有一种确认模式，用于确定如何确认应用程序收到的消息。 共有三种确认模式可用，确认模式的选择会影响应用程序的设计。

如果会话为非事务性，那么确认应用程序收到的消息的方法取决于该会话的确认模式。 以下段落描述了这三种确认模式：

AUTO_ACKNOWLEDGE

会话自动确认应用程序收到的每个消息。

如果消息同步发送到应用程序，会话会在每次 Receive 调用成功完成时确认收到消息。 如果消息是异步发送，会话会在每次调用消息侦听器的 onMessage() 方法成功完成时确认收到消息。

如果应用程序成功收到消息，但存在故障阻止进行确认，那么该消息将变为可重新发送。 因此，应用程序必须能够处理重新发送的消息。

DUPS_OK_ACKNOWLEDGE

会话在进行选择时确认应用程序收到消息。

使用此确认模式可减少会话必须执行的工作量，但是阻止消息确认的故障可能会导致多个消息变为可重新发送。 因此，应用程序必须能够处理重新发送的消息。

限制: 在 AUTO_ACKNOWLEDGE 和 DUPS_OK_ACKNOWLEDGE 模式下，JMS 不支持应用程序在消息侦听器中抛出未处理的异常。 这表示消息侦听器返回时消息始终会得到确认，无论消息处理是否成功（前提是所有故障都不是致命的，不会阻止应用程序继续运行）。 如果您需要更精细的控制消息确认，请使用 CLIENT_ACKNOWLEDGE 或事务性模式，这两种模式能让应用程序完全控制确认功能。

CLIENT_ACKNOWLEDGE

应用程序通过调用 Message 类的 Acknowledge 方法来确认它所收到的消息。

应用程序可以单独地确认收到每个消息，或者可以接收一批消息并只为收到的最后一条消息调用 Acknowledge 方法。 如果调用 Acknowledge 方法，自上次调用此方法以来收到的所有消息都将得到确认。

与以上任一确认模式配合使用，应用程序可以通过调用 Session 类的 Recover 方法停止并重新启动会话中的消息发送。 已收到但之前未确认过的消息将重新发送。 但是，这些消息的发送顺序可能会与之前的发送顺序不同。 在此期间，优先级较高的消息可能已到达，而某些原始消息可能已到期。 在点到点域中，某些原始消息可以已被另一个应用程序使用。

应用程序可以通过检查消息的 JMSRedelivered 头字段的内容来确定消息是否会重新发送。 应用程序通过调用 Message 类的 getJMSRedelivered() 方法实现此目的。

在 *JMS* 应用程序中创建目标

JMS 应用程序可以使用会话在运行时动态创建目标，而不是从 Java 命名和目录接口 (JNDI) 名称空间中检索目标作为受管对象。 应用程序可以使用统一资源标识 (URI) 识别 IBM MQ 队列或主题，也可以指定 Queue 或 Topic 对象的一个或多个属性（可选）。

使用会话创建 Queue 对象

如以下示例所示，要创建 Queue 对象，应用程序可以使用 Session 对象的 createQueue() 方法：

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

该代码可创建所有属性均为缺省值的 Queue 对象。 该对象表示名为 Q1 的 IBM MQ 队列，此队列属于本地队列管理器。 此队列可以是本地队列、别名队列或远程队列定义。

createQueue() 方法还可以接受队列 URI 作为参数。 队列 URI 是一个字符串，用于指定 IBM MQ 队列的名称，（可选）拥有此队列的队列管理器的名称以及 Queue 对象的一个或多个属性。 以下语句包含队列 URI 示例：

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

此语句创建的 Queue 对象表示名为 Q2 的 IBM MQ 队列（由队列管理器 QM2 拥有），发送至此目标的所有消息都是持久消息，并且其优先级为 5。 以这种方式识别的队列管理器可以是本地队列管理器或远程队列管理器。 如果是远程队列管理器，那么必须配置 IBM MQ，以便在应用程序向此目标发送消息时，WebSphere MQ 可以将该消息从本地队列管理器路由到队列管理器 QM2。 有关 URI 的更多信息，请参阅第 175 页的『统一资源标识 (URI)』。

请注意，createQueue() 方法上的参数可以包含特定于提供程序的信息。 因此，如果不是作为受管对象从 JNDI 名称空间中检索 Queue 对象，而是使用 createQueue() 方法创建 Queue 对象，可能会让您的应用程序的可移植性降低。

如以下示例所示，应用程序可以使用 Session 对象的 createTemporaryQueue() 方法创建 TemporaryQueue 对象：

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

尽管会话用于创建临时队列，但临时队列的作用域是用于创建会话的连接。 连接的任何会话都可以为临时队列创建消息生产者和消息使用者。 连接结束或应用程序使用 TemporaryQueue.delete() 方法显示删除临时队列之前（以先发生的为准），临时队列会一直保留。

在应用程序创建暂时队列时，IBM MQ classes for JMS 会在应用程序连接到的队列管理器中创建动态队列。 连接工厂的 TEMPMODEL 属性指定用于创建动态队列的模型队列的名称，连接工厂的 TEMPQPREFIX 属性指定用于组成动态队列名称的前缀。

使用会话创建 Topic 对象

如以下示例所示，要创建 Topic 对象，应用程序可以使用 Session 对象的 createTopic() 方法：

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

该代码可创建所有属性均为缺省值的 Topic 对象。 该对象表示名为 Sport/Football/Results 的主题。

createTopic() 方法还接受主题 URI 作为参数。 主题 URI 是一个字符串，用于指定主题的名称，同时可选择性地指定 Topic 对象的一个或多个属性。 以下代码包含主题 URI 示例：

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

此语句创建的 Topic 对象表示名为 Sport/Tennis/Results 的主题，发送至此目标的所有消息都是非持久消息，并且其优先级为 0。 有关主题 URI 的更多信息，请参阅 第 175 页的『统一资源标识 (URI)』。

请注意，createTopic() 方法上的参数可以包含特定于提供程序的信息。 因此，如果不是作为受管对象从 JNDI 名称空间中检索 Topic 对象，而是使用 createTopic() 方法创建 Topic 对象，可能会让您的应用程序的可移植性降低。

如以下示例所示，应用程序可以使用 Session 对象的 createTemporaryTopic() 方法创建 TemporaryTopic 对象：

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

尽管会话用于创建临时主题，但主题队列的作用域是用于创建会话的连接。 连接的任何会话都可以为临时主题创建消息生产者和消息使用者。 连接结束或应用程序使用 TemporaryTopic.delete() 方法显示删除临时主题之前（以先发生的为准），临时主题会一直保留。

当应用程序创建临时主题时， IBM MQ classes for JMS 将创建名称以字符 TEMP/*tempTopicPrefix* 开头的主题，其中 *tempTopicPrefix* 是连接工厂的 TEMPTOPICPREFIX 属性的值。

统一资源标识 (URI)

队列 URI 是一个字符串，用于指定 IBM MQ 队列的名称，同时可选择性地指定拥有此队列的队列管理器的名称以及该应用程序创建的 Queue 对象的一个或多个属性。 主题 URI 是一个字符串，用于指定主题的名称，同时可选择性地指定应用程序创建地 Topic 对象的一个或多个属性。

队列 URI 采用以下格式：

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

主题 URI 采用以下格式：

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

这些格式中的变量具有以下含义：

qMgrName
拥有 URI 识别的队列的队列管理器的名称。

该队列管理器可以是本地队列管理器或远程队列管理器。 如果是远程队列管理器，那么必须配置 IBM MQ，以便在应用程序向此队列发送消息时，WebSphere MQ 可以将该消息从本地队列管理器路由到远程队列管理器。

如果不指定任何名称，将采用本地队列管理器。

qName
IBM MQ 队列的名称。

该队列可以是本地队列、别名队列或远程队列定义。

有关创建队列名称的规则，请参阅 IBM MQ 对象的命名规则。

topicName
主题的名称。

有关创建主题名称的规则，请参阅 IBM MQ 对象的命名规则。 避免使用通配符 + ， # ， * 和? 在主题名称中。 在您预订主题时，主题名称中包含这些字符可能会导致意外结果。 请参阅使用主题字符串。

***propertyName1*, *propertyName2*, ...**
由应用程序创建的 Queue 或 Topic 对象的属性名称。 第 176 页的表 34 中列出了 URI 中可以使用的有效属性名称。

如果没有指定属性，Queue 或 Topic 对象将对其所有属性使用缺省值。

***propertyValue1*, *propertyValue2*, ...**
由应用程序创建的 Queue 或 Topic 对象的属性值。 第 176 页的表 34 中列出了 URI 中可以使用的有效属性值。

方括号 ([]) 表示可选组件，省略号 (...) 表示属性名称值对列表可以包含一个或多个名称值对（如果存在）。

第 176 页的表 34 中列出了主题 URI 中可以使用的有效属性名称和有效值。 尽管 IBM MQ JMS 管理工具使用符号常量作为属性值，但 URI 不能包含符号常量。

表 *34:* 用于队列和主题 *URI* 的属性名称和有效值

属性名	**描述**	**有效值**
CCSID	IBM MQ classes for JMS 将消息转发到目标时，如何表示消息体中的字符数据	• IBM MQ 支持的任何编码字符集标识。
编码	IBM MQ classes for JMS 将消息转发到目标时，如何表示消息体中的数字数据	• IBM MQ 消息描述符中 编码 字段的任何有效值。
到期	发送到目标的消息的生存时间	• -2 - 按照 send() 调用所指定，或者如果 send() 调用上没有指定，将使用消息生产者的缺省生存时间。 • 0 - 发送到目标的消息永不过期。 • 正整数，用于以毫秒为单位指定生存时间。
multicast	使用与代理程序的实时连接时，主题的多点广播设置	以下列表包含有效值。 与各个值相关联是指 IBM MQ JMS 管理工具中使用的 MULTICAST 属性的对应值。 有关 MULTICAST 属性及其有效值的描述，请参阅 IBM MQ classes for JMS 对象属性。 • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED
持久性	发送到目标的消息的持久性	• -2 - 按照 send() 调用所指定，或者如果 send() 调用上没有指定，将使用消息生产者的缺省持久性。 • -1-由 IBM MQ 队列或主题的 *DefPersistence* 属性指定。 • 1 - 非持久性。 • 2 - 持久性。 • 3 - 等同于在 IBM MQ JMS 管理工具中使用时 PERSISTENCE 属性的值 HIGH。 有关此值的说明，请参阅第 196 页的『JMS 持久消息』。

表 *34:* 用于队列和主题 *URI* 的属性名称和有效值 *(*继续*)*		
属性名	**描述**	**有效值**
priority	发送到目标的消息的优先级	• -2 - 按照 send() 调用所指定，或者如果 send() 调用上没有指定，将使用消息生产者的缺省优先级。 • -1-由 IBM MQ 队列或主题的 *DefPriority* 属性指定。 • 0-9 范围内的整数，用于指定发送到目标的消息优先级。
targetClient	发送到目标的消息是否包含 MQRFH2 头	• 0 - 消息包含 MQRFH2 头。 • 1 - 消息不包含 MQRFH2 头。

例如，以下 URI 标识本地队列管理器所拥有的名为 Q1 的 IBM MQ 队列。 使用此 URI 创建的 Queue 对象的所有属性均为缺省值。

```
queue:///Q1
```

以下 URI 标识队列管理器 QM2 所拥有的名为 Q2 的 IBM MQ 队列。 发送至此目标的所有消息的优先级为 6。 使用此 URI 创建的 Queue 对象的其余属性具有缺省值。

```
queue://QM2/Q2?priority=6
```

以下 URI 标识名为 Sport/Athletics/Results 的主题。 发送至此目标的所有消息都是非持久消息，并且其优先级为 0。 使用此 URI 创建的 Topic 对象的其余属性具有缺省值。

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

在 *JMS* 应用程序中发送消息

JMS 应用程序必须先为目标创建 MessageProducer 对象，然后才能向目标发送消息。 要向目标发送消息，应用程序要先创建 Message 对象，然后调用 MessageProducer 对象的 send() 方法。

应用程序使用 MessageProducer 对象发送消息。 应用程序通常为特定目标创建 MessageProducer 对象，此目标可以是队列或主题，这样一来使用消息生产者发送的所有消息都会发送到同一目标。 因此，应用程序必须先创建 Queue 或 Topic 对象，然后才能创建 MessageProducer 对象。 有关如何创建 Queue 或 Topic 对象的信息，请参阅以下主题：

- 第 163 页的『使用 JNDI 检索 JMS 应用程序中的受管对象』
- 第 164 页的『使用 IBM JMS 扩展』
- 第 169 页的『使用 IBM MQ JMS 扩展』
- 第 174 页的『在 JMS 应用程序中创建目标』

如以下示例所示，要创建 MessageProducer 对象，应用程序需使用 Session 对象的 createProducer() 方法：

```
MessageProducer producer = session.createProducer(destination);
```

参数 `destination` 是应用程序创建之前创建的 Queue 或 Topic 对象。

应用程序必须先创建 Message 对象，然后才能发送消息。 消息体包含应用程序数据，而 JMS 定义了五种类型的消息体：

- 字节
- 映射

- Object
- 流
- 文本

每种类型的消息体都有自己的 JMS 接口（这是 Message 接口的子接口），Session 中包含一个方法用于创建带体验该类型主体的消息。 例如，如以下语句所示，文本消息的接口称为 TextMessage，应用程序需使用 Session 对象的 createTextMessage() 方法创建文本消息：

```
TextMessage outMessage = session.createTextMessage(outString);
```

有关消息和消息体的更多信息，请参阅第 112 页的『JMS 消息』。

如以下示例所示，要发送连接，应用程序需使用 MessageProducer 对象的 send() 方法：

```
producer.send(outMessage);
```

应用程序可以使用 send() 方法在任何一个消息传递域内发送消息。 目标的性质决定了要使用哪个消息传递域。 但是，发布/预订域专用的 TopicPublisher（MessageProducer 的子接口）还有一个 publish() 方法，可用于代替 send() 方法。 这两个方法在功能上是相同的。

应用程序可以创建无指定目标的 MessageProducer 对象。 这种情况下，应用程序调用 send() 方法时必须指定目标。

如果应用程序在事务内发送消息，那么提交事务之前，该消息不会发送到目标。 这表示应用程序不能在同一事务中发送消息和接收对消息的回复。

可以对目标进行配置，以便应用程序向其发送消息时，IBM MQ classes for JMS 可以转发消息并将控制权返回应用程序，无需确定队列管理器是否已安全收到消息。 这一共能有时称为异步输入。 有关更多信息，请参阅 第 260 页的『在 IBM MQ classes for JMS 中异步放置消息』。

在 *JMS* 应用程序中接收消息

应用程序使用消息使用者接收消息。 持久主题订户是接收发送到目标的所有消息的消息使用者，包括使用者处于非活动状态时发送的消息。 应用程序可以使用消息选择器选择希望接收的消息，并可以使用消息侦听器异步接收消息。

应用程序使用 MessageConsumer 接收消息。 应用程序将为特定目标创建 MessageConsumer 对象，此目标可以是队列或主题，这样一来使用消息使用者接收的所有消息都会从同一目标接收。 因此，应用程序必须先创建 Queue 或 Topic 对象，然后才能创建 MessageConsumer 对象。 有关如何创建 Queue 或 Topic 对象的信息，请参阅以下主题：

- 第 163 页的『使用 JNDI 检索 JMS 应用程序中的受管对象』
- 第 164 页的『使用 IBM JMS 扩展』
- 第 169 页的『使用 IBM MQ JMS 扩展』
- 第 174 页的『在 JMS 应用程序中创建目标』

如以下示例所示，要创建 MessageConsumer 对象，应用程序需使用 Session 对象的 createConsumer() 方法：

```
MessageConsumer consumer = session.createConsumer(destination);
```

参数 `destination` 是应用程序创建之前创建的 Queue 或 Topic 对象。

然后，如以下示例所示，应用程序使用 MessageConsumer 对象的 receive() 方法接收来自目标的消息：

```
Message inMessage = consumer.receive(1000);
```

receive() 调用上的参数指定在没有马上可用的消息的情况下方法等待合适消息到达的时间（以毫秒为单位）。 如果您省略此参数，调用将无限期阻断，直至合适的消息到达。 如果您希望应用程序等待消息，请改为使用 receiveNoWait() 方法。

receive() 方法可返回特定类型的消息。 例如，当应用程序接收文本消息时，receive() 调用返回的对象为 TextMessage 对象。

但是，receive() 调用返回的对象的声明类型为 Message 对象。 因此，为了从刚刚接收到的消息体中抽取数据，应用程序必须从 Message 类转换到更具有针对性的子类，如 TextMessage。 如果不知道消息的类型，应用程序可以使用 `instanceof` 操作程序确定类型。 让应用程序确定消息类型，再进行强制转换以便可以正常地处理错误，始终是较为妥善的作法。

以下代码使用 `instanceof` 操作程序并显示了如何从文本消息主体抽取数据：

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

如果应用程序在事务内发送消息，那么提交事务之前，该消息不会发送到目标。 这表示应用程序不能在同一事务中发送消息和接收对消息的回复。

如果消息使用者从配置了预读的目标接收消息，那么预读缓冲区中的任何非持久性消息会在应用程序结束时被废弃。

在发布/预订域中，JMS 识别非持久主题订户和持久主题订户这两种类型的消息使用者，以下两节对这两种订户进行了描述。

非持久主题订户

非持久主题订户只接收订户处于活动状态时发布的消息。 非持久订户在应用程序创建非持久主题订户时开始，在应用程序关闭订户或订户超出作用域时结束。 作为 IBM MQ classes for JMS 中的扩展，非持久主题订户还会接收保留发布内容。

要创建非持久主题订户，应用程序可以使用独立于域的 createConsumer() 方法，此方法可指定 Topic 对象作为目标。 或者，如以下示例所示，应用程序可以使用特定于域的 createSubscriber() 方法：

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

参数 `topic` 是应用程序创建之前已创建的 Topic 对象。

持久主题订户

限制: 在使用与代理程序的实时连接时，应用程序不能创建持久主题订户。

持久主题订户接收持久预订期间发布的所有消息。 这些消息包括订户处于非活动状态时发布的所有消息。 作为 IBM MQ classes for JMS 中的扩展，持久主题订户还会接收保留发布内容。

如以下示例所示，要创建持久主题订户，应用程序需使用 Session 对象的 createDurableSubscriber() 方法：

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

在 createDurableSubscriber() 调用上，第一个参数为应用程序创建之前已创建的 Topic 对象，第二个参数为用于标识持久预订的名称。

用于创建持续主题订户的会话必须有关联的客户机标识。 与会话关联的客户机标识与用于创建此会话的连接的客户机标识相同。 客户机标识可以通过设置 ConnectionFactory 对象的 CLIENTID 属性来指定。 或者，应用程序可以通过调用 Connection 对象的 setClientID() 方法设置客户机标识。

用于标识持久预订的名称只需在客户机标识中必须唯一，因此客户机标识组成持久预订的完整且唯一标识符中的一部分。 要继续使用之前创建的持续预订，应用程序必须使用会话创建一个持久主题订户，该会话的客户机标识同与持久预订关联的客户机标识相同，并使用相同的预订名称。

持久预订在应用程序使用客户机标识和预订名称（当前不存在持续预订）创建持久主题订户时开始。 但是，持久预订不会在应用程序关闭持续主题订户时结束。 要结束持续预订，应用程序必须调用 Session 对象的 unsubscribe() 方法，该对象的客户机标识同与持久预订的关联客户机标识相同。 如以下示例所示，unsubscribe() 调用上的参数为预订名称：

```
session.unsubscribe("D_SUB_000001");
```

持久预订的作用域为队列管理器。 如果持续预订存在于一个队列管理器上，而连接到另一队列管理器的应用程序使用相同的客户机标识和预订名称创建持久预订，那么这两个持久预订完全不相关。

消息选择器

应用程序可以指定连续 receive() 调用只返回满足特定条件的消息。 在创建 MessageConsumer 对象时，应用程序可以指定用于确定要检索哪些消息的结构化查询语言 (SQL) 表达式。 该 SQL 表达式称为消息选择器。 消息选择器可以包含 JMS 消息头字段和消息属性的名称。 有关如何构造消息选择器的信息，请参阅 第 112 页的『JMS 中的消息选择器』。

以下示例显示了应用程序如何根据用户定义的名为 myProp 的属性来选择消息：

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

JMS 规范不允许应用程序更改消息使用者的消息选择器。 应用程序创建带有消息选择器的消息使用者后，消息选择器在该使用者的存续期间会一直存在。 如果应用程序需要多个消息选择器，那么应用程序必须为每个消息选择器创建消息使用者。

请注意，当应用程序连接到 IBM WebSphere MQ 7 队列管理器时，连接工厂的 MSGSELECTION 属性没有任何作用。 为了优化性能，所有消息选择均由队列管理器完成。

禁止本地发布内容

应用程序可以创建消息使用者，此消息使用者将忽略使用者自身连接上发布的发布内容。 如以下示例所示，应用程序通过将 createConsumer() 调用上的第三个参数设置为 `true` 来完成此操作：

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

如以下示例所示，在 createDurableSubscriber() 调用上，应用程序通过将第四个参数设置为 `true` 来完成此操作：

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

异步发送消息

通过向消息使用者注册消息侦听器，应用程序可以异步接收消息。 消息侦听器有一个称为 onMessage 的方法，此方法在出现合适的消息时调用，其目的是处理此消息。 以下代码展示了这种机制：

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
```

```
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

应用程序可以使用会话通过 receive() 调用异步接收消息，也可以使用消息侦听器异步接收消息，但两者不能同时使用。 如果应用程序需要同步和异步接收消息，必须创建单独的会话。

将会话设置为异步接收消息后，不能在此会话或从该会话创建的对象上调用以下方法：

- MessageConsumer.receive()
- MessageConsumer.receive(long)
- MessageConsumer.receiveNoWait()
- Session.acknowledge()
- MessageProducer.send(Destination, Message)
- MessageProducer.send(Destination, Message, int, int, long)
- MessageProducer.send(Message)
- MessageProducer.send(Message, int, int, long)
- MessageProducer.send(Destination, Message, CompletionListener)
- MessageProducer.send(Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send(Message, CompletionListener)
- MessageProducer.send(Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(Queue)
- Session.createBrowser(Queue, String)
- Session.createBytesMessage()
- Session.createConsumer(Destination)
- Session.createConsumer(Destination, String, boolean)
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Destination)
- Session.createQueue(String)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()

- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

如果调用以上任一方法，将抛出包含消息

```
JMSCC0033: 异步使用会话时，不允许同步方法调用: "method name"
```

的 JMSException。

接收有害消息

应用程序可能收到无法处理的消息。 可能有几种原因会导致消息无法处理，例如消息格式不正确。 此类消息称为有害消息，需要特殊处理才能阻止消息以递归方式处理。

有关如何处理有害消息的详细信息，请参阅第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』。

检索预订用户数据

如果通过以管理方式定义的持久预订放入了可供 IBM MQ classes for JMS 应用程序从队列使用的消息，那么该应用程序需要访问与该预订相关联的用户数据信息。 这会将此信息作为属性添加到消息中。

从包含带有 MQPS 文件夹的 RFH2 头的队列中使用消息时，与 Sud 键关联的值 (如果存在) 将作为字符串属性添加到返回到 IBM MQ classes for JMS 应用程序的 JMS 消息对象。 为了能够从消息中检索此属性，可将 JmsConstantsTo 接口中的常量 JMS_IBM_SUBSCRIPTION_USER_DATA 与方法 javax.jms.Message.getStringProperty(java.lang.String) 一起用于获取预订用户数据。

在以下示例中，使用 MQSC 命令 **DEFINE SUB** 定义了管理持久预订：

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

这会将发布到主题字符串 PUBLIC 的消息的副本放入队列 MY.SUBSCRIPTION.Q 中。 然后，会将与持久预订相关联的用户数据作为属性添加到消息中，此消息存储在具有 Sud 键的 RFH2 头的 MQPS 文件夹中。

IBM MQ classes for JMS 应用程序可调用：

```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

将返回以下字符串：

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

.

相关概念

第 116 页的『MQRFH2 头和 JMS』

相关任务

定义管理预订

相关参考

DEFINE SUB

接口 JmsConstants

关闭 *IBM MQ classes for JMS* 应用程序

IBM MQ classes for JMS 应用程序在停止前显式关闭某些 JMS 对象是非常重要的。 可能不会调用终结器，所以请不要依赖终结器来释放资源。 不允许应用程序在压缩跟踪处于活动状态时终止。

仅凭垃圾回收无法及时释放所有 IBM MQ classes for JMS 和 IBM MQ 资源，尤其是应用程序在会话级别或更低级别创建了多个短期存在的 JMS 对象的情况下。 因此应用程序及时关闭 Connection、Session、MessageConsumer 或 MessageProducer 对象（在不需要这些对象时）非常重要。

如果应用程序结束时没有关闭 Connection 对象，那么所有连接的事务性会话会发生隐式回滚。 为了确保提交应用程序所做的所有更改，请在关闭应用程序之前显式关闭 Connection 对象。

请不要使用应用程序中的终结器来关闭 JMS 对象。 因为可能不会调用终结器，所以资源可能无法释放。 Connection 对象关闭时，将关闭从该连接内创建的所有 Session 对象。 同样，Session 对象关闭时，从其中创建的 MessageConsumer 和 MessageProducer 对象也会关闭。 但是，请考虑显式关闭 Session、MessageConsumer 和 MessageProducer 对象以确保资源及时释放。

如果压缩跟踪已激活，System.Halt() 已关闭且不正常，那么不受控制的 JVM 终止很可能会导致跟踪文件损坏。 请尽可能在收集完需要的跟踪信息后关闭跟踪工具。 如果您要跟踪应用程序直至异常结束，请使用未压缩的跟踪输出。

注: 要从队列管理器断开连接，JMS 应用程序需在 Connection 对象上调用 close() 方法。

在 *IBM MQ classes for JMS* 中处理有害消息

有害消息是指接收应用程序无法处理的消息。 如果有害消息已传递到应用程序并且已回滚指定次数，那么 IBM MQ classes for JMS 可将其移到回退队列。

有害消息是指接收应用程序无法处理的消息。 消息可能具有意外类型，或者包含应用程序逻辑无法处理的信息。 如果有害消息传递至应用程序，那么应用程序将无法对其进行处理，并且会将其回滚到源队列。 缺省情况下，IBM MQ classes for JMS 会重复将此消息重新传递至应用程序。 这会导致应用程序陷入不断尝试处理有害消息并将其回滚的循环中。

为防止发生此情况，IBM MQ classes for JMS 可检测有害消息，并将其移至备用目标。 为此，IBM MQ classes for JMS 会使用以下属性：

- 已检测到消息的 MQMD 中 BackoutCount 字段的值。
- 包含消息的输入队列的 IBM MQ 队列属性 **BOTHRESH**（回退阈值）和 **BOQNAME**（回退重新排队队列）。

只要应用程序回滚消息，队列管理器就会自动递增消息的 BackoutCount 字段值。

如果 IBM MQ classes for JMS 检测到的消息具有的 BackoutCount 值大于 0，那么会将 BackoutCount 的值与 **BOTHRESH** 属性值进行比较。

- 如果 BackoutCount 值小于 **BOTHRESH** 属性值，那么 IBM MQ classes for JMS 会将其传递至应用程序以供处理。
- 但是，如果 BackoutCount 值大于或等于 **BOTHRESH** 属性值，那么此消息将被视为有害消息。 在此情况下，IBM MQ classes for JMS 会将此消息移至 **BOQNAME** 属性指定的队列中。 如果无法将消息放入回退队列，那么会将其移动到队列管理器的死信队列或将其废弃，具体取决于消息的报告选项。

注:

- 如果 **BOTHRESH** 属性保留其缺省值 0，那么将禁用有害消息处理操作。 这意味着所有有害消息都将被放回输入队列。
- 另外值得注意的是，IBM MQ classes for JMS 首次检测到 BackoutCount 值大于零的消息时，会查询队列的 **BOTHRESH** 和 **BOQNAME** 属性。 随后将缓存这些属性的值，并在 IBM MQ classes for JMS 遇到 BackoutCount 值大于零的消息时使用。

配置系统以执行有害消息处理

IBM MQ classes for JMS 在查询 **BOTHRESH** 和 **BOQNAME** 属性时所使用的队列取决于所执行消息传递的样式：

- 对于点到点消息传递，这是底层本地队列。 当 JMS 应用程序使用来自别名队列或集群队列的消息时，这一点十分重要。
- 对于发布/预订消息传递，会创建受管队列来保存应用程序的消息。 IBM MQ classes for JMS 会查询受管队列以确定 **BOTHRESH** 和 **BOQNAME** 属性的值。

  受管队列是根据与应用程序预订的“主题”对象所关联的模型队列创建的，并从该模型队列继承 **BOTHRESH** 和 **BOQNAME** 属性的值。 使用的模型队列取决于接收应用程序已取得持久预订还是非持久预订。

  – 用于持久预订的模型队列由“主题”的 **MDURMDL** 属性来指定。 该属性的缺省值为 SYSTEM.DURABLE.MODEL.QUEUE。

– 对于非持久预订，所使用的模型队列由 **MNDURMDL** 属性来指定。 **MNDURMDL** 属性的缺省值为 SYSTEM.NDURABLE.MODEL.QUEUE。

查询 **BOTHRESH** 和 **BOQNAME** 属性时，IBM MQ classes for JMS 会执行以下操作：

- 打开本地队列或别名队列的目标队列。
- 查询 **BOTHRESH** 和 **BOQNAME** 属性。
- 关闭本地队列或别名队列的目标队列。

打开本地队列或别名队列的目标队列时所使用的打开选项，取决于所使用的 IBM MQ classes for JMS 的版本：

- 对于 IBM MQ classes for JMS IBM MQ 9.1.0 Fix Pack 1 和更低版本或者 IBM MQ 9.1.1，如果本地队列或别名队列的目标队列为集群队列，那么 IBM MQ classes for JMS 会使用 MQOO_INPUT_AS_Q_DEF、MQOO_INQUIRE 和 MQOO_FAIL_IF_QUIESCING 选项来打开队列。 这意味着运行接收应用程序的用户必须对集群队列的本地实例具有查询和获取访问权。

  IBM MQ classes for JMS 会使用 MQOO_INQUIRE 和 MQOO_FAIL_IF_QUIESCING 打开选项来打开所有其他类型的本地队列。 为了使 IBM MQ classes for JMS 能够查询属性值，运行接收应用程序的用户必须对本地队列具有查询访问权。
- V 9.1.2 V 9.1.0.2 使用 IBM MQ classes for JMS IBM MQ 9.1.0 Fix Pack 2 和更高版本或者 IBM MQ 9.1.2 和更高版本时，运行接收应用程序的用户必须对本地队列具有查询访问权，而不论队列类型如何。

要将有害消息移至回退重排队列或队列管理器的死信队列，必须授予运行应用程序 put 和 passall 权限的用户。

为同步应用程序处理有害消息

如果应用程序通过调用以下任一方法来同步接收消息，那么 IBM MQ classes for JMS 会在应用程序尝试获取消息时处于活动状态的工作单元内将有害消息重新排队：

- JMSConsumer.receive()
- JMSConsumer.receive(long timeout)
- JMSConsumer.receiveBody(Class<T> c)
- JMSConsumer.receiveBody(Class<T> c, long timeout)
- JMSConsumer.receiveBodyNoWait Class<T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive()
- MessageConsumer.receive(long timeout)
- MessageConsumer.receiveNoWait()
- QueueReceiver.receive()
- QueueReceiver.receive(long timeout)
- QueueReceiver.receiveNoWait()
- TopicSubscriber.receive()
- TopicSubscriber.receive(long timeout)
- TopicSubscriber.receiveNoWait()

这意味着如果应用程序使用事务性 JMS 上下文或会话，那么落实事务后才会落实将消息移动到回退队列的操作。

如果 **BOTHRESH** 属性设置为零之外的值，那么还应设置 **BOQNAME** 属性。 如果 **BOTHRESH** 设置为大于零的值，并且 **BOQNAME** 尚未设置，那么由消息的报告选项来确定相关行为：

- 如果消息已设置 MQRO_DISCARD_MSG 报告选项，那么将丢弃此消息。

- 如果消息指定了报告选项 MQRO_DEAD_LETTER_Q，那么 IBM MQ classes for JMS 会尝试将消息移至队列管理器的死信队列。
- 如果此消息未设置 MQRO_DISCARD_MSG 或 MQRO_DEAD_LETTER_Q，那么 IBM MQ classes for JMS 会尝试将消息放入队列管理器的死信队列。

如果尝试将消息放入死信队列的操作因某些原因而失败，对消息执行的操作取决于接收应用程序是使用事务性还是非事务性 JMS 上下文或会话：

- 如果接收应用程序使用事务性 JMS 上下文或会话，并且已落实事务，那么会丢弃此消息。
- 如果接收应用程序使用事务性 JMS 上下文或会话，并且回滚事务，那么此消息会返回到输入队列。
- 如果接收应用程序已创建非事务性 JMS 上下文或会话，那么会丢弃此消息。

为异步应用程序处理有害消息

如果应用程序通过 MessageListener 异步接收消息，那么 IBM MQ classes for JMS 会将有害消息重新排队，而不会影响消息传递。 重新排队过程在任何与传递至应用程序的实际消息相关联的工作单元外部执行。

如果 **BOTHRESH** 设置为大于零的值，并且 **BOQNAME** 尚未设置，那么由消息的报告选项来确定相关行为：

- 如果消息已设置 MQRO_DISCARD_MSG 报告选项，那么将丢弃此消息。
- 如果消息指定了报告选项 MQRO_DEAD_LETTER_Q，那么 IBM MQ classes for JMS 会尝试将消息移至队列管理器的死信队列。
- 如果此消息未设置 MQRO_DISCARD_MSG 或 MQRO_DEAD_LETTER_Q，那么 IBM MQ classes for JMS 会尝试将消息放入队列管理器的死信队列。

如果尝试将消息放入死信队列的操作因某些原因而失败，那么 IBM MQ classes for JMS 会将消息返回到输入队列。

有关激活规范和 ConnectionConsumer 如何处理有害消息的信息，请参阅在 ASF 中从队列移除消息。

将消息移至回退队列时发生的情况

当有害消息在回退重新排队队列中重新排队时，IBM MQ classes for JMS 会为其添加一个 RFH2 头（如果尚未添加的话），并更新消息描述符 (MQMD) 中的部分字段。

如果有害消息包含 RFH2 头（例如，由于它原先属于 JMS 消息），那么 IBM MQ classes for JMS 会在将消息移至回退重新排队队列时更改 MQMD 中的以下字段：

- BackoutCount 字段重置为零。
- 更新消息的“到期时间”字段，以反映 JMS 应用程序接收有害消息时的剩余到期时间。

如果有害消息不包含 RFH2 头，那么 IBM MQ classes for JMS 会添加一个 RFH2 头，并在回退处理期间更新 MQMD 中的以下字段：

- BackoutCount 字段重置为零。
- 更新消息的“到期时间”字段，以反映 JMS 应用程序接收有害消息时的剩余到期时间。
- 消息的“格式”字段更改为 MQHRF2。
- CCSID 字段更改为 1208。
- “编码”字段修改为 273。

此外，来自有害消息的 CCSID 和“编码”字段将被复制到 RFH2 头的 CCSID 和“编码”字段，以确保回退重新排队队列上消息的头链接是正确的。

相关概念

第 272 页的『在 ASF 中处理有害消息』

在应用程序服务器工具中，处理有害消息的方式与在 IBM MQ classes for JMS 中其他位置处理的方式稍有不同。

IBM MQ classes for JMS 中的异常

IBM MQ classes for JMS 应用程序必须能够处理 JMS API 调用抛出的或发送到异常处理程序的异常。

IBM MQ classes for JMS 通过抛出异常来报告运行时问题。 JMSException 是 JMS 抛出的异常的 root 类，捕获 JMSException 异常能够使用通用的方法来处理与 JMS 有关的所有异常。

每个 JMSException 异常均封装了以下信息：

- 提供程序专用的异常消息，应用程序通过调用 Throwable.getMessage() 方法可获取此异常消息。
- 提供程序专用的错误代码，应用程序通过调用 JMSException.getErrorCode() 方法可获取此错误代码。
- 链接异常。 JMS API 调用抛出的异常通常是基本较低问题所致，此异常由链接到此异常的另一个异常报告。 应用程序通过调用 JMSException.getLinkedException() 或 Throwable.getCause() 方法获取链接异常。

IBM MQ classes for JMS 抛出的大多数异常是 JMSException 的子类实例。 这些子类可实现 com.ibm.msg.client.jms.JmsExceptionDetail 接口，此接口提供以下额外信息：

- 异常消息的解释，应用程序通过调用 JmsExceptionDetail.getExplanation() 方法可获取此解释。
- 推荐用户对异常进行的响应，应用程序通过调用 JmsExceptionDetail.getUserAction() 方法可获取此响应。
- 异常消息中消息插入内容的键。 应用程序通过调用 JmsExceptionDetail.getKeys() 方法可获取所有键的迭代器。
- 异常消息中的消息插入内容。 例如，消息插入内容可能是导致此异常的队列的名称，此内容可以有助于应用程序访问此名称。 应用程序通过调用 JmsExceptionDetail.getValue() 方法获取与指定键对应的消息插入内容。

如果没有可用的详细信息，JmsExceptionDetail 接口中的所有方法可能会返回 null。

例如，如果应用程序尝试为不存在的 IBM MQ 队列创建消息生产者，将抛出带有以下信息的异常：

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

抛出的异常 com.ibm.msg.client.jms.DetailedInvalidDestinationException 是 javax.jms.InvalidDestinationException 的子类，可实现 com.ibm.msg.client.jms.JmsExceptionDetail 接口。

链接异常

链接异常提供了有关运行时问题的更多信息。 因此，对于抛出的每个 JMSException 异常，应用程序应检查链接异常。 链接异常本身可能会包含另一个链接异常，因此，链接异常会形成一个指回原始底层问题的连锁链。 链接异常通过使用 java.lang.Throwable 类的连锁异常机制实现，而应用程序通过调用 Throwable.getCause() 方法获取链接异常。 对于 JMSException 异常，getLinkedException() 方法实际上代表 Throwable.getCause() 方法。

例如，如果应用程序连接队列管理器时指定了不正确的端口号，异常将形成以下连锁链：

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+- -->
com.ibm.mq.MQException
|
+- -->
com.ibm.mq.jmqi.JmqiException
|
+- -->
java.net.ConnectionException
```

通常，连锁链中的每个异常都是从代码的不同层抛出的。 例如，上述连锁链中的异常由以下层抛出：

- 第一个异常是 JMSException 的子类实例，由 IBM MQ classes for JMS 中的公用层排除。
- 下一个异常是 com.ibm.mq.MQException 的实例，由 IBM MQ 消息传递提供程序抛出。

- 下一个异常是 com.ibm.mq.jmqi.JmqiException 的实例，由 MQI 的公用 Java 接口抛出。
- 最后一个异常是 java.net.ConnectionException 的实例，由 Java 类库抛出。

有关 IBM MQ classes for JMS 的分层体系结构的更多信息，请参阅 IBM MQ classes for JMS 体系结构。

通过使用类似于以下代码的代码，应用程序可以迭代此连锁链以抽取所有相应的信息：

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
System.err.println("Caught JMSException");

// Check for linked exceptions in JMSException
Throwable t = je;
while (t != null) {
// Write out the message that is applicable to all exceptions
System.err.println("Exception Msg: " + t.getMessage());
// Write out the exception stack trace
t.printStackTrace(System.err);

// Add on specific information depending on the type of exception
if (t instanceof JMSException) {
JMSException je1 = (JMSException) t;
System.err.println("JMS Error code: " + je1.getErrorCode());

if (t instanceof JmsExceptionDetail){
JmsExceptionDetail jed = (JmsExceptionDetail)je1;
System.err.println("JMS Explanation: " + jed.getExplanation());
System.err.println("JMS Explanation: " + jed.getUserAction());
}
} else if (t instanceof MQException) {
MQException mqe = (MQException) t;
System.err.println("WMQ Completion code: " + mqe.getCompCode());
System.err.println("WMQ Reason code: " + mqe.getReason());
} else if (t instanceof JmqiException){
JmqiException jmqie = (JmqiException)t;
System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
System.err.println("WMQ Msg User Response: "
+ jmqie.getWmqMsgUserResponse());
System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
}

// Get the next cause
t = t.getCause();
}
}
```

请注意，应用程序应始终检查连锁链中每个异常的类型，因为异常类型可能会有变化，而不同的类型封装不同的信息。

获取特定于 IBM MQ 的问题信息

com.ibm.mq.MQException 和 com.ibm.mq.jmqi.JmqiException 的实例封装了特定于 IBM MQ 的问题信息

MQException 异常封装了以下信息：

- 完成代码，应用程序通过调用 getCompCode() 方法可获取此完成代码。
- 原因码，应用程序通过调用 getReason() 方法可获取此原因码。

JmqiException 异常也封装了完成代码和原因码。 但是此外，如果有消息与此异常关联，JmqiException 异常还会封装 AMQ *nnnn* 或 CSQ *nnnn* 消息中的信息。 通过调用异常的相应方法，应用程序可以获取此消息的各个组成部分，如严重性、解释和用户响应。

有关如何使用上述小节所提及的方法的示例，请参阅第 186 页的『链接异常』中的样本代码。

从 IBM MQ classes for JMS 的先前版本升级

与先前版本的 IBM MQ classes for JMS 相比，IBM WebSphere MQ 7.0 中的大多数错误代码和异常消息已发生更改。 发生这些更改的原始是从 IBM WebSphere MQ 7.0 开始，IBM MQ classes for JMS 具有分层结构，而异常是从代码的不同层中抛出的。

例如，如果应用程序尝试连接到不存在的队列管理器，先前版本的 IBM MQ classes for JMS 会抛出带有以下信息的 JMSException 异常：

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

此异常包含一个带有以下信息的链接 MQException 异常：

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

通过在相同情况下的比较， IBM MQ classes for JMS 版本 7.0 抛出 JMSException 异常，并提供以下信息:

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

此异常包含一个带有以下信息的链接 MQException 异常：

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

如果应用程序解析或测试 Throwable.getMessage() 方法返回的异常消息或 JMSException.getErrorCode() 方法返回的错误代码，并且您正在从 IBM WebSphere MQ 7.0 之前的发行版进行升级，那么可能需要修改应用程序以使用 IBM MQ classes for JMS 的 V 7.0 或更高版本。

异常侦听器

应用程序可以向 Connection 对象注册异常侦听器。 之后，如果出现使连接不可用的问题，IBM MQ classes for JMS 会通过调用异常侦听器的 onException() 方法向异常侦听器发送异常。 然后，应用程序便有机会重新建立连接。 此外，如果尝试异步发送消息时出现问题，IBM MQ classes for JMS 也可以将异常发送到异常侦听器。

从 IBM MQ 8.0.0 Fix Pack 2 开始，要维持配置了 JMS MessageListener 和 JMS ExceptionListener 的当前 JMS 应用程序的行为，并确保 IBM MQ classes for JMS 符合 JMS 规范，ASYNC_EXCEPTIONS JMS ConnectionFactory 属性的缺省值会针对 IBM MQ classes for JMS 更改为 ASYNC_EXCEPTIONS_CONNECTIONBROKEN。 因此，缺省情况下，仅向应用程序的 JMS ExceptionListener 传递与中断连接错误代码相对应的异常。

APAR IT14820(包含在 IBM MQ 9.0.0 Fix Pack 1 中) 更新了 IBM MQ classes for JMS ， 以便:

- 针对任何连接中断异常调用由应用程序注册的 ExceptionListener，无论应用程序使用的是同步消息使用者还是异步消息使用者。
- 如果 JMS 会话使用的 TCP/IP 套接字中断，将调用由应用程序注册的 ExceptionListener。
- 当应用程序使用异步消息使用者，并且应用程序使用的 JMS ConnectionFactory 将 ASYNC_EXCEPTIONS 属性设置为 ASYNC_EXCEPTIONS_ALL 值时，在消息传递期间发生的非连接中断异常（例如，MQRC_GET_INHIBITED）将传至该应用程序的 ExceptionListener。

注: 仅为连接中断异常调用一次 ExceptionListener，即使两个 TCP/IP 连接（一个由 JMS 连接使用，一个由 JMS 会话使用）均已中断。

对于其他类型的问题，当前 JMS API 调用会抛出 JMSException 异常。

如果应用程序不向 Connection 对象注册异常侦听器，那么应发送到异常侦听器的所有异常将写入 IBM MQ classes for JMS for JMS 日志。

相关参考

IBM MQ classes for JMS

ASYNCEXCEPTION

从 *IBM MQ classes for JMS* 应用程序访问 *IBM MQ* 功能部件

IBM MQ classes for JMS 提供了一些工具来利用 IBM MQ 的各种功能部件。

注意: 这些功能部件不在 JMS 规范内，或者，某些情况下违反 JMS 规范。 如果您使用这些功能部件，您的应用程序可能会无法兼容其他 JMS 提供程序。 不符合 JMS 规范的功能部件通过注意声明来标记。

在 *IBM MQ classes for JMS* 应用程序中读写消息描述符

可通过设置目标和消息的属性来控制访问消息描述符 (MQMD) 的功能。

某些 IBM MQ 应用程序要求在发送到这些应用程序的消息的 MQMD 中设置特定值。 IBM MQ classes for JMS 提供了一些消息属性，这些属性允许 JMS 应用程序设置 MQMD 字段，从而使 JMS 应用程序“驱动”IBM MQ 应用程序。

您必须将 Destination 对象属性 WMQ_MQMD_WRITE_ENABLED 设置为 true 才能使 MQMD 属性的设置生效。 然后，您可以使用该消息的属性设置方法（例如，setStringProperty）为 MQMD 字段指定值。 除 StrucId 和 Version 之外，将显示所有 MQMD 字段；BackoutCount 可以读取，但不能写入。

以下示例生成的消息将放入 MQMD.UserIdentifier 设置为“JoeBloggs”的队列或主题中。

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

设置 JMS_IBM_MQMD_UserIdentifier 之前需要先设置 WMQ_MQMD_MESSAGE_CONTEXT。 有关使用 WMQ_MQMD_MESSAGE_CONTEXT 的更多信息，请参阅 第 191 页的『JMS 消息对象属性』。

同样，在接收消息然后使用消息的 GET 方法（例如，getStringProperty）之前，可通过将 WMQ_MQMD_READ_ENABLED 设置为 true 来抽取 MQMD 字段的内容。 收到的任何属性均为只读。

以下示例生成的 *value* 字段保存从队列或主题获取的消息的 MQMD.ApplIdentityData 字段的值。

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...
```

```
// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

JMS Destination 对象属性

Destination 对象的两个属性控制对 JMS 中 MQMD 的访问，第三个控制消息上下文。

表 *35:* 属性名称和描述

属性	**缩写**	**描述**
WMQ_MQMD_WRITE_ENABLED	MDW	JMS 应用程序是否可以设置 MQMD 字段的值
WMQ_MQMD_READ_ENABLED	MDR	JMS 应用程序是否可以抽取 MQMD 字段的值
WMQ_MQMD_MESSAGE _CONTEXT	MDCTX	JMS 应用程序要设置的消息上下文级别。 应用程序必须以相应的上下文权限运行才能使属性生效

表 *36:* 属性名称、值和设置方法

属性	**管理工具中的有效值（缺省值以粗体显示）**	**程序中的有效值**	**设置方法**
WMQ_MQMD_WRITE _ENABLED	• **NO** 将忽略所有 JMS_IBM_MQMD* 属性，并且不会将它们的值复制到底层的 MQMD 结构。 • YES 将处理 JMS_IBM_MQMD* 属性。 它们的值将复制到底层的 MQMD 结构。	• **False** • True	setMQMDWriteEnabled
WMQ_MQMD_READ _ENABLED	• **NO** 发送消息时，将不会更新所发送消息中的 JMS_IBM_MQMD* 属性，从而不会反映 MQMD 中已更新的字段值。 接收消息时，在所接收到的消息中不存在任何 JMS_IBM_MQMD* 属性，即使发送方已设置了这些属性的一部分或全部也是如此。 • YES 发送消息时，将更新所发送消息中的所有 JMS_IBM_MQMD* 属性（包括发送方未显式设置的那些属性）以反映 MQMD 中已更新的字段值。 接收消息时，在所接收到的消息中提供了所有 JMS_IBM_MQMD* 属性（包括发送方未显式设置的那些属性）。	• **False** • True	setMQMDReadEnabled

表 *36:* 属性名称、值和设置方法 *(*继续*)*			
属性	**管理工具中的有效值（缺省值以粗体显示）**	**程序中的有效值**	**设置方法**
WMQ_MQMD _MESSAGE_CONTEXT	• **缺省值** MQOPEN API 调用和 MQPMO 结构未指定任何显式消息上下文选项 • SET_IDENTITY_CONTEXT MQOPEN API 调用指定消息上下文选项 MQOO_SET_IDENTITY_CONTEXT，而 MQPMO 结构则指定 MQPMO_SET_IDENTITY_CONTEXT • SET_ALL_CONTEXT MQOPEN API 调用指定消息上下文选项 MQOO_SET_ALL_CONTEXT，而 MQPMO 结构则指定 MQPMO_SET_ALL_CONTEXT	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

JMS 消息对象属性
前缀为 JMS_IBM_MQMD 的消息对象属性允许您设置或读取相应的 MQMD 字段。

发送消息

表示除 StrucId 和 Version 之外的所有 MQMD 字段。 这些属性仅仅是指 MQMD 字段；属性同时出现在 MQMD 和 MQRFH2 头中，不会设置或抽取 MQRFH2 中的版本。

除了 JMS_IBM_MQMD_BackoutCount 之外，以上任意属性都可以设置。 将忽略对 JMS_IBM_MQMD_BackoutCount 设置的任何值。

如果属性具有最大长度限制，而您提供的值过长，则该值会被截断。

对于某些属性，您还必须在 Destination 对象上设置 WMQ_MQMD_MESSAGE_CONTEXT 属性。 应用程序必须以相应的上下文权限运行才能使属性生效。 如果您不将 WMQ_MQMD_MESSAGE_CONTEXT 设置为相应的值，将忽略此属性。 如果您将 WMQ_MQMD_MESSAGE_CONTEXT 设置为相应的值，但是您没有队列管理器的足够上下文权限，系统将发出 JMSException 异常。 以下是需要 WMQ_MQMD_MESSAGE_CONTEXT 的特定值的属性。

以下属性要求 WMQ_MQMD_MESSAGE_CONTEXT 设置为 WMQ_MDCTX_SET_IDENTITY_CONTEXT 或 WMQ_MDCTX_SET_ALL_CONTEXT：

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

以下属性要求 WMQ_MQMD_MESSAGE_CONTEXT 设置为 WMQ_MDCTX_SET_ALL_CONTEXT：

- JMS_IBM_MQMD_PutApplType
- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOriginData

接收消息

如果 WMQ_MQMD_READ_ENABLED 属性设置为 true，那么无论生成应用程序已设置的实际属性是什么，以上所有属性都可用于收到的消息。 根据 JMS 规范，除非事先清除所有属性，否则应用程序不能修改已接收消息的属性。 可以在不修改属性的情况下转发已接收的消息。

注意: 如果您的应用程序从 WMQ_MQMD_READ_ENABLED 属性设置为 true 的目标接收消息，然后将其转发到 WMQ_MQMD_WRITE_ENABLED 设置为 true 的目标，那么已接收消息的所有 MQMD 字段值将复制到转发的消息中。

属性表

下表列出了表示 MQMD 字段的 Message 对象的属性。 请参见链接以获取这些字段及其允许值的完整描述。

表 *37:* 属性名称、描述和类型

属性	描述	Java 类型	完整描述的链接
JMS_IBM_MQMD_Report	报告消息的选项	整数	报告
JMS_IBM_MQMD_MsgType	消息类型	整数	MsgType
JMS_IBM_MQMD_Expiry	消息生命周期	整数	到期
JMS_IBM_MQMD_Feedback	反馈或原因码	整数	FEEDBACK
JMS_IBM_MQMD_Encoding	消息数据的数字编码	整数	编码
JMS_IBM_MQMD_CodedCharSetId	消息数据的字符集标识	整数	CodedCharSetId
JMS_IBM_MQMD_Format	消息数据的格式名称	字符串	FORMAT
JMS_IBM_MQMD_Priority [1]	消息优先级	整数	优先权
JMS_IBM_MQMD_Persistence	消息持久性	整数	PERSISTENCE
JMS_IBM_MQMD_MsgId [2]	消息标识	对象 (byte[]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	相关标识	对象 (byte[]) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	回退计数器	整数	BackoutCount
JMS_IBM_MQMD_ReplyToQ	应答队列的名称	字符串	ReplyToQ
JMS_IBM_MQMD_ReplyToQMgr	应答队列管理器的名称	字符串	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	用户标识	字符串	UserIdentifier
JMS_IBM_MQMD_AccountingToken	记帐标记	对象 (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	与身份有关的应用程序数据	字符串	ApplIdentityData
JMS_IBM_MQMD_PutApplType	放置消息的应用程序的类型	整数	PutApplType
JMS_IBM_MQMD_PutApplName	放置消息的应用程序的名称	字符串	PutApplName
JMS_IBM_MQMD_PutDate	消息的放置日期	字符串	PutDate
JMS_IBM_MQMD_PutTime	消息放置的时间	字符串	PutTime
JMS_IBM_MQMD_ApplOriginData	与源有关的应用程序数据	字符串	ApplOriginData
JMS_IBM_MQMD_GroupId	组标识	对象 (byte[]) [4]	GroupId
JMS_IBM_MQMD_MsgSeqNumber	组内逻辑消息的序列号	整数	MsgSeqNumber

表 37: 属性名称、描述和类型 (继续)			
属性	**描述**	**Java 类型**	**完整描述的链接**
JMS_IBM_MQMD_Offset	从逻辑消息开始的物理消息数据偏移量	整数	偏移量
JMS_IBM_MQMD_MsgFlags	消息标志	整数	MsgFlags
JMS_IBM_MQMD_OriginalLength	原始消息的长度	整数	OriginalLength

1. **注意:** 如果您为 JMS_IBM_MQMD_Priority 指定的值不在 0-9 范围内，将违反 JMS 规范。
2. **注意:** JMS 规范规定消息标识必须由 JMS 提供者设置，且该标识必须为唯一或 null。 如果您为 JMS_IBM_MQMD_MsgId 指定值，此值将复制到 JMSMessageID。 因此，该值不是由 JMS 提供者设置，可能并不唯一，这违反了 JMS 规范。
3. **注意:** 如果您为以字符串“ID”开头的 JMS_IBM_MQMD_CorrelId 指定值，将违反 JMS 规范。
4. **注意:** 在消息上使用字节数组属性违反 JMS 规范。

使用 *IBM MQ classes for JMS* 从应用程序访问 *IBM MQ* 消息数据

您可以使用 IBM MQ classes for JMS 访问应用程序中的完整 IBM MQ 消息数据。 要访问所有数据，消息必须是 JMSBytesMessage。 JMSBytesMessage 的主体包括任何 MQRFH2 头、其他任何 IBM MQ 头和以下消息数据。

将目标的 WMQ_MESSAGE_BODY 属性设置为 WMQ_MESSAGE_BODY_MQ，以接收 JMSBytesMessage 中的所有消息体数据。

如果 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_JMS 或 WMQ_MESSAGE_BODY_UNSPECIFIED，那么消息体返回时不带 JMS MQRFH2 头，而 JMSBytesMessage 的属性反映 RFH2 中设置的属性。

某些应用程序不能使用本主题中描述的功能。 如果应用程序连接到 IBM MQ V6 队列管理器，或者将 PROVIDERVERSION 设置为 6，那么这些功能不可用。

发送消息

在发送消息时，目标属性 WMQ_MESSAGE_BODY 的优先级高于 WMQ_TARGET_CLIENT。

如果 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_JMS，那么 IBM MQ classes for JMS 会根据 JMSMessage 属性和头字段的设置自动生成 MQRFH2 头。

如果 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_MQ，那么不会向消息体添加额外的头

如果 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_UNSPECIFIED，IBM MQ classes for JMS 会发送 MQRFH2 头，除非 WMQ_TARGET_CLIENT 设置为 WMQ_TARGET_DEST_MQ。 接收时，将 WMQ_TARGET_CLIENT 设置为 WMQ_TARGET_DEST_MQ 会导致所有 MQRFH2 头从消息体中移除。

注: JMSBytesMessage 和 JMSTextMessage 不需要 MQRFH2，但是 JMSStreamMessage、JMSMapMessage 和 JMSObjectMessage 需要。

WMQ_MESSAGE_BODY_UNSPECIFIED 是 WMQ_MESSAGE_BODY 的缺省设置，WMQ_TARGET_DEST_JMS 是 WMQ_TARGET_CLIENT 的缺省设置。

如果发送 JMSBytesMessage，那么在构造 IBM MQ 消息时，可以覆盖 JMS 消息体的缺省设置。 请使用以下属性:

- JMS_IBM_Format 或 JMS_IBM_MQMD_Format: 该属性指定 IBM MQ 头的格式；如果没有前置 WebSphere MQ 头，该属性指定 JMS 消息体开头的应用程序有效内容的格式。
- JMS_IBM_Character_Set 或 JMS_IBM_MQMD_CodedCharSetId: 此属性指定启动 JMS 消息体的 IBM MQ 头或应用程序有效内容的 CCSID (如果没有前面的 WebSphere MQ 头)。
- JMS_IBM_Encoding 或 JMS_IBM_MQMD_Encoding: 该属性指定 IBM MQ 头的编码；如果没有前置 WebSphere MQ 头，该属性指定 JMS 消息体开头的应用程序有效内容的编码。

如果同时指定两种类型的属性，JMS_IBM_MQMD_* 属性将覆盖对应的 JMS_IBM_* 属性，前提是目标属性 WMQ_MQMD_WRITE_ENABLED 设置为 true。

使用 JMS_IBM_MQMD_* 设置消息属性和 JMS_IBM_* 设置消息属性之间效果差别非常大：

1. JMS_IBM_MQMD_* 属性特定于 IBM MQ JMS 提供程序。
2. JMS_IBM_MQMD_* 属性只能在 MQMD 中设置。 仅当消息没有 MQRFH2 JMS 头时，才会在 MQMD 中设置 JMS_IBM_* 属性。 否则，此属性将在 JMS RFH2 头中设置。
3. JMS_IBM_MQMD_* 属性对写入 JMSMessage 的文本和数字的编码没有影响。

   接收方应用程序可能会假设 MQMD.Encoding 和 MQMD.CodedCharSetId 的值与消息体中数字和文本的编码和字符集对应。 如果使用 JMS_IBM_MQMD_* 属性，那么发送应用程序有责任使其对应。 消息体中数字和文本的编码和字符集由 JMS_IBM_* 属性设置。

   第 194 页的图 45 中的错误代码片段将发送以字符集 1208 编码的消息，同时 MQMD.CodedCharSetId 设置为 37。

   a. 发送编码错误的消息

```
TextMessage tmo = session.createTextMessage();
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQDestination)destination).setMQMDWriteEnabled(true);
tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
tmo.setText("String one");
producer.send(tmo);
```

   b. 依靠 MQMD.CodedCharSetId 值设置的 JMS_IBM_CHARACTER_SET 值接收消息：

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

   c. 生成的输出：

```
Message is "éÈÊ'>...??>?"
```

   图 *45:* 编码不一致的 *MQMD* 和消息数据

第 194 页的图 46 中的任一代码片段都会使消息放入队列或主题，同时其消息体中包含应用程序有效内容，该有效内容不包含自动生成的要添加的 MQRFH2 头。

1. 设置 WMQ_MESSAGE_BODY_MQ：

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. 设置 WMQ_TARGET_DEST_MQ：

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

图 *46:* 发送包含 *MQ* 消息体的消息。

接收消息

如果 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_JMS，那么入站 JMS 消息类型和消息体由所接收的 WebSphere MQ 消息的内容确定。 消息类型和消息体由 MQRFH2 头中的字段决定；如果没有 MQRFH2，那么由 MQMD 中的字段确定。

如果 WMQ_MESSAGE_BODY 设置为 WMQ_MESSAGE_BODY_MQ，那么入站 JMS 消息类型为 JMSBytesMessage。 JMS 消息体是底层 MQGET API 调用返回的消息数据。 消息体的长度为 MQGET 调用返回的长度。 消息体中的数据的字符集和编码由 MQMD 的 CodedCharSetId 和 Encoding 字段确定。 消息体中的数据的格式由 MQMD 的 Format 字段确定

如果 WMQ_MESSAGE_BODY 设置为缺省值 WMQ_MESSAGE_BODY_UNSPECIFIED，那么 IBM MQ classes for JMS 会将其设置为 WMQ_MESSAGE_BODY_JMS。

在接收 JMSBytesMessage 消息时，您可以参考以下属性对消息进行解码：

- JMS_IBM_Format 或 JMS_IBM_MQMD_Format：该属性指定 IBM MQ 头的格式；如果没有前置 WebSphere MQ 头，该属性指定 JMS 消息体开头的应用程序有效内容的格式。
- JMS_IBM_Character_Set 或 JMS_IBM_MQMD_CodedCharSetId: 此属性指定启动 JMS 消息体的 IBM MQ 头或应用程序有效内容的 CCSID (如果没有前面的 WebSphere MQ 头)。
- JMS_IBM_Encoding 或 JMS_IBM_MQMD_Encoding：该属性指定 IBM MQ 头的编码；如果没有前置 WebSphere MQ 头，该属性指定 JMS 消息体开头的应用程序有效内容的编码。

以下代码片段会使收到的消息为 JMSBytesMessage。 不管收到的消息内容以及收到的 MQMD 的字段内容如何，此消息都是 JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

目标属性 *WMQ_MESSAGE_BODY*
WMQ_MESSAGE_BODY 用于决定 JMS 应用程序是否将 IBM MQ 消息的 MQRFH2 作为消息有效内容的一部分处理（即，是否作为 JMS 消息体的一部分）。

表 *38:* 属性名称和描述

属性	缩写	描述
WMQ_MESSAGE_BODY	MBODY	JMS 应用程序是否将 IBM MQ 消息的 MQRFH2 作为消息有效内容的一部分处理（即，是否作为 JMS 消息体的一部分）。

表 39: 属性名称、值和设置方法			
属性	**管理工具中的有效值（缺省值以粗体显示）**	**程序中的有效值**	**设置方法**
WMQ_MESSAGE_BODY	• **UNSPECIFIED** 发送时，IBM MQ classes for JMS 是否生成并包含 MQRFH2 头取决于 WMQ_TARGET_CLIENT 的值。 接收时，充当值 JMS。 • JMS 发送时，IBM MQ classes for JMS 自动生成 MQRFH2 头并将其包含在 IBM MQ 消息中。 接收时，IBM MQ classes for JMS 按照 MQRFH2（如果存在）中的值设置 JMS 消息属性；它不会提供 MQRFH2 作为 JMS 消息体的一部分。 • MQ 发送时，IBM MQ classes for JMS 不生成 MQRFH2。 接收时，IBM MQ classes for JMS 会提供 MQRFH2 作为 JMS 消息体的一部分。	• **WMQ_MESSAGE_BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS 持久消息

IBM MQ classes for JMS 应用程序可以使用 **NonPersistentMessageClass** 队列属性为 JMS 持久消息提供更好的性能，但会牺牲一些可靠性。

IBM MQ 队列具有名为 **NonPersistentMessageClass** 的属性。 此属性的值确定队列管理器重新启动时是否废弃队列上的非持久消息。

您可以通过 IBM MQ 脚本 (MQSC) 命令 DEFINE QLOCAL，使用以下任一参数设置本地队列的属性：

NPMCLASS(NORMAL)
队列管理器重新启动时将废弃队列上的非持久消息。 这是缺省值。

NPMCLASS(HIGH)
队列管理器在静止或立即关闭后重新启动时不会废弃队列上的非持久消息。 但是，抢先关闭或故障后，非持久性消息可能会被废弃。

本主题描述 IBM MQ classes for JMS 应用程序如何使用此队列属性为 JMS 持久消息提供更好的性能。

Queue 或 Topic 对象的 PERSISTENCE 属性可以具有值 HIGH。 您可以使用 IBM MQ JMS 管理工具设置此值，或者应用程序可以调用 Destination.setPersistence() 方法作为参数传递值 WMQConstants.WMQ_PER_NPHIGH。

如果应用程序向 PERSISTENCE 属性值为 HIGH 的目标发送 JMS 持久消息或 JMS 非持久消息，并且底层 IBM MQ 队列设置为 NPMCLASS(HIGH)，那么该消息将作为 IBM MQ 非持久消息放入队列。 如果目标的 PERSISTENCE 属性值不是 HIGH，或者如果底层队列设置为 NPMCLASS(NORMAL)，JMS 持久消息将作为 IBM MQ 持久消息放入队列，而 JMS 非持久消息将作为 IBM MQ 非持久消息放入队列。

如果 JMS 持久消息作为 IBM MQ 非持久消息放入队列，并且您希望确保队列管理器静止或立即关闭后不会废弃此消息，那么该消息可能经过的所有队列都必须设置为 NPMCLASS(HIGH)。 在发布/预订域中，这些队列包括订户队列。 As an aid to enforcing this configuration, IBM MQ classes for JMS throws an InvalidDestinationException if an application tries to create a message consumer for a destination where

the PERSISTENCE property has the value HIGH and the underlying IBM MQ queue is set to NPMCLASS(NORMAL).

将目标的 PERSISTENCE 属性设置为 HIGH 不会影响从该目标接收消息的方式。 作为 JMS 持久消息发送的消息将作为 JMS 持久消息接收，而作为 JMS 非持久消息发送的消息将作为 JMS 非持久消息接收。

当应用程序向 PERSISTENCE 属性值为 HIGH 的目标发送第一条消息时，或者应用程序为 PERSISTENCE 属性值为 HIGH 的目标创建第一个消息使用者时，IBM MQ classes for JMS 会发出 MQINQ 调用来确定底层 IBM MQ 队列上是否设置了 NPMCLASS(HIGH)。 因此，应用程序必须有权问询此队列。 此外，删除目标前，IBM MQ classes for JMS 会一直保留 MQINQ 调用的结果，不会再发出其他 MQINQ 调用。 因此，如果在应用程序仍使用此目标时更改底层队列上的 NPMCLASS 设置，IBM MQ classes for JMS 不会注意到新设置。

通过允许将 JMS 持久消息作为 IBM MQ 非持久消息放入 IBM MQ 队列，您将以可靠性为代价获取更高的性能。 如果您需要最大程度的 JMS 持久消息可靠性，请不要将消息发送到 PERSISTENCE 属性值为 HIGH 的目标。

JMS 层可以使用 SYSTEM.JMS.TEMPQ.MODEL，而不是 SYSTEM.DEFAULT.MODEL.QUEUE。 因为 SYSTEM.DEFAULT.MODEL.QUEUE 无法接受持久消息，所以 SYSTEM.JMS.TEMPQ.MODEL 会创建接受持久消息的永久动态队列。 要使用临时队列来接受持久消息，必须使用 SYSTEM.JMS.TEMPQ.MODEL，或者将模型队列更改为您选择的替代队列。

对 *IBM MQ classes for JMS* 使用 *TLS*

IBM MQ classes for JMS 应用程序可以使用传输层安全性 (TLS) 加密。 为此，这些应用程序需要 JSSE 提供程序。

使用 TRANSPORT(CLIENT) 的 IBM MQ classes for JMS 连接支持 TLS 加密。 TLS 提供通信加密、认证和消息完整性。 它通常用于保护因特网或内部网上任何两个对等实体之间的通信。

IBM MQ classes for JMS 使用 Java 安全套接字扩展 (JSSE) 处理 TLS 加密，因此需要 JSSE 提供程序。 JSE v1.4 JVM 内置了 JSSE 提供程序。 有关如何管理和存储证书的详细信息根据提供程序不同而有所变化。 有关信息，请参阅 JSSE 提供程序的文档。

本部分假设您已正确安装和配置了 JSSE 提供程序，同时安装了合适的证书，并使其可供 JSSE 提供程序使用。 现在，您可以使用 JMSAdmin 设置大量管理属性。

如果您的 IBM MQ classes for JMS 应用程序使用客户机通道定义表 (CCDT) 连接到队列管理器，请参阅 第 231 页的『将客户机通道定义表用于 IBM MQ classes for JMS』。

SSLCIPHERSUITE 对象属性

设置 SSLCIPHERSUITE 可在 ConnectionFactory 对象上启用 TLS 加密。

要在 ConnectionFactory 对象上启用 TLS 加密，请使用 JMSAdmin 将 SSLCIPHERSUITE 属性设置为 JSSE 提供程序支持的 CipherSuite。 此 CipherSuite 必须与目标通道上设置的 CipherSpec 匹配。 但是，CipherSuite 与 CipherSpec 不同，因此具有不同的名称。 第 200 页的『IBM MQ classes for JMS 中的 TLS CipherSpec 和 CipherSuite』包含一个表，该表将 IBM MQ 支持的 CipherSpec 映射到 JSSE 知道的对等 CipherSuite。 有关 IBM MQ 的 CipherSpec 和 CipherSuite 的更多信息，请参阅保护 IBM MQ。

例如，要设置 ConnectionFactory 对象（该对象可用于创建连接，并且这一连接是基于一个支持 TLS 且 CipherSpec 为 TLS_RSA_WITH_AES_128_CBC_SHA 的 MQI 通道），请向 JMSAdmin 发出以下命令：

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

也可以从应用程序中使用 MQConnectionFactory 对象的 setSSLCipherSuite() 方法进行设置。

为了方便起见，如果 SSLCIPHERSUITE 属性上指定了 CipherSpec，JMSAdmin 会尝试将 CipherSpec 映射到相应的 CipherSuite 并发出警告。 如果该属性由应用程序指定，则不会尝试此映射。

或者，您可以使用使用定义表 (CCDT)。 有关更多信息，请参阅 第 231 页的『将客户机通道定义表用于 IBM MQ classes for JMS』。

SSLFIPSREQUIRED 对象属性

如果您要求连接使用 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS) 支持的 CipherSuite，请将连接工厂的 SSLFIPSREQUIRED 属性设置为 YES。

此属性的缺省值为 NO，表示连接可以使用 IBM MQ 支持的任何 CipherSuite。

如果应用程序使用多个连接，那么应用程序创建第一个连接时使用的 SSLFIPSREQUIRED 值决定了应用程序创建任何后续连接时要使用的值。 这表示用于创建后续连接的连接工厂的 SSLFIPSREQUIRED 属性值将被忽略。 如果您希望使用 SSLFIPSREQUIRED 的其他值，则必须重新启动应用程序。

应用程序可通过调用 ConnectionFactory 对象的 setSSLFipsRequired() 方法设置此属性。 如果未设置 CipherSuite，那么会忽略此属性。

相关任务

指定运行时在 MQI 客户机上仅使用经过 FIPS 认证的 CipherSpecs

相关参考

针对 UNIX, Linux, and Windows 的美国联邦信息处理标准 (FIPS)

SSLPEERNAME 对象属性

使用 SSLPEERNAME 指定专有名称模式，以确保您的 JMS 应用程序连接到正确的队列管理器。

JMS 应用程序可以通过指定专用名称 (DN) 模式来确保连接到正确的队列管理器。 只有队列管理器提交与模式匹配的 DN 时，连接才能成功。 有关此模式的格式的更多详细信息，请参阅相关主题。

DN 是使用 ConnectionFactory 对象的 SSLPEERNAME 属性设置的。 例如，以下 JMSAdmin 命令设置的 ConnectionFactory 对象期望队列管理器通过特定格式的公用名称识别自己，该名称以 `QMGR.` 字符开头，至少包含两个组织单元名称，第一个必须是 IBM，第二个必须是 WEBSPHERE：

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

检查不区分大小写，可以使用分号代替逗号。 也可以使用 MQConnectionFactory 对象上的 setSSLPeerName() 方法从应用程序中设置 SSLPEERNAME。 如果不设置此属性，则不会对队列管理器提供的专用名称执行检查。 如果未设置 CipherSuite，那么会忽略此属性。

SSLCERTSTORES 对象属性

使用 SSLCERTSTORES 指定用于证书撤销列表 (CRL) 检查的 LDAP 服务器列表。

通常使用证书撤销列表 (CRL) 来识别不再信任的证书。 CRL 通常托管在 LDAP 服务器上。 JMS 允许在 Java 2 v1.4 或更高版本下为 CRL 检查指定 LDAP 服务器。 以下 JMSAdmin 示例指示 JMS 使用名为 crl1.ibm.com 的 LDAP 服务器上托管的 CRL：

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

注: 要将 CertStore 成功用于 LDAP 服务器上托管的 CRL，请确保您的 Java 软件开发包 (SDK) 与此 CRL 兼容。 某些 SDK 要求 CRL 符合 RFC 2587（定义了 LDAP V2 的模式）。 大部分 LDAP V3 服务器改为使用 RFC 2256。

如果您的 LDAP 服务器不在缺省端口 389 上运行，您可以通过在主机名上追加冒号 (:) 和端口号来指定端口。 如果队列管理器提交的证书存在于 crl1.ibm.com 上托管的 CRL 中，则此连接未完成。 为了避免单点故障，JMS 允许通过提供 LDAP 服务器列表（由空格字符分隔）来提供多个 LDAP 服务器。 例如：

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

指定了多个 LDAP 服务器时，JMS 会轮流尝试每个服务器，直至找到可以通过其成功验证队列管理器证书的服务器。 每个服务器都必须包含相同的信息。

此格式的字符串可以由应用程序在 MQConnectionFactory.setSSLCertStores() 方法中提供。 或者，应用程序可以创建一个或多个 java.security.cert.CertStore 对象，并将这些对象放在合适的 Collection 对象中，然后将此 Collection 对象提供给 setSSLCertStores() 方法。 通过这种方式，应用程序可以定制 CRL 检查。 请参阅您的 JSSE 文档获取有关构造和使用 CertStore 对象的详细信息。

设置连接时队列管理器提交的证书按照如下过程进行验证：

1. sslCertStores 所识别的集合中的第一个 CertStore 对象用于识别 CRL 服务器。
2. 尝试联系 CRL 服务器。

3. 如果尝试成功，那么将搜索服务器以便匹配证书。
   a. 如果发现证书已撤销，那么搜索流程结束，连接请求失败，并出现原因码 MQRC_SSL_CERTIFICATE_REVOKED。
   b. 如果未找到证书，那么搜索流程结束，且允许继续处理连接。
4. 如果尝试联系服务器失败，那么会使用下一个 CertStore 对象来识别 CRL 服务器，并自步骤 2 起重复流程。

   如果这是 Collection 中的最后一个 CertStore，或者如果 Collection 不包含 CertStore 对象，那么搜索过程失败并且连接请求将失败，原因码为 MQRC_SSL_CERT_STORE_ERROR。

集合对象将决定使用 CertStore 的顺序。

如果您的应用程序使用 setSSLCertStores() 设置一组 CertStore 对象，那么 MQConnectionFactory 不能再绑定到 JNDI 名称空间。 尝试执行此操作会导致异常。 如果不设置 sslCertStores 属性，则不会对队列管理器提供的证书执行撤销检查。 如果未设置 CipherSuite，那么会忽略此属性。

SSLRESETCOUNT 对象属性
此属性表示在重新协商用于加密的密钥之前连接所发送和接收的字节总数。

发送的字节数是加密之前的字节数，接收的字节数是解密之后的字节数。 字节数还包含 IBM MQ classes for JMS 发送和接收的控制信息。

例如，要使用在流动的数据达到 4 MB 后重新协商的密钥配置 ConnectionFactory 对象（可用于创建通过启用 TLS 的 MQI 通道的连接），请向 JMSAdmin 发出以下命令：

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

应用程序可通过调用 ConnectionFactory 对象的 setSSLResetCount() 方法设置此属性。

如果此属性的值为零（缺省值），那么永不会重新协商密钥。 如果未设置 CipherSuite，那么会忽略此属性。

SSLSocketFactory 对象属性
要定制应用程序的 TLS 连接的其他方面，请创建 SSLSocketFactory 并配置 JMS 以使用此属性。

您可能希望定制应用程序的 TLS 连接的其他方面。 例如，可能希望初始化加密硬件或更改使用的密钥库和信任库。 要执行此操作，应用程序必须首先创建相应进行了定制的 javax.net.ssl.SSLSocketFactory 对象。 请参阅 JSSE 文档，以获取有关执行此操作的信息，因为可定制功能部件根据提供程序不同而有所差异。 获取适合的 SSLSocketFactory 对象后，使用 MQConnectionFactory.setSSLSocketFactory() 方法将 JMS 配置为使用定制的 SSLSocketFactory 对象。

如果应用程序使用 etSSLSocketFactory() 方法设置定制 SSLSocketFactory 对象，那么不再可将 MQConnectionFactory 对象绑定到 JNDI 名称空间。 尝试执行此操作会导致异常。 如果未设置此属性，那么会使用缺省 SSLSocketFactory 对象。 请参阅 JSSE 文档，以获取缺省 SSLSocketFactory 对象行为的详细信息。 如果未设置 CipherSuite，那么会忽略此属性。

要点: 请勿认为使用 SSL 属性可在从本身并不安全的 JNDI 名称空间检索 ConnectionFactory 对象时确保安全性。 尤其是，JNDI 的标准 LDAOP 实现并不安全。 攻击者可能会模仿 LDAP 服务器，将 JMS 应用程序错误地连接到不正确的服务器，而不发出通知。 通过恰当的安全措施，可以安全地执行 JDNI 的其他实现（例如，fscontext 实现）。

更改 *JSSE* 密钥库或信任库
如果更改密钥库或信任库，那么必须执行特定操作，才能获取更改。

如果更改 JSSE 密钥库或信任库的内容，或者更改密钥库或信任库文件的位置，那么当时正在运行的 IBM MQ classes for JMS 应用程序不会自动获取更改。 要使更改生效，必须执行以下操作：

- 应用程序必须关闭其所有连接，并破坏连接池中任何未使用的连接。
- 如果 JSSE 提供程序高速缓存来自密钥库和信任库的信息，那么必须刷新此信息。

执行这些操作后，应用程序可以重新创建其连接。

根据应用程序的设计方式以及 JSSE 提供程序所提供的功能，或许能够在不停止并重新启动应用程序的情况下执行这些操作。 然而，停止并重新启动应用程序可能是最简单的解决方案。

IBM MQ classes for JMS 中的 *TLS CipherSpec* 和 *CipherSuite*
IBM MQ classes for JMS 应用程序与队列管理器建立连接的能力取决于在 MQI 通道的服务器端指定的 CipherSpec，以及在客户端指定的 CipherSuite。

下表列出了 IBM MQ 支持的 CipherSpec 及其对应的 CipherSuite。

您应查看主题不推荐的 CipherSpec，以了解 IBM MQ 是否不推荐使用下表中列出的任何 CipherSpec，如果情况如此，请查看不推荐 CipherSpec 的更新。

要点: 列出的 CipherSuite 是 IBM MQ 随附的 IBM Java 运行时环境 (JRE) 支持的 CipherSuite。 列出的 CipherSuite 包括 Oracle Java JRE 所支持的 CipherSuite。 有关配置应用程序以使用 Oracle Java JRE 的更多信息，请参阅配置应用程序以使用 IBM Java 或 Oracle Java CipherSuite 映射。

下表还指明了用于通信的协议，以及 CipherSuite 是否符合 FIPS 140-2 标准。

如果应用程序未配置为强制实施 FIPS 140-2 合规性，可以使用标记为符合 FIPS 140-2 标准的 Ciphersuite，但如果为应用程序配置了 FIPS 140-2 合规性（请参阅下面有关配置的注释），那么只有标记为符合 FIPS 140-2 标准的那些 CipherSuite 才能配置；尝试使用其他 CipherSuite 会导致错误。

注: 每个 JRE 可以具有多个加密安全提供程序，其中每个提供程序都可以提供同一个 CipherSuite 的实现。 然而，并非所有安全提供程序都是经过 FIPS 140-2 认证的。 如果未对应用程序强制实施 FIPS 140-2 合规性，可能会使用未经过认证的 CipherSuite 实现。 未经过认证的实现可能无法与 FIPS 140-2 兼容，即使 CipherSuite 在理论上符合该标准要求的最低安全级别也不例外。请参阅以下说明，以获取有关在 IBM MQ JMS 应用程序中配置 FIPS 140-2 实施的更多信息。

有关 CipherSpec 和 CipherSuite 的 FIPS 140-2 和 Suite-B 合规性的更多信息，请参阅指定 CipherSpec。 您可能还需要了解有关美国联邦信息处理标准的信息。

要使用完整的 CipherSuite 集，并使用经过认证的 FIPS 140-2 和/或 Suite-B 合规性进行操作，需要一个合适的 JRE。IBM Java 7 服务刷新 4 修订包 2 或更高级别的 IBM JRE 为 第 201 页的表 40 中列出的 TLS 1.2 CipherSuites 提供相应支持。

V 9.1.5 为了能够使用 TLS v1.3 密码，运行您的应用程序的 JRE 必须支持 TLS v1.3。

注: 要使用某些 CipherSuite，需要在此 JRE 中配置“无限制”策略文件。 有关如何在 SDK 或 JRE 中设置策略文件的更多详细信息，请参阅您正在使用的版本的 *Security Reference for IBM SDK Java Technology Edition* 中的 *IBM SDK* 策略文件 主题。

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	否

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	否

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	否

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	否

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 227 页的『1』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	否
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	否

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	否
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	yes

表 40: IBM MQ 支持的 CipherSpec 及其对应的 CipherSuite (继续)

CipherSpec	对应的 CipherSuite (IBM JRE)	对应的 CipherSuite (Oracle JRE)	协议	是否符合 FIPS 140-2 标准
V 9.1.5 TLS_AES_128_GCM_SHA256 第 227 页的『2』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	否
V 9.1.5 TLS_AES_256_GCM_SHA384 第 227 页的『2』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	否

表 *40: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite* (继续)

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
V 9.1.5 TLS_CHACHA20_POLY1305_SHA256 第 227 页的『2』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	否

表 40: IBM MQ 支持的 CipherSpec 及其对应的 CipherSuite (继续)

CipherSpec	对应的 CipherSuite (IBM JRE)	对应的 CipherSuite (Oracle JRE)	协议	是否符合 FIPS 140-2 标准
V 9.1.5 TLS_AES_128_CCM_SHA256 第 227 页的『2』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	否
V 9.1.5 TLS_AES_128_CCM_8_SHA256 第 227 页的『2』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	否
V 9.1.5 ANY 第 227 页的『2』	*ANY	*ANY	多个	否

表 40: *IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite* (继续)

CipherSpec	对应的 CipherSuite (IBM JRE)	对应的 CipherSuite (Oracle JRE)	协议	是否符合 FIPS 140-2 标准
V 9.1.5 ANY_TLS13 第 227 页的『2』	*TLS13	*TLS13	TLS V13	否
V 9.1.5 ANY_TLS12_OR_HIGHER 第 227 页的『2』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS V1.2 及更高版本	否
V 9.1.5 ANY_TLS13_OR_HIGHER 第 227 页的『2』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS V1.3 及更高版本	否

注意:

1. 不推荐 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。
2. V 9.1.5 为了能够使用 TLS v1.3 密码，运行您的应用程序的 JRE 必须支持 TLS v1.3

在 IBM MQ classes for JMS 应用程序中配置 Ciphersuite 与 FIPS 合规性

- 使用 IBM MQ classes for JMS 的应用程序可以使用以下两种方法中的任何一种来设置 CipherSuite 以进行连接:

  - 调用 ConnectionFactory 对象的 setSSLCipherSuite 方法。
  - 使用 IBM MQ JMS 管理工具设置 ConnectionFactory 对象的 SSLCIPHERSUITE 属性。
- 使用 IBM MQ classes for JMS 的应用程序可以使用以下两种方法中的任何一种来强制实施 FIPS 140-2 合规性：
  - 调用 ConnectionFactory 对象的 setSSLFipsRequired 方法。
  - 使用 IBM MQ JMS 管理工具设置 ConnectionFactory 对象的 SSLFIPSREQUIRED 属性。

配置应用程序以使用 IBM Java 或 Oracle Java CipherSuite 映射

您可以配置应用程序是使用缺省的 IBM Java CipherSuite 到 IBM MQ CipherSpec 的映射，还是使用 Oracle CipherSuite 到 IBM MQ CipherSpec 的映射。 因此，无论应用程序是使用 IBM JRE 还是 Oracle JRE，您都可以使用 TLS CipherSuite。 Java 系统属性 `com.ibm.mq.cfg.useIBMCipherMappings` 控制使用哪些映射。 该属性可以具有以下某个值：

true
使用 IBM Java CipherSuite 到 IBM MQ CipherSpec 的映射。

该值为缺省值。

false
使用 Oracle CipherSuite 到 IBM MQ CipherSpec 的映射。

有关使用 IBM MQ Java 和 TLS 密码的更多信息，请参阅 MQdev 博客帖子 MQ Java， TLS 密码，非 IBM JRE 和 APAR IT06775， IV66840， IT09423， IT10837。

互操作性限制

根据所使用的协议，某些 CipherSuite 可能与多个 IBM MQ CipherSpec 兼容。 但是， 仅支持使用表 1 中指定的 TLS 版本的 CipherSuite/CipherSpec 组合。 尝试使用不支持的 CipherSuite 和 CipherSpec 组合会导致失败，并出现相应异常。 使用任何这些 CipherSuite/CipherSpec 组合的安装都应转为使用支持的组合。

下表显示了受此限制影响的 CipherSuite。

表 *41: CipherSuite* 及其支持和不支持的 *CipherSpec*

CipherSuite	**支持的 TLS CipherSpec**	**不支持的 SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 228 页的『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

注:

1. 不推荐此 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。

使用 *Java* 为 *IBM MQ classes for JMS* 编写通道出口
通过定义实现指定接口的 Java 类创建通道出口。

在 com.ibm.mq.exits 包中定义了三个接口：

- WMQSendExit，针对发送出口
- WMQReceiveExit，针对接收出口
- WMQSecurityExit，针对安全出口

以下样本代码定义了一个类，它实现了所有三个接口：

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the security exit here
  }
}
```

每个出口都将 MQCXP 对象和 MQCD 对象作为参数接收。 这些对象表示在过程接口中定义的 MQCXP 和 MQCD 结构。

调用发送出口时，agentBuffer 参数包含将发送到服务器队列管理器的数据。 由于 agentBuffer.limit() 表达式提供了数据长度，因此不需要 length 参数。 发送出口返回将发送到服务器队列管理器的数据作为其值。 但是，如果此发送出口不是发送出口序列中最后一个发送出口，那么改为将返回的数据传递到序列中下一个发送出口。 发送出口可返回其在 agentBuffer 参数中接收的数据的修改版本，也可返回保持不变的数据。 因此，最简单的出口代码是：

```
{ return agentBuffer; }
```

调用接收出口时，agentBuffer 参数包含已从服务器队列管理器接收的数据。 接收出口返回将由 IBM MQ classes for JMS 传递到应用程序的数据作为其值。 但是，如果此接收出口不是接收出口序列中最后一个接收出口，那么改为将返回的数据传递到序列中下一个接收出口。

调用安全出口时，agentBuffer 参数包含来自连接的服务器端安全出口的安全流中接收的数据。 安全出口返回将在安全流中发送到服务器安全出口的数据作为其值。

通道出口使用具有后备数组的缓冲区调用。 要获取最佳性能，出口必须返回具有后备数组的缓冲区。

调用通道出口时，最多可以向其传递包含 32 个字符的用户数据。 此出口通过调用 MQCXP 对象的 getExitData() 方法访问用户数据。 尽管此出口可通过调用 setExitData() 方法更改用户数据，但是每次调用出口时都会刷新用户数据。 因此，会丢失对用户数据所作的任何更改。 但是，出口可以通过使用 MQCXP 对象的出口用户区域将数据从一个调用传递到下一个调用。 出口通过调用 getExitUserArea() 方法按引用访问出口用户区域。

每个出口类必须具有一个构造函数。 构造函数可以是上一个示例中显示的缺省构造函数，也可以是具有字符串参数的构造函数。 将调用构造函数，为类中定义的每个出口创建出口类的实例。 因此，在先前示例中，为发送出口创建 MyMQExits 类的一个实例，为接收出口创建另一个实例，并为安全出口创建第三个实例。 调用具有字符串参数的构造函数时，该参数包含传递到将为其创建实例的通道出口的相同用户数据。 如果出口类同时包含缺省构造函数和单个参数构造函数，那么此单个参数构造函数将优先。

请勿从通道出口内部关闭连接。

将数据发送到连接的服务器端时，会在调用任何通道出口之后执行 TLS 加密。 相似地，从连接的服务器端接收数据时，会在调用任何通道出口之前执行 TLS 解密。

在低于 7.0 的 IBM MQ classes for JMS 版本中，通道出口是使用接口 MQSendExit， MQReceiveExit 和 MQSecurityExit 实现的。 您仍可使用这些接口，但首选使用新接口来改进功能和性能。

配置 *IBM MQ classes for JMS* 以使用通道出口

IBM MQ classes for JMS 应用程序可以在其连接到队列管理器时启动的 MQI 通道上使用通道安全出口、发送出口和接收出口。 应用程序可以使用以 Java、C 或 C++ 编写的出口。应用程序还可以使用连续运行的发送或接收出口序列。

以下属性用于指定供 JMS 连接使用的一个发送出口或一系列发送出口。

- MQConnectionFactory 对象的 **SENDEXIT** 属性。
- IBM MQ 资源适配器用于入站通信的激活规范上的 **sendexit** 属性。
- IBM MQ 资源适配器用于输出通信的 ConnectionFactory 对象上的 **sendexit** 属性。

属性的值是包含以逗号分隔的一个或多个项的字符串。 每个项都通过以下一种方式标识发送出口：

- 实现以 Java 编写的发送出口的 `WMQSendExit` 接口的类的名称。
- 用于以 C 或 C++ 编写的发送出口的字符串，格式为 *libraryName* (*entryPointName*)。

类似地，以下属性指定供连接使用的一个接收出口或一系列接收出口：

- MQConnectionFactory 对象的 **RECEXIT** 属性。
- IBM MQ 资源适配器用于入站通信的激活规范上的 **receiveexit** 属性。
- IBM MQ 资源适配器用于输出通信的 ConnectionFactory 对象上的 **receiveexit** 属性。

以下属性指定供连接使用的安全出口：

- MQConnectionFactory 对象的 **SECEXIT** 属性。
- IBM MQ 资源适配器用于入站通信的激活规范上的 **securityexit** 属性。
- IBM MQ 资源适配器用于输出通信的 ConnectionFactory 对象上的 **securityexit** 属性。

对于 MQConnectionFactories，可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 来设置 **SENDEXIT**, **RECEXIT** 和 **SECEXIT** 属性。 或者，应用程序可通过调用 `setSendExit()`、`setReceiveExit()` 和 `setSecurityExit()` 方法来设置属性。

通道出口通过各自的类装入器来装入。 为找到通道出口，类装入器会以指定顺序搜索以下位置。

1. 由属性 **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** 或 IBM MQ 客户机配置文件的 Channels 节中的 **JavaExitsClassPath** 属性指定的类路径。
2. 由 Java 系统属性 **com.ibm.mq.exitClasspath** 指定的类路径。 请注意，现在已不推荐使用此属性。
3. IBM MQ 出口目录，如 第 230 页的表 42 中所示。 类装入器会首先在目录中搜索未打包在 Java 归档 (JAR) 文件中的类文件。 如果找不到通道出口，那么类装入器会在目录中搜索 JAR 文件。

表 *42: IBM MQ* 出口目录

平台	目录
Linux UNIX UNIX and Linux	`/var/mqm/exits`（32 位通道出口） `/var/mqm/exits64`（64 位通道出口）
Windows Windows	*install_data_dir*\exits 其中，*install_data_dir* 是您在安装期间为 IBM MQ 数据文件所选的目录。 缺省目录为 C:\ProgramData\IBM\MQ。

注: 如果在多个位置存在通道出口，那么 IBM MQ classes for JMS 会装入它所发现的第一个实例。

类装入器的父代是用于装入 IBM MQ classes for JMS 的类装入器。 因此，如果在以上所有位置中都找不到通道出口，那么父类装入器可能会装入通道出口。 但是，在诸如 JEE 应用程序服务器之类的环境中使用 IBM MQ classes for JMS 时，您可能无法影响父类装入器的选择，因此应通过在应用程序服务器上设置 Java 系统属性 **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** 来配置类装入器。

如果运行应用程序时已启用 Java security manager，那么正在运行应用程序的 Java 运行时环境所使用的策略配置文件必须具有装入通道出口类的权限。 有关如何执行此操作的信息，请参阅在 Java 安全管理器下运行 IBM MQ classes for JMS 应用程序。

仍支持与低于 IBM WebSphere MQ 7.0 的 IBM WebSphere MQ 版本一起提供的 `MQSendExit`，`MQReceiveExit` 和 `MQSecurityExit` 接口。 如果使用实现这些接口的通道出口，那么类路径中必须存在 `com.ibm.mq.jar`。

有关如何使用 C 语言编写通道出口的信息，请参阅第 875 页的『消息传递通道的通道出口程序』。 必须在第 230 页的表 42 中显示的目录中存储以 C 或 C++ 编写的通道出口程序。

如果应用程序使用客户机通道定义表 (CCDT) 连接到队列管理器，请参阅第 231 页的『将客户机通道定义表用于 IBM MQ classes for JMS』。

指定使用 *IBM MQ classes for JMS* 时将传递到通道出口的用户数据

调用通道出口时，最多可以向其传递包含 32 个字符的用户数据。

MQConnectionFactory 对象的 SENDEXITINIT 属性指定在被调用时将传递到每个发送出口的用户数据。 属性的值是包含以逗号分隔的一个或多个用户数据项的字符串。 每个用户数据项在字符串中的位置确定了一系列发送出口中将向其传递用户数据的发送出口。 例如，字符串中的第一个用户数据项将传递到发送出口序列中的第一个发送出口。

您可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 来设置 SENDEXITINIT 属性。 此外，应用程序还可通过调用 setSendExitInit() 方法来设置该属性。

相似地，ConnectionFactory 对象的 RECEXITINIT 属性指定向每个接收出口传递的用户数据，而 SECEXITINIT 属性则指定传递到安全出口的用户数据。 您可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 来设置这些属性。 此外，应用程序还可通过调用 setReceiveExitInit() 和 setSecurityExitInit() 方法来设置属性。

指定传递到通道出口的用户数据时，请注意以下规则：

- 如果字符串中的用户数据项数多于序列中的出口数，那么将忽略超出此数量的用户数据项。
- 如果字符串中的用户数据项数少于序列中的出口数，那么会将每个未指定的用户数据项设置为空字符串。 字符串中连续两个逗号或字符串开头处的逗号也指示未指定的用户数据项。

如果应用程序使用客户机通道定义表 (CCDT) 连接到队列管理器，那么在调用通道出口时会向其传递客户机连接通道定义中指定的任何用户数据。 有关客户机通道定义表的更多信息，请参阅第 231 页的『将客户机通道定义表用于 IBM MQ classes for JMS』。

将客户机通道定义表用于 *IBM MQ classes for JMS*

IBM MQ classes for JMS 应用程序可使用客户机通道定义表 (CCDT) 中存储的客户机连接通道定义。 您可以将 ConnectionFactory 对象配置为使用 CCDT。 使用时需遵循一些限制。

作为通过设置 ConnectionFactory 对象的某些属性创建客户机连接通道定义的替代方法，IBM MQ classes for JMS 应用程序可使用客户机通道定义表中存储的客户机连接通道定义。 这些定义通过 IBM MQ 脚本 (MQSC) 命令或 IBM MQ 可编程命令格式 (PCF) 命令创建。 应用程序创建 Connection 对象时，IBM MQ classes for JMS 会在客户机通道定义表中搜索适合的客户机连接通道定义，并使用此通道定义启动 MQI 通道。 有关客户机通道定义表以及如何构造此表的更多信息，请参阅客户机通道定义表。

要使用客户机通道定义表，必须将 ConnectionFactory 对象的 CCDTURL 属性设置为 URL 对象。 IBM MQ classes for JMS 不会从 IBM MQ MQI client 配置文件读取有关 CCDT 的信息，尽管使用了此配置文件中的一些其他值也是如此（请参阅第 77 页的『IBM MQ classes for JMS 配置文件』，了解适用值）。 此 URL 对象会封装一个统一资源定位符 (URL)，用于确定包含客户机通道定义表的文件的名称和位置，并指定可以如何访问此文件。 您可以使用 IBM MQ JMS 管理工具设置 CCDTURL 属性，或者应用程序可以通过创建 URL 对象并调用 ConnectionFactory 对象的 setCCDTURL() 方法来设置此属性。

例如，如果文件 ccdt1.tab 包含客户机通道定义表并存储在正在运行应用程序的同一系统上，那么应用程序可按照下列方式设置 CCDTURL 属性：

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

再比如，假设文件 ccdt2.tab 中包含客户机通道定义表，并且存储该文件的系统不同于运行应用程序的系统。 如果可以使用 FTP 协议访问此文件，那么应用程序可以通过以下方式设置 CCDTURL 属性：

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

除了设置 ConnectionFactory 对象的 CCDTURL 属性，还必须将相同对象的 QMANAGER 属性设置为以下一个值：

- 队列管理器的名称
- 星号 (*) 后跟队列管理器组的名称

这些值也可以用于 MQCONN 调用上的 **QMgrName** 参数，此调用由使用消息队列接口 (MQI) 的客户机应用程序发出。 有关这些值的含义的更多信息，请参阅 MQCONN。 您可以使用 IBM MQ JMS 管理工具或 IBM MQ Explorer 设置 QMANAGER 属性。 此外，应用程序还可通过调用 ConnectionFactory 对象的 setQueueManager() 方法来设置该属性。

如果应用程序随后从 ConnectionFactory 对象创建 Connection 对象，IBM MQ classes for JMS 会访问 CCDTURL 属性标识的客户机通道定义表，使用 QMANAGER 属性在表中搜索适合的客户机连接通道定义，然后使用此通道定义启动队列管理器的 MQI 通道。

请注意，应用程序调用 createConnection() 方法时，无法同时设置 ConnectionFactory 对象的 CCDTURL 和 CHANNEL 属性。 如果同时设置了这两个属性，那么方法会抛出异常。 如果 CCDTURL 或 CHANNEL 属性的值不为 Null、空字符串或包含的全部是空白字符的字符串，那么会将其视为已设置。

IBM MQ classes for JMS 在客户机通道定义表中查找适合的客户机连接通道定义时，仅使用从表中提取的信息来启动 MQI 通道。 将忽略 ConnectionFactory 对象的任何通道相关属性。

尤其是，如果正使用 TLS，请注意以下几点：

- 仅当从客户机通道定义表中提取的通道定义指定 IBM MQ classes for JMS 支持的 CipherSpec 的名称时，MQI 通道才会使用 TLS。
- 客户机通道定义表还包含有关保存证书撤销列表 (CRL) 的轻量级目录访问协议 (LDAP) 服务器的位置信息。 IBM MQ classes for JMS 仅使用此信息来访问保存 CRL 的 LDAP 服务器。
- 客户机通道定义表还可以包含 OCSP 响应程序的位置。 IBM MQ classes for JMS 不能使用客户机通道定义表文件中的 OCSP 信息。 但是，可以按 Java 和 JMS 客户机应用程序中的联机证书状态协议 (OCSP) 部分中所述配置 OCSP。

有关将 TLS 与客户机通道定义表一起使用的更多信息，请参阅将扩展事务客户机与 TLS 通道一起使用。

如果您使用通道出口，还要注意以下几点：

- MQI 通道仅使用由从客户机通道定义表中提取的通道定义指定的通道出口和关联用户数据。
- 从客户机通道定义表中提取的通道定义可指定使用 Java 编写的通道出口。 例如，这表示用于创建客户机连接通道定义的 DEFINE CHANNEL 命令上的 SCYEXIT 参数可以指定用于实现 WMQSecurityExit 接口的类的名称。 同样，SENDEXIT 参数可指定实现 WMQSendExit 接口的类的名称，RCVEXIT 参数可指定实现 WMQReceiveExit 接口的类的名称。 有关如何使用 Java 编写通道出口的更多信息，请参阅第 228 页的『使用 Java 为 IBM MQ classes for JMS 编写通道出口』。

  同时还支持使用以非 Java 语言编写的通道出口。 有关如何针对通过其他语言编写的通道出口指定 DEFINE CHANNEL 命令上的 SCYEXIT、SENDEXIT 和 RCVEXIT 参数的信息，请参阅 DEFINE CHANNEL。

自动重新连接 *JMS* 客户机

将 JMS 客户机配置为在网络、队列管理器或服务器发生故障后自动重新连接。

通常情况下，如果独立 IBM MQ classes for JMS 应用程序使用客户机传输连接到队列管理器，且队列管理器由于某种原因（例如，网络中断、队列管理器故障或队列管理器停止）变为不可用，那么下次应用程序尝试与队列管理器通信时，IBM MQ classes for JMS 将抛出 JMSException。 应用程序必须捕获 JMSException，并尝试重新连接到队列管理器。 您可以通过启用自动客户机重新连接功能来简化应用程序的设计。 当队列管理器变为不可用时，IBM MQ classes for JMS 会尝试代表应用程序自动重新连接到队列管理器。 这表示应用程序不需要包含用于重新连接的逻辑。

在 Java Platform Enterprise Edition 应用程序服务器中，不支持使用此自动客户机重新连接功能。 请参阅第 237 页的『在 Java EE 环境中使用自动客户机重新连接』，获取备用实施信息。

使用 *JMS* 客户机自动重新连接

如果独立的 IBM MQ classes for JMS 应用程序使用设置了 CONNECTIONNAMELIST 或 CCDTURL 属性的连接工厂，应用程序将有资格使用自动客户机重新连接。

客户机自动重新连接可用于重新连接至队列管理器（包括属于高可用性 (HA) 配置的队列管理器）。 在 IBM MQ 设备上，HA 配置包含多实例队列管理器、RDQM 队列管理器或 HA 队列管理器。

IBM MQ classes for JMS 所提供的自动客户机重新连接功能的行为取决于如下属性：

JMS 连接工厂属性 TRANSPORT（短名称 TRAN）

TRANSPORT 指定使用连接工厂的应用程序如何连接到队列管理器。 必须将此属性设置为值 CLIENT 才能使用自动客户机重新连接。 对于使用将 TRANSPORT 属性设置为 BIND、DIRECT 或 DIRECTHTTP 的连接工厂来连接到队列管理器的应用程序，无法使用自动客户机重新连接。

JMS 连接工厂属性 QMANAGER（短名称 QMGR）

QMANAGER 属性指定连接工厂连接到的队列管理器的名称。

JMS 连接工厂属性 CONNECTIONNAMELIST（短名称 CRHOSTS）

CONNECTIONNAMELIST 属性是逗号分隔列表，其中每个条目都包含有关使用 CLIENT 传输时用于连接到 QMANAGER 属性所指定队列管理器的主机名和端口的信息。 列表采用以下格式：host name(port), host name(port)。

JMS 连接工厂属性 CCDTURL（短名称 CCDT）

CCDTURL 属性指向 IBM MQ classes for JMS 在使用 CCDT 连接到队列管理器时所使用的客户机通道定义表。

JMS 连接工厂属性 CLIENTRECONNECTOPTIONS（短名称 CROPT）

CLIENTRECONNECTOPTIONS 控制如果队列管理器变得可用，IBM MQ classes for JMS 是否代表应用程序尝试自动连接到队列管理器。

客户机配置文件中 Channels 节中的 DefRecon 属性

DefRecon 属性提供管理选项，使所有应用程序能够自动重新连接或对编写为自动重新连接的应用程序禁用自动重新连接。

只有当应用程序成功连接到队列管理器时，自动客户机重新连接才可用。

当应用程序使用 CLIENT 传输连接到队列管理器时，如果应用程序连接到的队列管理器变得不可用，IBM MQ classes for JMS 将使用连接工厂属性 CLIENTRECONNECTOPTIONS 的值来确定是否使用自动客户机重新连接。 表 1 显示了 CLIENTRECONNECTOPTIONS 属性的可能值以及针对其中每个值，IBM MQ classes for JMS 的行为：

表 *43:* 可能的 *CLIENTRECCECTOPTIONS* 属性值	
CLIENTRECONNECTOPTIONS	**IBM MQ classes for JMS 的行为**
ANY	如果 CONNECTIONNAMELIST 已设置，使用 CONNECTIONNAMELIST 属性的值来打开与主机名和端口组合的连接，并连接到任何队列管理器。 要使用该自动客户机重新连接选项，必须将 QMANAGER 属性设置为缺省值或者“*”。 如果 CCDTURL 已设置，打开使用 CCDTURL 属性指定的客户机通道定义表，在表中选取条目，然后使用该条目来启动到队列管理器的客户机连接通道。 要使用该自动客户机重新连接选项，必须将 QMANAGER 属性设置为： • 星号 (*) • 星号 (*) 后跟队列管理器组的名称 • 空字符串或者包含所有空白字符的字符串
ASDEF	使用 DefRecon 的值来确定自动客户机重新连接是否可用。
DISABLED	请勿执行任何自动客户机重新连接，并且向应用程序返回 JMSException。
QMGR	指定客户机必须重新连接到相同的队列管理器。 如果必须重新连接到相同队列管理器的另一个实例，那么该选项必须用于高可用性解决方案。 如果 CONNECTIONNAMELIST 已设置，使用 CONNECTIONNAMELIST 属性的值来打开与主机名和端口组合的连接，并连接到 QMANAGER 属性指定的队列管理器。 如果 CCDTURL 已设置，打开使用 CCDTURL 属性指定的客户机通道定义表，在表中找到与通过 QMANAGER 属性指定的队列管理器名称匹配的条目，然后使用这些条目来启动到该队列管理器的客户机连接通道。

如果 CONNECTIONNAMELIST 已设置，在执行自动客户机重新连接时，IBM MQ classes for JMS 使用连接工厂属性 CONNNECTIONNAMELIST 中的信息来确定要重新连接到的系统。

IBM MQ classes for JMS 最初会尝试使用 CONNECTIONNAMELIST 中的第一个条目指定的主机名和端口来重新连接。 如果连接已建立，那么 IBM MQ classes for JMS 会尝试连接到具有 QMANAGER 属性中所指定名称的队列管理器。 如果可以建立到队列管理器的连接，那么 IBM MQ classes for JMS 将在自动客户机重新连接之前重新打开应用程序曾打开的所有 IBM MQ 对象，并继续像以前一样运行。

如果无法使用 CONNECTIONNAMELIST 中的第一个条目建立连接，IBM MQ classes for JMS 会尝试 CONNECTIONNAMELIST 中的第二个条目，以此类推。

如果 IBM MQ classes for JMS 已尝试 CONNECTIONNAMELIST 中的所有条目，那么会在再次尝试重新连接之前等待一段时间。 要执行新的重新连接尝试，IBM MQ classes for JMS 将从 CONNECTIONNAMELIST 中的第一个条目开始。 然后，将轮流尝试 CONNECTIONNAMELIST 中的每个条目，直到重新连接或到达 CONNECTIONNAMELIST 的末尾，届时，IBM MQ classes for JMS 会在再次尝试之前等待一段时间。

如果 CCDTURL 已设置，在执行自动客户机重新连接时，IBM MQ classes for JMS 使用在 CCDTURL 属性中指定的客户机通道定义表来确定要重新连接到的系统。

IBM MQ classes for JMS 最初解析客户机通道定义表，并且找到与 QMANAGER 属性的值匹配的合适条目。 找到条目后，IBM MQ classes for JMS 将尝试使用该条目重新连接到所需的队列管理器。 如果可以建立到队

列管理器的连接，那么 IBM MQ classes for JMS 将在自动客户机重新连接之前重新打开应用程序曾打开的所有 IBM MQ 对象，并继续像以前一样运行。

如果无法建立到所需队列管理器的连接，那么 IBM MQ classes for JMS 将在客户机通道定义表中查找另一个合适条目并尝试使用该条目，以此类推。

如果 IBM MQ classes for JMS 已尝试客户机通道定义表中的所有合适条目，那么会在再次尝试重新连接之前等待一段时间。 要执行新的重新连接尝试，IBM MQ classes for JMS 将再次解析客户机通道定义表并尝试第一个合适条目。 然后，将轮流尝试客户机通道定义表中的每个合适条目，直到重新连接或已经尝试了客户机通道定义表中的最后一个合适条目，届时，IBM MQ classes for JMS 会在再次尝试之前等待一段时间。

无论使用 CONNECTIONNAMELIST 或 CCDTURL，该自动客户机重新连接过程都将继续，直至 IBM MQ classes for JMS 成功重新连接到 QMANAGER 属性指定的队列管理器。

缺省情况下，将按以下时间间隔尝试重新连接：

- 在初始延迟 1 秒后进行首次尝试，外加高达 250 毫秒的随机元素。
- 第二次尝试为 2 秒，外加首次尝试失败之后高达 500 毫秒的随机时间间隔。
- 第三次尝试为 4 秒，外加第二次尝试失败之后高达 1 秒的随机时间间隔。
- 第四次尝试为 8 秒，外加第三次尝试失败之后高达 2 秒的随机时间间隔。
- 第五次尝试为 16 秒，外加第四次尝试失败之后高达 4 秒的随机时间间隔。
- 第六次尝试以及所有后续尝试为 25 秒，外加上次尝试失败之后高达 6250 毫秒的随机时间间隔。

重新连接尝试按照部分固定、部分随机的时间间隔进行延迟。 这可防止连接到不再可用的队列管理器的所有 IBM MQ classes for JMS 应用程序同时重新连接。

如果需要增加缺省值以更准确地反映恢复队列管理器所需的时间量或备用队列管理器变为活动所需的时间量，请修改客户机配置文件的 Channel 节中的 ReconDelay 属性，有关更多信息，请参阅客户机配置文件的 CHANNELS 节。

IBM MQ classes for JMS 应用程序在自动重新连接后是否继续正确工作取决于其设计。 请阅读相关主题以了解如何设计可使用自动重新连接功能的应用程序。

指示队列管理器不再可用的原因码

哪些原因码指示队列管理器不再可用或无法访问，以及何时尝试 IBM MQ classes for JMS 自动重新连接。

第 232 页的『自动重新连接 JMS 客户机』 概述了 JMSException 以及您的应用程序可通过何种方式自动重新启动，第 233 页的『使用 JMS 客户机自动重新连接』中的信息详述了客户机自动重新连接的要求。

以下信息列出了您的应用程序应检查的 IBM MQ 原因码：

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

抛回至企业应用程序的大多数 JMSException 中都包含存储原因码的链接 MQException。 要针对上一个列表中的原因码实现重试逻辑，您的企业应用程序应使用类似于以下示例的代码检查此链接异常：

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

相关参考
IBM MQ classes for JMS

在 *Java SE* 和 *Java EE* 环境中使用自动客户机重新连接功能
您可以使用 IBM MQ 自动客户机重新连接来促进 Java SE 和 Java EE 环境中的各种高可用性 (HA) 和灾难恢复 (DR) 解决方案。

在不同平台上提供了各种 HA 和 DR 解决方案：

- Multi 多实例队列管理器是在不同服务器上配置的同一队列管理器的多个实例（请参阅多实例队列管理器）。 其中一个队列管理器实例定义为活动实例，另一个实例定义为备用实例。 如果活动实例发生故障，那么备用服务器上的多实例队列管理器将自动重新启动。

  活动队列管理器和备用队列管理器都具有相同的队列管理器标识 (QMID)。 可以将连接到多实例队列管理器的 IBM MQ 客户机应用程序配置为使用自动客户机重新连接自动重新连接到队列管理器的备用实例。

- Linux RDQM（复制的数据队列管理器）是 Linux 平台上提供的高可用性解决方案（请参阅 RDQM 高可用性）。 RDQM 配置由在高可用性 (HA) 组中配置的三台服务器组成，每台服务器都具有一个队列管理器实例。 其中一个实例是运行中的队列管理器，可将其数据同步复制到另外两个实例。 如果运行此队列管理器的服务器发生故障，那么另一个队列管理器实例就会启动，并且具有可操作的最新数据。 这三个队列管理器实例共享一个浮动 IP 地址，因此只需为客户机配置单个 IP 地址。 对于连接到 RDQM 队列管理器的客户机应用程序，可以将其配置为使用自动客户机重新连接功能来自动重新连接到队列管理器的备用实例。

- MQ Appliance HA 解决方案还可以由一对 IBM MQ 设备提供 (请参阅 IBM MQ Appliance 文档中的 高可用性和 灾难恢复)。 HA 队列管理器在其中一台设备上运行，同时将数据同步复制到另一台设备上的队列管理器的备用实例。 如果主设备发生故障，那么此队列管理器会在另一台设备上自动启动并运行。 这两个队列管理器实例可配置为共享一个浮动 IP 地址，因此只需为客户机配置单个 IP 地址。 对于连接到 IBM MQ Appliance 上的 HA 队列管理器的客户机应用程序，可以将其配置为使用自动客户机重新连接功能来自动重新连接到队列管理器的备用实例。

相关概念
多实例队列管理器
自动客户机重新连接
RDQM 高可用性

在 *Java SE* 环境中使用自动客户机重新连接
使用在 Java SE 环境中运行的 IBM MQ classes for JMS 的应用程序可以通过连接工厂属性 **CLIENTRECONNECTOPTIONS** 使用自动客户机重新连接功能。

连接工厂属性 **CLIENTRECONNECTOPTIONS** 将使用两个额外的连接工厂属性 **CONNECTIONNAMELIST** 和 **CCDTURL** 来确定如何连接到运行队列管理器的服务器。

CONNECTIONNAMELIST 属性

CONNECTIONNAMELIST 属性是包含主机名和端口信息的逗号分隔列表，这些信息用于以客户机方式连接到队列管理器。 此属性与 **QMANAGER** 和 **CHANNEL** 值结合使用。 当应用程序使用 **CONNECTIONNAMELIST** 属性来创建客户机连接时，IBM MQ classes for JMS 会尝试按照列出顺序来连接到各个主机。 如果第一个队列管理器主机不可用，IBM MQ classes for JMS 会尝试连接到列表中的下一个主机。 如果到达连接名称列表的末尾仍无法创建连接，那么 IBM MQ classes for JMS 会抛出 MQRC_QMGR_NOT_AVAILABLE IBM MQ 原因码。

如果应用程序连接到的队列管理器发生故障，那么使用 **CONNECTIONNAMELIST** 连接到该队列管理器的任何应用程序都会收到一个异常来用于表明队列管理器不可用。 应用程序必须捕获异常并清除之前所使用的任何资源。 要创建连接，应用程序必须使用连接工厂。 连接工厂将再次尝试按照列出顺序来连接到各个主机，之前发生故障的队列管理器现在不可用。 连接工厂会尝试连接到列表中的其他主机。

CCDTURL 属性

CCDTURL 属性包含指向客户机通道定义表 (CCDT) 的统一资源定位符 (URL)，此属性与 **QMANAGER** 属性结合使用。 CCDT 包含用于连接到 IBM MQ 系统上定义的队列管理器的客户机通道列表。 有关 IBM MQ classes for JMS 如何使用 CCDT 的信息，请参阅第 231 页的『将客户机通道定义表用于 IBM MQ classes for JMS』。

使用 CLIENTRECONNECTOPTIONS 属性在 IBM MQ classes for JMS 内启用自动客户机重新连接

CLIENTRECONNECTOPTIONS 属性用于在 IBM MQ classes for JMS 内启用自动客户机重新连接。 此属性的可能值如下所示：

ASDEF
自动客户机重新连接行为由 IBM MQ 客户机配置文件 (mqclient.ini) 的通道节中指定的缺省值来定义。

DISABLED
禁用自动客户机重新连接。

QMGR
IBM MQ classes for JMS 尝试使用以下任一选项来连接到与先前所连接队列管理器具有相同标识的队列管理器：

- **CONNECTIONNAMELIST** 属性以及 **CHANNEL** 属性中定义的通道。
- **CCDTURL** 属性中定义的 CCDT。

ANY
IBM MQ classes for JMS 尝试使用 **CONNECTIONNAMELIST** 属性或 **CCDTURL** 属性来重新连接到具有相同名称的队列管理器。

相关信息
客户机配置文件的 CHANNELS 节

在 *Java EE* 环境中使用自动客户机重新连接
可以部署到 Java EE (Java Platform, Enterprise Edition) 环境中的 IBM MQ 资源适配器和 WebSphere Application Server IBM MQ 消息传递提供程序使用 IBM MQ classes for JMS 与 IBM MQ 队列管理器进行通信。 IBM MQ 资源适配器和 WebSphere Application Server IBM MQ 消息传递提供程序都提供了自动客户机重新连接支持。

可用于在 Java EE 环境中提供自动客户机重新连接的选项包括：

- 激活规范
- WebSphere Application Server 侦听器端口
- Enterprise Java Bean 和基于 Web 的应用程序
- 在客户机容器内运行的应用程序

注: 不支持使用 IBM MQ classes for JMS 提供的功能，通过激活规范来进行自动客户机重新连接。 IBM MQ 资源适配器提供了自己的机制，用于在激活规范所连接到的队列管理器不可用时重新连接激活规范。

此机制由以下项控制：

- IBM MQ 资源适配器属性 **reconnectionRetryCount**。
- IBM MQ 资源适配器属性 **reconnectionRetryInterval**。
- 激活规范属性 **connectionNameList**。

有关这些属性的更多信息，请参阅第 376 页的『ResourceAdapter 对象属性的配置』。

不支持在消息驱动的 bean 应用程序的 onMessage() 方法中使用自动客户机重新连接，也不支持在 Java Platform, Enterprise Edition 环境中运行的任何其他应用程序中使用自动客户机重新连接。 如果应用程序所连接到的队列管理器不可用，那么该应用程序必须实施自己的重新连接逻辑。

在 *Java EE* 环境中支持自动客户机重新连接

在 Java EE 环境（例如 WebSphere Application Server）中，IBM MQ 资源适配器和 WebSphere Application Server IBM MQ 消息传递提供者提供了自动客户机重新连接支持。 但是，在某些情况下，此支持存在一些限制。

可以部署到 Java EE 环境和 WebSphere Application Server IBM MQ 消息传递提供程序的 IBM MQ 资源适配器使用 IBM MQ classes for JMS 与 IBM MQ 队列管理器进行通信。

下表汇总了 IBM MQ 资源适配器和 WebSphere Application Server IBM MQ 消息传递提供者所提供的自动客户机重新连接支持。

表 *44: Java EE* 环境中自动客户机重新连接选项支持的汇总

自动重新连接选项	CONNECTIONNAMELIST 属性	CCDTURL 属性	CLIENTRECONNECTOPTIONS 属性	自动客户机重新连接的替代方法
激活规范	支持但有限制	支持但有限制	不支持	Java EE 环境和激活规范可提供自己的重新连接机制
WebSphere Application Server 侦听器端口	支持但有限制	支持但有限制	不支持	WebSphere Application Server 提供自己的重新连接机制
Enterprise Java Bean 和基于 Web 的应用程序	支持但有限制	支持但有限制	不支持	应用程序必须实现自己的重新连接逻辑
在客户机容器内运行的应用程序	受支持	受支持	受支持	不适用

Java EE 环境（例如 IBM MQ classes for JMS）中安装的消息驱动的 bean 应用程序可以使用激活规范来处理 IBM MQ 系统上的消息。 激活规范用于检测 IBM MQ 系统上接收的消息，然后将消息传递到消息驱动的 bean 进行处理。 消息驱动的 Bean 还可以从其 **onMessage()** 方法内部与 IBM MQ 系统建立更多连接。 有关这些连接如何使用自动客户机重新连接的更多信息，请参阅 Enterprise Java Bean 和基于 Web 的应用程序。

激活规范

对于激活规范，在满足一定限制的情况下支持 **CONNECTIONNAMELIST** 和 **CCDTURL** 属性，但不支持 **CLIENTRECONNECTOPTIONS** 属性。

Java EE 环境（例如 WebSphere Application Server）中安装的消息驱动的 bean (MDB) 应用程序可以使用激活规范来处理 IBM MQ 系统上的消息。

激活规范用于检测 IBM MQ 系统上接收的消息，然后将消息传递到 MDB 进行处理。 本部分介绍了激活规范如何监视 IBM MQ 系统。

MDB 还可以从其 onMessage() 方法内部与 IBM MQ 系统建立其他连接。

可以在第 241 页的『Enterprise Java Bean 和基于 Web 的应用程序』中找到有关这些连接如何使用自动客户机重新连接的详细信息。

CONNECTIONNAMELIST 属性

在启动时，激活规范会尝试使用以下途径来连接到队列管理器：

- **QMANAGER** 属性中指定的队列管理器
- **CHANNEL** 属性中提到的通道
- 来自 **CONNECTIONNAMELIST** 中第一个条目的主机名和端口信息

如果激活规范无法使用列表中的第一个条目连接到队列管理器，那么激活规范会继续使用第二个条目并依次尝试后续条目，直到建立了与队列管理器的连接，或者到达该列表末尾为止。

如果激活规范无法使用 **CONNECTIONNAMELIST** 中的任何条目连接到指定的队列管理器，那么激活规范将停止并且必须重新启动。

运行激活规范后，激活规范会从 IBM MQ 系统中获取消息，并将消息传递到 MDB 进行处理。

如果在处理消息期间队列管理器发生故障，那么 Java EE 环境会检测到此故障并尝试重新连接到激活规范。

激活规范在执行重新连接尝试时，会像以前一样使用 **CONNECTIONNAMELIST** 属性中的信息。

如果激活规范尝试了 **CONNECTIONNAMELIST** 中的所有条目后仍无法连接到队列管理器，那么激活规范将等待 IBM MQ 资源适配器属性 **reconnectionRetryInterval** 所指定的时间段，然后再重试。

IBM MQ 资源适配器属性 **reconnectionRetryCount** 定义连续重新连接尝试的次数，达到该次数后将停止激活规范并需要手动重新启动。

激活规范重新连接到 IBM MQ 系统之后，Java EE 环境将执行任何必需的事务清除操作，然后继续将消息传递到 MDB 进行处理。

为使事务清除操作正常工作，Java EE 环境必须能够访问发生故障的队列管理器的日志。

如果激活规范与参与 XA 事务的事务 MDB 结合使用，并正在连接到多实例队列管理器，那么 **CONNECTIONNAMELIST** 必须同时包含活动和备用队列管理器实例的条目。

这表示 Java EE 环境在需要执行事务恢复时可以访问队列管理器日志，这与发生故障后该环境重新连接到哪个队列管理器无关。

如果事务 MDB 与独立队列管理器配合使用，那么 **CONNECTIONNAMELIST** 属性必须包含单个条目，以确保激活规范在发生故障后始终重新连接到同一系统上运行的同一队列管理器。

CCDTURL 属性

启动时，激活规范尝试使用客户机通道定义表 (CCDT) 中的第一个条目连接到 **QMANAGER** 属性中指定的队列管理器。

如果激活规范无法使用该表中的第一个条目连接到队列管理器，那么激活规范会继续使用第二个条目并依次尝试后续条目，直到建立了与队列管理器的连接，或者到达该表末尾为止。

如果激活规范无法使用 CCDT 中的任何条目连接到指定的队列管理器，那么激活规范将停止并且必须重新启动。

运行激活规范后，激活规范会从 IBM MQ 系统中获取消息，并将消息传递到 MDB 进行处理。

如果在处理消息期间队列管理器发生故障，那么 Java EE 环境会检测到此故障并尝试重新连接到激活规范。

激活规范在执行重新连接尝试时，会像以前一样使用 CCDT 属性中的信息。

如果激活规范尝试了 CCDT 中的所有条目后仍无法连接到队列管理器，那么激活规范将等待 IBM MQ 资源适配器属性 **reconnectionRetryInterval** 所指定的时间段，然后再重试。

IBM MQ 资源适配器属性 **reconnectionRetryCount** 定义连续重新连接尝试的次数，达到该次数后将停止激活规范并需要手动重新启动。

激活规范重新连接到 IBM MQ 系统之后，Java EE 环境将执行任何必需的事务清除操作，然后继续将消息传递到 MDB 进行处理。

为使事务清除操作正常工作，Java EE 环境必须能够访问发生故障的队列管理器的日志。

如果激活规范与参与 XA 事务的事务 MDB 结合使用，并正在连接到多实例队列管理器，那么 CCDT 必须同时包含活动和备用队列管理器实例的条目。

这表示 Java EE 环境在需要执行事务恢复时可以访问队列管理器日志，这与发生故障后该环境重新连接到哪个队列管理器无关。

如果事务 MDB 与独立队列管理器结合使用，那么 CCDT 必须包含单个条目，以确保在发生故障之后，激活规范始终重新连接到同一系统上运行的同一队列管理器。

确保为与激活规范结合使用的 CCDT 上的 **AFFINITY** 属性设置了缺省值 *PREFERRED*，以便建立与同一个活动队列管理器的连接。

CLIENTRECONNECTOPTIONS 属性

激活规范提供自己的重新连接功能。 通过此功能，这些规范在其连接到的队列管理器发生故障时自动重新连接到 IBM MQ 系统。

因此，不支持 IBM MQ classes for JMS 提供的自动客户机重新连接。

对于 Java EE 中使用的所有激活规范，必须将 **CLIENTRECONNECTOPTIONS** 属性设置为 *DISABLED* 。

WebSphere Application Server 侦听器端口
WebSphere Application Server 中安装的消息驱动的 bean (MDB) 应用程序也可以使用侦听器端口来处理 IBM MQ 系统上的消息。

侦听器端口用于检测 IBM MQ 系统上接收的消息，然后将消息传递到 MDB 进行处理。 本主题介绍了侦听器端口如何监视 IBM MQ 系统。

MDB 还可以从其 onMessage() 方法内部与 IBM MQ 系统建立其他连接。

请参阅第 241 页的『Enterprise Java Bean 和基于 Web 的应用程序』，以了解有关这些连接如何使用自动客户机重新连接的更多信息。

对于 WebSphere Application Server 侦听器端口：

- 在满足一定限制的情况下支持 **CONNECTIONNAMELIST** 和 **CCDTURL**
- 不支持 **CLIENTRECONNECTOPTIONS**

CONNECTIONNAMELIST 属性

侦听器端口在连接到 IBM MQ 时使用 JMS 连接池，因此受使用连接池的影响。 请参阅第 238 页的『激活规范』以获取进一步信息。

如果没有可用连接，并且从该连接工厂创建的连接数尚未达到最大数量，那么将使用 **CONNECTIONNAMELIST** 属性来尝试创建与 IBM MQ 的新连接。

如果 **CONNECTIONNAMELIST** 中的所有 IBM MQ 系统都不可访问，那么侦听器端口将停止。

然后，侦听器端口将等待消息侦听器服务定制属性 **RECOVERY.RETRY.INTERVAL** 指定的时间段，然后再次尝试重新连接。

该重新连接尝试将检查连接池中是否有任何空闲的连接，以防两次连接尝试期间返回了连接。 如果没有可用连接，那么侦听器端口会像以前一样使用 **CONNECTIONNAMELIST**。

侦听器端口重新连接到 IBM MQ 系统之后，Java EE 环境将执行任何必需的事务清除操作，然后继续将消息传递到 MDB 进行处理。

为使事务清除操作正常工作，Java EE 环境必须能够访问发生故障的队列管理器的日志。

如果侦听器端口与参与 XA 事务的事务 MDB 结合使用，并正在连接到**多实例队列管理器**，那么 **CONNECTIONNAMELIST** 必须同时包含活动和备用队列管理器实例的条目。

这表示 Java EE 环境在需要执行事务恢复时可以访问队列管理器日志，这与发生故障后该环境重新连接到哪个队列管理器无关。

如果事务 MDB 与独立队列管理器配合使用，那么 **CONNECTIONNAMELIST** 属性必须包含单个条目，以确保激活规范在发生故障后始终重新连接到同一系统上运行的同一队列管理器。

CCDTURL 属性

启动时，侦听器端口尝试使用 CCDT 中的第一个条目连接到 **QMANAGER** 属性中指定的队列管理器。

如果侦听器端口无法使用该表中的第一个条目连接到队列管理器，那么侦听器端口会继续使用第二个条目并依次尝试后续条目，直到建立了与队列管理器的连接，或者到达该表末尾为止。

如果侦听器端口无法使用 CCDT 中的任何条目连接到指定的队列管理器，那么侦听器端口将停止。

然后，侦听器端口将等待消息侦听器服务定制属性 **RECOVERY.RETRY.INTERVAL** 指定的时间段，然后再次尝试重新连接。

此重新连接尝试会像以前一样遍历 CCDT 中的所有条目。

运行侦听器端口之后，它会从 IBM MQ 系统中获取消息，并将消息传递到 MDB 进行处理。

如果在处理消息期间队列管理器发生故障，那么 Java EE 环境会检测到该故障并尝试重新连接到侦听器端口。 侦听器端口在执行重新连接尝试时使用 CCDT 中的信息。

如果侦听器端口尝试了 CCDT 中的所有条目后仍无法连接到队列管理器，那么该端口将等待 **RECOVERY.RETRY.INTERVAL** 属性指定的时间段，然后再重试。

消息侦听器服务属性 **MAX.RECOVERY.RETRIES** 定义连续重新连接尝试次数，达到该次数后将停止侦听器端口并需要手动重新启动。

侦听器端口重新连接到 IBM MQ 系统之后，Java EE 环境将执行任何必需的事务清除操作，然后继续将消息传递到 MDB 进行处理。

为使事务清除操作正常工作，Java EE 环境必须能够访问发生故障的队列管理器的日志。

如果侦听器端口与参与 XA 事务的事务 MDB 结合使用，并正在连接到多实例队列管理器，那么 CCDT 必须同时包含活动和备用队列管理器实例的条目。

这表示 Java EE 环境在需要执行事务恢复时可以访问队列管理器日志，这与发生故障后该环境重新连接到哪个队列管理器无关。

如果事务 MDB 与独立队列管理器结合使用，那么 CCDT 必须包含单个条目，以确保在发生故障之后，侦听器端口始终重新连接到同一系统上运行的同一队列管理器。

确保为与侦听器端口结合使用的 CCDT 上的 **AFFINITY** 属性设置了缺省值 *PREFERRED*，以便建立与同一个活动队列管理器的连接。

CLIENTRECONNECTOPTIONS 属性

侦听器端口提供自己的重新连接功能。 通过此功能，这些侦听器端口在其连接到的队列管理器发生故障时自动重新连接到 IBM MQ 系统。

因此，不支持 IBM MQ classes for JMS 提供的自动客户机重新连接。

对于 Java EE 中使用的所有侦听器端口，必须将 **CLIENTRECONNECTOPTIONS** 属性设置为 *DISABLED* 。

Enterprise Java Bean 和基于 *Web* 的应用程序
Enterprise Java Bean (EJB) 应用程序以及在 web 容器（例如 Servlet）内运行的应用程序都使用 JMS 连接工厂来创建与 IBM MQ 队列管理器的连接。

以下限制应用于 EJB 和基于 Web 的应用程序:

- 在满足一定限制的情况下支持 **CONNECTIONNAMELIST** 和 **CCDTURL**
- 不支持 **CLIENTRECONNECTOPTIONS**

CONNECTIONNAMELIST 属性

如果 Java EE 环境提供了用于 JMS 连接的连接池，请参阅第 243 页的『在连接池中使用 CONNECTIONNAMELIST 或 CCDT』，以了解这对 **CONNECTIONNAMELIST** 属性行为的影响。

如果 Java EE 环境未提供 JMS 连接池， 那么应用程序使用 **CONNECTIONNAMELIST** 属性的方式与 Java SE 应用程序相同。

如果应用程序与参与 XA 事务的事务 MDB 结合使用，并正在连接到多实例队列管理器，那么 **CONNECTIONNAMELIST** 必须同时包含活动和备用队列管理器实例的条目。

这表示 Java EE 环境在需要执行事务恢复时可以访问队列管理器日志，这与发生故障后该环境重新连接到哪个队列管理器无关。

如果应用程序与独立队列管理器配合使用，那么 **CONNECTIONNAMELIST** 属性必须包含单个条目，以确保应用程序在发生故障后始终重新连接到在同一系统上运行的同一队列管理器。

CCDTURL 属性

如果 Java EE 环境提供了用于 JMS 连接的连接池，请参阅第 243 页的『在连接池中使用 CONNECTIONNAMELIST 或 CCDT』，以了解这对 **CCDTURL** 属性行为的影响。

如果 Java EE 环境未提供 JMS 连接池， 那么应用程序使用 **CCDTURL** 属性的方式与 Java SE 应用程序相同。

如果应用程序与参与 XA 事务的事务 MDB 结合使用，并正在连接到“多实例队列管理器”，那么 CCDT 必须同时包含活动和备用队列管理器实例的条目。

这表示 Java EE 环境在需要执行事务恢复时可以访问队列管理器日志，这与发生故障后该环境重新连接到哪个队列管理器无关。

如果应用程序与“独立队列管理器”结合使用，那么 CCDT 必须包含单个条目，以确保在发生故障之后，激活规范始终重新连接到同一系统上运行的同一队列管理器。

CLIENTRECONNECTOPTIONS 属性

对于在 Web 容器中运行的 EJB 或应用程序所使用的所有 JMS 连接工厂，必须将 **CLIENTRECONNECTOPTIONS** 属性设置为 *DISABLED* 。

对于需要自动重新连接到新队列管理器的应用程序，当它们使用的队列管理器发生故障时，它们需要实现自己的重新连接逻辑。 请参阅 第 243 页的『在 Java EE 应用程序中实现重新连接逻辑』 以获取更多信息。

场景：WebSphere Application Server 与 IBM MQ
场景：WebSphere Application Server Liberty Profile 与 IBM MQ

在客户机容器内运行的应用程序
某些 Java EE 环境（例如 WebSphere Application Server）提供了可用于运行 Java SE 应用程序的客户机容器。

在这些环境内运行的应用程序会使用 JMS 连接工厂来连接到 IBM MQ 队列管理器。

对于在客户机容器内运行的应用程序：

- 完全支持 **CONNECTIONNAMELIST** 和 **CCDTURL**
- 完全支持 **CLIENTRECONNECTOPTIONS**

CONNECTIONNAMELIST 属性

如果 Java EE 环境提供了用于 JMS 连接的连接池，请参阅第 243 页的『在连接池中使用 CONNECTIONNAMELIST 或 CCDT』，以了解这对 **CONNECTIONNAMELIST** 属性行为的影响。

如果 Java EE 环境未提供 JMS 连接池， 那么应用程序使用 **CONNECTIONNAMELIST** 属性的方式与 Java SE 应用程序相同。

CCDTURL 属性

如果 Java EE 环境提供了用于 JMS 连接的连接池，请参阅第 243 页的『在连接池中使用 CONNECTIONNAMELIST 或 CCDT』，以了解这对 **CCDTURL** 属性行为的影响。

如果 Java EE 环境未提供 JMS 连接池， 那么应用程序使用 **CCDTURL** 属性的方式与 Java SE 应用程序相同。

在连接池中使用 *CONNECTIONNAMELIST* 或 *CCDT*

某些 Java EE 环境（例如 WebSphere Application Server）提供了 JMS 连接池。 这是可用于运行 Java SE 应用程序的容器。

对于使用 Java EE 环境中已定义的连接工厂创建连接的应用程序，它们要么从该连接工厂的连接池中获取现有的空闲连接，要么在连接池中没有合适连接的情况下建立新连接。

这可能暗示已配置的连接工厂是定义 **CONNECTIONNAMELIST** 属性还是 **CCDTURL** 属性。

首次使用连接工厂来创建连接时， Java EE 环境将使用 **CONNECTIONNAMELIST**。 或 **CCDTURL** 以创建与 IBM MQ 系统的新连接。 当不再需要该连接时，会将其返回到连接池中，该连接在连接池中可供复用。

如果其他实体使用连接工厂来创建了连接，那么 Java EE 环境会返回连接池中的该连接，而不是使用 **CONNECTIONNAMELIST** 或 **CCDTURL** 属性来创建新连接。

如果队列管理器实例在发生故障时正在使用某个连接，那么将废弃该连接。 但是，可能不会废弃连接池中的内容，这意味着连接池中可能仍包含与不再运行的队列管理器的连接。

在这种情况下，下次请求使用连接工厂来创建连接时，将返回与发生故障的队列管理器的连接。 任何使用此连接的尝试均会失败，因为该队列管理器不再运行，因而导致废弃了此连接。

仅当连接池为空时，Java EE 环境才会使用 **CONNECTIONNAMELIST** 或 **CCDTURL** 属性来创建与 IBM MQ 的新连接。

根据使用 **CONNECTIONNAMELIST** 和 CCDT 创建 JMS 连接的方式不同，连接池中还可能包含与不同 IBM MQ 系统的连接。

例如，假设在将 **CONNECTIONNAMELIST** 属性设置为以下值的情况下配置了连接工厂：

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

假设应用程序首次尝试使用该连接工厂创建与独立队列管理器的连接，那么无法访问系统 `hostname1(port1)` 上运行的队列管理器。 这意味着应用程序最终会连接到 `hostname2(port2)` 上运行的队列管理器。

此时有另一个应用程序在使用同一个连接工厂来创建 JMS 连接。 `hostname1(port1)` 上的队列管理器现在可用，因此将创建与此 IBM MQ 系统的新 JMS 连接并将其返回到应用程序。

这两个应用程序运行完毕后，它们会关闭其 JMS 连接，从而导致这些连接返回到连接池中。

最终，该连接工厂的连接池现在包含两个 JMS 连接：

- 一个是与 `hostname1(port1)` 上运行的队列管理器的连接
- 一个是与 `hostname2(port2)` 上运行的队列管理器的连接

这可能导致与事务恢复有关的问题。 如果 Java EE 系统需要回滚事务，那么必须能够连接到有权访问事务日志的队列管理器。

在 *Java EE* 应用程序中实现重新连接逻辑

在队列管理器发生故障时希望自动重新连接的 Enterprise Java Bean 和基于 Web 的应用程序需要实现自己的重新连接逻辑。

以下选项给出了有关如何实现此目的的更多信息。

允许应用程序失败

此方法不需要更改应用程序，但要求管理员将连接工厂定义重新配置为包含 **CONNECTIONNAMELIST** 属性。 但是，此方法要求调用者能够适当地处理故障。 请注意，对于与连接故障无关的故障（例如 MQRC_Q_FULL）也有此要求。

此过程的示例代码如下：

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
          try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
   }
 }
}
```

上述代码假设此 servlet 使用的连接工厂已定义了 **CONNECTIONNAMELIST** 属性。

当 servlet 首次处理时，将使用 **CONNECTIONNAMELIST** 属性创建新连接，假定没有来自连接到同一队列管理器的其他应用程序的合用连接可用。

当在 `close()` 调用之后释放连接时，会将此连接返回到池中，并在下次运行该 servlet 时进行复用（无需引用 **CONNECTIONNAMELIST**），直到发生连接故障（此时会生成 CONNECTION_ERROR_OCCURRED 事件）为止。 此事件提示池销毁发生故障的连接。

下次运行应用程序时，将没有可用的合用连接，并会使用 **CONNECTIONNAMELIST** 来连接到第一个可用的队列管理器。 如果发生队列管理器故障转移（例如，发生非暂时性网络故障），那么 servlet 将在备份实例可用时立即连接到该实例。

如果应用程序中包含其他资源（如数据库），那么可能需要指示应用程序服务器回滚该事务。

在应用程序内处理重新连接

如果调用者无法处理 servlet 中的故障，那么必须在应用程序内处理重新连接。 如下例所示，要处理应用程序中的重新连接需要应用程序请求新连接，以便它可以高速缓存从 JNDI 查找的连接工厂，并处理 `JMSException`，例如 `JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2009' ('MQRC_CONNECTION_BROKEN')`。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
```

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

在下面的示例中，setupResources() 将创建 JMS 对象，并包含一个“睡眠和重试”循环以处理非即时重新连接。 事实上，此方法会阻止许多重新连接尝试。 请注意，为了清晰起见，示例中已省略出口条件。

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

如果由应用程序管理重新连接，那么应用程序应尽可能地释放与其他资源的连接，无论这些资源是其他 IBM MQ 队列管理器还是数据库之类的其他后端服务。 重新连接到新的 IBM MQ 队列管理器实例之后，您必须重新建立这些连接。 如果没有重新建立连接，那么在重新连接尝试期间，会导致不必要地占用应用程序服务器资源，并导致这些资源在复用之前便已超时。

WorkManager 的用法

对于处理时间超过数十秒的长时间运行的应用程序（例如，批处理），可以使用 WebSphere Application Server WorkManager。 下面是 WebSphere Application Server 的代码片段示例：

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

其中 web.xml 包含：

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

而且，现在已通过 Work 接口实现了批处理：

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
               ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

例如，如果需要较长时间来运行批处理（例如，因为大型消息、网速较慢或大量数据库访问（尤其是伴随着缓慢的故障转移）），那么服务器会开始输出挂起线程警告，这些警告类似于以下示例：

WSVR0605W：线程“WorkManager.DefaultWorkManager : 0”(00000035) 已保持活动 694061 毫秒，可能已挂起。 服务器中总共有 1 个线程可能挂起。

通过减小批处理大小或增加挂起线程超时，可以尽可能减少这类警告。 但是，一般而言，最好是在 EJB（针对批处理发送）或消息驱动的 bean（针对使用或使用并回复）处理中实现此处理。

请注意，应用程序管理的重新连接未提供用于处理运行时错误的通用解决方案，应用程序仍必须处理与连接故障无关的错误。

例如，尝试将消息放入已满队列 (2053 MQRC_Q_FULL) 或者尝试使用无效的安全凭证连接到队列管理器 (2035 MQRC_NOT_AUTHORIZED)。

应用程序还必须处理在进行故障转移期间没有可立即使用的实例时发生的 2059 MQRC_Q_MGR_NOT_AVAILABLE 错误。 这可以通过应用程序在发生 JMS 异常时立即报告异常来实现，而不是以静默方式尝试重新连接。

IBM MQ classes for JMS 对象池

使用 Java EE 之外的连接池形式有助于降低总体负载，例如，从使用框架的某些独立应用程序或部署到云环境的应用程序，另外到 QueueManagers 的客户机连接数量的增加也会导致应用程序和队列管理器的服务器整合的增加。

在 Java EE 编程模型中，各种对象的定义完善的生命周期正在使用中。 消息驱动的 bean (MDB) 是最受约束的，而 Servlet 提供更多自由。 因此， Java EE 服务器中提供的池选项适合所用的各种编程模型。

使用 Java SE（或使用其他框架，例如 Spring），编程模型极其灵活。 因此，单个池策略并非适合所有模型。 您应该考虑是否应有一个框架，可以建立任何形式的池，例如 Spring。

要使用的池策略取决于您的应用程序的运行环境。

在 *Java EE* 环境建立对象池

Java EE 应用程序服务器提供连接池功能，消息驱动的 bean 应用程序 Enterprise Java Bean 和 Servlet 可以使用该功能。

WebSphere Application Server 负责维护与 JMS 提供者的连接的池，以便提高性能。 当应用程序创建 JMS 连接时，应用程序服务器将确定空闲连接池中是否存在连接。 如果存在，会将连接返回给应用程序；否则，将创建新连接。

第 247 页的图 47 显示了激活规范和侦听器端口如何建立 JMS 连接并使用该连接以正常方式监视目标中的消息。

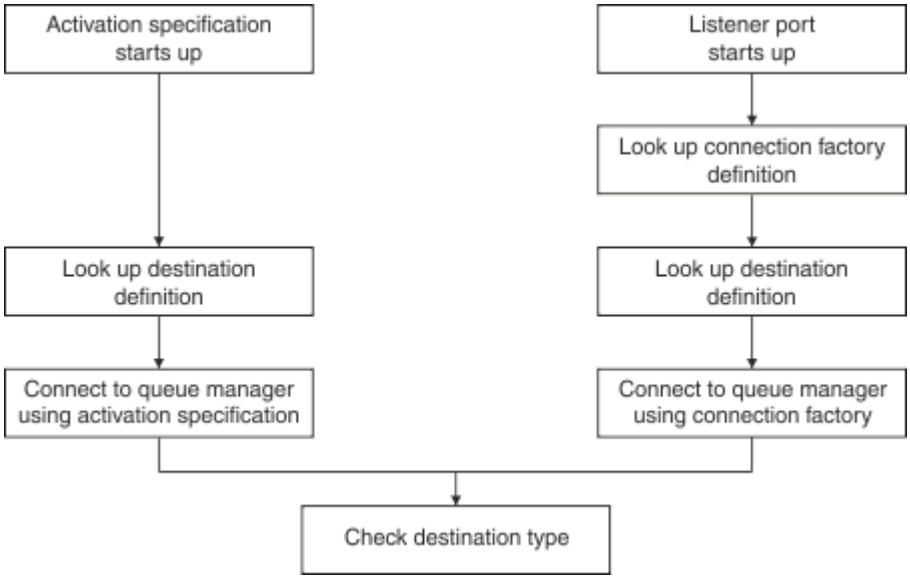


图 *47:* 正常方式

当使用 IBM MQ 消息传递提供者时，用于执行出站消息传递的应用程序（例如 Enterprise Java Bean 和 servlet）以及消息驱动的 bean 侦听器端口组件都可以使用这些连接池。

IBM MQ 消息传递提供者激活规范可使用 IBM MQ 资源适配器提供的连接池功能。 请参阅配置 WebSphere MQ 资源适配器的属性，以了解更多信息。

第 250 页的『使用连接池的示例』 介绍了用于执行出站消息传递的应用程序以及侦听器端口在创建 JMS 连接时如何使用空闲池。

第 253 页的『空闲连接池维护线程』 介绍了应用程序或侦听器端口使用完连接时将对这些连接执行的操作。

第 254 页的『池维护线程示例』 介绍了如何清理空闲连接池以防止 JMS 连接失效。

WebSphere Application Server 限制了可使用工厂创建的连接数，该限制由连接工厂的最大连接数属性指定。 该属性的缺省值为 10，表示任一时间最多可以使用工厂创建 10 个连接。

每个工厂都有关联的空闲连接池。 当启动应用程序服务器时，空闲连接池为空。 工厂的空闲池中可存在的最大连接数也由“最大连接数”属性指定。

提示: 通过 JMS 2.0，连接工厂可同时用于创建连接和上下文。 因此，连接池可以与包含连接和上下文的混合的连接工厂相关联。 建议连接工厂仅用于创建连接或创建上下文。 这确保该连接工厂的连接池仅包含单一类型的对象，从而使池更高效。

有关连接池如何在 WebSphere Application Server 中运作的信息，请参阅配置 JMS 连接的连接池。 对于其他应用程序服务器，请参阅相应的应用程序服务器文档。

如何使用连接池

每个 JMS 连接工厂都有关联的连接池，该连接池包含零个或更多个 JMS 连接。 每个 JMS 连接都有关联的 JMS 会话池，而每个 JMS 会话池包含零个或更多个 JMS 会话。

第 248 页的图 48 显示了这些对象之间的关系。

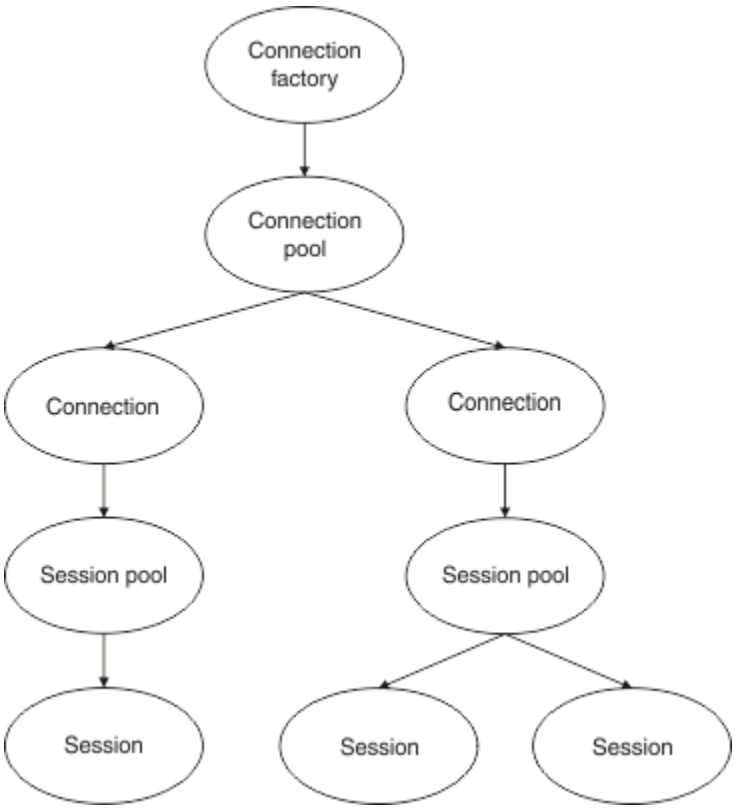


图 *48:* 连接池和会话池

启动侦听器端口，或用于执行出站消息传递的应用程序使用工厂创建连接时，该端口或应用程序会调用以下某个方法：

- **connectionFactory.createConnection()**
- **ConnectionFactory.createConnection(String, String)**
- **QueueConnectionFactory.createQueueConnection()**
- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

WebSphere Application Server 连接管理器会尝试从该工厂的空闲池中获取连接，并将其返回给应用程序。

如果池中没有空闲连接，并且使用该工厂创建的连接数尚未达到该工厂的最大连接数属性所指定的限制，那么连接管理器会创建新连接以供应用程序使用。

但是，如果应用程序尝试创建连接，但使用该工厂创建的连接数已达到该工厂的最大连接数属性值，那么应用程序将等待连接变为可用（即被放回到空闲池中）。

应用程序等待的时间在连接池的连接超时属性中指定，其缺省值为 180 秒。 如果在 180 秒内将连接放回到空闲池中，那么连接管理器会立即再次将该连接从池中取出并传递给应用程序。 但是，如果超过了超时时间段，那么会抛出 *ConnectionWaitTimeoutException*。

当应用程序使用完连接并通过调用以下方法来将其关闭时：

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

该连接实际上仍保持打开状态，并返回到空闲池中，以便可供其他应用程序复用。 因此，您可以将 WebSphere Application Server 与 JMS 提供者之间的连接保持打开状态，即使应用程序服务器上未在运行 JMS 应用程序也是如此。

高级连接池属性
可以使用一些高级属性来控制 JMS 连接池的行为。

浪涌保护

第 252 页的『用于执行出站消息传递的应用程序如何使用连接池』描述了 sendMessage() 方法（其中并入了 connectionFactory.createConnection()）的用法。

请考虑下列情况：有 50 个 EJB 在执行 ejbCreate() 方法的过程中全部使用同一个连接工厂来创建 JMS 连接。

如果所有这些 bean 都是同时创建的，并且该工厂的空闲连接池中没有连接，那么应用程序服务器将尝试同时创建与同一个 JMS 提供者的 50 个 JMS 连接。 结果是显著增加了 WebSphere Application Server 和 JMS 提供者上的负载。

浪涌保护属性通过限制任一时间可从连接工厂创建的 JMS 连接数，并与创建其他连接的时间错开，从而防止发生此情况。

可使用以下两个属性来限制任一时间的 JMS 连接数：

- 浪涌阈值
- 浪涌创建时间间隔。

当 EJB 应用程序尝试使用连接工厂创建 JMS 连接时，连接管理器将执行检查来了解正在创建的连接数。 如果该数字小于或等于 surge threshold 属性的值，那么连接管理器将继续打开新连接。

但是，如果要创建的连接数超过 surge threshold 属性，那么在创建和打开新连接之前，连接管理器将等待 surge creation interval 属性指定的时间段。

粘滞连接

如果 JMS 应用程序使用该连接向 JMS 提供程序发送请求，并且该提供程序在一定时间内未响应，那么会将 JMS 连接视为 stuck。

WebSphere Application Server 提供了一种检测 stuck JMS 连接的方法。要使用此功能，必须设置三个属性：

- 粘滞时间计时器
- 粘滞时间
- 粘滞阈值

第 254 页的『池维护线程示例』介绍了池维护线程如何定期运行并检查连接工厂空闲池的内容，以查找在一段时间内未使用或者存在时间过长的连接。

要检测粘滞连接，应用程序服务器还需要管理粘滞连接线程，该线程会检查从连接工厂创建的所有活动连接的状态，以了解它们当中是否有任何连接在等待 JMS 提供者回复。

粘滞连接线程何时运行由 Stuck time timer 属性确定。 此属性的缺省值为 0，表示从不运行粘滞连接检测。

如果线程找到一个正在等待响应的线程，那么它将确定它等待的时间长度，并将此时间与 Stuck time 属性的值进行比较。

如果 JMS 提供程序响应所花费的时间超过了 Stuck time 属性指定的时间，那么应用程序服务器会将 JMS 连接标记为卡住。

例如，假设连接工厂 jms/CF1 将 Stuck time timer 属性设置为 10 ，并将 Stuck time 属性设置为 15。

粘滞连接线程每 10 秒激活一次，并检查使用 jms/CF1 创建的任何连接是否等待来自 IBM MQ 的响应已超过 15 秒。

假设 EJB 使用 jms/CF1 创建到 IBM MQ 的 JMS 连接，然后通过调用 Connection.createSession()尝试使用该连接创建 JMS 会话。

但是，有不确定因素阻止 JMS 提供者对该请求作出响应。 也许是机器冻结，或者 JMS 提供者上运行的进程陷入死锁，从而阻止处理任何新工作：

在 EJB 调用 Connection.createSession() 10 秒后，粘滞连接计时器将变为活动状态，并查看使用 jms/CF1 创建的活动连接。

假设只有一个活动连接，例如，名为 c1 的连接。 第一个 EJB 已等待 10 秒来响应它发送到 c1 的请求，这小于 Stuck time 的值，因此卡住的连接计时器将忽略此连接并变为不活动状态。

再经过 10 秒，粘滞连接线程再次变为活动状态，并查看 jms/CF1 的活动连接。 像之前一样，假设仍只有一个连接 c1。

现在自第一个 EJB 调用 createSession() 后已过去 20 秒，而该 EJB 仍在等待响应。 20 秒比 Stuck time 属性中指定的时间长，因此粘滞连接线程将 c1 标记为粘滞。

如果再经过 5 秒，IBM MQ 最终做出响应，并且允许第一个 EJB 创建 JMS 会话，那么该连接将恢复使用。

应用程序服务器将计算从连接工厂创建的粘滞 JMS 连接的数量。 当应用程序使用该连接工厂来创建新的 JMS 连接，并且该工厂的空闲池中没有空闲连接时，连接管理器会将粘滞连接数与 Stuck threshold 属性的值进行比较。

如果粘滞连接数小于为 Stuck threshold 属性设置的值，那么连接管理器将创建新连接并将其提供给应用程序。

但是，如果粘滞连接数等于 Stuck threshold 属性的值，那么应用程序将发生资源异常。

池分区

WebSphere Application Server 提供了两个用于对连接工厂空闲连接池进行分区的属性：

- Number of free pool partitions 告诉应用程序服务器要将空闲连接池划分为多少个分区。
- Free pool distribution table size 确定如何对分区建立索引。

除非 IBM 支持中心要求您更改这些属性的值，否则请将这些属性保持为缺省值 0。

请注意， WebSphere Application Server 还有一个名为 Number of shared partitions 的高级连接池属性。 该属性指定用于存储共享连接的分区数。 但是，由于 JMS 连接始终为非共享连接，因此此属性不适用。

使用连接池的示例

消息驱动的 bean 侦听器端口组件和用于执行出站消息传递的应用程序都使用 JMS 连接池。

第 251 页的图 49 显示了 WebSphere Application Server V7.5 和 V8.0 中连接池的工作方式。

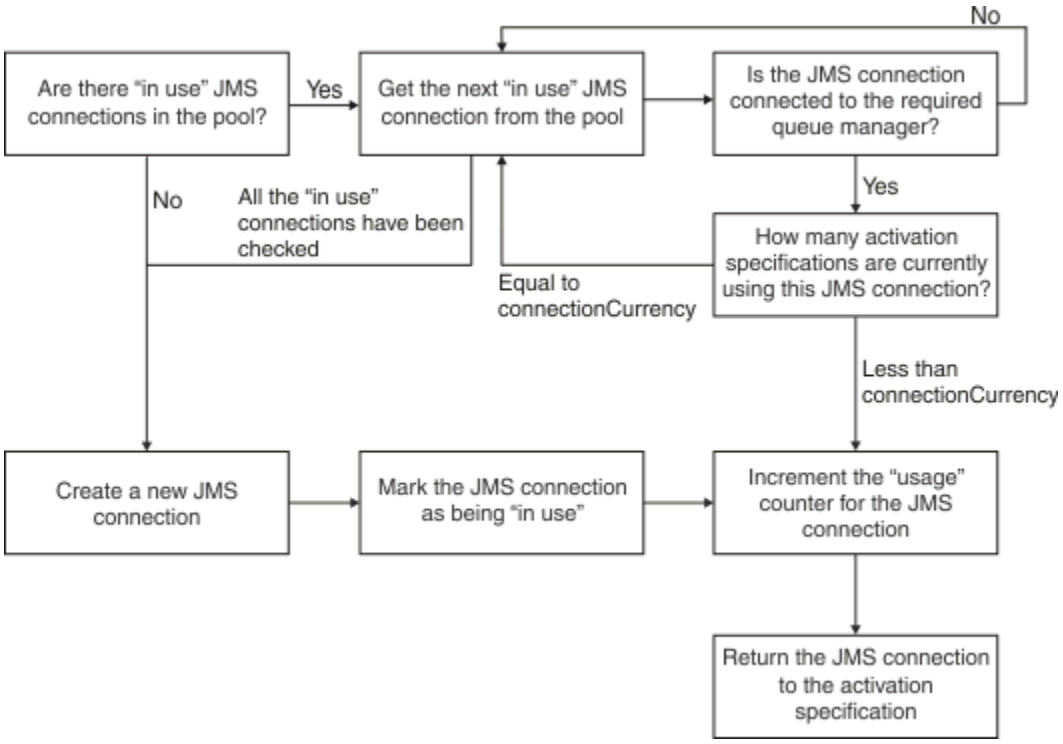


图 *49: WebSphere Application Server V7.5* 和 *V8.0* - 连接池的工作方式

第 251 页的图 50 显示了 WebSphere Application Server V8.5 中连接池的工作方式。

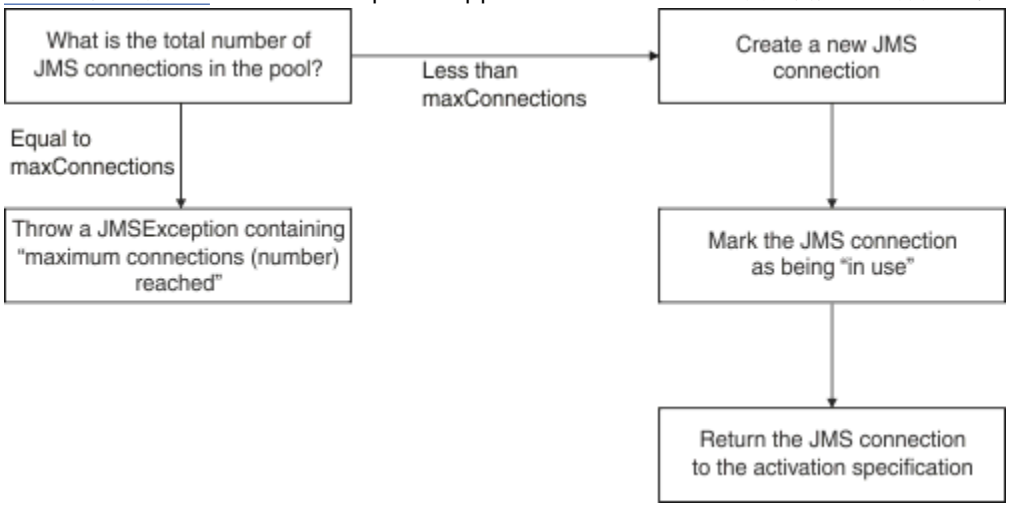


图 *50: WebSphere Application Server V8.5* - 连接池的工作方式

MDB 侦听器端口如何使用连接池

假定您在 WebSphere Application Server Network Deployment 系统上部署了 MDB，即使用 IBM MQ 作为 JMS 提供程序。 该 MDB 是针对使用某个连接工厂（例如名为 `jms/CF1`）的侦听器端口进行部署，并且最大连接数属性设置为 2，这意味着任一时间只能使用该工厂创建两个连接。

当侦听器端口启动时，该端口尝试使用 `jms/CF1` 连接工厂创建与 IBM MQ 的连接。

要执行此操作，该端口需要从连接管理器处请求连接。 由于这是第一次使用 `jms/CF1` 连接工厂，所以 `jms/CF1` 空闲连接池中没有连接，于是，连接管理器将创建新连接（例如名为 `c1`）。 请注意，此连接在侦听器端口的整个生命周期中始终存在。

现在，假设您使用 WebSphere Application Server 管理控制台停止了该侦听器端口。 在此情况下，连接管理器将获取该连接并将其放回到空闲池中。 但是，与 IBM MQ 的连接仍保持打开状态。

如果重新启动侦听器端口，该端口会再次要求连接管理器提供与队列管理器的连接。 由于空闲池中现在已有一个连接 (c1)，因此连接管理器将从池中取出此连接，使其可供侦听器端口使用。

现在，假设您在应用程序服务器中部署了第二个 MDB，并且它使用不同的侦听器端口。

再假设您尝试启动第三个侦听器端口，该端口也配置为使用 jms/CF1 连接工厂。 这第三个侦听器端口从连接管理器处请求连接，然后连接管理器会查看 jms/CF1 的空闲池，发现该池为空。 随后它检查使用 jms/CF1 工厂创建的连接数。

由于 jms/CF1 的“最大连接数”属性设置为 2，并且您已使用该工厂创建了两个连接，因此连接管理器将等待 180 秒（这是连接超时属性的缺省值），以等待有连接变为可用。

但是，如果您停止了第一个侦听器端口，那么其连接 c1 将被放入 jms/CF1 的空闲池中。 连接管理器会检索此连接并将其分配给第三个侦听器。

如果现在尝试重新启动第一个侦听器，那么该侦听器必须等待另外两个侦听器端口中的某一个停止，然后第一个侦听器才能重新启动。 如果正在运行的侦听器端口在 180 秒内均未停止，那么第一个侦听器会收到 ConnectionWaitTimeoutException 错误并停止。

用于执行出站消息传递的应用程序如何使用连接池

对于此选项，假设应用程序服务器中只安装了一个 EJB（例如名为 EJB1）。 该 bean 通过以下方法实现了名为 sendMessage() 的方法：

- 使用 connectionFactory.createConnection()从工厂 jms/CF1 创建到 IBM MQ 的 JMS 连接。
- 从该连接创建 JMS 会话。
- 从该会话创建消息生产者。
- 发送消息。
- 关闭生产者。
- 关闭会话。
- 调用 connection.close() 以关闭连接。

假设工厂 jms/CF1 的空闲池为空。 首次调用 EJB 时， Bean 尝试从工厂 jms/CF1 创建与 IBM MQ 的连接。 由于该工厂的空闲池为空，因此连接管理器会创建新连接并将其分配给 EJB1。

该方法恰好在退出之前调用 connection.close()。 连接管理器不会关闭 c1，而是获取该连接并将其放入 jms/CF1 的空闲池中。

下次调用 sendMessage() 时， connectionFactory.createConnection() 方法会将 c1 返回到应用程序。

假设在第一个实例运行的同时还有第二个 EJB 实例也在运行。 当这两个实例均调用 sendMessage() 时，会使用 jms/CF1 连接工厂创建两个连接。

现在假设创建了该 bean 的第三个实例。 当第三个 bean 调用 sendMessage() 时，该方法会调用 connectionFactory.createConnection() 以使用 jms/CF1 创建连接。

但是，当前已使用 jms/CF1 创建了两个连接，此数目已达到此工厂的最大连接数。 因此，createConnection() 方法将等待 180 秒（这是连接超时属性的缺省值），以等待有连接变为可用。

但是，如果第一个 EJB 的 sendMessage() 方法调用 connection.close() 并退出，那么它之前使用的连接 c1 将被放回到空闲连接池中。 连接管理器将从空闲池中取回该连接并将其分配给第三个 EJB。 然后将返回该 bean 对 connectionFactory.createConnection() 的调用，从而允许完成 sendMessage() 方法。

使用相同连接池的 MDB 侦听器端口和 EJB

上述两个示例显示了侦听器端口和 EJB 如何独立使用连接池。 但是，侦听器端口和 EJB 可以同时在同一个应用程序服务器中运行，并且使用同一个连接工厂创建 JMS 连接。

您需要考虑此情况的影响

需要记住的关键一点是连接工厂由侦听器端口和 EJB 共享。

例如，假设您的侦听器和 EJB 同时运行。 这二者均使用 jms/CF1 连接工厂，这意味着已达到该工厂的“最大连接数”属性指定的连接限制。

无论是尝试启动另一个侦听器端口还是 EJB 的另一个实例，都必须等待连接返回到 jms/CF1 的空闲连接池。

空闲连接池维护线程
每个空闲连接池都有关联的池维护线程，该线程用于监视空闲池以确保其中的连接仍然有效。

如果池维护线程决定需要废弃空闲池中的连接，那么该线程将以物理方式关闭与 IBM MQ 的 JMS 连接。

池维护线程的工作方式

池维护线程的行为由连接池的以下四个属性值决定：

过时超时
连接保持打开的时长。

最小连接数
连接管理器保留在连接工厂空闲池中的最小连接数。

收集时间
池维护线程运行的频率。

未使用超时
连接在关闭之前保留在空闲池中的时长。

缺省情况下，池维护的线程每 180 秒运行一次，尽管可以通过设置连接池 **Reap time** 属性来更改此值。

维护线程将查看池中的每个连接，检查连接保留在池中的时长，以及自创建和上次使用以来经过的时间。

如果未使用连接的时间段超过连接池的 **Unused timeout** 属性的值，那么维护线程将检查空闲池中当前的连接数。 如果该数量：

- 如果大于 **Minimum connections** 的值，那么连接管理器将关闭连接。
- 等于 **Minimum connections** 的值，连接未关闭并保留在空闲池中。

Minimum connections 属性的缺省值为 *1*，这意味着出于性能原因，连接管理器始终尝试在空闲池中至少保留一个连接。

Unused timeout 属性的缺省值为 1800 秒。 缺省情况下，如果将连接放回到空闲池，并且再次未使用时间至少为 1800 秒，那么将关闭该连接，但前提是关闭该连接后，空闲池中至少还有一个连接。

此过程可防止未使用的连接失效。 要关闭此功能，请将 **Unused timeout** 属性设置为零。

如果连接位于空闲池中，并且自其创建以来的耗用时间大于连接池的 **Aged timeout** 属性值，那么将关闭该连接，而无论自上次使用以来已经过了多长时间。

缺省情况下， **Aged timeout** 属性设置为零，这意味着维护线程从不执行此检查。 无论空闲池中将保留多少个连接，都将废弃存在时间超过 **Aged timeout** 属性的连接。 请注意， **Minimum connections** 属性在此情况下没有影响。

禁用池维护线程

从上述描述中可以看出，池维护线程在活动期间执行大量工作，当连接工厂空闲池中存在大量连接时尤为明显。

例如，假设有三个 JMS 连接工厂，每个工厂的 **Maximum connections** 属性都设置为 10。 每经过 180 秒，三个池维护线程就变为活动状态，并扫描各自连接工厂的空闲池。 如果空闲池包含许多连接，那么维护线程需要执行大量工作，这可能极大地影响性能。

您可以通过将单个空闲连接池的 **Reap time** 属性设置为零来禁用该连接池的池维护线程。

禁用维护线程意味着永远不会关闭连接，即使已经过 **Unused timeout** 也是如此。 但是，如果 **Aged timeout** 已通过，那么仍可以关闭连接。

当应用程序完成连接时，连接管理器将检查连接的存在时间，如果该时间段比 **Aged timeout** 属性的值长，那么连接管理器将关闭连接，而不是将其返回到空闲池。

过时超时的事务影响

如上一节中所述，**Aged timeout** 属性指定在连接管理器关闭与 JMS 提供程序的连接之前，该连接保持打开状态的时间长度。

Aged timeout 属性的缺省值为零，这意味着将永远不会关闭连接，因为它太旧。 您应该将 **Aged timeout** 属性保留在此值，因为在 EJB 中使用 JMS 时，启用 **Aged timeout** 可能会产生事务性影响。

在 JMS 中，事务单元是从 JMS 连接创建的 JMS 会话。 这是征调到事务中的 JMS 会话，而不是 JMS 连接。

由于应用程序服务器的设计，可以关闭 JMS 连接，因为已经过 **Aged timeout**，即使在事务中涉及从该连接创建的 JMS 个会话也是如此。

关闭 JMS 连接可能导致回滚 JMS 会话上任何未完成的事务工作，如 JMS 规范中所述。 但是，应用程序服务器并不知道从该连接创建的 JMS 会话不再有效。 当服务器尝试使用该会话来落实或回滚事务时，将发生 `IllegalStateException`。

要点: 如果要将 **Aged timeout** 与来自 EJB 内的 JMS 连接配合使用，请确保在执行 JMS 操作的 EJB 方法退出之前，在 JMS 会话上显式落实任何 JMS 工作。

池维护线程示例
使用 Enterprise Java Bean (EJB) 示例来了解池维护线程的工作方式。 请注意，您也可以使用消息驱动的 Bean (MDB) 和侦听器端口，您只需通过某种方式获取空闲池中的连接即可。

请参阅第 252 页的『用于执行出站消息传递的应用程序如何使用连接池』，以了解 `sendMessage()` 方法的更多详细信息。

您已为连接工厂配置了以下值:

- **Reap time** 的缺省值为 180 秒
- **Aged timeout** 的缺省值为零秒
- **Unused timeout** 设置为 300 秒

在启动应用程序服务器之后，将调用 `sendMessage()` 方法。

此方法使用 `jms/CF1` 工厂创建一个连接（例如名为 `c1`），使用该工厂发送消息，然后调用 `connection.close()` 以使 `c1` 能够被放入空闲池中。

180 秒之后，池维护线程将启动，并查看 `jms/CF1` 的空闲连接池。 该池中有一个空闲连接 `c1`，因此维护线程会查看放回该连接的时间，并将该值与当前时间进行比较。

自连接放入空闲池以来已经过 180 秒，这小于 `jms/CF1` 的 **Unused timeout** 属性的值。 因此，维护线程不对该连接进行处理。

再过 180 秒之后，池维护线程再次运行。 维护线程会找到连接 `c1`，并确定该连接已在池中 360 秒，这比设置的 **Unused timeout** 值长，因此连接管理器会关闭该连接。

如果您现在再次运行 `sendMessage()` 方法，那么当应用程序调用 `connectionFactory.createConnection()` 时，连接管理器会创建与 IBM MQ 的新连接，因为该连接工厂的空闲连接池为空。

上述示例显示了当 **Aged timeout** 属性设置为零时，维护线程如何使用 **Reap time** 和 **Unused timeout** 属性来防止旧连接。

Aged timeout 属性如何工作?

在以下示例中，假设您已经将:

- **Aged timeout** 属性到 300 秒
- **Unused timeout** 属性为零。

您调用 `sendMessage()` 方法，此方法尝试使用 `jms/CF1` 连接工厂创建连接。

由于该工厂的空闲池为空，因此连接管理器会创建新连接 `c1`，并将其返回给应用程序。 当 `sendMessage()` 调用 `connection.close()` 时，`c1` 将被放回到空闲连接池中。

180 秒之后，池维护线程将运行。 该线程在空闲连接池中发现 c1，并检查自其创建以来经过的时间。 连接已存在 180 秒，这小于 **Aged timeout**，因此池维护线程会将其单独保留并返回到休眠状态。

60 秒之后，再次调用 sendMessage()。 这一次，当该方法调用 connectionFactory.createConnection() 时，连接管理器将发现 jms/CF1 空闲池中有一个可用连接 c1。 连接管理器将 c1 从空闲池中取出，并将该连接分配给应用程序。

当 sendMessage() 退出时，该连接将被返回到空闲池中。 120 秒之后，池维护线程再次被唤醒，扫描 jms/CF1 的空闲池内容，并发现 c1。

虽然 120 秒前仅使用了该连接，但池维护线程会关闭该连接，因为该连接已存在总共 360 秒，这比您为 **Aged timeout** 属性设置的 300 秒值长。

“最低连接数”属性对池维护线程的影响

仍以 第 251 页的『MDB 侦听器端口如何使用连接池』为例，假设您在应用程序服务器中部署了两个 MDB，每个 MDB 使用不同的侦听器端口。

每个侦听器端口均配置为使用 jms/CF1 连接工厂，并且您为该连接工厂配置了以下属性值：

- **Unused timeout** 属性设置为 120 秒
- **Reap time** 属性设置为 180 秒
- **Minimum connections** 属性设置为 1

假设第一个侦听器已停止，其连接 c1 已被放入空闲池中。 180 秒后，池维护线程将唤醒，扫描空闲池的内容以查找 jms/CF1，并发现 c1 在空闲池中的时间长于连接工厂的 **Unused timeout** 属性值。

但是，在关闭 c1 之前，池维护线程将执行检查来了解在废弃此连接后池中将保留的连接数。 由于 c1 是空闲连接池中的唯一连接，因此连接管理器不会将其关闭，因为这样做会使保留在空闲池中的连接数小于为 **Minimum connections** 设置的值。

现在，假设第二个侦听器已停止。 空闲连接池现在包含两个空闲连接 - c1 和 c2。

再过 180 秒之后，池维护线程再次运行。 此时，c1 在空闲连接池中已存在 360 秒，c2 已存在 180 秒。

池维护线程将检查 c1 并发现它在池中的时间超过 **Unused timeout** 属性的值。

然后，线程将检查可用池中的连接数，并将其与 **Minimum connections** 属性的值进行比较。 由于池包含两个连接，并且 **Minimum connections** 设置为 1，因此连接管理器将关闭 c1。

维护线程现在会查看 c2。 这在空闲连接池中的时间也超过了 **Unused timeout** 属性的值。 但是，由于关闭 c2 将使空闲连接池中的连接数少于设置的最小连接数，因此连接管理器将单独保留 c2。

JMS 连接和 *IBM MQ*

下面提供了有关将 IBM MQ 用作 JMS 提供者的信息。

使用 BINDINGS 传输

如果已将连接工厂配置为使用绑定传输，那么每个 JMS 连接都会与 IBM MQ 建立对话 (也称为 **hconn**)。 该对话使用进程间通信（或共享内存）来与队列管理器进行通信。

使用 CLIENT 传输

当 IBM MQ 消息传递提供程序连接工厂配置为使用客户机传输时，从该工厂创建的每个连接都将建立到 IBM MQ 的新对话 (也称为 **hconn**)。

对于使用 IBM MQ 消息传递提供者正常方式连接到队列管理器的连接工厂，可以使用该连接工厂创建多个 JMS 连接以共享与 IBM MQ 的 TCP/IP 连接。 有关更多信息，请参阅第 257 页的『在 IBM MQ classes for JMS 中共享 TCP/IP 连接』。

要确定 JMS 连接在任一时间使用的最大客户机通道数，可以将指向同一队列管理器的所有连接工厂的最大连接数属性值相加。

例如，假设您有两个连接工厂（jms/CF1 和 jms/CF2），它们已配置为使用同一个 IBM MQ 通道连接到同一个 IBM MQ 队列管理器。

这两个工厂使用缺省连接池属性，这表示最大连接数设置为 10。 如果同时使用了来自 jms/CF1 和 jms/CF2 的所有连接，那么应用程序服务器和 IBM MQ 之间将有 20 个对话。

如果连接工厂使用 IBM MQ 消息传递提供者正常方式连接到队列管理器，那么对于这些连接工厂而言，应用程序服务器和队列管理器之间可以存在的最大 TCP/IP 连接数为：

```
20/the value of SHARECNV for the IBM MQ channel
```

如果连接工厂已配置为使用 IBM MQ 消息传递提供者迁移方式进行连接，那么对于这些连接工厂而言，应用程序服务器和 IBM MQ 之间的最大 TCP/IP 连接数将为 20（每个 TCP/IP 连接对应于两个工厂的连接池中的一个 JMS 连接）。

相关概念

第 67 页的『使用 IBM MQ classes for JMS』

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

Java SE 环境中的对象池

使用 Java SE（或使用其他框架，例如 Spring），编程模型极其灵活。 因此，单个池策略并非适合所有模型。 您应该考虑是否有一个框架，可以建立任何形式的池，例如 Spring。

否则，应用程序逻辑会执行此操作。 问问自己应用程序本身有多复杂？ 最好了解一下应用程序以及应用程序对指向消息传递系统的连接的要求什么。 另外，应用程序通常也会围绕 JMS API 在自己的包装器代码中进行编写。

这可能是非常敏感的方法，并且可以隐藏复杂性，但是需要引起注意的是它也可能引入问题。 例如，频繁调用的通用 `getMessage()` 方法不应仅打开和关闭使用者。

您应当考虑的几点：

- 应用程序访问 IBM MQ 需要多长时间？ 一直，还是只是偶尔。
- 消息发送的频率是多少？ 频率越低，就越有可能分享 IBM MQ 的单个连接。
- 连接断开异常通常表明需要重新创建合用的连接。 相关内容：
  - 安全性异常或主机不可用
  - 队列已满异常
- 如果发生连接中断异常，池中其他空闲连接将发生什么情况？ 应将它们关掉并重新创建吗？
- 例如，如果正在使用 TLS，您希望单个连接保持打开的时间为多少？
- 加入池的连接如何标识自己，使得队列管理器可以发现该连接并对其进行跟踪。

您应当考虑对所有 JMS 对象建立池，并随时将可加入池的对象加入到池中。 这些对象包括：

- JMS 连接
- Session
- 上下文
- 所有不同类型的生产者和使用者

使用客户机传输时， JMS 连接、会话和上下文与 IBM MQ 队列管理器通信时将使用套接字。 通过为这些对象建立池，节省了至队列管理器的一些入站 IBM MQ 连接 (hConns) 并降低了通道实例的数量。

使用队列管理器绑定传输彻底移除网络层。 但是，许多应用程序使用客户机传输提供可用性更高、工作负载更均衡的配置。

JMS 生产者和使用者打开队列管理器上的目的地。 如果打开的队列或主题数量较少，并且应用程序多个部分在使用这些对象，那么为这些项建立池可能非常有用。

从 IBM MQ 角度，此过程省去一系列 MQOPEN 和 MQCLOSE 操作。

连接、会话和上下文

这些对象都将 IBM MQ 连接句柄封装到队列管理器，并从 `ConnectionFactory` 生成。 您可以将逻辑添加到应用程序以将连接和从单个连接工厂创建的其他对象的数量限制为具体数量。

您可在应用程序中使用简单数据结构以包含所创建的连接。 需要使用其中一个数据结构的应用程序代码可以检出要使用的对象。

需考虑以下因素：

- 什么时候从池移除连接？ 通常，在连接上创建异常侦听器。 调用该侦听器来处理异常时，您应该重新创建连接以及之前从该连接创建的任何会话。
- 如果 CCDT 正在用于均衡工作负载，那么连接可能转到其他队列管理器。 这可能适用于池需求。

记住，JMS 规范指出，多个线程同时访问会话或上下文是编程错误。 IBM MQ JMS 代码确实试图在处理线程时严格一些。 但是，您应该向应用程序添加逻辑，以确保同时只有一个线程在使用会话或上下文对象。

生产者和使用者

所创建的每个生产者和使用者打开队列管理器上的目标。 如果同样的目标将用于各种任务，让使用者或生产者对象保持打开是有意义的。 仅在所有工作都完成时关闭对象。

尽管打开和关闭目标是短操作，但是如果频繁采取此操作，也可能花费很多时间。

这些对象的作用域在创建它们的会话或上下文中，因此需要将其保留在该作用域中。 通常，应用程序的编写方式使得这样做非常直接。

监视

应用程序将如何监视其对象池？ 此问题的答案很大程度上由实现的池解决方案的复杂性确定。

如果您考虑 JavaEE 池实现，有大量选项，包括：

- 池的当前大小
- 花费在池上的时间对象
- 清除池
- 连接的更新

您还应该考虑单个重复使用的会话如何显示在队列管理器上。 识别应用程序的连接工厂属性（例如 `appName`）可能非常有用。

第 67 页的『使用 IBM MQ classes for JMS』
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

在 *IBM MQ classes for JMS* 中共享 *TCP/IP* 连接
可以让 MQI 通道的多个实例共享一个 TCP/IP 连接。

可以使以下应用程序共享相同通道实例：正在相同 Java 运行时环境中运行的应用程序，以及使用 IBM MQ classes for JMS 或 IBM MQ 资源适配器通过 CLIENT 传输连接到队列管理器的应用程序。

通道实例和 TCP/IP 连接之间具有一对一关系。 将为每个通道实例创建一个 TCP/IP 连接。

如果在将 **SHARECNV** 参数设置为大于 1 的值的情况下定义通道，那么该数目的对话可以共享通道实例。 要使连接工厂或激活规范能够使用此功能，请将 **SHARECONVALLOWED** 属性设置为 YES。

JMS 应用程序创建的每个 JMS 连接和 JMS 会话都会创建自己与队列管理器的对话。

激活规范启动时，IBM MQ 资源适配器会启动与队列管理器的对话，以供激活规范使用。 服务器会话池中与此激活规范关联的每个服务器会话也会启动与队列管理器的对话。

SHARECNV 属性是尽力而为的连接共享方法。 因此，当大于 0 的 SHARECNV 值与 IBM MQ classes for JMS 一起使用时，则不能保证新的连接请求始终共享已经建立的连接。

计算通道实例数

使用以下公式确定将由应用程序创建的最大通道实例数。

激活规范

通道实例数 = (*maxPoolDepth_value* + 1) / *SHARECNV_value*

其中，*maxPoolDepth_value* 是 **maxPoolDepth** 属性的值，*SHARECNV_value* 是激活规范所使用的通道上的 **SHARECNV** 属性的值。

其他 JMS 应用程序

通道实例数 = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

其中，*jms_connections* 是由应用程序创建的连接数，*jms_sessions* 是由应用程序创建的 JMS 会话数，*SHARECNV_value* 是激活规范所使用的通道上 **SHARECNV** 属性的值。

示例

以下示例说明如何使用公式，计算应用程序通过 IBM MQ classes for JMS 或 IBM MQ 资源适配器在队列管理器上创建的通道实例数。

JMS 应用程序示例

JMS 应用程序连接通过使用 CLIENT 传输连接到队列管理器，创建一个 JMS 连接和三个 JMS 会话。 应用程序用于连接到队列管理器的通道的 **SHARECNV** 属性值设置为 10。 当应用程序正在运行时，应用程序和队列管理器之间具有四个对话，同时具有一个通道实例。 四个对话都共享此通道实例。

V 9.1.3 从 IBM MQ 9.1.3 开始，如果将应用程序配置为 `reconnectable`，那么只能在相关 JMS 对象（即 JMS 连接及其相关 JMS 会话）之间共享通道实例。 这可能需要配置其他通道实例才能支持此类应用程序。

例如，如果应用程序使用单个 JMS 连接和单个 JMS 会话，并且所使用的通道 **SHARECNV** 等于 10，在 IBM MQ 9.1.3 之前，最多可以有 5 个应用程序实例共享单个通道实例。 对于 IBM MQ 9.1.3 或更高版本，如果应用程序未配置为 `reconnectable`，但如果应用程序配置为 `reconnectable`，那么每个应用程序实例都将需要其自己的通道实例，因此总共需要 5 个通道实例。

激活规范示例

激活规范使用 CLIENT 传输连接到队列管理器。 激活规范已配置为将 **maxPoolDepth** 属性设置为 10。 将激活规范配置为使用的通道将 **SHARECNV** 属性设置为 10。 当激活规范正在运行且并行处理 10 条消息时，激活规范和队列管理器之间的对话数为 11（针对服务器会话具有 10 个对话，针对激活规范具有 1 个对话）。 激活规范所使用的通道实例数为 2。

激活规范示例

激活规范使用 CLIENT 传输连接到队列管理器。 激活规范已配置为将 **maxPoolDepth** 属性设置为 5。 将激活规范配置为使用的通道将 **SHARECNV** 属性设置为 0。 当激活规范正在运行并同时处理 5 条消息时，激活规范与队列管理器之间的对话数为 6（服务器会话的对话数为 5，激活规范的对话数为 1）。 激活规范所使用的通道实例数为 6，因为通道上的 **SHARECNV** 属性设置为 0，所以每个对话使用其自己的通道实例。

相关任务

第 417 页的『确定已创建的从 WebSphere Application Server 到 IBM MQ 的 TCP/IP 连接数』
通过使用共享对话功能，多个对话可以共享 MQI 通道实例，这也称为 TCP/IP 连接。

为 *IBM MQ classes for JMS* 中的客户机连接指定端口范围
使用 LOCALADDRESS 属性指定应用程序可绑定到的端口范围。

当 IBM MQ classes for JMS 应用程序尝试以客户机模式连接到 IBM MQ 队列管理器时，防火墙可能只允许从指定端口或端口范围发起的那些连接。 在此情况下，可使用 ConnectionFactory、QueueConnectionFactory 或 TopicConnectionFactory 对象的 LOCALADDRESS 属性，指定应用程序可绑定到的端口或端口范围。

您可以使用 IBM MQ JMS 管理工具或通过在 JMS 应用程序中调用 setLocalAddress() 方法来设置 LOCALADDRESS 属性。 此处是从应用程序中设置属性的示例：

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

当应用程序后续连接到队列管理器时，会绑定到范围在 192.0.2.0(2000) 到 192.0.2.0(3000) 之间的本地 IP 地址和端口号。

在具有多个网络接口的系统中，还可使用 LOCALADDRESS 属性指定必须用于连接的网络接口。

对于代理的实时连接，仅当使用多点广播时，LOCALADDRESS 属性才相关。 在此情况下，可以使用属性来指定必须用于连接的本地网络接口，但此属性的值不得包含端口号或端口号范围。

如果限制端口范围，那么可能会发生连接错误。 如果发生错误，那么会抛出 JMSException，并且嵌入了 MQException，其中包含 IBM MQ 原因码 MQRC_Q_MGR_NOT_AVAILABLE 和以下消息：

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

如果指定范围内的所有端口都在使用，或者指定的 IP 地址、主机名或端口号无效（例如，负端口号），可能会出现错误。

由于 IBM MQ classes for JMS 创建的连接可能不是应用程序所需的连接，请始终考虑指定端口范围。 通常，应用程序创建的每个会话都需要一个端口，IBM MQ classes for JMS 可能需要三个或四个额外的端口。 如果发生连接错误，请增大端口范围。

缺省情况下在 IBM MQ classes for JMS 中使用的连接池，可能会对可复用端口的速度产生影响。 因此，释放端口时，可能会发生连接错误。

IBM MQ classes for JMS 中的通道压缩

IBM MQ classes for JMS 应用程序可使用 IBM MQ 工具来压缩消息头或数据。

压缩 IBM MQ 通道上流动的数据可以提高通道性能并降低网络流量。 使用随 IBM MQ 提供的功能，您可以压缩消息通道和 MQI 通道上流动的数据。 对于每种通道类型，您都能独立压缩头数据和消息数据。 缺省情况下，通道上的数据都不会压缩。

IBM MQ classes for JMS 应用程序会通过创建 java.util.Collection 对象，指定用于在连接时压缩头或消息数据的方法。 每种压缩方法都是集合中的整数对象，且应用程序向集合添加压缩方法的顺序，就是压缩方法在应用程序创建连接时与队列管理器协商的顺序。 然后，应用程序可通过调用 setHdrCompList() 方法（对于头数据）或 setMsgCompList() 方法（对于消息数据）将集合传递到 ConnectionFactory 对象。 应用程序准备就绪后，可创建连接。

以下代码片段展示了所述方法。 第一个代码片段显示了如何执行头数据压缩：

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

第二个代码片段显示了如何执行消息数据压缩：

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

在第二个示例中，在创建连接时，将按顺序 RLE 和 ZLIBHIGH 协商压缩技术。 在 Connection 对象的生命周期内，选择的压缩方法无法更改。 要在连接上使用压缩，必须先调用 setHdrCompList() 和 setMsgCompList() 方法，然后再创建 Connection 对象。

在 *IBM MQ classes for JMS* 中异步放置消息

通常，当应用程序向目标发送消息时，该应用程序必须等待队列管理器确认它已处理该请求。 在某些情况下，可以通过改为选择异步放置消息来提高消息传递性能。 应用程序异步放置消息时，队列管理器不会返回每个调用的成功或失败状态，但可以改为定期检查错误。

在不确定队列管理器是否已安全接收消息的情况下，根据以下属性确定目标是否向应用程序返回控制：

JMS 目标属性 PUTASYNCALLOWED（短名称 - PAALD）。
PUTASYNCALLOWED 控制 JMS 应用程序是否可异步放置消息（如果 JMS 目标表示的底层队列或主题允许此选项）。

IBM MQ 队列或主题属性 DEFPRESP（缺省放置响应类型）。
DEFPRESP 指定向队列放置消息或将消息发布到主题的应用程序是否可使用异步放置功能。

下表显示 PUTASYNCALLOWED 和 DEFPRESP 属性的可能值以及用于启用异步放置功能的值组合：

表 *45: PUTASYNCALLOWED* 和 *DEFPRESP* 属性如何组合以确定是否将消息异步放置到目标。

IBM MQ 队列属性	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**PUTASYNCALLOWED = AS_DEST 或 AS_Q_DEF 或 AS_T_DEF**
DEFPRESP=SYNC	未启用异步放置功能	已启用异步放置功能	未启用异步放置功能
DEFPRESP=ASYNC	未启用异步放置功能	已启用异步放置功能	已启用异步放置功能

对于在事务性会话中发送的消息，应用程序最终确定队列管理器在调用 `commit()` 时是否安全地接收消息。

如果应用程序在事务性会话中发送持久消息，且未安全地接收一条或多条消息，那么事务未能落实，并会生成异常。 但是，如果应用程序在事务性会话中发送非持久消息，且未安全地接收一条或多条消息，那么事务会成功落实。 应用程序不会收到有关非持久消息未安全到达的任何反馈。

对于在非事务性会话中发送的非持久消息，*ConnectionFactory* 对象的 SENDCHECKCOUNT 属性指定在 IBM MQ classes for JMS 核实队列管理器安全地接收消息之前发送多少条消息。

如果检查发现未安全地接收一条或多条消息，且应用程序向连接注册了异常侦听器，那么 IBM MQ classes for JMS 会调用异常侦听器的 `onException()` 方法以向应用程序传递 JMS 异常。

JMS 异常的错误代码为 JMSWMQ0028，此代码显示以下消息：

```
At least one asynchronous put message failed or gave a warning.
```

JMS 异常也具有提供更多详细信息的链接异常。 SENDCHECKCOUNT 属性的缺省值为 0，这表示未执行此类检查。

如果应用程序以客户机模式连接到队列管理器，需要快速连续发送一系列消息，但不需要队列管理器立即对每条发送的消息做出反馈，此优化最为有利。 但是，即使应用程序以绑定模式连接到队列管理器，应用程序也仍然可以利用此优化，但预期性能优势不会那么大。

使用 *IBM MQ classes for JMS* 的预读功能

使用 IBM MQ 提供的预读功能，可在应用程序请求在事务外部接收的非持久消息之前，将这些消息发送到 IBM MQ classes for JMS。 IBM MQ classes for JMS 会在内部缓冲区中存储消息，当应用程序请求这些消息时将其传递到应用程序。

使用 `MessageConsumers` 或 `MessageListeners` 从事务外部的目标接收消息的 IBM MQ classes for JMS 应用程序可以使用预读功能。 通过使用预读功能，使用这些对象的应用程序可在接收消息时获得性能提升。

使用 `MessageConsumers` 或 `MessageListeners` 的应用程序是否可使用预读功能取决于以下属性：

JMS 目标属性 READAHEADALLOWED（短名称 - RAALD）。
READAHEADALLOWED 控制 JMS 应用程序是否可在事务外部获取或浏览非持久消息时使用预读功能（如果 JMS 目标表示的底层队列或主题允许此选项）。

IBM MQ 队列或主题属性 DEFREADA（缺省预读）。
DEFREADA 指定在事务外部接收或浏览非持久消息的应用程序是否可使用预读功能。

下表显示 READAHEADALLOWED 和 DEFREADA 属性的可能值以及用于启用预读功能的值组合：

表 *46: READAHEADALLOWED* 和 *DEFREADA* 属性如何组合以确定在事务外部接收或浏览非持久消息时是否使用预读功能。

IBM MQ 队列属性	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST 或 AS_Q_DEF 或 AS_T_DEF**
DEFREADA = NO	已启用预读功能	未启用预读功能	未启用预读功能
DEFREADA = YES	已启用预读功能	未启用预读功能	已启用预读功能
DEFREADA = DISABLED	未启用预读功能	未启用预读功能	未启用预读功能

如果启用了预读功能，那么在应用程序创建 `MessageConsumer` 或 `MessageListener` 时，IBM MQ classes for JMS 会为 `MessageConsumer` 或 `MessageListener` 监视的目标创建内部缓冲区。 每个 `MessageConsumer` 或 `MessageListener` 都有一个内部缓冲区。 应用程序调用以下一种方法时，队列管理器会开始向 IBM MQ classes for JMS 发送非持久消息：

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

IBM MQ classes for JMS 通过应用程序执行的方法调用，向应用程序自动返回第一条消息。 IBM MQ classes for JMS 会将其他非持久消息存储在为目标创建的内部缓冲区中。 应用程序请求处理下一条消息时，IBM MQ classes for JMS 将在内部缓冲区中返回下一条消息。

内部缓冲区为空时，IBM MQ classes for JMS 会从队列管理器请求更多非持久消息。

当应用程序关闭 `MessageConsumer` 或与 `MessageListener` 关联的 JMS 会话时，将删除 IBM MQ classes for JMS 所使用的内部缓冲区。

对于 `MessageConsumers`，将丢失内部缓冲区中的任何未处理消息。

使用 `MessageListeners` 时，对内部缓冲区中消息的处理取决于 JMS 目标属性 READAHEADCLOSEPOLICY（短名称 - RACP）。 此属性的缺省值为 DELIVER_ALL，这意味着在将内部缓冲区中的所有消息传递到应用程序之前，不会关闭用于创建 `MessageListener` 的 JMS 会话。 如果属性设置为 DELIVER_CURRENT，那么在应用程序已处理当前消息后将会关闭 JMS 会话，且会丢弃内部缓冲区中的所有剩余消息。

IBM MQ classes for JMS 中的保留发布内容

可以将 IBM MQ classes for JMS 客户机配置为使用保留发布内容。

发布者可以指定必须保留发布内容的副本，以便在未来订户表示对该主题感兴趣时将该副本发送给订户。 在 IBM MQ classes for JMS 中，可通过将整数属性 JMS_IBM_RETAIN 设置为值 1 来完成此操作。 已在 com.ibm.msg.client.jms.JmsConstants 接口中为这些值定义了一些常量。 例如，如果已创建消息 *msg*，要将其设置为保留发布内容，请使用以下代码：

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

现在，可正常发送消息。 同时还可在接收的消息中查询 JMS_IBM_RETAIN。 因此，可以查询接收的消息是否为保留发布内容。

IBM MQ classes for JMS 中的 *XA* 支持

JMS 通过 JEE 容器中受支持的事务管理器支持绑定和客户机方式中符合 XA 的事务。

如果在应用程序服务器环境中需要 XA 功能，那么必须相应配置应用程序。 请参阅应用程序服务器自带的文档，以获取有关如何将应用程序配置为使用分布式事务的信息。

IBM MQ 队列管理器不能充当 JMS 的事务管理器。

使用 *JMS 2.0* 功能

JMS 2.0 将多个新功能领域引入了 IBM MQ classes for JMS。

为 IBM MQ 8.0 或更高版本开发 JMS 应用程序时，可能需要考虑此功能对队列管理器的影响。

相关概念

IBM MQ Java 语言接口

JMS 2.0 交付延迟

通过 JMS 2.0，您可以在发送消息时指定传送延迟。 只有在经过指定传送延迟之后，队列管理器才会传送消息。

应用程序可以在发送消息时使用 `MessageProducer.setDeliveryDelay(long deliveryDelay)` 或 `JMSProducer.setDeliveryDelay(long deliveryDelay)` 指定交付延迟（以毫秒为单位）。 该值将添加到发送消息的时间，并提供任何其他应用程序可获取该消息的最早时间。

在 IBM MQ 8.0 及更高版本中，通过使用单个内部登台队列来实施交付延迟。 会将具有非零交付延迟的消息放置在该队列上，该队列有一个头指示交付延迟和关于目标队列的信息。 称为交付延迟处理器的队列管理器组件将监视登台队列上的消息。 当消息的交付延迟完成后，会将消息从登台队列中取出放置在目标队列上。

消息传递客户机

交付延迟的 IBM MQ 实施仅在您使用 JMS 客户机时可用。 如果将交付延迟与 IBM MQ 一起使用，以下限制适用。 这些限制同样适用于 `MessageProducers` 和 `JMSProducers`，但在 `JMSProducers` 情况下会抛出 `JMSRuntimeExceptions`。

- 当连接到早于 IBM MQ 8.0 的队列管理器时，任何使用非零值调用 `MessageProducer.setDeliveryDelay` 的尝试都会生成具有 MQRC_FUNCTION_NOT_SUPPORTED 消息的 `JMSException` 。
- 对于具有非 MQBND_BIND_NOT_FIXED 的 **DEFBIND** 值的集群目标，不支持交付延迟。 如果 `MessageProducer` 设置了非零交付延迟，并且尝试发送不满足该需求的目标，那么调用将导致生成 `JMSException` 和 MQRC_OPTIONS_ERROR 消息。
- 将时间设置为小于先前指定的非零交付延迟的生存时间值（反之亦然）的任何尝试都将导致生成 `JMSException` 和 MQRC_EXPIRY_ERROR 消息。 根据所选的准确操作集，在调用 `setTimeToLive`、`setDeliveryDelay` 或 `send` 方法时执行该检查。
- 不支持使用保留出版物和交付延迟。 如果已使用 `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` 将消息标记为保留，那么尝试使用交付延迟来发布该消息将导致生成 `JMSException` 和 MQRC_OPTIONS_ERROR 消息。
- 不支持交付延迟和消息分组，任何使用该组合的尝试到将导致生成 JMSException 和 MQRC_OPTIONS_ERROR 消息。

任何使用交付延迟发送消息的失败情况都将导致客户机抛出 `JMSException` 和一条合适的错误消息，例如，队列已满。 在某些情况下，错误消息可能适用于目标位置和/或登台队列。

注: IBM MQ 允许将消息放入工作单元中的应用程序再次获得相同的消息，即使未落实工作单元。 该方法对交付延迟无效，因为在落实工作单元前消息未放置在登台队列上，因此不会将消息发送至目标位置。

Authorization

在应用程序使用非零交付延迟发送消息时，IBM MQ 会对原始目标执行授权检查。 如果未授权应用程序，那么发送将失败。 当队列管理器检测到消息的交付延迟完成时，会打开目标队列。 此时不会执行任何授权检查。

SYSTEM.DDELAY.LOCAL.QUEUE

系统队列 SYSTEM.DDELAY.LOCAL.QUEUE 用于实施交付延迟。

- Multi 在 多平台上， SYSTEM.DDELAY.LOCAL.QUEUE 存在。 必须更改系统队列，以使其 MAXMSGL 和 MAXDEPTH 属性足以满足预期负载。
- z/OS 在 IBM MQ for z/OS 上， SYSTEM.DDELAY.LOCAL.QUEUE 用作将带有传递延迟的消息发送到本地队列和共享队列的登台队列。 在 z/OS 上，必须创建和定义队列，使其 MAXMSGL 和 MAXDEPTH 属性足以满足预期负载。

在创建此队列时，必须对其进行保护，以使尽可能少的用户具有访问权。 必须仅出于维护和监视目的授予该队列访问权。

当通过 JMS 应用程序使用非零交付延迟发送消息时，会使用新消息标识将消息放入该队列。 会将原始消息标识放入消息的相关标识中。 该相关标识允许应用程序在需要时（例如，错误地使用了较大的交付延迟时）从登台队列中检索消息。

z/OS 的注意事项

z/OS

如果您的系统正在 z/OS 上运行，如果要使用交付延迟，还需考虑额外的注意事项。

如果要使用交付延迟，那么必须定义系统队列 SYSTEM.DDELAY.LOCAL.QUEUE。 必须使用足以满足预期负载的存储类并指定 INDXTYPE(NONE) 和 MSGDLVSQ(FIFO) 来进行定义。 将在 CSQ4INSG JCL 中提供并注释掉系统的样本定义。

共享队列

支持使用交付延迟将消息发送至共享队列。 但是，不管目标队列是否是共享队列，都只会使用单个专用的登台队列。 拥有该专用队列的队列管理器必须一直运行，以在延迟完成时将延迟的消息发送到其目标共享队列。

注: 如果使用交付延迟将非持久性消息放入共享队列，并且拥有登台队列的队列管理器已关闭，那么原始消息将丢失。 因此，与不使用交付延迟将非持久性消息发送到共享队列相比，使用交付延迟将非持久性消息发送至共享队列时，消息更可能丢失。

目标位置解析

如果将消息发送至队列，将解析两次；在登台队列中提取消息并将消息发送至目标队列时，JMS 应用程序会解析一次，队列管理器会解析一次。

在 JMS 应用程序调用发送方法时，会匹配出版物的目标预订。

如果根据队列定义使用持久性或优先级发送消息，那么会在第一次解析而非第二次解析时设置该值。

到期时间间隔

交付延迟将保留到期属性 MQMD.Expiry 的行为。 例如，如果使用到期时间间隔 20,000 毫秒和交付延迟 5,000 毫秒通过 JMS 应用程序放入消息，并在耗用时间 10,000 毫秒之后获取消息，那么 MQMD.expiry 字段的值可能是一秒的大约十分之五十。 此值指示从放入消息开始到获得消息为止已耗用 15 秒。

如果消息在位于登台队列上时到期，并且设置了一个 MQRO_EXPIRATION_* 选项，那么生成的报告将针对应用程序发送的原始消息，将移除用于包含交付延迟信息的头。

停止和启动交付延迟处理器

z/OS 在 z/OS 上，交付延迟处理器将集成到队列管理器 MSTR 地址空间中。 在启动队列管理器时，交付延迟处理器也将启动。 如果可登台队列可用，将打开该队列并等待消息到达以进行处理。 如果尚未定义登台队列、已禁用获取或者发生另一个错误，那么将关闭交付延迟处理器。 如果稍后对登台队列进行定义或更改以将其启用，那么交付延迟处理器将重新启动。 如果交付延迟处理器出于任何其他原因关闭，那么可

以通过将登台队列的 PUT 属性从 ENABLED 更改为 DISABLED、再改回 ENABLED 来将其重新启动。 如果因任何原因需要停止交付延迟处理器，请将登台队列的 PUT 属性设置为 DISABLED。

Multi 在 多平台 上，延迟处理器将以队列管理器开始，并且如果出现可恢复的故障，将自动重新启动。

无法放到目标队列

如果在延迟完成后无法将延迟的消息放到目标队列，那么将按照报告选项中的指示处理消息：将丢弃消息或者将消息发送至死信队列。 如果此操作失败，那么将在稍后尝试放置消息。 如果操作成功，并且请求报告，那么会生成异常报告并且会将该报告发送到指定的队列。 如果无法发送报告消息，那么会将报告消息发送至死信队列。 如果将报告发送到死信队列失败并且消息是持久性消息，那么将丢弃所有更改并且将在稍后回滚并重新交付原始消息。 如果消息是非持久性消息，那么将丢弃报告消息，但是会落实其他更改。 如果由于订户取消预订而导致无法交付延迟发布，或者对于非持久订户（由于已断开连接），将以静默方式丢弃消息。 仍将按照先前描述生成报告消息。

如果无法将延迟发布交付给订户，而选择将其放入死信队列，但放入死信队列失败，那么将丢弃消息。

为减少交付延迟完成后放入目标队列失败的可能性，队列管理器将在 JMS 客户机使用非零交付延迟发送消息时执行一些基本检查。 这些检查包括队列是否被禁用、消息是否大于允许的最大消息长度以及队列是否已满。

发布／预订

在 JMS 应用程序使用非零交付延迟发送消息时，会将发布与可用预订进行匹配。 会将每个匹配订户的消息放入 SYSTEM.DDELAY.LOCAL.QUEUE 队列，消息将保留在该队列中直至交付延迟完成。 如果其中一个订户是其他队列管理器的代理预订，那么在交付延迟完成后将在该队列管理器上进行扇出。 这可能会导致其他队列管理器上的订户收到原本在预订前发布的出版物。 这偏离了 JMS 2.0 规范。

如果使用 (N)PMSGDLV = ALLAVAIL 配置了目标主题，那么仅支持具有发布/预订的交付延迟。 尝试使用任何其他值都将导致 MQRC_PUBLICATION_FAILURE 错误。 如果在将消息放入目标队列时交付延迟处理器发生故障，结果如“无法放入目标队列”部分中所述。

报告消息

交付处理器将支持并操作所有报告选项，以下将被忽略但会在将消息发送至目标队列时随消息一起传递的选项除外：

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

克隆和共享的预订
在 IBM MQ 8.0 或更高版本中，可以使用两种方法提供对同一个预订的多使用者访问权。 这两种方法是：使用克隆的预订或使用共享的预订。

克隆的预订

克隆的预订是 IBM MQ 扩展。 克隆的预订使不同的 Java 虚拟机 (JVM) 中的多个使用者能够同时访问预订。 可以通过在 connectionFactory 对象上将 **CLONESUPP** 属性设置为 Enabled 来使用此行为。 缺省情况下，**CLONESUPP** 为 Disabled。 只可以在持久预订上启用克隆的预订。 如果启用 **CLONESUPP**，那么使用该 connectionFactory 建立的每个后续连接都将被克隆。

如果创建了一个或多个使用者以从某个持久预订处接收消息（即，创建时指定了同一个预订名称），那么可以将该预订视为克隆的预订。 只有创建使用者时所使用的连接在 MQConnectionFactory 上将 **CLONESUPP** 设置为 Enabled，才可以这样做。 在预订的主题上发布消息时，会将该消息的副本发送至预订。 消息对所有使用者都可用，但只有一个使用者会收到该消息。

注: 启用克隆的预订将扩展 JMS 规范。

共享的预订

JMS 2.0 规范介绍了共享的预订，允许在多个使用者中共享来自主题预订的消息。 只会将来自预订的每条消息发送给该预订上的一个使用者。 通过对 JMS 2.0 API 的相关调用来启用共享预订。

可以采用以下方法之一来调用 API：

- 通过 Java SE 应用程序（或 Java EE 客户机容器）。
- 通过 servlet 或 MDB 的实现。

JMS 2.0 规范未定义通过 sharedSubscription 推动 MDB 的任何标准方法，因此 IBM MQ 8.0 或更高版本将出于此目的提供 sharedSubscription 激活规范属性。 有关该属性的更多信息，请参阅第 378 页的『配置入站通信的资源适配器』和第 390 页的『如何定义 sharedSubscription 属性的示例』。

如果启用共享的预订，那么将无法取消共享。

可以将共享的预订创建为持久预订或非持久预订。 无需在正常 JMS 配置之外的队列管理器端上单独创建任何对象，将动态创建所需的任何对象。

决定使用共享的预订还是克隆的预订

当您确定使用共享的预订还是克隆的预订时，请考虑两者的优势。 如果可以，请使用共享的预订，因为它是规范定义的行为，而不是特定于 IBM MQ 的扩展。

下表包含一些在决定使用共享的预订还是克隆的预订时需考虑的要点：

表 *47:* 选择使用共享的预订还是克隆的预订时的注意事项

共享的预订	**克隆的预订**
共享的预订是 JMS 2.0 规范的标准部分。	克隆的预订是特定于 IBM MQ 的扩展。
通过使用显式 API 方法调用来创建共享的预订。	在 ConnectionFactory 级别以管理方式控制克隆的预订。
共享的预订可以是持久预订也可以是非持久预订。	克隆的预订只可以是持久预订。
共享的预订以单个预订为基础显式创建。	克隆的预订在启用了功能的连接下用于任何持久预订。
如果将预订创建为共享预订，那么稍后无法将其更改为非共享，反之亦然。	如果所拥有连接的 **CLONESUPP** 属性已更改，那么每次重新打开预订时可以将其从克隆更改为非克隆。

相关概念

订户和预订

预订耐久性

相关任务

使用 JMS 2.0 共享的预订

相关参考

第 390 页的『如何定义 sharedSubscription 属性的示例』

您可以在 WebSphere Liberty `server.xml` 文件中定义激活规范的 sharedSubscription 属性。 或者，您可以使用注释在消息驱动的 Bean (MDB) 中定义属性。

CLONESUPP

SupportMQExtensions 属性

JMS 2.0 规范介绍了对某些行为工作方式的更改。 IBM MQ 8.0 及更高版本包含属性 `com.ibm.mq.jms.SupportMQExtensions`，可以将该属性设置为 *TRUE* 以将这些更改的行为还原回先前的实现。

通过将 `SupportMQExtensions` 设置为 *True* 来还原三个功能区域：

消息优先级
可以将消息优先级指定为 *0* - *9*。 在 JMS 2.0 之前，消息还可以使用值 *-1*，指示使用队列的缺省优先级。 JMS 2.0 不允许将消息优先级设置为 *-1*。 启用 SupportMQExtensions 允许使用值 *-1*。

客户机标识
JMS 2.0 规范要求在非空客户机标识进行连接时检查其唯一性。 开启 SupportMQExtensions 意味着将忽略此要求并可以复用客户机标识。

NoLocal
JMS 2.0 规范要求，在开启该常量的情况下使用者无法接收由同一个客户机标识发布的消息。 在 JMS 2.0 之前，会在订户上设置此属性，以防止其接收自己的连接发布的消息。 启用 SupportMQExtensions 将此行为还原到其先前实现。

属性 com.ibm.mq.jms.SupportMQExtensions 是包含在 com.ibm.mqjms.jar 中的布尔属性。 可以将该属性设置为如下：

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

该属性可以在 **java** 命令上设置为标准 JVM 系统属性，或包含在 IBM MQ classes for JMS 配置文件中。

相关概念
第 77 页的『IBM MQ classes for JMS 配置文件』
IBM MQ classes for JMS 配置文件指定用于配置 IBM MQ classes for JMS 的属性。

相关参考
第 82 页的『用于配置 JMS 客户机行为的属性』
使用这些属性来配置 JMS 客户机的行为。

在 *JMS 2.0* 中使用共享预订
JMS 2.0 引入了 shared subscriptions 概念，在此概念中，单个预订在多个使用者之间共享，其中只有一个使用者在任何时间点接收发布内容。 IBM MQ classes for JMS。

为 IBM MQ 8.0 或更高版本开发 JMS 应用程序时，可能需要考虑此功能对队列管理器的影响。

“共享预订”背后的想法主要是在多个使用者之间共享负载。 也可以在多个使用者之间共享持久预订。

例如，假定存在：

- 预订 SUB，其将预订主题 FIFA2014/UPDATES 以接收足球赛事更新，并在三个使用者 C1、C2 和 C3 之间进行共享
- 生产者 P1，其将在 FIFA2014/UPDATES 主题上进行发布

在 FIFA2014/UPDATES 上进行发布时，三个使用者（C1、C2 或 C3）中只有一个使用者（而非全部）会收到该发布。

以下样本演示了如何使用共享预订，还演示了如何在 JMS 2.0 中使用 API Message.receiveBody() 来仅检索消息体。

此样本会创建三个订户线程（用于创建对 FIFA2014/UPDATES 主题的共享预订）和一个发布者线程。

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
```

```
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
```

```
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
* Demonstrate JMS 2.0 Simplified API using IBM MQ v91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

相关概念

IBM MQ Java 语言接口

IBM MQ classes for JMS 应用程序服务器工具

此主题描述 IBM MQ classes for JMS 如何实现 ConnectionConsumer 类和 Session 类中的高级功能。 同时还概述了服务器会话池的功能。

要点: 该信息仅供参考。 应用程序不得编写为使用此接口：在 IBM MQ 资源适配器中，使用它来连接到 Java EE 服务器。 有关实际连接信息，请参阅第 366 页的『使用 IBM MQ 资源适配器』。

IBM MQ classes for JMS 支持在 *Java Message Service Specification*（请参阅 Oracle Technology Network for Java Developers）中指定的应用程序服务器工具 (ASF)。 此规范确定了该编程模型中的三个角色：

- **JMS 提供程序**提供 ConnectionConsumer 和高级 Session 功能。
- **应用程序服务器**提供 ServerSessionPool 和 ServerSession 功能。
- **客户机应用程序**使用 JMS 提供程序和应用程序服务器提供的功能。

如果应用程序使用代理的实时连接，那么本主题中的信息不适用。

JMS ConnectionConsumer

ConnectionConsumer 接口提供用于将消息同时传递到线程池的高性能方法。

JMS 规范使应用程序服务器能够使用 `ConnectionConsumer` 接口与 JMS 实现紧密集成。 此功能部件提供对消息的并行处理。 通常，应用程序服务器会创建线程池，JMS 实现则使消息可供这些线程使用。 JMS 感知的应用程序服务器（例如，WebSphere Application Server）可使用此功能部件来提供高级消息传递功能，例如，消息驱动的 Bean。

普通应用程序不会使用 ConnectionConsumer，而专业版 JMS 客户机则可能会使用。 对于此类客户机，ConnectionConsumer 提供用于将消息同时传递到线程池的高性能方法。 在消息到达队列或主题时，JMS 会从池中选择线程，并向其传递一批消息。 为执行此操作，JMS 会运行关联 MessageListener 的 `onMessage()` 方法。

您可以通过构造多个 Session 和 MessageConsumer 对象（每个对象带有已注册的 MessageListener）实现相同效果。 但是，ConnectionConsumer 的性能更佳，使用的资源更少，且更为灵活。 尤其是，需要的 Session 对象更少。

使用 *ASF* 规划应用程序

本节指导您如何规划应用程序，包括：

- 第 270 页的『使用 ASF 执行点到点消息传递的常规准则』
- 第 270 页的『使用 ASF 执行发布/预订消息传递的常规准则』
- 第 271 页的『在 ASF 中从队列移除消息』
- 在 ASF 中处理有害消息。 请参阅第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』。

使用 *ASF* 执行点到点消息传递的常规准则

使用本主题以获取有关使用 ASF 执行点到点消息传递的常规信息。

应用程序从 QueueConnection 对象创建 ConnectionConsumer 时，会指定 JMS 队列对象和选择器字符串。然后，ConnectionConsumer 会开始为关联 ServerSessionPool 中的会话提供消息。 消息到达队列，如果它们与选择器匹配，那么会将其传递到关联 ServerSessionPool 中的会话。

在 IBM MQ 术语中，队列对象是指本地队列管理器上的 QLOCAL 或 QALIAS。 如果是 QALIAS，那么 QALIAS 必须引用 QLOCAL。 完全解析的 IBM MQ QLOCAL 称为底层 *QLOCAL*。 如果 ConnectionConsumer 未关闭且启动了其父 QueueConnection，可以将其视为处于活动状态。

可以针对相同底层 QLOCAL 运行多个 ConnectionConsumer（每个都带有不同的选择器）。 为了保持性能，不得在队列上累积不需要的消息。 不需要的消息是其活动 ConnectionConsumer 无匹配选择器的消息。 可以设置 QueueConnectionFactory，以便从队列移除这些不需要的消息（有关详细信息，请参阅第 271 页的『在 ASF 中从队列移除消息』）。 您可以通过以下两种方法之一设置此行为：

- 使用 JMS 管理工具将 QueueConnectionFactory 设置为 MRET(NO)。
- 在程序中，使用：

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

如果未更改此设置，缺省情况下会在队列上保留此类不需要的消息。

设置 IBM MQ 队列管理器时，请考虑以下几点：

- 必须为共享输入启用底层 QLOCAL。 要执行此操作，请使用以下 MQSC 命令：

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- 队列管理器必须具有启用的死信队列。 如果 ConnectionConsumer 在将消息放入死信队列时遇到问题，那么会停止从底层 QLOCAL 传递消息。 要定义死信队列，请使用：

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- 运行 ConnectionConsumer 的用户必须有权使用 MQOO_SAVE_ALL_CONTEXT 和 MQOO_PASS_ALL_CONTEXT 执行 MQOPEN。 有关详细信息，请参阅适用于特定平台的 IBM MQ 文档。
- 如果队列上保留了不需要的消息，那么会降低系统性能。 因此，应规划消息选择器，以便在这些消息选择器之间，ConnectionConsumer 可从队列中移除所有消息。

有关 MQSC 命令的详细信息，请参阅 MQSC 命令。

使用 *ASF* 执行发布/预订消息传递的常规准则

ConnectionConsumer 接收指定 Topic 的消息。 ConnectionConsumer 可以是持久或非持久的。 必须指定 ConnectionConsumer 使用哪个队列或哪些队列。

应用程序从 TopicConnection 对象创建 ConnectionConsumer 时，会指定 Topic 对象和选择器字符串。 然后，ConnectionConsumer 会开始接收此 Topic 上与选择器匹配的消息，包括预订的主题的任何保留发布内容。

或者，应用程序可创建与特定名称关联的持久 ConnectionConsumer。 此 ConnectionConsumer 接收自持久 ConnectionConsumer 上次活动以来在 Topic 上发布的消息。 它会接收 Topic 上与选择器匹配的所有此类消息。 但是，如果 ConnectionConsumer 使用预读，那么会在关闭时丢失客户机缓冲区中的非持久消息。

如果 IBM MQ classes for JMS 处于 IBM MQ 消息传递提供程序迁移方式，那么会针对非持久 ConnectionConsumer 预订使用单独队列。TopicConnectionFactory 上的 CCSUB 可配置选项指定要使用的队列。 通常，CCSUB 指定单个队列，以供使用相同 TopicConnectionFactory 的所有 ConnectionConsumer 使用。 然而，可以通过指定队列名称前缀后跟一个星号 (*)，使每个 ConnectionConsumer 都生成一个临时队列。

如果 IBM MQ classes for JMS 处于 IBM MQ 消息传递提供程序迁移方式，那么 Topic 的 CCDSUB 属性指定要用于持久预订的队列。 同样，这可能是已存在的队列或后跟星号 (*) 的队列名称前缀。如果指定了已存在的队列，那么预订该主题的所有持久连接使用者都将使用此队列。 如果指定队列名称前缀后跟一个星号

(*)，那么首次使用此特定名称创建持久 ConnectionConsumer 时，会生成一个队列。 稍后会在使用相同名称创建持久 ConnectionConsumer 时复用此队列。

设置 IBM MQ 队列管理器时，请考虑以下几点：

- 队列管理器必须具有启用的死信队列。 如果 ConnectionConsumer 在将消息放入死信队列时遇到问题，那么会停止从底层 QLOCAL 传递消息。 要定义死信队列，请使用：

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- 运行 ConnectionConsumer 的用户必须有权使用 MQOO_SAVE_ALL_CONTEXT 和 MQOO_PASS_ALL_CONTEXT 执行 MQOPEN。 有关详细信息，请参阅适用于您平台的 IBM MQ 文档。
- 您可以通过为单个 ConnectionConsumer 创建单独的专用队列来优化其性能。 但是，这会耗用更多资源。

在 *ASF* 中从队列移除消息
当应用程序使用 ConnectionConsumer 时，在多种情况下，JMS 都可能需要从队列移除消息。

这些情况如下所示：

消息格式错误
可能收到 JMS 无法解析的消息。

有害消息
消息可能达到回退阈值，但 ConnectionConsumer 未能在回退队列上对其重新排队。

无相关 ConnectionConsumer
对于点到点消息传递，当设置了 QueueConnectionFactory 以便其不保留不需要的消息时，任何 ConnectionConsumer 都不需要的消息会到达。

在这些情况下，ConnectionConsumer 会尝试从队列移除消息。 消息的 MQMD 的报告字段中的处置选项用于设置准确的行为。 这些选项是：

MQRO_DEAD_LETTER_Q
消息将重新排队到队列管理器的死信队列。 这是缺省值。

MQRO_DISCARD_MSG
已丢弃消息。

ConnectionConsumer 也会生成报告消息，这也取决于消息的 MQMD 的报告字段。 此消息将发送到 ReplyToQmgr 上消息的 ReplyToQ。 如果发送报告消息时发生错误，那么会改为向死信队列发送消息。 消息的 MQMD 的报告字段中的异常报告选项用于设置报告消息的详细信息。 这些选项是：

MQRO_EXCEPTION
将生成包含原始消息的 MQMD 的报告消息。 此消息不包含任何消息体数据。

MQRO_EXCEPTION_WITH_DATA
将生成包含 MQMD、任何 MQ 头以及 100 字节主体数据的报告消息。

MQRO_EXCEPTION_WITH_FULL_DATA
将生成包含原始消息中所有数据的报告消息。

缺省
不会生成任何报告消息。

生成报告消息时，将使用以下选项：

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

如果无法对有害消息进行重新排队（可能是因为死信队列已满或者错误指定了授权），那么所产生的影响取决于消息的持久性。 如果消息是非持久消息，那么会丢弃消息，并且不会生成任何报告消息。 如果消息是持久消息，那么会停止将消息传递到侦听此目标的所有连接使用者。 必须关闭此类连接使用者并解决问题，才可重新创建这些使用者并重新启动消息传递。

请务必定义死信队列，并定期执行检查以确保不会发生问题。 尤其是，确保死信队列不会达到其最大深度，且其最大消息大小足够大，适用于所有消息。

将消息重新排队到死信队列时，会在消息前添加 IBM MQ 死信消息头 (MQDLH)。 请参阅 MQDLH - 死信消息头以获取有关 MQDLH 格式的详细信息。 您可以通过以下字段识别 ConnectionConsumer 已放置到死信队列上的消息，或 ConnectionConsumer 已生成的报告消息：

- PutApplType 是 `MQAT_JAVA （0x1C）`
- PutApplName 是“`MQ JMS ConnectionConsumer`”

这些字段位于死信队列上消息的 MQDLH 中以及报告消息的 MQMD 中。 MQMD 的反馈字段和 MQDLH 的原因字段包含描述错误的代码。 有关这些代码的详细信息，请参阅第 273 页的『ASF 中的原因码和反馈代码』。 其他字段在 MQDLH - 死信消息头中进行了描述。

在 *ASF* 中处理有害消息

在应用程序服务器工具中，处理有害消息的方式与在 IBM MQ classes for JMS 中其他位置处理的方式稍有不同。

有关在 IBM MQ classes for JMS 中处理有害消息的信息，请参阅第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』。

使用应用程序服务器工具 (ASF) 时，ConnectionConsumer 而不是 MessageConsumer 会处理有害消息。 ConnectionConsumer 会根据队列的 BackoutThreshold 和 BackoutRequeueQName 属性对消息重新排队。

应用程序使用 ConnectionConsumer 时，回退消息的环境取决于应用程序服务器提供的会话。

- 会话为具有 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE 的非事务性会话时，仅会在发生系统错误后或应用程序意外终止的情况下回退消息。
- 会话为具有 CLIENT_ACKNOWLEDGE 的非事务性会话时，调用 Session.recover() 的应用程序服务器可回退未确认的消息。

  通常，MessageListener 的客户机实现或应用程序服务器将调用 Message.acknowledge()。 Message.acknowledge() 确认到目前为止在会话上传递的所有消息。
- 会话为事务性会话时，调用 Session.rollback() 的应用程序服务器可回退未确认的消息。
- 如果应用程序服务器提供 XASession，那么会根据分布式事务落实或回退消息。 应用程序服务器负责完成事务。

相关概念

第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』
有害消息是指接收应用程序无法处理的消息。 如果有害消息已传递到应用程序并且已回滚指定次数，那么 IBM MQ classes for JMS 可将其移到回退队列。

错误处理

本章节介绍了错误处理的各个方面，包括第 272 页的『在 ASF 中从错误情况恢复』和第 273 页的『ASF 中的原因码和反馈代码』。

在 *ASF* 中从错误情况恢复

如果 ConnectionConsumer 遇到严重错误，那么会停止将消息传递到与相同 QLOCAL 相关的所有 ConnectionConsumer。 发生此情况时，会通知向受影响连接注册的任何 ExceptionListener。 应用程序可以通过两种方式从这些错误情况恢复。

通常，如果 ConnectionConsumer 无法将消息重新排队到死信队列，或者从 QLOCAL 读取消息时遇到错误，那么会发生此类严重错误。

由于会通知向受影响连接注册的任何 ExceptionListener，因此可使用它们来识别问题的原因。 在某些情况下，系统管理员必须介入以解决该问题。

使用以下一种方法从这些错误情况恢复：

- 针对所有受影响 ConnectionConsumer 调用 `close()`。 应用程序仅会在关闭所有受影响 ConnectionConsumer 并解决所有系统问题后，才可创建新的 ConnectionConsumer。

- 针对所有受影响连接调用 `stop()`。 所有连接已停止并解决所有系统问题后，应用程序可对其连接成功执行 `start()` 操作。

ASF 中的原因码和反馈代码
使用原因码和反馈代码来确定错误的原因。 此处提供了 ConnectionConsumer 生成的常见原因码。

要确定错误的原因，请使用以下信息：

- 任何报告消息中的反馈代码
- 死信队列中任何消息的 MQDLH 中的原因码

ConnectionConsumer 生成以下原因码。

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

原因
消息已达到在 QLOCAL 上定义的回退阈值，但未定义回退队列。

在无法定义回退队列的平台上，消息已达到 JMS 定义的回退阈值 20。

操作
如果不需要此项，请定义相关 QLOCAL 的回退队列。 同时还需了解多次回退的原因。

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

原因
在点到点消息传递中，具有一条消息与监视队列的 ConnectionConsumer 的任一选择器都不匹配。 为保持性能，需将消息重新排队到死信队列。

操作
要避免出现此情况，请确保使用队列的 ConnectionConsumer 提供一组用于处理所有消息的选择器，或者设置 QueueConnectionFactory 以保留消息。

或者，调查消息的原因。

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

原因
JMS 无法解释队列上的消息。

操作
调查消息的起因。 JMS 通常会将非预期格式的消息作为 BytesMessage 或 TextMessage 传递。 有时，消息格式严重错误时，这会失败。

这些字段中显示的其他代码是尝试将消息重新排队到回退队列失败所导致的。 在此情况下，代码描述了重新排队失败的原因。 要诊断这些错误的原因，请参阅 API 完成代码和原因码。

如果无法将报告消息放置到 ReplyToQ 上，那么会将其放置到死信队列上。 在此情况下，会按本主题中所述填写 MQMD 的反馈字段。 MQDLH 中的原因字段说明了无法将报告消息放置到 ReplyToQ 上的原因。

AFS 中服务器会话池的功能
本主题概述了服务器会话池的功能。

第 274 页的图 51 概述了 ServerSessionPool 和 ServerSession 功能的原理。

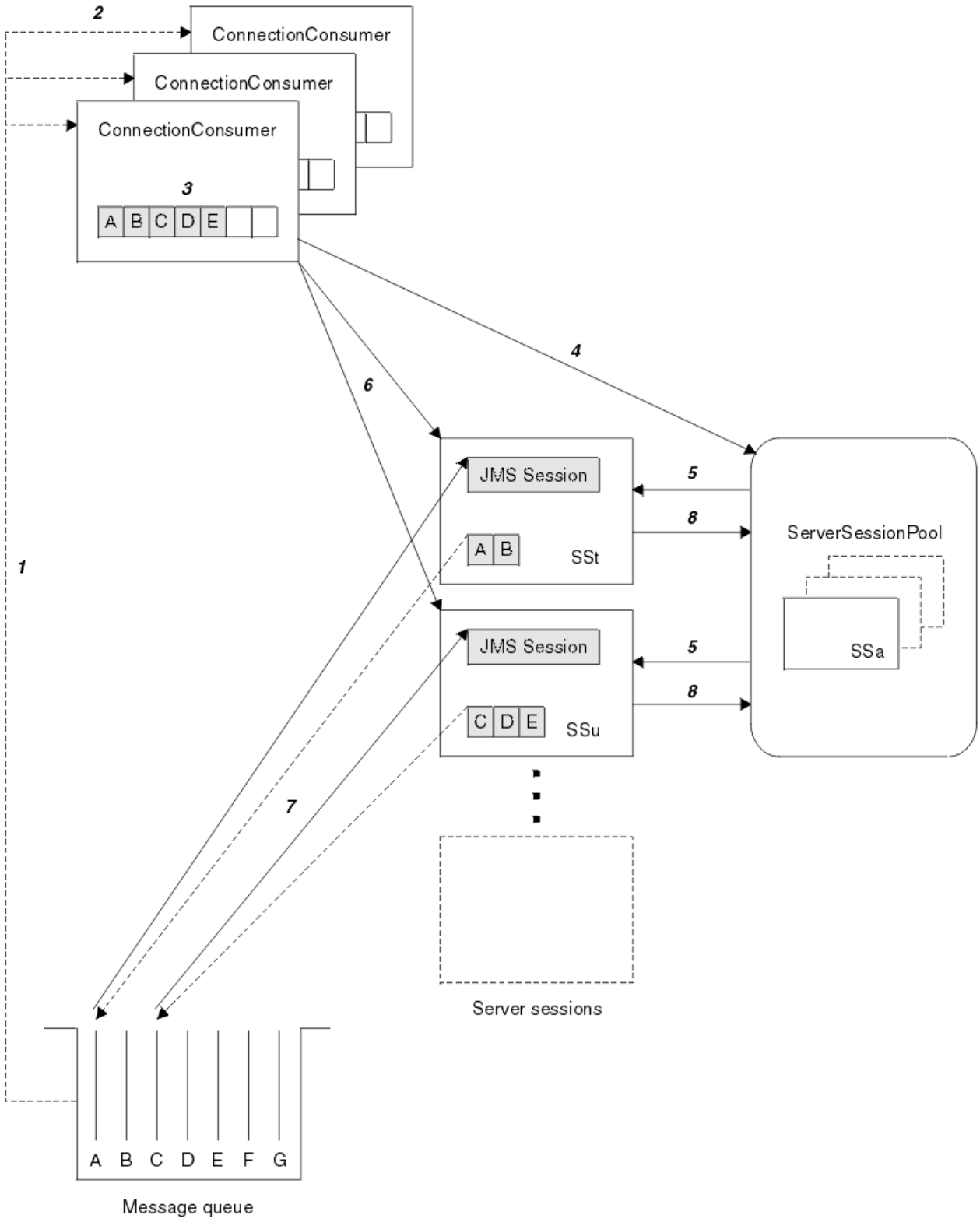


图 *51: ServerSessionPool* 和 *ServerSession* 功能

1. ConnectionConsumer 从队列取出消息引用。
2. 每个 ConnectionConsumer 选择特定消息引用。
3. ConnectionConsumer 缓冲区保存所选消息引用。
4. ConnectionConsumer 从 ServerSessionPool 请求一个或多个 ServerSession。

5. 从 ServerSessionPool 分配 ServerSession。
6. ConnectionConsumer 向 ServerSession 分配消息引用，启动 ServerSession 线程并使这些线程处于运行状态。
7. 每个 ServerSession 从队列检索其引用的消息。 它会将它们从与 JMS 会话关联的 MessageListener 传递到 onMessage 方法。
8. 完成其处理后，会向池返回 ServerSession。

通常，应用程序服务器会提供 ServerSessionPool 和 ServerSession 功能。

z/OS 在 CICS OSGi JVM 服务器中使用 IBM MQ classes for JMS

通过使用 IBM MQ classes for JMS 为在 CICS Open Services Gateway Initiative (OSGi) Server 环境中运行的应用程序提供基于标准的消息传递支持。

注意: 检查企业所使用的 CICS 系统的系统需求。 请参阅 CICSTransaction Server 的详细系统需求，以获取详细信息。

从 IBM MQ 8.0 开始， IBM MQ 支持在 CICS Open Services Gateway Initiative (OSGi) Java 虚拟机 (JVM) 服务器的特定版本中使用 IBM MQ classes for JMS 。

本主题是有关如何在 JVM 服务器环境中设置 IBM MQ classes for JMS 的简介。

请参阅 CICS 文档中的 在 OSGi JVM 服务器中使用 IBM MQ classes for JMS ，以获取有关设置和配置系统的详细信息。

常规限制

在 CICS OSGi JVM 服务器中使用 IBM MQ classes for JMS 时，以下限制适用:

- 不支持客户机方式连接。
- 仅对 IBM WebSphere MQ 7.1 或 IBM MQ 8.0 或更高版本的队列管理器支持连接。 连接工厂上的 **PROVIDERVERSION** 属性必须未指定或者值大于或等于 7。
- 不支持使用任何 XA 连接工厂（例如，`com.ibm.mq.jms.MQXAConnectionFactory`）。
- 仅在 CICS 5.2 或更高版本中支持在 CICS OSGi JVM 服务器中使用 IBM MQ classes for JMS 。 如果正在使用 CICS 5.2，那么必须应用 APAR PI32151。

V 9.1.0 在 IBM MQ 9.1.0 之前 (如果您是 Long Term Support 用户) 或 IBM MQ 9.0.1 之前 (如果您是 Continuous Delivery 用户) ，不支持在 Liberty JVM 服务器环境中使用 IBM MQ classes for JMS 。

相关任务

配置 JMS **PROVIDERVERSION** 属性

V 9.1.0 z/OS 在 CICS Liberty JVM 服务器中使用 IBM MQ classes for JMS

从 IBM MQ 9.1.0 开始，在 CICS Liberty JVM 服务器中运行的 Java 程序可以使用 IBM MQ classes for JMS 来访问 IBM MQ。

您必须使用 IBM MQ 9.1.0 版本的 IBM MQ 资源适配器。 可以从 Fix Central 中获取该资源适配器（请参阅第 373 页的『在 Liberty 中安装资源适配器』）。

有两种 Liberty Profile JVM 在 CICS 5.3 和更高版本中可用，对 IBM MQ 可用的连接类型限制如下:

CICS Liberty 标准

- IBM MQ 资源适配器可以在 CLIENT 方式下连接到任何服务中的 IBM MQ 版本
- 当没有 CICS 连接 (活动 CICS MQCONN 资源定义) 到同一 CICS 区域中的同一队列管理器时， IBM MQ 资源适配器可以在 BINDINGS 方式下连接到 IBM MQ for z/OS 的任何服务版本。

CICS Liberty 集成

- IBM MQ 资源适配器可以在 CLIENT 方式下连接到任何服务中的 IBM MQ 版本。

- 不支持 BINDINGS 方式连接。

有关设置和配置系统的详细信息，请参阅 CICS 文档中的 在 Liberty JVM 服务器中使用 IBM MQ classes for JMS 。

在 IMS 中使用 IBM MQ classes for JMS

IMS 13 环境中基于标准的消息传递支持是通过使用 IBM MQ classes for JMS 提供的。

查看您企业使用的 IMS 系统的系统需求。 有关更多信息，请参阅 IMS 13 的常规规划信息

从 IBM MQ 8.0.0 Fix Pack 4 开始， IBM MQ 支持在 IMS V13 和更高版本中使用 IBM MQ classes for JMS 。

这组主题描述了如何在 IMS 环境中设置 IBM MQ classes for JMS ， 以及在使用经典 (JMS 1.1) 和简化 (JMS 2.0) 接口时适用的 API 限制。 请参阅第 280 页的『JMS API 限制』以获取 API 特定信息的列表。

注: 类似的限制适用于传统的 (JMS 1.0.2) 特定于域的接口，但未在此处专门描述。

支持的 IMS 从属区域

支持以下从属区域类型:

- MPR
- BMP
- IFP
- JMP（仅限 31 位 Java 虚拟机 (JVM)，不支持 64 位 JVM）
- JBP（仅限 31 位 JVM，不支持 64 位 JVM）

除非在下列主题中特别地提及，IBM MQ classes for JMS 在所有区域类型中的行为是相同的。

支持的 Java 虚拟机

IBM MQ classes for JMS 要求使用 Java Platform, Standard Edition 7 (Java SE 7) 或更高版本。

其他限制

在 IMS 环境中使用 IBM MQ classes for JMS 时，存在以下限制:

- 不支持客户机方式连接。
- 仅支持连接到使用 IBM MQ 消息传递提供程序 `Normal` 或 `Version 8` 方式的 IBM MQ 8.0 队列管理器。

  连接工厂上的 **PROVIDERVERSION** 属性必须为 unspecified 或大于或等于 7 的值。
- 不支持使用任何 XA 连接工厂（例如，`com.ibm.mq.jms.MQXAConnectionFactory`）。

相关任务

将 IBM MQ 定义到 IMS

设置 *IMS* 适配器以用于 *IBM MQ classes for JMS*

IBM MQ classes for JMS 利用其他编程语言所使用的相同的 IBM MQ-IMS 适配器。 此适配器使用 IMS External Subsystem Attach Facility (ESAF)。

开始之前

完成以下过程之前，您必须为相关队列管理器、IMS 控制和从属区域配置 IMS 适配器，如设置 IMS 适配器中所述。

注意: 您不需要执行用于描述构建动态存根的步骤，除非您需要动态存根用于其他目的。

如果已配置 IMS 适配器，请执行以下过程。

过程

1. 更新 IMS PROCLIB 成员中的 LIBPATH 变量，从属区域 JCL（例如 DFSJVMEV）中的 ENVIRON 参数引用了该变量，这样，它可以包含 IBM MQ classes for JMS 本机库。

   即，包含 `libmqjims.so` 的 zFS 目录。 例如， DFSJVMEV 可能看起来如下所示，其中最后一行是包含 IBM MQ classes for JMS 本机库的目录：

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

2. 通过更新 `java.class.path` 选项，将 IBM MQ classes for JMS 添加到 IMS 从属区域所使用的 JVM 的类路径中。

   通过遵循 IMS PROCLIB 数据集的 DFSJVMMS 成员中的指示信息执行此操作。

   例如，您可以使用以下内容，其中粗体行指示更新：

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

   注: 虽然在包含 IBM MQ classes for JMS 的目录中提供了许多不同的 jar 文件，但您只需要 `com.ibm.mq.allclient.jar` 文件。

3. 停止并重新启动将使用 IBM MQ classes for JMS 的任何 IMS 从属区域。

下一步做什么

创建和配置连接工厂和目标。

可以使用三种方法来实例化连接工厂和目标的 IBM MQ 实现。 请参阅第 162 页的『在 IBM MQ classes for JMS 应用程序中创建并配置连接工厂和目标』，以了解详细信息。

请注意，这三个方法在 IMS 环境中都有效。

相关任务

设置 IMS 适配器

将 IBM MQ 定义到 IMS

事务行为

IBM MQ classes for JMS 在 IMS 环境中发送和接收的消息将始终与当前任务上活动的 IMS 工作单元 (UOW) 相关。

只能通过在 `com.ibm.ims.dli.tm.Transaction` 对象实例上调用落实或回滚方法，或者通过正常结束的 IMS 任务（在此情况下会隐式落实工作单元）来完成该工作单元。 如果 IMS 任务异常结束，那么该工作单元将回滚。

因此，在调用任何 `Connection.createSession` 或 `ConnectionFactory.createContext` 方法时，将忽略 **transacted** 和 **acknowledgeMode** 参数的值。 另外，不支持以下方法。 调用以下任何方法将在会话情况中导致 `IllegalStateException`：

- `javax.jms.Session.commit()`
- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

以及 JMS 上下文案例中的 `IllegalStateRuntimeSession`：

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

此行为存在一种例外情况。 如果使用以下某种机制来创建会话或 JMS 上下文：

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

那么该会话或 JMS 上下文的行为如下所示：

- 任何已发送消息都是在 IMS 工作单元外部发送。 即，这些消息将在目标位置上立即可用，或者当提供的交付延迟时间间隔结束时在目标位置上可用。
- 将在 IMS UOW 外部接收任何非持久消息，除非已在创建会话或 JMS 上下文的连接工厂上指定了 SYNCPOINT-tallgets 属性。
- 持久消息将始终在 IMS 工作单元内部接收。

此行为可能非常有用，例如，即使工作单元回滚您也想将审计消息写入到队列时。

IMS 同步点的影响

IBM MQ classes for JMS 构建基于利用 ESAF 的现有 IBM MQ 适配器支持。 这意味着记录的行为适用，包括同步点出现时由 IMS 适配器关闭的处于打开状态的句柄。

请参阅 第 57 页的『IMS 应用程序中的同步点』 以获取更多信息。

要解释这一点，请考虑在 JMP 环境运行的以下代码。 对 `mp.send()` 的第二次调用将生成 `JMSException`，因为 `messageQueue.getUnique(inputMessage)` 代码将导致所有打开的 IBM MQ 连接和对象句柄关闭。

如果 `getUnique()` 调用被 `Transaction.commit()` 替换，将观察到类似行为，但是使用 `Transaction.rollback()` 时不会观察到类似行为。

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

此方案中使用的正确代码如下所示。 在此案例中，IBM MQ 的连接在调用 `getUnique()` 之前关闭。 然后，将重新创建连接和会话以发送其他消息。

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);
```

```
//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

使用 *IMS* 适配器的注意事项

您需要注意到以下限制。 对于每个队列管理器，您只可以有一个连接句柄。 使用 JMS 和本机代码时，与 IBM MQ 的交互会产生影响。 对连接认证和授权也有一些限制。

每个队列管理器一个连接句柄

IMS 从属区域中对特定队列管理器一次仅允许使用一个连接句柄。 针对同一队列管理器的任何后续连接尝试都复用现有的句柄。

虽然此行为不应该在仅使用 IBM MQ classes for JMS 的应用程序中导致任何问题，但当在以语言 (例如 COBOL 或 C) 编写的本机代码中同时使用 IBM MQ classes for JMS 和 MQI 时，此行为会在与 IBM MQ 交互的应用程序中导致问题。

使用 JMS 和本机代码时与 IBM MQ 交互的含义

如果将使用 IBM MQ 功能的 Java 代码和本机代码交错以及如果指向 IBM MQ 的连接在离开本机或 Java 代码前未关闭，那么可能会出现问题。

例如，在以下伪代码中，队列管理器的连接句柄起初使用 IBM MQ classes for JMS 在 Java 代码中建立。 连接句柄在 COBOL 代码中重复使用并且通过调用 MQDISC 来使其失效。

下次 IBM MQ classes for JMS 使用连接句柄时，会生成原因码为 MQRC_HCONN_ERROR 的 `JMSException`。

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
       Get session from static variable
       Create a message consumer - fails as connection handle invalidated
```

存在其他类似使用模式，可以导致 MQRC_HCONN_ERROR。

虽然通常可以在本机和 Java 代码之间共享 IBM MQ 连接句柄（例如，如果没有 MQDISC 调用，先前示例将生效），最佳做法是在从 Java 更改为本机代码或执行相反的转换之前关闭任何连接句柄。

连接认证和授权

JMS 规范允许在创建连接或 JMS 上下文对象时，指定用户名和密码以进行认证和授权。

IMS 环境不支持此操作。 指定用户名和密码时尝试创建连接将导致抛出 JMS Exception。 指定用户名和密码时，尝试创建 JMS 上下文，将导致抛出 JMSRuntimeException。

相反，必须使用从 IMS 环境连接到 IBM MQ 时用于认证和授权的现有机制。

有关更多信息，请参阅在 z/OS 上设置安全性。 特别地，请参阅用于安全性检查的用户标识，其中描述了可以使用的用户标识。

相关任务

在 z/OS 上设置安全性

JMS API 限制

从 JMS 规范角度，IBM MQ classes for JMS 将 IMS 视为符合 Java EE 的应用程序服务器，总是存在进行中的 JTA 事务。

例如，您永远不可能在 IMS 中调用 javax.jms.Session.commit()，因为 JMS 规范规定，当 JTA 事务正在进行时，您不可以在 JEE EJB 或 Web 容器中调用它。

除了 第 277 页的『事务行为』中描述的限制外，这还会导致对 JMS API 的以下限制。

经典 API 限制

- javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int) 始终抛出 JMSException。
- javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) 始终抛出 JMSException。
- 如果连接现有一个活动的会话，javax.jms.Connection.createSession 的所有三个变体总是抛出 JMSException。
- javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) 始终抛出 JMSException。
- javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) 始终抛出 JMSException。
- javax.jms.Connection.setClientID() 始终抛出 JMSException。
- javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener) 始终抛出 JMSException。
- javax.jms.Connection.stop() 始终抛出 JMSException。
- javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener) 始终抛出 JMSException。
- javax.jms.MessageConsumer.getMessageListener() 始终抛出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener) 始终抛出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener) 始终抛出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener) 始终抛出 JMSException。
- javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener) 始终抛出 JMSException。
- javax.jms.Session.run() 始终抛出 JMSRuntimeException。
- javax.jms.Session.setMessageListener(javax.jms.MessageListener) 始终抛出 JMSException。
- javax.jms.Session.getMessageListener() 始终抛出 JMSException。

简化的 API 限制

- javax.jms.JMSContext.createContext(int) 始终抛出 JMSRuntimeException。

- `javax.jms.JMSContext.setClientID(String)` 始终抛出 `JMSRuntimeException`。
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` 始终抛出 `JMSRuntimeException`。
- `javax.jms.JMSContext.stop()` 始终抛出 `JMSRuntimeException`。
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` 始终抛出 `JMSRuntimeException`。

使用 IBM MQ classes for Java

在 Java 环境中使用 IBM MQ 。 IBM MQ classes for Java 允许 Java 应用程序作为 IBM MQ 客户机连接到 IBM MQ，或直接连接到 IBM MQ 队列管理器。

注:

IBM 将不对 IBM MQ classes for Java 进行进一步的增强，功能稳定在 IBM MQ 8.0 随附的级别上。 使用 IBM MQ classes for Java 的现有应用程序继续完全受支持，但是将不会添加新功能部件，并且会拒绝针对增强功能的请求。 完全受支持意味着，在完成 IBM MQ 系统需求变化所需的任何更改的同时修复缺陷。

IBM MQ classes for Java 在 IMS 中不受支持。

IBM MQ classes for Java 在 WebSphere Liberty 中不受支持。 不得与 IBM MQ Liberty 消息传递功能或通用的 JCA 支持一起使用。 有关更多信息，请参阅在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口。

IBM MQ classes for Java 是两个备用 API 中的一个，Java 应用程序可以使用它来访问 IBM MQ 资源。 另一个 API 是 IBM MQ classes for JMS。

从 IBM MQ 8.0 开始，IBM MQ classes for Java 使用 Java 7 进行构建。

Java 7 运行时环境支持运行较早的类文件版本。

IBM MQ classes for Java 封装了消息队列接口 (MQI) (本机 IBM MQ API) ，并使用类似于 C++ 的对象模型以及与 IBM MQ 的 .NET 接口。

可编程选项允许 IBM MQ classes for Java 采用以下任何方式连接到 IBM MQ:

- 在客户机方式下，通过使用传输控制协议/因特网协议 (TCP/IP) 作为 IBM MQ MQI client
- 在 绑定方式中，使用 Java 本机接口 (JNI) 直接连接到 IBM MQ

注: IBM MQ classes for Java 不支持自动客户机重新连接。

客户机方式连接

通过使用客户机方式，IBM MQ classes for Java 应用程序可以连接到任何受支持的队列管理器。

要在客户机方式下连接到队列管理器，IBM MQ classes for Java 应用程序可以在运行队列管理器的同一系统或不同系统上运行。 在每种情况下，IBM MQ classes for Java 均通过 TCP/IP 连接到队列管理器。

有关如何编写应用程序以使用客户机方式连接的更多信息，请参阅 第 301 页的『IBM MQ classes for Java 连接方式』。

绑定方式连接

在绑定方式下使用时，IBM MQ classes for Java 会使用 Java 本机接口 (JNI) 直接调用现有的队列管理器 API，而不是通过网络进行通信。 在大多数环境中，与在客户机方式下连接相比，在绑定方式下连接通过避免 TCP/IP 通信成本，为 IBM MQ classes for Java 应用程序提供了更佳的性能。

绑定方式下使用 IBM MQ classes for Java 进行连接的应用程序，必须与其连接的队列管理器在同一系统上运行。

必须将用于运行 IBM MQ classes for Java 应用程序的 Java 运行时环境配置为装入 IBM MQ classes for Java 库; 请参阅 第 289 页的『IBM MQ classes for Java 库』 以获取更多信息。

有关如何编写应用程序以使用绑定方式连接的更多信息，请参阅 第 301 页的『IBM MQ classes for Java 连接方式』。

相关概念

IBM MQ Java 语言接口

第 67 页的『使用 IBM MQ classes for JMS』

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

相关任务

跟踪 IBM MQ classes for Java 应用程序

Java 和 JMS 故障诊断

为什么应该使用 IBM MQ classes for Java?

Java 应用程序可以使用 IBM MQ classes for Java 或 IBM MQ classes for JMS 来访问 IBM MQ 资源。

注: 虽然继续完全支持那些使用 IBM MQ classes for Java 的现有应用程序，但新应用程序应该使用 IBM MQ classes for JMS。 最近添加到 IBM MQ 的功能（例如异步使用和自动重新连接）在 IBM MQ classes for Java 中不可用，但在 IBM MQ classes for JMS 中可用。 有关更多信息，请参阅第 68 页的『为什么应该使用 IBM MQ classes for JMS? 』。

注: IBM MQ classes for Java 功能稳定在 IBM MQ 8.0 随附的级别上。 使用 IBM MQ classes for Java 的现有应用程序仍完全受支持，但此 API 是稳定的，因此将不会添加新的功能部件，并会拒绝针对增强功能的请求。 完全受支持意味着，在完成 IBM MQ 系统需求变化所需的任何更改的同时修复缺陷。

IBM MQ classes for Java 的先决条件

要使用 IBM MQ classes for Java，您需要使用某些其他软件产品。

有关 IBM MQ classes for Java 的先决条件的信息，请访问 IBM MQ 的系统需求 Web 页面。

要开发 IBM MQ classes for Java 应用程序，您需要 Java 开发工具包 (JDK)。 可以在 IBM MQ 的系统需求 信息中找到您的操作系统所支持的 JDK 的详细信息。

要运行 IBM MQ classes for Java 应用程序，您需要以下软件组件：

- IBM MQ 队列管理器，适用于连接到队列管理器的应用程序。
- Java 运行时环境 (JRE)，适用于运行应用程序的每个系统。 合适的 JRE 与 IBM MQ 一起提供。
- IBM i 对于 IBM i，需要 QShell，这是操作系统的选项 30。
- z/OS 对于 z/OS，需要 UNIX and Linux 系统服务 (USS)。

如果要求 TLS 连接使用已通过 FIPS 140-2 认证的加密模块，那么就需要 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS)。 每个 V1.4.2 或更高版本的 IBM JDK 和 JRE 都包含 IBMJSSEFIPS。

您可以在操作系统上的 Java 虚拟机 (JVM) 和 TCP/IP 实现所支持的 IBM MQ classes for Java 应用程序 如果 IPv6 为 中使用 Internet Protocol V 6 (IPv6) 地址。

在 Java EE 中运行 IBM MQ classes for Java 应用程序

在 Java EE 中使用 IBM MQ classes for Java 之前，必须考虑某些限制和设计注意事项。

IBM MQ classes for Java 在 Java Platform, Enterprise Edition (Java EE) 环境中使用时具有限制。 在设计，实现和管理在 Java EE 环境中运行的 IBM MQ classes for Java 应用程序时，还必须考虑其他注意事项。 这些限制和注意事项在以下部分中进行了概述。

JTA 事务限制

针对使用 IBM MQ classes for Java 的应用程序的唯一受支持事务管理器是 IBM MQ 本身。 虽然受 JTA 控制的应用程序可以利用 IBM MQ classes for Java，但是通过这些类执行的任何工作不受 JTA 工作单元的控制。 而是，它们会形成与应用程序服务器通过 JTA 接口管理的工作单元分离的本地工作单元。 特别是，JTA 事务的任何回滚不会导致回滚任何已发送或接收的消息。 此限制适用于应用程序或 bean 管理的事务，以及容器管理的事务和所有 Java EE 容器。 要在应用程序服务器协调事务内直接使用 IBM MQ 执行消息传递工作，必须改用 IBM MQ classes for JMS。

线程创建

IBM MQ classes for Java 以内部方式为各种操作创建线程。 例如，以 BINDINGS 方式运行，从而直接在本地队列管理器上调用时，将在 IBM MQ classes for Java 以内部方式创建的“工作程序”线程上进行调用。 可以通过内部方式创建其他线程，例如，以从连接池清除未使用的连接或者移除针对已终止的发布/预订应用程序的预订。

某些 Java EE 应用程序（例如，在 EJB 和 Web 容器中运行的此类应用程序）不得创建新线程。 相反，所有工作都必须在由应用程序服务器管理的主应用程序线程上执行。 当应用程序使用 IBM MQ classes for Java 时，应用程序服务器可能无法区分应用程序代码和 IBM MQ classes for Java 代码，因此先前描述的线程导致应用程序与容器规范不符。 IBM MQ classes for JMS 不会违反这些 Java EE 规范，因此可以作为替代使用。

安全限制

应用程序服务器所实现的安全策略可能会阻止由 IBM MQ classes for Java API 执行的某些操作，例如创建和操作控件的新线程（如先前部分中所述）。

例如，应用程序服务器通常在缺省禁用 Java 安全性的情况下运行，并且允许通过一些特定于应用程序服务器的配置将其启用（某些应用程序服务器还允许对 Java 安全性中使用的策略进行更详细的配置）。 在启用 Java 安全性时，IBM MQ classes for Java 可能会违反为应用程序服务器定义的 Java 安全策略线程规则，并且 API 可能无法创建其正常运作所需的所有线程。 为了防止线程管理出现问题，在启用了 Java 安全性的环境中不支持使用 IBM MQ classes for Java 。

应用程序隔离注意事项

在 Java EE 环境中运行应用程序的预期益处是应用程序隔离。 IBM MQ classes for Java 的设计和实现早于 Java EE 环境。 可以通过不支持应用程序隔离概念的方式使用 IBM MQ classes for Java。 此方面的注意事项的特定示例包括：

- 在 MQEnvironment 类中使用静态（JVM 进程范围）设置，例如：
  - 用于连接识别和认证的用户标识和密码
  - 用于客户机连接的主机名、端口和通道
  - 受保护客户机连接的 TLS 配置

  为一个应用程序的利益修改任何 MQEnvironment 属性也影响使用相同属性的其他应用程序。 在多应用程序环境（例如 Java EE）中运行时，各应用程序必须通过创建具有特定属性集的 MQQueueManager 对象而不是缺省为进程范围 MQEnvironment 类中配置的属性来使用其自己的特有配置。
- MQEnvironment 类引入许多静态方法，这些方法用于对在同一 JVM 进程中使用 IBM MQ classes for Java 的所有应用程序进行全局处理，并且无法覆盖特定应用程序的此行为。 示例包括：
  - 配置 TLS 属性，例如密钥库的位置
  - 配置客户机通道出口
  - 启用或禁用诊断跟踪
  - 管理用于对队列管理器连接使用进行优化的缺省连接池

  调用此类方法会影响在同一 Java EE 环境中运行的所有应用程序。
- 连接池已启用，以优化对同一队列管理器进行多个连接的过程。 缺省连接池管理器为进程范围，并且由多个应用程序共享。 对连接池配置的更改（例如使用 MQEnvironment.setDefaultConnectionManager() 方法替换一个应用程序的缺省连接管理器）因此会影响在同一 Java EE 应用程序服务器中运行的其他应用程序。
- TLS 使用 MQEnvironment 类和 MQQueueManager 对象属性针对使用 IBM MQ classes for Java 的应用程序进行了配置。 它未与应用程序服务器本身的受管安全配置相集成。 必须确保适当配置 IBM MQ classes for Java 以提供所需的安全级别，并且不使用应用程序服务器配置。

绑定方式限制

可以将 IBM MQ 和 WebSphere Application Server 安装在同一台机器上，以使队列管理器的主要版本与 WebSphere Application Server 中提供的 IBM MQ 资源适配器 (RA) 的主要版本不同。 例如 WebSphere Application Server 7.0（随附 IBM MQ RA 级别 7.0.1）可以和 IBM WebSphere MQ 6 队列管理器安装在同一机器上。

如果队列管理器和资源适配器主版本不同，那么无法使用绑定连接。 从 WebSphere Application Server 到队列管理器的任何使用资源适配器的连接都必须使用客户机类型连接。 如果版本相同，那么可以使用绑定连接。

IBM MQ classes for Java 中的字符串转换

IBM MQ classes for Java 直接使用 CharsetEncoders 和 CharsetDecoders 进行字符串转换。 可以使用两个系统属性配置字符串转换的缺省行为。 可以通过 `com.ibm.mq.MQMD` 配置包含不可映射字符的消息的处理。

在 IBM MQ 8.0 之前， IBM MQ classes for Java 中的字符串转换是通过调用 `java.nio.charset.Charset.decode(ByteBuffer)` 和 `Charset.encode(CharBuffer)` 方法完成的。

使用其中任一方法都会导致对格式不正确或不可转换的数据进行缺省替换 (REPLACE)。 该行为可能会隐藏应用程序中的错误，并且导致已转换数据中产生意外的字符，例如 ?。

在 IBM MQ 8.0 中，为了更早更有效地检测此类问题，IBM MQ classes for Java 将直接使用 CharsetEncoder 和 CharsetDecoder 并明确配置对格式错误和不可转换数据的处理。 REPORT 的缺省行为是通过抛出适当的 `MQException` 来报告此类问题。

配置

从 UTF-16 (Java 中使用的字符表示) 到本机字符集 (例如 UTF-8) 的转换称为 `encoding`，而相反方向的转换称为 `decoding`。

当前，解码会采用 `CharsetDecoders` 的缺省行为，通过抛出异常来报告错误。

一个设置用于指定 `java.nio.charset.CodingErrorAction`，以控制编码和解码时的错误处理。 另一个设置用于在编码时控制一个或多个替换字节。 将在解码操作中使用缺省的 Java 替换字符串。

IBM MQ classes for Java 中不可翻译的数据处理的配置

IBM MQ 8.0 中，`com.ibm.mq.MQMD` 包括以下两个字段：

byte[] unMappableReplacement
在无法转换输入字符并且已指定 REPLACE 的情况下将写入编码字符串的字节序列。

缺省值："?".getBytes()
将在解码操作中使用缺省的 Java 替换字符串。

java.nio.charset.CodingErrorAction unmappableAction
指定在编码和解码时要对不可转换的数据采取的操作：

缺省值：CodingErrorAction.REPORT;

用于设置系统缺省值的系统属性

从 IBM MQ 8.0 开始，提供了以下两个 Java 系统属性，用于配置关于字符串转换的缺省行为。

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
指定编码和解码时要对不可转换的数据采取的操作。 值可以是 REPORT、REPLACE 或 IGNORE。

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
设置或获取无法在解码操作中映射字符时要应用的替换字节。将在解码操作中使用缺省的 Java 替换字符串。

为避免 Java 字符与本机字节表示之间混淆，您应将 `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` 指定为表示本机字符集中替换字节的十进制数。

例如，如果本机字符集基于 ASCII（如 ISO-8859-1），那么本机字节 ? 的十进制值为 63，而如果本机字符集为 EBCDIC，它仍为 111。

注: 请注意，如果 MQMD 或 MQMessage 对象设置了 **unmappableAction** 或 **unMappableReplacement** 字段，那么这些字段的值将优先于 Java 系统属性。 这将允许覆盖由 Java 系统属性指定的每条消息的值（如果需要）。

相关概念

第 108 页的『IBM MQ classes for JMS 中的字符串转换』
IBM MQ classes for JMS 直接使用 CharsetEncoders 和 CharsetDecoders 进行字符串转换。 可以使用两个系统属性配置字符串转换的缺省行为。 包含无法映射字符的消息处理可以通过消息属性配置，消息属性用于设置 UnmappableCharacterAction 和替换字节

安装和配置 IBM MQ classes for Java

本部分描述了在安装 IBM MQ classes for Java 时创建的目录和文件，并说明了在安装之后如何配置 IBM MQ classes for Java。

为 *IBM MQ classes for Java* 安装了什么

IBM MQ classes for Java 的最新版本随 IBM MQ 一起安装。 您可能需要覆盖缺省安装选项才能确保完成该操作。

有关安装 IBM MQ 的更多信息，请参阅：

- Multi 安装 IBM MQ
- z/OS 安装 IBM MQ for z/OS 产品

IBM MQ classes for Java 包含在 Java 归档 (JAR) 文件 `com.ibm.mq.jar` 和 `com.ibm.mq.jmqi.jar` 中。

对标准消息头（如可编程命令格式 (PCF)）的支持包含在 JAR 文件 `com.ibm.mq.headers.jar` 中。

对可编程命令格式 (PCF) 的支持包含在 JAR 文件 `com.ibm.mq.pcf.jar` 中。

注: 建议不要在应用程序服务器中使用 IBM MQ classes for Java。 有关在此环境中运行时适用的限制信息，请参阅第 282 页的『在 Java EE 中运行 IBM MQ classes for Java 应用程序』。 有关更多信息，请参阅 在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口。

要点: 除第 285 页的『IBM MQ classes for Java 可再定位的 JAR 文件』中描述的可再定位的 JAR 文件以外，不支持将 IBM MQ classes for Java JAR 文件或本机库复制到其他机器，或者复制到已安装 IBM MQ classes for Java 的机器上的其他位置。

IBM MQ classes for Java 可再定位的 *JAR* 文件

可以将可再定位的 JAR 文件移至需要运行 IBM MQ classes for Java 的系统。

要点:

- 除可再定位的 JAR 文件中描述的可再定位的 JAR 文件以外，不支持将 IBM MQ classes for Java JAR 文件或本机库复制到其他机器，或者复制到已安装 IBM MQ classes for Java 的机器上的其他位置。
- 为避免类装入器冲突，不建议将可再定位的 JAR 文件绑定在同一 Java 运行时内的多个应用程序中。 在此场景中，请考虑让 IBM MQ 可再定位的 JAR 文件在 Java 运行时的类路径上可用。
- 请勿在部署到 Java EE 应用程序服务器（如 WebSphere Application Server）的应用程序中包含可再定位的 JAR 文件。 在这些环境中，应该改为部署和使用 IBM MQ 资源适配器，因为这将包含 IBM MQ classes for Java。 请注意，WebSphere Application Server 将嵌入 IBM MQ 资源适配器，因此无需将其手动部署到此环境中。 除此之外，IBM MQ classes for Java 在 WebSphere Liberty 中不受支持。 有关更多信息，请参阅第 370 页的『Liberty 和 IBM MQ 资源适配器』。
- 如果要将可再定位的 JAR 文件绑定在应用程序中，请确保您包含所有必备的 JAR 文件，如可再定位的 JAR 文件中所述。 您还应确保具有相应的过程在应用程序维护期间更新绑定的 JAR 文件，以确保 IBM MQ classes for Java 保持最新状态，并且重新解决已知问题。

可再定位的 JAR 文件

在企业中，可以将以下文件移动到需要运行 IBM MQ classes for Java 应用程序的系统中：

- com.ibm.mq.allclient.jar
- com.ibm.mq.traceControl.jar
- bcpkix-jdk15on.jar
- bcprov-jdk15on.jar
- V 9.1.0.9 bcutil-jdk15on.jar
- V 9.1.2 org.json.jar

需要 Bouncy Castle 安全提供程序和 CMS 支持 JAR 文件。 有关更多信息，请参阅支持使用 AMS 的非 IBM JRE。 需要以下 JAR 文件：

- bcpkix-jdk15on.jar
- bcprov-jdk15on.jar
- V 9.1.0.9 bcutil-jdk15on.jar

V 9.1.2 如果 IBM MQ classes for Java 应用程序使用 JSON 格式的 CCDT，那么需要 org.json.jar 文件。

文件 com.ibm.mq.allclient.jar 包含 IBM MQ classes for JMS、IBM MQ classes for Java 以及 PCF 和 Headers 类。 如果您将此文件移动到新位置，请确保采取步骤使用新的 IBM MQ 修订包对此新位置进行维护。 此外，如果获取的是临时修订，请确保 IBM 支持人员知悉此文件的使用。

要确定 com.ibm.mq.allclient.jar 文件的版本，请使用以下命令：

```
java -jar com.ibm.mq.allclient.jar
```

以下示例显示此命令的某些样本输出：

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.1.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

com.ibm.mq.traceControl.jar 文件用于动态控制 IBM MQ classes for JMS 应用程序的跟踪。 有关更多信息，请参阅使用 IBM MQ classes for Java 和 IBM MQ classes for JMS 在运行中的流程中控制跟踪。

IBM MQ classes for Java 的安装目录

IBM MQ classes for Java 文件和样本根据平台安装在不同的位置。 随 IBM MQ 一起安装的 Java 运行时环境 (JRE) 的位置也因平台而异。

IBM MQ classes for Java 文件的安装目录

第 287 页的表 48 说明了 IBM MQ classes for Java 文件的安装位置。

表 *48: IBM MQ classes for Java* 安装目录	
平台	**目录**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib
Linux Linux	*MQ_INSTALLATION_PATH*/java/lib
Solaris Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

样本安装目录

IBM MQ 随附了一些样本应用程序，例如，安装验证程序 (IVP)。 第 287 页的表 49 显示了这些样本应用程序的安装位置。 IBM MQ classes for Java 样本位于名为 `wmqjava` 的子目录中。 PCF 样本位于一个名为 `pcf` 的子目录中。

表 *49:* 样本目录	
平台	**目录**
AIX AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i IBM i	/QIBM/ProdData/mqm/java/samples
Linux Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Solaris Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

JRE 的安装目录

IBM MQ classes for JMS 需要 Java 7（或更高版本）的 Java 运行时环境 (JRE)。 随 IBM MQ 安装了一个合适的 JRE。 第 287 页的表 50 显示了该 JRE 的安装位置。 要使用此 JRE 运行 Java 程序（例如，提供的样本），请显式调用 *JRE_LOCATION*`/bin/java` 或将 *JRE_LOCATION*`/bin` 添加到平台的 PATH 环境（或等效环境）中，其中 *JRE_LOCATION* 是 第 287 页的表 50 中给定的目录。

表 *50: JRE* 目录	
平台	**目录**
AIX AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i IBM i	/QIBM/ProdData/mqm/java/jre

表 *50: JRE* 目录 *(继续)*	
平台	**目录**
Linux Linux	*MQ_INSTALLATION_PATH*/java/jre
Solaris Solaris	*MQ_INSTALLATION_PATH*/java/jre
Windows Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

与 IBM MQ classes for Java 相关的环境变量
如果要运行 IBM MQ classes for Java 应用程序，那么它们的类路径中必须包含相应的 IBM MQ classes for Java 和样本目录。

要让 IBM MQ classes for Java 应用程序运行，它们的类路径中必须包含相应的 IBM MQ classes for Java 目录。 要运行样本应用程序，类路径中还必须包含相应的样本目录。 可以在 Java 调用命令或 CLASSPATH 环境变量中提供此信息。

要点: 不支持将 Java 选项 *-Xbootclasspath* 设置为包含 IBM MQ classes for Java。

第 288 页的表 51 显示了在每个平台上使用以运行 IBM MQ classes for Java 应用程序（包括样本应用程序）的相应 CLASSPATH 设置。

表 *51:* 运行 *IBM MQ classes for Java* 应用程序（包括 *IBM MQ classes for Java* 样本应用程序）的 *CLASSPATH* 设置	
平台	**CLASSPATH 设置**
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Solaris Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/samples/pcf

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

如果您使用 -Xlint 选项进行编译，可能会看到一条消息，警告您 com.ibm.mq.ese.jar 不存在。 您可以忽略此警告。 此文件仅在安装了 Advanced Message Security 的情况下存在。

随 IBM MQ classes for JMS 提供的脚本使用以下环境变量：

MQ_JAVA_DATA_PATH
此环境变量指定日志和跟踪输出的目录。

MQ_JAVA_INSTALL_PATH
此环境变量指定安装 IBM MQ classes for Java 的目录，如 IBM MQ classes for Java 安装目录中所示。

MQ_JAVA_LIB_PATH
此环境变量指定存储 IBM MQ classes for Java 库的目录，如每个平台的 IBM MQ classes for Java 库的位置中所示。 IBM MQ classes for Java 提供的某些脚本（例如，IVTRun）使用此环境变量。

Windows 在 Windows 上，所有环境变量都会在安装期间自动设置。

UNIX 在 UNIX 上，您可以使用脚本 **setjmsenv**（如果使用的是 32 位 JVM）或 **setjmsenv64**（如果使用的是 64 位 JVM）来设置环境变量。

Linux UNIX 在 UNIX and Linux 上，这些脚本位于 *MQ_INSTALLATION_PATH*/java/bin 目录中。

IBM i 在 IBM i 上，环境变量 QIBM_MULTI_THREADED 必须设置为 Y。 随后可使用运行单线程应用程序的方法来运行多线程应用程序。 有关更多信息，请参阅使用 Java 和 JMS 设置 IBM MQ。

IBM MQ classes for Java 需要 Java 7 Java 运行时环境 (JRE)。 有关随 IBM MQ 安装的合适 JRE 的位置信息，请参阅第 286 页的『IBM MQ classes for Java 的安装目录』。

IBM MQ classes for Java 库
IBM MQ classes for Java 库的位置因平台而异。 您启动应用程序时需指定此位置。

要指定 Java 本机接口 (JNI) 库的位置，请使用采用以下格式的 **java** 命令启动您的应用程序：

```
java -Djava.library.path= library_path application_name
```

其中，*library_path* 是包含这些 JNI 库的 IBM MQ classes for Java 的路径。 第 289 页的表 52 显示了每个平台的 IBM MQ classes for Java 库的位置。 在此表中，*MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。

表 *52:* 各平台的 *IBM MQ classes for Java* 库的位置

平台	包含 IBM MQ classes for Java 库的目录
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib（32 位库） *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
Linux Linux（x86 平台）	*MQ_INSTALLATION_PATH*/java/lib
Linux Linux（POWER、x86-64 和 zSeries s390x 平台）	*MQ_INSTALLATION_PATH*/java/lib（32 位库） *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
Solaris Solaris（x86-64 和 SPARC 平台）	*MQ_INSTALLATION_PATH*/java/lib（32 位库） *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib（32 位库） *MQ_INSTALLATION_PATH*\java\lib64（64 位库）
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib（32 位和 64 位库）

注:

1. Solaris Linux AIX 在 AIX、Linux（Power 平台）或 Solaris 上，使用 32 位库或 64 位库。 仅当在 64 位平台上的 64 位 Java 虚拟机 (JVM) 中运行应用程序时，才使用 64 位库。 否则，请使用 32 位库。
2. Windows 在 Windows 上，您可以使用 PATH 环境变量来指定 IBM MQ classes for Java 库的位置，而不是在 **java** 命令中指定它们的位置。
3. IBM i 要在 IBM i 上的绑定方式下使用 IBM MQ classes for Java ，请确保库 QMQMJAVA 位于库列表中。
4. z/OS 在 z/OS 上，您可以使用 32 位或 64 位 Java 虚拟机 (JVM)。 您不必指定要使用哪些库；IBM MQ classes for Java 可以自行确定装入哪些 JNI 库。

相关概念

使用 IBM MQ classes for Java
在安装 IBM MQ classes for Java 之后，您可以配置安装，以便运行自己的应用程序。

IBM MQ classes for Java 对 *OSGi* 的支持
OSGi 提供了一个框架，支持以捆绑软件形式部署应用程序。 在 IBM MQ classes for Java 中提供了三个 OSGi 捆绑软件。

OSGi 提供了一个通用的安全受管 Java 框架，此框架支持部署以捆绑软件形式提供的应用程序。 与 OSGi 兼容的设备可以下载和安装捆绑软件，并在不再需要时移除这些捆绑软件。 此框架能够以动态的可扩展方式管理捆绑软件的安装和更新。

IBM MQ classes for Java 中包含以下 OSGi 捆绑软件。

com.ibm.mq.osgi.java_*version_number*.jar
这些 JAR 文件允许应用程序使用 IBM MQ classes for Java。

com.ibm.mq.osgi.allclient_*version_number*.jar
此 JAR 文件允许应用程序同时使用 IBM MQ classes for JMS 和 IBM MQ classes for Java，并且还包含用于处理 PCF 消息的代码。

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
此 JAR 文件提供了 com.ibm.mq.osgi.allclient_*version_number*.jar 的先决条件。

其中 *version_number* 是已安装的 IBM MQ 的版本号。

这些捆绑软件安装在 IBM MQ 安装的 java/lib/OSGi 子目录或 Windows 上的 java\lib\OSGi 文件夹中。

从 IBM MQ 8.0 起，对任何新应用程序都使用捆绑软件 com.ibm.mq.osgi.allclient_8.0.0.0.jar 和 com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar。 使用这些捆绑软件可以打破不能在同一 OSGi 框架中同时运行 IBM MQ classes for JMS 和 IBM MQ classes for Java 的限制。 但所有其他限制仍然适用。 对于 IBM MQ 8.0 之前的版本，使用 IBM MQ classes for JMS 或 IBM MQ classes for Java 的限制适用。

另外 9 个捆绑软件也安装到 IBM MQ 安装的 java/lib/OSGi 子目录或 Windows 上的 java\lib\OSGi 文件夹。 这些捆绑软件是 IBM MQ classes for JMS 的一部分，不得装入到已装入了 IBM MQ classes for Java 捆绑软件的 OSGi 运行时环境中。 如果将 IBM MQ classes for Java OSGi 捆绑软件装入到同时装入了 IBM MQ classes for JMS 捆绑软件的 OSGi 运行时环境中，那么当使用 IBM MQ classes for Java 捆绑软件或 IBM MQ classes for JMS 捆绑软件的应用程序运行时，会出现以下示例中显示的错误：

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

IBM MQ classes for Java 的 OSGi 捆绑软件已写入 OSGi R4 规范中；不能用于 OSGi R3 环境。

必须正确设置系统路径或库路径，以便 OSGi 运行时环境能够找到任何必需的 DLL 文件或共享库。

如果您使用 IBM MQ classes for Java 的 OSGi 捆绑软件，将不支持以 Java 编写的通道出口类，因为在像 OSGi 这样包含多个类装入器的环境中装入类时存在固有问题。 用户捆绑软件可以感知到 IBM MQ classes for Java 捆绑软件，但 IBM MQ classes for Java 捆绑软件不会感知到任何用户捆绑软件。 因此，IBM MQ classes for Java 捆绑软件中使用的类装入器无法装入用户捆绑软件中的通道出口类。

有关 OSGi 的更多信息，请参阅 OSGi 联盟网站。

z/OS 安装 *IBM MQ classes for Java on z/OS*
在 z/OS 上，在运行时使用的 STEPLIB 必须包含 IBM MQ SCSQAUTH 和 SCSQANLE 库。

从 UNIX and Linux 系统服务中，您可以使用以下示例中显示的 `.profile` 中的一行来添加这些库，将 `thlqual` 替换为您在安装 IBM MQ 时选择的高级数据集限定符：

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

在其他环境中，您通常需要编辑启动 JCL 以在 STEPLIB 连接上包含 SCSQAUTH：

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

IBM MQ classes for Java 配置文件
IBM MQ classes for Java 配置文件指定用于配置 IBM MQ classes for Java 的属性。

IBM MQ classes for Java 配置文件的格式是标准 Java 属性文件的格式。

IBM MQ classes for Java 安装目录的 `bin` 子目录中提供了样本配置文件 `mqjava.config`。 该文件记录了所有受支持的属性及其缺省值。

注: 在将 IBM MQ 安装升级到未来修订包时，会覆盖该样本配置文件。 因此，建议生成该样本配置文件的副本以用于您自己的应用程序。

您可以选择 IBM MQ classes for Java 配置文件的名称和位置。 启动应用程序时，请使用以下格式的 **java** 命令：

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

在该命令中，*config_file_url* 是统一资源定位符 (URL)，指定 IBM MQ classes for Java 配置文件的名称和位置。 支持以下类型的 URL：`http`、`file`、`ftp` 和 `jar`。

以下示例显示了 **java** 命令：

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

此命令将 IBM MQ classes for Java 配置文件标识为本地 Windows 系统上的 `D:\mydir\mqjava.config` 文件。

在应用程序启动时，IBM MQ classes for Java 将读取配置文件的内容，并将指定的属性存储在内部属性库中。 如果 **java** 命令不识别配置文件，或者如果找不到配置文件，那么 IBM MQ classes for Java 使用所有属性的缺省值。 如果需要，您可以通过在 **java** 命令上将配置文件中的任何属性指定为系统属性来覆盖该属性。

IBM MQ classes for Java 配置文件可以与应用程序和队列管理器或代理程序之间的任何受支持的传输一起使用。

覆盖 IBM MQ MQI client 配置文件中指定的属性

IBM MQ MQI client 配置文件还可以指定用于配置 IBM MQ classes for Java 的属性。 但是，仅当应用程序以客户机方式连接到队列管理器时，IBM MQ MQI client 配置文件中指定的属性才适用。

如果需要，可以通过将 IBM MQ MQI client 配置文件中的任何属性 (attribute) 指定为 IBM MQ classes for Java 配置文件中的特性 (property) 来覆盖该属性。 要覆盖 IBM MQ MQI client 配置文件中的属性，请在 IBM MQ classes for Java 配置文件中使用具有以下格式的条目：

```
com.ibm.mq.cfg.stanza.propName=propValue
```

该条目中的变量具有以下含义：

stanza
IBM MQ MQI client 配置文件中包含该属性的节的名称。

propName
IBM MQ MQI client 配置文件中指定的属性的名称。

propValue
用于覆盖 IBM MQ MQI client 配置文件中指定属性值的特性的值。

或者，您可以通过在 **java** 命令中将属性指定为系统属性来覆盖 IBM MQ MQI client 配置文件中的属性。 使用先前格式将属性指定为系统属性。

IBM MQ MQI client 配置文件中只有以下属性与 IBM MQ classes for Java 相关。 如果您指定或覆盖其他属性，那么将无效。 具体而言，请注意，不会使用 客户机配置文件的 CHANNELS 节 中的 `ChannelDefinitionFile` 和 `ChannelDefinitionDirectory`。 请参阅第 305 页的『将客户机通道定义表用于 IBM MQ classes for Java』，以获取有关如何将 CCDT 与 IBM MQ classes for Java 一起使用的详细信息。

表 *53:* 客户机配置文件的哪节包含哪个属性

节	属性
客户机配置文件的 CHANNELS 节	Put1DefaultAlwaysSync
客户机配置文件的 CHANNELS 节	PasswordProtection
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath64
客户机配置文件的 ClientExitPath 节	JavaExitsClasspath
客户机配置文件的 JMQI 节	useMQCSPauthentication
客户机配置文件的 MessageBuffer 节	MaximumSize
客户机配置文件的 MessageBuffer 节	PurgeTime
客户机配置文件的 MessageBuffer 节	UpdatePercentage
客户机配置文件的 TCP 节	ClntRcvBuffSize
客户机配置文件的 TCP 节	ClntSndBuffSize
客户机配置文件的 TCP 节	Connect_Timeout
客户机配置文件的 TCP 节	KeepAlive

有关 IBM MQ MQI client 配置的更多信息，请参阅使用配置文件配置客户机。

相关任务
跟踪 IBM MQ Classes for Java 应用程序

Java 标准环境跟踪节
使用 Java 标准环境跟踪设置节配置 IBM MQ classes for Java 跟踪功能。

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputName*
`traceOutputName` 是要将跟踪输出发送至的目录和文件名。

缺省情况下，会将跟踪信息写入应用程序当前工作目录中的跟踪文件。 跟踪文件的名称取决于应用程序的运行环境：

- 对于 IBM MQ classes for Java for IBM MQ 9.0.0 Fix Pack 1 或更低版本，会将跟踪写入名为 `mqjms_%PID%.trc` 的文件。
- 从 IBM MQ 9.0.0 Fix Pack 2 起，如果应用程序已从 JAR 文件 `com.ibm.mq.jar` 装入 IBM MQ classes for Java，那么会将跟踪写入名为 `mqjava_%PID%.trc` 的文件。

- 从 IBM MQ 9.0.0 Fix Pack 2 起，如果应用程序已从可重定位的 JAR 文件 `com.ibm.mq.allclient.jar` 装入 IBM MQ classes for Java，那么会将跟踪写入名为 `mqjavaclient_%PID%.trc` 的文件。
- V 9.1.5 V 9.1.0.5 从 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5 开始，如果应用程序已从 JAR 文件 `com.ibm.mq.jar` 装入 IBM MQ classes for Java，那么会将跟踪写入名为 `mqjava_%PID%.cl%u.trc` 的文件。
- V 9.1.5 V 9.1.0.5 从 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5 开始，如果应用程序已从可重新定位的 JAR 文件 `com.ibm.mq.allclient.jar` 装入 IBM MQ classes for Java，那么会将跟踪写入名为 `mqjavaclient_%PID%.cl%u.trc` 的文件。

其中，*%PID%* 是正在跟踪的应用程序的进程标识，*%u* 是唯一编号，用于区分在不同 Java 类装入器下运行跟踪的线程之间的文件。

如果进程标识不可用，那么将生成一个随机数（以字母 `f` 为前缀）。 要将进程标识包含在您指定的文件名中，请使用字符串 `%PID%`。

如果指定备用目录，那么该目录必须存在并且您必须具有该目录的写许可权。 如果您没有写许可权，那么会将跟踪输出写入 `System.err`。

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList 是跟踪的程序包和类的列表，或者具有特殊值 `ALL` 或 `NONE`。

使用分号 `;` 分隔程序包或类名称。 *includeList* 缺省为 `ALL`，将跟踪 IBM MQ classes for Java 中的所有程序包和类。

注: 您可以包含某个程序包，但在随后排除该程序包的子程序包。 例如，如果您包含程序包 `a.b` 并排除程序包 `a.b.x`，那么跟踪将包含 `a.b.y` 和 `a.b.z` 中的所有内容，但不包含 `a.b.x` 和 `a.b.x.1` 中的内容。

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList 是不跟踪的程序包和类的列表，或者具有特殊值 `ALL` 或 `NONE`。

使用分号 `;` 分隔程序包或类名称。 *excludeList* 缺省为 `NONE`，因此跟踪 IBM MQ classes for JMS 中的所有程序包和类。

注: 您可以排除某个程序包，但在随后包含该程序包的子程序包。 例如，如果您排除程序包 `a.b` 并包含程序包 `a.b.x`，那么跟踪将包含 `a.b.x` 和 `a.b.x.1` 中的所有内容，但不包含 `a.b.y` 和 `a.b.z`。

将包含在同一级别（包含和排除）指定的任何程序包或类。

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes 是从任何字节数组跟踪的最大字节数。

如果将 *maxArrayBytes* 设置为正整数，将限制字节数组中写出到跟踪文件的字节数。 将在写出 *maxArrayBytes* 之后截断字节数组。 设置 *maxArrayBytes* 可减小所生成跟踪文件的大小，降低跟踪对应用程序性能的影响。

该属性的值如果为 `0`，表示不会将任何字节数组的内容发送至跟踪文件。

缺省值为 `-1`，这将移除对字节数组中发送至跟踪文件的字节数的任何限制。

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes 是写入跟踪输出文件的最大字节数。

maxTraceBytes 将使用 *traceCycles*。 如果写入的跟踪字节数接近限制，将关闭文件，并启动新的跟踪输出文件。

值为 `0` 表示跟踪输出文件的长度为零。 缺省值为 `-1`，表示写入跟踪输出文件的数据量不受限制。

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles 是要循环的跟踪输出文件数。

如果当前的跟踪输出文件达到了 *maxTraceBytes* 指定的限制，那么将关闭该文件。 会按顺序将进一步的跟踪输出写入下一个跟踪输出文件。 通过将数字后缀附加到文件名来区分每个跟踪输出文件。 当前或

最新的跟踪输出文件为 mqjms.trc.0，下一个最新的跟踪输出文件为 mqjms.trc.1。 较旧的跟踪文件将遵循相同的编号模式上限。

traceCycles 的缺省值为 1。 如果 *traceCycles* 为 1，那么在当前跟踪输出文件达到其最大大小时，将关闭并删除该文件。 将启动相同名称的新跟踪输出文件。 因此，同时只存在一个跟踪输出文件。

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters 将控制是否在跟踪中包含方法参数和返回值。

traceParameters 缺省为 TRUE。 如果将 *traceParameters* 设置为 FALSE，那么只会跟踪方法特征符。

com.ibm.msg.client.commonservices.trace.startup = *startup*
在分配资源期间存在 IBM MQ classes for Java 的初始化阶段。 将在资源分配阶段期间对主要跟踪功能进行初始化。

如果将 *startup* 设置为 TRUE，将使用启动跟踪。 跟踪信息将立即生成并包含所有组件的设置，包括跟踪功能自身。 启动跟踪信息可用于诊断配置问题。 启动跟踪信息始终会写入 System.err。

startup 缺省为 FALSE。

将在初始化完成前检查 *startup*。 鉴于此，仅在命令行上将属性指定为 Java 系统属性。 请勿在 IBM MQ classes for Java 配置文件中指定该属性。

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
将 *compressedTrace* 设置为 TRUE 以压缩跟踪输出。

compressedTrace 的缺省值为 FALSE。

如果将 *compressedTrace* 设置为 TRUE，那么将压缩跟踪输出。 缺省跟踪输出文件名具有扩展名 .trz。 如果将压缩设置为 FALSE（缺省值），那么该文件将具有扩展名 .trc 以指示它未压缩。 但是，如果在 *traceOutputName* 中指定了跟踪输出的文件名（将改用该名称）；将不会对该文件应用后缀。

压缩的跟踪输出小于未压缩的跟踪输出。 由于 I/O 较少，因此与未压缩的跟踪相比，写出速度更快。 与未压缩的跟踪相比，压缩的跟踪对 IBM MQ classes for Java 的性能影响较小。

如果设置了 *maxTraceBytes* 和 *traceCycles*，将创建多个压缩跟踪文件以替代多个平面文件。

如果 IBM MQ classes for Java 以不受控制的方式结束，那么压缩的跟踪文件可能无效。 鉴于此，只有当以受控方式关闭 IBM MQ classes for Java 时，才能使用跟踪压缩。 只有当被调查的问题不会导致 JVM 自身意外停止时，才可以使用跟踪压缩。 如果诊断问题可能会导致 System.Halt() 关闭或出现异常、不受控制的 JVM 终止，请勿使用跟踪压缩。

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel 指定跟踪的过滤级别。 定义的跟踪级别如下所示：

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

每个跟踪级别包含所有较低级别。 例如，如果将跟踪级别设置为 TRACE_INFO，那么具有 TRACE_EXCEPTION、TRACE_WARNING 或 TRACE_INFO 定义级别的任何跟踪点都将写入跟踪。 所有其他跟踪点将被排除。

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace 控制是否在 WebSphere Application Server 环境中使用 IBM MQ classes for Java 客户机跟踪服务。

如果将 *standaloneTrace* 设置为 TRUE，那么 IBM MQ classes for Java 客户机跟踪属性将用于确定跟踪配置。

如果将 *standaloneTrace* 设置为 FALSE，并且 IBM MQ classes for Java 客户机正在 WebSphere Application Server 容器中运行，那么将使用 WebSphere Application Server 跟踪服务。 生成的跟踪信息取决于应用程序服务器的跟踪设置。

standaloneTrace 的缺省值为 FALSE。

IBM MQ classes for Java 和软件管理工具
软件管理工具（如 Apache Maven）可以与 IBM MQ classes for Java 一起使用。

许多大型开发组织都使用这些工具来集中管理第三方库的存储库。

IBM MQ classes for Java 包含大量 JAR 文件。 在使用该 API 开发 Java 语言应用程序时，在开发应用程序的机器上需要安装 IBM MQ 服务器、IBM MQ 客户机或 IBM MQ 客户机 SupportPac。

如果要使用软件管理工具并将构成 IBM MQ classes for Java 的 JAR 文件添加到集中管理的存储库，必须遵守以下要点：

- 存储库或容器必须仅对组织内部的开发人员可用。 禁止组织外部的任何分发。
- 存储库需要包含来自单个 IBM MQ 发行版或修订包的一组完整且一致的 JAR 文件。
- 您负责通过 IBM 支持人员提供的任何维护服务来更新存储库。

从 IBM MQ 8.0 起，需要将 `com.ibm.mq.allclient.jar` JAR 文件安装到存储库中。

从 IBM MQ 9.0 开始，需要使用 Bouncy Castle 安全提供程序和 CMS 支持 JAR 文件。 有关更多信息，请参阅第 285 页的『IBM MQ classes for Java 可再定位的 JAR 文件』和对非 IBM JRE 的支持。

IBM MQ classes for Java 应用程序的后安装设置

在安装 IBM MQ classes for Java 之后，您可以配置安装，以便运行自己的应用程序。

请记得查看 IBM MQ 产品自述文件，以了解最新信息，或了解有关您环境的更多特定信息。 可以在 IBM MQ， WebSphere MQ 和 MQSeries 产品自述文件 网页上找到最新版本的产品自述文件。

在尝试以绑定方式运行 IBM MQ classes for Java 应用程序之前，请确保已按 配置中所述配置了 IBM MQ 。

配置您的队列管理器以接受来自 *IBM MQ classes for Java* 的客户机连接
要配置您的队列管理器以接受来自客户机的入局连接请求，请定义并允许使用服务器连接通道，然后启动侦听器程序。

请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』，以了解详细信息。

在 *Java security manager* 下运行 *IBM MQ classes for Java* 应用程序
IBM MQ classes for Java 可以在启用 Java security manager 的情况下运行。 要在启用 Java security manager 的情况下成功运行应用程序，您必须使用合适的策略定义文件配置您的 Java virtual machine (JVM)。

创建合适的策略定义文件的最简单方法是更改 Java runtime environment (JRE) 提供的策略文件。 在大多数系统上，此文件存储在 `path lib/security/java.policy`（相对于 JRE 目录）中。 您可以使用首选编辑器或者使用 JRE 提供的策略工具程序编辑策略文件。

您必须授予 `com.ibm.mq.jmqi.jar` 文件的权限，以便它可以：

- 创建套接字（以客户机模式）
- 装入本机库（以绑定模式）
- 从环境中读取各种属性

在 Java security manager 下运行时，系统属性 **os.name** 必须可供 IBM MQ classes for Java 使用。

如果 Java 应用程序使用 Java security manager，那么必须向应用程序使用的 `java.security.policy` 文件添加以下许可权，否则将向应用程序抛出异常：

```
permission java.lang.RuntimePermission "modifyThread";
```

在通过到队列管理器的 TCP/IP 连接管理多复用对话的分配和关闭过程中，客户机需要此 RuntimePermission。

策略文件条目示例

以下是一个策略文件条目示例，使 IBM MQ classes for Java 能够在缺省安全管理器下成功运行。 将此示例中的字符串 *MQ_INSTALLATION_PATH* 替换为您系统上安装 IBM MQ classes for Java 的位置。

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
```

```
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

此策略文件示例使 IBM MQ classes for Java 能够在安全管理器下正常工作，但您可能还需要先使自己的代码正确运行，然后应用程序才能正常工作。

目前还没有专门为用于此安全管理器而启用 IBM MQ classes for Java 提供的样本代码；不过，已经对此策略文件和适当的缺省安全管理器运行了 IVT 测试。

在 *CICS Transaction Server* 下运行 *IBM MQ classes for Java* 应用程序

可以将 IBM MQ classes for Java 应用程序作为事务在 CICS Transaction Server 下运行。

要在 CICS Transaction Server for z/OS 下作为事务运行 IBM MQ classes for Java 应用程序，请执行以下步骤:

1. 使用所提供的 CEDA 事务针对 CICS 定义应用程序和事务。
2. 确保已将 IBM MQ CICS 适配器安装到您的 CICS 系统。 z/OS （请参阅将 IBM MQ 与 CICS 一起使用，以获取详细信息。）
3. 确保 CICS 中指定的 JVM 环境中包含相应的 CLASSPATH 和 LIBPATH 条目。
4. 使用任何常规流程启动此事务。

有关运行 CICS Java 事务的更多信息，请参阅您的 CICS 系统文档。

验证 *IBM MQ classes for Java* 安装

IBM MQ classes for Java 提供了一个安装验证程序 MQIVP。 您可以使用此程序来测试 IBM MQ classes for Java 的所有连接方式。

此程序会提示进行一些选择并提供其他数据，以确定您希望验证哪种连接方式。 请使用以下过程验证您的安装:

1. 如果要以客户机方式运行程序，请按 第 972 页的『配置队列管理器以接受多平台上的客户机连接』中所述配置队列管理器。 要使用的队列为 SYSTEM.DEFAULT.LOCAL.QUEUE。
2. 如果您要以客户机模式运行程序，还需参考第 281 页的『使用 IBM MQ classes for Java』。

   在要运行此程序的系统上执行此过程的其余步骤。
3. 确保已根据第 288 页的『与 IBM MQ classes for Java 相关的环境变量』中的说明更新了您的 CLASSPATH 环境变量。
4. 将目录更改为 *MQ_INSTALLATION_PATH*/mqm/samp/wmqjava/samples，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 安装的路径。 然后，在命令提示符下，输入:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   其中，*library_path* 是 IBM MQ classes for Java 库的路径（请参阅 第 289 页的『IBM MQ classes for Java 库』）。

   在出现带有 (1) 标记的提示时:

   - 要使用 TCP/IP 连接，请输入 IBM MQ 服务器主机名。
   - 要使用本机连接（绑定模式），请保留字段为空（不输入名称）。

   该程序会尝试:

   - 1. 连接到队列管理器
   - 2. 打开队列 SYSTEM.DEFAULT.LOCAL.QUEUE，将消息放入队列中，从队列中获取消息，然后关闭队列

- 3. 断开与队列管理器的连接
- 4. 如果这些操作成功，那么会返回一条消息

以下是一个您可能会看到的提示和响应的示例。 实际提示和您的响应取决于您的 IBM MQ 网络。

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

注:

1. z/OS 在 z/OS 上，请在看到带有 (1) 标记的提示时将字段保留为空。
2. 如果您选择服务器连接，将不会看到标记为 (2) 的提示。
3. IBM i 在 IBM i 上，您只能从 QShell 发布 `java MQIVP` 命令。 或者，您也可以使用 CL 命令 `RUNJVA CLASS(MQIVP)` 运行此应用程序。

使用 *IBM MQ classes for Java* 样本应用程序

IBM MQ classes for Java 样本应用程序概述了 IBM MQ classes for Java API 的常用功能。 使用这些样本应用程序，可以验证安装和消息传递服务器设置，并帮助您构建自己的应用程序。

关于此任务

如果您在创建自己的应用程序时需要帮助，那么可以使用这些样本应用程序作为起点。 已提供了每个样本应用程序的源代码和已编译的版本。 请查看样本源代码，并了解为应用程序创建每个必需对象（MQQueueManager、MQConstants、MQMessage、MQPutMessageOptions 和 MQDestination）的关键步骤，并根据需要设置任何特定属性以指定应用程序的工作方式。 有关更多信息，请参阅第 301 页的『编写 IBM MQ classes for Java 应用程序』。 在 IBM MQ Java 的未来发行版中可随时更改这些样本。

第 298 页的表 54 显示了 IBM MQ classes for Java 样本应用程序在每个平台上的安装位置：

表 *54: IBM MQ classes for Java* 样本应用程序的安装目录

平台	目录
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/samples
Windows Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\samples
IBM i IBM i	/qibm/proddata/mqm/java/samples/wmqjava/samples
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/wmqjava

第 299 页的表 55 显示 IBM MQ classes for Java 随附的样本应用程序集合。

表 55: *IBM MQ classes for Java* 样本应用程序	
样本名称	**描述**
IMSBridgeSample.java	演示将 IMS Bridge 与 IBM MQ classes for Java 一起使用的简单程序。
MQIVP.java	IBM MQ Java 安装验证程序。
MQMessagePropertiesSample.java	演示对引入 IBM WebSphere MQ 7.0 的消息属性 API 的使用。
MQPubSubApiSample.java	演示对引入 IBM WebSphere MQ 7.0 的发布/预订 API 的使用。
MQSample.java	演示在队列中放置和获取消息的简单程序。
MQSampleMessageManager.java	用于在 IBM MQ 基本 Java 样本中处理消息的实用程序类。
mqjcivp.properties	此资源束包含 IBM MQ classes for Java 安装验证程序 (`MQIVP.java`) 使用的消息。

IBM MQ classes for Java 提供了一个名为 `runjms` 的脚本，此脚本可用于运行样本应用程序。 该脚本会设置 IBM MQ 环境，以便您可以运行 IBM MQ classes for Java 样本应用程序。

第 299 页的表 56 显示了每个平台上的脚本位置：

表 56: *runjms* 脚本的位置	
平台	**目录**
UNIX UNIX Linux Linux	*MQ_INSTALLATION_PATH*/java/bin/runjms
Windows Windows	*MQ_INSTALLATION_PATH*\java\bin\runjms.bat
IBM i IBM i	/qibm/proddata/mqm/java/bin/runjms 或者 /qibm/proddata/mqm/java/bin/runjms64
z/OS z/OS	*MQ_INSTALLATION_PATH*java/bin/runjms

要使用 `runjms` 脚本来调用样本应用程序，请完成以下步骤：

过程

1. 打开命令提示符，浏览至包含要运行的样本应用程序的目录。
2. 输入以下命令：

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   该样本应用程序会显示其需要的参数列表。
3. 输入以下命令以使用这些参数运行该样本：

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

示例

Linux 例如，要在 Linux 上运行 MQIVP 样本，请输入以下命令：

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

相关概念

第 70 页的『为 IBM MQ classes for JMS 安装了什么』
安装 IBM MQ classes for JMS 时，将会创建大量文件和目录。 在 Windows 上，通过自动设置环境变量在安装期间执行了某些配置。 在其他平台上的特定 Windows 环境中，必须先设置环境变量，然后才能运行 IBM MQ classes for JMS 应用程序。

解决 *IBM MQ classes for Java* 问题

首先运行安装验证程序。 您可能还需要使用跟踪功能。

如果应用程序没有成功完成，请运行安装验证程序，并按照诊断消息中给出的建议进行操作。 第 297 页的『验证 IBM MQ classes for Java 安装』中描述了安装验证程序。

如果问题继续存在，您需要联系 IBM 服务团队，他们可能会让您启用跟踪功能。 按照以下示例中的过程完成此操作。

要跟踪 MQIVP 程序，请执行以下操作：

- 创建 `com.ibm.mq.commonservices` 属性文件（请参阅使用 com.ibm.mq.commonservices）。
- 输入以下命令：

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

其中：

- *commonservices_properties_file* 是 `com.ibm.mq.commonservices` 属性文件的路径（包括文件名）。
- *library_path* 是 IBM MQ classes for Java 库的路径（请参阅第 289 页的『IBM MQ classes for Java 库』）。

有关如何使用跟踪的更多信息，请参阅跟踪 IBM MQ classes for Java 应用程序。

V 9.1.0 z/OS MQ Adv. VUE Java 客户机与在 z/OS 上运行的批处理应用程序的连接

通过使用客户机连接， z/OS 上的 IBM MQ classes for Java 应用程序可以连接到 z/OS 上具有 **ADVCAP**(ENABLED) 属性的队列管理器。 使用客户机连接可以简化 IBM MQ 拓扑。

值 **ADVCAP**(ENABLED) 仅适用于 z/OS 队列管理器，许可为 IBM MQ Advanced for z/OS Value Unit Edition (请参阅 IBM MQ 产品标识和导出信息)，并在 **QMGRPROD** 设置为 ADVANCEDVUE 的情况下运行。

请参阅 DISPLAY QMGR 以获取有关 **ADVCAP** 的更多信息，并参阅 START QMGR 以获取有关 **QMGRPROD** 的更多信息。

z/OS 上的 IBM MQ classes for Java 应用程序无法使用客户机方式连接来连接到未在 z/OS 上运行的队列管理器或未设置 **ADVCAP**(ENABLED) 选项的队列管理器。

如果 z/OS 上的 IBM MQ classes for Java 应用程序尝试使用客户机方式进行连接，并且不允许这样做，那么将返回 MQRC_ENVIRONMENT_ERROR 。

Advanced Message Security (AMS) 支持

V 9.1.0

从 IBM MQ 9.1 开始， IBM MQ classes for Java 客户机应用程序可以在连接到远程 z/OS 系统上的 IBM MQ Advanced for z/OS Value Unit Edition 队列管理器时使用 AMS 。

仅在 z/OS 上的 keystore.conf 中支持新的密钥库类型 jceracfks，其中：

- 属性名称前缀是 jceracfks，此名称前缀不区分大小写。
- 密钥库是 RACF 密钥环。
- 无需密码，因此将忽略密码。 原因是 RACF 密钥环不使用密码。
- 如果指定了提供程序，那么提供程序必须为 IBMJCE。

将 jceracfks 与 AMS 一起使用时，密钥库必须采用以下格式：safkeyring://*user*/*keyring*，其中：

- safkeyring 是字面值，此名称不区分大小写
- *user* 是拥有此密钥环的 RACF 用户标识
- *keyring* 是 RACF 密钥环的名称，密钥环名称区分大小写

以下示例将标准 AMS 密钥环用于用户 JOHNDOE：

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

编写 IBM MQ classes for Java 应用程序

这一组主题所提供的信息可帮助您编写 Java 应用程序，以便与 IBM MQ 系统交互。

要使用 IBM MQ classes for Java 访问 IBM MQ 队列，可以编写 Java 应用程序来包含一些调用，这些调用将在 IBM MQ 队列中放置和获取消息。 有关每个类的详细信息，请参阅 IBM MQ classes for Java。

注: IBM MQ classes for Java 不支持自动客户机重新连接。

IBM MQ classes for Java 接口

过程化的 IBM MQ 应用程序编程接口使用动词，这些动词作用于对象。 Java 编程接口使用对象，您可以通过调用方法对这些对象进行操作。

过程化的 IBM MQ 应用程序编程接口是围绕动词构建的，例如：

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

这些动词全部作为参数，会获取它们要进行操作的 IBM MQ 对象的句柄。 您的程序由一系列 IBM MQ 对象组成，通过调用这些对象的方法可以执行程序。

在使用过程接口时，通过调用 MQDISC(Hconn, CompCode, Reason) 与队列管理器断开连接，其中，*Hconn* 是队列管理器的句柄。

在 Java 接口中，队列管理器由一个 MQQueueManager 类对象表示。 通过调用该类的 Disconnect() 方法可以断开与队列管理器的连接。

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

IBM MQ classes for Java 连接方式

您为 IBM MQ classes for Java 编程的方法在某种程度上取决于要使用的连接方式。

如果您使用客户机连接，虽然与 IBM MQ MQI client 之间会有很多不同，但在概念上却相似。 如果您使用绑定模式，可以使用快速路径绑定，并且可以发布 MQBEGIN 命令。 您可以通过在 MQEnvironment 类中设置变量来指定要使用哪种模式。

IBM MQ classes for Java 客户机连接

将 IBM MQ classes for Java 作为客户机使用时，它就像 IBM MQ MQI client 一样，但有一些不同之处。

如果您为 *IBM MQ classes for Java* 编程以将其用作客户机，请注意以下不同之处：

- 它只支持 TCP/IP。
- 它在启动时不读取任何 IBM MQ 环境变量。
- 将存储到通道定义和环境变量中的信息可以存储在一个名为 Environment 的类中。 或者，也可以在连接后将该信息作为参数进行传递。
- 错误和异常条件将写入一个在 `MQException` 类中指定的日志中。 缺省错误目标是 Java 控制台。
- IBM MQ 客户机配置文件中只有以下属性与 IBM MQ classes for Java 相关。 如果您指定其他属性，这些属性将无效。

节	属性
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath64
客户机配置文件的 ClientExitPath 节	JavaExitsClasspath
客户机配置文件的 MessageBuffer 节	MaximumSize
客户机配置文件的 MessageBuffer 节	PurgeTime
客户机配置文件的 MessageBuffer 节	UpdatePercentage
客户机配置文件的 TCP 节	ClntRcvBuffSize
客户机配置文件的 TCP 节	ClntSndBuffSize
客户机配置文件的 TCP 节	Connect_Timeout
客户机配置文件的 TCP 节	KeepAlive

- 如果连接到需要转换字符数据的队列管理器，并且该队列管理器无法转换字符数据，那么 V7 Java 客户机现在能够执行转换。 客户机 JVM 必须支持客户机的 CCSID 和队列管理器的 CCSID 之间的转换。
- IBM MQ classes for Java 不支持客户机自动重新连接。

在客户机模式下使用时， *IBM MQ classes for Java* 不支持 MQBEGIN 调用。

IBM MQ classes for Java 绑定模式

IBM MQ classes for Java 的绑定模式与客户机模式在三个主要方面存在不同。

在绑定方式下使用时，IBM MQ classes for Java 会使用 Java 本机接口 (JNI) 直接调用现有的队列管理器 API，而不是通过网络进行通信。

缺省情况下，在绑定模式中使用 IBM MQ classes for Java 的应用程序使用 *ConnectOption* MQCNO_STANDARD_BINDINGS 连接到队列管理器。

IBM MQ classes for Java 支持以下 *ConnectOptions*：

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

有关 *ConnectOptions* 的更多信息，请参阅第 687 页的『使用 MQCONNX 调用连接到队列管理器』。

绑定模式支持在除 IBM MQ for IBM i 和 IBM MQ for z/OS 之外的所有平台上调用 MQBEGIN，以启动由队列管理器协调的全局工作单元。

大部分由 `MQEnvironment` 类提供的参数都与绑定模式无关，并且会被忽略。

定义要使用的 *IBM MQ classes for Java* 连接
要使用的连接类型通过设置 MQEnvironment 类中的变量来决定。

使用两个变量：

MQEnvironment.properties
连接类型由与键名称 CMQC.TRANSPORT_PROPERTY 相关联的值来决定。 可能的值如下所示：

CMQC.TRANSPORT_MQSERIES_BINDINGS
在绑定模式下连接

CMQC.TRANSPORT_MQSERIES_CLIENT
在客户机模式下连接

CMQC.TRANSPORT_MQSERIES
连接模式由 *hostname* 属性的值来决定

MQEnvironment.hostname
按照以下说明设置此变量的值：

- 对于客户机连接，请将此变量的值设置为要连接的 IBM MQ 服务器的主机名
- 对于绑定模式，请勿设置此变量，或将此变量设置为 Null

队列管理器上的操作

这组主题描述了如何使用 IBM MQ classes for Java 连接到队列管理器，以及如何与队列管理器断开连接。

为 *IBM MQ classes for Java* 设置 *IBM MQ* 环境
要让应用程序以客户机模式连接到队列管理器，该应用程序必须指定通道名称、主机名和端口号。

注: 仅当您的应用程序以客户机模式连接到队列管理器时，此主题中的信息才有用。 如果以绑定模式连接，该信息无用。 请参阅：第 86 页的『IBM MQ classes for JMS 的连接方式』

您可以通过以下两种方式之一来指定通道名称、主机名和端口号：以 MQEnvironment 类中的字段形式，或以 MQQueueManager 对象的属性形式。

如果您在 MQEnvironment 类中设置字段，这些字段会应用到整个应用程序中，被属性散列表覆盖的区域除外。 要在 MQEnvironment 中指定通道名称和主机名，请使用以下代码：

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

这相当于设置了一个 **MQSERVER** 环境变量：

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

缺省情况下， Java 客户机尝试连接到端口 1414 上的 IBM MQ 侦听器。 要指定不同端口，请使用以下代码：

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

其中，nnnn 是所需的端口号

如果您在创建队列管理器对象时将属性传递给该队列管理器对象，那么这些属性只应用于该队列管理器。 使用键 **hostname**、**channel** 和（可选）**port** 以及相应的值在 Hashtable 对象中创建条目。 要使用缺省端口 1414，您可以忽略 **port** 条目。 使用可接受属性散列表的构造函数创建 MQQueueManager 对象。

通过设置应用程序名称来确定与队列管理器的连接。

应用程序可以设置一个名称来确定与队列管理器的连接。 此应用程序名称由 **DISPLAY CONN MQSC/PCF** 命令显示（其中，字段名为 **APPLTAG**），或者在 IBM MQ Explorer“**应用程序连接**”显示屏幕中显示（其中，字段名为 **App name**）。

应用程序名称限制为 28 个字符，因此超过此长度的名称会被截断。 如果未指定应用程序名称，那么将提供缺省值。 缺省名称基于调用（主）类，但如果此信息不可用，就会使用文本 `IBM MQ Client for Java`。

如果使用调用类的名称，会通过移除前导包名称（根据需要）对该名称进行调整。 例如，如果调用类是 `com.example.MainApp`，将使用全名，但如果调用类是 `com.example.dictionaryAndThesaurus.multilingual.mainApp`，将使用名称 `multilingual.mainApp`，因为这是适合可用长度要求的类名称和最右侧包名称的最长组合。

如果类名称本身长度超过 28 个字符，就会被截断。 例如，`com.example.mainApplicationForSecondTestCase` 会被截断为 `mainApplicationForSecondTest`。

要在 MQEnvironment 类中设置应用程序名称，请通过以下代码使用键 **MQConstants.APPNAME_PROPERTY** 将该名称添加到 MQEnvironment.properties 散列表中：

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

要在传递到 MQQueueManager 构造函数的属性散列表中设置应用程序名称，请使用键 **MQConstants.APPNAME_PROPERTY** 将该名称添加到属性散列表中。

覆盖 IBM MQ 客户机配置文件中指定的属性

IBM MQ 客户机配置文件还可以指定用于配置 IBM MQ classes for Java 的属性。 但是，仅当应用程序以客户机方式连接到队列管理器时，IBM MQ MQI client 配置文件中指定的属性才适用。

如果需要，您可以通过以下任一方式，覆盖 IBM MQ 配置文件中的任何属性。 这些选项按照优先顺序显示。

- 为配置属性设置 Java 系统属性
- 在 MQEnvironment.properties 映射中设置此属性。
- 在 Java 5 及更高发行版上，设置系统环境变量。

IBM MQ 客户机配置文件中只有以下属性与 IBM MQ classes for Java 相关。 如果您指定或覆盖其他属性，那么将无效。

节	属性
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath
客户机配置文件的 ClientExitPath 节	ExitsDefaultPath64
客户机配置文件的 ClientExitPath 节	JavaExitsClasspath
客户机配置文件的 MessageBuffer 节	MaximumSize
客户机配置文件的 MessageBuffer 节	PurgeTime
客户机配置文件的 MessageBuffer 节	UpdatePercentage
客户机配置文件的 TCP 节	ClntRcvBufSize
客户机配置文件的 TCP 节	ClntSndBufSize
客户机配置文件的 TCP 节	Connect_Timeout
客户机配置文件的 TCP 节	KeepAlive

连接到 *IBM MQ classes for Java* 中的队列管理器
通过创建 MQQueueManager 类的新实例连接到队列管理器。 通过调用 disconnect() 方法与队列管理器断开连接。

您现在已经准备就绪，可以通过创建 MQQueueManager 类的新实例连接到队列管理器：

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

要与队列管理器断开连接，请在队列管理器上调用 disconnect() 方法：

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

如果您调用此 disconnect 方法，通过该队列管理器访问过的所有已打开的队列和进程都将关闭。 但是，良好的编程实践是在用完这些资源后将其明确关闭。 要执行此操作，请在相关对象上使用 close() 方法。

在队列管理器上使用 commit() 和 backout() 方法相当于在过程接口上调用 MQCMIT 和 MQBACK。

将客户机通道定义表用于 *IBM MQ classes for Java*

IBM MQ classes for Java 客户机应用程序可以使用存储在客户机通道定义表 (CCDT) 中的客户机连接通道定义。

可以通过在 MQEnvironment 类中设置某些字段和环境属性，或将这些字段和属性传递到属性散列表中的 MQQueueManager 来创建客户机连接通道定义，作为这种方法的备选方法，IBM MQ classes for Java 客户机应用程序可以使用存储在客户机通道定义表中的客户机连接通道定义。 这些定义是通过 IBM MQ Script (MQSC) 命令或 IBM MQ 可编程命令格式 (PCF) 命令，或者使用 IBM MQ Explorer 来创建的。

应用程序创建 MQQueueManager 对象时，IBM MQ classes for Java 客户机会搜索客户机通道定义表，寻找合适的客户机连接通道定义，并使用通道定义来启动 MQI 通道。 有关客户机通道定义表以及如何构造此表的更多信息，请参阅客户机通道定义表。

要使用客户机通道定义表，应用程序必须先创建一个 URL 对象。 此 URL 对象会封装一个统一资源定位符 (URL)，用于确定包含客户机通道定义表的文件的名称和位置，并指定可以如何访问此文件。

例如，如果文件 ccdt1.tab 包含客户机通道定义表，并且存储在用于运行应用程序的系统上，那么应用程序可以通过以下方式创建 URL 对象：

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

作为另一个示例，假设文件 ccdt2.tab 包含客户机通道定义表，并且存储在与用于运行应用程序的系统不同的系统上。 如果此文件可以使用 FTP 协议来访问，那么该应用程序可以通过以下方式创建 URL 对象：

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

在应用程序创建了 URL 对象后，该应用程序可以使用将 URL 对象用作参数的其中一个构造函数来创建一个 MQQueueManager 对象。 例如：

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

此语句可导致 IBM MQ classes for Java 客户机访问由 URL 对象 chanTab2 确定的客户机通道定义表，在此表中搜索合适的客户机连接通道定义，然后使用通道定义来启动到名为 MARS 的队列管理器的 MQI 通道。

如果应用程序使用客户机通道定义表，请注意以下几点适用：

- 在应用程序使用将 URL 对象用作参数的构造函数来创建 MQQueueManager 对象时，不得在 MQEnvironment 类中设置通道名称，无论是作为字段还是作为环境属性。 如果设置通道名称，IBM MQ classes for Java 客户机会抛出一个 MQException。 如果指定通道名称的字段或环境属性的值不为 Null、空字符串或包含的全部是空白字符的字符串，那么会将其视为已设置。
- MQQueueManager 构造函数上的 **queueManagerName** 参数可以包含以下值之一：
  - 队列管理器的名称
  - 星号 (*) 后跟队列管理器组的名称
  - 星号 (*)
  - Null、空字符串或包含的全部是空白字符的字符串

  这些值也可以用于 MQCONN 调用上的 **QMgrName** 参数，此调用由使用消息队列接口 (MQI) 的客户机应用程序发出。 有关这些值的含义的更多信息，请参阅第 673 页的『消息队列接口概述』。

  如果您的应用程序使用连接池，请参阅第 322 页的『在 IBM MQ classes for Java 中控制缺省连接池』。

- 当 IBM MQ classes for Java 客户机在客户机通道定义表中找到合适的客户机连接通道定义时，它只使用从此通道定义中提取的信息来启动 MQI 通道。 应用程序可能已经在 MQEnvironment 类中设置的任何通道相关字段或环境属性都会被忽略。

  如果您使用传输层安全性 (TLS)，尤其要注意以下几点：

  – 仅当从客户机通道定义表中提取的通道定义指定 IBM MQ classes for Java 客户机支持的 CipherSpec 名称时，MQI 通道才使用 TLS。
  – 客户机通道定义表还包含有关保存证书撤销列表 (CRL) 的轻量级目录访问协议 (LDAP) 服务器的位置信息。 IBM MQ classes for Java 客户机只使用此信息来访问保存 CRL 的 LDAP 服务器。
  – 客户机通道定义表还可以包含 OCSP 响应程序的位置。 IBM MQ classes for Java 不能使用客户机通道定义表文件中的 OCSP 信息。 但是，您可以按照使用联机证书协议部分中的说明来配置 OCSP

  有关将 TLS 与客户机通道定义表一起使用的更多信息，请参阅指定 MQI 通道使用 TLS。

  如果您使用通道出口，还要注意以下几点：

  – MQI 通道使用由从客户机通道定义表中提取的通道定义指定的通道出口和相关用户数据，这优先于使用其他方法指定的通道出口和数据。
  – 从客户机通道定义表中抽取的通道定义可以指定用 Java、C 或 C++ 编写的通道出口。有关如何使用 Java 编写通道出口的更多信息，请参阅第 317 页的『在 IBM MQ classes for Java 中创建通道出口』。有关如何使用其他语言编写通道出口的更多信息，请参阅第 320 页的『将未以 Java 编写的通道出口与 IBM MQ classes for Java 一起使用』。

指定 *IBM MQ classes for Java* 客户机连接的端口范围
您可以使用以下两种方式之一指定应用程序可以绑定的端口或端口范围。

当 IBM MQ classes for Java 应用程序尝试以客户机模式连接到 IBM MQ 队列管理器时，防火墙可能只允许从指定端口或端口范围发起的那些连接。 在此情况下，您可以指定应用程序可以绑定到的端口或端口范围。您可以通过以下方式指定端口：

- 您可以在 MQEnvironment 类中设置 localAddressSetting 字段。 例如：

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- 您可以设置环境属性 CMQC.LOCAL_ADDRESS_PROPERTY。 例如：

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- 当可以构建 MQQueueManager 对象时，您可以传递包含值为 "192.0.2.0(2000,3000)" 的 LOCAL_ADDRESS_PROPERTY 的属性散列表

在每个示例中，当应用程序稍后连接到队列管理器时，应用程序会绑定到范围在 192.0.2.0(2000) 到 192.0.2.0(3000) 之间的本地 IP 地址和端口号。

在带有多个网络接口的系统中，您还可以使用 localAddressSetting 字段，或者环境属性 CMQC.LOCAL_ADDRESS_PROPERTY 来指定必须将哪个网络接口用于连接。

如果限制端口范围，那么可能会发生连接错误。 如果出现错误，会抛出一个 MQException，其中包含 IBM MQ 原因码 MQRC_Q_MGR_NOT_AVAILABLE 以及以下消息：

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

如果指定范围内的所有端口都在使用，或者指定的 IP 地址、主机名或端口号无效（例如，负端口号），可能会出现错误。

访问 *IBM MQ classes for Java* 中的队列、主题和进程

要访问队列、主题和进程，请使用 MQQueueManager 类的方法。 MQOD（对象描述符结构）折叠在这些方法的参数中。

队列

要打开队列，您可以使用 MQQueueManager 类的 accessQueue 方法。 例如，在名为 queueManager 的队列管理器上，使用以下代码：

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

accessQueue 方法会返回类 MQQueue 的一个新对象。

使用完队列后，请使用 close() 方法将该队列关闭，如下所示：

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

您也可以使用 MQQueue 构造函数创建一个队列。 这些参数与用于 accessQueue 方法的参数完全相同，只是增加了一个队列管理器参数。 例如：

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

在创建队列时，您可以指定选项的数量。 有关详细信息，请参阅 Class.com.ibm.mq.MQQueue。 以这种方式构建队列对象时，您可以编写自己的 MQQueue 子类。

主题

同样，您可以使用 MQQueueManager 类的 accessTopic 方法打开主题。 例如，在一个名为 queueManager 的队列管理器上，可以使用以下代码创建一个订户和发布者：

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

使用完主题后，请使用 close() 方法将其关闭。

您还可以使用 MQTopic 构造函数来创建主题。 这些参数与用于 accessTopic 方法的参数完全相同，只是增加了一个队列管理器参数。 例如：

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

在创建主题时，您可以指定选项的数量。 有关详细信息，请参阅 Class com.ibm.mq.MQTopic。 以这种方式构造主题对象时，您可以编写自己的 MQTopic 子类。

必须为发布或预订打开一个主题。 MQQueueManager 类有 8 种 accessTopic 方法，Topic 类有 8 个构造函数。 在每种情况下，4 种方法有 **destination** 参数，4 种方法有 **subscriptionName** 参数（包括两种同时包含这两者的方法）。 这些方法只能用来为预订打开主题。 其余两种方法有一个 **openAs** 参数，可以根据 **openAs** 参数的值为发布或预订打开主题。

要作为持久订户创建主题，请使用 MQQueueManager 类的 accessTopic 方法或接受预订名称的 MQTopic 构造函数，无论哪一种情况，都要设置 CMQC.MQSO_DURABLE 选项。

进程

要访问进程，请使用 MQQueueManager 的 accessProcess 方法。 例如，在一个名为 queueManager 的队列管理器上，使用以下代码来创建一个 MQProcess 对象：

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

要访问进程，请使用 MQQueueManager 的 accessProcess 方法。

accessProcess 方法会返回类 MQProcess 的一个新对象。

使用完进程对象后，请使用 close() 方法将其关闭，如下所示：

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

您还可以通过使用 MQProcess 构造函数来创建进程。 这些参数与用于 accessProcess 方法的参数完全相同，只是增加了一个队列管理器参数。 例如：

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

以这种方式构造进程对象时，您可以编写自己的 MQProcess 子类。

处理 *IBM MQ classes for Java* 中的消息

消息由 MQMessage 类表示。 您可以使用 MQDestination 类的方法来放置和获取消息，该类带有 MQQueue 和 MQTopic 这两个子类。

可以使用 MQDestination 类的 put() 方法将消息放在队列或主题上。 您可以使用 MQDestination 类的 get() 方法从队列或主题中获取消息。 与 MQPUT 和 MQGET 放置和获取字节数组的过程接口不同，Java 编程语言放置和获取 MQMessage 类的实例。 MQMessage 类封装包含实际消息数据的数据缓冲区，以及描述该消息的所有 MQMD（消息描述符）参数和消息属性。

要构建新消息，请创建 MQMessage 类的一个新实例，并使用 writeXXX 方法将数据放入消息缓冲区中。

创建新的消息实例后，会自动将所有 MQMD 参数设置为其缺省值，如 MQMD 的初始值和语言声明中所定义。 MQDestination 的 put() 方法还会将 MQPutMessageOptions 类的实例作为参数。 该类表示 MQPMO 结构。 以下示例会创建一条消息并将其放入队列：

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage,pmo);
```

MQDestination 的 get() 方法将返回 MQMessage 的一个新实例，该实例表示刚刚从队列获取的消息。 它还会获取一个 MQGetMessageOptions 类实例作为参数。 该类表示 MQGMO 结构。

您无需指定最大消息大小，因为 get() 方法会自动调整其内部缓冲区的大小以容纳入局消息。 使用 MQMessage 类的 readXXX 方法来访问所返回消息中的数据。

以下示例显示如何从队列中获取消息：

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
```

```
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

通过设置 *encoding* 成员变量，您可以更改读、写方法使用的数字格式。

通过设置 *characterSet* 成员变量，您可以更改用于读、写字符串的字符集。

请参阅第 597 页的『MQMessage 类』，了解更多信息。

注: MQMessage 的 writeUTF() 方法会自动为字符串的长度以及其包含的 Unicode 字节编码。 当您的消息被另一个 Java 程序（使用 readUTF()）读取时，这是发送字符串信息的最简单方法。

改进 *IBM MQ classes for Java* 中非持久消息的性能
要改进在浏览消息或使用来自客户机应用程序的非持久消息时的性能，可以使用预读。 在浏览消息或使用非持久消息时，使用 MQGET 或异步消费的客户机应用程序将从性能提高中获益。

有关预读工具的常规信息，请参阅相关主题。

在 IBM MQ classes for Java 中，您可以使用 MQQueue 或 MQTopic 对象的 CMQC.MQSO_READ_AHEAD 和 CMQC.MQSO_NO_READ_AHEAD 属性来决定是否允许消息使用者和队列浏览器在该对象上使用预读。

使用 *IBM MQ classes for Java* 异步放置消息
要异步放置消息，请设置 MQPMO_ASYNC_RESPONSE。

您可以使用 MQDestination 类的 put() 方法将消息放在队列或主题上。 要异步放置消息，即，允许放置操作完成，而无需等待来自队列管理器的响应，您可以在 MQPutMessageOptions 的选项字段中设置 MQPMO_ASYNC_RESPONSE。 要确定异步放置操作是成功还是失败，请使用 MQQueueManager.getAsyncStatus 调用。

在 *IBM MQ classes for Java* 中发布/预订

在 IBM MQ classes for Java 中，主题由 MQTopic 类表示，您可以使用 MQTopic.put() 方法向其进行发布。

有关 IBM MQ 发布/预订的常规信息，请参阅发布/预订消息传递。

使用 *IBM MQ classes for Java* 处理 *IBM MQ* 消息头

提供了一些 Java 类，它们表示不同类型的消息头。 同时还提供了两个助手类。

MQHeader 接口

头对象由 MQHeader 接口描述，该接口提供了一些常规方法来访问头字段，以及读写消息内容。 每个头类型都有其自己的类，该类能够实现 MQHeader 接口，并为单个字段添加 getter 和 setter 方法。 例如，MQRFH2 头类型由 MQRFH2 类表示；MQDLH 头类型由 MQDLH 类表示等等。 头类会自动执行任何必要的数据转换，并且可以在任何指定的数字编码或字符集 (CCSID) 中读或写数据。

要点: MQRFH2 头类会将消息视为随机访问文件，这意味着必须将光标放在消息的起始位置。 在使用内部消息头类 (例如 MQRFH， MQRFH2， MQCIH， MQDEAD， MQIIH 或 MQXMIT) 之前，请确保在将消息传递到类之前将消息的光标位置更新为正确的位置。

助手类

两个助手类 MQHeaderIterator 和 MQHeaderList 可以帮助读取消息中的头内容，并对内容进行解码（解析）：

- MQHeaderIterator 类的工作方式与 java.util.Iterator 相似。 因为只要消息中有多个头，next() 方法就会返回 true，nextHeader() 或 next() 方法则会返回下一个头对象。

- MQHeaderList 的工作方式与 java.util.List 相似。 与 MQHeaderIterator 一样，它会解析头内容，但还允许您搜索特定头，添加新头，移除现有头，更新头字段，然后再将头内容写回到消息中。 或者，您也可以创建一个空 MQHeaderList，然后使用头实例进行填充，并一次或重复多次将其写入到消息中。

MQHeaderIterator 和 MQHeaderList 类使用 MQHeaderRegistry 中的信息来了解哪些 IBM MQ 头类与特定消息类型和格式相关。 MQHeaderRegistry 是使用所有当前 IBM MQ 格式和头类型及其实现类这些信息来配置的，您也可以注册自己的头类型。

支持以下常用 IBM MQ 头

- MQRFH - 规则和格式化头
- MQRFH2 - 与 MQRFH 一样，用来与属于 IBM Integration Bus 的消息代理之间传递消息。 同时还用于包含消息属性
- MQCIH - CICS 网桥
- MQDLH - 死信头
- MQIIH - IMS 信息头
- MQRMH - 参考消息头
- MQSAPH - SAP 头
- MQWIH - 工作信息头
- MQXQH - 传输队列头
- MQDH - 分发头
- MQEPH - 已封装的 PCF 头

您还可以定义表示自己的头的类。

要使用 MQHeaderIterator 来获取 RFH2 头，可以在 GetMessageOptions 中设置 MQGMO_PROPERTIES_FORCE_MQRFH2，也可以将队列属性 PROPCTL 设置为 FORCE。

使用 *IBM MQ classes for Java* 打印消息中的所有头
在此示例中，MQHeaderIterator 的实例会解析从队列中获取的 MQMessage 中的头。 在 nextHeader() 方法返回 MQHeader 对象后，如果调用这些对象的 toString 方法，那么将显示它们的结构和内容。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

使用 *IBM MQ classes for Java* 跳过消息中的头
在本示例中，MQHeaderIterator 的 skipHeaders() 方法会将消息读取游标置于紧随最后一个头之后的位置。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

使用 *IBM MQ classes for Java* 查找死信消息中的原因码
在此示例中，读取方法会通过从消息中读取内容来填充 MQDLH 对象。 读取操作完成之后，会将消息读取光标直接置于 MQDLH 头内容之后。

队列管理器的死信队列上的消息带有死信消息头 (MQDLH) 作为前缀。 要决定如何处理这些消息，例如，决定是重试还是放弃这些消息，死信处理应用程序必须查看包含在 MQDLH 中的原因码。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

所有头类还提供了一个方便的构造函数，只需一个步骤即可直接从消息中初始化这些头类本身。 所以，可以按照以下方法简化此示例中的代码：

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

使用 *IBM MQ classes for Java* 从死信消息中读取和移除头
在此示例中，MQDLH 用于从死信消息中移除头。

如果被拒绝消息的原因码表示存在瞬时错误，死信处理应用程序通常会重新提交消息。 在重新提交消息之前，它必须移除 MQDLH 头。

此示例将执行以下步骤（请参阅示例代码中的注释）：

1. MQHeaderList 读取整条消息，并且在消息中遇到的每个头都成为列表中的项。
2. 死信消息中包含一个 MQDLH 作为其第一个头，因此，这可以在头列表的第一项中找到。 MQDLH 已在构建 MQHeaderList 时通过消息进行了填充，因此，无需调用其读取方法。
3. 使用 MQDLH 类提供的 getReason() 方法提取原因码。
4. 已对原因码进行检查，表明适合重新提交此消息。 将使用 MQHeaderList remove() 方法来除去 MQDLH。
5. MQHeaderList 会将其剩余的内容写入新消息对象中。 新消息中现在包含原始消息中除 MQDLH 之外的所有内容，并且可以写入到队列中。 构造函数和写入方法的 **true** 自变量表示消息体将保留在 MQHeaderList 中，并且将再次写出。
6. 新消息的消息描述符中的格式字段现在包含之前位于 MQDLH 格式字段中的值。 消息数据与消息描述符中设置的数字编码和 CCSID 相匹配。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

使用 *IBM MQ classes for Java* 打印消息内容
此示例使用 MQHeaderList 打印出消息内容，包括其头。

输出中包含所有头内容和消息体的视图。 MQHeaderList 类会一次性对所有头进行解码，而 MQHeaderIterator 则会在应用程序控件下一次对一个头进行单步调试。 您可以使用此方法在编写 WebSphere MQ 应用程序时提供一个简单的调试工具。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

此示例还将使用 MQMD 类打印出消息描述符字段。 com.ibm.mq.headers.MQMD 类的 copyFrom() 方法会通过 MQMessage 的消息描述符字段填充头对象，而不是通过读取消息体。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

使用 *IBM MQ classes for Java* 在消息中查找特定类型的头
本示例使用 MQHeaderList 的 indexOf(String) 方法在消息中查找 MQRFH2 头（如果存在）。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
        ...
}
```

使用 *IBM MQ classes for Java* 分析 *MQRFH2* 头
此示例说明了如何使用 MQRFH2 类访问命名文件夹中的已知字段值。

MQRFH2 类提供了一系列方法，不仅能够访问结构的固定部分中的字段，还能访问 NameValueData 字段中包含的由 XML 编码的文件夹内容。 此示例说明了如何访问命名文件夹中的已知字段值，在此实例中，为 jms 文件夹中的 Rto 字段，它表示 MQ JMS 消息中的应答队列名称。

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

要发现 MQRFH2 中的内容（而不是直接请求特定字段），您可以使用 getFolders 方法返回 MQRFH2.Element 列表，后者表示可以包含字段和其他文件夹的文件夹结构。 将字段或文件夹设置为 Null 可将其从 MQRFH2 中移除。 以此方法操作 NameValueData 文件夹内容时，会相应地自动更新 StrucLength 字段。

使用 *IBM MQ classes for Java* 读写字节流而不是 *MQMessage* 对象
当数据源不是 MQMessage 对象时，以下示例使用头类来解析和处理 IBM MQ 头内容。

即使数据源不是 MQMessage 对象，您也可以使用头类来解析和处理 IBM MQ 头内容。 由每个头类实现的 MQHeader 接口提供了方法 `int read (java.io.DataInput message, int encoding, int characterSet)` 和 `int write (java.io.DataOutput message, int encoding, int characterSet)`。 com.ibm.mq.MQMessage 类实现了 java.io.DataInput 和 java.io.DataOutput 接口。 这

意味着您可以使用两种 MQHeader 方法读写 MQMessage 内容，从而覆盖在消息描述符中指定的编码和 CCSID。 这对于包含一系列使用不同编码的头的消息非常有用。

您也可以从其他数据流中获取 DataInput 和 DataOutput 对象，例如，文件或套接字流，或者 JMS 消息中包含的字节数组。 java.io.DataInputStream 类实现了 DataInput，java.io.DataOutputStream 类实现了 DataOutput。 以下示例从字节数组中读取 IBM MQ 头内容：

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

以 MQHeaderIterator 开头的行可以替换为

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

以下示例使用 DataOutputStream 将内容写入字节数组：

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

以这种方式使用流时，请注意要使用正确的编码和 characterSet 自变量值。 读取头时，请指定最初编写字节内容时使用的编码和 CCSID。 编写头时，请指定您要生成的编码和 CCSID。 头类会自动执行数据转换。

使用 *IBM MQ classes for Java* 为新头类型创建类

您可以为未随 IBM MQ classes for Java 提供的头类型创建 Java 类。

要添加一个表示新头类型的 Java 类（您可以使用与 IBM MQ classes for Java 提供的任何头类相同的方式使用新头类型），您可以创建一个实现 MQHeader 接口的类。 这个非常简单的方法将会扩展 com.ibm.mq.headers.impl.Header 类。 以下示例将生成一个表示 MQTM 头结构的全功能类。 您不必为每个字段添加单独的 getter 和 setter 方法，但这对于头类用户却很方便有用。 常规的使用字符串作为字段名称的 getValue 和 setValue 方法将适用于在头类型中定义的所有字段。 所继承的读写和大小方法使新头类型的实例能够被读写，并将根据其字段定义正确地计算头大小。 虽然类型定义只创建一次，但是却创建了这个头类的多个实例。 要使新头定义能够使用 MQHeaderIterator 或 MQHeaderList 类进行解码，您需要使用 MQHeaderRegistry 进行注册。 不过要注意，MQTM 头类实际上已经在此包中提供，并在缺省注册表中注册。

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);

    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
```

```
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

使用 *IBM MQ classes for Java* 处理 *PCF* 消息

提供了一些 Java 类，用于创建和解析 PCF 结构的消息，并帮助发送 PCF 请求和收集 PCF 响应。

类 PCFMessage 和 MQCFGR 表示 PCF 参数结构数组。 它们提供了一些便捷的方法来添加和检索 PCF 参数。

PCF 参数结构由类 MQCFH、MQCFIN、MQCFIN64、MQCFST、MQCFBS、MQCFIL、MQCFIL64 MQCFSL 和 MQCFGR 表示。 这些类共享基本的操作接口：

- 读写消息内容的方法：read ()、write () 和 size ()
- 处理参数的方法：getValue ()、setValue ()、getParameter () 以及其他方法
- 枚举符方法 .nextParameter ()，该方法可解析 MQMessage 中的 PCF 内容

PCF 过滤器参数在查询命令中用于提供过滤功能。 它被封装在以下类中：

- MQCFIF - 整数过滤器
- MQCFSF - 字符串过滤器
- MQCFBF - 字节过滤器

提供两个代理类 PCFAgent 和 PCFMessageAgent 来管理与队列管理器、命令服务器队列和相关响应队列的连接。 PCFMessageAgent 可扩展 PCFAgent，通常情况下应优先使用。 PCFMessageAgent 类可转换所接收到的 MQMessages，并将它们作为 PCFMessage 数组传回给调用者。 PCFAgent 会返回一个 MQMessages 数组，您需要先对其进行解析，然后才能使用。

处理 *IBM MQ classes for Java* 中的消息属性

IBM MQ classes for Java 中没有等同的处理消息句柄的函数调用方法。 要设置、返回或删除消息句柄属性，请使用 MQMessage 类的方法。

有关消息属性的常规信息，请参阅第 22 页的『属性名称』。

在 IBM MQ classes for Java 中，访问消息是通过 MQMessage 类实现的。 因此，在 Java 环境中没有提供消息句柄，也就没有与 IBM MQ 函数调用 MQCRTMH、MQDLTMH、MQMHBUF 和 MQBUFMH 等同的函数

要在过程接口中设置消息句柄属性，您可以使用调用 MQSETMP。 在 IBM MQ classes for Java 中，使用 MQMessage 类的适当方法：

- setBooleanProperty
- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property
- setInt8Property
- setLongProperty
- setFloatProperty
- setDoubleProperty

- setStringProperty
- setObjectProperty

这些属性有时统称为 *set*property* 方法。

要在过程接口中返回消息句柄属性的值，您可以使用调用 MQINQMP。 在 IBM MQ classes for Java 中，使用 MQMessage 类的适当方法：

- getBooleanProperty
- getByteProperty
- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty
- getDoubleProperty
- getStringProperty
- getObjectProperty

这些属性有时统称为 *get*property* 方法。

要删除过程接口中的消息句柄属性的值，您可以使用调用 MQDLTMP。 在 IBM MQ classes for Java 中，使用 MQMessage 类的 deleteProperty 方法。

处理 *IBM MQ classes for Java* 中的错误

使用 Java `try` 和 `catch` 块处理 IBM MQ classes for Java 引起的错误。

Java 接口中的方法不会返回完成代码和原因码。 而是每当 IBM MQ 调用产生的完成代码和原因码都不为零时抛出异常。 这样简化了程序逻辑，您不必在每次调用 IBM MQ 之后都检查返回码。 您可以决定希望在程序中的哪些位置处理可能出现的故障。 您可以在这些位置使用 `try` 和 `catch` 块将代码包围起来，如下例中所示：

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                           "CC = " + ex.completionCode +
                           "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

z/OS 的 Java 异常中报告的 IBM MQ 调用原因码记录在 API 完成代码和原因码中。

在运行 IBM MQ classes for Java 应用程序时抛出的异常也写入此日志中。 不过，应用程序可以通过调用 MQException.logExclude() 方法来阻止记录与特定原因码相关的异常。 如果您希望抛出很多与特定原因码相关的异常，但不希望日志中出现这些异常，可以使用这种方法。 例如，如果应用程序在每次迭代循环时都尝试从队列中获取消息，并且对于其中大部分尝试，您都预计没有要放入队列的合适消息，那么可能会希望阻止记录与原因码 MQRC_NO_MSG_AVAILABLE 相关的异常。 如果应用程序以前阻止记录与特定原因码相关的异常，它可以通过调用方法 MQException.logInclude() 允许再次记录这些异常。

有时候，原因码不会传达所有与错误有关的详细信息。 对于所抛出的每个异常，应用程序都应检查所链接的异常。 链接异常本身可能会包含另一个链接异常，因此，链接异常会形成一个指回原始底层问题的连锁链。

链接异常通过使用 java.lang.Throwable 类的连锁异常机制实现，而应用程序通过调用 Throwable.getCause() 方法获取链接异常。 MQException.getCause() 可以从作为 MQException 实例的异常中检索底层的 com.ibm.mq.jmqi.JmqiException 实例，getCause 可以从此异常中检索导致此错误的底层 java.lang.Exception。

在 *IBM MQ classes for Java* 中获取并设置属性值

我们为多种常见属性提供了 getXXX() 和 setXXX() 方法。 其他属性则可以使用通用的 inquire() 和 set() 方法来访问。

对于多种常见属性，类 MQManagedObject、MQDestination、MQQueue、MQTopic、MQProcess 和 MQQueueManager 包含 getXXX() 和 setXXX() 方法。 您可以通过这些方法获取并设置其属性值。 请注意，对于 MQDestination、 MQQueue 和 MQTopic，仅在打开对象时指定了适当的查询和设置标志时，这些方法才起作用。

对于不太常见的属性，MQQueueManager、MQDestination、MQQueue、MQTopic 和 MQProcess 类都继承自名为 MQManagedObject 的类。 此类将定义 inquire() 和 set() 接口。

使用 *new* 操作符创建新队列管理器对象时，它将自动打开以供查询。 使用 accessProcess() 方法访问进程对象时，该对象将自动打开以供查询。 使用 accessQueue() 方法访问队列对象时，该对象不会自动打开以供查询或设置操作。 这是因为自动添加这些选项可能会导致某些类型的远程队列出现问题。 要针对队列使用查询、设置、getXXX 和 setXXX 方法，您必须在 accessQueue() 方法的 openOptions 参数中指定相应的查询和设置标志。 目标和主题对象同样如此。

查询和设置方法接受三种参数：

- selectors 数组
- intAttrs 数组
- charAttrs 数组

您不需要在 MQINQ 中找到的 SelectorCount、IntAttrCount 和 CharAttrLength 参数，因为 Java 中数组的长度始终是已知的。 以下示例显示了如何查询队列：

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

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

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

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Java 中的多线程程序

Java 运行时环境是固有的多线程环境。 IBM MQ classes for Java 允许多个线程共享队列管理器对象，但会确保所有对目标队列管理器的访问都是同步的。

在 Java 中，多线程程序难以避免。 考虑一个能连接到队列管理器并在启动时可以打开队列的简单程序。 该程序在屏幕上显示一个按钮。 用户单击该按钮时，程序将从队列中访存消息。

Java 运行时环境是固有的多线程环境。 因此，应用程序初始化发生在一个线程中，按下按钮后执行的代码则会在独立线程（用户界面线程）中执行。

使用基于 C 的 IBM MQ MQI client 时，这将引发问题，因为通过多个线程共享句柄将存在限制。 IBM MQ classes for Java 放宽了这种约束，允许通过多个线程共享队列管理器对象（及其相关联的队列、主题和进程对象）。

实施 IBM MQ classes for Java 可以确保对于特定连接（MQQueueManager 对象实例），对目标 IBM MQ 队列管理器的所有访问都是同步的。 将会阻止要向队列管理器发出调用的线程，直到针对该连接的所有进行中

的调用都完成为止。 如果您需要在应用程序中通过多个线程同时访问同一个队列管理器，请针对需要并行访问的每个线程创建新的 MQQueueManager 对象。 （这等同于对每个线程发出一个单独的 MQCONN 调用。）

注: 不得在同时请求消息的线程间共享类 `com.ibm.mq.MQGetMessageOptions` 的实例。 此类的实例将在相应的 MQGET 请求期间通过数据进行更新，如果多个线程在同一个对象实例上同时运行，这可能导致意外后果。

在 *IBM MQ classes for Java* 中使用通道出口

有关如何通过 IBM MQ classes for Java 在应用程序中使用通道出口的概述。

以下主题描述了如何使用 Java 编写通道出口，如何分配该通道出口，以及如何向其传递数据。 之后描述了如何使用通过 C 语言编写的通道出口以及如何使用通道出口序列。

要装入通道出口类，您的应用程序必须具有正确的安全许可权。

在 *IBM MQ classes for Java* 中创建通道出口

您可以通过定义可实施相应接口的 Java 类，提供自己的通道出口。

要实现一个出口，您应定义可实现相应接口的新 Java 类。 com.ibm.mq.exits 包中定义了三个出口接口：

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

注: 仅客户机连接支持通道出口；绑定连接不支持。 不能在 IBM MQ classes for Java 外部使用 Java 通道出口，例如，如果您正在使用以 C 编写的客户机应用程序。

任何针对连接定义的 TLS 加密都将在调用了发送和安全出口之后执行。 同样，解密将在调用了接收和安全出口之前执行。

以下样本定义了实现所有三个接口的类：

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                  MQCD channelDefinition,
                                  ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                     MQCD channelDefinition,
                                     ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                      MQCD channelDefinition,
                                      ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

每个出口都传递了一个 MQCXP 对象和一个 MQCD 对象。 这些对象表示在过程接口中定义的 MQCXP 和 MQCD 结构。

您所编写的任何出口类都必须具备构造函数。 这可以是缺省构造函数，也可以是采用字符串自变量的构造函数。 如果采用字符串，那么用户数据将在创建出口类后传递到该出口类。 如果出口类同时包含缺省构造函数和单个自变量构造函数，那么单个自变量构造函数将优先。

对于发送和安全出口，您的退出代码必须返回要发送到服务器的数据。 对于接收出口，您的退出代码必须返回想要 IBM MQ 解释的修改后数据。

最简单的出口代码是：

```
{ return agentBuffer; }
```

请勿从通道出口内部关闭队列管理器。

使用现有的通道出口类

在 7.0 之前的 IBM MQ 版本中，您可以使用接口 MQSendExit、MQReceiveExit 和 MQSecurityExit 实现这些出口，如以下示例中所示。 此方法仍然有效，但为了改善功能和性能，将首选采用新方法。

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

在 *IBM MQ classes for Java* 中分配通道出口
您可以使用 IBM MQ classes for Java 分配通道出口。

IBM MQ classes for Java 中没有与 IBM MQ 通道直接等效的通道。 通道出口将被分配给 MQQueueManager。 例如，定义了实现 WMQSecurityExit 接口的类之后，应用程序可以通过以下四种方式之一使用安全出口：

- 通过向 MQEnvironment.channelSecurityExit 字段分配类实例，然后创建 MQQueueManager 对象
- 通过将 MQEnvironment.channelSecurityExit 字段设置为表示安全出口类的字符串，然后创建 MQQueueManager 对象
- 通过在传递给 MQQueueManager 的属性散列表中，使用键 CMQC.SECURITY_EXIT_PROPERTY 创建键/值对
- 使用客户机通道定义表 (CCDT)

通过将 MQEnvironment.channelSecurityExit 字段设置为字符串、在属性散列表中创建键/值对或使用 CCDT 分配的任何出口，都必须使用缺省构造函数来编写。 作为类实例分配的出口不需要缺省构造函数，具体取决于应用程序。

应用程序可以通过相似的方法使用发送或接收出口。 例如，以下代码片段为您展示了如何通过 MQEnvironment 使用在类 MyMQExits（之前定义的）中实现的安全、发送和接收出口：

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
```

```
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

如果使用了多种方法分配通道出口，那么优先顺序如下所示：

1. 如果将 CCDT 的 URL 传递给 MQQueueManager，那么 CCDT 的内容将确定要使用的通道出口，并且将忽略 MQEnvironment 或属性散列表中的任何出口定义。
2. 如果没有传递 CCDT URL，那么将合并 MQEnvironment 和散列表中的出口定义。
   - 如果在 MQEnvironment 和散列表中定义了相同的出口类型，那么将使用散列表中的定义。
   - 如果指定了等效的旧出口类型和新出口类型（例如 sendExit 字段，它只能用于 IBM WebSphere MQ 7.0 之前版本中使用的出口类型，而 channelSendExit 字段则可用于任何发送出口），那么将使用新出口 (channelSendExit) 而不是旧出口。

如果您已经将通道出口声明为字符串，那么必须启用 IBM MQ 来找到通道出口程序。 您可以通过多种方法执行此操作，具体取决于应用程序所运行的环境以及通道出口程序的打包方式。

- 对于在应用程序服务器中运行的应用程序，您必须将文件存储在 第 319 页的表 57 中显示的目录中，或者打包为由 **exitClasspath** 引用的 JAR 文件。
- 对于不在应用程序服务器中运行的应用程序，以下规则适用：
  - 如果您的通道出口类被打包在独立的 JAR 文件中，这些 JAR 文件必须包含在 **exitClasspath** 中。
  - 如果您的通道出口类没有打包在 JAR 文件中，这些类文件可以存储在第 319 页的表 57 中显示的目录中，或者 JVM 系统类路径或 **exitClasspath** 中的任何目录中。

可以通过四种方法指定 **exitClasspath** 属性。 这些方法按优先级顺序显示如下：

1. 系统属性 com.ibm.mq.exitClasspath（使用 -D 选项在命令行中定义）
2. mqclient.ini 文件的 exitPath 节
3. 键为 CMQC.EXIT_CLASSPATH_PROPERTY 的散列表条目
4. MQEnvironment 变量 **exitClasspath**

多个路径将使用 java.io.File.pathSeparator 字符分隔。

表 *57:* 通道出口程序的目录

平台	目录
AIX AIX Linux Linux Solaris Solaris	/var/mqm/exits（32 位通道出口程序） /var/mqm/exits64（64 位通道出口程序）
Windows	*install_data_dir*\exits

注: *install_data_dir* 是安装期间为 IBM MQ 数据文件选择的目录。 缺省目录为 C:\ProgramData\IBM\MQ。

在 *IBM MQ classes for Java* 中将数据传递到通道出口
您可以将数据传递到通道出口，也可以使通道出口中的数据返回到应用程序。

agentBuffer 参数

对于发送出口，*agentBuffer* 参数包含待发送的数据。 对于接收出口或安全出口，*agentBuffer* 参数包含刚刚接收到的数据。 您无需 length 参数，因为表达式 agentBuffer.limit() 指示了数组长度。

对于发送和安全出口，您的退出代码必须返回要发送到服务器的数据。 对于接收出口，您的退出代码必须返回想要 IBM MQ 解释的修改后数据。

最简单的出口代码是：

```
{ return agentBuffer; }
```

通道出口使用具有后备数组的缓冲区调用。 要获取最佳性能，出口必须返回具有后备数组的缓冲区。

用户数据

如果应用程序通过设置 channelSecurityExit、channelSendExit 或 channelReceiveExit 连接到队列管理器，那么可以使用 channelSecurityExitUserData、channelSendExitUserData 或 channelReceiveExitUserData 字段在调用相应通道出口类时将 32 字节用户数据传递到该通道出口类。 此用户数据可用于通道出口类，但每次调用出口时都会被刷新。 因此，对通道出口中的用户数据所做的任何更改都会丢失。 如果您想对通道出口中的数据执行永久性更改，请使用 MQCXP exitUserArea。 此字段中的数据将在调用出口的间隙得到维护。

如果应用程序设置 securityExit、sendExit 或 receiveExit，那么不会向这些通道出口类传递任何用户数据。

如果应用程序使用客户机通道定义表 (CCDT) 来连接至队列管理器，那么在客户机连接通道定义中指定的任何用户数据都将在调用通道出口类时传递到这些通道出口类。 有关客户机通道定义表的更多信息，请参阅第 305 页的『将客户机通道定义表用于 IBM MQ classes for Java』。

将未以 *Java* 编写的通道出口与 *IBM MQ classes for Java* 一起使用

如何通过 Java 应用程序使用以 C 语言编写的通道出口程序。

在 IBM WebSphere MQ 7.0 中，您可以指定以 C 语言编写的通道出口程序的名称，作为字符串传递到 MQEnvironment 对象或属性散列表中的 channelSecurityExit、channelSendExit 或 channelReceiveExit 字段。 然而，您无法在以其他语言编写的应用程序中使用以 Java 编写的通道出口。

以 `library(function)` 格式指定出口程序名称，并确保按出口路径中所述指定出口程序的位置。

有关如何使用 C 语言编写通道出口的信息，请参阅第 875 页的『消息传递通道的通道出口程序』。

使用外部出口类

在 IBM WebSphere MQ 7.0 之前的版本中，提供了三个类，使您能够使用以除了 Java 之外的语言编写的通道出口：

- MQExternalSecurityExit，实现 MQSecurityExit 接口
- MQExternalSendExit，实现 MQSendExit 接口
- MQExternalReceiveExit，实现 MQReceiveExit 接口

这些类依然可供使用，但将首选采用新方法。

要使用未以 Java 编写的安全出口，应用程序首先应创建 MQExternalSecurityExit 对象。 作为 MQExternalSecurityExit 构造函数上的参数，该应用程序指定了包含安全出口的库的名称、安全出口的入口点名称，以及在调用安全出口时将传递到该安全出口的用户数据。 未以 Java 编写的通道出口程序存储在第 319 页的表 57 中显示的目录中。

在 *IBM MQ classes for Java* 中使用通道发送或接收出口序列

IBM MQ classes for Java 应用程序可以使用连续运行的一系列通道发送出口或接收出口。

为使用发送出口序列，应用程序可以创建包含发送出口的列表或字符串。 如果使用了列表，那么列表中的每个元素都可以是以下任意项：

- 实现 WMQSendExit 接口的用户定义类的实例
- 实现 MQSendExit 接口的用户定义类的实例（针对以 Java 编写的发送出口）
- MQExternalSendExit 类实例（针对未以 Java 编写的发送出口）
- MQSendExitChain 类实例
- 字符串类实例

列表无法包含另一个列表。

应用程序可以通过类似的方式使用接收出口序列。

如果使用了字符串，该字符串必须包含一个或多个以逗号分隔的出口定义，每个定义都可以是 Java 类或 `library(function)` 格式的 C 语言程序的名称。

应用程序之后会向 MQEnvironment.channelSendExit 字段分配列表或字符串对象，然后创建 MQQueueManager 对象。

传递给出口的信息上下文仅存在于出口域中。 例如，如果 Java 出口和 C 出口相链接，那么 Java 出口的存在对 C 出口没有影响。

使用出口链类

在 IBM WebSphere MQ 7.0 之前的版本中，提供了允许出口序列的两个类：

- MQSendExitChain，实现 MQSendExit 接口
- MQReceiveExitChain，实现 MQReceiveExit 接口

这些类依然可供使用，但将首选采用新方法。 使用 IBM MQ Classes for Java 意味着应用程序仍依赖于 `com.ibm.mq.jar`。如果使用了 `com.ibm.mq.exits` 包中的新接口集，那么不依赖于 `com.ibm.mq.jar`。

为使用发送出口序列，应用程序创建了对象列表，其中每个对象都是以下项之一：

- 实现 MQSendExit 接口的用户定义类的实例（针对以 Java 编写的发送出口）
- MQExternalSendExit 类实例（针对未以 Java 编写的发送出口）
- MQSendExitChain 类实例

应用程序通过将此对象列表作为构造函数上的参数传递，创建了 MQSendExitChain 对象。 应用程序之后会向 MQEnvironment.sendExit 字段分配 MQSendExitChain 对象，然后创建 MQQueueManager 对象。

IBM MQ classes for Java 中的通道压缩

压缩通道上流动的数据可以提高通道性能并降低网络流量。 IBM MQ classes for Java 使用 IBM MQ 中内置的压缩功能。

使用随 IBM MQ 提供的功能，您可以压缩消息通道和 MQI 通道上流动的数据，对于每种通道类型，您都能独立压缩头数据和消息数据。 缺省情况下，通道上的数据都不会压缩。 要了解关于通道压缩的完整说明，包括在 IBM MQ 中的实现方式，请参阅数据压缩 (COMPMSG) 和头压缩 (COMPHDR)。

IBM MQ classes for Java 应用程序会通过创建 java.util.Collection 对象，指定用于在客户机连接时压缩头或消息数据的方法。 每种压缩方法都是集合中的整数对象，且应用程序向集合添加压缩方法的顺序，就是压缩方法在客户机连接开始时与队列管理器协商的顺序。 应用程序之后可以在 MQEnvironment 类中向 hdrCompList 字段（对于头数据）或 msgCompList 字段（对于消息数据）分配集合。 应用程序准备就绪后，它可以通过创建 MQQueueManager 对象来启动客户机连接。

以下代码片段展示了所述方法。 第一个代码片段显示了如何执行头数据压缩：

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

第二个代码片段显示了如何执行消息数据压缩：

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));
:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

在第二个示例中，当启动客户机连接时，会按先 RLE 后 ZLIBHIGH 的顺序协商压缩方法。 在 MQQueueManager 对象的生命周期内，选择的压缩方法无法更改。

在客户机连接中，客户机和队列管理器都支持的头和消息数据的压缩方法将被传递到通道出口，作为 MQChannelDefinition 对象的 hdrCompList 和 msgCompList 字段中的集合。 目前在客户机连接上用于压缩头和消息数据的实际方法，将被传递到 MQChannelExit 对象的 CurHdrCompression 和 CurMsgCompression 字段中的通道出口。

如果在客户机连接上使用压缩，那么数据将在处理和提取了任何通道发送出口之前且在处理了任何通道接收出口之后得到压缩。 因此，传递到发送和接收出口的数据都处于压缩状态。

有关指定压缩方法以及可用压缩方法的更多信息，请参阅类 com.ibm.mq.MQEnvironment 和接口 com.ibm.mq.MQC。

在 *IBM MQ classes for Java* 中共享 *TCP/IP* 连接

可以让 MQI 通道的多个实例共享一个 TCP/IP 连接。

在 IBM MQ classes for Java 中，使用 MQEnvironment.sharingConversations 变量来控制可共享一个 TCP/IP 连接的对话数量。

SHARECNV 属性是尽力而为的连接共享方法。 因此，当大于 0 的 SHARECNV 值与 IBM MQ classes for Java 一起使用时，则不能保证新的连接请求始终共享已经建立的连接。

IBM MQ classes for Java 中的连接池

IBM MQ classes for Java 允许将多余的连接聚集为池，以便复用。

IBM MQ classes for Java 为处理与 IBM MQ 队列管理器的多个连接的应用程序提供了其他支持。 当连接不再需要时，该连接不会被破坏，而是被聚集到池中，以便以后复用。 对于串行连接到任意队列管理器的应用程序和中间件，这可以大幅提升性能。

IBM MQ 提供缺省连接池。 应用程序可以通过 MQEnvironment 类注册和注销令牌，从而激活或取消激活此连接池。 如果在 IBM MQ classes for Java 构造 MQQueueManager 对象时，该池为激活状态，那么它将搜索此缺省池并复用任何适用的连接。 发生 MQQueueManager.disconnect() 调用时，基础连接将返回到池中。

或者，应用程序可以针对特定用途构造 MQSimpleConnectionManager 连接池。 之后，应用程序可以在构造 MQQueueManager 对象期间指定该池，也可以将该池传递到 MQEnvironment，以作为缺省连接池使用。

为了防止连接使用太多资源，您可以限制 MQSimpleConnectionManager 对象可以处理的连接总数，也可以限制连接池的大小。 如果 JVM 中的连接需求存在冲突，那么设置限制就非常有用。

缺省情况下，getMaxConnections() 方法将返回值“0”，这意味着 MQSimpleConnectionManager 对象可以处理的连接数量没有限制。 您可以使用 setMaxConnections() 方法设置限制。 如果设置了限制，那么在达到该限制后，进一步连接请求可能会导致抛出 MQException，并出现原因码 MQRC_MAX_CONNS_LIMIT_REACHED。

在 *IBM MQ classes for Java* 中控制缺省连接池

此示例显示了如何使用缺省连接池。

请考虑以下示例应用程序 MQApp1：

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

MQApp1 从命令行获取本地队列管理器列表，依次连接到每个本地队列管理器，然后执行某些操作。 然而，如果命令行多次列出同一个队列管理器，那么更有效的方法是仅连接一次，然后多次复用该连接。

IBM MQ classes for Java 提供了缺省连接池，可用于执行此操作。 要启用池，请使用 MQEnvironment.addConnectionPoolToken() 方法之一。 要禁用池，请使用 MQEnvironment.removeConnectionPoolToken()。

以下示例应用程序 MQApp2 的功能与 MQApp1 相同，但仅连接到每个队列管理器一次。

```
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);

      }
}
```

第一个粗体行通过向 MQEnvironment 注册 MQPoolToken 对象来激活缺省连接池。

MQQueueManager 构造函数现在将在该池中搜索相应的连接，如果无法找到现有连接，将仅创建一个指向队列管理器的连接。 qmgr.disconnect() 调用会将连接返回到池中，以便以后复用。 这些 API 调用与样本应用程序 MQApp1 相同。

第二个突出显示的行将取消激活缺省连接池，此操作将破坏池中存储的任何队列管理器连接。 这一操作非常重要，因为如果不这么做的话，应用程序将会因为池中的众多活动队列管理器连接而终止。 此情况可能会导致出现在队列管理器日志中的错误。

如果应用程序使用客户机通道定义表 (CCDT) 来连接至队列管理器，那么 MQQueueManager 构造函数首先会在表中搜索合适的客户机连接通道定义。 如果找到一个定义，那么构造函数会在缺省连接池中搜索可用于通道的连接。 如果构造函数无法在池中找到合适的连接，那么会在客户机通道定义表中搜索下一个合适的客户机连接通道定义，并按照前述方法继续。 如果构造函数完成了客户机通道定义表搜索且无法在池中找到任何合适的连接，那么构造函数会启动对表的第二次搜索。 在此搜索期间，构造函数会尝试依次为每个合适的客户机连接通道定义创建新连接，然后使用它设法创建的第一个连接。

缺省连接池最多可存储十个未使用的连接，且未使用的连接最多可保持活动状态五分钟。 应用程序可以更改此设置（有关详细信息，请参阅第 324 页的『在 IBM MQ classes for Java 中提供不同的连接池』）。

应用程序不会使用 MQEnvironment 来提供 MQPoolToken，而是会自行构造：

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

某些应用程序或中间件供应商会提供 MQPoolToken 子类，从而向定制连接池传递信息。 它们可以采用这种方式构造并传递给 addConnectionPoolToken()，以便可将额外的信息传递到连接池。

IBM MQ classes for Java 中的缺省连接池和多个组件

此示例显示了如何通过静态的已注册 MQPoolToken 对象集添加或移除 MQPoolToken。

MQEnvironment 包含静态的已注册 MQPoolToken 对象集。 要通过此集添加或移除 MQPoolToken，请使用以下方法：

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

应用程序可能包含多个组件，这些组件独立存在，并使用队列管理器执行工作。 在此类应用程序中，每个组件都应该在其生命周期中向 MQEnvironment 集添加一个 MQPoolToken。

例如，示例应用程序 MQApp3 将创建十个线程并启动每一个线程。 每个线程都将注册自己的 MQPoolToken，等待一段时间，然后连接到队列管理器。 线程断开连接后，会移除自己的 MQPoolToken。

缺省连接池仍然处于活动状态，而 MQPoolToken 集中至少具有一个令牌，因此它在此应用程序使用期间仍然处于活动状态。 应用程序不需要保留主对象来全面控制线程。

```
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);
      }
}
```

在 *IBM MQ classes for Java* 中提供不同的连接池
此示例显示了如何使用类 **com.ibm.mq.MQSimpleConnectionManager** 来提供不同的连接池。

此类提供了连接池的基本功能，且应用程序可以使用此类来定制池的行为。

一旦实例化，就可以在 MQQueueManager 构造函数中指定 MQSimpleConnectionManager。 MQSimpleConnectionManager 随后将管理作为已构造 MQQueueManager 基础的连接。 如果 MQSimpleConnectionManager 包含合适的池连接，那么该连接将在 MQQueueManager.disconnect() 调用后被复用并返回到 MQSimpleConnectionManager。

以下代码片段说明了此行为：

```
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);
```

在第一个 MQQueueManager 构造函数中伪造的连接将在 qmgr.disconnect() 调用后存储在 myConnMan 中。 该连接随后将在第二次调用 MQQueueManager 构造函数期间被复用。

第二行将启用 MQSimpleConnectionManager。 最后一行将禁用 MQSimpleConnectionManager，破坏池中包含的任何连接。 在缺省情况下，MQSimpleConnectionManager 处于 MODE_AUTO 状态，本部分稍后将作说明。

MQSimpleConnectionManager 将以最近最多使用为基础分配连接，并以最近最少使用为基础破坏连接。 缺省情况下，如果连接在五分钟内未被使用，或者池中存在超过十个未使用的连接，那么就会破坏连接。 您可以通过调用 MQSimpleConnectionManager.setTimeout() 来改变这些值。

您也可以设置 MQSimpleConnectionManager 作为缺省连接池使用，如果 MQQueueManager 构造函数上未提供连接管理器，就将使用该连接池。

以下应用程序说明了此操作：

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

粗体行将创建并配置 MQSimpleConnectionManager 对象。 配置过程将执行以下任务：

- 结束在一个小时内未使用的连接
- 将 myConnMan 管理的连接数量限制为 75
- 将池中未使用的连接数量限制为 50
- 设置 MODE_AUTO，这是缺省值。 这意味着仅当池作为缺省连接管理器，且 MQEnvironment 包含的 MQPoolToken 集中至少有一个令牌时，该池才处于活动状态。

新的 MQSimpleConnectionManager 随后会设置为缺省连接管理器。

在最后一行中，应用程序将调用 MQApp3.main()。这会运行多个线程，其中每个线程都独立使用 IBM MQ。这些线程会在伪造连接时使用 myConnMan。

使用 *IBM MQ classes for Java* 进行 *JTA/JDBC* 协调

IBM MQ classes for Java 支持 MQQueueManager.begin() 方法，允许 IBM MQ 充当数据库（提供 JDBC 类型 2 或 JDBC 类型 4 兼容驱动程序）的协调程序。

这种支持不适用于所有平台。 要检查哪些平台支持 JDBC 协调，请参阅 IBM MQ 的系统需求。

要使用 XA-JTA 支持，您必须使用特殊 JTA 切换库。 使用这种库的方法各不相同，具体取决于您是使用 Windows 还是其他某种平台。

在 *Windows* 上配置 *JTA/JDBC* 协调

XA 库作为 DLL 提供，名称格式为 `jdbcxxx.dll`。

提供的 `jdbcora12.dll` 与 Oracle 12C 兼容，适用于 IBM MQ for Windows 服务器安装。

在 Windows 系统上，XA 库作为完整的 DLL 提供。 此 DLL 的名称为 `jdbcxxx.dll`，其中 `xxx` 表示已编译切换库的数据库。 此库位于 IBM MQ classes for Java 安装的 `java\lib\jdbc` 或 `java\lib64\jdbc` 目录中。 您必须向队列管理器声明 XA 库（也称为切换装入文件）。 使用 IBM MQ Explorer。 在队列管理器属性面板中的 XA 资源管理器下指定切换装入文件的详细信息。 您必须仅提供库的名称。 例如：

对于 Db2 数据库，将 SwitchFile 字段设置为：`dbcdb2`

对于 Oracle 数据库，将 SwitchFile 字段设置为：`jdbcora`

注意:

1. Oracle 12C 受 IBM MQ classes for Java 支持，仅在 IBM MQ for Windows 上受支持。
2. 受支持的 Oracle 12C 版本为 12.1.0.1.0 Enterprise Edition 和未来的修订包。
3. 64 位 Windows 上的 Oracle 64 位数据库需要 32 位 Oracle 客户机。
4. 通过使用 IBM MQ classes for Java，IBM MQ 可以充当事务协调程序。 但无法参与 JTA 样式事务。

在 *Windows* 以外的平台上配置 *JTA/JDBC* 协调
将提供对象文件。 使用提供的 Makefile 链接合适的对象文件，并使用配置文件向队列管理器声明该对象文件。

对于每个数据库管理系统，IBM MQ 提供两个对象文件。 您必须链接一个对象文件以创建 32 位切换库，并链接另一个对象文件以创建 64 位切换库。 对于 Db2，每个对象文件的名称为 jdbcdb2.o，对于 Oracle，每个对象文件的名称为 jdbcora.o。

您必须使用随 IBM MQ 提供的相应 Makefile 链接每个对象文件。 切换库需要其他库，这些库可能存储于不同系统上的不同位置。 然而，切换库无法使用库路径环境变量来查找这些库，因为切换库是由队列管理器装入的，该队列管理器在 setuid 环境中运行。 正因如此，提供的 Makefile 会确保切换库包含这些库的标准路径名。

要创建切换库，请使用以下格式输入 **make** 命令。 要创建 32 位切换库，请在 IBM MQ 安装的 /java/lib/jdbc 目录中输入命令。 要创建 64 位切换库，请在 /java/lib64/jdbc 目录中输入命令。

```
make DBMS
```

其中，*DBMS* 是您为其创建切换库的数据库管理系统。 有效值为 db2 for Db2 和 `oracle` for Oracle。

以下是 **make** 命令的示例：

```
make db2
```

请注意以下几点：

- 要运行 32 位应用程序，您必须为正在使用的每个数据库管理系统创建 32 位和 64 位切换库。 要运行 64 位应用程序，您只需创建 64 位切换库。 对于 Db2，每个切换库的名称为 jdbcdb2，对于 Oracle，每个切换库的名称为 jdbcora。 Makefile 可确保 32 位 和 64 位切换库存储于不同的 IBM MQ 目录。 32 位切换库存储于 /java/lib/jdbc 目录，而 64 位切换库则存储于 /java/lib64/jdbc 目录。
- 因为您可以将 Oracle 安装在系统上的任何位置，Makefile 会使用 ORACLE_HOME 环境变量来查找 Oracle 的安装位置。

针对 Db2 和/或 Oracle 创建切换库之后，您必须向队列管理器声明这些切换库。 如果队列管理器配置文件 (qm.ini) 已经包含适用于 Db2 或 Oracle 数据库的 XAResourceManager 节，那么您必须使用以下一项替换每个节中的 SwitchFile 条目：

对于 Db2 数据库

```
SwitchFile=jdbcdb2
```

对于 Oracle 数据库

```
SwitchFile=jdbcora
```

请勿指定 32 位或 64 位切换库的标准路径名。 仅指定库的名称。

如果队列管理器配置文件尚不包含适用于 Db2 或 Oracle 数据库的 XAResourceManager 节，或者您希望添加其他 XAResourceManager 节，请参阅管理，以获取关于如何构造 XAResourceManager 节的信息。 然而，新 XAResourceManager 节中的每个 SwitchFile 条目都必须与之前针对 Db2 或 Oracle 数据库所描述的完全一致。 此外，还必须包含条目 `ThreadOfControl=PROCESS`。

在更新了队列管理器配置文件并确保设置了所有合适的数据库环境变量后，您可以重新启动队列管理器。

使用 *JTA/JDBC* 协调
按照提供的示例对 API 调用进行编码。

针对用户应用程序的 API 调用的基本序列为：

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

getJDBCConnection 调用中的 xads 是特定于数据库的 XADataSource 接口实现，可定义要连接的数据库的详细信息。 请参阅您的数据库文档，确定如何创建要传递到 getJDBCConnection 的相应 XADataSource 对象。

您也可以使用相应的特定于数据库的 JAR 文件更新类路径，从而执行 JDBC 工作。

如果必须连接到多个数据库，那么必须几次调用 getJDBCConnection，以便在几个不同的连接之间执行事务。

有两种形式的 getJDBCConnection，反映了 XADataSource.getXAConnection 的两种形式：

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

这些方法会在其抛出子句中声明异常，从而避免未使用 JTA 函数的客户的 JVM 验证器出现问题。 实际抛出的异常为 javax.transaction.xa.XAException，对于之前未作要求的程序，这会需要向类路径添加 jta.jar 文件。

要使用 JTA/JDBC 支持，您必须在应用程序中包含以下语句：

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

JTA/JDBC 协调的已知问题和限制
JTA/JDBC 支持的某些问题和限制取决于所使用的数据库管理系统，例如，如果数据库在应用程序运行时关闭，所测试的 JDBC 驱动程序的行为将有所不同。 如果应用程序所使用的与数据库的连接中断，那么应用程序可以执行几个步骤来重新建立与队列管理器和数据库的新连接，以便之后可以使用这些新连接来执行所需的事务工作。

因为 JTA/JDBC 支持会对 JDBC 驱动程序发出调用，所以这些 JDBC 驱动程序的实现会对系统行为产生重大影响。 特别是如果数据库在应用程序运行时关闭，那么所测试的 JDBC 驱动程序的行为将有所不同。

要点： 请始终避免在应用程序仍然保持与数据库的开放连接时突然关闭该数据库。

注： IBM MQ classes for Java 应用程序必须使用绑定模式来连接，以便使 IBM MQ 充当数据库协调程序。

多个 XAResourceManager 节
不支持在队列管理器配置文件 qm.ini 中使用多个 XAResourceManager 节。 将会忽略除了第一个 XAResourceManager 节之外的任何 XAResourceManager 节。

Db2

有时，Db2 会返回 SQL0805N 错误。 可通过以下 CLP 命令解决此问题：

```
DB2 bind @db2cli.lst blocking all grant public
```

要获取更多信息，请参阅 Db2 文档。

必须将 XAResourceManager 节配置为使用 ThreadOfControl=PROCESS。 对于 Db2 8.1 及更高版本，这与 Db2 控制设置的缺省线程不匹配，因此必须在 XA 开放字符串中指定 toc=p。 针对使用 JTA/JDBC 协调的 Db2 的 XAResourceManager 节示例如下：

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

这不会防止使用 JTA/JDBC 协调的 Java 应用程序为自己编写多个线程。

Oracle
在 MQQueueManager.disconnect() 生成 SQLException 后调用 JDBC Connection.close() 方法。 可以在 MQQueueManager.disconnect() 之前调用 Connection.close()，也可以忽略对 Connection.close() 的调用。

处理数据库连接问题

当 IBM MQ classes for Java 应用程序使用 IBM MQ 提供的 JTA/JDBC 支持时，通常会执行以下步骤：

1. 创建新的 MQQueueManager 对象，以表示与将充当事务管理器的队列管理器的连接。
2. 构造 XADataSource 对象，其中包含关于如何连接到将在事务中征调的数据库的详细信息。
3. 调用在之前创建的 XADataSource 中传递的 MQQueueManager.getJDBCConnection(XADataSource) 方法。 这会导致 IBM MQ classes for Java 与数据库建立连接。
4. 调用 MQQueueManager.begin() 方法以启动 XA 事务。
5. 执行消息传递和数据库工作。
6. 在完成所有必需工作后，调用 MQQueueManager.commit() 方法。这将完成 XA 事务。
7. 如果此时需要新的 XA 事务，该应用程序可以重复步骤 4、5 和 6。
8. 在应用程序完成后，应关闭在步骤 3 中创建的数据库连接，然后调用 MQQueueManager.disconnect() 方法以断开与队列管理器的连接。

IBM MQ classes for Java 将维护在应用程序调用 MQQueueManager.getJDBCConnection(XADataSource) 时已创建的所有数据库连接的内部列表。 如果队列管理器在处理 XA 事务期间需要与数据库通信，那么会进行以下处理：

1. 队列管理器调用 IBM MQ classes for Java，传递需要传递到数据库的 XA 调用的详细信息。
2. IBM MQ classes for Java 随后查找列表中的相应连接，然后使用该连接将 XA 调用传递到数据库。

如果与数据库的连接在此处理期间的任意时刻丢失，那么应用程序应该：

1. 通过调用 MQQueueManager.backout() 方法回退在此事务下完成的任何现有工作。
2. 关闭数据库连接。 这应该导致 IBM MQ classes for Java 从其内部列表中移除中断的数据库连接的详细信息。
3. 通过调用 MQQueueManager.disconnect() 方法断开与队列管理器的连接。
4. 通过构造新的 MQQueueManager 对象建立与队列管理器的新连接。
5. 通过调用 MQQueueManager.getJDBCConnection(XADataSource) 方法创建新的数据库连接。
6. 再次执行事务工作。

这会允许应用程序重新建立与队列管理器和数据库的新连接，然后使用这些连接来执行所需的事务工作。

IBM MQ classes for Java 中的传输层安全性 *(TLS)* 支持

IBM MQ classes for Java 客户机应用程序支持 TLS 加密。 要使用 TLS 加密，您需要 JSSE 提供程序。

使用 TRANSPORT(CLIENT) 的 IBM MQ classes for Java 客户机应用程序支持 TLS 加密。 TLS 提供通信加密、认证和消息完整性。 它通常用于保护因特网或内部网上任何两个对等实体之间的通信。

IBM MQ classes for Java 使用 Java 安全套接字扩展 (JSSE) 处理 TLS 加密，因此需要 JSSE 提供程序。 JSE V1.4 JVM 具有内置的 JSSE 提供程序。 有关如何管理和存储证书的详细信息根据提供程序不同而有所变化。 要获取相关信息，请参阅 JSSE 提供程序的文档。

本部分假设您已正确安装和配置了 JSSE 提供程序，同时安装了合适的证书，并使其可供 JSSE 提供程序使用。

如果您的 IBM MQ classes for Java 客户机应用程序使用客户机通道定义表 (CCDT) 连接到队列管理器，请参阅第 305 页的『将客户机通道定义表用于 IBM MQ classes for Java』。

在 *IBM MQ classes for Java* 中启用 *TLS*

要启用 TLS，您需要指定 CipherSuite。 可以通过两种方法来指定 CipherSuite。

仅客户机连接支持 TLS。 要启用 TLS，您必须在与队列管理器通信时指定要使用的 CipherSuite，且该 CipherSuite 必须与在目标通道上设置的 CipherSpec 匹配。 此外，您的 JSSE 提供程序必须支持该指定的 CipherSuite。 但是，CipherSuite 与 CipherSpec 不同，因此两者的名称也不同。 第 332 页的『IBM MQ classes for Java 中的 TLS CipherSpec 和 CipherSuite』包含一个表，该表将 IBM MQ 支持的 CipherSpec 映射到 JSSE 知道的对等 CipherSuite。

要启用 TLS，请使用 MQEnvironment 的 sslCipherSuite 静态成员变量指定 CipherSuite。 以下示例与名为 SECURE.SVRCONN.CHANNEL 的 SVRCONN 通道有关，该通道已设置为需要使用 CipherSpec 为 TLS_RSA_WITH_AES_128_CBC_SHA256 的 TLS：

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

尽管通道的 CipherSpec 为 TLS_RSA_WITH_AES_128_CBC_SHA256，但 Java 应用程序仍必须将 CipherSuite 指定为 SSL_RSA_WITH_AES_128_CBC_SHA256。 请参阅第 332 页的『IBM MQ classes for Java 中的 TLS CipherSpec 和 CipherSuite』，获取 CipherSpec 和 CipherSuite 之间的映射列表。

应用程序也可以通过设置环境属性 CMQC.SSL_CIPHER_SUITE_PROPERTY 来指定 CipherSuite。

或者，使用客户机通道定义表 (CCDT)。 有关更多信息，请参阅第 305 页的『将客户机通道定义表用于 IBM MQ classes for Java』。

如果您需要客户机连接来使用受 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS) 支持的 CipherSuite，那么应用程序可以将 MQEnvironment 类中的 sslFipsRequired 字段设置为 `true`。 或者，应用程序可以设置环境属性 CMQC.SSL_FIPS_REQUIRED_PROPERTY。 缺省值为 `false`，这表明客户机连接可以使用 IBM MQ 支持的任何 CipherSuite。

如果应用程序使用多个客户机连接，那么在应用程序创建第一个客户机连接时使用的 sslFipsRequired 字段值，将决定在应用程序创建任何后续客户机连接时所使用的值。 因此，在应用程序创建后续客户机连接时，会忽略 sslFipsRequired 字段值。 如果想要针对 sslFipsRequired 字段使用不同的值，那么必须重新启动应用程序。

要使用 TLS 成功连接，必须使用认证中心根证书（可以对队列管理器显示的证书进行认证）设置 JSSE 信任库。 类似地，如果 SVRCONN 通道上的 SSLClientAuth 已被设为 MQSSL_CLIENT_AUTH_REQUIRED，那么 JSSE 密钥库必须包含一个队列管理器信任的标识证书。

相关参考

针对 UNIX, Linux, and Windows 的美国联邦信息处理标准 (FIPS)

在 *IBM MQ classes for Java* 中使用队列管理器的专有名称

队列管理器可使用包含专有名称 (DN) 的 TLS 证书识别自身。 IBM MQ classes for Java 客户机应用程序可以使用该 DN 确保与正确的队列管理器通信。

使用 MQEnvironment 的 sslPeerName 变量指定 DN 模式。 例如，设置：

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

仅当队列管理器提供公共名称以 QMGR. 开头的证书时，才允许连接成功。 以及至少两个组织单元名称，其中第一个必须是 IBM ，第二个必须是 WebSphere。

如果设置 sslPeerName，仅当它被设为有效模式并且队列管理器提供一个匹配证书时，连接才成功。

应用程序也可以通过设置环境属性 CMQC.SSL_PEER_NAME_PROPERTY 来指定队列管理器的专有名称。 有关专有名称的更多信息，请参阅专有名称。

在 *IBM MQ classes for Java* 中使用证书撤销列表

通过 java.security.cert.CertStore 类指定要使用的证书撤销列表。 IBM MQ classes for Java 之后会根据指定的 CRL 检查证书。

证书撤销列表是由颁发证书的认证中心或本地组织所撤销的一系列证书。 CRL 通常托管在 LDAP 服务器上。 利用 Java 2 V1.4，可以在连接时指定 CRL 服务器，并且在根据 CRL 检查了队列管理器所显示的证书后，才允许连接。 有关证书撤销列表和 IBM MQ 的更多信息，请参阅使用证书撤销列表和机构撤销列表以及通过 IBM MQ classes for Java 和 IBM MQ classes for JMS 访问 CRL 和 ARL。

注: 要将 CertStore 成功用于 LDAP 服务器上托管的 CRL，请确保您的 Java 软件开发包 (SDK) 与此 CRL 兼容。 某些 SDK 要求 CRL 符合 RFC 2587（定义了 LDAP V2 的模式）。 大部分 LDAP V3 服务器改为使用 RFC 2256。

可通过 java.security.cert.CertStore 类指定要使用的 CRL。 有关如何获取 CertStore 实例的完整详细信息，请参阅关于此类的文档。 要基于 LDAP 服务器创建 CertStore，首先请创建 LDAPCertStoreParameters 实例，并通过要使用的服务器和端口设置进行初始化。 例如：

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

创建了 CertStoreParameters 实例之后，可使用 CertStore 上的静态构造函数来创建 LDAP 类型的 CertStore：

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

同时也支持其他类型的 CertStore（例如集合）。 在通常情况下，会使用同样的 CRL 信息设置几个 CRL 服务器，从而提供冗余性。 如果您拥有针对每个 CRL 服务器的 CertStore 对象，请将其全部置于合适的集合中。 以下示例显示了置于 ArrayList 中的 CertStore 对象：

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

在连接以启用 CRL 检查前，可以将此集合设置为 MQEnvironment 静态变量 sslCertStores：

```
MQEnvironment.sslCertStores = crls;
```

设置连接时队列管理器提交的证书按照如下过程进行验证：

1. sslCertStores 所识别的集合中的第一个 CertStore 对象用于识别 CRL 服务器。
2. 尝试联系 CRL 服务器。
3. 如果尝试成功，那么将搜索服务器以便匹配证书。
   a. 如果发现证书已撤销，那么搜索流程结束，连接请求失败，并出现原因码 MQRC_SSL_CERTIFICATE_REVOKED。
   b. 如果未找到证书，那么搜索流程结束，且允许继续处理连接。
4. 如果尝试联系服务器失败，那么会使用下一个 CertStore 对象来识别 CRL 服务器，并自步骤 2 起重复流程。

   如果这是集合中的最后一个 CertStore，或者如果集合不包含 CertStore 对象，那么搜索流程失败，且连接请求失败，并出现原因码 MQRC_SSL_CERT_STORE_ERROR。

集合对象将决定使用 CertStore 的顺序。

同时也可以使用 CMQC.SSL_CERT_STORE_PROPERTY 来设置 CertStore 集合。 为方便起见，此属性也允许指定并非集合成员的单个 CertStore。

如果将 sslCertStores 设置为 Null，那么不会执行 CRL 检查。 如果未设置 sslCipherSuite，那么忽略该属性。

在 *IBM MQ classes for Java* 中重新协商密钥

IBM MQ classes for Java 客户机应用程序可以根据发送和接收的字节总数，控制何时对用于客户机连接加密的密钥进行重新协商。

应用程序可以通过以下任一方式执行此操作：如果应用程序使用其中多种方式，那么采用常规优先顺序规则。

- 通过在 MQEnvironment 类中设置 sslResetCount 字段。
- 通过在 Hashtable 对象中设置环境属性 MQC.SSL_RESET_COUNT_PROPERTY。 应用程序之后会向 MQEnvironment 类中的 `properties` 字段分配散列表，或者将散列表传递到其构造函数上的 MQQueueManager 对象。

sslResetCount 字段的值或环境属性 MQC.SSL_RESET_COUNT_PROPERTY，表示 IBM MQ classes for Java 客户机代码在重新协商密钥前发送和接收的字节总数。 发送的字节数是加密之前的字节数，接收的字节数是解密之后的字节数。 字节数也包括 IBM MQ classes for Java 客户机发送和接收的控制信息。

如果重置计数是零（即缺省值），那么始终不会重新协商密钥。 如果没有指定 CipherSuite，将忽略重置计数。

在 *IBM MQ classes for Java* 中提供定制的 *SSLSocketFactory*

如果使用定制的 JSSE 套接字工厂，请将 MQEnvironment.sslSocketFactory 设置为定制工厂对象。 不同 JSSE 实现的详细信息各有不同。

不同的 JSSE 实现可提供不同的功能。 例如，专门的 JSSE 实现可能允许配置特定型号的加密硬件。 此外，某些 JSSE 提供程序允许通过程序定制密钥库和信任库，或者允许从密钥库中选择要更改的身份证书。 在 JSSE 中，所有这些定制都被抽象到一个工厂类 javax.net.ssl.SSLSocketFactory 中。

请参阅您的 JSSE 文档，了解关于如何创建定制的 SSLSocketFactory 实现的详细信息。 根据提供程序的不同，详细信息也有所不同，但一般的步骤顺序可能如下：

1. 在 SSLContext 上使用静态方法创建 SSLContext 对象
2. 使用适当的 KeyManager 和 TrustManager 实现（从其自己的工厂类创建）初始化此 SSLContext
3. 从 SSLContext 创建 SSLSocketFactory

如果拥有 SSLSocketFactory 对象，请将 MQEnvironment.sslSocketFactory 设置为定制工厂对象。 例如：

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java 使用此 SSLSocketFactory 与 IBM MQ 队列管理器相连。 同时也可以使用 CMQC.SSL_SOCKET_FACTORY_PROPERTY 设置此属性。 如果将 sslSocketFactory 设置为 Null，那么将使用 JVM 的缺省 SSLSocketFactory。 如果未设置 sslCipherSuite，那么忽略该属性。

使用定制 SSLSocketFactory 时，请考虑 TCP/IP 连接共享的影响。 如果可以实现连接共享，那么不会针对所提供的 SSLSocketFactory 请求新套接字，即使在后续连接请求的上下文中，产生的套接字可能会在某些方面有所不同。 例如，如果后续连接中将出现不同的客户机证书，那么不得允许连接共享。

在 *IBM MQ classes for Java* 中更改 *JSSE* 密钥库或信任库

如果更改 JSSE 密钥库或信任库，那么要使更改生效，就必须执行某些操作。

如果更改 JSSE 密钥库或信任库的内容，或者更改密钥库或信任库文件的位置，那么当时正在运行的 IBM MQ classes for Java 应用程序不会自动获取更改。 要使更改生效，必须执行以下操作：

- 应用程序必须关闭其所有连接，并破坏连接池中任何未使用的连接。
- 如果 JSSE 提供程序高速缓存来自密钥库和信任库的信息，那么必须刷新此信息。

执行这些操作后，应用程序可以重新创建其连接。

根据应用程序的设计方式以及 JSSE 提供程序所提供的功能，或许能够在不停止并重新启动应用程序的情况下执行这些操作。 然而，停止并重新启动应用程序可能是最简单的解决方案。

将 *TLS* 与 *IBM MQ classes for Java* 一起使用时的错误处理

在使用 TLS 连接到队列管理器时，IBM MQ classes for Java 会发出一系列原因码。

以下列表对这些原因码进行了说明：

MQRC_SSL_NOT_ALLOWED
设置了 sslCipherSuite 属性，但使用了绑定连接。 仅客户机连接支持 TLS。

MQRC_JSSE_ERROR
JSSE 提供程序报告了一个 IBM MQ 无法处理的错误。 这可能是由 JSSE 的配置问题造成的，也可能是因为无法验证队列管理器所显示的证书。 可以使用 getCause() 方法在 MQException 上检索 JSSE 所产生的异常。

MQRC_SSL_INITIALIZATION_ERROR
使用指定的 TLS 配置选项发出了 MQCONN 或 MQCONNX 调用，但初始化 TLS 环境期间发生错误。

MQRC_SSL_PEER_NAME_MISMATCH
在 sslPeerName 属性中指定的 DN 模式与队列管理器提供的 DN 不匹配。

MQRC_SSL_PEER_NAME_ERROR
在 sslPeerName 属性中指定的 DN 模式无效。

MQRC_UNSUPPORTED_CIPHER_SUITE
JSSE 提供程序未识别在 sslCipherSuite 中命名的 CipherSuite。 可通过使用 SSLSocketFactory.getSupportedCipherSuites() 方法的程序获取受 JSSE 提供程序支持的完整 CipherSuite 列表。 可以在 第 332 页的『IBM MQ classes for Java 中的 TLS CipherSpec 和 CipherSuite』中找到可用于与 IBM MQ 通信的 CipherSuites 列表。

MQRC_SSL_CERTIFICATE_REVOKED
可以在使用 sslCertStores 属性指定的 CRL 中找到队列管理器所显示的证书。 更新队列管理器，以使用可信证书。

MQRC_SSL_CERT_STORE_ERROR
无法在提供的 CertStore 中搜索到队列管理器所显示的证书。 MQException.getCause() 方法返回在尝试搜索第一个 CertStore 时发生的错误。 如果原因异常是 NoSuchElementException、ClassCastException 或 NullPointerException，那么检查在 sslCertStores 属性上指定的 Collection 是否至少包含一个有效的 CertStore 对象。

IBM MQ classes for Java 中的 *TLS CipherSpec* 和 *CipherSuite*

IBM MQ classes for Java 应用程序与队列管理器建立连接的能力取决于在 MQI 通道的服务器端指定的 CipherSpec，以及在客户端指定的 CipherSuite。

下表列出了 IBM MQ 支持的 CipherSpec 及其对应的 CipherSuite。

您应查看主题不推荐的 CipherSpec，以了解 IBM MQ 是否不推荐使用下表中列出的任何 CipherSpec，如果情况如此，请查看不推荐 CipherSpec 的更新。

要点: 列出的 CipherSuite 是 IBM MQ 随附的 IBM Java 运行时环境 (JRE) 支持的 CipherSuite。 列出的 CipherSuite 包括 Oracle Java JRE 所支持的 CipherSuite。 有关配置应用程序以使用 Oracle Java JRE 的更多信息，请参阅第 361 页的『配置应用程序以使用 IBM Java 或 Oracle Java CipherSuite 映射』。

下表还指明了用于通信的协议，以及 CipherSuite 是否符合 FIPS 140-2 标准。

如果应用程序未配置为强制实施 FIPS 140-2 合规性，可以使用标记为符合 FIPS 140-2 标准的 Ciphersuite，但如果为应用程序配置了 FIPS 140-2 合规性（请参阅下面有关配置的注释），那么只有标记为符合 FIPS 140-2 标准的那些 CipherSuite 才能配置；尝试使用其他 CipherSuite 会导致错误。

注: 每个 JRE 可以具有多个加密安全提供程序，其中每个提供程序都可以提供同一个 CipherSuite 的实现。 然而，并非所有安全提供程序都是经过 FIPS 140-2 认证的。 如果未对应用程序强制实施 FIPS 140-2 合规性，可能会使用未经过认证的 CipherSuite 实现。 未经过认证的实现可能无法与 FIPS 140-2 兼容，即使 CipherSuite 在理论上符合该标准要求的最低安全级别也不例外。 请参阅以下注释，了解有关在 IBM MQ Java 应用程序中配置 FIPS 140-2 强制实施的更多信息。

有关 CipherSpec 和 CipherSuite 的 FIPS 140-2 和 Suite-B 合规性的更多信息，请参阅指定 CipherSpec。您可能还需要了解有关美国联邦信息处理标准的信息。

要使用完整的 CipherSuite 集，并使用经过认证的 FIPS 140-2 和/或 Suite-B 合规性进行操作，需要一个合适的 JRE。IBM Java 7 服务刷新 4 修订包 2 或更高级别的 IBM JRE 为 第 334 页的表 58 中列出的 TLS 1.2 CipherSuites 提供相应支持。

V 9.1.5 为了能够使用 TLS v1.3 密码，运行您的应用程序的 JRE 必须支持 TLS v1.3。

注: 要使用某些 CipherSuite，需要在此 JRE 中配置“无限制”策略文件。 有关如何在 SDK 或 JRE 中设置策略文件的更多详细信息，请参阅您正在使用的版本的 *Security Reference for IBM SDK Java Technology Edition* 中的 *IBM SDK* 策略文件 主题。

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	否

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	否

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	否

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	否

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 360 页的『1』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	否
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	否

表 *58: IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite (*继续*)*

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	否
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	yes

表 58: *IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite* (继续)

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
V 9.1.5 TLS_AES_128_GCM_SHA256 第 360 页的『2』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	否
V 9.1.5 TLS_AES_256_GCM_SHA384 第 360 页的『2』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	否

表 58: IBM MQ 支持的 CipherSpec 及其对应的 CipherSuite (继续)

CipherSpec	对应的 CipherSuite (IBM JRE)	对应的 CipherSuite (Oracle JRE)	协议	是否符合 FIPS 140-2 标准
V 9.1.5 TLS_CHACHA20_POLY1305_SHA256 第 360 页的『2』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	否

表 58: *IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite* (继续)

CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
V 9.1.5 TLS_AES_128_CCM_SHA256 第 360 页的『2』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	否
V 9.1.5 TLS_AES_128_CCM_8_SHA256 第 360 页的『2』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	否
V 9.1.5 ANY 第 360 页的『2』	*ANY	*ANY	多个	否

表 58: *IBM MQ* 支持的 *CipherSpec* 及其对应的 *CipherSuite* (继续)				
CipherSpec	**对应的 CipherSuite (IBM JRE)**	**对应的 CipherSuite (Oracle JRE)**	**协议**	**是否符合 FIPS 140-2 标准**
V 9.1.5 ANY_TLS13 第 360 页的『2』	*TLS13	*TLS13	TLS V13	否
V 9.1.5 ANY_TLS12_OR_HIGHER 第 360 页的『2』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS V1.2 及更高版本	否
V 9.1.5 ANY_TLS13_OR_HIGHER 第 360 页的『2』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS V1.3 及更高版本	否

注意:

1. 不推荐 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。
2. V 9.1.5 为了能够使用 TLS v1.3 密码，运行您的应用程序的 JRE 必须支持 TLS v1.3

在 IBM MQ classes for Java 应用程序中配置 Ciphersuite 与 FIPS 合规性

- 使用 IBM MQ classes for Java 的应用程序可以使用以下两种方法中的任何一种来设置 CipherSuite 以进行连接：

  - 将 MQEnvironment 类中的 sslCipherSuite 字段设置为 CipherSuite 名称。
  - 将传递到 MQQueueManager 构造函数的属性哈希表中的属性 CMQC.SSL_CIPHER_SUITE_PROPERTY 设置为 CipherSuite 名称。
- 使用 IBM MQ classes for Java 的应用程序可以使用以下两种方法中的任何一种来强制实施 FIPS 140-2 合规性：
  - 将 MQEnvironment 类中的 sslFipsRequired 字段设置为 True。
  - 将传递到 MQQueueManager 构造函数的属性哈希表中的属性 CMQC.SSL_FIPS_REQUIRED_PROPERTY 设置为 True。

配置应用程序以使用 IBM Java 或 Oracle Java CipherSuite 映射

您可以配置应用程序是使用缺省的 IBM Java CipherSuite 到 IBM MQ CipherSpec 的映射，还是使用 Oracle CipherSuite 到 IBM MQ CipherSpec 的映射。 因此，无论应用程序是使用 IBM JRE 还是 Oracle JRE，您都可以使用 TLS CipherSuite。 Java 系统属性 `com.ibm.mq.cfg.useIBMCipherMappings` 控制使用哪些映射。 该属性可以具有以下某个值：

true
使用 IBM Java CipherSuite 到 IBM MQ CipherSpec 的映射。

该值为缺省值。

false
使用 Oracle CipherSuite 到 IBM MQ CipherSpec 的映射。

有关使用 IBM MQ Java 和 TLS 密码的更多信息，请参阅 MQdev 博客帖子 MQ Java， TLS 密码，非 IBM JRE 和 APAR IT06775， IV66840， IT09423， IT10837。

互操作性限制

根据所使用的协议，某些 CipherSuite 可能与多个 IBM MQ CipherSpec 兼容。 但是， 仅支持使用表 1 中指定的 TLS 版本的 CipherSuite/CipherSpec 组合。 尝试使用不支持的 CipherSuite 和 CipherSpec 组合会导致失败，并出现相应异常。 使用任何这些 CipherSuite/CipherSpec 组合的安装都应转为使用支持的组合。

下表显示了受此限制影响的 CipherSuite。

表 *59: CipherSuite* 及其支持和不支持的 *CipherSpec*

CipherSuite	**支持的 TLS CipherSpec**	**不支持的 SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA 第 361 页的『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

注:

1. 不推荐此 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。

运行 *IBM MQ classes for Java* 应用程序

如果使用客户机或绑定模式编写应用程序（包含 main() 方法的类），请通过 Java 解释器运行您的程序。

使用以下命令：

```
java -Djava.library.path= library_path MyClass
```

其中，*library_path* 是指向 IBM MQ classes for Java 库的路径。 有关更多信息，请参阅 第 289 页的『IBM MQ classes for Java 库』。

相关任务

跟踪 IBM MQ classes for Java 应用程序

跟踪 IBM MQ 资源适配器

依赖于 IBM MQ classes for Java 环境的行为

IBM MQ classes for Java 允许您创建可针对不同 IBM MQ 版本运行的应用程序。 本主题集合说明了依赖于这些不同版本的 Java 类的行为。

IBM MQ classes for Java 提供了类核心，可在所有环境中提供一致的功能和行为。 该核心之外的功能部件取决于应用程序所连接的队列管理器的功能。

除了在此注明外，所展示的行为在适用于队列管理器的 MQI 应用程序参考中有相关描述。

IBM MQ classes for Java 中的核心类

IBM MQ classes for Java 包含一个核心类集合，可用于所有环境。

以下类集合被视为核心类，只需要执行 第 363 页的『IBM MQ classes for Java 核心类的限制和变化』中列出的较小变动即可用于所有环境。

- MQEnvironment
- MQException
- MQGetMessageOptions

  不包括：

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - 分段
- MQManagedObject

  不包括：

  - inquire()
  - set()
- MQMessage

  不包括：

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

  不包括：

  - knownDestCount
  - unknownDestCount
  - invalidDestCount

  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  不包括：

  - begin( )
  - accessDistributionList()
- MQSimpleConnectionManager
- MQTopic
- MQC

注：

1. 某些常量不包含在核心中（请参阅 第 363 页的『IBM MQ classes for Java 核心类的限制和变化』以了解详细信息）；请勿在完全可移植的程序中加以使用。
2. 有些平台不支持所有连接模式。 在这些平台上，您只能使用与所支持的模式有关的核心类和选项。

IBM MQ classes for Java 核心类的限制和变化

即使同等的 MQI 调用一般存在环境差异，核心类在所有环境中的行为通常是一致的。 该行为就如同使用了 Windows、UNIX 或 Linux IBM MQ 队列管理器一样，除了存在以下细微限制和变化之外。

IBM MQ classes for Java 中 *MQGMO_** 值的限制
某些 MQGMO_* 值不受所有队列管理器支持。

使用以下 MQGMO_* 值可能会导致从 MQQueue.get() 中抛出 MQException：

MQGMO_SYNCPOINT_IF _PERSISTENT
MQGMO_MARK_SKIP _BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_ AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

此外，从 Java 中使用时，不支持 MQGMO_SET_SIGNAL。

IBM MQ classes for Java 中 *MQPMRF_** 值的限制
仅当将消息纳入分发列表中时才会使用这些限制，且只有支持分发列表的队列管理器支持这些限制。 例如，z/OS 队列管理器不支持分发列表。

IBM MQ classes for Java 中 *MQPMO_** 值的限制
某些 MQPMO_* 值不受所有队列管理器支持

使用以下 MQPMO_* 值可能会导致从 MQQueue.put() 或 MQQueueManager.put() 中抛出 MQException：

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

IBM MQ classes for Java 中 *MQCNO_** 值的限制和变化
某些 MQCNO_* 值不受支持。

- IBM MQ classes for Java 不支持自动客户机重新连接。 无论设置的 MQCNO_RECONNECT_* 值如何，连接的行为方式仍然会如同您设置了 MQCNO_RECONNECT_DISABLED 一样。
- 在不支持 MQCNO_FASTPATH 的队列管理器上，将忽略 MQCNO_FASTPATH。 它也将被客户机连接忽略。

IBM MQ classes for Java 中 *MQRO_** 值的限制
可以设置以下报告选项。

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

有关更多信息，请参阅报告。

z/OS 上的 *IBM MQ classes for Java* 与其他平台之间的其他差异
在某些方面，IBM MQ for z/OS 的行为不同于其他平台上的 IBM MQ。

BackoutCount
z/OS 队列管理器返回的 BackoutCount 最大值为 255，即使消息已被回退超过 225 次也是如此。

缺省动态队列前缀
使用绑定连接来连接到 z/OS 队列管理器时，缺省动态队列前缀是 CSQ.*。 否则，缺省动态队列前缀是 AMQ.*。

MQQueueManager 构造函数
z/OS 上不支持客户机连接。 尝试使用客户机选项连接会导致出现 MQException 以及 MQCC_FAILED 和 MQRC_ENVIRONMENT_ERROR。

MQQueueManager 构造函数也可能会失败，并返回 MQRC_CHAR_CONVERSION_ERROR（如果初始化 IBM-1047 和 ISO8859-1 代码页之间的转换失败）或 MQRC_UCS2_CONVERSION_ERROR（如果初始化队列管理器的代码页和 Unicode 之间的转换失败）错误。 如果您的应用程序失败，并返回这些原因码之一，请确保安装了语言环境的国家语言资源组件，并确保有正确的转换表可用。

针对 Unicode 的转换表将作为 z/OS C/C++ 可选功能部件的一部分进行安装。 请参阅 *z/OS C/C++ Programming Guide* (SC09-4765)，了解关于启用 UCS-2 转换的更多信息。

IBM MQ classes for Java 核心类外部的功能部件

IBM MQ classes for Java 包含某些功能，这些功能专门设计为使用不受所有队列管理器支持的 API 扩展。 本主题集合描述了在使用不支持这些功能部件的队列管理器时它们的行为方式。

MQQueueManager 构造函数选项中的变化
某些 MQQueueManager 构造函数包括可选整数自变量。 并非所有平台都接受该自变量的某些值。

如果 MQQueueManager 构造函数包含可选整数自变量，它将映射至 MQI 的 MQCNO 选项字段，并用于在常规和快速路径连接之间切换。 如果唯一使用的选项为 MQCNO_STANDARD_BINDING 或 MQCNO_FASTPATH_BINDING，那么所有环境均接受构造函数的这种扩展形式。 其他任意选项都将导致构造函数失败，返回 MQRC_OPTIONS_ERROR。 快速路径选项 CMQC.MQCNO_FASTPATH_BINDING 仅支持通过绑定连接来连接到支持它的队列管理器。 在其他环境中，它将被忽略。

MQQueueManager.begin() 方法的限制
此方法只能用于处于绑定方式的 UNIX， Linux 或 Windows 系统上的 IBM MQ 队列管理器。 否则，它将失败并产生 MQRC_ENVIRONMENT_ERROR。

有关详细信息，请参阅第 325 页的『使用 IBM MQ classes for Java 进行 JTA/JDBC 协调』。

MQGetMessageOptions 字段中的变体
某些队列管理器不支持 V2 MQGMO 结构，因此必须将一些字段设置为其缺省值。

使用不支持 V2 MQGMO 结构的队列管理器时，请保持以下字段设置为其缺省值：

GroupStatus
SegmentStatus
分段

此外，MatchOptions 字段仅支持 MQMO_MATCH_MSG_ID 和 MQMO_MATCH_CORREL_ID。 如果将不受支持的值放入这些字段中，那么后续 MQDestination.get() 将由于错误 MQRC_GMO_ERROR 而失败。 如果队列管理器不支持 V2 MQGMO 结构，那么在成功执行 MQDestination.get() 之后，将不更新这些字段。

IBM MQ classes for Java 中的分发列表内的限制
并非所有队列管理器都允许打开 MQDistributionList。

以下类用于创建分发列表：

MQDistributionList
MQDistributionListItem
MQMessageTracker

您可以在任何环境中创建和填充 MQDistributionLists 和 MQDistributionListItems，但是并非所有队列管理器都允许打开 MQDistributionList。 特别是，z/OS 队列管理器不支持分发列表。 在使用这种队列管理器时，如果尝试打开一个 MQDistributionList，会导致 MQRC_OD_ERROR。

MQPutMessageOptions 字段中的变体
如果队列管理器不支持分发列表，那么某些 MQPMO 字段以不同方式处理。

MQPMO 中的四个字段呈现为 MQPutMessageOptions 类中的以下成员变量：

knownDestCount
unknownDestCount
invalidDestCount
recordFields

这些字段主要旨在与分发列表结合使用。 但是，支持分发列表的队列管理器在对单个队列执行 MQPUT 之后还会填充 DestCount 字段。 例如，如果队列解析成本地队列，那么 knownDestCount 设为 1，其他两个计数字段设为 0。

如果队列管理器不支持分发列表，那么这些值模拟如下：

- 如果 put() 成功，那么 unknownDestCount 设置为 1，并且其他字段设置为 0。
- 如果 put() 失败，那么 invalidDestCount 设置为 1，并且其他字段设置为 0。

recordFields 变量与分发列表结合使用。 无论环境如何，都可以随时将值写入到 recordFields 中。 如果在随后的 MQDestination.put() 或 MQQueueManager.put() 上使用 MQPutMessageOptions 对象，而不是在 MQDistributionList.put() 上使用它，那么会将其忽略。

IBM MQ classes for Java 的 *MQMD* 字段中的限制
使用不支持分段的队列管理器时，涉及消息分段的某些 MQMD 字段应保留为其缺省值。

以下 MQMD 字段主要涉及消息分段：

GroupId
MsgSeqNumber
抵销
MsgFlags
OriginalLength

如果应用程序将其中任何 MQMD 字段设置为除其缺省值以外的其他值，然后在不支持这些字段的队列管理器上执行 put() 或 get()，那么 put() 或 get() 将由于 MQRC_MD_ERROR 而引发 MQException。 通过此类队列管理器成功执行 put() 或 get() 始终保持将 MQMD 字段设置为其缺省值。 请勿将已分组或已分段消息发送到针对不支持消息分组和分段的队列管理器运行的 Java 应用程序。

如果 Java 应用程序尝试通过 get() 从不支持这些字段的队列管理器获取消息，并且要检索的物理消息属于一组已分段消息（即，具有 MQMD 字段的非缺省值）的一部分，那么对它的检索不会出错。 然而，

MQMessage 中的 MQMD 字段未被更新，MQMessage 格式属性被设为 MQFMT_MD_EXTENSION，并且真实消息数据的前面放有一个包含了新字段值的 MQMDE 结构。

CICS Transaction Server 下 *IBM MQ classes for Java* 的限制

在 CICS Transaction Server for z/OS 环境中，仅允许主（第一个）线程发出 CICS 或 IBM MQ 调用。

请注意，不支持在 CICS Java 应用程序中使用 IBM MQ JMS 类。

因此，无法在此环境中的线程之间共享 MQQueueManager 或 MQQueue 对象，或者在子线程上创建新的 MQQueueManager。

z/OS 第 364 页的『z/OS 上的 IBM MQ classes for Java 与其他平台之间的其他差异』 标识了针对 z/OS 队列管理器运行时适用于 IBM MQ classes for Java 的一些限制和变体。 此外，在 CICS 下运行时，不支持 MQQueueManager 上的事务控制方法。 应用程序将使用 JCICS 任务同步方法 Task.commit() 和 Task.rollback()，而不是发出 MQQueueManager.commit() 或 MQQueueManager.backout()。Task 类由 com.ibm.cics.server 软件包中的 JCICS 提供。

使用 IBM MQ 资源适配器

通过资源适配器，在应用程序服务器中运行的应用程序可以访问 IBM MQ 资源。 它支持入站和出站通信。

资源适配器包含的内容

Java Platform, Enterprise Edition Connector Architecture (JCA) 提供将在 Java EE 环境中运行的应用程序连接到诸如 IBM MQ 或 Db2 之类的企业信息系统 (EIS) 的标准方式。 IBM MQ 资源适配器实现 JCA 1.7 接口并包含 IBM MQ classes for JMS。 它允许在应用程序服务器中运行的 JMS 应用程序和消息驱动 Bean (MDB) 访问 IBM MQ 队列管理器的资源。 资源适配器同时支持点到点域和发布/预订域。

IBM MQ 资源适配器在应用程序和队列管理器之间支持两种类型的通信：

出站通信
应用程序启动与队列管理器的连接，然后以同步方式将 JMS 消息发送到 JMS 目标并从 JMS 目标接收 JMS 消息。

入站通信
到达 JMS 目标的 JMS 消息传送到异步处理消息的 MDB。

资源适配器还包含 IBM MQ classes for Java。 这些类自动可用于在资源适配器已部署到的应用程序服务器中运行的应用程序，并允许在该应用程序服务器中运行的应用程序在访问 IBM MQ 队列管理器的资源时使用 IBM MQ classes for Java API。

支持在 Java EE 环境中使用 IBM MQ classes for Java ，但存在限制。 有关这些限制的信息，请参阅第 282 页的『在 Java EE 中运行 IBM MQ classes for Java 应用程序』。

要使用的资源适配器版本

您正在使用的应用程序服务器的 Java Platform, Enterprise Edition (Java EE) 版本确定必须使用的资源适配器的版本：

Java EE 7
IBM MQ 8.0 和更高版本的资源适配器支持 JCA V1.7 并提供 JMS 2.0 支持。 需要在 Java EE 7 和更高版本的应用程序服务器中部署此资源适配器（请参阅 第 367 页的『IBM MQ 资源适配器支持声明』）。

可以将 IBM MQ 8.0 或更高版本资源适配器安装在认证为符合 Java Platform, Enterprise Edition 7 规范的任何应用程序服务器上。 使用 IBM MQ 8.0 或更高版本的资源适配器，应用程序可以通过 BINDINGS 或 CLIENT 传输连接到 IBM WebSphere MQ 7.0 或更高版本的队列管理器，或者通过仅 CLIENT 传输连接到 IBM WebSphere MQ 6.0 队列管理器。

要点: IBM MQ 8.0 或更高版本资源适配器只能部署到支持 JMS 2.0 的应用程序服务器中。

Java EE 5 和 Java EE 6
IBM WebSphere MQ 7.5 资源适配器支持 Java EE Connector Architecture (JCA) V1.5 并提供 JMS 1.1 支持。 为提供与 WebSphere Liberty 的完全集成， IBM WebSphere MQ 7.5 资源适配器将从 IBM WebSphere MQ 7.5.0 Fix Pack 2 更新为 APAR IC92914 。 此资源适配器保留与其他 Java EE 5 和更高

版本的应用程序服务器的完全兼容性（请参阅 WebSphere MQ 资源适配器 v7.1 和更高版本的支持声明）。

将资源适配器与 WebSphere Application Server traditional 结合使用

从 IBM MQ 9.0 起，IBM MQ 资源适配器预安装在 WebSphere Application Server traditional 9.0 或更高版本中。 因此，无需安装新资源适配器。

注: IBM MQ 9.0 或更高版本的资源适配器可在 CLIENT 或 BINDINGS 传输方式下连接到任何运作中的 IBM MQ 队列管理器。

将资源适配器与 WebSphere Liberty 结合使用

要从 WebSphere Liberty 连接到 IBM MQ ，必须使用 IBM MQ 资源适配器。 由于 Liberty 不包含 IBM MQ 资源适配器，因此必须从 Fix Central 单独获取。 您使用的资源适配器的版本取决于应用程序服务器的 Java EE 版本。

有关如何下载和安装资源适配器的更多信息，请参阅第 373 页的『在 Liberty 中安装资源适配器』。

相关概念

第 378 页的『配置入站通信的资源适配器』
要配置入站通信，请定义一个或多个 ActivationSpec 对象的属性。

第 390 页的『配置出站通信的资源适配器』
要配置出站通信，请定义 ConnectionFactory 对象和受管目标对象的属性。

第 67 页的『使用 IBM MQ classes for JMS』
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) 是 IBM MQ 随附的 JMS 提供程序。 除实现 javax.jms 包中定义的接口以外，IBM MQ classes for JMS 还提供两个 JMS API 扩展集。

第 281 页的『使用 IBM MQ classes for Java』
在 Java 环境中使用 IBM MQ 。 IBM MQ classes for Java 允许 Java 应用程序作为 IBM MQ 客户机连接到 IBM MQ，或直接连接到 IBM MQ 队列管理器。

相关任务

配置应用程序服务器以使用最新的资源适配器维护级别

IBM MQ 资源适配器的问题确定

WebSphere Application Server 主题

维护 IBM MQ 资源适配器

将 JMS 应用程序部署到 Liberty 以使用 IBM MQ 消息传递提供程序

IBM MQ 资源适配器支持声明

随 IBM MQ 8.0 或更高版本提供的资源适配器已实现了 JMS 2.0 规范。 它仅可以部署到兼容 Java Platform, Enterprise Edition 7 (Java EE 7) 并支持 JMS 2.0 的应用程序服务器。

Oracle Web 站点上维护认证的应用程序服务器列表。

在 WebSphere Liberty 中部署

WebSphere Liberty 8.5.5 Fix Pack 6 和更高版本以及 WebSphere Application Server Liberty 9.0 和更高版本是 Java EE 7 认证应用程序服务器，因此可以在其上部署 IBM MQ 9.0 资源适配器。

WebSphere Liberty 具有两项功能部件可用于处理资源适配器：

- wmqJmsClient-1.1 功能部件允许处理 JMS 1.1 资源适配器。
- wmqJmsClient-2.0 功能部件允许处理 JMS 2.0 资源适配器。

要点: 必须使用 wmqJmsClient-2.0 功能部件部署 IBM MQ 8.0 或更高版本的资源适配器。

关于此配置的信息位于方案将 WebSphere Liberty Liberty 连接到 IBM MQ 中。

在 WebSphere Application Server traditional 中部署

WebSphere Application Server traditional 9.0 随已安装的 IBM MQ 9.0 资源适配器一起提供。 因此，无需安装新资源适配器。 已安装的资源适配器可以在 CLIENT 或 BINDINGS 传输方式下连接到 IBM MQ 或 IBM WebSphere MQ 受支持的版本上运行的任何队列管理器。 有关更多信息，请参阅第 368 页的『到 IBM MQ 8.0 或更高版本队列管理器的连接』。

要点: 无法将 IBM MQ 9.0 资源适配器部署到 IBM MQ 9.0 之前的 WebSphere Application Server traditional 版本中，因为这些版本未经 Java EE 7 认证。

任何受支持的 WebSphere Application Server 版本都可以使用与其捆绑在一起的 IBM MQ 资源适配器来连接到任何受支持的 IBM MQ 版本。

将资源适配器与其他应用程序服务器结合使用

对于所有其他符合 Java EE 7 的应用程序服务器，成功完成 IBM MQ 资源适配器 安装验证测试 (IVT) 之后产生的问题可以报告到 IBM 以调查 IBM MQ 产品跟踪和其他 IBM MQ 诊断信息。 如果 IBM MQ 资源适配器 IVT 可以成功运行，遇到的任何问题可能由于特定于应用程序服务器的错误部署或错误资源定义导致，并且必须使用应用程序服务器文档和/或该应用程序服务器的支持组织调查问题。

Java 运行时

用于运行应用程序服务器的 Java Runtime (JRE) 必须是 IBM MQ 9.0 或更高版本的客户机支持的运行时。 有关更多信息，请参阅 IBM MQ 的系统需求。 （选择要查看的版本和操作系统或组件报告，然后进入**支持软件**选项卡下方列出的 **Java** 链接。）

到 IBM MQ 8.0 或更高版本队列管理器的连接

使用已部署到 Java EE 7 认证的应用程序服务器中的资源适配器连接到 IBM MQ 8.0 或更高版本的队列管理器时，提供了完整范围的 JMS 2.0 功能。 有关 WebSphere Application Server 随附的资源适配器版本的更多信息，请参阅技术说明 Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server?。

要利用 JMS 2.0 功能，需要使用 IBM MQ 消息传递提供程序正常方式将资源适配器连接到队列管理器。 有关更多信息，请参阅配置 JMS **PROVIDERVERSION** 属性。

连接 IBM WebSphere MQ 7.5 或更低版本的队列管理器

支持将 IBM MQ 9.0 或更高版本的资源适配器部署到支持 JMS 2.0 的 Java EE 7 认证应用程序服务器中，并将该资源适配器连接到正在运行 IBM WebSphere MQ 7.5 或更低版本的队列管理器。 队列管理器的能力限制了可用功能。 有关更多信息，请参阅配置 JMS **PROVIDERVERSION** 属性。

MQ 扩展名

JMS 2.0 规范更改了特定行为的运作方式。 由于 IBM MQ 8.0 或更高版本实现了此规范，因此 IBM MQ 8.0 与更高版本以及 IBM MQ 8.0 之前的版本之间存在行为更改。 在 IBM MQ 8.0 和更高版本中， IBM MQ classes for JMS 包括对 Java 系统属性 `com.ibm.mq.jms.SupportMQExtensions` 的支持，当设置为 TRUE 时，会使这些版本的 IBM MQ 将这些行为还原为 IBM WebSphere MQ 7.5 或更低版本的行为。 属性的缺省值是 FALSE。

IBM MQ 9.0 或更高版本的资源适配器还包括资源适配器属性 `supportMQExtensions`，该属性具有与 `com.ibm.mq.jms.SupportMQExtensions` Java 系统属性相同的效果和缺省值。 缺省情况下，在 `ra.xml` 中将此资源适配器属性设置为 false。

如果同时设置了资源适配器属性和 Java 系统属性，那么系统属性优先。

请注意，在已经部署到 WebSphere Application Server traditional 9.0 的资源适配器中，此属性自动地设置为 TRUE 以协助迁移。

有关更多信息，请参阅第 265 页的『SupportMQExtensions 属性』。

一般问题

不支持会话交错功能

部分应用程序服务器提供会话交错功能，使用此功能时同样的 JMS 会话可以在多个事务中使用，尽管一次仅在一个事务中列出。 IBM MQ 资源适配器不支持此功能，其可以导致以下问题：

`尝试将消息放入 MQ 队列失败，原因码 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE)。`

调用 xa_close() 失败，原因码 -3 (XAER_PROTO)，将在应用程序服务器访问的 IBM MQ 队列管理器上生成探测器标识为 AT040010 的 FDC。 有关如何禁用此功能的信息，请参阅您的应用程序服务器文档。

有关如何恢复 XA 资源以恢复 XA 事务的 Java Transaction API (JTA) 规范

JTA 规范的第 3.4.8 节未定义用于重新创建 XA 资源以执行 XA 事务恢复的特定机制。 这样，XA 事务中涉及的 XA 资源的恢复方式由每个单个事务管理器（以及应用程序服务器）决定。 对于一些应用程序服务器， IBM MQ 9.0 资源适配器很可能不实现用于执行 XA 事务恢复的特定于应用程序服务器的机制。

匹配 ManagedConnectionFactory 中的连接

应用程序服务器可以在 IBM MQ 资源适配器提供的 ManagedConnectionFactory 实例上调用 matchManagedConnections 方法。 只有当此方法发现与应用程序服务器传递给此方法的 **javax.security.auth.Subject** 和 **javax.resource.spi.ConnectionRequestInfo** 自变量都匹配的一个项时，才会返回 ManagedConnection。

IBM MQ 资源适配器的限制

所有 IBM MQ 平台均支持 IBM MQ 资源适配器。 但是，在使用 IBM MQ 资源适配器时，IBM MQ 的一些功能将不可用或受限。

IBM MQ 资源适配器具有以下限制：

- 从 IBM MQ 8.0 开始，资源适配器是提供 JMS 2.0 功能的 Java Platform, Enterprise Edition 7 (Java EE 7) 资源适配器。 因此，IBM MQ 8.0 或更高版本的资源适配器必须安装在 Java EE 7 或更高版本认证的应用程序服务器上。 它可以在客户机或绑定传输方式下连接到任何运作中的队列管理器。
- 在 WebSphere Liberty 应用程序服务器内部运行时，将不支持稳定的 IBM MQ classes for Java。 在其他应用程序服务器中，不建议使用 IBM MQ classes for Java。 请参阅 IBM 技术说明 在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口 ，以获取 Java EE 中 IBM MQ classes for Java 注意事项的详细信息。
- 在 z/OS 上的 WebSphere Liberty 应用程序服务器内运行时，必须使用 wmqJmsClient-2.0 功能部件。 通用 JCA 支持针对 z/OS 不可用。
- IBM MQ 资源适配器不支持以除 Java 以外的语言编写的通道出口程序。
- 应用程序服务器运行时，sslFipsRequired 属性值必须对于所有 JCA 资源均为 true，或对于所有 JCA 资源均为 false。 即使未同时使用 JCA 资源，这也是需要满足的一项要求。 如果 sslFipsRequired 属性对于不同的 JCA 资源具有不同的值，那么 IBM MQ 会发出原因码 MQRC_UNSUPPORTED_CIPHER_SUITE，即使未使用 TLS 连接也如此。
- 您不能为应用程序服务器指定多个密钥库。 如果与多个队列管理器建立了连接，那么所有连接都必须使用相同的密钥库。 此限制不适用于 WebSphere Application Server。
- 如果将客户机通道定义表 (CCDT) 与多个适当的客户机连接通道定义一起使用，那么当发生故障时，资源适配器可能会选择不同的通道定义，并从而从 CCDT 中选择不同的队列管理器，这将导致发生事务恢复问题。 资源适配器不采取任何措施来防止使用此类配置，您应负责避免使用可能导致发生事务恢复问题的配置。
- 在 Java EE 容器 (EJB/Servlet) 中运行时，出站连接不支持 IBM WebSphere MQ 7.0.1 中引入的连接重试功能。 在 JEE 容器上下文中使用适配器时，对于出站 JMS ，无论事务配置或非事务性使用，都完全不支持连接重试。
- 如 Java EE Connector Architecture V1.7 规范的第 9.1.9 部分中定义的那样，不支持重新认证 JMS 连接。 IBM MQ 资源 adpater 中的 `ra.xml` 文件必须将名为 **`reauthentication-support`** 的属性设置为值 `false`。 应用程序服务器尝试重新认证 JMS 连接将导致 IBM MQ 资源适配器抛出 javax.resource.spi.SecurityException 和 MQJCA1028 消息代码。

相关任务

指定运行时在 MQI 客户机上仅使用经过 FIPS 认证的 CipherSpecs

相关参考

适用于 UNIX、Linux 和 Windows 的美国联邦信息处理标准 (FIPS)

WebSphere Application Server 和 IBM MQ 资源适配器

IBM MQ 资源适配器由通过 WebSphere Application Server 中的 IBM MQ 消息传递提供程序执行 JMS 消息传递的应用程序使用。

要点: 请勿将 IBM MQ 或 IBM WebSphere MQ 资源适配器与 WebSphere Application Server 6.0 或 WebSphere Application Server 6.1 结合使用。

WebSphere Application Server 7.0 和 WebSphere Application Server 8.0 包含 IBM WebSphere MQ 7.0 资源适配器的版本。

WebSphere Application Server 8.5.5 包含 IBM WebSphere MQ 7.1 资源适配器的版本。

WebSphere Application Server traditional 9.0 包含 IBM MQ 9.0 资源适配器的版本。 IBM MQ 9.0 或更高版本的资源适配器无法部署到更低版本的 WebSphere Application Server 中，因为这些版本未经 Java EE 7 认证。

如果要使用 JMS 应用程序从 WebSphere Application Server 中访问 IBM MQ 队列管理器的资源，请使用 WebSphere Application Server 中的 IBM MQ 消息传递提供程序。 IBM MQ 消息传递提供程序包含 IBM MQ classes for JMS 的版本。 有关更多信息，请参阅 Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ?。

要点: 请勿在应用程序中包含任何 IBM MQ classes for JMS 或 IBM MQ classes for Java JAR 文件。 这样做可能会导致 ClassCastExceptions 并且可能难以维护。

Liberty 和 IBM MQ 资源适配器

根据要安装的资源适配器的版本，可以使用 wmqJmsClient-1.1 或 wmqJmsClient-2.0 功能将 IBM MQ 资源适配器安装到 WebSphere Liberty 8.5.5 Fix Pack 2 或更高版本中。 或者，您可以遵守一些限制使用通用的 Java Platform, Enterprise Edition Connector Architecture (Java EE JCA) 支持来安装资源适配器。

将资源适配器安装到 Liberty 时的常规限制

如果使用 wmqJmsClient-1.1 或 wmqJmsClient-2.0 功能并且同时使用通用的 JCA 支持，那么以下限制适用于资源适配器:

- IBM MQ classes for Java 在 Liberty 中不受支持。 不得与 IBM MQ Liberty 消息传递功能或通用的 JCA 支持一起使用。 有关更多信息，请参阅 在 J2EE/JEE 环境中使用 WebSphere MQ Java 接口。
- IBM MQ 资源适配器具有传输类型 BINDINGS_THEN_CLIENT。 此传输类型在 IBM MQ Liberty 消息传递功能中不受支持。
- 在 IBM MQ 9.0 之前，Advanced Message Security (AMS) 功能未包含在 IBM MQ Liberty 消息传递功能中。 然而，AMS 受 IBM MQ 9.0 或更高版本资源适配器支持。

使用 Liberty 功能时的限制

从 WebSphere Liberty 8.5.5 Fix Pack 2 到 WebSphere Liberty 8.5.5 Fix Pack 5（含），只有 wmqJmsClient-1.1 功能可用，并且只可以使用 JMS 1.1。 WebSphere Liberty 8.5.5 Fix Pack 6 增加了 wmqJmsClient-2.0 功能，因此可以使用 JMS 2.0。

但是，您必须使用的功能取决于您正在使用的资源适配器的版本:

- The IBM WebSphere MQ 7.5.0 Fix Pack 6 和更高版本的 IBM WebSphere MQ 7.5 资源适配器只可以与 wmqJmsClient-1.1 功能一起使用。
- IBM MQ 8.0.0 Fix Pack 3 和更高版本的 IBM MQ 8.0 资源适配器只可以与 wmqJmsClient-2.0 功能一起使用。
- IBM MQ 9.0 资源适配器只可以与 wmqJmsClient-2.0 功能一起使用。

使用通用 JCA 支持时的限制

如果您正在使用通用 JCA 支持，那么以下限制适用：

- 使用通用 JCA 支持时，必须指定 JMS 的级别：
  - JMS 1.1 和 JCA 1.6 必须仅与 IBM WebSphere MQ 7.5.0 Fix Pack 6 和更高版本的 IBM WebSphere MQ 7.5 资源适配器一起使用。
  - JMS 2.0 和 JCA 1.7 必须仅与 IBM MQ 8.0.0 Fix Pack 3 和更高版本的 IBM MQ 8.0 资源适配器一起使用。
- 无法使用通用 JCA 支持在 z/OS 上运行 IBM MQ 资源适配器。 要在 z/OS 上运行 IBM MQ 资源适配器，必须使用 wmqJmsClient-1.1 或 wmqJmsClient-2.0 功能部件来运行该资源适配器。
- 使用以下 XML 元素来指定资源适配器的位置：

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

要点: ID 标记的值可以是 wmqJms 以外的任何值。 如果一定要使用 wmqJms 作为 ID，那么 Liberty 将无法正确装入资源适配器。 这是因为 wmqJms 是在内部用于引用 IBM MQ 特定功能的 ID。 实际上将创建 NullPointerException。

以下示例显示了来自 server.xml 文件的一些片段：

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

提示: 请注意，使用 jca-1.7 和 jms-2.0 功能，缺少 wmqJmsClient-2.0 功能。

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

提示: 请注意，对 ID 使用 mqJms（首选值）。 不要使用 wmqJms。

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

提示: 请注意，通过 id mqJms 将 classloaderProviderRef 返回至资源适配器；这将允许加载 IBM MQ 特定类。

使用通用 JCA 支持进行跟踪时的限制

跟踪和日志记录未集成在 Liberty 跟踪系统中。 相反，必须使用 Java 系统属性或 IBM MQ classes for JMS 配置文件来启用 IBM MQ 资源适配器跟踪，如 跟踪 IBM MQ classes for JMS 应用程序中所述。 有关如何在 Liberty 中设置 Java 系统属性的详细信息，请参阅 WebSphere Liberty 文档。

例如，为了在 Liberty 19.0.0.9 中启用 IBM MQ 资源适配器的跟踪，请向 Liberty 文件 jvm.options 添加一个条目：

1. 创建名为 jvm.options 的文本文件。
2. 将以下 JVM 选项 (每行一个) 插入到此文件中以启用跟踪：

```
-Dcom.ibm.msg.client.commonservices.trace.status=ON
-Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
```

3. 要将这些设置应用于单个服务器，请在以下位置保存 jvm.options：

   ```
   ${server.config.dir}/jvm.options
   ```

   要将这些更改应用于所有 Liberty，请在以下位置保存 jvm.options：

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   这将对没有本地定义的 jvm.options 文件的所有 JVM 生效。
4. 重新启动服务器以启用更改。

这会导致将跟踪写入到目录 <path_to_trace_to> 中名为 MQRA-WLP_<process identifier>.trc 的跟踪文件中。

含客户机通道定义表的完整 Liberty XA 支持

V 9.1.2

将 WebSphere Liberty 18.0.0.2 及更高版本与 IBM MQ 9.1.2 配合使用时，可将客户机通道定义表 (CCDT) 中的队列管理器组与 XA 事务配合使用。 这意味着现在可以利用队列管理器组所提供的工作负载分配和可用性同时保持事务完整性。

如果队列管理器发生连接错误，那么队列管理器需要重新变为可用以便解决事务。 事务恢复由 Liberty 管理，您可能需要配置事务管理器以便提供相应的时间段以供队列管理器重新恢复可用。 有关更多信息，请参阅 WebSphere Liberty 产品文档中的 事务管理器 (事务) 。

这是客户机端功能，即，您需要 IBM MQ 9.1.2 资源适配器而不是 IBM MQ 9.1.2 队列管理器。

安装 IBM MQ 资源适配器

IBM MQ 资源适配器作为资源归档 (RAR) 文件提供。 请在您的应用程序服务器中安装该 RAR 文件。 您可能需要将目录添加到系统路径。

关于此任务

IBM MQ 资源适配器将作为名为 wmq.jmsra.rar 的资源归档 (RAR) 文件提供。 RAR 文件包含 Java EE Connector Architecture (JCA) 接口的 IBM MQ classes for JMS 和 IBM MQ 实现。

将资源适配器安装为 IBM MQ 产品安装的一部分时，wmq.jmsra.rar 将随 IBM MQ classes for JMS 一起安装在 第 372 页的表 60 中显示的目录中。

表 *60:* 各平台的包含 *wmq.jmsra.rar* 的目录

平台	目录
UNIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

必须使用 IBM MQ 资源适配器从应用程序服务器连接到 IBM MQ。 根据您使用的应用程序服务器，资源适配器可能已预安装，或者您可能需要自行安装。

表 61: 应用程序服务器中的资源适配器安装	
应用程序服务器	**已预安装还是需要安装?**
WebSphere Application Server traditional 9.0	IBM MQ 9.0 资源适配器预安装在 WebSphere Application Server traditional 9.0 中。 因此，无需在 WebSphere Application Server traditional 9.0 中安装新的资源适配器。
WebSphere Liberty	WebSphere Liberty 不包含 IBM MQ 资源适配器，因此必须从 Fix Central 单独获取。
其他 Java EE 应用程序服务器	从 Fix Central 单独获取资源适配器，如 WebSphere Liberty。

过程

- 如果要从 WebSphere Liberty 或其他 Java EE 应用程序服务器连接到 IBM MQ，请下载并安装 IBM MQ 资源适配器，如 第 373 页的『在 Liberty 中安装资源适配器』中所述。
- Linux UNIX

  对于 UNIX and Linux 系统上的绑定连接，请确保包含 Java 本机接口 (JNI) 库的目录位于系统路径中。

  有关此目录（其中也包含 IBM MQ classes for JMS 库）的位置，请参阅 第 75 页的『配置 Java 本机接口 (JNI) 库』。

  Windows 在 Windows 上，此目录将在安装 IBM MQ classes for JMS 期间自动添加到系统路径中。

  提示: 作为设置系统路径的替代方法，IBM MQ 资源适配器具有名为 nativeLibraryPath 的属性，该属性可用于指定 JNI 库的位置。 例如，在 WebSphere Liberty 中，这将如以下示例中所示进行配置:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

  事务在客户机和绑定方式下均受支持。

在 *Liberty* 中安装资源适配器

要从 WebSphere Liberty 或其他 Java EE 应用程序服务器连接到 IBM MQ，必须使用 IBM MQ 资源适配器。由于 Liberty 不包含 IBM MQ 资源适配器，因此必须从 Fix Central 单独获取。

开始之前

注: 本主题中的信息不适用于 WebSphere Application Server traditional 9.0。 IBM MQ 9.0 资源适配器预安装在 WebSphere Application Server traditional 9.0 中。 因此，在这种情况下不需要安装新的资源适配器。

启动此任务之前，确保您已经在机器上安装 Java runtime environment (JRE) 并且 JRE 已经添加到系统路径。

此安装过程中使用的 Java 安装程序不需要以 root 用户或任何特定用户身份运行。 只要求运行它的用户具有您想要放入文件的目录的写入权限。

关于此任务

您可以从 Fix Central 下载的资源适配器的 JAR 文件是可执行文件。 在您运行此可执行文件时，它显示 IBM MQ 许可协议，必须接受该许可协议。 安装过程中要求指定安装 IBM MQ 资源适配器的目录。 然后，资源适配器 RAR 文件和安装验证测试 (IVT) 程序安装到该目录。 您可以接受缺省目录或指定其他目录，可以是应用程序服务器的资源适配器目录，或系统上任何其它目录。 如果不存在目录，将在安装过程中创建目录。

在 IBM MQ 9.0 之前，要下载的文件的名称格式为 *V.R.M.F*-WS-MQ-Java-InstallRA.jar，例如 8.0.0.6-WS-MQ-Java-InstallRA.jar。 从 IBM MQ 9.0 中，文件名的格式为 *V.R.M.F*-IBM-MQ-Java-InstallRA.jar，例如 9.0.0.0-IBM-MQ-Java-InstallRA.jar。

已经下载和安装资源适配器之后，您就可以在 WebSphere Liberty 中进行配置了。

过程

1. 从 Fix Central 下载 IBM MQ 资源适配器。
   a) 单击此链接：IBM MQ 资源适配器。
   b) 在显示的可用修订列表中查找您的 IBM MQ 版本的资源适配器。
      例如：

```
release level: 9.1.4.0-IBM-MQ-Java-InstallRA
Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
Servers
```

      然后，单击资源适配器文件名并执行下载过程。
2. 从您下载文件的目录，输入以下命令来启动安装。
   从 IBM MQ 9.0 开始，命令的格式如下所示：

```
java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
```

   其中 *V.R.M.F* 是版本、发行版、修订版和修订包编号，*V.R.M.F*-IBM-MQ-Java-InstallRA.jar 是从 Fix Central 下载的文件的名称。
   例如，要安装 V9.1.4.0 版本的 IBM MQ 资源适配器，您将使用以下命令：

```
java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
```

   注: 要执行此安装，您必须在机器上安装 JRE 并将其添加到系统路径。

   在您输入命令时，将显示以下信息：

```
使用、解压缩或安装 IBM MQ V9.1 之前，您必须接受下述协议中的条款：
1 的条款。 IBM 国际评估许可协议
程序 2。 IBM 国际程序许可协议和其他
许可信息。 请仔细阅读以下许可协议。

使用
--viewLicenseInfo 选项可以单独查看许可协议。
按“回车”以立即显示许可条款，或按“x”跳过。
```

3. 阅读并且接受许可条款：
   a) 要显示许可，按 Enter 键。
      或者，按 x 跳过显示许可。
      显示许可之后或者紧跟着选择 X 之后，将显示以下消息，告知您可以选择显示其他许可条款：

```
使用
--viewLicenseInfo 选项可以单独查看其他许可信息。
按“回车”以立即显示附加许可信息，或按“x”跳过。
```

   b) 要显示其他许可条款，请按 Enter 键。
      或者，按 x 跳过显示其他许可条款。
      在显示其他许可条款或者紧跟着选择 X 之后，将显示以下消息，要求您接受许可协议：

```
通过选择下面的“我同意”选项，您同意
许可协议条款和非 IBM 条款（如果适用）。 如果不
选择“我不同意”。

选择 [1] 我同意，或 [2] 我不同意：
```

   c) 要接受许可协议并且继续选择安装目录，请选择 1。
      或者还可以选择 2 立即终止安装。
      如果选择 1，将显示以下消息，要求您选择目标安装目录：

```
输入产品文件的目录或保留空白以接受缺省值。
缺省目标目录为：H:\Liberty\WMQ
是否作为产品文件的目标目录？
```

4. 指定资源适配器的安装目录：
   - 如果您想要将资源适配器安装在缺省位置，那么可不指定值直接按 Enter 键。

- 如果想要将资源适配器安装在与缺省位置不同的其他位置，指定要安装资源适配器的目录的名称，然后按 Enter 键。

在所选位置安装文件后，将显示确认消息，如下示例所示：

```
正在解压缩文件到 H:\Liberty\WMQ\wmq
成功解压缩所有产品文件。
```

在安装期间，会在所选安装目录中创建名为 wmq 的新目录，然后将以下文件安装在 wmq 目录中：

- 安装验证测试程序 `wmq.jmsra.ivt`。
- IBM MQ RAR 文件 `wmq.jmsra.rar`。

5. 在 WebSphere Liberty 中配置资源适配器。

在 Liberty 中配置资源适配器必须采取的步骤如下所示。 有关更多信息，请参阅 WebSphere Application Server 产品文档。

a) 将 wmqJmsClient-2.0 功能部件添加到 `server.xml` 文件，以便使用 IBM MQ 9.1 资源适配器。
有关更多信息，请参阅第 366 页的『要使用的资源适配器版本』。

b) 添加对已安装的 `wmq.jmsra.rar` 文件的引用。

注: 对于 WebSphere Liberty 8.5.5 Fix Pack 1 之前的 Liberty 版本，如果仅使用 `ejb-jar.xml` 中的配置来部署 EJB，那么 Liberty 概要文件使用的 WebSphere Application Server 版本必须已应用 APAR PM89890。 此配置方法用于资源适配器的安装验证程序 (IVT)，因此，为了让 IVT 运行，需要此 APAR。

JNDI 支持 servlet 和 MDB 的示例配置可能如下所示：

```
 <featureManager>
    <feature>wmqJmsClient-2.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

配置 IBM MQ 资源适配器

要配置 IBM MQ 资源适配器，您需要定义各种 Java Platform, Enterprise Edition Connector Architecture (JCA) 资源以及系统属性（可选）。 您还必须配置资源适配器以运行安装验证测试 (IVT) 程序。 这很重要，因为 IBM 服务可能需要运行此程序来指示已正确配置任何非 IBM 应用程序服务器。

开始之前

此任务假设您已经熟悉 JMS 和 IBM MQ classes for JMS。 用于配置 IBM MQ 资源适配器的许多属性与 IBM MQ classes for JMS 对象的属性等效并具有相同功能。

关于此任务

每个应用程序服务器提供各自的管理接口集。 某些应用程序服务器提供图形用户界面来定义 JCA 资源，但是其他应用程序服务器需要管理员编写 XML 部署计划。 因此，提供有关如何为各应用程序服务器配置 IBM MQ 资源适配器的信息超出本文档的范围。

以下步骤因此仅专注于需要配置的内容。 请参阅应用程序服务器随附的文档，以获取有关如何配置 JCA 资源适配器的信息。

过程

在以下类别中定义 JCA 资源：

- 定义 ResourceAdapter 对象的属性。
  这些属性（表示资源适配器的全局属性，例如诊断跟踪的级别）在第 376 页的『ResourceAdapter 对象属性的配置』中进行了描述。
- 定义 ActivationSpec 对象的属性。

这些属性确定如何为入站通信激活 MDB。 有关更多信息，请参阅第 378 页的『配置入站通信的资源适配器』。

- 定义 ConnectionFactory 对象的属性。

  应用程序服务器使用这些属性为出站通信创建 JMS ConnectionFactory 对象。 有关更多信息，请参阅第 390 页的『配置出站通信的资源适配器』。

- 定义受管目标对象的属性。

  应用程序服务器使用这些属性为出站通信创建 JMS Queue 对象或 JMS Topic 对象。 有关更多信息，请参阅第 390 页的『配置出站通信的资源适配器』。

- 可选： 定义资源适配器的部署计划。

  IBM MQ 资源适配器 RAR 文件包含名为 `META-INF/ra.xml` 的文件，其中包含资源适配器的部署描述符。 此部署描述符由 XML 模式在 https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd 中定义，并且包含有关资源适配器及其提供的服务的信息。 应用程序服务器还可能需要资源适配器的部署计划。 此部署计划特定于应用程序服务器。

根据需要指定 JVM 系统属性：

- 如果使用的是传输层安全性 (TLS)，请将密钥库文件和信任库文件的位置指定为 JVM 系统属性，如以下示例中所示：

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  这些属性不能是 ActivationSpec 或 ConnectionFactory 对象的属性，并且不能为应用程序服务器指定多个密钥库。 属性适用于整个 JVM，因此如果在应用程序服务器中运行的其他应用程序使用的是 TLS 连接，那么可能会影响应用程序服务器。 应用程序服务器也可能将这些属性重置为不同值。 有关将 TLS 与 IBM MQ classes for JMS 结合使用的更多信息，请参阅第 197 页的『对 IBM MQ classes for JMS 使用 TLS』。

- 可选： 如果需要，请配置资源适配器以将警告消息记录到应用程序服务器的标准输出日志。

  资源适配器日志、警告和错误消息使用与 IBM MQ classes for JMS 相同的机制。 有关更多信息，请参阅 IBM MQ classes for JMS 的日志记录错误。 这意味着在默认情况下，消息转至名为 `mqjms.log` 的文件。 要配置资源适配器以将日志警告消息另外记录到应用程序服务器的标准输出日志，请为应用程序服务器设置以下 JVM 系统属性：

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  这与用于控制对 IBM MQ classes for JMS 的跟踪的属性为同一属性。 与 IBM MQ classes for JMS 一样，可以使用指向 `jms.config` 文件的系统属性（请参阅 第 77 页的『IBM MQ classes for JMS 配置文件』）。 有关如何设置 JVM 系统属性的信息，请参阅应用程序服务器文档。

配置资源适配器以运行安装验证测试

- 配置资源适配器以运行随附于 IBM MQ 资源适配器的安装验证测试 (IVT) 程序。

  有关为运行 IVT 程序而需要配置的内容的信息，请参阅 第 405 页的『验证资源适配器安装』。

  这很重要，因为 IBM 服务可能需要运行此程序来指示已正确配置任何非 IBM 应用程序服务器。

  要点: 您必须先配置资源适配器，然后才能运行程序。

ResourceAdapter 对象属性的配置

ResourceAdapter 对象包含 IBM MQ 资源适配器的全局属性（例如诊断跟踪的级别）。 要定义这些属性，请使用资源适配器的工具，如应用程序服务器随附的文档中所述。

ResourceAdapter 对象具有两个属性集：

- 与诊断跟踪相关联的属性
- 与资源适配器管理的连接池相关联的属性

定义这些属性的方式取决于应用程序服务器提供的管理接口。 如果使用的是 WebSphere Application Server traditional，请参阅 第 378 页的『WebSphere Application Server traditional 配置』；如果使用的是

WebSphere Liberty，请参阅 第 378 页的『WebSphere Liberty 配置』。 对于其他应用程序服务器，请参阅您的应用程序服务器的产品文档。

有关定义与诊断跟踪相关联属性的更多信息，请参阅跟踪 IBM MQ 资源适配器

资源适配器管理用于将消息传送到 MDB 的 JMS 连接的内部连接池。 第 377 页的表 62 列出与连接池相关联的 ResourceAdapter 对象的属性。

表 *62:* 与连接池相关联的 *ResourceAdapter* 对象的属性

属性的名称	类型	缺省值	描述
maxConnections	字符串	50	与 IBM MQ 队列管理器的连接的最大数量和已部署的 MDB 的最大数量。
connectionConcurrency	字符串	1	共享 JMS 连接的 MDB 的最大数量。 无法共享连接，并且此属性始终具有值 1。
reconnectionRetryCount	字符串	5	资源适配器为重新连接到 IBM MQ 队列管理器（如果连接失败）进行的最大尝试数。
reconnectionRetryInterval	字符串	300 000	资源适配器在尝试重新连接到 IBM MQ 队列管理器之前等待的时间（以毫秒为单位）。
startupRetryCount	字符串	0	在启动时尝试连接 MDB 的缺省次数（如果启动应用程序服务器时队列管理器未在运行）。
startupRetryInterval	字符串	30 000	两次启动连接尝试之间间隔的缺省休眠时间（以毫秒为单位）。
supportMQExtensions	字符串	false	将 IBM MQ JMS 行为还原为 JMS 2.0 之前版本的行为。 有关更多信息，请参阅第 265 页的『SupportMQExtensions 属性』。
nativeLibraryPath	字符串	<空>	用于装入 IBM MQ JNI 库以允许绑定方式连接的路径。 Windows 在 Windows 上，系统路径还需要包含匹配的 IBM MQ 安装的位置。

在应用程序服务器中部署 MDB 时，将会创建新 JMS 连接并启动与队列管理器的对话，前提是不超过 maxConnection 属性指定的最大连接数。 因此，MDB 的最大数量等于最大连接数。 如果已部署的 MDB 的数量达到此最大值，那么部署其他 MDB 的任何尝试都将失败。 如果某个 MDB 停止，那么其连接可供其他 MDB 使用。

一般而言，如果要部署许多 MDB，那么必须增大 maxConnections 属性的值。

reconnectionRetryCount 和 reconnectionRetryInterval 属性监管在与 IBM MQ 队列管理器的连接失败（例如，由于网络故障）时资源适配器的行为。 当连接失败时，资源适配器在 reconnectionRetryInterval 属性指定的时间间隔内暂挂指向该连接提供的所有 MDB 的消息传送。 然后，资源适配器尝试重新连接到队列管理器。 如果尝试失败，那么资源适配器按 reconnectionRetryInterval 属性指定的时间间隔进一步尝试重新连接，直至达到 reconnectionRetryCount 属性施加的限制。 如果所有尝试都失败，那么传送永久停止，直至手动重新启动 MDB。

一般而言，ResourceAdapter 对象无需管理。 不过，要在诸如 UNIX and Linux 的系统上启用诊断跟踪，可以设置以下属性：

```
traceEnabled:    true
traceLevel:      10
```

如果还未启动资源适配器，那么这些属性无效（例如，当使用 IBM MQ 资源的应用程序仅在客户机容器中运行时就会发生这种情况）。 在此情况下，可以将诊断跟踪的属性设置为 Java 虚拟机 (JVM) 系统属性。 可以通过在 **java** 命令上使用 -D 标志来设置属性，如以下示例所示：

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

无需定义 ResourceAdapter 对象的所有属性。 保留未指定的任何属性都采用其缺省值。 在受管环境中，最好不要混合使用两种指定属性的方式。 如果一定要混用，那么 JVM 系统属性优先于 ResourceAdapter 对象的属性。

WebSphere Application Server traditional 配置

对于 WebSphere Application Server traditional 中的资源适配器，提供了相同的属性，但应该在资源适配器的属性面板中设置这些属性 (请参阅 WebSphere Application Server traditional 产品文档中的 JMS 提供程序设置)。 跟踪由 WebSphere Application Server traditional 配置的诊断部分控制。 有关更多信息，请参阅 WebSphere Application Server traditional 产品文档中的 使用诊断提供程序 。

WebSphere Liberty 配置

可使用 `server.xml` 文件中的 XML 元素来配置资源适配器，如以下示例中所示：

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

通过添加以下 XML 元素来启用跟踪：

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

配置入站通信的资源适配器

要配置入站通信，请定义一个或多个 ActivationSpec 对象的属性。

ActivationSpec 对象的属性确定消息驱动 Bean (MDB) 如何从 IBM MQ 队列接收 JMS 消息。 MDB 的事务行为在其部署描述符中进行定义。

ActivationSpec 对象具有两个属性集：

- 用于创建与 IBM MQ 队列管理器的 JMS 连接的属性
- 用于创建在消息到达指定队列时对其进行异步传送的 JMS 连接使用者的属性

定义 ActivationSpec 对象的属性的方式取决于应用程序服务器提供的管理接口。

JMS 2.0 中的新 ActivationSpec 属性

JMS 2.0 规范引入两个新的 ActivationSpec 属性。 可以通过优先于其他 ActivationSpec 属性进行使用的受管对象的 JNDI 名称来提供 connectionFactoryLookup 和 destinationLookup 属性。

例如，假设在 JNDI 中定义了连接工厂，并在激活规范的 connectionFactoryLookup 属性中指定了该对象的 JNDI 名称。 JNDI 中为连接工厂定义的所有属性优先于第 379 页的表 63 中的属性进行使用。

如果在 JNDI 中定义了目标，并且在 ActivationSpec 的 destinationLookup 属性中设置了 JNDI 名称，那么使用的值优先于 第 385 页的表 64 中的值。 有关如何使用这两个属性的更多信息，请参阅 第 388 页的『ActivationSpec connectionFactoryLookup 和 destinationLookup 属性』。

用于创建与 IBM MQ 队列管理器的 JMS 连接的属性

第 379 页的表 63 中的所有属性都是可选的。

表 *63:* 用于创建 *JMS* 连接的 *ActivationSpec* 对象的属性

属性的名称	类型	有效值（缺省值为粗体形式）	描述
applicationName	字符串	• 调用类名（如果可用）调整为长度不超过 28 个字符。 如果它不可用，那么使用字符串 `WebSphere MQ Client for Java`。	应用程序向队列管理器进行注册所使用的名称。 此应用程序名称由 **DISPLAY CONN MQSC/PCF** 命令显示 (其中该字段称为 **APPLTAG**) 或在 IBM MQ Explorer **应用程序连接** 屏幕中 (其中该字段称为 **App name**)。
brokerCCDurSubQueue [1]	字符串	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • 队列名称	连接使用者从中接收持久预订消息的队列的名称
brokerCCSubQueue [1]	字符串	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • 队列名称	连接使用者从中接收非持久预订消息的队列的名称
brokerControlQueue [1]	字符串	• **SYSTEM.BROKER.CONTROL.QUEUE** • 队列名称	代理程序控制队列的名称
brokerQueueManager [1]	字符串	• **""（空字符串）** • 队列管理器名称	代理程序运行所在的队列管理器的名称
brokerSubQueue [1]	字符串	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • 队列名称	非持久消息使用者从中接收消息的队列的名称
brokerVersion [1]	字符串	• **未指定** - 在代理程序从 V6 迁移到 V7 之后，设置此属性，以便不再使用 RFH2 头。 迁移后，此属性不再相关。 • **V1** - 使用 IBM MQ 发布/预订代理程序。如果 TRANSPORT 设置为 BIND 或 CLIENT，那么此值为缺省值。 • **V2** - 以本机方式使用 IBM Integration Bus 的代理程序。 如果 TRANSPORT 设置为 DIRECT 或 DIRECTHTTP，那么此值为缺省值。	正在使用的代理程序的版本
ccdtURL	字符串	• **null** • 统一资源定位符 (URL)	一个 URL，用于标识包含客户机通道定义表 (CCDT) 的文件的名称和位置并指定如何可以访问该文件
CCSID	字符串	• **819** • Java 虚拟机 (JVM) 支持的编码字符集标识	连接的编码字符集标识
通道	字符串	• **SYSTEM.DEF.SVRCONN** • MQI 通道的名称	要使用的 MQI 通道的名称

表 63: 用于创建 JMS 连接的 ActivationSpec 对象的属性 (继续)			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
cleanupInterval [1]	int	• **3 600 000** • 正整数	发布/预订清除实用程序的后台运行之间的时间间隔（以毫秒为单位）
cleanupLevel [1]	字符串	• **SAFE** • 无 • STRONG • FORCE • NONDUR	基于代理程序的预订库的清除级别
clientID	字符串	• **null** • 客户机标识	连接的客户机标识
cloneSupport	字符串	• **DISABLED** - 一次只能运行持久主题订户的一个实例。 • ENABLED - 同一持久主题订户的两个或更多实例可以同时运行，但是各实例必须在单独的 Java 虚拟机 (JVM) 中运行。	同一持久主题订户的两个或更多实例是否可以同时运行
connectionFactoryLookup	字符串	• **null** • ConnectionFactory 对象的 JNDI 名称	如果设置了此属性，那么 ActivationSpec 会在应用程序服务器的 JNDI 名称空间中查找具有指定 JNDI 名称的 JMS ConnectionFactory 对象，然后使用该对象的属性创建与 IBM MQ 队列管理器的 JMS 连接，但有一个异常。 创建 JMS 连接时将使用的 ActivationSpec 中唯一属性是 clientID。 有关更多信息，请参阅 第 388 页的『ActivationSpec connectionFactoryLookup 和 destinationLookup 属性』。

表 63: 用于创建 JMS 连接的 ActivationSpec 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
connectionNameList	字符串	• **localhost(1414)** • 由以逗号分隔的项组成的字符串，其中各项采用以下格式： `HOSTNAME(PORT)` 其中 *HOSTNAME* 是 DNS 名称或 IP 地址。	用于入站通信的 TCP/IP 连接名称列表。 在指定时，**connectionNameList** 取代 **hostname** 和 **port** 属性。 此属性用于重新连接到多实例队列管理器。 **connectionNameList** 在形式上类似于 **localAddress**，但是不得与其混淆。 **localAddress** 指定本地通信的特征，而 **connectionNameList** 指定如何访问远程队列管理器。
failIfQuiesce	布尔	• **true** • false	如果队列管理器处于停顿状态，那么对某些方法的调用是否失败
headerCompression	字符串	• **NONE** • SYSTEM - 执行 RLE 消息头压缩	可用于压缩连接上的头数据的方法列表
hostName	字符串	• **localhost** • 主机名 • IP 地址	队列管理器所在系统的主机名或 IP 地址。 在指定时，**hostname** 和 **port** 属性由 **connectionNameList** 属性取代。
localAddress	字符串	• **null** • 以下格式的字符串： `[ host_name ][(low_port [, high_port ])]` 其中 *host_name* 是主机名或 IP 地址，*low_port* 和 *high_port* 是 TCP 端口号，括号表示可选组件	对于与队列管理器的连接，此属性指定以下任一项或同时指定两项： • 要使用的本地网络接口 • 要使用的本地端口或本地端口范围 **localAddress** 在形式上类似于 **connectionNameList**，但是不得与其混淆。 **localAddress** 指定本地通信的特征，而 **connectionNameList** 指定如何访问远程队列管理器。

表 63: 用于创建 JMS 连接的 ActivationSpec 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
messageCompression	字符串	• **NONE** • 以空白字符分隔的下列一个或多个值的列表: RLE ZLIBFAST ZLIBHIGH	可用于压缩连接上的消息数据的方法列表
messageRetention [1]	布尔	• **true** - 不需要的消息保留在输入队列上 • false - 根据不需要的消息的处置选项对其进行处理	连接使用者是否将不需要的消息保留在输入队列上
messageSelection [1]	字符串	• **CLIENT** • BROKER	确定由 IBM MQ classes for JMS 还是由代理程序进行消息选择。 当 brokerVersion 值为 1 时，不支持由代理程序进行消息选择。
密码	字符串	• **null** • 密码	创建与队列管理器的连接时要使用的缺省密码
pollingInterval [1]	int	• **5000** • 任何正整数	如果会话中的各消息侦听器在其队列上没有合适的消息，那么此值是各消息侦听器再次尝试从其队列中获取消息前经过的最大时间间隔（以毫秒为单位）。 如果没有合适的消息可用于会话中的任何消息侦听器的情况频繁发生，请考虑增大此属性的值。 仅当 TRANSPORT 的值为 BIND 或 CLIENT 时，此属性才相关。
port	int	• **1414** • TCP 端口号	队列管理器在其上进行侦听的端口。 在指定时，**hostname** 和 **port** 属性由 **connectionNameList** 属性取代。
providerVersion	字符串	• **unspecified** • 以下格式之一的字符串 – V.R.M.F – V.R.M – V.R – V 其中 V、R、M 和 F 是大于或等于零的整数值。	MDB 计划连接到的队列管理器的版本、发行版、修改级别和修订包。

表 63: 用于创建 JMS 连接的 ActivationSpec 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
queueManager	字符串	• **""（空字符串）** • 队列管理器名称	要连接到的队列管理器的名称
receiveExit [3]	字符串	• **null** • 由以逗号分隔的一项或多项组成的字符串，其中各项是实现 IBM MQ classes for Java 接口 MQReceiveExit 的类的标准名称	标识通道接收出口程序或要连续运行的一系列接收出口程序
receiveExitInit	字符串	• **null** • 由以逗号分隔的一项或多项用户数据组成的字符串	调用通道接收出口程序时传递到这些程序的用户数据
rescanInterval [1]	int	• **5000** • 任何正整数	当点到点域中的消息使用者使用消息选择器来选择要接收的消息时， IBM MQ classes for JMS 将按队列的 **MsgDeliverySequence** 属性确定的顺序在 IBM MQ 队列中搜索适合的消息。 当 IBM MQ classes for JMS 找到合适的消息并将其传递到使用者时，IBM MQ classes for JMS 从它在队列中的当前位置继续搜索下一条合适消息。 IBM MQ classes for JMS 继续以此方式搜索队列，直至到达队列的结尾，或者直至此属性的值所确定的时间间隔（以毫秒为单位）已到期。 在每种情况下，IBM MQ classes for JMS 会返回到队列的开头以继续其搜索，并且开始新的时间间隔。
securityExit [3]	字符串	• **null** • 实现 IBM MQ classes for Java 接口 MQSecurityExit 的类的标准名称	标识通道安全出口程序
securityExitInit	字符串	• **null** • 用户数据的字符串	调用通道安全出口程序时传递到该程序的用户数据
sendExit [3]	字符串	• **null** • 由以逗号分隔的一项或多项组成的字符串，其中各项是实现 IBM MQ classes for Java 接口 MQSendExit 的类的标准名称	标识通道发送出口程序或要连续运行的一系列发送出口程序
sendExitInit	字符串	• **null** • 由以逗号分隔的一项或多项用户数据组成的字符串	调用通道发送出口程序时传递到这些出口程序的用户数据

表 63: 用于创建 *JMS* 连接的 *ActivationSpec* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
shareConvAllowed	布尔	• NO-客户机连接不能共享其套接字。 • **YES** -客户机连接可以共享其套接字。	如果通道定义匹配，那么客户机连接是否可以将其套接字与从同一进程到同一队列管理器的其他顶级 JMS 连接共享
sparseSubscriptions [1]	布尔	• **false** - 预订接收常用匹配消息。 • true - 预订接收不常用匹配消息。 该值要求可以打开预订队列以供浏览。	控制 TopicSubscriber 对象的消息检索策略
sslCertStores	字符串	• **null** • 以空格分隔的一个或多个 LDAP URL 组成的字符串。 各 LDAP URL 采用以下格式： `ldap://host_name [: port ]` 其中 *host_name* 是主机名或 IP 地址，*port* 是 TCP 端口号，括号表示可选组件。	轻量级目录访问协议 (LDAP) 服务器，其中存放供在 TLS 连接上使用的证书撤销列表 (CRL)
sslCipherSuite	字符串	• **null** • CipherSuite 的名称	要用于 TLS 连接的 CipherSuite
sslFipsRequired [2]	布尔	• **false** • true	TLS 连接是否必须使用 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS) 支持的 CipherSuite
sslPeerName	字符串	• **null** • 专有名称的模板	对于 TLS 连接，用于检查由队列管理器提供的的数字证书中的专有名称的模板
sslResetCount	int	• **0** • 范围 0 - 999 999 999 中的整数	在重新协商 TLS 使用的密钥之前由 TLS 连接发送和接收的总字节数
sslSocketFactory	字符串	表示提供 javax.net.ssl.SSLSocketFactory 接口实现的类的标准类名的字符串。 （可选）包括要传递到构造方法的自变量，用括号括起来。	受管对象的范围内建立的任何连接使用从 SSLSocketFactory 接口的此实现获取的套接字。
statusRefreshInterval [1]	int	• **60000** • 任何正整数	长时间运行的事务的刷新之间的时间间隔（以毫秒为单位），该事务检测订户何时断开与队列管理器的连接。 仅当 **subscriptionStore** 的值为 QUEUE 时，此属性才相关。
subscriptionStore [1]	字符串	• **BROKER** • MIGRATE • 队列	确定 IBM MQ classes for JMS 存储有关活动预订的持久数据的位置

表 63: 用于创建 JMS 连接的 ActivationSpec 对象的属性 (继续)			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
transportType	字符串	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	与队列管理器的连接使用客户机方式还是绑定方式。 如果指定值 BINDINGS_THEN_CLIENT，那么资源适配器首先尝试以绑定方式进行连接。 如果此连接尝试失败，那么资源适配器将尝试建立客户机方式连接。 z/OS 如果在 WebSphere Application Server for z/OS 系统上运行的激活规范已配置为使用 BINDINGS_THEN_CLIENT 传输方式，并且先前建立的连接已中断，那么由此激活规范所作的任何重新连接尝试都会首先尝试使用 BINDINGS 传输方式。 如果 BINDINGS 传输方式连接尝试不成功，那么激活规范随后会尝试进行 CLIENT 传输方式连接。
username	字符串	• **null** • 用户名	创建与队列管理器的连接时要使用的缺省用户名
wildcardFormat	字符串	• CHAR - 仅识别代理程序版本 1 中所使用的字符通配符 • **TOPIC** - 仅识别代理程序版本 2 中所使用的主题级别通配符	要使用的通配符语法版本

注意:

1. 此属性可以与 IBM MQ classes for JMS 的 V 7.0 配合使用。 除非 **providerVersion** 属性设置为小于 7 的版本号，否则不会影响连接到 IBM WebSphere MQ 7.0 队列管理器的应用程序。
2. 有关使用 sslFipsRequired 属性的重要信息，请参阅 第 369 页的『IBM MQ 资源适配器的限制』。
3. 有关如何配置资源适配器以使其可以找到出口的信息，请参阅第 230 页的『配置 IBM MQ classes for JMS 以使用通道出口』。

用于创建 JMS 连接使用者的属性

注: 必须显式定义 **destination** 和 **destinationType**。 第 385 页的表 64 中的所有其他属性都是可选的。

表 64: 用于创建 JMS 连接使用者的 ActivationSpec 对象的属性			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
destination	字符串	目标名称	要从中接收消息的目标。 **useJNDI** 属性确定如何解释此属性值。

表 64: 用于创建 *JMS* 连接使用者的 *ActivationSpec* 对象的属性 *(*继续*)*

属性的名称	类型	有效值（缺省值为粗体形式）	描述
destinationLookup	字符串	• **null** • 目标对象的 JNDI 名称	如果已设置此属性，那么 ActivationSpec 将在应用程序服务器的 JNDI 名称空间中查找具有指定 JNDI 名称的 JMS 目标对象，然后使用该对象的属性来创建 JMS 连接使用者，优先级高于 ActivationSpec 上指定的其他属性。 有关更多信息，请参阅第 388 页的『ActivationSpec connectionFactoryLookup 和 destinationLookup 属性』。
destinationType	字符串	• javax.jms.Queue • javax.jms.Topic	目标（队列或主题）的类型
maxMessages	int	• **1** • 正整数	一次可以向服务器会话分配的最大消息数。 如果激活规范正在 XA 事务中向 MDB 传送消息，那么无论此属性的设置如何，都会使用值 1。
maxPoolDepth	int	• **10** • 正整数	连接使用者所使用的服务器会话池中的最大服务器会话数
messageSelector	字符串	• **null** • SQL92 消息选择器表达式	指定要传送哪些消息的消息选择器表达式
nonASFTimeout	int	• **0** • 正整数	正值指示使用非 ASF 传送。 该值表示 Get 请求等待可能尚未到达的消息（带等待调用的 Get）的时间（以毫秒为单位）。 缺省值 0 指示使用 ASF 传送。 在下列情况下，此参数有效: • 应用程序正在 WebSphere Application Server 7.0 或更高版本上运行。 • 应用程序正在 WebSphere Liberty 中使用相应级别的 wmqJmsClient 功能部件运行。 有关更多信息，请参阅第 370 页的『Liberty 和 IBM MQ 资源适配器』。
nonASFRollbackEnabled	布尔	• **false** - 即使 MDB 失败，也会消耗消息 • true - MDB 中的故障导致消息回滚到队列。	如果 MDB 是非事务性，那么消息传送是否在 IBM MQ 同步点内。 如果 MDB 是事务性，或者如果 **nonASFTimeout** 设置为 0，那么将会忽略。
poolTimeout	int	• **300000** • 正整数	未使用的服务器会话在由于不活动而关闭前，在服务器会话池中保持打开状态的时间（以毫秒为单位）

表 64: 用于创建 JMS 连接使用者的 ActivationSpec 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
readAheadAllowed	int	• **DESTINATION** - 通过引用队列或主题定义来确定是否允许预读。 • DISABLED - 不允许预读。 • ENABLED - 允许预读。 • QUEUE - 通过引用队列定义来确定是否允许预读。 • TOPIC - 通过引用主题定义来确定是否允许预读。	是否允许 MDB 在接收来自目标的非持久消息之前使用预读将其放入内部缓冲区
readAheadClosePolicy	int	• **ALL** - 在 MDB 停止之前，内部预读缓冲区中的所有消息都传送到 MDB。 • CURRENT - 仅当前 MDB 调用完成，从而可能将消息保留在内部预读缓冲区中，然后将丢弃这些消息。	管理员停止 MDB 后，内部预读缓冲区中的消息发生的情况。
receiveCCSID	int	• **0** - 使用 JVM `Charset.defaultCharset` • 1208 - UTF-8 • 支持的编码字符集标识	用于设置队列管理器消息转换的目标 CCSID 的目标属性。 除非 **receiveConversion** 设置为 QMGR，否则忽略该值。
receiveConversion	字符串	• **CLIENT_MSG** • QMGR	用于确定数据转换即将由队列管理器执行的目标属性。
sharedSubscription	布尔	• **False** - MDB 不应将预订作为共享的预订打开。 • True-MDB 应将预订作为共享预订打开 (使用 JMS 2.0 所暗示的规则，请参阅 Java.net 上的 JMS 2.0 规范)。	控制如何从共享的预订驱动 MDB。 有关如何使用此属性的更多信息，请参阅第 390 页的『如何定义 sharedSubscription 属性的示例』。
startTimeout	int	• **10 000** • 正整数	以毫秒为单位的时间，在该时间内，在已调度传送消息的工作之后必须启动消息到 MDB 的传送。 如果经过此时间，那么消息回滚到队列上。
subscriptionDurability	字符串	• **NonDurable** - 使用非持久预订将消息传送到预订主题的 MDB。 • Durable - 使用持久预订将消息传送到预订主题的 MDB。	使用持久还是非持久预订将消息传送到预订主题的 MDB
subscriptionName	字符串	• **""（空字符串）** • 预订名称	持久预订的名称

表 64: 用于创建 JMS 连接使用者的 ActivationSpec 对象的属性 (继续)			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
useJNDI	布尔	• **false** - 名为 destination 的属性解释为 IBM MQ 队列或主题的名称。 • true - 名为 destination 的属性解释为应用程序服务器的 JNDI 名称空间中的 javax.jms.Queue 对象或 javax.jms.Topic 对象的名称。	确定如何解释名为 destination 的属性的值 **注:** IBM MQ 9.0 中不推荐使用此属性。应改用 destinationLookup 属性。

属性冲突和依赖关系

ActivationSpec 对象可能具有冲突的属性。 例如，可以指定绑定方式下的连接的 TLS 属性。 在此情况下，该行为由传输类型和消息传递域确定，它可以是 **destinationType** 属性确定的点到点或发布/预订。 将忽略不适用于指定传输类型或消息传递域的任何属性。

如果定义的属性要求定义其他属性，但是您不定义这些其他属性，那么在 MDB 部署期间调用 ActivationSpec 对象的 validate() 方法时该对象将抛出 InvalidPropertyException 异常。 异常以根据应用程序服务器而定的方式报告给应用程序服务器的管理员。 例如，如果将 subscriptionDurability 属性设置为 Durable，表示您想要使用持续预订，那么还必须定义 **subscriptionName** 属性。

如果定义了名为 **ccdtURL** 和 **channel** 的属性，那么会抛出 InvalidPropertyException 异常。 但是，如果仅定义 **ccdtURL** 属性，那么将名为 **channel** 的属性保留为其缺省值 SYSTEM.DEF.SVRCONN，不会抛出异常，并且使用 **ccdtURL** 属性标识的客户机通道定义表来启动 JMS 连接。

ActivationSpec connectionFactoryLookup 和 destinationLookup 属性

这两个属性可用于指定 ConnectionFactory 和 Destination 对象的 JNDI 名称，这些对象优先于 第 379 页的表 63 和 第 385 页的表 64 中定义的 ActivationSpec 的属性。

请务必注意以下要点，其中详细描述了这些属性的工作方式。

connectionFactoryLookup
从 JNDI 查找的 ConnectionFactory 用作第 379 页的表 63 中所列的属性的源。 ConnectionFactory 对象未用于实际创建任何 JMS 连接，仅会查询该对象的属性。 来自 ConnectionFactory 的这些属性会覆盖 ActivationSpec 上定义的任何属性。 对此存在一个例外情况。 如果 ActivationSpec 设置了 **ClientID** 属性，那么此属性的值将覆盖 ConnectionFactory 中指定的值。 这是因为常见的场景是将一个 ConnectionFactory 用于多个 ActivationSpec。 这可简化管理过程。 但是，JMS 2.0 规范指出从 ConnectionFactory 创建的每个 JMS 连接均应具有唯一的 **ClientID**。 因此，ActivationSpec 需要能够覆盖 ConnectionFactory 上设置的任何值。 如果在 ActivationSpec 上未设置 **ClientID**，那么将使用连接工厂上的任何值。

destinationLookup
ActivationSpec 上已定义 **Destination** 和 **UseJndi** 属性。 如果将 **UseJndi** 标志设置为 true，那么在目标属性中指定的文本将视为 JNDI 名称，并且将从 JNDI 查找具有该 JNDI 名称的目标对象。

destinationLookup 属性的行为方式完全相同。 如果已设置此属性，那么将从 JNDI 查找具有由此属性指定的 JNDI 名称的目标对象。 此属性优先于 **useJNDI** 属性。

在 IBM MQ 9.0 中不推荐使用 useJNDI 属性，因为 **destinationLookup** 属性是执行相同功能的 JMS 2.0 等效指定项。

在 IBM MQ classes for JMS 中无等效项的 ActivationSpec 属性

ActivationSpec 对象的大多数属性与 IBM MQ classes for JMS 对象的属性或 IBM MQ classes for JMS 方法的参数等效。 但是，三个调整属性和一个易用性属性在 IBM MQ classes for JMS 中没有等效项：

startTimeout
在资源适配器调度工作对象将消息传送到 MDB 之后，应用程序服务器的工作管理器等待资源可供使用的时间（以毫秒为单位）。 如果在消息传送开始之前经过此时间，那么工作对象超时，消息回滚到队列上，然后资源适配器可尝试再次传送消息。 警告写入到诊断跟踪（如果已启用），但是不会以其他方式影响传送消息的过程。 您可能预期仅在应用程序服务器负载非常高时才会发生此情况。 如果定期发生该情况，请考虑增大此属性的值来为工作管理器提供更长时间调度消息传送。

maxPoolDepth
连接使用者所使用的服务器会话池中的最大服务器会话数。 创建服务器会话后，它会启动与队列管理器的对话。 连接使用者使用服务器会话将消息传送到 MDB。 较大的池深度允许在有大量消息情况下并发传送更多消息，但会使用应用程序服务器的更多资源。 如果要部署许多 MDB，请考虑减小池深度，以便将应用程序服务器上的负载维持在可管理级别。 各连接使用者使用其自己的服务器会话池，以便此属性不定义可用于所有连接使用者的服务器会话的总数。

poolTimeout
未使用的服务器会话在由于不活动而关闭前，在服务器会话池中保持打开状态的时间（以毫秒为单位）。 消息工作负载的瞬态增加导致创建其他服务器会话，以便分发负载，但在消息工作负载恢复正常之后，其他服务器会话保留在池中且未使用。

每次使用服务器会话时，将使用时间戳记对其进行标记。 清扫器线程定期检查在此属性指定的时间段内是否已使用各服务器会话。 如果未使用服务器会话，那么会将其关闭并从服务器会话池中移除。 在经过指定的时间段之后，服务器会话可能不会立即关闭，此属性表示移除之前的最小不活动时间段。

useJNDI
有关此属性的描述，请参阅第 385 页的表 64。

部署 MDB

要部署 MDB，请首先定义 ActivationSpec 对象的属性，从而指定 MDB 所需的属性。 以下示例是您可能显式定义的典型属性集：

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

应用程序服务器使用属性来创建 ActivationSpec 对象，该对象然后与 MDB 相关联。 ActivationSpec 对象的属性确定如何将消息传送到 MDB。 如果 MDB 需要分布式事务，但是资源适配器不支持分布式事务，那么 MDB 的部署失败。 有关如何安装资源适配器以便支持分布式事务的信息，请参阅第 372 页的『安装 IBM MQ 资源适配器』。

如果多个 MDB 从同一目标接收消息，那么在点到点域中发送的消息仅由一个 MDB 接收，即使其他 MDB 有资格接收该消息也如此。 特别是，如果两个 MDB 使用的是不同的消息选择器，并且入局消息与两个消息选择器均匹配，那么仅其中一个 MDB 接收消息。 选择用于接收消息的 MDB 未定义，因此无法依靠特定 MDB 接收消息。 在发布/预订域中发送的消息由所有符合资格的 MDB 接收。

在某些情况下，传送到 MDB 的消息可能会回滚到 IBM MQ 队列上。 例如，如果在之后将回滚的工作单元中传送消息，那么可能会发生此回滚。 回滚的消息会再次传送，但是错误格式化的消息可能会重复导致 MDB 失败，因此无法传送。 此类消息称为有害消息。 您可以配置 IBM MQ，以便 IBM MQ classes for JMS 自动将有害消息传送到其他队列以进一步调查，或者丢弃消息。

有关如何处理有害消息的详细信息，请参阅第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』。

相关任务
指定运行时在 MQI 客户机上仅使用经过 FIPS 认证的 CipherSpecs
在 WebSphere Application Server 中配置 JMS 资源
相关参考
适用于 UNIX、Linux 和 Windows 的美国联邦信息处理标准 (FIPS)

如何定义 *sharedSubscription* 属性的示例
您可以在 WebSphere Liberty `server.xml` 文件中定义激活规范的 sharedSubscription 属性。 或者，您可以使用注释在消息驱动的 Bean (MDB) 中定义属性。

示例：在 Liberty `server.xml` 文件中定义

在 WebSphere Liberty `server.xml` 文件中，定义激活规范，如以下示例中所示。 此示例在本地主机/端口 1490 上创建队列管理器的持久共享预订。

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

示例，在 MDB 中定义

您还可以使用注释在 MDB 中定义 sharedSubscription 属性，如下面的示例所示：

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

下面的示例显示使用注释方法的 MDB 代码部分：

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

相关概念
订户和预订
预订耐久性
第 264 页的『克隆和共享的预订』
在 IBM MQ 8.0 或更高版本中，可以使用两种方法提供对同一个预订的多使用者访问权。 这两种方法是：使用克隆的预订或使用共享的预订。

配置出站通信的资源适配器

要配置出站通信，请定义 ConnectionFactory 对象和受管目标对象的属性。

出站通信使用示例

使用出站通信时，在应用程序服务器中运行的应用程序启动与队列管理器的连接，然后以同步方式将消息发送到其队列并从其队列接收消息。 例如，以下 servlet 方法 doGet() 使用出站通信：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

在 servlet 收到 HTTP GET 请求时，它从 JNDI 名称空间检索 ConnectionFactory 对象和 Queue 对象，并且使用这些对象将消息发送到 IBM MQ 队列。 然后，servlet 接收其已发送的消息。

出站通信所需的资源

要配置出站通信，请在以下类别中定义 Java EE Connector Architecture (JCA) 资源：

- ConnectionFactory 对象的属性，供应用程序服务器用于创建 JMS ConnectionFactory 对象。
- 受管目标对象的属性，供应用程序服务器用于创建 JMS Queue 对象或 JMS Topic 对象。

定义这些属性的方式取决于应用程序服务器提供的管理接口。 应用程序服务器创建的 ConnectionFactory、Queue 和 Topic 对象绑定到 JNDI 名称空间中，应用程序可从中对这些对象进行检索。

通常，为应用程序可能需要连接到的各队列管理器定义一个 ConnectionFactory 对象。 为应用程序可能需要在点到点域中访问的各队列定义一个 Queue 对象。 并且，为应用程序可能要发布到或预订的各主题定义一个 Topic 对象。 ConnectionFactory 对象可以独立于域。 或者，它可以特定于域，针对点到点域使用 QueueConnectionFactory 对象，或者针对发布/预订域使用 TopicConnectionFactory 对象。

提示: 通过 JMS 2.0，连接工厂可同时用于创建连接和上下文。 因此，连接池可以与包含连接和上下文的混合的连接工厂相关联。 建议连接工厂仅用于创建连接或创建上下文。 这确保该连接工厂的连接池仅包含单一类型的对象，从而使池更高效。

ConnectionFactory 对象的属性

第 392 页的表 65 列出 ConnectionFactory 对象的属性。 应用程序服务器使用这些属性来创建 JMS ConnectionFactory 对象。

表 65: *ConnectionFactory* 对象的属性			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
applicationName	字符串	• 调用类名（如果可用）调整为长度不超过 28 个字符。 如果它不可用，那么使用字符串 `WebSphere MQ Client for Java`。	应用程序向队列管理器进行注册所使用的名称。 此应用程序名称由 **DISPLAY CONN MQSC/PCF** 命令显示（其中，字段名为 **APPLTAG**），或者在 IBM MQ Explorer“**应用程序连接**”显示屏幕中显示（其中，字段名为 **App name**）。
brokerCCSubQueue [1]	字符串	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • 队列名称	连接使用者从中接收非持久预订消息的队列的名称。
brokerControlQueue [1]	字符串	• **SYSTEM.BROKER.CONTROL.QUEUE** • 队列名称	代理程序控制队列的名称。
brokerPubQueue [1]	字符串	• **SYSTEM.BROKER.DEFAULT.STREAM** • 队列名称	发送发布的消息的队列的名称（流队列）。
brokerQueueManager [1]	字符串	• **""（空字符串）** • 队列管理器名称	运行代理程序的队列管理器的名称。
brokerSubQueue [1]	字符串	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • 队列名称	非持久消息使用者从中接收消息的队列的名称。 请参阅 BROKERSUBQ 属性以获取更多信息。
brokerVersion [1]	字符串	• **未指定** - 在代理程序从 V6 迁移到 V7 之后，设置此属性，以便不再使用 RFH2 头。 迁移后，此属性不再相关。 • **V1** - 使用 IBM MQ 发布/预订代理程序。 如果 TRANSPORT 设置为 BIND 或 CLIENT，那么此值为缺省值。 • **V2** - 以本机方式使用 IBM Integration Bus 的代理程序。 如果 TRANSPORT 设置为 DIRECT 或 DIRECTHTTP，那么此值为缺省值。	正在使用的代理程序的版本。
ccdtURL	字符串	• **null** • 统一资源定位符 (URL)	一个 URL，用于标识包含客户机通道定义表 (CCDT) 的文件的名称和位置并指定如何可以访问该文件。
CCSID	字符串	• **819** • Java 虚拟机 (JVM) 支持的编码字符集标识	连接的编码字符集标识。
通道	字符串	• **SYSTEM.DEF.SVRCONN** • MQI 通道的名称	要使用的 MQI 通道的名称。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
cleanupInterval [1]	int	• **3 600 000** • 正整数	发布/预订清除实用程序的后台运行之间的时间间隔（以毫秒为单位）。
cleanupLevel [1]	字符串	• **SAFE** • 无 • STRONG • FORCE • NONDUR	基于代理程序的预订库的清除级别。
clientID	字符串	• **null** • 客户机标识	连接的客户机标识。
cloneSupport	字符串	• **DISABLED** - 一次只能运行持久主题订户的一个实例。 • ENABLED - 同一持久主题订户的两个或更多实例可以同时运行，但是各实例必须在单独的 Java 虚拟机 (JVM) 中运行。	同一持久主题订户的两个或更多实例是否可以同时运行。
connectionNameList	字符串	• **localhost(1414)** • 由以逗号分隔的项组成的字符串，其中各项采用以下格式： `HOSTNAME(PORT)` 其中 *HOSTNAME* 是 DNS 名称或 IP 地址。	用于出站通信的 TCP/IP 连接名称列表。 **connectionNameList** 取代 **hostname** 和 **port** 属性。 此属性用于重新连接到多实例队列管理器。 **connectionNameList** 在形式上类似于 **localAddress**，但是不得与其混淆。 **localAddress** 指定本地通信的特征，而 **connectionNameList** 指定如何访问远程队列管理器。
failIfQuiesce	布尔	• **true** • false	如果队列管理器处于停顿状态，那么对某些方法的调用是否失败。
headerCompression	字符串	• **NONE** • SYSTEM - 执行 RLE 消息头压缩。	可用于压缩连接上的头数据的方法列表。
hostName	字符串	• **localhost** • 主机名 • IP 地址	队列管理器所在系统的主机名或 IP 地址。 在指定时，**hostname** 和 **port** 属性由 **connectionNameList** 属性取代。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
localAddress	字符串	• **null** • 以下格式的字符串： `[ host_name ][(low_port [, high_port ])]` 其中 *host_name* 是主机名或 IP 地址，*low_port* 和 *high_port* 是 TCP 端口号，括号表示可选组件	对于与队列管理器的连接，此属性指定以下任一项或同时指定两项： • 要使用的本地网络接口 • 要使用的本地端口或本地端口范围 **localAddress** 在形式上类似于 **connectionNameList**，但是不得与其混淆。 **localAddress** 指定本地通信的特征，而 **connectionNameList** 指定如何访问远程队列管理器。
messageCompression	字符串	• **NONE** • 以空白字符分隔的下列一个或多个值的列表： RLE ZLIBFAST ZLIBHIGH	可用于压缩连接上的消息数据的方法列表。
messageSelection [1]	字符串	• **CLIENT** • BROKER	确定由 IBM MQ classes for JMS 还是由代理程序进行消息选择。 当 **brokerVersion** 具有值 1 时，不支持由代理作出消息选择。
密码	字符串	• **null** • 密码	创建与队列管理器的连接时要使用的缺省密码。
pollingInterval [1]	int	• **5000** • 任何正整数	如果会话中的各消息侦听器在其队列上没有合适的消息，那么此值是各消息侦听器再次尝试从其队列中获取消息前经过的最大时间间隔（以毫秒为单位）。 如果没有合适的消息可用于会话中的任何消息侦听器的情况频繁发生，请考虑增大此属性的值。 仅当 **TRANSPORT** 的值为 BIND 或 CLIENT 时，此属性才相关。
port	int	• **1414** • TCP 端口号	队列管理器在其上进行侦听的端口。 在指定时，**hostname** 和 **port** 属性由 **connectionNameList** 属性取代。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
providerVersion	字符串	• **unspecified** • 以下格式之一的字符串 – V.R.M.F – V.R.M – V.R – V 其中 V、R、M 和 F 是大于或等于零的整数值。	应用程序计划连接到的队列管理器的版本、发行版、修改级别和修订包。
pubAckInterval [1]	int	• **25** • 正整数	在 IBM MQ classes for JMS 请求来自代理程序的应答之前由发布者发布的消息数。
queueManager	字符串	• **""（空字符串）** • 队列管理器名称	要连接到的队列管理器的名称。
receiveExit [3]	字符串	• **null** • 由以逗号分隔的一项或多项组成的字符串，其中各项是实现 IBM MQ classes for Java 接口 MQReceiveExit 的类的标准名称	标识通道接收出口程序或要连续运行的一系列接收出口程序。
receiveExitInit	字符串	• **null** • 由以逗号分隔的一项或多项用户数据组成的字符串	调用通道接收出口程序时传递到这些程序的用户数据。
rescanInterval [1]	int	• **5000** • 任何正整数	当点到点域中的消息使用者使用消息选择器来选择要接收的消息时，IBM MQ classes for JMS 按 IBM MQ 队列的 **MsgDeliverySequence** 属性确定的顺序搜索该队列以获取合适的消息。 当 IBM MQ classes for JMS 找到合适的消息并将其传递到使用者时，IBM MQ classes for JMS 从它在队列中的当前位置继续搜索下一条合适消息。 IBM MQ classes for JMS 继续以此方式搜索队列，直至到达队列的结尾，或者直至此属性的值所确定的时间间隔（以毫秒为单位）已到期。 在每种情况下，IBM MQ classes for JMS 会返回到队列的开头以继续其搜索，并且开始新的时间间隔。
securityExit [3]	字符串	• **null** • 实现 IBM MQ classes for Java 接口 MQSecurityExit 的类的标准名称	标识通道安全出口程序。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
securityExitInit	字符串	• **null** • 用户数据的字符串	调用通道安全出口程序时传递到该程序的用户数据。
sendCheckCount	int	• **0** • 任何正整数	在单个非事务性 JMS 会话内两次异步放置错误检查之间允许执行的发送调用的次数。
sendExit [3]	字符串	• **null** • 由以逗号分隔的一项或多项组成的字符串，其中各项是实现 IBM MQ classes for Java 接口 MQSendExit 的类的标准名称	标识通道发送出口程序或要连续运行的一系列发送出口程序。
sendExitInit	字符串	• **null** • 由以逗号分隔的一项或多项用户数据组成的字符串	调用通道接收出口程序时传递到这些程序的用户数据。
shareConvAllowed	布尔	• NO-客户机连接不能共享其套接字。 • **YES** -客户机连接可以共享其套接字。	如果通道定义匹配，那么客户机连接是否可以将其套接字与从同一进程到同一队列管理器的其他顶级 JMS 连接共享。
sparseSubscriptions [1]	布尔	• **false** - 预订接收常用匹配消息。 • true - 预订接收不常用匹配消息。 该值要求可以打开预订队列以供浏览。	控制 TopicSubscriber 对象的消息检索策略。
sslCertStores	字符串	• **null** • 以空格分隔的一个或多个 LDAP URL 组成的字符串。 各 LDAP URL 采用以下格式： `ldap://host_name [: port ]` 其中 *host_name* 是主机名或 IP 地址，*port* 是 TCP 端口号，括号表示可选组件。	轻量级目录访问协议 (LDAP) 服务器，其中存放供在 TLS 连接上使用的证书撤销列表 (CRL)。
sslCipherSuite	字符串	• **null** • CipherSuite 的名称	要用于 TLS 连接的 CipherSuite。
sslFipsRequired [2]	布尔	• **false** • true	TLS 连接是否必须使用 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS) 支持的 CipherSuite。
sslPeerName	字符串	• **null** • 专有名称的模板	对于 TLS 连接，用于检查由队列管理器提供的的数字证书中的专有名称的模板。
sslResetCount	int	• **0** • 范围 0 - 999 999 999 中的整数	在重新协商 TLS 使用的密钥之前由 TLS 连接发送和接收的总字节数。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
sslSocketFactory	字符串	表示提供 javax.net.ssl.SSLSocketFactory 接口实现的类的标准类名的字符串，（可选）包括要传递到构造方法的自变量，用括号括起来。	受管目标对象的范围内建立的任何连接使用从 SSLSocketFactory 接口的此实现获取的套接字。
statusRefreshInterval [1]	int	• **60000** • 任何正整数	长时间运行的事务的刷新之间的时间间隔（以毫秒为单位），该事务检测订户何时断开与队列管理器的连接。 仅当 **SUBSTORE** 的值为 QUEUE 时，此属性才相关。
subscriptionStore [1]	字符串	• **BROKER** • MIGRATE • 队列	确定 IBM MQ classes for JMS 存储有关活动预订的持久数据的位置。
targetClientMatching	布尔	• **true** • false	发送到由入局消息的 JMSReplyTo 头字段标识的队列的应答消息是否仅当该入局消息具有 MQRFH2 头时才具有 MQRFH2 头。 您还可以为激活规范配置此属性。有关更多信息，请参阅第 404 页的『为激活规范配置 targetClientMatching 属性』。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
temporaryModel	字符串	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • 任何字符串	创建 JMS 临时队列所基于的模型队列的名称。 如果以下两个语句都为 true，请使用 SYSTEM.DEFAULT.MODEL.QUEUE： • 您的应用程序使用将接受非持久消息的临时队列。 • 一次仅有一个应用程序将在 ConnectionFactory 指向的队列管理器上创建临时队列。 请注意，一次只能由一个应用程序打开 SYSTEM.DEFAULT.MODEL.QUEUE。 在下列情况下使用 SYSTEM.JMS.TEMPQ.MODEL： • 您的应用程序使用将接受持久消息的临时队列。 • 如果多个应用程序可以连接到 ConnectionFactory 指向的队列管理器，并且这些应用程序需要同时创建临时队列。 在以下情况下定义新模型队列，其中 **DEFPSIST** 属性设置为 YES，**DEFSOPT** 属性设置为 SHARED： • 您的应用程序使用将接受非持久消息的临时队列，多个应用程序将连接到 ConnectionFactory 指向的队列管理器，并且这些应用程序需要同时创建临时队列。 创建新模型队列后，将 **temporaryModel** 属性设置为新模型队列的名称。
tempQPrefix	字符串	• **""（空字符串）** • 可用于构成 IBM MQ 动态队列名称的前缀。 构成前缀的规则与构成 IBM MQ 对象描述符结构 MQOD 中 **DynamicQName** 字段内容的规则相同，但最后一个非空白字符必须是星号 (*)。如果该属性的值为空字符串，那么 IBM MQ classes for JMS 将使用值 AMQ.* 创建动态队列时。	用于构成 IBM MQ 动态队列名称的前缀。

表 65: *ConnectionFactory* 对象的属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
tempTopicPrefix	字符串	仅由 IBM MQ 主题字符串的有效字符组成的任何非空字符串	创建临时主题时， JMS 会生成格式为 "TEMP/*TEMPTOPICPREFIX*/*unique_id* " 的主题字符串，或者如果此属性保留缺省值，那么仅生成 "TEMP/*unique_id*"。 指定非空 **TEMPTOPICPREFIX** 允许定义特定模型队列，以便为在此连接下面创建的临时主题的订户创建受管队列。
transportType	字符串	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	与队列管理器的连接使用客户机方式还是绑定方式。 如果指定值 BINDINGS_THEN_CLIENT，那么资源适配器首先尝试以绑定方式进行连接。 如果此连接尝试失败，那么资源适配器将尝试建立客户机方式连接。 z/OS 如果在 WebSphere Application Server for z/OS 系统上运行的激活规范已配置为使用 BINDINGS_THEN_CLIENT 传输方式，并且先前建立的连接已中断，那么由此激活规范所作的任何重新连接尝试都会首先尝试使用 BINDINGS 传输方式。 如果 BINDINGS 传输方式连接尝试不成功，那么激活规范随后会尝试进行 CLIENT 传输方式连接。
username	字符串	• **null** • 用户名	创建与队列管理器的连接时要使用的缺省用户名。
wildcardFormat	int	• CHAR - 仅识别代理程序版本 1 中所使用的字符通配符 • TOPIC - 仅识别代理程序版本 2 中所使用的主题级别通配符	要使用的通配符语法版本。

注意:

1. 此属性可以与 IBM WebSphere MQ classes for JMS 的 V 7.0 配合使用，但不会影响连接到 IBM WebSphere MQ 7.0 队列管理器的应用程序，除非 providerVersion 属性设置为小于 7 的版本号。
2. 有关使用 sslFipsRequired 属性的重要信息，请参阅 第 369 页的『IBM MQ 资源适配器的限制』。
3. 有关如何配置资源适配器以使其可以找到出口的信息，请参阅第 230 页的『配置 IBM MQ classes for JMS 以使用通道出口』。

以下示例显示 ConnectionFactory 对象的典型属性集:

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

受管目标对象的属性

应用程序服务器使用受管目标对象的属性来创建 JMS Queue 对象或 JMS Topic 对象。

第 400 页的表 66 列出 Queue 对象和 Topic 对象的公共属性。

表 *66: Queue* 对象和 *Topic* 对象的公共属性

属性的名称	类型	有效值（缺省值为粗体形式）	描述
CCSID	字符串	• **1208** • Java 虚拟机 (JVM) 支持的编码字符集标识	目标的编码字符集标识。
编码	字符串	• **NATIVE** • 包含三个字符的字符串： – 第一个字符指定二进制整数的表示： - *N* 表示正常编码。 - *R* 表示反向编码。 – 第二个字符指定压缩十进制整数的表示： - *N* 表示正常编码。 - *R* 表示反向编码。 – 第三个字符指定浮点数的表示： - *N* 表示标准 IEEE 编码。 - *R* 表示反向 IEEE 编码。 - *3* 表示 zSeries 编码。 NATIVE 与字符串 NNN 等效。	目标的二进制整数、压缩十进制整数和浮点数的表示。
到期	字符串	• **APP** - 由消息生产者确定消息到期时间。 • UNLIM - 消息从不到期。 • 0 - 消息从不到期。 • 表示消息到期时间的正整数（以毫秒为单位）。	发送到目标的消息的到期时间。
failIfQuiesce	字符串	• **true** • false	如果队列管理器处于停顿状态，那么尝试访问目标是否失败。

表 66: *Queue* 对象和 *Topic* 对象的公共属性 (继续)

属性的名称	类型	有效值（缺省值为粗体形式）	描述
messageBodyStyle	字符串	• **UNSPECIFIED** • JMS • MQ	您可以在 JMS 队列和主题上设置 **messageBodyStyle** 属性: UNSPECIFIED (缺省值) • 在发送时，IBM MQ classes for JMS 生成并包含 MQRFH2 头，具体取决于 WMQ_TARGET_CLIENT 的值。 • 在接收时，IBM MQ classes for JMS 根据 MQRFH2（如果存在）中的值设置 JMS 消息属性。 MQRFH2 不作为 JMS 消息体的一部分存在。 JMS • 在发送时，IBM MQ classes for JMS 自动生成 MQRFH2 头并将该头包含在 IBM MQ 消息中。 • 在接收时，IBM MQ classes for JMS 根据 MQRFH2（如果存在）中的值设置 JMS 消息属性。 MQRFH2 不作为 JMS 消息体的一部分存在。 MQ • 在发送时，IBM MQ classes for JMS 不生成 MQRFH2。 • 在接收时，IBM MQ classes for JMS 提供 MQRFH2 作为 JMS 消息体的一部分。
持久性	字符串	• **APP** - 由消息生产者确定消息持久性。 • QDEF-消息的持久性由 IBM MQ 队列的 **DefPersistence** 属性确定。 • PERS - 消息持久。 • NON - 消息不持久。 • HIGH-根据 第 196 页的『JMS 持久消息』中的说明，消息的持久性由 IBM MQ 队列的 **NonPersistentMessageClass** 属性确定。	发送到目标的消息的持久性。
priority	字符串	• **APP** - 由消息生产者确定消息优先级。 • QDEF-消息的优先级由 IBM MQ 队列的 **DefPriority** 属性确定。 • 范围在 0（最低优先级）到 9（最高优先级）中的整数。	发送到目标的消息的优先级。

表 66: *Queue* 对象和 *Topic* 对象的公共属性 *(继续)*			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
putAsyncAllowed	字符串	• QUEUE - 通过引用队列定义来确定是否允许异步放置。 • TOPIC - 通过引用主题定义来确定是否允许异步放置。 • DESTINATION - 通过引用队列或主题定义来确定是否允许异步放置。 • DISABLED - 不允许异步放置。 • ENABLED - 允许异步放置。	是否允许消息生产者使用异步放置将消息发送到此目标。
readAheadAllowed	int	• **DESTINATION** - 通过引用队列或主题定义来确定是否允许预读。 • DISABLED - 不允许预读。 • ENABLED - 允许预读。 • QUEUE - 通过引用队列定义来确定是否允许预读。 • TOPIC - 通过引用主题定义来确定是否允许预读。	是否允许消息使用者和队列浏览器在接收来自目标的非持久消息之前使用预读将其放入内部缓冲区。
receiveCCSID	int	• **0** - 使用 JVM `Charset.defaultCharset` • 1208 - UTF-8 • 支持的编码字符集标识	用于设置队列管理器消息转换的目标 CCSID 的目标属性。除非 **receiveConversion** 设置为 QMGR，否则忽略该值。
receiveConversion	字符串	• **CLIENT_MSG** • QMGR	用于确定数据转换即将由队列管理器执行的目标属性。
targetClient	字符串	• **JMS** - 消息的目标是 JMS 应用程序。 • MQ - 消息的目标是非 JMS IBM MQ 应用程序。	发送到目的地的消息的目标是否是 JMS 应用程序。 以 JMS 应用程序为目标的消息包含 MQRFH2 头。

第 402 页的表 67 列出特定于 Queue 对象的属性。

表 67: 特定于 *Queue* 对象的属性			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
baseQueueManagerName	字符串	• **""（空字符串）** • 队列管理器名称	拥有底层 IBM MQ 队列的队列管理器的名称。
baseQueueName	字符串	• **""（空字符串）** • 队列名称	底层 IBM MQ 队列的名称。

第 403 页的表 68 列出特定于 Topic 对象的属性。

表 68: 特定于 *Topic* 对象的属性			
属性的名称	**类型**	**有效值（缺省值为粗体形式）**	**描述**
baseTopicName	字符串	• **""（空字符串）** • 主题名称	底层主题的名称。
brokerCCDurSubQueue [1]	字符串	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • 队列名称	连接使用者从中接收持久预订消息的队列的名称。
brokerDurSubQueue [1]	字符串	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • 队列名称	持久主题订户从中接收消息的队列的名称。 请参阅 IBM MQ Explorer 文档中的 **BROKEDURRSUBQ** 属性以获取更多信息。
brokerPubQueue [1]	字符串	• **未设置** • 队列名称	发送发布的消息的队列的名称（流队列）。 此属性的值将覆盖 ConnectionFactory 对象的 **brokerPubQueue** 属性的值。 但是，如果不设置此属性值，则将使用 ConnectionFactory 对象的 **brokerPubQueue** 属性值。
brokerPubQueueManager [1]	字符串	• **""（空字符串）** • 队列管理器名称	队列管理器的名称，该队列管理器拥有其中发送主题上发布的消息的队列。
brokerVersion [1]	字符串	• **未设置** • 1 • 2	正在使用的代理程序的版本。 此属性的值将覆盖 ConnectionFactory 对象的 **brokerVersion** 属性的值。 但是，如果不设置此属性值，则将使用 ConnectionFactory 对象的 **brokerVersion** 属性值。

注:

1. 此属性可以与 IBM WebSphere MQ classes for JMS 的 V 7.0 配合使用，但不会影响连接到 IBM WebSphere MQ 7.0 队列管理器的应用程序，除非 ConnectionFactory 对象的 providerVersion 属性设置为小于 7 的版本号。

以下示例显示 Queue 对象的属性集：

```
expiry:               UNLIM
persistence:          QDEF
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

以下示例显示 Topic 对象的属性集：

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

相关任务

指定运行时在 MQI 客户机上仅使用经过 FIPS 认证的 CipherSpecs

在 WebSphere Application Server 中配置 JMS 资源

相关参考

针对 UNIX, Linux, and Windows 的美国联邦信息处理标准 (FIPS)

V 9.1.0 为激活规范配置 *targetClientMatching* 属性

您可以为激活规范配置 **targetClientMatching** 属性，以便在请求消息不包含 MQRFH2 头时在应答消息中包含 MQRFH2 头。 这意味着，在发送消息时将包含应用程序在应答消息上定义的所有消息属性。

关于此任务

如果消息驱动的 bean (MDB) 应用程序通过 IBM MQ JCA 资源适配器激活规范使用的消息不包含 MQRFH2 头，并随后将应答消息发送到根据请求消息的 JMSReplyTo 字段创建的 JMS 目标，那么这些应答消息必须包含 MQRFH2 头（即使请求消息不包含此头也是如此），否则此应用程序在应答消息上定义的所有消息属性都将丢失。

对于由入局消息的 JMSReplyTo 头字段标识的队列，**targetClientMatching** 属性可定义发送到此队列的应答消息是否仅当该入局消息具有 MQRFH2 头时才具有 MQRFH2 头。 您可以在 WebSphere Application Server traditional 和 WebSphere Liberty 中为激活规范配置该属性。

如果将 **targetClientMatching** 属性的值设置为 `false`，那么对于根据不包含 MQRFH2 的入局请求消息的 JMSReplyTo 头创建的 JMS 目标，在发送到此 JMS 目标的应答消息中可包含 MQRFH2 头。 这是因为 JMS 目标上的 **targetClient** 属性设置为值 `0`，这意味着此消息包含 MQRFH2 头。 出站消息中存在 MQRFH2 头即可允许在发送到 IBM MQ 队列的消息上存储用户定义的消息属性。

如果 **targetClientMatching** 属性设置为 `true`，并且请求消息不包含 MQRFH2 头，那么在应答消息中不包含 MQRFH2 头。

过程

- 在 WebSphere Application Server traditional 中，使用管理控制台来将 **targetClientMatching** 属性定义为 IBM MQ 激活规范上的定制属性：
  a) 在导航窗格中，单击**资源 -> JMS -> 激活规范**。
  b) 选择要查看或更改的激活规范的名称。
  c) 单击**定制属性 -> 新建**，然后输入新定制属性的详细信息。
  将属性名称设置为 `targetClientMatching`，将类型设置为 `java.lang.Boolean`，并将值设置为 `false`。
- 在 WebSphere Liberty 中，在 `server.xml` 中的激活规范定义上指定 **targetClientMatching** 属性。例如：

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

相关概念

第 174 页的『在 JMS 应用程序中创建目标』
JMS 应用程序可以使用会话在运行时动态创建目标，而不是从 Java 命名和目录接口 (JNDI) 名称空间中检索目标作为受管对象。 应用程序可以使用统一资源标识 (URI) 识别 IBM MQ 队列或主题，也可以指定 Queue 或 Topic 对象的一个或多个属性（可选）。

第 390 页的『配置出站通信的资源适配器』
要配置出站通信，请定义 ConnectionFactory 对象和受管目标对象的属性。

V 9.1.1 *IBM MQ* 消息驱动的 *Bean* 在 *WebSphere Liberty* 中暂停

针对激活规范的 **maxSequentialDeliveryFailures** 属性定义在暂停消息驱动的 bean (MDB) 之前资源适配器容许 MDB 实例发生的顺序消息传递失败的最大次数。

开始之前

您需要注意可能导致 MDB 在 WebSphere Liberty 中暂停的事件集。 资源适配器会将以下任何一种情况视为消息传递失败：

- 从 MDB 的 **onMessage** 方法中抛出的未经检查的异常。
- 在将消息传递到 MDB 之前，在处理资源适配器时发生的 `JMSException`。
- 在将消息传递到 MDB 之后，在处理资源适配器时发生的 `JMSException`。
- 用于使用所回滚消息的 XA 事务或本地事务。
- 应用程序服务器中没有可用于将消息传递到 MDB 的线程。

关于此任务

maxSequentialDeliveryFailures 属性的缺省值为 *-1*，这表示永不暂停 MDB。 任何其他负值均被视为 *-1*。 值：

- *0* 表示 MDB 将在发生第一个错误时暂停
- *1* 表示 MDB 将在发生连续两个错误时暂停
- *2* 表示 MDB 将在发生连续三个错误时暂停，以此类推

仅在 WebSphere Liberty 中以及当 Liberty 级别为 18.0.0.4 或更高级别时，才可以为激活规范配置此属性。

注意: 如果在 Liberty 以外的任何其他应用程序服务器环境中将此属性设置为非缺省值，那么将忽略该值，并会向日志中写入一条警告消息。

此外，还可以将 IBM MQ 资源适配器作为通用资源适配器安装到 WebSphere Liberty 中。 这样做将禁用所有 IBM MQ 和 WebSphere Application Server 集成功能，并且会使资源适配器无法检测到它是否正在 Liberty 中运行。 因此，不支持将 **maxSequentialDeliveryFailures** 设置为大于等于 *0* 的值，这样做会导致向日志中写入一条警告消息。

过程

- 在 WebSphere Liberty 中，在 `server.xml` 中的激活规范定义上指定 **maxSequentialDeliveryFailures** 属性。
  例如：

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

相关概念

第 390 页的『配置出站通信的资源适配器』
要配置出站通信，请定义 ConnectionFactory 对象和受管目标对象的属性。

验证资源适配器安装

IBM MQ 资源适配器的安装验证测试 (IVT) 程序作为 EAR 文件提供。 要使用该程序，必须部署它并将一些对象定义为 JCA 资源。

关于此任务

安装验证测试 (IVT) 程序以企业归档 (EAR) 文件（名为 `wmq.jmsra.ivt.ear`）形式提供。 此文件随 IBM MQ classes for JMS 一起安装在 IBM MQ 资源适配器 RAR 文件 `wmq.jmsra.rar` 所在的目录中。 有关安装这些文件的位置的信息，请参阅第 372 页的『安装 IBM MQ 资源适配器』。

必须在应用程序服务器上部署 IVT 程序。 IVT 程序包含 servlet 以及用于测试是否可将消息发送到 IBM MQ 队列及从该队列中接收消息的 MDB。 可使用 IVT 程序来验证是否正确配置了 IBM MQ 资源适配器来支持分

布式事务。 如果要在非 IBM 应用程序服务器中部署 IBM MQ 资源适配器，那么 IBM 服务可能会要求您演示 IVT 工作以验证是否正确配置了应用程序服务器。

在可以运行 IVT 程序前，必须将 ConnectionFactory 对象、Queue 对象以及可能有的 Activation Specification 对象定义为 JCA 资源，并确保应用程序服务器会通过这些定义来创建 JMS 对象，并将其绑定至 JNDI 名称空间。 可选择对象属性以与您自己的 QueueManager 的主机和端口设置相匹配，下面一组属性仅是一个简单示例：

```
ConnectionFactory object:
channel:               SYSTEM.DEF.SVRCONN
hostName:              localhost
port:                  1550
queueManager:          QM1
transportType:         CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:         TEST.QUEUE
```

用于定义 ConnectionFactory、队列和激活规范对象的机制因应用程序服务器而异。 例如，要在 WebSphere Liberty 中设置这些属性，请将以下条目添加到应用程序服务器的 `server.xml` 文件中：

```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

缺省情况下，IVT 程序预期将在名称为 jms/ivt/IVTCF 的 JNDI 名称空间中绑定 ConnectionFactory 对象，并将 Queue 对象与 jms/ivt/IVTQueue 名称绑定。 可使用不同的名称，但如果这样操作，就必须在 IVT 程序的初始页面上输入对象名称，并适当地修改 EAR 文件。

在部署 IVT 程序，并且应用程序服务器创建 JMS 对象并将其绑定至 JNDI 名称空间后，可完成以下步骤来启动 IVT 程序。

过程

1. 通过在 Web 浏览器中输入以下格式的 URL，来启动 IVT 程序：

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   其中 *app_server_host* 是运行应用程序服务器的系统的 IP 地址或主机名，*port* 是应用程序服务器正在侦听的 TCP 端口号。 例如：

   ```
   http://localhost:9080/WMQ_IVT/
   ```

   第 407 页的图 52 显示 IVT 程序的初始页面。

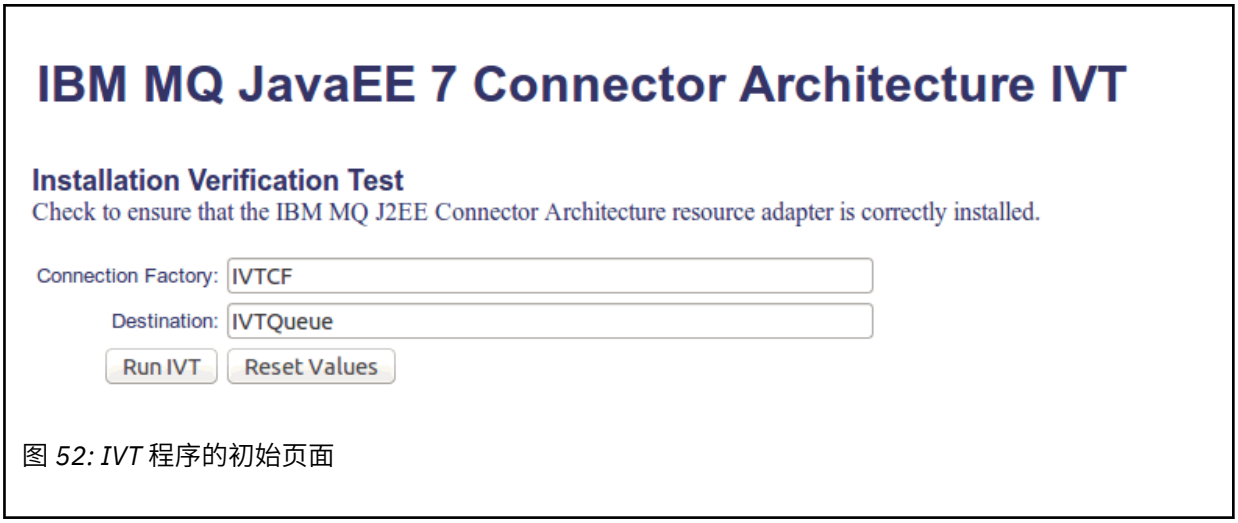


图 *52: IVT* 程序的初始页面

2. 要运行测试，请单击**运行 IVT**。

   第 407 页的图 53 显示在 IVT 成功时显示的页面。

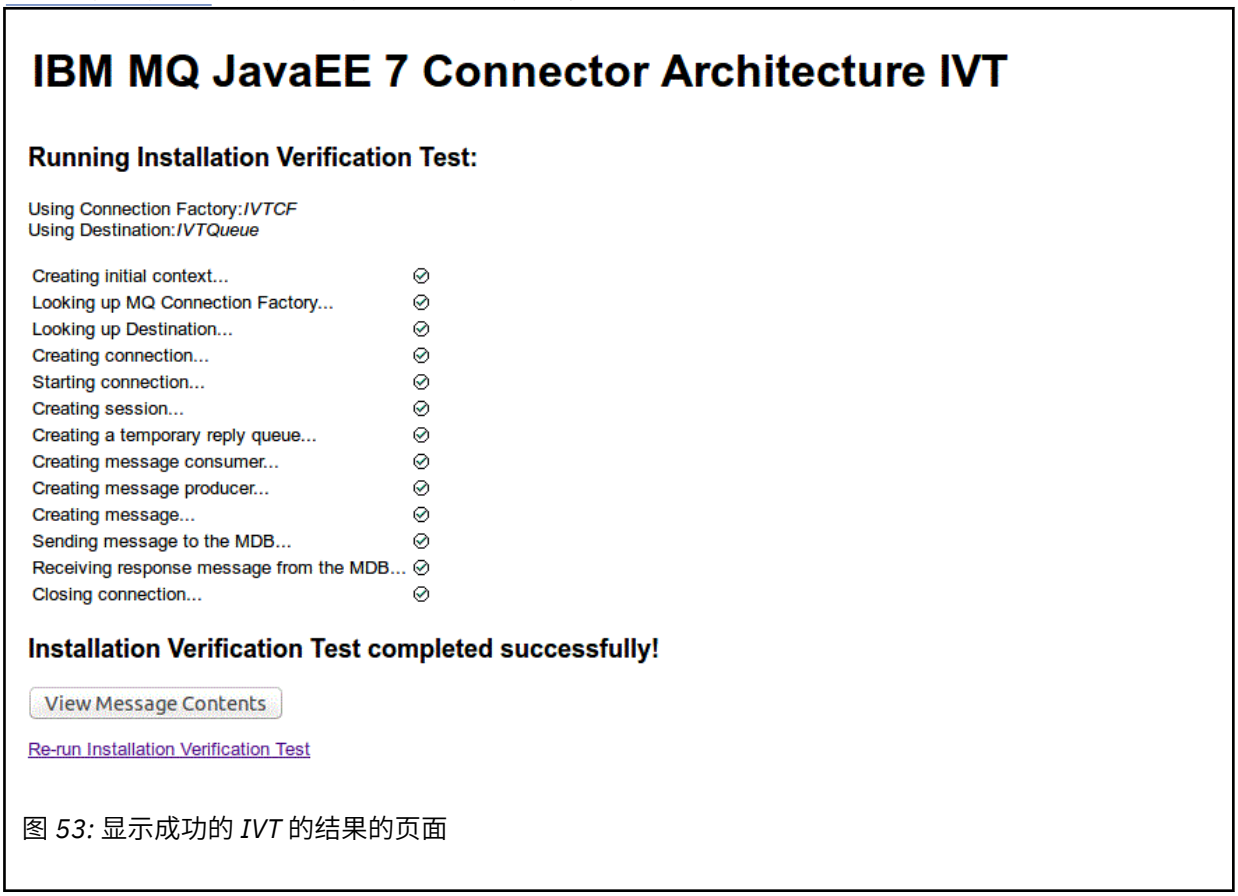


图 *53:* 显示成功的 *IVT* 的结果的页面

如果 IVT 失败，那么将显示类似第 408 页的图 54 中显示的页面的页面。 要获取有关故障原因的更多信息，请单击**查看堆栈跟踪**。

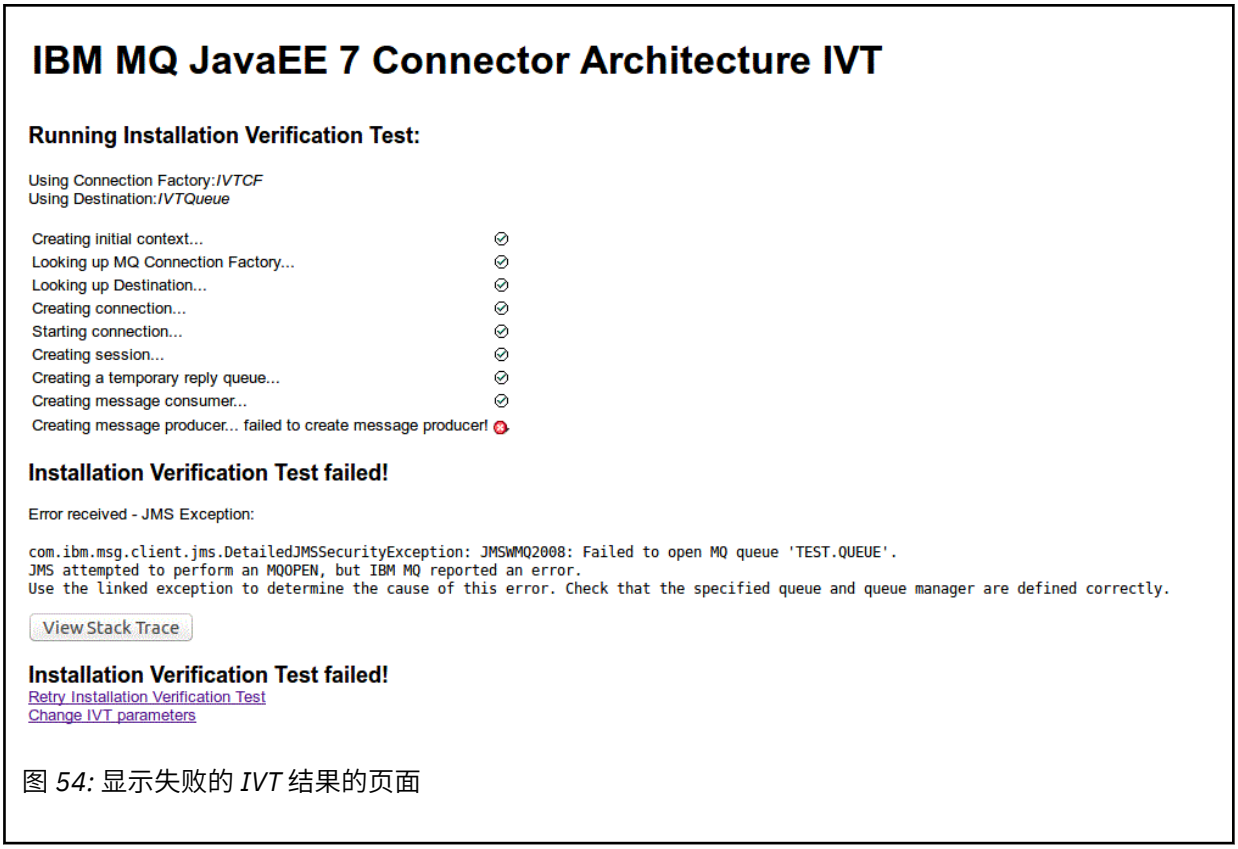


图 *54:* 显示失败的 *IVT* 结果的页面

Windows 在 GlassFish Server 中安装和测试资源适配器

要在 Windows 操作系统上的 GlassFish Server 中安装 IBM MQ 资源适配器，必须首先创建并启动域。 然后，可部署和配置资源适配器，并部署和运行安装验证测试 (IVT) 应用程序。

关于此任务

要点: 这些指示信息适用于 GlassFish Server V4。

此任务假定您正在运行 GlassFish Server 应用程序服务器，并且熟悉它的标准管理任务。 此任务还假定您在本地系统上安装了 IBM MQ 并且假定您熟悉标准管理任务。

注: 为了完成以下任务步骤，您必须具备运行正常的 IBM MQ 安装，并且配置了以下对象:

- 名为 QM 的队列管理器，在端口 1414 上启动，使用通道 SYSTEM.DEF.SVRCONN，并且使用客户机传输进行连接。
- 名为 Q1 的队列。

过程

1. 启动 GlassFish Server **asadmin** shell 程序。
   a) 打开 Windows 命令行并浏览至 *GlassFish*/bin 目录，其中 *GlassFish* 是 GlassFish Server V4 的安装目录。
   b) 在命令行中输入命令 **asadmin**。
   asadmin 命令将在命令行中打开支持您创建新域的 shell 程序。
   GlassFish Server V4 将在您的系统上启动。
2. 创建并启动域。
   a) 使用 **create-domain** 命令通过指定端口和域名来创建新域。 在命令行上输入以下命令:

```
create-domain --adminport port domain_name
```

其中，*port* 是端口号，*domain_name* 是希望域使用的名称。

注: create-domain 命令具有许多与之关联的可选参数。 但是，对于此任务，仅需要 `--adminport` 参数。 有关更多信息，请参阅 GlassFish Server V4 的产品文档。

如果您指定的端口已被使用，将显示以下消息：

`Port for domain_name port is in use`

如果您指定的域名正在使用中，您收到一条消息，告知您指定的名称已在使用中，并提供当前不可用的所有域名的列表。

b) 在提示输入用户名和密码时，请输入用于通过 Web 浏览器登录到应用程序服务器的凭证。

如果该命令成功完成，那么将在命令行上显示一条总结域创建的消息，包括消息 `Command create-domain executed successfully.`

您已成功创建域。

c) 通过在命令行中输入以下命令来启动您的域：

```
start-domain domain_name
```

其中，*domain_name* 是您先前指定的域名。

3. 使用 Web 浏览器访问 GlassFish 应用程序服务器。

a) 在 Web 浏览器的地址栏中，输入以下命令：

```
localhost:port
```

其中，*port* 是您先前在创建域时指定的端口。

将显示 GlassFish Console。

b) 如果 GlassFish Console 已加载，并且提示您输入用户名和密码，请输入您在步骤 2b 中指定的凭证。

4. 将资源适配器上载至 GlassFish Server 4。

a) 在工具栏上的**常见任务**中，选择**应用程序**菜单项以显示**“应用程序”**页面。

b) 单击**部署**按钮以打开**“部署应用程序或模块”**页面。

c) 单击**浏览**按钮，然后浏览至 `wmq.jmsra.rar` 文件所在的位置。 选择该文件，然后单击**确定**。

5. 创建连接池。

a) 在工具栏上的**资源**下，选择**连接器**菜单项。

b) 然后，选择**连接器连接池**菜单项，以打开**连接器连接池**页面。

c) 单击**新建**以打开**“新建连接器连接池（第 1 步，共 2 步）”**页面。

d) 在**“新建连接器连接池（第 1 步，共 2 步）”**页面上，在**池名称**字段中输入 jms/ivt/IVTCF-Connection-Pool 作为池名称。

e) 在**资源适配器**字段中选择 **wmq.jmsra**。

f) 在**连接定义**字段中输入 `javax.jms.ConnectionFactory`。

g) 选择**下一步**，然后选择**完成**。

6. 创建连接器资源。

a) 在工具栏上的**连接器**菜单下，选择**连接器资源**选项以打开**“连接器资源”**页面。

b) 选择**新建**，以打开**新建连接器资源**页面。

c) 在 **JNDI 名称**字段中，输入 `IVTCF`。

d) 在**池名称**字段中，输入 `jms/ivt/IVTCF-Connection-Pool`。

e) 所有其他字段留空。

f) 对于以下每个“属性/值”对，单击**添加属性**，并输入属性名称和值，如以下示例中所示：

- 名称：host；值：localhost
- 名称：port；值：1414
- 名称：channel；值：SYSTEM.DEF.SVRCONN
- 名称：queueManager；值：QM
- 名称：transportType；值：CLIENT

注: 确保对自己的配置设置（可能与此示例中所示的配置设置不同）使用正确的值。

g) 在工具栏上的**连接器**下，选择**管理对象资源**菜单项以打开**管理对象资源**页面。

h) 在**管理对象资源**页面中，单击**新建**以打开**新建管理对象资源**页面。

i) 在 **JNDI 名称**字段中，输入 IVTQueue。

j) 在**资源适配器**字段中，输入 wmq.jmsra。

k) 在**资源类型**字段中，输入 javax.jms.Queue。

l) 将**类名**字段保留现状。

m) 对于以下每个“属性/值”对，单击**添加属性**，并输入属性名称和值，如以下示例中所示：

- 名称：name；值：IVTQueue
- 名称：baseQueueManagerName；值：QM
- 名称：baseQueueName；值：Q1

注: 确保对自己的配置设置（可能与此示例中所示的配置设置不同）使用正确的值。

n) 单击**确定**。

o) 选中**已启用**复选框，然后单击**启用**。

7. 将 EAR 文件 wmq.jmsra.ivt.ear 部署到 GlassFish Server。
   a) 单击工具栏上的**应用程序**选项以显示“**应用程序**”页面。
   b) 单击**部署**以添加 IVT 应用程序。
   c) 在**位置**字段中，浏览并选择 wmq.jmsra.ivt.ear。
   d) 在**虚拟服务器**字段中，选择**服务器**，然后单击**确定**。
8. 启动 IVT 程序。
   a) 单击工具栏上的**应用程序**选项以显示“**应用程序**”页面。
   b) 单击“已部署的应用程序”表中的 wmq.jmsra.ivt。
   c) 单击“模块和组件”表中的**启动**按钮。
   d) 选择 http: 链接。
   e) 单击**运行 IVT**。

   您已启动 IVT 程序，如果成功，会显示以下输出：

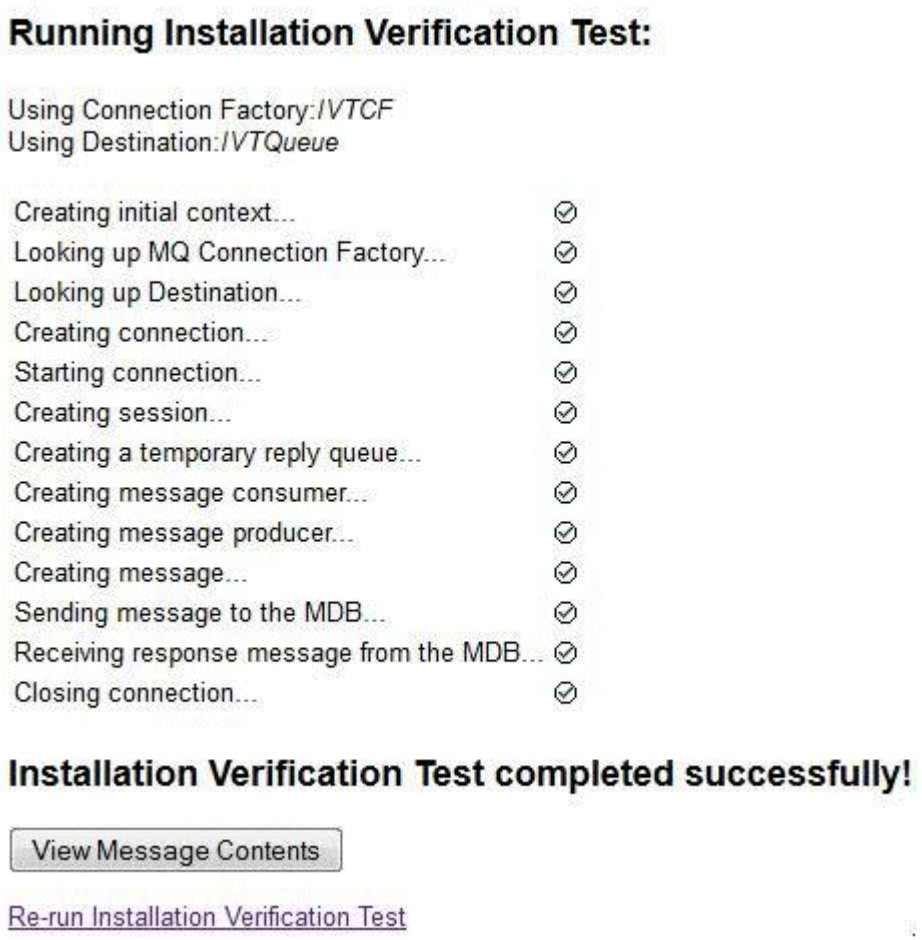


图 *55:* 成功的 *IVT* 输出

在 Wildfly 中安装和测试资源适配器

如果要在 Wildfly V10 中安装 IBM MQ 资源适配器，那么必须先对配置文件进行一些更改，以便为 IBM MQ 资源适配器添加子系统定义。 然后，可通过安装并运行“安装验证测试”(IVT) 应用程序来部署资源适配器并对其进行测试。

关于此任务

要点: 这些指示信息适用于 Wildfly V10。

此任务假定您有一个正在运行的 WildFly 应用程序服务器，并且熟悉其标准管理任务。 此任务还假定您有一个 IBM MQ 安装，并且熟悉标准管理任务。

过程

1. 创建一个名为 ExampleQM 的 IBM MQ 队列管理器，并如第 972 页的『配置队列管理器以接受多平台上的客户机连接』中所述对其进行设置。

   设置队列管理器时，请注意以下几点：

   - 必须在端口 1414 上启动侦听器。
   - 要使用的通道名为 SYSTEM.DEF.SVRCONN。
   - IVT 应用程序使用的队列名为 TEST.QUEUE。

   另外，还需要向模型队列 SYSTEM.DEFAULT.MODEL.QUEUE 授予 DSP 和 PUT 权限，以便此应用程序能够创建临时回复队列。

2. 编辑配置文件 *WildFly_Home*/standalone/configuration/standalone-full.xml 并添加以下子系统:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

3. 通过将 wmq.jmsra.rar 文件复制到目录 *WildFly_Home*/standalone/deployments 以将资源适配器部署到服务器。
4. 通过将 wmq.jmsra.ivt.ear 文件复制到目录 *WildFly_Home*/standalone/deployments 以部署 IVT 应用程序。
5. 启动应用程序服务器，方法是启动命令提示符，浏览至目录 *WildFly_Home*/bin 并运行以下命令:

```
standalone.bat -c standalone-full.xml
```

6. 运行 IVT 应用程序。

   有关更多信息，请参阅第 405 页的『验证资源适配器安装』。 对于 Wildfly，缺省 URL 为 http://localhost:8080/WMQ_IVT/。

将 IBM MQ 与 WebSphere Application Server 结合使用

通过 WebSphere Application Server 中的 IBM MQ 消息传递提供程序， Java Message Service (JMS) 消息传递应用程序可以将 IBM MQ 系统用作 JMS 消息传递资源的外部提供程序。

关于此任务

在 WebSphere Application Server 下运行的 Java 中编写的应用程序可以使用 Java 消息传递服务 (JMS) 规范来执行消息传递。 此环境中的消息传递可由 IBM MQ 队列管理器提供。

使用 IBM MQ 队列管理器的好处在于连接 JMS 应用程序可以充分享受 IBM MQ 网络的功能，这使得应用程序可以与众多平台上运行的队列管理器交换消息。

应用程序可以为队列连接工厂对象使用客户机传输或绑定传输。 对于绑定传输，队列管理器必须存在于需要连接的应用程序本地。

缺省情况下，保留在 IBM MQ 队列上的 JMS 消息使用 MQRFH2 头来保存某些 JMS 消息头信息。 很多旧 IBM MQ 应用程序不能处理带有这些头的消息，它们需要使用自己的特性头，例如 MQCIH 用于 CICS 桥接应用程序，或者 MQWIH 用于 IBM MQ 工作流程应用程序。 有关这些特殊注意事项的更多信息，请参阅将 JMS 消息映射到 IBM MQ 消息。

相关任务

在 WebSphere Application Server 中配置 JMS 资源

配置应用程序服务器以使用最新的资源适配器维护级别

将 WebSphere Application Server 与 IBM MQ 结合使用

IBM MQ 和 IBM MQ for z/OS 可以与 WebSphere Application Server 随附的缺省消息传递提供程序结合使用，也可以用作其替代选项。

IBM MQ 消息传递提供程序将作为 WebSphere Application Server 的一部分进行安装。 其中包括一个 IBM MQ 资源适配器版本以及 IBM MQ Extended Transactional Client 功能，后者允许队列管理器参与由应用程序服务器管理的 XA 事务。 通过使用资源适配器，可将消息驱动 bean 配置为使用激活规范或侦听器端口。

为支持应用程序服务器，IBM MQ 资源适配器安装验证测试程序必须部署到应用程序服务器中并成功运行。 在成功运行 IBM MQ 资源适配器安装验证测试程序之后，IBM MQ 资源适配器可以连接到任何受支持的 IBM MQ 队列管理器。

从 WebSphere Application Server 到 IBM MQ 的 JMS 连接

在考虑可与 WebSphere Application Server 配合使用的 IBM MQ 级别之前，了解在应用程序服务器中运行的 Java Message Service (JMS) 应用程序如何连接到 IBM MQ 队列管理器很重要。

需要访问 IBM MQ 队列管理器资源的 JMS 应用程序可以使用下列其中一种传输类型来执行此操作:

BINDINGS

当应用程序服务器和队列管理器安装在同一台机器和操作系统映像上时，可以使用此传输。 当使用 BINDINGS 方式时，这两个产品之间的所有通信都是使用进程间通信 (IPC) 来完成的。

IBM MQ 消息传递提供程序不包含以 BINDINGS 方式连接到 IBM MQ 队列管理器所需的本机库。 要使用 BINDINGS 方式连接，必须将 IBM MQ 安装到应用程序服务器所在的机器上，并且必须将资源适配器的本机库路径配置为指向这些库所在的 IBM MQ 目录。 有关更多信息，请参阅 WebSphere Application Server 产品文档:

- 对于 WebSphere Application Server traditional，请参阅 Configuring the IBM MQ messaging provider with native libraries。
- 对于 WebSphere Liberty，请参阅 Deploying JMS applications to Liberty to use the IBM MQ messaging provider。

z/OS 在 z/OS 上，如果要以绑定方式将 WebSphere Application Server 连接工厂连接到 IBM MQ 队列管理器，那么必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM MQ 库。 有关更多信息，请参阅 WebSphere Application Server 产品文档中的 IBM MQ 库和 WebSphere Application Server for z/OS STEPLIB 。

CLIENT
CLIENT 传输使用 TCP/IP 在 WebSphere Application Server 和 IBM MQ 之间进行通信。 当应用程序服务器和队列管理器位于不同机器上时，或者当这两个产品安装在同一台机器和操作系统映像上时，均可使用 CLIENT 方式。

JMS 应用程序也可以指定 BINDINGS_THEN_CLIENT 传输类型。 当使用此传输类型时，应用程序最初会尝试使用 BINDINGS 方式连接到队列管理器，如果无法进行连接，那么会再尝试 CLIENT 传输。

如何查找安装在 WebSphere Application Server 中的 IBM MQ 资源适配器版本

有关在 WebSphere Application Server 中安装了哪个版本的 IBM MQ 资源适配器的信息，请参阅技术说明 WebSphere Application Server 随附了哪个版本的 WebSphere MQ 资源适配器 (RA)?。

您可以使用以下 Jython 和 JACL 命令来确定 WebSphere Application Server 当前使用的资源适配器级别：

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

注: 在输入此命令之后需要单击两次**回车键**才能运行此命令。

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

更新资源适配器

随应用程序服务器一起安装的 IBM MQ 资源适配器的更新包含在 WebSphere Application Server 修订包中。 使用 **更新资源适配器 ...** 更新 IBM MQ 资源适配器 建议不要使用 WebSphere Application Server 管理控制台中的工具，因为这样做将意味着 WebSphere Application Server 修订包中提供的更新将无效。

MQ_INSTALL_ROOT 变量

在版本 7.0 之前，可以将 WebSphere Application Server 配置为使用位于外部 IBM WebSphere MQ 安装中的 IBM WebSphere MQ classes for JMS ，通过设置 WebSphere 变量 MQ_INSTALL_ROOT 来连接到队列管理器。

从 WebSphere Application Server 7.0 开始，MQ_INSTALL_ROOT 仅用于查找本机库，可以被资源适配器上配置的任何本机库路径覆盖。

从 WebSphere Application Server 连接到 IBM MQ

注意:

1. 任何受支持的 WebSphere Application Server 版本都可以使用与其捆绑在一起的 IBM MQ 资源适配器来连接到任何受支持的 IBM MQ 版本。
2. 如果使用绑定方式，那么 WebSphere Application Server 中的某些库需要与所连接到的队列管理器的版本匹配：
   - WebSphere Application Server 必须配置为装入 IBM MQ 9.1 随附的本机库。 有关更多信息，请参阅 第 75 页的『配置 Java 本机接口 (JNI) 库』。
   - z/OS 在 z/OS 上，必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM MQ 库。

有关您需要的 IBM MQ 库的详细信息，请参阅 IBM MQ 库和 WebSphere Application Server for z/OS STEPLIB。

如果您在 LINKLIST (LINKLST) 中具有某个版本的 IBM MQ 的库，那么可以通过使用 STEPLIB 覆盖库来连接到其他版本的 IBM MQ。

3. IBM MQ 资源适配器版本独立于队列管理器安装提供的本机（共享）库版本。

   例如，具有 IBM WebSphere MQ 7.1 资源适配器的 WebSphere Application Server 8.5 仍可以使用 IBM MQ 9.0 本机库来管理与 IBM MQ 9.0 队列管理器的绑定连接。

有关更多信息，请参阅第 367 页的『IBM MQ 资源适配器支持声明』。

下表显示了可用于从 WebSphere Application Server 的所有版本连接到 IBM MQ 的传输类型。

IBM MQ 或 IBM WebSphere MQ 的版本	**BINDINGS 传输**	**CLIENT 传输**
IBM MQ 9.1	支持。 • IBM MQ 9.1 必须与应用程序服务器安装在同一台机器上。 • WebSphere Application Server 必须配置为装入 IBM MQ 9.1 随附的本机库。 • z/OS 在 z/OS 上，如果要以绑定方式将 WebSphere Application Server 连接工厂连接到 IBM MQ 队列管理器，那么必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM MQ 库。	受支持
IBM MQ 9.0	支持。 • IBM MQ 9.0 必须与应用程序服务器安装在同一台机器上。 • WebSphere Application Server 必须配置为装入 IBM MQ 9.0 随附的本机库。 • z/OS 在 z/OS 上，如果要以绑定方式将 WebSphere Application Server 连接工厂连接到 IBM MQ 队列管理器，那么必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM MQ 库。	受支持
IBM MQ 8.0	支持。 • IBM MQ 8.0 必须与应用程序服务器安装在同一台机器上。 • WebSphere Application Server 必须配置为装入 IBM MQ 8.0 随附的本机库。 • z/OS 在 z/OS 上，如果要以绑定方式将 WebSphere	受支持

IBM MQ 或 IBM WebSphere MQ 的版本	BINDINGS 传输	CLIENT 传输
	Application Server 连接工厂连接到 IBM MQ 队列管理器，那么必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM MQ 库。	
IBM WebSphere MQ 7.5	支持。 • IBM WebSphere MQ 7.5 必须与应用程序服务器安装在同一台机器上。 • WebSphere Application Server 必须配置为装入 IBM WebSphere MQ 7.5 随附的本机库。 • z/OS 在 z/OS 上，如果要将 WebSphere Application Server 连接工厂以绑定方式连接到 IBM WebSphere MQ 队列管理器，那么必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM WebSphere MQ 库。	受支持
IBM WebSphere MQ 7.1	支持。 • IBM WebSphere MQ 7.1 必须与应用程序服务器安装在同一台机器上。 • WebSphere Application Server 必须配置为装入 IBM WebSphere MQ 7.1 随附的本机库。 • z/OS 在 z/OS 上，如果要将 WebSphere Application Server 连接工厂以绑定方式连接到 IBM WebSphere MQ 队列管理器，那么必须在 WebSphere Application Server STEPLIB 并置中指定正确的 IBM WebSphere MQ 库。	受支持

下表显示了支持 IBM MQ 资源适配器在其中运行的 WebSphere Application Server 版本。

IBM MQ 资源适配器的版本	此版本的资源适配器可以在哪个版本的 WebSphere Application Server 中运行?
IBM MQ 9.1	可在以下版本中运行资源适配器: • 任何符合 Java EE 7 的 WebSphere Liberty 版本。 • WebSphere Application Server traditional 9.0.

IBM MQ 资源适配器的版本	此版本的资源适配器可以在哪个版本的 WebSphere Application Server 中运行?
IBM MQ 9.0	可在以下版本中运行资源适配器: • 任何符合 Java EE 7 的 WebSphere Liberty 版本。 • WebSphere Application Server traditional 9.0
IBM MQ 8.0	可以在 WebSphere Liberty 的任何 Java EE 7 兼容版本中运行资源适配器。 不支持在 WebSphere Application Server traditional 中运行 IBM MQ 8.0 资源适配器。 应该使用 WebSphere Application Server traditional 中安装的资源适配器来连接到 IBM MQ 8.0 队列管理器。
IBM WebSphere MQ 7.5	可以在 J2EE 1.4 或更高版本兼容的应用程序服务器中使用资源适配器。 应在这些环境中使用 WebSphere Application Server 8.0 和 7.0 中包含的 IBM WebSphere MQ 资源适配器版本。 IBM WebSphere MQ 7.5.0 Fix Pack 2 和 APAR IC92914 新增了将资源适配器部署到 WebSphere Liberty 的支持。
IBM WebSphere MQ 7.1	可以在 J2EE 1.4 或更高版本兼容的应用程序服务器中使用资源适配器。 应在这些环境中使用 WebSphere Application Server 8.0 和 7.0 中包含的 IBM WebSphere MQ 资源适配器版本。

相关概念

第 367 页的『IBM MQ 资源适配器支持声明』
随 IBM MQ 8.0 或更高版本提供的资源适配器已实现了 JMS 2.0 规范。 它仅可以部署到兼容 Java Platform, Enterprise Edition 7 (Java EE 7) 并支持 JMS 2.0 的应用程序服务器。

相关信息

IBM MQ 系统需求

确定已创建的从 WebSphere Application Server 到 IBM MQ 的 TCP/IP 连接数

通过使用共享对话功能，多个对话可以共享 MQI 通道实例，这也称为 TCP/IP 连接。

关于此任务

在使用 IBM MQ 消息传递提供程序正常方式的 WebSphere Application Server 7 和 8 中运行的应用程序将自动使用此功能。 这意味着在同一个应用程序服务器实例内运行并连接到同一个 IBM MQ 队列管理器的多个应用程序能够共享同一个通道实例。

可通过单个通道实例共享的对话数量由 IBM MQ 通道属性 **SHARECNV** 确定。 服务器连接通道的该属性的缺省值为 10。

通过查看 WebSphere Application Server 7 和 8 创建的对话数，可以确定创建的通道实例数。

有关 IBM MQ 消息传递提供者方式的更多信息，请参阅 PROVIDERVERSION 正常方式。

相关概念

使用共享对话
在允许共享对话的环境中，对话可以共享 MQI 通道实例。

第 257 页的『在 IBM MQ classes for JMS 中共享 TCP/IP 连接』
可以让 MQI 通道的多个实例共享一个 TCP/IP 连接。

JMS 连接工厂

对于在 WebSphere Application Server 内运行并使用 IBM MQ 消息传递提供者连接工厂来创建连接和会话的应用程序，它们将让用于从连接工厂创建的每个 JMS 连接以及用于从 JMS 连接创建的每个 JMS 会话的对话都保持活动状态。

从连接工厂创建的每个 JMS 连接都对应有一个对话

每个 JMS 连接工厂都有关联的连接池，该连接池划分为空闲池和活动池两部分。 这两个池最初都为空。

当应用程序从连接工厂创建 JMS 连接时，WebSphere Application Server 将执行检查以了解空闲池中是否有 JMS 连接。 如果有，那么会将该连接移到活动池，然后将其分配给应用程序。 否则，将创建新的 JMS 连接，将其放入活动池，然后将其返回给应用程序。 可以从连接工厂创建的最大连接数由连接工厂连接池属性 **Maximum connections** 指定。 该属性的缺省值为 10。

应用程序使用完 JMS 连接并将其关闭后，该连接将从活动池移到可供复用的空闲池中。 连接池属性 **Unused timeout** 定义 JMS 连接在断开连接之前可以在空闲池中停留的时间。 该属性的缺省值是 1800 秒（30 分钟）。

首次创建 JMS 连接时，会启动 WebSphere Application Server 和 IBM MQ 之间的对话。 当超出空闲池的 **Unused timeout** 属性值时，该对话将保持活动状态，直到连接关闭为止。

从 JMS 连接创建的每个 JMS 会话都对应有一个对话

从 IBM MQ 消息传递提供程序连接工厂创建的每个 JMS 连接都具有关联的 JMS 会话池。 这些会话池的工作方式与连接池相同。 可从单个 JMS 连接创建的最大 JMS 会话数由连接工厂会话池属性 **Maximum connections** 确定。 该属性的缺省值为 10。

首次创建 JMS 会话时，将启动对话，此对话将保持活动状态，直到 JMS 会话关闭为止，因为它在空闲池中的保留时间超过会话池的 **Unused timeout** 属性值。

计算 SHARECNV 属性的值

可以使用以下公式计算从单个连接工厂到 IBM MQ 的最大对话数：

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

可以使用以下计算公式来确定为支持此数量的对话而创建的通道实例数：

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

可以对此计算的余数部分进行上舍入处理。

对于使用连接池 **Maximum connections** 和会话池 **Maximum connections** 属性的缺省值的简单连接工厂，此连接工厂的 WebSphere Application Server 和 IBM MQ 之间可以存在的最大对话数为：

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

例如：

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

如果此连接工厂正在使用将 **SHARECNV** 属性设置为 10 的通道连接到 IBM MQ，那么将为此连接工厂创建的最大通道实例数为:

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

例如:

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

激活规范

对于配置为使用激活规范的消息驱动的 bean 应用程序，它们会让用于激活规范的对话保持活动状态以监视 JMS 目标，并让用于每个服务器会话（用于运行消息驱动的 bean 实例）的对话保持活动状态以处理消息。

对于配置为使用激活规范的消息驱动的 bean 应用程序，以下对话将保持活动状态:

- 供激活规范用来监视 JMS 目标中是否有合适消息的对话。 此对话在激活规范启动后立即启动并一直保持活动状态，直到激活规范停止为止。
- 供每个服务器会话用来运行消息驱动 bean 实例以处理消息的对话。

激活规范高级属性 **Maximum server sessions** 指定在给定激活规范的任意时刻可以处于活动状态的服务器会话的最大数目。 此属性的缺省值为 10。 将根据需要创建服务器会话，如果这些会话在激活规范高级属性 **Server session pool timeout** 指定的时间段内处于空闲状态，那么将关闭这些会话。 该属性的缺省值为 300000 毫秒（5 分钟）。

对话在创建服务器会话时启动，并在激活规范停止或服务器会话超时时停止。

这表示可以使用以下公式来计算从单个激活规范到 IBM MQ 的最大对话数:

```
Maximum number of conversations = Maximum server sessions + 1
```

可以使用以下计算公式来确定为支持此数量的对话而创建的通道实例数:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

可以对此计算的余数部分进行上舍入处理。

对于使用 **Maximum server sessions** 属性的缺省值的简单激活规范，此激活规范的 WebSphere Application Server 和 IBM MQ 之间可以存在的最大对话数计算如下:

```
Maximum number of conversations = Maximum server sessions + 1
```

例如:

```
    = 10 + 1
    = 11
```

如果此激活规范正在使用将 **SHARECNV** 属性设置为 10 的通道连接到 IBM MQ，那么会将创建的通道实例数计算为:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

例如:

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

在应用程序服务器工具 (ASF) 方式中运行的侦听器端口

消息驱动的 bean 应用程序使用的、以 ASF 方式运行的侦听器端口将会为每个服务器会话创建对话。 一个对话用于监视目标中是否有合适的消息，另一个对话用于运行消息驱动的 bean 实例以处理消息。 可以根据最大会话数来计算每个侦听器端口的对话数。

缺省情况下，侦听器端口将作为 1.1 规范的一部分以 ASF 方式运行，该规范定义了一种机制，供应用程序服务器用来检测消息和将消息传递到消息驱动的 bean 进行处理。 设置为以此缺省运行方式使用侦听器端口的消息驱动的 bean 应用程序将创建以下对话：

供侦听器端口用来监视目标中是否有合适消息的对话
侦听器端口已配置为使用 JMS 连接工厂。 在启动侦听器端口时，将从连接工厂空闲池请求 JMS 连接。 当侦听器端口停止时，该连接将返回到空闲池中。 有关连接池的使用方式以及这对 IBM MQ 对话数量的影响的更多信息，请参阅 第 418 页的『JMS 连接工厂』。

供每个服务器会话用来运行消息驱动 bean 实例以处理消息的对话
侦听器端口属性 **Maximum sessions** 指定在给定侦听器端口的任意时刻可以处于活动状态的服务器会话的最大数目。 此属性的缺省值为 10。 服务器会话在需要时创建，并使用从侦听器端口所用 JMS 连接的关联会话池中获取的 JMS 会话。

如果服务器会话空闲时间达到消息侦听器服务定制属性 **SERVER.SESSION.POOL.UNUSED.TIMEOUT** 指定的时间段，那么将关闭该会话，并将使用的 JMS 会话返回到会话空闲池中。 JMS 会话将保留在会话池空闲池中，直到需要该会话池为止，或者由于它在空闲池中的空闲时间超过了会话池的 **Unused timeout** 属性的值而关闭该会话。

有关会话池的使用方式以及 WebSphere Application Server 与 IBM MQ 对话的管理方式的更多信息，请参阅 第 418 页的『JMS 连接工厂』。

有关消息侦听器服务定制属性 **SERVER.SESSION.POOL.UNUSED.TIMEOUT**，请参阅 WebSphere Application Server 产品文档中的 Monitoring server session pools for 侦听器端口 。

计算从单个侦听器端口到 IBM MQ 的最大对话数

可以使用以下公式计算从单个侦听器端口到 IBM MQ 的最大对话数：

```
Maximum number of conversations = Maximum sessions + 1
```

可以使用以下计算公式来确定为支持此数量的对话而创建的通道实例数：

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

可以对此计算的余数部分进行上舍入处理。

对于使用 **Maximum sessions** 属性的缺省值的简单侦听器端口，此侦听器端口的 WebSphere Application Server 和 IBM MQ 之间可以存在的最大对话数计算为：

```
Maximum number of conversations = Maximum sessions + 1
```

例如：

```
    = 10 + 1
    = 11
```

如果此侦听器端口正在使用 **SHARECNV** 属性设置为 10 的通道连接到 IBM MQ ，那么将创建的通道实例数计算为：

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

例如：

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

在非应用程序服务器工具（非 *ASF*）方式中运行的侦听器端口

以非 ASF 方式运行的侦听器端口可以配置为使用服务器会话来监视队列目标和主题目标。 服务器会话可以有多个对话，最大对话数视具体情况而定。

侦听器端口可以配置为以非 ASF 方式运行，这会更改侦听器端口监视 JMS 目标的方式。 以非 ASF 运行方式使用侦听器端口的消息驱动的 bean 应用程序将会为每个服务器会话创建对话以运行消息驱动 bean 实例来处理消息。 侦听器端口属性**最大会话数**指定了给定侦听器端口在任一时间可保持活动的最大服务器会话数。 该属性的缺省值为 10。

当以非 ASF 方式运行时，监视队列目标的侦听器端口将自动创建侦听器端口属性**最大会话数**所指定数量的服务器会话。 所有这些服务器会话都使用从侦听器端口所用 JMS 连接的关联会话池获取的 JMS 会话，并持续监视 JMS 目标中是否有合适的消息。

如果侦听器端口配置为监视主题目标，那么将忽略**最大会话数**值，并且只使用一个服务器会话。

以非 ASF 方式运行的侦听器端口所使用的服务器会话将一直保持活动状态，直到侦听器端口停止为止，此时会将先前使用的 JMS 会话返回到侦听器端口所用的 JMS 连接的会话空闲池中。

有关会话池的使用方式以及 WebSphere Application Server 与 IBM MQ 对话的管理方式的更多信息，请参阅第 418 页的『JMS 连接工厂』。

有关使用 WebSphere Application Server 的 ASF 和非 ASF 操作方式以及如何配置侦听器端口以使用非 ASF 方式的更多信息，请参阅 以 ASF 方式和非 ASF 方式进行消息处理。

计算监视队列目标时的最大对话数

可以使用以下公式来计算从单个侦听器端口（以非 ASF 方式运行，并用于监视队列目标）到 IBM MQ 的最大对话数：

```
Maximum number of conversations = Maximum sessions
```

可以使用以下计算公式来确定为支持此数量的对话而创建的通道实例数：

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

可以对此计算的余数部分进行上舍入处理。

对于以非 ASF 方式运行、使用**最大会话数**属性缺省值并监视目标队列的简单侦听器端口，WebSphere Application Server 和 IBM MQ 之间针对此侦听器端口可以存在的最大对话数为：

```
Maximum number of conversations = Maximum sessions
```

例如：

```
    = 10
```

如果此侦听器端口使用将 **SHARECNV** 属性设置为 10 的通道连接到 IBM MQ ，那么创建的通道实例数计算为：

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

例如：

```
    = 10 / 10
    = 1
```

计算监视主题目标时的最大对话数

对于以非 ASF 方式运行并配置为监视主题目标的侦听器端口，从该侦听器端口到 IBM MQ 的对话数为：

```
Maximum number of conversations = 1
```

可以使用以下计算公式来确定为支持此数量的对话而创建的通道实例数：

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

可以对此计算的余数部分进行上舍入处理。

对于以非 ASF 方式运行、使用**最大会话数**属性缺省值并监视主题目标的简单侦听器端口，WebSphere Application Server 和 IBM MQ 之间针对此侦听器端口可以存在的最大对话数为：

```
Maximum number of conversations = Maximum sessions
```

例如：

```
= 10
```

如果此侦听器端口使用将 **SHARECNV** 属性设置为 10 的通道连接到 IBM MQ，那么创建的通道实例数计算为：

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

例如：

```
= 10 / 10
= 1
```

配置认证别名以保护 WebSphere Application Server 与 IBM MQ 的连接

认证别名将映射到用户名和密码组合，可使用此组合来保护 WebSphere Application Server 与 IBM MQ 的连接。 您可以对连接工厂配置认证别名。

将认证别名与企业应用程序结合使用

在 WebSphere Application Server 中运行的企业应用程序尝试创建与 IBM MQ 的 JMS 连接时，应用程序会从应用程序服务器的 Java Naming Directory Interface (JNDI) 存储库中查找 IBM MQ 消息传递提供程序连接工厂定义。

当 IBM MQ 消息传递提供程序连接工厂定义位于应用程序服务器的 JNDI 存储库中时，将调用下列其中一种方法：

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

如果在定义了 J2C 认证别名的情况下配置了连接工厂，那么当使用该连接工厂创建连接时，该认证别名中的用户名和密码将向下流至 IBM MQ。

连接工厂和认证别名

IBM MQ 消息传递提供者连接工厂包含有关如何连接到 IBM MQ 队列管理器的信息。 在 WebSphere Application Server 中运行的企业应用程序可以使用连接工厂来创建与 IBM MQ 的 JMS 连接。

WebSphere Application Server 将连接工厂定义存储在可使用 JNDI 访问的存储库中。 在创建连接工厂时，系统会为该连接工厂分配 JNDI 名称，以便在定义该连接工厂的应用程序服务器范围（“单元”、“节点”或“服务器”范围）内唯一标识该连接工厂。

例如，在 WebSphere Application Server 单元作用域定义的 IBM MQ 消息传递提供程序连接工厂包含有关如何使用 BINDINGS 传输连接到队列管理器 (myQM) 的信息。 已为此连接工厂指定了由 jms/myCF 用来唯一标识它的 JNDI 名称。

也可以将连接工厂配置为使用认证别名。 认证别名将映射到用户名和密码组合。 根据使用连接工厂的方式，在创建 JMS 连接时，认证别名中的用户名和密码可能会向下流至 IBM MQ，也可能不会向下流至。

要点: 在 IBM MQ 8.0 之前，缺省的 IBM MQ 对象权限管理器 (OAM) 执行了一次授权检查，目的只是为了确保在建立连接时向下传递至 IBM MQ 的用户名有权访问队列管理器。

不会执行任何检查来验证所指定的密码。 要执行认证检查并验证用户标识和密码是否匹配，您需要编写 IBM MQ 通道安全出口。 可在通道安全出口程序中找到有关如何执行此操作的详细信息。

从 IBM MQ 8.0 开始，除了用户名之外，队列管理器还会检查密码。

使用连接工厂

以下主题包含有关通过直接查找和间接查找来使用连接工厂的信息：

- 第 425 页的『通过直接查找来使用连接工厂』
- 第 426 页的『通过间接查找来使用连接工厂』

使用 CLIENT 传输

配置为使用 CLIENT 传输的连接工厂必须指定其要使用哪个 IBM MQ 服务器连接通道 (SVRCONN) 来连接到队列管理器。

对于连接工厂配置为使用的通道，如果其 IBM MQ 通道代理程序用户标识 (MCAUSER) 属性留空，那么可以通过直接查找或间接查找来使用该连接工厂。

如果 MCAUSER 属性设置为用户标识，那么在使用连接工厂创建与 IBM MQ 的连接时，此用户标识将向下传递至 IBM MQ，这与企业应用程序是使用直接查找还是间接查找无关。

总结表

以下各表汇总了在使用 BINDINGS 传输和 CLIENT 传输时分别向下流至 IBM MQ 的用户标识：

表 *69: BINDINGS* 方式

配置	**应用程序调用 ConnectionFactory.createConnection()**	**应用程序调用 ConnectionFactory.createConnection(String username, String password)**
应用程序部署描述符不包含连接工厂的资源引用	应用程序服务器进程的用户标识向下流至 IBM MQ。	传递到 ConnectionFactory.createConnection(String username, String password) 方法的用户标识和密码向下流至 IBM MQ。
应用程序部署描述符包含连接工厂的资源引用，并且 **res-auth** 属性设置为“Application”	应用程序服务器进程的用户标识向下流至 IBM MQ。	传递到 ConnectionFactory.createConnection(String username, String password) 方法的用户标识和密码向下流至 IBM MQ。
应用程序部署描述符包含连接工厂的资源引用，并且 **res-auth** 属性设置为“Container”	连接工厂认证别名中指定的用户标识和密码向下流至 IBM MQ。	连接工厂认证别名中指定的用户标识和密码向下流至 IBM MQ。

表 *69: BINDINGS* 方式 *(*继续*)*

配置	应用程序调用 **ConnectionFactory.createConnection()**	应用程序调用 **ConnectionFactory.createConnection(String username, String password)**
应用程序部署描述符包含 **res-auth** 属性设置为“Container”的连接工厂的资源引用，并且为应用程序配置了认证别名	应用程序配置为使用的认证别名中指定的用户标识和密码向下流至 IBM MQ。	应用程序配置为使用的认证别名中指定的用户标识和密码向下流至 IBM MQ。

表 *70: CLIENT* 方式

配置	应用程序调用 **ConnectionFactory.createConnection()**	应用程序调用 **ConnectionFactory.createConnection(String username, String password)**
应用程序部署描述符不包含连接工厂的资源引用，并且连接工厂配置为使用未设置 MCAUSER 属性的 IBM MQ 通道	应用程序服务器进程的用户标识向下流至 IBM MQ。	传递到 `ConnectionFactory.createConnection(String username, String password)` 方法的用户标识和密码向下流至 IBM MQ。
应用程序部署描述符不包含连接工厂的资源引用，并且连接工厂配置为使用 MCAUSER 属性设置为用户标识的 IBM MQ 通道	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。
应用程序部署描述符包含 **res-auth** 属性设置为 *Application* 的连接工厂的资源引用，并且连接工厂配置为使用未设置 MCAUSER 属性的 IBM MQ 通道	应用程序服务器进程的用户标识向下流至 IBM MQ。	传递到 `ConnectionFactory.createConnection(String username, String password)` 方法的用户标识和密码向下流至 IBM MQ。
应用程序部署描述符包含 **res-auth** 属性设置为 *Application* 的连接工厂的资源引用，并且连接工厂配置为使用 MCAUSER 属性设置为用户标识的 IBM MQ 通道	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。
应用程序部署描述符包含 **res-auth** 属性设置为 *Container* 的连接工厂的资源引用，并且连接工厂配置为使用未设置 MCAUSER 属性的 IBM MQ 通道	连接工厂认证别名中指定的用户标识和密码向下流至 IBM MQ。	连接工厂认证别名中指定的用户标识和密码向下流至 IBM MQ。
应用程序部署描述符包含 **res-auth** 属性设置为 *Container* 的连接工厂的资源引用，并且连接工厂配置为使用 MCAUSER 属性设置为用户标识的 IBM MQ 通道	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。

表 70: CLIENT 方式 (继续)		
配置	**应用程序调用 ConnectionFactory.createConnection()**	**应用程序调用 ConnectionFactory.createConnection(String username, String password)**
应用程序部署描述符包含 **res-auth** 属性设置为 *Container* 的连接工厂的资源引用，为应用程序配置了认证别名，并且连接工厂配置为使用未设置 MCAUSER 属性的 IBM MQ 通道	应用程序配置为使用的认证别名中指定的用户标识和密码向下流至 IBM MQ。	应用程序配置为使用的认证别名中指定的用户标识和密码向下流至 IBM MQ。
应用程序部署描述符包含 **res-auth** 属性设置为 *Container* 的连接工厂的资源引用，为应用程序配置了认证别名，并且连接工厂配置为使用 MCAUSER 属性设置为用户标识的 IBM MQ 通道	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。	连接工厂配置为使用的 IBM MQ 通道上 MCAUSER 属性所指定的用户标识向下流至 IBM MQ。

通过直接查找来使用连接工厂

定义 IBM MQ 消息传递提供程序连接工厂后，企业应用程序可以查找连接工厂定义，并使用它来创建与 IBM MQ 队列管理器的 JMS 连接。 可以通过直接查找来执行此操作。

要使用直接查找，企业应用程序可通过发出以下方法调用来连接到应用程序服务器的 JNDI 存储库：

```
InitialContext ctx = new InitialContext();
```

连接到 JNDI 存储库之后，企业应用程序便可以使用连接工厂的 JNDI 名称来识别连接工厂定义，如下所示：

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

注意：

- 在开发企业应用程序期间，应用程序开发者需要知道所需连接工厂的 JNDI 名称。 由于 JNDI 名称已硬编码到应用程序中，因此，如果 JNDI 名称发生更改，那么您需要重新编写并重新部署该应用程序。
- 当通过这种方式使用连接工厂定义时，连接工厂配置为使用的认证别名中指定的用户名和密码不会向下流至 IBM MQ。 这是为了防止未经授权的应用程序识别该连接工厂并用它进行连接，因此这可保护 IBM MQ 系统的安全。

向下流至 IBM MQ 的用户名和密码取决于从连接工厂创建 JMS 连接时所使用的方法。

如果应用程序使用以下方法来创建 JMS 连接：

```
ConnectionFactory.createConnection()
```

那么会将缺省用户身份向下传递到 IBM MQ。 这是用于启动企业应用程序运行所在的应用程序服务器的用户名和密码。

或者，应用程序可以调用以下方法来创建 JMS 连接：

```
ConnectionFactory.createConnection(String username, String password)
```

如果应用程序对连接工厂执行了直接查找，然后调用了此方法，那么传递到 `createConnection()` 方法的用户名和密码将向下流至 IBM MQ。

要点: 在 IBM MQ 8.0 之前，IBM MQ 执行了一次授权检查，目的只是为了确保向下流动的用户名有权访问队列管理器。

但未执行任何密码检查。 要执行认证检查并验证用户名和密码是否有效，必须编写 IBM MQ 通道安全出口。 可在通道安全出口程序中找到有关如何执行此操作的详细信息。

从 IBM MQ 8.0 开始，除了用户名之外，队列管理器还会检查密码。

通过间接查找来使用连接工厂

在编写企业应用程序时，如果连接工厂的 JNDI 名称未知，或者如果使用其他连接工厂（具有不同的 JNDI 名称，这取决于要安装到的应用程序服务器）将应用程序安装在不同的应用程序服务器上，那么可以使用资源引用来查找连接工厂。 可以通过间接查找来执行此操作。

示例

作为使用 jms/myCF 直接查找连接工厂的替代方法，企业应用程序将包含具有本地 JNDI 名称 jms/myResourceReferenceCF 的资源引用。

要使用该 JNDI 名称，应用程序可像执行直接查找一样连接到应用程序服务器的 JNDI 存储库：

```
InitialContext ctx = new InitialContext();
```

现在，应用程序将标识资源引用的 JNDI 名称，而不是直接标识 jms/myCF：

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

您需要本地 JNDI 名称的 java:comp/env 前缀，以告知应用程序服务器企业应用程序正在执行间接查找。

当部署应用程序时，用户将资源引用的 JNDI 名称 jms/myResourceReferenceCF 映射到应用程序已创建的连接工厂的 JNDI 名称 jms/myCF。

运行应用程序时，它将使用应用程序服务器映射到的本地 JNDI 名称来查找 JMS 连接工厂: jms/myCF。 然后，应用程序将使用此连接工厂来创建与 IBM MQ 的连接。

认证别名和间接查找

资源引用还允许定义其他属性来改变所提供的连接工厂的行为。 资源引用的属性之一是 **res-auth**。 此属性的值指定企业应用程序是否应该使用在创建与 IBM MQ 的连接时（如果已定义认证别名）或在应用程序指定自己的用户名和密码时资源引用映射到的连接工厂的认证别名。

此属性的缺省值为 *Application*。 这表示在创建 JMS 连接时，向下流至队列管理器的用户名和密码由应用程序自身决定。 将不使用资源引用所映射到的连接工厂的认证别名。

应用程序可使用以下某个方法来创建 JMS 连接：

- ConnectionFactory.createConnection()
- ConnectionFactory.createConnection(String username, String password)

如果应用程序使用 ConnectionFactory.createConnection()，并且 **res-auth** 设置为 *Application*，那么缺省用户身份将向下流至 IBM MQ。 这是用于启动企业应用程序运行所在的应用程序服务器的用户名和密码。

如果应用程序使用 ConnectionFactory.createConnection(String username, String password)，并且 **res-auth** 设置为 应用程序，那么传递到方法的用户名和密码将向下发送到 IBM MQ。

要在创建连接时使用资源引用所映射到的连接工厂上定义的认证别名，您需要将 **res-auth** 属性设置为 *Container*。 当应用程序创建 JMS 连接时，即使 createConnection 调用指定了用户名和密码，也会使用认证别名详细信息。

使用间接查找时覆盖认证别名

如果应用程序使用 **res-auth** 属性设置为 *Container* 的资源引用，那么可以覆盖在创建 JMS 连接时使用的认证别名。

要覆盖认证别名，资源引用需要包含一个称为 **authDataAlias** 的额外属性，该属性会映射到在其中部署应用程序的应用程序服务器环境中已创建的现有认证别名。 您可以对使用 IBM 提供的 Rational 工具创建的任何资源引用指定此属性。

通过使用此方法，可以在使用已间接查找过的 JMS 连接工厂时使用其他认证别名。 如果指定的认证别名不存在，那么可以在安装企业应用程序之后指定新的认证别名。 有关更多信息，请参阅 WebSphere Application Server 产品文档中的 资源引用 。

WebSphere Application Server 8.5.5 的相关信息

资源引用

WebSphere Application Server 8.0 的相关信息

资源引用

WebSphere Application Server 7.0 的相关信息

资源引用

使用 WebSphere Application Server 集群时用于消息驱动的 bean 的工作负载均衡

使用部署在 WebSphere Application Server 7.0 和 8.0 集群中并配置为以 IBM WebSphere MQ 消息传递提供程序正常方式运行的消息驱动的 bean 应用程序时，其中一个集群成员将处理大部分消息。 您可以均衡集群成员的工作负载，以便在多个集群成员之间分配消息处理。

IBM WebSphere MQ 7.0 引入了名为 **Asynchronous consume** 的新功能，该功能允许应用程序使用名为 **MQCB** 和 **MQCTL** 的 API 异步使用来自队列的消息。

在 WebSphere Application Server 7.0 和 8.0 中运行的使用 IBM WebSphere MQ 消息传递提供程序正常方式的消息驱动的 bean 应用程序将自动使用此功能。 当应用程序启动时，它们将在已配置为通过调用 **MQCB** 进行监视的 JMS 目标上设置异步使用者。 然后调用 **MQCTL** API 来指示应用程序已准备好从 JMS 目标接收消息。

在消息驱动的 bean 应用程序已部署到 WebSphere Application Server 集群后，对于消息驱动的 bean 在其中监视消息的 JMS 目标，每个集群成员都会设置对应的异步使用者。 然后，主管 JMS 目标的 IBM WebSphere MQ 7.0 队列管理器负责在 JMS 目标上存在适合其处理的消息时通知集群成员。

在 IBM WebSphere MQ 7.0.1 Fix Pack 6 之前，队列管理器首选由第一个集群成员在 JMS 目标上设置异步使用者。 当 JMS 目标上收到合适的消息时，该集群成员将第一个收到通知。 然后，启动消息驱动的 bean 应用程序的第一个集群成员将处理 JMS 目标上收到的大部分合适消息。

当 WebSphere Application Server 连接到 IBM WebSphere MQ 7.0.1 Fix Pack 6 或后续版本的队列管理器时，JMS 目标上收到的消息将以更平均的方式分配到该 JMS 目标上已注册的所有异步使用者。 对于在 WebSphere Application Server 7.0 和 IBM MQ 8.0 集群中部署的消息驱动的 bean 应用程序，这意味着将以更平均的方式在集群成员间分配消息。

相关任务

配置 JMS **PROVIDERVERSION** 属性

使用 IBM MQ 头包

IBM MQ 头包提供一组帮助程序接口和类，可用其来处理消息的 IBM MQ 头。 通常，可使用 IBM MQ 头包，因为您希望使用命令服务器（使用可编程命令格式 (PCF) 消息）来执行管理服务。

关于此任务

IBM MQ 头包位于 `com.ibm.mq.headers` 和 `com.ibm.mq.headers.pcf` 包中。 可以将此工具同时用于两个替代 API，IBM MQ 提供这两个 API 供在 Java 应用程序中使用：

- IBM MQ classes for Java（又称为“IBM MQ 基本 Java”）。

- IBM MQ classes for Java Message Service （IBM MQ classes for JMS，又称为“IBM MQ JMS”）。

IBM MQ 基本 Java 应用程序通常会处理 MQMessage 对象，头支持类可与这些对象直接交互，因为其原生理 IBM MQ 基本 Java 接口。

在 IBM MQ JMS 中，消息的有效内容通常是字符串或字节数组对象，可使用 DataInput 和 DataOutput 流对其进行操作。 IBM MQ 头包可用于与这些数据流交互，并适用于处理由 IBM MQ JMS 应用程序发送和接收的任何 MQ 消息。

因此，虽然 IBM MQ 头包包含对 IBM MQ 基本 Java 包的引用，但它也能在 IBM MQ JMS 应用程序内部使用，并适合于在 Java Platform, Enterprise Edition (Java EE) 环境内使用。

您可以使用 IBM MQ 头包的一种典型的方式是处理可编程命令格式 (PCF) 的管理消息，如出于以下任一原因：

- 访问有关 IBM MQ 资源的详细信息。
- 监控队列深度。
- 禁止访问队列。

通过将 PCF 消息与 IBM MQ JMS API 配合使用，可以从 Java EE 应用程序中执行这种以应用程序为中心的资源管理，而不必使用 IBM MQ 基本 Java API。

过程

- 要使用 IBM MQ 头包为 IBM MQ classes for Java 处理消息头，请参阅第 428 页的『与 IBM MQ classes for Java 配合使用』。
- 要使用 IBM MQ 头包为 IBM MQ classes for JMS 处理消息头，请参阅第 429 页的『与 IBM MQ classes for JMS 配合使用』。

与 IBM MQ classes for Java 配合使用

IBM MQ classes for Java 应用程序通常会操作 MQMessage 对象，头支持类可与这些对象直接交互，因为其原生理解 IBM MQ classes for Java 接口。

关于此任务

IBM MQ 提供了一些样本应用程序，演示如何将 IBM MQ 头包与 IBM MQ 基本 Java API (IBM MQ classes for Java) 一起使用。

样本显示两方面信息：

- 如何创建 PCF 消息，以执行管理操作并解析响应消息。
- 如何使用 IBM MQ classes for Java 发送此 PCF 消息。

根据您使用的平台，这些样本安装在 IBM MQ 安装的 `samples` 或 `tools` 目录中的 `pcf` 目录下（请参阅第 286 页的『IBM MQ classes for Java 的安装目录』）。

过程

1. 创建 PCF 消息以执行管理操作并解析响应消息。
2. 使用 IBM MQ classes for Java 发送此 PCF 消息。

相关概念

第 309 页的『使用 IBM MQ classes for Java 处理 IBM MQ 消息头』
提供了一些 Java 类，它们表示不同类型的消息头。 同时还提供了两个助手类。

第 314 页的『使用 IBM MQ classes for Java 处理 PCF 消息』
提供了一些 Java 类，用于创建和解析 PCF 结构的消息，并帮助发送 PCF 请求和收集 PCF 响应。

与 IBM MQ classes for JMS 配合使用

要将 IBM MQ 头与 IBM MQ classes for JMS 一起使用，可执行与 IBM MQ classes for Java 的情况相同的基本步骤。 可使用 IBM MQ 头包以及与 IBM MQ classes for Java 的情况相同的样本代码，以完全相同的方式创建 PCF 消息和解析响应。

关于此任务

要使用 IBM MQ API 发送 PCF 消息，必须将消息有效内容写入 JMS 字节消息，并使用标准 JMS API 发送。 唯一的注意事项是消息不得包含 JMS RFH2，也不得包含 MQMD 中具有特定值的任何其他头。

要发送 PCF 消息，请完成以下步骤。 创建 PCF 消息并从响应消息提取信息的方式与 IBM MQ classes for Java 的情况相同（请参阅第 428 页的『与 IBM MQ classes for Java 配合使用』）。

过程

1. 创建表示 SYSTEM.ADMIN.COMMAND.QUEUE 的 JMS 队列目标。

   IBM MQ JMS 应用程序将 PCF 消息发送到 SYSTEM.ADMIN.COMMAND.QUEUE，需要访问表示此队列的 JMS Destination 对象。 目标必须设置以下属性：

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   如果您正在使用 WebSphere Application Server，那么必须将这些属性定义为目标上的定制属性。

   要以编程方式从应用程序内部创建目标，请使用下列代码：

   ```
   Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
   ((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
       WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQQueue) q1).setMQMDWriteEnabled(true);
   ```

2. 将 PCF 消息转换为包含正确的 MQMD 值的 JMS 字节消息。

   需创建一条 JMS 字节消息，并向其写入 PCF 消息。 需创建响应队列，但这不需要特定的设置。

   以下样本代码片段显示如何创建 JMS 字节消息，并向其写入 com.ibm.mq.headers,pcf.PCFMessage 对象。 先前已使用 IBM MQ 头包构建 PCFMessage 对象 (pcfCmd)。 （请注意，装入 PCFMessage 的包为 com.ibm.mq.headers.pcf.PCFMessage）。

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();

   // Create the wrapping streams to put the bytes into the message payload
   ByteArrayOutputStream baos = new ByteArrayOutputStream();
   DataOutput dataOutput = new DataOutputStream(baos);

   // Set the JMSReplyTo so the answer comes back
   msg.setJMSReplyTo(new MQQueue("adminResp"));

   // write the pcf into the stream
   pcfCmd.write(dataOutput);
   baos.flush();
   msg.writeBytes(baos.toByteArray());

   // we have taken control of the MD, so need to set all
   // flags in the MD that we require - main one is the format
   msg.setJMSPriority(4);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
     CMQC.MQPER_NOT_PERSISTENT);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
   msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
       CMQC.MQRO_PASS_CORREL_ID);
   msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

   // and send the message
   sender.send(msg);
   ```

3. 发送消息，并使用标准 JMS API 接收响应。
4. 将响应消息转换为要处理的 PCF 消息。

要检索响应消息并作为 PCF 消息进行处理，请使用下列代码：

```
 // Get the message back
BytesMessage msg = (BytesMessage) consumer.receive();

// get the size of the bytes message & read into an array
int bodySize = (int) msg.getBodyLength();
byte[] data = new byte[bodySize];
msg.readBytes(data);

// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

相关概念

第 112 页的『JMS 消息』

JMS 消息由头、属性以及主体组成。 JMS 定义了五类消息体。

IBM i 使用 Java 和 JMS 在 IBM i 上设置 IBM MQ

此主题集合概述了如何使用 CL 命令或 qshell 环境在 IBM i 上使用 Java 和 JMS 来设置和测试 IBM MQ 。

注: 从 IBM MQ 8.0 起，`ldap.jar`、`jndi.jar` 和 `jta.jar` 是 JDK 的一部分。

使用 CL 命令

您设置的 CLASSPATH 用于测试 MQ 基础 Java、JMS（带 JNDI）和 JMS（不带 JNDI）。

如果在 `/home/Userprofile` 目录下不使用 `.profile` 文件，那么需要在下面设置系统级别环境变量。 您可以检查以了解其是否是使用 **WRKENVVAR** 命令设置的。

1. 要查看整个系统的环境变量，请发出命令：**WRKENVVAR LEVEL(*SYS)**
2. 要查看特定于您的作业的环境变量，请发出命令：**WRKENVVAR LEVEL(*JOB)**
3. 如果不设置 CLASSPATH，请发出以下命令：

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. 如果不设置 QIBM_MULTI_THREADED，请发出以下命令：

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. 如果不设置 QIBM_USE_DESCRIPTOR_STDIO，请发出以下命令：

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. 如果不设置 QSH_REDIRECTION_TEXTDATA，请发出以下命令：

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

使用 qshell 环境

如果使用 QSHELL 环境，那么可以在 `/home/Userprofile` 目录中设置 `.profile`。 有关更多信息，请参考 Qshell 解释器 (qsh) 文档。

在 .profile 中指定以下内容。 请注意，CLASSPATH 语句必须位于单个行上，或使用 \ 字符按如下所示分到多个行上。

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

通过发出命令 **DSPLIBL** 确保 QMQMJAVA 库位于库列表中。

如果 QMQMJAVA 库不在列表中，使用命令 **ADDLIBLE LIB**（**QMQMJAVA**） 将其添加到列表中。

IBM i 在 IBM i 上使用 Java 测试 IBM MQ

如何使用 MQIVP 样本程序测试 IBM MQ Java。

测试 IBM MQ 基础 Java

执行以下过程：

1. 通过发出以下命令验证队列管理器已启动并且队列管理器的状态为“活动”：

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. 通过发出以下命令验证 JAVA.CHANNEL 服务器连接通道是否已创建：

   ```
   WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

   a. 如果 JAVA.CHANNEL 不存在，可发出以下命令：

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

3. 通过发出 **WRKMQMLSR** 命令验证是否正在对端口 1414 或您使用的任何端口运行队列管理器侦听器。

   a. 如果尚未启动队列管理器的侦听器，请发出以下命令：

   ```
   STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
   ```

运行 MQIVP 样本测试程序

1. 通过发出命令 STRQSH 从命令行启动 qshell
2. 通过发出 **export** 命令验证是否已设置正确的 CLASSPATH，然后如下所示发出 **cd** 命令：

   ```
   cd /qibm/proddata/mqm/java/samples/wmqjava/samples
   ```

3. 通过发出以下命令运行 **java** 程序：

   ```
   java MQIVP
   ```

提示输入以下值时：

- 连接类型

- IP 地址
- 队列管理器名称

您可以按 ENTER 键以使用缺省值。 这会验证产品绑定，可以在 QMQMJAVA 库中找到产品绑定。

您收到与以下示例类似的输出。 请注意，版权声明取决于您正在使用的产品版本。

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

测试 IBM MQ Java 客户机连接

您必须指定:

- 连接类型
- IP 地址
- 端口
- 服务器连接通道
- 队列管理器

您收到与以下示例类似的输出。 请注意，版权声明取决于您正在使用的产品版本。

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i 在 IBM i 上使用 JMS 测试 IBM MQ

如何测试 IBM MQ JMS（带有或不带有 JNDI）

使用 IVTRun 样本测试 JMS（不带 JNDI）

执行以下过程:

1. 通过发出以下命令验证队列管理器已启动并且队列管理器的状态为“活动”：

```
WRKMQM MQMNAME(QMGRNAME)
```

2. 通过发出 **STRQSH** 命令从命令行启动 qshell。
3. 使用 **cd** 命令更改目录，如下所示：

```
cd /qibm/proddata/mqm/java/bin
```

4. 运行脚本文件：

```
IVTRun -nojndi [-m qmgrname]
```

您收到与以下示例类似的输出。 请注意，版权声明取决于您正在使用的产品版本。

```
> IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 5.300
Installation Verification Test


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

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

测试 IBM MQ JMS 客户机模式（不带 JNDI）

执行以下过程：

1. 通过发出以下命令验证队列管理器已启动并且队列管理器的状态为“活动”：

```
WRKMQM MQMNAME(QMGRNAME)
```

2. 通过发出以下命令验证服务器连接通道是否已创建：

```
WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
MQMNAME(QMGRNAME)
```

3. 通过发出 **WRKMQMLSR** 命令验证是否已针对正确的端口启动侦听器：
4. 通过发出 **STRQSH** 命令从命令行启动 qshell。
5. 通过发出 **export** 命令验证 CLASSPATH 是否正确。
6. 使用 **cd** 命令更改目录，如下所示：

```
cd /qibm/proddata/mqm/java/bin
```

7. 运行脚本文件：

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

您收到与以下示例类似的输出。 请注意，版权声明取决于您正在使用的产品版本。

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 5.300
Installation Verification Test


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

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

测试 IBM MQ JMS（带有 JNDI）

通过发出以下命令验证队列管理器已启动并且队列管理器的状态为“活动”：

```
WRKMQM MQMNAME(QMGRNAME)
```

使用 IVTRun 样本测试脚本

执行以下过程：

1. 对 JMSAdmin.config 文件进行适当的更改。 要编辑此文件，请使用 IBM i 命令行中的 **EDTF** (编辑文件) 命令

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. 要针对 WebLogic 使用 LDAP，请从以下行除去注释：

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
   ```

   b. 要针对 WebSphere Application Server 使用 LDAP，请从以下行除去注释：

   ```
   INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
   ```

   c. 要测试文件系统，请从以下行除去注释：

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   ```

   d. 通过从适当的行中除去注释，来确保您已经选择正确的 PROVIDER_URL。

   e. 使用 # 符号注释掉所有其他行。

   f. 完成所有更改之后，按 **F2=Save** 和 **F3=Exit**。

2. 通过发出 **STRQSH** 命令从命令行启动 qshell。
3. 通过发出 **export** 命令验证 CLASSPATH 是否正确。
4. 使用 **cd** 命令更改目录，如下所示：

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

5. 通过发出 **IVTSetup** 命令启动 **IVTSetup** 脚本以创建受管对象（*MQQueueConnectionFactory* 和 *MQQueue*）。
6. 通过发出以下命令运行 IVTRun 脚本：

   ```
   IVTRun -url providerURL [-icf initCtxFact]
   ```

您收到与以下示例类似的输出。 请注意，版权声明取决于您正在使用的产品版本。

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java(tm) Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java(tm) Message Service Administration
```

```
+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java(tm) Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

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

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

使用 Maven 存储库的 Java 应用程序开发

为 IBM MQ 开发 Java 应用程序时，通过使用 Maven 存储库自动安装依赖关系，您无需在使用 IBM MQ 接口之前显式安装任何内容。

Maven 中央存储库

Maven 是一款用于构建应用程序的工具，它还提供了一个存储库来存放应用程序可能需要访问的工件。

Maven 存储库（或中央存储库）的结构允许文件（如 JAR 文件）具有不同的版本，并且可通过众所周知的命名机制来轻松发现这些版本。 然后，构建工具可使用这些名称来动态提取应用程序的依赖项。 在应用程序定义（在使用 Maven 作为构建工具时被称为 POM 文件）中，只需指定依赖项，然后构建进程便知道要在其中执行哪些操作。

IBM MQ 客户机文件

在 `com.ibm.mq GroupId` 下的中央存储库中提供了 IBM MQ Java 客户机接口的副本。 您可以同时找到 `com.ibm.mq.allclient.jar`（通常用于独立程序）和 `wmq.jmsra.rar`（用于 Java EE 应用程序服务器）。 `allclient.jar` 包含 IBM MQ classes for JMS 和 IBM MQ classes for Java。

要点: 不支持使用 Apache Maven 组合件插件 *jar-with-dependencies* 格式构建应用程序，包括 IBM MQ 可再定位的 JAR 文件。

在 Maven 命令处理的 pom.xml 文件中，添加对这些 JAR 文件的依赖关系，如以下示例中所示：

- 要显示应用程序代码与 com.ibm.mq.allclient.jar 之间的关系：

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.1.3.0</version>
    </dependency>
```

- 要使用 Java EE 资源适配器：

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.1.3.0</version>
 <dependency/>
```

有关在 Eclipse 中运行 JMS 项目的简单项目示例，请参阅以下 IBM Developer 文章：Developing Java applications for MQ just got easier with Maven。

开发 C++ 应用程序

IBM MQ 提供等同于 IBM MQ 对象的 C++ 类，以及等同于数组数据类型的一些其他类。 它提供许多不能通过 MQI 使用的功能。

IBM WebSphere MQ 7.0 对 IBM MQ 编程接口的增强功能并不适用于 C++ 类。

IBM MQ C++ 提供以下功能：

- 自动初始化 IBM MQ 数据结构。
- 及时的队列管理器连接和队列开启。
- 隐式队列关闭和队列管理器断开连接。
- 死信消息头传输和接收。
- IMS 网桥头传输和接收。
- 参考消息头传输和接收。
- 触发器消息接收。
- CICS bridge 头传输和接收。
- 工作头传输和接收。
- 客户机通道定义。

下面的 Booch 类图表明，所有类与过程 MQI（例如使用 C）中具有句柄或数据结构的这些 IBM MQ 实体大致相当。 所有类都从 ImqError 类继承而来（请参阅 ImqError C++ 类），这将允许错误条件与每个对象相关联。

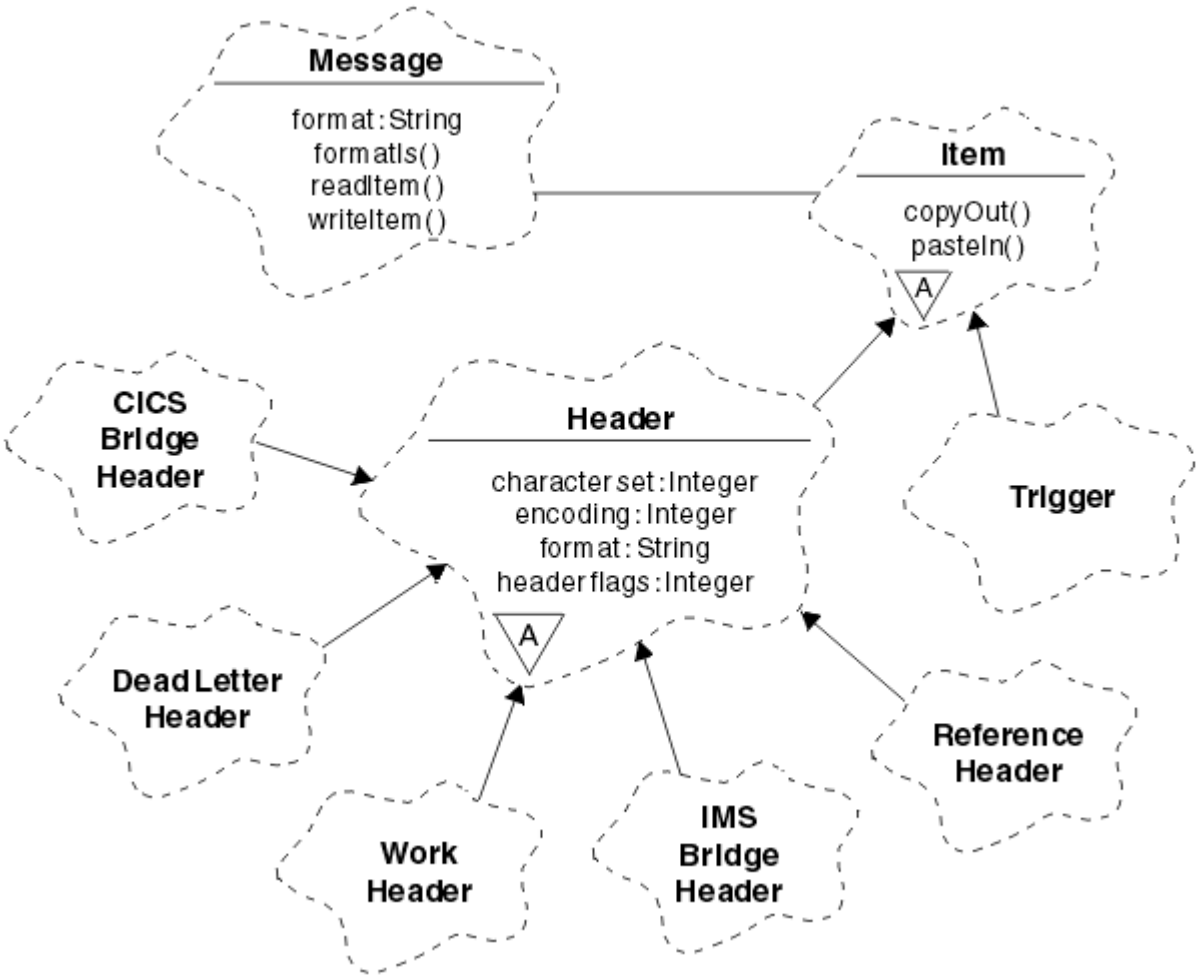


图 *56: IBM MQ C++* 类（项处理）

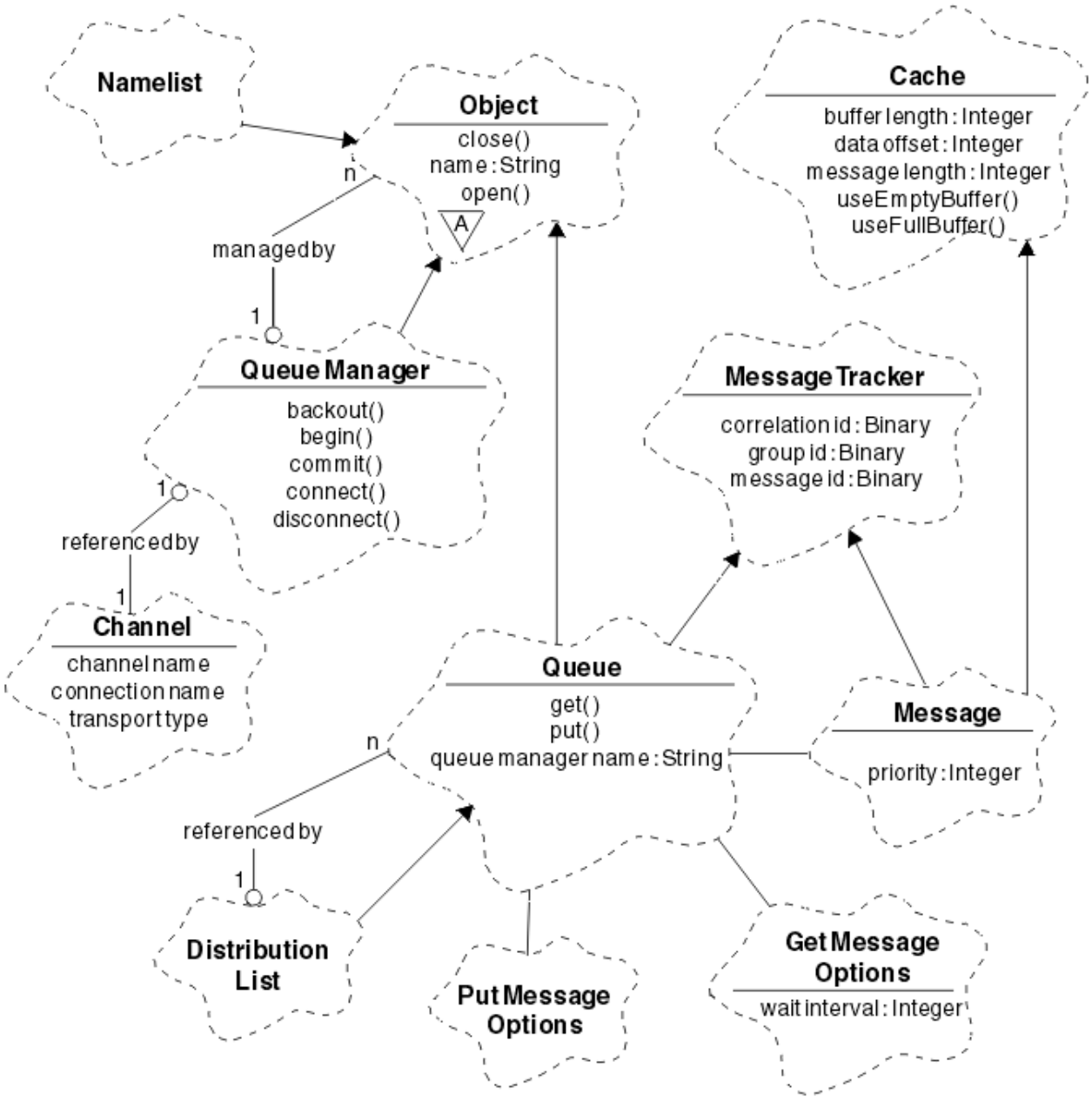


图 *57: IBM MQ C++* 类（队列管理）

要正确解释 Booch 类图，请注意以下约定：

- 方法和需要注意的属性显示在类名称之下。
- 云内的小三角形表示抽象类。
- 继承由指向父类的箭头来表示。
- 云之间未加装饰的线表示类之间的协作关系。
- 使用数字装饰的线表示两个类之间的引用关系。 数字表示在任何一个时刻可以参与特定关系的对象数量。

在队列管理类（请参阅第 439 页的图 57）和项处理类（请参阅第 438 页的图 56）的 C++ 方法特征符中使用以下类和数据类型：

- ImqBinary 类（请参阅 ImqBinary C++ 类），封装诸如 MQBYTE24 之类的字节数组。
- ImqBoolean 数据类型，定义为 **typedef unsigned char ImqBoolean**。
- ImqString 类（请参阅 ImqString C++ 类），封装诸如 MQCHAR64 之类的字符数组。

具有数据结构的实体包含在适当的对象类中。 使用方法访问单独的数据结构字段（请参阅 C++ 和 MQI 交叉引用）。

具有句柄的实体位于 ImqObject 类层次结构之下（请参阅 ImqObject C++ 类），并向 MQI 提供封装的接口。 这些类的对象显示智能行为，可以减少相对于过程 MQI 所需的方法调用数量。 例如，您可以根据需要建立和丢弃队列管理器连接，也可以使用适当的选项打开队列，然后再将其关闭。

ImqMessage 类（请参阅 ImqMessage C++ 类）用于封装 MQMD 数据结构，也通过提供高速缓存区设施来充当用户数据和项（请参阅第 448 页的『使用 C++ 读取消息』）的暂留点。 您可以为用户数据提供固定长度的缓冲区，并多次使用缓冲区。 存在于缓冲区中的数据量会随着使用而变化。 或者，系统可以提供和管理灵活长度的缓冲区。 缓冲区的大小（可用于接收消息的量）和实际使用量（传输的字节数或实际接收的字节数）都是重要的考虑因素。

相关概念

技术概述

第 440 页的『C++ 样本程序』
提供四个样本程序以演示获取和放置消息。

第 444 页的『C++ 语言注意事项』
本主题集合详细描述了在编写使用消息队列接口 (MQI) 的应用程序时，您必须考虑的 C++ 语言用法和约定方面。

第 448 页的『使用 C++ 准备消息数据』
在由系统或应用程序提供的缓冲区中准备消息数据。 任何方法都有一些优势。 这里给出了缓冲区使用示例。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

相关参考

第 453 页的『构建 IBM MQ C++ 程序』
列出支持的编译器的 URL，以及用于在 IBM MQ 平台上编译、链接和运行 C++ 程序和样本的命令。

C++ 和 MQI 交叉引用

IBM MQ C++ 类

C++ 样本程序

提供四个样本程序以演示获取和放置消息。

样本程序是:

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

样本程序位于第 440 页的表 71 所示的目录中。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

表 *71:* 样本程序的位置

环境	包含源的目录	包含构建的目标程序
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
IBM i	/QIBM/ProdData/mqm/samp/	（请参阅注释 第 441 页的『1』）

表 71: 样本程序的位置 (继续)		
环境	**包含源的目录**	**包含构建的目标程序**
Linux Linux	*MQ_INSTALLATION_PATH*/samp	无
Solaris Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/as
Windows Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn （请参阅注释 第 441 页的『2』）
z/OS z/OS	thlqual.SCSQCPPS	

注意:

1. IBM i 使用面向 IBM i 的 ILE C++ 编译器构建的程序位于库 QMQM 中。 样本文件位于 `/QIBM/ProdData/mqm/samp` 中。
2. Windows 使用 Microsoft Visual Studio Visual Studio 构建的程序位于 *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn` 中。 有关这些编译器的更多信息，请参阅第 459 页的『在 Windows 上构建 C++ 程序』。

样本程序 HELLO WORLD (imqwrld.cpp)

该 C++ 样本程序显示如何使用 ImqMessage 类放置和获取常规数据报（C 结构）。

该程序显示如何使用 ImqMessage 类放置和获取常规数据报（C 结构）。 该样本使用了很少的方法调用，利用了隐式方法调用，例如**打开**、**关闭**和**断开连接**。

在除 z/OS 之外的所有平台上

如果正在使用到 IBM MQ 的服务器连接，请执行以下过程之一：

- 要使用现有的缺省队列 SYSTEM.DEFAULT.LOCAL.QUEUE，请运行程序 **imqwrlds**，无需传递任何参数
- 要使用临时动态分配的队列，请运行 **imqwrlds**，传递缺省模型队列的名称 SYSTEM.DEFAULT.MODEL.QUEUE。

如果正在使用到 IBM MQ 的客户机连接，请执行以下过程之一：

- 设置 MQSERVER 环境变量（请参阅 MQSERVER 以了解更多信息）并运行 **imqwrldc**，或者
- 运行 **imqwrldc** ，将 **queue-name**， **queue-manager-name** 和 **channel-definition** 作为参数传递，其中典型的 **channel-definition** 可能是 SYSTEM.DEF.SVRCONN/TCP/*hostname* (1414)

在 z/OS 上

z/OS

使用样本 JCL **imqwrldr** 构造并运行批处理作业。

请参阅 z/OS 批处理、RRS 批处理和 CICS ，获取更多信息。

样本代码

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
```

```
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

样本程序 SPUT (imqsput.cpp) 和 SGET (imqsget.cpp)

这些 C++ 程序会将消息放置到已命名的队列并从中检索消息。

这些样本显示了使用以下类:

- ImqError（请参阅 ImqError C++ 类）
- ImqMessage（请参阅 ImqMessage C++ 类）
- ImqObject（请参阅 ImqObject C++ 类）
- ImqQueue（请参阅 ImqQueue C++ 类）
- ImqQueueManager（请参阅 ImqQueueManager C++ 类）

遵循相应的指示信息运行这些程序。

在除 z/OS 之外的所有平台上

1. 运行 **imqsputs** *queue-name*。
2. 在控制台上输入文本行。 这些行作为消息放置到指定的队列上。
3. 输入空行以结束输入。
4. 运行 **imqsgets** *queue-name* 以检索所有行，并将其显示在控制台上。

z/OS 请参阅第 461 页的『在 z/OS 批处理、RRS 批处理和 CICS 上构建 C++ 程序』，了解更多信息。

在 z/OS 上

z/OS

1. 使用样本 JCL **imqsputr** 构造并运行批处理作业。 从 SYSIN 数据集读取消息。
2. 使用样本 JCL **imqsgetr** 构造并运行批处理作业。 从队列检索消息并将这些消息发送到 SYSPRINT 数据集。

样本程序 DPUT (imqdput.cpp)

该 C++ 样本程序将消息放置到包含两个队列的分发列表中。

DPUT 显示了 ImqDistributionList 类的用法（请参阅 ImqDistributionList C++ 类）。 z/OS 上不支持该样本。

1. 运行 **imqdputs** *queue-name-1 queue-name-2*，以将消息放置到两个已命名的队列上。
2. 运行 **imqsgets** *queue-name-1* 和 **imqsgets** *queue-name-2*，以从这些队列中检索消息。

C++ 语言注意事项

本主题集合详细描述了在编写使用消息队列接口 (MQI) 的应用程序时，您必须考虑的 C++ 语言用法和约定方面。

C++ 头文件

头文件作为 MQI 定义的一部分提供，能够帮助您使用 C++ 语言编写 IBM MQ 应用程序。

下表中汇总了这些头文件。

表 *72: C/C++* 头文件

文件名	内容
IMQI.HPP	C++ MQI 类（包括 CMQC.H 和 IMQTYPE.H）
IMQTYPE.H	定义了 **ImqBoolean** 数据类型
CMQC.H	MQI 数据结构和常量声明

为提高应用程序的可移植性，请在 **#include** 预处理器伪指令上以小写形式对头文件名称进行编码：

```
#include <imqi.hpp> // C++ classes
```

C++ 方法和属性

方法名称为混合大小写形式。 各种注意事项适用于参数和返回值。 可在适当情况下使用 set 和 get 方法来访问属性。

作为 *const* 的方法参数仅用于输入。 具有包括指针 (*) 或引用 (&) 的特征符的参数通过引用来传递。 不包括指针或引用的返回值通过值来传递，就返回对象而言，这些是由调用者负责的新实体。

一些方法特征符包含未指定时采用缺省值的项。 这类项总是位于特征符的末尾并由等号 (=) 来表示，等号之后的值表示忽略该项时应用的缺省值。

这些类中的所有方法名称都是混合大小写形式，并以小写字母开头。 除了方法名称中的第一个单词之外，每个单词都以大写字母开头。 除非这些单词的含义为众人所理解，否则不会使用缩写。 所用的缩写包含 *id*（用于身份）和 *sync*（用于同步）。

使用 set 和 get 方法访问对象属性。 set 方法以单词 *set* 开头，get 方法没有任何前缀。 如果属性为只读，那么不使用 set 方法。

在对象构造期间，会将属性初始化为有效状态，且对象的状态总是一致的。

C++ 中的数据类型

所有数据类型都由 C **typedef** 语句来定义。

在 IMQTYPE.H 中，将类型 **ImqBoolean** 定义为 **unsigned char**，并且可具有 TRUE 和 FALSE 值。 您可以使用 **ImqBinary** 类对象代替 **MQBYTE** 数组，使用 **ImqString** 类对象代替 **char ***。 许多方法都会返回对象而不是 **char** 或 **MQBYTE** 指针，以简化存储器管理。 所有返回值都由调用者负责，就返回对象而言，可以使用删除来处理存储器。

在 C++ 中处理二进制字符串

二进制数据的字符串可声明为 **ImqBinary** 类的对象。 可使用熟悉的 C 操作程序来复制、比较和设置此类的对象。 提供了示例代码。

以下代码样本显示了对二进制字符串执行的操作：

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
...
```

在 C++ 中处理字符串

通常在 **ImqString** 类对象中返回字符数据，可使用转换运算符将这些类对象转换为 **char ***。 ImqString 类包含用于辅助处理字符串的方法。

在使用 MQI C++ 方法接受或返回字符数据时，字符数据总是以 Null 结束，并且可以为任意长度。 但是，IBM MQ 施加的一些限制可能会导致信息被截断。 为了简化存储器管理，通常在 **ImqString** 类对象中返回字符数据。 使用提供的转换运算符可将这些对象转换为 **char ***，并在需要 **char *** 的许多情况下用于只读目的。

注: 从 **ImqString** 类对象产生的 **char *** 转换结果可能为 Null。

虽然可以在 **char *** 上使用 C 函数，但仍有优先的 **ImqString** 类的特殊方法；**operator length** ( ) 等同于 **strlen**，**storage** ( ) 表示为字符数据分配的内存。

C++ 中对象的初始状态

所有对象都由其属性反映出一致初始状态。 初始值在类描述中定义。

通过 C++ 使用 C

在通过 C++ 程序使用 C 函数时，请包含相应的头。

以下示例显示了 C++ 程序中包含的 `string.h`：

```
extern "C" {
#include <string.h>
}
```

C++ 表示法约定

该示例显示了如何调用方法和声明参数。

此代码样本使用方法和参数 **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

如下所示声明并使用参数:

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

...
}
```

C++ 中的隐式操作

可及时执行几个隐式操作，以满足成功执行某种方法的先决条件。 这些隐式操作是连接、打开、重新打开、关闭和断开连接。 您可以使用类属性来控制连接和打开隐式行为。

连接

对于导致任何 MQI 调用的任何方法，都会自动连接一个 ImqQueueManager 对象（请参阅 C++ 和 MQI 交叉引用）。

打开

对于导致 MQGET、MQINQ、MQPUT 或 MQSET 调用的任何方法，都会自动打开一个 ImqObject 对象。 使用 **openFor** 方法以指定一个或多个相关的**打开选项**值。

重新打开

对于导致 MQGET、MQINQ、MQPUT 或 MQSET 调用的任何方法，都会自动重新打开一个 ImqObject，其中对象已打开，但现有的**打开选项**不足以成功调用 MQI。 使用临时**关闭选项**值 MQCO_NONE 临时关闭对象。 使用 **openFor** 方法来添加相关的 **open 选项**。

在特定情况下，重新打开可能会导致问题:

- 临时动态队列在关闭时会被销毁，且永远无法重新打开。
- 在关闭和重新打开期间，其他人可能有机会访问打开的用于独占输入的队列（显式或缺省情况下）。
- 当队列关闭时，会丢失浏览光标位置。 此情况不会阻止关闭和重新打开，但会阻止后续使用光标，直到再次使用 MQGMO_BROWSE_FIRST。
- 当队列关闭时，会丢失检索的最后一条消息的上下文。

如果其中任何一种情况发生或可预见，请在打开对象（显式或隐式）之前通过显式设置足够的**打开选项**来避免重新打开。

为复杂的队列处理情况显式设置**打开选项**能够提高性能，并避免与使用重新打开相关的问题。

关闭

ImqObject 在对象状态不再可行的任意时刻都会自动关闭，例如，ImqObject 连接引用被切断，或 ImqObject 对象被销毁。

断开连接

ImqQueueManager 在连接不再可行的任意时刻都会自动断开连接，例如，ImqObject 连接引用被切断，或 ImqQueueManager 对象被销毁。

C++ 中的二进制和字符串

ImqString 类用于封装传统的 *char ** 数据格式。 ImqBinary 类用于封装二进制字节数组。 用于设置字符数据的一些方法可能会截断数据。

用于设置字符 (**char ***) 数据的方法总是会获取数据的副本，但由于 IBM MQ 施加了某些限制，因此某些方法可能会截断副本。

ImqString 类（请参阅 ImqString C++ 类 ）用于封装传统的 **char ***，并提供以下支持：

- 比较
- 连接
- 复制
- 整数到文本和文本到整数的转换
- 令牌（单词）抽取
- 大写转换

ImqBinary 类（请参阅 ImqBinary C++ 类）用于封装任意大小的二进制字节数组。 尤其是用于保存以下属性：

- **记帐令牌** (MQBYTE32)
- **连接标记** (MQBYTE128)
- **相关标识** (MQBYTE24)
- **工具令牌** (MQBYTE8)
- **组标识** (MQBYTE24)
- **实例标识** (MQBYTE24)
- **消息标识** (MQBYTE24)
- **消息令牌** (MQBYTE16)
- **事务实例标识** (MQBYTE16)

其中这些属性属于以下类的对象：

- ImqCICSBridgeHeader（请参阅 ImqCICSBridgeHeader C++ 类）
- ImqGetMessageOptions（请参阅 ImqGetMessageOptions C++ 类）
- ImqIMSBridgeHeader（请参阅 ImqIMSBridgeHeader C++ 类）
- ImqMessageTracker（请参阅 ImqMessageTracker C++ 类）
- ImqQueueManager（请参阅 ImqQueueManager C++ 类）
- ImqReferenceHeader（请参阅 ImqReferenceHeader C++ 类）
- ImqWorkHeader（请参阅 ImqWorkHeader C++ 类）

ImqBinary 类同样也支持比较和复制。

C++ 中不支持的函数

IBM MQ C++ 类和方法不依赖于 IBM MQ 平台。 可能会因此提供一些在某些平台上不支持的函数。

如果尝试在不支持某个函数的平台上使用该函数，那么会通过 IBM MQ 而不是 C++ 语言绑定检测到该函数。 IBM MQ 会向您的程序报告错误，类似于其他任何 MQI 错误。

C++ 中的消息传递

该主题集合详细描述了如何使用 C++ 准备、读取以及编写消息。

使用 C++ 准备消息数据

在由系统或应用程序提供的缓冲区中准备消息数据。 任何方法都有一些优势。 这里给出了缓冲区使用示例。

在发送消息时，先在由 ImqCache 对象管理的缓冲区中准备消息数据（请参阅 ImqCache C++ 类）。 与每个 ImqMessage 对象关联缓冲区（通过继承）（请参阅 ImqMessage C++ 类）：可由应用程序（使用 **useEmptyBuffer** 或 **useFullBuffer** 方法）或系统自动提供。 通过应用程序提供消息缓冲区的优势在于，在许多情况下，都不需要数据复制，因为应用程序可以直接使用准备就绪的数据区。 劣势在于提供的缓冲区的长度是固定的。

缓冲区可以复用，并且在传输之前通过使用 **setMessageLength** 方法，每次传输的字节数可以变化。

在系统自动提供缓冲区时，可用的字节数由系统管理，并且可使用一些方法（如 ImqCache **write** 方法或 ImqMessage **writeItem** 方法）将数据复制到消息缓冲区。 消息缓冲区可根据需求增长。 随着缓冲区的增长，之前编写的数据不会丢失。 大型或多部分消息可按顺序写入。

以下示例显示了简化的消息发送。

1. 在用户提供的缓冲区中使用准备好的数据

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. 在用户提供的缓冲区中使用准备好的数据，该缓冲区的大小超过了数据大小

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. 将数据复制到用户提供的缓冲区

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. 将数据复制到系统提供的缓冲区

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. 使用对象（对象设置了消息格式以及内容）将数据复制到系统提供的缓冲区

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

使用 C++ 读取消息

缓冲区可由应用程序或系统提供。 可直接从缓冲区访问数据或按顺序读取数据。 有一个等同于每个消息类型的类。 这里给出了样本代码。

在收到数据时，应用程序和系统可以提供合适的消息缓冲区。 同一缓冲区可用于特定 ImqMessage 对象的多次传输和多次接收。 如果自动提供消息缓冲区，它会增长以容纳接收的任何长度的数据。 但是，应用程序提供的消息缓冲区可能不够大，不足以保存接收的数据。 这样可能就会发生截断或故障，具体取决于用于接收消息的选项。

可直接从消息缓冲区访问入局数据，在这种情况下，数据长度表示入局数据的总量。 或者，可从消息缓存区按顺序读取入局数据。 在这种情况下，数据指针会寻址入局数据的下一个字节，且每次读取数据时都会更新数据指针和数据长度。

项属于消息部分，这些全都位于消息缓冲区的用户区域，需要有序且单独处理。 除常规用户数据之外，项还可能是死信消息头或触发器消息。 项总是与消息格式关联，但消息格式却**不**一定与项关联。

与可识别 IBM MQ 消息格式对应的每个项都有一个对象类。 死信消息头和触发器消息各有一个对象类。 用户数据没有对象类。 即，在用完可识别格式之后，处理剩余部分的任务将留给应用程序。 可通过专门化 ImqItem 类来编写用户数据的类。

以下示例显示了在假设情况下，考虑在用户数据之前的潜在项数目的消息接收情况。 将非项用户数据定义为在可识别项之后出现的任何内容。 自动缓冲区（缺省情况下）用于保存任意数量的消息数据。

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */
```

```
        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

在此示例中，FMT_USERCLASS 是常量，表示与 UserClass 类的对象相关联的 8 字符格式名称，并由应用程序定义。

UserClass 来源于 ImqItem 类（请参阅 ImqItem C++ 类），并实施了该类的虚拟 **copyOut** 和 **pasteIn** 方法。

接下来的两个示例显示了 ImqDeadLetterHeader 类的代码（请参阅 ImqDeadLetterHeader C++ 类）。 第一个示例显示了定制封装消息编写代码。

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }

  return bSuccess ;
}
```

第二个示例显示了定制封装消息读取代码。

```
// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
```

```
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

对于自动缓冲区，缓冲存储是不稳定的。 即，每次调用 **get** 方法之后，缓冲区数据可能保存在不同的物理位置。 因此，每次引用缓冲区数据时，都会使用 **bufferPointer** 或 **dataPointer** 方法来访问消息数据。

您可能希望程序为接受消息数据留出固定区域。 在这种情况下，可在使用 **get** 方法之前调用 **useEmptyBuffer** 方法。

因为使用固定非自动区域将消息限制为最大大小，所以务必要考虑使用 ImqGetMessageOptions 对象的 MQGMO_ACCEPT_TRUNCATED_MSG 选项。 如果未指定该选项（缺省情况下），可能会出现 MQRC_TRUNCATED_MSG_FAILED 原因码。 如果指定了该选项，可能会出现 MQRC_TRUNCATED_MSG_ACCEPTED 原因码，具体取决于应用程序的设计。

接下来的示例显示了如何使用存储器的固定区域来接收消息：

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

在此代码片段中，始终可以使用 *pszBuffer* 直接寻址缓冲区，而不需要使用 **bufferPointer** 方法。 但是，最好使用 **dataPointer** 方法进行通用访问。 应用程序必须（不是 ImqCache 类对象）必须丢弃用户定义（非自动）缓冲区。

注意： 使用 **useEmptyBuffer** 指定空指针和零长度不会按预期指定长度为零的固定长度缓冲区。 这种组合被解释为忽略任何先前用户定义缓冲区的请求，而是恢复为使用自动缓冲区。

使用 C++ 将消息写入死信队列

将消息写入死信队列的示例程序代码。

通常情况下，多部分消息包含死信消息头。 来自无法处理的消息的数据将会附加到死信消息头。

```
ImqQueueManager mgr ;        // The queue manager.
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

使用 C++ 将消息写入 IMS 网桥

将消息写入 IMS 网桥的示例程序代码。

发送至 IBM MQ - IMS 网桥的消息可能会使用特殊的头。 在常规消息数据之前附加 IMS 网桥头。

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
```

```
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

使用 C++ 将消息写入 CICS bridge

将消息写入 CICS bridge 的示例程序代码。

使用 CICS bridge 发送到 IBM MQ for z/OS 的消息需要特殊头。 在常规消息数据之前附加 CICS bridge 头。

```
ImqQueueManager mgr ;         // The queue manager.
ImqQueue queueIn ;            // Incoming message queue.
ImqQueue queueBridge ;        // CICS bridge message queue.
ImqMessage msg ;              // Incoming and outgoing message.
ImqCicsBridgeHeader header ;  // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

使用 C++ 编写带有工作头的消息

用于编写消息的示例程序代码，该消息预定发送至由 z/OS 工作负载管理器管理的队列。

发送到 IBM MQ for z/OS 的消息（预定发送至 z/OS 工作负载管理器管理的队列）需要一个特殊头。 在常规消息数据之前附加工作头。

```
ImqQueueManager mgr ;             // The queue manager.
ImqQueue queueIn ;                // Incoming message queue.
ImqQueue queueWLM ;               // WLM managed queue.
ImqMessage msg ;                  // Incoming and outgoing message.
ImqWorkHeader header ;            // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

构建 IBM MQ C++ 程序

列出支持的编译器的 URL，以及用于在 IBM MQ 平台上编译、链接和运行 C++ 程序和样本的命令。

有关 IBM MQ 支持的每个平台和版本的编译器列表，请参阅 IBM MQ 的系统需求。

编译和链接 IBM MQ C++ 程序所需的命令取决于您的安装和需求。 以下示例显示了一些编译器的典型编译和链接命令，这些命令在多个平台上均使用 IBM MQ 的缺省安装。

AIX 在 AIX 上构建 C++ 程序

使用 XL C Enterprise Edition 编译器在 AIX 上构建 IBM MQ C++ 程序。

客户机

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

32 位非线程应用程序

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32 位线程应用程序

```
xlC_r -o imqsputc_32_r imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64 位非线程应用程序

```
xlC -q64 -o imqsputc_64 imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64 位线程应用程序

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

服务器

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

32 位非线程应用程序

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32 位线程应用程序

```
xlC_r -o imqsput_32_r imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64 位非线程应用程序

```
xlC -q64 -o imqsput_64 imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64 位线程应用程序

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

IBM i 在 IBM i 上构建 C++ 程序

使用 ILE C++ 编译器在 IBM i 上构建 IBM MQ C++ 程序。

IBM ILE C++ for IBM i 是用于 C++ 程序的本机编译器。 以下指示信息描述如何使用此编译器通过 *Hello World!* 创建 IBM MQ C++ 应用程序 以 IBM MQ 样本程序为例。

1. 按照 首先读*!* 中的指示安装 ILE C++ for IBM i 编译器 与产品相关的手册。

2. 确保 QCXXN 库位于您的库列表中。
3. 创建 HELLO WORLD 样本程序：

   a. 创建模块：

```
CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
TERASPACE(*YES)
```

   可在 /QIBM/ProdData/mqm/samp 中找到 C++ 样本程序的源，并在 /QIBM/ProdData/mqm/inc 中找到包含文件。

   或者，也可在库 SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD) 中找到该源。

   b. 将其与 IBM MQ 提供的服务程序绑定，以生成程序对象：

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

   使用可重入服务程序构建线程应用程序：

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

   c. 使用 SYSTEM.DEFAULT.LOCAL.QUEUE 执行 HELLO WORLD 样本程序：

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux 在 Linux 上构建 C++ 程序

使用 GNU g + + 编译器在 Linux 上构建 IBM MQ C++ 程序。

System p

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

客户机：System p

32 位非线程应用程序

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

32 位线程应用程序

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

64 位非线程应用程序

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 位线程应用程序

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

服务器：System p

32 位非线程应用程序

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

32 位线程应用程序

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

64 位非线程应用程序

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 位线程应用程序

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

客户机：IBM Z

32 位非线程应用程序

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32 位线程应用程序

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

64 位非线程应用程序

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 位线程应用程序

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

服务器：IBM Z

32 位非线程应用程序

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

32 位线程应用程序

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64 位非线程应用程序

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 位线程应用程序

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64（32 位）

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

客户机：x86-64（32 位）

32 位非线程应用程序

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

32 位线程应用程序

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

64 位非线程应用程序

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

64 位线程应用程序

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

服务器：x86-64（32 位）

32 位非线程应用程序

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

32 位线程应用程序

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

64 位非线程应用程序

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64 位线程应用程序

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Solaris 在 Solaris 上构建 C++ 程序

使用 Sun ONE 编译器在 Solaris 上构建 IBM MQ C++ 程序。

SPARC

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

客户机：SPARC

32 位应用程序

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

64 位应用程序

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

服务器：SPARC

32 位应用程序

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

64 位应用程序

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

客户机：x86-64

32 位应用程序

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

64 位应用程序

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

服务器：x86-64

32 位应用程序

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

64 位应用程序

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

Windows 在 Windows 上构建 C++ 程序

使用 Microsoft Visual Studio C++ 编译器在 Windows 上构建 IBM MQ C++ 程序。

注意: IBM MQ 随附的库为动态库，而非静态库。 IBM MQ 提供了一些称为 "`import libraries`" 的内容，仅供您在编译时使用。 对于运行时，您必须使用动态库。

从 IBM MQ 8.0.0 Fix Pack 4 起，IBM MQ 随附可再分发的客户机，其中包含运行 IBM MQ 应用程序所需的库。 这些库随客户机应用程序一起打包和再分发。 有关更多信息，请参阅 Windows 上的可再分发的客户机。

用于 32 位应用程序的库 (.lib) 文件和 dll 文件安装在 *MQ_INSTALLATION_PATH*/Tools/Lib 中。 用于 64 位应用程序的文件都安装在 *MQ_INSTALLATION_PATH*/Tools/Lib64 中。 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。

客户机

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

服务器

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

安装 Universal C Runtime

如果使用 Windows 8.1 或 Windows Server 2012 R2，那么必须安装来自 Microsoft 的 Universal C Runtime 更新 (Universal CRT)。 Windows 10 和 Windows Server 2016 中都包含此运行时。

Universal CRT 更新即是 Microsoft 更新 KB3118401。 您可以通过在 C:\Windows\System32 目录中搜索名为 ucrtbase.dll 的文件来检查是否有此更新。 如果未安装，可从以下 Microsoft 页面中下载此更新：https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401。

在未安装此运行时的情况下尝试运行 IBM MQ 程序或您使用 Microsoft Visual Studio 2017 自行编译的程序将导致如下错误：

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

为 Microsoft Visual Studio 2012 程序提供运行时

如果使用 Microsoft Visual Studio 2012 编译了 IBM MQ 程序，请注意 IBM MQ 安装程序不会安装 Microsoft Visual Studio 2012 C/C++ 运行时。 如果在该计算机上已安装先前版本的 IBM MQ，那么可从该安装中获取 Microsoft Visual Studio 2012 运行时。

但是，如果使用的程序是使用 Microsoft Visual Studio 2012 构建的，并且未安装先前版本的 IBM MQ，那么必须执行以下某项操作：

- 从 Microsoft 下载并安装 **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)**。
- 使用 Microsoft Visual Studio 2017 或其他已安装这些运行时的 Microsoft Visual Studio 版本来重新编译程序。

使用 Microsoft Visual Studio 2015 编译器构建的 C++ 客户机库

IBM MQ 提供了使用 Microsoft Visual Studio 2015 C++ 编译器和 Microsoft Visual Studio 2017 C++ 编译器构建的 C++ 客户机库。

同时提供了 32 位和 64 位版本的 IBM MQ C++ 库。 32 位库安装在 bin\vs2015 文件夹下，64 位库安装在 bin64\vs2015 文件夹下。

缺省情况下，IBM MQ 已配置为使用 Microsoft Visual Studio 2017 库。 要使用 Microsoft Visual Studio 2015 库，必须先将 MQ_PREFIX_VS_LIBRARIES 环境变量设置为 MQ_PREFIX_VS_LIBRARIES=vs2015，然后才能安装 IBM MQ 或者使用 **setmqenv** 或 **setmqinst** 命令。

使用命名不同的 IBM MQ C++ 库

IBM MQ 额外提供了一些命名不同的 C++ 客户机库。 这些库是使用 Microsoft Visual Studio 2015 和 Microsoft Visual Studio 2017 C++ 编译器构建的。 这些库是在使用 Microsoft Visual Studio 2017 C++ 编译

器构建的现有 C++ 库之外额外提供的。 由于这些额外的 IBM MQ C++ 库具有不同的名称，因此您可以在同一计算机上运行 IBM MQ C++ 应用程序，这些应用程序是使用 IBM MQ C++ 构建的，并使用 Microsoft Visual Studio 2017 和更低版本的产品进行编译。

附加的 Microsoft Visual Studio 2017 库具有以下名称：

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

附加的 Microsoft Visual Studio 2015 库具有以下名称：

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

同时提供了这些库的 32 位和 64 位版本。 32 位库安装在 `bin` 文件夹下，64 位库安装在 `bin64` 文件夹下。 相应的导入库将安装在 `Tools\lib` 和 `Tools\lib64` 目录下。

如果应用程序使用 `imq*vs2015.lib` 文件，那么必须使用 Microsoft Visual Studio 2015 编译器编译该应用程序。 要运行使用 Microsoft Visual Studio 2015 编译的 IBM MQ C++ 应用程序或在同一计算机上使用较低版本的产品编译的应用程序，必须将 PATH 环境变量作为前缀，如以下示例中所示：

- 对于 32 位应用程序：

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- 对于 64 位应用程序：

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

相关参考

Windows：IBM MQ 8.0 以来的更改

z/OS 在 z/OS 批处理、RRS 批处理和 CICS 上构建 C++ 程序

针对批处理， RRS 批处理或 CICS 环境在 z/OS 上构建 IBM MQ C++ 程序，并运行样本程序。

您可以为 IBM MQ for z/OS 支持的三种环境编写 C++ 程序：

- 批处理
- RRS 批处理
- CICS

编译、预链接和链接

通过编译、预链接以及链接编辑您的 C++ 源代码来创建 z/OS 应用程序。

IBM MQ C++ for z/OS 是作为 z/OS DLL 针对 IBM C++ for z/OS 语言实现的。 使用 DLL，您可以在预链接时间将提供的定义备卡与编译器输出连接。 这使链接程序能够检查对 IBM MQ C++ 成员函数的调用。

注: 对于三种环境中的每种环境，都有三组备卡。

要构建 IBM MQ for z/OS C++ 应用程序，请创建并运行 JCL。 使用以下过程：

1. 如果您的应用程序在 CICS 下运行，请在程序中使用 CICS 提供的过程来转换 CICS 命令。

   此外，对于 CICS 应用程序，您需要：

   a. 将 SCSQLOAD 库添加到 DFHRPL 连接。

b. 使用 SCSQPROC 库中的成员 IMQ4B100 定义 CSQCAT1 CEDA 组。

c. 安装 CSQCAT1。

2. 编译程序以生成对象代码。 用于编译的 JCL 必须包括使产品数据定义文件可供编译器使用的语句。 数据定义在以下 IBM MQ for z/OS 库中提供：

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   确保指定了 /cxx 编译器选项。

   注: 名称 **thlqual** 是 z/OS 上 IBM MQ 安装库的高级限定符。

3. 预链接步骤 第 462 页的『2』 中创建的对象代码，包括 **thlqual**.SCSQDEFS 中提供的以下定义备卡：

   a. 用于批处理的 imqs23dm 和 imqb23dm

   b. 用于 RRS 批处理的 imqs23dr 和 imqb23dr

   c. 用于 CICS 的 imqs23dc 和 imqb23dc

   以下是相应的 DLL。

   a. 用于批处理的 imqs23im 和 imqb23im

   b. 用于 RRS 批处理的 imqs23ir 和 imqb23ir

   c. 用于 CICS 的 imqs23ic 和 imqb23ic

4. 链接编辑步骤 第 462 页的『3』 中创建的对象代码以生成装入模块，并将该模块存储在应用程序装入库中。

要运行批处理或 RRS 批处理程序，请将库 **thlqual**.SCSQAUTH 和 **thlqual**.SCSQLOAD 包含在 STEPLIB 或 JOBLIB 数据集连接中。

要运行 CICS 程序，首先请让系统管理员针对 CICS 将其定义为 IBM MQ 程序和事务。 然后，您可以照常运行该程序。

运行样本程序

第 440 页的『C++ 样本程序』 中描述了这些程序。

样本应用程序仅以源形式提供。 这些文件是：

表 *73: z/OS* 样本程序文件

样本	源程序（位于库 thlqual.SCSQCPPS 中）	JCL（位于库 thlqual.SCSQPROC 中）
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

要运行样本，请如同使用任何 C++ 程序一样编译和链接编辑这些样本（请参阅第 461 页的『在 z/OS 批处理、RRS 批处理和 CICS 上构建 C++ 程序』）。 使用提供的 JCL 来构造和运行批处理作业。 您必须先通过遵循 JCL 包含的注释来定制 JCL。

z/OS 在 z/OS UNIX 系统服务上构建 C++ 程序

在 z/OS 上为 Unix 系统服务构建 IBM MQ C++ 程序。

要在 UNIX 系统服务 shell 下构建应用程序，您必须为编译器提供对 IBM MQ 包含文件（位于 `thlqual.SCSQC370` 和 `hlqual.SCSQHPPS` 中）的访问权，并针对两个 DLL 备卡（位于 `thlqual.SCSQDEFS` 中）进行链接。 在运行时，应用程序需要访问 IBM MQ 数据集 `thlqual.SCSQLOAD`、`thlqual.SCSQAUTH` 以及某个特定于语言的数据集，例如 `thlqual.SCSQANLE`[6].

[6] 您可以与 "预链接对象代码 " 中列出的任何侧板链接，以在以下三个环境中的任何环境中运行 UNIX 系统服务：第 461 页的『在 z/OS 批处理、RRS 批处理和 CICS 上构建 C++ 程序』

编译

1. 使用 TSO **oput** 命令或使用 FTP 将样本复制到文件系统中。 本示例的其余部分假定您已将样本复制到名为 /u/fred/sample 的目录中，并将其命名为 imqwrld.cpp。
2. 登录到 UNIX 系统服务 shell，然后切换到您放置样本的目录。
3. 设置 C++ 编译器，以便它可以接受 DLL 备卡和 .cpp 文件作为输入：

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. 编译并链接样本程序。 以下命令将程序与批处理备卡链接，可改为使用 RRS 批处理备卡。 \ 字符用于将命令拆成多行。 请勿输入该字符，以单行输入该命令：

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

有关 TSO **oput** 命令的更多信息，请参阅 z/OS UNIX System Services Command Reference。

您也可以使用 make 实用程序来简化 C++ 程序的构建过程。 这是用于构建 HELLO WORLD C++ 样本程序的样本 Makefile。 它会将编译阶段和链接阶段分开。 在运行 make 实用程序之前，请先按照步骤 第 463 页的『3』 设置环境。

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

请参阅 z/OS UNIX System Services Programming Tools，以获取有关使用 make 的更多信息。

正在运行

1. 登录到 UNIX 系统服务 shell，然后切换到构建样本的目录。
2. 设置 STEPLIB 环境变量以包含 IBM MQ 数据集：

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. 运行该样本：

```
/u/fred/sample:> ./imqwrld
```

开发 .NET 应用程序

IBM MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 IBM MQ MQI client 连接到 IBM MQ 或直接连接到 IBM MQ 服务器。

如果具有使用 Microsoft .NET Framework 的应用程序，并且希望充分利用 IBM MQ 的功能，那么必须使用 IBM MQ classes for .NET。 有关更多信息，请参阅第 468 页的『安装 IBM MQ classes for .NET Framework』。

V 9.1.1 从 IBM MQ 9.1.1 开始，对于 Windows 环境中的应用程序， IBM MQ 支持 .NET Core 。 有关更多信息，请参阅第 465 页的『安装 IBM MQ classes for .NET Standard』。

V 9.1.2 从 IBM MQ 9.1.2 开始，对于 Linux 环境中的应用程序， IBM MQ 支持 .NET Core 。

V 9.1.4 从 IBM MQ 9.1.4 开始，IBM MQ .NET 受管应用程序能够自动均衡集群队列管理器中的连接。 .NET Framework 和 .NET Standard 库都受支持。 有关更多信息，请参阅统一集群和自动应用程序均衡。

面向对象的 IBM MQ .NET 接口与 MQI 接口不同，它使用对象的方法而不是 MQI 动词。

过程化 IBM MQ 应用程序编程接口基于动词而构建，例如以下列表中的动词：

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

这些动词全部作为参数，会获取它们要进行操作的 IBM MQ 对象的句柄。 由于 .NET 面向对象，因此 .NET 编程接口会将其进行转换。 您的程序由一系列 IBM MQ 对象组成，通过调用这些对象的方法可以执行程序。 您可以使用 .NET 支持的任何语言编写程序。

使用程序化接口时，通过调用 MQDISC(*Hconn*, CompCode, Reason) 可断开与队列管理器的连接，其中 *Hconn* 是到队列管理器的句柄。

在 .NET 接口中，队列管理器由一个 MQQueueManager 类对象表示。 通过调用该类的 Disconnect() 方法可以断开与队列管理器的连接。

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

IBM MQ classes for .NET 是一套支持 .NET 应用程序与 IBM MQ 交互的类。 它们表示您的应用程序使用的各种 IBM MQ 组件，如队列管理器、队列、通道以及消息。 有关这些类的详细信息，请参阅 IBM MQ .NET 类和接口。

必须首先安装 .NET Framework，之后才能编译您编写的任何应用程序。 有关安装 IBM MQ classes for .NET 和 .NET 框架的指示信息，请参阅第 468 页的『安装 IBM MQ classes for .NET Framework』。

相关概念

第 465 页的『将 IBM MQ classes for .NET 连接至队列管理器的选项』
通过三种方式可以将 IBM MQ classes for .NET 连接至队列管理器。 请考虑哪种类型的连接最适合您的需求。

第 480 页的『编写和部署 IBM MQ .NET 程序』
要使用 IBM MQ classes for .NET 访问 IBM MQ 队列，需要使用 .NET 支持的任一语言编写程序，并在程序中包含向 IBM MQ 队列放入消息以及从中获取消息的调用。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

相关任务

IBM MQ .NET 问题故障诊断

第 1141 页的『使用 IBM MQ 开发 Microsoft Windows Communication Foundation 应用程序』
IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 定制通道在 WCF 客户机和服务之间发送和接收消息。

相关参考

技术概述

开始使用 IBM MQ classes for .NET

IBM MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 IBM MQ MQI client 连接到 IBM MQ 或直接连接到 IBM MQ 服务器。

将 IBM MQ classes for .NET 连接至队列管理器的选项

通过三种方式可以将 IBM MQ classes for .NET 连接至队列管理器。 请考虑哪种类型的连接最适合您的需求。

客户机绑定连接

要将 IBM MQ classes for .NET 用作 IBM MQ MQI client，您可以将其与 IBM MQ MQI client 一起安装在 IBM MQ 服务器上，或者安装在独立的机器上。 客户机绑定连接可以使用 XA 事务或非 XA 事务

服务器绑定连接

在服务器绑定方式下使用时，IBM MQ classes for .NET 使用队列管理器 API（而不是通过网络）进行通信。 这能够为 IBM MQ 应用程序提供比网络连接更好的性能。

要使用绑定连接，必须在 IBM MQ 服务器上安装 IBM MQ classes for .NET 。

受管客户机连接

以此方式建立的连接作为 IBM MQ 客户机连接到在本地或远程机器上运行的 IBM MQ 服务器。

以此方式连接的 IBM MQ classes for .NET 保留在 .NET 受管代码中，并且不调用任何本机服务。 有关受管代码的更多信息，请参阅 Microsoft 文档。

使用受管客户机时有若干限制。 有关这些限制的更多信息，请参阅 第 481 页的『受管客户机连接』。

Windows V 9.1.1 Linux 安装 IBM MQ classes for .NET Standard

从 IBM MQ 9.1.1 开始，IBM MQ classes for .NET Standard（包括样本）随 IBM MQ 一起安装在 Windows 上。 V 9.1.2 从 IBM MQ 9.1.2 开始， IBM MQ classes for .NET Standard 在 Linux 平台上也可用。 必备软件为 Microsoft.NET Core for IBM MQ classes for .NET Standard。

必备软件和安装

缺省情况下，最新版本的 IBM MQ classes for .NET Standard 会作为标准 IBM MQ 安装的一部分安装在 *Java and .NET Messaging and Web Services* 功能中。

在 IBM MQ 9.1.1 时，IBM MQ classes for .NET Standard 只可以在 Windows 上使用。

V 9.1.2 从 IBM MQ 9.1.2 开始，IBM MQ classes for .NET Standard 可以在 Linux 和 Windows 平台上使用。

要运行 IBM MQ classes for .NET Standard，必须安装 Microsoft .NET Core。

Windows 有关必备软件以及在 Windows 上进行安装的更多信息：

- 请参阅 IBM MQ classes for .NET 需求，以了解运行 IBM MQ classes for .NET Standard 的必备软件。
- 请参阅在 Windows 上安装 IBM MQ 服务器或在 Windows 系统上安装 IBM MQ 客户机，以获取安装指示信息。

Linux V 9.1.2 有关必备软件以及在 Linux 上进行安装的更多信息：

- 请参阅 IBM MQ classes for .NET 需求，以了解运行 IBM MQ classes for .NET Standard 的必备软件。
- 有关 rpm 安装指示信息，请参阅 在 Linux 系统上安装 IBM MQ 客户机。
- 对于 Linux Ubuntu，使用 Debian 软件包，请参阅 在 Linux 系统上安装 IBM MQ 客户机。

V 9.1.4 从 IBM MQ 9.1.4 开始，可从 NuGet 存储库下载 IBM MQ classes for .NET Standard 库 `amqmdnetstd.dll`。 有关更多信息，请参阅第 468 页的『从 NuGet 存储库下载 IBM MQ classes for .NET Standard』。

amqmdnetstd.dll 库

Windows 从 IBM MQ 9.1.1 开始， amqmdnetstd.dll 库可用于 Windows 上的 .NET Standard 支持。此外还提供了样本应用程序（包含源文件）；请参阅第 469 页的『.NET 的样本应用程序』。

Linux V 9.1.2 从 IBM MQ 9.1.2 中，Linux 上也提供了 amqmdnetstd.dll 库。 当 IBM MQ 客户机安装在 Linux 上时，该库将安装到 /&MQINSTALL_PATH&/lib64 path 中。 .NET 样本位于 &MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base 中。

根据 Microsoft .NET Standard 规范构建的任何库均可用于开发 .NET Framework 应用程序和 .NET Core 应用程序。

注意: 仍会提供用于 .NET Framework 的 amqmdnet.dll 库，但该库已稳定下来（即，不会引入任何新功能）。

对于任何最新功能，必须将其迁移到 amqmdnetstd.dll 库。 但是，您可以继续在 IBM MQ 9.1 Long Term Support 或 Continuous Delivery 发行版上使用 amqmdnet.dll 库。

dspmqver 命令

从 IBM MQ 9.1.1 开始，可以使用 **dspmqver** 命令来显示 .NET Core 组件的版本和构建信息。

IBM MQ classes for .NET Framework 和 IBM MQ classes for .NET Standard 功能

下表列出了 IBM MQ 9.1.1 for IBM MQ classes for .NET Framework 和 IBM MQ classes for .NET Standard 提供的适用功能部件。

表 *74: IBM MQ classes for .NET Framework* 和 *IBM MQ classes for .NET Standard* 功能部件之间的差异

功能部件	IBM MQ classes for .NET Framework	IBM MQ classes for .NET Standard
类名 (API)	所有类在每个网络中都保持不变。	所有类在每个网络中都保持不变。
操作系统	Windows	Windows Dockerized 容器 V 9.1.2 Linux V 9.1.3 MacOS
app.config 文件（用于在可再分发客户机中启用跟踪的配置文件）	app.config 文件用于为可再分发软件包启用跟踪	app.config 在 .NET Standard 中不受支持。 您必须使用环境变量来代替 app.config。
跟踪	**strmqtrc** 命令和（对于可再发行客户机）app.config 文件用于启用跟踪。	**strmqtrc** 命令。 要对可再发行软件包启用跟踪，必须使用环境变量。 对于 IBM MQ .NET，环境变量 **MQDOTNET_TRACE_ON** 用于对可再发行的客户机启用跟踪。 等于或小于 0 的值不会启用跟踪。 值 1 将启用缺省级别跟踪。 大于 1 的值将启用详细跟踪。 将此环境变量设置为任何其他值 (例如，字符串) 不会启用跟踪。
传输方式	受管、非受管和绑定	受管

表 74: *IBM MQ classes for .NET Framework* 和 *IBM MQ classes for .NET Standard* 功能部件之间的差异 (继续)

功能部件	IBM MQ classes for .NET Framework	IBM MQ classes for .NET Standard
TLS	Windows 密钥库用于存储证书。	Windows 在 Windows 上，必须使用密钥库来存储证书。 允许的值为 *USER 或 *SYSTEM。 根据输入，IBM MQ .NET 客户机将查看当前用户或系统范围内的 Windows 密钥库。 Linux V 9.1.2 在 Linux 上，建议使用 X509Store 类来安装证书，.NET Core 会将证书安装到以下位置：`".dotnet/corefx/cryptography/x509stores"`。
CCDT	支持	支持，并且 CCDT 路径的设置对与 .NET Framework 类的相同。
客户机自动重新连接	支持	支持
分布式事务	支持	不支持
将动态链接库 (dll's) 安装到全局组合件高速缓存 (GAC) 中	Dll's 将作为 IBM MQ 安装的一部分安装在 GAC 中。	Dll's 不作为 IBM MQ 安装的一部分安装在 GAC 中。

注: Windows Windows 安全标识 (SID):

IBM MQ .NET Standard 类不支持域级别认证。 将使用已登录的用户标识进行认证。

除了用于在 IBM MQ .NET Standard 上启用跟踪的环境变量 **MQDOTNET_TRACE_ON** 外，还可以将用于 IBM MQ classes for .NET Framework 的其他环境变量 (包括 **MQERRORPATH**， **MQLOGLEVEL** 和 **MQSERVER** 等) 用于 IBM MQ classes for .NET Standard 。 对于 IBM MQ classes for .NET Standard 和 IBM MQ classes for .NET Framework，这些变量的工作方式相同。

在 IBM MQ classes for .NET Standard 中， **MQDOTNET_TRACE_ON** 环境变量检查 IBM MQ 跟踪目录是否可用。 如果跟踪目录可用，那么将在跟踪目录中生成跟踪文件。 但是，如果未安装 IBM MQ ，那么会将跟踪文件复制到当前工作目录。

请参阅 第 508 页的『使用独立的 IBM MQ .NET 客户机』 ，以获取有关用于跟踪的变量的更多信息，包括 **MQTRACEPATH** 和 **MQTRACELEVEL**。

在 MacOS 上开发 IBM MQ .NET Core 应用程序

V 9.1.3

从 IBM MQ 9.1.3 开始，可以在 MacOS 上开发 IBM MQ .NET Core 应用程序。

IBM MQ .NET 库没有与 MacOS 工具箱打包在一起，因此必须将其从 Windows 或 Linux IBM MQ 客户机复制到 MacOS 上。 然后，可以使用这些库在 MacOS 上开发 IBM MQ .NET Core 应用程序。

在完成开发后，支持在 Windows 或 Linux 环境中运行这些应用程序。

相关概念

第 516 页的『使用 IBM MQ classes for XMS .NET Standard』
如何将 XMS 与 Microsoft .NET Standard 结合使用，以及使用 IBM MQ classes for XMS .NET Framework 和 IBM MQ classes for XMS .NET Standard 的差别。 必备软件为 Microsoft.NET Core for IBM MQ classes for XMS .NET Standard。

Windows V 9.1.4 Linux 从 *NuGet* 存储库下载 *IBM MQ classes for .NET Standard*

从 IBM MQ 9.1.4 开始，可以从 NuGet 存储库下载 IBM MQ classes for .NET Standard，以便 .NET 开发者可以轻松使用这些内容。

关于此任务

NuGet 是用于 Microsoft 开发平台（包括 .NET）的软件包管理器。可通过 NuGet 客户机工具来生成和使用软件包。NuGet 软件包是具有 .nupkg 扩展名的单个压缩文件，其中包含已编译的代码 (DLL)、与该代码相关的其他文件以及包含诸如包版本号之类的信息的描述性清单。

从 IBM MQ 9.1.4，您可以从 NuGet Gallery 下载包含 amqmdnetstd.dll 库的 IBMMQDotnetClient NuGet 软件包，这是所有软件包作者和使用者使用的中央软件包存储库。

可使用三种方式来下载 IBMMQDotnetClient 软件包：

- 使用 Microsoft Visual Studio。NuGet 作为 Microsoft Visual Studio 扩展分发。从 Microsoft Visual Studio 2012 开始，缺省情况下已预安装了 NuGet。
- 从命令行中使用 NuGet Package Manager 或 .NET CLI。
- 使用 Web 浏览器。

对于可再分发软件包，您可以使用环境变量 **MQDOTNET_TRACE_ON** 来启用跟踪。

过程

- 要使用 Microsoft Visual Studio 中的 Package Manager UI 来下载 IBMMQDotnetClient 软件包，请完成以下步骤：

  a) 右键单击 .NET 项目，然后单击**管理 Nuget 软件包**。

  b) 单击**浏览**选项卡，并搜索“IBMMQDotnetClient”。

  c) 选择软件包，然后单击**安装**。

  在安装期间，Package Manager 将以控制台语句形式提供进度信息。
- 要从命令行中下载 IBMMQDotnetClient 软件包，请选择以下某个选项：
  - 使用 NuGet Package Manager，输入以下命令：

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    在安装期间，Package Manager 将以控制台语句形式提供进度信息。您可以将输出重定向到某个日志文件。
  - 使用 .NET CLI，输入以下命令：

    ```
    dotnet add package IBMMQDotnetClient --version 9.1.4
    ```
- 使用 Web 浏览器从 https://www.nuget.org/packages/IBMMQDotnetClient 下载 IBMMQDotnetClient 软件包。

相关任务

第 518 页的『从 NuGet 存储库下载 IBM MQ classes for XMS .NET Standard』

从 IBM MQ 9.1.4 开始，可以从 NuGet 存储库下载 IBM MQ classes for XMS .NET Standard，以便 .NET 开发者可以轻松使用这些内容。

相关参考

IBM MQ Client for .NET 许可证信息

Windows 安装 IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework（包括样本）随 IBM MQ 一起安装。Windows 上存在 Microsoft.NET Framework 的先决条件。

缺省情况下，最新版本的 IBM MQ classes for .NET Framework 会作为标准 IBM MQ 安装的一部分安装在 *Java and .NET Messaging and Web Services* 功能中。 有关安装指示信息，请参阅 在 Windows 上安装 IBM MQ 服务器 或 在 Windows 系统上安装 IBM MQ 客户机。

V 9.1.0 从 IBM MQ 9.1 开始，要运行 IBM MQ classes for .NET Framework，您必须安装 Microsoft.NET Framework V4.5.1 或更高版本。

通过在应用程序的 `app.config` 文件中添加以下标记，可以运行使用 Microsoft.NET Framework V3.5 编译的现有应用程序，而无需重新编译：

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.1"/>
 </startup>
</configuration>
```

注： 如果在安装 IBM MQ 之前未安装 Microsoft .NET Framework V4.5.1 或更高版本，那么 IBM MQ 产品安装将继续进行而不会发生错误，但 IBM MQ classes for .NET 不可用。 如果在安装 IBM MQ 之后安装了 .NET Framework，那么必须通过运行 *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` 脚本来注册 IBM MQ.NET 组合件，其中 *WMQInstallDir* 是 IBM MQ 的安装目录。 该脚本将在全局程序集缓存 (GAC) 中安装必需的程序集。 将在 %TEMP% 目录中创建用于记录所执行操作的一组 `amqi*.log` 文件。 如果 .NET 已从较低版本（例如，从 .NET v3.5）升级到 v4.5.1 或更高版本，那么不必重新运行 `amqiRegisterdotNet.cmd` 脚本。

在多重安装环境中，如果先前已安装 IBM MQ classes for .NET 作为支持包，那么除非先卸载该支持包，否则无法安装 IBM MQ。 随 IBM MQ 一起安装的 IBM MQ classes for .NET 功能部件包含与支持包相同的功能。

此外还提供了样本应用程序（包含源文件）；请参阅第 469 页的『.NET 的样本应用程序』。

有关将 IBM MQ 定制通道用于具有 .NET 的 Microsoft WCF 的信息，请参阅 第 1141 页的『使用 IBM MQ 开发 Microsoft Windows Communication Foundation 应用程序』

相关概念

第 465 页的『安装 IBM MQ classes for .NET Standard』

从 IBM MQ 9.1.1 开始，IBM MQ classes for .NET Standard（包括样本）随 IBM MQ 一起安装在 Windows 上。 V 9.1.2 从 IBM MQ 9.1.2 开始， IBM MQ classes for .NET Standard 在 Linux 平台上也可用。 必备软件为 Microsoft.NET Core for IBM MQ classes for .NET Standard。

.NET 的样本应用程序

要运行您自己的 .NET 应用程序，请使用有关验证程序的指示信息，并将样本应用程序替换为您自己的应用程序。

提供了以下样本应用程序：

- put message 应用程序
- get message 应用程序
- “hello world”应用程序
- publish/subscribe 应用程序
- 使用消息属性的应用程序

所有这些样本应用程序都提供了 C# 语言版本，部分也提供了 C++ 和 Visual Basic 语言版本。 您可以使用 .NET 支持的任意语言编写应用程序。

“Put message”程序 SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)

该程序显示如何将消息放入指定的队列。 该程序具有以下三个参数：

- 队列名称（必需），例如 SYSTEM.DEFAULT.LOCAL.QUEUE
- 队列管理器名称（可选）
- 通道定义（可选），例如 SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

如果未提供队列管理器名称，那么队列管理器将缺省为缺省本地队列管理器。 如果定义了通道，那么它具有与 MQSERVER 环境变量相同的格式。

“Get message”程序 SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
该程序显示如何从指定队列中获取消息。 该程序具有以下三个参数：

- 队列名称（必需），例如 SYSTEM.DEFAULT.LOCAL.QUEUE
- 队列管理器名称（可选）
- 通道定义（可选），例如 SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

如果未提供队列管理器名称，那么队列管理器将缺省为缺省本地队列管理器。 如果定义了通道，那么它具有与 MQSERVER 环境变量相同的格式。

“Hello World”程序 (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
该程序显示如何放入和获取消息。 该程序具有以下三个参数：

- 队列名称（可选），例如 SYSTEM.DEFAULT.LOCAL.QUEUE 或 SYSTEM.DEFAULT.MODEL.QUEUE
- 队列管理器名称（可选）
- 通道定义（可选），例如 SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

如果未提供队列名称，那么名称将缺省为 SYSTEM.DEFAULT.LOCAL.QUEUE。 如果未提供队列管理器名称，那么队列管理器将缺省为缺省本地队列管理器。

“Publish/subscribe”程序 (MQPubSubSample.cs)
该程序显示如何使用 IBM MQ 发布/预订。 该程序仅提供 C# 版本。 该程序具有两个参数：

- 队列管理器名称（可选）
- 通道定义（可选）

“Message properties”程序 (MQMessagePropertiesSample.cs)
该程序显示如何使用消息属性。 该程序仅提供 C# 版本。 该程序具有两个参数：

- 队列管理器名称（可选）
- 通道定义（可选）

您可以通过编译和运行这些应用程序来验证您的安装。

安装位置

根据编写样本应用程序时使用的语言，样本应用程序安装在以下位置。 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

托管 C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

构建样本应用程序

为构建样本应用程序，为每种语言提供了一个批处理文件。

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

bldcssamp.bat 文件针对每个样本都包含一行内容，它是构建该样本程序所必需的：

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

托管 C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

bldmcpsamp.bat 文件针对每个样本都包含一行内容，它是构建该样本程序所必需的：

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

如果要在 Microsoft Visual Studio 2003/.NET SDKv1.1 上编译这些应用程序，请将编译命令：

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

替换为

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

bldvbsamp.bat 文件针对每个样本都包含一行内容，它是构建该样本程序所必需的：

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

将 IBM MQ 与 Microsoft .NET Core 一起使用的样本

V 9.1.1

从 IBM MQ 9.1.1 开始， IBM MQ 在 Windows 环境中支持 .NET Core for IBM MQ .NET 应用程序。 缺省情况下，IBM MQ classes for .NET Standard（包括样本）作为 IBM MQ 标准安装的一部分进行安装。

IBM MQ .NET 的样本应用程序安装在 &MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base 中。 脚本同样已提供，可用于编译样本。

您可以使用提供的 build.bat 文件来构建样本。 每个样本在 Windows 上的以下位置中都有一个 build.bat：

- MQ\tools\dotnet\samples\cs\core\base\SimpleGet
- MQ\tools\dotnet\samples\cs\core\base\SimplePut

Linux V 9.1.2 从 IBM MQ 9.1.2 开始， IBM MQ 还支持 .NET Core for applications in Linux 环境。

有关将 IBM MQ 与 Microsoft .NET Core 结合使用的更多信息，请参阅第 465 页的『安装 IBM MQ classes for .NET Standard』。

将队列管理器配置为接受 TCP/IP 客户机连接

将队列管理器配置为接受来自客户机的传入连接请求。

关于此任务

此任务说明配置队列管理器以接受 TCP/IP 客户机连接的基本步骤。 对于生产系统，在配置队列管理器时，还必须考虑安全隐患。

过程

1. 定义服务器连接通道：
   a. 启动队列管理器。
   b. 定义名为“NET.CHANNEL”的样本通道：

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   要点: 此样本仅适用于沙箱环境，因为它不包括任何安全隐患注意事项。 对于生产系统，请考虑使用 TLS 或安全出口。 请参阅保护 IBM MQ 以获取更多信息。
2. 启动侦听器：

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   注: 方括号指示可选参数；对于缺省队列管理器，*qmname* 不是必需的，如果您使用缺省值 (1414)，那么端口号 *portnum* 也不是必需的。

.NET 中的分布式事务

分布式事务或全局事务允许客户机应用程序在一个事务中包含两个或更多个联网系统上的不同数据源。

在分布式事务中，事务管理器协调和管理两个或更多个资源管理器间的事务。

事务可以是单阶段落实过程，也可以是两阶段落实过程。 单阶段落实是只有一个资源管理器参与事务的过程，而两阶段落实过程是一个以上资源管理器参与事务的过程。 在两阶段落实过程中，事务管理器会发送一个准备调用，以便检查所有的资源管理器是否都准备好落实。 当收到来自所有资源管理器的确认后，便会发出落实调用。 否则，会执行针对整个事务的回滚。 有关更多详细信息，请参阅事务管理和支持。 资源管理器应当就自身对事务的参与通知事务管理器。 在资源管理器就自身的参与通知事务管理器时，资源管理器会在事务将落实或回调时从事务管理器处获得回调。

IBM MQ .NET 类已在非管理和服务器绑定方式连接中支持分布式事务。 在这些方式中，IBM MQ .NET 类会将其所有调用委派给 C 扩展事务客户机，该客户机将代表 .NET 管理事务处理。

IBM MQ.NET 类现在受管方式下支持分布式事务，此时 IBM MQ .NET 类针对分布式事务支持使用 System.Transactions 名称空间。 System.Transactions 基础结构支持在所有资源管理器（包括 IBM MQ）中启动的事务，使事务编程变得简单高效。 IBM MQ .NET 应用程序可使用 .NET 隐式事务编程或显式事务编程模式放入和获取消息。 在隐式事务中，事务边界由决定何时落实、回滚（针对显式事务）或完成事务的应用程序来创建。 在显式事务中，您必须明确指定是否希望落实、回滚和完成事务。

IBM MQ.NET 使用 Microsoft 分布式事务协调程序 (MS DTC) 作为事务管理器，用于协调和管理多个资源管理器间的事务。 IBM MQ 用作资源管理器。 注意，不能将 TLS 与 XA 事务一起使用。 必须使用 CCDT。 有关更多信息，请参阅将扩展事务客户机与 TLS 通道结合使用。

IBM MQ.NET 遵循 X/Open 分布式事务处理 (DTP) 模型。 X/Open 分布式事务处理模型是由 Open Group（一个供应商联盟）提议的一种分布式事务处理模型。 该模型是事务处理和数据库领域中大部分商业供应商采用的标准。 大部分商业事务管理产品都支持 X/DTP 模型。

事务方式

- 第 473 页的『.NET 受管方式下的分布式事务』
- 非受管方式的分布式事务

协调各种场景中的事务

- 一条连接可能会参与多个事务，但在任何时间点都只有一个事务处于活动状态。
- 事务期间，采用 MQQueueManager.Disconnect 调用。 在这种情况下，会要求事务回滚。
- 事务期间，采用 MQQueue.Close 或 MQTopic.Close 调用。 在这种情况下，会要求事务回滚。
- 事务边界由决定何时落实、回滚（针对显式事务）或完成（针对隐式事务）事务的应用程序来创建。
- 如果在针对队列或主题调用发出 Put 或 Get 调用之前，客户机应用程序在事务期间中断并返回意外错误，那么将回滚事务并抛出 MQException。
- 如果针对队列或主题调用发出 Put 或 Get 调用期间返回 MQCC_FAILED 原因码，那么将与原因码一起抛出 MQException，并且回滚事务。 如果事务管理器已发出准备调用，那么 IBM MQ .NET 将通过强制回滚事务来返回准备请求。 之后，事务管理器 DTC 会使当前环境事务中的所有资源管理器回滚当前工作。
- 在涉及多个资源管理器的事务期间，如果由于某些环境原因而导致 Put 或 Get 调用无限期挂起，那么事务管理器将一直等待到规定的时间。 在超时后，它会使当前环境事务中的所有资源管理器回滚当前的所有工作。 如果在准备阶段发生无限制等待，那么事务管理器可能会超时，或针对资源发出不确定的调用（在此情况下将回滚事务）。
- 使用这些事务的应用程序必须在 SYNC_POINT 下对消息进行 Put 或 Get 操作。 如果在不是 SYNC_POINT 下的事务上下文下发出了消息 Put 或 Get 调用，那么该调用操作将失败并显示 MQRC_UNIT_OF_WORK_NOT_STARTED 原因码。

使用 Microsoft.NET System.Transactions 名称空间的受管和非受管客户机事务支持之间的行为差异

嵌套事务在一个 TransactionScope 内具有另一个 TransactionScope

- IBM MQ .NET 完全受管客户机不支持嵌套 TransactionScope
- IBM MQ .NET 非受管客户机不支持嵌套 TransactionScope

System.Transactions 中的从属事务

- IBM MQ .NET 完全受管客户机不支持 System.Transactions 提供的从属事务设施。
- IBM MQ .NET 非受管客户机不支持 System.Transactions 提供的从属事务设施。

产品样本

`WebSphere MQ\tools\dotnet\samples\cs\base` 下提供了新的产品样本 `SimpleXAPut` 和 `SimpleXAGet`。 这些样本是 C# 应用程序，演示在使用 SystemTransactions 名称空间的分布式事务下使用 MQPUT 和 MQGET。 有关这些样本的更多信息，请参阅第 476 页的『在 TransactionScope 中创建简单的 put 和 get 消息』

.NET 受管方式下的分布式事务

IBM MQ .NET 类针对受管方式下的分布式事务支持使用 System.Transactions 名称空间。 在受管方式中，MS DTC 协调和管理在事务中登记的所有服务器上的分布式事务。

IBM MQ .NET 类提供一个基于 System.Transactions.Transaction 类的显式编程模型和一个使用 System.Transactions.TransactionScope 类（其中的事务由基础结构自动管理）的隐式编程模型。

隐式事务

以下代码片段描述 IBM MQ .NET 应用程序如何使用 .NET 隐式事务编程放入消息。

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
```

```
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

隐式事务代码流说明

此代码创建了 *TransactionScope* 并将消息放在了该作用域下。 其随后调用 *Complete*，以向事务协调程序通知事务已完成的消息。 此时，事务协调程序发出 *prepare* 和 *commit*，用于完成事务。 如果检测到任何问题，那么将调用 *rollback*。

显式事务

以下代码描述 IBM MQ .NET 应用程序如何使用 .NET 显式事务编程模型放入消息。

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

显式事务代码流说明

该代码片段使用 *CommitableTransaction* 类创建事务。 它将消息放到了该区域下，然后显式调用 *commit* 以完成该事务。 如果出现任何问题，那么将调用 *rollback*。

.NET 非受管方式下的分布式事务

IBM MQ.NET 类使用扩展事务客户机和 COM+/MTS 作为事务协调程序，使用隐式或显式事务编程模型来支持非受管连接（客户机）。 在非受管方式下，IBM MQ .NET 类会将其所有调用委派给 C 扩展事务客户机，该客户机将代表 .NET 管理所有事务处理。

事务处理由外部事务管理器控制，该管理器在事务管理器的 API 控制下协调全局工作单元。 MQBEGIN、MQCMIT 和 MQBACK 动词均不可用。IBM MQ .NET 类通过其非受管传输方式（C 客户机）来公开此支持。 请参阅配置 XA 兼容的事务管理器

MTS 已演变为事务处理 (TP) 系统，在 Windows NT（在 CICS、Tuxedo 和其他平台上可用）上提供相同功能。 安装 MTS 后，会将单独的服务添加到名为 Microsoft 分布式事务协调程序 (MSDTC) 的 Windows NT。 MSDTC 将协调跨不同数据存储器或资源的事务。 工作时，它要求每个数据存储都实施其自己的专有资源管理器。

IBM MQ 通过实现接口 (专有资源管理器接口) 来与 MSDTC 兼容，在该接口中，它管理将 DTC XA 调用映射到 IBM MQ(X/Open) 调用。 IBM MQ 充当资源管理器的角色。

当诸如 COM+ 等组件请求访问 IBM MQ 时，如果需要事务，COM 通常会检查相应的 MTS 上下文对象。 如果需要事务，COM 会通知 DTC 并针对该操作自动启动一个完整的 IBM MQ 事务。 之后，COM 会通过 MQMTS 软件处理数据，并根据需要放入或获取消息。 从 COM 获取的对象实例将在所有对数据的操作都结束后调用 SetComplete 或 SetAbort 方法。 当应用程序发出 SetComplete 时，调用会示意 DTC 应用程序已完成了事务，且 DTC 可以进行两阶段落实过程。 接着，DTC 会向 MQMTS 发出调用，MQMTS 又向 IBM MQ 发出调用以落实或回滚事务。

使用非受管客户机编写 IBM MQ .NET 应用程序

要在 COM+ 上下文中运行，必须从 System .EnterpriseServices.ServicedComponent 继承 .NET 类。 在创建使用服务组件的组合件时，请遵循下列规则和建议：

注: 仅当使用 System.EnterpriseServices 模式时，以下步骤才有意义：

- 在 COM+ 中启动的类和方法必须是公共的（非内部类，不受保护并且也不是静态方法）。
- 类和方法属性：TransactionOption 属性指示类的事务级别，即该事务是已禁用、受支持还是必需的。 ExecuteUOW() 方法的 AutoComplete 属性指示 COM+ 在没有抛出任何未处理的异常的情况下落实该事务。
- 强命名组合件：组合件必须在全局组合件高速缓存 (GAC) 中已强命名且注册。 组合件已在 COM+ 中显式注册，或者已在 GAC 中注册后通过延迟注册方式注册。
- 在 COM+ 中注册组合件：准备向 COM 客户机公开的组合件。 然后，通过使用组合件注册工具 regasm.exe 创建类型库。

```
regasm UnmanagedToManagedXa.dll
```

- 在 GAC `gacutil /i UnmanagedToManagedXa.dll` 中注册该组合件。
- 通过使用 .NET 服务安装程序工具 regsvcs.exe 在 COM+ 中注册组合件。 请参阅 regasm.exe 创建的类型库：

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- 组合件将部署到 GAC 中，稍后将通过延迟注册在 COM+ 中注册该组合件。 .NET 框架会在首次运行代码后执行注册。

下列部分中描述了将 System.EnterpriseServices 模型和 System.Transactions 与 COM+ 配合使用的代码流示例：

使用 System.EnterpriseServices 模型的代码流示例

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
```

```
   xa.ExecuteUOW();
}
```

使用 System.Transactions 与 COM+ 进行交互的代码流示例

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

在 *TransactionScope* 中创建简单的 *put* 和 *get* 消息

IBM MQ 中提供了产品样本 C# 应用程序。 这些简单的应用程序演示在 TransactionScope 中放入和获取消息。 完成任务时，您也将能够在队列或主题中放入或获取消息。

开始之前

MSDTC 服务必须正在运行，且已针对 XA 事务启用。

关于此任务

示例是简单的应用程序 SimpleXAPut 和 SimpleXAGet。 程序 SimpleXAPut 和 SimpleXAGet 是 C# 应用程序，可以从 IBM MQ 中获取。 SimpleXAPut 演示在使用 SystemTransactions 名称空间的分布式事务下使用 MQPUT。 SimpleXAGet 演示在使用 SystemTransactions 名称空间的分布式事务下使用 MQGET。

SimpleXAPut 位于 MQ\tools\dotnet\samples\cs\base 中

过程

可以使用命令行参数从 tools\dotnet\samples\cs\base\bin 中运行这些应用程序

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

其中，参数为：

-destinationURI

可以是队列或主题。 对于队列，指定为 queue://queueName；对于主题，指定为 topic://topicName。

-host

这可以是主机名，如 localhost 或 IP 地址。

-port

正在运行队列管理器的端口。

-channel

使用的连接通道。 缺省值为 SYSTEM.DEF.SVRCONN

-transaction

事务结果，例如落实或回滚。

-mode

传输方式，例如受管或非受管。

-numberOfMsgs

消息数目。 缺省值是 1。

示例

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

在 *IBM MQ .NET* 中恢复事务

本部分描述使用受管方式在 IBM MQ .NET XA 中恢复事务的过程。

概述

在分布式事务处理中，可以成功完成事务。 但是，在有些场景中，也会由于种种理由而导致事务失败。 这些原因可能包括系统故障、硬件故障、网络错误、数据错误或无效、应用程序错误或自然或人为灾害。 防止事务失败是不可能的。 分布式事务系统必须有能力处理这些失败情况。 它必须能够在错误出现时检测并纠正这些错误。 该过程称为“事务恢复”。

分布式事务处理的一个重要方面是恢复未完成和不确定的事务。 由于特定事务的工作单元部分会在恢复前一直保持锁定状态，因此运行恢复是非常必要的。 Microsoft.NET 通过其 System.Transactions 类库提供用于恢复未完成/不确定事务的选项。 此恢复支持期望资源管理器来维护事务日志并在需要时运行恢复。

恢复模式

在 Microsoft .NET 事务恢复模型中，事务管理器（System.Transactions 和/或 Microsoft 分布式事务协调程序 (MS DTC)）负责启动、协调和控制事务恢复。 基于资源管理器的 OLE Tx 协议（Microsoft XA 协议）提供选项，用于配置 DTC 以推动、协调和控制事务的恢复。 要完成此操作，资源管理器必须通过使用本机接口向 MS DTC 注册 XA_Switch。

XA_Switch 在资源管理器中向分布式事务协调程序提供 XA 功能（如 xa_start、xa_end 和 xa_recover）的入口点。

使用 Microsoft 分布式事务协调程序 (DTC) 的恢复：

Microsoft 分布式事务协调程序提供两种类型的恢复过程。

冷恢复

如果与 XA 资源管理器的连接打开，此时事务管理器进程失败，那么将执行冷恢复。 事务管理器重新启动时，它将读取事务管理器日志并重新建立到 XA 资源管理器的连接，然后再启动恢复。

热恢复

如果在由于 XA 资源管理器或网络故障而导致事务管理器和 XA 资源管理器之间的连接失败期间，事务管理器仍保持运行状态，那么将执行热恢复。 连接失败后，事务管理器会周期性尝试重新连接到 XA 资源管理器。 重新建立连接后，事务管理器会启动 XA 恢复。

System.Transactions 名称空间为以作为事务管理器的 MS DTC 为基础的分布式事务提供受管实施。 它提供与 MS DTC 的本机接口类似的功能，但处于完全受管的环境中。 唯一的区别是事务恢复。 System.Transactions 期望资源管理器自己推动恢复，然后与事务管理器 (MS DTC) 协调。 资源管理器必须要求恢复特定的未完成事务，然后事务管理器接受该请求并根据特定事务的实际结果进行协调。

IBM MQ .NET 的事务恢复过程
本部分描述了如何使用 IBM MQ .NET 类来恢复分布式事务。

概述

要恢复未完成的事务，恢复信息是必需的。 资源管理器必须将事务恢复信息记录到存储器中。IBM MQ .NET 类遵循类似的路径。 事务恢复信息被记录在名为 SYSTEM.DOTNET.XARECOVERY.QUEUE 的系统队列中。

IBM MQ .NET 中的事务恢复是一个两阶段过程。

1. 记录事务恢复信息。
   - 对于每个事务，都会在准备阶段将包含恢复信息的持久消息添加到 SYSTEM.DOTNET.XARECOVERY.QUEUE 中。
   - 如果成功落实调用，那么将删除该消息。
2. 使用监视器应用程序 WmqDotnetXAMonitor 恢复事务。
   - WmqDotnetXAMonitor 是 .NET 管理的应用程序，用于处理 SYSTEM.DOTNET.XARECOVERY.QUEUE 并恢复不完整的事务

   如果 MCA 无法将消息放入到目标队列，那么它将生成一个包含原始消息的异常报告，并将该报告放入传输队列中，此传输队列随后发送至原始消息中指定的应答队列。 （如果应答队列与 MCA 在相同的队列管理器上，那么该消息将直接放入该队列，而不是传输队列。）

SYSTEM.DOTNET.XARECOVERY.QUEUE

这是一个系统队列，用于保持未完成事务的事务恢复消息。 该队列是在创建队列管理器时创建的。

注: 您不应当删除 SYSTEM.DOTNET.XARECOVERY.QUEUE 队列。

WMQDotnetXAMonitor 应用程序

IBM MQ .NET XA 监视器应用程序负责监视给定的队列管理器和恢复未完成的事务（如果有）。 以下项将被视为未完成的事务并将恢复：

未完成的事务

- 如果准备了事务，但在超时时间段内未完成 COMMIT。
- 如果准备了事务，但 IBM MQ 队列管理器已关闭。
- 如果准备了事务，但事务管理器已关闭。

必须在运行 IBM MQ .NET 客户机应用程序的系统上运行监视器应用程序。 如果在连接到同一队列管理器的多个系统上运行应用程序，那么必须在所有这些系统上运行监视器应用程序。 虽然每个客户机都运行监视器应用程序以恢复应用程序，但是每个监视器都应该能够识别与当前监视器的本地 MS DTC 协调的事务相对应的消息，以使其可以重新列出并完成事务。

IBM MQ .NET 的事务恢复用例
可能需要恢复事务的用例有多种。

- **IBM MQ 应用程序使用单个 DTC 和单个队列管理器实例：** 在此用例中，当连接到队列管理器并在事务下运行工作单元 (UoW) 时，如果事务失败并变得不完整，那么监视器应用程序将恢复事务并完成该事务。

在此用例中，由于单个队列管理器与该事务关联，因此将运行监视器应用程序的单个实例。

- **多个 IBM MQ 应用程序使用单个 DTC 和单个队列管理器实例：** 在此用例中，单个 DTC 下存在多个 WMQ 应用程序并且它们都连接到同一个队列管理器并在事务下运行 UoW。

  如果事务失败并且变得不完整，那么监视器应用程序将恢复它们并完成与所有应用程序相关的事务。

  在此用例中，由于事务中使用一个队列管理器，因此只有一个监视器应用程序运行。

- **多个 IBM MQ 应用程序、多个 DTC、不同队列管理器实例：** 在此用例中，多个 WMQ 应用程序位于不同 DTC 下（即，每个应用程序运行在不同的机器上）并且连接到不同的队列管理器。

  如果发生故障并且事务变得不完整，那么监视器应用程序会检查消息中的 TransactionManagerWhereabouts 以确定 DTC 地址。 如果 TransactionManagerWhereabouts 值与正在其下运行监视器的 DTC 地址匹配，那么该监视器会完成恢复，否则它会继续搜索，直到找到与其 DTC 对应的消息。

  在此用例中，由于每个客户机都在事务中使用自己的队列管理器，因此每个客户机（用户或计算机）只运行监视器应用程序的一个实例。

- **多个 IBM MQ 应用程序、多个 DTC、多个相同的队列管理器实例：** 在此用例中，多个 WMQ 应用程序位于不同 DTC 下（即，每个应用程序运行在不同的机器上）并且全部连接到同一个队列管理器。

  如果发生故障并且事务变得不完整，那么监视器应用程序会验证消息中的 TransactionManagerWhereabouts 以检查 DTC 地址和值是否与用于运行监视器的 DTC 匹配。 如果两个值都匹配，那么它将完成恢复，否则会继续搜索，直到找到与其 DTC 对应的消息。

  在此用例中，由于每个客户机都在事务中使用自己的队列管理器关联，因此每个客户机（用户或计算机）只有一个监视器应用程序实例运行。

- **多个 IBM MQ 应用程序、单个 DTC、不同队列管理器实例：** 在此用例中，多个 WMQ 应用程序位于单个 DTC 下（即在一个计算机上，有多个 WMQ 应用程序运行）并且连接到不同的队列管理器。

  如果事务失败并变得不完整，那么监视器应用程序将恢复该事务。

  在此用例中，由于每个应用程序都在事务中使用自己的队列管理器并且每个都必须恢复，因此连接了多少个队列管理器，就需要运行多少个监视器应用程序实例。

  注: 如果监视器应用程序未在后台运行，您可以将其启动。

使用 *WMQDotnetXAMonitor* 应用程序
必须手动运行 WMQDotnetXAMonitor 应用程序。 可以随时将其启动。 当您在 SYSTEM.DOTNET.XARECOVERY.QUEUE 或您可以在对使用 IBM MQ .NET 类编写的应用程序执行任何事务性工作之前保持其在后台运行。

使用以下命令启动监视器应用程序

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
```

其中：

- **-m *QueueManagerName***
  队列管理器名称。

  可选

  -n *ConnectionName*
  采用 host (port) 格式的连接名称。 Connection Name 可以包含多个连接名称。 多个连接名称必须以逗号分隔列表形式提供，例如 `localhost (1414), localhost (1415), localhost (1416)`。 监视器应用程序针对逗号分隔列表中指定的每个连接名称运行恢复。

  -c *ChannelName*
  通道名称。

  -h
  启发式分支完成。

  可选

监视器应用程序执行以下操作：

1. 每隔 100 秒检查一次 SYSTEM.DOTNET.XARECOVERY.QUEUE 的队列深度。
2. 如果队列深度大于零，那么 XA 监视器将浏览队列以获取消息并检查该消息是否满足未完成事务的条件。
3. 如果任何消息满足未完成事务的标准，那么监视器会将其拉出并检索事务恢复信息。
4. 之后，将确定恢复信息是否与本地 MS DTC 相关。 如果相关，那么将继续恢复该事务。 否则，将返回以浏览下一条消息。
5. 之后将调用队列管理器以恢复未完成的事务。

WmqDotNETXAMonitor 应用程序配置文件设置
要监视应用程序，也可以使用应用程序配置文件来提供输入。 IBM MQ .NET 随附了一个样本应用程序配置文件。 您可以根据自己的需求来修改该文件。

在考虑输入值时，应用程序配置文件具有最高优先顺序。 如果在命令行和应用程序配置文件中同时提供了输入值，那么将考虑使用应用程序配置文件中的值。

样本应用程序配置文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

WmqDotNetXAMonitor 应用程序日志
监视器应用程序会在应用程序目录中创建一个日志文件，用于记录监视器的进度和事务恢复状态。 日志记录以连接名称和通道详细信息开始，显示对其运行恢复的当前队列管理器。

恢复一旦启动，便会记录事务恢复消息的 MessageId、未完成事务的 TransactionId 以及按照事务管理器协调的实际事务结果。

样本日志文件：

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

编写和部署 IBM MQ .NET 程序

要使用 IBM MQ classes for .NET 访问 IBM MQ 队列，需要使用 .NET 支持的任一语言编写程序，并在程序中包含向 IBM MQ 队列放入消息以及从中获取消息的调用。

IBM MQ 文档仅包含有关 C#、C++ 和 Visual Basic 语言的参考信息。

本主题集提供的信息可帮助您编写与 IBM MQ 系统进行交互的应用程序。 有关个别类的详细信息，请参阅 IBM MQ .NET 类和接口。

连接差异

您为 IBM MQ.NET 编程的方式一定程度上依赖于要使用的连接方式。

当 IBM MQ classes for .NET 用作受管客户机时，由于部分功能对受管客户机不可用，因此它与标准 IBM MQ MQI client 之间存在一些差异。

IBM MQ.NET 通过您为连接名称、通道名称、定制值 NMQ_MQ_LIB 和属性 MQC.TRANSPORT_PROPERTY 指定的设置确定要使用的连接类型。

受管客户机连接

当 IBM MQ classes for .NET 用作受管客户机时，与标准 IBM MQ MQI client 之间将存在一些差异。

以下功能对受管客户机不可用：

- 通道压缩
- 通道出口链

如果尝试将这些功能用于受管客户机，那么将返回 MQException。 如果在连接的客户机端检测到该错误，那么将使用原因码 MQRC_ENVIRONMENT_ERROR。 如果在服务器端检测到该错误，那么将使用服务器返回的原因码。

为非受管客户机编写的通道出口没有作用。 您必须专门为受管客户机编写新的出口。 请检查您的客户机通道定义表 (CCDT)，确定未指定无效的通道出口。

受管通道出口的名称长度最多为 999 个字符。 但是，如果使用 CCDT 来指定通道出口名称，那么名称将限制为 128 个字符。

仅支持通过 TCP/IP 进行通信。

使用 **endmqm** 命令停止队列管理器时，到 .NET 受管客户机的服务器连接通道的关闭时间可能比到其他客户机的服务器连接通道的时间长。

如果已将 *NMQ_MQ_LIB* 设置为 `managed` 以便使用受管 IBM MQ 问题诊断，那么不支持 **strmqtrc** 命令的任何参数 -i， -p， -s， -b 或 -c。

使用 XA 事务的受管 .NET 应用程序将不能使用 z/OS 队列管理器。 尝试连接到 z/OS 队列管理器的受管 .NET 客户机在 MQOPEN 调用上失败，发生错误 MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354)。 但是，使用 XA 事务的受管 .NET 应用程序可使用分布式队列管理器。

定义要使用的连接类型

通过连接名称、通道名称、定制值 NMQ_MQ_LIB 和属性 MQC.TRANSPORT_PROPERTY 的设置可确定连接类型。

您可以按如下所示指定连接名称：

- 在 MQQueueManager 构造函数上明确指定：

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- 通过设置 MQQueueManager 构造函数的散列表条目中的属性 MQC.HOST_NAME_PROPERTY 和（可选）MQC.PORT_PROPERTY：

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 作为明确的 MQEnvironment 值

```
MQEnvironment.Hostname
```

`MQEnvironment.Port`（可选）。

- 通过设置 MQEnvironment.properties 散列表中的属性 MQC.HOST_NAME_PROPERTY 和（可选）MQC.PORT_PROPERTY。

您可以按如下所示指定通道名称：

- 在 MQQueueManager 构造函数上明确指定：

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- 通过设置 MQQueueManager 构造函数的散列表条目中的属性 MQC.CHANNEL_PROPERTY：

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 作为显式 MQEnvironment 值

```
MQEnvironment.Channel
```

- 通过设置 MQEnvironment.properties 散列表中的属性 MQC.CHANNEL_PROPERTY。

您可以按如下指定传输属性：

- 通过设置 MQQueueManager 构造函数的散列表条目中的属性 MQC.TRANSPORT_PROPERTY：

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 通过设置 MQEnvironment.properties 散列表中的属性 MQC.TRANSPORT_PROPERTY。

通过使用以下值之一选择所需的连接类型：

MQC.TRANSPORT_MQSERIES_BINDINGS - 作为服务器连接
MQC.TRANSPORT_MQSERIES_CLIENT - 作为非 XA 客户机连接
MQC.TRANSPORT_MQSERIES_XACLIENT - 作为 XA 客户机连接
MQC.TRANSPORT_MQSERIES_MANAGED - 作为非 XA 受管客户机连接

您可以设置定制值 NMQ_MQ_LIB 以如下表所示显式选择连接类型。

NMQ_MQ_LIB 值	**连接类型**
mqic.dll	作为非 XA 客户机连接
mqicxa.dll	作为 XA 客户机连接
mqm.dll	作为服务器或非 XA 客户机连接
受管	作为非 XA 受管客户机连接

注: 接受作为 mqic.dll 和 mqicxa.dll 同义词的 mqic32.dll 和 mqic32xa.dll 的值，以与早期版本兼容。 但是，mqm.dll 和 mqm.pdb 仅是 IBM WebSphere MQ 7.1 之前的客户机软件包的一部分。

如果选择了在您的环境中不可用的连接类型，例如您指定了 mqic32xa.dll 并且环境中不支持 XA，那么 IBM MQ.NET 将抛出异常。

将 NMQ_MQ_LIB 设置为“managed”会导致客户机使用受管 IBM MQ 问题诊断测试、.NET 数据转换和其他受管的低级 IBM MQ 功能。

NMQ_MQ_LIB 的所有其他值都会使 .NET 进程使用非受管的 IBM MQ 问题诊断测试和数据转换，以及其他非受管的低级 IBM MQ 功能（假定系统上安装了 IBM MQ MQI client 或服务器）。

IBM MQ.NET 按如下所示选择连接类型：

1. 如果指定了 MQC.TRANSPORT_PROPERTY，那么它将根据 MQC.TRANSPORT_PROPERTY 的值进行连接。

   但请注意，将 MQC.TRANSPORT_PROPERTY 设置为 MQC.TRANSPORT_MQSERIES_MANAGED 并不保证客户机进程以受管方式运行。 即使使用此设置，在以下情况下，客户机也不会以受管方式运行：

   - 如果进程中的另一个线程在连接时将 MQC.TRANSPORT_PROPERTY 设置为 MQC.TRANSPORT_MQSERIES_MANAGED 以外的内容。
   - 如果 NMQ_MQ_LIB 未设置为“managed”，那么问题诊断测试、数据转换和其他低级功能将不会完全受管（假定系统中安装了 IBM MQ MQI client 或服务器）。
2. 如果指定了连接名称但未指定通道名称，或者指定了通道名称但未指定连接名称，那么将抛出错误。
3. 如果已指定了连接名称和通道名称：
   - 如果 NMQ_MQ_LIB 设置为 mqic32xa.dll，那么将作为 XA 客户机连接。
   - 如果 NMQ_MQ_LIB 设置为 managed，那么将作为受管客户机连接。
   - 否则将作为非 XA 客户机连接。
4. 如果指定了 NMQ_MQ_LIB，那么将根据 NMQ_MQ_LIB 的值进行连接。
5. 如果安装了 IBM MQ 服务器，那么将作为服务器连接。
6. 如果安装了 IBM MQ MQI client，那么将作为非 XA 客户机连接。
7. 否则，将作为受管客户机连接。

Windows V 9.1.5 使用 IBM MQ .NET 项目模板

从 IBM MQ 9.1.5 开始，IBM MQ .NET 客户机支持您借助项目模板开发 .NET Core 应用程序。

开始之前

您的系统上必须具有 Microsoft Visual Studio 2017 或更高版本以及 .NET Core 2.1。

您必须将 .NET 模板从

```
&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip
```

目录复制到

```
&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates
```

目录，其中：

- *&MQ_INSTALL_ROOT* 是安装的根目录
- *&USER_HOME_DIRECTORY* 是主目录。

您必须停止并重新启动 Microsoft Visual Studio 以选取模板。

关于此任务

.NET 项目模板包含一些可用于帮助您开发应用程序的通用代码。 通过内置代码，您可以连接到 IBM MQ 队列管理器，并且只需修改内置代码中的属性即可执行放置或获取操作。

过程

1. 打开 Microsoft Visual Studio。

2. 单击**文件**，接着单击**新建**和**项目**。
3. 在 " 创建新项目 " 窗口中，选择 IBM MQ .NET Client App（.NET Core），然后单击 **下一步**。
4. 在配置您的新项目窗口中，更改项目的项目名称（如果需要），然后单击**创建**以创建 .NET 项目。

   MQDotnetApp.cs 是随项目文件一起创建的文件。 该文件包含连接到队列管理器并执行放置和获取操作的代码。

   连接属性将设置为缺省值：

   - MQC.CONNECTION_NAME_PROPERTY 将设置为 *localhost(1414)*
   - MQC.CHANNEL_PROPERTY 将设置为 *DOTNET.SVRCONN*

   队列将设置为 *Q1*，您可以相应地修改这些属性。
5. 编译并运行应用程序。

相关概念

IBM MQ 组件和功能

相关参考

.NET 应用程序运行时（仅限 Windows）

IBM MQ classes for .NET 的配置文件

.NET 客户机应用程序可以使用 IBM MQ MQI client 配置文件，如果您使用的是受管连接类型，那么可以使用 .NET 应用程序配置文件。 应用程序配置文件中的设置具有优先权。

客户机配置文件

IBM MQ classes for .NET 客户机应用程序可以像其他任何 IBM MQ MQI client 一样使用客户机配置文件。 此文件通常称为 mqclient.ini，但您可以指定不同的文件名。 有关客户机配置文件的更多信息，请参阅使用配置文件配置客户机。

IBM MQ MQI client 配置文件中只有以下属性与 IBM MQ classes for .NET 相关。 如果指定其他属性，那么将没有任何效果。

表 *75: 与 IBM MQ classes for .NET* 相关的客户机配置文件属性

节	属性
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ReconDelay
CHANNELS	DefRecon
CHANNELS	MQReconnectTimeout
CHANNELS	ServerConnectionParms
CHANNELS	Put1DefaultAlwaysSync
CHANNELS	PasswordProtection
ClientExitPath	ExitsDefaultPath
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage

表 75: 与 IBM MQ classes for .NET 相关的客户机配置文件属性 (继续)	
节	**属性**
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

您可以使用相应的环境变量覆盖任何这些属性。

应用程序配置文件

如果使用受管连接类型运行，那么还可以使用 .NET 应用程序配置文件覆盖 IBM MQ 客户机配置文件和等效环境变量。

仅当使用受管连接类型运行时，.NET 应用程序配置文件设置才会生效，而对于其他连接类型，这些设置将被忽略。

.NET 应用程序配置文件及其格式由 Microsoft 定义，通常用于 .NET 框架，但本文档中提及的特定节名称、关键字和值都特定于 IBM MQ。

.NET 应用程序配置文件的格式涵盖多个节。 每节都包含一个或多个关键字，每个关键字都具有关联的值。以下示例显示 .NET 应用程序配置文件中用于控制 TCP/IP KeepAlive 属性的节、关键字和值：

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

.NET 应用程序配置文件节名称和键中使用的关键字与客户机配置文件中定义的“节”和“属性”的关键字完全匹配。

节 <configSections> 必须是 <configuration> 元素的第一个子元素。

请参阅 Microsoft 文档以获取更多信息。

可与 .NET 一起使用的 C# 代码片段示例

一个 C# 代码片段，演示应用程序连接到队列管理器、将消息放入队列并接收回复。

以下 C# 代码片段演示一个执行三个操作的应用程序：

1. 连接到队列管理器
2. 将消息放入 SYSTEM.DEFAULT.LOCAL.QUEUE
3. 获取消息回复

它还显示如何更改连接类型。

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
```

```
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define an IBM MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                           // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again
```

```
      // First define an IBM MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

设置 IBM MQ 环境

使用客户机连接来连接到队列管理器之前，必须设置 IBM MQ 环境。

注: 以服务器绑定方式使用 IBM MQ classes for .NET 时，不需要执行本步骤。

.NET 编程接口允许您使用 NMQ_MQ_LIB 定制值，但也包含类 MQEnvironment。 该类允许您指定将在连接尝试期间使用的详细信息，如以下列表中的各项:

- 通道名称
- 主机名
- 端口号
- 通道出口
- SSL 参数
- 用户标识和密码

有关 MQEnvironment 类的完整信息，请参阅 MQEnvironment.NET 类

要指定通道名称和主机名，请使用以下代码:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

缺省情况下，客户机会尝试通过端口 1414 连接到 IBM MQ 侦听器。 要指定其他端口，请使用代码:

```
MQEnvironment.Port = nnnn;
```

连接到队列管理器和从队列管理器断开连接

配置 IBM MQ 环境后，即可准备连接到队列管理器。

要连接到队列管理器，请创建 MQQueueManager 类的新实例：

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

要断开与队列管理器的连接，请在队列管理器上调用 `Disconnect` 方法：

```
queueManager.Disconnect();
```

尝试连接到队列管理器时，您必须具有队列管理器的查询 (`inq`) 权限。 如果没有查询权限，那么连接尝试将失败。

如果调用 `Disconnect` 方法，那么将关闭通过该队列管理器访问的所有打开的队列和进程。 但是，良好的编程实践是在用完这些资源后将其明确关闭。 要关闭资源，请使用与每个资源关联的对象上的 `Close` 方法。

队列管理器上的 `Commit` 和 `Backout` 方法将取代用于程序接口的 MQCMIT 和 MQBACK 调用。

访问队列和主题

使用 MQQueueManager 或相应构造函数的方法，可以访问队列和主题。

要访问队列，可使用 MQQueueManager 类的方法。 MQOD（对象描述符结构）折叠在这些方法的参数中。 例如，要在称为 queueManager 的 MQQueueManager 对象表示的队列管理器上打开队列，请使用以下代码：

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

选项参数与 MQOPEN 调用中的选项参数相同。

AccessQueue 方法将返回 MQQueue 类的新对象。

使用完队列后，请使用 Close() 方法将其关闭，如以下示例中所示：

```
queue.Close();
```

使用 IBM MQ .NET，您还可以通过 MQQueue 构造函数创建队列。 参数与 accessQueue 方法的完全相同，只是添加了用于指定要使用的实例化 MQQueueManager 对象的队列管理器参数。 例如：

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

以这种方式构建队列对象时，您可以编写自己的 MQQueue 子类。

同样，您也能使用 MQQueueManager 类的方法访问主题。 使用 AccessTopic() 方法打开主题。 这会返回 MQTopic 类的新对象。 使用完主题后，请使用 MQTopic 的 Close() 方法将其关闭。

您也可以使用 MQTopic 构造函数创建主题。 用于主题的构造函数有许多；有关更多信息，请参阅 MQTopic.NET 类。

处理消息

消息是使用队列或主题类的方法进行处理的。 要处理新的消息，需创建新的 MQMessageobject。

使用 MQQueue 或 MQTopic 类的 Put() 方法将消息放入队列或主题。 使用 MQQueue 或 MQTopic 类的 Get() 方法从队列或主题中获取消息。 与使用 MQPUT 和 MQGET 放入或获取字节数组的程序接口不同，IBM MQ classes for .NET 放入和获取 MQMessage 类的实例。 MQMessage 类会将包含实际消息数据的数据缓冲区和描述该消息的所有 MQMD（消息描述符）参数封装在一起。

要构建新的消息，需创建新的 MQMessage 类实例并使用 WriteXXX 方法将数据放入消息缓冲区。

创建新的消息实例后，会自动将所有 MQMD 参数设置为其缺省值，如 MQMD 的初始值和语言声明中所定义。 MQQueue 的 Put() 方法还会将 MQPutMessageOptions 类的实例作为参数。 该类表示 MQPMO 结构。 以下示例会创建一条消息并将其放入队列：

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage,pmo);
```

MQQueue 的 Get() 方法会返回一个新的 MQMessage 实例，表示刚从队列中获取该消息。 它还会获取一个 MQGetMessageOptions 类实例作为参数。 该类表示 MQGMO 结构。

您无需指定最大消息大小，因为 get() 方法会自动调整其内部缓冲区的大小以容纳入局消息。 使用 MQMessage 类的 ReadXXX 方法可访问返回的消息中的数据。

以下示例显示如何从队列中获取消息：

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

通过设置 *encoding* 成员变量，您可以更改读、写方法使用的数字格式。

通过设置 *characterSet* 成员变量，您可以更改用于读、写字符串的字符集。

有关更多详细信息，请参阅 MQMessage.NET 类。

注: MQMessage 的 WriteUTF() 方法会自动为字符串的长度以及其包含的 Unicode 字节编码。 当您的消息被另一个 .NET 程序（使用 readUTF()）读取时，这是发送字符串信息的最简单方法。

处理消息属性

消息属性使您在无需访问其标头的情况下选择消息或检索有关消息的信息。 MQMessage 类包含用于获取和设置属性的方法。

通过使用消息属性，可以使应用程序选择要处理的消息，或在无需访问 MQMD 或 MQRFH2 标头的情况下检索有关消息的信息。 这些属性也便于 IBM MQ 和 JMS 应用程序之间的通信。 有关 IBM MQ 中的消息属性的更多信息，请参阅消息属性。

MQMessage 类根据属性的数据类型，提供了用于获取和设置属性的大量方法。 get 方法名称格式为 Get*Property，set 方法名称格式为 Set*Property，其中星号 (*) 代表以下字符串之一：

- 布尔
- Byte

- 字节
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Long
- Object
- 短
- 字符串

例如，要获取 IBM MQ 属性 myproperty（一个字符串），请使用调用 `message.GetStringProperty('myproperty')`。 您可以选择传递 IBM MQ 将要完成的属性描述符。

处理错误

处理由于 IBM MQ classes for .NET 使用 `try` 和 `catch` 块而产生的错误。

.NET 接口中的方法不会返回完成代码和原因码。 而是每当 IBM MQ 调用产生的完成代码和原因码都不为零时抛出异常。 这样简化了程序逻辑，您不必在每次调用 IBM MQ 之后都检查返回码。 您可以决定希望在程序中的哪些位置处理可能出现的故障。 您可以在这些位置使用 `try` 和 `catch` 块将代码包围起来，如下例中所示：

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                        "CC = " + ex.CompletionCode +
                        "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

获取和设置属性值

类 MQManagedObject、MQQueue 和 MQQueueManager 包含使您能够获取和设置其属性值的方法。 请注意，仅当在打开队列时指定了适当的 inquire 和 set 标志时，MQQueue 的这些方法才起作用。

对于公共属性，MQQueueManager 和 MQQueue 类从名为“MQManagedObject”的类继承。 此类定义了 Inquire() 和 Set() 接口。

使用 *new* 操作符创建新队列管理器对象时，它将自动打开以供查询。 在使用 AccessQueue() 方法访问队列对象时，该对象不会自动打开来进行查询或设置操作，这可能导致某些类型的远程队列出现问题。 要使用 Inquire 和 Set 方法并在队列上设置属性，您必须在 AccessQueue() 方法的 openOptions 参数中指定合适的 inquire 和 set 标志。

查询和设置方法接受三种参数：

- selectors 数组
- intAttrs 数组
- charAttrs 数组

您不需要在 MQINQ 中找到的 SelectorCount、IntAttrCount 和 CharAttrLength 参数，因为数组的长度都是已知的。 以下示例显示了如何查询队列：

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

可以查询该队列上所有对象的属性。 属性的子集作为对象的属性公开。 有关对象属性的列表，请参阅对象的属性。 有关对象属性，请参阅相应的类描述。

多线程程序

.NET 运行时环境是固有的多线程环境。 IBM MQ classes for .NET 允许在多个线程之间共享队列管理器对象，但要确保对目标队列管理器的所有访问都是同步的。

考虑一个能连接到队列管理器并在启动时可以打开队列的简单程序。 该程序在屏幕上显示一个按钮。 用户单击该按钮时，程序将从队列中访存消息。 在这种情况下，会在一个线程上发生应用程序初始化，且用于响应按钮按下执行的代码在单独的线程（用户界面线程）中执行。

实施 IBM MQ .NET 可以确保对于特定连接（MQQueueManager 对象实例），对目标 IBM MQ 队列管理器的所有访问都是同步的。 缺省行为是，会阻止想要向队列管理器发出调用的线程，直到针对该连接正在进行的所有其他调用都已完成。 如果您需要在应用程序中通过多个线程同时访问同一个队列管理器，请针对需要并行访问的每个线程创建新的 MQQueueManager 对象。 （这等同于对每个线程发出一个单独的 MQCONN 调用。）

如果缺省线程选项被 MQC.MQCNO_HANDLE_SHARE_NONE 或 MQC.MQCNO_SHARE_NO_BLOCK 覆盖，那么队列管理器将不再同步。

将客户机通道定义表用于 .NET

您可以将客户机通道定义表 (CCDT) 与 IBM MQ 的 .NET 类配合使用。 可以通过不同的方式指定 CCDT 的位置，具体取决于您是使用受管连接还是非受管连接。

非 XA 或 XA 非受管客户机连接类型

对于非受管连接类型，您可以通过两种方式指定 CCDT 的位置：

- 使用环境变量 MQCHLLIB 指定表所在的目录，并使用 MQCHLTAB 指定表的文件名。
- 使用客户机配置文件。 在 CHANNELS 节中，使用属性 ChannelDefinitionDirectory 指定表所在的目录，并使用 ChannelDefinitionFile 指定文件名。

如果都通过客户机配置文件和使用环境变量指定位置，那么环境变量优先。 您可以使用此功能在客户机配置文件中指定标准位置，并在必要时使用环境变量覆盖该位置。

受管客户机连接类型

对于受管连接类型，您可以通过三种方式指定 CCDT 的位置：

- 使用 .NET 应用程序配置文件。 在 CHANNELS 部分中，使用关键字 ChannelDefinitionDirectory 指定表所在的目录，并使用 ChannelDefinitionFile 指定文件名称。
- 使用环境变量 MQCHLLIB 指定表所在的目录，并使用 MQCHLTAB 指定表的文件名。
- 使用客户机配置文件。 在 CHANNELS 节中，使用属性 ChannelDefinitionDirectory 指定表所在的目录，并使用 ChannelDefinitionFile 指定文件名。

如果使用多种方式指定位置，那么环境变量将优先于客户机配置文件，且 .NET 应用程序配置文件将优先于其他方法。 您可以使用此功能在客户机配置文件中指定标准位置，并在必要时使用环境变量或应用程序配置文件覆盖该位置。

.NET 应用程序如何确定要使用的通道定义。

在 IBM MQ .NET 客户机应用环境中，可通过许多不同的方式指定要使用的通道定义。 可存在多个用于指定通道定义的规范。 应用程序从一个或多个源中获取通道定义。

如果存在多个通道定义，则按以下优先级顺序选择所使用的通道定义：

1. 在 MQQueueManager 构造器中指定的属性（显式指定），或在属性散列表中包含 *MQC.CHANNEL_PROPERTY*。
2. MQEnvironment.properties 散列表中的 *MQC.CHANNEL_PROPERTY* 属性
3. MQEnvironment 中的 *Channel* 属性
4. .NET 应用程序配置文件、段名称 CHANNELS 以及关键字 ServerConnectionParms（仅适用于受管连接）
5. *MQSERVER* 环境变量
6. 客户机配置文件、节 CHANNELS 以及 Attribute ServerConnectionParms
7. 客户机通道定义表 (CCDT)。 CCDT 的位置是在 .NET 应用程序配置文件中指定（仅适用受管连接）
8. 客户机通道定义表 (CCDT)。 使用环境变量 *MQCHLIB* 和 *MQCHLTAB* 指定 CCDT 的位置
9. 客户机通道定义表 (CCDT)。 CCDT 的位置是使用客户机配置文件指定

对 1-3 项，通过应用程序提供的值，逐个字段地构建通道定义。 可通过使用不同的接口提供这些值，并且每个接口可存在多个值。 按以下所给的优先级顺序，将字段值添加到通道定义：

1. MQQueueManager 构造函数的 *connName* 值
2. MQQueueManager.properties 散列表中属性的值
3. MQEnvironment.properties 散列表中属性的值
4. 值设置为 MQEnvironment 字段（例如 MQEnvironment.Hostname 和 MQEnvironment.Port）

对于 4-6 项，将整个通道定作为值提供。 通道定义上未指定的字段会采用系统缺省值。 来自定义通道及其字段的其他方法的值没有与这些指定内容合并。

对于 7-9 项，整个通道定义取自 CCDT。 在定义通道时未明确指定的字段将采用系统缺省值。 来自定义通道及其字段的其他方法的值没有与这些指定内容合并。

在 IBM MQ .NET 中使用通道出口

如果使用客户机绑定，则可以像使用任何其他客户机连接一样使用通道出口。 如果使用受管绑定，那么必须编写实现适当接口的出口程序。

客户机绑定

如果使用客户机绑定，那么也可使用通道出口描述的通道出口。 您不能使用为受管绑定编写的通道定义。

受管绑定

如果使用受管连接来实现出口，那么需要定义可以实现相应接口的新 .NET 类。 IBM MQ 包中定义了三个出口接口：

- MQSendExit
- MQReceiveExit
- MQSecurityExit

注: 不支持将使用这些接口编写的用户出口作为非受管环境中的通道出口。

以下样本定义了一个类，它实现了所有三个接口：

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // complete the body of the send exit here
  }


  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // complete the body of the receive exit here
  }


  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefParms,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
  {
     // complete the body of the security exit here
  }

}
```

每个出口都传递了一个 MQChannelExit 和一个 MQChannelDefinition 对象实例。 这些对象表示在过程接口中定义的 MQCXP 和 MQCD 结构。

使用出口参数指定要通过发送出口发送的数据以及在安全和接收出口接收的数据。

在入口处，字节数组 *dataBuffer* 中的偏移量 *dataOffset* 上长度为 *dataLength* 的数据将通过发送出口发送，并在安全和接收出口中接收数据。 参数 *dataMaxLength* 给定了 *dataBuffer* 中出口可用的最大长度（根据 *dataOffset*）。 请注意：对于安全出口，如果是第一次调用该出口，或合作伙伴最终选择不发送任何数据，那么该出口可能为空。

在返回时，*dataOffset* 和 *dataLength* 的值应设置为指向 .NET 类之后应该使用的返回字节数组中的偏移量和长度 对于发送出口，这指示它应该发送的数据，以及对于安全或接收出口，这指示应该解释的数据。 出口通常应当返回一个字节数组；异常的安全出口可以选择不发送数据，以及因为 INIT 或 TERM 原因调用的任何出口。 因此，最简单的出口形式就是返回 dataBuffer：

最简单的出口代码是：

```
{
  return dataBuffer;
}
```

MQChannelDefinition 类

使用受管 .NET 客户机应用程序指定的用户标识和密码在传递到客户机安全出口的 IBM MQ .NET MQChannelDefinition 类中进行了设置。 安全出口将用户标识和密码复制到 MQCD.RemoteUserIdentifier 和 MQCD.RemotePassword 字段中（请参阅第 886 页的『编写安全出口』）。

指定通道出口（受管客户机）

如果在创建 MQQueueManager 对象时指定通道名称或连接名称（以 MQEnvironment 或 MQQueueManager 构造函数），那么您可通过两种方式指定通道出口。

按照优先顺序，依次为：

1. 在 MQQueueManager 构造函数中传递散列表属性 MQC.SECURITY_EXIT_PROPERTY、MQC.SEND_EXIT_PROPERTY 或 MQC.RECEIVE_EXIT_PROPERTY。
2. 设置 MQEnvironment SecurityExit、SendExit 或 ReceiveExit 属性。

如果您未指定通道名称和连接名称，那么要使用的通道出口来自从客户机通道定义表 (CCDT) 中选取的通道定义。 不能覆盖在通道定义中存储的值。 请参阅客户机通道定义表和第 491 页的『将客户机通道定义表用于 .NET』以获取有关通道定义表的更多详细信息。

在每种情况下，规范都要采用具有以下格式的字符串形式：

```
Assembly_name(Class_name)
```

Class_name 是实现 IBM.WMQ.MQSecurityExit、IBM.WMQ.MQSendExit 或 IBM.WMQ.MQReceiveExit 接口（视情况而定）的 .NET 类的标准名称，包括空间名称规范。 *Assembly_name* 是容纳该类的组合件的完全限定位置，包括文件扩展名。 如果使用 MQEnvironment 或 MQQueueManager 的属性，那么字符串的长度将限制为 999 个字符。 但是，如果在 CCDT 中指定通道出口名称，那么字符串长度则限制为 128 个字符。 如果需要，.NET 客户机代码会通过解析字符串规范装入和创建指定类的实例。

指定通道出口用户数据（受管客户机）

通道出口可以具有与其相关联的用户数据。 如果在创建 MQQueueManager 对象时指定通道名称和连接名称（在 MQEnvironment 或 MQQueueManager 构造函数中），则可以通过两种方式指定用户数据。

按照优先顺序，依次为：

1. 在 MQQueueManager 构造函数中传递散列表属性 MQC.SECURITY_USERDATA_PROPERTY、MQC.SEND_USERDATA_PROPERTY 和 MQC.RECEIVE_USERDATA_PROPERTY。
2. 设置 MQEnvironment SecurityUserData、SendUserData 或 ReceiveUserData 属性。

如果您未指定通道名称和连接名称，那么要使用的出口用户数据值来自从客户机通道定义表 (CCDT) 中选取的通道定义。 不能覆盖在通道定义中存储的值。 请参阅客户机通道定义表和第 491 页的『将客户机通道定义表用于 .NET』以获取有关通道定义表的更多详细信息。

在所有情况下，规范是一个字符串，限制为 32 个字符。

.NET 中自动客户机重接

您可以在连接意外断开期间使客户机自动重新连接到队列管理器。

在某些情况下客户机可能意外与队列管理器断开连接，例如，在队列管理器停止或网络或服务器出现故障的情况下。

如果没有自动重新连接客户机，则连接失败时会产生错误。 您可以使用错误代码帮助重新建立连接。

使用自动客户机连接功能的客户机称为可重新连接的客户机。 要创建可重新连接的客户机，请在连接到队列管理器时指定名为 reconnect 选项的特定选项。

如果客户机应用程序是 IBM MQ .NET 客户机，那么可以在使用 MQQueueManager 类创建队列管理器时，可通过为 CONNECT_OPTIONS_PROPERTY 指定合适的值来选择自动化应用程序重新连接。 请参阅重新连接选项以获取 CONNECT_OPTIONS_PROPERTY 值的详细信息。

您可以选择总是连接客户机应用程序以及重新连接到具有相同名称的队列管理器或同一队列管理器，或连接到使用客户机连接表中同一 QMNAME 定义的任何队列管理器设置（请参阅 CCDT 的队列管理器组以获取详细信息）。

针对 .NET 的传输层安全性 (TLS) 支持

IBM MQ classes for .NET 客户机应用程序支持传输层安全性 (TLS) 加密。 TLS 协议通过因特网提供通信安全性，并且允许客户机/服务器应用程序以是保密且可靠的方式通信。

相关概念

IBM MQ.NET 受管客户机 TLS 支持

加密安全性协议：TLS

TLS 支持非受管 *.NET* 客户机

TLS 支持非受管 .NET 客户机是基于 C MQI 和 GSKit。 C MQI 处理 TLS 操作，GSKit 实现 TLS 安全套接字协议。

对非受管 *.NET* 客户机启用 *TLS*

仅客户机连接支持 TLS。 要启用 TLS，必须指定在与队列管理器通信时要使用的 CipherSpec，此 CipherSpec 必须与目标通道上设置的 CipherSpec 相匹配。

要启用 TLS，请使用 MQEnvironment 的 SSLCipherSpec 静态成员变量指定 CipherSpec。 以下示例连接到了名称为 SECURE.SVRCONN.CHANNEL 的 SVRCONN 通道，该通道已设置为要求 TLS 具有 TLS_RSA_WITH_AES_128_CBC_SHA 的 CipherSpec：

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

请参阅指定 CipherSpecs 获取 CipherSpec 的列表。

也可使用连接属性散列表中的 MQC.SSL_CIPHER_SPEC_PROPERTY 设置 SSLCipherSpec 属性。

要使用 TLS 成功连接，客户机密钥库必须设置有“认证中心”根证书链，从中可以认证队列管理器提供的证书。 同样，如果 SVRCONN 通道上的 SSLClientAuth 已设置为 SSL_CLIENT AUTH_REQUIRED，那么客户机密钥库必须包含队列管理器信任的标识个人证书。

使用队列管理器的专有名称

队列管理器使用包含专有名称 (DN) 的 TLS 证书来标识自身。

IBM MQ .NET 客户机应用程序可以使用该 DN 以确保自身与正确的队列管理器通信。 使用 MQEnvironment 的 sslPeerName 变量指定 DN 模式。 例如，设置：

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

仅当队列管理器提供公共名称以 QMGR. 开头的证书时，才允许连接成功。 和至少两个组织单元名称，其中第一个必须是 IBM 和第二个 WEBSPHERE。

还可以使用连接属性散列表中的 MQC.SSL_PEER_NAME_PROPERTY 来设置 SSLPeerName 属性。 有关用于设置对等名称的专有名称和规则的更多信息，请参阅 保护 IBM MQ。

如果设置了 SSLPeerName，只有将其设置为有效模式且队列管理器提供匹配的证书时，连接才会成功。

使用 *TLS* 时的错误处理

在使用 TLS 连接到队列管理器时，可通过 IBM MQ classes for .NET 发出以下原因码：

MQRC_SSL_NOT_ALLOWED
已设置 SSLCipherSpec 属性，但使用了绑定连接。 仅客户机连接支持 TLS。

MQRC_SSL_PEER_NAME_MISMATCH
在 sslPeerName 属性中指定的 DN 模式与队列管理器所显示的 DN 不匹配。

MQRC_SSL_PEER_NAME_ERROR
SSLPeerName 属性中指定的 DN 模式无效。

针对受管 *.NET* 客户机的 *TLS* 支持

受管 .NET 客户机使用 Microsoft.NET Framework 库实施 TLS 安全套接字协议。 Microsoft System.Net.SecuritySslStream 类将通过连接的 TCP 套接字作为流运行，并且会通过该套接字连接发送和接收数据。

最低必需 .NET Framework 级别为 .NET Framework v3.5。 密码算法支持的级别基于应用程序所使用的 .NET Framework 级别：

- 对于基于 .NET Framework 级别 3.5 和 4.0 的应用程序，可用的安全套接字协议为 SSL 3.0 和 TSL 1.0。

- 对于基于 .NET Framework 级别 4.5 的应用程序，可用的安全套接字协议为 SSL 3.0、TLS 1.1 和 TLS 1.2。

您可能需要将期望更高 TLS 协议支持的应用程序移至为 .NET 框架中的 Microsoft 安全性支持定义的更高版本的框架。

受管 .NET 客户机的 TLS 支持的主要特性如下所示：

TLS 协议支持
通过 .NET SSLStream 类定义针对 .NET 受管客户机的 TLS 支持，并且取决于应用程序所使用的 .NET Framework。 有关更多信息，请参阅第 496 页的『针对受管 .NET 客户机的 TLS 协议支持』。

CipherSpec 支持
.NET 受管客户机的 TLS 设置针对 Microsoft.NET TLS 流。 有关更多信息，请参阅第 497 页的『针对受管 .NET 客户机的 CipherSpec 支持』和第 498 页的『受管 .NET 客户机的 CipherSpec 映射』。

密钥存储库
客户机端的密钥存储库是 Windows 密钥库。 服务器端存储库是加密消息语法 (CMS) 类型的存储库。 有关更多信息，请参阅第 499 页的『针对受管 .NET 客户机的密钥存储库』。

证书
您可以使用自签名 TLS 证书在客户机与队列管理器之间实施相互认证。 有关更多信息，请参阅第 499 页的『对受管 .NET 客户机使用证书』。

SSLPEERNAME
在 .NET 中，应用程序可以使用可选的 SSLPEERNAME 属性来指定专有名称 (DN) 模式。 有关更多信息，请参阅第 500 页的『SSLPEERNAME』。

FIPS 合规性
Microsoft.NET 安全库不支持以编程方式启用 FIPS。 通过 Windows 组策略设置来控制 FIPS 启用。

NSA Suite B 合规性
IBM MQ 实施 RFC 6460。 针对 NSA Suite B 的 Microsoft.NET 实现为 5430。 从 .NET Framework 3.5 起受支持。

密钥重置或重新协商
虽然 SSLStream 类不支持密钥重置或重新协商，但为了确保一致性，对于其他 IBM MQ 客户机，.NET 受管客户机允许应用程序设置 SSLKeyResetCount。 有关更多信息，请参阅第 501 页的『受管 .NET 客户机的密钥重置或重新协商』。

撤销检查
SSLStream 类支持证书撤销检查，将通过证书链锁引擎自动执行证书撤销检查。 有关更多信息，请参阅第 501 页的『撤销检查』。

IBM MQ 安全出口支持
SSLStream 类提供对 IBM MQ 安全出口的有限支持。 查询本地证书和远程证书以获取 SSLPeerNamePtr（主题 DN）和 SSLRemCertIssNamePtr（颁发者 DN）是可能的，因为这在 Microsoft.NET 中受支持。 但是，不支持获取诸如 DNQ、UNSTRUCTUREDNAME 和 UNSTRUCTUREDADDRESS 之类属性，因此无法使用出口检索这些值。

加密硬件支持
针对受管 .NET 客户机，不支持加密硬件。

针对受管 *.NET* 客户机的 *TLS* 协议支持

IBM MQ.NET TLS 支持基于 .NET SSLStream 类。

注: 针对受管 .NET 客户机的 TLS 协议支持取决于应用程序所使用的 .NET Framework 级别。 有关更多信息，请参阅 第 495 页的『针对受管 .NET 客户机的 TLS 支持』。

要使 Microsoft.NET SSLStream 类初始化 TLS 并与队列管理器执行握手，您必须设置的一个必需参数为 **SSLProtocol**，您必须在其中指定 TLS 版本号，必须为以下值之一：

- SSL3.0
- TLS1.0
- TLS1.2

该参数的值与首选的 CipherSpec 所属于的协议系列紧密耦合。 当 SSLStream 开始与服务器（队列管理器）开始 TLS 握手时，将使用 **SSLProtocol** 中指定的 TLS 版本来标识要用于协商的 CipherSpec 的列表。

IBM MQ.NET 不会使任何属性供应用程序用于设置该值。 而是，IBM MQ 会使用映射表在内部将 CipherSpec 集映射到协议系列，并标识要使用的 SSLProtocol 版本。 该表显示了 Microsoft.NET 与 IBM MQ 之间的每个受支持 CipherSpec 的映射，以及所属于的协议版本。 有关更多信息，请参阅 第 498 页的『受管 .NET 客户机的 CipherSpec 映射』。

针对受管 *.NET* 客户机的 *CipherSpec* 支持
在与服务器握手期间使用应用程序的 CipherSpec 设置。

IBM MQ 客户机允许您设置在与队列管理器握手期间使用的 CipherSpec 值。 IBM MQ 客户机应该设置有效的 CipherSpec 以建立安全连接，首选 Windows 组策略中指定的 CipherSpec。 将该字段留空表示没有任何套接字安全性的明文通道。

对于 IBM MQ.NET 受管客户机，TLS 设置针对 Microsoft.NET SSLStream 类。 对于 SSLStream，只可以在 Windows 组策略（一种计算机范围内的设置）中设置 CipherSpec 或 CipherSpec 的首选项列表。 然后，SSLStream 会在与服务器握手期间使用指定的 CipherSpec 或首选项列表。 对于其他 IBM MQ 客户机，可以在 IBM MQ 通道定义上的应用程序中设置 CipherSpec 属性，并且相同的设置将用于 TLS 协商。 由于此限制，TLS 握手可能会与任何受支持的 CipherSpec 协商，而不管 IBM MQ 通道配置中指定的内容。 因此，很可能会导致队列管理器上出现错误 AMQ9631。 为避免此错误，请将与您在应用程序中设置的 CipherSpec 相同的 CipherSpec 设置为 Windows 组策略中的 TLS 配置。

新的 IBM MQ.NET TLS 客户机代码只会检查是否协商正确协议版本。 TLS 协议版本可从应用程序设置的 CipherSpec 中获取，并且用于与服务器（队列管理器）进行 TLS 握手。 因此，设计上需要在 IBM MQ.NET 受管客户机应用程序中设置 CipherSpec。 如果 IBM MQ 客户机设置的 CipherSpec 不是 SSL 3.0、TLS 1.0 和 TLS 1.2 协议所提供的 CipherSpec，那么 IBM MQ 受管 .NET 客户机缺省情况下将与 SSL 3.0 或 TLS 1.0 协议提供的任何密码进行协商，并且不会报告错误。

注: 如果应用程序提供的 CipherSpec 值不是 IBM MQ 可识别的 CipherSpec，那么 IBM MQ 受管 .NET 客户机将忽略该值，并且会根据 Windows 系统的组策略来协商连接。

设置 CipherSpec

可以使用三种方式设置 CipherSpec：

MQEnvironment .NET 类
以下示例显示如何使用 MQEnvironment 类设置 CipherSpec。

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

TLS CipherSpec 属性
以下示例显示如何通过将 hashtable 参数添加到 MQQueueManager 构造函数中来设置 CipherSpec。

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows 组策略
在 Windows 组策略上设置 CipherSpec 时，必须为 SVRCONN 通道和应用程序中的 SSLCipherSpec 属性值设置相同的 CipherSpec。 如果将 Windows 组策略设置为缺省组策略（即没有为 CipherSpec 设置启用/编辑的组策略），那么应用程序必须在 MQEnvironment 类或 MQQueueManager 构造函数 hashtable 属性中通过 Windows 组策略 TLS 配置来设置相同的 CipherSpec 缺省值。

CCDT 用法

IBM MQ.NET 仅支持位于本地计算机上的客户机通道定义表（.TAB 文件）。 设置了 CipherSpec 值的现有 CCDT 文件可用于 IBM MQ.NET 连接。 但是，在客户机连接通道上设置的 CipherSpec 值将确定 TLS 协议版本，并且还必须与 Windows 组策略中设置的 CipherSpec 匹配。

相关概念

第 487 页的『设置 IBM MQ 环境』
使用客户机连接来连接到队列管理器之前，必须设置 IBM MQ 环境。

相关任务

指定 CipherSpec

相关参考

MQEnvironment .NET 类

受管 *.NET* 客户机的 *CipherSpec* 映射

IBM MQ.NET 接口维护 IBM MQ 到 Microsoft.NET 映射表，用于确定受管客户机建立与队列管理器的安全连接时需要使用的 TLS 协议版本。

如果在 SVRCONN 通道上指定了 CipherSpec，那么在 TLS 握手完成后，队列管理器将尝试将该 CipherSpec 与客户机应用程序正在使用的协商 CipherSpec 相匹配。 如果队列管理器找不到匹配的 CipherSpec，那么通信将失败，报告错误 AMQ9631。

IBM MQ.NET 接口维护 IBM MQ 到 Microsoft.NET CipherSpec 映射表。 该表用于确定客户机用于建立与队列管理器的安全套接字连接的 TLS 协议版本。 基于 SSLCipherSpec 值，SSLProtocol 版本可能是 TLS 1.0 或 TLS 1.2，这取决于您正在使用的 Microsoft.NET Framework 的版本。

请确保提供正确的 SSLCipherSpec 值，因为指定错误的值可能会导致使用 SSL 3.0 或 TLS 1.0 协议。

表 *76: IBM MQ* 和 *Microsoft.NET* 映射表

IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS 版本**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
V 9.1.0.6 ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
V 9.1.0.6 ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2

表 76: *IBM MQ* 和 *Microsoft.NET* 映射表 *(*继续*)*		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS 版本**
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2

注意:

1. 不推荐此 CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。

针对受管 *.NET* 客户机的密钥存储库
受管 .NET 客户机使用的密钥存储库是 Windows 密钥库。 证书和专用密钥必须在用户或系统密钥库中可用，客户机应用程序才能在 TLS 握手期间将其用于身份和信任。

客户机端

IBM MQ.NET 中 TLS 配置的客户机端包含客户机端密钥存储库、客户机证书以及应用程序所做的选择。

- 客户机端密钥存储库始终是 Windows 密钥库。 可以是证书可存储在其下的用户或计算机帐户。
- 在应用程序中，您可以为密钥存储库设置以下任一值:
  - “*USER”：IBM MQ.NET 访问当前用户的证书库以检索客户机证书。
  - “*SYSTEM”：IBM MQ.NET 访问本地计算机帐户以检索证书。
- 客户机的证书必须存储在“用户”或“计算机”帐户的“我的证书库”中。 所有服务器 (CA) 证书必须存储在证书库的根目录中。

注: 您可以使用以下格式将多个证书存储在单个文件中:

- 个人信息交换 - PKCS #12（.PFX、.P12）
- 加密消息语法标准 - PKCS #7 证书 (.P7B)
- Microsoft 序列化证书库 (.SST)

对受管 *.NET* 客户机使用证书
对于客户机证书，IBM MQ 管理的 .NET 客户机将访问 Windows 密钥库并装入按证书标签或按字符串匹配的所有客户机证书。

在选择要使用的证书时，IBM MQ 受管 .NET 客户机始终使用第一个匹配证书进行 SSLStream TLS 握手。

按证书标签匹配证书

如果已设置证书标签，那么 IBM MQ managed .NET 客户机将搜索具有给定标签名称的 Windows 证书库以标识客户机证书。 将装入所有匹配证书并使用列表上的第一个证书。 可以使用两个选项来设置证书标签：

- 可以在访问 MQEnvironment.CertificateLabel 的 MQEnvironment 类上设置证书标签。
- 也可以在作为 MQQueueManager 构造函数的输入参数提供的散列表属性中设置证书标签，如以下示例所示。

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

name("CertificateLabel") 和值区分大小写。

按字符串匹配证书

如果未设置证书标签，那么将搜索并使用与字符串 "ibmwebspheremq" 和当前登录的用户（小写）匹配的证书。

相关任务

安全地将客户机连接到队列管理器

相关参考

MQEnvironment .NET 类

SSLPEERNAME

SSLPEERNAME 属性用于从对等队列管理器检查证书的专有名称 (DN)。

在 IBM MQ.NET 中，应用程序可以使用 SSLPEERNAME 指定专有名称模式，如以下示例所示。

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

对于其他 IBM MQ 客户机，SSLPEERNAME 是可选参数。

如果未设置 SSLPEERNAME 值，那么 IBM MQ.NET 受管客户机不会执行任何 Remote(Server) 证书验证，并且受管客户机只按现状接受 Remote(/server) 证书。

您设置 SSLPEERNAME 的方式取决于所使用的 IBM MQ 堆栈产品。

IBM MQ classes for .NET

有如下三个选项。

1. 在 MQEnvironment 类中设置 MQEnvironment.SSLPeerName。
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. 使用队列管理器构造函数 `MQQueueManager (String queueManagerName, Hashtable properties)`。 对于选项 2，在 `Hashtable properties` 中提供 SSLPEERNAME。

XMS .NET

在连接工厂中设置 SSL 对等名称：

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF

在 URI 中包含 SslPeerName 作为分号分隔的字段。

相关参考

MQEnvironment .NET 类

受管 *.NET* 客户机的密钥重置或重新协商
SSLStream 类不支持密钥重置/重新协商。 但是，为了与其他 IBM MQ 客户机保持一致，IBM MQ 受管 .NET 客户机允许应用程序设置 SSLKeyResetCount。

在达到限制后，IBM MQ.NET 将从队列管理器断开连接，并且会将其作为异常通知应用程序，并显示 MQRC_CONNECTION_BROKEN 作为原因码。 应用程序可以选择处理异常并重新建立连接，或对 IBM MQ.NET 启用 MQCNO_RECONNECT 选项以自动重新连接到队列管理器。

启用自动客户机重新连接功能意味着，在达到密钥重置计数时，所有现有连接都会断开并且 IBM MQ.NET 客户机会重新创建所有连接。 有关自动客户机重新连接的更多信息，请参阅自动客户机重新连接。

相关概念
重置 SSL 和 TLS 密钥

撤销检查
SSLStream 类支持证书撤销检查。

将通过证书链锁引擎自动执行证书撤销检查。 这将适用于联机证书状态协议 (OCSP) 和证书撤销列表 (CRL)。 SSLStream 类使用证书撤销，证书撤销仅使用证书中指定的服务器，该服务器是证书本身指定的服务器。 HTTP CDP 扩展和 OCSP HTTP 请求可以通过 HTTP 代理服务器进行代理。

设置撤销检查的方法取决于您所使用的 IBM MQ 堆栈产品。

IBM MQ.NET
可以通过访问 `MQEnvironment.cs` 类文件上的 **MQEnvironment.SSLCertRevocationCheck** 属性来设置撤销检查。

XMS .NET
可以在连接工厂属性上下文上设置撤销检查，如以下示例所示。

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF
可以使用以下命名约定在 URI 上设置撤销检查。

```
"SslCertRevocationCheck=true"
```

为受管 *IBM MQ .NET* 配置 *TLS*
为受管 IBM MQ .NET 配置 TLS 包括创建签署者证书，然后配置服务器端、客户机端和应用程序。

关于此任务

要配置 TLS，必须先创建合适的签署者证书。 签署者证书可以是自签名证书或由认证中心提供的证书。 虽然可以在开发、测试或预生产系统上使用自签名证书，但请勿在生产系统上使用自签名证书。 在生产系统上，请使用您从可信的外部认证中心 (CA) 获得的证书。

过程

1. 创建签署者证书。
   a) 要创建自签名证书，请使用 IBM MQ 随附的以下工具之一：
   使用 **strmqikm** GUI，或者从命令行使用 **runmqckm** 或 **runmqakm**。 有关使用这些工具的更多信息，请参阅使用 **runmqckm**、**runmqakm** 和 **strmqikm** 来管理数字证书。
   b) 要从认证中心 (CA) 获取针对队列管理器和客户机的证书，请遵循从认证中心获取个人证书中的指示信息。
2. 配置服务器端。
   a) 在队列管理器上使用 GSKit 配置 TLS，如将客户机安全地连接到队列管理器中所述。
   b) 设置 SVRCONN 通道 TLS 属性：
   - 将 **SSLCAUTH** 设置为 "REQUIRED/OPTIONAL"。

- 将 **SSLCIPH** 设置为合适的 CipherSpec。

有关更多信息，请参阅第 495 页的『对非受管 .NET 客户机启用 TLS』。

3. 配置客户机端。
   a) 将客户机证书导入 Windows 证书库（在“用户/计算机帐户”下）。

   IBM MQ .NET 将从 Windows 证书库访问客户机证书，因此您必须将自己的证书导入 Windows 证书库以与 IBM MQ 建立安全套接字连接。 有关如何访问 Windows 密钥库并导入客户机端证书的更多信息，请参阅导入或导出证书和专用密钥。

   b) 提供 CertificateLabel，如将客户机安全地连接到队列管理器中所述。

   c) 如果需要，请编辑 Windows 组策略以设置 CipherSpec，然后，为了使 Windows 组策略更新生效，请重新启动计算机。
4. 配置应用程序。
   a) 设置 MQEnvironment 或 SSLCipherSpec 值来将连接表示为安全连接。

   您指定的值用于标识正在使用的协议 (TLS)。 CipherSpec 集应该是受支持的 SSLProtocol 版本的 CipherSpec 之一，并且可以首选与 Windows 组策略中指定的 CipherSpec 相同。 （受支持的 SSLProtocol 版本取决于所使用的 .NET Framework。 SSLProtocol 版本可能是 TLS 1.0 或 TLS 1.2，这取决于您正在使用的 Microsoft .NET Framework 版本。）

   注: 如果应用程序提供的 CipherSpec 值不是 IBM MQ 可识别的 CipherSpec，那么 IBM MQ 受管 .NET 客户机将忽略该值，并且会根据 Windows 系统的组策略来协商连接。

   b) 将 SSLKeyRepository 属性设置为 "*SYSTEM" 或 "*USER"。

   c) 可选： 将 SSLPEERNAME 设置为服务器证书的专有名称 (DN)。

   d) 提供 CertificateLabel，如将客户机安全地连接到队列管理器中所述。

   e) 设置您需要的任何其他可选参数（如 KeyResetCount、CertificationRevocationCheck），然后启用 FIPS。

如何设置 TLS 协议和 TLS 密钥存储库的示例

对于基本 .NET，您可以通过 MQEnvironment 类来设置 TLS 协议和 TLS 密钥存储库，如以下示例所示：

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

或者，您可以通过提供散列表作为 MQQueueManager 构造函数的一部分来设置 TLS 协议和 TLS 密钥存储库，如以下示例所示：

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

下一步做什么

有关开发 IBM MQ .NET 受管 TLS 应用程序入门的更多信息，请参阅第 502 页的『编写简单的应用程序』。

相关参考

MQEnvironment .NET 类
KeyResetCount (MQLONG)
针对 UNIX, Linux, and Windows 的美国联邦信息处理标准 (FIPS)

编写简单的应用程序
用于编写简单的 IBM MQ 受管 .NET TLS 应用程序的提示，包括为连接工厂设置 SSL 属性、创建队列管理器实例、连接、会话和目标以及发送测试消息等的示例。

开始之前

您必须先为受管 IBM MQ.NET 配置 TLS，如第 501 页的『为受管 IBM MQ .NET 配置 TLS』中所述。

对于基本 .NET 中的应用程序配置，请通过使用 MQEnvironment 类或提供 hashtable 作为 MQQueueManager 构造函数的一部分来设置 SSL 属性。

对于 XMS .NET 中的应用程序配置，请在连接工厂的属性上下文上设置 SSL 属性。

过程

1. 为连接工厂设置 SSL 属性，如以下示例所示。

   IBM MQ.NET 的示例

   ```
   properties = new Hashtable();
   properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
   properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
   properties.Add(MQC.PORT_PROPERTY, port);
   properties.Add(MQC.CHANNEL_PROPERTY, channelName);
   properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
   properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
   properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
   properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
   properties.Add("CertificateLabel", "ibmwebspheremq");
   MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
   ```

   XMS .NET 的示例

   ```
   cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
   cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
   cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
   cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
   cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
   ```

2. 创建队列管理器实例、连接、会话和目标，如以下示例所示。

   MQ .NET 的示例

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
   MQC.MQOO_FAIL_IF_QUIESCING);
   Console.WriteLine("done");
   ```

   XMS .NET 的示例

   ```
   connectionWMQ = cf.CreateConnection();
   // Create session
   sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

   // Create destination
   destination = sessionWMQ.CreateQueue(destinationName);

   // Create producer
   producer = sessionWMQ.CreateProducer(destination);
   ```

3. 发送消息，如以下示例所示。

   MQ .NET 的示例

   ```
   // creating a message object
   message = new MQMessage();
   message.WriteString(messageString);

   // putting messages continuously
   for (int i = 1; i <= numberOfMsgs; i++)
   ```

```
{
Console.Write("Message " + i + " <" + messageString + ">.. ");
queue.Put(message);
Console.WriteLine("put");
}
```

XMS .NET 的示例

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. 验证 TLS 连接。

   检查通道状态以验证 TLS 连接是否已建立以及是否运行正常。

为 *SSLStream* 配置跟踪

要捕获与 SSLStream 类相关的跟踪事件和消息，必须将系统诊断的配置部分添加到应用程序的应用程序配置文件中。

关于此任务

如果未将系统诊断的配置部分添加到应用程序配置文件中，那么 IBM MQ 管理的 .NET 客户机将不会捕获与 TLS 和 SSLStream 类相关的任何事件、跟踪或调试点。

注: 使用 **strmqtrc** 启动 IBM MQ 跟踪不会捕获所有必需的 TLS 跟踪。

过程

1. 为应用程序项目创建应用程序配置 (App.Config) 文件。
2. 添加系统诊断配置部分，如以下示例中所示。

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
           </sources>
           <switches>
                 <add name="System.Net" value="Verbose"/>
                 <add name="System.Net.Sockets" value="Verbose"/>
                 <add name="System.Net.Cache" value="Verbose"/>
                 <add name="System.Security" value="Verbose"/>
                 <add name="System.Net.Security" value="Verbose"/>
           </switches>

           <sharedListeners>
                 <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
           </sharedListeners>
```

```
        <trace autoflush="true"/>
    </system.diagnostics>
```

注意: add name 条目的 Version 字段需要正在使用的 .net amqmdnet.dll 文件的任一版本。

用于在受管 *.NET* 中实施 *TLS* 的样本应用程序
提供了样本应用程序以显示 IBM MQ classes for .NET， XMS .NET 和 IBM MQ WCF 定制通道中受管 .NET 的 TLS 实现。

下表显示了这些样本应用程序的位置。 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。

表 *77:* 用于在受管 *.NET* 中实施 *TLS* 的样本应用程序的位置

IBM MQ.NET 堆栈产品	**样本的位置**
基本 .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
面向 WCF 的 IBM MQ 定制通道	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs

Windows 使用 .NET 监视器

.NET 监视是与 IBM MQ 触发器监视器相似的应用程序。

要点: See Features that can be used only with the primary installation on Windows for important information.

您可以创建 .NET 组件，每当在受监视队列上收到消息时会将这些组件序列化，然后处理该消息。 .NET Monitor 可以通过 **runmqdnm** 命令来启动，并通过 **endmqdnm** 命令来停止。 有关这些命令的详细信息，请参阅 runmqdnm 和 endmqdnm。

要使用 .NET Monitor，您可以编写用于实现 IMQObjectTrigger 接口的组件，该接口在 amqmdnm.dll 中定义。

组件可以是事务型或者非事务型。 事务型组件必须从 System.EnterpriseServices.ServicedComponent 继承，且必须注册为 RequiresTransaction 或 SupportsTransaction。 因为 .NET Monitor 已经启动一个事务，所以不能将该组件注册为 RequiresNew。

该组件会接收来自 **runmqdnm** 的 MQQueueManager、MQQueue 和 MQMessage 对象。 如果已经使用 *-u* 命令行选项指定了一个“用户参数”字符串，那么在 runmqdnm 启动时，也会收到“用户参数”字符串。 请注意，组件收到 MQMessage 对象中的受监视队列的消息内容。 组件不需要连接到队列管理器，打开队列或获取消息本身。 然后该组件必须在适当的情况下处理消息并将控制权返回给 .NET Monitor。

如果已将组件编写为事务型组件，那么其将使用由 System.EnterpriseServices.ServicedComponent 提供的工具注册提交或回滚事务

由于组件接收 MQQueueManager 和 MQQueue 对象以及消息，因此其具有该消息的完整上下文信息，并且可以在同一队列管理器上打开另一个队列，而无需单独连接到 IBM MQ。

Windows 代码片段示例

本主题包含两个组件示例，用于从 .NET 监视器获取并打印消息，其中一个使用事务处理，另一个使用非事务处理。 第三个示例显示了适用于前两个示例的常见实用程序例程。 所有示例均采用 C# 编写。

示例 1：事务处理

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

示例 2：非事务处理

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;
```

```
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

示例 3：常见例程

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ---------------------------------------------------------------- */
    /* Default prefix string of the namespace.                           */
    /* ---------------------------------------------------------------- */
    private string prefixText = "dnmsamp";

    /* ---------------------------------------------------------------- */
    /* Constructor that takes the replacement prefix string to use.     */
    /* ---------------------------------------------------------------- */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ---------------------------------------------------------------- */
    /* Display an arbitrary string to the console.                      */
    /* ---------------------------------------------------------------- */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }


 /* ---------------------------------------------------------------- */
    /* Display the content of the message passed to the console.        */
    /* ---------------------------------------------------------------- */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
```

```
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

编译 IBM MQ .NET 程序

用于编译使用各种语言编写的 .NET 应用程序的样本命令。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

要使用 IBM MQ classes for .NET 构建 C# 应用程序，请使用以下命令：

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

要使用 IBM MQ classes for .NET 构建 Visual Basic 应用程序，请使用以下命令：

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

要使用 IBM MQ classes for .NET 构建受管 C++ 应用程序，请使用以下命令：

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

有关其他语言，请语言供应商提供的文档。

使用独立的 IBM MQ .NET 客户机

从 IBM MQ 8.0.0 Fix Pack 2 开始，IBM MQ .NET 客户机可以打包和部署 IBM MQ .NET 组合件，无需在生产系统上使用完整的 IBM MQ 客户机安装即可运行您的应用程序。

开始之前

Windows V 9.1.1 从 IBM MQ 9.1.1 中，`amqmdnetstd.dll` 库可用于 Windows 上的 .NET Standard 支持（请参阅 第 465 页的『安装 IBM MQ classes for .NET Standard』）。 仍会提供 `amqmdnet.dll` 库，但该库已稳定下来（即，不会引入任何新功能）。 对于任何最新功能，必须将其迁移到 `amqmdnetstd.dll` 库。 但是，您可以继续在 IBM MQ 9.1 Long Term Support 或 Continuous Delivery 发行版上使用 `amqmdnet.dll` 库。

Linux V 9.1.2 从 IBM MQ 9.1.2 中，Linux 上也提供了 `amqmdnetstd.dll` 库。

关于此任务

从 IBM MQ 8.0.0 Fix Pack 2 中，您可以在安装了完整 IBM MQ 客户机的计算机上构建 IBM MQ .NET 应用程序，然后将 IBM MQ .NET 组合件（即 amqmdnet.dll）与应用程序一起打包，并将其部署到生产系统上。

您构建和部署的应用程序可以是传统的 Windows .NET 应用程序、服务或 Microsoft Azure Web/Worker 应用程序。

在这样的部署中，IBM MQ .NET 客户机仅支持以受管方式连接到队列管理器。 服务器绑定方式和非受管客户机方式的连接均不可用，因为这两种方式都需要完整的 IBM MQ 客户机安装。 使用这两种方式的任何尝试都将导致应用程序异常。

过程

在应用程序中引用 IBM MQ .NET 客户机组合件

- 以先前发行版的相同方式在应用程序中引用 amqmdnet.dll 组合件。

  将 amqmdnet 组合件的 **CopyLocal** 属性设置为 True，以确保已将 amqmdnet 组合件复制到应用程序的 bin 目录中。 设置此属性还有助于应用程序打包工具打包所需的二进制文件，以便在生产系统和 Microsoft Azure PaaS 云环境上进行部署。

添加全局事务支持

- 确保在机器上随应用程序自身一起部署了监视器应用程序 WMQDotnetXAMonitor。

  如果应用程序使用 IBM MQ .NET 管理的全局事务功能，那么还必须在机器上随应用程序自身一起部署 WMQDotnetXAMonitor。 需要使用此实用程序来恢复任何不确定的事务。

使用应用程序配置文件来启动和停止跟踪

- 要启动和停止跟踪，请使用应用程序配置文件和特定于 IBM MQ 的跟踪配置文件。

  您必须使用应用程序配置文件和特定于 IBM MQ 的跟踪配置文件，因为既没有完整的 IBM MQ 客户机安装，也未提供用于启动和停止跟踪的标准工具 **strmqtrc** 和 **endmqtrc**。

  注意:

  – 用于生成跟踪的以下步骤适用于 .NET 可重新分发的受管客户机和独立的 .NET 客户机。

  – V 9.1.1 在 .NET Standard 中不支持应用程序配置文件。 要在 IBM MQ .NET Standard 上启用跟踪，请使用 **MQDOTNET_TRACE_ON** 环境变量。

  应用程序配置文件（app.config 或 web.config）

  应用程序需要在应用程序配置文件（即 app.config 或 web.config 文件）的 <appSettings> 部分下定义 **MQTRACECONFIGFILEPATH** 属性。 （应用程序配置文件的实际名称取决于您的应用程序名称。） **MQTRACECONFIGFILEPATH** 属性的值将指定特定于 IBM MQ 的跟踪配置文件 mqtrace.config 的位置路径，如以下示例中所示:

  ```
  <appSettings>
  <add key="MQTRACECONFIGFILEPATH" value="C:\MQTRACECONFIG" />
  </appSettings>
  ```

  如果在所指定应用程序配置文件的路径中找不到 mqtrace.config 文件，那么将禁用跟踪。 但是，如果应用程序有权写入到当前目录，那么会在应用程序所在目录中创建 First Failure Support Technology (FFST) 和错误日志。

  IBM MQ 特定跟踪配置文件（mqtrace.config）

  mqtrace.config 文件是一个 XML 文件，其中定义了用于启动和停止跟踪的属性、跟踪文件的路径以及错误日志的路径。 下表描述了这些属性。

表 78: *mqtrace.config* 文件中定义的属性	
属性	**描述**
MQTRACELEVEL	0：停止跟踪（缺省值）。 1：启动包含较少详细信息的跟踪。 2：启动包含完整详细信息的跟踪（推荐）。
MQTRACEPATH	指向将在其中创建跟踪文件的文件夹。 如果路径为空白或未定义 **MQTRACEPATH** 属性，那么将使用应用程序的当前目录。
MQERRORPATH	指向将在其中创建错误日志文件的文件夹。 如果路径为空白或未定义 **MQERRORPATH** 属性，那么将使用应用程序的当前目录。

以下示例显示了样本 mqtrace.config 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<traceSettings>
  <MQTRACELEVEL>2</MQTRACELEVEL>
  <MQTRACEPATH>C:\MQTRACEPATH</MQTRACEPATH>
  <MQERRORPATH>C:\MQERRORLOGPATH</MQERRORPATH>
</traceSettings>
```

当应用程序正在运行时，可以通过更改 mqtrace.config 文件中 **MQTRACELEVEL** 属性的值来动态启动和停止跟踪。

运行的应用程序必须具有 **MQTRACELEVEL** 属性所指定的文件夹的创建和写许可权，才能生成跟踪文件。 在 Microsoft Azure PaaS 环境中运行的应用程序还必须确保具有类似的访问许可权，因为使用在 Microsoft Azure PaaS 中运行的 IBM MQ .NET 组合件的 Web 应用程序可能没有创建和写许可权。 如果应用程序不具有所指定文件夹的创建和写许可权（必需），那么将无法生成跟踪、首次故障数据捕获 (FDC) 和错误日志。

在应用程序配置文件中启用绑定重定向

- 要启用从 IBM MQ .NET 组合件到更高版本的组合件的编译时绑定引用，请将 <dependentAssembly> 属性添加到应用程序配置文件。

  app.config 文件中的以下示例片段将重定向使用 IBM MQ 8.0.0 Fix Pack 2 (8.0.0.2) 版本的 IBM MQ .NET 组合件编译的应用程序，但随后将应用修订包 IBM MQ 8.0.0 Fix Pack 3 将 IBM MQ.NET 组合件更新为 8.0.0.3。

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

相关概念

IBM MQ 组件和功能

可重新分发的客户机

第 479 页的『使用 WMQDotnetXAMonitor 应用程序』

必须手动运行 WMQDotnetXAMonitor 应用程序。 可以随时将其启动。 当您在 SYSTEM.DOTNET.XARECOVERY.QUEUE 或您可以在对使用 IBM MQ .NET 类编写的应用程序执行任何事务性工作之前保持其在后台运行。

相关参考

.NET 应用程序运行时（仅限 Windows）

开发 XMS .NET 应用程序

IBM Message Service Client for .NET (XMS .NET) 提供了称为 XMS 的应用程序编程接口 (API)，其接口集与 Java Message Service (JMS) 相同 API。 IBM Message Service Client for .NET 包含一个完全受管的 XMS 实现，此实现可由任何与 .NET 兼容的语言使用。

关于此任务

XMS 支持：

- 点到点消息传递
- 发布/预订消息传递
- 同步消息传递
- 异步消息传递

XMS 应用程序可以与以下类型的应用程序交换消息：

- XMS 应用程序
- IBM MQ classes for JMS 应用程序
- 本机 IBM MQ 应用程序
- 使用 IBM MQ 缺省消息传递提供程序的 JMS 应用程序

XMS 应用程序可以连接到以下任何消息传递服务器并使用其资源：

IBM MQ 队列管理器
应用程序可以在绑定或客户机方式下连接。

WebSphere Application Server service integration bus
应用程序可以使用直接 TCP/IP 连接，也可以使用“基于 TCP/IP 的 HTTP”。

IBM Integration Bus
将使用 WebSphere MQ Real-Time Transport 在应用程序和代理程序之间传输消息。 可使用 WebSphere MQ Multicast Transport 将消息传递到应用程序。

通过连接到 IBM MQ 队列管理器，XMS 应用程序可以使用 WebSphere MQ Enterprise Transport 来与 IBM Integration Bus 进行通信。 或者，XMS 应用程序可以通过连接到 IBM MQ 来进行发布和预订。

V 9.1.1 从 IBM MQ 9.1.1 开始，对于 Windows 环境中的应用程序， IBM MQ 支持 .NET Core 。 有关更多信息，请参阅第 516 页的『使用 IBM MQ classes for XMS .NET Standard』。

V 9.1.2 从 IBM MQ 9.1.2 开始，对于 Linux 环境中的应用程序， IBM MQ 支持 .NET Core 。

V 9.1.4 从 IBM MQ 9.1.4 开始，XMS .NET 受管应用程序能够自动均衡集群队列管理器中的连接。 .NET Framework 和 .NET Standard 库都受支持。 有关更多信息，请参阅统一集群和自动应用程序均衡。

XMS 支持的消息传递样式

XMS 支持点到点和发布/预订样式的消息传递。

消息传递类型也称为消息传递域。

点到点消息传递

常见形式的点到点消息传递使用排队。 在最简单的情况下，一个应用程序通过隐式或显式标识目标队列来将消息发送到另一个应用程序。 底层的消息传递和排队系统将从发送应用程序接收消息，然后将此消息传递至其目标队列。 然后，接收应用程序可以从该队列中检索此消息。

如果底层的消息传递和排队系统包含 IBM Integration Bus，那么 IBM Integration Bus 会复制消息并将消息副本传递到不同的队列。 因此，可能有多个应用程序收到该消息。 IBM Integration Bus 可能还会转换消息并向其添加数据。

点到点消息传递的一个关键特性是，应用程序在发送消息时将消息放置到本地队列上。 底层的消息传递和排队系统会确定要将消息发送到哪个目标队列。 接收应用程序将从目标队列中检索消息。

发布/预订消息传递

在发布/预订消息传递中，有两种类型的应用程序：发布者和订户。

发布程序采用发布消息的形式提供信息。 当发布程序发布消息时，它会指定主题以用于标识消息内信息的主题。

预订程序是所发布信息的使用者。 预订程序通过创建预订来指定其感兴趣的主题。

发布/预订系统将接收来自发布程序的发布和来自预订程序的预订。 它会将发布内容传递到预订程序。 预订程序仅接收其预订的那些主题的发布内容。

发布/预订消息传递的一个关键特性是，发布程序在发布消息时标识主题。 它不会标识预订程序。 如果针对某个主题发布的消息没有预订程序，那么所有应用程序都不会收到该消息。

应用程序可以既是发布者也是订户。

XMS 对象模型

XMS API 是一个面向对象的接口。 XMS 对象模型基于 JMS 1.1 对象模型。

XMS 主类

以下是 XMS 主类或对象类型：

ConnectionFactory
ConnectionFactory 对象封装了连接的一组参数。 应用程序使用 ConnectionFactory 来创建连接。 应用程序可以在运行时提供参数并创建 ConnectionFactory 对象。 或者，可以将连接参数存储在受管对象存储库中。 应用程序可以从该存储库中检索对象，并通过该对象来创建 ConnectionFactory 对象。

Connection
Connection 对象封装了从应用程序到消息传递服务器的活动连接。 应用程序使用连接来创建会话。

Destination
应用程序使用 Destination 对象来发送消息或接收消息。 在发布/预订域中，Destination 对象封装了主题，在点到点域中， Destination 对象封装了队列。 应用程序可以提供参数以在运行时创建 Destination 对象。 或者，可以通过存储在受管对象存储库中的对象定义来创建 Destination 对象。

Session
Session 对象是用于发送和接收消息的单线程上下文。 应用程序使用 Session 对象创建 Message、MessageProducer 和 MessageConsumer 对象。

消息
Message 对象封装了应用程序使用 MessageProducer 对象发送或使用 MessageConsumer 对象接收的 Message 对象。

MessageProducer
应用程序使用 MessageProducer 对象将消息发送到目标。

MessageConsumer
应用程序使用 MessageConsumer 对象接收已发送到目标的消息。

XMS 对象及其关系

第 513 页的图 58 显示 XMS 对象的主要类型：ConnectionFactory、Connection、Session、MessageProducer、MessageConsumer、Message 和 Destination。 应用程序使用连接工厂来创建连接，并使用连接来创建会话。 然后，应用程序可以使用会话来创建消息、消息生产者和消息使用者。 应用程序使用消息生产者向目标发送消息，并使用消息使用者接收向目标发送的消息。

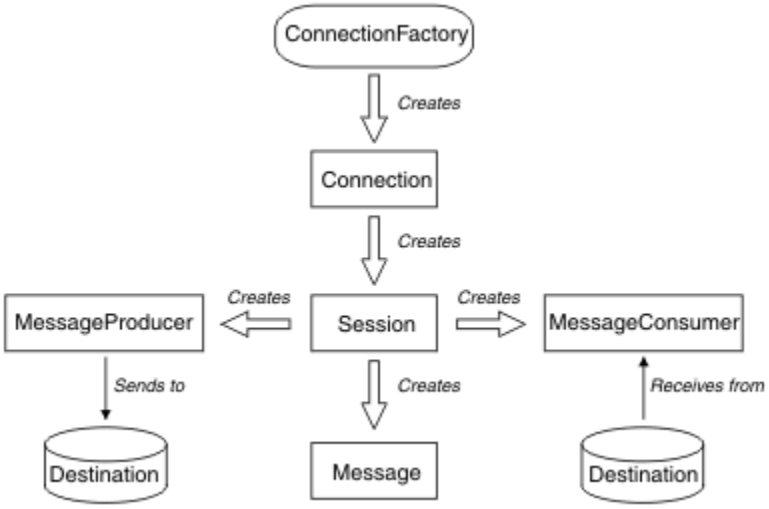


图 *58: XMS* 对象及其关系

在 XMS .NET 中，XMS 类定义为一组 .NET 接口。 当对 XMS .NET 应用程序进行编码时，只需要声明的接口。

XMS 对象模型基于 Java Message Service Specification V1.1 中描述的独立于域的接口。 未提供特定于域的类，如 `Topic`、`TopicPublisher` 和 `TopicSubscriber`。

XMS 对象的属性和特性

XMS 对象可以具有属性和特性，这些是对象的特征，以不同方式实现：

属性
始终存在且占用存储空间（即使属性不含值也是如此）的对象特征。 从这方面讲，属性与固定长度数据结构中的字段类似。 属性的显著特点是：每个属性都有自己的用于设置和获取其值的方法。

属性
对象的特性存在且仅在设置其值后才占用存储空间。 在设置特性的值之后，无法删除该属性或恢复其存储空间。 您可以更改其值。 XMS 提供一组常规方法来用于设置和获取特性值。

受管对象

通过使用受管对象，可以在中央存储库中管理您希望管理的客户机应用程序所使用的连接设置。 应用程序从中央存储库中检索对象定义，然后使用这些定义来创建 `ConnectionFactory` 和 `Destination` 对象。 通过使用受管对象，可以将应用程序与其在运行时使用的资源分离开来。

例如，可以使用受管对象（引用了测试环境中的一组连接和目标）来编写和测试 XMS 应用程序。 在部署应用程序时，可以更改受管对象，以将应用程序配置为引用生产环境中的连接和目标。

XMS 支持以下两种类型的受管对象：

- `ConnectionFactory` 对象，可供应用程序用来与服务器建立初始连接。
- `Destination` 对象，可供应用程序用来指定所发送消息的目标以及所接收消息的源。 目标是应用程序要连接到的服务器上的主题或队列。

IBM MQ 随附了管理工具 **JMSAdmin** 。 它用于在受管对象的中央存储库中创建和管理受管对象。

该存储库中的受管对象可由 IBM MQ classes for JMS 和 XMS 应用程序使用。 XMS 应用程序可以使用 `ConnectionFactory` 和 `Destination` 对象来连接到 IBM MQ 队列管理器。 管理员可以更改该存储库中保存的对象定义，而不影响应用程序代码。

下图显示了 XMS 应用程序通常如何使用受管对象。 该图的左侧显示存储库，其中包含使用管理控制台管理的 ConnectionFactory 和 Destination 对象定义。 该图的右侧显示 XMS 应用程序，此应用程序在该存储库中查找对象定义，然后在连接到消息传递服务器时使用这些对象定义。

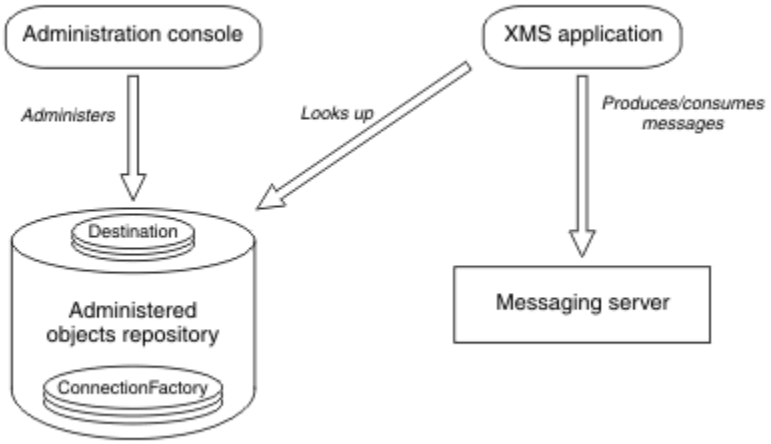


图 *59:* 受管对象在 *XMS* 应用程序中的典型用途

XMS 消息模型

XMS 消息模型与 IBM MQ classes for JMS 消息模型相同。

特别是，XMS 实现的消息头字段和消息属性与 IBM MQ classes for JMS 实现的相同：

- JMS 头字段。 这些字段的名称以前缀 JMS 开头。
- JMS 定义的属性。 这些字段的属性名以前缀 JMSX 开头。
- IBM 定义的属性。 这些字段的属性名以前缀 JMS_IBM_ 开头。

因此，XMS 应用程序可以与 IBM MQ classes for JMS 应用程序交换消息。 在每条消息中，某些头字段和属性由应用程序设置，而其他头字段和属性由 XMS 或 IBM MQ classes for JMS 设置。 由 XMS 或 IBM MQ classes for JMS 设置的某些字段是在发送消息时设置，而其他字段是在接收消息时设置。 在适用情况下，通过消息传递服务器，将头字段和属性随消息一起传播。 它们可用于任何接收消息的应用程序。

相关概念

IBM MQ classes for JMS

设置消息传递服务器环境

本部分中的主题描述了如何设置消息传递服务器环境以允许 XMS 应用程序连接到服务器。

关于此任务

对于连接到 IBM MQ 队列管理器的应用程序，需要 IBM MQ 客户机（或者，如果是绑定方式，那么需要队列管理器）。

对于使用与代理程序的实时连接的应用程序，目前不存在任何先决条件。

运行任何 XMS 应用程序（包括 XMS 随附的样本应用程序）之前，必须设置消息传递服务器环境。

本部分包含以下主题：

- 第 519 页的『为连接到 IBM MQ 队列管理器的应用程序配置队列管理器和代理程序』

- V 9.1.1 第 516 页的『使用 IBM MQ classes for XMS .NET Standard』
- 第 520 页的『为使用与代理程序的实时连接的应用程序配置代理程序』
- 第 521 页的『为连接到 WebSphere Application Server 的应用程序配置服务集成总线』

XMS .NET 中的消息侦听器

消息侦听器用于异步接收消息。 与 `MessageConsumer.receive()` 调用不同，消息侦听器不会阻止调用线程，而是将消息传递到应用程序指定的回调方法 (通常是 **onMessage** 方法)。

调用 **Connection.Start()** 方法后，将启动消息传递。 可以分别使用 **Connection.Stop()** 和 **Connection.Start()** 方法来停止和恢复消息传递。

在将消息侦听器设置为会话中的至少一个使用者之后，调用 **Connection.Start()** 方法后，该会话将变为异步会话。 一旦会话变为异步，就无法调用任何 XMS .NET 同步方法。 例如，**MessageProducer.Send()**。 这样做会导致异常，原因码为 IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500)。

异步会话中的同步调用

`Session.Close` 是异步会话中允许的唯一同步调用。 应用程序还可以使用消息侦听器回调方法 (即 **onMessage** 方法) 进行同步调用 (`Session.Close` 除外)。

除了这两个选项以外，您必须使用应用程序的 **Connection.Stop()** 方法停止连接以进行任何同步调用。 进行调用后，必须使用 **Connection.Start()** 方法再次恢复连接。 这将重新启动消息传递。

会话可以具有多少异步消息使用者?

一个会话可以具有多个异步消息使用者。 但在任何时候，一条消息仅传递给一个使用者。 这实际上意味着，当第二条消息到达时，当 XMS .NET 已调用使用者的 **onMessage()** 方法来传递第一条消息时，第二条消息将不会在会话中传递给使用者，直到 **onMessage()** 方法返回为止。

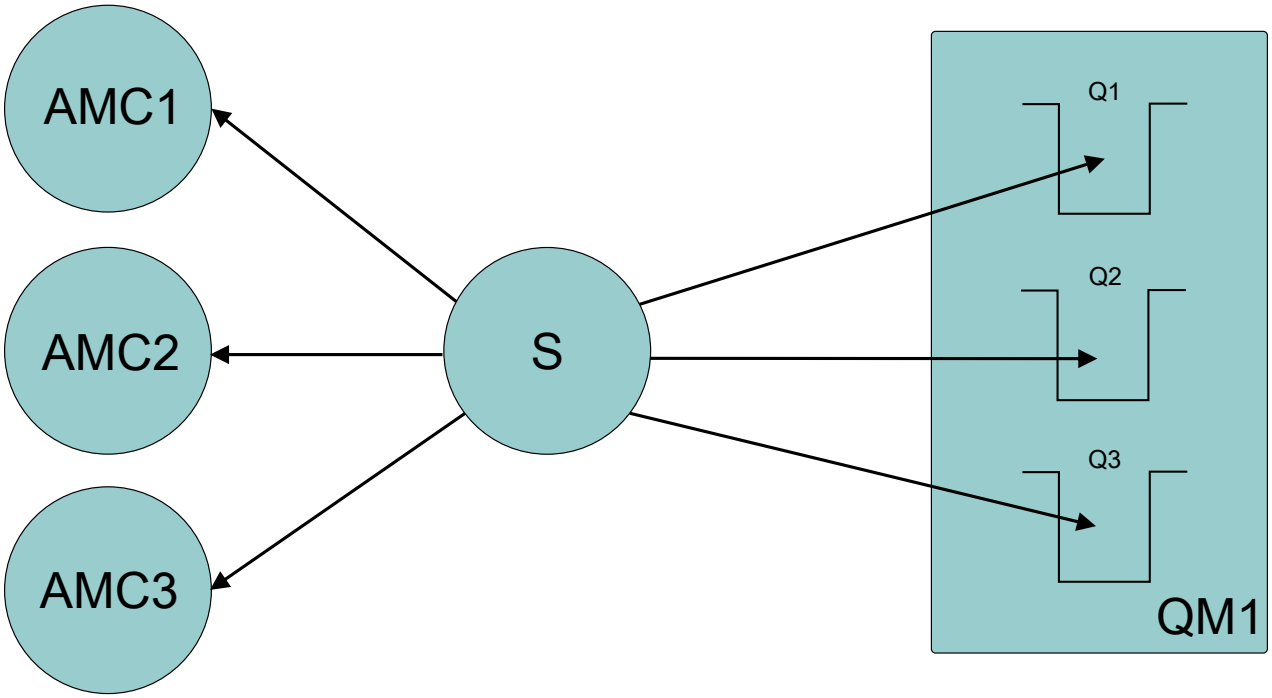
只有在 **onMessage()** 方法返回后，才会将第二条消息传递到会话中的使用者。 这是因为会话仅使用一个线程来管理对使用者的消息传递。 这意味着一次只能传递一条消息，而消费者可能是任何一条。

如果应用程序需要并发消息传递，即所有使用者必须同时接收消息，那么应用程序必须创建多个会话，并且每个会话都必须具有一个异步消息使用者。

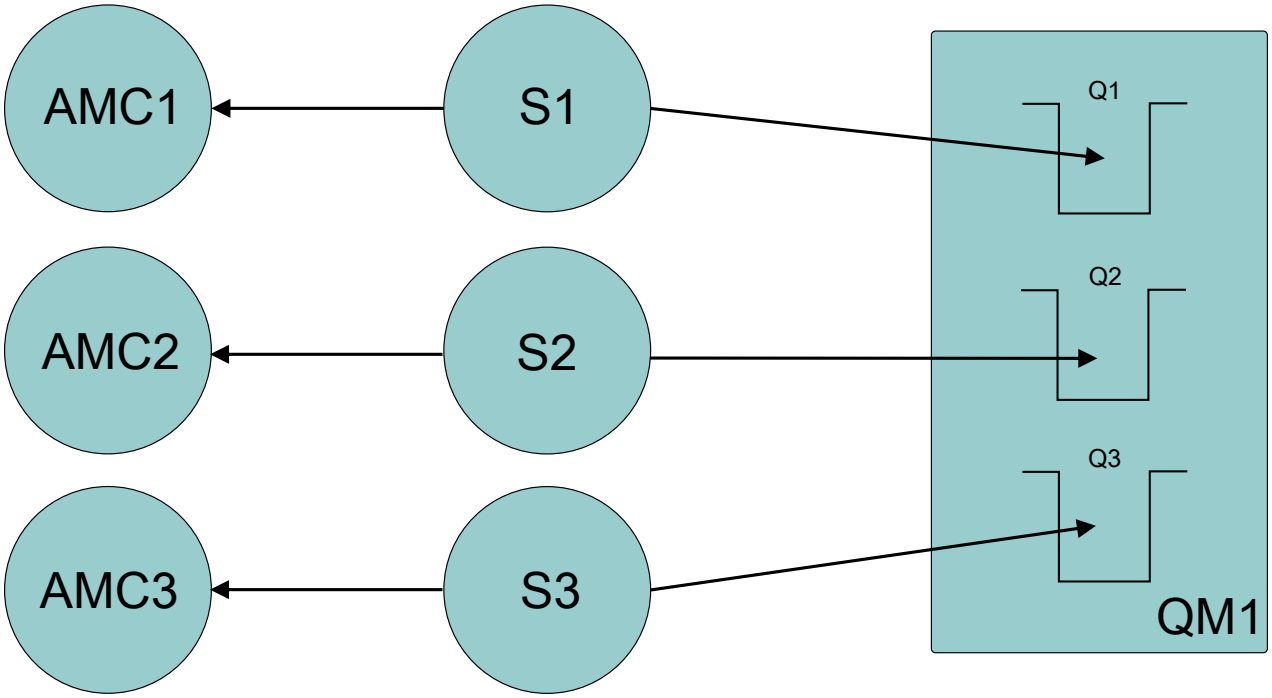
以下示例更清楚地显示了此功能。

在第一个示例中，一个会话中有多个异步消息使用者。 会话 S 具有三个异步消息使用者: AMC1， AMC2 和 AMC3 ，它们从三个不同的目标 Q1， Q2 和 Q3 接收消息。

由于只有一个会话 S，因此只有消息传递线程将消息传递到使用者 AMC1， AMC2 和 AMC3。 当会话将消息传递到 AMC1 时，其他两个使用者 AMC2 和 AMC3 等待，即使 Q2 和 Q3 中有消息可供传递。



在第二种情况下，有多个会话 S1， S2 和 S3，每个会话分别具有一个异步消息使用者 AMC1， AMC2 和 AMC3 。 由于每个会话都有一个使用者，因此会同时将消息传递给使用者。



这表明如果需要并发消息传递，那么需要多个会话。

Windows V 9.1.1 Linux

使用 IBM MQ classes for XMS .NET Standard

如何将 XMS 与 Microsoft .NET Standard 结合使用，以及使用 IBM MQ classes for XMS .NET Framework 和 IBM MQ classes for XMS .NET Standard 的差别。 必备软件为 Microsoft.NET Core for IBM MQ classes for XMS .NET Standard。

amqmxmsstd.dll 库

Windows 从 IBM MQ 9.1.1 开始， IBM MQ classes for XMS .NET Standard 库 amqmxmsstd.dll 可用于 Windows 上的 XMS .NET Standard 支持。

Linux V 9.1.2 从 IBM MQ 9.1.2 中，Linux 上也提供了 amqmxmsstd.dll 库。 当 IBM MQ 客户机安装在 Linux 上时，库将安装到 /&MQINSTALL_PATH&/lib64 path 中。 XMS .NET 样本位于 &MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms 中。

有关更多信息，请参阅第 465 页的『安装 IBM MQ classes for .NET Standard』。

注意: 仍会提供所有的 IBM.XMS.* 库，但这些库已稳定下来（即，不会引入任何新功能）。

对于任何最新功能，必须将其迁移到 amqmxmsstd.dll 库。 但是，您可以在 IBM MQ 9.1 Long Term Support 或 Continuous Delivery 发行版上继续使用现有库。

V 9.1.4 从 IBM MQ 9.1.4 开始，可从 NuGet 存储库下载 IBM MQ classes for XMS .NET Standard 。 NuGet 包同时包含 amqmxmsstd.dll 库和 amqmdnetstd.dll 库。 amqmxmsstd.dll 依赖于 amqmdnetstd.dll，并且在打包 XMS .NET Core 应用程序时，amqmxmsstd.dll 和 amqmdnetstd.dll 应该与 XMS .NET Core 应用程序一起打包。 有关更多信息，请参阅第 518 页的『从 NuGet 存储库下载 IBM MQ classes for XMS .NET Standard』。

dspmqver 命令

从 IBM MQ 9.1.1 开始，可以使用 **dspmqver** 命令来显示 .NET Core 组件的版本和构建信息。

IBM MQ classes for XMS .NET Framework 和 IBM MQ classes for XMS .NET Standard 功能

下表列出了 IBM MQ 9.1.1 for IBM MQ classes for XMS .NET Framework 和 IBM MQ classes for XMS .NET Standard 提供的适用功能部件。

表 *79: IBM MQ classes for XMS .NET Framework* 和 *IBM MQ classes for XMS .NET Standard* 功能部件之间的差异

功能部件	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET Standard
类名 (API)	所有类在每个网络中都保持不变。	所有类在每个网络中都保持不变。
操作系统	Windows	Windows Dockerized 容器 V 9.1.2 Linux V 9.1.3 MacOS
app.config 文件（用于在可再分发客户机中启用跟踪的配置文件）	app.config 文件用于对可重新分发的软件包启用跟踪。	app.config 在 .NET Standard 中不受支持。 您必须使用环境变量来代替 app.config。
跟踪	要跟踪 XMS .NET 客户机，可以使用现有环境变量，例如用于启用跟踪的环境变量 **XMS_TRACE_ON** 。 有关更多信息，请参阅 使用 XMS 环境变量配置 XMS .NET 跟踪。 app.config 文件可用于对可重新分发的软件包启用跟踪。	要跟踪 XMS .NET 客户机，可以使用现有环境变量，例如用于启用跟踪的环境变量 **XMS_TRACE_ON** 。 有关更多信息，请参阅 使用 XMS 环境变量配置 XMS .NET 跟踪。

表 79: *IBM MQ classes for XMS .NET Framework* 和 *IBM MQ classes for XMS .NET Standard* 功能部件之间的差异 (继续)

功能部件	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET Standard
传输方式	受管、非受管和绑定	受管
TLS	Windows 密钥库用于存储证书。	Windows 在 Windows 上，必须使用密钥库来存储证书。 允许的值为 *USER 或 *SYSTEM。 根据输入，IBM MQ .NET 客户机将查看当前用户或系统范围内的 Windows 密钥库。 Linux V 9.1.2 在 Linux 上，建议使用 X509Store 类来安装证书，.NET Core 会将证书安装到以下位置：".dotnet/corefx/cryptography/x509stores"。
CCDT	支持	支持，并且 CCDT 路径的设置对与 .NET Framework 类的相同。
客户机自动重新连接	支持	支持
分布式事务	支持	不支持
将动态链接库 (dll's) 安装到全局组合件高速缓存 (GAC) 中	Dll's 将作为 IBM MQ 安装的一部分安装在 GAC 中。	Dll's 不作为 IBM MQ 安装的一部分安装在 GAC 中。
支持 WMQ、WPM 和 RTT 连接类型	支持 WMQ、WPM 和 RTT 连接类型	仅支持 WMQ
JNDI 受管对象	支持 LDAP 和 FileSystem	仅支持 FileSystem

相关任务

第 522 页的『使用 XMS 样本应用程序』

XMS .NET 样本应用程序概述了每个 API 的常用功能。 使用这些样本应用程序，可以验证安装和消息传递服务器设置，并帮助您构建自己的应用程序。

Windows V 9.1.4 Linux 从 *NuGet* 存储库下载 *IBM MQ classes for XMS .NET Standard*

从 IBM MQ 9.1.4 开始，可以从 NuGet 存储库下载 IBM MQ classes for XMS .NET Standard ， 以便 .NET 开发者可以轻松使用这些内容。

关于此任务

NuGet 是用于 Microsoft 开发平台（包括 .NET）的软件包管理器。 可通过 NuGet 客户机工具来生成和使用软件包。 NuGet 软件包是具有 `.nupkg` 扩展名的单个压缩文件，其中包含已编译的代码 (DLL)、与该代码相关的其他文件以及包含诸如包版本号之类的信息的描述性清单。

从 IBM MQ 9.1.4，您可以从 NuGet Gallery 下载 `IBMXMSDotnetClient` NuGet 软件包，该软件包同时包含 `amqmdnetstd.dll` 库和 `amqmxmsstd.dll` 库，这是所有软件包作者和使用者使用的中央软件包存储库。

可使用三种方式来下载 `IBMXMSDotnetClient` 软件包：

- 使用 Microsoft Visual Studio。 NuGet 作为 Microsoft Visual Studio 扩展分发。 从 Microsoft Visual Studio 2012 开始，缺省情况下已预安装了 NuGet。

- 从命令行中使用 NuGet Package Manager 或 .NET CLI。
- 使用 Web 浏览器。

对于可再分发的软件包，可以使用环境变量 **XMS_TRACE_ON** 来启用跟踪。

过程

- 要使用 Microsoft Visual Studio 中的 Package Manager UI 来下载 `IBMXMSDotnetClient` 软件包，请完成以下步骤：

  a) 右键单击 .NET 项目，然后单击**管理 Nuget 软件包**。

  b) 单击**浏览**选项卡，并搜索“`IBMXMSDotnetClient`”。

  c) 选择软件包，然后单击**安装**。

  在安装期间，Package Manager 将以控制台语句形式提供进度信息。
- 要从命令行中下载 `IBMXMSDotnetClient` 软件包，请选择以下某个选项：
  - 使用 NuGet Package Manager，输入以下命令：

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    在安装期间，Package Manager 将以控制台语句形式提供进度信息。 您可以将输出重定向到某个日志文件。
  - 使用 .NET CLI，输入以下命令：

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- 使用 Web 浏览器从 https://www.nuget.org/packages/IBMXMSDotnetClient 下载 `IBMXMSDotnetClient` 软件包。

相关概念

第 465 页的『安装 IBM MQ classes for .NET Standard』

从 IBM MQ 9.1.1 开始，IBM MQ classes for .NET Standard（包括样本）随 IBM MQ 一起安装在 Windows 上。 V 9.1.2 从 IBM MQ 9.1.2 开始， IBM MQ classes for .NET Standard 在 Linux 平台上也可用。 必备软件为 Microsoft.NET Core for IBM MQ classes for .NET Standard。

相关任务

第 468 页的『从 NuGet 存储库下载 IBM MQ classes for .NET Standard』

从 IBM MQ 9.1.4 开始，可以从 NuGet 存储库下载 IBM MQ classes for .NET Standard ， 以便 .NET 开发者可以轻松使用这些内容。

相关参考

IBM MQ Client for .NET 许可证信息

为连接到 IBM MQ 队列管理器的应用程序配置队列管理器和代理程序

本部分假定您使用的是 IBM WebSphere MQ 7.0.1 或更高版本。 在运行连接到 IBM MQ 队列管理器的应用程序之前，必须先配置该队列管理器。 对于发布/预订应用程序，如果使用的是排队式发布/预订接口，那么需要进行一些额外配置。

开始之前

XMS 使用 IBM Integration Bus 或 WebSphere Message Broker 6.1 或更高版本

在开始此任务之前，请执行以下步骤：

- 确保应用程序有权访问正在运行的队列管理器。
- 如果应用程序是发布/预订应用程序并且使用排队式发布/预订接口，请确保队列管理器上的 **PSMODE** 属性设置为 ENABLED。

- 确保已相应地设置了应用程序所用连接工厂的属性以连接到队列管理器。 如果应用程序是发布/预订应用程序，请确保已设置了相应的连接工厂属性，以便使用代理程序。 有关连接工厂的属性的更多信息，请参阅 ConnectionFactory 的属性。

关于此任务

配置队列管理器和代理以运行 XMS 应用程序的方式与配置队列管理器和排队的发布/预订接口以运行 IBM MQ JMS 应用程序的方式相同。 以下步骤概括了您需要执行的操作。

过程

1. 在队列管理器上，创建应用程序所需的队列。

   有关如何创建队列的概述，请参阅定义队列。

   如果应用程序是发布/预订应用程序，并且使用需要访问 IBM MQ classes for JMS 系统队列的排队式发布/预订接口，请等待完成步骤 4a，然后才能创建队列。
2. 向与应用程序相关的用户标识授予可连接到队列管理器和访问队列的相应权限。

   有关授权的概述，请参阅保护。 如果应用程序以客户机方式连接到队列管理器，请参阅客户机和服务器。
3. 如果应用程序以客户机方式连接到队列管理器，请确保已在队列管理器中定义服务器连接通道，并且已启动侦听器。

   您无需针对连接到队列管理器的每个应用程序都执行此步骤。 通过一个服务器连接通道定义和一个侦听器，就可以支持在客户机方式下连接的所有应用程序。
4. 如果应用程序是发布/预订应用程序，并且使用排队式发布/预订接口，请执行以下步骤。
   a) 在队列管理器上，通过运行随 IBM MQ 提供的 MQSC 命令的脚本来创建 IBM MQ classes for JMS 系统队列。 确保与 IBM Integration Bus 或 WebSphere Message Broker 相关的用户标识有权访问这些队列。

      有关可在何处找到该脚本以及如何运行该脚本的信息，请参阅使用 IBM MQ classes for Java。

      针对队列管理器仅执行该步骤一次。 通过同一组 IBM MQ classes for JMS 系统队列，就可以支持连接到队列管理器的所有 XMS 和 IBM MQ classes for JMS 应用程序。

   b) 向与应用程序相关的用户标识授予可访问 IBM MQ classes for JMS 系统队列的相应权限。

      有关用户标识需要哪些权限的信息，请参阅使用 IBM MQ classes for JMS。

   c) 对于 IBM Integration Bus 或 WebSphere Message Broker 代理程序，请创建并部署消息流，以便服务于可供应用程序发送其发布的消息的队列。

      基本消息流包括一个用于读取已发布消息的 MQInput 消息处理节点和一个用于发布消息的 Publication 消息处理节点。

      有关如何创建和部署消息流的信息，请参阅 IBM Integration Bus 产品文档库 Web 页面中提供的 IBM Integration Bus 或 WebSphere Message Broker 产品文档。

      如果代理程序上已部署了合适的消息流，那么无需执行该步骤。

结果

现在，您可以启动应用程序。

为使用与代理程序的实时连接的应用程序配置代理程序

在运行使用与代理程序的实时连接的应用程序之前，必须先配置该代理程序。

开始之前

在开始此任务之前，请执行以下步骤：

- 确保应用程序有权访问正在运行的代理程序。

- 确保已根据与代理程序的实时连接相应地设置了应用程序所用连接工厂的属性。 有关连接工厂的属性的更多信息，请参阅 ConnectionFactory 的属性。

关于此任务

配置代理以运行 XMS 应用程序的方式与配置代理以运行 IBM MQ classes for JMS 应用程序的方式相同。 以下步骤概括了您需要执行的操作：

过程

1. 创建并部署消息流，以从代理程序侦听的 TCP/IP 端口读取消息并发布这些消息。

   您可以使用以下任一方式来执行此操作：

   - 创建包含 **Real-timeOptimizedFlow** 消息处理节点的消息流。
   - 创建包含 **Real-timeInput** 消息处理节点和 Publication 消息处理节点的消息流。

   必须配置 **Real-timeOptimizedFlow** 或 **Real-timeInput** 节点，才能侦听用于实时连接的端口。 在 XMS 中，实时连接的缺省端口号为 1506。

   如果代理程序上已部署了合适的消息流，那么无需执行该步骤。
2. 如果需要使用 IBM MQ classes for JMS 将消息传递到应用程序，请配置代理程序以启用多点广播。 配置那些需要启用多点广播的主题，为需要可靠多点广播的主题指定可靠的服务质量。
3. 如果应用程序在连接到代理程序时提供用户标识和密码，并且您希望代理程序使用这些信息来认证应用程序，请对用户名服务器和代理程序配置简单的 telnet 类型密码认证。

结果

现在，您可以启动应用程序。

为连接到 WebSphere Application Server 的应用程序配置服务集成总线

在运行连接到 WebSphere Application Server service integration technologies 服务集成总线的应用程序之前，必须先配置该服务集成总线，其配置方法与配置服务集成总线以运行使用缺省消息传递提供程序的 JMS 应用程序的方式相同。

开始之前

在开始此任务之前，必须执行以下步骤：

- 确保已创建消息传递总线，并且您的服务器已作为总线成员添加到该总线中。
- 确保应用程序有权访问至少包含一个正在运行的消息传递引擎的服务集成总线。
- 如果需要 HTTP 操作，那么必须定义 HTTP 消息传递引擎入站传输通道。 缺省情况下，会在服务器安装期间定义用于 SSL 和 TCP 的通道。
- 确保已相应地设置了应用程序所用连接工厂的属性，以便使用引导程序服务器连接到服务集成总线。 必须至少提供以下信息：
  - 提供程序端点，用于描述在协商与消息传递服务器的连接（即，通过引导程序服务器）时要使用的位置和协议。 在最简单的形式中，对于使用缺省设置安装的服务器，提供程序端点可设置为服务器的主机名。
  - 用于发送消息的总线的名称。

  有关连接工厂的属性的更多信息，请参阅 ConnectionFactory 的属性。

关于此任务

必须定义所需的任何队列或主题空间。 缺省情况下，会在服务器安装期间定义名为 Default.Topic.Space 的主题空间，但如果需要更多主题空间，您必须自行创建。 您不需要预定义主题空间中的主题，因为服务器会根据需要动态实例化这些主题。

以下步骤概括了您需要执行的操作。

过程

1. 创建在应用程序执行点到点消息传递时需要的队列。
2. 创建在应用程序执行发布/预订消息传递时需要的任何额外主题空间。

结果

现在，您可以启动应用程序。

使用 XMS 样本应用程序

XMS .NET 样本应用程序概述了每个 API 的常用功能。 使用这些样本应用程序，可以验证安装和消息传递服务器设置，并帮助您构建自己的应用程序。

关于此任务

如果您在创建自己的应用程序时需要帮助，那么可以使用这些样本应用程序作为起点。 已提供了每个样本应用程序的源代码和已编译的版本。 请查看样本源代码，并确定关键步骤，以便为应用程序创建所有必需对象（ConnectionFactory、Connection、Session、Destination 以及 Producer 和/或 Consumer）并根据需要设置任何特定属性以指定应用程序的工作方式。 有关更多信息，请参阅第 524 页的『编写 XMS 应用程序』。 在 XMS 的未来发行版中可随时更改这些样本。

下表显示了 XMS 随附的样本应用程序集合（每个 API 一个集合）。

表 *80: XMS .NET* 的样本应用程序

样本名称	**描述**
SampleConsumerCS	从队列中获取消息或预订主题的消息使用者应用程序。
SampleProducerCS	将消息生成到队列中或针对主题生成消息的消息生产者应用程序。
SampleConfigCS	可用于创建基于文件的受管对象存储库的配置应用程序。 该应用程序包含适用于特殊连接设置的连接工厂和目标。 然后，可以将此受管对象存储库与各个样本使用者应用程序和生成者应用程序一起使用。

支持各种 API 中相同功能的样本存在一些语法差异。

- 样本消息使用者和生产者应用程序都支持以下功能：
  - 连接到 IBM MQ、IBM Integration Bus（使用与代理程序的实时连接）和 WebSphere Application Server service integration bus
  - 使用初始上下文接口执行受管对象存储库查找
  - 连接到队列（IBM MQ 和 WebSphere Application Server service integration bus）和主题（IBM MQ、与代理程序的实时连接和 WebSphere Application Server service integration bus）
  - 基本、字节、映射、对象、流和文本消息
- 样本消息使用者应用程序支持同步和异步接收方式以及 SQL 选择器语句。
- 样本消息生产者应用程序支持持久性和非持久性传递方式。

这些样本可以在以下两种方式下运行：

简单方式
您可以使用最少的用户输入来运行样本。

高级方式
您可以更细微地定制样本的运行方式。

所有样本都兼容，因此可以跨语言运行。

Windows 从 IBM MQ 9.1.1 开始， IBM MQ 在 Windows 环境中支持 .NET Core for XMS .NET 应用程序。 缺省情况下，IBM MQ classes for .NET Standard（包括样本）作为 IBM MQ 标准安装的一部分进行安装。

Linux V 9.1.2 从 IBM MQ 9.1.2 开始， IBM MQ 还支持 .NET Core for applications in Linux 环境。

XMS .NET 的样本应用程序安装在 `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms` 中。

有关更多信息，请参阅第 516 页的『使用 IBM MQ classes for XMS .NET Standard』。

运行 .NET 样本应用程序

您可以在简单或高级方式下，以交互方式运行 .NET 样本应用程序，或者通过自动生成或定制的响应文件，以非交互方式运行这些样本应用程序。

开始之前

在运行所提供的任何样本应用程序之前，必须首先设置消息传递服务器环境，以便应用程序可以连接到服务器。 请参阅 第 514 页的『设置消息传递服务器环境』。

过程

要运行 .NET 样本应用程序，请完成以下步骤：

提示: 在运行样本应用程序时，输入? 随时获取有关下一步操作的帮助。

1. 选择要用于运行样本应用程序的方式。

   输入 `Advanced` 或 `Simple`。
2. 回答问题。

   要选择缺省值（显示在问题结尾处的方括号中），请按 Enter 键。 要选择其他值，请输入相应的值，然后按 Enter 键。

   下面是一个示例问题：

   ```
   Enter connection type [wpm]:
   ```

   在此情况下，缺省值为 `wpm`（与 WebSphere Application Server service integration bus 的连接）。

结果

运行样本应用程序时，会在当前工作目录中自动生成响应文件。 响应文件名的格式为 *connection_type-sample_type*`.rsp`；例如，`wpm-producer.rsp`。 如果需要，可以使用生成的响应文件，采用相同的选项来重新运行样本应用程序，因此不需要再次输入这些选项。

相关任务

构建 .NET 样本应用程序
构建 .NET 样本应用程序时，会创建所选样本的可执行版本。

构建自己的应用程序
您可以像构建样本应用程序那样构建自己的应用程序。

构建 .NET 样本应用程序

构建 .NET 样本应用程序时，会创建所选样本的可执行版本。

开始之前

安装相应的编译器。 此任务假定您已安装了 Microsoft Visual Studio 2012，并且熟悉它的使用方法。

过程

要构建 .NET 样本应用程序，请完成以下步骤：

1. 单击 .NET 样本随附的 `Samples.sln` 解决方案文件。
2. 在“解决方案资源管理器”窗口中右键单击解决方案**样本**，然后选择**构建解决方案**。

结果

此时会在该样本的相应子文件夹（`bin/Debug` 或 `bin/Release`，取决于您所选的配置）中创建可执行程序。 此程序与该文件夹同名，其后缀为 `CS`。 例如，如果要构建 C# 版本的消息生产者样本应用程序，那么会在 `SampleProducer` 文件夹中创建 `SampleProducerCS.exe`。

相关任务

运行 .NET 样本应用程序
您可以在简单或高级方式下，以交互方式运行 .NET 样本应用程序，或者通过自动生成或定制的响应文件，以非交互方式运行这些样本应用程序。

构建自己的应用程序
您可以像构建样本应用程序那样构建自己的应用程序。

第 524 页的『构建自己的应用程序』
您可以像构建样本应用程序那样构建自己的应用程序。

构建自己的应用程序

您可以像构建样本应用程序那样构建自己的应用程序。

开始之前

安装相应的编译器。 此任务假定您已安装了 Microsoft Visual Studio 2012，并且熟悉它的使用方法。

过程

- 构建 .NET 应用程序，如 第 523 页的『构建 .NET 样本应用程序』中所述。
  有关如何构建自己的应用程序的其他指导，请使用为每个样本应用程序提供的 makefile。

  提示: 为了在出现故障的情况下帮助诊断问题，您可能会发现使用包含的符号编译应用程序很有帮助。

相关任务

运行 .NET 样本应用程序
您可以在简单或高级方式下，以交互方式运行 .NET 样本应用程序，或者通过自动生成或定制的响应文件，以非交互方式运行这些样本应用程序。

构建 .NET 样本应用程序
构建 .NET 样本应用程序时，会创建所选样本的可执行版本。

编写 XMS 应用程序

本部分中的主题所提供的信息通常可帮助您编写 XMS 应用程序。

关于此任务

本部分包含编写 XMS 应用程序时涉及的常用概念。 另请参阅第 540 页的『编写 XMS .NET 应用程序』，以获取与创建 XMS .NET 应用程序有关的信息。

本部分包含以下主题：

- 第 526 页的『线程技术模型』
- 第 526 页的『ConnectionFactories 和 Connection 对象』
- 第 527 页的『会话』
- 第 530 页的『目标』
- 第 532 页的『消息生产者』

- 第 533 页的『消息使用者』
- 第 535 页的『队列浏览器』
- 第 536 页的『请求程序』
- 第 536 页的『删除对象』
- 第 537 页的『XMS 基本类型』
- 第 537 页的『属性值从一种数据类型到另一种数据类型的隐式转换』
- 第 539 页的『迭代器』
- 第 539 页的『编码字符集标识』
- 第 539 页的『XMS 错误和异常代码』
- 第 524 页的『构建自己的应用程序』

Windows V 9.1.5 使用 IBM MQ XMS .NET 项目模板

从 IBM MQ 9.1.5 开始，IBM MQ XMS .NET 客户机支持您借助项目模板开发 XMS .NET Core 应用程序。

开始之前

您的系统上必须具有 Microsoft Visual Studio 2017 或更高版本以及 .NET Core 2.1。

您必须将 XMS .NET 模板从

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip

目录复制到

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

目录，其中：

- *&MQ_INSTALL_ROOT* 是安装的根目录
- *&USER_HOME_DIRECTORY* 是主目录。

您必须停止并重新启动 Microsoft Visual Studio 以选取模板。

关于此任务

XMS .NET 项目模板包含一些可用于帮助您开发应用程序的通用代码。 通过内置代码，您可以连接到 IBM MQ 队列管理器，并且只需修改内置代码中的属性即可执行放置或获取操作。

过程

1. 打开 Microsoft Visual Studio。
2. 单击**文件**，接着单击**新建**和**项目**。
3. 在 " 创建新项目" 窗口中，选择 IBM XMS .NET Client App (.NET Core)，然后单击 **下一步**。
4. 在配置您的新项目窗口中，更改项目的项目名称（如果需要），然后单击**创建**以创建 XMS .NET 项目。

   XMSDotnetApp.cs 是随项目文件一起创建的文件。 该文件包含连接到队列管理器并执行发送和接收操作的代码。

   连接属性将设置为缺省值：

   - WMQ_CONNECTION_NAME_LIST 将设置为 *localhost(1414)*
   - XMSC.WMQ_CHANNEL 将设置为 *DOTNET.SVRCONN*

   队列将设置为 *Q1*，您可以相应地修改这些属性。
5. 编译并运行应用程序。

相关概念
IBM MQ 组件和功能
.NET 应用程序运行时（仅限 Windows）

线程技术模型

一般规则可控制多线程应用程序如何使用 XMS 对象。

- 只能在不同线程上同时使用以下类型的对象：
  - ConnectionFactory
  - Connection
  - ConnectionMetaData
  - Destination
- 在任何时候，都只能在单个线程上使用 Session 对象。

这些规则的例外情况由 IBM Message Service Client for .NET 参考中方法的接口定义中标记为“线程上下文”的条目指示。

ConnectionFactories 和 Connection 对象

ConnectionFactory 对象提供一种模板以供应用程序用于创建 Connection 对象。 应用程序可使用 Connection 对象来创建 Session 对象。

对于 .NET，XMS 应用程序首先使用 XMSFactoryFactory 对象获取对所需协议类型对应的 ConnectionFactory 对象的引用。 然后，该 ConnectionFactory 对象会建立仅适用于该协议类型的连接。

XMS 应用程序可以创建多个连接，而多线程应用程序可在多个线程上同时使用同一个 Connection 对象。 Connection 对象用于封装应用程序与消息传递服务器之间的通信连接。

连接可提供多种用途：

- 在应用程序创建连接时，可认证该应用程序。
- 应用程序可将唯一的客户机标识与连接相关联。 客户机标识用于支持发布/预订域中的持久预订。 可通过以下两种方式设置客户机标识：

  分配连接客户机标识的首选方法是使用属性在特定于客户机的 ConnectionFactory 对象中进行配置，并采用透明方式将该标识分配给其创建的连接。

  分配客户机标识的另一种方法是使用 Connection 对象上设置的特定于提供者的值。 该值不会覆盖由管理人员配置的标识。 在管理人员未指定标识的情况下，可提供该值。 如果管理人员指定的标识确实存在，那么尝试使用特定于提供者的值覆盖该值会导致抛出异常。 如果应用程序要显式设置标识，那么必须在创建连接之后且在连接上执行其他操作之前立即执行该操作，否则会抛出异常。

XMS 应用程序通常会创建一个连接、一个或多个会话，以及大量的消息生产者和消息使用者。

创建连接需要使用大量的系统资源，因为这不仅要建立通信连接，还要认证应用程序。

连接启动和停止方式

可以在启动或停止方式下运行连接。

应用程序创建连接时，该连接处于停止方式。 如果连接处于停止方式，那么应用程序可以初始化会话，可以发送消息但无法接收消息（同步或异步）。

应用程序可以通过调用 `Start Connection` 方法来启动连接。 如果连接处于启动方式，那么应用程序可以发送和接收消息。 应用程序随后可通过调用 Stop Connection 和 `Start Connection` 方法来停止和重新启动连接。

关闭连接

应用程序可通过调用 Close Connection 方法来关闭连接。 在应用程序关闭连接时，XMS 会执行以下操作：

- 关闭与连接相关联的所有会话，并删除与这些会话相关联的某些对象。 有关删除哪些对象的更多信息，请参阅第 536 页的『删除对象』。 同时，XMS 会回滚这些会话中当前正在执行的所有事务。
- 结束与消息传递服务器的通信连接。
- 释放连接所使用的内存及其他内部资源。

在关闭连接之前，对于在会话期间确认失败的消息，XMS 不会确认收到这类消息。 有关确认收到消息的更多信息，请参阅第 528 页的『消息确认』。

异常处理

XMS .NET 异常都派生自 System.Exception。 有关更多信息，请参阅第 543 页的『.NET 中的错误处理』。

连接到服务集成总线

XMS 应用程序可以通过使用直接 TCP/IP 连接或使用“基于 TCP/IP 的 HTTP”来连接到 WebSphere Application Server 服务集成总线。

HTTP 协议可在无法使用直接 TCP/IP 连接的情况下使用。 一种常见的情况是通过防火墙进行通信，例如，当两个企业交换消息时。 使用 HTTP 利用防火墙通信通常称为 *HTTP* 隧道。 但是，与使用直接 TCP/IP 连接相比，HTTP 隧道的速度要慢一些，因为 HTTP 头会显著增加传输的数据量，而且 HTTP 协议需要的通信量也多于 TCP/IP。

要创建 TCP/IP 连接，应用程序可以使用其 XMSC_WPM_TARGET_TRANSPORT_CHAIN 属性设置为 XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC 的连接工厂。 这是该属性的缺省值。 如果成功创建连接，那么该连接的 XMSC_WPM_CONNECTION_PROTOCOL 属性设置为 XMSC_WPM_CP_TCP。

要创建使用 HTTP 的连接，应用程序必须使用其 XMSC_WPM_TARGET_TRANSPORT_CHAIN 属性设置为入站传输链名称（即配置为使用 HTTP 传输通道）的连接工厂。 如果成功创建连接，那么该连接的 XMSC_WPM_CONNECTION_PROTOCOL 属性设置为 XMSC_WPM_CP_HTTP。 有关如何配置传输链的信息，请参阅 WebSphere Application Server 产品文档中的 配置传输链 。

应用程序在连接到引导程序服务器时可使用类似的通信协议选项。 连接工厂的 XMSC_WPM_PROVIDER_ENDPOINTS 属性是引导程序服务器的一个或多个端点地址的序列。 每个端点地址的引导程序传输链组件可以是 XMSC_WPM_BOOTSTRAP_TCP（针对到引导程序服务器的 TCP/IP 连接）或 XMSC_WPM_BOOTSTRAP_HTTP（针对使用 HTTP 的连接）。

会话

会话是用于发送和接收消息的单线程上下文。

应用程序可以使用会话来创建消息、消息生产者、消息使用者、队列浏览器和临时目标。 应用程序还可以使用会话来运行本地事务。

应用程序可以创建多个会话，其中每个会话独立于其他会话来生成和使用消息。 如果不同会话（甚至是同一个会话）中的两个消息使用者预订同一个主题，那么每个消息使用者都会收到该主题上任何已发布消息的副本。

与 Connection 对象不同，不能在不同的线程中同时使用 Session 对象。 只可以通过 Session 对象所使用线程以外的其他线程来调用 Session 对象的 Close Session 方法。 Close Session 方法将结束会话，并释放已分配给该会话的所有系统资源。

如果应用程序必须在多个线程上同时处理消息，那么应用程序必须在每个线程上创建一个会话，然后在该线程上使用该会话执行任何发送或接收操作。

事务性会话

XMS 应用程序可以运行本地事务。 本地事务只涉及更改应用程序连接到的队列管理器或服务集成总线的资源。

仅当应用程序连接到 IBM MQ 队列管理器或 WebSphere Application Server 服务集成总线时，此主题中的信息才有用。 这些信息不适用于与代理程序的实时连接。

要运行本地事务，应用程序必须首先通过调用 Connection 对象的 Create Session 方法并使用参数将会话指定为事务性来创建事务性会话。 然后，会话内发送和接收的所有消息将分组到事务序列中。 应用程序提交或回滚事务开始后所发送和接收的消息时，表明事务结束。

要落实事务，应用程序可调用 Session 对象的 Commit 方法。 提交事务时，事务中发送的所有消息可用于发送到其他应用程序，而事务内收到的所有消息将得到确认，这样一来消息传递服务器便不会尝试再次将这些消息发送到应用程序。 在点到点域中，消息传递服务器还会从队列中移除收到的消息。

要回滚事务，应用程序可调用 Session 对象的 Rollback 方法。 回滚事务时，消息传递服务器将废弃事务内发送的所有消息，而事务内收到的所有消息可用于再次发送。 在点到点域中，之前收到的消息将放回队列并可再次被其他应用程序看到。

在应用程序创建事务性会话或调用 Commit 或 Rollback 方法时，会自动启动新的事务。 因此，事务性会话总是有活动事务。

当应用程序关闭事务性会话时，会发生隐式回滚。 当应用程序关闭连接时，所有连接的事务性会话将发生隐式回滚。

事务完全包含在事务性会话中。 事务不能跨越多个会话。 这表示一个应用程序不能在两个或多个事务性会话中发送和接收消息然后作为单个事务提交或回滚所有操作。

相关概念

消息确认
每个非事务性会话都有一种确认模式，用于确定如何确认应用程序收到的消息。 共有三种确认模式可用，确认模式的选择会影响应用程序的设计。

消息传递
XMS 支持消息传递的持久和非持久方式，以及消息的异步和同步传递。

消息确认

每个非事务性会话都有一种确认模式，用于确定如何确认应用程序收到的消息。 共有三种确认模式可用，确认模式的选择会影响应用程序的设计。

仅当应用程序连接到 IBM MQ 队列管理器或 WebSphere Application Server 服务集成总线时，此主题中的信息才有用。 这些信息不适用于与代理程序的实时连接。

XMS 使用相同的机制来确认收到 JMS 使用的消息。

如果会话为非事务性，那么确认应用程序收到的消息的方法取决于该会话的确认模式。 以下段落描述了这三种确认模式：

XMSC_AUTO_ACKNOWLEDGE
会话自动确认应用程序收到的每个消息。

如果消息同步发送到应用程序，会话会在每次 Receive 调用成功完成时确认收到消息。

如果应用程序成功收到消息，但存在故障阻止进行确认，那么该消息将变为可重新发送。 因此，应用程序必须能够处理重新传递的消息。

XMSC_DUPS_OK_ACKNOWLEDGE
会话在进行选择时确认应用程序收到消息。

使用此确认模式可减少会话必须执行的工作量，但是阻止消息确认的故障可能会导致多个消息变为可重新发送。 因此，应用程序必须能够处理重新传递的消息。

XMSC_CLIENT_ACKNOWLEDGE
应用程序通过调用 Message 类的 Acknowledge 方法来确认它所收到的消息。

应用程序可以单独地确认收到每个消息，或者可以接收一批消息并只为收到的最后一条消息调用 Acknowledge 方法。 如果调用 Acknowledge 方法，自上次调用此方法以来收到的所有消息都将得到确认。

与以上任一确认模式配合使用，应用程序可以通过调用 Session 类的 Recover 方法停止并重新启动会话中的消息发送。 将重新传递先前未确认接收的消息。 但是，这些消息的发送顺序可能会与之前的发送顺序不同。 在此期间，优先级较高的消息可能已到达，而某些原始消息可能已到期。 在点到点域中，某些原始消息可以已被另一个应用程序使用。

应用程序可以通过检查消息的 JMSRedelivered 头字段的内容来确定消息是否会重新发送。 应用程序通过调用 Message 类的 Get JMSRedelivered 方法来实现此目标。

相关概念

事务性会话
XMS 应用程序可以运行本地事务。 本地事务只涉及更改应用程序连接到的队列管理器或服务集成总线的资源。

消息传递
XMS 支持消息传递的持久和非持久方式，以及消息的异步和同步传递。

消息传递

XMS 支持消息传递的持久和非持久方式，以及消息的异步和同步传递。

消息传递方式

XMS 支持两种消息传递方式：

持久
持久消息都会传递一次。 消息传递服务器会采取特殊的预防措施（例如，记录消息），以确保在传输过程中不会丢失持久消息，即使在发生故障的情况下也是如此。

非持久
非持久消息最多传递一次。 与持久消息相比，非持久消息的可靠性差一些，因为可能由于发生故障而导致在传输过程中丢失非持久消息。

可通过权衡可靠性与性能来选择合适的传递方式。 通常，非持久消息比持久消息传输得更快。

异步消息传递

XMS 使用一个线程来处理会话的所有异步消息传递。 这意味着每次只能运行一个消息侦听器函数或一个 `onMessage()` 方法。

如果会话中的多个消息使用者正在异步接收消息，并且消息侦听器函数或 `onMessage()` 方法正在将消息传递给某个消息使用者，那么等待同一消息的其他消息使用者必须继续等待。 等待传递给会话的其他消息也必须继续等待。

如果应用程序需要同时传递消息，请创建多个会话，以便 XMS 能够使用多个线程来处理异步消息传递。 通过使用这种方式，可同时运行多个消息侦听器函数或 `onMessage()` 方法。

无法通过将消息侦听器分配给使用者来使会话异步。 只有在调用 `Connection.Start` 方法时，才能使会话异步。 在调用 `Connection.Start` 方法之前，允许所有同步调用。 在调用 `Connection.Start` 时，会开始将消息传递给使用者。

如果必须在异步会话上进行同步调用 (例如，创建使用者或生产者)，那么必须调用 `Connection.Stop` 。 可以通过调用 `Connection.Start` 方法来恢复会话，以开始传递消息。 这里的唯一例外情况是 Session 消息传递线程（即将消息传递到回调函数的线程）。 这个线程可以在消息回调函数中对会话发出任何调用（Close 调用除外）。

注: 在非受管方式下，回调函数中的 MQDISC 调用不受 IBM MQ .NET 客户机支持。 因此，客户机应用程序无法在异步接收方式下在 MessageListener 回调内创建或关闭会话。 可在 `MessageListener` 方法外创建和处置会话。

同步消息传递

如果应用程序使用 MessageConsumer 对象的 Receive 方法，那么会将消息同步传递到该应用程序。

通过使用 Receive 方法，应用程序可以在指定的时间段内等待消息，也可以无限期地等待。 或者，如果应用程序不想等待消息，那么可以使用“Receive with No Wait”方法。

相关概念

事务性会话
XMS 应用程序可以运行本地事务。 本地事务只涉及更改应用程序连接到的队列管理器或服务集成总线的资源。

消息确认
每个非事务性会话都有一种确认模式，用于确定如何确认应用程序收到的消息。 共有三种确认模式可用，确认模式的选择会影响应用程序的设计。

目标

XMS 应用程序可使用 Destination 对象来指定所发送消息的目标以及所接收消息的源。

XMS 应用程序可以在运行时创建 Destination 对象，也可以从受管对象存储库中获取预定义的目标。

对于 ConnectionFactory，XMS 应用程序指定目标的最灵活方式是将其定义为受管对象。 通过使用此方法，用 C、C++、.NET 语言以及 Java 编写的应用程序可以共享目标的定义。 可以在不更改任何代码的情况下更改受管 Destination 对象的属性。

对于 .NET 应用程序，可以使用 CreateTopic 或 CreateQueue 方法来创建目标。 .NET API 中的 ISession 和 XMSFactoryFactory 对象中都提供了这两个方法。 有关更多信息，请参阅第 542 页的『.NET 中的目标』和../com.ibm.mq.ref.dev.doc/sapidest.dita#sapidest。

主题统一资源标识

主题统一资源标识 (URI) 指定主题的名称，还可以指定主题的一个或多个属性。

主题 URI 以序列 topic:// 开头，后跟主题名称以及（可选）用于设置其余主题属性的名称/值对列表。 主题名称不能为空。

以下示例显示了 .NET 代码片段：

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

有关主题的属性（包括可以在 URI 中使用的名称和有效值）的更多信息，请参阅目标的属性。

在指定要在预订中使用的主题 URI 时，可以使用通配符。 这些通配符的语法取决于连接类型和代理程序版本；提供了以下选项：

- 具有字符级别通配符格式的 IBM WebSphere MQ 7.0 队列管理器
- 具有主题级别通配符格式的 IBM WebSphere MQ 7.0 队列管理器
- WebSphere Application Server 服务集成总线

具有字符级别通配符格式的 IBM WebSphere MQ 7.0 队列管理器

具有字符级别通配符格式的 IBM WebSphere MQ 7.0 队列管理器使用以下通配符：

* 表示 0 个或更多个字符
? 表示 1 个字符
% 表示转义字符

第 530 页的表 81 提供了一些有关如何使用此通配符方案的示例。

表 *81:* 对 *IBM WebSphere MQ 7.0* 队列管理器使用字符级别通配符方案的示例 *URI*

统一资源标识	匹配项	示例
"topic://Sport*Results"	以“Sport”开头且以“Results”结束的所有主题	"topic://SportsResults" 和 "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport?Results"	以“Sport”开头、后跟单个字符、再后跟“Results”的所有主题	"topic://SportsResults" 和 "topic://SportXResults"
"topic://Sport/*ball*/Div?/Results/*/???"	主题	"topic://Sport/Football/Div1/Results/2002/Nov" 和 "topic://Sport/Netball/National/Div3/Results/02/Jan"

具有主题级别通配符格式的 IBM WebSphere MQ 7.0 队列管理器

具有主题级别通配符格式的 IBM WebSphere MQ 7.0 队列管理器使用以下通配符：

可匹配多个级别
+ 可匹配单个级别

第 531 页的表 82 提供了一些有关如何使用此通配符方案的示例。

表 *82:* 对 *IBM WebSphere MQ 7.0* 队列管理器使用主题级别通配符方案的示例 *URI*

统一资源标识	匹配项	示例
"topic://Sport/+/Results"	在 Sport 与 Results 之间具有单个分层级别名称的所有主题	"topic://Sport/Football/Results" 和 "topic://Sport/Ju-Jitsu/Results"
"topic://Sport/#/Results"	以“Sport/”开头且以“/Results”结束的所有主题	"topic://Sport/Football/Results" 和 "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football/#"	以“Sport/Football/”开头的所有主题	"topic://Sport/Football/Results" 和 "topic://Sport/Football/TeamNews/Signings/Managerial"

WebSphere Application Server 服务集成总线

WebSphere Application Server 服务集成总线使用以下通配符：

* 可与层次结构中一个级别上的任意字符匹配
// 可与 0 个或更多个级别匹配
//. 可与 0 个或更多个级别匹配（位于主题表达式末尾）

第 531 页的表 83 提供了一些有关如何使用此通配符方案的示例。

表 *83:* 对 *WebSphere Application Server* 服务集成总线使用通配符方案的示例 *URI*

统一资源标识	匹配项	示例
"topic://Sport/*ball/Results"	在 Sport 与 Results 之间具有以“ball”结尾的单个分层级别名称的所有主题	"topic://Sport/Football/Results" 和 "topic://Sport/Netball/Results"
"topic://Sport//Results"	以“Sport/”开头且以“/Results”结束的所有主题	"topic://Sport/Football/Results" 和 "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	以“Sport/Football/”开头的所有主题	"topic://Sport/Football/Results" 和 "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/*ball//Results//."	主题	"topic://Sport/Football/Results" 和 "topic://Sport/Netball/National/Div3/Results/2002/November"

相关概念

队列统一资源标识
队列 URI 指定队列的名称，还可以指定队列的一个或多个属性。

临时目标
XMS 应用程序可以创建和使用临时目标。

队列统一资源标识

队列 URI 指定队列的名称，还可以指定队列的一个或多个属性。

队列的 URI 以序列 queue:// 开头，后跟队列的名称；它还可能包含用于设置其余队列属性的“名称/值”对列表。

对于 IBM MQ 队列（但不针对 WebSphere Application Server 缺省消息传递提供程序队列），可以在队列前指定队列所在的队列管理器，使用 / 将队列管理器名称与队列名称分隔开。

如果指定队列管理器，那么对于使用该队列的连接而言，它必须是 XMS 直接连接到的队列管理器，或者必须可通过该队列进行访问。 远程队列管理器仅支持从队列中检索消息，而不支持将消息放入队列中。 有关完整详细信息，请参阅 IBM MQ 队列管理器文档。

如果未指定任何队列管理器，那么可以选择使用额外的 / 分隔符，它存在与否对于队列定义没有影响。

以下队列定义都等价于名为 QM_A 的队列管理器上名为 QB 的 IBM MQ 队列， XMS 直接连接到该队列管理器:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

相关概念

主题统一资源标识
主题统一资源标识 (URI) 指定主题的名称，还可以指定主题的一个或多个属性。

临时目标
XMS 应用程序可以创建和使用临时目标。

临时目标

XMS 应用程序可以创建和使用临时目标。

应用程序通常使用临时目标来接收请求消息的应答。 要指定将请求消息的应答发送到的目标，应用程序可调用表示请求消息的 Message 对象的 Set JMSReplyTo 方法。 该调用上指定的目标可以是临时目标。

虽然是使用会话来创建临时目标，但是临时目标的作用域实际上是用于创建该会话的连接。 连接的任何会话都可以为临时目标创建消息生产者和消息使用者。 临时目标会一直保留，直至将其显式删除或连接终止（以先发生者为准）。

在应用程序创建临时队列时，将在应用程序所连接到的消息传递服务器中创建队列。 如果应用程序连接到队列管理器，那么将根据模型队列创建动态队列，该模型队列的名称由 XMSC_WMQ_TEMPORARY_MODEL 属性指定，而用于构成动态队列的名称的前缀由 XMSC_WMQ_TEMP_Q_PREFIX 属性指定。 如果应用程序连接到服务集成总线，那么将在总线中创建临时队列，并由 XMSC_WPM_TEMP_Q_PREFIX 属性指定用于构成临时队列名称的前缀。

当连接到服务集成总线的应用程序创建临时主题时，用于构成临时主题的名称的前缀由 XMSC_WPM_TEMP_TOPIC_PREFIX 属性指定。

相关概念

主题统一资源标识
主题统一资源标识 (URI) 指定主题的名称，还可以指定主题的一个或多个属性。

队列统一资源标识
队列 URI 指定队列的名称，还可以指定队列的一个或多个属性。

消息生产者

在 XMS 中，可以创建消息生产者（具有有效目标或无相关目标）。 在使用空目标创建消息生产者时，需要在发送消息时指定有效目标。

具有相关目标的消息生产者

在此场景中，将使用有效目标来创建消息生产者。 在发送操作期间，无需指定目标。

无相关目标的消息生产者

在 XMS .NET 中，可使用空目标创建消息生产者。

要在使用 .NET API 时创建无相关目标的消息生产者时，必须将 NULl 作为参数传递到 ISession 对象的 `CreateProducer()` 方法（例如 `session.CreateProducer(null)`）。 但是，在发送消息时必须指定有效目标。

消息使用者

消息使用者可以分为持久和非持久订户以及同步和异步消息使用者。

持久订户

持久订户是接收主题上所有已发布消息（包括当订户处于不活动状态时发布的消息）的消息使用者。

仅当应用程序连接到 IBM MQ 队列管理器或 WebSphere Application Server 服务集成总线时，此主题中的信息才有用。 这些信息不适用于与代理程序的实时连接。

要为主题创建持久订户，应用程序将调用 Session 对象的 Create Durable Subscriber 方法，并将用于标识持久预订的名称和表示主题的 Destination 对象指定为参数。 应用程序可以使用也可以不使用消息选择器来创建持久订户，并且可以指定持久订户是否接收自己的连接所发布的消息。

用于创建持久订户的会话必须具有相关的客户机标识。 客户机标识与用于创建会话的连接的相关标识相同；其指定方式如 第 526 页的『ConnectionFactories 和 Connection 对象』中所述。

用于标识持久预订的名称在客户机标识中必须唯一，因此客户机标识将构成完整且唯一的持久预订标识的一部分。 消息传递服务器将维护持久预订的记录，并确保一直保留主题上发布的所有消息，直至持久订户确认这些消息或这些消息到期。

甚至在持久订户关闭之后，消息传递服务器仍将继续维护持久预订的记录。 要复用先前创建的持久预订，应用程序必须创建一个持久订户，方法是指定与持久预订相关的预订名称并使用具有与持久预订相关的客户机标识的会话。 每个会话针对特殊的持久预订每次只能有一个持久订户。

持久预订的作用域是负责维护预订记录的消息传递服务器。 如果连接到不同消息传递服务器的两个应用程序使用相同的预订名称和客户机标识各自创建了一个持久订户，那么将创建两个完全独立的持久预订。

要删除持久预订，应用程序可调用 Session 对象的 Unsubscribe 方法，并将用于标识持久预订的名称指定为参数。 与会话相关的客户机标识必须与持久预订的相关标识相同。 消息传递服务器将删除其维护的持久预订记录，并且不会向持久订户再发送任何消息。

要更改现有预订，应用程序可以使用相同的预订名称和客户机标识但指定不同的主题和/或消息选择器来创建持久订户。 更改持久预订等效于删除预订并创建新预订。

对于连接到 IBM WebSphere MQ 7.0 或更高版本队列管理器的应用程序，XMS 将管理订户队列。 因此，应用程序无需指定订户队列。 如果指定了订户队列，XMS 将忽略该订户队列。

请注意，您不能更改持久预订的订户队列。 更改订户队列的唯一方式是删除预订并创建新预订。

对于连接到服务集成总线的应用程序，每个持久订户都必须具有指定的主持久预订。 要为使用同一个连接的所有持久订户指定主持久预订，请设置用于创建连接的 ConnectionFactory 对象的 XMSC_WPM_DUR_SUB_HOME 属性。 要为个别主题指定主持久预订，请设置表示该主题的 Destination 对象的 XMSC_WPM_DUR_SUB_HOME 属性。 必须先为连接指定持久预订宿主，然后应用程序才能创建使用此连接的持久订户。 为目标指定的任何值将覆盖为连接指定的值。

非持久订户

非持久订户是仅接收在订户处于活动状态时发布的消息的消息使用者。 在订户处于不活动状态时传递的消息将会丢失。

仅当通过 IBM WebSphere MQ 6.0 队列管理器使用发布/预订消息传递时，此主题中的信息才有用。

如果未在关闭连接之前或期间删除使用者对象，那么消息可以保留在不再活动的订户的代理程序队列中。

在此情况下，可以使用“用于 JMS 的 IBM WebSphere MQ classes for JMS 类”随附的清除实用程序来清除队列中的这些消息。 *IBM WebSphere MQ* 使用 *Java* 中提供了有关如何使用此实用程序的详细信息。 如果订户队列中保留了大量消息，那么可能还需要增加该队列的队列深度。

同步和异步消息使用者

同步消息使用者从队列同步接收消息，异步消息使用者从队列异步接收消息。

同步消息使用者

同步消息使用者每次接收一条消息。 如果使用 `Receive(wait interval)` 方法，那么此调用仅在指定的时间段（以毫秒计）内等待消息或者直至关闭消息使用者为止。

如果使用 ReceiveNoWait() 方法，那么同步消息使用者会立即接收消息而无任何延迟；如果下一条消息可用，将立即接收该消息，否则将返回指向空 Message 对象的指针。

异步消息使用者

只要队列中有可用的新消息，就会调用由应用程序注册的消息侦听器。

XMS 中的有害消息

有害消息是指接收 MDB 应用程序无法处理的消息。 如果遇到有害消息，XMS MessageConsumer 对象可以根据队列属性 BOQUEUE 和 BOTHRESH 将其重新排队。

在某些情况下，传送到 MDB 的消息可能会回滚到 IBM MQ 队列上。 例如，如果在工作单元内传递消息，随后进行回滚，那么可能会发生此情况。 通常，将会再次传递已回滚的消息，但格式错误的消息可能会不断地导致 MDB 发生故障，从而无法进行传递。 此类消息称为有害消息。 您可以配置 IBM MQ，以便将有害消息自动传输到另一个队列以供进一步调查或丢弃。 有关如何使用此方式配置 IBM MQ 的信息，请参阅在 ASF 中处理有害消息。

有时队列中会收到格式不正确的消息。 这种情况下，格式错误表示接收方应用程序无法正确处理此消息。 此类消息可能会导致接收方应用程序故障并回退此格式错误的消息。 该消息随后将重复传递到输入队列并被应用程序重复回退。 这些消息称为有害消息。 XMS MessageConsumer 对象将检测有害消息并将其重新发送到备用目标。

IBM MQ 队列管理器会记录每个消息的回退次数。 当该次数达到配置的阈值时，消息使用者会将消息重新排入指定回退队列。 如果重新排队因为任何原因失败，该消息将从输入队列中移除或重新排入死信队列，或被废弃。

XMS ConnectionConsumer 对象使用相同的队列属性以相同的方式处理有害消息。 如果有多个连接使用者监视同一队列，可能是重新排队发生之前中毒消息发送到应用程序的次数已超过阈值。 此行为是因为单个连接使用者监视队列和对中毒消息重新排队的方式。

回退队列的阈值和名称是 IBM MQ 队列的属性。 属性名称为 BackoutThreshold 和 BackoutRequeueQName。 它们适用于的队列如下所示：

- 对于点到点消息传递，这是底层本地队列。 在消息使用者和连接使用者使用队列别名时，这非常重要。
- 对于 IBM MQ 消息传递提供程序正常方式下的发布/预订消息传递，它是用于创建 Topic 受管队列的模型队列。
- 对于 IBM MQ 消息传递提供程序迁移模式中的发布/预订消息传递，适用队列为 TopicConnectionFactory 对象中定义的 CCSUB 队列，或者 Topic 主题上定义的 CCDSUB 队列。

要设置 BackoutThreshold 和 BackoutRequeueQName 属性，请发出以下 MQSC 命令：

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

对于发布/预订消息传递，如果系统为每个预订创建动态队列，那么将从 IBM MQ classes for JMS 模型队列 SYSTEM.JMS.MODEL.QUEUE。 要更改这些设置，请使用：

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

如果回退阈值是 0，那么将禁用有害消息处理，并将有害消息保留在输入队列中。 否则，回退计数值达到阈值时，消息将被发送到指定的回退队列。

如果回退计数达到阈值，但是消息无法转到回退队列中，那么会将该消息发送到死信队列，或者如果该消息是非持久消息，那么会将其丢弃。

如果没有定义回退队列，或者如果 MessageConsumer 对象无法将消息发送至回退队列，会出现此情况。

配置系统以执行有害消息处理

XMS .NET 在查询 **BOTHRESH** 和 **BOQNAME** 属性时所使用的队列取决于所执行消息传递的样式：

- 对于点到点消息传递，这是底层本地队列。 当 XMS .NET 应用程序使用来自别名队列或集群队列的消息时，这一点十分重要。
- 对于发布/预订消息传递，会创建受管队列来保存应用程序的消息。 XMS .NET 会查询受管队列以确定 **BOTHRESH** 和 **BOQNAME** 属性的值。

  受管队列是根据与应用程序预订的“主题”对象所关联的模型队列创建的，并从该模型队列继承 **BOTHRESH** 和 **BOQNAME** 属性的值。 使用的模型队列取决于接收应用程序已取得持久预订还是非持久预订。

  – 用于持久预订的模型队列由“主题”的 **MDURMDL** 属性来指定。 该属性的缺省值为 SYSTEM.DURABLE.MODEL.QUEUE。
  – 对于非持久预订，所使用的模型队列由 **MNDURMDL** 属性来指定。 **MNDURMDL** 属性的缺省值为 SYSTEM.NDURABLE.MODEL.QUEUE。

查询 **BOTHRESH** 和 **BOQNAME** 属性时，XMS .NET 会执行以下操作：

- 打开本地队列或别名队列的目标队列。
- 查询 **BOTHRESH** 和 **BOQNAME** 属性。
- 关闭本地队列或别名队列的目标队列。

打开本地队列或别名队列的目标队列时所使用的打开选项，取决于所使用的 IBM MQ 的版本：

- 对于 IBM MQ 9.1.0 Fix Pack 4 和更低版本的 Long Term Support， 和 IBM MQ 9.1.4 以及更低版本的 Continuous Delivery， 和更低版本，如果本地队列或别名队列的目标队列是集群队列，那么 XMS .NET 使用 MQOO_INPUT_AS_Q_DEF， MQOO_INQUIRE 和 MQOO_FAIL_IF_QUIESCING 选项打开队列。 这意味着运行接收应用程序的用户必须对集群队列的本地实例具有查询和获取访问权。

  XMS .NET 会使用 MQOO_INQUIRE 和 MQOO_FAIL_IF_QUIESCING 打开选项来打开所有其他类型的本地队列。 为了使 XMS .NET 能够查询属性值，运行接收应用程序的用户必须对本地队列具有查询访问权。
- V 9.1.5 V 9.1.0.5 从 IBM MQ 9.1.5 和 IBM MQ 9.1.0 Fix Pack 5 使用 XMS .NET 时，运行接收应用程序的用户必须具有对本地队列的查询访问权，而不考虑队列的类型。

要将有害消息移至回退重排队列或队列管理器的死信队列，必须授予运行应用程序 put 和 passall 权限的用户。

在 *ASF* 中处理有害消息
使用应用程序服务器工具 (ASF) 时，ConnectionConsumer 而不是 MessageConsumer 会处理有害消息。 ConnectionConsumer 会根据队列的 BackoutThreshold 和 BackoutRequeueQName 属性对消息重新排队。

应用程序使用 ConnectionConsumer 时，回退消息的环境取决于应用程序服务器提供的会话。

- 会话为具有 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE 的非事务性会话时，仅会在发生系统错误后或应用程序意外终止的情况下回退消息。
- 如果会话是带有 CLIENT_ACKNOWLEDGE 的非事务性会话，那么可通过应用程序服务器调用 Session.recover() 来回退未确认的消息。

  通常，MessageListener 的客户机实现或应用程序服务器将调用 Message.acknowledge()。 Message.acknowledge() 会确认到目前为止在会话上传递的所有消息。
- 如果会话是事务性会话，那么可通过应用程序服务器调用 Session.rollback() 来回退未确认的消息。

队列浏览器

应用程序使用队列浏览器浏览而不移除队列中的消息。

要创建队列浏览器，应用程序可调用 ISession 对象的 Create Queue Browser 方法，并将用于标识要浏览队列的 Destination 对象指定为参数。 应用程序可以使用也可以不使用消息选择器来创建队列浏览器。

在创建队列浏览器之后，应用程序可调用 IQueueBrowser 对象的 GetEnumerator 方法以获取队列中消息的列表。 此方法将返回用于封装 Message 对象列表的枚举符。 列表中 Message 对象的顺序与从队列中检索消息的顺序相同。 然后，应用程序可以使用该枚举符来依次浏览各条消息。

在将消息放入队列中以及从队列中除去消息时会动态更新该枚举符。 每次应用程序调用 IEnumerator.MoveNext() 来浏览队列上的下一条消息时，该消息会反映队列的当前内容。

针对给定的队列浏览器，应用程序可以多次调用 GetEnumerator 方法。 每个调用都会返回一个新枚举符。 因此，应用程序可以使用多个枚举符来浏览队列中的消息，并维护队列内的多个位置。

应用程序可以使用队列浏览器来搜索要从队列中移除的适当消息，然后将消息使用者与消息选择器一起用于移除该消息。 消息选择器可以根据 JMSMessageID 头字段的值来选择消息。 有关该头字段及其他 JMS 消息头字段的信息，请参阅 第 554 页的『XMS 消息中的头字段』。

请求程序

应用程序使用请求程序来发送请求消息，然后等待接收应答。

许多消息传递应用程序都基于用于发送请求消息并等待应答的算法。 XMS 提供一个名为 Requestor 的类，以帮助开发这种类型的应用程序。

要创建请求程序，应用程序可调用 Requestor 类的 Create Requestor 构造函数，并将用于标识请求消息要发送到何处的 Session 对象和 Destination 对象指定为参数。 此会话不能是事务性会话，也不能使用确认方式 XMSC_CLIENT_ACKNOWLEDGE。 构造函数将自动创建要将应答消息发送到的临时队列或主题。

在创建请求程序之后，应用程序可调用 Requestor 对象的 Request 方法来发送请求消息，然后等待接收来自负责接收请求消息的应用程序的应答。 此调用将一直等待，直至收到应答或该会话结束（以先发生者为准）。 针对每个请求消息，请求程序只需要一个应答。

在应用程序关闭请求程序后，将删除临时队列或主题。 但是，不会关闭相关会话。

删除对象

在应用程序删除其创建的 XMS 对象时，XMS 会释放已分配给该对象的内部资源。

在应用程序创建 XMS 对象时，XMS 会为该对象分配内存和其他内部资源。 XMS 将一直保留这些内部资源，直至应用程序通过调用该对象的 close 或 delete 方法来显式删除该对象，届时 XMS 会释放这些内部资源。 如果应用程序尝试删除已删除的对象，那么将忽略此调用。

在应用程序删除 Connection 或 Session 对象时，XMS 会自动删除某些相关对象并释放其内部资源。 这些是由 Connection 或 Session 对象创建的对象，并且其功能都依赖于该对象。 第 536 页的表 84 中显示了这些对象。

注: 如果应用程序关闭与被依赖会话的连接，那么还会删除依赖于这些会话的所有对象。 只有 Connection 或 Session 对象可以具有从属对象。

表 *84:* 自动删除的对象

已删除的对象	**方法**	**自动删除的从属对象**
Connection	关闭连接	ConnectionMetaData 和 Session 对象
Session	关闭会话	MessageConsumer、MessageProducer、QueueBrowser 和 Requestor 对象

通过 XMS 使用受管的 IBM MQ XA 事务

可以通过 XMS 使用受管 IBM MQ XA 事务。

要通过 XMS 使用 XA 事务，必须创建事务性会话。 当使用 XA 事务时，事务控制是通过分布式事务协调程序 (DTC) 全局事务进行的，而不是通过 XMS 会话进行的。 使用 XA 事务时，无法在 XMS 会话上发出 `Session.commit` 或 `Session.rollback`。 请改为使用 `Transscope.Commit` 或 `Transscope.Rollback` DTC 方法来落实或回滚事务。 如果对 XA 事务使用会话，那么使用该会话创建的生产者或使用者必须是 XA 事务的一部分。 它们不能用于 XA 事务作用域外的任何操作。 它们不能用于 XA 事务外的操作，如 `Producer.send` 或 `Consumer.receive`。

如果出现以下情况，将抛出 IllegalStateException 异常对象：

- XA 事务性会话用于 Session.commit 或 Session.rollback。
- 在 XA 事务作用域外使用曾在 XA 事务性会话中使用的生产者或使用者对象。

在异步使用者中不支持 XA 事务。

注：

1. 在落实 XA 事务之前，可以在 Producer、Consumer、Session 或 Connection 对象上发出 close 调用。 在这样的情况下，将回滚事务中的消息。 同样，如果在 XA 事务落实之前连接中断，那么也会回滚事务中的所有消息。 对于 Producer 对象，回滚意味着不将消息放入队列中。 对于 Consumer 对象，回滚意味着消息保留在队列中。
2. 如果 Producer 对象将具有 TimeToLive 的消息放入 TransactionScope，并且在该时间过后发出 commit，那么该消息可能会在发出 commit 之前到期。 在此情况下，该消息将不可用于 Consumer 对象。
3. 在线程中不支持 Session 对象。 不支持将事务与在线程间共享的 Session 对象一起使用。

XMS 基本类型

XMS 提供了 8 种 Java 基本类型（byte、short、int、long、float、double、char 和 boolean）的等效项。 这样便可以在 XMS 与 JMS 之间交换消息而不丢失或损坏数据。

第 537 页的表 85 列出了每个 XMS 基本类型的 Java 等效数据类型，大小以及最小值和最大值。

表 *85: XMS* 数据类型及其 *Java* 等效项

XMS 数据类型	**与 Java 兼容的数据类型**	**大小**	**最小值：**	**最大值：**
System.Boolean	布尔值	32 位	false	true
System.SBYTE	byte	8 位	-2^7 (-128)	2^7-1 (127)
System.BYTE	byte	8 位	-2^7 (-128)	2^7-1 (127)
System.CHAR	byte	8 位	-2^7 (-128)	2^7-1 (127)
System.Int16	短整型	16 位	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 位	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	long	64 位	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	浮点值	32 位	-3.402823E-38（精度为 7 位数）	3.402823E+38（精度为 7 位数）
System.Double	双精度值	64 位	-1.79769313486231E-308（精度为 15 位数）	1.79769313486231E+308（精度为 15 位数）

属性值从一种数据类型到另一种数据类型的隐式转换

在应用程序获取某个属性的值后，XMS 可以将该值转换为另一种数据类型。 许多规则可控制受支持的转换以及 XMS 执行转换的方式。

对象属性具有名称和值；值具有相关的数据类型，属性的值也称为属性类型。

应用程序使用 PropertyContext 类的方法来获取和设置对象的属性。 要获取某个属性的值，应用程序可调用适用于此属性类型的方法。 例如，要获取整数属性的值，应用程序通常会调用 GetIntProperty 方法。

但是，在应用程序获取某个属性的值后，XMS 可以将该值转换为另一种数据类型。 例如，要获取整数属性的值，应用程序可以调用 GetStringProperty 方法以将此属性的值作为字符串返回。 XMS 支持的转换显示在第 538 页的表 86 中。

表 *86:* 从属性类型到其他数据类型的受支持转换	
属性类型	**受支持的目标数据类型**
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte 数组	System.String
System.Int16	System.String, System.Int32, System.Int64

以下一般规则可控制受支持的转换：

- 如果在转换期间未丢失数据，那么可以将数字属性值从一种数据类型转换为另一种数据类型。 例如，可以将数据类型为 System.Int32 的属性值转换为数据类型为 System.Int64 的值，但不能将其转换为数据类型为 System.Int16 的值。
- 可以将任何数据类型的属性值转换为字符串。
- 可以将字符串属性值转换为任何其他数据类型，前提是已针对转换正确格式化了该字符串。 如果应用程序尝试转换未正确格式化的字符串属性值，那么 XMS 可能会返回错误。
- 如果应用程序尝试不受支持的转换，那么 XMS 可能会返回错误。

将属性值从一种数据类型转换为另一种数据类型时，以下规则适用：

- 将布尔属性值转换为字符串时，值 true 将转换为字符串 “true”，值 false 将转换为字符串 “false”。
- 在将布尔属性值转换为数字数据类型（包括 System.SByte）时，会将值 true 转换为 1，并将值 false 转换为 0。
- 将字符串属性值转换为布尔值时，字符串 “true”（不区分大小写）或 “1” 将转换为 true，字符串 “false”（不区分大小写）或 “0” 将转换为 false。 无法转换所有其他字符串。
- 在将字符串属性值转换为数据类型为 System.Int32、System.Int64、System.SByte 或 System.Int16 的值时，该字符串必须采用以下格式：

  [*blanks*][*sign*] *digits*

  此字符串的各个组成部分定义如下：

  blanks
  可选前导空白字符

  sign
  可选的加号 (+) 或减号 (-) 字符。

  digits
  连续的数字字符序列 (0-9)。 必须至少存在一个数字字符。

  在数字字符序列之后，该字符串可包含其他非数字字符，但是在到达这些字符中的第一个字符后会立即停止转换。 假设该字符串表示十进制整数。

  如果未正确格式化该字符串，那么 XMS 可能会返回错误。
- 在将字符串属性值转换为数据类型为 System.Double 或 System.Float 的值时，该字符串必须采用以下格式：

[*blanks*][*sign*][*digits*][*point*[*d_digits*]][*e_char*[*e_sign*]*e_digits*]

此字符串的各个组成部分定义如下：

blanks
（可选）前导空格字符。

sign
（可选）加号 (+) 或减号 (-) 字符。

digits
连续的数字字符序列 (0-9)。 *digits* 或 *d_digits* 中必须至少存在一个数字字符。

point
（可选）小数点 (.)。

d_digits
连续的数字字符序列 (0-9)。 *digits* 或 *d_digits* 中必须至少存在一个数字字符。

e_char
阶符，为 *E* 或 *e*。

e_sign
（可选）该指数的加号 (+) 或减号 (-) 字符。

e_digits
该指数的连续数字字符序列 (0-9)。 如果该字符串包含指数字符，那么必须至少存在一个数字字符。

在数字字符序列或表示指数的可选字符之后，该字符串可包含其他非数字字符，但是在到达这些字符中的第一个字符后会立即停止转换。 假设该字符串表示十进制浮点数，指数幂为 10。

如果未正确格式化该字符串，那么 XMS 可能会返回错误。

- 在将数字属性值转换为字符串（包括数据类型为 System.SByte 的属性值）时，会将该值转换为字符串表示形式（即十进制数字），而不是转换为包含该值的 ASCII 字符的字符串。 例如，整数 65 将转换为字符串 “65”，而不是字符串 “A”。
- 在将字节数组属性值转换为字符串时，会将每个字节转换为表示该字节的 2 个十六进制字符。 例如，字节数组 {0xF1, 0x12, 0x00, 0xFF} 将转换为字符串 “F11200FF”。

Property 和 PropertyContext 类的方法支持从属性类型转换为其他数据类型。

迭代器

迭代器封装了一个对象列表和一个用于维护列表中当前位置的光标。 IBM Message Service Client for C/C++ 中提供的迭代器的概念是通过在 IBM Message Service Client for .NET 中使用 IEnumerator 接口实现的。

在创建迭代器时，光标的位置位于第一个对象之前。 应用程序使用迭代器来依次检索每个对象。

IBM Message Service Client for C/C++ 的 Iterator 类等同于 Java 中的 Enumerator 类。 IBM Message Service Client for .NET 类似于 Java 并使用 IEnumerator 接口。

应用程序可以使用 IEnumerator 来执行以下任务：

- 获取消息的属性
- 获取映射消息主体中的名称/值对
- 浏览队列中的消息
- 获取连接支持的由 JMS 定义的消息属性的名称

编码字符集标识

在 XMS .NET 中，所有字符串均使用本机 .NET 字符串进行传递。 由于这采用固定编码，所以不需要其他信息即可解释该字符串。 因此，XMS .NET 应用程序不需要 XMSC_CLIENT_CCSID 属性。

XMS 错误和异常代码

XMS 可使用各种错误代码来指示故障。 本文档并未明确列出这些错误代码，因为它们可能因发行版而异。 仅记录 XMS 异常代码（格式为 XMS_X_...），因为它们在 XMS 的各发行版中保持相同。

通过 XMS 自动重新连接 IBM MQ 客户机

配置 XMS 客户机以在将 IBM WebSphere MQ 7.1 客户机和更高版本用作消息提供程序时，在发生网络，队列管理器或服务器故障后自动重新连接。

可使用 MQConnectionFactory 类的 WMQ_CONNECTION_NAME_LIST 和 WMQ_CLIENT_RECONNECT_OPTIONS 属性来将客户机配置为自动重新连接。 自动客户机重新连接功能可在发生连接故障后重新连接客户机，也可以作为一个选项在停止队列管理器后使用。 某些客户机应用程序的设计可能导致其不适合于自动重新连接。

可自动重新连接的客户机连接在建立连接后变为可重新连接。

注: 也可以通过客户机通道定义表 (CCDT) 或通过 mqclient.ini 文件启用客户机重新连接来设置属性**客户机重新连接选项**、**客户机重新连接超时**和**连接名称列表**。

注: 如果在 ConnectionFactory 对象上以及在 CCDT 中设置了重新连接属性，那么优先规则如下。 如果在 ConnectionFactory 对象上设置了“连接名称列表”属性的缺省值，那么优先采用 CCDT。 如果未将“连接名称列表”属性设置为其缺省值，那么优先采用 ConnectionFactory 对象上设置的属性值。 “连接名称列表”属性的缺省值为 localhost(1414)。

编写 XMS .NET 应用程序

本部分中的主题所提供的信息可帮助您编写 XMS .NET 应用程序。

关于此任务

本部分提供了与编写 XMS .NET 应用程序有关的信息。 有关编写 XMS 应用程序的常规信息，请参阅第 524 页的『编写 XMS 应用程序』。

本部分包含以下主题:

- 第 540 页的『.NET 的数据类型』
- 第 541 页的『.NET 中的受管和非受管操作』
- 第 542 页的『.NET 中的目标』
- 第 542 页的『.NET 中的属性』
- 第 543 页的『在 .NET 中处理不存在的属性』
- 第 543 页的『.NET 中的错误处理』
- 第 543 页的『在 .NET 中使用消息和异常侦听器』

.NET 的数据类型

XMS .NET 支持 System.Boolean、System.Byte、System.SByte、System.Char、System.String、System.Single、System.Double、System.Decimal、System.Int16、System.Int32、System.Int64、System.UInt16、System.UInt32、System.UInt64 和 System.Object。 XMS .NET 的数据类型与 XMS C/C++ 的数据类型有所不同。您可以使用本主题来标识相应的数据类型。

下表显示了对应的 XMS .NET 和 XMS C/C++ 数据类型及其简要描述。

表 *87: XMS .NET* 和 *XMS C/C++* 的数据类型

XMS .NET 类型	**XMS C/C++ 类型**	**描述**
System.SByte	xmsSBYTE xmsINT8	有符号的 8 位值
System.Byte	xmsBYTE xmsUINT8	无符号的 8 位值

表 87: *XMS .NET* 和 *XMS C/C++* 的数据类型 *(继续)*		
XMS .NET 类型	**XMS C/C++ 类型**	**描述**
System.Int16	xmsINT16 xmsSHORT	有符号的 16 位值
System.UInt16	xmsUINT16 xmsUSHORT	无符号的 16 位值
System.Int32	xmsINT32 xmsINT	有符号的 32 位值
System.UInt32	xmsUINT32 xmsUINT	无符号的 32 位值
System.Int64	xmsLONG xmsINT64	有符号的 64 位值
System.UInt64	xmsULONG xmsUINT64	无符号的 64 位值
System.Char	xmsCHAR16	无符号的 16 位字符（针对 .NET 的 Unicode）
System.Single	xmsFLOAT	IEEE 32 位浮点数
System.Double	xmsDOUBLE	IEEE 64 位浮点数
System.Boolean	xmsBOOL	True/False 值
不适用	xmsCHAR	有符号或无符号的 8 位值（有符号或无符号取决于平台）
System.Decimal	不适用	96 位有符号的整数倍（10^0 到 10^{28}）
System.Object	不适用	所有类型的基础
System.String	不适用	字符串类型

.NET 中的受管和非受管操作

受管代码专门在 .NET 通用语言运行时环境中执行，并且完全依赖于该运行时提供的服务。 如果应用程序的任何部分运行或调用 .NET 通用语言运行时环境外部的服务，那么此应用程序归类为非受管应用程序。

受管 .NET 环境中当前不支持某些高级功能。

如果应用程序需要使用完全受管环境中当前不支持的某些功能，那么可以将应用程序更改为使用非受管环境，而无需对应用程序进行实质性更改。 但请注意，做出此选择后，XMS 堆栈将使用非受管代码。

与 IBM MQ 队列管理器的连接

与 WMQ_CM_CLIENT 的受管连接不支持非 TCP 通信和通道压缩。 但是，可通过非受管连接 (WMQ_CM_CLIENT_UNMANAGED) 来支持这些连接。 有关更多信息，请参阅 开发 .NET 应用程序。

如果通过非受管环境中的受管对象来创建连接工厂，那么必须手动将连接方式的值更改为 XMSC_WMQ_CM_CLIENT_UNMANAGED。

与 WebSphere Application Server 服务集成总线消息传递引擎的连接

对于需要使用 SSL 协议（包括 HTTPS）的与 WebSphere Application Server 服务集成总线消息传递引擎的连接，当前不支持将其用作受管代码。

.NET 中的目标

在 .NET 中，可根据协议类型创建目标，而且只能在创建目标的协议类型上使用该目标。

提供了两个用于创建目标的函数：一个用于主题，另一个用于队列：

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

API 中的以下两个对象上提供了这两个函数：

- ISession
- XMSFactoryFactory

在这两种情况下，这些方法都可以接受 URI 样式字符串，它可以包含以下格式的参数：

```
"topic://some/topic/name?priority=5"
```

或者，这些方法只能接受目标名称（即不带 topic:// 或 queue:// 前缀且不含参数的名称）。

这意味着将使用以下 URI 样式字符串：

```
CreateTopic("topic://some/topic/name");
```

将生成与以下目标名称相同的结果：

```
CreateTopic("some/topic/name");
```

对于 WebSphere Application Server 服务集成总线 JMS，还可以指定简短格式的主题，其中包括 *topicname* 和 *topicspace*，但不能包含参数：

```
CreateTopic("topicspace:topicname");
```

.NET 中的属性

.NET 应用程序使用 PropertyContext 接口中的方法来获取和设置对象的属性。

PropertyContext 接口封装了用于获取和设置属性的方法。 以下类直接或间接继承了这些方法：

- BytesMessage
- 连接
- ConnectionFactory
- ConnectionMetaData
- 目标
- MapMessage
- 消息
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Session

- StreamMessage
- TextMessage

如果应用程序设置了某个属性的值，那么新值将替换此属性的旧值。

有关 XMS 属性的更多信息，请参阅 XMS 对象属性。

为便于使用，XMS 中的 XMS 属性名称和值已预定义为 XMSC struct 中的公共常量。 这些常量的名称采用 XMSC.*constant* 格式；例如，XMSC.USERID（属性名称常量）和 XMSC.DELIVERY_AS_APP（值常量）。

另外，您可以使用 IBM.XMS.MQC struct 访问 IBM MQ 常量。 如果已导入 IBM.XMS 名称空间，那么可以访问这些属性的值（采用格式 MQC.*constant*）。 例如，MQC.MQRO_COA_WITH_FULL_DATA。

此外，如果您有一个混合应用程序同时将 XMS .NET 和 IBM MQ 类用于 .NET，并同时导入两个 IBM.XMS 和 IBM.WMQ 名称空间，那么必须完全限定 MQC struct 名称空间，以确保每次出现都是唯一的。

受管 .NET 环境中当前不支持某些高级功能。 请参阅 第 541 页的『.NET 中的受管和非受管操作』，以获取更多详细信息。

在 .NET 中处理不存在的属性

XMS .NET 中针对不存在属性的处理方式与 JMS 规范大体一致，也与 XMS 的 C 和 C++ 实现一致。

在 JMS 中，当方法尝试将不存在的（空）值转换为所需类型时，访问不存在的属性可能会导致 Java 系统异常。 如果属性不存在，那么会发生以下异常：

- getStringProperty 和 getObjectProperty 返回空值
- getBooleanProperty 返回 false，因为 Boolean.valueOf(null) 返回 false
- getIntProperty 等抛出 java.lang.NumberFormatException，因为 Integer.valueOf(null) 抛出异常

如果属性不存在于 XMS .NET 中，那么会发生以下异常：

- GetStringProperty 和 GetObjectProperty（和 GetBytesProperty）返回空值（与 Java 相同）
- GetBooleanProperty 抛出 System.NullReferenceException
- GetIntProperty 等抛出 System.NullReferenceException

此实现与 Java 不同，但与 JMS 规范大体一致，并且与 XMS C 和 C++ 接口一致。 与 Java 实现相似，XMS .NET 会将任何异常从 System.Convert 调用传播到调用者。 但与 Java 不同，XMS 会通过将空值传递到系统转换例程来显式抛出 NullReferenceExceptions，而不仅仅使用 .NET 框架的本机行为。 如果应用程序将属性设置为诸如 “abc” 之类的字符串并调用 GetIntProperty，那么 Convert.ToInt32("abc") 抛出的 System.FormatException 将传播到调用者（这与 Java 一致）。 只有当用于 setProperty 和 getProperty 的类型不兼容时，才会抛出 MessageFormatException。 此行为也与 Java 一致。

.NET 中的错误处理

XMS .NET 异常都派生自 System.Exception。 XMS 方法调用可以抛出特定的 XMS 异常（如 MessageFormatException）、常规 XMSException 或系统异常（如 NullReferenceException）。

编写应用程序，以根据应用程序需求在特定的 catch 块或常规 `System.Exception` catch 块中捕获所有这些错误。

在 .NET 中使用消息和异常侦听器

.NET 应用程序使用消息侦听器异步接收消息，并使用异常侦听器异步通知连接问题。

关于此任务

对于 .NET 和 C++，消息和异常侦听器的功能是相同的。但是，存在一些小的实现差异。

过程

- 要设置消息侦听器以异步接收消息，请完成以下步骤：

a) 定义与消息侦听器委托的特征符匹配的方法。
您定义的方法可以是静态方法，也可以是实例方法，并且可以在任何可访问类中进行定义。 委托特征符如下所示：

```
public delegate void MessageListener(IMessage msg);
```

因此，您可以将方法定义为：

```
void SomeMethodName(IMessage msg);
```

b) 使用类似于以下示例的内容，将此方法实例化为委托：

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

c) 通过设置为使用者的 MessageListener 属性，向一个或多个使用者注册该委托：

```
consumer.MessageListener = OnMsgMethod;
```

您可以通过将 MessageListener 重新设置为 null 来移除该委托：

```
consumer.MessageListener = null;
```

- 要设置异常侦听器，请完成以下步骤。
异常侦听器的工作方式与消息侦听器大致相同，它具有不同的委托定义，并且分配给连接而不是分配给消息使用者。 这与 C++ 相同。
a) 定义方法。
委托特征符如下所示：

```
public delegate void ExceptionListener(Exception ex);
```

因此，定义的方法可以是：

```
void SomeMethodName(Exception ex);
```

b) 使用类似于以下示例的内容，将此方法实例化为委托：

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

c) 通过设置 ExceptionListener 属性来向连接注册该委托：

```
connection.ExceptionListener = OnExMethod ;
```

您可以按如下方式重新设置 ExceptionListener 来移除该委托：

```
null:  connection.ExceptionListener = null;
```

使用 XMS .NET 受管对象

本部分中的主题提供了有关受管对象的信息。 XMS 应用程序可以从中央受管对象存储库中检索对象定义，并使用这些定义来创建连接工厂和目标。

关于此任务

本部分提供的信息可帮助创建和管理受管对象，并描述了 XMS 支持的受管对象储存库类型。 本部分还说明了 XMS 应用程序如何与受管对象存储库建立连接以检索所需的受管对象。

本部分包含以下主题：

- 第 545 页的『XMS .NET 支持的受管对象存储库的类型』
- 第 545 页的『受管对象的 XMS .NET 属性映射』
- 第 547 页的『XMS .NET 需要的受管 ConnectionFactory 对象的属性』
- 第 548 页的『XMS .NET 需要的受管 Destination 对象的属性』
- 第 548 页的『XMS .NET 创建受管对象』
- 第 549 页的『XMS .NET 创建 InitialContext 对象』
- 第 549 页的『XMS .NET InitialContext 属性』
- 第 549 页的『XMS 初始上下文的 URI 格式』
- 第 550 页的『XMS .NET 的 JNDI 查询 web 服务』
- 第 551 页的『XMS .NET 检索受管对象』

XMS .NET 支持的受管对象存储库的类型

“文件系统”和 LDAP 受管对象可用于连接到 IBM MQ 和 WebSphere Application Server，而“COS 命名”只能用于连接到 WebSphere Application Server。

“文件系统”对象目录可采用序列化的 Java Naming Directory Interface (JNDI) 对象形式。 LDAP 对象目录是包含 JNDI 对象的目录。 可使用 JMSAdmin 工具（随 IBM WebSphere MQ 6.0 一起提供）或 IBM MQ Explorer（随 IBM WebSphere MQ 7.0 和更高版本一起提供）来管理“文件系统”和 LDAP 对象目录。 文件系统和 LDAP 对象目录都可用于通过集中 IBM WebSphere MQ 连接工厂和目标来管理客户机连接。 网络管理员可以部署引用了同一中央存储库的多个应用程序，这些应用程序将自动进行更新以反映中央存储库中的连接设置更改。

“COS 命名”目录包含 WebSphere Application Server service integration bus 连接工厂和目标，并且可使用 WebSphere Application Server 管理控制台进行管理。 如果 XMS 应用程序需要从“COS 命名”目录中检索对象，那么必须部署 JNDI 查找 Web Service。 此 Web Service 并不适用于所有 WebSphere Application Server service integration technologies。 请参阅产品文档以了解详细信息。

注: 重新启动应用程序连接，使该对象目录的更改生效。

受管对象的 XMS .NET 属性映射

要使 XMS .NET 应用程序能够使用 IBM MQ JMS 和 WebSphere Application Server 连接工厂和目标对象定义，必须将从这些定义中检索的属性映射到可以在 XMS 连接工厂和目标上设置的相应 XMS 属性。

例如，要创建具有从 IBM MQ JMS 连接工厂检索的属性的 XMS 连接工厂，必须在两者之间映射这些属性。

将自动执行所有属性映射。

第 545 页的表 88 演示连接工厂和目的地的一些最常见属性之间的映射。 此表中显示的属性只是一小部分示例，并非所有这些属性都适用于所有连接类型和服务器。

表 *88:* 连接工厂和目标属性的名称映射示例

IBM MQ JMS 属性名称	**XMS 属性名称**	**WebSphere Application Server service integration bus 属性名称**
PERSISTENCE (PER)	XMSC_DELIVERY_MODE	
EXPIRY (EXP)	XMSC_TIME_TO_LIVE	
PRIORITY (PRI)	XMSC_PRIORITY	

表 *88:* 连接工厂和目标属性的名称映射示例 *(继续)*		
IBM MQ JMS 属性名称	**XMS 属性名称**	**WebSphere Application Server service integration bus 属性名称**
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

注: 第 546 页的表 89 中显示的属性可用于 JMS 以及 XMS .NET。

表 *89: XMS .NET* 属性					
属性	**对象类型**				
	CF	**QCF**	**TCF**	**队列**	**Topic**
APPLICATIONNAME	Y	Y	Y	N/A	N/A
ASYNCEXCEPTION	Y	Y	Y	N/A	N/A
CCDTURL	Y	Y	Y	N/A	N/A
CHANNEL	Y	Y	Y	N/A	N/A
CONNECTIONNAMELIST	Y	Y	Y	N/A	N/A
CLIENTRECONNECTOPTIONS	Y	Y	Y	N/A	N/A
CLIENTRECONNECTTIMEOUT	Y	Y	Y	N/A	N/A
CLIENTID	N/A	Y	N/A	N/A	N/A
COMPHDR 第 547 页的『1』	Y	N/A	Y	N/A	N/A
COMPMSG 第 547 页的『1』	Y	Y	Y	N/A	N/A
CONNOPT 第 547 页的『1』	Y	Y	Y	N/A	N/A
CONNTAG 第 547 页的『1』	Y	Y	Y	N/A	N/A
DESCRIPTION 第 547 页的『1』	N/A	Y	N/A	Y	Y
到期第 547 页的『1』	N/A	N/A	N/A	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
HOSTNAME	N/A	Y	N/A	N/A	N/A
LOCALADDRESS	N/A	Y	N/A	N/A	N/A
PERSISTENCE	N/A	N/A	N/A	Y	Y

表 89: XMS .NET 属性 (继续)					
属性	**对象类型**				
	CF	**QCF**	**TCF**	**队列**	**Topic**
PORT	N/A	Y	N/A	N/A	N/A
优先级第 547 页的『1』	N/A	N/A	N/A	Y	Y
PROVIDERVERSION 第 547 页的『1』	N/A	Y	N/A	N/A	N/A
QMANAGER	Y	Y	Y	Y	N/A
队列第 547 页的『1』	N/A	N/A	N/A	Y	N/A
SHARECONVALLOWED	Y	Y	Y	N/A	N/A
主题第 547 页的『1』	N/A	N/A	N/A	N/A	Y
Transport 第 547 页的『1』	N/A	Y	N/A	N/A	N/A

注:

1. 这些属性没有应用程序级别属性，但可以选择使用受管属性进行设置。

XMS .NET 需要的受管 ConnectionFactory 对象的属性

应用程序创建连接工厂时，必须定义一些属性来与消息传递服务器建立连接。

以下表格中列出的属性是应用程序与消息传递服务器建立连接的最低设置要求。 如果要定制用于建立连接的方式，那么应用程序可以根据需要来设置 ConnectionFactory 对象的任何其他属性。 有关更多信息，请参阅 ConnectionFactory 属性。 其中提供了可用属性的完整列表。

与 IBM MQ 队列管理器的连接

表 90: 用于与 IBM MQ 队列管理器建立连接的受管 ConnectionFactory 对象的属性设置	
必需 XMS 属性	**等效的必需 IBM MQ JMS 属性**
XMSC_CONNECTION_TYPE	XMS 通过连接工厂类名和 TRANSPORT (TRAN) 属性来确定此项。
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	端口
XMSC_WMQ_QUEUE_MANAGER	队列管理器的名称

与代理程序的实时连接

表 91: 用于与代理程序建立实时连接的受管 ConnectionFactory 对象的属性设置	
必需 XMS	**等效的必需 IBM MQ JMS 属性**
XMSC_CONNECTION_TYPE	XMS 通过连接工厂类名和 TRANSPORT (TRAN) 属性来确定此项。

表 *91:* 用于与代理程序建立实时连接的受管 *ConnectionFactory* 对象的属性设置 *(*继续*)*	
必需 XMS	**等效的必需 IBM MQ JMS 属性**
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	端口

与 WebSphere Application Server service integration bus 的连接

表 *92:* 用于与 *WebSphere Application Server service integration bus* 建立连接的受管 *ConnectionFactory* 对象的属性设置	
XMS 属性	**描述**
XMSC_CONNECTION_TYPE	与应用程序相连的消息传递服务器的类型。。这可通过连接工厂类名称来确定。
XMSC_WPM_BUS_NAME	对于连接工厂，这是应用程序连接到的服务集成总线的名称；对于目标，这是存在目标的服务集成总线的名称。

XMS .NET 需要的受管 Destination 对象的属性

用于创建目标的应用程序必须在受管 Destination 对象上设置多个属性。

表 *93:* 受管 *Destination* 对象的属性设置		
连接类型	**属性**	**描述**
IBM MQ 队列管理器	QUEUE (QU) TOPIC (TOP)	要连接到的队列 应用程序用作目标的主题
与代理程序的实时连接	TOPIC (TOP)	应用程序用作目标的主题
WebSphere Application Server service integration bus	topicName queueName	如果应用程序正在连接到主题 如果应用程序正在连接到队列

XMS .NET 创建受管对象

必须使用相应的管理工具来创建 XMS 应用程序与消息传递服务器建立连接时所需的 ConnectionFactory 和 Destination 对象定义。

开始之前

有关 XMS 支持的不同类型的受管对象存储库的更多详细信息，请参阅第 545 页的『XMS .NET 支持的受管对象存储库的类型』。

关于此任务

要创建 IBM MQ 的受管对象，请使用 IBM MQ Explorer 或 IBM MQ JMS 管理 (JMSAdmin) 工具。

要创建 IBM MQ 或 IBM Integration Bus 的受管对象，请使用 IBM MQ JMS 管理 (JMSAdmin) 工具。

要创建 WebSphere Application Server service integration bus 的受管对象，请使用 WebSphere Application Server 管理控制台。

V 9.1.2 在管理工具中，该属性简称为 **APPLICATIONNAME** 或 **APPNAME**。

注: 您不能使用 JMSAdmin 设置 TRANSPORT(UNMANAGED)。 因此，要使用以管理方式选择的应用程序名称获取不受管的 XMS 客户机，您需要输入以下命令：

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

以下步骤概括了在创建受管对象时要执行的操作。

过程

1. 创建连接工厂并定义必需的属性，以创建应用程序到所选服务器的连接。
   在 第 547 页的『XMS .NET 需要的受管 ConnectionFactory 对象的属性』中定义了 XMS 建立连接所需的最低属性。
2. 在应用程序连接到的消息传递服务器上创建必需的目标：
   - 对于与 IBM MQ 队列管理器的连接，请创建队列或主题。
   - 对于与代理程序的实时连接，请创建主题。
   - 对于与 WebSphere Application Server service integration bus 的连接，请创建队列或主题。

   在 第 548 页的『XMS .NET 需要的受管 Destination 对象的属性』中定义了 XMS 建立连接所需的最低属性。

XMS .NET 创建 InitialContext 对象

应用程序必须创建用于与受管对象存储库建立连接的初始上下文，以便检索所需的受管对象。

关于此任务

InitialContext 对象封装了与该存储库的连接。 XMS API 提供了用于执行以下任务的方法：

- 创建 InitialContext 对象
- 在受管对象存储库中查找受管对象。

过程

- 有关创建 InitialContext 对象的更多详细信息，请参阅 InitialContext for .NET 和 InitialContext 的属性。

XMS .NET InitialContext 属性

InitialContext 构造函数的参数中包括受管对象存储库的位置（其指定为统一资源指示符 (URI)）。 如果要使应用程序与该存储库建立连接，那么需要提供的信息可能要多于 URI 中包含的信息。

在 JNDI 和 XMS 的 .NET 实现中，将在环境 Hashtable 中向构造函数提供其他信息。

受管对象存储库的位置是在 XMSC_IC_URL 属性中定义的。 通常会将该属性传递给 Create 调用，但可以修改该属性以在查找之前连接到其他命名目录。 对于 FileSystem 或 LDAP 上下文，该属性将定义该目录的地址。 对于 COS 命名，这是使用这些属性连接到 JNDI 目录的 Web Service 的地址。

以下属性在不做修改的情况下传递给 Web Service，后者将使用这些属性来连接到 JNDI 目录。

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

XMS 初始上下文的 URI 格式

将采用统一资源指示符 (URI) 形式提供受管对象存储库的位置。 URI 的格式取决于上下文类型。

FileSystem 上下文

对于 FileSystem 上下文，URL 提供基于文件系统的目录的位置。 URL 的结构由 RFC 1738“统一资源定位符 *(URL)*”定义：URL 具有前缀 `file://`，此前缀后面的语法是在运行 XMS 的系统上可打开的文件的有效定义。

该语法可特定于平台，并且可使用“/”或“\”分隔符。 如果使用“\”，那么需要使用额外的“\”来转义每个分隔符。 这可防止 .NET 框架尝试将分隔符解释为其后面内容的转义字符。

以下示例说明了该语法：

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

LDAP 上下文

对于 LDAP 上下文，URL 的基本结构由 RFC 2255“*LDAP URL* 格式”定义，它具有不区分大小写的前缀 `ldap://`。

以下示例说明了精确的语法：

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

此语法由该 RFC 定义，但不支持任何属性、作用域、过滤器或扩展。

此语法的示例包括：

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

WSS 上下文

对于 WSS 上下文，URL 采用 Web Service 端点形式，它具有前缀 `http://`。

也可以使用前缀 `cosnaming://` 或 `wsvc://`。

这两个前缀可解释为以下含义：您正在使用可通过 http 访问其 URL 的 WSS 上下文，这样便可直接从 URL 轻松派生出初始上下文类型。

此语法的示例包括：

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

XMS .NET 的 JNDI 查询 web 服务

要从 XMS 访问 COS 命名目录，必须在 WebSphere Application Server service integration bus 服务器上部署 JNDI Lookup Web Service。 此 Web Service 可将来自 COS 命名服务的 Java 信息转换为 XMS 应用程序可读格式。

该 Web Service 以企业归档文件 SIBXJndiLookupEAR.ear（位于安装目录中）形式提供。 对于当前 IBM Message Service Client for .NET 发行版，SIBXJndiLookupEAR.ear 位于 *install_dir*`\java\lib` 目录中。 可使用管理控制台或 wsaadmin 脚本编制工具，将该 Web Service 安装在 WebSphere Application Server service integration bus 服务器中。 请参阅产品文档，以获取有关部署 Web Service 应用程序的更多信息。

要在 XMS 应用程序中定义该 Web Service，只需将 InitialContext 对象的 XMSC_IC_URL 属性设置为 Web Service 端点 URL。 例如，如果在名为 MyServer 的服务器主机上部署该 Web Service，那么 Web Service 端点 URL 示例如下：

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

通过设置 XMSC_IC_URL 属性，InitialContext 查找调用可以在定义的端点上调用该 Web Service，进而从 COS 命名服务中查找所需的受管对象。

.NET 应用程序可以使用该 Web Service。 XMS C、/C++ 和 XMS .NET 的服务器端部署相同。XMS .NET 通过 Microsoft .NET Framework 直接调用 Web Service。

XMS .NET 检索受管对象

XMS 使用在创建 InitialContext 对象时提供的地址或 InitialContext 属性中的地址，从存储库中检索受管对象。

要检索的对象可能具有以下类型的名称：

- 用于描述 Destination 对象的简单名称（例如，名为 SalesOrders 的队列目标）
- 复合名称（可能由 SubContext 构成，用“/”分隔，且必须以对象名结尾）。 复合名称的示例是 “Warehouse/PickLists/DispatchQueue2”（其中 Warehouse 和 Picklists 是命名目录中的 SubContext，DispatchQueue2 是 Destination 对象的名称）。

防止应用程序使用较新的 XMS 版本

缺省情况下，安装较新的 XMS 版本时，使用先前版本的应用程序会自动切换到较新版本，而无需重新编译。 但是，通过在应用程序配置文件中设置属性，可以防止应用程序使用较新的版本。

关于此任务

“多版本共存”功能可确保安装较新的 XMS 版本时不会覆盖先前的 XMS 版本。 相似 XMS .NET 组合件的多个实例共存于全局组合件高速缓存 (GAC) 中，但具有不同的版本号。 在内部，GAC 使用策略文件将应用程序调用传递到最新版本的 XMS。 应用程序无需重新编译即可运行，并且可以使用较新的 XMS .NET 版本中提供的新功能。

过程

- 如果要求某个应用程序使用之前 XMS .NET 版本，请在应用程序配置文件中将 `publisherpolicy` 属性设置为 no。

  注: 应用程序配置文件的名称包含相关可执行程序的名称且后缀为 .config。 例如，text.exe 的应用程序配置文件的名称为 text.exe.config。

  但是，在任何时候，系统的所有应用程序都使用相同版本的 XMS .NET。

确保 XMS 应用程序安全通信

本部分介绍了如何设置安全通信以使 XMS 应用程序能够通过安全套接字层 (SSL) 连接到 WebSphere Application Server service integration bus 消息传递引擎或 IBM MQ 队列管理器。

关于此任务

本部分包含以下主题：

- 第 552 页的『与 IBM MQ 队列管理器的安全连接』
- 第 552 页的『XMS 到 IBM MQ 队列管理器的连接的 CipherSuite 和 CipherSpec 名称映射』
- 第 553 页的『与 WebSphere Application Server service integration bus 消息传递引擎的安全连接』
- 第 553 页的『与 WebSphere Application Server service integration bus 的连接的 CipherSuite 和 CipherSpec 名称映射』

与 IBM MQ 队列管理器的安全连接

要使 XMS .NET 应用程序能够与 IBM MQ 队列管理器建立安全连接，必须在 ConnectionFactory 对象中定义相关属性。

在加密协商中使用的协议可以是安全套接字层 (SSL) 或传输层安全性 (TLS)，这取决于您在 ConnectionFactory 对象中指定的 CipherSuite。

如果您使用 IBM WebSphere MQ 7.0.0 Fix Pack 1 和更高版本的客户机库并连接到 IBM WebSphere MQ 7.0 队列管理器，那么可以在 XMS 应用程序中与同一队列管理器建立多个连接。 但是，不允许与不同的队列管理器建立连接。 如果您尝试这样做，那么会收到 MQRC_SSL_ALREADY_INITIALIZED 错误。

如果您使用 IBM WebSphere MQ 6.0 和更高版本的客户机库，那么只有在先关闭任何先前的 SSL 连接时才可以创建 SSL 连接。 不允许在同一个进程中与相同或不同的队列管理器建立多个并发 SSL 连接。 如果您尝试多个请求，那么会收到警告 MQRC_SSL_ALREADY_INITIALIZED，这意味着已忽略 SSL 连接的某些请求参数。

下表显示了通过 SSL 与 IBM MQ 队列管理器建立的连接的 ConnectionFactory 属性以及简短描述。

表 *94:* 通过 *SSL* 与 *IBM MQ* 队列管理器建立的连接的 *ConnectionFactory* 属性

属性的名称	**描述**
XMSC_WMQ_SSL_CERT_STORES	用于保存与队列管理器的 SSL 连接上使用的证书撤销列表 (CRL) 的服务器的位置。
XMSC_WMQ_SSL_CIPHER_SPEC	到队列管理器的安全连接上要使用的 CipherSpec 名称。
XMSC_WMQ_SSL_CIPHER_SUITE	到队列管理器的 TLS 连接上要使用的 CipherSuite 的名称。 协商安全连接时使用的协议取决于指定的 CipherSuite。
XMSC_WMQ_SSL_CRYPTO_HW	下面是连接到客户机系统的加密硬件的配置详细信息。
XMSC_WMQ_SSL_FIPS_REQUIRED	该属性的值用于确定应用程序能否使用符合非 FIPS 标准的密码套件。 如果将该属性设置为 true，那么客户机/服务器连接只能使用 FIPS 算法。
XMSC_WMQ_SSL_KEY_REPOSITORY	用于存储密钥和证书的密钥数据库文件的位置。
XMSC_WMQ_SSL_KEY_RESETCOUNT	KeyResetCount 表示在重新协商密钥之前在 SSL 对话期间发送和接收的未加密字节总数。
XMSC_WMQ_SSL_PEER_NAME	到队列管理器的 SSL 连接上要使用的对等方名称。

XMS 到 *IBM MQ* 队列管理器的连接的 *CipherSuite* 和 *CipherSpec* 名称映射

InitialContext 可在 JMSAdmin 连接工厂属性 SSLCIPHERSUITE 与 XMS 近似等效项 XMSC_WMQ_SSL_CIPHER_SPEC 之间进行转换。 如果指定了 XMSC_WMQ_SSL_CIPHER_SUITE 的值但省略了 XMSC_WMQ_SSL_CIPHER_SPEC 的值，那么需要进行类似转换。

第 552 页的表 95 列出了可用的 CipherSpec 及其 JSSE CipherSuite 等效项。

表 *95:* 可用的 *CipherSpec* 及其 *JSSE CipherSuite* 等效项

CipherSpec	**等效的 JSSE CipherSuite**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

注: 不推荐 TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。

与 WebSphere Application Server service integration bus 消息传递引擎的安全连接

要使 XMS .NET 应用程序能够与 WebSphere Application Server service integration bus 消息传递引擎建立安全连接，必须在 ConnectionFactory 对象中定义相关属性。

XMS 为与 WebSphere Application Server service integration bus 的连接提供 SSL 和 HTTPS 支持。 SSL 和 HTTPS 提供安全连接以便进行认证和确保机密性。

与 WebSphere 安全性相似，XMS 安全性是根据 JSSE 安全标准和命名约定进行配置，包括使用 CipherSuite 指定在协商安全连接时使用的算法。 在加密协商中使用的协议可以是 SSL 或 TLS，这取决于您在 ConnectionFactory 对象中指定的 CipherSuite。

第 553 页的表 96 列出了必须在 ConnectionFactory 对象中定义的属性。

表 *96: 与 WebSphere Application Server service integration bus* 消息传递引擎的安全连接的 *ConnectionFactory* 属性

属性的名称	描述
XMSC_WPM_SSL_CIPHER_SUITE	要在与 WebSphere Application Server service integration bus 消息传递引擎的 TLS 连接上使用的 CipherSuite 的名称。 协商安全连接时使用的协议取决于指定的 CipherSuite。
XMSC_WPM_SSL_KEYRING_LABEL	向服务器进行认证时要使用的证书。

下面是与 WebSphere Application Server service integration bus 消息传递引擎的安全连接的 ConnectionFactory 属性示例：

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

其中，应将 chain_name 设置为 BootstrapTunneledSecureMessaging 或 BootstrapSecureMessaging，port_number 是引导程序服务器侦听入局请求时所使用的端口号。

下面是与 WebSphere Application Server service integration bus 消息传递引擎的安全连接的 ConnectionFactory 属性示例（已插入示例值）：

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

与 *WebSphere Application Server service integration bus* 的连接的 *CipherSuite* 和 *CipherSpec* 名称映射

由于 GSKit 使用 CipherSpec（而不是 CipherSuite），因此必须将 XMSC_WPM_SSL_CIPHER_SUITE 属性中指定的 JSSE 样式的 CipherSuite 名称映射到 GSKit 样式的 CipherSpec 名称。

第 553 页的表 97 列出了每个公认 CipherSuite 的等效 CipherSpec。

表 *97:* 可用的 *CipherSuite* 及其等效的 *CipherSpec*

密码套件	CipherSpec 等效项
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA

表 97: 可用的 *CipherSuite* 及其等效的 *CipherSpec (继续)*	
密码套件	**CipherSpec 等效项**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

注: 不推荐 TLS_RSA_WITH_3DES_EDE_CBC_SHA。 但是，它仍可用于传输最多 32 GB 数据，超过此数据量之后，连接将因错误 AMQ9288 而终止。 要避免此错误，您需要避免使用三重 DES，或在使用此 CipherSpec 时启用密钥重置。

XMS 消息

本部分描述了 XMS 消息的结构和内容，并解释了应用程序如何处理 XMS 消息。

本部分包含以下主题:

- 第 554 页的『XMS 消息的组成部分』
- 第 554 页的『XMS 消息中的头字段』
- 第 555 页的『XMS 消息的属性』
- 第 557 页的『XMS 消息的主体』
- 第 560 页的『消息选择器』
- 第 561 页的『将 XMS 消息映射到 IBM MQ 消息』

XMS 消息的组成部分

XMS 消息由一个头、一组属性和一个主体组成。

头
消息的头包含字段，并且所有消息都包含同一组头字段。 XMS 和应用程序使用头字段的值来识别和传递消息。 有关头字段的更多信息，请参阅 第 554 页的『XMS 消息中的头字段』。

属性集
消息的属性指定有关消息的其他信息。 虽然所有消息都具有同一组头字段，但是每条消息可以具有一组不同的属性。 有关更多信息，请参阅 第 555 页的『XMS 消息的属性』。

正文
消息的主体包含应用程序数据。 有关更多信息，请参阅 第 557 页的『XMS 消息的主体』。

应用程序可以选择要接收的消息。 使用可指定选择条件的消息选择器。 条件可以基于某些头字段的值以及消息的任何属性的值。 有关消息选择器的更多信息，请参阅第 560 页的『消息选择器』。

XMS 消息中的头字段

要允许 XMS 应用程序与 WebSphere JMS 应用程序交换消息，XMS 消息的头包含 JMS 消息头字段。

这些头字段的名称以前缀 JMS 开头。 有关 JMS 消息头字段的描述，请参阅 *Java Message Service* 规范。

XMS 会将 JMS 消息头字段实现为 Message 对象的属性。 每个头字段都有自己的用于设置和获取其值的方法。 有关这些方法的描述，请参阅 IMessage。 头字段始终可读且可写。

第 554 页的表 98 列出了 JMS 消息头字段，并指示如何为传输的消息设置每个字段的值。 在应用程序发送消息或者在 JMSRedelivered 情况下应用程序接收消息时，XMS 会自动设置其中某些字段。

表 98: *JMS* 消息头字段.]	
JMS 消息头字段的名称	**如何为传输的消息设置值（采用格式 *method* [*class*]）**
JMSCorrelationID	Set JMSCorrelationID [Message]
JMSDeliveryMode	Send [MessageProducer]
JMSDestination	Send [MessageProducer]

表 *98: JMS* 消息头字段.] *(继续)*	
JMS 消息头字段的名称	**如何为传输的消息设置值（采用格式 *method* [*class*]）**
JMSExpiration	Send [MessageProducer]
JMSMessageID	Send [MessageProducer]
JMSPriority	Send [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Set JMSReplyTo [Message]
JMSTimestamp	Send [MessageProducer]
JMSType	Set JMSType [Message]

XMS 消息的属性

XMS 支持三类消息属性：JMS 定义的属性、IBM 定义的属性和应用程序定义的属性。

XMS 应用程序可以与 WebSphere JMS 应用程序交换消息，因为 XMS 支持 Message 对象的下列预定义属性：

- WebSphere JMS 支持的相同 JMS 定义的属性。 这些属性的名称以前缀 JMSX 开头。
- WebSphere JMS 支持的相同 IBM 定义的属性。 这些属性的名称以前缀 JMS_IBM_ 开头。

每个预定义属性都具有两个名称：

- JMS 名称（针对 JMS 定义的属性）或 WebSphere JMS 名称（针对 IBM 定义的属性）。

  这是在 JMS 或 WebSphere JMS 中用于标识属性的名称，也是与具有此属性的消息一起传输的名称。 XMS 应用程序使用此名称识别消息选择器表达式中的属性。
- 在所有情况（消息选择器表达式除外）下用于标识属性的 XMS 名称。 将每个 XMS 名称定义为 `IBM.XMS.XMSC` 类中的命名常量。 命名常量的值是对应的 JMS 或 WebSphere JMS 名称。

除了预定义的属性外，XMS 应用程序还可以创建和使用自己的消息属性集。 这些属性称为应用程序定义的属性。

在应用程序创建消息之后，消息属性为可读且可写。 在应用程序发送消息之后，这些属性仍然可读且可写。 在应用程序接收消息时，消息属性为只读。 如果应用程序在消息的属性为只读时调用 Message 类的 `Clear Properties` 方法，那么这些属性将变为可读且可写。 该方法还会清除这些属性。

在清除消息属性后转发已收到消息的行为方式与转发已清除消息属性的 .NET BytesMessage 的标准 WMQ XMS 的行为方式一致。

但是，由于将会丢失以下属性，所以建议不要这样做：

- JMS_IBM_Encoding 属性值，意味着无法采用有意义的方式解码消息数据。
- JMS_IBM_Format 属性值，意味着（MQMD 或新的 MQRFH2）消息头与现有头之间的头链将断开。

要确定消息的所有属性的值，应用程序可以调用 Message 类的 Get Properties 方法。 该方法将创建用于封装 Property 对象列表的迭代器，其中每个 Property 对象都表示消息的一个属性。 然后，应用程序可以使用 Iterator 类的方法来依次检索各个 Property 对象，它可以使用 Property 类的方法来检索每个属性的名称、数据类型和值。

消息的 *JMS* 定义的属性

XMS 和 WebSphere JMS 支持消息的多个 JMS 定义的属性。

第 556 页的表 99 列出了 XMS 和 WebSphere JMS 支持的消息的 JMS 定义的属性。 有关 JMS 定义的属性的描述，请参阅 *Java Message Service* 规范。 JMS 定义的属性不适用于与代理程序的实时连接。

该表列出了每个属性的数据类型，并指示如何为传输的消息设置此属性的值。 当应用程序发送消息或者在 JMSXDeliveryCount 情况下应用程序接收消息时，XMS 会自动设置其中某些属性。

表 *99:* 消息的 *JMS* 定义的属性			
JMS 定义的属性的 XMS 名称	**JMS 名称**	**数据类型**	**如何为传输的消息设置值（采用格式 *method* [*class*]）**
JMSX_APPID	JMSXAppID	System.String	Send [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Set String Property [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Set Integer Property [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Send [MessageProducer]

消息的 *IBM* 定义的属性

XMS 和 WebSphere JMS 支持消息的多个 IBM 定义的属性。

第 556 页的表 100 列出了 XMS 和 WebSphere JMS 都支持的 IBM 定义的消息属性。 有关 IBM 定义的属性的更多信息，请参阅 IBM MQ 或 WebSphere Application Server 产品文档。

该表列出了每个属性的数据类型，并指示如何为传输的消息设置此属性的值。 应用程序发送消息时，XMS 会自动设置其中某些属性。

表 *100:* 消息的 *IBM* 定义的属性			
IBM 定义的属性的 XMS 名称	**WebSphere JMS 名称**	**数据类型**	**如何为传输的消息设置值（采用格式 *method* [*class*]）**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEM DESTINATION	JMS_IBM_ExceptionProblemDestination	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Set String Property [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Set Integer Property [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Set Integer Property [PropertyContext]

表 *100:* 消息的 *IBM* 定义的属性 *(*继续*)*

IBM 定义的属性的 XMS 名称	WebSphere JMS 名称	数据类型	如何为传输的消息设置值（采用格式 *method* [*class*]）
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ 标识	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Send [MessageProducer]

消息的应用程序定义的属性

XMS 应用程序可以创建和使用自己的消息属性集。 在应用程序发送消息时，这些属性也随该消息一起传输。 然后，接收应用程序可以根据这些属性的值，使用消息选择器来选择要接收的消息。

要允许 WebSphere JMS 应用程序选择和处理 XMS 应用程序发送的消息，应用程序定义的属性的名称必须遵循在消息选择器表达式中构成标识的相关规则。 有关更多信息，请参阅第 112 页的『JMS 中的消息选择器』。 应用程序定义的属性的值必须具有以下某种数据类型：System.Boolean、System.SByte、System.Int16、System.Int32、System.Int64、System.Float、System.Double 或 System.String。

XMS 消息的主体

消息的主体包含应用程序数据。 但是，消息也可以没有主体，而是只包含头字段和属性。

XMS 支持以下五种类型的消息体：

字节

主体包含字节流。 具有此类型主体的消息称为字节消息。 IBytesMessage 接口包含用于处理字节消息体的方法。

映射
主体包含一组名称/值对，其中每个值都具有相关的数据类型。 具有此类型主体的消息称为映射消息。 IMapMessage 接口包含用于处理映射消息体的方法。

Object
主体包含序列化的 Java 或 .NET 对象。 具有此类型主体的消息称为对象消息。 IObjectMessage 接口包含用于处理对象消息体的方法。

流
主体包含值流，其中每个值都具有相关的数据类型。 具有此类型主体的消息称为流消息。 IStreamMessage 接口包含用于处理流消息体的方法。

文本
主体包含字符串。 具有此类型主体的消息称为文本消息。 ITextMessage 接口包含用于处理文本消息体的方法。

IMessage 接口是所有 Message 对象的父项，可以在消息传递函数中用于表示任何 XMS 消息类型。

有关所有这些数据类型的大小以及最大值和最小值的信息，请参阅 第 537 页的表 85。

字节消息

字节消息的主体包含一连串字节。 主体仅包含实际数据，由发送和接收应用程序负责解释这些数据。

如果 XMS 应用程序需要与不使用 XMS 或 JMS 应用程序编程接口的应用程序交换消息，那么字节消息将很有用。

在应用程序创建字节消息之后，消息体为只写。 应用程序通过对 .NET 调用 IBytesMessage 接口的相应写方法，将应用程序数据组合到主体中。 每次应用程序将值写入字节消息流时，都会直接在应用程序写入的前一个值后面组合该值。 XMS 会维护一个内部光标来记住组合的最后一个字节的位置。

在应用程序发送消息时，消息体将变为只读。 在此方式下，应用程序可以重复发送消息。

在应用程序接收字节消息时，消息体为只读。 应用程序可以使用 IBytesMessage 接口的相应读方法来读取字节消息流的内容。 应用程序将按顺序读取字节，并且 XMS 会维护一个内部光标来记住读取的最后一个字节的位置。

如果应用程序在字节消息体为可写时调用 IBytesMessage 接口的 Reset 方法，该主体将变为只读。 该方法还会将光标重新定位到字节消息流的开头。

当字节消息的主体为只读时，如果应用程序调用 .NET 的 IMessage 接口的 `Clear Body` 方法，那么该主体变为可写。 该方法还会清除主体。

映射消息

映射消息的主体包含一组名称/值对，其中每个值都具有相关的数据类型。

在每个 "名称/值" 对中， "名称" 是用于标识值的字符串，而 "值" 是具有 第 559 页的表 101 中列出的其中一种 XMS 数据类型的应用程序数据元素。 未定义名称/值对的顺序。 MapMessage 类包含用于设置和获取名称/值对的方法。

应用程序可以通过指定名称来随机访问名称/值对。

.NET 应用程序可以使用 MapNames 属性来获取映射消息体中名称的枚举。

在应用程序获取名称/值对的值后，XMS 可以将该值转换为另一种数据类型。 例如，要从映射消息体中获取某个整数，应用程序可以调用 MapMessage 类的 GetString 方法，以将该整数作为字符串返回。 受支持的转换与 XMS 将属性值从一种数据类型转换为另一种数据类型时支持的转换相同。 有关受支持的转换的更多信息，请参阅第 537 页的『属性值从一种数据类型到另一种数据类型的隐式转换』。

在应用程序创建映射消息之后，消息体为可读且可写。 在应用程序发送消息之后，主体仍然可读且可写。 在应用程序接收映射消息时，消息体为只读。 如果应用程序在映射消息体为只读时调用 Message 类的 Clear Body 方法，该主体将变为可读且可写。 该方法还会清除主体。

对象消息

对象消息的主体包含序列化的 Java 或 .NET 对象。

XMS 应用程序可以接收对象消息，更改其头字段和属性，然后将其发送到其他目标。 应用程序也可以复制对象消息的主体，并使用它来构成其他对象消息。 XMS 会将对象消息的主体视为字节数组。

在应用程序创建对象消息之后，消息体为可读且可写。 在应用程序发送消息之后，主体仍然可读且可写。 在应用程序接收对象消息时，消息体为只读。 如果应用程序在对象消息的主体为只读时调用 .NET 的 IMessage 接口的 `Clear Body` 方法，那么该主体将变为可读且可写。 该方法还会清除主体。

流消息

流消息的主体包含一连串值，其中每个值都具有相关的数据类型。

值的数据类型是 第 559 页的表 101 中列出的其中一种 XMS 数据类型。

在应用程序创建流消息之后，消息体为可写。 应用程序通过调用 .NET IStreamMessage 接口的相应写方法，将应用程序数据组合到主体中。 每次应用程序将值写入消息流时，都会直接在应用程序写入的前一个值后面组合该值及其数据类型。 XMS 会维护一个内部光标来记住组合的最后一个值的位置。

在应用程序发送消息时，消息体将变为只读。 在此方式下，应用程序可以多次发送消息。

在应用程序接收流消息时，消息体为只读。 应用程序可以使用 .NET IStreamMessage 接口的相应读方法来读取消息流的内容。 应用程序将按顺序读取值，并且 XMS 会维护一个内部光标来记住读取的最后一个值的位置。

在应用程序从消息流中读取值时，该值可由 XMS 转换为其他数据类型。 例如，要从消息流中读取某个整数，应用程序可以调用 ReadString 方法以将该整数作为字符串返回。 受支持的转换与 XMS 将属性值从一种数据类型转换为另一种数据类型时支持的转换相同。 有关受支持的转换的更多信息，请参阅第 537 页的『属性值从一种数据类型到另一种数据类型的隐式转换』。

如果在应用程序尝试从消息流中读取值时发生错误，那么不会将光标前移。 应用程序可以通过尝试将值作为另一种数据类型读取来从该错误中恢复。

如果应用程序在流消息体为只写时调用 XMS IStreamMessage 接口的 Reset 方法，该主体将变为只读。 该方法还会将光标重新定位到消息流的开头。

如果应用程序在流消息的主体为只读时调用 XMS 的 IMessage 接口的 `Clear Body` 方法，那么该主体将变为只读。 该方法还会清除主体。

文本消息

文本消息的主体包含一个字符串。

在应用程序创建文本消息之后，消息体为可读且可写。 在应用程序发送消息之后，主体仍然可读且可写。 在应用程序接收文本消息时，消息体为只读。 如果应用程序在文本消息体为只读时调用 .NET IMessage 接口的 Clear Body 方法，该主体将变为可读且可写。 该方法还会清除主体。

应用程序数据元素的数据类型

为确保 XMS 应用程序可以与 IBM MQ classes for JMS 应用程序交换消息，这两个应用程序必须能够以相同的方式解释消息体中的应用程序数据。

由于这个原因，XMS 应用程序在消息体中写入的应用程序数据的每个元素都必须具有第 559 页的表 101 中所列的一种数据类型。 对于每种数据类型，该表显示了兼容的 Java 数据类型。 XMS 提供了仅使用这些数据类型来写入应用程序数据元素的方法。

表 *101: 与 Java* 数据类型兼容的 *XMS* 数据类型

XMS 数据类型	**表示**	**与 Java 兼容的数据类型**
System.Boolean	布尔值 true 或 false	布尔值
System.Char16	双字节字符	char

表 101: 与 Java 数据类型兼容的 XMS 数据类型 (继续)		
XMS 数据类型	**表示**	**与 Java 兼容的数据类型**
System.SByte	有符号的 8 位整数	byte
System.Int16	有符号的 16 位整数	短整型
System.Int32	有符号的 32 位整数	int
System.Int64	有符号的 64 位整数	long
System.Float	有符号的浮点数	浮点值
System.Double	有符号的双精度浮点数	双精度值
System.String	字符串	字符串

有关所有这些数据类型的大小以及最大值和最小值的信息，请参阅 第 537 页的『XMS 基本类型』。

消息选择器

XMS 应用程序使用消息选择器来选择要接收的消息。

在应用程序创建消息使用者时，它可以将消息选择器表达式与该使用者相关联。 消息选择器表达式可指定选择条件。

当应用程序连接到 IBM WebSphere MQ 7.0 队列管理器时，将在队列管理器端完成消息选择。 XMS 将不做任何选择，只是传递其从队列管理器收到的消息，因此能够提供更好的性能。

应用程序可以创建多个消息使用者，每个都有自己的消息选择器表达式。 如果入局消息满足多个消息使用者的选择条件，那么 XMS 会将该消息传递到所有这些使用者。

消息选择器表达式可以引用消息的以下属性:

- JMS 定义的属性
- IBM 定义的属性
- 应用程序定义的属性

它还可以引用以下消息头字段:

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

但是，消息选择器表达式无法引用消息体中的数据。

下面是消息选择器表达式的示例:

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

仅当消息的优先级大于 3 时， XMS 才会使用此消息选择器表达式将消息传递给消息使用者; 应用程序定义的属性，制造商 (值为 `Jaguar;`) 和另一个应用程序定义的属性，模型 (值为 `xj6` 或 `xj12.`)

XMS 中构成消息选择器表达式的语法规则与 IBM MQ classes for JMS 中的相同。 有关如何构造消息选择器表达式的信息，请参阅 IBM MQ 产品文档。请注意，在消息选择器表达式中，JMS 定义的属性的名称必须为 JMS 名称，IBM 定义的属性的名称必须为 IBM MQ classes for JMS 名称。 不能在消息选择器表达式中使用 XMS 名称。

将 XMS 消息映射到 IBM MQ 消息

将 XMS 消息的 JMS 头字段和属性映射到 IBM MQ 消息的头结构中的字段。

当 XMS 应用程序连接到 IBM MQ 队列管理器时，发送到队列管理器的消息将以与在类似情况下 IBM MQ classes for JMS 消息映射到 IBM MQ 消息相同的方式映射到 IBM MQ 消息。

如果将 Destination 对象的 XMSC_WMQ_TARGET_CLIENT 属性设置为 XMSC_WMQ_TARGET_DEST_JMS，那么会将发送到目标的消息的 JMS 头字段和属性映射到 IBM MQ 消息的 MQMD 和 MQRFH2 头结构中的字段。 如果使用该方式设置 XMSC_WMQ_TARGET_CLIENT 属性，即是假定接收消息的应用程序可以处理 MQRFH2 头。 因此，接收应用程序可能是另一个 XMS 应用程序、IBM MQ classes for JMS 应用程序或设计为处理 MQRFH2 头的本机 IBM MQ 应用程序。

如果将 Destination 对象的 XMSC_WMQ_TARGET_CLIENT 属性转而设置为 XMSC_WMQ_TARGET_DEST_MQ，那么会将发送到目标的消息的 JMS 头字段和属性映射到 IBM MQ 消息的 MQMD 头结构中的字段。 消息不包含 MQRFH2 头，并且将忽略那些无法映射到 MQMD 头结构中字段的 JMS 头字段和属性。 因此，接收消息的应用程序可能是未设计为处理 MQRFH2 头的本机 IBM MQ。

从队列管理器接收的 IBM MQ 消息映射到 XMS 消息的方式与在类似情况下将 IBM MQ 消息映射到 IBM MQ classes for JMS 消息的方式相同。

如果入局 IBM MQ 消息具有 MQRFH2 头，那么生成的 XMS 消息具有主体，其类型由 MQRFH2 头的 mcd 文件夹中包含的 **Msd** 属性的值确定。 如果 **Msd** 属性不存在于 MQRFH2 头中，或者如果 IBM MQ 消息不包含 MQRFH2 头，那么生成的 XMS 消息的主体类型由 MQMD 头中的 *Format* 字段值确定。 如果将 *Format* 字段设置为 MQFMT_STRING，那么 XMS 消息为文本消息。 否则，XMS 消息为字节消息。 如果 IBM MQ 消息不包含 MQRFH2 头，那么只会设置可从 MQMD 头中的字段派生的 JMS 头字段和属性。

有关将 IBM MQ classes for JMS 消息映射到 IBM MQ 消息的更多信息，请参阅第 115 页的『将 JMS 消息映射到 IBM MQ 消息』。

通过 *IBM Message Service Client for .NET* 应用程序读写消息描述符

您可以访问 IBM MQ 消息的除 StrucId 和 Version 以外的所有其他消息描述符 (MQMD) 字段；可以读取但不能写入 BackoutCount。 仅当连接到 IBM WebSphere MQ 6.0 或更高版本的队列管理器时，此功能才可用，它由稍后所述的目标属性来控制。

IBM Message Service Client for .NET 提供的消息属性不仅可以帮助 XMS 应用程序设置 MQMD 字段，还可以驱动 IBM WebSphere MQ 应用程序。

使用发布/预订消息传递时，存在一些限制。 例如，将忽略诸如 MsgID 和 CorrelId 的 MQMD 字段。

当连接到 IBM WebSphere MQ 6.0 队列管理器时，此主题中描述的功能不可用于发布/预订消息传递。 当 **PROVIDERVERSION** 属性设置为 6 时，该功能同样不可用。

从 *IBM Message Service Client for .NET* 应用程序访问 *IBM MQ* 消息数据

您可以在作为 JMSBytesMessage 主体的 IBM Message Service Client for .NET 应用程序中访问完整的 IBM MQ 消息数据，包括 MQRFH2 头 (如果存在) 和任何其他 IBM MQ 头 (如果存在)。

仅当连接到 IBM WebSphere MQ 7.0 或更高版本的队列管理器并且 IBM MQ 消息传递提供程序处于正常方式时，本主题中描述的功能才可用。

Destination 对象属性将确定 XMS 应用程序如何访问充当 JMSBytesMessage 主体的整个 IBM MQ 消息（包括 MQRFH2 头（如果存在））。

使用组件对象模型接口 (IBM MQ Automation Classes for ActiveX)

IBM MQ Automation Classes for ActiveX (MQAX) 是一些 ActiveX 组件，这些组件提供了一些类，您可以在应用程序中使用这些类来访问 IBM MQ。

从 IBM MQ 9.0 开始，不推荐使用对 Microsoft Active X 的 IBM MQ 支持。 IBM MQ classes for .NET 是建议使用的替代技术。 有关更多信息，请参阅开发 .NET 应用程序。

MQAX 需要 IBM MQ 环境，还需要一个与之通信的相应 IBM MQ 应用程序。

它让您的 ActiveX 应用程序能够在任何您能够通过 IBM MQ 访问的企业系统上运行事务和访问数据。

IBM MQ Automation Classes for ActiveX：

- 让您能够访问 IBM MQ API 的功能和特性，从而实现与其他 IBM MQ 平台的完全互联。
- 符合对 ActiveX 组件期待的一般约定。
- 符合 IBM MQ 对象模型，还可用于 .NET、C++、Java 和 LotusScript。

提供了 MQAX 启动器样本。 您可以使用这些样本初步检查您的 MQAX 安装是否成功，以及是否准备好了基本的 IBM MQ 环境。 样本还演示了如何使用 MQAX。

COM 和 ActiveX 脚本编制

组件对象模型 (COM) 是一个由 Microsoft 定义的、基于对象的编程模型。 它指定通过怎样的方法提供软件组件，能够允许它们互相定位和通信，而不考虑用于编写它们的计算机语言或它们的位置。

ActiveX 是一组基于 COM 的技术，它将应用程序开发、可重复使用的组件，以及互联网技术集成到 Microsoft Windows 平台上。 ActiveX 组件提供应用程序可动态地访问的接口。 ActiveX 脚本编制客户机是应用程序（例如编译器），可构建或执行使用 ActiveX（或 COM）组件所提供接口的程序或脚本。

IBM MQ 环境支持

IBM MQ Automation Classes for ActiveX 只能用于 **32 位** ActiveX 脚本客户机。

COM 组件只能用于 **32 位**应用程序。 如果您希望编写 64 位 COM 应用程序，可以使用 .NET 接口。

要在 IBM MQ 服务器环境中运行 MQAX，您必须在系统上安装 Windows 2000 或更高版本。

要在安装在系统上的 Windows 2000 或更高版本上的 IBM MQ MQI client 环境中运行 MQAX ，您需要 IBM MQ MQI client :

IBM MQ MQI client 需要访问至少一个 IBM MQ 服务器。 如果在系统上同时安装了 IBM MQ MQI client 和 the IBM MQ 服务器，MQAX 应用程序始终会在服务器上运行。 只有 IBM MQ 服务器环境中才能使用 MQAI ActiveX 接口。

使用 IBM MQ Automation Classes for ActiveX 进行设计和编程

设计访问非 ActiveX 应用程序的 MQAX 应用程序

IBM MQ 自动化类提供了对 IBM MQ API 的访问途径。 由于使用 IBM MQ 可以访问您的 Windows 应用程序，因此，您可以享受到其中的全部优势。

由于您应用程序的总体设计与任何 IBM MQ 应用程序相同，因此，请考虑第 40 页的『IBM MQ 应用程序的设计注意事项』一节中描述的所有设计方面。

要使用 IBM MQ 自动化类，您需要在应用程序中使用支持创建和使用 COM 对象的语言来编写 Windows 程序的代码。 例如，Visual Basic、Java 和其他 ActiveX 脚本编写客户机。 然后，可以轻松地将这些类集成到您的应用程序中，因为可以使用实现语言的本机语法来编写您需要的 IBM MQ 对象的代码。

使用 IBM MQ Automation Classes for ActiveX

设计使用 IBM MQ Automation Classes for ActiveX 的 ActiveX 应用程序时，最重要的信息项目是与远程 IBM MQ 系统发送或接收的消息。 因此您必须知道插入消息的项的格式。 要让 MQAX 脚本能够工作，该脚本和收发消息的 IBM MQ 应用程序必须知道该消息的结构。

如果您使用 MQAX 应用程序发送消息，并且要在 MQAX 端执行数据转换，那么必须还要知道：

- 远程系统使用的代码页
- 远程系统使用的编码

要让您的代码保持可移植，设置代码页和编码是一种很好的做法，即使两者当前在发送和接收系统中相同也不例外。

在考虑如何设计实施系统的结构时，请记住您的 MQAX 脚本在安装 IBM MQ 队列管理器或 IBM MQ 客户机同一台计算机上运行。

编程提示和技巧

下列提示和技巧并非按重要性排序。 如果这些主题与您从事的工作相关，可能会节省您的时间。

消息描述符属性

如果在程序中使用消息描述符属性，最好使用这些字段的十六进制等效形式。

本节中的信息指的是以下属性：

- AccountingToken
- CorrelationId
- GroupId
- MessageId

如果 IBM MQ 应用程序发起消息，IBM MQ 还生成了这些属性，并且您希望查看这些属性的值，或者以任何方式使用这些属性，包括将它们放到消息中传递回 IBM MQ，最好使用 AccountingTokenHex、CorrelationIdHex、GroupIdHex 和 MessageIdHex 属性。 这是因为 IBM MQ 生成的值是字节数在 0 到 255 之间（包含 0 和 255）的字符串，不是由可打印字符组成的字符串。

如果您的 MQAX 脚本是消息的发起方，您可以使用 AccountingToken、CorrelationId、GroupId 和 MessageId 属性或其相应的十六进制等效形式。

IBM MQ 常量

IBM MQ 常量是作为库 MQAX200 中 enum IBM MQ 的成员提供的。

IBM MQ 字符串常量

使用 IBM MQ Automation Classes for ActiveX 时，IBM MQ 字符串常量不可用。 您必须对下表中所示的字符串，以及其他可能需要的字符串使用显式字符串。 必须使用空格来将命令填充至八个字符：

表 *102: IBM MQ* 字符串常量及其相应的字符串。

字符串常量	**相应的字符串**
MQFMT_NONE	" "
MQFMT_ADMIN	"MQADMIN "
MQFMT_CHANNEL_COMPLETED	"MQCHCOM "
MQFMT_CICS	"MQCICS "
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD "
MQFMT_DIST_HEADER	"MQHDIST "
MQFMT_EVENT	"MQEVENT "
MQFMT_IMS	"MQIMS "
MQFMT_IMS_VAR_STRING	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "
MQFMT_REF_MSG_HEADER	"MQHREF "
MQFMT_RF_HEADER	"MQHRF "

表 *102: IBM MQ* 字符串常量及其相应的字符串。 *(继续)*	
字符串常量	**相应的字符串**
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	"MQXMIT "

空字符串常量

IBM MQ Automation Classes for ActiveX 不支持 IBM MQ 常量，这些常量用于初始化四个 MQMessage 属性：MQMI_NONE（24 个空字符）、MQCI_NONE （24 个空字符）、MQGI_NONE（24 个空字符）和 MQACT_NONE（32 个空字符）。 将它们设置为空字符串也有相同效果。

例如，要将 MQMessage 的各个标识设置为以下值：*mymessage*。 **MessageId** = "" *mymessage*. **CorrelationId** = "" *mymessage*. **AccountingToken** = ""

接收来自 IBM MQ 的消息

有几种方法可接收来自 IBM MQ 的消息：

- 使用 Visual Basic TIMER 函数，通过发出 GET 再发出 Wait 进行轮询。
- 发出带 Wait 选项的 GET；您可通过设置 WaitInterval 属性指定等待持续时间。 即使您将系统设置为在多线程环境中运行，当时运行的软件可能只运行一个线程，这时，请考虑使用这种方法。 这可避免系统无限期地锁定。

  其他线程操作不受影响。 然而，如果您的其他线程需要访问 IBM MQ，它们会使用其他 MQAX 队列管理器和队列对象请求与 IBM MQ 的第二个连接。

如果此进程为单线程，使用 Wait 选项，并将 WaitInterval 设置为 MQWI_UNLIMITED 发出 GET 可导致您的系统锁定，直到 GET 调用完成。

使用数据转换

IBM MQ Automation Classes for ActiveX 支持两种形式的数据转换 - 数字编码和字符集转换。

数字编码

如果您设置 MQMessage Encoding 属性，那么下列方法会在不同数字编码系统之间转换：

- ReadDecimal2 方法
- ReadDecimal4 方法
- ReadDouble 方法
- ReadDouble4 方法
- ReadFloat 方法
- ReadInt2 方法
- ReadInt4 方法
- ReadLong 方法
- ReadShort 方法
- ReadUInt2 方法
- WriteDecimal2 方法
- WriteDecimal4 方法
- WriteDouble 方法

- WriteDouble4 方法
- WriteFloat 方法
- WriteInt2 方法
- WriteInt4 方法
- WriteLong 方法
- WriteShort 方法
- WriteUInt2 方法

可以使用所提供的 IBM MQ 常量设置和解释 Encoding 属性。 第 565 页的图 60 显示了它们的示例：

```
/* Encodings for Binary Integers */
MQENC_INTEGER_UNDEFINED
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
MQENC_DECIMAL_UNDEFINED
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
MQENC_FLOAT_UNDEFINED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_S390
```

图 *60:* 所提供的用于编码的 *IBM MQ* 常量

例如，要将来自 Intel 系统的整数发送到使用 System/390 编码的 System/390 操作系统：

```
Dim msg As New MQMessage  'Define an IBM MQ message for our use..
Print msg. Encoding   'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg. Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg. Encoding   'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg. WriteLong (local_num)       'Write the number into the message
```

字符集转换

当您从一个系统向另一个有不同代码页的系统发送消息时，字符集转换是必需的。 代码页转换用于：

- ReadString 方法
- ReadNullTerminatedString 方法
- WriteString 方法
- WriteNullTerminatedString 方法
- MessageData 属性

您必须将 MQMessage CharacterSet 属性设置为支持的字符集值（CCSID）。

IBM MQ Automation Classes for ActiveX 使用转换表来执行字符集转换。

例如，将字符串自动转换为代码页 437：

```
Dim msg As New MQMessage                'Define an IBM MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

WriteString 方法可以 Unicode 字符串的形式检索字符串数据（例如，`A character string`）。 然后它使用转换表 34B001B5.TBL 将此数据从 Unicode 转换为代码页 437。

对 Unicode 字符串中代码页 437 不支持的字符给出了代码页 437 的标准替换字符。

同样，使用 ReadString 方法时，传入消息会使用 IBM MQ 消息描述符 (MQMD) 值建立一个字符集，并将此代码页转换为 Unicode 格式，然后再将其传递回您的脚本语言。

线程化

IBM MQ Automation Classes for ActiveX 可实现一个自由线程模型，在该模型中，可以在线程之间使用对象。

虽然 MQAX 允许使用 MQQueue 和 MQQueueManager 对象，但 IBM MQ 当前不允许在不同线程之间共享句柄。

尝试在另一个线程上使用它们会导致错误，并且 IBM MQ 会返回一个返回码 MQRC_HCONN_ERROR。

注: 每个进程只有一个 MQSession 对象。 我们不建议您在多线程环境中使用 MQSession CompletionCode 和 ReasonCode。 MQSession 错误值可能会被第一个线程上所引发和检测到的错误之间的第二个线程覆盖。 针对每个方法调用或属性访问的持续时间对线程进行了序列化。 所以，使用 Wait 选项发出 GET 会导致其他访问 MQAX 对象的线程被挂起，直到操作完成。

错误处理

此信息描述了 MQAX 对象属性，错误处理如何工作，描述如何处理引发异常的规则以及获取属性。

每个 MQAX 对象都包含保存错误信息的属性和复位或清除它们的方法。 这些属性是:

- CompletionCode
- ReasonCode
- ReasonName

方法是:

- ClearErrorCodes

错误处理如何工作

您的 MQAX 脚本或应用程序可调用 MQAX 对象的方法，或处理或更新 MQAX 对象的属性:

1. 更新对象中的有关 ReasonCode 和 CompletionCode。
2. 还以相同信息更新 MQSession 对象中的 ReasonCode 和 CompletionCode。

   注: 请参阅第 566 页的『线程化』了解在线程应用程序中使用 MQSession 错误代码的限制。

   如果 CompletionCode 大于或等于 MQSession 的 ExceptionThreshold 属性，那么 MQAX 抛出一个异常（号码 32000）。 在您的脚本中使用它时可使用 On Error 语句（或等价语句）处理它。
3. 可以使用 Error 函数检索关联的错误字符串，其格式为:

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   有关如何使用 On Error 语句的更多信息，请参阅您 ActiveX 脚本语言的相关文档。

   对于简单错误处理程序，在 MQSession 对象中使用 CompletionCode 和 ReasonCode 很方便。

   ReasonName 属性会为 ReasonCode 的当前值返回 IBM MQ 符号名称。

产生异常

下列规则描述如何处理产生异常:

- 无论何时只要属性或方法将完成代码设置为大于或等于异常阈值（通常设置为 2）的值，就会产生异常。
- 所有方法调用和属性集都设置完成代码。

获取属性

这是一个特例，因为并不总是更新 CompletionCode 和 ReasonCode：

- 如果属性获得成功，那么对象、MQSession 对象 ReasonCode 和 CompletionCode 保持不变。
- 如果属性失败并带有警告的 CompletionCode，那么 ReasonCode 和 CompletionCode 保持不变。
- 如果属性失败并带有错误的 CompletionCode，那么更新 ReasonCode 和 CompletionCode 以反映真实值，并且错误处理按所描述的那样继续。

MQSession 类有一种方法 *ReasonCodeName*，也许可将这种方法用于将 IBM MQ 原因码替换为符号名称。在开发的程序可能发生意外错误时，这种方法尤其有用。 但该名称对于用户并不是理想的表示。

每个类还有属性 *ReasonName*，它返回该类的当前原因码的符号名称。

IBM MQ Automation Classes for ActiveX 参考

这部分描述了为 ActiveX 开发的 IBM MQ Automation Classes for ActiveX (MQAX) 的类。 您可以使用 IBM MQ 通过这些类编写 ActiveX 应用程序，这些应用程序可以访问其他在您的非 ActiveX 环境中运行的应用程序。

IBM MQ Automation Classes for ActiveX 接口

IBM MQ Automation Classes for ActiveX 提供了一些在使用这些类时需要的预定义数字 ActiveX 常量（例如，MQMT_REQUEST）。

ActiveX 自动化类由下列类组成：

- 第 568 页的『MQSession 类』
- 第 571 页的『MQQueueManager 类』
- 第 582 页的『MQQueue 类』
- 第 597 页的『MQMessage 类』
- 第 618 页的『MQPutMessageOptions 类』
- 第 620 页的『MQGetMessageOptions 类』
- 第 622 页的『MQDistributionList 类』
- 第 627 页的『MQDistributionListItem 类』

此外，IBM MQ Automation Classes for ActiveX 还提供了一些使用这些类时需要的预定义数字 ActiveX 常量（例如，MQMT_REQUEST）。 这些常量在库 MQAX200 中的 enum MQ 中提供。 这些常量是 IBM MQ C 头文件 (cmqc*.h) 中定义的集合的子集，带有其他一些 IBM MQ Automation Classes for ActiveX 原因码。

关于 IBM MQ Automation Classes for ActiveX 类

阅读这些信息，同时参考开发应用程序参考下的主题。

See Features that can be used only with the primary installation on Windows for important information.

MQSession 类提供一个根对象，包含对任何 MQAX 对象执行的最近操作的状态。 请参阅第 566 页的『错误处理』，了解更多信息。

MQQueueManager 和 MQQueue 类提供了对基本 IBM MQ 对象的访问权。 对这些类的方法或属性访问通常会导致在整个 IBM MQ MQI 中进行调用。

MQMessage、MQPutMessageOptions 和 MQGetMessageOptions 类会封装 MQMD、MQPMO 和 MQGMO 数据结构，可用于帮助您将消息发送到队列以及从队列中检索消息。

MQDistributionList 类封装队列 — 本地、远程或别名的集合以供输出。 MQDistributionListItem 类封装 MQOR、MQRR 和 MQPMR 结构并将其与属主分发列表关联。

参数传递

方法调用上的参数都是通过值传递的，但参数是对象时例外，在此情况下传递的是引用。

所提供的类定义列出了每个参数或属性的数据类型。 对于许多 ActiveX 客户机（如 Visual Basic）来说，如果使用的变量不属于必需类型，只要有可能，该值会在其所属类型与必需类型之间自动来回转换。 这符合客户机的标准规则；MQAX 不提供此类转换。

许多方法都使用定长字符串参数，或返回定长字符串。 转换规则按如下所示：

- 如果用户提供错误长度的定长字符串作为输入参数或返回值，那么会根据需要截断该值或用结尾空格填充该值。
- 如果用户提供长度不正确的可变长度字符串作为输入参数，那么会截断该值或用结尾空格填充该值。
- 如果用户提供长度不正确的可变长度字符串作为返回值，会将该字符串调整为必需长度（因为不管怎样返回值都会破坏字符串中的前一个值）。
- 作为输入参数提供的字符串可包含嵌入的 Null。

这些类可在 MQAX200 库中找到。

对象访问方法

这些方法不直接与任何单个 IBM MQ 调用相关。 每种方法都会创建一个对象，在该对象中会保留参考信息，之后会连接到或打开一个 IBM MQ 对象：

连接到队列管理器时，它会保留 IBM MQ 生成的“连接句柄”属性。

打开队列后，它会保留 IBM MQ 生成的“对象句柄”属性。

MQAX 程序不能直接使用这些 IBM MQ 属性。

错误

ActiveX 客户端在编译时和运行时可能会检测到有关参数传递的语法错误。 在 Visual Basic 中可使用 On Error 对错误设陷阱。

所有 IBM MQ ActiveX 类都包含两个特殊的只读属性：ReasonCode 和 CompletionCode。 可以在任何时候读取这两个属性。

尝试访问任何其他属性，或尝试发布任何方法调用都可能会导致 IBM MQ 生成错误。

如果属性集合或方法调用成功，那么拥有对象的 ReasonCode 将设置为 MQRC_NONE，CompletionCode 将设置为 MQCC_OK。

如果属性访问或方法调用不成功，那么在这些字段中设置原因和完成代码。

MQSession 类

这是 IBM MQ Automation Classes for ActiveX 的一个根类。

每个 ActiveX 客户机进程总是只有一个 MQSession 对象。 尝试创建第二个对象时会创建对原始对象的第二个引用。

创建

新建会创建一个新的 MQSession 对象。

语法

Dim *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

属性

- 第 569 页的『CompletionCode 属性』.

- 第 569 页的『ExceptionThreshold 属性』.
- 第 569 页的『ReasonCode 属性』.
- 第 570 页的『ReasonName 属性』.

方法

- 第 570 页的『AccessGetMessageOptions 方法』.
- 第 570 页的『AccessMessage 方法』.
- 第 570 页的『AccessPutMessageOptions 方法』.
- 第 570 页的『AccessQueueManager 方法』.
- 第 571 页的『ClearErrorCodes 方法』.
- 第 571 页的『ReasonCodeName 方法』.

CompletionCode 属性

只读。 返回由针对任何 IBM MQ 对象发出的最新方法或属性集设置的 IBM MQ 完成代码。

当对任何 MQAX 对象成功调用方法或属性集后，会将其复位为 MQCC_OK。

错误事件处理程序可以检查此属性以诊断错误，而不必知道涉及哪个对象。

在 MQSession 对象中使用 CompletionCode 和 ReasonCode 对于简单错误处理程序是很方便的。

注: 请参阅第 566 页的『线程化』了解在线程应用程序中使用 MQSession 错误代码的限制。

定义于:
MQSession 类

数据类型:
Long

值:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法:
要获取: *completioncode& = MQSession*。CompletionCode

ExceptionThreshold 属性

读 — 写。 定义 MQAX 将对其抛出异常的 IBM MQ 错误的级别。 缺省设置为 MQCC_FAILED。 大于 MQCC_FAILED 的值可有效阻止异常处理，让程序员对 CompletionCode 和 ReasonCode 执行检查。

定义于: MQSession 类

数据类型: 长整型

值:

- 任何值，但应考虑 MQCC_WARNING、MQCC_FAILED 或更大值。

语法:

要获取: *ExceptionThreshold& = MQSession*。 **ExceptionThreshold**

设置：*MQSession.* **ExceptionThreshold** = *ExceptionThreshold$*

ReasonCode 属性

只读。 返回由针对任何 IBM MQ 对象发出的最新方法或属性集设置的原因码。

错误事件处理程序可以检查此属性以诊断错误，而不必知道涉及哪个对象。

在 MQSession 对象中使用 CompletionCode 和 ReasonCode 对于简单错误处理程序是很方便的。

注: 请参阅第 566 页的『线程化』了解在线程应用程序中使用 MQSession 错误代码的限制。

定义于: MQSession 类

数据类型: 长整型

值:

- 请参阅原因 (MQLONG)，以及第 633 页的『IBM MQ Automation Classes for ActiveX 原因码』下列出的其他 MQAX 值。

语法: 要获取: *reasoncode& = MQSession*。 **ReasonCode**

ReasonName 属性

只读。 返回最新原因码的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。

注: 请参阅第 566 页的『线程化』了解在线程应用程序中使用 MQSession 错误代码的限制。

定义于: MQSession 类

数据类型: 字符串

值:

- 请参阅 API 完成代码和原因码。

语法: 获取: *reasonname$ = MQSession* **.ReasonName**

AccessGetMessageOptions 方法

创建新的 MQGetMessageOptions 对象。

定义于:
MQSession 类

语法:
gmo = MQSession **.AccessGetMessageOptions()**

AccessMessage 方法

创建新的 MQMessage 对象。

定义于:
MQSession 类

语法:
msg = MQSession **.AccessMessage()**

AccessPutMessageOptions 方法

创建新的 MQPutMessageOptions 对象。

定义于:
MQSession 类

语法:
pmo = MQSession **.AccessPutMessageOptions()**

AccessQueueManager 方法

创建一个新的 MQQueueManager 对象，并通过 IBM MQ MQI client 或 IBM MQ 服务器将其连接到一个真实的队列管理器。 除了执行连接外，此方法还为队列管理器对象执行打开。

在您的系统上安装完 IBM MQ MQI client 和 IBM MQ 服务器后，缺省情况下，MQAX 应用程序将在服务器上运行。 要在客户端上运行 MQAX，必须在 `GMQ_MQ_LIB` 环境变量中指定客户端绑定库，例如，设置 `GMQ_MQ_LIB=mqic.dll`。

对于仅客户端安装，不必设置 GMQ_MQ_LIB 环境变量。 如果不设置此变量，IBM MQ 会尝试加载 amqzst.dll。 如果不存在此 DLL（例如，仅客户端安装的情况），IBM MQ 会尝试加载 mqic.dll。

如果成功，它会将 MQQueueManager 的 ConnectionStatus 设置为 TRUE。

队列管理器对每个 ActiveX 实例最多可连接到一个 MQQueueManager 对象。

如果与队列管理器的连接失败，那么发生错误事件，并设置 MQSession 对象的 ReasonCode 和 CompletionCode。

定义于： MQSession 类

语法： *set qm = MQSession* **.AccessQueueManager** (*Name$*)

参数： *Name$* 字符串。 要连接的队列管理器名称。

ClearErrorCodes 方法

将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于： MQSession 类

语法：

```
Call MQSession.ClearErrorCodes()
```

ReasonCodeName 方法

以给定的数值返回原因码的名称。 它对于给用户更清楚的错误条件指示很有用。 该名称仍然有些难以理解（例如，ReasonCodeName(2059) 是 **MQRC_Q_MGR_NOT_AVAILABLE**），因此如果有可能，应该捕捉错误并将其替换为与应用程序相应的描述性文本。

定义于： MQSession 类

语法: *errname $= MQSession* **.ReasonCode 名称** (*reasonCode&*)

参数: *reasoncode&* 长整型。 需要符号名称的原因码。

MQQueueManager 类

此类代表与队列管理器的连接。 此队列管理器可以在本地运行（一个 IBM MQ 服务器），也可以通过由 IBM MQ 客户端提供的访问权远程运行。 应用程序必须创建此类的对象并将其连接到队列管理器。 当此类的对象被破坏时，它会自动从其队列管理器断开连接。

包含

MQQueue 类对象与此类关联。

“新建”会创建一个新的 MQQueueManager 对象，并将所有属性设置为初始值。 或者可使用 MQSession 类的 AccessQueueManager 方法。

创建

“新建”可创建一个**新** MQQueueManager 对象，并将所有属性设置为初始值。 或者可使用 MQSession 类的 AccessQueueManager 方法。

语法

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

属性

- 第 573 页的『AlternateUserId 属性』.

- 第 573 页的『AuthorityEvent 属性』.
- 第 573 页的『BeginOptions 属性』.
- 第 574 页的『ChannelAutoDefinition 属性』.
- 第 574 页的『ChannelAutoDefinitionEvent 属性』.
- 第 574 页的『ChannelAutoDefinitionExit 属性』.
- 第 574 页的『CharacterSet 属性』.
- 第 574 页的『CloseOptions 属性』.
- 第 575 页的『CommandInputQueueName 属性』.
- 第 575 页的『CommandLevel 属性』.
- 第 575 页的『CompletionCode 属性』.
- 第 575 页的『ConnectionHandle 属性』.
- 第 575 页的『ConnectionStatus 属性』.
- 第 576 页的『ConnectOptions 属性』.
- 第 576 页的『DeadLetterQueueName 属性』.
- 第 576 页的『DefaultTransmissionQueueName 属性』.
- 第 576 页的『Description 属性』.
- 第 576 页的『DistributionLists 属性』.
- 第 577 页的『InhibitEvent 属性』.
- 第 577 页的『IsConnected 属性』.
- 第 577 页的『IsOpen 属性』.
- 第 577 页的『LocalEvent 属性』.
- 第 577 页的『MaximumHandles 属性』.
- 第 578 页的『MaximumMessageLength 属性』.
- 第 578 页的『MaximumPriority 属性』.
- 第 578 页的『MaximumUncommittedMessages 属性』.
- 第 578 页的『Name 属性』.
- 第 578 页的『ObjectHandle 属性』.
- 第 578 页的『PerformanceEvent 属性』.
- 第 579 页的『Platform 属性』.
- 第 579 页的『ReasonCode 属性』.
- 第 579 页的『ReasonName 属性』.
- 第 579 页的『RemoteEvent 属性』.
- 第 579 页的『StartStopEvent 属性』.
- 第 580 页的『SyncPointAvailability 属性』.
- 第 580 页的『TriggerInterval 属性』.

方法

- 第 580 页的『AccessQueue 方法』.
- 第 581 页的『AddDistributionList 方法』.
- 第 581 页的『Backout 方法』.
- 第 581 页的『Begin 方法』.
- 第 581 页的『ClearErrorCodes 方法』.
- 第 581 页的『Commit 方法』.

- 第 581 页的『Connect 方法』.
- 第 582 页的『Disconnect 方法』.

属性访问

以下属性可随时访问。

- 第 573 页的『AlternateUserId 属性』.
- 第 575 页的『CompletionCode 属性』.
- 第 575 页的『ConnectionStatus 属性』.
- 第 579 页的『ReasonCode 属性』.

仅当对象连接到队列管理器，并且已授权用户标识对队列管理器进行查询时才可访问其余属性。 如果设置了备用用户标识并且授权当前用户标识使用它，那么会对备用用户标识进行检查授权以供查询

如果这些条件不适用，IBM MQ Automation Classes for ActiveX 会尝试连接队列管理器，并自动打开进行查询。 如果此操作不成功，那么调用设置 CompletionCode MQCC_FAILED 和下列 ReasonCode 之一：

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

AlternateUserId 属性

读 — 写。 用于确认对队列管理器属性的访问的备用用户标识。

如果 IsConnected 为 TRUE，不能设置此属性。

在对象打开时不能设置此属性。

Defined in: MQQueueManager 类

Data Type: 由 12 个字符组成的字符串

Syntax: 要获取: *altuser $= MQQueueManager* **.AlternateUserId** 要设置:
MQQueueManager **.AlternateUserId** = *altuser $*

AuthorityEvent 属性

只读。 MQI AuthorityEvent 属性。

定义于:
MQQueueManager 类

数据类型:
长

值:

- MQEVR_DISABLED
- MQEVR_ENABLED

语法: 获取: *authevent = MQQueueManager* **.AuthorityEvent**

BeginOptions 属性

读 — 写。 这些是应用于 Begin 方法的选项。 初始为 MQBO_NONE。

定义于:
MQQueueManager 类

数据类型:
长

值：

- MQBO_NONE

语法： 要获取: *beginoptions&=MQQueueManager*。 **BeginOptions**

要设置: *MQQueueManager* **.BeginOptions** = *beginoptions&*

ChannelAutoDefinition 属性

只读。 它控制是否允许自动通道定义。

定义于：
MQQueueManager 类

数据类型：
长

值：

- MQCHAD_DISABLED
- MQCHAD_ENABLED

语法： 要获取: *channelautodef&= MQQueueManager*。 **ChannelAutoDefinition**

ChannelAutoDefinitionEvent 属性

只读。 它控制是否生成自动通道定义事件。

定义于：
MQQueueManager 类

数据类型：
长

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法： 要获取: *channelautodefevent&=MQQueueManager*。 **ChannelAutoDefinitionEvent**

ChannelAutoDefinitionExit 属性

只读。 用于自动通道定义的用户出口名称。

定义于：
MQQueueManager 类

数据类型：
字符串

语法： 获取： *channelautodefexit$= MQQueueManager*. **ChannelAutoDefinitionExit**

CharacterSet 属性

只读。 MQI CodedCharSetId 属性。

定义于： MQQueueManager 类

数据类型： 长整型

语法： 要获取: *characterset&= MQQueueManager* **.CharacterSet**

CloseOptions 属性

读 — 写。 用于控制关闭队列管理器时所发生事件的选项。 初始值是 MQCO_NONE。

定义于：
MQQueueManager 类

数据类型：
长

值：

- MQCO_NONE

语法： 要获取: *closeopt& = MQQueueManager* **.CloseOptions**

要设置: *MQQueueManager* **.CloseOptions** = *closeopt&*

CommandInputQueueName 属性

只读。 MQI CommandInputQName 属性。

定义于： MQQueueManager 类

数据类型： 由 48 个字符组成的字符串

语法： 获取： *commandinputqname$ = MQQueueManager* **.CommandInputQueueName**

CommandLevel 属性

只读。 返回 IBM MQ 队列管理器实施（MQI CommandLevel 属性）的版本和级别。

定义于： MQQueueManager 类

数据类型： 长整型

语法： 要获取: *level& = MQQueueManager* **.CommandLevel**

CompletionCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的完成代码。

定义于： MQQueueManager 类

数据类型： 长整型

值：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法： 要获取: *completioncode& = MQQueueManager* **.CompletionCode**

ConnectionHandle 属性

只读。 IBM MQ 队列管理器对象的连接句柄。

定义于：
MQQueueManager 类

数据类型：
Long

语法： 要获取: *hconn& = MQQueueManager.* **ConnectionHandle**

ConnectionStatus 属性

只读。 表明对象是否连接到其队列管理器。

定义于： MQQueueManager 类

数据类型： 布尔

值：

- TRUE (-1)
- FALSE (0)

语法：获取：*status = MQQueueManager* **.ConnectionStatus**

ConnectOptions 属性

读 — 写。 这些选项应用于 Connect 方法。 初始为 MQCNO_NONE。

定义于：
MQQueueManager 类

数据类型：
长

值：

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

语法: 要获取: *connectoptions&=MQQueueManager*。 **ConnectOptions**

要设置: *MQQueueManager* **.ConnectOptions** = 连接和

DeadLetterQueueName 属性

只读。 MQI DeadLetterQName 属性。

定义于：MQQueueManager 类

数据类型：由 48 个字符组成的字符串

语法：获取：*dlqname$ = MQQueueManager* **.DeadLetterQueueName**

DefaultTransmissionQueueName 属性

只读。 MQI DefXmitQName 属性。

定义于：MQQueueManager 类

数据类型：由 48 个字符组成的字符串

语法：获取：*defxmitqname$ = MQQueueManager* **.DefaultTransmissionQueueName**

Description 属性

只读。 MQI QMgrDesc 属性。

定义于：MQQueueManager 类

数据类型：由 64 个字符组成的字符串

语法：获取：*description$ = MQQueueManager* **.Description**

DistributionLists 属性

只读。 这是队列管理器支持分发列表的功能。

定义于：
MQQueueManager 类

数据类型：
布尔

值：

- TRUE (-1)
- FALSE (0)

语法：获取：*distributionlists= MQQueueManager*. **DistributionLists**

InhibitEvent 属性

只读。 MQI InhibitEvent 属性。

定义于：MQQueueManager 类

数据类型：长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法：要获取: *inhibevent& = MQQueueManager* **.InhibitEvent**

IsConnected 属性

表明队列管理器当前是否连接的值。

只读。

定义于：MQQueueManager 类

数据类型：布尔

值：

- TRUE (-1)
- FALSE (0)

语法：获取：*isconnected = MQQueueManager* **.IsConnected**

IsOpen 属性

表明队列管理器当前是否打开以供查询的值。

只读。

定义于：
MQQueueManager 类

数据类型：
布尔

值：

- TRUE (-1)
- FALSE (0)

语法：获取：*IsOpen = MQQueueManager*. **IsOpen**

LocalEvent 属性

只读。 MQI LocalEvent 属性。

定义于：MQQueueManager 类

数据类型：长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法：要获取: *localevent& = MQQueueManager* **.LocalEvent**

MaximumHandles 属性

只读。 MQI MaxHandles 属性。

定义于： MQQueueManager 类

数据类型： 长整型

语法: 要获取: *maxhandles& = MQQueueManager* **.MaximumHandles**

MaximumMessageLength 属性

只读。 MQI MaxMsgLength Queue Manager 属性。

定义于： MQQueueManager 类

数据类型： 长整型

语法: 要获取: *maxmessagelength& = MQQueueManager* **.MaximumMessage 长度**

MaximumPriority 属性

只读。 MQI MaxPriority 属性。

定义于： MQQueueManager 类

数据类型： 长整型

语法: 要获取: *maxpriority& = MQQueueManager* **.MaximumPriority**

MaximumUncommittedMessages 属性

只读。 MQI MaxUncommittedMsgs 属性。

定义于： MQQueueManager 类

数据类型： 长整型

语法: 要获取: *maxuncommitted& = MQQueueManager* **.MaximumUncommitted 消息**

Name 属性

读 — 写。 MQI QMgrName 属性。 一旦连接了 MQQueueManager，就不能写此属性。

定义于： MQQueueManager 类

数据类型： 由 48 个字符组成的字符串

语法： 获取： *name$ = MQQueueManager* **.name**

设置： *MQQueueManager* **.name** = *name$*

注: Visual Basic 保留“Name”属性供可视界面中使用。 因此当在使用小写的 Visual Basic 中使用此属性时，它就是“name”。

ObjectHandle 属性

只读。 IBM MQ 队列管理器对象的对象句柄。

定义于：
MQQueueManager 类

数据类型
Long

语法: 要获取: *hobj& = MQQueueManager.* **ObjectHandle**

PerformanceEvent 属性

只读。 MQI PerformanceEvent 属性。

定义于： MQQueueManager 类

数据类型：长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法：要获取：*perfevent& = MQQueueManager*。PerformanceEvent

Platform 属性

只读。 MQI Platform 属性。

定义于：MQQueueManager 类

数据类型：长整型

值：

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

语法：要获取：*platform& = MQQueueManager* **. 平台**

ReasonCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的原因码。

定义于：MQQueueManager 类

数据类型：长整型

值：

- 请参阅 API 完成代码和原因码。

语法：要获取：*reasoncode& = MQQueueManager* **.ReasonCode**

ReasonName 属性

只读。 返回最新原因码的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。

定义于：MQQueueManager 类

数据类型：字符串

值：

- 请参阅 API 完成代码和原因码。

语法：获取：*reasonname$ = MQQueueManager* **.ReasonName**

RemoteEvent 属性

只读。 MQI RemoteEvent 属性。

定义于：MQQueueManager 类

数据类型：长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法：要获取：*remoteevent& = MQQueueManager* **.RemoteEvent**

StartStopEvent 属性

只读。 MQI StartStopEvent 属性。

定义于： MQQueueManager 类

数据类型： 长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法： 要获取: *strstpevent& = MQQueueManager* **.StartStop 事件**

SyncPointAvailability 属性

只读。 MQI SyncPoint 属性。

定义于： MQQueueManager 类

数据类型： 长整型

值：

- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

语法： 要获取: *syncpointavailability& = MQQueueManager* **.SyncPoint 可用性**

TriggerInterval 属性

只读。 MQI TriggerInterval 属性。

定义于： MQQueueManager 类

数据类型： 长整型

语法： 要获取: *trigint& = MQQueueManager* **.TriggerInterval**

AccessQueue 方法

创建一个 MQQueue 对象，并通过设置此队列的连接参考属性，将其与这个 MQQueueManager 对象相关联。 它将 MQQueue 对象的 Name、OpenOptions、DynamicQueueName 和 AlternateUserId 属性设置为提供的值，并尝试打开它。

如果打开不成功，那么调用失败。 会对此对象引发一个错误事件。 已设置对象的 ReasonCode 和 CompletionCode，以及 MQSession ReasonCode 和 CompletionCode。

DynamicQueueName、QueueManagerName 和 AlternateUserId 参数是可选的，并缺省为 ""。

如果要读取队列属性，除其他选项外还应指定 OpenOption MQOO_INQUIRE。

如果要打开的队列是本地的，那么不要设置 QueueManagerName，或应将其设置为 ""。 否则，将其设置为拥有该列队的远程队列管理器的名称，并尝试打开远程队列的本地定义。 有关远程队列名称解析和队列管理器别名判别的更多信息，请参阅什么是别名？ 。

如果 Name 属性设置为模型队列名称，请在 DynamicQueueName$ 参数中指定要创建的动态队列的名称。 如果在 DynamicQueueName$ 参数中提供的值为 ""，那么设置到队列对象中并在打开调用时使用的值就是 “AMQ.*”。 请参阅第 699 页的『创建动态队列』，以了解有关命名动态队列的更多信息。

定义

定义于：MQQueueManager 类。

语法

语法： set queue = MQQueueManager. **AccessQueue** (Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

参数

Name$ 字符串。 IBM MQ 队列的名称。

OpenOptions: 长整型。 当队列打开时要使用的选项。 请参阅 OpenOptions (MQLONG)。

QueueManagerName$ 字符串。 拥有要打开的队列的队列管理器名。 值 "" 暗示队列管理器是本地的。

*DynamicQueueName$* 字符串。 如果 Name$ 参数指定模型队列，在队列打开时对动态队列指定的名称。

AlternateUserId$ 字符串。 在打开队列时，用于确认访问权的备用用户标识。

AddDistributionList 方法

创建新的 MQDistributionList 对象并将其连接引用设置为属主队列管理器。

定义于：
MQQueueManager 类

语法： *set distributionlist* = **MQQueueManager.** *AddDistributionList*

Backout 方法

回退自最后一个同步点以来作为工作单元一部分发生的任何未落实的消息放入和取出。

定义于： MQQueueManager 类

语法：

```
Call MQQueueManager.Backout()
```

Begin 方法

开始由队列管理器协调的工作单元。 begin 选项影响此方法的行为。

定义于：
MQQueueManager 类

语法：

```
Call MQQueueManager.Begin()
```

ClearErrorCodes 方法

为 MQQueueManager 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于：
MQQueueManager 类

语法：
Call *MQQueueManager* **.ClearErrorCodes()**

Commit 方法

落实任何自最后一个同步点以来作为工作单元一部分发生的消息放入和取出。

定义于： MQQueueManager 类

语法：

```
Call MQQueueManager.Commit()
```

Connect 方法

通过 IBM MQ MQI client 或服务器将 MQQueueManager 对象连接到实际队列管理器。 进行连接时，此方法还会打开队列管理器对象，以便能够对其进行查询。

将 IsConnected 设置为 TRUE。

最多允许每个 ActiveX 实例有一个 MQQueueManager 对象连接到一个队列管理器。

定义于： MQQueueManager 类

语法：

```
Call MQQueueManager.Connect()
```

Disconnect 方法

将 MQQueueManager 对象从队列管理器断开连接。

将 IsConnected 设置为 FALSE。

使所有与 MQQueueManager 对象关联的队列对象都不可用并无法重新打开。

落实任何未落实的更改（消息放入和取出）。

定义于： MQQueueManager 类

语法：

```
Call MQQueueManager.Disconnect()
```

MQQueue 类

此类代表对 IBM MQ 队列的访问。 此连接由关联的 MQQueueManager 对象提供。 当此类的对象被破坏时，它会自动关闭。

包含

MQQueueManager 类中包含 MQQueue 类。

创建

New 创建新的 MQQueue 对象，并将所有属性设置为初始值。 也可以使用 MQQueueManager 类的 AccessQueue 方法。

语法

```
Dim que As New MQQueue Set que = New MQQueue
```

属性

- 第 585 页的『AlternateUserId 属性』.
- 第 585 页的『BackoutRequeueName 属性』.
- 第 585 页的『BackoutThreshold 属性』.
- 第 585 页的『BaseQueueName 属性』.
- 第 585 页的『CloseOptions 属性』.
- 第 586 页的『CompletionCode 属性』.
- 第 586 页的『ConnectionReference 属性』.
- 第 586 页的『CreationDateTime 属性』.

- 第 586 页的『CurrentDepth 属性』.
- 第 586 页的『DefaultInputOpenOption 属性』.
- 第 587 页的『DefaultPersistence 属性』.
- 第 587 页的『DefaultPriority 属性』.
- 第 587 页的『DefinitionType 属性』.
- 第 587 页的『DepthHighEvent 属性』.
- 第 587 页的『DepthHighLimit 属性』.
- 第 587 页的『DepthLowEvent 属性』.
- 第 588 页的『DepthLowLimit 属性』.
- 第 588 页的『DepthMaximumEvent 属性』.
- 第 587 页的『DepthHighEvent 属性』.
- 第 587 页的『DepthHighLimit 属性』.
- 第 587 页的『DepthLowEvent 属性』.
- 第 588 页的『DepthLowLimit 属性』.
- 第 588 页的『DepthMaximumEvent 属性』.
- 第 588 页的『Description 属性』.
- 第 588 页的『DynamicQueueName 属性』.
- 第 588 页的『HardenGetBackout 属性』.
- 第 589 页的『InhibitGet 属性』.
- 第 589 页的『InhibitPut 属性』.
- 第 589 页的『InitiationQueueName 属性』.
- 第 589 页的『IsOpen 属性』.
- 第 589 页的『MaximumDepth 属性』.
- 第 590 页的『MaximumMessageLength 属性』.
- 第 590 页的『MessageDeliverySequence 属性』.
- 第 590 页的『ObjectHandle 属性』.
- 第 590 页的『OpenInputCount 属性』.
- 第 591 页的『OpenOptions 属性』.
- 第 591 页的『OpenOutputCount 属性』.
- 第 591 页的『OpenStatus 属性』.
- 第 591 页的『ProcessName 属性』.
- 第 591 页的『QueueManagerName 属性』.
- 第 592 页的『QueueType 属性』.
- 第 592 页的『ReasonCode 属性』.
- 第 592 页的『ReasonName 属性』.
- 第 592 页的『RemoteQueueManagerName 属性』.
- 第 592 页的『RemoteQueueName 属性』.
- 第 592 页的『ResolvedQueueManagerName 属性』.
- 第 593 页的『ResolvedQueueName 属性』.
- 第 593 页的『RetentionInterval 属性』.
- 第 593 页的『Scope 属性』.
- 第 593 页的『ServiceInterval 属性』.
- 第 593 页的『ServiceIntervalEvent 属性』.

- 第 593 页的『Shareability 属性』.
- 第 594 页的『TransmissionQueueName 属性』.
- 第 594 页的『TriggerControl 属性』.
- 第 594 页的『TriggerData 属性』.
- 第 594 页的『TriggerDepth 属性』.
- 第 594 页的『TriggerMessagePriority 属性』.
- 第 595 页的『TriggerType 属性』.
- 第 595 页的『Usage 属性』.

方法

- 第 595 页的『ClearErrorCodes 方法』
- 第 595 页的『Close 方法』
- 第 595 页的『Get 方法』
- 第 596 页的『Open 方法』
- 第 596 页的『Put 方法』

属性访问

如果队列对象未连接到队列管理器，您可以读取以下属性：

- 第 586 页的『CompletionCode 属性』
- 第 591 页的『OpenStatus 属性』
- 第 592 页的『ReasonCode 属性』

并且您可以读和写：

- 第 585 页的『AlternateUserId 属性』
- 第 585 页的『CloseOptions 属性』
- 第 586 页的『ConnectionReference 属性』
- 第 590 页的『Name 属性』
- 第 591 页的『OpenOptions 属性』

如果队列对象已连接到队列管理器，您可以读取所有属性。

队列属性 Attribute 属性

未在前一部分中列出的属性都是基本 IBM MQ 队列的属性。 仅当对象连接到队列管理器，并且已授权用户的用户标识对该队列进行查询或设置时才可访问它们。 如果设置了备用用户标识，并且授权当前用户标识使用它，那么会对备用用户标识进行检查以供授权。

属性必须是给定 QueueType 的相应属性。 请参阅队列属性以了解更多信息。

如果这些条件不适用，那么属性访问将设置 CompletionCode MQCC_FAILED 和下列 ReasonCode 之一：

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE（CompletionCode 是 MQCC_WARNING）

打开队列

创建 MQQueue 对象的办法只有使用 MQQueueManager AccessQueue 方法或“新建”。 打开的 MQQueue 对象会保持打开（OpenStatus=TRUE）直到它被关闭或删除，或直到删除创建队列管理器对象或丢失与队列管理器的连接。 MQQueue CloseOptions 属性的值控制当删除 MQQueue 对象时发生的关闭操作的行为。

MQQueueManager AccessQueue 方法使用 OpenOptions 参数打开队列。 MQQueue.Open 方法使用 OpenOptions 属性打开队列。 IBM MQ 会在打开队列过程中根据用户授权来验证 OpenOptions。

AlternateUserId 属性

读 — 写。 在队列打开时，用于确认对其访问权的备用用户标识。

当对象打开时（即 IsOpen 为 TRUE 时）不能设置此属性。

定义于： MQQueue 类

数据类型： 由 12 个字符组成的字符串

语法： 获取：*altuser$ = MQQueue* **.AlternateUserId**

设置：*MQQueue.* **AlternateUserId** = *altuser$*

BackoutRequeueName 属性

只读。 MQI BackOutRequeueQName 属性。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*backoutrequeuename$ = MQQueue* **.BackoutRequeueName**

BackoutThreshold 属性

只读。 MQI BackoutThreshold 属性。

定义于： MQQueue 类

数据类型： 长整型

值：

- 请参阅 BackoutThreshold (MQLONG)。

语法: 要获取: *backoutthreshold& = MQQueue.* **BackoutThreshold**

BaseQueueName 属性

只读。 别名所指的队列名。

仅对别名队列有效。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*baseqname$ = MQQueue* **.BaseQueueName**

CloseOptions 属性

读 — 写。 用于控制队列关闭时所发生事件的选项。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQCO_NONE
- MQCO_DELETE

- MQCO_DELETE_PURGE

MQCO_DELETE 和 MQCO_DELETE_PURGE 仅对动态队列有效。

语法: 要获取: *closeopt& = MQQueue* **.CloseOptions**

要设置: *MQ* 队列 **.CloseOptions** = *closeopt&*

CompletionCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的完成代码。

定义于: MQQueue 类

数据类型: 长整型

值:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法: 要获取: *completioncode& = MQQueue* **.CompletionCode**

ConnectionReference 属性

读 — 写。 定义队列对象所属的队列管理器对象。 在队列打开时不能写连接引用。

定义于: MQQueue 类

数据类型: MQQueueManager

值:

- 对活动 IBM MQ 队列管理器对象的引用

语法: 设置: *set MQQueue* **.ConnectionReference** = *ConnectionReference*

获取: *set ConnectionReference = MQQueue* **.ConnectionReference**

CreationDateTime 属性

只读。 创建此队列的日期和时间。

定义于: MQQueue 类

数据类型: 类型 7 的变体（日期/时间）或 EMPTY

语法: 获取: *datetime = MQQueue* **.CreationDateTime**

CurrentDepth 属性

只读。 当前在队列上的消息数。

定义于: MQQueue 类

数据类型: 长整型

语法: 要获取: *currentdepth& = MQQueue* **.CurrentDepth**

DefaultInputOpenOption 属性

只读。 如果 OpenOptions 指定 MQOO_INPUT_AS_Q_DEF，那么它控制打开队列的方式。

定义于: MQQueue 类

数据类型: 长整型

值:

- MQOO_INPUT_EXCLUSIVE

- MQOO_INPUT_SHARED

语法: 要获取: *defaultinop& = MQQueue* **.DefaultInputOpenOption**

DefaultPersistence 属性

只读。 队列中消息的缺省持久状态。

定义于: MQQueue 类

数据类型: 长整型

语法: 要获取: *defpersistence& = MQQueue* **.DefaultPersistence**

DefaultPriority 属性

只读。 队列中消息的缺省优先级。

定义于: MQQueue 类

数据类型: 长整型

语法: 要获取: *defpriority& = MQQueue* **.DefaultPriority**

DefinitionType 属性

只读。 队列定义类型。

定义于: MQQueue 类

数据类型: 长整型

值:

- MQQDT_PREDEFINED
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

语法: 要获取: *deftype& = MQQueue* **.DefinitionType**

DepthHighEvent 属性

只读。 MQI QDepthHighEvent 属性。

定义于: MQQueue 类

数据类型: 长整型

值:

- MQEVR_DISABLED
- MQEVR_ENABLED

语法: 要获取: *depthhighevent& = MQQueue.* **DepthHighEvent**

DepthHighLimit 属性

只读。 MQI QDepthHighLimit 属性。

定义于: MQQueue 类

数据类型: 长整型

语法: 要获取: *depthhighlimit& = MQQueue.* **DepthHighLimit**

DepthLowEvent 属性

只读。 MQI QDepthLowEvent 属性。

定义于: MQQueue 类

数据类型：长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法：要获取：*depthlowevent& = MQQueue.* **DepthLowEvent**

DepthLowLimit 属性

只读。 MQI QDepthLowLimit 属性。

定义于：MQQueue 类

数据类型：长整型

语法：要获取：*depthlowlimit& = MQQueue.* **DepthLowLimit**

DepthMaximumEvent 属性

只读。 MQI QDepthMaxEvent 属性。

定义于：MQQueue 类

数据类型：长整型

值：

- MQEVR_DISABLED
- MQEVR_ENABLED

语法：要获取：*depthmaximumevent& = MQQueue.* **DepthMaximumEvent**

Description 属性

只读。 队列的描述。

定义于：MQQueue 类

数据类型：由 64 个字符组成的字符串

语法：获取：*description$ = MQQueue* **.Description**

DynamicQueueName 属性

读 — 写，当队列打开时是只读。

它控制当打开模型队列时使用的动态队列名称。 用户可以使用通配符将它作为属性集（仅当队列关闭时）设置或作为 MQQueueManager.AccessQueue() 的参数设置。

动态队列的实际名称是通过查询 QueueName 找到的。

定义于：MQQueue 类

数据类型：由 48 个字符组成的字符串

值：

- 任何有效的 IBM MQ 队列名称。

语法：设置：*MQQueue* **.DynamicQueueName** = *dynamicqueuename$*

获取：*dynamicqueuename$* = *MQQueue* **.DynamicQueueName**

HardenGetBackout 属性

只读。 是否维护精确的回退计数。

定义于：MQQueue 类

数据类型：长整型

值：

- MQQA_BACKOUT_HARDENED
- MQQA_BACKOUT_NOT HARDENED

语法：要获取：*hardengetback& = MQQueue* **.HardenGet 回退**

InhibitGet 属性

读 — 写。 MQI InhibitGet 属性。

定义于：MQQueue 类

数据类型：长整型

值：

- MQQA_GET_INHIBITED
- MQQA_GET_ALLOWED

语法：要获取：*getstatus& = MQQueue* **.InhibitGet**

要设置：*MQ* 队列 **.InhibitGet** = *getstatus&*

InhibitPut 属性

读 — 写。 MQI InhibitPut 属性。

定义于：MQQueue 类

数据类型：长整型

值：

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

语法：要获取：*putstatus& = MQQueue* **.InhibitPut**

要设置：*MQ* 队列 **.InhibitPut** = *putstatus&*

InitiationQueueName 属性

只读。 启动队列的名称。

定义于：MQQueue 类

数据类型：由 48 个字符组成的字符串

语法：获取：*initqname$ = MQQueue* **.InitiationQueueName**

IsOpen 属性

返回队列是否打开。

只读。

定义于：MQQueue 类

数据类型：布尔

值：

- TRUE (-1)
- FALSE (0)

语法：获取：*open = MQQueue* **.IsOpen**

MaximumDepth 属性

只读。 最大队列深度。

定义于： MQQueue 类

数据类型： 长整型

语法: 要获取: *maxdepth& = MQQueue* **.MaximumDepth**

MaximumMessageLength 属性

只读。 对此队列允许的最大消息长度，以字节为单位。

定义于： MQQueue 类

数据类型： 长整型

语法: 要获取: *maxmlength& = MQQueue* **.MaximumMessage 长度**

MessageDeliverySequence 属性

只读。 消息传递顺序。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQMDS_PRIORITY
- MQMDS_FIFO

语法: 要获取: *messdelseq& = MQQueue* **.MessageDelivery 序列**

Name 属性

读 — 写。 MQI Queue 属性。 在打开 MQQueue 后，就不能写此属性。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取： *name$ = MQQueue* **.name**

设置： *MQQueue* **.name** = *name$*

注: Visual Basic 保留“Name”属性供可视界面中使用。 因此当在使用小写的 Visual Basic 中使用此属性时，它就是“name”。

ObjectHandle 属性

只读。 IBM MQ 队列对象的对象句柄。

定义于： MQQueue 类

数据类型： 长整型

语法: 要获取: *hobj& = MQQueue*。 **ObjectHandle**

OpenInputCount 属性

只读。 用于输入的打开数。

定义于： MQQueue 类

数据类型： 长整型

语法： 获取：

```
openincount& = MQQueue.OpenInputCount
```

OpenOptions 属性

读 — 写。 要用于打开队列的选项。

定义于： MQQueue 类

数据类型： 长整型

值：

- 请参阅 OpenOptions (MQLONG)。

语法： 获取：

```
openopt& = MQQueue.OpenOptions
```

设置：*MQQueue.* **OpenOptions** = *openopt&*

OpenOutputCount 属性

只读。 用于输出的打开数。

定义于： MQQueue 类

数据类型： 长整型

语法： 获取：

```
openoutcount& = MQQueue.OpenOutputCount
```

OpenStatus 属性

只读。 表明队列是否打开。 在 AccessQueue 方法后初始值是 TRUE，在“新建”后初始值是 FALSE。

定义于： MQQueue 类

数据类型： 布尔

值：

- TRUE (-1)
- FALSE (0)

语法： 获取：

```
status& = MQQueue.OpenStatus
```

ProcessName 属性

只读。 MQI ProcessName 属性。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*procname$ =MQQueue* **.ProcessName**

QueueManagerName 属性

读 — 写。 IBM MQ 队列管理器名称。

定义于： MQQueue 类

数据类型： 字符串

语法： 获取：*QueueManagerName$* = *MQQueue* **.QueueManagerName**

设置：*MQQueue* **.QueueManagerName** = *QueueManagerName$*

QueueType 属性

只读。 MQI QType 属性。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQQT_ALIAS
- MQQT_LOCAL
- MQQT_MODEL
- MQQT_REMOTE

语法： 要获取: *queuetype& = MQQueue* **.QueueType**

ReasonCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的原因码。

定义于： MQQueue 类

数据类型： 长整型

值：

- 请参阅 API 完成代码和原因码。

语法： 要获取: *reasoncode& = MQQueue* **.ReasonCode**

ReasonName 属性

只读。 返回最新原因码的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。

定义于： MQQueue 类

数据类型： 字符串

值：

- 请参阅 API 完成代码和原因码。

语法： 获取：*reasonname$ = MQQueue* **.ReasonName**

RemoteQueueManagerName 属性

只读。 远程队列管理器的名称。 仅对远程队列有效。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*remqmanname$ = MQQueue* **.RemoteQueueManagerName**

RemoteQueueName 属性

只读。 在远程队列管理器上所知的队列名称。 仅对远程队列有效。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*remqname$ = MQQueue* **.RemoteQueueName**

ResolvedQueueManagerName 属性

只读。 本地队列管理器所知的最终目标队列管理器名称。

定义于：MQQueue 类

数据类型：由 48 个字符组成的字符串

语法：获取：*resqmanname$ = MQQueue* **.ResolvedQueueManagerName**

ResolvedQueueName 属性

只读。 本地队列管理器所知的最终目标队列名称。

定义于：MQQueue 类

数据类型：由 48 个字符组成的字符串

语法：获取：*resqname$ = MQQueue* **.ResolvedQueueName**

RetentionInterval 属性

只读。 队列应当被保留的一段时间。

定义于：MQQueue 类

数据类型：长整型

语法：要获取: *retinterval& = MQQueue* **.RetentionInterval**

Scope 属性

只读。 控制此队列的条目是否也在单元目录中存在。

定义于：MQQueue 类

数据类型：长整型

值：

- MQSCO_Q_MGR
- MQSCO_CELL

语法：要获取: *scope& = MQQueue* **. 范围**

ServiceInterval 属性

只读。 MQI QServiceInterval 属性。

定义于：MQQueue 类

数据类型：长整型

语法：要获取: *serviceinterval& = MQQueue.* **ServiceInterval**

ServiceIntervalEvent 属性

只读。 MQI QServiceIntervalEvent 属性。

定义于：MQQueue 类

数据类型：长整型

值：

- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

语法：要获取: *serviceintervalevent& = MQQueue.* **ServiceIntervalEvent**

Shareability 属性

只读。 队列可共享性。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

语法： 要获取: *shareability& = MQQueue* **. 可共享性**

TransmissionQueueName 属性

只读。 传输队列名称。 仅对远程队列有效。

定义于： MQQueue 类

数据类型： 由 48 个字符组成的字符串

语法： 获取： *transqname$ = MQQueue* **.TransmissionQueueName**

TriggerControl 属性

读 — 写。 触发器控制。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQTC_OFF
- MQTC_ON

语法： 要获取: *trigcontrol& = MQQueue* **.TriggerControl**

要设置: *MQ* 队列 **.TriggerControl** = 三联控制公司

TriggerData 属性

读 — 写。 触发器数据。

定义于： MQQueue 类

数据类型： 由 64 个字符组成的字符串

语法： 获取： *trigdata$ = MQQueue* **.TriggerData**

设置： *MQQueue* **.TriggerData** = *trigdata$*

TriggerDepth 属性

读 — 写。 在写触发器消息之前必须位于队列上的消息数。

定义于： MQQueue 类

数据类型： 长整型

语法： 要获取: *trigdepth& = MQQueue* **.TriggerDepth**

要设置: *MQ* 队列 **.TriggerDepth** = *trigdepth&*

TriggerMessagePriority 属性

读 — 写。 触发器的阈值消息优先级。

定义于： MQQueue 类

数据类型： 长整型

语法： 要获取: *trigmesspriority& = MQQueue* **.TriggerMessage 优先级**

要设置: *MQ* 队列 **.TriggerMessage 优先级** = 特里梅斯普里亚特公司

TriggerType 属性

读 — 写。 触发器类型。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

语法: 要获取: *trigtype& = MQQueue* **.TriggerType**

要设置: *MQ* 队列 **.TriggerType** = *Trigtype&*

Usage 属性

只读。 表明队列将作何用。

定义于： MQQueue 类

数据类型： 长整型

值：

- MQUS_NORMAL
- MQUS_TRANSMISSION

语法: 要获取: *usage& = MQQueue* **. 使用情况**

ClearErrorCodes 方法

为 MQQueue 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于： MQQueue 类

语法：

```
Call MQQueue.ClearErrorCodes()
```

Close 方法

使用 CloseOptions 的当前值关闭队列。

定义于： MQQueue 类

语法：

```
Call MQQueue.Close()
```

Get 方法

从队列检索消息。

此方法将 MQMessage 对象中的某些字段用作输入参数，从而将该对象的 MQMD 中的某些字段作为参数使用。 尤其是由于使用了 MessageId 和 CorrelId 字段，因此，一定要确保按照要求设置这些字段。 有关这些字段的更多信息，请参阅 MsgId (MQBYTE24) 和 CorrelId (MQBYTE24)。

如果方法失败，那么 MQMessage 对象不变。 如果成功，MQMessage 对象的 MQMD 和消息数据部分会被替换为来自传入消息的 MQMD 和消息数据。 MQMessage 控制属性的设置按如下所示

- **MessageLength** 设置为 IBM MQ 消息的长度
- **DataLength** 设置为 IBM MQ 消息的长度
- **DataOffset** 设置为零

定义于：
MQQueue 类

语法：

```
Call MQQueue.Get(Message, GetMsgOptions, GetMsgLength)
```

参数
消息：

代表要检索的消息的 MQMessage 对象。

GetMsgOptions：

用于控制获取操作的可选 MQGetMessageOptions 对象。 如果未指定此参数，将使用缺省 MQGetMessageOptions。

GetMsgLength：

可选的 2 或 4 字节长度值来控制从队列检索的 IBM MQ 消息的最大长度。

如果指定了 MQGMO_ACCEPT_TRUNCATED_MSG 选项，并且消息大小超出指定长度，那么，GET 会成功，并显示完成代码 MQCC_WARNING 和原因码 MQRC_TRUNCATED_MSG_ACCEPTED。

MessageData 保存数据的前 GetMsgLength 个字节。

如果 **未** 指定 MQGMO_ACCEPT_TRUNCATED_MSG，而消息大小超出指定长度，那么返回完成代码 MQCC_FAILED 和原因码 MQRC_TRUNCATED_MESSAGE_FAILED。

如果未定义消息缓冲区的内容，那么总计消息长度设置为要检索的消息的全长。

如果未指定消息长度参数，那么消息缓冲区的长度自动调整为至少达到入局消息的大小。

Open 方法

使用以下属性的当前值打开队列：

1. QueueName
2. 队列管理器名称
3. AlternateUserId
4. DynamicQueueName

定义于：
MQQueue 类

语法：

```
Call MQQueue.Open()
```

Put 方法

将消息放入队列。

此方法使用 MQMessage 对象作为参数。 使用此方法的结果是该对象的消息描述符 (MQMD) 属性可能会改变。 运行此方法后立即获得的值是放入 IBM MQ 中的值。

在完成 Put 方法后对 MQMessage 对象进行修改不会影响 IBM MQ 队列上的实际消息。

定义于：
MQQueue 类

语法：

```
Call MQQueue.Put(Message, PutMsgOptions)
```

参数
消息

表示要放入的消息的 MQMessage 对象。

PutMsgOptions

包含控制放入操作的选项的 MQPutMessageOptions 对象。 如果未指定它们，那么使用缺省 PutMessageOptions。

MQMessage 类

此类代表一个 IBM MQ 消息。 它包含一些可封装 IBM MQ 消息描述符 (MQMD) 的属性，并提供了一个缓冲区，用于保存应用程序定义的消息数据。

此类中包含一些 Write 方法，可将数据从 ActiveX 应用程序复制到 MQMessage 对象。 同样，该类中也包含一些 Read 方法，可将 MQMessage 对象中的数据复制到 ActiveX 应用程序。 该类自动管理缓冲区内存的分配和释放。 MQMessage 对象创建时应用程序不必声明缓冲区的大小，因为缓冲区会随着写入数据的增加而扩大。

如果缓冲区大小超出 IBM MQ 队列的 MaximumMessageLength 属性指定的大小，那么无法将消息放到该队列中。

构建 MQMessage 对象后，可以使用 MQQueue.Put 方法将该对象 Put 到 IBM MQ 队列中。 此方法会复制该对象的 MQMD 和消息数据部分，并将副本放到队列中。 因此，应用程序可以在 Put 操作之后修改或删除 MQMessage 对象，而不会影响 IBM MQ 队列上的消息。 队列管理器在将消息复制到 IBM MQ 队列中时，可以调整 MQMD 中的某些字段。

可以使用 MQQueue.Get 方法将入局消息读取到 MQMessage 对象中。 这会使用来自入局消息的值替换 MQMessage 对象中任何可能已经存在的 MQMD 或消息数据。 它会调整 MQMessage 对象的数据缓冲区大小，以匹配入局消息数据的大小。

包含

MQSession 类包含消息。

创建

新建可创建 MQMessage 对象。 它的消息描述符属性的初始设置为缺省值，并且它的消息数据缓冲区是空的。

语法

```
Dim msg As New MQMessage
```

或者

```
Set msg = New MQMessage
```

属性

控制属性为：

- 第 600 页的『CompletionCode 属性』
- 第 600 页的『DataLength 属性』
- 第 600 页的『DataOffset 属性』
- 第 601 页的『MessageLength 属性』
- 第 601 页的『ReasonCode 属性』
- 第 601 页的『ReasonName 属性』

消息描述符属性为：

- 第 601 页的『AccountingToken 属性』
- 第 601 页的『AccountingTokenHex 属性』
- 第 602 页的『ApplicationIdData 属性』
- 第 602 页的『ApplicationOriginData 属性』
- 第 602 页的『BackoutCount 属性』
- 第 602 页的『CharacterSet 属性』
- 第 603 页的『CorrelationId 属性』
- 第 603 页的『CorrelationIdHex 属性』
- 第 603 页的『Encoding 属性』
- 第 604 页的『Expiry 属性』
- 第 604 页的『Feedback 属性』
- 第 604 页的『Format 属性』
- 第 605 页的『GroupId 属性』
- 第 605 页的『GroupIdHex 属性』
- 第 605 页的『MessageData 属性』
- 第 605 页的『MessageFlags 属性』
- 第 606 页的『MessageId 属性』
- 第 606 页的『MessageIdHex 属性』
- 第 606 页的『MessageSequenceNumber 属性』
- 第 606 页的『MessageType 属性』
- 第 607 页的『Offset 属性』
- 第 607 页的『OriginalLength 属性』
- 第 607 页的『Persistence 属性』
- 第 607 页的『Priority 属性』
- 第 607 页的『PutApplicationName 属性』
- 第 608 页的『PutApplicationType 属性』
- 第 608 页的『PutDateTime 属性』
- 第 608 页的『ReplyToQueueManagerName 属性』
- 第 608 页的『ReplyToQueueName 属性』
- 第 609 页的『Report 属性』
- 第 609 页的『TotalMessageLength 属性』
- 第 609 页的『UserId 属性』

方法

- 第 609 页的『ClearErrorCodes 方法』
- 第 609 页的『ClearMessage 方法』

- 第 610 页的『Read 方法』
- 第 610 页的『ReadBoolean 方法』
- 第 610 页的『ReadByte 方法』
- 第 610 页的『ReadDecimal2 方法』
- 第 610 页的『ReadDecimal4 方法』
- 第 611 页的『ReadDouble 方法』
- 第 611 页的『ReadDouble4 方法』
- 第 611 页的『ReadFloat 方法』
- 第 611 页的『ReadInt2 方法』
- 第 611 页的『ReadInt4 方法』
- 第 612 页的『ReadLong 方法』
- 第 612 页的『ReadNullTerminatedString 方法』
- 第 612 页的『ReadShort 方法』
- 第 612 页的『ReadString 方法』
- 第 613 页的『ReadUInt2 方法』
- 第 613 页的『ReadUnsignedByte 方法』
- 第 613 页的『ReadUTF 方法』
- 第 613 页的『ResizeBuffer 方法』
- 第 614 页的『Write 方法』
- 第 614 页的『WriteBoolean 方法』
- 第 614 页的『WriteByte 方法』
- 第 614 页的『WriteDecimal2 方法』
- 第 615 页的『WriteDecimal4 方法』
- 第 615 页的『WriteDouble 方法』
- 第 615 页的『WriteDouble4 方法』
- 第 615 页的『WriteFloat 方法』
- 第 616 页的『WriteInt2 方法』
- 第 616 页的『WriteInt4 方法』
- 第 616 页的『WriteLong 方法』
- 第 616 页的『WriteNullTerminatedString 方法』
- 第 617 页的『WriteShort 方法』
- 第 617 页的『WriteString 方法』
- 第 617 页的『WriteUInt2 方法』
- 第 617 页的『WriteUnsignedByte 方法』
- 第 618 页的『WriteUTF 方法』

属性访问

任何时候都可读取所有属性。

除了 DataOffset 是读/写的以外，所有控制属性都是只读的。 除了 BackoutCount 和 TotalMessageLength 都是只读的之外，所有消息描述符属性都是读/写的。

但是请注意，在将消息放入 IBM MQ 队列中时，队列管理器可能会修改某些 MQMD 属性。 请参阅 MQMD 中的字段，以了解有关如何修改这些字段的详细信息。

数据转换

通过将 **CharacterSet** 属性设置为与队列管理器 (MQCCSI_Q_MGR) 的编码字符集标识匹配，并将数据作为字符串传递到消息，可以将二进制数据传递到 IBM MQ 消息。 如果该字符串需要包含 Unicode 或 ASCII 编码数字，您可以使用 chr$ 函数将它们转换为字符串格式。

Read 和 Write 方法执行数据转换。 这两种方法可以在 ActiveX 内部格式，以及由来自消息描述符的 Encoding 和 CharacterSet 属性定义的 IBM MQ 消息格式之间转换。 在写消息时，应在发出 Write 方法前在 Encoding 和 CharacterSet 中设置值以匹配消息接收方的特征。 读消息时通常不必这行此步骤，因为已经从传入 MQMD 中的那些属性设置了这些值。

这是在 MQQueue.Get 方法执行的任何转换之后发生的附加数据转换步骤。

CompletionCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的 IBM MQ 完成代码。

定义于： MQMessage 类

数据类型： 长整型

值：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法: 要获取: *completioncode& = MQMessage* **.CompletionCode**

DataLength 属性

只读。 此属性返回值:

```
MQMessage.MessageLength - MQMessage.DataOffset
```

它可在 Read 方法之前使用，用于检查期待的字符数是否实际存在于缓冲区中。

初始值为零。

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *bytesleft& = MQMessage* **.DataLength**

DataOffset 属性

读 — 写。 消息对象的消息数据部分中的当前位置。

该值表达为从消息数据缓冲区开始处计的字节偏移量；缓冲区中的第一个字符对应的是 DataOffset 值零。

read 或 write 方法在 DataOffset 引用的字符处开始其操作。 这些方法从此位置逐个处理缓冲区中的数据，并更新 DataOffset 以指向最后一个被处理的字节后面紧挨着的字节（如果有的话）。

DataOffset 只能接受零到 MessageLength 之间的值（包含两者）。 当 DataOffset = MessageLength 时它指向结尾，即缓冲区的第一个无效字符。 在此情况下允许 Write 方法 — 它们扩展缓冲区中的数据并使 MessageLength 增加所添加的字节数。 缓冲区结尾以外的读取都是无效的。

初始值为零。

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *currpos& = MQMessage* **.DataOffset**

要设置: *MQ* 消息 **.DataOffset** = 柯尔波斯公司

MessageLength 属性

只读。 返回消息对象的消息数据部分的总计长度，以字符为单位，不考虑 DataOffset 的值。

初始值为零。 在引用此消息对象的 Get 方法调用后，它设置为入局消息长度。 如果应用程序使用 Write 方法将数据添加到对象，它就会递增。 Read 方法对它没有影响。

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *msglength& = MQMessage* **.MessageLength**

ReasonCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的原因码。

定义于： MQMessage 类

数据类型： 长整型

值：

- 请参阅 API 完成代码和原因码。

语法: 要获取: *reasoncode& = MQMessage* **.ReasonCode**

ReasonName 属性

只读。 返回最新原因码的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。 **定义于：** MQMessage 类

数据类型： 字符串

值：

- 请参阅 API 完成代码和原因码。

语法： 获取： *reasonname$ = MQMessage* **.ReasonName**

AccountingToken 属性

读 — 写。 MQMD AccountingToken — 消息身份上下文的一部分。

其初始值全为空。

定义于： MQMessage 类

数据类型： 由 32 个字符组成的字符串

语法： 获取： *actoken$ = MQMessage* **.AccountingToken**

设置： *MQMessage.* **AccountingToken** = *actoken$*

请参阅第 563 页的『消息描述符属性』， 了解关于何时必须使用 AccountingTokenHex 代替 AccountingToken 属性的更多信息。

AccountingTokenHex 属性

读 — 写。 MQMD AccountingToken — 消息身份上下文的一部分。

每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 64 个有效的十六进制字符。

其初始值为“0.....0”

定义于： MQMessage 类

数据类型： 由 64 个十六进制字符组成的字符串，表示 32 个 ASCII 字符

语法： 获取： *actokenh$ = MQMessage* **.AccountingTokenHex**

设置：*MQMessage.* **AccountingTokenHex** = *actokenh$*

请参阅第 563 页的『消息描述符属性』， 了解关于何时必须使用 AccountingTokenHex 代替 AccountingToken 属性的更多信息。

ApplicationIdData 属性

读 — 写。 MQMD ApplIdentityData — 消息身份上下文的一部分。

其初始值全为空白。

定义于： MQMessage 类

数据类型： 由 32 个字符组成的字符串

语法： 获取：*applid$ = MQMessage* **.ApplicationIdData**

设置：*MQMessage* **.ApplicationIdData** = *applid$*

ApplicationOriginData 属性

读 — 写。 MQMD ApplOriginData — 消息原始上下文的一部分。

其初始值全为空白。

定义于： MQMessage 类

数据类型： 由 4 个字符组成的字符串

语法： 获取：*applor$ = MQMessage* **.ApplicationOriginData**

设置：*MQMessage* **.ApplicationOriginData** = *applor$*

BackoutCount 属性

只读。 MQMD BackoutCount。

其初始值为 0

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *backoutct& = MQMessage* **.BackoutCount**

CharacterSet 属性

读 — 写。 MQMD CodedCharSetId。

其初始值是特殊值 **MQCCSI_Q_MGR**。

如果将 CharacterSet 设置为 **MQCCSI_Q_MGR**，会将当前语言环境的代码页用于 WriteString 方法中的字符转换。 对于服务器应用程序，所使用的代码页是队列管理器的代码页。 对于客户机应用程序，所使用的代码页是缺省的当前语言环境代码页。

例如：

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

其中“n”大于或等于 0 并且小于或等于 255，所以写入缓冲区的“n”值是一个字节。

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *:30ccid& = MQMessage* **.CharacterSet**

要设置: *MQ* 消息 **.CharacterSet** = *ccid&*

示例

如果您要以代码页 437 编写字符串，那么请发出：

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

在发出任何 WriteString 调用前在 CharacterSet 中设置您需要的值。

CorrelationId 属性

读 — 写。 将消息放入队列中时，要包含在消息的 MQMD 中的 CorrelationId。 也是从队列中获取消息时要匹配的标识。

其初始值为空。

定义于： MQMessage 类

数据类型： 由 24 个字符组成的字符串

语法： 获取：*correlid$ = MQMessage* **.CorrelationId** 设置：*MQMessage* **.CorrelationId** = *correlid$*

请参阅第 563 页的『消息描述符属性』，了解关于何时必须使用 CorrelationIdHex 代替 CorrelationId 属性的更多信息。

CorrelationIdHex 属性

读 — 写。 将消息放入队列中时，要包含在消息的 MQMD 中的 CorrelationId。 也是在从队列中获取消息时要匹配的 CorrelationId。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 48 个有效的十六进制字符。

其初始值为“0.....0”。

定义于： MQMessage 类

数据类型： 由 48 个十六进制字符组成的字符串，表示 24 个 ASCII 字符

语法： 获取：*correlidh$ = MQMessage* **.CorrelationIdHex**

设置：*MQMessage* **.CorrelationIdHex** = *correlidh$*

请参阅第 563 页的『消息描述符属性』了解关于何时必须使用 CorrelationIdHex 代替 CorrelationId 属性的讨论。

Encoding 属性

读 — 写。 标识用于应用程序消息数据中数值的表示法的 MQMD 字段。

其初始值是特殊值 MQENC_NATIVE，因平台而异。

此属性由下列方法使用：

- ReadDecimal2 方法
- ReadDecimal4 方法
- ReadDouble 方法
- ReadDouble4 方法
- ReadFloat 方法
- ReadInt2 方法
- ReadInt4 方法
- ReadLong 方法
- ReadShort 方法

- ReadUInt2 方法
- WriteDecimal2 方法
- WriteDecimal4 方法
- WriteDouble 方法
- WriteDouble4 方法
- WriteFloat 方法
- WriteInt2 方法
- WriteInt4 方法
- WriteLong 方法
- WriteShort 方法
- WriteUInt2 方法

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *encoding& = MQMessage* **. 编码** 要设置: *MQ* 消息 **. 编码** = *encoding&*

如果您准备将数据写入消息缓冲区，而接收队列管理器不能执行自己的数据转换，那么您应该设置此字段以符合接收队列管理器平台的特征。

Expiry 属性

读 — 写。 MQMD 到期时间字段，预计值在十分之一秒内。

其初始值是特殊值 MQEI_UNLIMITED

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *expiry& = MQMessage* **. 到期**

要设置: *MQ* 消息 **. 到期** = *expiry&*

Feedback 属性

读 — 写。 MQMD 反馈字段。

其初始值是特殊值 MQFB_NONE。

定义于： MQMessage 类

数据类型： 长整型

值：

- 请参阅反馈。

语法: 要获取: *feedback& = MQMessage* **. 反馈**

要设置: *MQMessage* 。**反馈** = 反馈和

Format 属性

读 — 写。 MQMD 格式字段。 给出描述消息数据性质的内置格式名称或用户定义的格式名称。

其初始值是特殊值 MQFMT_NONE。

定义于： MQMessage 类

数据类型： 由 8 个字符组成的字符串

语法： 获取： *format$ = MQMessage* **.Format**

设置： *MQMessage* **.Format** = *format$*

GroupId 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 GroupId。 也是从队列中获取消息时要匹配的标识。 其初始值全为空。

定义于：
MQMessage 类

数据类型：
由 24 个字符组成的字符串

语法： 获取：*groupid$ = MQMessage*. **GroupId**

设置：*MQMessage*. **GroupId** = *groupid$*

请参阅第 563 页的『消息描述符属性』，了解关于何时必须使用 GroupIdHex 代替 GroupId 属性的更多信息。

GroupIdHex 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 GroupId。 也是从队列中获取消息时要匹配的标识。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 48 个有效的十六进制字符。

其初始值为“0.....0”。

定义于：
MQMessage 类

数据类型：
由 48 个十六进制字符组成的字符串，表示 24 个 ASCII 字符。

语法： 获取：*groupidh$ = MQMessage*. **GroupIdHex**

设置：*MQMessage*. **GroupIdHex** = *groupidh$*

请参阅第 563 页的『消息描述符属性』，了解关于何时必须使用 GroupIdHex 代替 GroupId 属性的更多信息。

MessageData 属性

读 — 写。 将消息的整个内容作为一个字符串来检索或设置。

定义于： MQMessage 类

数据类型： 变体

注： 此属性使用的数据类型是变体，但 MQAX 期望它是字符串的变体类型。 如果您以除此类型以外的变体传递，将返回错误 MQRC_OBJECT_TYPE_ERROR。

语法： 获取：*String$ = MQMessage* **.MessageData**

设置：*MQMessage* **.MessageData** = *String$*

MessageFlags 属性

读 — 写。 指定分段控制信息的消息标志。 初始值为 0。

定义于：
MQMessage 类

数据类型：
长

值：
请参阅 MsgFlags (MQLONG)。

语法： 要获取: *messageflags& = MQMessage*。 **MessageFlags**

设置：*MQMessage*. **MessageFlags** = *messageflags&*

MessageId 属性

读 — 写。 将消息放入队列中时，要包含在消息的 MQMD 中的 MessageId。 也是从队列中获取消息时要匹配的标识。

其初始值全为空。

定义于： MQMessage 类

数据类型： 由 24 个字符组成的字符串

语法： 获取：*messageid$ = MQMessage* **.MessageId**

设置：*MQMessage* **.MessageId** = *messageid$*

请参阅第 563 页的『消息描述符属性』，了解关于何时必须使用 MessageIdHex 代替 MessageId 属性的更多信息。

MessageIdHex 属性

读 — 写。 将消息放入队列中时，要包含在消息的 MQMD 中的 MessageId。 也是在从队列中获取消息时要匹配的 MessageId。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 48 个有效的十六进制字符。

其初始值为“0.....0”。

定义于： MQMessage 类

数据类型： 由 48 个十六进制字符组成的字符串，表示 24 个 ASCII 字符

语法： 获取：*messageidh$ = MQMessage* **.MessageIdHex**

设置：*MQMessage* **.MessageIdHex** = *messageidh$*

请参阅第 563 页的『消息描述符属性』，了解关于何时必须使用 MessageIdHex 代替 MessageId 属性的更多信息。

MessageSequenceNumber 属性

读 — 写。 标识消息在组中的顺序的信息。 初始值为 1。

定义于：
MQMessage 类

数据类型：
长

值：
请参阅 MsgSeqNumber (MQLONG)。

语法： 要获取: *sequencenumber& = MQMessage*。 **SequenceNumber**

设置：*MQMessage*. **SequenceNumber** = 序列枚举器和

MessageType 属性

读 — 写。 MQMD MsgType 字段。

其初始值是 MQMT_DATAGRAM。

定义于： MQMessage 类

数据类型： 长整型

值：

- 请参阅 MsgType (MQLONG)。

语法： 要获取: *msgtype& = MQMessage* **.MessageType**

要设置: *MQ* 消息 **.MessageType** = *msgtype&*

Offset 属性

读 — 写。 在分段消息中的偏移。 初始值为 0。

定义于：
MQMessage 类

数据类型：
长

值：
请参阅 Offset (MQLONG)。

语法： 要获取: *offset& = MQMessage*。 **Offset**

设置： *MQMessage*. **偏移量** = *offset&*

OriginalLength 属性

读 — 写。 分段消息的原始长度。 初始值为 MQOL_UNDEFINED。

定义于：
MQMessage 类

数据类型：
长

值：
请参阅 OriginalLength (MQLONG)。

语法： 要获取: *originallength& = MQMessage*。 **OriginalLength**

设置： *MQMessage*. **OriginalLength** = *originallength&*

Persistence 属性

读 — 写。 消息的持久性设置。

其初始值为 MQPER_PERSISTENCE_AS_Q_DEF。

定义于： MQMessage 类

数据类型： 长整型

语法： 要获取: *persist& = MQMessage* **. 持久性**

要设置: *MQ* 消息 **. 持久性** = *persist&*

Priority 属性

读 — 写。 消息的优先级。

其初始值是特殊值 MQPRI_PRIORITY_AS_Q_DEF

定义于： MQMessage 类

数据类型： 长整型

语法： 要获取: *priority& = MQMessage* **. 优先级**

要设置: *MQ* 消息 **. 优先级** = *priority&*

PutApplicationName 属性

读 — 写。 MQMD PutApplName — 消息原始上下文的一部分。

其初始值全为空白。

定义于： MQMessage 类

数据类型： 由 28 个字符组成的字符串

语法： 获取： *putapplnm$ = MQMessage* **.PutApplicationName**

设置： *MQMessage* **.PutApplicationName** = *putapplnm$*

PutApplicationType 属性

读 — 写。 MQMD PutApplType — 消息原始上下文的一部分。

其初始值为 MQAT_NO_CONTEXT

定义于： MQMessage 类

数据类型： 长整型

值：

- 请参阅 PutApplType (MQLONG)。

语法： 要获取: *putappltp& = MQMessage* **.PutApplication 类型**

要设置: *MQ* 消息 **.PutApplication 类型** = 普塔普利特普公司

PutDateTime 属性

读/写。 此属性组合 MQMD PutDate 和 PutTime 字段。 它们是表明何时放入消息的消息原始上下文的一部分。

ActiveX 扩展在 ActiveX 日期/时间格式与 IBM MQ MQMD 中使用的日期和时间格式之间转换。 如果接收到 PutDate 或 PutTime 无效的消息，那么 get 方法之后的 PutDateTime 属性将设置为 EMPTY。

其初始值为 EMPTY。

定义于： MQMessage 类

数据类型： 类型 7 的变体（日期/时间）或 EMPTY。

语法： 获取： *datetime = MQMessage* **.PutDateTime**

设置： *MQMessage* **.PutDateTime** = *datetime*

ReplyToQueueManagerName 属性

读 — 写。 MQMD ReplyToQMgr 字段。

其初始值全为空白。

定义于： MQMessage 类

数据类型： 由 48 个字符组成的字符串

语法： 获取： *replytoqmgr$ = MQMessage* **.ReplyToQueueManagerName**

设置： *MQMessage* **.ReplyToQueueManagerName** = *replytoqmgr$*

ReplyToQueueName 属性

读 — 写。 MQMD ReplyToQ 字段。

其初始值全为空白。

定义于： MQMessage 类

数据类型： 由 48 个字符组成的字符串

语法： 获取： *replytoq$ = MQMessage* **.ReplyToQueueName**

设置：*MQMessage* **.ReplyToQueueName** = *replytoq$*

Report 属性

读 — 写。 消息的报告选项。

其初始值为 MQRO_NONE。

定义于： MQMessage 类

数据类型： 长整型

值：

- 请参阅报告。

语法: 要获取: *report& = MQMessage* **. 报告**

要设置: *MQMessage* 。**报告** = 报告和

TotalMessageLength 属性

只读。 获取由 MQGET 接收到的最后一个消息的长度。 如果该消息未被截断，那么此值等于 MessageLength 属性的值。

定义于： MQMessage 类

数据类型： 长整型

语法: 要获取: *totalmessagelength& = MQ* 消息 **.TotalMessage 长度**

UserId 属性

读 — 写。 MQMD UserIdentifier — 消息身份上下文的一部分。

其初始值全为空白。

定义于： MQMessage 类

数据类型： 由 12 个字符组成的字符串

语法： 获取： *userid$ = MQMessage* **.UserId**

设置： *MQMessage* **.UserId** = *userid$*

ClearErrorCodes 方法

为 MQMessage 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于： MQMessage 类

语法：

```
Call MQMessage.ClearErrorCodes()
```

ClearMessage 方法

此方法清除 MQMessage 对象的数据缓冲区部分。 数据缓冲区中的任何消息数据都会丢失，因为 MessageLength、DataLength 和 DataOffset 都设置为零。

消息描述符 (MQMD) 部分不受影响；应用程序可能需要先修改某些 MQMD 字段，然后才能重新使用 MQMessage 对象。 要将 MQMD 字段设置回，可使用“新建”将对象替换为新实例。

定义于： MQMessage 类

语法：

```
Call MQMessage.ClearMessage()
```

Read 方法

将一序列字节从消息缓冲区读到字节数组中。 DataOffset 增加所读取的字节数，而 Data Length 减少同样的数。

定义于：
MQMessage 类

语法： Data = MQMessage。 **读取** (len&)

参数：
len&: 长整型。 要读取的数据长度，单位是字节。

ReadBoolean 方法

从消息缓冲区中的当前位置读取一个 1 字节的布尔值，并返回一个 2 字节的布尔值 TRUE(-1)/FALSE(0)。 DataOffset 加一，而 Data Length 减一。

定义于：
MQMessage 类

语法： *value = MQMessage*. **ReadBoolean**

ReadByte 方法

ReadByte 方法从消息数据缓冲区中读取从 DataOffset 所指字节开始的 1 个字节，并将其作为整型（带符号的 2 字节）整数值返回，其范围在 -128 到 127 之间。

如果发出此方法时 MQMessage.DataLength 小于 1，那么此方法失败。

如果方法成功，那么 DataOffset 加 1，DataLength 减 1。

假设消息数据的字节是带符号的二进制整数。

定义于：
MQMessage 类

语法：
integerv% = MQMessage **.ReadByte**

ReadDecimal2 方法

读取 2 字节的压缩十进制数并将其作为带符号的 2 字节整数值返回。 DataOffset 加二，而 Data Length 减二。

定义于：
MQMessage 类

语法： *value% = MQMessage*. **ReadDecimal2**

ReadDecimal4 方法

读取 4 字节的压缩十进制数并将其作为带符号的 4 字节整数值返回。 DataOffset 加四，而 Data Length 减四。

定义于：
MQMessage 类

语法：

```
Call value& = MQMessage.ReadDecimal4
```

ReadDouble 方法

ReadDouble 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 8 个字节，并将其作为双精度（带符号的 8 字节）浮点值返回。

如果发出此方法时 MQMessage.DataLength 小于 8，那么此方法失败。

如果方法成功，那么 DataOffset 加 8，DataLength 减 8。

假设 8 字符的消息数据是二进制浮点数。 编码由 MQMessage.Encoding 属性指定。 请注意，不支持从 System/360 格式进行转换。

定义于：
MQMessage 类

语法：
doublev# = MQMessage **.ReadDouble**

ReadDouble4 方法

ReadDouble4 和 WriteDouble4 方法是 ReadFloat 和 WriteFloat 的备用方法。 这是因为它们支持由于太大而无法转换为 4 字节 IEEE 浮点格式的 4 字节 System/390 浮点消息值。

ReadDouble4 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 4 个字节，并将它们作为双精度（带符号的 8 字节）浮点值返回。

如果发出此方法时 MQMessage.DataLength 小于 4，那么此方法失败。

如果方法成功，那么 DataOffset 加 4，DataLength 减 4。

假设 4 字符的消息数据是一个二进制浮点数。 编码由 MQMessage.Encoding 属性指定。 请注意，不支持从 System/360 格式进行转换。

定义于：
MQMessage 类

语法：
doublev# = MQMessage. **ReadDouble4**

ReadFloat 方法

ReadFloat 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 4 个字节，并将它们作为单精度（带符号的 4 字节）浮点值返回。

如果发出此方法时 MQMessage.DataLength 小于 4，那么此方法失败。

如果方法成功，那么 DataOffset 加 4，DataLength 减 4。

假设 4 字符的消息数据是一个浮点数。 编码由 MQMessage.Encoding 属性指定。 请注意，不支持从 System/360 格式进行转换。

定义于：
MQMessage 类

语法：
singlev! = MQMessage 。**ReadFloat**

ReadInt2 方法

该方法等同于 ReadShort 方法。

语法：
integerv% = MQMessage **.ReadInt2**

ReadInt4 方法

此方法等同于 ReadLong 方法。

语法：
bigint& = MQMessage **.ReadInt4**

ReadLong 方法

ReadLong 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 4 个字节，并将它们作为长整形（带符号的 4 字节）整数值返回。

如果发出此方法时 MQMessage.DataLength 小于 4，那么此方法失败。

如果方法成功，那么 DataOffset 加 4，DataLength 减 4。

假设 4 个字符的消息数据是一个二进制整数。 编码由 MQMessage.Encoding 属性指定。

定义于：
MQMessage 类

语法：
bigint& = MQMessage **.ReadLong**

ReadNullTerminatedString 方法

此方法是在字符串可能包含嵌入空字符时用于替代 ReadString 的。

此方法从消息数据缓冲区读取以 DataOffset 所指字节开始的指定数目的字节，并将其作为 ActiveX 字符串返回。 如果字符串在结束前包含嵌入空，那么返回的字符串长度会减少以仅反映在空前面的字符。

无论字符串是否包含嵌入的空字符，DataOffset 都会增加指定的值，而 DataLength 减少同样的值。

假设消息数据中的字符为代码页中的一个字符串，该代码页由 MQMessage.CharacterSet 属性指定。 会为应用程序执行到 ActiveX 表示法的转换。

定义于：
MQMessage 类

语法: *string $ = MQMessage*。 **ReadNullTerminatedString(length&)**

参数：
length& 长长的 字符串字段的长度，单位是字节。

ReadShort 方法

ReadShort 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 2 个字节，并将它们作为整型（带符号的 2 字节）值返回。

如果发出此方法时 MQMessage.DataLength 小于 2，那么此方法失败。

如果方法成功，那么 DataOffset 加 2，DataLength 减 2。

假设 2 字符的消息数据为一个二进制整数。 编码由 MQMessage.Encoding 属性指定。

定义于：
MQMessage 类

语法：
integerv% = MQMessage **.ReadShort**

ReadString 方法

此方法从消息数据缓冲区读取以 DataOffset 所指字节开始的 n 个字节，并将其作为 ActiveX 字符串返回。

如果发出此方法时 MQMessage.DataLength 小于 n，那么此方法失败。

如果方法成功，那么 DataOffset 加 n，DataLength 减 n。

假设 n 个字符的消息数据是代码页中的一个字符串，该代码页由 MQMessage.CharacterSet 属性指定。 会为应用程序执行到 ActiveX 表示法的转换。

定义于： MQMessage 类

语法: *stringv $= MQMessage* **.ReadString** *(length&)*

参数

length& 长整型。 字符串字段的长度，单位是字节。

ReadUInt2 方法

ReadUInt2 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 2 个字节，并将它们作为长整型（带符号的 4 字节）整数值返回。

如果发出此方法时 MQMessage.DataLength 小于 2，那么此方法失败。

如果方法成功，那么 DataOffset 加 2，DataLength 减 2。

假设 2 个字节的消息数据是一个不带符号的二进制整数。 编码由 MQMessage.Encoding 属性指定。

定义于：

MQMessage 类

语法：

bigint& = MQMessage **.ReadUInt2**

ReadUnsignedByte 方法

ReadUnsignedByte 方法从消息数据缓冲区中读取从 DataOffset 指定字节开始的 1 个字节，并将其作为一个整型（带符号的 2 字节）整数值返回，其范围在 0 到 255 之间。

如果发出此方法时 MQMessage.DataLength 小于 1，那么此方法失败。

如果方法成功，那么 DataOffset 加 1，DataLength 减 1。

假设消息数据的 1 个字符是不带符号的二进制整数。

定义于：

MQMessage 类

语法：

integerv% = MQMessage **.ReadUnsignedByte**

ReadUTF 方法

此方法从消息中读取从 **DataOffset** 指定字节开始的 UTF 格式字符串，并将 UTF 格式的字符串作为 ActiveX 字符串返回。 所读取的 UTF 格式的字符串由 2 字节数据组成，这些数据表示该字符串的长度，后跟 UTF 字符数据。

如果在发出此方法时 **MQMessage.DataLength** 小于字符串长度，那么此方法将失败。

如果此方法成功，**DataOffset** 将按照字符串长度递增，**DataLength** 将按照字符串长度递减。

定义于：

MQMessage 类

语法：

```
value$ = MQMessage.ReadUTF
```

ResizeBuffer 方法

此方法改变当前内部分配用于保留消息数据缓冲区的存储量。 它使应用程序在自动缓冲区管理之上还可有一些控制能力，这样如果应用程序知道它要处理大型消息时，就可确保分配足够大的缓冲区。 应用程序不一定要使用此调用 — 如果它不使用，那么自动缓冲区管理代码会增加缓冲区大小以适应需求。

如果您将缓冲区的大小调整为小于当前 MessageLength，就要冒丢失数据的风险。 如果您确实丢失了数据，那么方法返回 CompletionCode MQCC_WARNING 和 ReasonCode MQRC_DATA_TRUNCATED。

如果您将缓冲区的大小调整为小于 **DataOffset** 属性的值，那么：

- **DataOffset** 属性会被更改以指向新缓冲区的结尾
- **DataLength** 属性设置为零

- **MessageLength** 属性会被更改为新缓冲区大小

定义于：
MQMessage 类

语法： *MQ* 消息 **.ResizeBuffer** (*Length&*)

参数：
Length& 长整型。 必需大小，单位是字符。

Write 方法

将一系列字节从 Data Offset 所指位置的字节数组写入消息缓冲区。 若有必要，会扩展缓冲区的长度（MQMessage.MQMessageLength）以容纳字节数组的全长。 如果方法成功，那么 DataOffset 增加所写的字节数。

定义于：
MQMessage 类

语法：

```
Call MQMessage.Write(value)
```

参数：
data：字节数组或引用字节数组的变体

WriteBoolean 方法

从一个 2 字节的布尔值在消息缓冲区中的当前位置写一个 1 字节的布尔值。 DataOffset 加一。

定义于：
MQMessage 类

语法：

```
Call MQMessage.WriteBoolean(value)
```

参数：
value：布尔（2 字节）。 要写的值。

WriteByte 方法

此方法获取一个带符号的 2 字节整数值，并将它作为一个 1 字节二进制数在 DataOffset 所指位置写入消息数据缓冲区。 它替换任何已经在缓冲区中该位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加一。

指定值的范围应该是从 -128 到 127。 如果不是，那么方法返回 CompletionCode MQCC_FAILED 和 ReasonCode MQRC_WRITE_VALUE_ERROR。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteByte(value%)
```

参数： *value%* Integer。 要写的值。

WriteDecimal2 方法

将带符号的 2 字节整数作为 2 字节压缩十进制数写入。 DataOffset 加二。

定义于：
MQMessage 类

语法：

```
Call MQMessage.WriteDecimal2(value%)
```

参数：
value% Integer。 要写的值。

WriteDecimal4 方法

将带符号的 4 字节整数作为 4 字节压缩十进制数写入。 DataOffset 加四。

定义于：
MQMessage 类

语法：

```
Call MQMessage.WritedDecimal4(value&)
```

参数：
值和长整型。 要写的值。

WriteDouble 方法

此方法获取一个带符号的 8 字节浮点值，并将它作为一个 8 字节浮点数在 DataOffset 所指位置开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加 8。

该方法转换到 MQMessage.Encoding 属性指定的浮点表示法。 不支持转换为 *System/360* 格式。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteDouble(value#)
```

参数：
value# Double。 要写的值。

WriteDouble4 方法

请参阅 第 611 页的『ReadDouble4 方法』获取何时应该使用 ReadDouble4 和 WriteDouble4 代替 ReadFloat 和 WriteFloat 的描述。

此方法获取一个带符号的 8 字节浮点值，并将它作为一个 4 字节浮点数在 DataOffset 所指位置开始写入消息数据缓冲区。

如果方法成功，那么 DataOffset 加 4。

它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

该方法转换到 MQMessage.Encoding 属性指定的浮点表示法。 不支持转换为 *System/360* 格式。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteDouble4(value#)
```

参数： *value#* Double。 要写的值。

WriteFloat 方法

此方法获取一个带符号的 4 字节浮点值，并将它作为一个 4 字节浮点数在 DataOffset 所指字符处开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加 4。

该方法转换到 MQMessage.Encoding 属性指定的二进制表示法。 不支持转换为 *System/360* 格式。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteFloat(value!)
```

参数 *value!* Float。 要写的值。

WriteInt2 方法

此方法等同于 WriteShort 方法。

语法：

```
Call MQMessage.WriteInt2(value%)
```

参数 *value%* Integer。 要写的值。

WriteInt4 方法

此方法等同于 WriteLong 方法。

语法：

```
Call MQMessage.WriteInt4(value&)
```

参数 *value&* 长整型。 要写的值。

WriteLong 方法

此方法获取一个带符号的 4 字节整数值，并将它作为一个 4 字节二进制数在 DataOffset 所指字节开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加 4。

该方法转换到 MQMessage.Encoding 属性指定的二进制表示法。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteLong(value&)
```

参数 *value&* 长整型。 要写的值。

WriteNullTerminatedString 方法

此方法执行正常 WriteString，并向任何剩余字节填充空直至达到指定长度。 如果初始的 write string 所写的字节数等于指定长度，那么不写空。 如果字节数超出指定长度，那么设置错误（原因码 MQRC_WRITE_VALUE_ERROR）。

如果方法成功，那么 DataOffset 增加指定的长度。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteNullTerminatedString(value$, length&)
```

参数：
value$ 字符串。 要写的值。

length& 长整型。 字符串字段的长度，单位是字节。

WriteShort 方法

此方法获取一个带符号的 2 字节整数值，并将它作为一个 2 字节二进制数在 DataOffset 所指字节开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加 2。

该方法转换到 MQMessage.Encoding 属性指定的二进制表示法。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteShort(value%)
```

参数 *value%* Integer。 要写的值。

WriteString 方法

此方法获取一个 ActiveX 字符串，并将它在 DataOffset 所指字节开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 增加字符串的字节长度。

该方法将字符转换为 MQMessage.CharacterSet 属性指定的代码页。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteString(value$)
```

参数 *value$* 字符串。 要写的值。

WriteUInt2 方法

此方法获取一个带符号的 2 字节整数值，并将它作为一个不带符号的 2 字节二进制数在 DataOffset 所指字节开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加 2。

该方法转换到 MQMessage.Encoding 属性指定的二进制表示法。 指定值的范围应该是从 0 到 2**16-1。 如果不是，那么方法返回 CompletionCode MQCC_FAILED 和 ReasonCode MQRC_WRITE_VALUE_ERROR。

定义于： MQMessage 类

语法：

```
Call MQMessage.WriteUInt2(value&)
```

参数 *value&* 长整型。 要写的值。

WriteUnsignedByte 方法

此方法获取一个带符号的 2 字节整数值，并将它作为一个不带符号的 1 字节二进制数在 DataOffset 所指字节开始写入消息数据缓冲区。 它替换任何已经在缓冲区中这些位置的数据，若有必要，还会扩展缓冲区的长度（MQMessage.MessageLength）。

如果方法成功，那么 DataOffset 加 1。

指定值的范围应该是从 0 到 255。 如果不是，那么方法返回 CompletionCode MQCC_FAILED 和 ReasonCode MQRC_WRITE_VALUE_ERROR。

定义于：
MQMessage 类

语法：

```
Call MQMessage.WriteUnsignedByte(value%)
```

参数 *value%* Integer。 要写的值。

WriteUTF 方法

此方法获取一个 ActiveX 字符串，并将它在当前位置以 UTF 格式写入消息数据缓冲区。 所写的数据由 2 字节长度后跟字符数据组成。 如果方法成功，那么 DataOffset 增加字符串的长度。

定义于：
MQMessage 类

语法: Call *MQMessage*。 **WriteUTF** (值 $)

参数：
value$ 字符串。 要写的值。

MQPutMessageOptions 类

此类可封装各种选项，这些选项可控制将消息放入 IBM MQ 队列的操作。

包含

MQPutMessageOptions 类包含在 MQSession 类中。

创建

新建会创建新的 MQPutMessageOptions 对象，并将其所有属性设置为初始值。

或者，使用 MQSession 类的 AccessPutMessageOptions 方法。

语法

Dim *pmo* **As New MQPutMessageOptions** 或

Set *pmo* = **New MQPutMessageOptions**

属性

- 第 619 页的『CompletionCode 属性』.
- 第 619 页的『Options 属性』.
- 第 619 页的『ReasonCode 属性』.
- 第 619 页的『ReasonName 属性』.
- 第 619 页的『RecordFields 属性』.
- 第 620 页的『ResolvedQueueManagerName 属性』.
- 第 620 页的『ResolvedQueueName 属性』.

方法

- 第 620 页的『ClearErrorCodes 方法』.

CompletionCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的完成代码。

定义于： MQPutMessageOptions 类

数据类型： 长整型

值：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法： 要获取: *completioncode& = PutOpts* **.CompletionCode**

Options 属性

读 — 写。 MQPMO Options 字段。 此字段的初始值是 MQPMO_NONE。 有关更多信息，请参阅 MQPMO 选项。

定义于： MQPutMessageOptions 类。

数据类型： 长整型

语法： 要获取: *options& = PutOpts* **. 选项**

要设置: *PutOpts* **. 选项** = *options&*

不支持 MQPMO_PASS_IDENTITY_CONTEXT 和 MQPMO_PASS_ALL_CONTEXT 选项。

ReasonCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的原因码。

定义于： MQPutMessageOptions 类

数据类型： 长整型

值：

- 请参阅 API 完成代码和原因码。

语法： 要获取: *reasoncode& = PutOpts* **.ReasonCode**

ReasonName 属性

只读。 返回最新原因码的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。

定义于： MQPutMessageOptions 类

数据类型： 字符串

值：

- 请参阅 API 完成代码和原因码。

语法： 获取： *reasonname$ = PutOpts* **.ReasonName**

RecordFields 属性

读 — 写。 当将消息放入分发列表时，表明要在每队列基础上定制哪个字段的标志。 初始值为零。

此属性对应于 MQI MQPMO 结构中的 PutMsgRecFields 标记。 在 MQI 中，这些标志控制哪些字段存在和由 MQPUT 使用（在 MQPMR 结构中）。 在 MQPutMessageOptions 对象中总是存在这些字段，因此这些标志仅影响哪些字段被 Put 使用。

定义于：
MQPutMessageOptions 类

数据类型：
长

语法： 要获取: *recordfields& = PutOpts* **.RecordFields**

设置：*PutOpts*. **RecordFields** = *recordfields&*

ResolvedQueueManagerName 属性

只读。 MQPMO ResolvedQMgrName 字段。 请参阅 ResolvedQMgrName (MQCHAR48)，以了解详细信息。 初始值全为空白。

定义于： MQPutMessageOptions 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*qmgr$ = PutOpts* **.ResolvedQueueManagerName**

ResolvedQueueName 属性

只读。 MQPMO ResolvedQName 字段。 请参阅 ResolvedQName (MQCHAR48)，以了解详细信息。 初始值全为空白。

定义于： MQPutMessageOptions 类

数据类型： 由 48 个字符组成的字符串

语法： 获取：*qname$ = PutOpts* **.ResolvedQueueName**

ClearErrorCodes 方法

为 MQPutMessageOptions 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于：
MQPutMessageOptions 类

语法：
Call *PutOpts* **.ClearErrorCodes()**

MQGetMessageOptions 类

此类可封装各种选项，这些选项可控制从 IBM MQ 队列获取消息的操作。

包含

MQGetMessageOptions 类包含在 MQSession 类中。

创建

新建会创建新的 MQGetMessageOptions 对象，并将其所有属性设置为初始值。

或者，使用 MQSession 类的 AccessGetMessageOptions 方法。

属性

- 第 621 页的『CompletionCode 属性』
- 第 621 页的『MatchOptions 属性』
- 第 621 页的『Options 属性』
- 第 621 页的『ReasonCode 属性』

- 第 622 页的『ReasonName 属性』
- 第 622 页的『ResolvedQueueName 属性』
- 第 622 页的『WaitInterval 属性』

方法

- 第 622 页的『ClearErrorCodes 方法』

语法

Dim *gmo* **As New MQGetMessageOptions** 或

Set *gmo* = **New MQGetMessageOptions**

CompletionCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的完成代码。

定义于： MQGetMessageOptions 类。

数据类型： 长整型

值：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法： 要获取: *completioncode& = GetOpts* **.CompletionCode**

MatchOptions 属性

读 — 写。 控制用于 MQGET 的选择标准的选项。 初始值是 MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID。

定义于：
MQGetMessageOptions 类

数据类型：
长

值：
请参阅 MatchOptions (MQLONG)。

语法： 要获取: *matchoptions& = GetOpts*。 **MatchOptions**

设置： *GetOpts*. **MatchOptions** = *matchoptions&*

Options 属性

读 — 写。 MQGMO Options 字段。 请参阅选项，以了解详细信息。 初始值为 MQGMO_NO_WAIT。

定义于： MQGetMessageOptions 类。

数据类型： 长整型

语法： 要获取: *options& = GetOpts* **. 选项** 要设置: *GetOpts* **. 选项** = *options&*

ReasonCode 属性

只读。 返回由对此对象发出的最新方法或属性访问所设置的原因码。

定义于： MQGetMessageOptions 类

数据类型： 长整型

值：

- 请参阅 API 完成代码和原因码。

语法: 要获取: *reasoncode& = GetOpts* **.ReasonCode**

ReasonName 属性

只读。 返回最新原因码的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。 **定义于:** MQGetMessageOptions 类

数据类型: 字符串

值:

- 请参阅 API 完成代码和原因码。

语法: 获取: *reasonname$ = MQGetMessageOptions* **.ReasonName**

ResolvedQueueName 属性

只读。 MQGMO ResolvedQName 字段。 请参阅 ResolvedQName (MQCHAR48)，以了解详细信息。 初始值全为空白。

定义于: MQGetMessageOptions 类

数据类型: 由 48 个字符组成的字符串

语法: 获取: *qname$ = GetOpts* **.ResolvedQueueName**

WaitInterval 属性

读/写。 MQGMO WaitInterval 字段。 如果 Options 属性已请求等待操作，Get 将等待适当消息到达的最长时间（单位是毫秒）。 此字段的初始值为 0。 有关 MQGMO 选项的详细信息，请参阅 MQGMO。

定义于: MQGetMessageOptions 类

数据类型: 长整型

语法: 要获取: *wait& = GetOpts* **.WaitInterval**

To set: *GetOpts* **.WaitInterval** = wait&

ClearErrorCodes 方法

为 MQGetMessageOptions 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于:
MQGetMessageOptions 类

语法:
Call *GetOpts* **.ClearErrorCodes()**

MQDistributionList 类

此类封装队列 — 本地、远程或别名的集合以供输出。

创建

新建会创建一个新的 MQDistributionList 对象。

或者，也可以使用 MQQueueManager 类的 AddDistributionList 方法

属性

- 第 623 页的『AlternateUserId 属性』
- 第 623 页的『CloseOptions 属性』

- 第 623 页的『CompletionCode 属性』
- 第 624 页的『ConnectionReference 属性』
- 第 624 页的『FirstDistributionListItem 属性』
- 第 624 页的『IsOpen 属性』
- 第 624 页的『OpenOptions 属性』
- 第 625 页的『ReasonCode 属性』
- 第 625 页的『ReasonName 属性』

方法

- 第 625 页的『AddDistributionListItem 方法』
- 第 625 页的『ClearErrorCodes 方法』
- 第 626 页的『Close 方法』
- 第 626 页的『Open 方法』
- 第 626 页的『Put 方法』

语法

Dim *distlist.* **A s New MQDistributionList** 或 **Set** *distlist* = **New MQDistributionList**

AlternateUserId 属性

读 — 写。 在队列打开时，用于确认对其列表的访问的备用用户标识。

定义于：
MQDistributionList 类

数据类型：
由 12 个字符组成的字符串

语法：获取：*altuser$ = MQDistributionList*. **AlternateUserId**

设置：*MQDistributionList*. **AlternateUserId** = *altuser$*

CloseOptions 属性

读 — 写。 用于控制分发列表关闭时所发生事件的选项。 初始值是 MQCO_NONE。

定义于：
MQDistributionList 类

数据类型：
长

值：

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

语法: 要获取: *closeopt& = MQDistributionList*。 **CloseOptions**

设置：*MQDistributionList*. **CloseOptions** = *closeopt&*

CompletionCode 属性

只读。 由对此对象发出的最新方法或属性访问所设置的完成代码。

定义于：
MQDistributionList 类

数据类型：
长

值：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法： 要获取: *completioncode& = MQDistributionList*。 **CompletionCode**

ConnectionReference 属性

读 — 写。 分发列表所属的队列管理器。

定义于：
MQDistributionList 类

数据类型：
MQQueueManager

语法： 获取： *set queuemanager = MQDistributionList*. **ConnectionReference**

设置： *set MQDistributionList*. **ConnectionReference** = *queuemanager*

FirstDistributionListItem 属性

只读。 与分发列表关联的第一个分发列表项对象。

定义于：
MQDistributionList 类

数据类型：
MQDistributionListItem

值：

语法： 获取: *set distributionlistitem = MQDistributionList*。 **FirstDistributionListItem**

IsOpen 属性

只读。

定义于：
MQDistributionList 类

数据类型：
布尔

值：

- TRUE (-1)
- FALSE (0)

语法： 获取： *IsOpen = MQDistributionList*. **IsOpen**

OpenOptions 属性

读 — 写。 当分发列表打开时要使用的选项。

定义于：
MQDistributionList 类

数据类型：
长

值：
请参阅 MQPMO 选项。

语法： 要获取: *openopt& = MQDistributionList*。 **OpenOptions**

设置：*MQDistributionList.* **OpenOptions** = openopt&

ReasonCode 属性

只读。 由对此对象发出的最新方法或属性访问所设置的原因码。

定义于：
MQDistributionList 类

数据类型：
长

值：
请参阅 API 完成代码和原因码。

语法： 要获取: *reasoncode& = MQDistributionList*。 **ReasonCode**

ReasonName 属性

只读。 ReasonCode 的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。

定义于：
MQDistributionList 类

数据类型：
字符串

值：
请参阅 API 完成代码和原因码。

语法： 获取：*reasonname$ = MQDistributionList*. **ReasonName**

AddDistributionListItem 方法

使用此方法可创建新的 `MQDistributionListItem` 对象，并将该对象与分发列表对象相关联。 队列名称参数是必需的。

此方法可将新分发列表项作为第一项插入现有列表中。 具体来说就是，此方法可创建以下配置：

- 在分发列表中，它会将 **FirstDistributionListItem** 属性设置为指向新分发列表项。
- 在新分发列表项中，它会设置以下属性：
  - 它会将 **DistributionList** 属性设置为指向此分发列表。
  - 它会将 **PreviousDistributionListItem** 属性设置为空。
  - 它会将 **NextDistributionListItem** 属性设置为指向之前为第一项的分发列表项，如果列表中之前没有项目，会将该属性设置为空。

当分发列表打开时，您不能使用此方法添加新项。

定义于：
MQDistributionList 类

语法： set distributionlistitem = *MQDistributionList* **.AddDistributionListItem** (QName$, QMgrName$)

参数：

QName$ 字符串。 IBM MQ 队列的名称。

QMgrName$ 字符串。 IBM MQ 队列管理器的名称。

ClearErrorCodes 方法

为 MQDistributionList 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于：
MQDistributionList 类

语法：

```
Call MQDistributionList.ClearErrorCodes()
```

Close 方法

使用当前的 Close 选项值关闭分发列表。

定义于：
MQDistributionList 类

语法：
Call *MQDistributionList*. **Close** ()

Open 方法

使用当前的 AlternateUserId 值打开与当前对象关联的分发列表项的 **QueueName** 和（相应的）**QueueManagerName** 属性所指定的每个队列。

定义于：
MQDistributionList 类

语法：

```
Call MQDistributionList.Open()
```

Put 方法

在与分发列表关联的分发列表项所标识的每个队列上放置消息。

定义于：

MQDistributionList 类

语法

Call MQDistributionList. **放入** (消息， PutMsg 选项和)

参数

Message 表示要放入的消息的 MQMessage 对象。

PutMsgOptions 包含控制放入操作的选项的 MQPutMessageOptions 对象。 如果未指定，那么使用缺省 PutMessageOptions。

此方法使用 MQMessage 对象作为参数。 使用此方法后可能会改变以下分发列表项属性：

- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex
- GroupId
- GroupIdHex
- Feedback
- AccountingToken

- AccountingTokenHex

MQDistributionListItem 类

此类封装 MQOR、MQRR 和 MQPMR 结构并将其与属主分发列表关联。

创建

使用 MQDistributionList 类的 AddDistributionListItem 方法

属性

方法

- 第 628 页的『AccountingToken 属性』.
- 第 628 页的『AccountingTokenHex 属性』.
- 第 628 页的『CompletionCode 属性』.
- 第 628 页的『CorrelationId 属性』.
- 第 629 页的『CorrelationIdHex 属性』.
- 第 629 页的『DistributionList 属性』.
- 第 629 页的『Feedback 属性』.
- 第 629 页的『GroupId 属性』.
- 第 630 页的『GroupIdHex 属性』.
- 第 630 页的『MessageId 属性』.
- 第 630 页的『MessageIdHex 属性』.
- 第 630 页的『NextDistributionListItem 属性』.
- 第 631 页的『PreviousDistributionListItem 属性』.
- 第 631 页的『QueueManagerName 属性』.
- 第 631 页的『QueueName 属性』.
- 第 631 页的『ReasonCode 属性』.
- 第 631 页的『ReasonName 属性』.
- 第 632 页的『ClearErrorCodes 方法』.

属性：

- AccountingToken 属性
- AccountingTokenHex 属性
- CompletionCode 属性
- CorrelationId 属性
- CorrelationIdHex 属性
- DistributionList 属性
- Feedback 属性
- GroupId 属性
- GroupIdHex 属性
- MessageId 属性
- MessageIdHex 属性
- NextDistributionListItem 属性
- PreviousDistributionListItem 属性

- QueueManagerName 属性
- QueueName 属性
- ReasonCode 属性
- ReasonName 属性

方法：

- ClearErrorCodes 方法

创建：

使用 MQDistributionList 类的 AddDistributionListItem 方法

AccountingToken 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 AccountingToken。 其初始值全为空。

定义于：
MQDistributionListItem 类

数据类型：
由 32 个字符组成的字符串

语法：获取：*accountingtoken$ = MQDistributionListItem*. **AccountingToken**

设置：*MQDistributionListItem*. **AccountingToken** = *accountingtoken$*

AccountingTokenHex 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 AccountingToken。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 64 个有效的十六进制字符。

其初始值为“0.....0”。

定义于：
MQDistributionListItem 类

数据类型：
由 64 个十六进制字符组成的字符串，表示 32 个 ASCII 字符。

语法：获取：*accountingtokenh$ = MQDistributionListItem*. **AccountingTokenHex**

设置：*MQDistributionListItem*. **AccountingTokenHex** = *accountingtokenh$*

CompletionCode 属性

只读。 由对属主分发列表对象发出的最新打开或放入请求所设置的完成代码。

定义于：
MQDistributionListItem 类

数据类型：
长

值：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法：获取：*completioncode$ = MQDistributionListItem*. **CompletionCode**

CorrelationId 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 CorrelId。 其初始值全为空。

定义于：
MQDistributionListItem 类

数据类型：
由 24 个字符组成的字符串

语法：获取：*correlid$ = MQDistributionListItem*. **CorrelationId**

设置：*MQDistributionListItem*. **CorrelationId** = *correlid$*

CorrelationIdHex 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 CorrelId。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 48 个有效的十六进制字符。

其初始值为“0.....0”。

定义于：
MQDistributionListItem 类

数据类型：
由 48 个十六进制字符组成的字符串，表示 24 个 ASCII 字符。

语法：获取：*correlidh$ = MQDistributionListItem*. **CorrelationIdHex**

设置：*MQDistributionListItem*. **CorrelationIdHex** = *correlidh$*

DistributionList 属性

只读。 与此分发列表项关联的分发列表。

定义于：
MQDistributionListItem 类

数据类型：
MQDistributionList

语法：获取：*set distributionlist = MQDistributionListItem*. **DistributionList**

Feedback 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的反馈值。

定义于：
MQDistributionListItem 类

数据类型：
长

值：
请参阅 Feedback (MQLONG)。

语法: 要获取: *feedback& = MQDistributionListItem*。 **Feedback**

设置：*MQDistributionListItem*. **反馈** = 反馈和

GroupId 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 GroupId。 其初始值全为空。

定义于：
MQDistributionListItem 类

数据类型：
由 24 个字符组成的字符串

语法：获取：*groupid$ = MQDistributionListItem*. **GroupId**

设置：*MQDistributionListItem*. **GroupId** = *groupid$*

GroupIdHex 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 GroupId。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 48 个有效的十六进制字符。

其初始值为“0.....0”。

定义于：
MQDistributionListItem 类

数据类型：
由 48 个十六进制字符组成的字符串，表示 24 个 ASCII 字符。

语法：获取：*groupidh$ = MQDistributionListItem*. **GroupIdHex**

设置：*MQDistributionListItem*. **GroupIdHex** = *groupidh$*

MessageId 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 MessageId。 其初始值全为空。

定义于：
MQDistributionListItem 类

数据类型：
由 24 个字符组成的字符串

语法：获取：*messageid$ = MQDistributionListItem*. **MessageId**

设置：*MQDistributionListItem*. **MessageId** = *messageid$*

MessageIdHex 属性

读 — 写。 当消息放入队列时消息的 MQPMR 要包含的 MessageId。

字符串的每两个字符代表等于单个 ASCII 字符的十六进制数。 例如，字符“6”和“1”的字符对代表单个字符“A”，字符“6”和“2”的字符对代表单个字符“B”，依次类推。

您必须提供 48 个有效的十六进制字符。

其初始值为“0.....0”。

定义于：
MQDistributionListItem 类

数据类型：
由 48 个十六进制字符组成的字符串，表示 24 个 ASCII 字符。

语法：获取：*messageidh$ = MQDistributionListItem*. **MessageIdHex**

设置：*MQDistributionListItem*. **MessageIdHex** = *messageidh$*

NextDistributionListItem 属性

只读。 与同一分发列表关联的下一个分发列表项对象。

定义于：
MQDistributionListItem 类

数据类型：
MQDistributionListItem

语法：获取：*set distributionlistitem = MQDistributionListItem.* **NextDistributionListItem**

PreviousDistributionListItem 属性

只读。 与同一分发列表关联的前一个分发列表项对象。

定义于：
MQDistributionListItem 类

数据类型：
MQDistributionListItem

语法：获取：*set distributionlistitem = MQDistributionListItem*. **PreviousDistributionListItem**

QueueManagerName 属性

读 — 写。 IBM MQ 队列管理器名称。

定义于：
MQDistributionListItem 类

数据类型：
由 48 个字符组成的字符串。

语法：获取：*qmname$ = MQDistributionListItem*. **QueueManagerName**

设置：*MQDistributionListItem*. **QueueManagerName** = *qmname$*

QueueName 属性

读 — 写。 IBM MQ 队列名称。

定义于：
MQDistributionListItem 类

数据类型：
由 48 个字符组成的字符串。

语法：获取：*qname$ = MQDistributionListItem*. **QueueName**

设置：*MQDistributionListItem*. **QueueName** = *qname$*

ReasonCode 属性

只读。 由对属主分发列表对象发出的最新打开或放入所设置的原因码。

定义于：
MQDistributionListItem 类

数据类型：
长

值：
请参阅 API 完成代码和原因码。

语法：获取：

```
reasoncode& = MQDistributionListItem.ReasonCode
```

ReasonName 属性

只读。 ReasonCode 的符号名称。 例如“MQRC_QMGR_NOT_AVAILABLE”。

定义于：
MQDistributionListItem 类

数据类型：
字符串

值：
请参阅 API 完成代码和原因码。

语法：获取：*reasonname$ = MQDistributionListItem*. **ReasonName**

ClearErrorCodes 方法

为 MQDistributionListItem 类和 MQSession 类将 CompletionCode 复位为 MQCC_OK 和将 ReasonCode 复位为 MQRC_NONE。

定义于：
MQDistributionListItem 类

语法：

```
Call MQDistributionListItem.ClearErrorCodes
```

跟踪 IBM MQ Automation Classes for ActiveX

下面提供了有关针对 IBM MQ Automation Classes for ActiveX 提供的跟踪功能以及常见缺陷的信息，还提供了有关如何避免这些缺陷的帮助。

从 IBM MQ 9.0 开始，不推荐使用对 Microsoft Active X 的 IBM MQ 支持。 IBM MQ classes for .NET 是建议使用的替代技术。 有关更多信息，请参阅开发 .NET 应用程序。

以下部分说明了所提供的跟踪设备，并详细介绍了常见的缺陷，还提供了如何避免这些缺陷的帮助：

- 第 632 页的『控制 IBM MQ Automation Classes for ActiveX 的跟踪』
- 第 633 页的『当您的 IBM MQ Automation Classes for ActiveX 脚本失败时』
- 第 633 页的『IBM MQ Automation Classes for ActiveX 原因码』
- 第 635 页的『代码级工具』

控制 IBM MQ Automation Classes for ActiveX 的跟踪

IBM MQ Automation Classes for ActiveX (MQAX) 提供一个跟踪功能，可帮助服务组织确定当您遇到问题时发生了什么情况。

它会显示当您运行您的 MQAX 脚本时使用的路径。 除非遇到问题，否则请在运行脚本时禁用跟踪，以避免任何不必要的系统资源使用。

有三个环境变量可供您设置用于控制跟踪：

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

注：为 **OMQ_TRACE** 变量指定任何值都可打开跟踪功能。 即使将 **OMQ_TRACE** 变量设置为 OFF，跟踪功能仍处于活动状态。 要关闭跟踪功能，请不要为 **OMQ_TRACE** 指定任何值。

1. 单击**开始**
2. 单击**控制面板**
3. 双击**系统**
4. 单击**高级**
5. 单击**环境**
6. 在标题为“**（用户名）的用户变量**”的部分中，单击**新建**
7. 在相应的字段中输入变量名和有效值，并单击**确定**
8. 单击**确定**以关闭“环境变量”窗口
9. 单击**确定**以关闭“系统属性”窗口
10. 关闭“控制面板”窗口

在确定写入跟踪文件的位置时，应确保您有足够的权限可以写入（以及读取）该磁盘。

启用跟踪时，它会减慢 MQAX 的运行速度，但这不会影响 ActiveX 或 IBM MQ 环境的性能。 当您不再需要某个跟踪文件时，可将其删除。

要更改 OMQ_TRACE 变量的状态，您必须先停止运行 MQAX。

跟踪文件名和目录

跟踪文件名的格式为 `OMQ`*nnnnn*`.trc`，其中 *nnnnn* 是当时运行的 ActiveX 进程的标识。

表 *103:* 命令及其效果

命令	结果
SET OMQ_TRACE_PATH=drive:\directory	设置写入跟踪文件的跟踪目录。
SET OMQ_TRACE_PATH =	移除跟踪目录的所有现有设置。 如果不设置跟踪目录，将使用当前工作目录（当 ActiveX 启动时）。
ECHO %OMQ_TRACE_PATH%	显示 Windows 上跟踪目录的当前设置。
SET OMQ_TRACE = xxxxxxxx	打开跟踪。 您可以通过在“=”符号后面加上一个或多个字符来启用跟踪。 例如: `SET OMQ_TRACE=yes` 和 `SET OMQ_TRACE=no`。 在这两个示例中，都启用了跟踪。 此设置仅对一个窗口/会话有效。
SET OMQ_TRACE=	关闭跟踪。
ECHO %OMQ_TRACE%	显示 Windows 上环境变量的内容。
SET	显示 Windows 上所有环境变量的内容。
SET OMQ_TRACE_LEVEL = 9	将跟踪级别设置为 9。 大于 9 的值不会在跟踪文件中生成任何其他信息。

当您的 IBM MQ Automation Classes for ActiveX 脚本失败时

如果 IBM MQ Automation Classes for ActiveX 脚本失败，那么可查找大量信息源。

首次故障症状报告

对于意外错误和内部错误，可能会生成首次故障症状报告（与跟踪功能无关）。

可在名为 `OMQ`*nnnnn*`.fdc`（其中，*nnnnn* 是当时运行的 ActiveX 进程的编号）的文件中找到此报告。 您可在启动 ActiveX 的工作目录或 OMQ_PATH 环境变量中指定的路径中找到该文件。

其他信息来源

IBM MQ 提供了各种错误日志和跟踪信息，具体取决于所涉及的平台。 请参阅您的 Windows 应用程序事件日志。

IBM MQ Automation Classes for ActiveX 原因码

除了 IBM MQ MQI 原因码外，还可能会出现 IBM MQ Automation Classes for ActiveX (MQAX) 原因码。

除了为 IBM MQ MQI 记录的代码之外，可能还会出现以下原因码。 关于其他代码，请参阅您的 IBM MQ 应用程序事件日志。

表 *104:* 原因码及其含义

原因码	说明
MQRC_LIBRARY_LOAD_ERROR (6000)	一个或多个 IBM MQ 库无法载入。 请检查所有的 IBM MQ 库是否都在您所使用系统上的正确搜索路径中。 例如，确保包含 IBM MQ 库的目录在 PATH 中。
MQRC_CLASS_LIBRARY_ERROR (6001)	其中一个 IBM MQ `classlibrary` 调用返回一个意外的 **ReasonCode** 或 **CompletionCode** 值。 请检查首次故障症状报告以获取详细信息。 记录最后使用的方法/属性和类，并将此问题通知 IBM 支持。

表 *104:* 原因码及其含义 *(*继续*)*

原因码	说明
MQRC_STRING_LENGTH_TOO_BIG (6002)	试图向消息缓冲区写入长度大于 65535 个字节的 UTF 格式字符串。
MQRC_WRITE_VALUE_ERROR (6003)	使用的值超出范围; 例如 `msg.WriteByte (240)`。
MQRC_PACKED_DECIMAL_ERROR (6004)	试图从消息缓冲区读取压缩十进制数，但是该数据指针处的数据格式不是有效的压缩数据格式。
MQRC_FLOAT_CONVERSION_ERROR (6005)	试图从消息缓冲区读取单精度或双精度浮点数，但是该数据指针处的数据格式不是相应的浮点格式。
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	打开的对象没有正确的 **OpenOptions** 设置，因此，需要一个或多个附加选项。 因为此队列对独占输入打开，而关闭将呈现一个窗口，从而为其他输入提供可能获取访问该队列的机会，所以需要隐式重新打开，但不能关闭。 显式地设置 **OpenOptions** 值以涵盖所有可能性，这样便不需要隐式地重新打开。
MQRC_REOPEN_INQUIRE_ERROR (6101)	打开的对象没有正确的 **OpenOptions** 设置，因此，需要一个或多个附加选项。 因为关闭之前要动态地检查对象的一个或多个特征，且 **OpenOptions** 值不包含 **MQOO_INQUIRE** 选项，因此，需要隐式重新打开，但不能关闭。 显式地设置 **OpenOptions** 值以包含 **MQOO_INQUIRE** 选项。
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	打开的对象没有正确的 **OpenOptions** 设置，因此，需要一个或多个附加选项。 因为已使用 **MQOO_SAVE_ALL_CONTEXT** 选项打开此队列，且之前执行了破坏性的 Get 调用，所以需要隐式重新打开，但不能关闭。 这已使保留的状态信息与开放式队列关联，此信息将在关闭时被破坏。 显式地设置 **OpenOptions** 值以涵盖所有可能性，这样便不需要隐式地重新打开。
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	打开的对象没有正确的 **OpenOptions** 设置，因此，需要一个或多个附加选项。 因为队列是定义类型为 **MQQDT_TEMPORARY_DYNAMIC** 的本地队列，而它将在关闭时被破坏，因此，需要隐式重新打开，但不能关闭。 显式地设置 **OpenOptions** 值以涵盖所有可能性，这样便不需要隐式地重新打开。
MQRC_ATTRIBUTE_LOCKED (6104)	试图在对象打开时更改该对象的值或属性。 某些属性（如 **AlternateUserId**）不能在对象打开时进行更改。
MQRC_CURSOR_NOT_VALID (6105)	自隐式重新打开上次使用已打开队列的浏览光标以来，该浏览光标已失效。 显式地设置 **OpenOptions** 值以涵盖所有可能性，这样便不需要隐式地重新打开。
MQRC_ENCODING_ERROR (6106)	下一个消息项的编码需要是 **MQENC_NATIVE** 编码才能读取。
MQRC_STRUCID_ERROR (6107)	缺少下一个消息项的标识结构（由数据指针开始处的 4 个字符派生而来），或者该结构与在其中读取项的变量类型不一致。
MQRC_NULL_POINTER (6108)	在需要或者暗指非空指针的位置提供了空指针。 这可能是由于为从 Visual Basic 或 Excel 中作为调用参数使用的 IBM MQ 对象使用了显式声明导致的。 例如: • 从 Visual Basic 开始， `dim msg as Object` 正常工作，而 `dim msg as MqMessage` 可能无法正常工作。 • 在 Visual Basic 中，定义和设置队列后，`dim msg as MqMessageq.put msg` 正常工作，而在 Excel 中，此命令会生成一个 MQRC_NULL_POINTER 异常。
MQRC_NO_CONNECTION_REFERENCE (6109)	MQQueue 对象丢失与 MQQueueManager 对象的连接。 如果队列管理器断开连接，会发生这种情况。 删除此 MQQueue 对象。
MQRC_NO_BUFFER (6110)	没有可用的缓冲区。 对于 MQMessage 对象，无法分配缓冲区，因为对象状态中存在内部不一致。
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	二进制数据的长度与目标属性的长度不一致。 对于所有属性而言，0 是一个正确的长度。 **CorrelationId** 属性和 **MessageId** 属性的正确长度是 24。 **AccountingToken** 属性的正确长度是 32。
MQRC_BUFFER_NOT_AUTOMATIC (6112)	无法调整用户定义和管理的缓冲区的大小。 因为消息缓冲区是系统管理的，这表明存在内部不一致。
MQRC_INSUFFICIENT_BUFFER (6113)	在向请求提供数据指针后，可用的缓冲区空间不足。 这可能是因为无法调整缓冲区的大小。
MQRC_INSUFFICIENT_DATA (6114)	在为读请求提供数据指针后，数据不足。 将此缓冲区减小到合适的大小后再读取该数据。

表 *104:* 原因码及其含义 *(*继续*)*

原因码	说明
MQRC_DATA_TRUNCATED (6115)	将数据从一个缓冲区复制到另一个缓冲区时，数据被截断。 这可能是因为无法调整目标缓冲区的大小，或是因为寻址这个缓冲区或其他缓冲区的问题，或是因为缓冲区的大小缩小。
MQRC_ZERO_LENGTH (6116)	在要求或暗示长度为正的位置提供的长度为零。
MQRC_NEGATIVE_LENGTH (6117)	在要求长度为零或为正的位置提供的长度为负。
MQRC_NEGATIVE_OFFSET (6118)	在要求偏移量为零或为正的位置提供的偏移量为负。
MQRC_INCONSISTENT_FORMAT (6119)	下一个消息项的格式与读取项的变量的类型不一致。
MQRC_INCONSISTENT_OBJECT_STATE (6120)	此对象（已打开）和引用的 `MQQueueManager` 对象（未连接）之间存在不一致。
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	`MQPutMessageOptions` 上下文引用没有引用有效的 `MQQueue` 对象。 此对象之前被破坏过。
MQRC_CONTEXT_OPEN_ERROR (6122)	`MQPutMessageOptions` 上下文引用引用了无法打开以建立上下文的 `MQQueue` 对象。 这可能是由于 `MQQueue` 对象具有不恰当的打开选项。 请检查引用的对象的原因码以确定问题原因。
MQRC_STRUC_LENGTH_ERROR (6123)	内部数据结构的长度与其内容不一致。 对于 `MQRMH` 标头，长度不足以包含固定字段和所有偏移量数据。
MQRC_NOT_CONNECTED (6124)	因为到队列管理器的必需连接不可用，所以方法失败，且无法隐式地建立连接。
MQRC_NOT_OPEN (6125)	因为没有打开 IBM MQ 对象，所以方法失败，且无法隐式地完成打开过程。
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	因为分发列表中没有 `MQDistributionListItem` 对象，因此打开 `MQDistributionList` 失败。 更正操作：至少向分发列表中添加一个 `MQDistributionListItem` 对象。
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	因为对象已打开，所以方法失败，且此打开选项与所需的操作不一致。 更正操作：使用适当的打开选项打开此对象，然后重试。
MQRC_WRONG_VERSION (6128)	由于指定或遇到的版本号不正确或不受支持，因此方法失败。

代码级工具

IBM Service Team 可能会向您询问您安装的代码级别。

要确定您安装的代码级别，请运行“MQAXLEV”实用程序。

在命令提示符中，切换到包含 MQAX200.dll 的目录或添加完整路径长度并输入：

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

其中 MQAXLEV.OUT 是输出文件的名称。

如果您没有指定输出文件，那么详细信息将显示在屏幕上。

来自代码级工具的输出文件示例如下：

来自代码级工具的输出文件示例

```
5639-B43 (C) Copyright IBM Corp. 1996, 2024. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 *****
lib/mqole/mqole.cpp, mqole, p000, p000 L981119       1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212      1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212   1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216        1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212       1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212       1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212      1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212      1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212      1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219        1.11 99/02/18 12:12:59
```

到 MQAI 的 ActiveX 接口

要获取 COM 接口的简要概述和它们在 MQAI 中的用途，请参阅 第 561 页的『使用组件对象模型接口 (IBM MQ Automation Classes for ActiveX)』。

MQAI 使应用程序能构建和发送可编程命令格式（PCF）命令，而不必直接获取和格式化 PCF 必需的变量长度缓冲区。 有关 MQAI 的更多信息，请参阅 IBM MQ 管理界面 (MQAI)。 MQAI ActiveX MQBag 类封装 MQAI 支持的数据包时所用的方式可在任何支持创建 COM 对象的语言中使用；例如， Visual Basic、C++、Java 和其他 ActiveX 脚本编制客户机。

MQAI ActiveX 接口可与 MQAX 类一起使用，这些类提供到 MQI 的 COM 接口。 要获取更多关于 MQAX 类的信息，请参阅第 562 页的『设计访问非 ActiveX 应用程序的 MQAX 应用程序』。

ActiveX 接口提供一个名为 MQBag 的类。 此类用于创建 MQAI 数据包及其属性，和用于在每个数据包中创建和使用数据项的方法。 MQBag Execute 方法将包数据作为 PCF 消息发送到 IBM MQ 队列管理器并收集应答。

有关 MQBag 类、其属性和方法的更多信息，请参阅 第 636 页的『MQBag 类』。

可以选择使用指定的请求队列和应答队列将 PCF 消息发送到指定的队列管理器对象。 应答在新的 MQBag 对象中返回。 在可编程命令格式的定义中描述了命令和应答的完整集合。 通过选择相应的请求队列和应答队列，可将命令发送到 IBM MQ 网络中的任何队列管理器。

MQBag 类

MQBag 类用于按需要创建 MQBag 对象。 当实例化时，MQBag 类会返回新的 MQBag 对象引用。

按如下所示在 Visual Basic 中创建 MQBag 对象：

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

MQBag 属性

以下列表中解释了 MQBag 对象的属性：

- 第 637 页的『Item 属性』.
- 第 638 页的『Count 属性』.
- 第 638 页的『Options 属性』.

MQBag 方法

以下列表中解释了 MQBag 对象的方法：

- 第 639 页的『Add 方法』.

- 第 640 页的『AddInquiry 方法』.
- 第 640 页的『Clear 方法』.
- 第 640 页的『Execute 方法』.
- 第 641 页的『FromMessage 方法』.
- 第 641 页的『ItemType 方法』.
- 第 642 页的『Remove 方法』.
- 第 642 页的『Selector 方法』.
- 第 643 页的『ToMessage 方法』.
- 第 644 页的『Truncate 方法』.

错误处理

如果在对 MQBag 对象的操作期间检测到错误，包括底层 MQAX 或 MQAI 对象返回到包中的错误，就会产生错误异常。 MQBag 类支持 COM ISupportErrorInfo 接口，因此下列信息可用于您的错误处理例程：

- 错误号码：由所检测到错误的 IBM MQ 原因码和 COM 工具代码组成。 作为 COM 标准的工具字段指明了错误的责任区域。 对于由 IBM MQ 检测到的错误，它始终为 FACILITY_ITF。
- 错误源：标识检测到错误的对象的类型和版本。 对于在 MQBag 操作期间检测到的错误，错误源始终为 MQBag.MQBag1。
- 错误描述：给出 IBM MQ 原因码的符号名称的字符串。

如何访问错误信息取决于您的脚本语言；例如，在 Visual Basic 中，信息在 Err 对象中返回，而 IBM MQ 原因码要通过从 Err.Number 减去常量 vbObjectError 来获取。

ReasonCode = Err.Number - vbObjectError

如果 MQBag Execute 方法发送 PCF 消息并接收到应答，那么即使发送的命令可能已失败，也将认为该操作是成功的。 在这种情况下，应答包本身包含完成代码和错误原因码，如 可编程命令格式的定义中所述。

Item 属性

用途

Item 属性代表包中的一项。 它用于设置或查询关于项的值。 此属性的使用相当于以下 MQAI 调用：

- “mqSetString”
- “mqSetInteger”
- “mqInquireInteger”
- “mqInquireString”
- “mqInquireBag”

在可编程命令格式参考中描述。

格式

Item (Selector, ItemIndex, Value)

参数

Selector（变体）- 输入
要设置或查询的项的选择器。

当查询项时，缺省值是 MQSEL_ANY_USER_SELECTOR。 当设置项时，缺省值是 MQIA_LIST 或 MQCA_LIST。

如果 `Selector` 不属于长整型，那么产生 MQRC_SELECTOR_TYPE_ERROR。

此参数是可选的。

ItemIndex（长整型）- 输入
此值标识要设置或查询的指定选择器项的出现。 缺省值是 MQIND_NONE。

此参数是可选的。

Value（变体）- 输入
返回的值或要设置的值。 当查询项时，返回值的类型可以是长整型、字符串或 MQBag。 但是当设置项时，值的类型必须是长整型或字符串；如果不是，那么产生 MQRC_ITEM_VALUE_ERROR。

Visual Basic 语言调用

当在包中查询一个项的值时：

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

对于 MQBag 引用：

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

设置包中项的值：

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

Count 属性

用途

Count 属性表示包中的数据项数。 此属性相当于可编程命令格式参考中的 MQAI 调用“mqCountItems”。

格式

Count (*Selector, Value*)

参数

Selector（变体）- 输入
要包含在计数中的数据项的选择器。

缺省值是 MQSEL_ALL_USER_SELECTORS。

如果 Selector 不属于长整型，那么返回 MQRC_SELECTOR_TYPE_ERROR。

Value（长整型）- 输出
Selector 包含的包中的项数。

Visual Basic 语言调用

返回包中的项数：

```
ItemCount = mqbag.Count([Selector])
```

Options 属性

用途

Options 属性设置供包使用的选项。 此属性对应于 可编程命令格式参考中 MQAI 调用的 **Options** 参数 "mqCreateBag"。

格式

Options (*Options*)

参数

Options（长整型）- 输入/输出
包选项。

注: 必须在数据项添加到包中或在包中设置 **之前** 设置包选项。 如果在包不是空的时候更改选项，那么产生 MQRC_OPTIONS_ERROR。 即使随后清除该包也是如此。

Visual Basic 语言调用

当在包中查询一个项的选项时：

```
Options = mqbag.Options
```

设置包中项的选项：

```
mqbag.Options = Options
```

MQBag 方法

以下几页解释 MQBag 对象的方法。

Add 方法

用途

Add 方法将数据项添加到包。 此方法相当于可编程命令格式参考中的 MQAI 调用“mqAddInteger”和“mqAddString”。

格式

Add (*Value, Selector*)

参数

Value（变体）- 输入
数据项的整数或字符串值。

Selector（变体）- 输入
标识要添加的项的选择器。

根据 Value 的类型，缺省值是 MQIA_LIST 或 MQCA_LIST。 如果 **Selector** 参数不属于长整型，那么产生 MQRC_SELECTOR_TYPE_ERROR。

Visual Basic 语言调用

向包中添加项：

```
mqbag.Add(Value,[Selector])
```

AddInquiry 方法

用途

AddInquiry 方法添加一个选择器，指定在发送管理包以执行 INQUIRE 命令时要返回的属性。 此方法相当于可编程命令格式参考中的 MQAI 调用“mqAddInquiry”。

格式

AddInquiry *(Inquiry)*

参数

Inquiry（长整型）- 输入
INQUIRE 管理命令要返回的 IBM MQ 属性的选择器。

Visual Basic 语言调用

使用 AddInquiry 方法：

```
mqbag.AddInquiry(Inquiry)
```

Clear 方法

用途

Clear 方法从包中删除所有数据项。 此方法相当于可编程命令格式参考中的 MQAI 调用“mqClearBag”。

格式

清除

Visual Basic 语言调用

从包中删除所有数据项：

```
mqbag.Clear
```

Execute 方法

用途

Execute 方法向命令服务器发送管理命令消息，并等待任何应答消息。 此方法相当于可编程命令格式参考中的 MQAI 调用“mqExecute”。

格式

Execute *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag)*

参数

QueueManager (MQQueueManager) - 输入
应用程序所连接的队列管理器。

Command (LONG) - 输入
要执行的命令。

OptionsBag (MQBag) - 输入
包含影响调用处理的选项的包。

RequestQ (MQQueue) - 输入
将放置管理命令消息的队列。

ReplyQ (MQQueue) - 输入
接收到应答消息的队列。

ReplyBag (MQBag) - 输出
包含来自应答消息的数据的包引用。

Visual Basic 语言调用

发送管理命令消息并等待任何应答消息：

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

FromMessage 方法

用途

FromMessage 方法将数据从消息装入包中。 此方法相当于可编程命令格式参考中的 MQAI 调用 “mqBufferToBag”。

格式

FromMessage (*Message, OptionsBag*)

参数

Message (MQMessage) - 输入
包含要转换的数据的消息。

OptionsBag (MQBag) - 输入
用于控制调用处理的选项。

Visual Basic 语言调用

将数据从消息装入包中：

```
mqbag.FromMessage(Message,[OptionsBag])
```

ItemType 方法

用途

ItemType 方法返回包中指定项中的值类型。 此方法相当于可编程命令格式参考中的 MQAI 调用 “mqInquireItemInfo”。

格式

ItemType (*Selector, ItemIndex, ItemType*)

参数

Selector（变体）- 输入
标识要查询的项的选择器。

缺省值是 MQSEL_ANY_USER_SELECTOR。 如果 **Selector** 参数不属于长整型，那么产生 MQRC_SELECTOR_TYPE_ERROR。

ItemIndex（长整型）- 输入
要查询的项的索引。

缺省值是 MQIND_NONE。

ItemType（长整型）- 输出
所指定项的数据类型。

注: 必须指定 **Selector** 参数和/或 **ItemIndex** 参数。 如果两个参数都不存在，那么产生 MQRC_PARAMETER_MISSING。

Visual Basic 语言调用

返回值的类型:

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

Remove 方法

用途

Remove 方法从包中删除项。 此方法相当于可编程命令格式参考中的 MQAI 调用“mqDeleteItem”。

格式

Remove (*Selector, ItemIndex*)

参数

Selector（变体）- 输入
标识要删除的项的选择器。

缺省值是 MQSEL_ANY_USER_SELECTOR。 如果 **Selector** 参数不属于长整型，那么产生 MQRC_SELECTOR_TYPE_ERROR。

ItemIndex（长整型）- 输入
要删除的项的索引。

缺省值是 MQIND_NONE。

注: 必须指定 **Selector** 参数和/或 **ItemIndex** 参数。 如果两个参数都不存在，那么产生 MQRC_PARAMETER_MISSING。

Visual Basic 语言调用

从包中删除项:

```
mqbag.Remove([Selector],[ItemIndex])
```

Selector 方法

用途

Selector 方法返回包中指定项的选择器。 此方法相当于可编程命令格式参考中的 MQAI 调用 “mqInquireItemInfo”。

格式

Selector (***Selector, ItemIndex, OutSelector***)

参数

Selector（变体）- 输入
标识要查询的项的选择器。

缺省值是 MQSEL_ANY_USER_SELECTOR。 如果 **Selector** 参数不属于长整型，那么产生 MQRC_SELECTOR_TYPE_ERROR。

ItemIndex（长整型）- 输入
要查询的项的索引。

缺省值是 MQIND_NONE。

OutSelector（变体）- 输出
指定项的选择器。

注: 必须指定 **Selector** 参数和/或 **ItemIndex** 参数。 如果两个参数都不存在，那么产生 MQRC_PARAMETER_MISSING。

Visual Basic 语言调用

返回项的选择器:

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

ToMessage 方法

用途

ToMessage 方法返回对 MQMessage 对象的引用。 引用包含来自包中的数据。 此方法相当于可编程命令格式参考中的 MQAI 调用“mqBagToBuffer”。

格式

ToMessage (*OptionsBag, Message*)

参数

OptionsBag (MQBag) - 输入
包含控制方法处理的选项的包。

Message (MQMessage) - 输出
包含来自包中的数据的 MQMessage 对象引用。

Visual Basic 语言调用

使用 ToMessage 方法:

```
Set Message = mqbag.ToMessage([OptionsBag])
```

Truncate 方法

用途

Truncate 方法减少包中的用户项数。 此方法相当于可编程命令格式参考中的 MQAI 调用 “mqTruncateBag”。

格式

Truncate (*ItemCount*)

参数

ItemCount（长整型）- 输入
在发生截断后要保留在包中的用户项数。

Visual Basic 语言调用

减少包中的用户项数:

```
mqbag.Truncate(ItemCount)
```

关于 IBM MQ Automation Classes for ActiveX 启动程序样本

此附录描述了 IBM MQ Automation Classes for ActiveX 启动程序样本，并说明了如何使用这些样本。

IBM MQ for Windows 提供了以下 Visual Basic 样本程序:

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

这些样本在 Visual Basic 4 或 Visual Basic 5 上运行。 您将在目录中找到它们 ... \tools\mqax\samples\vb。

在相同目录中，您还能找到 Microsoft Excel 和 html 的样本。 这些字段为:

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

注: 如果使用 Visual Basic 5，您**必须**选择并安装 Visual Basic 组件 grid32.ocx。

样本演示了什么

这些样本演示了如何使用 IBM MQ Automation Classes for ActiveX 来完成以下任务:

- 连接到队列管理器
- 访问队列
- 在队列中放入消息
- 从队列中取出消息

Visual Basic 样本的中心部分在下列页面中显示。

第 645 页的『准备运行样本』 和

第 645 页的『样本中的错误处理』

运行 ActiveX 启动程序样本

运行 IBM MQ Automation Classes for ActiveX 启动程序样本之前，请检查您是否已经运行缺省队列管理器，并已经创建所需的队列定义。 有关创建和运行队列管理器，以及创建队列的详细信息，请参阅管理。 此样本使用应在任何正常设置的 IBM MQ 服务器上定义的队列 SYSTEM.DEFAULT.LOCAL.QUEUE 。

下面列出了使用数据包的不同方式：

- 连接到队列管理器
- 访问队列
- 在队列中放入消息
- 从队列中取出消息

有关 Microsoft 基本 4 或更高版本的 MQAX 入门模板样本的信息，请参阅 第 645 页的『运行 MQAXTRIV 样本』

有关允许您浏览队列管理器和队列对象的属性和方法的样本的信息，请参阅第 647 页的『启动 MQAXCLSS 样本』

有关 MQAXDLST 样本的信息，请参阅第 647 页的『MQAXDLST 样本』

有关运行适用于 Microsoft Excel 95 或更高版本的 MQAX 启动程序样本 MQAXTRIV.XLS 的信息，请参阅第 647 页的『运行 MQAXTRIV.XLS 样本』。

有关使用 MQAX.XLS 运行银行演示的信息，请参阅第 647 页的『使用 MQAX.XLS 运行银行演示』

有关使用与 ActiveX 兼容的 WWW 浏览器的启动程序样本的信息，请参阅第 648 页的『使用 ActiveX 兼容 WWW 浏览器的启动程序样本』

准备运行样本

要运行任一样本，您需要下列其中一个软件，这取决于您准备要运行的样本。

- Microsoft Visual Basic 4 (或更高版本)
- Microsoft Excel 95（或更高版本）
- Web 浏览器

您还需要：

- 运行一个 IBM MQ 队列管理器。
- 定义一个 IBM MQ 队列。

样本中的错误处理

IBM MQ Automation Classes for ActiveX 软件包中提供的大多数样本只显示了少量错误处理过程或者未显示任何错误处理过程。 有关错误处理的更多信息，请参阅第 566 页的『错误处理』。

运行 MQAXTRIV 样本

1. 启动队列管理器。
2. 在 Windows 资源管理器或文件管理器中，选择样本 MQAXTRIV.VBP（Visual Basic 工程文件）的图标并打开该文件。

   将启动 Visual Basic 程序并打开文件 MQAXTRIV.VBP。
3. 在 Visual Basic 中，按功能键 5（F5）以运行此样本。
4. 单击窗口表单“**MQAX trivial tester**”中的任何地方。

如果一切运行正常，那么窗口背景应该变成绿色。 如果您的设置有问题，窗口背景会变成红色并显示错误信息。

下图显示了 Visual Basic 样本的中心部分。

```
Option Explicit

Private Sub Form_Click()

'*****************************************************************************
'* This simple example illustrates how to put and get an IBM MQ message to
'* and from an IBM MQ message queue. The data from the message returned by the
'* get is read and compared with that from the original message.
'*****************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* put message options
Dim GetOptions As MQGetMessageOptions   '* get message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*****************************************************************************
'* Handle errors
'*****************************************************************************
On Error GoTo HandleError

'*****************************************************************************
'* Initialize the current position for the form
'*****************************************************************************
CurrentX = 0
CurrentY = 0

'*****************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*****************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                        MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*****************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*****************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions

'*****************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*****************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*****************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*****************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127) '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
```

```
End If

Exit Sub
'*************************************************************************
'* Handle errors
'*************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "IBM MQ Completion Code = " & MQSess.CompletionCode
    Print "IBM MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
Else
    Print "Visual Basic error: " & Err
    Print Error(Err)
End If

Exit Sub

End Sub
```

启动 MQAXCLSS 样本

此样本允许您浏览队列管理器和队列对象的属性和方法。

1. 启动队列管理器。
2. 在 Windows 资源管理器中双击文档图标，或者从 Visual Basic 的“文件”菜单中单击“文件 - 打开”，以打开文件 MQAXCLSS.VBP。
3. 启动此样本。
4. 输入适当的队列管理器名称和队列名称，然后单击相应的按钮。

MQAXDLST 样本

Visual Basic MQAXDLST 样本演示了如何使用分发列表以在使用一次放入操作的情况下向两个队列发送相同的消息。 要运行此样本，请对 MQAXCLSS 样本执行上述相同的操作。

适用于 Microsoft Excel 95 或更高版本的 MQAX 启动程序样本

这部分说明了如何运行适用于 Microsoft Excel 95 或更高版本的 MQAX 启动程序样本 MQAXTRIV.XLS。

运行 *MQAXTRIV.XLS* 样本

1. 启动队列管理器。
2. 在资源管理器或文件管理器中，选择 MQAX 样本 MQAXTRIV.XLS 的图标。
3. 单击电子表格中的按钮。
4. 屏幕被更新以显示成功（或故障）消息。

使用 *MQAX.XLS* 运行银行演示

遵循以下步骤以运行银行演示。

1. 启动队列管理器。

2. 运行 IBM MQ MQSC 命令文件 BANK.TST。 这将设置必要的 IBM MQ 队列定义。

   要了解如何使用 MQSC 命令文件，请参阅使用 MQSC 命令管理。
3. 运行 MQAXBSRV.VBP。 此样本程序是模拟后端应用程序的服务器，它必须使用 Microsoft Excel 运行。
4. 运行 MQAX.XLS。 此样本是客户机 IBM MQ 演示。
5. 从列表中选择一个客户。
6. 单击**提交**。

暂停（大约 3 秒钟）之后，这些字段会填充值，并会显示一个条形图。

使用 ActiveX 兼容 WWW 浏览器的启动程序样本

注: 要运行此样本，您必须运行与 ActiveX 兼容的 Web 浏览器。 Microsoft Edge 是一个兼容的 Web 浏览器（但 Netscape Navigator 不是）。

运行 HTML 样本

此样本演示如何从 VBScript 和 JavaScript 调用 MQAX。

1. 启动队列管理器。
2. 在您的与 ActiveX 兼容的 Web 浏览器中打开文件“MQAXTRIV.HTM”。

   您可以通过在 Windows 资源管理器中双击此文件图标，或者从与 ActiveX 兼容的 Web 浏览器的“文件”菜单中选择“文件”-“打开”来打开此文件。
3. 按照屏幕上的指示信息进行操作。

ULW 开发 AMQP 客户机应用程序

由于 IBM MQ 支持 AMQP API（包括 MQ Light API），因此 IBM MQ 管理员可以创建 AMQP 通道。 此通道在启动时会定义一个端口号以用于接受来自 AMQP 客户机应用程序的连接。

您可以在 UNIX、Linux 或 Windows 上安装 AMQP 通道，但 AMQP 通道在 IBM i 或 z/OS 上不可用。

MQ Light API 基于 Oasis AMQP 1.0 协议。 针对 Node.js、Java、Ruby 和 Python 都提供了消息传递 API。

为使用 MQ Light API 而开发的应用程序可以连接到 MQ Light 运行时，具有 AMQP 通道的 IBM MQ 队列管理器或 IBM Cloud (formerly Bluemix)中 MQ Light 服务的实例。

开发 AMQP 客户机

MQ Light API 旨在快速轻松地建立业务应用程序的原型并开发业务应用程序。 有 MQ Light API for Node.js， Java， Ruby 和 Python，可从 https://github.com/mqlight 获取。

下载样本 AMQP 客户机

虽然 IBM MQ 未随附 MQ Light 客户机，但是您可以下载并安装以下 MQ Light 客户机：

Node.js
使用 npm 将 MQ Light Node.js API 安装到您的工作目录：`npm install mqlight@1.0`

Java
从 Maven Central 下载所需版本的 mqlight-distribution 软件包，并抽取其中的内容。 您可以在 Maven Central 上查找可用版本的 mqlight-distribution 软件包。

Ruby
使用 gem 将 MQ Light Ruby API 安装到您的工作目录：`gem install mqlight --pre`

Python
使用 pip 将 MQ Light Python API 安装到您的工作目录：`pip install mqlight --pre`

所有 MQ Light 客户机下载均包含若干样本，用于展示不同的消息传递功能：

- 发送样本
- 接收样本
- UI Workout 样本

您还可以下载基于 Apache Qpid 库的其他开放式源代码 AMQP 客户机。 有关更多信息，请参阅 https://qpid.apache.org/index.html。

注意: IBM 支持人员无法为这些客户机包提供配置或缺陷支持，任何使用问题或代码缺陷报告都应指向相应的项目。

保护 AMQP 客户机安全

有关保护 MQ Light 应用程序的信息，请参阅 保护 AMQP 客户机。

将 AMQP 客户机部署到 IBM MQ

当准备好部署应用程序时，此应用程序需要具备其他企业应用程序的所有监视、可靠性和安全性功能。 它还可以与其他企业应用程序交换数据。 您可以将 MQ Light 应用程序部署到 IBM MQ 队列管理器。 请参阅 第 663 页的『将 MQ Light 应用程序部署到本地 IBM MQ 环境』 。

在部署 AMQP 客户机后，您就可以与 IBM MQ 应用程序交换消息。 例如，如果您使用 MQ Light Node.js 客户机发送 JavaScript 字符串消息，那么 IBM MQ 应用程序会收到已将 MQMD 格式字段设置为 MQSTR 的 MQ 消息。

管理 AMQP 通道

可以采用与其他 MQ 通道相同的方式来管理 AMQP 通道。 您可以使用 MQSC 命令、PCF 命令消息或 IBM MQ Explorer 来定义、启动、停止和管理这些通道。 在创建和使用 AMQP 通道中，提供了一些示例命令来定义和启动客户机到队列管理器的连接。

在启动 AMQP 通道后，您可以通过连接 MQ Light 应用程序或使用以下任意方法来对其进行测试：

- 使用 IBM MQ Light Client for Node.js 和 Java。
- 使用针对 Ruby 和 Python 的 IBM MQ Light 客户机。
- 使用其他 AMQP 1.0 客户机。 例如，Apache Qpid Proton。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW MQ Light 和 AMQP（高级消息排队协议）

IBM MQ Light API 基于 OASIS 标准 AMQP 1.0 有线协议。 AMQP 指定了在发送方和接收方之间发送消息的方式。 如果应用程序向消息代理（例如 IBM MQ）发送消息，那么该应用程序将充当发送方。 如果 IBM MQ 向 AMQP 应用程序发送消息，那么它将充当发送方。

以下是 AMQP 的一些优点：

- 它是开放式标准化协议
- 与其他开放式源代码 AMQP 1.0 客户机兼容
- 提供多种开放式源代码客户机实施

虽然任何 AMQP 1.0 客户机都能够连接到 AMQP 通道，但某些 AMQP 功能（例如事务或多个会话）不受支持。

有关更多信息，请参阅 AMQP.org Web 站点和 OASIS 标准 AMQP 1.0 PDF。

MQ Light 消息传递 API 基于 AMQP 1.0。 该 API 提供了大部分发布/预订和点到点消息传递流所需的大多数消息传递功能。

MQ Light API 具有以下消息传递功能：

- 至多一次消息传递
- 至少一次消息传递
- 主题字符串目标寻址
- 消息和目标耐久性
- 可使多个订户共享工作负载的共享目标
- 便于解决挂起客户机的客户机接管功能
- 可配置的消息预读功能
- 可配置的消息确认功能

要获取完整的 MQ Light API 文档，请访问以下 Web 站点：

- 要获取 Node.js API 文档，请参阅 https://www.npmjs.org/package/mqlight
- 要获取 Ruby API 文档，请参阅 https://www.rubydoc.info/github/mqlight/ruby-mqlight
- 要获取 Python API 文档，请参阅 https://python-mqlight.readthedocs.org
- 对于 Java API 文档，请参阅 https://mqlight.github.io/java-mqlight

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW AMQP 1.0 支持

AMQP 通道为兼容 AMQP 1.0 的应用程序提供某种级别的支持。

AMQP 通道提供了一组 AMQP 1.0 协议子集。 您可以将 MQ Light 客户机或其他兼容 AMQP 1.0 的客户机连接到 IBM MQ AMQP 通道。 要使用 AMQP 通道支持的所有消息传递功能，必须正确设置某些 AMQP 1.0 字段的值。

此信息概括了格式化 AMQP 字段的方式，并列出了 AMQP 通道不支持的 AMQP 1.0 规范的特征。

不支持以下 AMQP 1.0 规范的特征或者限制其使用：

链接名称

AMQP 通道期望 AMQP 链接名称遵循以下三种格式之一：

- 普通主题（用于发布和预订）
  - 发布消息：纯主题字符串（例如，链接名称 "/sports/football"）会导致在 /sports/football 主题上发布消息。
  - 预订主题以接收消息：纯文本主题字符串（例如，链接名称 "/sports/football"）导致在 /sports/football 主题上定义预订。
- 专用详细主题（用于预订）
  - 用于描述专用预订的详细主题字符串，格式为："private:*topic string*"（例如，"private:/sports/football"）。此行为与普通主题字符串相同。 private 声明用于区分特定于特定 AMQP 客户机的预订与客户机之间共享的预订。
- 共享详细主题（用于预订）
  - 用于描述共享预订的详细主题字符串，格式为："share:*share name*:*topic string*"（例如："share:bbc:/sports/football"）。

有关 AMQP 消息与 IBM MQ 消息之间映射方式的更多信息，请参阅将 AMQP 字段映射到 IBM MQ 字段上（入局消息）。

主题字符串、共享名称和客户机标识的最大长度

主题字符串、共享名称和客户机标识的总长度必须不超过 10237 字节。 另外，客户机标识的最大长度为 256 个字符。

这些最大长度意味着您可以采用以下某种组合：

- 较短的共享名称配上较长的主题字符串
- 较长的共享名称配上较短的主题字符串

容器标识

AMQP 通道预期 AMQP Open 行为原语的容器标识包含唯一的 MQ Light 客户机标识。 MQ Light 客户机标识的最大长度为 256 个字符，此标识可包含字母数字字符、百分号 (%)、斜杠 (/)、句点 (.) 和下划线 (_)。

会话

AMQP 通道仅支持单个 AMQP 会话。 尝试创建多个 AMQP 会话的 AMQP 客户机会收到错误消息，并断开与通道的连接。

事务

AMQP 通道不支持 AMQP 事务。 尝试协调新事务的 AMQP 附加帧或尝试声明新事务的 AMQP 传输帧会遭拒绝并收到错误消息。

交付状态

AMQP 通道只支持处置帧的“已接受”交付状态。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 映射 AMQP 和 IBM MQ 消息字段

AMQP 消息包括头、交付注释、消息注释、属性、应用程序属性、主体以及页脚。

AMQP 消息包括以下部分：

头

可选的头包含消息的五个固定属性：

- **持久性** - 指定持久性要求
- **优先级** - 相关消息优先级
- **生存时间** - 生存时间（以毫秒为单位）
- **第一获取方** - 如果这是 true，消息没有被任何其他链接获取。
- **交付计数** - 先前失败的交付尝试次数。

交付注释

可选。 为不同目标受众指定消息的非标准头属性。 交付注释将信息从发送对等项传达给接收对等项。

消息注释

可选。 为不同目标受众指定消息的非标准头属性。 消息注释部分用于针对基础架构的消息属性，应跨每个交付步骤传播。

属性

可选。 此部分等同于 MQ 消息描述符。 它包含以下固定字段：

- **消息标识** - 应用程序消息标识
- **用户标识** - 所创建用户的标识
- **目标** - 消息发往的节点地址
- **主题** - 消息的主题
- **回复** - 回复发送至的节点
- **相关标识** - 应用程序相关标识
- **内容类型** - MIME 内容类型

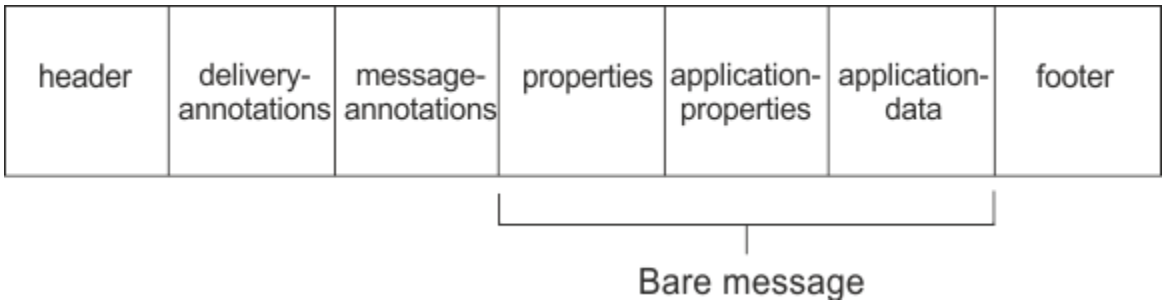
- **内容编码** - MIME 内容类型。 用作内容类型的修饰符。
- **绝对到期时间** - 将此消息视为到期的时间
- **创建时间** - 创建此消息的时间
- **组标识** - 此消息所属的组
- **组序列** - 此消息在组中的序列号
- **回复组标识** - 回复消息所属的组

应用程序属性
等同于 MQ 消息属性。

正文
等同于 MQ 用户有效内容。

页脚
可选。 页脚用于说明消息或传送的详细信息，此详细信息仅在已构建或查看整个裸消息后才可以计算或求值（例如，消息散列、HMAC、签名和加密详细信息）。

下图演示了 AMQP 消息格式：



这些属性、应用程序属性和应用程序数据部分称为“裸消息”。 这是发送方发送的消息，是不可变的。 接收方可看到整个消息，包括头、页脚、传送注释和消息注释。

有关 AMQP 1.0 消息格式的完整描述，请参阅位于以下站点的“OASIS 标准”：https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf。

相关任务
创建和使用 AMQP 通道
保护 AMQP 客户机安全

ULW 将 IBM MQ 字段映射到 AMQP 字段（外发消息）

发布 IBM MQ 消息时，IBM MQ 会将其发送到 AMQP 使用者，并将 IBM MQ 消息的部分属性填充到同等的 AMQP 消息属性。

头

仅在头中的五个字段之一包含非缺省值时包括头。 头中仅包含具有非缺省值的字段。 五个头字段起初从同等的 `mq_amqp.Hdr` 属性衍生（如果已设置），然后按下面表格所示进行修改：

表 *105:* 头字段映射

字段	缺省值	值
持久	false	在 `MQMD.Persistence` 设置为 `MQPER_PERSISTENT` 时为 true，否则为 false。
priority	4	来自 `mq_amqp.Hdr.Pri`（如果设置），否则来自 `MQMD.Priority`（如果设置）。 如果均未设置，设为 4。

表 *105:* 头字段映射 *(*继续*)*		
字段	**缺省值**	**值**
ttl	不适用	MQMD.Expiry（以毫秒为单位）。 如果 MQMD.Expiry 的值为 MQEI_UNLIMITED，那么设为 AMQP ttl 字段的最大值
first-acquirer	false	来自 mq_amqp.Hdr.Fac（如果设置），否则为 false。
delivery-count	0	来自 mq_amqp.Hdr.Dct（如果设置），否则为 0。

delivery-annotation

AMQP 通道根据需要设置。

message-annotation

未包括。

属性

如果**属性**已设置，那么它们将取自同等的 mq_amqp.Prp 属性，不经过修改。 如果消息最初不是 AMQP 消息（也就是说，PutApplType 不是 MQAT_AMQP），那么将生成属性部分，如下表所描：

表 *106:* 属性字段映射	
名称	**值**
message-id	MQMD.MsgId 设置为二进制。
用户标识	UTF-8 格式的 MQMD.UserIdentifier 在网络字节顺序设置为二进制。
到	从中获取消息的队列，或针对发布内容，更改为主题字符串。
主题	未设置。
应答	如果非空，为 MQMD.ReplyToQ，否则未设置。
correlation-id	如果非空，MQMD.CorrelId 设置为二进制，否则不设置。
content-type	未设置。
content-encoding	未设置。
absolute-expiry-time	未设置。
creation-time	MQMD.PutDate 和 MQMD.PutTime 字段用于生成时间戳记。
group-id	未设置。
group-sequence	未设置。
reply-to-group-id	未设置。

application-properties

“usr” 组中的所有 IBM MQ 属性都添加为 **application-properties**。

主体

AMQP 通道通过转换执行获取，以将 IBM MQ 有效内容转换为 UTF-8 格式。

如果 IBM MQ 有效内容不包含 AMQP 消息，那么 IBM MQ 有效内容在主体中设置为格式为 MQFMT_STRING（前提是转换为 UTF-8 成功）的单个字符串数据部分，或者设置为单个二进制数据部分。

如果包括 AMQP 格式消息，那么将其设置为主体。 如果主体是 AMQP 序列，那么 AMQP 消息之前的任何 IBM MQ 头（不包括消息句柄中返回的消息属性）都会在前面添加一个二进制值。 否则，将丢弃 IBM MQ 头。

页脚

不包括页脚。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

相关参考

MQMD - 消息描述符

ULW 映射 AMQP 字段到 IBM MQ 字段（入局消息）

当 AMQP 通道收到消息，并将它放入 IBM MQ 时，它会将 AMQP 消息的部分属性传播到同等的 IBM MQ 消息属性。

映射入站 AMQP 消息时，以下限制适用：

- 如果属性部分中的 `message-id` 或 `correlation-id` 字段是 uuid 或 ulong ，那么将拒绝该消息。
- 任何 `message-annotations` 都会导致消息被拒绝。
- 允许使用 `delivery-annotations` 和 `footer` 部分，但不会将其传播到 IBM MQ 消息中。

以下子小节显示 AMQP 消息的 IBM MQ 表达式。

消息描述符

表 *107: AMQP* 消息的消息描述符

字段	值
StrucId	MQMD_STRUC_ID
版本	MQMD_VERSION_1
报告	MQRO_NONE
MsgType	MQMT_DATAGRAM
到期	AMQP 消息头中 `ttl` 字段获取的值
Feedback	MQFB_NONE
编码	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
格式	参见有效内容
优先级	从 AMQP 消息头中的 `priority` 字段获取的值。 如果设置了该项，那么限制为最多 9 个。 如果未设置该项，那么将使用缺省值 4。
持久	如果 AMQP 消息头中的 `durable` 字段设置为 true ，请设置为 MQPER_PERSISTENT。 否则，设为 MQPER_NOT_PERSISTENT。
MagId	队列管理器分配唯一的 24 字节 MsgId。
Correlld	从 AMQP 属性的 `correlation-id` 字段中获取的值 (如果已设置)。 设置为 24 字节二进制值。 否者，设置为 MQCI_NONE/。

表 *107: AMQP* 消息的消息描述符 *(*继续*)*	
字段	**值**
BackoutCount	0
ReplyToQ	""
ReplyToQMgr	""
UserIdentifier	设置为连接 AMQP 通道的已认证的用户的标识
AccountingToken	MQACT_NONE
ApplIdentityData	十六进制字符串。 设置为 AMQP 通道的 MQ 连接标识的最后 8 个字节。
PutApplType	MQAT_AMQP
PutApplName	
PutDate	从 AMQP 属性的 `creation-time` 字段中获取的值 (如果已设置)。 否则，设为当前日期。
PutTime	从 AMQP 属性的 `creation-time` 字段中获取的值 (如果已设置)。 否则，设为当前时间。
ApplOriginData	""

消息属性

设置消息属性有两个原因：

- 允许通过队列管理器传播 AMQP 消息的部分，而不影响消息的有效内容。
- 允许选择 `application-properties`。

下表显示从 AMQP 消息设置的属性：

表 *108: AMQP* 消息属性			
属性名	**MQRFH2 名称**	**类型**	**描述**
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	AMQP 通道的识别字符串。 用于生成消息，以便相关方（例如诊断问题的服务团队）可以知道是哪个版本放入消息。 值不由队列管理器进行验证，且不得在外部进行记录。
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	AMQP 消息的版本。 如果不存在，假定为“1.0”。 值不由队列管理器进行验证。
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	API 的识别字符串。 用于发送 AMQP 消息至通道，以便相关方（例如诊断问题的服务团队）可以知道是哪个版本放入消息。 值不由队列管理器进行验证，且不得在外部进行记录。
AMQPDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	AMQP 消息头中 `durable` 字段的值 (如果已设置)。
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	AMQP 消息头中 `priority` 字段的值 (如果已设置)。
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	AMQP 消息头中 `ttl` 字段的值 (如果已设置)。

表 108: AMQP 消息属性 (继续)			
属性名	**MQRFH2 名称**	**类型**	**描述**
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLEAN	AMQP 消息头中 `first-acquirer` 字段的值 (如果已设置)。
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	AMQP 消息头中 `delivery-count` 字段的值 (如果已设置)。
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	AMQP 属性 `message-id` 字段的值 (如果设置为字符串)。
		MQTYPE_BYTE_STRING	AMQP 属性 `message-id` 字段的值 (如果设置为字节字符串)。
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	AMQP 属性 (如果已设置) 中 `user-id` 字段的值。
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `to` 字段的值。
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `subject` 字段的值。
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `reply-to` 字段的值。
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	AMQP 属性 `correlation-id` 字段的值 (如果设置为字符串)。
		MQTYPE_BYTE_STRING	AMQP 属性 `correlation-id` 字段的值 (如果设置为字节字符串)。
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `content-type` 字段的值。
AMQPContentEncoding	mq_amqp.Prp.Cneing	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `content-encoding` 字段的值。
AMQPAbsoluteExpiryTime	mq_amqp.Prp.Aet	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `absolute-expiry-time` 字段的值。
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `creation-time` 字段的值。
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `group-id` 字段的值。
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	AMQP 属性 (如果已设置) 中 `group-sequence` 字段的值。
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	AMQP 属性 (如果已设置) 中 `reply-to-group-id` 字段的值。

AMQP 消息的每个应用程序属性被设为 IBM MQ 消息属性。 必须以相同的字节对字节重新构成 `application-properties` 部分，因此以下限制适用:

- 如果 MQSETMP 验证代码拒绝应用程序属性，消息将被拒绝。 例如:
  - 属性名称长度限制为 MQ_MAX_PROPERTY_NAME_LENGTH。
  - 属性名称必须遵守 Java 标识的 Java 语言规范定义的规则。
  - 属性名不得以 `JMS` 或 `usr.JMS` 开头，但已记录的可设置的 JMS 属性除外。
  - 属性名不能是 SQL 关键字。

- 包含 Unicode 字符 U+002E (".") 的应用程序属性导致消息被拒绝。 属性必须在 JMS 使用的“usr”属性组中可以表述。
- 仅支持空值、布尔值、字节、简短型、整数、长整型、浮点型、双精度值、二进制和字符串属性。 任何其他类型的应用程序属性将导致消息被拒绝。

有效内容

- 对于具有单个二进制数据部分的 AMQP body ， 二进制数据 (不包括 AMQP 位) 将作为 IBM MQ 有效内容放置，格式为 MQFMT_NONE。
- 对于具有单个字符串数据部分的 AMQP body ， 字符串数据 (不包括 AMQP 位) 将作为 IBM MQ 有效内容放置，格式为 MQFMT_STRING。
- 否则， AMQP body 将以 MQFMT_AMQP 格式按原样构成有效内容。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 具有 AMQP 的消息传递可靠性

IBM MQ Light API 有四种功能，用于控制与 AMQP 应用程序之间的消息传递的可靠性。

这些字段为:

- 第 657 页的『消息服务质量 (QOS)』
- 第 658 页的『订户自动确认』
- 第 658 页的『预订生存时间』
- 第 658 页的『消息持久性』

消息服务质量 (QOS)

MQ Light API 提供两种服务质量:

- 至多一次
- 至少一次

您可以选择希望发布者和订户使用的服务质量。

如果您要使用 MQ Light 客户机，请将客户机或预订 **qos** 选项设置为 *QOS_AT_MOST_ONCE* 或 *QOS_AT_LEAST_ONCE*。

如果您要使用其他 AMQP 客户机，请根据您想要实现服务质量，将传输帧（对于发布者）或处置帧（对于订户）的 **settled** 属性设置为 *true* 或 *false*。

服务质量决定何时废弃来自对话的 `sending` 端的消息。

发布

如果发布者选择 **QOS** *0*（最多一次），那么发布者在废弃其消息副本之前不会等待来自队列管理器的确认。

如果在发送完成之前到队列管理器的连接失败，那么订户可能不会收到消息。

如果发布者选择 **QOS** *1*（至少一次），那么发布者将等待队列管理器确认消息已写入订户队列，然后再废弃其消息副本。

如果在发送期间到队列管理器的连接失败，发布者将在其重新连接到队列管理器后重新发送消息。

预订

如果订户选择 **QOS** *0*，那么队列管理器在废弃其消息副本之前不会等待来自订户的确认。

如果在订户收到消息之前到订户的连接失败，该消息可能会丢失。

如果订户选择 **QOS** *1*，那么队列管理器在废弃其消息副本之前将等待来自订户的确认。

如果在订户收到消息之前到订户的连接失败，队列管理器将保留该消息。 当队列管理器重新连接到该订户或另一个订户（如果共享预订）时，队列管理器会将消息重新发送给订户。

订户自动确认

如果订户选择 **QOS** *1*（至少一次），那么它必须在队列管理器废弃其副本之前确认收到每条消息。 订户可以决定何时确认消息。

auto-confirm 设置为 *true* 时，一旦客户机通过网络成功接收到消息，MQ Light 客户机将自动确认每个消息的传递。

这将确保如果存在网络故障，会将消息重新传递到应用程序。 但是，如果应用程序在 MQ Light 客户机确认消息和其自身处理消息之间出现故障，应用程序仍可能会丢失消息。

auto-confirm 设置为 *false* 时，MQ Light 客户机不会自动确认消息的传递，而是将其留给应用程序来决定何时应确认。

这将允许应用程序先对外部资源（如数据库或文件）进行更新，然后再向队列管理器确认消息现在已处理并且可以丢弃。

预订生存时间

应用程序进行预订时，它会选择预订以及该预订的消息存储到的目标在应用程序断开连接后是否继续存在。

MQ Light 预订选项 **ttl** 用于指定在应用程序断开连接后预订继续存在的时间（以毫秒为单位）。 如果应用程序在此之前重新连接，那么将恢复预订，并且应用程序可继续使用来自该预订的消息。

如果生存时间内应用程序未重新连接，那么将移除预订并丢失其目标上存储的所有消息，即使这些消息是持久消息。

如果消息不该丢失，那么必须向应用程序指定生存时间值，该值应足够高以确保消息不会在系统停运期间丢失。

消息持久性

消息的持久性由发布应用程序、预订应用程序和 IBM MQ 主题对象的配置控制。

如果 AMQP 订户使用 **QOS** *0*（最多一次）并创建非持久预订，那么 AMQP 通道始终会将非持久消息放置到订户队列中，而不考虑以下文本中描述的其他选项。

请注意，如果队列管理器已停止，那么预订和消息将丢失。

如果 AMQP 发布者将 AMQP **durable** 标头设置为 *true*，那么 AMQP 通道会将持久消息放入到订户队列中。

如果队列管理器因任何原因停止，在重新启动队列管理器后，消息仍对订户可用。

如果未设置 **durable** 标头，那么 AMQP 通道会根据相关 IBM MQ 主题对象的 **DEFPSIST** 属性来选择是否保存已发布消息。

缺省情况下，这是 SYSTEM.BASE.TOPIC，它使用的 **DEFPSIST** 属性值 *NO*（不保存）。

注意: 最新版的 MQ Light 客户机不支持设置 AMQP 持久头。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 用于将 AMQP 客户机与 IBM MQ 配合使用的拓扑

以下拓扑示例将帮助您开发可与 IBM MQ 配合使用的 AMQP 客户机。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 通过 IBM MQ 进行通信的 AMQP 客户机

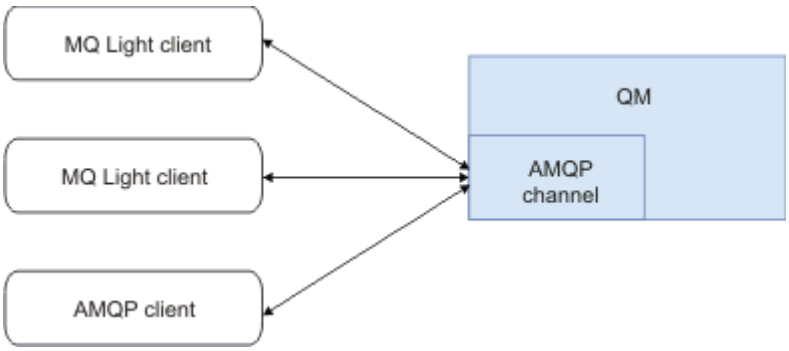
您可以将 IBM MQ 用作 IBM MQ Light 或符合 AMQP 1.0 的任何应用程序的消息传递提供程序。 虽然任何 AMQP 1.0 客户机都能够连接到 AMQP 通道，但某些 AMQP 功能（例如事务或多个会话）不受支持。

通过定义一个或多个 AMQP 通道，AMQP 1.0 客户机可以连接到队列管理器，并将消息发送到主题字符串。 客户机还可以预订主题模式以接收与该模式匹配的消息。

在以下场景中，发送和接收消息的唯一应用程序是 MQ Light 或 AMQP 1.0 应用程序。

应用程序可以选择通过预订主题字符串创建的目标是否具有持久性，以便应用程序暂时与队列管理器断开连接时不会丢失消息。

应用程序还可以选择消息的保留时间，在此时间之后将从目标中清除消息。



相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW AMQP 客户机与 IBM MQ 应用程序交换消息

通过定义和启动 AMQP 通道，MQ Light 或 AMQP 1.0 应用程序可以发布将由现有 MQ 应用程序接收的消息。 通过 AMQP 通道发布的消息将全部发送到 MQ 主题，而非 MQ 队列。 通过 MQSUB API 调用来创建预订的 MQ 应用程序将接收 AMQP 1.0 应用程序发布的消息，前提是该 MQ 应用程序使用的主题字符串或主题对象与该 AMQP 客户机发布的主题字符串匹配。

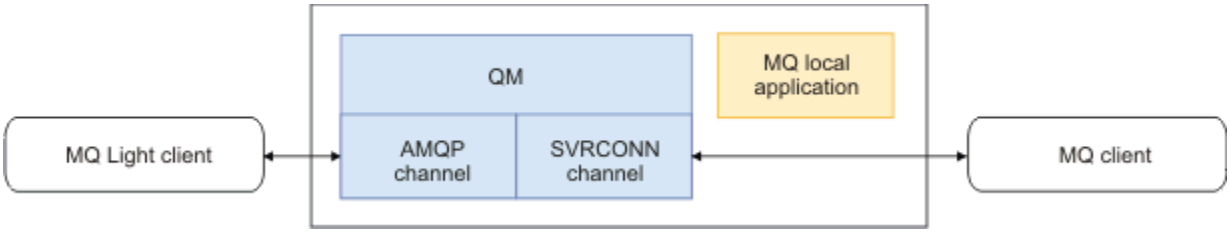
AMQP 消息数据、特性和属性都在 MQ 应用程序收到的 MQ 消息上进行设置。 有关 AMQP 至 MQ 消息映射的更多信息，请参阅将 AMQP 字段映射到 IBM MQ 字段上（入局消息）。

如果 MQ 应用程序创建了持久预订，那么 AMQP 应用程序发布的消息将存储在支持该预订的队列上。 然后，在 MQ 应用程序恢复预订后，此应用程序将接收到这些消息。 如果 AMQP 应用程序指定了消息生存时间，而 MQ 应用程序未在该生存时间内重新连接，那么消息将在队列中到期。

MQ Light 或 AMQP 1.0 应用程序还可以使用现有 MQ 应用程序发布的消息。 MQ 应用程序发布到 MQ 主题或主题字符串的消息将被 AMQP 1.0 应用程序接收，前提是该应用程序预订了与所发布主题字符串匹配的主题模式。

如果 AMQP 1.0 应用程序为预订指定了生存时间值，而 AMQP 应用程序断开连接的时间超过该生存时间，那么预订将在队列管理器中到期，并会丢失预订队列上存储的所有消息。

MQMD 字段、消息属性和应用程序数据都在 AMQP 应用程序接收的 AMQP 消息上进行设置。 有关 MQ 至 AMQP 消息映射的更多信息，请参阅将 AMQP 字段映射到 IBM MQ 字段上（出局消息）。



相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 将 AMQP 客户机配置为可在 IBM MQ 队列上直接与应用程序交互

IBM MQ AMQP 实施仅支持发布/预订机制。 AMQP 客户机无法直接将消息放入 IBM MQ 队列中或者从这些队列中获取消息。 为了让发布 AMQP 客户机能够向现有队列发送消息，或者让预订 AMQP 客户机能够接收已放入队列的消息，必须创建一些其他对象。

概述

例如，假定有一个应用程序从输入队列中获取消息 IN_QUEUE 并将这些消息放入输出队列 OUT_QUEUE。 AMQP 客户机可以将消息放置到 IN_QUEUE，并从 OUT_QUEUE 获取消息。

注: 无需对应用程序本身进行任何更改。

为了让 AMQP 发布者能够将消息放入队列中，需要针对 AMQP 客户机发布到（以预定队列为目标）的主题字符串创建一个管理预订；请参阅第 660 页的『将消息发送到应用程序：』。

为了让 AMQP 订户能够从队列中获取消息，需要用同名的别名队列替换该队列，并以主题对象（表示 AMQP 客户机所预订的主题字符串）为目标；请参阅第 660 页的『从应用程序获取消息：』

将消息发送到应用程序：

应用程序已从 IN_QUEUE 获取消息，并且您希望 AMQP 客户机能够发布消息，以便这些消息进入此队列由应用程序进行处理。

要执行此操作，请创建一个新的管理预订，此预订要从中接收消息的主题字符串就是 AMQP 客户机要发布到的主题字符串。 此预订的目标队列是应用程序的输入队列：IN_QUEUE。

发布到该管理预订的已定义主题字符串的任何消息都将路由到已定义目标（在本例中为 IN_QUEUE）。

假定 AMQP 客户机发布到主题字符串 /application/in，那么您可以使用以下 MQSC 命令来创建管理预订 APP_IN:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

定义此对象后，发布到 /application/in 的所有消息都将路由到目标 IN_QUEUE，这些消息将由应用程序以与其他应用程序放入此队列中的任何其他消息相同的方式来获取。

从应用程序获取消息：

应用程序正将消息放入 OUT_QUEUE 中，其他客户机可以在其中选取和处理这些消息。

但是，在此情况下，您希望将消息传递到 AMQP 客户机，但是 AMQP 客户机仅使用发布/预订机制，所以无法直接从队列中选取消息。

要将先前接收消息的客户机替换为预订 AMQP 客户机，需要针对 AMQP 客户机所预订的主题字符串创建一个主题对象并创建一个别名队列。

注意: 如果定义了别名队列，并在任何 AMQP 客户机有机会预订之前启动生产应用程序，那么由生产应用程序发送到“队列”（现在为“主题”）的消息将会丢失，因为没有订户。

本文中描述的更改仅用预订 AMQP 客户机替换先前接收消息的客户机。 要结合使用 AMQP 客户机和其他客户机来获取消息，需要进行更广泛的更改。

假定 AMQP 客户机预订了主题字符串 `/application/out`，您可以使用以下 MQSC 命令来定义主题对象 APP_OUT：

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

传递到此主题对象的任何消息都会传递到预订相同主题字符串的 AMQP 客户机。

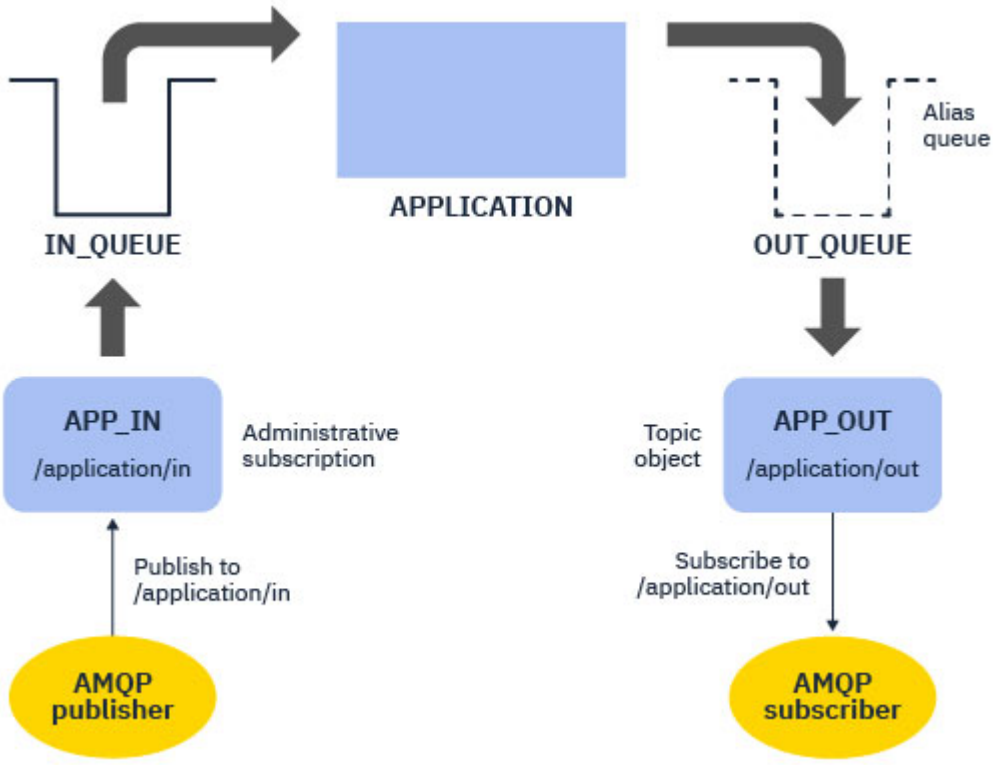
然后，您需要确保将应用程序放入到 OUT_QUEUE 的消息传递到这个新的主题对象，以便将这些消息发送到预订客户机。

要执行此操作，请使用以下 MQSC 命令将现有队列 OUT_QUEUE 替换为同名的别名队列，并将其替换为刚刚创建的主题对象的目标类型：

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

现在，应用程序放入到 OUT_QUEUE 的消息不会在队列上等待被选取；而是会传递到此别名队列的目标，即，新的主题对象 APP_OUT。

然后，AMQP 客户机（预订此主题对象 `/application/out` 所表示的主题字符串）从别名队列接收发送到此主题对象的任何消息。



相关任务

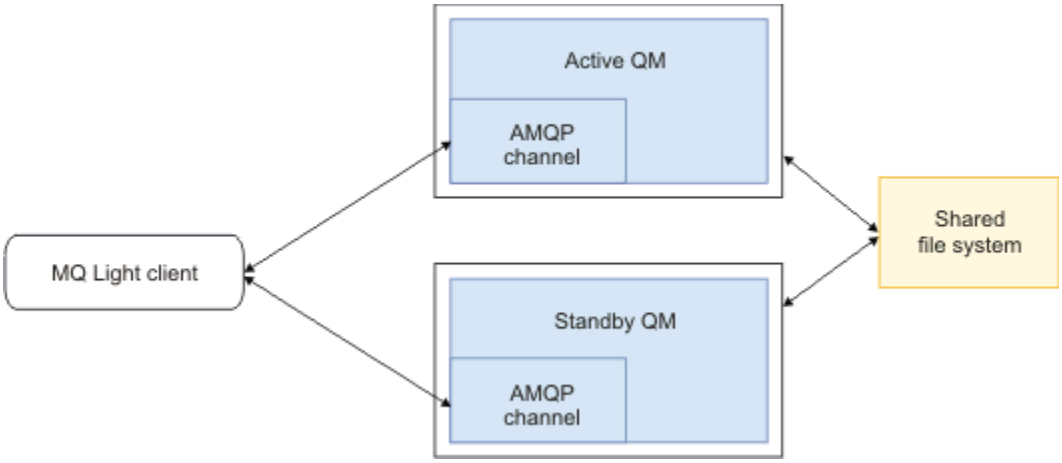
创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 配置 AMQP 客户机以实现高可用性

您可以配置 MQ Light 或 AMQP 1.0 应用程序以连接至 IBM MQ 多实例队列管理器的活动实例，并故障转移到高可用性 (HA) 对中多实例队列管理器的备用实例。 为此，为 AMQP 应用程序配置两对 IP 地址和端口。

您可以使用定制函数来配置 MQ Light API，当客户机与服务器的连接断开时将调用该函数。 该函数可以连接到备选 IP 地址（例如，备用 IBM MQ 队列管理器）或原始 IP 地址。 对于其他 AMQP 客户机，如果支持包含多个连接端点的配置，那么可以为该应用程序配置两对主机端口，并使用 AMQP 库提供的重新连接功能来切换到备用队列管理器。



相关任务

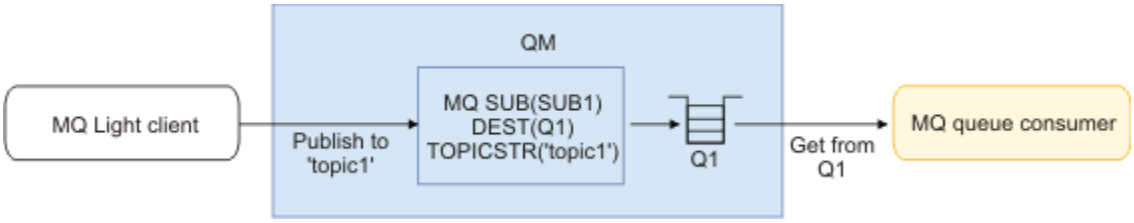
创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 为 AMQP 客户机配置发布/预订

AMQP 客户机可以通过 IBM MQ 预订（用于将消息传递到现有应用程序读取的 IBM MQ 队列）发布到主题。 如果您希望 MQ Light 或 AMQP 1.0 应用程序将消息发送到现有 IBM MQ 应用程序，并且该应用程序配置为从队列中读取消息，那么必须在队列管理器上定义受管 IBM MQ 预订。

将预订配置为使用与 AMQP 应用程序所用主题字符串匹配的主题模式。 将预订目标设置为 IBM MQ 应用程序从中获取或浏览消息的队列的名称。



相关任务

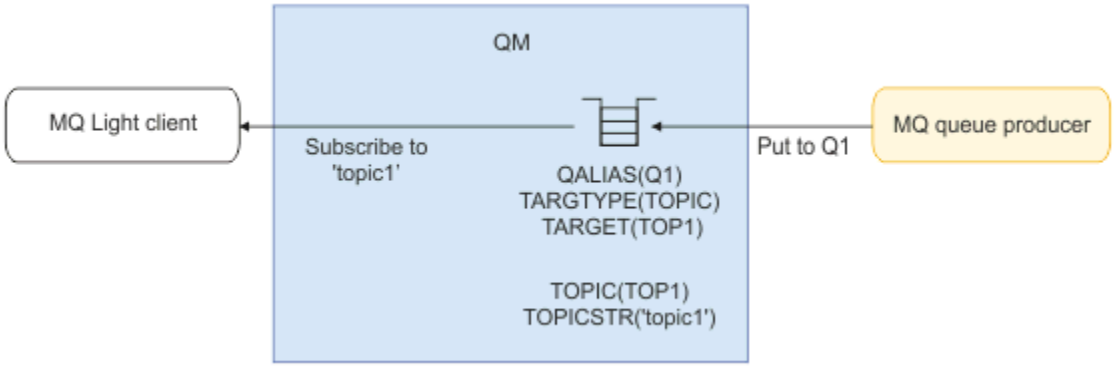
创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 用于使用队列别名从 IBM MQ 应用程序接收消息的 AMQP 客户机

AMQP 客户机可以预订主题，并接收由 IBM MQ 应用程序放入别名队列的消息。 如果您希望 MQ Light 或 AMQP 1.0 应用程序从现有 IBM MQ 应用程序接收消息，而该应用程序配置为将消息放入队列，那么必须在队列管理器上定义队列别名 (QALIAS)。

该队列别名必须与 IBM MQ 应用程序为放入消息而打开的队列同名。 该队列别名必须指定基本类型 TOPIC 和基本对象“IBM MQ 主题对象”，该主题对象中的主题字符串与 AMQP 应用程序所预订的主题模式匹配。



相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 用于将请求提交到应用程序服务器并使用来自应用程序服务器的响应的 AMQP 客户机

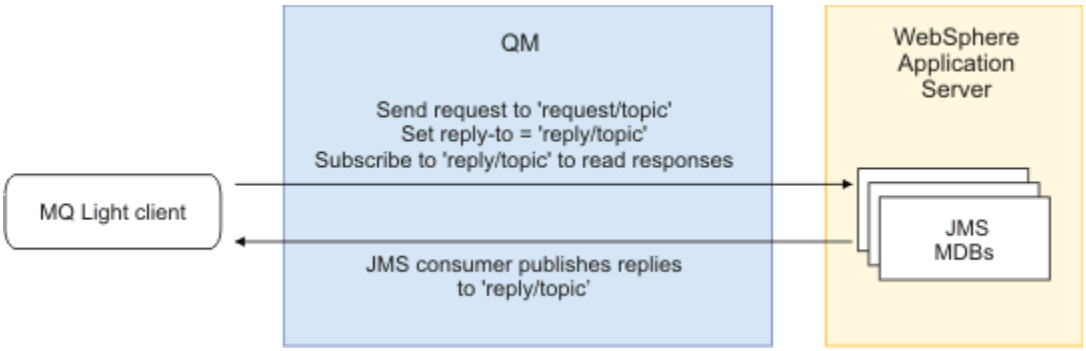
MQ Light 或其他 AMQP 客户机可以将请求提交到应用程序服务器中运行的消息驱动 bean，并使用来自回复主题的响应。 IBM MQ 支持可在 IBM MQ 所发布消息中设置回复主题的 AMQP 1.0 应用程序。 当发布已设置回复属性的 AMQP 消息时，回复字段的值会设置为一个 JMS 属性，以便发送给 JMS 使用者。 此设置允许 JMS 使用者从消息中读取回复主题，并将响应消息发回给 AMQP 客户机。

JMS 属性为 **JMSReplyTo**。 AMQP 回复字符串必须是以下某种类型：

- 主题字符串。 例如，`'reply/topic'`
- 格式为 `amqp://host:port/[topic-string]` 的 AMQP 地址 URL。 例如，`amqp://localhost:5672/reply/topic`

如果您将 AMQP 地址 URL 指定为回复字段，那么在设置 **JMSReplyTo** 属性之前，URL 末尾的除主题字符串以外的所有内容都将移除。

有关从 AMQP 回复地址至 **JMSReplyTo** 属性的映射的更多信息，请参阅将 AMQP 字段映射到 IBM MQ 字段（入局消息）。



相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

ULW 将 MQ Light 应用程序部署到本地 IBM MQ 环境

IBM MQ 支持 IBM MQ Light 消息传递 API，因此您可以使用 IBM MQ 将 MQ Light 应用程序部署到本地 IBM MQ 环境中。

您可以将 MQ Light 应用程序部署到 IBM MQ 队列管理器，从而使 MQ Light 应用程序能够与已连接到 IBM MQ 的现有企业应用程序通信，如下图所示：

MQ Light 应用程序可以与现有 IBM MQ 应用程序共享主题空间，从而使它们能够与现有企业系统进行交互。

相关任务

创建和使用 AMQP 通道

保护 AMQP 客户机安全

V 9.1.0 使用 IBM MQ 开发 REST 应用程序

您可以开发 REST 应用程序以发送和接收消息。 根据平台和功能的不同，IBM MQ 支持不同的 REST API。

以下选项是 IBM MQ 支持的选项（可选择这些选项以向 IBM MQ 发送消息或者从 IBM MQ 接收消息）：

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

IBM MQ messaging REST API

您可以使用 messaging REST API 以纯文本格式发送，接收和发布 IBM MQ 消息。 缺省情况下，已启用 messaging REST API。

支持多种不同的 HTTP 头，可使用这些头来设置通用消息属性。

messaging REST API 已经与 IBM MQ 安全性完全集成。 要使用 messaging REST API，用户必须向 mqweb 服务器进行认证，并且必须是具有 MQWebUser 角色的成员。

有关更多信息，请参阅第 665 页的『使用 REST API 进行消息传递』。 另请参阅 IBM Developer 上的教程：IBM MQ 消息传递 REST API 入门，其中包括用于使用消息传递 REST API 的 Go 和 Node.js 示例。

IBM z/OS Connect EE

IBM z/OS Connect EE (zCEE) 是一种 z/OS 产品，支持在现有 z/OS 资产（例如，CICS 或 IMS 事务）以及 IBM MQ 队列和主题上构建 REST API。 针对用户隐藏了现有 z/OS 资产。 这允许您通过 REST 启用资产，而无需更改这些资产或使用这些资产的任何现有应用程序。

利用 zCEE，您可以轻松地构建 REST API，以用于发送和接收 JSON 数据并可选择将此数据转换为众多大型机应用程序期望的更传统的语言结构。 例如，COBOL 副本。

基于 Eclipse 的 API 编辑器可用于构建全面的 RESTful API，使用查询参数和 URL 路径段，在流经 zCEE 运行时期间处理 JSON 格式。

zCEE 可用于将 IBM MQ 队列和主题作为服务进行公开。 支持以下两种不同的服务类型：

- 单向服务：其提供的功能类似于 IBM MQ Bridge for HTTP 附带的功能，在其中，HTTP POST 用于发送消息，HTTP DELETE 用于以中断性方式获取消息。 该网桥的主要优势是内置数据转换支持以及可使用 API 编辑器来构建更全面的 RESTful API。
- 双向服务：可在请求-响应样式的后端应用程序所使用的一对队列上提供 REST API。 调用者向双向服务发出 HTTP POST 并发送 JSON 数据。 此数据将放入由后端应用程序处理的请求队列中，并且将响应放入响应队列中。 将检索此响应并将其作为 POST 响应主体发回给调用者。

zCEE 在 IBM MQ 8 和更高版本上受支持。 有关更多信息，请参阅针对 z/OS Connect 的 IBM MQ for z/OS 服务提供程序。

IBM Integration Bus

IBM Integration Bus 是 IBM 领先的集成技术，可用于将应用程序和系统连接在一起，而不考虑它们支持的消息格式和协议。

IBM Integration Bus 始终支持 IBM MQ，并提供 *HTTPInput* 和 *HTTPRequest* 节点（可用于在 IBM MQ 上构建 RESTful 接口）以及许多其他系统（例如，数据库）。

IBM Integration Bus 的功能远不止于在 IBM MQ 上提供简单 REST 接口。 其功能包括提供高级有效内容处理、有效内容扩充以及作为 REST API 一部分提供的许多其他增强功能。

有关更多信息，请参阅技术样本，此样本在期望 XML 有效内容的 IBM MQ 应用程序上公开 JSON over REST 接口。

DataPower

DataPower 网关是一个多通道网关，可帮助保护、控制、集成和优化访问一系列系统（包括 IBM MQ）。 其同时提供了硬件和虚拟规格。

多协议网关是 DataPower 提供的一项服务，它可采用一种协议来接收输入并采用另一种协议来生成输出。 特别是，DataPower 可配置为接受 HTTP(S) 数据并通过客户机连接将其路由至 IBM MQ，您可在 IBM MQ 上使用该数据构建 REST 接口。 其他 DataPower 服务（例如，转换）也可用于增强 REST 接口。

有关更多信息，请参阅多协议网关。

V 9.1.0 使用 REST API 进行消息传递

您可以使用 messaging REST API 来执行简单的点到点和发布消息传递。 您可以向主题发布消息、向队列发送消息、浏览队列上的消息以及以中断性方式从队列获取消息。 与 messaging REST API 之间传递的信息采用纯文本格式。

开始之前

注:

- 缺省情况下，已启用 messaging REST API。 您可以禁用 messaging REST API 以阻止所有消息传递。 有关启用或禁用 messaging REST API 的更多信息，请参阅配置 messaging REST API。
- messaging REST API 已与 IBM MQ 安全性相集成。 要使用 messaging REST API，用户必须向 mqweb 服务器进行认证，并且必须是具有 MQWebUser 角色的成员。 用户还必须有权访问指定的队列或主题。 有关 REST API 安全性的更多信息，请参阅 IBM MQ Console 和 REST API 安全性。
- 如果将 Advanced Message Security (AMS) 用于 messaging REST API，请注意将使用 mqweb 服务器的上下文来加密所有消息，而不是使用发布消息的用户的上下文。
- 在接收或浏览消息时，仅支持 IBM MQ MQSTR 格式的消息。 随后，将在同步点下以中断性方式接收所有消息，并且任何未处理的消息都会留在队列中。 IBM MQ 队列可配置为将这些有害消息移至备用目标。 有关更多信息，请参阅第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』。
- messaging REST API 不提供具有事务性支持的一次性消息交付。 如果发出 HTTP POST 并且在客户机接收 HTTP 响应之前连接失败，那么客户机将无法立即告知消息是否已发送到指定的队列或发布到指定的主题。 如果发出 HTTP DELETE 并且在客户机接收 HTTP 响应之前连接失败，那么可能会从队列中以中断性方式获取了一条消息并丢失了此消息，因为无法回滚中断性获取。
- 在 HTTP POST 操作上，将除去传入字符串中的换行符。 REST 应用程序

  不应在使用 REST API 发送或发布的消息中使用换行符，因为它们将丢失。

过程

- 第 666 页的『开始使用 messaging REST API』
- 第 668 页的『使用 messaging REST API』
- REST API 错误处理
- REST API 发现
- REST API 本地语言支持

相关参考

消息传递 REST API 参考

相关信息

教程：IBM MQ 消息传递 REST API 入门

V 9.1.0 开始使用 messaging REST API

快速开始使用 messaging REST API，并通过 cURL 来试用一些示例命令。

开始之前

要开始使用 messaging REST API，此任务中的示例具有以下要求：

- 这些示例使用 cURL 来发送用于在队列中放置和获取消息的 REST 请求。 因此，要完成此任务，您需要在系统上安装 cURL。
- 这些示例使用队列管理器 QM1。 请创建同名的队列管理器，或者替换系统上的现有队列管理器。 该队列管理器必须位于与 mqweb 服务器相同的机器上。
- 要完成此任务，您必须是具有特定权限的用户，才能使用 **dspmqweb** 命令：
  - z/OS 在 z/OS 上，您必须有权运行 **dspmqweb** 命令，并且必须具有 mqwebuser.xml 文件的写访问权。
  - Multi 在所有其他操作系统上，您必须是特权用户。

IBM i 在 IBM i 上，命令应该在 QSHELL 中运行。

过程

1. 确保已为 messaging REST API 配置 mqweb 服务器：
   - 如果尚未配置 mqweb 服务器以供 administrative REST API， administrative REST API for MFT, messaging REST API 或 IBM MQ Console 使用，请配置 mqweb 服务器。有关使用基本注册表为 mqweb 服务器创建基本配置的更多信息，请参阅 mqweb 服务器的基本配置。
   - 如果已配置 mqweb 服务器，请确保已添加相应的用户，以便在步骤 5 中启用消息传递
2. z/OS
   在 z/OS 上，设置 WLP_USER_DIR 环境变量，以便您可以使用 **dspmqweb** 命令。 通过输入以下命令，将变量设置为指向您的 mqweb 服务器配置：

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   ，其中 *WLP_user_directory* 是传递到 crtmqweb 的目录的名称。 例如：

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   有关更多信息，请参阅创建 mqweb 服务器。
3. 通过输入以下命令确定 REST API URL：

   ```
   dspmqweb status
   ```

   以下步骤中的示例假定 REST API URL 是缺省 URL https://localhost:9443/ibmmq/rest/v1/。 如果您的 URL 与缺省 URL 不同，请在以下步骤中替换您的 URL。
4. 在队列管理器 QM1 上创建队列 MSGQ。 此队列用于消息传递。 请使用以下方法之一：
   - 对 administrative REST API 的 queue 资源使用 POST 请求，并以 mqadmin 用户身份进行认证：

     ```
     curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
     data "{\"name\":\"MSGQ\"}"
     ```

   - 使用 MQSC 命令：

z/OS 在 z/OS 上，使用 2CR 源代码代替 **runmqsc** 命令。 有关更多信息，请参阅在 z/OS 上可从中发出 MQSC 命令的源。

a. 输入以下命令以针对队列管理器启动 **runmqsc**：

```
runmqsc QM1
```

b. 使用 **DEFINE QLOCAL** MQSC 命令创建队列：

```
DEFINE QLOCAL(MSGQ)
```

c. 输入以下命令以退出 **runmqsc**：

```
end
```

5. 对您在 mqweb 服务器的基本配置的步骤 5 中添加至 mqwebuser.xml 的用户授予访问队列 MSGQ 的权限。 在使用 myuser 的位置替换您的用户：

- z/OS 在 z/OS 上：

  a. 向用户授予访问队列的权限：

  ```
  RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
  PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
  ```

  b. 向 mqweb 启动的任务用户标识授予在队列上设置所有上下文的权限：

  ```
  RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
  PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
  ```

- Multi 在所有其他操作系统上，如果用户属于 mqm 组，那么已向其授予权限。 否则，请输入以下命令：

  a. 输入以下命令以针对队列管理器启动 **runmqsc**：

  ```
  runmqsc QM1
  ```

  b. 使用 **SET AUTHREC** MQSC 命令向用户授予队列的浏览、查询、获取和放置权限：

  ```
  SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
  PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
  ```

  c. 输入以下命令以退出 **runmqsc**：

  ```
  end
  ```

6. 通过对 message 资源使用 POST 请求，将内容为 Hello World! 的消息放入至队列管理器 QM1 上的队列 MSGQ。 使用 mqwebuser.xml 中的用户标识和密码替换 myuser 和 mypassword：

   已使用基本认证，并且在 cURL REST 请求中设置了具有任意值的 ibm-mq-rest-csrf-token HTTP 标头。 POST、PATCH 和 DELETE 请求需要此附加头。

   ```
   curl -k https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/MSGQ/message -X POST
   -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
   plain;charset=utf-8" --data "Hello World!"
   ```

7. 通过对 message 资源使用 DELETE 请求，以破坏性方式从队列管理器 QM1 上的队列 MSGQ 上的队列 Hello World! 获取消息。 使用 mqwebuser.xml 中的用户标识和密码替换 myuser 和 mypassword：

   ```
   curl -k https://localhost:9443/ibmmq/rest/v1/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
   -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
   ```

   将返回消息 Hello World!。

下一步做什么

- 示例使用基本认证保护请求。 您可以改用基于令牌的认证或基于客户机的认证。 有关更多信息，请参阅对 REST API 和 IBM MQ Console 使用客户机证书认证，以及对 REST API 使用基于令牌的认证。
- 了解有关使用 messaging REST API 和通过查询参数构造 URL 的更多信息：第 668 页的『使用 messaging REST API』。
- V 9.1.2 使用 messaging REST API 时，与队列管理器的连接会汇聚到池中以优化性能。 您可配置最大池大小以及使用池中所有连接时所采取的操作：配置 messaging REST API。
- 浏览可用 messaging REST API 资源和所有可用查询参数的参考信息：messaging REST API 参考。
- 了解 administrative REST API（这是用于 IBM MQ 管理的 RESTful 接口）：使用 REST API 进行管理。
- 了解 IBM MQ Console（这是基于浏览器的 GUI）：使用 IBM MQ Console 进行管理。

V 9.1.0 使用 messaging REST API

在使用 messaging REST API 时，将在 URL 上调用 HTTP 方法以发送和接收 IBM MQ 消息。 HTTP 方法（如 POST）表示要在 URL 所表示的对象上执行的操作类型。 可在查询参数中编码有关操作的其他信息。 操作的执行结果信息可作为 HTTP 响应主体返回。

开始之前

在使用 messaging REST API 之前，请考虑以下事项：

- 您必须向 mqweb 服务器进行认证才能使用 messaging REST API。 您可以使用 HTTP 基本认证、客户机证书认证或基于令牌的认证来进行认证。 有关如何使用这些认证方法的更多信息，请参阅 IBM MQ Console 和 REST API 安全性。
- REST API 区分大小写。 例如，如果队列管理器名为 `qmgr1`，那么在以下 URL 上执行 HTTP POST 会导致错误。

  ```
  /ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- 并非 IBM MQ 对象名称中可使用的所有字符都能在 URL 中直接进行编码。 要直接对这些字符进行编码，必须使用相应的 URL 编码：
  - 必须将正斜杠编码为 %2F。
  - 必须将百分号编码为 %25。
  - 必须将句点编码为 %2E。
  - 必须将问号编码为 %3F。
- 在接收或浏览消息时，仅支持 IBM MQ MQSTR 格式的消息。 随后，将在同步点下以中断性方式接收所有消息，并且任何未处理的消息都会留在队列中。 IBM MQ 队列可配置为将这些有害消息移至备用目标。 有关更多信息，请参阅第 183 页的『在 IBM MQ classes for JMS 中处理有害消息』。

关于此任务

在使用 REST API 对 IBM MQ 队列对象执行消息传递操作时，首先需要构造 URL 来表示此对象。 每个 URL 都以一个前缀开头，此前缀描述了要向其发送请求的主机名和端口。 URL 的其余部分描述了一个特定对象或到此对象的路径（称为资源）。

要在资源上执行的消息传递操作将定义 URL 是否需要查询参数。 它还会定义所使用的 HTTP 方法，以及是将附加信息发送到 URL 还是从 URL 返回附加信息。 此附加信息可作为 HTTP 请求的一部分，或者作为 HTTP 响应的一部分返回。

在构造 URL 后，您可以将 HTTP 请求发送到 IBM MQ。 您可以使用所选编程语言内置的 HTTP 实现来发送请求。 您还可以使用命令行工具（例如，cURL、Web 浏览器或 Web 浏览器附加组件）来发送请求。

要点: 您必须至少执行步骤第 669 页的『1.a』 和第 669 页的『1.b』。

过程

1. 构造 URL：
   a) 通过输入以下命令来确定前缀 URL：

   ```
   dspmqweb status
   ```

   您要使用的 URL 包括 `/ibmmq/rest/` 短语。
   b) 添加队列和关联的队列管理器资源以用于将消息传递到 URL 路径。

   在消息传递引用中，变量段可以在 URL 中由包围其 *{ }*的花括号标识。有关更多信息，请参阅 /messaging/qmgr/{qmgrName}/queue/{queueName}/message。

   例如，要与队列管理器 *QM1* 的关联队列 *Q1* 进行交互，请将 `/qmgr` 和 `/queue` 添加到前缀 URL 以创建以下 URL：

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
   ```

   c) 可选： 将可选查询参数添加到 URL 中。

   向 URL 添加问号、 查询参数、等号和值。

   例如，要最多等待 30 秒使下一条消息变为可用，请创建以下 URL：

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
   ```

   d) 可选： 将更多的可选查询参数添加到 URL 中。

   向 URL 添加和号 (&)，然后重复步骤 1c。
2. 在 URL 上调用相关的 HTTP 方法。 指定任何可选消息有效内容，并提供相应的安全凭证以进行认证。 例如：
   - 使用所选编程语言的 HTTP/REST 实现。
   - 使用 REST 客户机浏览器附加组件或 cURL 之类的工具。

使用 IBM MQ 开发 MQT 应用程序

IBM MQ 提供了对 C、Visual Basic、COBOL、Assembler、RPG、pTAL 和 PL/I 的支持。 这些过程语言使用消息队列接口 (MQI) 来访问消息排队服务。

有关如何采用所选语言编写应用程序的详细信息，请参阅子主题。

有关过程语言的调用接口的概述，请参阅调用描述。 本主题包含 MQI 调用列表，并且每个调用说明如何采用其中各语言对调用进行编码。

IBM MQ 提供数据定义文件来帮助编写应用程序。 有关完整描述，请参阅 第 670 页的『IBM MQ 数据定义文件』。

要帮助选择采用哪种过程语言对程序进行编码，请考虑程序将处理的消息的最大长度。 如果程序将仅处理最大长度已知的消息，那么可以采用任何受支持语言对其进行编码。 如果不知道程序必须处理的消息的最大长度，那么选择的语言将取决于编写的是 CICS、IMS 还是批处理程序：

IMS 和批处理
采用 C、PL/I 或汇编语言对程序进行编码，以使用这些语言提供的工具来获取和释放任意内存量。 或者，可以采用 COBOL 对程序进行编码，但是使用汇编语言、PL/I 或 C 子例程来获取和释放存储空间。

CICS
采用 CICS 支持的任何语言对程序进行编码。 如有必要，EXEC CICS 接口提供调用来管理内存。

相关概念
第 13 页的『面向对象的应用程序』
IBM MQ 提供了对 JMS， Java， C++ ， .NET 和 ActiveX 的支持。 这些语言和框架使用 IBM MQ 对象模型，它提供的类与 IBM MQ 调用和结构提供相同的功能。

技术概述

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

相关参考
应用程序开发参考

IBM MQ 数据定义文件

IBM MQ 提供数据定义文件来帮助编写应用程序。

数据定义文件也称为：

语言	数据定义
C	包含文件或头文件
Visual Basic	模块文件（仅 32 位版本）
COBOL	副本文件
汇编程序	宏
PL/I	包含文件

IBM MQ COPY、头、包含和模块文件中描述了用于帮助编写通道出口的数据定义文件。

第 853 页的『用户出口、API 出口和 IBM MQ 可安装服务』中描述了用于帮助编写可安装服务出口的数据定义文件。

有关在 C++ 上受支持的数据定义文件，请参阅使用 C++。

IBM i

有关在 RPG 上受支持的数据定义文件，请参阅 IBM i 应用程序编程参考 (ILE/RPG)。

数据定义文件的名称具有前缀 CMQ，以及由编程语言确定的后缀：

后缀	语言
a	汇编语言
b	Visual Basic
c	C
l	COBOL（不具有初始化值）
p	PL/I
v	COBOL（具有缺省值集）

安装库

名称 **thlqual** 是 z/OS 上的安装库的高级限定符。

本主题在以下标题下介绍 IBM MQ 数据定义文件：

- 第 671 页的『C 语言包含文件』
- 第 671 页的『Visual Basic 模块文件』
- 第 671 页的『COBOL 副本文件』
- z/OS 第 672 页的『System/390 汇编语言宏』
- z/OS 第 672 页的『PL/I 包含文件』

C 语言包含文件

IBM MQ C 包含文件列出在 C 头文件中。 它们安装在以下目录或库中：

平台	安装目录或库
IBM i IBM i	QMQM/H
UNIX UNIX	*MQ_INSTALLATION_PATH*/inc/
Windows Windows 系统	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS z/OS	**thlqual**.SCSQC370

其中，*MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。

注: 对于 UNIX，包含文件以符号形式链接到 `/usr/include` 中。

有关目录结构的更多信息，请参阅规划文件系统支持。

Visual Basic 模块文件

IBM MQ for Windows 提供四个 Visual Basic 模块文件。

这些文件列出在 Visual Basic 模块文件中并安装在

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

COBOL 副本文件

对于 COBOL，IBM MQ 提供包含命名常量的单独副本文件，并为各结构提供两个副本文件。

每个结构有两个副本文件，因为各副本文件均附带和不附带初始值：

- 在 COBOL 程序的 WORKING-STORAGE SECTION 中，使用用于将结构字段初始化为缺省值的文件。 这些结构在名称以字母 V（值）为后缀的副本文件中进行定义。
- 在 COBOL 程序的 LINKAGE SECTION 中，使用没有初始值的结构。 这些结构在名称以字母 L（链接）为后缀的副本文件中进行定义。

IBM i 对于使用 MQI 原型调用的 ILE COBOL 程序，提供了包含 IBM i 的数据和接口定义的副本文件。 这些文件存在于成员名称具有后缀 L（对于没有初始值的结构）或后缀 V（对于具有初始值的结构）的 QMQM/QCBLLESRC 中。

IBM MQ COBOL 副本文件列出在 COBOL COPY 文件中。 它们安装在以下目录中：

平台	安装目录或库
UNIX 其他 UNIX 平台	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook（对于 Micro Focus COBOL） *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol（对于 IBM VisualAge COBOL）
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

请在程序中仅包含需要的文件。 通过在 01 级声明后使用一个或多个 COPY 语句来实现此目的。 这意味着必要时可以在程序中包含多个版本的结构。 请注意，CMQV 是大文件。

以下是用于包含 CMQMDV 副本文件的 COBOL 代码示例：

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

各结构声明以 01 级项开头；可以通过对 01 级声明进行编码，后跟用于在结构声明的其余部分中进行复制的 COPY 语句来声明结构的若干实例。 要参考相应的实例，请使用 IN 关键字。

以下是用于包含两个 CMQMDV 实例的 COBOL 代码示例：

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

在 4 字节边界上将结构对齐。 如果使用 COPY 语句在不是 01 级项的项后包含结构，请确保该结构是 4 字节的倍数（从 01 级项的开始起算）。 如果不这样做，那么可能会降低应用程序的性能。

结构在 MQI 中使用的数据类型中进行了描述。 结构中字段的描述显示没有前缀的字段名称。 在 COBOL 程序中，使用结构名称后跟连字符作为字段名称的前缀，如 COBOL 声明中所示。 结构副本文件中的字段通过此方式添加前缀。

结构副本文件中的声明内的字段名称为大写形式。 可以改用混合大小写或小写。 例如，MQGMO 结构的字段 *StrucId* 在 COBOL 声明和副本文件中显示为 MQGMO-STRUCID。

V 后缀结构使用所有字段的初始值进行声明，因此需要仅设置所需值不同于初始值的字段。

System/390 汇编语言宏

z/OS

IBM MQ for z/OS 提供两个包含命名常量的汇编语言宏，以及一个用于生成各结构的宏。

这些宏列出在 z/OS 汇编程序 COPY 文件中并安装在 **thlqual**.SCSQMACS 中。

这些宏使用如下代码进行调用：

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

PL/I 包含文件

z/OS

IBM MQ for z/OS 提供包含文件，其中包含在采用 PL/I 编写 IBM MQ 应用程序时需要的所有定义。

这些文件列出在 PL/I 包含文件中并安装在 **thlqual**.SCSQPLIC 目录中：

如果要将 IBM MQ 存根链接到您的程序，请在程序中包含这些文件（请参阅第 934 页的『准备要运行的程序』）。 如果试图动态链接 IBM MQ 调用，请仅包含 CMQP（请参阅第 940 页的『动态调用 IBM MQ 存根』）。 只能对批处理和 IMS 程序执行动态链接。

编写用于排队的过程应用程序

使用此信息来了解有关编写排队应用程序、连接和断开队列管理器、发布/预订以及打开和关闭对象的信息。

使用以下链接了解有关编写应用程序的更多信息：

- 第 673 页的『消息队列接口概述』

- 第 685 页的『连接到队列管理器和从队列管理器断开连接』
- 第 692 页的『打开和关闭对象』
- 第 700 页的『将消息放置到队列上』
- 第 713 页的『从队列取出消息』
- 第 746 页的『编写发布/预订应用程序』
- 第 783 页的『查询和设置对象属性』
- 第 786 页的『落实和回退工作单元』
- 第 795 页的『使用触发器启动 IBM MQ 应用程序』
- 第 811 页的『与 MQI 和集群配合使用』
- z/OS 第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
- z/OS 第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』

相关概念

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

第 40 页的『IBM MQ 应用程序的设计注意事项』
当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

第 834 页的『编写客户机过程应用程序』
使用过程语言在 IBM MQ 上编写客户机应用程序时需要知道的内容。

第 908 页的『构建过程应用程序』
您可以使用若干过程语言之一编写 IBM MQ 应用程序，并在几个不同平台上运行该应用程序。

第 947 页的『处理过程程序错误』
本信息说明与应用程序 MQI 调用关联的错误，这些错误可能是应用程序进行调用时产生的，也可能是将其消息传递给最终目标时产生的。

相关任务

第 963 页的『使用 IBM MQ 样本过程程序』
这些样本程序采用过程语言编写，并演示了消息队列接口 (MQI) 的典型用法。 IBM MQ 在不同平台上编程。

消息队列接口概述

了解有关消息队列接口 (MQI) 组件的信息。

消息队列接口由以下各项组成：

- 调用，程序可以通过其访问队列管理器和它的设施
- 结构，供程序用于将数据传递到队列管理器和从中获取数据
- 基本数据类型，用于将数据传递到队列管理器和从中获取数据

z/OS IBM MQ for z/OS 还提供：

- 两个额外调用，z/OS 批处理程序可以通过其落实和回退更改。
- 数据定义文件（有时称为副本文件、宏、包含文件和头文件），用于定义 IBM MQ for z/OS 随附的常量的值。
- 存根程序，用于对应用程序进行链接编辑。
- 样本程序套件，用于演示如何在 z/OS 平台上使用 MQI。 有关这些样本的更多信息，请参阅第 1054 页的『针对 z/OS 使用样本程序』。

IBM i IBM MQ for IBM i 还提供:

- 数据定义文件（有时称为副本文件、宏、包含文件和头文件），用于定义 IBM MQ for IBM i 随附的常量的值。
- 三个存根程序，用于对 ILE C、ILE COBOL 和 ILE RPG 应用程序进行链接编辑。
- 样本程序套件，用于演示如何在 IBM i 平台上使用 MQI。

UNIX and Linux 系统上的 IBM MQ for Windows 和 IBM MQ 还提供:

- 调用， IBM MQ for Windows 和 IBM MQ on UNIX and Linux 系统程序可以通过这些调用落实和回退更改。
- 包含文件，用于定义这些平台上提供的常量的值。
- 库文件，用于链接到应用程序。
- 样本程序套件，用于演示如何在这些平台上使用 MQI。 有关这些样本的更多信息，请参阅第 964 页的『在 Multiplatforms 上使用样本程序』。
- 样本源和可执行文件代码，用于绑定到外部事务管理器。

使用以下链接了解有关 MQI 的更多信息:

- 第 675 页的『MQI 调用』
- 第 675 页的『同步点调用』
- 第 676 页的『数据转换、数据类型、数据定义和结构』
- 第 677 页的『IBM MQ 存根程序和库文件』
- 第 682 页的『所有调用的通用参数』
- 第 682 页的『指定缓冲区』
- z/OS 第 682 页的『z/OS 批处理注意事项』
- 第 683 页的『UNIX and Linux 信号处理』

相关概念

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

MQI 调用

使用此信息来了解有关消息队列接口 (MQI) 中的调用的信息。

MQI 中的调用可以按如下分组：

MQCONN、MQCONNX 和 MQDISC
这些调用用于将程序连接到（带有或不带选项）队列管理器或将程序与队列管理器断开连接。 如果为 z/OS 编写 CICS 程序，那么不需要使用这些调用。 但是，如果要将应用程序移植到其他平台，那么建议使用这些调用。

MQOPEN 和 MQCLOSE
这些调用用于打开和关闭对象（例如队列）。

MQPUT 和 MQPUT1
这些调用用于将消息放在队列上。

MQGET
此调用用于浏览队列上的消息，或者从队列中移除消息。

MQSUB 和 MQSUBRQ
这些调用用于注册主题预订以及请求与预订相匹配的发布。

MQINQ
此调用用于查询有关对象属性的信息。

MQSET
此调用用于设置队列的某些属性。 不能设置其他类型的对象的属性。

MQBEGIN、MQCMIT 和 MQBACK
当 IBM MQ 是工作单元的协调程序时使用这些调用。 MQBEGIN 启动工作单元。 MQCMIT 和 MQBACK 通过落实或回滚工作单元期间进行的更新来结束工作单元。 IBM i 落实控制器用于协调 IBM MQ for IBM i 上的全局工作单元。 使用本机启动落实控制、落实和回滚命令。

MQCRTMH、MQBUFMH、MQMHBUF、MQDLTMH
这些调用用于创建消息句柄，以将消息句柄转换为缓冲区或将缓冲区转换为消息句柄以及删除消息句柄。

MQSETMP、MQINQMP、MQDLTMP
这些调用用于设置消息句柄上的消息属性、查询消息属性以及从消息句柄中删除属性。

MQCB、MQCB_FUNCTION、MQCTL
这些调用用于注册或控制回调函数。

MQSTAT
此调用用于检索有关先前异步放置操作的状态信息。

请参阅调用描述以获取 MQI 调用的描述。

同步点调用

使用此信息来了解有关不同平台上的同步点调用的信息。

同步点调用按如下可用：

IBM MQ for z/OS 调用

z/OS

IBM MQ for z/OS 提供 MQCMIT 和 MQBACK 调用。

在 z/OS 批处理程序中使用这些调用，以告诉队列管理器自上个同步点以来的所有 MQGET 和 MQPUT 操作都将成为永久（已落实）或将回退。 要在其他环境中落实和回退更改，请执行以下操作：

CICS
使用诸如 EXEC CICS SYNCPOINT 和 EXEC CICS SYNCPOINT ROLLBACK 之类的命令。

IMS
将诸如 GU（获取唯一）之类的 IMS 同步点设施用于 IOPCB、CHKP（检查点）和 ROLB（回滚）调用。

RRS

根据情况使用 MQCMIT 和 MQBACK 或 SRRCMIT 和 SRRBACK。（请参阅第 789 页的『事务管理和可恢复的资源管理器服务』。）

注: SRRCMIT 和 SRRBACK 是本机 RRS 命令，它们并非 MQI 调用。

IBM i 调用

IBM i

IBM MQ for IBM i 提供 MQCMIT 和 MQBACK 命令。 您也可以使用 IBM i COMMIT 和 ROLLBACK 命令，或者任何其他用于启动 IBM i 落实控制设施（例如，EXEC CICS SYNCPOINT）的命令或调用。

Windows， UNIX and Linux 平台上的 IBM MQ 调用

ULW

以下产品提供 MQCMIT 和 MQBACK 调用：

- IBM MQ for Windows
- UNIX and Linux 系统上的 IBM MQ

在程序中使用同步点调用，以告诉队列管理器自上个同步点以来的所有 MQGET 和 MQPUT 操作都将成为永久（已落实）或将回退。 要在 CICS 环境中落实和回退更改，请使用诸如 EXEC CICS SYNCPOINT 和 EXEC CICS SYNCPOINT ROLLBACK 之类的命令。

数据转换、数据类型、数据定义和结构

使用消息队列接口时，使用此信息来了解有关数据转换、基本数据类型、IBM MQ 数据定义和结构的信息。

数据转换

MQXCNVC（转换字符）调用将消息字符数据从一个字符集转换为另一个字符集。（在 IBM MQ for z/OS 上除外）此调用仅从数据转换出口进行使用。

请参阅 MQXCNVC - 转换字符以了解用于 MQXCNVC 调用的语法，并参阅第 893 页的『编写数据转换出口』以获取有关编写和调用数据转换出口的指导。

基本数据类型

对于受支持的编程语言，MQI 提供基本数据类型或非结构化字段。

这些数据类型在基本数据类型中进行了完整描述。

IBM MQ 数据定义

z/OS IBM MQ for z/OS 以 COBOL 副本文件、汇编语言宏、单个 PL/I 包含文件、单个 C 语言包含文件以及 C++ 语言包含文件的形式提供数据定义。

IBM i IBM MQ for IBM i 以 COBOL 副本文件、RPG 副本文件、C 语言包含文件以及 C++ 语言包含文件的形式提供数据定义。

IBM MQ 随附的数据定义文件包含：

- 所有 IBM MQ 常量和返回码的定义
- IBM MQ 结构和数据类型的定义
- 用于初始化结构的常量定义
- 各调用的函数原型（仅限 PL/I 和 C 语言）

有关 IBM MQ 数据定义文件的完整描述，请参阅 第 670 页的『IBM MQ 数据定义文件』。

结构

在数据定义文件中针对受支持的各编程语言提供了与 第 675 页的『MQI 调用』 中所列的 MQI 调用结合使用的结构。 IBM i z/OS IBM MQ for z/OS 和 IBM MQ for IBM i 提供包含供您在完成这些结构的某些字段时使用的常量的文件。 有关这些结构的更多信息，请参阅 IBM MQ 数据定义。

请参阅结构数据类型摘要以获取结构的摘要。

IBM MQ 存根程序和库文件

此处列出了针对每个平台提供的存根程序和库文件。

有关在构建可执行应用程序时如何使用存根程序和库文件的更多信息，请参阅第 908 页的『构建过程应用程序』。 有关链接到 C++ 库文件的信息，请参阅使用 C++ *IBM MQ* 使用 *C++*。

AIX *IBM MQ for AIX* 库文件

在 IBM MQ for AIX 中，必须将程序链接到针对运行应用程序的环境提供的 MQI 库文件以及操作系统所提供的 MQI 库文件。

在非线程应用程序中，将链接到以下某个库：

表 *109:* 非线程 *AIX* 应用程序的库文件

库文件	**环境**
libmqm.a	针对 C 的服务器
libmqic.a 和 libmqm.a	针对 C 的客户机
libmqmzf.a	针对 C 的可安装服务出口
libmqmxa.a	服务器 XA 接口
libmqmxa64.a	服务器备用 XA 接口
libmqcxa.a	客户机 XA 接口
libmqcxa64.a	客户机备用 XA 接口
libmqmcbrt.o	针对 Micro Focus COBOL 支持的 IBM MQ 运行时库
libmqmcb.a	针对 COBOL 的服务器
libmqicb.a	针对 COBOL 的客户机
libimqc23ia.a	针对 C++ 的客户机
libimqs23ia.a	针对 C++ 的服务器

在线程应用程序中，将链接到以下某个库：

表 *110:* 线程 *AIX* 应用程序的库文件.

一个包含两列的表，列出库文件以及每个库文件的环境。

库文件	**环境**
libmqm_r.a	针对 C 的服务器
libmqic_r.a 和 libmqm_r.a	针对 C 的客户机
libmqmzf_r.a	针对 C 的可安装服务出口
libmqmxa_r.a	服务器 XA 接口
libmqmxa64_r.a	服务器备用 XA 接口
libmqcxa_r.a	客户机 XA 接口

表 *110:* 线程 *AIX* 应用程序的库文件. 一个包含两列的表，列出库文件以及每个库文件的环境。 *(继续)*	
库文件	**环境**
libmqcxa64_r.a	客户机备用 XA 接口
libimqc23ia_r.a	针对 C++ 的客户机
libimqs23ia_r.a	针对 C++ 的服务器

注: 不能链接到多个库。 即，不能同时链接到线程库和非线程库。

IBM i *IBM MQ for IBM i* 库文件

在 IBM MQ for IBM i 中，将程序链接到为运行应用程序的环境提供的 MQI 库文件以及操作系统所提供的 MQI 库文件。

对于非线程应用程序:

表 *111:* 非线程 *IBM i* 应用程序的库文件	
库文件	**环境**
LIBMQM	服务器和客户机服务程序
LIBMQIC	客户机服务程序
IMQB23I4	C++ 库服务程序
IMQS23I4	C++ 服务器服务程序
LIBMQMZF	针对 C 的可安装出口

在线程应用程序中:

表 *112:* 线程 *IBM i* 应用程序的库文件	
库文件	**环境**
LIBMQM_R	服务器和客户机服务程序
IMQB23I4_R	C++ 库服务程序
IMQS23I4_R	C++ 服务器服务程序
LIBMQMZF_R	针对 C 的可安装出口
LIBMQIC_R	客户机服务程序

在 IBM MQ for IBM i 上，您可以用 C++ 编写应用程序。要了解如何链接 C++ 应用程序以及有关使用 C++ 的各个方面的完整详细信息，请参阅 使用 C++。

Linux *IBM MQ for Linux* 库文件

在 IBM MQ for Linux 中，必须将程序链接到为运行应用程序的环境提供的 MQI 库文件以及操作系统提供的 MQI 库文件。

在非线程应用程序中，将链接到以下某个库:

表 *113:* 非线程 *Linux* 应用程序的库文件	
库文件	**环境**
libmqm.so	针对 C 的服务器

表 113: 非线程 *Linux* 应用程序的库文件 (继续)	
库文件	**环境**
libmqic.so 和 libmqm.so	针对 C 的客户机
libmqmzf.so	针对 C 的可安装服务出口
libmqmxa.so	服务器 XA 接口
libmqmxa64.so	服务器备用 XA 接口
libmqcxa.so	客户机 XA 接口
libmqcxa64.so	客户机备用 XA 接口
libimqc23gl.so	针对 C++ 的客户机
libimqs23gl.so	针对 C++ 的服务器

在线程应用程序中，将链接到以下某个库：

表 114: 线程 *Linux* 应用程序的库文件	
库文件	**环境**
libmqm_r.so	针对 C 的服务器
libmqic_r.so 和 libmqm_r.so	针对 C 的客户机
libmqmzf_r.so	针对 C 的可安装服务出口
libmqmxa_r.so	服务器 XA 接口
libmqmxa64_r.so	服务器备用 XA 接口
libmqcxa_r.so	客户机 XA 接口
libmqcxa64_r.so	客户机备用 XA 接口
libimqc23gl_r.so	针对 C++ 的客户机
libimqs23gl_r.so	针对 C++ 的服务器

注: 不能链接到多个库。 即，不能同时链接到线程库和非线程库。

Solaris *IBM MQ for Solaris* 库文件

在 IBM MQ for Solaris 中，必须将程序链接到针对运行应用程序的环境提供的 MQI 库文件以及操作系统所提供的 MQI 库文件。

表 115: *Solaris* 应用程序的库文件	
库文件	**环境**
libmqm.so	针对 C 的服务器和客户机
libmqmzse.so	针对 C
libmqic.so	针对 C 的客户机
libmqmcs.so	针对 C 的通用服务
libmqmzf.so	针对 C 的可安装服务出口
libmqmxa.so	服务器 XA 接口
libmqmxa64.so	服务器备用 XA 接口
libmqcxa.so	客户机 XA 接口

表 115: Solaris 应用程序的库文件 (继续)	
库文件	**环境**
libmqcxa64.so	客户机备用 XA 接口
libimqc23as.a	针对 C++ 的客户机
libimqs23as.a	针对 C++ 的服务器

Windows *IBM MQ for Windows* 库文件

在 IBM MQ for Windows 上，除了操作系统提供的文件外，还必须将程序链接到为用于运行应用程序的环境提供的 MQI 库文件。

表 116: Windows 应用程序的库文件	
库文件	**环境**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	针对 C 的服务器（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	针对 C 的客户机（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	针对 C 的服务器 XA 接口（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	针对 C 的客户机 XA 接口（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	针对 C 的客户机 MTS（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	针对 C 的服务器 TXSeries CICS 支持（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	针对 C 的客户机 TXSeries CICS 支持（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	针对 C 的可安装服务出口（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	针对 IBM COBOL 的服务器（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	针对 Micro Focus COBOL 的服务器（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	针对 IBM COBOL 的客户机（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	针对 Micro Focus COBOL 的客户机（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	针对 C++ 的服务器（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	针对 C++ 的客户机（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	针对 C++ 的库（32 位）
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	针对 C++ 的客户机 MTS（32 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	针对 C 的服务器（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	针对 C 的客户机（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	针对 C 的服务器 XA 接口（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	针对 C 的客户机 XA 接口（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	针对 C 的客户机 MTS（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	针对 IBM COBOL 的服务器（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	针对 Micro Focus COBOL 的服务器（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	针对 IBM COBOL 的客户机（64 位）

表 116: Windows 应用程序的库文件 (继续)	
库文件	**环境**
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	针对 Micro Focus COBOL 的客户机（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	针对 C++ 的服务器（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	针对 C++ 的客户机（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	针对 C++ 的库（64 位）
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	针对 C++ 的客户机 MTS（64 位）

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

使用 amqmdnet.dll 来编译 .NET 程序。 请参阅部分第 463 页的『开发 .NET 应用程序』中的第 508 页的『编译 IBM MQ .NET 程序』，以获取更多信息。

交付这些文件以与先前发行版兼容：

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS* 存根程序

可运行使用 IBM MQ for z/OS 编写的程序之前，必须对其执行链接编辑，使其指向适用于运行应用程序的环境随 IBM MQ for z/OS 提供的存根程序。

存根程序提供处理对 IBM MQ for z/OS 可处理的请求的调用的第一个阶段。

IBM MQ for z/OS 提供以下存根程序：

CSQBSTUB
z/OS 批处理程序的存根程序

CSQBRRSI
通过 MQI 使用 RRS 的 z/OS 批处理程序的存根程序

CSQBRSTB
直接使用 RRS 的 z/OS 批处理程序的存根程序

CSQCSTUB
CICS 程序的存根程序

CSQQSTUB
IMS 程序的存根程序

CSQXSTUB
分布式排队非 CICS 出口的存根程序

CSQASTUB
数据转换出口的存根程序

注意: 如果使用的存根程序不是为特定环境列出的存根程序，那么可能会产生不可预测的结果。

注: 如果使用 CSQBRSTB 存根程序，请从 SYS1.CSSLIB 中使用 ATRSCSS 执行链接编辑。 （SYS1.CSSLIB 也称为可调用服务库）。 有关 RRS 的更多信息，请参阅第 789 页的『事务管理和可恢复的资源管理器服务』。

或者，可从程序中动态调用存根。 此方法在第 940 页的『动态调用 IBM MQ 存根』中进行了描述。

在 IMS 中，可能还需要使用 IBM MQ 提供的特殊语言接口模块。

请勿在相同 IMS MPP 区域中运行使用 CSQBSTUB 和 CSQQSTUB 执行链接编辑的应用程序。 这会导致问题，例如，DFS3607I 或 CSQQ005E 消息。 地址空间中第一个 MQCONN 调用确定使用的接口，因此，CSQQSTUB 和 CSQBSTUB 事务必须在不同 IMS 消息区域中运行。

所有调用的通用参数

针对所有调用，具有两种类型的公用参数：句柄和返回码。

使用句柄

所有 MQI 调用使用一个或多个句柄。 这些句柄识别与调用对应的队列管理器、队列或其他对象、消息或预订。

要使程序与队列管理器通信，程序必须具有用于识别此队列管理器的唯一标识。 此标识称为连接句柄，有时称为 *Hconn*。 针对 CICS 程序，连接句柄始终为零。 针对程序的所有其他平台或样式，当程序连接到队列管理器时，会通过 MQCONN 或 MQCONNX 调用返回连接句柄。 程序使用其他调用时，会将连接句柄作为输入参数传递。

要使程序与 IBM MQ 对象配合使用，程序必须具有用于识别此对象的唯一标识。 此标识称为对象句柄，有时称为 *Hobj*。 程序打开对象以配合使用时，会通过 MQOPEN 调用返回句柄。 程序使用后续 MQPUT、MQGET、MQINQ、MQSET 或 MQCLOSE 调用时，会将对象句柄作为输入参数传递。

同样，MQSUB 调用返回预订句柄或 *Hsub*（用于识别后续 MQGET、MQCB 或 MQSUBRQ 调用中的预订），某些处理消息属性的调用使用消息句柄或 *Hmsg*。

了解返回码

每个调用都会将完成代码和原因码作为输出参数返回。 这些完成代码和原因码统称为返回码。

要说明调用是否成功，每个调用完成时都会返回完成代码。 通常，完成代码 MQCC_OK 指示成功，MQCC_FAILED 指示失败。 一些调用可返回中间状态 MQCC_WARNING 以指示部分成功。

每个调用还会返回原因码，说明调用失败或部分成功的原因。 有很多原因码，包括如下情况：队列将满，队列不允许执行获取操作以及没有为队列管理器定义特定队列。 程序可使用原因码来决定如何继续。 例如，可提示用户更改其输入数据，然后再次执行调用，或者可向用户返回错误消息。

当完成代码为 MQCC_OK 时，原因码始终为 MQRC_NONE。

将列出每个调用的完成代码和原因码以及此调用的描述。 请参阅调用描述，并从列表选择相应调用。

有关更多详细信息（包括更正操作建议），请参阅：

- z/OS IBM MQ for z/OS 消息，完成和原因码 for IBM MQ for z/OS
- 消息和原因码（对于所有其他 IBM MQ 平台）

指定缓冲区

队列管理器仅在必要时会引用缓冲区。 如果在调用上不需要缓冲区，或者缓冲区长度为零，那么可以对缓冲区使用空指针。

指定所需缓冲区大小时，始终使用 datalength。

使用缓冲区保存调用的输出（例如，保存 MQGET 调用的消息数据或 MQINQ 调用查询的属性的值）时，如果指定的缓冲区无效或在只读存储器中，那么队列管理器会尝试返回原因码。 但是，它可能不能始终返回原因码。

z/OS *z/OS* 批处理注意事项

用于调用 MQI 的 z/OS 批处理程序可以处于主管或问题状态。

但是，必须满足以下条件：

- 它们必须处于任务方式，而不是服务请求块 (SRB) 方式。
- 它们必须处于主地址空间控制 (ASC) 方式（而不是存取寄存器 ASC 方式）。
- 它们不能处于跨内存方式。 主地址空间编号 (ASN) 必须等于辅助 ASN 和主 ASN。

- 它们不能用作 MPF 出口程序。
- 不能持有 z/OS 锁定。
- FRR 堆栈上不能具有函数恢复例程 (FRR)。
- 针对 MQCONN 或 MQCONNX 调用，任何程序状态字 (PSW) 关键字会生效（前提是此关键字与使用 TCB 关键字中的存储器兼容），但使用 MQCONN 或 MQCONNX 返回的连接句柄的后续调用必须满足以下条件：
  - 必须具有针对 MQCONN 或 MQCONNX 调用使用的相同 PSW 关键字
  - 必须具有可在相同 PSW 关键字下访问（适当时可写入）的参数
  - 必须在相同任务 (TCB) 而不是任务的任何子任务下发出这些调用
- 这些调用可以为 24 位或 31 位寻址方式。 但是，如果 24 位寻址方式生效，那么参数地址必须解释为有效 31 位地址。

如果不满足以上任一条件，那么会执行程序检查。 在某些情况下，调用会失败，且会返回原因码。

Linux UNIX *UNIX and Linux* 注意事项

需要考虑的注释事项。

开发 UNIX and Linux 应用程序时注意以下几点：

Linux UNIX *UNIX and Linux* 系统中的派生系统调用

在 IBM MQ 应用程序中使用派生系统调用时，请留意这些注意事项。

如果应用程序想要使用 `fork`，那么该应用程序的父进程应在进行任何 IBM MQ 调用 (例如， `MQCONN`) 或使用 **`ImqQueueManager`** 创建 IBM MQ 对象之前调用 `fork` 。

如果应用程序希望在执行任何 IBM MQ 调用后创建子进程，那么应用程序代码必须将 `fork()` 与 `exec()` 配合使用，以确保子进程为新实例，而不是与父进程完全相同的副本。

如果应用程序未使用 `exec()`，那么在子进程中执行的 IBM MQ API 调用会返回 MQRC_ENVIRONMENT_ERROR。

Linux UNIX *UNIX and Linux* 信号处理

这不适用于 IBM MQ for z/OS 或 IBM MQ for Windows。

通常，UNIX、Linux 和 IBM i 系统已从非线程（进程）环境移至多线程环境。 在非线程环境中，某些功能仅可使用信号实现，而多数应用程序不需要关注信号和信号处理。 在多线程环境中，基于线程的原语支持用于使用信号在非线程环境中实现的一些函数。

在很多情况下，虽然多线程环境支持信号和信号处理，但并不是很适用，存在各种限制。 在各个中间件库（作为应用程序的一部分运行）尝试处理信号的多线程环境中，将应用程序代码与这些库集成时，可能会发生问题。 仅当一个进程中仅具有一个执行线程时，用于保存和复原信号处理程序（按进程定义）的传统方法适用，而在多线程环境中，则不适用。 这是因为很多执行线程可能会尝试保存和复原进程范围的资源，产生不可预测的结果。

UNIX 非线程应用程序

不适用于 Solaris，因为所有应用程序被视为线程应用程序，即使它们仅使用一个线程。

每个 MQI 函数针对信号设置其自己的信号处理程序：

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

将在 MQI 函数调用期间替换这些信号的用户处理程序。 其他信号可通过用户编写的处理程序按常规方式进行捕获。 如果未安装处理程序，那么会保留缺省操作（例如，忽略、核心转储或退出）。

IBM MQ 处理同步信号（SIGSEGV、SIGBUS、SIGFPE 和 SIGILL）后，会尝试将信号传递到任何注册的信号处理程序，再执行 MQI 函数调用。

Linux UNIX线程应用程序

线程被视为连接到 IBM MQ，从 MQCONN（或 MQCONNX）直到 MQDISC。

同步信号

同步信号在一个特定线程中出现。

UNIX and Linux 系统允许安全地为整个进程设置此类信号的信号处理程序。 但是，在任何线程连接到 IBM MQ 时，IBM MQ 在应用程序进程中为以下信号设置其自己的处理程序：

SIGBUS
SIGFPE
SIGSEGV
SIGILL

如果要编写多线程应用程序，那么对于每个信号仅具有一个进程范围信号处理程序。 IBM MQ 设置其自己的同步信号处理程序时，可保存每个信号的任何先前注册的处理程序。 IBM MQ 处理其中列出的一个信号后，IBM MQ 会尝试调用在进程中执行第一个 IBM MQ 连接时生效的信号处理程序。 所有应用程序线程从 IBM MQ 断开连接时，会复原先前注册的处理程序。

由于 IBM MQ 会保存和复原信号处理程序，因此如果可能还有相同进程的其他线程连接到 IBM MQ，那么应用程序线程不得为这些信号建立信号处理程序。

注: 当有线程连接到 IBM MQ 的情况下应用程序或中间件库（作为应用程序的一部分运行）建立信号处理程序时，在处理此信号期间，应用程序的信号处理程序必须调用相应 IBM MQ 处理程序。

通常，建立和复原信号处理程序时，将保存的最后一个信号处理程序必须第一个复原：

- 应用程序在连接到 IBM MQ 后建立信号处理程序时，在应用程序从 IBM MQ 断开连接之前，必须复原前一个信号处理程序。
- 应用程序连接到 IBM MQ 之前建立信号处理程序时，必须从 IBM MQ 断开连接，再复原其信号处理程序。

注: 如果未遵从将保存的最后一个信号处理程序必须第一个复原这一常规准则，那么会导致应用程序中意外信号处理，并且应用程序可能会丢失信号。

异步信号

IBM MQ 不会在线程应用程序中使用任何异步信号，除非其是客户机应用程序。

线程客户机应用程序的其他注意事项

IBM MQ 在对服务器执行 I/O 操作期间处理以下信号。 这些信号由通信堆栈定义。 当有线程连接到队列管理器时，应用程序不得为这些信号建立信号处理程序：

SIGPIPE（针对 TCP/IP）

UNIX在 *MQI* 中使用 *UNIX* 信号处理的其他注意事项

使用 UNIX 信号处理时，请留意这些注意事项。

快速路径（可信）应用程序

快速路径应用程序可在 IBM MQ 所在的相同进程中运行，因此在多线程环境中运行。

在此环境中，IBM MQ 处理同步信号 SIGSEGV、SIGBUS、SIGFPE 和 SIGILL。 不得向连接到 IBM MQ 的快速路径应用程序传递所有其他信号。 相反，这些信号必须由应用程序阻塞或处理。 如果快速路径应用程序拦截此类事件，那么必须停止并重新启动队列管理器，否则它会保持处于未定义状态。 有关 MQCONNX 下快速路径应用程序的完整限制列表，请参阅第 687 页的『使用 MQCONNX 调用连接到队列管理器』。

信号处理程序中的 MQI 函数调用

当您处于信号处理程序中时，请勿调用 MQI 函数。

如果尝试在一个 MQI 函数处于活动状态时从信号处理程序调用另一个 MQI 函数，那么会返回 MQRC_CALL_IN_PROGRESS。 如果尝试在没有任何其他 MQI 函数处于活动状态时从信号处理程序调用一个 MQI 函数，有时在操作期间可能会失败，因为存在操作系统限制，仅可从处理程序或在其中发出所选调用。

对于 C++ 析构函数方法（在程序退出期间可能会自动调用），可能无法停止调用 MQI 函数。 忽略有关 MQRC_CALL_IN_PROGRESS 的任何错误。 如果信号处理程序调用 exit()，那么 IBM MQ 通常会回退同步点中的未落实消息并关闭任何打开的队列。

MQI 调用期间的信号

MQI 函数不会向应用程序返回代码 EINTR 或任何等效代码。

如果 MQI 调用期间出现信号，且处理程序调用 *return*，那么会继续执行调用，好像未出现信号一样。 尤其是，MQGET 不能被信号中断，导致控制立即返回给应用程序。 如果要突破 MQGET 这一限制，请将队列设置为 GET_DISABLED；或者，使用有限到期时间对 MQGET 的调用使用循环（设置了 gmo.WaitInterval 的 MQGMO_WAIT），并使用信号处理程序（在非线程环境中）或线程环境中的等效函数来设置用于断开循环的标记。

AIX 在 AIX 环境中，IBM MQ 需要重新启动信号中断的系统调用。 与 sigaction(2) 建立您自己的信号处理程序时，在新操作结构的 sa_flags 字段中设置 SA_RESTART 标记，否则，IBM MQ 可能无法完成信号中断的任何调用。

用户出口和可安装服务

在多线程环境中作为 IBM MQ 进程一部分运行的用户出口和可安装服务具有与快速路径应用程序相同的限制。 要永久连接到 IBM MQ 以便不使用信号或非线程安全操作系统调用，请考虑这些注意事项。

连接到队列管理器和从队列管理器断开连接

要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

建立此连接的方式取决于运行程序所在的平台和环境：

Multi **IBM MQ for Multiplatforms**

在这些环境中运行的程序可使用 MQCONN MQI 调用连接到队列管理器，使用 MQDISC 调用从队列管理器断开连接。 或者，程序可使用 MQCONNX 调用。

z/OS **IBM MQ for z/OS 批处理**

在此环境中运行的程序可使用 MQCONN MQI 调用连接到队列管理器，使用 MQDISC 调用从队列管理器断开连接。 或者，程序可使用 MQCONNX 调用。

z/OS 批处理程序可连续或并行连接到相同 TCB 上的多个队列管理器。

z/OS **IMS**

IMS 控制区域在启动时连接到一个或多个队列管理器。 此连接通过 IMS 命令控制。 有关如何控制 z/OS 上的 IMS 适配器的信息，请参阅 管理 IBM MQ for z/OS。 但是，消息排队 IMS 程序的编写者必须使用 MQCONN MQI 调用指定其要连接的队列管理器。 它们可使用 MQDISC 调用从此队列管理器断开连接。

在用于建立同步点的 IMS 调用之后，处理另一个用户的消息之前，IMS 适配器确保应用程序关闭句柄并从队列管理器断开连接。 请参阅第 789 页的『IMS 应用程序中的同步点』。

IMS 程序可连续或并行连接到相同 TCB 上的多个队列管理器。

z/OS **CICS 事务服务器 z/OS**

CICS 程序不需要执行任何操作也可连接到队列管理器，因为 CICS 系统本身已连接。 通常，在初始化时会自动建立此连接，但也可使用 IBM MQ for z/OS 提供的 CKQC 事务。 有关 CKQC 的更多信息，请参阅 管理 IBM MQ for z/OS。

CICS 任务只能连接到 CICS 区域已连接到的队列管理器。

CICS 程序还可使用 MQI 连接和断开连接调用（MQCONN 和 MQDISC）。 您可能希望执行此操作以便通过最少的重新编码量将这些应用程序移植到非 CICS 环境。 但是，这些调用始终会在 CICS 环境中成功完成。 这表示返回码可能不会反映队列管理器的连接的真实状态。

Windows 和开放式系统的 TXSeries

这些程序不需要执行任何操作也可连接到队列管理器，因为 CICS 系统本身已连接。 因此，一次仅支持一个连接。 CICS 应用程序必须发出 MQCONN 调用以获取连接句柄，在退出之前必须发出 MQDISC 调用。

使用以下链接，以了解有关连接到队列管理器和从其中断开连接的更多信息：

- 第 686 页的『使用 MQCONN 调用连接到队列管理器』
- 第 687 页的『使用 MQCONNX 调用连接到队列管理器』
- 第 691 页的『使用 MQDISC 从队列管理器断开程序连接』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

使用 *MQCONN* 调用连接到队列管理器

使用此信息以了解如何使用 MQCONN 调用连接到队列管理器。

通常，可连接到特定队列管理器或缺省队列管理器：

- 针对 IBM MQ for z/OS，在批处理环境中，在 CSQBDEFV 模块中指定缺省队列管理器。
- 针对 IBM MQ for Windows、IBM i、UNIX 和 Linux 系统，在 mqs.ini 文件中指定缺省队列管理器。

或者，在 z/OS MVS 批处理、TSO 和 RRS 环境中，可连接到队列共享组中任何一个队列管理器。 MQCONN 或 MQCONNX 请求选择组的任何一个活动成员。

连接到队列管理器时，此调用对于任务必须是本地的。 它必须属于 IBM MQ 应用程序所在的相同系统。

在 IMS 环境中，必须将队列管理器连接到 IMS 控制区域以及程序使用的从属区域。 安装了 IBM MQ for z/OS 时，在 CSQQDEFV 模块中指定缺省队列管理器。

针对 TXSeries CICS 环境以及 TXSeries for Windows 和 AIX，必须将队列管理器定义为 CICS 的 XA 资源。

要连接到缺省队列管理器，请调用 MQCONN，同时指定全部为空白的名称或以空 (X'00') 字符开头的名称。

应用程序必须获得授权，才可成功连接到队列管理器。 有关更多信息，请参阅保护。

MQCONN 的输出为：

- 连接句柄 (**Hconn**)
- 完成代码
- 原因码

针对后续 MQI 调用使用连接句柄。

如果原因码指示应用程序已连接到此队列管理器，那么返回的连接句柄与首次连接应用程序时返回的连接句柄相同。 在此情况下，应用程序不能发出 MQDISC 调用，因为调用应用程序预期会保持为已连接。

连接句柄的作用域与对象句柄的作用域相同（请参阅第 693 页的『使用 MQOPEN 调用打开对象』）。

在 MQCONN 的 MQCONN 调用描述中提供了参数的描述。

如果在发出调用时队列管理器处于停顿状态，或者如果队列管理器将关闭，那么 MQCONN 调用会失败。

MQCONN 或 MQCONNX 的作用域

通常，MQCONN 或 MQCONNX 调用的作用域为发出此调用的线程。 即，从调用返回的连接句柄仅在发出此调用的线程中有效。 一次仅可使用句柄执行一个调用。 如果从其他线程使用调用，那么会因无效将其拒绝。 如果在应用程序中具有多个线程且每个线程都希望使用 IBM MQ 调用，那么每个线程都必须发出 MQCONN 或 MQCONNX。

一个进程执行多个 MQCONN 调用时，不需要对相同队列管理器执行每个调用。 但是，一次仅可从一个线程建立一个 IBM MQ 连接。 或者，考虑第 690 页的『使用 MQCONNX 的共享（独立于线程）连接』，以允许从单个线程使用多个 IBM MQ 连接，以及从任何线程使用 IBM MQ 连接。[7]

如果应用程序正作为客户机运行，那么可连接到一个线程中的多个队列管理器。

使用 *MQCONNX* 调用连接到队列管理器

MQCONNX 调用与 MQCONN 调用类似，但包含用于控制调用工作方式的选项。

您可以提供队列管理器名称 z/OS 或 z/OS 共享队列系统上队列共享组名作为 MQCONNX 的输入。

MQCONNX 的输出为：

- 连接句柄 (Hconn)
- 完成代码
- 原因码

针对后续 MQI 调用使用连接句柄。

在 MQCONNX 中提供了 MQCONNX 的所有参数的描述。 使用 *Options* 字段，可为任何版本的 MQCNO 设置 STANDARD_BINDING、FASTPATH_BINDING、SHARED_BINDING 或 ISOLATED_BINDING。 还可使用 MQCONNX 调用执行共享（独立于线程的）连接。 请参阅第 690 页的『使用 MQCONNX 的共享（独立于线程）连接』，以获取有关这些对象的更多信息。

MQCNO_STANDARD_BINDING

缺省情况下，MQCONNX（类似于 MQCONN）暗示两个逻辑线程，其中 IBM MQ 应用程序和本地队列管理器代理程序在不同进程中运行。 IBM MQ 应用程序请求 IBM MQ 操作，本地队列管理器代理程序为请求提供服务。 这由 MQCONNX 调用上的 MQCNO_STANDARD_BINDING 选项定义。

如果指定 MQCNO_STANDARD_BINDING，那么 MQCONNX 调用根据队列管理器的 **DefaultBindType** 属性值（在 qm.ini 中定义）使用 MQCNO_SHARED_BINDING 或 MQCNO_ISOLATED_BINDING。

这是缺省值。

[7] 将多线程应用程序与 UNIX and Linux 系统上的 IBM MQ 配合使用时，需要确保这些应用程序具有足够的线程堆栈大小。 当多线程应用程序通过自身或其他信号处理程序（例如 CICS）发出 MQI 调用时，请考虑使用 256 KB 或更大的堆栈大小。

如果链接到 mqm 库，那么会首先尝试使用缺省绑定类型的标准服务器连接。 如果底层服务器库未能装入，那么会改为尝试客户机连接。

- 如果指定了 MQ_CONNECT_TYPE 环境变量，那么可提供以下其中一个选项以更改 MQCONN 或 MQCONNX（如果指定了 MQCNO_STANDARD_BINDING）的行为。 （存在例外情况：如果指定了 MQCNO_FASTPATH_BINDING 且 MQ_CONNECT_TYPE 设置为 LOCAL 或 STANDARD 以允许管理员对快速路径连接降级，而不对应用程序进行相关更改：

值	含义
CLIENT	仅尝试执行客户机连接。
FASTPATH	此值在先前发行版中受支持，但现在即使指定此值也会将其忽略。
LOCAL	仅尝试执行服务器连接。 快速路径连接降级为标准服务器连接。
标准	支持兼容先前发行版。 此值现在被视为 LOCAL。

- 如果调用 MQCONN 时未设置 MQ_CONNECT_TYPE 环境变量，那么会尝试使用缺省绑定类型的标准服务器连接。 如果服务器库未能装入，那么会尝试客户机连接。

MQCNO_FASTPATH_BINDING

可信应用程序表示 IBM MQ 应用程序和本地队列管理器代理程序变为相同进程。 由于代理程序进程不再需要使用接口来访问队列管理器，因此这些应用程序变为队列管理器的扩展。 这由 MQCONNX 调用上的 MQCNO_FASTPATH_BINDING 选项定义。

您需要将受信任的应用程序链接到线程 IBM MQ 库。 有关如何将 IBM MQ 应用程序设置为作为可信应用程序运行的指示信息，请参阅 MQCNO 选项。

使用此选项，可实现最高性能。

注: 此选项会损害队列管理器的完整性：无法避免覆盖其存储器。 如果应用程序包含可向消息公开的错误以及队列管理器中的其他数据，这也适用。 使用此选项之前，请考虑这些问题。

MQCNO_SHARED_BINDING

指定此选项以使应用程序和本地队列管理器代理程序在不同进程中运行。 这会维护队列管理器的完整性，即，保护队列管理器免受错误程序的损害。 但是，应用程序和本地队列管理器代理程序共享一些资源。

此选项是 MQCNO_FASTPATH_BINDING 和 MQCNO_ISOLATED_BINDING 之间的中间选项，但都可保护队列管理器的完整性和 MQI 调用的性能。

如果队列管理器不支持此类型的绑定，那么会忽略 MQCNO_SHARED_BINDING。 将继续处理，好像未指定此选项一样。

如果应用程序已使用 MQCNO_SHARED_BINDING 连接到本地队列管理器，那么在应用程序正在运行时，可停止队列管理器。 如果在应用程序仍在运行时，重新启动队列管理器，那么尝试启动队列管理器会失败，且会返回错误 AMQ7018，因为应用程序仍在占用队列管理器所需的资源。

为了启动队列管理器，必须停止应用程序。

MQCNO_ISOLATED_BINDING

指定此选项以使应用程序和本地队列管理器代理程序在不同进程中运行，与 MQCNO_SHARED_BINDING 一样。 但是，在此情况下，应用程序进程和本地队列管理器代理程序会相互隔离，因为它们不共享资源。

这是用于保护队列管理器的完整性最为安全的选项，但所提供的 MQI 调用性能最低。

如果队列管理器不支持这种绑定，那么会忽略 MQCNO_ISOLATED_BINDING。 将继续处理，好像未指定此选项一样。

MQCNO_CLIENT_BINDING

指定此选项以使应用程序仅尝试执行客户机连接。 此选项具有以下局限性：

- z/OS 在 z/OS 上会忽略 MQCNO_CLIENT_BINDING。
- 如果使用非 MQCNO_STANDARD_BINDING 的任何 MQCNO 绑定选项指定 MQCNO_CLIENT_BINDING，那么会将其拒绝，并返回 MQRC_OPTIONS_ERROR。
- MQCNO_CLIENT_BINDING 不适用于 Java，因为它具有其自己的选择绑定类型的机制。
- 如果调用 MQCONN 时未设置 MQ_CONNECT_TYPE 环境变量，那么会尝试使用缺省绑定类型的标准服务器连接。 如果服务器库未能装入，那么会尝试客户机连接。

MQCNO_LOCAL_BINDING

指定此选项以使应用程序尝试执行服务器连接。 如果还指定了 MQCNO_FASTPATH_BINDING、MQCNO_ISOLATED_BINDING 或 MQCNO_SHARED_BINDING，那么连接会是此类型的连接，并在此部分中记录。 否则，会使用缺省绑定类型尝试执行标准服务器连接。 MQCNO_LOCAL_BINDING 具有以下局限性：

- z/OS 在 z/OS 上会忽略 MQCNO_CLIENT_BINDING。
- 如果使用非 MQCNO_STANDARD_BINDING 的任何 MQCNO 重新连接选项指定 MQCNO_LOCAL_BINDING，那么会将其拒绝，并返回 MQRC_OPTIONS_ERROR。
- MQCNO_LOCAL_BINDING 不适用于 Java，因为它具有其自己的选择绑定类型的机制。
- 如果调用 MQCONN 时未设置 MQ_CONNECT_TYPE 环境变量，那么会尝试使用缺省绑定类型的标准服务器连接。 如果服务器库未能装入，那么会尝试客户机连接。

z/OS 在 z/OS 上，允许使用这些选项，但仅会执行标准绑定连接。

z/OS MQCNO V 3 for z/OS 允许四个不同的选项：

MQCNO_SERIALIZE_CONN_TAG_QSG

这允许应用程序请求一次仅可在队列共享组中运行应用程序的一个实例。 这通过使用应用程序指定或衍生的值注册使用连接标记实现。 标记为 MQCNO V3 中指定的 128 字节字符串。

MQCNO_RESTRICT_CONN_TAG_QSG

应用程序包含可连接到队列管理器的多个进程（或 TCB）时，会使用此选项。 仅当当前未使用标记或请求的应用程序在相同处理作用域中时，才允许连接。 这是与标记所有者相同的队列共享组中的 MVS 地址空间。

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

这类似于 MQCNO_SERIALIZE_CONN_TAG_QSG，但仅会查询本地队列管理器以查看是否已在使用请求的标记。

MQCNO_RESTRICT_CONN_TAG_Q_MGR

这类似于 MQCNO_RESTRICT_CONN_TAG_QSG，但仅会查询本地队列管理器以查看是否已在使用请求的标记。

可信应用程序的限制

以下限制适用于可信应用程序：

- 必须将可信应用程序从队列管理器显式断开连接。
- 必须先停止可信应用程序，再使用 endmqm 命令结束队列管理器。
- 不得将异步信号和计时器中断（例如，`sigkill`）与 MQCNO_FASTPATH_BINDING 配合使用。
- 在所有平台上，当可信应用程序进程中的一个线程连接到队列管理器时，相同进程中另一个线程不能连接到队列管理器。

- Linux UNIX 在 UNIX and Linux 系统上，必须将 mqm 用作所有 MQI 调用的有效 userID 和 groupID。 您可以更改这些标识然后执行需要认证的非 MQI 调用（例如，打开文件），但在执行下一个 MQI 调用之前必须将其更改回 mqm。
- IBM i 在 IBM i 上：
  1. 可信应用程序必须在 QMQM 用户概要文件下运行。 用户概要文件是 QMQM 组的成员或程序采用 QMQM 权限都不足以运行可信应用程序。 无法使用 QMQM 用户概要文件登录到交互式作业，或无法在运行可信应用程序的作业的作业描述中指定 QMQM 用户概要文件。 在这种情况下，一种方法是使用 IBM i 概要文件交换 API 函数， QSYGETPH， QWTSETP 和 QSYRLSPH 在 MQ 程序运行时临时将作业的当前用户更改为 QMQM。 在《*IBM i 系统 API 参考*》的“安全性 API”部分中提供了这些函数的详细信息及其使用示例。
  2. 请勿通过使用系统请求选项 2 或通过使用 ENDJOB 结束运行可信应用程序的作业，来取消可信应用程序。
- Windows Linux UNIX 在 UNIX, Linux, and Windows 系统上，不支持可信 32 位应用程序。 如果尝试运行可信 32 位应用程序，那么会降级为标准绑定连接。

使用 *MQCONNX* 的共享（独立于线程）连接
使用此信息以了解使用 MQCONNX 的共享连接以及要考虑的一些使用说明。

注: IBM MQ for z/OS 上不支持。

在除 IBM MQ for z/OS 以外的 IBM MQ 平台上，使用 MQCONN 建立的连接仅可用于建立该连接的线程。 针对 MQCONNX 调用的选项允许创建可由进程中所有线程共享的连接。 如果运行应用程序的事务环境需要在相同线程上发出 MQI 调用，那么必须使用以下缺省选项：

MQCNO_HANDLE_SHARE_NONE
创建非共享连接。

在其他多数环境中，可使用以下其中一个独立于线程的共享连接选项：

MQCNO_HANDLE_SHARE_BLOCK
创建共享连接。 在 MQCNO_HANDLE_SHARE_BLOCK 连接上，如果连接当前由其他线程上的 MQI 调用使用，那么 MQI 调用会等待，直到完成当前 MQI 调用。

MQCNO_HANDLE_SHARE_NO_BLOCK
创建共享连接。 在 MQCNO_HANDLE_SHARE_NO_BLOCK 连接上，如果连接当前由另一个线程上的 MQI 调用使用，那么 MQI 调用会立即失败，且会返回原因 MQRC_CALL_IN_PROGRESS。

在非 MTS (Microsoft Transaction Server) 环境中，缺省值为 MQCNO_HANDLE_SHARE_NONE。 在 MTS 环境中，缺省值为 MQCNO_HANDLE_SHARE_BLOCK。

将从 MQCONNX 调用返回连接句柄。 此句柄可由来自进程内任何线程的后续 MQI 调用使用，方法是将这些调用与从 MQCONNX 返回的句柄关联。 使用单个共享句柄的 MQI 调用在线程中进行序列化。

例如，通过共享句柄，以下活动序列是可行的：

1. 线程 1 发出 MQCONNX 并获取共享句柄 *h1*
2. 线程 1 打开队列并使用 *h1* 发出 get 请求
3. 线程 2 使用 *h1* 发出 put 请求
4. 线程 3 使用 *h1* 发出 put 请求
5. 线程 2 使用 *h1* 发出 MQDISC

句柄由任何线程使用时，对连接的访问权不可用于其他线程。 如果可接受线程等待来自其他线程的任何先前调用完成，请使用带选项 MQCNO_HANDLE_SHARE_BLOCK 的 MQCONNX。

但是，阻塞会导致出现困难。 假定在步骤 第 690 页的『2』 中，线程 1 发出 get 请求，此请求等待可能尚未到达的消息（带 wait 的 get）。 在此情况下，线程 2 和 3 也会保持等待（已阻塞）线程 1 上 get 请求等待所花的时间。 如果偏向在其他 MQI 调用已在句柄上运行时 MQI 调用返回错误，请使用带 MQCNO_HANDLE_SHARE_NO_BLOCK 选项的 MQCONNX。

共享连接使用说明

1. 任何通过打开对象创建的对象句柄 (Hobj) 都与 Hconn 关联；因此针对共享 Hconn，任何线程还可使用 Hconn 共享和使用 Hobjs。 同样，任何在 Hconn 下启动的工作单元都与此 Hconn 关联；因此，此工作单元也可在具有共享 Hconn 的线程中共享。
2. 任何线程而不仅仅是调用相应 MQCONNX 的线程，都可调用 MQDISC 以与共享 Hconn 断开连接。 MQDISC 终止 Hconn，使其不可用于所有线程。
3. 单个线程可按顺序使用多个共享 Hconn，例如，使用 MQPUT 将一个消息放置在一个共享 Hconn 下，然后使用另一个共享 Hconn 放置另一个消息，每个操作都在不同本地工作单元下执行。
4. 无法在全局工作单元中使用共享 Hconn。

将 *MQCONNX* 调用选项与 *MQ_CONNECT_TYPE* 配合使用

使用此信息来了解不同的 MQCONNX 调用选项以及它们如何与 MQ_CONNECT_TYPE 环境变量配合使用。

注: MQ_CONNECT_TYPE 仅对 STANDARD 绑定有任何影响。 对于其他绑定，会忽略 MQ_CONNECT_TYPE。

在 UNIX and Linux 系统上的 IBM MQ for IBM i， IBM MQ for Windows 和 IBM MQ 上，可以将环境变量 MQ_CONNECT_TYPE 与 MQCONNX 调用上使用的 MQCNO 结构的 *Options* 字段中指定的绑定类型结合使用。

表 *117: MQCONNX* 调用选项如何与 ***MQ_CONNECT_TYPE*** 环境变量配合使用

MQCONNX 调用选项	**MQ_CONNECT_TYPE 环境变量**	**结果**
标准	未定义	标准
标准	标准	标准
标准	FASTPATH	标准
标准	CLIENT	CLIENT
标准	LOCAL	标准

如果未指定 MQCNO_STANDARD_BINDING，那么可使用 MQCNO_NONE（其缺省值为 MQCNO_STANDARD_BINDING）。

使用 *MQDISC* 从队列管理器断开程序连接

使用此信息以了解如何使用 MQDISC 将程序从队列管理器断开连接。

在已使用 MQCONN 或 MQCONNX 调用连接到队列管理器的程序已完成与队列管理器的所有交互时，会使用 MQDISC 调用断开此连接，除了以下情况：

- 在 CICS Transaction Server for z/OS 应用程序上（在其中调用是可选的，除非使用 MQCONNX 且您希望在应用程序结束之前删除连接标记）。
- 在 IBM MQ for IBM i 上（在其中从操作系统注销时，会执行隐式 MQDISC 调用）。

必须提供连接到队列管理器时 MQCONN 或 MQCONNX 返回的连接句柄 (Hconn)，作为 MQDISC 调用的输入。

除了在 z/OS 上的 CICS 上，在调用 MQDISC 之后，连接句柄 (Hconn) 不再有效，并且直到再次调用 MQCONN 或 MQCONNX 之后才能发出任何进一步的 MQI 调用。 对于仍使用此句柄打开的任何对象，MQDISC 会执行隐式 MQCLOSE。

z/OS 对于连接到 z/OS 的客户机，发出 MQDISC 调用时，会执行隐式落实，但不会关闭仍打开的任何队列句柄，直到通道实际结束。

如果使用 MQCONNX 在 IBM MQ for z/OS 上连接，那么 MQDISC 还会结束 MQCONNX 建立的连接标记的作用域。 但是，在 CICS、IMS 或 RRS 应用程序中，如果具有与连接标记关联的活动恢复单元，那么会拒绝 MQDISC，且会返回原因码 MQRC_CONN_TAG_NOT_RELEASE。

在 MQDISC 的 MQDISC 调用描述中提供了参数的描述。

未发出 MQDISC 时

创建线程终止时，会清除标准非共享连接 (Hconn)。 仅当整个进程终止时，才会隐式回退和断开共享连接。 如果仍存在 Hconn 时创建共享 Hconn 的线程终止，那么 Hconn 仍可使用。

权限检查

通常，MQCLOSE 和 MQDISC 调用不会执行权限检查。

在正常事件过程中，有权打开或连接到 IBM MQ 对象的作业可关闭此对象或从此对象断开连接。 即使连接到或打开 IBM MQ 对象的作业权限已撤销，也会接受 MQCLOSE 和 MQDISC 调用。

打开和关闭对象

使用此信息，可深入了解打开和关闭 IBM MQ 对象。

要执行以下任一操作，必须先打开相关 IBM MQ 对象：

- 将消息放置到队列上
- 从队列获取（浏览或检索）消息
- 设置对象的属性
- 查询任何对象的属性

使用 MQOPEN 调用打开对象，同时使用此调用的选项指定要对此对象执行的操作。 唯一的例外是，希望在队列上放置单条消息，然后立即关闭此队列。 在此情况下，通过使用 MQPUT1 调用绕过打开阶段（请参阅第 708 页的『使用 MQPUT1 调用将一条消息放置到队列上』）。

使用 MQOPEN 调用打开对象之前，必须将程序连接到队列管理器。 这在第 685 页的『连接到队列管理器和从队列管理器断开连接』中针对所有环境进行了详细说明。

可以打开四种类型的 IBM MQ 对象：

- 队列
- 名称列表
- 进程定义
- 队列管理器

通过使用 MQOPEN 调用所采用的相同方式打开所有这些对象。 有关 IBM MQ 对象的更多信息，请参阅对象类型。

您可以多次打开相同对象，每次会获得一个新的对象句柄。 您可能希望使用一个句柄浏览队列上的消息，使用另一个句柄从相同队列移除消息。 这可节省关闭和重新打开相同对象时使用的资源，防止耗尽资源。 还可打开队列以同时浏览和移除消息。

此外，可以使用单个 MQOPEN 打开多个对象，使用 MQCLOSE 关闭多个对象。 请参阅第 709 页的『分发列表』，以获取有关如何完成该操作的信息。

尝试打开对象时，队列管理器会检查针对在 MQOPEN 调用中指定的选项，您是否有权打开此对象。

程序从队列管理器断开连接时，会自动关闭对象。 在 IMS 环境中，在执行 GU（获取唯一项）IMS 调用后，程序开始针对新用户执行处理时，会强制断开连接。 在 IBM i 平台上，作业结束时会自动关闭对象。

关闭打开的对象是很好的编程实践。 使用 MQCLOSE 调用执行此操作。

使用以下链接了解有关打开和关闭对象的更多信息：

- 第 693 页的『使用 MQOPEN 调用打开对象』
- 第 699 页的『创建动态队列』
- 第 699 页的『打开远程队列』
- 第 700 页的『使用 MQCLOSE 调用关闭对象』

相关概念

第 673 页的『消息队列接口概述』

了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

使用 *MQOPEN* 调用打开对象

使用此信息以了解如何使用 MQOPEN 调用打开对象。

作为 MQOPEN 调用的输入，必须提供：

- 连接句柄。 对于 z/OS 上的 CICS 应用程序，可以指定常量 MQHC_DEF_HCONN (值为零)，或者使用 MQCONN 或 MQCONNX 调用返回的连接句柄。 对于其他程序，始终使用 MQCONN 或 MQCONNX 调用返回的连接句柄。
- 要打开的使用对象描述符结构 (MQOD) 的对象的描述。
- 用于控制调用操作的一个或多个选项。

MQOPEN 的输出为：

- 表示您对此对象的访问权的对象句柄。 在任何后续 MQI 调用的输入中使用此对象句柄。
- 修改的对象描述符结构（如果要创建在平台上支持的动态队列）。
- 完成代码。
- 原因码。

对象句柄的作用域

对象句柄 (Hobj) 的作用域与连接句柄 (Hconn) 的作用域相同。

这在第 687 页的『MQCONN 或 MQCONNX 的作用域』和第 690 页的『使用 MQCONNX 的共享（独立于线程）连接』中进行了描述。 但是，在一些环境中，存在其他注意事项：

CICS
在 CICS 程序中，只能在从其执行 MQOPEN 调用的相同 CICS 任务中使用此句柄。

IMS 和 z/OS 批处理
在 IMS 和批处理环境中，可在相同任务而不是任何子任务中使用句柄。

在 MQOPEN 中提供了 MQOPEN 调用的参数的描述。

以下部分描述了必须作为 MQOPEN 的输入提供的信息。

标识对象（MQOD 结构）

使用 MQOD 结构标识要打开的对象。 此结构是 MQOPEN 调用的输入参数。 （使用 MQOPEN 调用创建动态队列时，队列管理器会修改此结构。）

有关 MQOD 结构的完整详细信息，请参阅 MQOD

有关对分发列表使用 MQOD 结构的信息，请参阅第 709 页的『分发列表』下的第 710 页的『使用 MQOD 结构』。

名称解析
MQOPEN 调用如何解析队列及队列管理器名称。

注: 队列管理器别名是不具有 RNAME 字段的远程队列定义。

打开 IBM MQ 队列时，MQOPEN 调用会针对指定的队列名称执行名称解析函数。 这确定队列管理器对其执行后续操作的队列。 这表示，在对象描述符 (MQOD) 中指定别名队列或远程队列的名称时，调用会将名称解析为本地队列或传输队列。 如果针对任何类型的输入、浏览或设置打开队列，那么名称会解析为本地队列（如果存在一个本地队列），如果不存在一个本地队列，那么名称解析会失败。 仅当针对仅输出、仅查询或仅输出和查询打开队列时，名称会解析为非本地队列。 请参阅第 694 页的表 118，以获取名称解析过程的概述。 在 *ObjectQMgrName* 中提供的名称会在 *ObjectName* 中提供的名称之前解析。

第 694 页的表 118 还说明可如何使用远程队列的本地定义来定义队列管理器的名称的别名。 这允许您在将消息放置到远程队列上时选择使用的传输队列，从而可以实现多个目的，例如，针对目标为很多远程队列管理器的消息使用单个传输队列。

要使用下表，请先往下读取 **MQOD 的输入**标题下左侧两个列，并选择相应案例。 然后读取相应行，同时遵循任何指示信息。 遵循**解析名称**列中的指示信息，可返回到 **MQOD 的输入**列并根据指示插入值，或者可退出提供了结果的表。 例如，可能需要您输入 *ObjectName*。

表 *118:* 使用 *MQOPEN* 时解析队列名称

MQOD 的输入	**MQOD 的输入**	**解析名称**	**解析名称**	**解析名称**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
空或本地队列管理器	不具有 CLUSTER 属性的本地队列	本地队列管理器	输入 *ObjectName*	不适用（使用了本地队列）
空队列管理器	具有 CLUSTER 属性的本地队列	选择了工作负载管理的集群队列管理器或执行 PUT 时选择的特定集群队列管理器	输入 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE 和使用的本地队列 SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）
本地队列管理器	具有 CLUSTER 属性的本地队列	本地队列管理器	输入 *ObjectName*	不适用（使用了本地队列）
空或本地队列管理器	模型队列	本地队列管理器	生成的名称	不适用（使用了本地队列）

表 *118:* 使用 *MQOPEN* 时解析队列名称 *(*继续*)*

MQOD 的输入	**MQOD 的输入**	**解析名称**	**解析名称**	**解析名称**
空或本地队列管理器	具有/不具有 CLUSTER 属性的别名队列	再次执行名称解析，同时在别名队列定义对象中不更改 *ObjectQMgrName*，输入 *ObjectName*（设置为 *BaseQName*）。 指定了 *ObjectQMgrName* 时，不得解析为本地定义的别名，*ObjectQMgrName* 为空时，可解析为集群别名（在其他队列管理器上托管）。		
本地队列管理器	具有 CLUSTER 属性的别名队列	别名不得解析为非本地定义的集群队列或具有与别名相同的 *ObjectName* 的集群队列。		
空队列管理器	具有 CLUSTER 属性的别名队列	别名可解析为 *ObjectName* 与别名相同的集群队列。		
空或本地队列管理器	远程队列的本地定义	再次执行名称解析，同时 *ObjectQMgrName* 设置为 *RemoteQMgrName*，*ObjectName* 设置为 *RemoteQName*。 不得解析远程队列		如果非空，那么为 *XmitQName* 属性的名称；否则，为远程队列定义对象中的 *RemoteQMgrName*。 SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）
空队列管理器	无匹配的本地对象；找到集群队列	选择了工作负载管理的集群队列管理器或执行 PUT 时选择的特定集群队列管理器	输入 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）
空或本地队列管理器	无匹配的本地对象；找不到集群队列		错误，找不到队列	不适用
本地队列管理器所在的队列共享组中队列管理器的名称	本地共享队列	本地队列管理器	输入 *ObjectName*	不适用
本地传输队列的名称	（未解析）	输入 *ObjectQMgrName*	输入 *ObjectName*	输入 *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）

表 118: 使用 *MQOPEN* 时解析队列名称 (继续)				
MQOD 的输入	**MQOD 的输入**	**解析名称**	**解析名称**	**解析名称**
队列管理器别名定义（*RemoteQMgrName* 可为本地队列管理器）	（未解析，远程队列）	再次执行名称解析，同时 *ObjectQMgrName* 设置为 *RemoteQMgrName*。 不得解析为远程队列	输入 *ObjectName*	如果非空，那么为 *XmitQName* 属性的名称；否则，为远程队列定义对象中的 *RemoteQMgrName*。 SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）
队列管理器不是任何本地对象的名称；找到集群队列管理器或队列管理器别名	（未解析）	*ObjectQMgrName* 或执行 PUT 时选择的特定集群队列管理器	输入 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）
队列管理器不是任何本地对象的名称；找不到任何集群对象	（未解析）	输入 *ObjectQMgrName*	输入 *ObjectName*	支持 *DefXmitQName* 的队列管理器的 *DefXmitQName* 属性。 SYSTEM.QSG.TRANSMIT.QUEUE（请参阅注释）

注意:

1. *BaseQName* 是别名队列定义中的基本队列的名称。
2. *RemoteQName* 是远程队列的本地定义中的远程队列的名称。
3. *RemoteQMgrName* 是远程队列的本地定义中的远程队列管理器的名称。
4. *XmitQName* 是来自远程队列的本地定义的传输队列的名称。
5. 使用作为队列共享组 (QSG) 一部分的 IBM MQ for z/OS 队列管理器时，可以使用队列共享组的名称而不是第 694 页的表 118 中的本地队列管理器名称。

   如果本地队列管理器无法打开目标队列或无法将消息放置到此队列，那么会通过组内排队或 IBM MQ 通道，将消息传输到指定 ObjectQMgrName。
6. 在表的 *ObjectName* 列中， CLUSTER 指的是队列的 CLUSTER 和 CLUSNL 属性。
7. 如果本地队列管理器和远程队列管理器处于相同队列共享组中且启用了组内排队，那么使用 SYSTEM.QSG.TRANSMIT.QUEUE。
8. 如果为每个集群发送方通道指定了不同集群传输队列，那么 SYSTEM.CLUSTER.TRANSMIT.QUEUE 可能不是集群传输队列的名称。 有关多个集群传输队列的更多信息，请参阅集群：计划如何配置集群传输队列。
9. 在队列管理器不是任何本地对象的名称；找到集群队列管理器或队列管理器别名的情况下。

   如果使用 **ObjectQMgrName** 提供了队列管理器名称，且多个集群通道的不同集群名称被将到达此目标的本地队列管理器所知，那么可以使用这些通道中任何一个通道来移动消息，不管目标队列的集群名称是什么。

   如果希望此队列的消息仅通过集群名称与队列相同的通道发送，那么这可能是意外情况。

   但是，在此情况下，**ObjectQMgrName** 优先，集群工作负载均衡会考虑可能到达此队列管理器的所有通道，与其所处的集群的名称无关。

打开别名队列也会打开别名解析为的基本队列，打开远程队列也会打开传输队列。 因此，在打开一个队列时，不能删除指定的其他队列或解析为的其他队列。

别名队列无法解析为其他本地定义的别名队列（不管是否在集群中共享）时，允许解析为远程定义的集群别名队列，因此可指定为基本队列。

解析的队列名称和解析的队列管理器名称存储在 MQOD 的 *ResolvedQName* 和 *ResolvedQMgrName* 字段中。

有关分布式排队环境中名称解析的更多信息，请参阅什么是队列名称解析？。

使用 *MQOPEN* 调用选项

在 MQOPEN 调用的 **Options** 参数中，必须选择一个或多个选项来控制为将打开的对象提供的访问权。 使用这些选项，可执行以下操作：

- 打开队列，并指定放置到此队列的所有消息必须指向其相同实例
- 打开队列以允许在其上放置消息
- 打开队列以允许在其上浏览消息
- 打开队列以允许从其移除消息
- 打开对象以允许查询和设置其属性（但仅可设置队列的属性）
- 打开主题或主题字符串以向其发布消息
- 将上下文信息与消息关联
- 指定将用于安全检查的备用用户标识
- 如果队列管理器处于停顿状态，可控制调用

集群队列的 *MQOPEN* 选项
用于队列句柄的绑定取自 **DefBind** 队列属性（可采用值 MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED 或 MQBND_BIND_ON_GROUP）。

要将使用 MQPUT 放置到队列上的所有消息通过相同路径发送到相同队列管理器，请针对 MQOPEN 调用使用 MQOO_BIND_ON_OPEN 选项。

要指定在 MQPUT 时选择目标（即，基于逐条消息），请针对 MQOPEN 调用使用 MQOO_BIND_NOT_FIXED 选项。

要指定将消息组中使用 MQPUT 放置到队列的所有消息分配到相同目标实例，请针对 MQOPEN 调用使用 MQOO_BIND_ON_GROUP 选项。

将 消息组 与集群配合使用时，必须指定 MQOO_BIND_ON_OPEN 或 MQOO_BIND_ON_GROUP，以确保在同一目标处处理组中的所有消息。

如果未将这些选项中任一选项指定为缺省值，那么会使用 MQOO_BIND_AS_Q_DEF。

如果在 MQOD 中指定队列管理器的名称，那么会选择此队列管理器处的队列。 如果队列管理器名称为空，那么可选择任何实例。 请参阅第 812 页的『MQOPEN 和集群』，了解更多信息。

如果使用 QALIAS 定义打开集群队列，那么一些队列属性通过别名队列而不是基本队列定义。 集群属性属于别名队列覆盖的基本队列定义的属性。 例如，在以下片段中，集群队列使用 MQOO_BIND_NOT_FIXED 而不是 MQOO_BIND_ON_OPEN 打开。 集群队列定义在集群中公开时，别名队列定义对于队列管理器是本地的。

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

用于放入消息的 *MQOPEN* 选项
要打开队列或主题以将消息放入其中，请使用 MQOO_OUTPUT 选项。

用于浏览消息的 *MQOPEN* 选项
要打开队列以便您可以浏览其中的消息，请使用带有 MQOO_BROWSE 选项的 MQOPEN 调用。

这会创建浏览光标，队列管理器用于识别队列上下一条消息。 有关更多信息，请参阅第 741 页的『浏览队列中的消息』。

注：

1. 无法浏览远程队列上的消息；请勿使用 MQOO_BROWSE 选项打开远程队列。
2. 打开分发列表时，无法指定此选项。 有关分发列表的更多信息，请参阅第 709 页的『分发列表』。
3. 如果要使用协作浏览，请将 MQOO_CO_OP 与 MQOO_BROWSE 配合使用；请参阅选项

用于移除消息的 *MQOPEN* 选项
提供了三个选项来控制打开队列以从其移除消息这一操作。

在任何 MQOPEN 调用中仅可使用其中一个选项。 这些选项定义程序对队列具有专用访问权还是共享访问权。 专用访问权表示在关闭队列之前， 仅可从其移除消息。 如果另一个程序尝试打开队列以移除消息，那么其 MQOPEN 调用会失败。 共享访问权表示多个程序可移除 队列中的消息。

最明智的方法是接受定义队列时预期用于队列的访问权类型。 队列定义涉及 设置 **Shareability** 和 **DefInputOpenOption** 属性。 要接受此访问权，请使用 MQOO_INPUT_AS_Q_DEF 选项。 请参阅第 698 页的表 119，以了解这些属性的设置如何影响将在使用此选项时指定的访问权的类型。

表 *119: MQOPEN* 调用的队列属性和选项如何影响对队列的访问权

队列属性		**具有 MQOPEN 选项的访问权的类型**		
可共享性	**DefInputOpenOption**	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
SHAREABLE	SHARED	共享	共享	专用
SHAREABLE	EXCLUSIVE	专用	共享	专用
NOT_SHAREABLE*	SHARED*	专用	专用	专用
NOT_SHAREABLE	EXCLUSIVE	专用	专用	专用

注: * 尽管可定义队列具有此属性组合，缺省输入打开选项也会由 shareability 属性覆盖。

或者：

- 如果您知道即使其他程序可以同时从队列中移除消息，您的应用程序也可以成功运行，请使用 MQOO_INPUT_SHARED 选项。第 698 页的表 119 显示在某些情况中如何为您分配对队列的独占访问权（即使使用此选项）。
- 如果知道仅当阻止其他程序同时从队列移除消息时应用程序才会成功运行，请使用 MQOO_INPUT_EXCLUSIVE 选项。

注:

1. 无法从远程队列移除消息。 因此，无法使用任何 MQOO_INPUT_* 选项打开远程队列。
2. 打开分发列表时，无法指定此选项。 有关更多信息，请参阅第 709 页的『分发列表』。

用于设置和查询属性的 *MQOPEN* 选项
要打开队列以便您可以设置其属性，请使用 MQOO_SET 选项。

无法设置任何其他类型的对象的属性（请参阅第 783 页的『查询和设置对象属性』）。

要打开对象以便可查询其属性，请使用 MQOO_INQUIRE 选项。

注: 打开分发列表时，无法指定此选项。

用于关联消息上下文的 *MQOPEN* 选项
如果要可以在将消息放置到队列上时将上下文信息与消息关联，必须在打开队列时使用其中一个消息上下文选项。

使用这些选项，可区分与发出消息的用户相关的上下文信息以及与发出消息的应用程序相关的上下文信息。此外，在将消息放置到队列上时，可选择设置上下文信息，或者可选择自动从其他队列句柄获取上下文。

相关概念
第 39 页的『消息上下文』
消息上下文信息使得检索消息的应用程序能够了解有关消息发起方的信息。

第 706 页的『控制消息上下文信息』

使用 MQPUT 或 MQPUT1 调用将消息放入队列时，您可以指定队列管理器向消息描述符中添加某些缺省上下文信息。 拥有相应权限级别的应用程序可以添加额外的上下文信息。 您可以在 MQPMO 结构中使用 options 字段来控制上下文信息。

备用用户权限的 *MQOPEN* 选项
尝试使用 MQOPEN 调用打开对象时，队列管理器会检查您是否有权打开此对象。 如果您未获得授权，那么该调用将失败。

但是，服务器程序可能希望队列管理器检查正为其工作的用户的权限，而不是服务器自己的权限。 要执行此操作，必须使用 MQOPEN 调用的 MQOO_ALTERNATE_USER_AUTHORITY 选项，并在 MQOD 结构的 *AlternateUserId* 字段中指定备用用户标识。 通常，服务器将从其正处理的消息的上下文信息中获取用户标识。

z/OS 用于停顿队列管理器的 *MQOPEN* 选项
如果在队列管理器处于停顿状态时使用 MQOPEN 调用，那么根据您所使用的环境，调用可能会失败。

在 z/OS 上的 CICS 环境中，如果在队列管理器处于停顿状态时使用 MQOPEN 调用，那么调用始终失败。

在其他 z/OS 环境、IBM i、Windows 系统和 UNIX and Linux 系统环境中，仅当使用 MQOPEN 调用的 MQOO_FAIL_IF_QUIESCING 选项且队列管理器停顿时，调用会失败。

用于解析本地队列名称的 *MQOPEN* 选项
打开本地别名或模型队列时，将返回本地队列。

但是，打开远程队列或集群队列时，会使用远程队列定义中发现的远程队列名称和远程队列管理器名称或者所选远程集群队列名称填充 MQOD 结构的 *ResolvedQName* 和 *ResolvedQMgrName* 字段。

通过 MQOPEN 调用的 MQOO_RESOLVE_LOCAL_Q 选项，使用打开的本地队列的名称填充 MQOD 结构中的 *ResolvedQName*。 将通过相似方式使用托管本地队列的本地队列管理器的名称填充 *ResolvedQMgrName*。 此字段仅适用于 MQOD V3 结构；如果结构低于 V3，那么会忽略 MQOO_RESOLVE_LOCAL_Q，且不会返回错误。

如果在打开对象（例如，远程队列）时指定 MQOO_RESOLVE_LOCAL_Q，那么 *ResolvedQName* 是消息将放置到的传输队列的名称。 *ResolvedQMgrName* 是托管传输队列的本地队列管理器的名称。

创建动态队列
应用程序结束后不需要队列时，使用动态队列。

例如，可以针对应答队列使用动态队列。 将消息放置到队列上时，在 MQMD 结构的 *ReplyToQ* 字段中指定应答队列的名称（请参阅第 702 页的『使用 MQMD 结构定义消息』）。

要创建动态队列，将称为模型队列的模板与 MQOPEN 调用配合使用。 使用 IBM MQ 命令或操作以及控制面板创建模型队列。 创建的动态队列采用模型队列的属性。

调用 MQOPEN 时，在 MQOD 结构的 *ObjectName* 字段中指定模型队列的名称。 调用完成时，*ObjectName* 字段设置为创建的动态队列的名称。 此外，*ObjectQMgrName* 字段设置为本地队列管理器的名称。

可以通过三种方式指定创建的动态队列的名称：

- 在 MQOD 结构的 *DynamicQName* 字段中提供所需的全名。
- 为名称指定前缀（少于 33 个字符），并允许队列管理器生成名称剩余部分。 这表示队列管理器生成唯一名称，但您仍具有一些控制权（例如，可能希望每个用户使用特定前缀，或可能希望通过其名称中的特定前缀，为队列提供特殊安全分类）。 要使用此方式，请为 *DynamicQName* 字段的最后一个非空字符指定星号 (*)。 不要为动态队列名称指定单个星号 (*)。
- 允许队列管理器生成全名。 要使用此方式，请在 *DynamicQName* 字段的第一个字符位置指定星号 (*)。

有关这些方法的更多信息，请参阅 DynamicQName 字段的描述。

在动态队列和模型队列中提供了有关动态队列的更多信息。

打开远程队列
远程队列是非应用程序连接到的队列管理器所拥有的队列。

要打开远程队列，请使用用于本地队列的 MQOPEN 调用。 您可以按如下所示指定队列的名称：

1. 在 MQOD 结构的 *ObjectName* 字段中，指定本地队列管理器已知的远程队列的名称。

   注： 在此情况下，将 *ObjectQMgrName* 字段留空。
2. 在 MQOD 结构的 *ObjectName* 字段中，指定远程队列管理器已知的远程队列的名称。 在 *ObjectQMgrName* 字段中，指定：
   - 名称与远程队列管理器相同的传输队列的名称。 名称和大小写（大写、小写或混合大小写）必须精确匹配。
   - 解析为目标队列管理器或传输队列的队列管理器别名对象的名称。

   这将告知队列管理器消息的目标以及为到达此目标而需要将其放置到的传输队列。
3. 如果支持 *DefXmitQname*，请在 MQOD 结构的 *ObjectName* 字段中，指定远程队列管理器已知的远程队列的名称。

   注： 将 *ObjectQMgrName* 字段设置为远程队列管理器的名称（在此情况下，不能将其留空）。

仅在调用 MQOPEN 时会验证本地名称；最后一个检查是检查是否存在将使用的传输队列。

第 694 页的表 118 中概述了这些方法。

使用 *MQCLOSE* 调用关闭对象

要关闭对象，请使用 MQCLOSE 调用。

如果对象是队列，请注意以下情况：

- 关闭临时动态队列之前，不需要将其清空。

  关闭临时动态队列时，会删除队列以及其中可能仍具有的任何消息。 即使针对队列存在有待完成的未落实 MQGET、MQPUT 或 MQPUT1 调用，这也适用。
- 在 IBM MQ for z/OS 上，如果针对此队列存在有待完成的带 MQGMO_SET_SIGNAL 选项的任何 MQGET 请求，那么会将其取消。
- 如果使用 MQOO_BROWSE 选项打开了队列，那么浏览光标会损坏。

关闭操作与同步点无关，因此您可以在同步点之前或之后关闭队列。

作为 MQCLOSE 调用的输入，必须提供：

- 连接句柄。 使用用于打开它的同一连接句柄，或者对于 z/OS 上的 CICS 应用程序，可以指定常量 MQHC_DEF_HCONN (值为零)。
- 要关闭的对象的句柄。 从 MQOPEN 调用的输出获取此句柄。
- *Options* 字段中的 MQCO_NONE（除非将关闭永久动态队列）。
- 控制选项，用于确定队列管理器是否应该删除队列，即使队列上仍有消息（关闭永久动态队列时）。

MQCLOSE 的输出为：

- 完成代码
- 原因码
- 对象句柄，重置为值 MQHO_UNUSABLE_HOBJ

在 MQCLOSE 中提供了 MQCLOSE 调用的参数的描述。

将消息放置到队列上

使用此信息以了解如何将消息放置到队列上。

使用 MQPUT 调用将消息放置到队列上。 在初始 MQOPEN 调用后，您可以重复使用 MQPUT 以将很多消息放置到相同队列上。 将所有消息放置到队列上后，调用 MQCLOSE。

如果希望将单条消息放置到队列上并之后立即关闭队列，那么可使用 MQPUT1 调用。 MQPUT1 按照以下调用序列执行功能：

- MQOPEN
- MQPUT
- MQCLOSE

但是，通常，如果具有多条消息要放置到队列上，那么使用 MQPUT 调用会更有效。 这取决于消息的大小和所使用的平台。

使用以下链接，以了解有关将消息放置到队列上的更多信息：

- 第 701 页的『使用 MQPUT 调用将消息放置到本地队列上』
- 第 705 页的『将消息放置到远程队列上』
- 第 706 页的『设置消息的属性』
- 第 706 页的『控制消息上下文信息』
- 第 708 页的『使用 MQPUT1 调用将一条消息放置到队列上』
- 第 709 页的『分发列表』
- 第 713 页的『放置调用失败的一些情况』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

使用 *MQPUT* 调用将消息放置到本地队列上

使用此信息以了解如何使用 MQPUT 调用将消息放置到本地队列上。

作为 MQPUT 调用的输入，必须提供：

- 连接句柄 (Hconn)。
- 队列句柄 (Hobj)。
- 要放置到队列上的消息的描述。 此消息采用消息描述符结构 (MQMD) 的形式。
- 控制信息，采用放置消息选项结构 (MQPMO) 的形式。
- 消息中包含的数据的长度 (MQLONG)。
- 消息数据本身。

MQPUT 调用的输出如下所示：

- 原因码 (MQLONG)
- 完成代码 (MQLONG)

如果调用成功完成，那么还会返回选项结构以及消息描述符结构。 调用修改选项结构以显示要向其发送消息的队列的名称和队列管理器的名称。 如果请求队列管理器为将放置的消息的标识生成唯一值（通过在 MQMD 结构的 *MsgId* 字段中指定二进制零），那么调用会先在 *MsgId* 字段中插入值，再向您返回此结构。 发出另一个 MQPUT 之前重置此值。

MQPUT 中提供了 MQPUT 调用的描述。

有关作为 MQPUT 调用的输入所需的信息的更多描述，请参见以下链接：

- 第 702 页的『指定句柄』
- 第 702 页的『使用 MQMD 结构定义消息』
- 第 702 页的『使用 MQPMO 结构指定选项』
- 第 705 页的『消息中的数据』
- 第 705 页的『放置消息：使用消息句柄』

指定句柄

对于 z/OS 应用程序上的 CICS 中的连接句柄 (*Hconn*)，可以指定常量 MQHC_DEF_HCONN (值为零)，也可以使用 MQCONN 或 MQCONNX 调用返回的连接句柄。 对于其他应用程序，始终使用 MQCONN 或 MQCONNX 调用返回的连接句柄。

无论您处于什么环境，请使用 MQOPEN 调用返回的相同队列句柄 (*Hobj*)。

使用 MQMD 结构定义消息

消息描述符结构 (MQMD) 是 MQPUT 和 MQPUT1 调用的输入/输出参数。 使用它来定义要放置到队列上的消息。

如果为消息指定了 MQPRI_PRIORITY_AS_Q_DEF 或 MQPER_PERSISTENCE_AS_Q_DEF 且队列是集群队列，那么使用的值为 MQPUT 解析为的队列的值。 如果对此队列禁用 MQPUT，那么调用会失败。 请参阅配置队列管理器集群，以获取更多信息。

注: 放置新消息之前使用 MQPMO_NEW_MSG_ID 和 MQPMO_NEW_CORREL_ID，以确保 *MsgId* 和 *CorrelId* 唯一。 成功执行 MQPUT 时，会返回这些字段中的值。

第 15 页的『IBM MQ 消息』中提供了 MQMD 描述的消息属性的介绍，MQMD 中提供了结构本身的描述。

使用 MQPMO 结构指定选项

使用 MQPMO（放置消息选项）结构将选项传递到 MQPUT 和 MQPUT1 调用。

以下部分为您提供填充此结构的字段的帮助。 MQPMO 中提供了结构的描述。

此结构包含以下字段：

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*

- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

这些字段的内容如下所示：

StrucId
此字段将结构标识为放置消息选项结构。 这是一个 4 字符的字段。 始终指定 MQPMO_STRUC_ID。

版本
这描述了结构的版本号。 缺省值为 MQPMO_VERSION_1。 如果输入 MQPMO_VERSION_2，那么可使用分发列表（请参阅第 709 页的『分发列表』）。 如果输入 MQPMO_VERSION_3，那么可使用消息句柄和消息属性。 如果输入 MQPMO_CURRENT_VERSION，那么应用程序始终设置为使用最新级别。

选项
此字段控制以下内容：

- 是否在工作单元中包含放置操作
- 将多少上下文信息与消息关联
- 从其中获取上下文信息的位置
- 调用在队列管理器处于停顿状态时是否会失败
- 是否允许分组或分段
- 生成新消息标识和相关标识
- 将消息和分段放置到队列上的顺序
- 是否解析本地队列名称

如果将 *Options* 字段保留设置为缺省值 (MQPMO_NONE)，那么放置的消息具有关联的缺省上下文信息。

此外，调用采用同步点运行的方式由平台确定。 在 z/OS 中，缺省情况下，同步点控制为“是”；对于其他平台，为“否”。

Context
此字段说明希望从其复制上下文信息（如果在 *Options* 字段中请求）的队列句柄的名称。

有关消息上下文的介绍，请参阅第 39 页的『消息上下文』。 有关使用 MQPMO 结构来控制消息中上下文信息的信息，请参阅第 706 页的『控制消息上下文信息』。

ResolvedQName
此字段包含为接收消息而打开的队列的名称（解析任何别名后）。 这是输出字段。

ResolvedQMgrName
此字段包含在 *ResolvedQName* 中拥有队列的队列管理器的名称（在解析任何别名后）。 这是输出字段。

MQPMO 还可包含分发列表所需的字段（请参阅第 709 页的『分发列表』）。 如果要使用此工具，可使用 MQPMO V2 结构。 此结构包含以下字段：

RecsPresent
此字段包含分发列表中的队列数；即，存在的放置消息记录 (MQPMR) 和相应响应记录 (MQRR) 的数目。

输入的值可与 MQOPEN 提供的对象记录数相同。 但是，如果值小于执行 MQOPEN 调用时提供的对象记录数，或者如果未提供放置消息记录，那么未定义的队列的值取自消息描述符提供的缺省值。 此外，如果值大于提供的对象记录数，那么会忽略额外的放置消息记录。

建议执行以下其中一项操作：

- 如果要接收来自目标的报告或回复，请输入 MQOR 结构中所显示的相同值，并使用包含 *MsgId* 字段的 MQPMR。 将这些 *MsgId* 字段初始化为零或指定 MQPMO_NEW_MSG_ID。

  已将消息放置到队列上时，队列管理器创建的 *MsgId* 值在 MQPMR 中将可用；可以使用这些值来识别与各个报告或回复关联的目标。

- 如果不希望接收报告或回复，请选择以下项之一：
  1. 如果要立即识别失败的目标，那么可能仍要在 *RecsPresent* 字段中输入 MQOR 结构中显示的相同值，并提供 MQRR 以识别这些目标。 不要指定任何 MQPMR。
  2. 如果不希望识别失败的目标，请在 *RecsPresent* 字段中输入零，且不要提供 MQPMR 和 MQRR。

  注: 如果正使用 MQPUT1，那么响应记录指针数和响应记录偏移量数必须为零。

  有关放置消息记录 (MQPMR) 和响应记录 (MQRR) 的完整描述，请参阅 MQPMR 和 MQRR。

PutMsgRecFields
这指示每个放置消息记录 (MQPMR) 中存在哪些字段。 有关这些字段的列表，请参阅第 712 页的『使用 MQPMR 结构』。

PutMsgRecOffset* 和 *PutMsgRecPtr
指针（通常在 C 中）和偏移量（通常在 COBOL 中）用于处理放置消息记录（请参阅第 712 页的『使用 MQPMR 结构』以获取 MQPMR 结构的概述）。

使用 *PutMsgRecPtr* 字段指定指向第一个放置消息记录的指针，或者使用 *PutMsgRecOffset* 字段指定第一个放置消息记录的偏移量。 这是距离 MQPMO 开始的偏移量。 根据 *PutMsgRecFields* 字段，输入 *PutMsgRecOffset* 或 *PutMsgRecPtr* 的非空值。

ResponseRecOffset 和 ResponseRecPtr
还可使用指针和偏移量来处理响应记录（请参阅第 711 页的『使用 MQRR 结构』，以获取有关响应记录的更多信息）。

使用 *ResponseRecPtr* 字段指定指向第一个响应记录的指针，或者使用 *ResponseRecOffset* 字段指定第一个响应记录的偏移量。 这是距离 MQPMO 结构开始的偏移量。 输入 *ResponseRecOffset* 或 *ResponseRecPtr* 的非空值。

注: 如果正使用 MQPUT1 将消息放置到分发列表，那么 *ResponseRecPtr* 必须为 null 或零，*ResponseRecOffset* 必须为零。

此外，MQPMO V3 结构还包含以下字段：

OriginalMsgHandle
对此字段的使用取决于 *Action* 字段的值。 如果要放置具有关联消息属性的新消息，请将此字段设置为先前创建的消息句柄，并将属性设置为启用。 如果要转发、回复或生成报告以响应先前检索的消息，那么此字段包含此消息的消息句柄。

NewMsgHandle
如果指定 *NewMsgHandle*，那么与句柄关联的任何属性将覆盖与 *OriginalMsgHandle* 关联的属性。 有关更多信息，请参阅 Action (MQLONG)。

操作
使用此字段指定将执行的放置操作的类型。 可能值及其含义如下所示：

MQACTP_NEW
这是与其他任何消息不相关的新消息。

MQACTP_FORWARD
先前检索了此消息，现在将转发此消息。

MQACTP_REPLY
此消息是对先前检索的消息的回复。

MQACTP_REPORT
此消息是由于先前检索到的消息生成的报告。

有关更多信息，请参阅 Action (MQLONG)。

PubLevel
如果此消息是发布消息，那么可设置此字段以确定哪些预订接收此消息。 仅 *SubLevel* 小于或等于此值的预订将接收此发布。 缺省值为 9（最高级别），表示具有任何 *SubLevel* 的预订都可接收此发布。

消息中的数据

在 MQPUT 调用的 **Buffer** 参数中提供包含数据的缓冲区的地址。 您可以在消息的数据中包含任何内容。 但是，消息中的数据量会影响处理这些消息的应用程序的性能。

数据的最大大小由以下内容确定：

- 队列管理器的 **MaxMsgLength** 属性
- 将在其上放置消息的队列的 **MaxMsgLength** 属性
- IBM MQ 所添加的任何消息头（包括死信消息头 MQDLH 和分发列表头 MQDH）的大小

队列管理器的 **MaxMsgLength** 属性具有队列管理器可处理的消息的大小。 对于 V6 或更高版本的所有 IBM MQ 产品，该属性的缺省值为 100 MB。

要确定此属性的值，请针对队列管理器对象使用 MQINQ 调用。 对于较大的消息，可更改此值。

队列的 **MaxMsgLength** 属性确定可放置到队列上的消息的最大大小。 如果尝试放置大小大于此属性值的消息，那么 MQPUT 调用会失败。 如果要将消息放置到远程队列上，那么可成功放置的消息的最大大小由以下对象确定：远程队列、沿着目标路径放置消息的任何中间传输队列以及使用的通道的 **MaxMsgLength** 属性。

对于 MQPUT 操作，消息的大小必须小于或等于队列和队列管理器的 **MaxMsgLength** 属性。 这些属性的值是独立的，但建议您将队列的 *MaxMsgLength* 设置为小于或等于队列管理器的对应属性。

在以下情况下，IBM MQ 向消息添加头信息：

- 将消息放置到远程队列上时，IBM MQ 会向消息添加传输头结构 (MQXQH)。 此结构包含目标队列的名称及其拥有的队列管理器的名称。
- 如果 IBM MQ 无法将消息传递到远程队列，那么它会尝试将消息放置到死信（未传递的消息）队列上。 它会将 MQDLH 结构添加到消息。 此结构包含目标队列的名称以及将消息放置到死信队列上的原因。
- 如果要将消息发送到多个目标队列，那么 IBM MQ 会将 MQDH 头添加到消息。 这描述了消息中存在的数据，消息属于分发列表，在传输队列上。 为最大消息长度选择最优值时，考虑此情况。
- 如果消息为组中分段或消息，那么 IBM MQ 可添加 MQMDE。

这些结构在 MQDH 和 MQMDE 中进行了描述。

如果消息超出这些队列允许的最大大小，那么添加这些头意味着，由于消息目前太大会导致放置操作失败。 要降低放置操作失败的可能性，请执行以下操作：

- 使消息大小小于传输队列和死信队列的 **MaxMsgLength** 属性。 至少允许 MQ_MSG_HEADER_LENGTH 常量的值（对于较大分发列表，还需允许其他值）。
- 确保死信队列的 **MaxMsgLength** 属性设置为与拥有死信队列的队列管理器的 *MaxMsgLength* 相同。

队列管理器的属性和消息排队常量在队列管理器的属性中进行了描述。

z/OS 有关如何在分布式排队环境中处理未送达的消息的信息，请参阅未送达/未处理的消息。

放置消息：使用消息句柄

MQPMO 结构中提供了两个消息句柄：*OriginalMsgHandle* 和 *NewMsgHandle*。 这些消息句柄之间的关系由 MQPMO *Action* 字段的值定义。

有关完整详细信息，请参阅 Action (MQLONG)。 消息句柄对于放置消息不是必要的。 其目的是将属性与消息关联，因此，仅当使用消息属性时，才需要消息句柄。

将消息放置到远程队列上

要在远程队列（即，非应用程序连接到的队列管理器所拥有的队列）而不是本地队列上放置消息，唯一额外需要考虑的是打开队列时如何指定队列的名称。在第 699 页的『打开远程队列』中对此进行了描述。 对于本地队列使用 MQPUT 或 MQPUT1 调用则没有变化。

有关使用远程队列和传输队列的更多信息，请参阅 IBM MQ 分布式排队方法。

设置消息的属性

对于要设置的每个属性，调用 MQSETMP。 放置消息时，设置 MQPMO 结构的消息句柄和操作字段。

要将属性与消息关联，消息必须具有消息句柄。 使用 MQCRTMH 函数调用创建消息句柄。 通过对要设置的每个属性指定此消息句柄来调用 MQSETMP。 提供了样本程序 amqsstma.c 来说明 MQSETMP 的使用。

如果这是新消息，那么使用 MQPUT 或 MQPUT1 将其放置到队列时，将 MQPMO 中的 OriginalMsgHandle 字段设置为此消息句柄的值，将 MQPMO Action 字段设置为 MQACTP_NEW（这是缺省值）。

如果这是您以前检索到的消息，且现在将转发或回复此消息或者将发送报告以响应此消息，那么将原始消息句柄放置到 MQPMO 的 OriginalMsgHandle 字段中，将新消息句柄放置到 NewMsgHandle 字段中。 将 Action 字段相应设置为 MQACTP_FORWARD、MQACTP_REPLY 或 MQACTP_REPORT。

如果 MQRFH2 头中的属性来自先前检索到的消息，那么可使用 MQBUFMH 调用将其转换为消息句柄属性。

如果要将消息放置到级别早于 IBM WebSphere MQ 7.0 的队列管理器上的队列（无法处理消息属性），那么可在通道定义中设置 PropertyControl 参数以指定如何处理这些属性。

控制消息上下文信息

使用 MQPUT 或 MQPUT1 调用将消息放入队列时，您可以指定队列管理器向消息描述符中添加某些缺省上下文信息。 拥有相应权限级别的应用程序可以添加额外的上下文信息。 您可以在 MQPMO 结构中使用 options 字段来控制上下文信息。

消息上下文信息允许检索消息的应用程序查找有关消息的发起方的信息。 所有上下文信息都存储在消息描述符的上下文字段中。 信息类型划分为身份信息、源信息和用户上下文信息。

要控制上下文信息，请在 MQPMO 结构中使用 *Options* 字段。

如果没有为上下文信息指定任何选项，那么队列管理器会使用其已为消息生成的身份和上下文信息覆盖消息描述符中可能已存在的上下文信息。 这与指定 MQPMO_DEFAULT_CONTEXT 选项的效果相同。创建新消息时（例如，处理查询屏幕中的用户输入时），可能需要此缺省上下文信息。

如果不需要与消息关联的上下文信息，请使用 MQPMO_NO_CONTEX 选项。 放置不带任何上下文的消息时，会使用空用户标识执行 IBM MQ 执行的任何权限检查。 无法为空用户标识分配对 IBM MQ 资源的显式权限，但会将此标识视为特殊组“nobody”的成员。 有关特殊组 nobody 的更多详细信息，请参阅可安装服务接口参考信息。

可以通过使用 MQOPEN 后跟 MQPUT，同时使用以下部分中指示的 MQOO_ 选项和 MQPMO_ 选项来执行上下文设置。 还可仅使用 MQPUT1 执行上下文设置，在此情况下，仅需要选择以下部分中指示的 MQPMO_ 选项。

此主题的以下部分说明了身份上下文、用户上下文和所有上下文的使用。

- 第 706 页的『传递身份上下文』
- 第 707 页的『传递用户上下文』
- 第 707 页的『传递所有上下文』
- 第 707 页的『设置身份上下文』
- 第 707 页的『设置用户上下文』
- 第 707 页的『设置所有上下文』

传递身份上下文

通常，程序应该在应用程序中消息之间传递身份上下文信息，直到数据到达其最终目标。

程序在每次更改数据时都应该更改源上下文信息。 但是，需要更改或设置任何上下文信息的应用程序必须具有相应的权限级别。 应用程序打开队列时，队列管理器会检查此权限；这些应用程序必须有权使用 MQOPEN 调用的相应上下文选项。

如果应用程序获取消息，处理消息中的数据，然后将更改的数据放置到其他消息中（可能供其他应用程序处理），那么应用程序必须将身份上下文信息从原始消息传递到新消息。 可以允许队列管理器创建源上下文信息。

要保存来自原始消息的上下文信息，请在打开队列以获取消息时使用 MQOO_SAVE_ALL_CONTEXT 选项。 还使用与 MQOPEN 调用配合使用的任何其他选项。 但是，请注意，如果仅浏览消息，那么无法保存上下文信息。

创建第二条消息时：

- 使用 MQOO_PASS_IDENTITY_CONTEXT 选项（以及 MQOO_OUTPUT 选项）打开队列。
- 在放置消息选项结构的 *Context* 字段中，提供保存了其中上下文信息的队列的句柄。
- 在放置消息选项结构的 *Options* 字段中，指定 MQPMO_PASS_IDENTITY_CONTEXT 选项。

传递用户上下文

无法选择仅传递用户上下文。 要在放置消息时传递用户上下文，请指定 MQPMO_PASS_ALL_CONTEXT。 用户上下文中任何属性以源上下文中相同的方式传递。

发生 MQPUT 或 MQPUT1 调用且传递上下文时，会将用户上下文中的所有属性从检索到的消息传递到放置消息。 放置应用程序已更改的任何用户上下文属性将使用其原始值进行放置。 放置应用程序已删除的任何用户上下文属性将在放置消息中复原。 将保留放置应用程序已添加到消息的任何用户上下文属性。

传递所有上下文

如果应用程序获取消息，将消息数据（未更改）放置到其他消息中，那么应用程序必须将所有上下文信息（身份、源和用户）从原始消息传递到新消息。 可执行此操作的应用程序示例为消息移动程序，其将消息从一个队列移动到另一个队列。

遵循与传递身份上下文相同的过程，不同之处在于使用 MQOPEN 选项 MQOO_PASS_ALL_CONTEXT 和放置消息选项 MQPMO_PASS_ALL_CONTEX。

设置身份上下文

如果要为消息设置身份上下文信息，请执行以下操作：

- 使用 MQOO_SET_IDENTITY_CONTEXT 选项打开队列。
- 将消息放置到队列上，同时指定 MQPMO_SET_IDENTITY_CONTEXT 选项。 在消息描述符中，指定所需的任何身份上下文信息。

注: 使用 MQOO_SET_IDENTITY_CONTEXT 和 MQPMO_SET_IDENTITY_CONTEXT 选项设置一些（而不是全部）身份上下文字段时，了解队列管理器不会设置任何其他字段这一点很重要。

要修改任何消息上下文选项，必须具有相应权限来发出调用。 例如，要使用 MQOO_SET_IDENTITY_CONTEXT 或 MQPMO_SET_IDENTITY_CONTEXT，必须具有 `+setid` 许可权。

设置用户上下文

要在用户上下文中设置属性，请在执行 MQSETMP 调用时，将消息属性描述符 (MQPD) 的 Context 字段设置为 MQPD_USER_CONTEXT。

不需要任何特殊权限，即可在用户上下文中设置属性。 用户上下文没有 MQOO_SET_* 或 MQPMO_SET_* 上下文选项。

设置所有上下文

如果要为消息设置身份上下文信息和源上下文信息，请执行以下操作：

1. 使用 MQOO_SET_ALL_CONTEXT 选项打开队列。
2. 将消息放置到队列上，同时指定 MQPMO_SET_ALL_CONTEXT 选项。 在消息描述符中，指定所需的任何身份上下文信息和源上下文信息。

每种类型的上下文设置需要相应权限。

相关概念

第 39 页的『消息上下文』

消息上下文信息使得检索消息的应用程序能够了解有关消息发起方的信息。

相关参考

第 698 页的『用于关联消息上下文的 MQOPEN 选项』

如果要可以在将消息放置到队列上时将上下文信息与消息关联，必须在打开队列时使用其中一个消息上下文选项。

使用 *MQPUT1* 调用将一条消息放置到队列上

如果希望在将单条消息放置到队列上后立即关闭队列，请使用 MQPUT1 调用。 例如，服务器应用程序在向各个队列发送回复时，可能会使用 MQPUT1 调用。

MQPUT1 的功能相当于调用 MQOPEN 后跟 MQPUT 后跟 MQCLOSE 的功能。 MQPUT 和 MQPUT1 调用的语法中唯一差异在于对于 MQPUT 指定对象句柄，而对于 MQPUT1 则指定 MQOPEN 中定义的对象描述符结构 (MQOD)（请参阅第 694 页的『标识对象（MQOD 结构）』）。 这是因为，需要为 MQPUT1 调用提供有关其必须打开的队列的信息，而调用 MQPUT 时，此队列必须已打开。

作为 MQPUT1 调用的输入，必须提供：

- 连接句柄。
- 要打开的对象的描述。 这采用了对象描述符结构 (MQOD) 的形式。
- 要放置到队列上的消息的描述。 此消息采用消息描述符结构 (MQMD) 的形式。
- 控制信息，采用放置消息选项结构 (MQPMO) 的形式。
- 消息中包含的数据的长度 (MQLONG)。
- 消息数据的地址。

MQPUT1 的输出为：

- 完成代码
- 原因码

如果调用成功完成，那么还会返回选项结构以及消息描述符结构。 调用修改选项结构以显示要向其发送消息的队列的名称和队列管理器的名称。 如果请求队列管理器为将放置的消息的标识生成唯一值（通过在 MQMD 结构的 *MsgId* 字段中指定二进制零），那么调用会先在 *MsgId* 字段中插入值，再向您返回此结构。

注: 无法将 MQPUT1 与模型队列名称配合使用；但是，打开模型队列后，可向动态队列发出 MQPUT1。

MQPUT1 的六个输入参数为：

Hconn
这是连接句柄。 对于 CICS 应用程序，可以指定常量 MQHC_DEF_HCONN（具有值零），或使用 MQCONN 或 MQCONNX 调用返回的连接句柄。 对于其他程序，始终使用 MQCONN 或 MQCONNX 调用返回的连接句柄。

ObjDesc
这是对象描述符结构 (MQOD)。

在 *ObjectName* 和 *ObjectQMgrName* 字段中，提供希望在其上放置消息的队列的名称以及拥有此队列的队列管理器的名称。

针对 MQPUT1 调用，将忽略 *DynamicQName* 字段，因为此调用无法使用模型队列。

如果希望指定将用于测试打开队列的权限的备用用户标识，请使用 *AlternateUserId* 字段。

MsgDesc
这是消息描述符结构 (MQMD)。 与 MQPUT 调用一样，使用此结构定义将放置到队列上的消息。

PutMsgOpts
这是放置消息选项结构 (MQPMO)。 使用方式与 MQPUT 调用一样（请参阅第 702 页的『使用 MQPMO 结构指定选项』）。

Options 字段设置为零时，队列管理器会使用您自己的用户标识对访问队列的权限执行测试。 此外，队列管理器会忽略 MQOD 结构的 *AlternateUserId* 字段中提供的任何备用用户标识。

BufferLength
这是消息的长度。

Buffer
这是包含消息文本的缓冲区区域。

使用集群时，MQPUT1 会运行，好像 MQOO_BIND_NOT_FIXED 已生效。 应用程序必须使用 MQPMO 结构而不是 MQOD 结构中的解析字段来确定消息的发送位置。 请参阅配置队列管理器集群，以获取更多信息。

MQPUT1 中提供了 MQPUT1 调用的描述。

分发列表

在 IBM MQ for z/OS 上不受支持。 分发列表允许您在单个 MQPUT 或 MQPUT1 调用中将消息放置到多个目标中。单个 MQOPEN 调用可打开多个队列，然后，单个 MQPUT 调用可将消息放置到每个队列上。 用于此进程的 MQI 结构中一些通用信息可替代为与分发列表中包含的个别目标相关的特定信息。

注意: 分发列表不支持使用指向主题对象的别名队列。 如果别名队列指向分发列表中的主题对象，那么 IBM MQ 将返回 MQRC_ALIAS_BASE_Q_TYPE_ERROR。

发出 MQOPEN 调用时，通用信息取自对象描述符 (MQOD)。 如果在 *Version* 字段中指定 MQOD_VERSION_2，在 *RecsPresent* 字段中指定大于零的值，那么可以将 *Hobj* 定义为（一个或多个队列的）列表的句柄而不是队列的句柄。 在此情况下，通过对象记录 (MQOR) 提供特定信息，对象记录提供了目标的详细信息（即 *ObjectName* 和 *ObjectQMgrName*）。

将向 MQPUT 调用传递对象句柄 (*Hobj*)，这允许您将消息放置到列表中而不是单个队列中。

将消息放置到队列上时 (MQPUT)，通用信息取自放置消息选项结构 (MQPMO) 和消息描述符 (MQMD)。 特定信息采用放置消息记录 (MQPMR) 的形式提供。

响应记录 (MQRR) 可以接收特定于每个目标队列的完成代码和原因码。

第 709 页的图 61 显示分发列表如何工作。

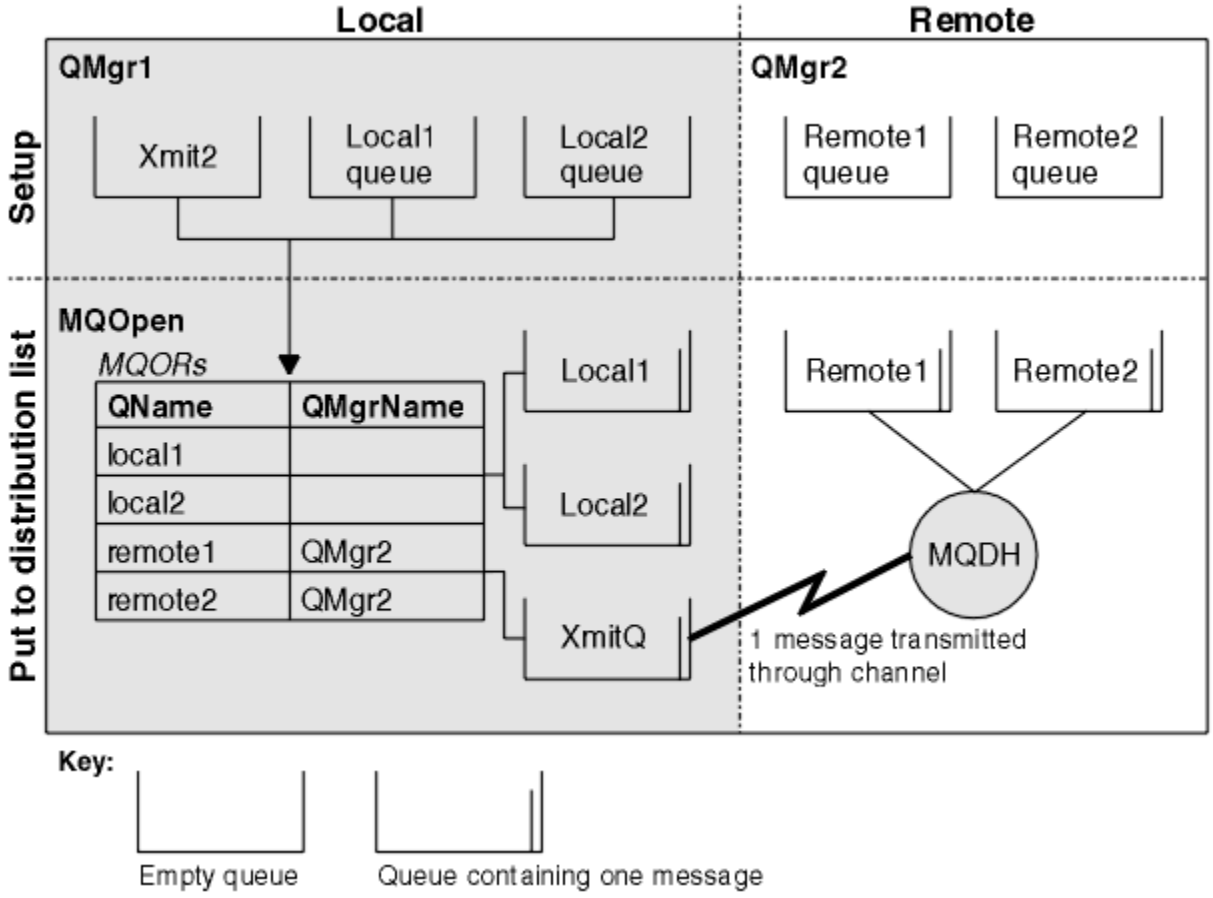


图 *61:* 分发列表如何工作

打开分发列表
使用 MQOPEN 调用打开分发列表，使用调用的选项指定要对列表执行的操作。

作为 MQOPEN 的输入，必须提供：

- 连接句柄（请参阅第 700 页的『将消息放置到队列上』以获取描述）
- 对象描述符结构 (MQOD) 中的通用信息
- 要使用对象记录结构 (MQOR) 打开的每个队列的名称

MQOPEN 的输出为：

- 表示您对此分发列表的访问权的对象句柄。
- 通用完成代码
- 通用原因码
- 响应记录（可选），包含每个目标的完成代码和原因

使用 MQOD 结构

使用 MQOD 结构标识要打开的队列。

要定义分发列表，必须在 *Version* 字段中指定 MQOD_VERSION_2，在 *RecsPresent* 字段中指定大于 0 的值，在 *ObjectType* 字段中指定 MQOT_Q。 请参阅 MQOD，以获取 MQOD 结构的所有字段的描述。

使用 MQOR 结构

提供每个目标的 MQOR 结构。

此结构包含目标队列名称和队列管理器名称。 不会将 MQOD 中 *ObjectName* 和 *ObjectQMgrName* 字段用于分发列表。 必须具有一条或多条对象记录。 如果 *ObjectQMgrName* 保留为空，那么会使用本地队列管理器。 请参阅 ObjectName 和 ObjectQMgrName，以获取有关这些字段的更多信息。

您可以通过两种方式指定目标队列：

- 通过使用偏移量字段 *ObjectRecOffset*。

  在此情况下，应用程序必须声明其自己的结构（包含 MQOD 结构），此结构后跟 MQOR 记录的数组（以及所需数量的数组元素），将 *ObjectRecOffset* 设置为从 MQOD 开始的数组中第一个元素的偏移量。 确保此偏移量正确。

  建议使用编程语言提供的内置工具（前提是这些工具在运行应用程序的所有环境中可用）。 以下代码针对 COBOL 编程语言说明了此方法：

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  或者，如果编程语言在所有相关环境中不支持必需的内置工具，请使用常量 MQOD_CURRENT_LENGTH。 以下代码说明此方法：

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  但是，仅当 MQOD 结构和 MQOR 记录的数组连续时，这才有效；如果编译器在 MQOD 和 MQOR 数组之间插入跳过字节，那么必须将这些字节添加到 *ObjectRecOffset* 中存储的值。

  对于不支持指针数据类型的编程语言或实现指针数据类型的方式不适用于其他环境的编程语言（例如，COBOL 编程语言），建议使用 *ObjectRecOffset*。

- 通过使用指针字段 *ObjectRecPtr*。

在此情况下，应用程序可独立于 MQOD 结构声明 MQOR 结构的数组，并将 *ObjectRecPtr* 设置为数组的地址。 以下代码针对 C 编程语言说明了此方法：

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

对于支持指针数据类型的方式不适用于其他环境的编程语言（例如，C 编程语言），建议使用 *ObjectRecPtr*。

无论选择哪种方法，必须使用 *ObjectRecOffset* 和 *ObjectRecPtr* 之一；如果这两者都为零或为非零，那么调用会失败，且会返回原因码 MQRC_OBJECT_RECORDS_ERROR。

使用 MQRR 结构

这些结构特定于目标；针对分发列表的每个队列，每条响应记录都包含 *CompCode* 和 *Reason* 字段。 必须使用此结构辨别任何问题的发生位置。

例如，如果收到原因码 MQRC_MULTIPLE_REASONS，且分发列表包含五个目标队列，在不使用此结构的情况下，您不会知道发生问题的队列。 但是，如果针对每个目标具有完成代码和原因码，便可更轻松地找到错误。

请参阅 MQRR，以获取有关 MQRR 结构的更多信息。

第 711 页的图 62 说明您如何在 C 中打开分发列表。

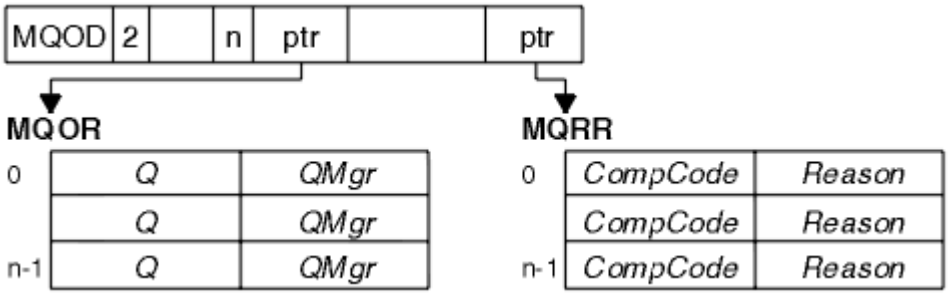


图 *62:* 在 *C* 中打开分发列表

第 711 页的图 63 说明您如何在 COBOL 中打开分发列表。

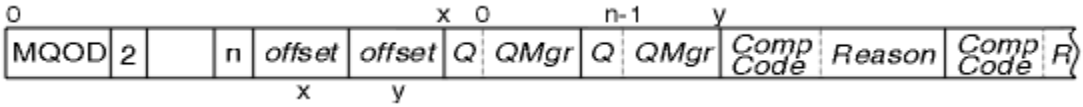


图 *63:* 在 *COBOL* 中打开分发列表

使用 MQOPEN 选项

打开分发列表时，可指定以下选项：

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING（可选）
- MQOO_ALTERNATE_USER_AUTHORITY（可选）
- MQOO_*_CONTEXT（可选）

请参阅第 692 页的『打开和关闭对象』，以获取这些选项的描述。

将消息放到分发列表
要将消息放置到分发列表中，可使用 MQPUT 或 MQPUT1。

作为输入，必须提供：

- 连接句柄（请参阅第 700 页的『将消息放置到队列上』以获取描述）。

- 对象句柄。 如果使用 MQOPEN 打开了分发列表，那么 *Hobj* 仅允许您将消息放置到列表中。
- 消息描述符结构 (MQMD)。 请参阅 MQMD，以获取有关此结构的描述。
- 控制信息，采用放置消息选项结构 (MQPMO) 的形式。 请参阅第 702 页的『使用 MQPMO 结构指定选项』，以获取有关填写 MQPMO 结构的字段的信息。
- 控制信息，采用放置消息记录 (MQPMR) 的形式。
- 消息中包含的数据的长度 (MQLONG)。
- 消息数据本身。

输出为：

- 完成代码
- 原因码
- 响应记录（可选）

使用 MQPMR 结构

此结构是可选的，为可能希望以不同于在 MQMD 中标识字段方式来标识的一些字段提供特定于目标的信息。

有关这些字段的描述，请参阅 MQPMR。

每条记录的内容取决于在 MQPMO 的 *PutMsgRecFields* 字段中提供的信息。 例如，在说明分发列表的使用的样本程序 AMQSPTL0.C（请参阅第 990 页的『“分发列表”样本程序』以获取描述）中，样本选择为 MQPMR 中的 *MsgId* 和 *CorrelId* 提供值。 样本程序的此部分看似如下：

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

这表示为分发列表的每个目标提供了 *MsgId* 和 *CorrelId*。 放置消息记录作为数组提供。

第 712 页的图 64 说明您如何在 C 中将消息放置到分发列表中。

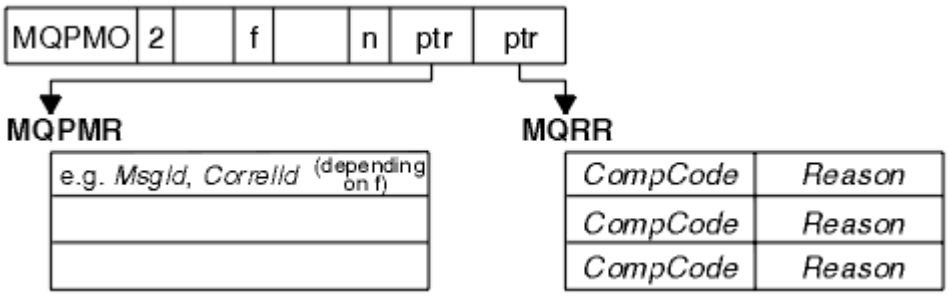


图 *64:* 在 *C* 中将消息放置到分发列表中

第 712 页的图 65 说明您如何在 COBOL 中将消息放置到分发列表中。

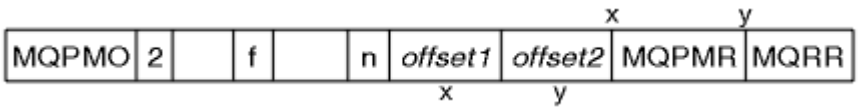


图 *65:* 在 *COBOL* 中将消息放置到分发列表中

使用 MQPUT1

如果要使用 MQPUT1，请考虑以下几点：

1. *ResponseRecOffset* 和 *ResponseRecPtr* 字段的值必须为 null 或零。
2. 必要时，必须从 MQOD 处理响应记录。

放置调用失败的一些情况

如果在发出 MQOPEN 和 MQPUT 调用之间的时间间隔内在命令上使用 FORCE 选项更改了队列的某些属性，那么 MQPUT 调用将失败并返回 MQRC_OBJECT_CHANGED 原因码。

队列管理器将对象句柄标记为不再有效。 如果在处理 MQPUT1 调用时执行了更改，或者如果更改应用于队列名称解析到的任何队列，那么也可能会发生此情况。 MQOPEN 中的 MQOPEN 调用描述中列出以此方式影响句柄的属性。 如果调用返回 MQRC_OBJECT_CHANGED 原因码，请关闭队列后将其重新打开，然后再次尝试放置消息。

如果禁止针对将尝试在其上放置消息的队列（或队列名称解析到的任何队列）执行放置操作，那么 MQPUT 或 MQPUT1 调用会失败，且会返回 MQRC_PUT_INHIBITED 原因码。 如果以后尝试执行此调用且应用程序设计为其他程序定期更改队列的属性，那么可能可以成功放置消息。

此外，如果尝试在其上放置消息的队列已满，那么 MQPUT 或 MQPUT1 调用会失败且会返回 MQRC_Q_FULL。

如果已删除动态队列（临时或永久），那么使用先前获取的对象句柄的 MQPUT 调用会失败，且会返回 MQRC_Q_DELETED 原因码。 在此情况下，最好关闭对象句柄，因为不再有用。

如果使用分发列表，那么会在单个请求中出现多个完成代码和原因码。 无法通过仅针对 MQOPEN 和 MQPUT 使用 *CompCode* 和 *Reason* 输出字段处理这些完成代码和原因码。

使用分发列表将消息放置到多个目标时，响应记录包含每个目标的特定 *CompCode* 和 *Reason*。 如果收到完成代码 MQCC_FAILED，表明在任何目标队列上放置消息均失败。 如果完成代码为 MQCC_WARNING，表明已在一个或多个目标队列上成功放置消息。 如果收到返回码 MQRC_MULTIPLE_REASONS，表明各个目标的原因码并不全部相同。 因此，建议使用 MQRR 结构，以便可确定哪个队列或哪些队列导致了错误以及每个错误的原因。

从队列取出消息

使用此信息以了解如何从队列取出消息。

您可以通过以下两种方式从队列取出消息：

1. 可以从队列移除消息，以使其他程序不再看到此消息。
2. 您可以复制消息，同时保留队列上的原始消息。 这称为浏览。 浏览后，可移除消息。

在这两种情况下，使用 MQGET 调用，但首先应用程序必须连接到队列管理器，必须使用 MQOPEN 调用打开队列（用于输入和/或浏览）。 这些操作在第 685 页的『连接到队列管理器和从队列管理器断开连接』和第 692 页的『打开和关闭对象』中进行了描述。

打开队列后，可重复使用 MQGET 调用来浏览或移除相同队列上的消息。 从队列获取所需的所有消息后，调用 MQCLOSE。

使用以下链接以了解有关从队列取出消息的更多信息：

- 第 714 页的『使用 MQGET 调用从队列取出消息』
- 第 718 页的『从队列检索消息的顺序』
- 第 727 页的『获取特定消息』
- 第 728 页的『提高非持久消息的性能』
- z/OS 第 732 页的『索引类型』
- 第 732 页的『处理消息长度大于 4 MB 的消息』
- 第 737 页的『等待消息』
- z/OS 第 737 页的『发信号』
- 第 738 页的『跳过回退』
- 第 740 页的『应用程序数据转换』

- 第 741 页的『浏览队列中的消息』
- 第 746 页的『MQGET 调用失败的一些情况』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

使用 *MQGET* 调用从队列取出消息

MQGET 调用可从打开的本地队列获取消息。 它无法从其他系统上的队列获取消息。

作为对 MQGET 调用的输入，您必须提供：

- 连接句柄。
- 队列句柄。
- 要从队列获取的消息的描述。 它采用消息描述符 (MQMD) 结构形式。
- 采用“获取消息选项 (MQGMO) ”结构的控制信息。
- 已分配用于保存消息 (MQLONG) 的缓冲区的大小。
- 放入消息的存储器的地址。

来自 MQGET 的输出为：

- 原因码
- 完成代码
- 指定的缓冲区中的消息（如成功完成调用）
- 选项结构，将对其进行修改以显示从中检索消息的队列的名称
- 消息描述符结构，具有已修改用于描述检索的消息的字段内容
- 消息长度 (MQLONG)

MQGET 中有 MQGET 调用描述。

以下部分描述了必须作为对 MQGET 调用的输入提供的信息。

- 第 715 页的『指定连接句柄』
- 第 715 页的『使用 MQMD 结构和 MQGET 调用描述消息』

- 第 715 页的『使用 MQGMO 结构指定 MQGET 选项』
- 第 717 页的『指定缓冲区的大小』

指定连接句柄

z/OS 对于 z/OS 应用程序上的 CICS，可以指定常量 MQHC_DEF_HCONN (值为零)，或者使用 MQCONN 或 MQCONNX 调用返回的连接句柄。 对于其他应用程序，始终使用 MQCONN 或 MQCONNX 调用返回的连接句柄。

使用调用 MQOPEN 时返回的队列句柄 (*Hobj*)。

使用 MQMD 结构和 MQGET 调用描述消息

要识别希望从队列获取的消息，请使用消息描述符结构 (MQMD)。

这是 MQGET 调用的输入/输出参数。 第 15 页的『IBM MQ 消息』中提供了 MQMD 描述的消息属性的介绍，MQMD 中提供了结构本身的描述。

如果您知道要从队列获取哪一条消息，请参阅第 727 页的『获取特定消息』。

如果不指定特定消息，MQGET 将检索队列中的第一条消息。 第 718 页的『从队列检索消息的顺序』描述了消息优先级、队列的 **MsgDeliverySequence** 属性以及 MQGMO_LOGICAL_ORDER 选项如何确定队列中消息的顺序。

注: 如果要多次使用 MQGET（例如，逐句获取队列中的消息），必须在每次调用后将此结构的 *MsgId* 和 *CorrelId* 字段设置为空。 这将清除这些所检索消息标识的字段。

但是，如果要为消息分组，*GroupId* 对于同一组中的消息必须相同，以便调用过程查找与前一条消息标识相同的消息，以构成整个组。

使用 MQGMO 结构指定 MQGET 选项

MQGMO 结构是用于将选项传递到 MQGET 调用的输入/输出变量。 以下部分可帮助您填写此结构的一些字段。

MQGMO 中有 MQGMO 结构描述。

StrucId
StrucId 是一个 4 字符字段，用于将结构标识为 get-message 选项结构。 请始终指定 MQGMO_STRUC_ID。

Version
Version 描述结构的版本号。 缺省情况下为 MQGMO_VERSION_1。 如果要使用 Version 2 字段或以逻辑顺序检索消息，请指定 MQGMO_VERSION_2。 如果要使用 Version 3 字段或以逻辑顺序检索消息，请指定 MQGMO_VERSION_3。 MQGMO_CURRENT_VERSION 设置应用程序以使用最新的级别。

Options
您可以在代码内按任意顺序选择选项；每个选项均由 *Options* 字段中的一位来表示。

Options 字段可控制：

- MQGET 调用在完成前是否等待消息到达队列（请参阅第 737 页的『等待消息』）
- 工作单元中是否包含获取操作。
- 是否在同步点之外检索非持久消息，允许快速消息传递
- z/OS 在 IBM MQ for z/OS 上，所检索消息是否标记为“跳过回退”（请参阅第 738 页的『跳过回退』）
- 是否从队列中除去消息，或仅浏览消息
- 是否使用浏览光标或其他选择标准选择消息
- 即使在消息长度超过缓冲区所允许的长度时，是否也能成功调用

- z/OS 在 IBM MQ for z/OS 上，是否允许完成调用。 该选项还设置了一个信号，指示您是否想在消息到达时收到通知
- 调用在队列管理器处于停顿状态时是否会失败
- z/OS 在 IBM MQ for z/OS 上，如果连接处于停顿状态，调用是否失败
- 是否需要转换应用程序消息数据（请参阅第 740 页的『应用程序数据转换』）
- 从队列检索消息和段的顺序 z/OS （IBM MQ for z/OS 除外）
- 是否只能检索完整的逻辑消息 z/OS （IBM MQ for z/OS 除外）
- 是否只能在组中的所有消息均可用时检索该组中的消息。
- 是否只能在逻辑消息中的所有段均可用时检索此逻辑消息中的段 z/OS （IBM MQ for z/OS 除外）

如果保持将 *Options* 字段设置为缺省值 (MQGMO_NO_WAIT)，那么 MQGET 调用将以此方式进行：

- 如果队列上没有消息与选择标准匹配，那么调用不会等待消息到达，而是立即完成。 z/OS 同样，在 IBM MQ for z/OS 中，调用未设置信号以在此类消息到达时请求通知。
- 通过同步点执行调用的方式由平台决定：

平台	在同步点控制下
IBM i	否
UNIX and Linux 系统	否
z/OS z/OS z/OS	Yes
Windows 系统	否

- z/OS 在 IBM MQ for z/OS 上，检索的消息未标记为“跳过回退”。
- 从队列中除去所选消息（不浏览）。
- 无需转换应用程序消息数据。
- 如果消息长度大于缓冲区允许的长度，调用失败。

WaitInterval

WaitInterval 字段指定 MQGET 调用在使用 MQGMO_WAIT 选项时等待消息到达队列的最长时间 (以毫秒计)。 如果在 *WaitInterval* 中指定的时间内没有消息到达，那么调用将完成并返回原因码，显示队列上没有与您的选择标准匹配的消息。

z/OS 在 IBM MQ for z/OS 上，如果使用 MQGMO_SET_SIGNAL 选项，那么 *WaitInterval* 字段指定设置信号的时间。

有关这些选项的更多信息，请参阅第 737 页的『等待消息』 z/OS 和第 737 页的『发信号』。

Signal1

Signal1 仅在 z/OS IBM MQ for z/OS 上受支持。

如果使用 MQGMO_SET_SIGNAL 选项请求在适当的消息到达时通知应用程序，请在 *Signal1* 字段中指定信号类型。 在所有其他平台上的 IBM MQ 中， *Signal1* 字段是保留的，其值并不重要。

z/OS 有关更多信息，请参阅第 737 页的『发信号』。

Signal2

Signal2 字段在所有平台上均为保留字段，其值并不重要。

z/OS 有关更多信息，请参阅第 737 页的『发信号』。

ResolvedQName
ResolvedQName 是一个输出字段，队列管理器在其中将返回从中检索消息的队列的名称（解析任何别名后）。

MatchOptions
MatchOptions 控制用于 MQGET 的选择标准。

GroupStatus
GroupStatus 指示检索的消息是否在组中。

SegmentStatus
SegmentStatus 指示检索的项是否为逻辑消息段。

Segmentation
Segmentation 指示检索的消息是否允许分段。

MsgToken

MsgToken 可唯一标识消息。

ReturnedLength
ReturnedLength 是一个输出字段，队列管理器在其中返回所返回的消息数据的长度（字节）。

MsgHandle
消息句柄，将在其中填充从队列检索的消息属性。 句柄先前由 MQCRTMH 调用创建。 检索消息前，将清除已与句柄关联的所有属性。

指定缓冲区的大小

在 MQGET 调用的 **BufferLength** 参数中，指定保存检索的消息数据的缓冲区的大小。 您可使用如下三种方式来决定其大小：

1. 您可能已知道该程序预期的消息长度。 如果知道，请指定具有此大小的缓冲区。

   但是，即使消息对于缓冲区而言过大，如果您要完成 MQGET 调用，也可以在 MQGMO 结构中使用 MQGMO_ACCEPT_TRUNCATED_MSG 选项。 在这种情况下：

   - 向缓冲区尽可能多地填充消息
   - 调用返回警告完成代码
   - 从队列中除去消息（丢弃其余消息），或推进浏览光标（如您正在浏览队列）
   - *DataLength* 中返回消息的真实长度

   如果没有该选项，调用仍将完成且发出警告，但不会从队列中除去消息（或推进浏览光标）。

2. 估算缓冲区大小（甚至指定零字节大小），不要使用 MQGMO_ACCEPT_TRUNCATED_MSG 选项。 如果 MQGET 调用失败（例如，由于缓冲区过小），那么调用的 **DataLength** 参数中将返回消息长度。 （缓冲区仍尽可能多地填充消息，但未完成调用处理。） 存储此消息的 *MsgId*，然后重复进行 MQGET 调用，指定具有正确大小的缓冲区，以及第一次调用期间注释的 *MsgId*。

   如果您的程序服务的队列同样由其他程序提供服务，那么这些其他程序中的某一程序可能会在您的程序可发出其他 MQGET 调用前除去所需要的消息。 程序可能会浪费时间来搜索不再存在的消息。 为避免此情况，请首先浏览队列，直至找到需要的消息，指定 *BufferLength* 为零并使用 MQGMO_ACCEPT_TRUNCATED_MSG 选项。 这会将浏览光标放在需要的消息下方。 然后，可重新调用 MQGET 并指定 MQGMO_MSG_UNDER_CURSOR 选项来检索消息。 如果其他程序除去“浏览与除去”调用间的消息，那么第二个 MQGET 将立即失败（未搜索整个队列），因为在浏览光标下无任何消息。

3. *MaxMsgLength queue* 属性可确定此队列接受的消息的最大长度；*MaxMsgLength queue manager* 属性可确定此队列管理器接受的消息的最大长度。 如果您不清楚预期的消息长度，可询问 **MaxMsgLength** 属性（使用 MQINQ 调用），然后指定具有此大小的缓冲区。

   尝试使缓冲区大小尽可能接近实际消息大小，以避免降低性能。

   有关 **MaxMsgLength** 属性的更多信息，请参阅第 732 页的『增加最大消息长度』。

从队列检索消息的顺序

您可以控制从队列检索消息的顺序。 本部分将介绍这些选项。

优先级

程序在将消息放在队列上时，可向此消息分配优先级（请参阅第 21 页的『消息优先级』）。 优先级相同的消息会按到达顺序（而不是落实顺序）存储在队列中。

队列管理器可以按严格的 FIFO（先入先出）顺序维护队列，或者在优先级序列中以 FIFO 顺序维护队列。 这取决于队列的 **MsgDeliverySequence** 属性设置。 消息到达队列时，将紧跟在具有相同优先级的最后一条消息之后插入此消息。

程序可从队列中获取第一条消息，或可从队列中获取特定的消息，忽略这些消息的优先级。 例如，程序可能要处理对先前发送的特定消息的回复。 有关更多信息，请参阅 第 727 页的『获取特定消息』。

如果应用程序将一连串消息放到队列上，那么其他应用程序可以按照放入消息的顺序来检索这些消息，前提是：

- 所有消息具有相同的优先级
- 所有消息均放在同一工作单元中，或均放在工作单元外
- 队列对于放入应用程序是本地队列

如果不符合这些条件，并且应用程序取决于以特定顺序检索的消息，那么应用程序必须在消息数据中包含序列信息，或在发送下一条消息前建立一种确认收到消息的方式。

z/OS 在 IBM MQ for z/OS 上，可使用队列属性 *IndexType*，来提高队列上 MQGET 操作的速度。 有关更多信息，请参阅 第 732 页的『索引类型』。

逻辑与物理排序

队列上的消息可以物理或逻辑顺序（在每个优先级内）显示。

物理顺序是消息到达队列的顺序。 逻辑顺序是满足下列条件时的顺序：当组中的所有消息和段位于其逻辑序列中由属于该组的第一个项的物理位置确定的位置且彼此相邻时。

有关组、消息和段的描述，请参阅第 36 页的『消息组』。 这些物理和逻辑顺序可能会有所不同，因为：

- 组可以在相近的时间从不同的应用程序到达目标，从而丢失任何明确的物理顺序。
- 即使在单个组内，消息也可能因为组中的某些消息重排或延迟而变得无序。

例如，逻辑顺序可能类似图第 719 页的图 66：

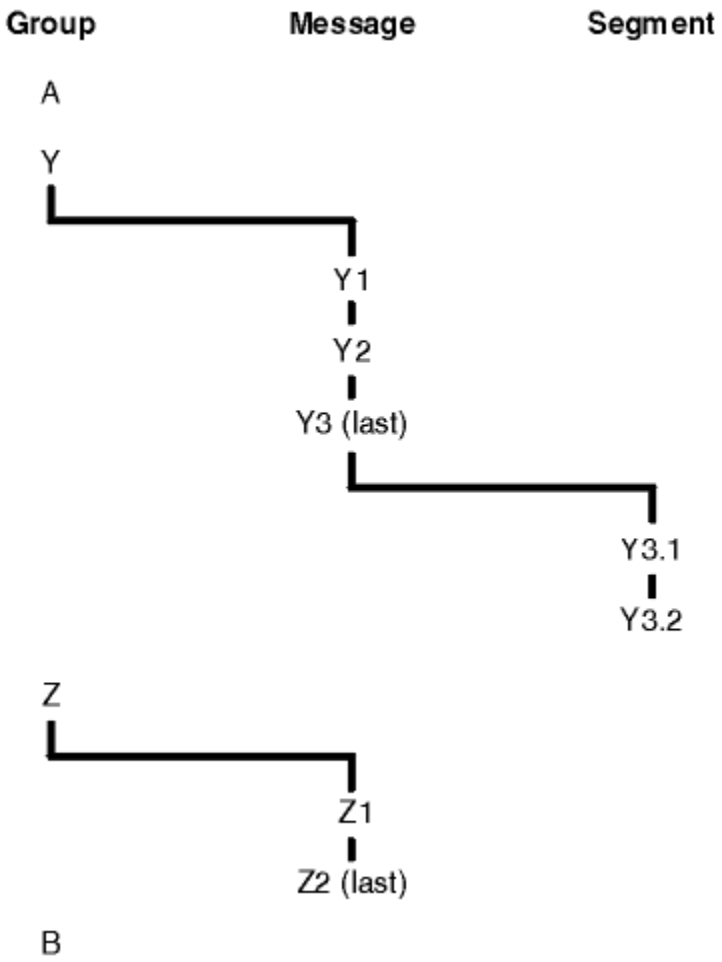


图 *66:* 队列上的逻辑顺序

这些消息将以如下逻辑顺序在队列上显示:

1. 消息 A（不在组中）
2. 组 Y 的逻辑消息 1
3. 组 Y 的逻辑消息 2
4. 组 Y 的逻辑消息 3（最后一条消息）的段 1
5. 组 Y 的逻辑消息 3（最后一条消息）的段 2（最后一个段）
6. 组 Z 的逻辑消息 1
7. 组 Z 的逻辑消息 2（最后一条消息）
8. 消息 B（不在组中）

但是，物理顺序可能完全不同。 每个组中第一个项的物理位置确定整个组的逻辑位置。 例如，如果组 Y 和组 Z 到达的时间接近，并且组 Z 的消息 2 位于同一组的消息 1 之前，那么物理顺序将类似图第 720 页的图 67:

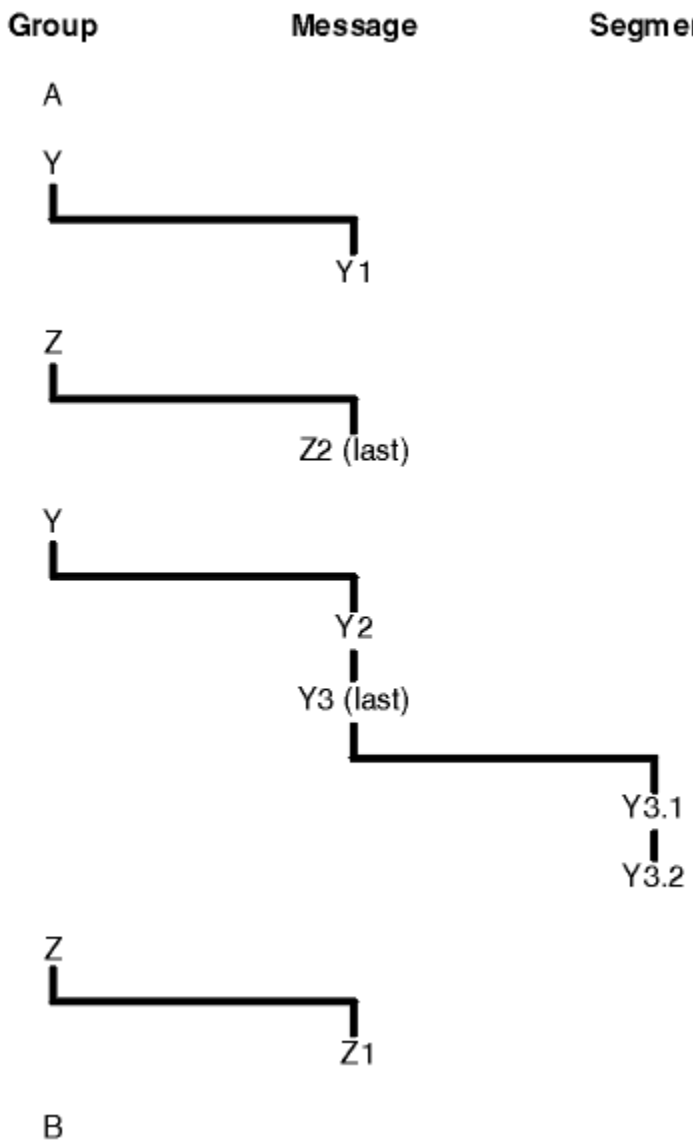


图 *67:* 队列上的物理顺序

这些消息以如下物理顺序在队列上显示:

1. 消息 A（不在组中）
2. 组 Y 的逻辑消息 1
3. 组 Z 的逻辑消息 2
4. 组 Y 的逻辑消息 2
5. 组 Y 的逻辑消息 3（最后一条消息）的段 1
6. 组 Y 的逻辑消息 3（最后一条消息）的段 2（最后一个段）
7. 组 Z 的逻辑消息 1
8. 消息 B（不在组中）

注: 在 IBM MQ for z/OS 上，如果按照 GROUPID 为队列建立索引，那么将不保证此队列上的消息的物理顺序。

获取消息时，可指定 MQGMO_LOGICAL_ORDER 以逻辑顺序（而不是物理顺序）来检索消息。

如果发出具有 MQGMO_BROWSE_FIRST 和 MQGMO_LOGICAL_ORDER 的 MQGET 调用，那么具有 MQGMO_BROWSE_NEXT 的后续 MQGET 调用也必须指定 MQGMO_LOGICAL_ORDER。 相反，如果具有

MQGMO_BROWSE_FIRST 的 MQGET 未指定 MQGMO_LOGICAL_ORDER，那么具有 MQGMO_BROWSE_NEXT 的以下 MQGET 也不能指定 MQGMO_LOGICAL_ORDER。

队列管理器为浏览队列上的消息的 MQGET 调用保留的组和段信息与队列管理器为从队列中移除消息的 MQGET 调用保留的组和段信息不同。 指定 MQGMO_BROWSE_FIRST 时，队列管理器会忽略要浏览的组和段信息并扫描队列，如同无当前组和当前逻辑消息一样。

注: 请勿在不指定 MQGMO_LOGICAL_ORDER 的情况下，使用 MQGET 调用在消息组（或不在组中的逻辑消息）末尾以外浏览。 例如，如果组中的最后一条消息在队列上的组中的第一条消息之前，那么使用 MQGMO_BROWSE_NEXT 在组末尾以外浏览，指定 MQMO_MATCH_MSG_SEQ_NUMBER 并将 *MsgSeqNumber* 设置为 1（以查找下一个组的第一条消息），将再次返回已浏览组中的第一条消息。 这可能会即时进行，或在多次 MQGET 调用之后进行（如存在中间组）。

两次打开队列进行浏览，避免无限循环的可能性：

- 使用第一个句柄来仅浏览每个组中的第一条消息。
- 使用第二个句柄来仅浏览特定组中的消息。
- 使用 MQMO_* 选项将第二个浏览光标移至第一个浏览光标的位置，然后浏览组中的消息。
- 请勿在组末尾以外使用 MQGMO_BROWSE_NEXT 浏览。

有关其更多信息，请参阅 MQGET、MQMD 和 验证 MQI 选项的规则。

对于大多数应用程序，浏览时您将可能选择逻辑排序或物理排序。 但是，如果要在这些方式之间切换，请记住在使用 MQGMO_LOGICAL_ORDER 第一次发出浏览时，将确定您在逻辑序列中的位置。

如果组内的第一个项此时不存在，那么不会将您所在的组视为逻辑序列的一部分。

一旦浏览光标在组内，便可在同一组内继续操作，即使移除第一条消息也是如此。 但是一开始，您绝不能使用 MQGMO_LOGICAL_ORDER 移至第一个项不存在的组中。

MQPMO_LOGICAL_ORDER

MQPMO 选项会告知队列管理器应用程序如何将消息放在逻辑消息的组和段中。 只能对 MQPUT 调用指定此选项；它对于 MQPUT1 调用是无效的。

如果指定 MQPMO_LOGICAL_ORDER，那么指示应用程序使用连续的 MQPUT 调用来执行以下操作：

1. 按段偏移量的递增顺序（从 0 开始，无间隔）放置每条逻辑消息中的段。
2. 先放置一条逻辑消息中的所有段，然后再放置下一条逻辑消息中的段。
3. 按消息序号的递增顺序（从 1 开始，无间隔）放置每个消息组中的逻辑消息。 IBM MQ 将自动递增消息序号。
4. 先放置一个消息组中的所有逻辑消息，然后再放置下一个消息组中的逻辑消息。

由于应用程序已告知队列管理器它如何将消息放在逻辑消息的组和段中，因此应用程序无需保留和更新有关每个 MQPUT 调用的组和段信息，因为队列管理器会保留和更新此信息。 特别地，这意味着应用程序无需在 MQMD 中设置 *GroupId*、*MsgSeqNumber* 和 *Offset* 字段，因为队列管理器将这些字段设置为相应的值。 应用程序必须仅在 MQMD 中设置 *MsgFlags* 字段，以指示消息何时属于组或成为逻辑消息段，并指示组中的最后一条消息或逻辑消息的最后一个段。

启动消息组或逻辑消息后，后续的 MQPUT 调用必须在 MQMD 中的 *MsgFlags* 中指定相应的 MQMF_* 标志。 如果应用程序尝试在存在未结束的消息组时放入不在组中的消息，或在存在未结束的逻辑消息时放入非段消息，那么调用将失败，并根据情况返回原因码 MQRC_INCOMPLETE_GROUP 或 MQRC_INCOMPLETE_MSG。 但是，队列管理器将保留有关当前消息组或当前逻辑消息的信息，应用程序可通过发送消息（可能无任何应用程序消息数据），根据情况指定 MQMF_LAST_MSG_IN_GROUP 或 MQMF_LAST_SEGMENT，然后重新发出 MQPUT 调用来放入不在组中或非段的消息来进行终止。

第 720 页的图 67 显示了有效的选项和标志组合，以及队列管理器在每种情况下使用的 *GroupId*、*MsgSeqNumber* 和 *Offset* 字段值。 此表中未显示的选项和标志组合无效。 此表中的列具有以下含义；“是”或“否”：

LOG ORD
是否在调用上指定 MQPMO_LOGICAL_ORDER 选项。

MIG
是否在调用上指定 MQMF_MSG_IN_GROUP 或 MQMF_LAST_MSG_IN_GROUP 选项。

SEG
是否在调用上指定 MQMF_SEGMENT 或 MQMF_LAST_SEGMENT 选项。

SEG OK
是否在调用上指定 MQMF_SEGMENTATION_ALLOWED 选项。

Cur grp
是否在调用前存在当前消息组。

Cur log msg
是否在调用前存在当前逻辑消息。

其他列
显示队列管理器使用的值。 上一项表示用于队列句柄的上一条消息中的字段的值。

表 *120:* 与逻辑消息的组和段中的消息相关的 *MQPUT* 选项

指定的选项	指定的选项	指定的选项	指定的选项	调用前的组和 log-msg 状态	调用前的组和 log-msg 状态	队列管理器使用的值	队列管理器使用的值	队列管理器使用的值
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	**GroupId**	**MsgSeqNumber**	**Offset**
Yes	否	否	否	否	否	MQGI_NONE	1	0
Yes	否	否	Yes	否	否	新组标识	1	0
Yes	否	Yes	任一	否	否	新组标识	1	0
Yes	否	Yes	任一	否	Yes	前一个组标识	1	前一个偏移量 + 前一个段长
Yes	Yes	任一	任一	否	否	新组标识	1	0
Yes	Yes	任一	任一	Yes	否	前一个组标识	前一个序列号 + 1	0
Yes	Yes	Yes	任一	Yes	Yes	前一个组标识	前一个序列号	前一个偏移量 + 前一个段长
否	否	否	否	任一	任一	MQGI_NONE	1	0
否	否	否	Yes	任一	任一	新组标识（如果指定 MQGI_NONE），否则为字段中的值	1	0
否	否	Yes	任一	任一	任一	新组标识（如果指定 MQGI_NONE），否则为字段中的值	1	字段中的值
否	Yes	否	任一	任一	任一	新组标识（如果指定 MQGI_NONE），否则为字段中的值	字段中的值	0
否	Yes	Yes	任一	任一	任一	新组标识（如果指定 MQGI_NONE），否则为字段中的值	字段中的值	字段中的值

注:

- MQPMO_LOGICAL_ORDER 在 MQPUT1 调用上无效。
- 对于 *MsgId* 字段，如果已指定 MQPMO_NEW_MSG_ID 或 MQMI_NONE，那么队列管理器将生成新的消息标识，如果未指定，那么使用该字段中的值。
- 对于 *CorrelId* 字段，如果已指定 MQPMO_NEW_CORREL_ID，那么队列管理器将生成新的相关标识，如果未指定，那么使用该字段中的值。

指定 MQPMO_LOGICAL_ORDER 时，队列管理器要求使用 MQMD 中的 *Persistence* 字段中的相同值，放入逻辑消息中的组和段中的所有消息,也就是说，所有项都必须是持久性的，或必须是非持久性的。 如果不满足此条件，那么 MQPUT 调用将失败，显示原因码 MQRC_INCONSISTENT_PERSISTENCE。

MQPMO_LOGICAL_ORDER 选项会影响工作单元，如下所述：

- 如果将组或逻辑消息中的第一条物理消息放入一个工作单元内，那么该组或逻辑消息中的所有其他物理消息都必须放入一个工作单元内（如果使用相同的队列句柄）。 但是，无需将它们放入同一工作单元内，这表示允许包含多条物理消息的消息组或逻辑消息拆分到队列句柄的两个或更多连续工作单元中。
- 如果未将组或逻辑消息中的第一条物理消息放入一个工作单元内，那么该组或逻辑消息中的所有其他物理消息都不能放入一个工作单元内（如果使用相同的队列句柄）。

如果不满足这些条件，那么 MQPUT 调用将失败，显示原因码 MQRC_INCONSISTENT_UOW。

指定 MQPMO_LOGICAL_ORDER 时，MQPUT 调用上提供的 MQMD 不得低于 MQMD_VERSION_2。 如果不满足此条件，那么调用将失败，显示原因码 MQRC_WRONG_MD_VERSION。

如未指定 MQPMO_LOGICAL_ORDER，那么可以任何顺序放入逻辑消息的组和段中的消息，并且无需放入完整的消息组或完整的逻辑消息。 应用程序应确保 *GroupId*、*MsgSeqNumber*、*Offset* 和 *MsgFlags* 字段具有相应的值。

系统故障发生后，请使用此方法在此期间重新启动消息组或逻辑消息。 系统重新启动时，应用程序可将 *GroupId*、*MsgSeqNumber*、*Offset*、*MsgFlags* 和 *Persistence* 字段设置为相应的值，然后根据需要发出设置了 MQPMO_SYNCPOINT 或 MQPMO_NO_SYNCPOINT 的 MQPUT 调用，但不指定 MQPMO_LOGICAL_ORDER。 如果成功进行此调用，队列管理器将保留组和段信息，并且使用此队列句柄的后续 MQPUT 调用可正常指定 MQPMO_LOGICAL_ORDER。

队列管理器为 MQPUT 调用保留的组和段信息与其为 MQGET 调用保留的组和段信息不同。

对于任意给定的队列句柄，应用程序可混合指定了 MQPMO_LOGICAL_ORDER 的 MQPUT 调用和未指定 MQPMO_LOGICAL_ORDER 的 MQPUT 调用，但请注意以下几点：

- 如果未指定 MQPMO_LOGICAL_ORDER，那么每个成功的 MQPUT 调用都会使队列管理器将队列句柄的组和段信息设置为应用程序指定的值，替换队列管理器为队列句柄保留的现有组和段信息。
- 如果未指定 MQPMO_LOGICAL_ORDER，那么在存在当前消息组或逻辑消息的情况下，调用不会失败；可能会成功调用，显示完成代码 MQCC_WARNING。 第 723 页的表 121 显示可能出现的各种情况。 在这些情况下，如果完成代码不是 MQCC_OK，那么原因码为以下项之一（视情况而定）：
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_PERSISTENCE
  - MQRC_INCONSISTENT_UOW

  注: 队列管理器未对 MQPUT1 调用检查组和段信息。

表 *121: MQPUT* 或 *MQCLOSE* 调用与组和段信息不一致时的结果

当前调用是	前一调用是 MQPUT，使用 MQPMO_LOGICAL_ORDER	前一调用是 MQPUT，不使用 MQPMO_LOGICAL_ORDER
MQPUT，使用 MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED

表 121: *MQPUT* 或 *MQCLOSE* 调用与组和段信息不一致时的结果 (继续)		
当前调用是	**前一调用是 MQPUT，使用 MQPMO_LOGICAL_ORDER**	**前一调用是 MQPUT，不使用 MQPMO_LOGICAL_ORDER**
MQPUT，不使用 MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE，使用未结束的组或逻辑消息	MQCC_WARNING	MQCC_OK

对于以逻辑顺序放入消息和段的应用程序，指定 MQPMO_LOGICAL_ORDER，因为它是可使用的最简单的选项。 该选项可使应用程序无需管理组和段信息，因为队列管理器会管理此信息。 但是，特殊的应用程序可能需要较之 MQPMO_LOGICAL_ORDER 选项提供的功能更大的控制能力，这可通过不指定该选项来实现；如果不指定，那么必须确保正确设置 MQMD 中的 *GroupId*、*MsgSeqNumber*、*Offset* 和 *MsgFlags* 字段，然后再进行每个 MQPUT 或 MQPUT1 调用。

例如，要转发所接收的物理消息（而不考虑这些消息是否在逻辑消息的组或段中）的应用程序，不得指定 MQPMO_LOGICAL_ORDER，原因有二：

- 如果按顺序检索和放入消息，那么指定 MQPMO_LOGICAL_ORDER 会将新组标识分配给消息，这会使消息发起方很难或无法关联由消息组生成的任何回复或报告消息。
- 在发送和接收队列管理器间具有多条路径的复杂网络中，物理消息到达时可能杂乱无序。 通过在 MQGET 调用上不指定 MQPMO_LOGICAL_ORDER 和 MQGMO_LOGICAL_ORDER，转发应用程序可在每条物理消息到达时立即对其进行检索和转发，而无需等待下一条采用逻辑顺序的消息到达。

为逻辑消息的组或段中的消息生成报告消息的应用程序也不得在放入报告消息时指定 MQPMO_LOGICAL_ORDER。

可以使用其他 MQPMO_* 选项中的任一选项来指定 MQPMO_LOGICAL_ORDER。

将逻辑排序的组放入集群队列 (MQOO_BIND_ON_GROUP)

MQOO_BIND_ON_OPEN 选项确保将此应用程序的所有消息（所有组）路由至单个实例。 它的缺点是，应用程序流量在集群队列的多个实例间未均衡负载。 为了在保持消息组完整的同时启用工作负载均衡，您必须设置以下选项：

- MQPUT 调用必须指定 MQPMO_LOGICAL_ORDER
- MQOPEN 调用必须指定以下两个选项之一：
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF，队列定义必须指定 DEFBIND(GROUP)

随后会在消息的组间驱动工作负载均衡，无需队列的 MQCLOSE 和 MQOPEN。 组间意味着在 MQMD(v2) 或 MQMDE 中设置 MQMF_MSG_IN_GROUP，没有正在进行的未完成组。 当组正在进行中时，将复用解析的队列管理器和对象句柄中的队列名称。

如果先前的消息是 MQPMO_LOGICAL_ORDER 且/或设置了 MQMF_MSG_IN_GROUP，但当前消息不是组的一部分，那么 PUT 调用将失败并显示 MQRC_INCOMPLETE_GROUP。

如果单个 MQPUT 未指定 MQPMO_LOGICAL_ORDER，并且没有处于活动状态的当前组，那么将对此消息驱动工作负载均衡（如同已对 MQOPEN 调用指定了 MQOO_BIND_NOT_FIXED 一样）。

不会对使用 MQOO_BIND_ON_GROUP 绑定到目标的消息进行重新分配。 有关重新分配的更多信息，请参阅第 36 页的『消息组』。

对逻辑消息分组

在组中使用逻辑消息有两个主要原因：

- 不会对使用 MQOO_BIND_ON_GROUP 绑定到。
- 您可能需要以相关方式处理组中的每一条消息。

在任一情况下，均使用相同的获取应用程序实例来检索整个组。

例如，假定组包含四条逻辑消息。 放入应用程序类似如下：

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

获取应用程序为组中的第一条消息指定 MQGMO_ALL_MSGS_AVAILABLE 选项。 这将确保直到组内的所有消息到达后才开始处理。 将为组内的后续消息忽略 MQGMO_ALL_MSGS_AVAILABLE 选项。

检索组的第一条逻辑消息时，可使用 MQGMO_LOGICAL_ORDER 来确保按顺序检索该组的其余逻辑消息。

因此，获取应用程序类似如下：

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

要获取有关消息分组的更多示例，请参阅第 735 页的『逻辑消息的应用程序分段』和第 725 页的『放入和取出覆盖多个工作单元的组』。

有关允许应用程序请求向集群队列的目标实例分配全部一组消息的信息，请参阅 DefBind。

放入和取出覆盖多个工作单元的组

在上一情况中，只有在放入整个组并落实工作单元后，才能使消息或段开始脱离节点（如果其目标为远程目标）或开始对其进行检索。 如果放入整个组的时间较长，或节点上的队列空间受到限制，那么这可能不是您希望看到的情况。 要解决这一问题，请将组放入多个工作单元中。

如果在多个工作单元中放入组，那么即使放入应用程序失败，也能落实其中一部分组。 因此，应用程序必须保存状态信息，使用每个工作单元来落实，重新启动后可用它来恢复不完整的组。 记录此信息最简单的位置是在 STATUS 队列中。 如果已成功放入完整的组，那么 STATUS 队列将为空。

如果涉及分段，那么逻辑是相似的。 在这种情况下，**StatusInfo** 必须包含 *Offset*。

以下是将组放入多个工作单元的示例：

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
```

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

如果已落实所有工作单元，那么已成功放入整个组，并且 STATUS 队列为空。 如果未落实，那么必须在状态信息所指示的点上恢复组。 MQPMO_LOGICAL_ORDER 无法用于第一次放入操作，但可用于之后的放入操作。

重新启动处理如下所示：

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

在整个组到达之前，您可能希望从获取应用程序开始处理组中的消息。 这将缩短组中消息的响应时间，也意味着整个组无需存储。 为了实现这些优势，请对每一组的消息使用多个工作单元。 出于恢复原因，必须检索一个工作单元内的每一条消息。

同相应的放入应用程序一样，这需要在落实每一个工作单元时，将状态信息自动记录到某个地点。 同样，记录此信息最简单的位置是在 STATUS 队列上。 如果已成功处理完整的组，那么 STATUS 队列将为空。

注: 对于中间工作单元，可通过指定对状态队列的每个 MQPUT 均为消息段（也即，通过设置 MQMF_SEGMENT 标志），而不为每个工作单元放入完整的新消息，来避免从 STATUS 队列调用 MQGET。 在最后一个工作单元中，最终的段将放入指定 MQMF_LAST_SEGMENT 的状态队列中，然后使用指定了 MQGMO_COMPLETE_MSG 的 MQGET 清除状态信息。

重新启动处理期间，请浏览具有 MQGMO_LOGICAL_ORDER 的状态队列，直至到达最后一个段（也即，直至不返回任何更多的段为止），而不要使用单个 MQGET 来获取可能存在的状态消息。 在重新启动后的第一个工作单元中，也需要明确指定放入状态段时的偏移量。

在以下示例中，仅考虑了某一组内的消息，假定应用程序缓冲区始终都足够大，能够容纳整个消息（无论消息是否分段）。 因此，在每个 MQGET 上均指定了 MQGMO_COMPLETE_MSG。 涉及分段时将应用相同的原则（在这种情况下，StatusInfo 必须包含 *Offset*）。

为简便起见，假定在一个工作单元中最多检索 4 条消息：

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
```

```
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

如果已落实所有工作单元，那么已成功检索整个组，并且 STATUS 队列为空。 如果未落实，那么必须在状态信息所指示的点上恢复组。 MQGMO_LOGICAL_ORDER 无法用于第一次检索操作，但可用于之后的放入操作。

重新启动处理如下所示：

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   /* The next message on the group must be retrieved by matching
      the sequence number and group ID with those retrieved from the
      status information. */
   GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
   MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
         MQMD.GroupId      = value from Status message,
         MQMD.MsgSeqNumber = value from Status message plus 1
   msgs = 1
   /* Process this message */
   ...

   /* Now normal processing is resumed */
   /* Retrieve remaining messages in the group */
   do while ( GroupStatus == MQGS_MSG_IN_GROUP )

      /* Process up to 4 messages in the group */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                  | MQGMO_LOGICAL_ORDER
      do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
         MQGET
         msgs = msgs + 1
         /* Process this message */
         ...

      /* Have retrieved last message or 4 messages  */
      /* Update status message if not last in group */
      MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
      if ( GroupStatus == MQGS_MSG_IN_GROUP )
         StatusInfo = GroupId,MsgSeqNumber from MQMD
         MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
      MQCMIT
      msgs = 0
```

获取特定消息

可通过多种方法从队列中获取特定消息。 包括选择 MsgId 和 CorrelId、选择 GroupId、MsgSeqNumber 和 Offset 以及选择 MsgToken。 还可以在打开队列时使用选择字符串。

要从队列获取特定消息，请使用 MQMD 结构的 MsgId 和 CorrelId 字段。 但是，应用程序可明确设置这些字段，因此指定的值可能不能识别唯一的消息。 第 727 页的表 122 显示了为这些字段的可能的设置检索哪一条消息。 如果在 MQGET 调用的 **GetMsgOpts** 参数中指定 MQGMO_MSG_UNDER_CURSOR，那么输入上将忽略这些字段。

表 *122:* 使用消息和相关标识

要检索...	**MsgId**	**CorrelId**
队列中的第一条消息	MQMI_NONE	MQCI_NONE
与 *MsgId* 匹配的第一条消息	非零	MQCI_NONE
与 *CorrelId* 匹配的第一条消息	MQMI_NONE	非零

表 122: 使用消息和相关标识 (继续)		
要检索...	**MsgId**	**CorrelId**
与 *MsgId* 和 *CorrelId* 匹配的第一条消息	非零	非零

在每种情况下，第一条均指符合选择标准的第一条消息（除非指定 MQGMO_BROWSE_NEXT，此时指序列中符合选择标准的下一条消息）。

返回时，MQGET 调用会将 *MsgId* 和 *CorrelId* 字段设置为所返回的消息的消息和相关标识（如有）。

如果将 MQMD 结构的 *Version* 字段设置为 2，那么可使用 *GroupId*、*MsgSeqNumber* 和 *Offset* 字段。第 728 页的表 123 显示了为这些字段的可能的设置检索哪一条消息。

表 123: 使用组标识	
要检索...	**匹配选项**
队列中的第一条消息	MQMO_NONE
与 MsgId 匹配的第一条消息	MQMO_MATCH_MSG_ID
与 CorrelId 匹配的第一条消息	MQMO_MATCH_CORREL_ID
与 GroupId 匹配的第一条消息	MQMO_MATCH_GROUP_ID
与 MsgSeqNumber 匹配的第一条消息	MQMO_MATCH_MSG_SEQ_NUMBER
与 MsgToken 匹配的第一条消息	MQMO_MATCH_MSG_TOKEN
与 Offset 匹配的第一条消息	MQMO_MATCH_OFFSET

注意:

1. MQMO_MATCH_XXX 意味着将 MQMD 结构中的 *XXX* 字段设置为要匹配的值。
2. 可组合使用 MQMO 标志。 例如，可同时使用 MQMO_MATCH_GROUP_ID、MQMO_MATCH_MSG_SEQ_NUMBER 和 MQMO_MATCH_OFFSET 提供由 GroupId、MsgSeqNumber 和 Offset 字段标识的段。
3. 如果指定 MQGMO_LOGICAL_ORDER，那么您尝试检索的消息将受到影响，因为选项取决于为队列句柄控制的状态信息。 相关信息请参阅第 718 页的『逻辑与物理排序』和选项。

MQGET 调用通常检索来自队列的第一条消息。 如果在使用 MQGET 调用时指定特定消息，那么队列管理器必须搜索队列直至找到该消息。 这可影响应用程序的性能。

如果使用 V2 或更高版本的 MQGMO 结构，并且未指定 MQMO_MATCH_MSG_ID 或 MQMO_MATCH_CORREL_ID 标志，那么无需在 MQGET 间重置 MsgId 或 CorrelId 字段。

z/OS 在 IBM MQ for z/OS 上，可使用队列属性 IndexType，来提高队列上 MQGET 操作的速度。有关更多信息，请参阅第 732 页的『索引类型』。

可通过在 MQGMO 结构中指定消息的 MsgToken 和 MatchOption MQMO_MATCH_MSG_TOKEN，从队列中获取特定的消息。 除非重新启动队列管理器，否则 MsgToken 将由最初将此消息放入队列的 MQPUT 调用或先前的 MQGET 操作返回并保持不变。

如果您仅对队列上的一部分消息感兴趣，可通过将选择字符串用于 MQOPEN 或 MQSUB 调用，来指定要处理的消息。 然后，MQGET 会检索满足选择字符串要求的下一条消息。 有关选择字符串的更多信息，请参阅第 25 页的『选择器』。

提高非持久消息的性能

当客户机需要从服务器获取消息时，它会将请求发送到该服务器。 它为要使用的每条消息单独发送一个请求。 要通过避免必须发送这些请求消息来提高使用非持久消息的客户机的性能，可以将客户机配置为使用预读。 预读允许将消息发送到客户机，而不必使应用程序请求这些消息。

启用预读时，系统会将消息发送到名为预读缓冲区的客户机上的内存缓冲区。 在预读已启用的情况下，客户机将为它打开的每个队列都提供一个预读缓冲区。 预读缓冲区中的消息为非持久消息。 客户机会定期为服务器更新有关它已使用的数据量的信息。

在通过 MQOO_READ_AHEAD 调用 MQOPEN 时，IBM MQ 客户机仅在满足某些条件时才会启用预读。 这些条件包括：

- 客户机和远程队列管理器都必须处于 IBM WebSphere MQ 7.0 或更高版本。
- 必须针对线程化的 IBM MQ MQI 客户机库编译和链接客户机应用程序。
- 客户机通道必须使用的是 TCP/IP 协议
- 该通道必须在客户机和服务器通道定义中具有非零 SharingConversations (SHARECNV) 设置。

在通过客户机应用程序使用非持久消息时，使用预读可提高性能。 此性能提升适用于 MQI 和 JMS 应用程序。 使用 MQGET 或异步使用的客户机应用程序将在使用非持久消息时从性能提升中获益。

并非所有客户机应用程序设计都适合使用预读，这是因为并非支持所有选项与预读配合使用，当预读已启用时，部分选项需要在 MQGET 调用之间保持一致。 如果客户机在 MQGET 调用之间改变其选择标准，那么存储在预读缓冲区中的消息将滞留在客户机预读缓冲区中。

如果不再需要具有先前选择标准的滞留消息集合，可在客户机上设置可配置清除时间间隔，以便从客户机中自动清除这些消息。 清除时间间隔是由客户机确定的一组预读调整选项之一。 可调整这些选项以符合您的要求。

如果重新启动客户机应用程序，那么预读缓冲区中的消息可能会丢失。 相反，可能会从底层队列中删除移至预读缓冲区的消息；这不会导致从缓冲区中除去消息，因此使用预读功能的 MQGET 调用可能会返回不再存在的消息。

仅针对客户机绑定执行预读。 会对所有其他绑定忽略此属性。

预读对触发没有任何影响。 消息由客户机预读时，将不会生成任何触发器消息。 启用预读时，将不会生成会计和统计信息。

将预读与发布预订消息传递一起使用

预订应用程序指定向其发送发布的目标队列时，所指定的队列的 DEFREADA 值将用作缺省预读值。

预订应用程序请求 IBM MQ 管理向其发送发布的目标时，将根据预定义模型队列创建受管队列作为动态队列。 它是用作缺省预读值的模型队列的 DEFREADA 值。 将使用缺省模型队列 SYSTEM.DURABLE.PUBLICATIONS.MODEL 或 SYSTEM.NONDURABLE.PUBLICATIONS.MODEL，除非为此主题或某个父主题定义了模型队列。

相关概念

第 731 页的『为 AIX 上的非持久消息调整性能』
如果使用 AIX V5.3 或更高版本，请考虑设置调整参数，以使用非持久消息的完整性能。

相关任务

第 730 页的『启用和禁用预读』
缺省情况下，禁用预读。 您可以在队列或应用程序级别启用预读。

相关参考

第 729 页的『MQGET 选项和预读』
启用预读时，并非支持所有 MQGET 选项；部分选项需要在 MQGET 调用之间保持一致。

MQGET 选项和预读
启用预读时，并非支持所有 MQGET 选项；部分选项需要在 MQGET 调用之间保持一致。

在通过 MQOO_READ_AHEAD 调用 MQOPEN 时，IBM MQ 客户机仅在满足某些条件时才会启用预读。 这些条件包括：

- 客户机和远程队列管理器都必须处于 IBM WebSphere MQ 7.0 或更高版本。
- 必须针对线程化的 IBM MQ MQI 客户机库编译和链接客户机应用程序。
- 客户机通道必须使用的是 TCP/IP 协议

- 该通道必须在客户机和服务器通道定义中具有非零 SharingConversations (SHARECNV) 设置。

下表指出支持哪些选项用于预读，以及是否能够在 MQGET 调用之间更改这些选项。

表 *124: MQGET* 选项和预读

MQGET 值和选项	**启用预读时允许，并可在 MQGET 调用之间更改 [5]**	**启用预读时允许，但不能在 MQGET 调用之间更改 [1]**	**启用预读时不允许使用的 MQGET 选项 [2]**
MQGET MQMD 值	MsgId [3] CorrelId [3]	Encoding CodedCharSetId	
MQGET MQGMO 选项	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED_M SG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR [4] • MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

注释:

1. 如果在 MQGET 调用之间更改了这些选项，那么将返回 MQRC_OPTIONS_CHANGED 原因码。
2. 如果在第一个 MQGET 调用上指定这些选项，那么将禁用预读。 如果在后续 MQGET 调用上指定这些选项，那么将返回原因码 MQRC_OPTIONS_ERROR。
3. 如果客户机应用程序在 MQGET 调用之间更改了 MsgId 和 CorrelId 值，那么具有先前值的消息可能已发送至客户机，并将保留在客户机预读缓冲区中，直至被使用（或自动清除）为止。
4. MQGMO_MSG_UNDER_CURSOR 不能与预读配合使用。 当打开队列时指定了 MQOO_BROWSE 以及 MQOO_INPUT_SHARED 或 MQOO_INPUT_EXCLUSIVE 选项的其中一个选项时，将禁用预读。
5. 启用预读时，第一个 MQGET 可确定是否要从队列中浏览或获取消息。 如果客户机应用程序随后使用更改了选项的 MQGET（例如尝试在初次获取后浏览，或尝试在初次浏览后获取），那么将返回 MQRC_OPTIONS_CHANGED 原因码。

如果客户机在 MQGET 调用之间更改其选择标准，那么与初始选择标准匹配且存储在预读缓冲区中的消息不会由客户机应用程序使用，并滞留在客户机预读缓冲区中。 在客户机预读缓冲区包含多条滞留消息的情况下，预读的优势将丢失，并且使用每条消息均需要单独请求服务器。 要确定是否正在有效使用预读，可使用连接状态参数 READA。

如果第一个 MQGET 调用上指定了不兼容的选项，那么应用程序可请求禁止预读。 在此情况下，连接状态会显示预读被禁止。

如果由于 MQGET 上存在这些限制您确定客户机应用程序设计不适合于预读，请指定 MQOPEN 选项 MQOO_READ_AHEAD_NO。 或者，将打开的队列的缺省预读值变更为 NO 或 DISABLED。

启用和禁用预读
缺省情况下，禁用预读。 您可以在队列或应用程序级别启用预读。

关于此任务

在通过 MQOO_READ_AHEAD 调用 MQOPEN 时，IBM MQ 客户机仅在满足某些条件时才会启用预读。 这些条件包括:

- 客户机和远程队列管理器都必须处于 IBM WebSphere MQ 7.0 或更高版本。
- 必须针对线程化的 IBM MQ MQI 客户机库编译和链接客户机应用程序。
- 客户机通道必须使用的是 TCP/IP 协议

- 该通道必须在客户机和服务器通道定义中具有非零 SharingConversations (SHARECNV) 设置。

要启用预读：

- 要在队列级别配置预读，请将队列属性 DEFREADA 设置为 YES。
- 要在应用程序级别配置预读：
  - 要在尽可能的情况下使用预读，请在调用 MQOPEN 函数时使用 MQOO_READ_AHEAD 选项。 如果已将 DEFREADA 队列属性设置为 DISABLED，那么客户机应用程序将无法使用预读。
  - 要仅在队列上启用预读时使用预读，请在调用 MQOPEN 函数时使用 MQOO_READ_AHEAD_AS_Q_DEF 选项。

如果客户机应用程序设计不适用于预读，可通过以下方式禁用它：

- 如果您不想使用预读（除非客户机应用程序请求它），可在队列级别将队列属性 DEFREADA 设置为 NO；或者如果您不想使用预读（无论客户机应用程序是否请求预读），可在队列级别将队列属性 DEFREADA 设置为 DISABLED。
- 在应用程序级别，在调用 MQOPEN 函数时使用 MQOO_NO_READ_AHEAD 选项。

如果队列已关闭，存在两个 MQCLOSE 选项，使用这两个选项可配置存储在预读缓冲区中的任何消息发生的情况。

- 使用 MQCO_IMMEDIATE 来废弃预读缓冲区中的消息。
- 使用 MQCO_QUIESCE 来确保应用程序在队列关闭前使用预读缓冲区中的消息。 当发出具有 MQCO_QUIESCE 的 MQCLOSE，并且预读缓冲区上留有消息时，MQRC_READ_AHEAD_MSGS 将返回 MQCC_WARNING。

为 *AIX* 上的非持久消息调整性能
如果使用 AIX V5.3 或更高版本，请考虑设置调整参数，以使用非持久消息的完整性能。

要设置调整参数以便即时生效，可以 root 用户身份发出以下命令：

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

要设置调整参数以便即时生效并使该设置在重新引导后持续有效，可以 root 用户身份发出以下命令：

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

通常，仅在内存中保存非持久消息，但在一些情况下，AIX 可安排将非持久消息写入磁盘。 在写入磁盘完成之前，已安排要写入磁盘的消息对于 MQGET 是不可用的。 建议的调整命令改变该阈值；只有在机器上的实存储器几乎快满时才发生写入磁盘事件，而不是在有 16 千字节的数据排队时，安排将消息写入磁盘。 这是全局性的改变，可能会影响其他软件组件。

在 AIX 上，当使用多线程应用程序时（尤其是在具有多个处理器的机器上运行时），我们强烈建议在 mqm ID `.profile` 中设置 AIXTHREAD_SCOPE=S 或在启动应用程序之前在环境中设置 AIXTHREAD_SCOPE=S，以便获得更好的性能和更可靠的调度。 例如：

```
export AIXTHREAD_SCOPE=S
```

设置 AIXTHREAD_SCOPE=S 意味着使用缺省属性创建的用户线程位于系统范围内的争用作用域中。 如果用户线程创建时处于系统范围内的争用作用域，它将绑定到内核线程并由该内核进行安排。 底层内核线程不与任何其他用户线程共享。

文件描述符

运行诸如代理程序进程的多线程进程时，您可能会达到文件描述符的软限制。 此限制会为您提供 IBM MQ 原因码 MQRC_UNEXPECTED_ERROR (2195)；如果有足够的文件描述符，则会提供一个 IBM MQ FFST™ 文件。

要避免此问题，可以增加文件描述符数目的进程限制。 为此，请为 mqm 用户标识或在缺省节中，将 `/etc/security/limits` 中的 `nofiles` 属性更改为 10,000。

系统资源限制

在命令提示符中使用下列命令将数据段和堆栈段的系统资源限制设置为无限制：

```
ulimit -d unlimited
ulimit -s unlimited
```

索引类型

队列属性 *IndexType* 指定队列管理器为了加快队列上 MQGET 操作速度而保留的索引类型。

注： 仅在 IBM MQ for z/OS 上受支持。

您有五个选项：

值	描述
NONE	没有维护索引。 在按顺序检索消息时使用此值（请参阅第 718 页的『优先级』）。
GROUPID	维护组标识的索引。 如果要对消息组进行逻辑排序，必须使用此索引类型（请参阅第 718 页的『逻辑与物理排序』）。
MSGID	维护消息标识的索引。 当使用 *MsgId* 字段作为 MQGET 调用上的选择标准来检索消息时，请使用此值（请参阅第 727 页的『获取特定消息』）。
MSGTOKEN	维护消息令牌的索引。
CORRELID	维护相关标识的索引。 当使用 *CorrelId* 字段作为 MQGET 调用上的选择标准来检索消息时，请使用此值（请参阅第 727 页的『获取特定消息』）。

注：

1. 如果您正在使用 MSGID 选项或 CORRELID 选项建立索引，请设置 MQMD 中的相关 **MsgId** 或 **CorrelId** 参数。 同时设置二者并不会获益。
2. 如果队列与以下所有条件相匹配，那么浏览将使用索引机制来查找消息：
   - 它具有索引类型 MSGID、CORRELID 或 GROUPID
   - 使用相同类型标识浏览它
   - 它具有仅有一个优先级的消息
3. 避免包含数千条消息的队列（通过 *MsgId* 或 *CorrelId* 建立索引），因为这将影响重新启动时间。 （这不适用于非持久消息，因为在重新启动时将删除非持久消息。）
4. MSGTOKEN 用于定义由 z/OS 工作负载管理器管理的队列。

有关 **IndexType** 属性的完整描述，请参阅 IndexType。 有关 **IndexType** 属性的更多信息，请参阅第 53 页的『z/OS 应用程序的设计和性能注意事项』。

处理消息长度大于 *4 MB* 的消息

消息对于应用程序、队列或队列管理器可能过大。 根据不同的环境，IBM MQ 会提供多种方式来处理长度大于 4 MB 的消息。

在 V6 或更高版本的所有 IBM MQ 系统上，可以将 **MaxMsgLength** 属性增加到 100 MB。 请设置此值，以反映使用队列的消息的大小。 在 IBM MQ for z/OS 以外的 IBM MQ 系统上，还可以执行以下操作：

1. 使用分段消息。 （消息可由应用程序或队列管理器分段。）
2. 使用参考消息。

本节其余部分描述了这些方法中的每一种方法。

增加最大消息长度

MaxMsgLength 队列管理器属性定义可由队列管理器处理的消息的最大长度。 类似地，**MaxMsgLength** 队列属性是可由队列处理的消息的最大长度。 所支持的缺省最大消息长度取决于您在运行的环境。

如果您正在处理大型消息，那么可独立地在 z/OS 之外的平台上更改这些属性。 可设置在 32768 字节到 100 MB 范围内的队列管理器属性值。

注意: 在 IBM MQ for z/OS 上，队列管理器的 **MaxMsgLength** 属性采用 100 MB 硬编码。

在所有平台上，您可以设置 0 到 100 MB 范围内的队列属性值。

更改一个或两个 **MaxMsgLength** 属性后，重新启动应用程序和通道以确保更改生效。

做出这些更改时，消息长度必须小于或等于队列和队列管理器 **MaxMsgLength** 属性。 然而，现有消息可能比任何一个属性都长。

如果消息对于队列而言过大，那么将返回 MQRC_MSG_TOO_BIG_FOR_Q。 类似地，如果消息对于队列管理器而言过大，那么将返回 MQRC_MSG_TOO_BIG_FOR_Q_MGR。

处理大型消息的这一方法非常简单和方便。 但是，在使用前应考虑以下因素：

- 降低队列管理器间的一致性。 消息数据的最大大小由消息所处的每个队列（包含传输队列）的 *MaxMsgLength* 确定。 缺省情况下，该值通常为队列管理器的 *MaxMsgLength*，对于传输队列尤其如此。 这使得您很难预测消息在传输到远程队列管理器时是否过大。
- 增加使用系统资源。 例如，应用程序需要更大的缓冲区，在一些平台上，可能会增加使用共享存储器。 只有在大型消息实际需要时，才会影响队列存储器。
- 将影响通道分批。 大型消息在批次计数中仍仅视为一条消息，但需要更长的传输时间，因此会增加其他消息的响应时间。

Multi 消息分段

使用此信息来了解消息分段。 在 IBM MQ for z/OS 中或使用 IBM MQ classes for JMS 的应用程序中不支持此功能。

按第 732 页的『增加最大消息长度』主题中所述增加最大消息长度具有一些负面影响。 同样，仍可能导致消息对于队列或队列管理器过大。 在这些情况下，可以对消息进行分段。 有关分段的信息，请参阅第 36 页的『消息组』。

以下几部分讨论了消息分段的常见使用情况。 对于放入和破坏性获取，会假定 MQPUT 或 MQGET 调用始终在一个工作单元中进行。 请始终考虑使用此方法来降低网络中存在不完整组的可能性。 将假定按队列管理器进行单阶段落实，但其他协调方法同等有效。

此外，在获取应用程序中，将假定如果多台服务器正在处理同一个队列，那么每台服务器都会执行类似的代码，以便一台服务器始终能够找到其预期使用的消息或分段（因为它已在先前指定了 MQGMO_ALL_MSGS_AVAILABLE 或 MQGMO_ALL_SEGMENTS_AVAILABLE）。

放入和获取跨多个工作单元的分段消息

您可以按第 725 页的『放入和取出覆盖多个工作单元的组』中类似的方式放入和获取跨一个工作单元的分段消息。

但是，您无法在全局工作单元中放入或获取分段消息。

Multi 按队列管理器分段和组装

这是最简单的场景，在此场景中一个应用程序会放入将由另一个应用程序检索的消息。 消息可能会很大：对于要在单个缓冲区中处理消息的放入或获取应用程序不太大，但对于要放入消息的队列管理器或队列则过大。

这些应用程序所需的唯一更改是在需要时，使放入应用程序授权队列管理器执行分段：

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

使获取应用程序在已对消息分段时，请求队列管理器重新汇编消息：

```
GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
MQGET
```

在这一最简单的场景中，应用程序必须在调用 MQPUT 前将 GroupId 字段重置为 MQGI_NONE，以便队列管理器能够为每条消息生成唯一的组标识。 如果没有这样做，那么无关的消息可能具有相同的组标识，这可能会导致后续处理不正确。

应用程序缓冲区必须足够大，能够包含重新汇编的消息（除非包含 MQGMO_ACCEPT_TRUNCATED_MSG 选项）。

如果将修改队列的 MAXMSGLEN 属性以支持消息分段，请考虑：

- 本地队列上支持的最小消息段为 16 个字节。
- 对于传输队列，MAXMSGLEN 还必须包含头所需的空间。 考虑在可以放在传输队列上的任何消息段中，使用至少大于用户数据的最大预期长度 4000 个字节的值。

如果需要进行数据转换，那么获取应用程序可能必须要通过指定 MQGMO_CONVERT 来完成转换。 这一过程应当是简单的，因为完整的消息提供了数据转换出口。 如果将消息分段，并且数据格式导致数据转换出口无法对不完整数据执行转换，那么请勿尝试在发送方通道中转换数据。

Multi 应用程序分段

如果队列管理器分段不足以满足需求，或当应用程序需要在特定的分段范围内进行数据转换时，将使用应用程序分段。

使用应用程序分段主要有以下两方面原因：

1. 仅使用队列管理器分段不足以满足需求，因为消息过大，无法由应用程序在单个缓冲区中处理。
2. 数据转换必须由发送方通道执行，并且格式要求放入应用程序必须对段范围做出规定，以便能够转换单个段。

但是，如果数据转换不是问题，或如果获取应用程序始终使用 MQGMO_COMPLETE_MSG，那么还可以通过指定 MQMF_SEGMENTATION_ALLOWED 来允许队列管理器分段。 在我们的示例中，应用程序将消息分为四段：

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

如果未使用 MQPMO_LOGICAL_ORDER，那么应用程序必须设置每个段的 *Offset* 和长度。 在这种情况下，系统不会自动保持逻辑状态。

获取应用程序无法保证具有足够大的缓冲区来容纳所有重新汇编消息。 因此，必须精心准备以单独处理各段。

对于分段消息，应用程序不希望在构成逻辑消息的所有段均存在时才开始处理某个段。 因此，为第一个段指定 MQGMO_ALL_SEGMENTS_AVAILABLE。 如果指定 MQGMO_LOGICAL_ORDER，并且有一条当前的逻辑消息，那么将忽略 MQGMO_ALL_SEGMENTS_AVAILABLE。

在检索逻辑消息的第一个段后，使用 MQGMO_LOGICAL_ORDER 来确保按顺序检索逻辑消息的其余各段。

未考虑不同组中的消息。 如果出现此类消息，将按照每一条消息的第一个段在队列上出现的顺序对其进行处理。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...
```

```
MQCMIT
```

Multi逻辑消息的应用程序分段
这些消息必须按逻辑顺序成组进行维护，由于其中部分或全部消息可能非常大，因此需要使用应用程序分段。

在我们的示例中，将放入一组四条逻辑消息。 除第三条消息外，所有消息都非常大，需要分段，这将由放入应用程序执行：

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

在获取应用程序中，在第一个 MQGET 上指定 MQGMO_ALL_MSGS_AVAILABLE。 这意味着，只有整个组可用时，才会检索该组的消息或段。 检索某组的第一条物理消息时，将使用 MQGMO_LOGICAL_ORDER 来确保按顺序检索该组的段和消息：

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
             (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

注: 如果指定 MQGMO_LOGICAL_ORDER 且有一个当前组，那么将忽略 MQGMO_ALL_MSGS_AVAILABLE。

参考消息
使用此信息了解有关参考消息的更多内容。

注: IBM MQ for z/OS 中不支持参考消息。

该方法允许在不将大型对象存储在源或目标节点的 IBM MQ 队列上的情况下，将该对象从一个节点传输到另一个节点。 这在数据以其他形式存在时尤为有用，如用于邮件应用程序。

为此，可在通道两端指定消息出口。 有关如何执行此操作的信息，请参阅第 890 页的『通道消息出口程序』。

IBM MQ 定义参考消息头 (MQRMH) 格式。 请参阅 MQRMH，以获取此内容的描述。 它使用定义的格式名称进行识别，并且可能后跟实际的数据。

要启动大对象传输，应用程序可放入包含参考消息头（后无任何数据）的消息。 由于此消息离开节点，消息出口会以相应的方式检索对象，并将其附加到参考消息。 然后，它会将消息（现在大于以前）返回到发送消息通道代理程序，以传输到接收 MCA。

接收的 MCA 上配置了其他消息出口。 当此消息出口接收这些消息中的一条消息时，它会使用附加的对象数据来创建对象，并在不含它的参考消息上传递。 应用程序可接收参考消息，其了解已在此节点上创建对象（或至少是由此参考消息表示的一部分对象）。

发送消息出口可附加到参考消息上的最大对象数据量受通道的协商最大消息长度限制。 出口只能为其传递的每条消息仅返回一条消息到 MCA，因此放入应用程序可放入多条消息以传输一个对象。 每条消息都必须确定要附加的对象的逻辑长度及偏移量。 但是，如果不可能知道对象总大小或通道允许的最大大小，请设计发送消息出口，以便放入应用程序仅放入一条消息，出口本身在将尽可能多的数据附加到其传递的消息上时，将下一条消息放在传输队列上。

在使用该方法处理大型消息之前，请考虑以下几点：

- MCA 和消息出口在 IBM MQ 用户标识下运行。 消息出口（以及相应的用户标识）需要访问对象以在发送端检索对象，或在接收端创建对象；这可能只有在普遍可访问对象的情况下可行。 这将会产生安全问题。
- 如果附加了批量数据的参考消息在到达其目标之前必须经过多个队列管理器，那么批量数据将存在于中间节点的 IBM MQ 队列上。 但是，在这些情况下无需提供特殊支持或出口。
- 如果允许重排或死信排队，那么设计消息出口将非常困难。 在这些情况下，对象的各部分可能会不按顺序到达。
- 当参考消息到达其目标时，接收消息出口将创建对象。 但是，这与 MCA 工作单元不同步，因此如果回退批量，那么包含此相同部分对象的其他参考消息将在后续批量中到达，消息出口可能会尝试重新创建相同部分的对象。 例如，如果对象是一系列数据库更新，这可能是不能接受的。 如果发生这种情况，消息出口必须保留已应用更新的日志；这可能需要使用 IBM MQ 队列。
- 根据对象类型的特征，消息出口和应用程序可能需要在保留使用计数时协作，以便能够在不再需要对象时将其删除。 还可能需要实例标识；在参考消息头中为此提供一个字段（请参阅 MQRMH）。
- 如果放入参考消息作为分发列表，那么必须能够为此节点上生成的每个分发列表或单个目标检索对象。 您可能需要保留使用计数。 还要考虑节点对于列表中的一些目标是最终节点，但对于其他目标是中间节点的可能性。
- 通常，不会转换批量数据。 这是因为在调用消息出口前已进行转换。 因此，不得在起始发送方通道上请求转换。 如果参考消息通过中间节点，那么批量数据将在从中间节点发送时进行转换（如果请求转换的话）。
- 参考消息无法进行分段。

使用 MQRMH 和 MQMD 结构

请参阅 MQRMH 和 MQMD，以获取有关参考消息头和消息描述符中的字段的描述。

在 MQMD 结构中，将 *Format* 字段设置为 MQFMT_REF_MSG_HEADER。 当 MQGET 上请求时，MQHREF 格式会与其后的所有批量数据一起由 IBM MQ 自动转换。

以下是使用 MQRMH 的 *DataLogicalOffset* 和 *DataLogicalLength* 字段的示例：

放入应用程序可能会放入满足下列条件的参考消息：

- 无物理数据
- *DataLogicalLength* = 0（此消息表示整个对象）
- *DataLogicalOffset* = 0。

假定对象长度为 70,000 字节，发送消息出口会在参考消息中沿通道发送前 40,000 个字节，包含：

- MQRMH 之后的 40,000 字节物理数据
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0（从对象开头）。

然后将其他消息放在传输队列上，包含：

- 无物理数据
- *DataLogicalLength* = 0（到对象末端）。 可在此指定值 30,000。
- *DataLogicalOffset* = 40000（从该点开始）。

在发送消息出口看到此消息出口时，将附加其余的 30,000 字节数据，并将字段设置为：

- MQRMH 之后的 30,000 字节物理数据
- *DataLogicalLength* = 30000

- *DataLogicalOffset* = 40000（从该点开始）。

还将设置 MQRMHF_LAST 标志。

有关为使用参考消息提供的样本程序的描述，请参阅第 964 页的『在 Multiplatforms 上使用样本程序』。

等待消息

如果您希望程序等到消息到达队列，请在 MQGMO 结构的 *Options* 字段中指定 MQGMO_WAIT 选项。

使用 MQGMO 结构的 *WaitInterval* 字段来指定 您希望 MQGET 调用等待消息到达队列的最大时间（毫秒）。

如果消息未在此时间内到达，那么将完成 MQGET 调用，显示 MQRC_NO_MSG_AVAILABLE 原因码。

可使用 *WaitInterval* 字段中的 MQWI_UNLIMITED 常量，来指定无限的等待时间间隔。 但是，超出您控制范围的事件可能会使程序等待很长的一段时间，因此应谨慎使用此常量。 IMS 应用程序不得指定无限的等待时间间隔，因为这将使 IMS 系统无法终止运行。 （当 IMS 终止运行时，它需要终止所有从属区域。） 相反，IMS 应用程序可指定有限的等待时间间隔；然后，如果在此时间间隔后，完成调用但未检索消息，请发出具有等待选项的另一 MQGET 调用。

注: 如果多个程序正在同一共享队列上等待除去消息，那么到达的消息只能激活一个程序。 但是，如果多个程序正在等待浏览消息，那么可激活所有程序。 有关更多信息，请参阅 MQGMO 中 MQGMO 结构的 *Options* 字段描述。

如果在等待时间间隔到期之前更改队列或队列管理器状态，将发生以下情况：

- 如果队列管理器进入停顿状态，并且使用了 MQGMO_FAIL_IF_QUIESCING 选项，那么将取消等待并完成 MQGET 调用，显示 MQRC_Q_MGR_QUIESCING 原因码。 如果没有此选项，调用将保持等待状态。
- z/OS 在 z/OS 上，如果连接（针对 CICS 或 IMS 应用程序）进入停顿状态，并且使用了 MQGMO_FAIL_IF_QUIESCING 选项，那么将取消等待并完成 MQGET 调用，显示 MQRC_CONN_QUIESCING 原因码。 如果没有此选项，调用将保持等待状态。
- 如果强制使队列管理器停止运行或取消队列管理器，那么将完成 MQGET 调用，显示 MQRC_Q_MGR_STOPPING 或 MQRC_CONNECTION_BROKEN 原因码。
- 如果更改了队列（或针对其解析队列名称的队列）的属性，以致于现在获取请求受到禁止，那么将取消等待并完成 MQGET 调用，显示 MQRC_GET_INHIBITED 原因码。
- 如果以需要 FORCE 选项的方式更改队列（或针对其解析队列名称的队列）的属性，那么将取消等待并完成 MQGET 调用，显示 MQRC_OBJECT_CHANGED 原因码。

z/OS 如果您希望应用程序在多个队列上等待，请使用 IBM MQ for z/OS 的信号功能（请参阅第 737 页的『发信号』）。 有关进行这些操作的情况的更多信息，请参阅 MQGMO。

发信号

仅 IBM MQ for z/OS 上支持发信号。

发信号是 MQGET 调用上的一个选项，允许操作系统在预期的消息到达队列时通知（或发信号到）程序。 这类似于主题第 737 页的『等待消息』中所述的获取及等待功能，因为它允许程序在等待信号的同时继续处理其他工作。 但是，如果使用发信号，那么可释放应用程序线程并依靠操作系统来通知程序消息何时到达。

设置信号

要设置信号，请在 MQGET 调用上使用的 MQGMO 结构中完成以下操作：

1. 在 *Options* 字段中设置 MQGMO_SET_SIGNAL 选项。
2. 在 *WaitInterval* 字段中设置信号的最大使用时长。 这可设置您希望 IBM MQ 监控队列的时长（毫秒）。 使用 MQWI_UNLIMITED 值来指定无限的使用时长。

   注: IMS 应用程序不得指定无限的等待时间间隔，因为这将使 IMS 系统无法终止运行。 （当 IMS 终止运行时，它需要终止所有从属区域。） 相反，IMS 应用程序可定期检查 ECB 的状态（请参阅步骤 3）。 一个程序同时可以在多个队列句柄上设置信号：
3. 在 *Signal1* 字段中指定事件控制块 (ECB) 的地址。 这将通知您信号结果。 在关闭队列前，ECB 存储必须保持可用。

注: 不能将 MQGMO_SET_SIGNAL 选项与 MQGMO_WAIT 选项一起使用。

消息到达时

当合适的消息到达时，将向 ECB 返回完成代码。

完成代码描述下列其中一项：

- 为其设置信号的消息已到达队列。 未针对请求了信号的程序保留该消息，因此程序必须重新发出 MQGET 调用以获取消息。

  注: 其他应用程序可在接收信号与发出其他 MQGET 调用之间的时间内获取消息。
- 所设置的等待时间间隔已到期，为其设置信号的消息未到达队列。 IBM MQ 已取消信号。
- 信号已取消。 例如，如果队列管理器停止运行或更改了队列属性，以致于不再允许进行 MQGET 调用，那么将发生此情况。

队列上已有合适的消息时，MQGET 调用会以与在不发信号的情况下进行 MQGET 调用相同的方式完成。 此外，如果立即检测到错误，那么将完成调用并设置返回码。

接受调用并且无消息即时可用时，控制将返回到程序，以便其能够继续完成其他工作。 未设置消息描述符中的任何输出字段，但 **CompCode** 参数已设置为 MQCC_WARNING，**Reason** 参数设置为 MQRC_SIGNAL_REQUEST_ACCEPTED。

有关在使用发信号进行 MQGET 调用时，IBM MQ 可以向应用程序返回的内容的信息，请参阅 MQGET。

如果程序在等待发布 ECB 时没有其他工作需要完成，可使用以下项等待 ECB：

- 对于 CICS Transaction Server for z/OS 程序，可使用 EXEC CICS WAIT EXTERNAL 命令
- 对于批量和 IMS 程序，可使用 z/OS WAIT 宏

如果在设置信号时队列或队列管理器状态发生更改（也即，尚未发布 ECB），将发生以下情况：

- 如果队列管理器进入停顿状态，并且使用了 MQGMO_FAIL_IF_QUIESCING 选项，那么将取消信号。 将发布 ECB 并发出 MQEC_Q_MGR_QUIESCING 完成代码。 如果没有此选项，信号将保持设置状态。
- 如果强制使队列管理器停止运行或取消队列管理器，那么将取消信号。 将交付信号并发出 MQEC_WAIT_CANCELED 完成代码。
- 如果更改了队列（或针对其解析队列名称的队列）的属性，以致于现在获取请求受到禁止，那么将取消信号。 将交付信号并发出 MQEC_WAIT_CANCELED 完成代码。

注:

1. 如果多个程序已在同一共享队列上设置信号来除去消息，那么到达的消息只能激活一个程序。 但是，如果多个程序正在等待浏览消息，那么可激活所有程序。 队列管理器在确定要激活哪些应用程序时遵循的规则与等待应用程序遵循的规则相同：有关更多信息，请参阅 MQGMO - Get-message 选项中 MQGMO 结构的 *Options* 字段描述。
2. 如果有多个 MQGET 调用等待同一消息（同时具有等待和信号选项），那么将同等考虑每一个等待的调用。 有关更多信息，请参阅 MQGMO - Get-message 选项中 MQGMO 结构的 *Options* 字段描述。
3. 在某些情况下，MQGET 调用可检索消息，也可交付信号（由同一消息到达生成）。 这意味着当程序发出其他 MQGET 调用（由于已交付信号）时，可能没有可用消息。 为此情况设计要测试的程序。

有关如何设置信号的信息，请参阅 Signal1 中的 MQGMO_SET_SIGNAL 选项及 *Signal1* 字段描述。

跳过回退

可通过在 MQGET 调用上指定 **MQGMO_MARK_SKIP_BACKOUT** 选项，来防止应用程序进入 *MQGET-error-backout* 循环。

注: 仅在 IBM MQ for z/OS 上受支持。

作为工作单元的一部分，应用程序可发出一个或多个 MQGET 调用以从队列中获取消息。 如果应用程序检测到错误，可回退工作单元。 这会将运行此工作单元期间更新的所有资源都复原到启动工作单元前的状态，并恢复 MQGET 调用检索的消息。

一旦恢复，应用程序发出的后续 MQGET 调用便可使用这些消息。 在许多情况下，这都不会导致应用程序出现问题。 但是，如果无法规避导致发生回退的错误，那么在队列上恢复消息可能会使应用程序进入 *MQGET-error-backout* 循环。

要避免出现此问题，请在 MQGET 调用上指定 MQGMO_MARK_SKIP_BACKOUT 选项。 这会将 MQGET 请求标记为不包含在应用程序启动的回退中，也即，不得进行回退。 使用该选项意味着在进行回退时，将根据需要回退对其他资源的更新，但会将标记的消息视为如同在新的工作单元下检索的消息一样。

应用程序必须发出 IBM MQ 调用，以落实新的工作单元或回退新的工作单元。 例如，程序可执行异常处理（如通知发起方已丢弃消息），并落实工作单元以便从队列中除去消息。如果（出于任何原因）回退新的工作单元，那么队列上将恢复消息。

在一个工作单元内，只能将一条 MQGET 请求标记为“跳过回退”，但可以有多条其他消息未标记为“跳过回退”。 一旦将消息标记为“跳过回退”，那么工作单元内指定 MQGMO_MARK_SKIP_BACKOUT 的任何进一步调用 MQGET 的操作都会失败，显示原因码 MQRC_SECOND_MARK_NOT_ALLOWED。

注:

1. 只有包含标记的消息的工作单元因应用程序请求回退而终止时，此消息才会跳过回退。 如果工作单元出于任何其他原因而回退，那么消息将以与未将其标记为“跳过回退”时相同的方式回退到队列上。
2. 参与受 RRS 控制的工作单元的 Db2 存储过程内不支持跳过回退。 例如，具有 MQGMO_MARK_SKIP_BACKOUT 选项的 MQGET 调用将失败，显示原因码 MQRC_OPTION_ENVIRONMENT_ERROR。

第 739 页的图 68 说明了在需要 MQGET 请求跳过回退时，应用程序可能包含的典型的一系列步骤。

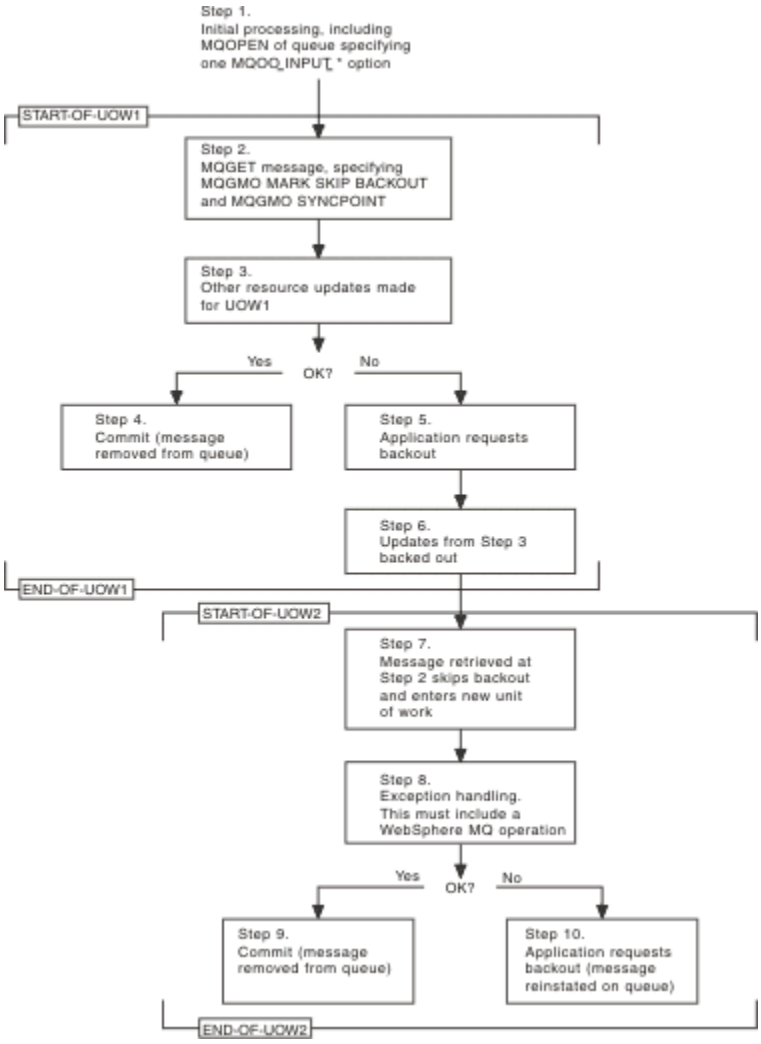


图 *68:* 使用 *MQGMO_MARK_SKIP_BACKOUT* 跳过回退

第 739 页的图 68 中的步骤是：

步骤 1
在事务内进行初次处理，包含调用 MQOPEN 以打开队列（指定某一 MQOO_INPUT_* 选项，以便在步骤 2 中从队列取出消息）。

步骤 2
将使用 MQGMO_SYNCPOINT 和 MQGMO_MARK_SKIP_BACKOUT 调用 MQGET。 需要 MQGMO_SYNCPOINT，因为 MQGET 必须位于工作单元中，才能使 MQGMO_MARK_SKIP_BACKOUT 生效。 在第 739 页的图 68 中，此工作单元称为 UOW1。

步骤 3
其他资源更新作为 UOW1 的一部分进行。 可包含更多的 MQGET 调用（在无 MQGMO_MARK_SKIP_BACKOUT 的情况下发出）。

步骤 4
根据需要完成步骤 2 和步骤 3 的所有更新。 应用程序落实更新，UOW1 结束。 步骤 2 中检索的消息将从队列中除去。

步骤 5
未根据需要完成步骤 2 和步骤 3 的部分更新。 应用程序请求将执行这些步骤期间所做的更新回退。

步骤 6
步骤 3 中所做的更新将回退。

步骤 7
步骤 2 中所做的 MQGET 请求会跳过回退，并成为新的工作单元 UOW2 的一部分。

步骤 8
UOW2 执行异常处理，以响应正在回退的 UOW1。 （例如，对其他队列进行 MQPUT 调用，指示发生了导致 UOW1 回退的问题。）

步骤 9
根据需要完成步骤 8，应用程序落实活动，UOW2 结束。 由于 MQGET 请求是 UOW2 的一部分（请参阅步骤 7），此落实将导致从队列中除去消息。

步骤 10
步骤 8 未根据需要完成，应用程序回退 UOW2。 由于获取消息请求是 UOW2 的一部分（请参阅步骤 7），它也将回退并在队列上恢复。 它现在可用于由此应用程序或其他应用程序发出的更多的 MQGET 调用（以与队列上的任何其他消息相同的方式可用）。

应用程序数据转换

需要时，MCA 会将消息描述符和头数据转换为所需的字符集和编码。 链路的任一端（即本地 MCA 或远程 MCA）均可完成转换。

应用程序将消息放在队列上时，本地队列管理器会将控制信息添加到消息描述符中，以简化队列管理器和 MCA 处理消息时对消息的控制过程。 根据环境的不同，将在本地系统的字符集和编码中创建消息头数据字段。

在系统间移动消息时，您有时需要将应用程序数据转换为接收系统所需的字符集和编码。 这可以从接收系统上的应用程序内部完成，也可以由发送系统上的 MCA 来完成。 如果接收系统上支持数据转换，请使用应用程序来转换应用程序数据，而不要依赖于已在发送系统上进行的转换。

当在传递到 MQGET 调用的 MQGMO 结构的 *Options* 字段中指定 MQGMO_CONVERT 选项，并且以下所有语句均为 true 时，将在应用程序内部转换应用程序数据：

- 与队列上的消息关联的 MQMD 结构中设置的 *CodedCharSetId* 或 *Encoding* 字段与 MQGET 调用上指定的 MQMD 结构中设置的 *CodedCharSetId* 或 *Encoding* 字段不同。
- 与消息关联的 MQMD 结构中的 *Format* 字段不是 MQFMT_NONE。
- MQGET 调用上指定的 *BufferLength* 不为零。
- 消息数据长度不为零。
- 队列管理器支持在与消息及 MQGET 调用关联的 MQMD 结构中指定的 *CodedCharSetId* 和 *Encoding* 字段之间进行转换。 请参阅 CodedCharSetId 和 Encoding，以获取所支持的编码字符集标识和机器编码的详细信息。

- 队列管理器支持转换消息格式。 如果与消息关联的 MQMD 结构的 *Format* 字段使用其中一种内置格式，那么队列管理器可转换此消息。 如果 *Format* 未使用其中一种内置格式，那么需要编写数据转换出口以转换消息。

如果发送 MCA 要转换数据，请在需要转换的每条发送方或服务器通道定义上指定 CONVERT(YES) 关键字。 如果数据转换失败，那么消息将发送到发送队列管理器上的 DLQ，MQDLH 结构的 *Feedback* 字段将指出原因。 如果无法在 DLQ 上放入消息，那么通道将关闭，并且未转换消息将保留在传输队列上。 在应用程序内部进行数据转换（而不是在发送 MCA 上转换数据）可避免此情况出现。

作为一条规则，由内置格式或数据转换出口描述为字符数据的消息中的数据将从消息所使用的编码字符集转换为请求的编码字符集，数字字段将转换为请求的编码。

有关在转换内置格式时使用的转换处理约定的更多信息，以及有关编写您自己的数据转换出口的信息，请参阅第 893 页的『编写数据转换出口』。 另请参阅本地语言和机器编码，以获取有关语言支持表以及受支持的机器编码的信息。

EBCDIC 换行符转换

如果需要确保从 EBCDIC 平台发送到 ASCII 平台的数据与您再次收到的数据相同，那么必须控制 EBCDIC 换行符转换过程。

可使用强制 IBM MQ 使用未修改转换表的与平台相关的开关来完成此操作，但您必须清楚这可能会造成行为不相一致。

出现这一问题是由于未在平台或转换表之间一致地转换 EBCDIC 换行符。 因此，如果在 ASCII 平台上显示数据，那么格式可能会不正确。 例如，这将使得您很难使用 RUNMQSC 从 ASCII 平台上远程管理 IBM i 系统。

请参阅数据转换，以获取有关将 EBCDIC 格式数据转换为 ASCII 格式的更多信息。

浏览队列中的消息

使用此信息来查找有关使用 MQGET 调用浏览队列中的消息的内容。

要使用 MQGET 调用浏览队列中的消息：

1. 调用 MQOPEN 以打开队列进行浏览，指定 MQOO_BROWSE 选项。
2. 要浏览队列中的第一条消息，请调用具有 MQGMO_BROWSE_FIRST 选项的 MQGET。 要查找您想要的消息，请重复调用具有 MQGMO_BROWSE_NEXT 选项的 MQGET，以逐句通过多条消息。

   每次调用 MQGET 后，都必须将 MQMD 结构的 *MsgId* 和 *CorrelId* 字段设置为空，以查看所有消息。
3. 调用 MQCLOSE 以关闭队列。

浏览光标

打开 (MQOPEN) 队列进行浏览时，调用过程会建立浏览光标，以与使用某一浏览选项的 MQGET 调用搭配使用。 您可以将浏览光标视为位于队列上的第一条消息前的逻辑指针。

您可以通过为相同的队列发出多条 MQOPEN 请求，（从单个程序）激活多个浏览光标。

调用 MQGET 进行浏览时，请在 MQGMO 结构中使用下列选项之一：

MQGMO_BROWSE_FIRST
获取满足 MQMD 结构中指定的条件的第一条消息的副本。

MQGMO_BROWSE_NEXT
获取满足 MQMD 结构中指定的条件的下一条消息的副本。

MQGMO_BROWSE_MSG_UNDER_CURSOR
获取光标当前指向的消息的副本，也即，使用 MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 选项上一次检索的消息副本。

在所有情况下，消息都会保留在队列上。

打开队列时，浏览光标在逻辑上正位于队列上的第一条消息前。 这意味着，如果在 MQOPEN 调用后立即进行 MQGET 调用，可使用 MQGMO_BROWSE_NEXT 选项来浏览第一条消息；无需使用 MQGMO_BROWSE_FIRST 选项。

从队列复制消息的顺序由此队列的 **MsgDeliverySequence** 属性确定。（有关更多信息，请参阅第 718 页的『从队列检索消息的顺序』。）

- 第 742 页的『FIFO（先入先出）序列中的队列』
- 第 742 页的『优先序列中的队列』
- 第 742 页的『未落实的消息数』
- 第 742 页的『队列序列的更改』
- 第 742 页的『使用队列的索引』

FIFO（先入先出）序列中的队列

此序列中的队列中的第一条消息是位于最长的队列上的消息。

使用 MQGMO_BROWSE_NEXT 在队列中按顺序读取消息。 您将看到浏览时放入队列中的所有消息，因为此序列中的队列将消息置于末尾。 光标确定到达队列末尾时，浏览光标会停在那里并返回 MQRC_NO_MSG_AVAILABLE。 然后，可将其留在那里等待更多的消息，或通过 MQGMO_BROWSE_FIRST 调用将其重置到队列开头。

优先序列中的队列

此序列中的队列中的第一条消息是位于最长的队列上的消息，在发出 MQOPEN 调用时具有最高的优先级。

使用 MQGMO_BROWSE_NEXT 读取队列中的消息。

浏览光标指向下一条消息，按优先级从第一条消息进行处理直到优先级最低的消息。 只要放入队列的消息优先级等于或低于当前浏览光标所识别的消息，它就会在此期间浏览所有放入队列中的消息。

放入队列中且具有更高优先级的所有消息只能通过以下操作进行浏览：

- 重新打开要浏览的队列，此时将建立新的浏览光标
- 使用 MQGMO_BROWSE_FIRST 选项

未落实的消息数

未落实消息一律不可见而无法浏览；浏览光标会跳过它。

只有落实工作单元后，才能浏览其中的消息。 落实消息时不会更改其在队列上的位置，因此将看不到跳过或未落实的消息，即使在落实后也是如此，除非使用 MQGMO_BROWSE_FIRST 选项并重新处理队列。

队列序列的更改

如果在队列上有多条消息时，将消息交付序列从优先级更改为 FIFO，那么将不会更改已排队的消息的顺序。 稍后添加到队列的消息将采用队列的缺省优先级。

使用队列的索引

浏览仅包含单一优先级的消息（持久和/或非持久）的索引队列时，队列管理器将使用要在使用特定形式浏览时浏览的索引。

注: 仅在 IBM MQ for z/OS 上受支持。

在索引队列仅包含单一优先级的消息时，将使用以下任何形式的浏览：

1. 如果按 MSGID 为队列建立索引，那么将使用索引处理在 MQMD 结构中传递 MSGID 的浏览请求，以查找目标消息。
2. 如果按 CORRELID 为队列建立索引，那么将使用索引处理在 MQMD 结构中传递 CORRELID 的浏览请求，以查找目标消息。
3. 如果按 GROUPID 为队列建立索引，那么将使用索引处理在 MQMD 结构中传递 GROUPID 的浏览请求，以查找目标消息。

如果浏览请求未在 MQMD 结构中传递 MSGID、CORRELID 或 GROUPID，并为队列建立索引且返回消息，那么必须找到消息的索引条目，并使用其中的信息来更新浏览光标。 如果使用选择范围广的索引值，那么这不会向浏览请求添加重要的额外处理。

消息长度未知时浏览消息

要在您不知道消息大小，并且不希望使用 *MsgId*、*CorrelId* 或 *GroupId* 字段来查找消息的情况下浏览消息，可使用 MQGMO_BROWSE_MSG_UNDER_CURSOR 选项：

1. 发出具有以下选项的 MQGET：
   - MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 选项
   - MQGMO_ACCEPT_TRUNCATED_MSG 选项
   - 缓冲区长度零

   注： 如果其他程序有可能获取相同的消息，也可考虑使用 MQGMO_LOCK 选项。 应返回 MQRC_TRUNCATED_MSG_ACCEPTED。
2. 使用返回的 *DataLength* 来分配所需存储量。
3. 发出具有 MQGMO_BROWSE_MSG_UNDER_CURSOR 的 MQGET。

所指向的消息是检索的最后一条消息；将不移动浏览光标。 可选择使用 MQGMO_LOCK 选项锁定消息，或使用 MQGMO_UNLOCK 选项解锁锁定的消息。

如果自打开队列起未成功发出具有 MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 选项的 MQGET，那么调用将失败。

除去已浏览消息

您可以从队列中除去已浏览消息，前提是已打开队列用于除去消息以及浏览消息。 （您必须在 MQOPEN 调用上指定一个 MQOO_INPUT_* 选项以及 MQOO_BROWSE 选项。）

要除去消息，请重新调用 MQGET，但在 MQGMO 结构的***选项***字段中，请指定 MQGMO_MSG_UNDER_CURSOR。在此情况下，MQGET 调用会忽略 MQMD 结构的 *MsgId*、*CorrelId* 和 *GroupId* 字段。

在浏览和除去步骤之间，其他程序可能已从队列中除去消息，包括位于浏览光标下方的消息。 在此情况下，调用 MQGET 将返回原因码，指出消息不可用。

按逻辑顺序浏览消息

第 718 页的『逻辑与物理排序』说明队列上消息的逻辑顺序与物理顺序之间的差异。 浏览队列时，这一差异尤为重要，因为一般来说不会正在删除消息，并且浏览操作不一定从队列开头处开始。

如果应用程序浏览一个组的不同消息（使用逻辑顺序），那么应遵循逻辑顺序以到达下一个组的开头，这一点非常重要，因为一个组的最后一条消息可能会实际出现在下一组的第一条消息后。 MQGMO_LOGICAL_ORDER 选项可确保在扫描队列时遵循逻辑顺序。

小心使用 MQGMO_ALL_MSGS_AVAILABLE（或 MQGMO_ALL_SEGMENTS_AVAILABLE）进行浏览操作。考虑具有 MQGMO_ALL_MSGS_AVAILABLE 的逻辑消息的情况。 此情况的结果是，只有组中所有的剩余消息同样存在时，逻辑消息才可用。 如果剩余消息不存在，将忽略逻辑消息。 这可能意味着，在缺少的消息之后到达时，将不会引起“浏览下一项”操作的注意。

例如，如果存在以下逻辑消息，

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

并使用 MQGMO_ALL_MSGS_AVAILABLE 发出浏览函数，那么将返回组 456 的第一条逻辑消息，将浏览光标留在此逻辑消息上。 如果组 123 的第二条（最后一条）消息现在到达：

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
```

```
   Logical message 1 (not last) of group 456  <=== browse cursor
   Logical message 2 (last)     of group 456
```

并发出同一“浏览下一项”函数，那么将不会注意到组 123 现已完成，因为该组的第一条消息位于浏览光标前。

在一些情况下（例如，如果在组完整存在时以破坏性方式检索消息），那么可结合使用 MQGMO_ALL_MSGS_AVAILABLE 和 MQGMO_BROWSE_FIRST。 否则，必须重复浏览扫描以注意到已错过的新到达的消息；仅发出 MQGMO_WAIT 以及 MQGMO_BROWSE_NEXT 和 MQGMO_ALL_MSGS_AVAILABLE 不会考虑这些消息。 （这一情况也可能出现在扫描消息完成后可能到达的高优先级消息上。）

以下几部分说明了处理未分段消息的浏览示例；分段消息遵循相似的原则。

浏览组中的消息

在此示例中，应用程序将按逻辑顺序浏览队列中的每一条消息。

可以对队列上的消息分组。 对于分组消息，只有任一组内的所有消息均到达后，应用程序才要开始处理该组。 因此，将为组中的第一条消息指定 MQGMO_ALL_MSGS_AVAILABLE；对于组中的后续消息，不需要该选项。

此示例中使用了 MQGMO_WAIT。 但是，虽然出于第 743 页的『按逻辑顺序浏览消息』中的原因，在新组到达时可满足等待要求，但如果浏览光标已通过组中的第一条逻辑消息，并且其余消息均已到达，那么将不满足此要求。 然而，等待适当的时间间隔可确保应用程序在等待新消息或新段的同时不会陷入往复循环。

始终使用 MQGMO_LOGICAL_ORDER 来确保以逻辑顺序执行扫描。 这与破坏性的 MQGET 示例形成对比，在后者中，由于正在除去每个组，因此在组中查找第一条（或唯一一条）消息时，将不使用 MQGMO_LOGICAL_ORDER。

将假定应用程序缓冲区始终都足够大，能够容纳整个消息（无论消息是否分段）。 因此，在每个 MQGET 上均指定了 MQGMO_COMPLETE_MSG。

以下给出浏览组中的逻辑消息的示例：

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

将重复该组，直至返回 MQRC_NO_MSG_AVAILABLE。

以中断性方式浏览和检索

在此示例中，应用程序会浏览组中的每个逻辑消息，然后决定是否以中断性方式检索该组。

此示例的第一部分与前一示例类似。 但是，在此案例中，在浏览整个组后，我们会决定返回并以中断性方式检索。

由于此示例中除去了每个组，因此在组中查找第一条或唯一一条消息时，将不使用 MQGMO_LOGICAL_ORDER。

以下给出浏览继而以中断性方式检索的示例：

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...
```

```
if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

避免重复传递已浏览过的消息
通过使用特定的 open 选项和 get-message 选项，可将消息标记为“已浏览”，以便当前或其他合作应用程序不会重新检索此消息。 可明确或自动取消标记消息，以便能够重新进行浏览。

如果浏览队列上的消息，那么可以与在以中断性方式获取消息时进行检索的顺序不同的顺序来检索这些消息。 特别地，可多次浏览相同的消息，如果从队列中除去它，将无法浏览。 要避免此情况，可在浏览消息时进行标记，并避免检索已标记的消息。 这有时称为浏览并标记。 要标记已浏览的消息，请使用获取消息选项 MQGMO_MARK_BROWSE_HANDLE，要仅检索未标记的消息，请使用 MQGMO_UNMARKED_BROWSE_MSG。 如果使用 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 和 MQGMO_MARK_BROWSE_HANDLE 选项组合，并发出重复的 MQGET，那么将依次检索队列上的每一条消息。 即使使用 MQGMO_BROWSE_FIRST 来确保不跳过消息，这也可防止重复交付消息。 此选项组合可由单个常量 MQGMO_BROWSE_HANDLE 来表示。 如果未浏览的队列上没有消息，那么将返回 MQRC_NO_MSG_AVAILABLE。

如果多个应用程序正在浏览同一队列，那么可使用 MQOO_CO_OP 和 MQOO_BROWSE 选项打开队列。 每个 MQOPEN 返回的对象句柄将被视为协作组的一部分。 由指定 MQGMO_MARK_BROWSE_CO_OP 选项的 MQGET 调用返回的任何消息都将被视为针对此协作的一组句柄进行标记。

如果已将消息标记了一段时间，那么可由队列管理器自动取消标记，使其再次可供浏览。 队列管理器属性 MsgMarkBrowseInterval 提供针对协作的一组句柄使消息保持标记状态的时间（毫秒）。 MsgMarkBrowseInterval 为 -1，意味着永不自动取消标记消息。

当标记消息的单个流程或一组协作流程停止时，将取消标记任何已标记的消息。

协作浏览示例

您可以运行多个分派器应用程序副本来浏览队列上的消息，并根据每一条消息的内容启动使用者。 在每个分派器中，使用 MQOO_CO_OP 打开队列。 这表示分派器正在进行协作，并了解彼此的已标记消息。 然后，每个分派器都会进行重复的 MQGET 调用，指定 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 和 MQGMO_MARK_BROWSE_CO_OP 选项（可使用单个常量 MQGMO_BROWSE_CO_OP 来表示此选项组合）。 然后，每个分派器应用程序只会检索尚未由其他协作分派器标记的消息。 分派器会初始化使用者，并将 MQGET 返回的 MsgToken 传递到使用者，这将以中断性方式从队列中获取消息。 如果使用者回退消息的 MQGET，那么其中一个浏览者可重新分派消息，因为该消息不再处于标记状态。 如果使用者未对消息进行 MQGET 调用，那么在经历 MsgMarkBrowseInterval 后，队列管理器将取消标记协作的一组句柄的消息，并可重新分派消息。

您可能有多个不同的分派器应用程序（而不是同一分派器应用程序的多个副本）浏览队列，每一个都适合处理队列上的一部分消息。 在每个分派器中，使用 MQOO_CO_OP 打开队列。 这表示分派器正在进行协作，并了解彼此的已标记消息。

- 如果单个分派器的消息处理顺序非常重要，那么每个分派器都会进行重复的 MQGET 调用，指定 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 和 MQGMO_MARK_BROWSE_HANDLE（或 MQGMO_BROWSE_HANDLE）选项。 如果浏览的消息适合此分派器进行处理，那么它会进行 MQGET 调用，指定 MQMO_MATCH_MSG_TOKEN、MQGMO_MARK_BROWSE_CO_OP 以及由先前的 MQGET 调用返回的 MsgToken。 如果成功调用，那么分派器将初始化使用者，将 MsgToken 传给使用者。
- 如果消息处理顺序不重要，并且预期分派器将处理其遇到的大部分消息，请使用 MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG 和 MQGMO_MARK_BROWSE_CO_OP（或 MQGMO_BROWSE_CO_OP）选项。 如果分派器浏览其无法处理的消息，那么可通过调用具有 MQMO_MATCH_MSG_TOKEN、MQGMO_UNMARK_BROWSE_CO_OP 选项和先前返回的 MsgToken 的 MQGET 来取消标记此消息。

MQGET 调用失败的一些情况

如果在发出 MQOPEN 和 MQGET 调用之间在命令上使用 FORCE 选项更改了队列的某些属性，那么 MQGET 调用将失败并返回 MQRC_OBJECT_CHANGED 原因码。

队列管理器将对象句柄标记为不再有效。 如果更改适用于针对其解析队列名称的任何队列，那么也将出现这一情况。 MQOPEN 中的 MQOPEN 调用描述中列出以此方式影响句柄的属性。 如果调用返回 MQRC_OBJECT_CHANGED 原因码，请关闭队列，重新打开，然后尝试重新获取消息。

如果针对您尝试从中获取消息的队列（或针对其解析队列名称的任何队列）禁止获取操作，那么 MQGET 调用将失败并返回 MQRC_GET_INHIBITED 原因码。 即使您正在使用 MQGET 调用进行浏览，也会出现这一情况。 如果您尝试稍后进行 MQGET 调用，并且如果应用程序设计使其他程序定期更改队列属性，那么您可能能够成功获取消息。

如果已删除动态队列（临时或永久队列），那么使用先前获取的对象句柄的 MQGET 调用将失败，并返回 MQRC_Q_DELETED 原因码。

编写发布/预订应用程序

开始编写发布/预订 IBM MQ 应用程序。

有关发布/预订概念的概述，请参阅发布/预订消息传递。

请参阅以下主题，以获取有关编写不同类型的发布/预订应用程序的信息：

- 第 747 页的『编写发布者应用程序』
- 第 753 页的『编写订户应用程序』
- 第 769 页的『发布/预订生命周期』
- 第 772 页的『发布/预订消息属性』
- 第 774 页的『消息排序』
- 第 774 页的『拦截发布内容』
- 第 781 页的『发布选项』
- 第 782 页的『预订选项』

相关概念

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

第 40 页的『IBM MQ 应用程序的设计注意事项』
当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

第 672 页的『编写用于排队的过程应用程序』
使用此信息来了解有关编写排队应用程序、连接和断开队列管理器、发布/预订以及打开和关闭对象的信息。

第 834 页的『编写客户机过程应用程序』
使用过程语言在 IBM MQ 上编写客户机应用程序时需要知道的内容。

第 908 页的『构建过程应用程序』
您可以使用若干过程语言之一编写 IBM MQ 应用程序，并在几个不同平台上运行该应用程序。

第 947 页的『处理过程程序错误』
本信息说明与应用程序 MQI 调用关联的错误，这些错误可能是应用程序进行调用时产生的，也可能是将其消息传递给最终目标时产生的。

相关任务

第 963 页的『使用 IBM MQ 样本过程程序』
这些样本程序采用过程语言编写，并演示了消息队列接口 (MQI) 的典型用法。 IBM MQ 在不同平台上编程。

编写发布者应用程序

通过研究两个示例来开始编写发布者应用程序。 第一个示例尽可能地按照将消息放入队列的点到点应用程序进行建模，第二个示例动态地创建主题 - 这是更为常见的发布者应用程序模式。

编写一个简单的 IBM MQ 发布者应用程序，就像编写用于将消息放入队列 (第 747 页的表 125) 的 IBM MQ 点到点应用程序。差别在于将 MQPUT 消息传递到主题（而不是队列）。

表 *125:* 点到点与发布/预订 *IBM MQ* 程序模式。

步骤	**点到点 MQ 调用**	**发布 MQ 调用**
连接到队列管理器	MQCONN	MQCONN
打开队列	MQOPEN	
打开主题		MQOPEN
放置消息	MQPUT	MQPUT
关闭主题		MQCLOSE
关闭队列	MQCLOSE	
断开与队列管理器的连接	MQDISC	MQDISC

具体而言，共有两个用于发布股价的应用程序示例。 第一个示例（第 747 页的『示例 1：固定主题的发布者』）以接近于将消息放入队列的方式进行建模，管理员以类似于创建队列的方式来创建主题定义。 程序员编码 MQPUT 以便将消息写至主题，而不是将其写入队列。 在第二个示例（第 750 页的『示例 2：可变主题的发布者』）中，程序与 IBM MQ 之间的交互模式相似。 差别在于，消息被写至的主题由程序员提供，而不是由管理员提供。 实际上，这通常意味着主题字符串由内容定义或由其他源提供，如人通过浏览器输入。

相关概念

第 753 页的『编写订户应用程序』
请通过学习三个示例开始编写订户应用程序：使用队列中的消息的 IBM MQ 应用程序、创建预订并且不要求了解排队的应用程序以及最后的同时使用排队和预订的示例。

相关参考

DEFINE TOPIC
DISPLAY TOPIC
DISPLAY TPSTATUS

示例 *1*：固定主题的发布者
IBM MQ 程序演示如何发布到以管理方式定义的主题。

注: 紧凑编码样式是为了便于您阅读，而不是用于生产目的。

请参阅第 748 页的图 70 中的输出。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
                publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

图 *69:* 固定主题的简单 *IBM MQ* 发布者。

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

图 *70:* 第一个发布者示例的样本输出

以下选择的代码行说明为 IBM MQ 编写发布者应用程序的各个方面。

char topicNameDefault[] = "IBMSTOCKPRICE";
在程序中定义了缺省的主题名。 您可以通过提供另一个主题对象的名称作为此程序的第一个自变量来覆盖此名称。

MQCHAR resTopicStr[151];
resTopicStr 由 td.ResObjectString.VSPtr 指向，并且由 MQOPEN 用于返回所解析的主题字符串。 请使 resTopicStr 的长度比 td.ResObjectString.VSBufSize 中传递的长度大 1，以便为 null 终止符提供空间。

memset (resTopicStr, 0, sizeof(resTopicStr));
将 resTopicStr 初始化为 null，以确保 MQCHARV 中返回的已解析主题字符串以 null 终止。

td.ObjectType = MQOT_TOPIC
有一种新的用于发布/预订的对象类型：主题对象。

td.Version = MQOD_VERSION_4;
要使用新对象类型，必须至少使用对象描述符的 *V4* 版本。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
topicName 是主题对象（有时被称为“管理主题对象”）的名称。 在此示例中，需要使用 IBM MQ 资源管理器或以下 MQSC 命令提前创建主题对象：

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
程序中的最后一个 printf 将回传已解析的主题字符串。 为 IBM MQ 设置 MQCHARV ResObjectString 结构以将解析后的字符串返回到程序。

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
打开主题以便进行输出；这就像打开队列以便进行输出一样。

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
您希望新订户能够接收到发布内容，并且，通过在发布者中指定 MQPMO_RETAIN，当您启动订户时，该订户将接收到在其启动前发布的最新发布内容作为其第一个匹配发布内容。 另一种方法是，只向订户提供在该订户启动后发布的发布内容。 另外，订户可以通过在其预订中指定 MQSO_NEW_PUBLICATIONS_ONLY 来拒绝接收保留式发布内容。

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
对传递到 MQPUT 的字符串的长度加 1，以便将 null 终止字符作为消息缓冲区的组成部分传递给 IBM MQ。

第一个示例演示什么？ 此示例尽可能接近地模仿您已尝试并测试过的有关编写点到点 IBM MQ 程序的传统模式。 IBM MQ 编程模式的一个重要特征是，程序员不关心消息的发送目标。 程序员的任务是，连接到队列管理器并向其传递要分发给接收方的消息。 在点到点范例中，程序员打开由管理员配置的队列（可能是别名队列）。 此别名队列将消息路由到本地队列管理器上的目标队列或者远程队列管理器。 在消息等待传递时，它们存储在源与目标之间某个位置处的队列中。

在发布/预订模式中，程序员打开主题，而不是打开队列。 在我们的示例中，管理员使主题与主题字符串相关联。 队列管理器使用队列将发布内容转发到本地或远程订户，那些订户具有与发布内容的主题字符串匹配的预订。 如果保留发布内容，队列管理器将保留该发布内容的最新副本，即使当前没有该内容的订户亦如此。 保留式发布内容可以转发给将来的订户。 发布者应用程序并不负责选择发布内容或者将发布内容路由到目标；它的任务是创建发布内容并将其放到管理员定义的主题。

这个固定主题示例是一个典型的发布/预订应用程序：它是静态的。 它要求管理员定义主题字符串以及更改要发布的主题。 通常，发布/预订应用程序需要了解主题树的某些或全部内容。 主题可能会频繁地更改，另一种可能情况是，虽然主题不会经常更改，但主题组合的数目非常大，导致管理员难以为每个可能需要发布的主题字符串定义主题节点。 主题字符串在发布前可能处于未知状态；发布者应用程序可以使用发布内容中的信息来指定主题字符串，它也可能有关于要从其他源（如人从浏览器输入）发布的主题字符串的信息。 为了适应更动态的发布样式，下一个示例将说明如何在发布者应用程序中动态地创建主题。

主题将发布者和订户耦合到一起。 在发布/预订解决方案的开发过程中，设计用于命名主题并在主题树中对其进行组织的规则（即体系结构）是重要的步骤。 请仔细审视主题树的组织将发布者和订户程序绑定到一起以及将它们与主题树内容绑定的程度。 问自己一个问题，主题树中的更改是否会影响发布者和订户应用程序

以及可如何最大程度地降低此影响。 在 IBM MQ 发布/预订模型的体系结构中，提供了管理主题对象的内置表示法，该对象用于提供主题的根部分（即根子树）。 主题对象允许您选择以管理方式定义主题树的根部分，这将简化应用程序编程和操作，从而提高可维护性。 例如，如果您正在部署多个具有隔离式主题树的发布/预订应用程序，那么通过以管理方式定义主题树的根部分，可以确保主题树的隔离，即使不同应用程序采用的各种主题命名约定并不一致亦如此。

实际上，发布者应用程序既可以只使用固定主题（如本示例所示），也可以使用可变主题（如下一个示例所示）。 第 750 页的『示例 2：可变主题的发布者』还演示了如何将主题与主题字符串配合使用。

相关概念

第 750 页的『示例 2：可变主题的发布者』
这是一个 WebSphere MQ 程序，它演示了如何发布到以编程方式定义的主题。

第 753 页的『编写订户应用程序』
请通过学习三个示例开始编写订户应用程序：使用队列中的消息的 IBM MQ 应用程序、创建预订并且不要求了解排队的应用程序以及最后的同时使用排队和预订的示例。

示例 *2*：可变主题的发布者

这是一个 WebSphere MQ 程序，它演示了如何发布到以编程方式定义的主题。

注: 紧凑编码样式是为了便于您阅读，而不是用于生产目的。

请参阅第 751 页的图 72 中的输出。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

图 *71:* 可变主题的简单 *IBM MQ* 发布者。

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

图 *72:* 第二个发布者示例的样本输出

关于此示例，需要注意下列几点。

char topicNameDefault[] = "STOCKS";
缺省主题名 STOCKS 定义主题字符串的组成部分。 您可以通过提供主题名作为此程序的第一个自变量来覆盖此主题名，也可以通过提供 / 作为第一个参数不使用主题名。

char topicString[101] = "IBM/PRICE";
IBM/PRICE 是缺省的主题字符串。 您可以通过提供主题字符串作为此程序的第二个自变量来覆盖此主题字符串。

队列管理器将 STOCKS 主题对象 “NYSE” 提供的主题字符串与程序 “IBM/PRICE” 提供的主题字符串组合在一起，并在两个主题字符串之间插入 “/”。 结果是已解析的主题字符串 “NYSE/IBM/PRICE”。 生成的主题字符串与 IBMSTOCKPRICE 主题对象中定义的主题字符串相同，并且作用也完全相同。

已解析的主题字符串的相关联管理主题对象不必是发布者传递到 MQOPEN 的主题对象。 IBM MQ 将使用已解析的主题字符串中的隐式树来确定哪个管理主题对象定义了与发布内容相关联的属性。

假设有两个主题对象 A 和 B， A 定义主题 “a”， B 定义主题 “a/b” (第 752 页的图 73)。 如果发布程序引用主题对象 A 并提供主题字符串 “b”(将主题解析为主题字符串 “a/b”) ，那么发布程序将从主题对象 B 继承其属性，因为主题与为 B 定义的主题字符串 “a/b” 相匹配。

if (strcmp(argv[1],"/"))
argv[1] 是以可选方式提供的 topicName。 “/” 作为主题名称无效; 此处表示没有主题名称，主题字符串完全由程序提供。 第 751 页的图 72 中的输出表明，整个主题字符串由程序动态提供。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
对于缺省情况，需要使用 IBM MQ Explorer 或此 MQSC 命令预先创建可选 topicName :

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
主题字符串是主题描述符中的 MQCHARV 字段。

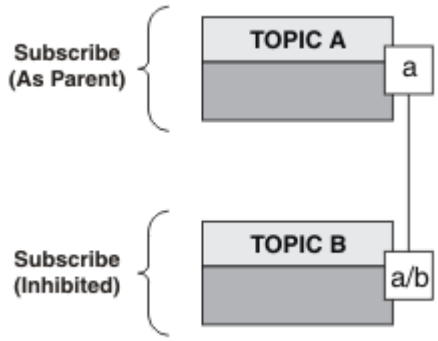


图 *73:* 主题对象关联

第二个示例演示什么？ 虽然代码与第一个示例很相似（实际上，只有两行有所不同），但生成的程序与第一个示例差异很大。 发布内容所发送到的目标由程序员控制。 与用于设计订户应用程序的最少量管理员输入相结合，不需要预定义任何主题或队列即可将发布内容从发布者路由到订户。

在点到点消息传递范例中，必须先定义队列，这样消息才能流动。 对于发布/预订而言，虽然 IBM MQ 使用底层排队系统来实现发布/预订，但却不必定义队列；发布/预订应用程序将继承与消息传递和排队相关联的有保证传递、事务性和松耦合优势。

设计者必须确定发布者、订户和程序是否了解底层的主题树，并确定订户程序是否了解排队。 请接着研究订户示例应用程序。 它们设计成与发布者示例配合使用，并且通常发布和预订 NYSE/IBM/PRICE。

相关概念
第 747 页的『示例 1：固定主题的发布者』
IBM MQ 程序演示如何发布到以管理方式定义的主题。

第 753 页的『编写订户应用程序』

请通过学习三个示例开始编写订户应用程序：使用队列中的消息的 IBM MQ 应用程序、创建预订并且不要求了解排队的应用程序以及最后的同时使用排队和预订的示例。

编写订户应用程序

请通过学习三个示例开始编写订户应用程序：使用队列中的消息的 IBM MQ 应用程序、创建预订并且不要求了解排队的应用程序以及最后的同时使用排队和预订的示例。

第 753 页的表 126 列出了三种样式的使用者和订户以及它们所特有的 IBM MQ 函数调用顺序。

1. 第一种样式“MQ 发布内容使用者”与只执行 MQGET 的点到点 MQ 程序完全相同。 此应用程序不知道它正在使用发布内容 — 它只是从队列中读取消息。 导致发布内容被路由到队列的预订是使用 IBM MQ 资源管理器以管理方式创建的，或者是使用命令创建的。
2. 第二种样式是大多数订户应用程序的首选模式。 订户应用程序创建预订，然后获取发布内容。 队列管理完全由队列管理器执行。
3. 在第三种样式中，订户应用程序选择打开并关闭用于存放发布内容的底层队列，并发出预订以便在该队列中填充发布内容。

理解这些样式的一种方法是，研究第 753 页的表 126 中列出的每种样式的示例 C 程序。 这些示例设计成与第 747 页的『编写发布者应用程序』中的发布者示例配合运行。

表 *126:* 点到点与预订 *IBM MQ* 程序模式。

步骤	MQ 消息使用者	第 753 页的『示例 1：MQ 发布内容使用者』	第 756 页的『示例 2：受管 MQ 订户』	第 761 页的『示例 3：非受管 MQ 订户』
连接到队列管理器	MQCONN	MQCONN	MQCONN	MQCONN
打开队列	MQOPEN	MQOPEN		MQOPEN
预订			MQSUB	MQSUB
获取消息	MQGET	MQGET	MQGET	MQGET
关闭队列	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
关闭预订			MQCLOSE	MQCLOSE
断开与队列管理器的连接	MQDISC	MQDISC	MQDISC	MQDISC

使用 MQCLOSE 始终是可选的，它用于释放资源、传递 MQCLOSE 选项或者仅仅与 MQOPEN 对称。 在“受管 MQ 订户”用例中，由于您不大可能需要在预订队列处于关闭状态时指定 MQCLOSE 选项，并且对称自变量没有意义，因此示例 2：受管 MQ 订户不以显式方式关闭订户队列。

另一种理解发布/预订应用程序模式的方法是，研究所涉及的不同实体之间的交互。 生命线或 UML 时序图是一种很好的研究交互的方法。 在第 769 页的『发布/预订生命周期』中描述了三个生命线示例。

示例 *1*：*MQ* 发布内容使用者

MQ 发布内容使用者是一个 IBM MQ 消息使用者，它不预订主题本身。

要为此示例创建预订和发布内容队列，请运行下列命令，或者使用 IBM MQ 资源管理器来定义对象。

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

IBMSTOCKPRICESUB 预订引用了为发布者示例创建的 IBMSTOCK 主题对象以及本地队列 STOCKTICKER。 主题对象 IBMSTOCK 定义了此预订中使用的主题字符串 NYSE/IBM/PRICE。 注意，在创建此预订之前，需要定义用于接收发布内容的主题对象和队列。

MQ 发布内容使用者模式包含多个有价值的构面：

1. 多处理：共享读取发布内容这一工作。 发布内容全都进入与预订主题相关联的单一队列。 多个使用者可以使用 MQOO_INPUT_SHARED 来打开该队列。

2. 集中管理的预订。 应用程序不需要构造它们自己的预订主题或预订；管理员负责确定发布内容的发送目标。
3. 预订并置：可以将多个不同的预订发送到单一队列。
4. 预订耐久性：队列将接收到所有发布内容，而无论使用者是否处于活动状态。
5. 迁移和共存：使用者代码在点到点和发布/预订方案中的工作表现同样好。

预订在主题字符串 NYSE/IBM/PRICE 与队列 STOCKTICKER 之间创建关系。 创建预订之后，发布内容（包括任何当前的保留式发布内容）将被转发到 STOCKTICKER。

以管理方式创建的预订可以是受管预订或非受管预订。 就像非受管预订一样，受管预订将在创建时立即生效。 但是，并非所有模式构面都适用于受管预订。 请参阅 第 761 页的『示例 3：非受管 MQ 订户』

注: 紧凑编码样式是为了便于您阅读，而不是用于生产目的。

第 755 页的图 75 显示了结果。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

图 *74: MQ* 发布内容使用者。

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

图 *75: MQ* 发布内容使用者的输出

您应该了解几项标准的 IBM MQ *C* 语言编程技巧：

memset(publication, 0, sizeof(publicationBuffer));
确保消息包含尾部 null，以便于使用 printf 进行格式化。 发布者示例通过对 strlen(publication) 加 1 在传递到 MQPUT 的消息缓冲区中包括尾部 null。 对于使用缓冲区来存储字符串的 IBM MQ C 程序，将 MQCHAR 缓冲区设置为 null 是很好的编程样式，确保空值跟在未完全填充缓冲区的字符数组之后。

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
在消息缓冲区末尾保留一个 null，以确保在 if (messlen == strlen(publication)); 为 true 时，返回的消息包含尾部 null。 此技巧对上一个技巧进行补充，用于确保未被 publication 的内容覆盖的 publicationBuffer 至少包含一个 null。

相关概念

第 756 页的『示例 2：受管 MQ 订户』
受管 MQ 订户是大多数订户应用程序的首选模式。 此示例不需要队列、主题或预订的管理定义。

第 761 页的『示例 3：非受管 MQ 订户』
非受管订户是一种重要的订户应用程序类型。 借助非受管订户，可以将发布/预订的优势与发布内容的排队和使用控制权相结合。 本示例演示预订和队列的不同组合方法。

第 747 页的『编写发布者应用程序』
通过研究两个示例来开始编写发布者应用程序。 第一个示例尽可能地按照将消息放入队列的点到点应用程序进行建模，第二个示例动态地创建主题 - 这是更为常见的发布者应用程序模式。

示例 *2*：受管 *MQ* 订户

受管 MQ 订户是大多数订户应用程序的首选模式。 此示例不需要队列、主题或预订的管理定义。

这种最简单的受管订户类型通常使用非持久预订。 此示例侧重于非持久预订。 此预订只在来自 MQSUB 的预订句柄的生存期内存在。 在此预订的生存期内，任何与主题字符串匹配的发布内容都将被发送到预订队列（如果符合以下条件，那么此发布内容将是保留式发布内容：未设置 MQSO_NEW_PUBLICATIONS_ONLY 标志或者此标志具有缺省值，已保留与该主题字符串匹配的先前发布内容，并且，该发布内容是持久发布内容或者队列管理器在该发布内容创建后未曾终止）。

另外，还可以将持久预订与此模式配合使用。 通常，使用受管持久预订时，这样做可以提高可靠性，而不必建立不发生任何错误时生命周期将比订户长的预订。 有关与受管预订、非受管预订、持久预订和非持久预订相关联的不同生命周期的更多信息，请参阅相关主题章节。

持久预订通常与持久发布内容相关联，非持久预订与非持久发布内容相关联，但预订耐久性与发布内容持久性之间不必存在任何关系。 持久性与耐久性的全部四种组合都可行。

对于所考虑的受管非持久情况，队列管理器将创建一个预订队列，该队列在关闭时将被清除并删除。 关闭非持久预订时，将从该队列中除去发布内容。

此代码所演示的受管非持久模式的有价值构面列示如下：

1. 随需应变预订：预订主题字符串是动态的。 它由应用程序在运行时提供。
2. 自我管理队列：预订队列将进行自我定义和管理。
3. 自我管理预订生命周期：非持久预订只在订户应用程序的持续时间存在。
   - 如果定义持久受管预订，那么将生成一个永久预订队列，并且发布内容将继续存储在该队列中，而没有任何订户程序处于活动状态。 只有在应用程序或管理员选择删除该预订之后，队列管理器才会删除该队列并从中清除任何未被检索的发布内容。 您可以使用管理命令来删除该预订，也可以使用 MQCO_REMOVE_SUB 选项来关闭该预订。
   - 考虑对持久预订设置 SubExpiry，以便在除去该预订并导致队列管理器删除该队列以及其中的任何其余发布内容之前，停止将发布内容发送到该队列，并且订户可以使用任何其余发布内容。
4. 灵活的主题字符串部署：通过使用以管理方式定义的主题来定义预订的根部分，简化了预订主题管理工作。 于是，将在应用程序中隐藏主题树的根部分。 通过隐藏根部分，部署应用程序时，该应用程序就不会意外地创建与另一实例或另一应用程序所创建的另一主题树重叠的主题树。
5. 受管主题：通过使用第一部分与以管理方式定义的主题对象匹配的主题字符串，可以根据该主题对象的属性来管理发布内容。

- 例如，如果主题字符串的第一部分与集群主题对象的相关联主题字符串匹配，那么预订可以接收到来自集群中其他成员的发布内容。
- 以管理方式定义的主题对象与通过程序定义的预订之间的选择性匹配使您能够将二者的优势相结合。管理员提供主题的属性，程序员以动态方式定义子主题，而不必关心主题的管理工作。
- 主题的相关联属性由用于匹配主题对象的结果主题字符串提供，而不必由 sd.Objectname 中指定的主题对象提供，虽然它们通常被证实是同一个对象或者相同。 请参阅 第 750 页的『示例 2：可变主题的发布者』。

此示例将预订设置为持久预订，因此，在订户使用 MQCO_KEEP_SUB 选项关闭该预订之后，发布内容将继续被发送到订户队列。 此队列将在订户处于不活动状态期间继续接收发布内容。 通过使用 MQSO_PUBLICATIONS_ON_REQUEST 选项来创建预订并使用 MQSUBRQ 来请求获取保留式发布内容，可以覆盖此行为。

以后，可以通过使用 MQCO_RESUME 选项打开该预订将其恢复。

您可以通过多种方法来使用 MQSUB 所返回的队列句柄 Hobj。 在此示例中，使用队列句柄来查询预订队列的名称。 并且，使用缺省模型队列 SYSTEM.NDURABLE.MODEL.QUEUE 或 SYSTEM.DURABLE.MODEL.QUEUE 来打开受管队列。 通过逐个主题地提供您自己的持久和非持久模型队列作为此预订的相关联主题对象的属性，可以覆盖缺省行为。

无论从模型队列继承哪些属性，您都无法通过复用受管队列句柄来创建另一个预订。 您也无法通过使用所返回的队列名再次打开该受管队列来获取该受管队列的另一个句柄。 此队列的行为就像是，它已打开以便进行独占输入。

非受管队列比受管队列更灵活。 例如，您可以共享非受管队列或者对一个队列定义多个预订。 下一个示例将演示如何将预订与非受管预订队列相结合。

注: 紧凑编码样式是为了便于您阅读，而不是用于生产目的。

第 759 页的图 78 显示了结果。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){                /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
}
```

图 *76:* 受管 *MQ* 订户 — 第 *1* 部分：声明和参数处理。

关于此示例中的声明，还有另外一些注释。

MQHOBJ Hobj = MQHO_NONE;
您无法显式地打开非持久受管预订队列以接收发布内容，但需要为队列管理器打开该队列时返回的对象句柄分配存储器。 将此句柄初始化为 MQHO_OBJECT 至关重要。 这向队列管理器表明，它需要返回预订队列的队列句柄。

MQSD sd = {MQSD_DEFAULT};
MQSUB 中使用的新预订描述符。

MQCHAR48 qName;
虽然此示例不要求了解预订队列，但此示例查询预订队列的名称 - MQINQ 绑定在 C 语言中不够灵活，因此您会发现此部分的示例值得研究。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

图 *77:* 受管 *MQ* 订户 — 第 *2* 部分：代码主体。

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020    "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

图 *78: MQ* 订户

关于此示例中的代码，还有另外一些注释。

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
如果 topicName 为 null 或为空（缺省值），那么将不会使用主题名来计算已解析的主题字符串。

sd.ObjectString.VSPtr = topicString;
在此示例中，程序员提供由 MQSUB 组合的主题对象和主题字符串，而不是只使用预定义的主题对象。注意，主题字符串是 MQCHARV 结构。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
这是设置 MQCHARV 字段长度的替代方法。

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
定义主题字符串之后，您需要特别注意 sd.Options 标志。 存在许多选项，此示例仅指定最常用的选项。 其他选项都使用缺省值。

1. 由于此预订为非持久预订，即，它具有应用程序中打开的预订的生存期，因此请设置 MQSO_CREATE 标志。 您还可以设置缺省的 MQSO_NON_DURABLE 标志以便提高可读性。
2. MQSO_RESUME 是对 MQSO_CREATE 的补充。 您可以同时设置这两个标志；根据情况不同，队列管理器将创建新预订或者恢复现有预订。 但是，如果指定了 MQSO_RESUME，那么还必须为 sd.SubName 初始化 MQCHARV 结构，即使没有要恢复的预订亦如此。 未能初始化 SubName 将导致 MQSUB 发出返回码 2440：MQRC_SUB_NAME_ERROR。

   注: 对于非持久受管预订，将始终忽略 MQSO_RESUME：但是，在没有为 sd.SubName 初始化 MQCHARV 结构的情况下指定此标志却会引起错误。
3. 另外，还有第三个影响预订打开方式的标志，即 MQSO_ALTER。 在具有适当许可权的情况下，所恢复的预订的属性将更改为与 MQSUB 中指定的其他属性匹配。

   注: 必须至少指定 MQSO_CREATE、MQSO_RESUME 和 MQSO_ALTER 标志中的一个。 请参阅选项 (MQLONG)。 第 761 页的『示例 3：非受管 MQ 订户』提供了有关使用全部这三个标志的示例。
4. 设置 MQSO_MANAGED，以便让队列管理器自动管理预订。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
（可选）对于以 null 终止的字符串，省略设置 MQCHARV 的长度并改为使用 null 终止符标志。

sd.ResObjectString.VSPtr = resTopicStr;
程序中的第一个 printf 将回传所生成的主题字符串。 为 IBM MQ 设置 MQCHARV ResObjectString 以将解析的字符串返回到程序。

注: 在 memset(resTopicStr, 0, sizeof(resTopicStrBuffer)) 中，已将 resTopicStringBuffer 初始化为 null。 返回的主题字符串未以尾部 null 结尾。

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
将 sd.ResObjectString 的缓冲区大小设置为比它的实际大小要小 1。 这是为了避免覆盖提供的 null 终止符，以防已解析的主题字符串填满整个缓冲区。

注: 即使主题字符串长于 sizeof(resTopicStrBuffer)-1，也不会返回错误。 即使 VSLength > VSBufSiz，sd.ResObjectString.VSLength 中返回的长度也是整个字符串的长度，而不必是所返回字符串的长度。 测试 sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz 以确认主题字符串已完成。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
MQSUB 函数创建预订。 如果这是非持久预订，那么您可能对它的名称不感兴趣，尽管可以在 IBM MQ 资源管理器中检查它的状态。 您可以提供 sd.SubName 参数作为输入，以便了解所要查找的名称；显然，您必须避免名称与其他预订发生冲突。

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
同时关闭预订和预订队列是可选的。 在此示例中，关闭预订，但不关闭队列。 在本例中，MQCLOSE MQCO_REMOVE_SUB 是缺省选项，尽管该预订是非持久预订。 使用 MQCO_KEEP_SUB 是错误的。

注: MQSUB 不关闭预订队列，该预订队列的句柄 Hobj 在该队列被 MQCLOSE 或 MQDISC 关闭之前将保持有效。 如果应用程序过早终止，那么队列管理器将在应用程序终止后的某个时间清除队列和预订。

相关概念
第 753 页的『示例 1：MQ 发布内容使用者』
MQ 发布内容使用者是一个 IBM MQ 消息使用者，它不预订主题本身。

第 761 页的『示例 3：非受管 MQ 订户』
非受管订户是一种重要的订户应用程序类型。 借助非受管订户，可以将发布/预订的优势与发布内容的排队和使用控制权相结合。 本示例演示预订和队列的不同组合方法。

第 747 页的『编写发布者应用程序』
通过研究两个示例来开始编写发布者应用程序。 第一个示例尽可能地按照将消息放入队列的点到点应用程序进行建模，第二个示例动态地创建主题 - 这是更为常见的发布者应用程序模式。

示例 *3*：非受管 *MQ* 订户

非受管订户是一种重要的订户应用程序类型。 借助非受管订户，可以将发布/预订的优势与发布内容的排队和使用控制权相结合。 本示例演示预订和队列的不同组合方法。

非受管模式通常与持久预订而非非持久预订相关联。 通常，非受管订户所创建的预订的生命周期与预订应用程序本身的生命周期无关。 通过使预订持久，即使没有处于活动状态的预订应用程序，该预订也将接收到发布内容。

您可以通过创建持久受管预订来实现同一结果，但某些应用程序要求受管预订所无法实现的灵活性以及队列和消息控制权。 对于持久受管预订，队列管理器将为与预订主题匹配的发布内容创建一个永久队列。 它将在该预订被删除时删除该队列以及相关联的发布内容。

通常，如果应用程序和预订的生命周期基本相同但难以保证，那么使用持久受管预订。 通过使预订持久并使用共享预订，共享预订的每个应用程序都会打开同一个受管队列，并从中获取消息。

受管 预订是 IBM MQ 处理预订并为您进行注册和注销的预订，而在 非受管 预订中，应用程序负责指定存储预订的队列。

队列管理器以隐式方式打开订户的持久受管预订队列，这使得不可能对该队列进行共享处理。 另外，不能为每个受管队列创建多个预订，您将发现由于对队列名的控制权下降而导致队列更难以管理。 因此，对于需要持久预订的应用程序，请考虑非受管 MQ 订户是否比受管 MQ 订户更合适。

第 766 页的图 81 中的代码演示非受管持久预订模式。 为了进行举例说明，此代码还将创建非受管非持久预订。 该示例展示了以下模式构面：

- 随需应变预订：预订主题字符串是动态的。 它们由应用程序在运行时提供。
- 简化的预订主题管理：通过使用以管理方式定义的主题来定义预订主题字符串的根部分，简化了预订主题管理工作。 这将在应用程序中隐藏主题树的根部分。 通过隐藏根部分，可以将订户部署到不同的主题树。
- 灵活的预订管理：您可以通过管理方式来定义预订，也可以在订户程序中随需应变地创建预订。 以管理方式创建的预订与通过程序创建的预订没有区别，但有一个属性将指示该预订的创建方式。 还有第三种预订，即，队列管理器为了分发预订而自动创建的预订。 所有预订都将显示在 IBM MQ 资源管理器中。
- 预订与队列之间灵活的关联：MQSUB 函数使预定义的本地队列与预订相关联。 您可以通过不同方法来使用 MQSUB 使预订与队列相关联：
  - 将预订与具有 *no* 现有预订 `MQSO_CREATE + (Hobj from MQOPEN)`的队列相关联。
  - 将 新 预订与具有现有预订 `MQSO_CREATE + (Hobj from MQOPEN)`的队列相关联。
  - 将现有预订移至另一个队列 `MQSO_ALTER + (Hobj from MQOPEN)`。
  - 恢复与现有队列 `MQSO_RESUME + (Hobj = MQHO_NONE)`或 `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`关联的现有预订。
  - 通过按不同的组合来组合 `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER`，可以适应预订和队列的不同输入状态，而不必编码具有不同 `sd.Options` 值的多个 MQSUB 版本。
  - 另外，通过编码特定的 `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` 选项，如果作为输入提供给 MQSUB 的预订和队列的状态与 `sd.Options` 的值不一致，那么队列管理器将返回错误（第 762 页的表 127）。 第 768 页的图 87 显示了在使用不同 `sd.Options` 标志设置的情况下对预订 X 发出 MQSUB 并传递三个不同对象句柄的结果。

  请探查第 765 页的图 80 中的示例程序的不同输入，以便熟悉这些不同类型的错误。 常见错误 `RC = 2440`（未包括在表中列出的范例中）是预订名错误。 此错误的原因通常是，传递到 MQSO_RESUME 或 MQSO_ALTER 的预订名为 null 或无效。
- 多处理：可以在多个使用者之间共享读取发布内容这一工作。 发布内容全都进入与预订主题相关联的单一队列。 使用者可以选择使用 MQOPEN 来直接打开队列，也可以通过 MQSUB 来使用预订。

- 预订并置：可以在同一个队列中创建多个预订。 您务必谨慎地使用此功能，这是因为，此功能会导致预订重叠以及多次接收到相同的发布内容。 MQSO_GROUP_SUB 选项可以消除重叠预订所引起的重复发布内容。
- 订户与使用者分离：除了示例所演示的三种使用者模型以外，还有另一种模型，即，将使用者与订户分离。 这是非受管 MQ 订户的变体，即，一个程序预订发布内容，另一个程序使用这些内容，而不是在同一个程序中发出 MQOPEN 和 MQSUB。 例如，订户可能是发布/预订集群的组成部分，使用者连接到队列管理器集群外部的队列管理器。 通过将预订队列定义为远程队列定义，使用者通过标准的分布式排队功能来接收发布内容。

理解 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER 的行为至关重要，您计划通过使用这些选项的组合来简化代码时尤其如此。 请研究第 762 页的表 127，此表显示了将不同队列句柄传递到 MQSUB 的结果以及运行第 767 页的图 82 到第 768 页的图 87 所示示例程序的结果。

用于构造表的方案具有一个预订 (X) 和两个队列 (A 和 B)。 预订名称参数 sd.SubName 设置为 X，这是连接到队列 A 的预订的名称。 队列 B 没有附加任何预订。

在 第 762 页的表 127 中，MQSUB 会将预订 X 和队列句柄传递至队列 A。 预订选项的结果如下所示：

- MQSO_CREATE 失败，因为队列句柄对应于已预订 X 的队列 A 。 将此行为与成功调用进行对比。 此调用将成功，这是因为，没有任何对 X 的预订与队列 B 相连接。
- MQSO_RESUME 成功，因为队列句柄与已预订 X 的队列 A 相对应。 相反，如果预订 X 不存在于队列 A 上，那么调用将失败。
- 在打开预订和队列方面，MQSO_ALTER 的行为与 MQSO_RESUME 相似。 但是，如果传递到 MQSUB 的预订描述符中包含的属性与此预订的属性不同，那么 MQSO_RESUME 将失败，而只要程序实例有权改变属性，MQSO_ALTER 就会成功。 注意，您永远无法更改预订中的主题字符串；但 MQSUB 将忽略预订描述符中的主题名和主题字符串值并使用现有预订中的值，而不是返回错误。

接下来，查看 第 762 页的表 127，其中 MQSUB 将预订 X 和队列句柄传递给队列 B。 预订选项的结果如下所示：

- MQSO_CREATE 将成功并在队列 B 中创建预订 X，这是因为，这是队列 B 中的新队列。
- MQSO_RESUME 将失败。 MQSUB 将在队列 B 中查找预订 X 并且找不到，但是，它将返回 *RC = 2019* — 预订队列与队列对象句柄不匹配而不是 *RC = 2428* — 预订 *X* 不存在。 第三个选项 MQSO_ALTER 的行为暗示了这个意外错误发生的原因。 MQSUB 期望队列句柄指向包含预订的队列。 它将先检查此情况，然后再检查 sd.SubName 中指定的预订是否存在。
- MQSO_ALTER 将成功并将该预订从队列 A 移至队列 B。

此表未列出的一种情况是，队列 A 中的预订的预订名与 sd.SubName 中的预订名不匹配。 该调用将失败并返回 *RC = 2428* — 预订 *X* 在队列 *A* 中不存在。

表 *127:* 使用不同的队列句柄与预订组合时 *MQSUB* 返回的错误

队列句柄	**队列 A** **预订 X** **队列 B** **无预订**	**队列 A** **无预订** **队列 B** **无预订**
将队列 *A* 的 Hobj 传递到 MQSUB	**MQSO_CREATE** RC = 2432 — 预订 X 在队列 A 中已存在 **MQSO_RESUME** 在队列 A 中恢复预订 X **MQSO_ALTER** 在队列 A 中恢复预订 X 并进行允许的改变	**MQSO_CREATE** 在队列 A 中创建预订 X **MQSO_RESUME** RC = 2428 — 预订 X 在队列 A 中不存在 **MQSO_ALTER** RC = 2428 — 预订 X 在队列 A 中不存在

表 *127:* 使用不同的队列句柄与预订组合时 *MQSUB* 返回的错误 *(*继续*)*		
队列句柄	**队列 A** **预订 X** **队列 B** **无预订**	**队列 A** **无预订** **队列 B** **无预订**
将队列 *B* 的 Hobj 传递到 MQSUB	**MQSO_CREATE** 在队列 B 中创建新预订 X **MQSO_RESUME** RC = 2019 — 预订队列与队列对象句柄不匹配 **MQSO_ALTER** 将预订 X 从队列 A 移至队列 B	**MQSO_CREATE** 在队列 B 中创建新预订 X **MQSO_RESUME** RC = 2428 — 预订 X 在队列 B 中不存在 **MQSO_ALTER** RC = 2428 — 预订 X 在队列 B 中不存在
将 MQHO_NONE 传递到 MQSUB	**MQSO_CREATE** RC = 2019 — 对象句柄错误：设置 MQSO_MANAGED 标志以创建受管预订并创建受管队列 **MQSO_RESUME** 在队列 A 中恢复预订 X 并返回队列 A 的 Hobj **MQSO_ALTER** 在队列 A 中恢复预订 X，返回队列 A 的 Hobj 并进行允许的改变	**MQSO_CREATE** RC = 2019 — 对象句柄错误：设置 MQSO_MANAGED 标志以创建受管预订并创建受管队列 **MQSO_RESUME** RC = 2428 — 没有预订 X **MQSO_ALTER** RC = 2019 — 对象句柄错误：没有队列 A 或 B

注: 紧凑编码样式是为了便于您阅读，而不是用于生产目的。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

图 *79:* 非受管 *MQ* 订户 — 第 *1* 部分：声明。

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

图 *80:* 非受管 *MQ* 订户 — 第 *2* 部分：参数处理。

在此示例中有关参数处理的其他注释如下：

switch((argv[5][0]))
您可以选择在参数 5 中输入 A lter | C reate | R esume，以测试覆盖示例中缺省使用的 MQSUB 选项设置的部分的效果。 此示例使用的缺省设置是 MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE。

注: 在未设置 MQSO_DURABLE 的情况下设置 MQSO_ALTER 或 MQSO_RESUME 是错误的，并且，必须设置 sd.SubName 并使其引用可以恢复或改变的预订。

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
如果缺省预订队列 STOCKTICKER 被空字符串替换，那么只要设置了 MQSO_CREATE，此示例就会设置 MQSO_MANAGED 标志并创建动态预订队列。 如果在第五个参数中设置了 Alter or Resume，那么示例的行为将取决于 subscriptionName 的值。

***subscriptionName = '\0';**
sdOptions = sdOptions - MQSO_DURABLE;
如果缺省预订 IBMSTOCKPRICESUB 被空字符串替换，那么此示例将移除 MQSO_DURABLE 标志。 如果在对其他参数提供缺省值的情况下运行此示例，那么将创建一个以 STOCKTICKER 为目标的附加临时预订，该预订将接收重复的发布内容。 下次在不指定任何参数的情况下运行此示例时，您将只接收到一份发布内容。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                  &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
&CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes           */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
&CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

图 *81:* 非受管 *MQ* 订户 — 第 *3* 部分：代码主体。

在此示例中有关代码的其他注释如下：

if (strlen(subscriptionQueue))
如果没有预订队列名，那么此示例将使用 MQHO_NONE 作为 Hobj 的值。

MQOPEN(...);
打开预订队列并将队列句柄保存在 Hobj 中。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
使用从 MQOPEN 传递的 Hobj（或者，如果没有预订队列名，那么将使用 MQHO_NONE）来打开预订。 可以在不使用 MQOPEN 显式地打开非受管队列的情况下将其恢复。

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
使用预订句柄来关闭预订。 根据该预订是否持久预订，将使用隐式的 MQCO_KEEP_SUB 或 MQCO_REMOVE_SUB 将其关闭。 您可以使用 MQCO_REMOVE_SUB 来关闭持久预订，但无法使用 MQCO_KEEP_SUB 来关闭非持久预订。 MQCO_REMOVE_SUB 的操作是移除该预订，这将停止将任何更多发布内容发送到预订队列。

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
如果该预订是非受管预订，那么不执行任何特殊操作。 如果该队列是受管队列，并且使用显式或隐式的 MQCO_REMOVE_SUB 来关闭预订，那么将从该队列中清除所有发布内容并在此时删除该队列。

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
确保接收到的消息是针对我们预订的消息。

此示例的结果演示了发布/预订的各个方面：

在第 767 页的图 82 中，此示例先在主题 NYSE/IBM/PRICE 上发布 130。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

图 *82:* 将 *130* 发布到 *NYSE/IBM/PRICE*

在第 767 页的图 83 中，使用缺省参数执行示例时，将接收到保留式发布内容 130。 提供的主题对象和主题字符串将被忽略，如第 768 页的图 87 所示。 如果提供了预订对象，并且主题字符串不可变，那么将始终从该预订对象获取主题对象和主题字符串。 此示例的实际行为取决于对 MQSO_CREATE、MQSO_RESUME 和 MQSO_ALTER 的选择和组合。 在此示例中，选择的选项是 MQSO_RESUME。

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

图 *83:* 接收保留式发布内容

在第 767 页的图 84 中，不接收任何发布内容，这是因为，持久预订已接收到保留式发布内容。 在此示例中，通过只提供预订名而不提供队列名来恢复预订。 如果提供了队列名，那么将先打开该队列，然后将句柄传递到 MQSUB。

注: MQINQ 发生错误 2038 的原因是，MQSUB 对 STOCKTICKER 执行的隐式 MQOPEN 未包括 MQOO_INQUIRE 选项。 请通过显式地打开队列来避免 MQINQ 发出返回码 2038。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

图 *84:* 恢复预订

在第 768 页的图 85 中，此示例在使用 STOCKTICKER 作为目标的情况下创建非持久非受管预订。 由于这是新预订，因此它将接收到保留式发布内容。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

图 *85:* 使用新的非受管非持久预订来接收保留式发布内容

在第 768 页的图 86 中，为了演示重叠预订，发送了另一个发布内容，从而更改保留式发布内容。 接着，通过不提供预订名来创建新的非持久非受管预订。 保留式发布内容将被接收两次，即，为新预订接收一次，并为仍在 STOCKTICKER 队列中处于活动状态的持久 IBMSTOCKPRICESUB 预订接收一次。 此示例是一个例证，它是包含预订的队列，而不是应用程序。 尽管应用程序的此调用不引用 IBMSTOCKPRICESUB 预订，但应用程序仍接收到发布内容两次：从以管理方式创建的持久预订接收一次，从由应用程序本身创建的非持久预订接收一次。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

图 *86:* 重叠预订

在第 768 页的图 87 中，此示例演示：提供新主题字符串和现有预订不会导致更改预订。

1. 在第一种情况下，Resume 将按您的期望采用现有预订并忽略已更改的主题字符串。
2. 在第二种情况下， Alter 会导致错误 RC = 2510, Topic not alterable。
3. 在第三个示例中， Create 导致错误 RC = 2432, Sub already exists。

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

图 *87:* 无法更改预订主题

相关概念

第 753 页的『示例 1：MQ 发布内容使用者』
MQ 发布内容使用者是一个 IBM MQ 消息使用者，它不预订主题本身。

第 756 页的『示例 2：受管 MQ 订户』
受管 MQ 订户是大多数订户应用程序的首选模式。 此示例不需要队列、主题或预订的管理定义。

第 747 页的『编写发布者应用程序』

通过研究两个示例来开始编写发布者应用程序。 第一个示例尽可能地按照将消息放入队列的点到点应用程序进行建模，第二个示例动态地创建主题 - 这是更为常见的发布者应用程序模式。

发布/预订生命周期

在设计发布/预订应用程序时，请考虑主题、预订、订户、发布内容、发布者和队列的生命周期。

对象（例如预订）的生命周期从该对象被创建时开始，并在该对象被删除时结束。 生命周期还可以包括它所经历的其他状态和更改，例如临时暂挂、包含父主题和子主题、到期以及删除。

传统上，IBM MQ 对象（例如队列）是以管理方式创建的，或者由管理程序使用可编程命令格式 (PCF) 创建。 发布/预订的区别是，它提供了用于创建和删除预订的 MQSUB 和 MQCLOSE API 动词，具有受管预订这一概念（不仅创建和删除队列，而且清除未使用的消息），并且，在以管理方式创建的主题对象与通过程序或以管理方式创建的主题字符串之间存在关联。

功能方面的丰富使您能够满足各种各样的发布/预订需求，并且还能简化某些常用发布/预订应用程序模式的设计。 例如，受管预订能够简化对那些只应该与其创建程序具有相同生命周期的预订进行的编程和管理。 非受管预订能够简化预订发布内容与使用发布内容之间存在松散连接时的编程。 如果模式是根据中央化控制模型将发布内容流量路由到使用者（例如，将航班信息发送到自动登机门），那么集中创建的预订非常有用；但是，如果门卫负责通过在登机门处输入航班号来预订该航班的乘客记录，那么可以使用通过程序创建的预订。

在这个示例（最后一个）中，可能适合使用托管式持久预订：之所以选择“托管式”预订，是因为需要非常频繁地创建预订，并在登机门关闭并且能够以编程方式移除预订时具有明确的端点； 之所以选择“持久”预订，是为了避免由于登机门订户程序因某种原因停止而丢失乘客记录 [8] 要开始将乘客记录发布到登机门，一种可能的设计是让登机门应用程序使用登机门号码预订乘客记录，并使用登机门号码发布登机门打开事件。 发布者通过发布乘客记录来响应登机门打开事件 — 接着，还可以将这些记录发送给其他感兴趣的相关方（例如开帐单）以记录正在登机，或者发送给客户服务以便向乘客的移动电话发送有关登机门编号的文本通知。

集中管理的预订可以使用持久非受管模型，从而使用每个登机门的预定义队列将乘客列表路由到登机门。

下面这三个发布/预订生命周期示例演示受管非持久、受管持久和非受管持久订户如何与预订、主题、队列、发布者和队列管理器进行交互以及如何在管理程序与订户程序之间划分职责。

受管非持久订户

第 770 页的图 88 显示了一个应用程序，此应用程序创建受管非持久预订、获取两条发布到该预订所标识主题的消息并终止。 以灰色斜体字体标记并带有虚线箭头的交互是隐式交互。

请注意下面这些要点。

1. 此应用程序对一个已经两次作为发布目标的主题创建预订。 当订户接收到它的第一份发布内容时，它将接收到第二份发布内容，该发布内容当前是保留式发布内容。
2. 队列管理器将创建临时预订队列，并且将为主题创建预订。
3. 此预订具有到期时间。 此预订到期后，不会将该主题的更多发布内容发送到此预订，但订户将继续获取在此预订到期前发布的消息。 发布内容到期不受预订到期影响。
4. 第四份发布内容不会被放入预订队列，因此最后一个 MQGET 不会返回发布内容。
5. 虽然订户将关闭它的预订，但不会关闭它与队列或队列管理器的连接。
6. 在应用程序终止后不久，队列管理器将执行清除操作。 由于此预订是受管的非持久预订，因此预订队列将被删除。

[8] 当然，发布者必须将乘客记录作为持久消息发送，以避免发生其他可能的故障。

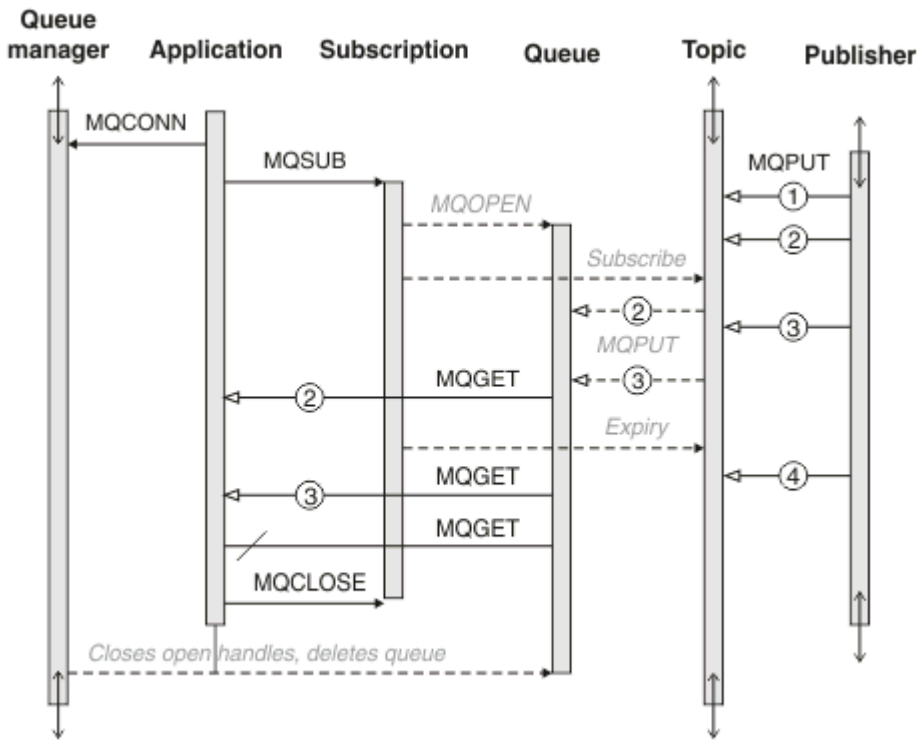


图 *88:* 受管非持久订户生命线

受管持久订户

受管持久订户相对于上一个示例更进一步，并显示了不会由于预订应用程序终止和重新启动而丢失的受管预订。

请注意下面这些新要点。

1. 与上一个示例不同，在此示例中，在预订中定义发布内容主题之前，该主题不存在。
2. 订户第一次终止时，它将使用 MQCO_KEEP_SUB 选项来关闭预订。 对于隐式地关闭受管持久预订而言，这是缺省行为。
3. 订户恢复预订时，将重新打开预订队列。
4. 在重新打开该队列之前放入该队列的新发布内容 2 可供 MQGET 使用，即使在该预订被除去后亦如此。

   尽管此预订可持久，但仅当此预订可持久并且消息也持久时，订户才能可靠地接收到发布者所发送的所有消息。 消息持久性取决于发布者所发送消息的 MQMD 中的 Persistent 字段设置。 订户无法对此进行控制。
5. 使用标志 MQCO_REMOVE_SUB 来关闭预订将除去该预订，从而停止将更多发布内容放入预订队列。 关闭预订队列时，队列管理器将除去未读取的发布内容 3，然后删除该队列。 此操作相当于以管理方式删除该预订。

   注: 请不要手动地删除该队列或者在指定选项 MQCO_DELETE 或 MQCO_PURGE_DELETE 的情况下发出 MQCLOSE。 受管预订的可视实现详细信息不是受支持的 IBM MQ 接口的组成部分。 除非队列管理器具有完全的控制权，否则无法可靠地管理预订。

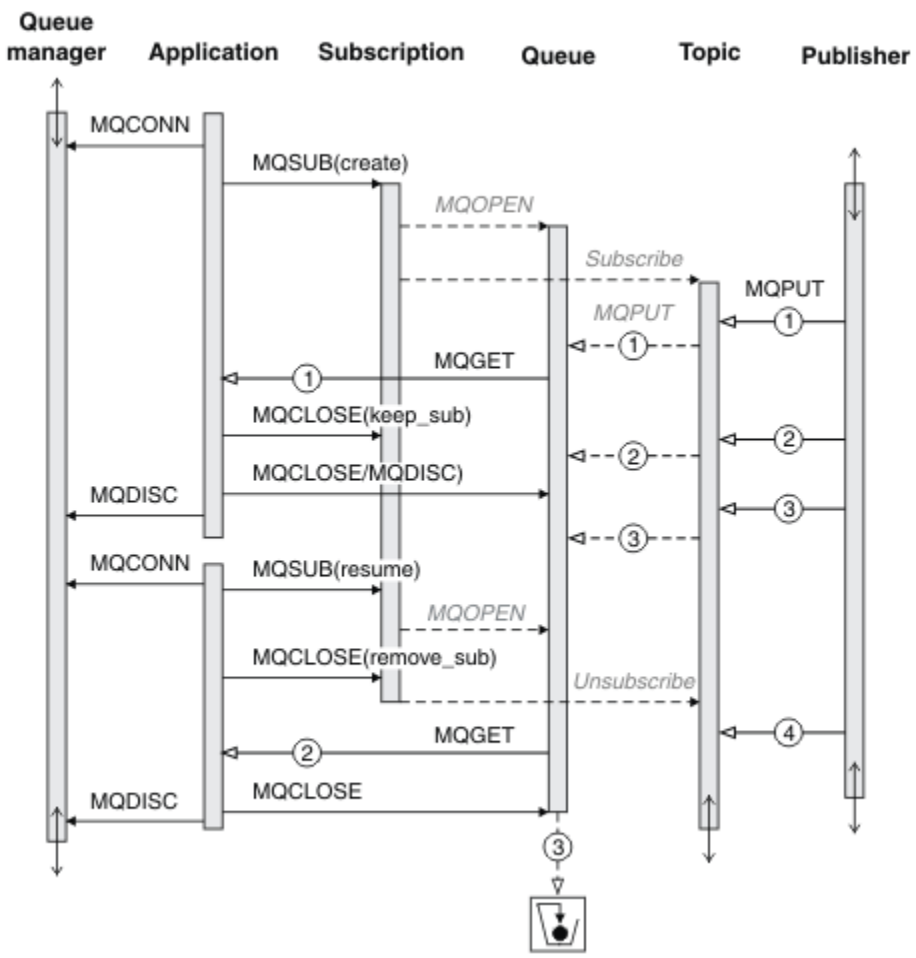


图 *89:* 受管持久订户生命线

非受管持久订户

在第三个示例“非受管持久订户”中，添加了管理员。 此示例能够很好地说明管理员如何与发布/预订应用程序进行交互。

要注意的事项如下所示。

1. 发布者将消息 1 放至一个主题，以后，该主题将与用于预订的主题对象相关联。 此主题对象定义了主题字符串，该主题字符串通过使用通配符与发布到的主题相匹配。
2. 此主题具有保留式发布内容。
3. 管理员创建主题对象、队列和预订。 必须在定义预订之前定义主题对象和队列。
4. 应用程序打开与此预订相关联的队列并将该队列的句柄传递到 MQSUB。 另外，它也可以简单地打开此预订并向其传递队列句柄 MQHO_NONE。 反之则不成立，它无法在只向预订传递队列句柄而不传递预订名的情况下恢复预订 — 一个队列可能包含多个预订。
5. 应用程序使用 MQSO_RESUME 选项来打开预订，即使是第一次打开该预订亦如此。 这是恢复以管理方式创建的预订。
6. 订户将接收到保留式发布内容 1。 发布内容 2 虽然是在任何发布内容被订户接收之前发布的，但是在预订启动后发布的，并且是预订队列中的第二份发布内容。

   注: 如果保留式发布内容未作为持久消息发布，那么它将在队列管理器重新启动后丢失。
7. 在此示例中，预订可持久。 程序可以创建非受管非持久预订；显而易见，管理员无法完成此任务。

8. 关闭预订时，选项 MQCO_REMOVE_SUB 的效果是除去该预订，就像管理员已将其删除一样。 这会停止进一步将发布内容发送到此队列，但与受管持久预订不同，这不会影响已在队列中的发布内容，即使关闭队列时亦如此。
9. 然后，管理员删除其余消息 3 并删除该队列。

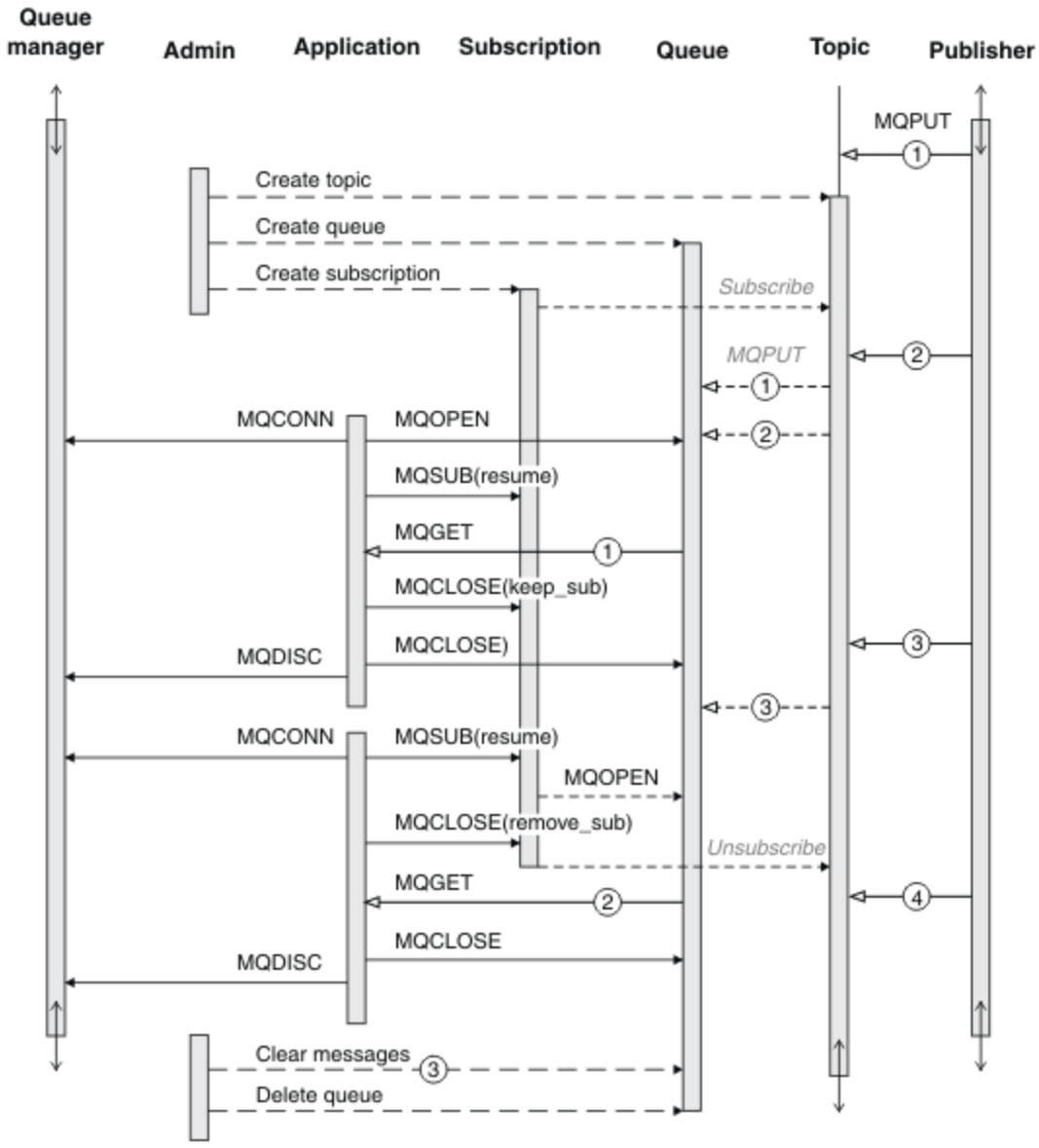


图 *90:* 非受管持久订户生命线

非受管预订的正常模式是，队列和预订的维护工作由管理员执行。 通常，用户不会尝试在应用程序代码中通过程序来模仿受管订户的行为以及整理队列和预订。 如果您发现需要编写管理逻辑，请确定能否使用受管模式来实现相同的结果。 编写紧密同步并完全可靠的管理代码并不容易。 以后，当您确定可以简单地删除消息、预订和队列而不必考虑其状态时，以手动方式或使用自动化管理程序进行整理更为简单。

发布/预订消息属性

多个消息属性与 IBM MQ 发布/预订消息传递相关。

PubAccountingToken

对于所有与此预订匹配的发布内容消息，其消息描述符 (MQMD) 的 AccountingToken 字段都将包含此值。 AccountingToken 是消息的身份上下文的组成部分。 有关消息上下文的更多信息，请参阅第 39 页的『消息上下文』。 有关 MQMD 中的 AccountingToken 字段的更多信息，请参阅 AccountingToken。

PubApplIdentityData

对于所有与此预订匹配的发布内容消息，其消息描述符 (MQMD) 的 ApplIdentityData 字段都将包含此值。 ApplIdentityData 是消息的身份上下文的组成部分。 有关消息上下文的更多信息，请参阅第 39 页的『消息上下文』。 有关 MQMD 中的 ApplIdentityData 字段的更多信息，请参阅 ApplIdentityData。

如果未指定 MQSO_SET_IDENTITY_CONTEXT 选项，那么在每条为此预订发布的消息中设置的 ApplIdentityData 都是空白，即缺省上下文信息。

如果指定了 MQSO_SET_IDENTITY_CONTEXT 选项，那么 PubApplIdentityData 由用户生成，此字段是输入字段并包含要在此预订的每份发布内容中设置的 ApplIdentityData。

PubPriority

对于所有与此预订匹配的发布内容消息，其消息描述符 (MQMD) 的 Priority 字段都将包含此值。 有关 MQMD 中的 Priority 字段的更多信息，请参阅 Priority。

此值必须大于或等于零；零是最低优先级。 并且，还可以使用下列特殊值：

- MQPRI_PRIORITY_AS_Q_DEF — 如果在 MQSUB 调用中的 Hobj 字段中提供了预订队列，并且该队列不是受管句柄，那么消息的优先级由此队列的 DefProirity 属性确定。 如果标识的队列是集群队列，或者队列名解析路径中有多个定义，那么优先级在发布内容消息被放入队列时确定，如 MQMD 中的优先级所述。 如果 MQSUB 调用使用了受管句柄，那么消息的优先级由所预订主题的相关联模型队列的 DefPriority 属性确定。
- MQPRI_PRIORITY_AS_PUBLISHED — 消息的优先级是原始发布内容的优先级。 这是此字段的初始值。

SubCorrelId

注意: 只能在发布/预订集群中的队列管理器之间传递相关标识，而不能在层次结构中的队列管理器之间传递该标识。

所发送的所有与此预订匹配的发布内容都将在消息描述符中包含此相关标识。 如果多个预订使用同一个队列来从中获取发布内容，那么按相关标识使用 MQGET 将只允许获取特定预订的发布内容。 此相关标识可以由队列管理器或用户生成。

如果未指定 MQSO_SET_CORREL_ID 选项，那么相关标识由队列管理器生成，此字段是输出字段并包含将在每条为此预订发布的消息中设置的相关标识。

如果指定了 MQSO_SET_CORREL_ID 选项，那么相关标识由用户生成，此字段是输入字段并包含要在此预订的每份发布内容中设置的相关标识。 在这种情况下，如果此字段包含 MQCI_NONE，那么在为此预订发布的每条消息中设置的相关标识将是最初放置该消息时创建的相关标识。

如果指定了 MQSO_GROUP_SUB 选项，并且指定的相关标识与正在使用同一队列和重叠主题字符串的现有分组预订相同，那么将仅随该发布内容的副本一起提供该组中最重要的预订。

SubUserData

这是预订用户数据。 在此字段中为预订提供的数据将作为发送到此预订的每份发布内容的 MQSubUserData 消息属性。

发布属性

第 774 页的表 128 列出了随发布消息提供的发布属性。

您可以从 **MQRFH2** 文件夹直接访问这些属性，或者使用 MQINQMP 来检索这些属性。 MQINQMP 接受属性名称或 **MQRFH2** 名称作为要查询的属性的名称。

表 128: 发布属性			
属性名	**MQRFH2 名称**	**类型**	**描述**
MQTopicString	mqps.Top	MQTYPE_STRING	主题字符串
MQSubUserData	mqps.Sud	MQTYPE_STRING	订户用户数据
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	保留发布
MQPubOptions	mqps.Pub	MQTYPE_INT32	发布选项
MQPubLevel	mqps.Pbl	MQTYPE_INT32	发布内容级别
MQPubTime	mqpse.Pts	MQTYPE_STRING	发布时间
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	发布序列号
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	由发布者添加的字符串/整数数据
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	消息格式: MQRFH1 MQRFH2 PCF

消息排序

对于特定的主题，消息由队列管理器按从发布应用程序接收它们的顺序发布（根据消息优先级进行重新排序）。

消息排序通常意味着，每个订户都将从特定队列管理器接收到来自特定发布者的有关特定主题的消息，并按该发布者发布那些消息的顺序接收。

但是，对于所有 IBM MQ 消息，消息的传递有可能偶尔失序。 在下列情况下，可能会发生此问题:

- 如果网络中的链路断开，那么后续消息将沿另一链路重新路由。
- 某个队列暂时变满或者被禁止放入消息，以致一条消息被放入死信队列并因此延迟，而后续消息仍直接传递。
- 如果管理员在发布者和订户仍在运行期间删除队列管理器，将导致已排队的消息被放入死信队列，并且导致预订中断。

如果不可能发生这些情况，那么发布内容始终按顺序传递。

注: 无法将已分组的消息或已分段的消息与“发布/预订”配合使用。

拦截发布内容

在发布内容到达任何其他订户之前，您可以对其进行拦截、修改并重新发布，

您可能想在发布内容到达订户前对其进行拦截，以便执行下列其中一项操作:

- 对消息附加其他信息
- 阻塞消息
- 变换消息

您可以对每条消息执行相同的操作，也可以根据预订、消息或消息头来执行不同的操作。

相关参考

MQ_PUBLISH_EXIT - 发布出口

预订级别

设置预定的预定级别以在发布内容到达最终订户前对其进行拦截。 拦截订户可在较高的预定级别进行预定，并在较低的发布级别重新发布。 构建拦截订户链，以在将发布内容交付至最终订户之前对发布内容执行消息处理。

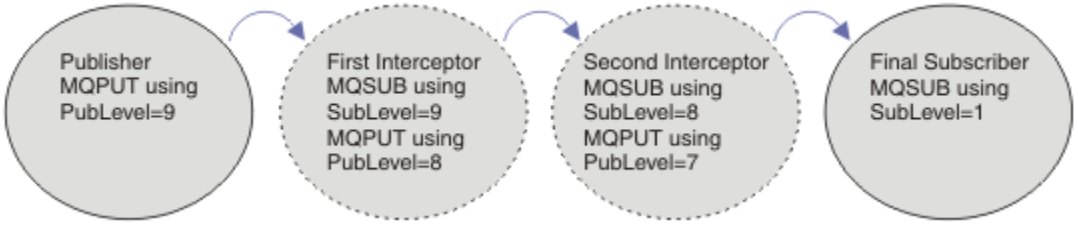


图 *91:* 拦截订户的顺序

要拦截发布内容，请使用 **MQSD** SubLevel 属性。 拦截消息后，可以通过更改 **MQPMO** PubLevel 属性将其转换并以更低的发布内容级别重新发布。 随后，此消息将转至最终订户，或者由中间订户在更低的预定级别再次拦截。

拦截订户通常在重新发布消息前会将其进行转换。 拦截订户序列组成消息流。 或者，您可能不想重新发布拦截的发布内容：更低预定级别的订户将不会收到消息。

请确保拦截器在任何其他订户之前先接收到发布内容。 将拦截器的预定级别设置为高于其他订户的预定级别。 缺省情况下，订户的**子级别**为 1。 最大值为 9。 发布必须以至少与最高**子级别**一样高的**发布级别**开始。 最初请使用 PubLevel 的缺省值 9 来进行发布。

- 如果对于一个主题，如果有一个拦截订户，请将 SubLevel 设置为 9。
- 如果对于一个主题有多个拦截应用程序，请为后续每个拦截订户设置更低的 SubLevel。
- 最多可以实现 8 个拦截应用程序，预订级别从 9 降到 2（含）。 消息的最终接收方的 SubLevel 为 1。

等于或低于发布内容的 PubLevel 的最高预定级别的拦截器会首先收到发布内容。 在特定预定级别仅为每个主题配置一个拦截订户。 在特定预定级别有多个订户会导致将发布内容的多个副本发送到最终预定应用程序集。

SubLevel 为 0 的订户将用作为回收器。 如果没有最终订户收到消息，那么它会接收到发布内容。 SubLevel 为 0 的订户可用于监视没有其他订户接收的发布内容。

对拦截订户进行编程

使用预订选项，如第 775 页的表 129 中所述。

表 *129:* 拦截用户的预订选项

预订选项	**注意**
MQSO_SET_CORREL_ID 和 SubCorrelId 设置为 MQCI_NONE	保留拦截的发布内容的 CorrelId。使其与原始发布内容相同。 **注:** 不能在层次结构中传递发布内容的相关标识。 该字段供队列管理器使用。
PubPriority 设置为 MQPRI_PRIORITY_AS_PUBLISHED	保留拦截的发布内容的优先级，使其与原始发布内容相同。

第 775 页的表 129 中的选项必须由所有拦截订户使用。 从而确保不能从原始发布者的设置中修改相关标识和消息优先级。

当拦截订户已处理发布内容后，它会将消息发布到 PubLevel 比其自己的预定的 SubLevel 低一级的同一主题。 如果拦截订户将 SubLevel 设置为 9，那么它会使用 PubLevel 8 来重新发布此消息。

要正确重新发布消息，需要来自原始发布内容的多项信息。 复用与原始消息相同的 **MQMD**，并设置 MQPMO_PASS_ALL_CONTEXT 以确保将 **MQMD** 中的所有信息都传递到下一个订户。 将第 776 页的表 130 中显示的消息属性中的值复制到重新发布的消息的对应字段中。 拦截订户可以更改这些值。 使用 OR 运算符可向 **MQPMO** 添加其他值。 Options 字段用于组合放入消息选项。

您必须显式打开发布内容队列，而不是使用受管发布内容队列。 不能为受管队列设置 MQSO_SET_CORREL_ID。 并且不能在受管队列上设置 MQOO_SAVE_ALL_CONTEXT。 请参阅第 776 页的『示例』中列出的代码片段。

表 *130:* 已重新发布的消息的 *MQPUT* 值	
使用 MQPUT 来重新发布消息	**发布内容消息中的信息**
MQOD. ObjectString	消息属性 MQTopicString
MQPMO. Options	消息属性 MQPubOptions

最终订户可以选择将其预定选项设置为其他选项。 例如，可以显式设置发布内容优先级，而不是设置为 MQPRI_PRIORITY_AS_PUBLISHED。 最终订户的设置仅影响来自链中最终拦截订户的发布内容。

保留的发布内容

保留的发布内容在受到拦截后必须保留，方法是将原始 put-message 选项复制到重新发布的消息中。

MQPMO_RETAIN 选项由发布者设置。 每个拦截订户都必须将 MQPubOptions 传输到重新发布的消息的 put-message 选项，如第 776 页的表 130 中所示。 复制 put-message 选项会保留由原始发布者设置的选项，包括是否保留发布内容。

当发布内容完成沿拦截订户链向下发布并交付至最终订户后，最终会保留。 SubLevel 为 1 的新订户在请求保留的发布内容时，会收到这些内容，不会再进行拦截。 不会向 SubLevel 大于 1 的订户发送保留的发布内容。 因此，拦截订户链不会对保留的发布内容进行第二轮修改。

示例

示例是一些代码片段，这些代码片段可以组合以构建拦截订户。 撰写的代码较为简短，而非生产质量的代码。

第 776 页的图 92 中的预处理器伪指令定义了要从 MQINQMP MQI 调用所需的发布消息中提取的两个属性。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS         (MQPTR)(char*) "MQPubOptions",\
                                  0,\
                                  12,\
                                  MQVS_NULL_TERMINATED,\
                                  MQCCSI_APPL
#define      MQTOPICSTRING        (MQPTR)(char*) "MQTopicString",\
                                  0,\
                                  13,\
                                  MQVS_NULL_TERMINATED,\
                                  MQCCSI_APPL
```

图 *92:* 预处理器伪指令

第 777 页的图 93 列出了代码片段中使用的声明。 除突出显示的项外，此声明针对 IBM MQ 应用程序为标准声明。

突出显示的 Put 和 Get 选项已初始化为传递所有上下文。 突出显示的 MQTOPICSTRING 和 MQPUBOPTIONS 为针对预处理器伪指令中定义的属性名称的 MQCHARV 初始化程序。 这些名称会传递至 MQINQMP。

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

图 *93:* 声明

在第 778 页的图 94 中显示了声明中不轻易执行的初始化。 突出显示的值需要说明。

SYSTEM.NDURABLE.MODEL.QUEUE
在此示例中，并非使用 MQSUB 打开受管非持久预定，而是使用模型队列 SYSTEM.NDURABLE.MODEL.QUEUE 来创建临时动态队列。 其句柄将传递到 MQSUB。 通过直接打开队列，您可以保存所有上下文，并设置预定选项 MQSO_SET_CORREL_ID。

MQGMO_CURRENT_VERSION
重要的是使用大部分 IBM MQ 结构的最新版本。 仅在控制结构的最新版本中才提供 gmo.MsgHandle 之类的字段。

MQGMO_PROPERTIES_IN_HANDLE
原始发布内容中设置的主题字符串和放入消息选项将由拦截订户使用消息属性来检索。 替代方法是直接在消息中读取 **MQRFH2** 结构。

MQSO_SET_CORREL_ID
使用 MQSO_SET_CORREL_ID 与以下选项相结合：

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

这些选项的效果是传递相关标识。 由原始发布者设置的相关标识将放入由拦截订户接收到的发布内容的相关标识字段中。 每个拦截订户都传递相同的相关标识。 随后，最终订户可以选择接收相同的相关标识。

注: 如果通过发布/预订层次结构来传递发布内容，那么将从不保留相关标识。

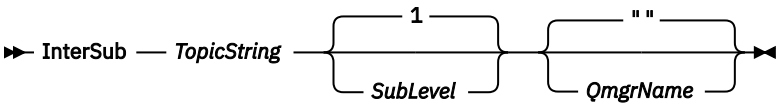
MQPRI_PRIORITY_AS_PUBLISHED
发布内容置于与其发布时处于相同消息优先级的发布内容队列中。

```
    strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
            sizeof(getOD.ObjectName));
    gmo.Version                = MQGMO_VERSION_4;
    gmo.Options                = MQGMO_WAIT
                               | MQGMO_PROPERTIES_IN_HANDLE
                               | MQGMO_CONVERT;
    gmo.WaitInterval           = 30000;
    sd.Options                 = MQSO_CREATE
                               | MQSO_FAIL_IF_QUIESCING
                               | MQSO_SET_CORREL_ID;
    sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
    sd.Version                 = MQSD_VERSION_1;
    memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
    putOD.ObjectType             = MQOT_TOPIC;
    putOD.ObjectString.VSPtr     = &TopStrBuf;
    putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
    putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
    putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
    putOD.Version                = MQOD_VERSION_4;
    pmo.Version                = MQPMO_VERSION_3;
```

图 *94:* 初始化

第 779 页的图 95 显示了读取命令行参数、完成初始化和创建拦截预定的代码片段。

使用以下命令运行程序：



为了使错误处理尽可能不明显，来自每个 MQI 调用的原因码会存储在不同数组元素中。 测试每个调用的完成代码后，如果值为 MQCC_FAIL，那么控制会退出 `do { } while(0)` 代码块。

以下两行代码值得注意：

`pmo.PubLevel = sd.SubLevel - 1;`
将已重新发布的消息的发布内容级别设置为比拦截订户的预定级别低一级。

`gmo.MsgHandle = Hmsg;`
提供消息句柄，以供 MQGET 返回消息属性。

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

图 *95:* 准备拦截发布内容

主要代码片段第 780 页的图 96 会从发布内容队列获取消息。 它会使用发布内容的主题字符串和原始 **MQPMO**. option 属性查询消息属性并重新发布消息。

在此示例中，不对发布内容执行任何转换。 重新发布的发布内容的主题字符串与拦截订户预定的主题字符串始终匹配。 如果拦截订户负责拦截发送到发布同一个内容队列的多个预定，那么可能需要查询主题字符串以区分匹配不同预定的发布内容。

突出显示了对 MQINQMP 的调用。 主题字符串和发布内容放入消息选项直接写入输出控制结构。 将 putOD.ObjectString 的 MQCHARV 长度从显式长度更改为 null terminated 字符串的唯一原因是使用 printf 来输出字符串。

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

图 *96:* 拦截发布内容并重新发布

最终代码片段如第 780 页的图 97 中所示。

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

图 *97:* 完成

拦截发布和分布式发布/预订

将拦截订户或发布出口部署到分布式发布/预订拓扑时，请遵循简单模式。 在与发布者相同的队列管理器上部署拦截订户，在与最终订户相同的队列管理器上部署发布出口。

第 781 页的图 98 显示了已发布的预定集群中连接的两个队列管理器。 发布者在发布级别 9 为集群主题创建发布。 带编号的箭头显示了发布在流向集群主题订户时所执行的步骤序列。 该发布由**子级别**为 9 的订户拦截，并在**发布级别** 8 重新发布。 它由**子级别为** 8 的订户再次拦截。 该订户在**发布级别** 7 重新发布。 队列管理器提供的代理订户将发布转发至队列管理器 B，其中除了最终订户外，还部署了发布出口。 发布出口在**子级别**为 1 的最终订户接收发布之前处理该发布。 将以虚线显示拦截订户和发布出口。

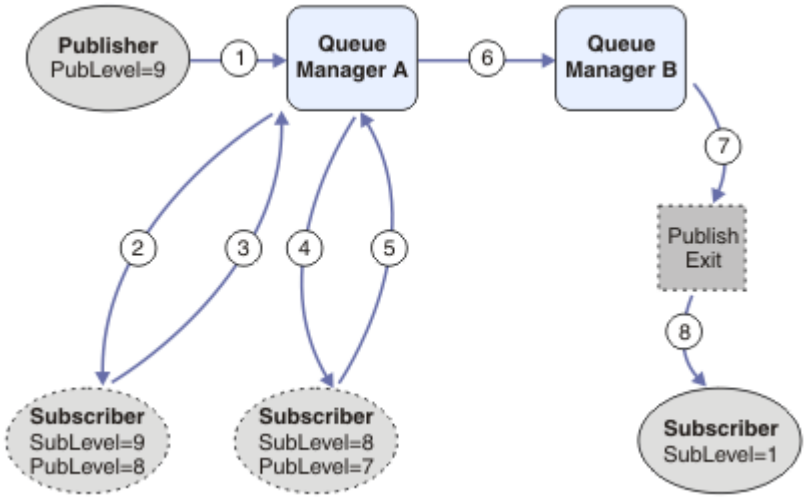


图 *98:* 集群中的拦截和发布出口

简单模式的目的是使每个接收发布内容的订户都可接收到相同的发布内容。 无论订户连接于何处，发布内容经过的转换顺序都是相同的。 根据发布者或最终订户的连接位置，您可能想要避免转换顺序的变化。 合理的例外情况是对最终交付给每个订户的发布内容进行定制。 使用发布出口来基于发布内容最终交付到的队列定制发布内容。

您必须仔细考量拦截订户和发布出口在分布式发布/预订拓扑中的部署位置。 简单模式会将拦截订户部署到与发布者相同的队列管理器，并将发布出口部署到与最终订户相同的队列管理器。

反模式

第 781 页的图 99 显示了不遵循简单模式的情况下可能出现的偏差。 为使部署变得更复杂，将最终订户添加到队列管理器 A 中，并将两个额外拦截订户添加到队列管理器 B 中。

该发布将转发到**发布级别** 7 的队列管理器 B，它由**子级别**为 5 的订户拦截，然后由**子级别**为 1 的最终订户使用。 发布出口拦截发布，然后将其传递到拦截使用者和队列管理器 B 中的最终使用者。 发布在未经发布出口处理的情况下到达队列管理器 A 的最终订户。

在发布/预订拓扑中，代理订户使用 SubLevel 1 进行预定，并使用最后一个拦截订户设置的 PubLevel 进行传递。 在第 781 页的图 99 中，结果是位于队列管理器 B 的 SubLevel 为 9 的订户未拦截发布内容。

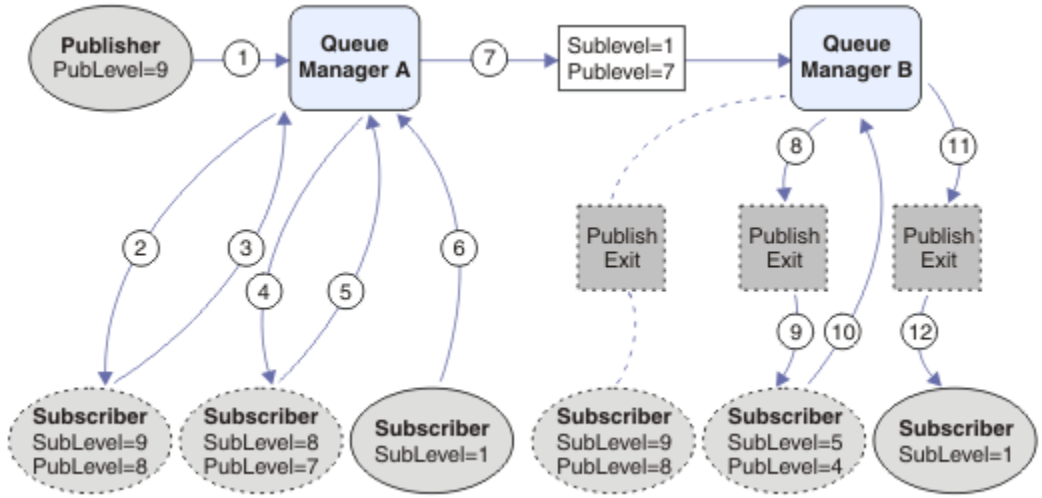


图 *99:* 拦截订户的复杂部署

发布选项

您可以通过多个选项来控制消息的发布方式。

抑制来自订户的应答信息

如果您不希望订户能够应答它们所接收到的发布内容，那么可以使用 MQPMO_SUPPRESS_REPLYTO put-message 选项来禁止 MQMD 的 ReplyToQ 和 ReplyToQmgr 字段中的信息。 如果使用此选项，那么队列管理器将在接收到发布内容时从 MQMD 中除去该信息，然后再将该发布内容转发给任何订户。

此选项不能与需要 ReplyToQ 的报告选项配合使用，如果尝试这么做，那么此调用将失败并返回 MQRC_MISSING_REPLY_TO_Q。

发布内容级别

使用发布内容级别是一种用于控制要让哪些订户接收发布内容的方法。 发布内容级别指定作为发布内容目标的预订的级别。 只有那些最高预订级别低于或等于发布内容的发布内容级别的预订才会接收到该发布内容。 此值必须在 0 到 9 之间；0 是最低的发布内容级别。 此字段的初始值为 9。发布和预订级别的一种用途是拦截发布。

检查是否未将发布内容传递到任何订户

要检查是否未将某份发布内容传递到任何订户，请将 MQPMO_WARN_IF_NO_SUBS_MATCHED 放置消息选项与 MQPUT 调用配合使用。 如果放置操作返回了完成码 MQCC_WARNING 和原因码 MQRC_NO_SUBS_MATCHED，那么表示未将发布内容传递到任何预订。 如果对放置操作指定了 MQPMO_RETAIN 选项，那么将保留该消息并将其传递到后续定义的任何匹配预订。 在分布式发布/预订系统中，仅当队列管理器上没有为主题注册代理预订时，才会返回 MQRC_NO_SUBS_MATCHED 原因码。

预订选项

有多个选项可用于控制消息预定的处理方式。

消息持久性

队列管理器将维护它们向订户转发的发布内容的持久性（由发布者设置）。 发布者将持久性设置为以下选项之一：

0
非持久

1
持久

2
具有队列/主题定义的持久性

对于发布/预订，发布者会将主题对象和 **topicString** 解析为已解析的主题对象。 如果发布者指定“持久性”作为队列/主题定义，那么将针对发布内容设置来自已解析主题对象的缺省持久性。

保留的发布内容

要控制保留发布内容的接收时间，订户可以使用两个预订选项：

仅在请求时发布 (MQSO_PUBLICATIONS_ON_REQUEST)

如果要让订户控制它接收发布内容的时间，那么可以使用 MQSO_PUBLICATIONS_ON_REQUEST 预订选项。 于是，订户可以通过使用 MQSUBRQ 调用来控制它接收发布内容的时间（指定原始 MQSUB 调用所返回的 Hsub 句柄），以便请求向其发送主题的保留式发布内容。 使用 MQSO_PUBLICATIONS_ON_REQUEST 预订选项的订户不会接收到任何非保留式发布内容。

如果指定了 MQSO_PUBLICATIONS_ON_REQUEST，那么必须使用 MQSUBRQ 来检索任何发布内容。 如果未使用 MQSO_PUBLICATIONS_ON_REQUEST，那么您将在消息发布时获取那些消息。

如果订户使用 MQSUBRQ 调用并在预订的主题中使用通配符，那么该预订可能与主题树中的多个主题或节点匹配，它们的所有保留式消息（如果存在）都将被发送到该订户。

此选项与持久预订配合使用时可能特别有用，这是因为，队列管理器会继续将发布内容发送给订户（如果该订户以持久方式进行预订），即使该订户应用程序未处于运行状态亦如此。 这可能会导致在订户队列中积压消息。 如果订户使用 MQSO_PUBLICATIONS_ON_REQUEST 选项进行注册，那么可以避免此

积压问题。 另外，如果非持久预订适合于应用程序，那么还可以使用非持久预订来避免不想要的消息积压。

对于持久预订，如果发布者使用保留式发布内容，那么订户应用程序在重新启动后可以使用 MQSUBRQ 调用来刷新其状态信息。 然后，订户必须定期使用 MQSUBRQ 调用来刷新其状态。

使用此选项的 MQSUB 调用的结果是，不发送任何发布内容。 如果将原始预订配置为使用 MQSO_PUBLICATIONS_ON_REQUEST 选项，那么在断开连接后恢复的持久预订将使用此选项。

仅限于新发布内容 (MQSO_NEW_PUBLICATIONS_ONLY)

如果存在某个主题的保留式发布内容，那么任何在该发布内容发布后进行预订的订户都将接收到该发布内容的副本。 如果订户不想接收任何在进行预订前发布的发布内容，那么该订户可以使用 MQSO_NEW_PUBLICATIONS_ONLY 预订选项。

对预订进行分组

如果您设置了一个队列来接收发布内容，并且有许多重叠的预订将发布内容馈送到同一个队列，那么请考虑对预订进行分组。 这种情况类似于重叠预定中的示例。

可以通过在预订主题时设置 MQSO_GROUP_SUB 选项来避免接收重复的发布内容。 结果是，当组中的多个预订与某个发布内容的主题匹配时，只有一个预订负责将该发布内容放入队列。 其他与该发布内容主题匹配的预订都将被忽略。

按以下标准选择负责将发布内容放入队列的预订：在遇到任何通配符之前，匹配主题字符串最长。 可以将其想像成最接近的匹配预订。 它的属性将传播到发布内容，包括它是否具有 MQSO_NOT_OWN_PUBS 属性。 如果它具有此属性，那么不会将任何发布内容传递到队列中，即使其他匹配的预订不具有 MQSO_NOT_OWN_PUBS 属性亦如此。

不能为了消除重复发布内容而将所有预订都放入一个组中。 分组的预订必须满足下列条件：

1. 没有任何预订是受管预订。
2. 一组预订将发布内容传递到同一个队列。
3. 各个预订必须处于同一预订级别。
4. 组中每个预订的发布内容消息具有相同的相关标识。

   要确保每个预订生成具有相同相关标识的发布内容消息，请设置 MQSO_SET_CORREL_ID 以便在发布内容中创建您自己的相关标识，并在每个预订中的 **SubCorrelId** 字段中设置相同的值。 不要将 **SubCorrelId** 设置为值 MQCI_NONE。

请参阅../com.ibm.mq.ref.dev.doc/q100080_.dita#q100080_/mqso_group_sub，以获取更多信息。

查询和设置对象属性

属性是用于定义 IBM MQ 对象特征的特性。

属性会影响队列管理器处理对象的方式。 对象的属性中详细描述了每种类型的 IBM MQ 对象的属性。

某些属性可在定义对象时设置，并且只能使用 IBM MQ 命令进行更改；此类属性的示例是放入队列的消息的缺省优先级。 其他属性受到队列管理器操作的影响并且可随时间而更改；例如，一个示例便是队列的当前深度。

您可以使用 MQINQ 调用查询大部分属性的当前值。 MQI 也提供一个 MQSET 调用，可供您用于更改某些队列属性。 您无法使用 MQI 调用来更改任何其他类型的对象的属性。 而您必须使用以下资源之一：

- ULW MQSC 工具，在 MQSC 命令中对此进行了描述。
- IBM i CHGMQMx CL 命令（在 IBM i 的 CL 命令参考中进行了描述）或 MQSC 工具。
- z/OS ALTER 操作程序命令（或者含有 REPLACE 选项的 DEFINE 命令），在 MQSC 命令中对这些命令进行了描述。

注: 在本文档中，以配合 MQINQ 和 MQSET 调用使用的格式来显示对象属性名称。 使用 IBM MQ 命令来定义、更改或显示属性时，必须使用主题链接中的命令描述中显示的关键字来识别属性。

MQINQ 和 MQSET 调用都使用选择器的数组来识别 要查询或设置的属性。 针对么使用的每个属性都存在一个选择器。 选择器名称包含由属性性质决定的前缀。

表 *131:* 选择器名称前缀

前缀	描述
MQCA_	这些选择器表示包含字符数据（例如，队列名称）的属性。
MQIA_	这些选择器表示包含数字值（例如，队列上的消息数 *CurrentQueueDepth*）或常量值（例如，表示队列管理器是否支持同步点的 *SyncPoint*）。

在使用 MQINQ 或 MQSET 调用之前，应用程序必须连接到队列管理器，并且必须使用 MQOPEN 调用来打开要设置或查询属性的对象。 这些操作在第 685 页的『连接到队列管理器和从队列管理器断开连接』和第 692 页的『打开和关闭对象』中进行了描述。

请使用以下链接来查找有关查询或设置对象属性的更多信息。

- 第 784 页的『查询对象的属性』
- 第 785 页的『MQINQ 调用失败的一些情况』
- 第 785 页的『设置队列属性』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

查询对象的属性

使用 MQINQ 调用来查询有关任何类型的 IBM MQ 的属性。

作为此调用的输入，必须提供：

- 连接句柄。
- 对象句柄。
- 选择器数量。

- 一个属性选择器数组，其中每个选择器的格式为 MQCA_* 或 MQIA_*。每个选择器都代表一个具有您要查询的值的属性，并且每个选择器必须适用于对象句柄所代表的对象类型。 您可以按任何顺序指定选择器。
- 要查询的整数属性的数量。 如果不查询整数属性，请指定零值。
- *CharAttrLength* 中字符属性缓冲区的长度。 该值必须至少为保存每个字符属性字符串所需的长度总和。 如果不查询字符属性，请指定零值。

来自 MQINQ 的输出为：

- 复制到此数组的一组整数属性值。 值的数量由 *IntAttrCount* 确定。 如果 *IntAttrCount* 或 *SelectorCount* 为零，那么不使用此参数。
- 在其中返回字符属性的缓冲区。 缓冲区的长度由 **CharAttrLength** 参数提供。 如果 *CharAttrLength* 或 *SelectorCount* 为零，那么不使用此参数。
- 完成代码。 如果完成代码提供警告，那么这表示调用仅部分完成。 在此情况下，请检查原因码。
- 原因码。 存在三种部分完成情况：
  - 选择器不适用于队列类型
  - 没有足够空间可供用于整数属性
  - 没有足够空间可供用于字符属性

  如果出现以上多种情况，那么将返回适用的第一种情况。

如果打开队列以获取输出或者进行查询，并且队列解析为非本地集群队列，那么只能查询队列名称、队列类型和公共属性。 公共属性值为使用 MQOO_BIND_ON_OPEN 时所选队列的属性值。 如果使用了 MQOO_BIND_NOT_FIXED 或 MQOO_BIND_ON_GROUP 或者如果使用了 MQOO_BIND_AS_Q_DEF 并且 **DefBind** 队列属性为 MQBND_BIND_NOT_FIXED，那么属性值为可能的集群队列的任意一个值。 请参阅 第 812 页的『MQOPEN 和集群』和 MQOPEN，以获取更多信息。

注: 此调用返回的值为所选属性的快照。 可在程序对所返回的值执行操作之前更改属性。

在 MQINQ 中提供了 MQINQ 调用的描述。

MQINQ 调用失败的一些情况

如果打开别名以查询其属性，那么将返回别名队列的属性（用于访问其他队列的 IBM MQ 对象），而不是基本队列的属性。

但是，队列管理器还会打开别名解析到的基本队列的定义，并且如果其他程序在 MQOPEN 和 MQINQ 调用期间更改了基本队列的使用，那么 MQINQ 调用将失败，并返回 MQRC_OBJECT_CHANGED 原因码。 如果别名队列对象的属性发生更改，那么调用也会失败。

同样，打开远程队列以查询其属性时，将仅返回远程队列的本地定义的属性。

如果指定一个或多个对于要查询的队列属性类型无效的选择器，那么 MQINQ 调用完成，但显示警告，并且设置如下输出：

- 对于整数属性，*IntAttrs* 的对应元素设置为 MQIAV_NOT_APPLICABLE。
- 对于字符属性，*CharAttrs* 字符串的对应部分设置为星号。

如果指定一个或多个对于要查询的对象属性类型无效的选择器，那么 MQINQ 调用失败，并返回 MQRC_SELECTOR_ERROR 原因码。

无法调用 MQINQ 以查看模型队列；请使用 MQSC 工具或者平台上可用的命令。

设置队列属性

使用此信息来了解如何使用 MQSET 调用设置队列属性。

只能使用 MQSET 调用设置以下队列属性：

- *InhibitGet*（对于远程队列不可用）
- *DistList*（在 z/OS 上不可用）
- *InhibitPut*
- *TriggerControl*

- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

MQSET 调用的参数与 MQINQ 调用的参数相同。 但是对于 MQSET，除完成代码和原因码以外的所有参数都是输入参数。 不存在部分完成的情况。

注: 不能使用 MQI 来设置除本地定义的队列以外的 IBM MQ 对象的属性。

有关 MQSET 调用的更多详细信息，请参阅 MQSET。

落实和回退工作单元

此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

在本主题中使用了以下术语:

- Commit
- 回退
- 同步点协调
- 同步点
- 工作单元
- 单阶段落实
- 两阶段落实

如果您熟悉这些事务处理术语，可以跳至 第 787 页的『IBM MQ 应用程序中同步点注意事项』。

落实和回退
当程序将一条消息放置在工作单元中的队列上时，仅当程序落实此工作单元时此消息才对其他程序可见。 要落实工作单元，所有更新都必须成功，才能保持数据完整性。 如果程序检测到错误，并且确定放置操作未永久生效，那么将回退工作单元。 当程序执行回退时，IBM MQ 将通过除去由此工作单元放置在队列上的消息来复原队列。 程序执行落实和回退的方式取决于程序运行的环境。

同样，当程序从工作单元中的队列获取消息时，此消息将保留在队列上直至程序落实工作单元为止，但是其他程序不可检索此消息。 当程序落实工作单元后，将永久删除此消息。 如果程序回退工作单元，IBM MQ 将通过使其他程序可检索这些消息来复原队列。

同步点协调、同步点、工作单元
同步点协调使工作单元落实或回退以确保数据完整性的过程。

最简单的示例是在事务结束时决定采取落实还是回退更改的操作。 但是，它更适用于在事务内其他逻辑点同步数据更改的应用程序。 这些逻辑点称为同步点（或同步点），处理两个同步点之间的一组更新的时间段称为工作单元。 可以有几个 MQGET 调用和 MQPUT 调用作为一个工作单元的一部分。

工作单元中的最大消息数可由 ALTER QMGR 命令的 MAXUMSGS 属性控制。

单阶段落实
单阶段落实是程序可将更新落实到队列而无需将其更改与其他资源管理器相协调的过程。

两阶段落实
两阶段落实是程序对 IBM MQ 队列进行的更新可与其他资源（例如，受 Db2 控制的数据库）的更新相协调的过程。 在此类过程下，将同时落实或回退对所有资源进行的更新。

为了帮助处理工作单元，IBM MQ 提供了 **BackoutCount** 属性。 每次回退工作单元中的一条消息时，该属性都会递增。 如果此消息重复导致工作单元异常结束，那么 *BackoutCount* 的值最终会超出 *BackoutThreshold* 的值。 在定义队列时设置该值。 在此情况下，该应用程序可以从工作单元中除去此消息，并将其放入其他队列，如 *BackoutRequeueQName* 中定义的那样。 移动消息时，可以落实工作单元。

使用以下链接来查找有关落实和回退工作单元的更多信息。

- 第 787 页的『IBM MQ 应用程序中同步点注意事项』

- z/OS 第 788 页的『IBM MQ for z/OS 应用程序中的同步点』
- IBM i 第 790 页的『CICS for IBM i 应用程序中的同步点』
- 第 790 页的『IBM MQ for Multiplatforms 中的同步点』
- IBM i 第 794 页的『到 IBM i 外部同步点管理器的接口』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

IBM MQ 应用程序中同步点注意事项

使用此信息来了解有关使用 IBM MQ 应用程序中的同步点。

以下环境支持两阶段落实：

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ for Linux
- Solaris IBM MQ for Solaris
- Windows IBM MQ for Windows
- z/OS CICS 事务服务器 z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS z/OS 批处理（含 RRS）
- 使用 X/Open XA 接口的其他外部协调程序

以下环境支持单阶段落实：

- IBM i IBM MQ for IBM i
- UNIX IBM MQ 开 UNIX
- Windows IBM MQ for Windows
- z/OS z/OS 批处理

有关外部接口的更多信息，请参阅第 793 页的『Multiplatforms 上到外部同步点管理器的接口』和由 The Open Group 发布的 XA 文档 *CAE Specification Distributed Transaction Processing: The XA Specification*。 事务管理器（如 CICS、IMS、Encina 和 Tuxedo）可在与其他可恢复资源协调的情况下参与两阶段落实。 这意味着可在工作单元范围内引入 IBM MQ 所提供的排队功能，并由事务管理器进行管理。

随 IBM MQ 提供的样本显示了协调兼容 XA 的数据库的 IBM MQ。 有关这些样本的更多信息，请参阅第 963 页的『使用 IBM MQ 样本过程程序』。

在 IBM MQ 应用程序中，可以在每个 put 和 get 调用上指定是否希望调用接受同步点控制。 要使 put 操作接受同步点控制，请在调用 MQPUT 时在 MQPMO 结构的 *Options* 字段中使用 MQPMO_SYNCPOINT 值。 对于 get 操作，请在 MQGMO 结构的 *Options* 字段中使用 MQGMO_SYNCPOINT 值。 如果不显式选择选项，那么缺省操作取决于平台：

- Multi 同步点控件缺省设置为 NO。
- z/OS 同步点控件缺省设置为 YES。

发出带有 MQPMO_SYNCPOINT 的 MQPUT1 调用时，缺省行为会发生更改，因此 put 操作将异步完成。 这可能导致依靠返回的 MQOD 和 MQMD 结构中的某些字段的部分应用程序行为发生更改（但是现在这些字段现在包含未定义的值）。 应用程序可以指定 MQPMO_SYNC_RESPONSE 以确保同步执行 put 操作，并且完成所有相应字段值。

当应用程序接收到 MQRC_BACKED_OUT 原因码以响应同步点下的 MQPUT 或 MQGET 时，该应用程序应使用 MQBACK 正常回退当前事务，然后（如果适用）重新尝试整个事务。 如果该应用程序接收到 MQRC_BACKED_OUT 以响应 MQCMIT 或 MQDISC 调用，那么无需调用 MQBACK。

每次回退 MQGET 调用时，受影响的消息的 MQMD 结构的 *BackoutCount* 字段都会递增。 *BackoutCount* 值较高表示某条消息已重复回退。 这可能表示此消息存在问题，您应当进行调查。 请参阅 BackoutCount，以获取 *BackoutCount* 的详细信息。

除了在 z/OS 批处理（含 RRS）上之外，如果程序在存在未落实的请求时发出 MQDISC 调用，那么将发生隐式同步点。 如果此程序异常终止，那么将发生隐式回退。

z/OS 在 z/OS 上，如果程序在未首先调用 MQDISC 的情况下异常终止，那么还将发生隐式同步点。 如果连接到 MQ 的 TCB 正常结束，那么此程序将被视为正常结束。 在 z/OS UNIX System Services and Language Environment (LE) 下运行时，将针对异常终止或信号调用缺省条件处理。 LE 条件处理程序会处理错误条件，TCB 正常结束。 在这些条件下，MQ 会落实工作单元。 有关更多信息，请参阅 Language Environment Condition Handling 简介。

z/OS 对于 IBM MQ for z/OS 程序，可以使用 MQGMO_MARK_SKIP_BACKOUT 选项来指定发生回退时不得回退消息（以避免 *MQGET-error-backout* 循环）。 有关使用该选项的信息，请参阅第 738 页的『跳过回退』。

对查询属性的更改（通过 MQSET 或通过命令）均不受到落实或回退工作单元的影响。

z/OS *IBM MQ for z/OS* 应用程序中的同步点

本主题说明了如何在事务管理器中使用同步点（CICS 和 IMS）和批处理应用程序。

z/OS *CICS Transaction Server for z/OS* 应用程序中的同步点

在 CICS 应用程序中，通过使用 EXEC CICS SYNCPOINT 命令来建立同步点。

要回退对先前同步点进行的所有更改，可以使用 EXEC CICS SYNCPOINT ROLLBACK 命令。 有关更多信息，请参阅《*CICS* 应用程序编程参考》。

如果在工作单元中涉及其他可恢复资源，那么队列管理器将与 CICS 同步点管理器一起参与两阶段落实协议；否则，队列管理器将执行单阶段落实流程。

如果 CICS 应用程序发出 MQDISC 调用，那么不生成隐式同步点。 如果应用程序正常关闭，那么将关闭任何已打开的队列，并发生隐式落实。 如果应用程序异常关闭，那么将关闭任何已打开的队列，并发生隐式回退。

z/OS *IMS* 应用程序中的同步点

在 IMS 应用程序中，使用 IMS 调用（如 GU（获取唯一））建立到 IOPCB 和 CHKP（检查点）的同步点。

要回退从先前检查点开始的所有更改，可以使用 IMS ROLB（回滚）调用。 有关更多信息，请参阅 IMS 文档。

如果在工作单元中涉及其他可恢复资源，那么队列管理器将与 IMS 同步点管理器一起参与两阶段落实协议。

所有打开的句柄由 IMS 适配器在同步点时关闭（除了批处理或非消息驱动的 BMP 环境）。 这是因为执行 MQCONN、MQCONNX 和 MQOPEN 调用（而非 MQPUT 或 MQGET 调用）时不同用户可以启动下一个工作单元，并且执行 IBM MQ 安全性检查。

但在等待输入 (WFI) 或伪等待输入 (PWFI) 环境中，只有收到下一条消息或者将 QC 状态码返回至应用程序之后，IMS 才会通知 IBM MQ 关闭句柄。 如果应用程序在 IMS 区域内等待，并且所有句柄都属于已触发队列，那么由于队列处于打开状态，将不会发生触发。 因此，WFI 或 PWFI 环境中运行的应用程序应在对下一条消息的 IOPCB 执行 GU 之前，对队列句柄执行显式 MQCLOSE。

如果 IMS 应用程序（BMP 或 MPP）发出 MQDISC 调用，那么将关闭打开的队列，但不会生成隐式同步点。 如果应用程序正常关闭，那么将关闭任何已打开的队列，并发生隐式落实。 如果应用程序异常关闭，那么将关闭任何已打开的队列，并发生隐式回退。

z/OS *z/OS* 批处理应用程序中的同步点

对于批处理应用程序，可以使用 IBM MQ 同步点管理调用：MQCMIT 和 MQBACK。 为了与早期版本兼容，CSQBCMT 和 CSQBBAK 可用作同义词。

注: 如果需要将更新落实或回退到不同资源管理器（例如，IBM MQ 和 Db2）所管理的资源，那么可在单个工作单元内使用 RRS。 有关更多信息，请参阅第 789 页的『事务管理和可恢复的资源管理器服务』。

使用 MQCMIT 调用落实更改

必须提供 MQCONN 或 MQCONNX 调用返回的连接句柄 (*Hconn*) 作为输入。

MQCMIT 的输出是完成代码和原因码。 如果同步点已完成但由于先前同步点导致队列管理器回退放置和获取操作，那么调用完成，但显示警告。

成功完成 MQCMIT 调用即指示队列管理器，应用程序已到达同步点，并且自从上一个同步点开始的所有放置和获取操作都已永久生效。

并非所有失败响应都表示 MQCMIT 未完成。 例如，应用程序可能会收到 MQRC_CONNECTION_BROKEN。

在 MQCMIT 中提供了 MQCMIT 调用的描述。

使用 MQBACK 调用回退更改

作为输入，必须提供连接句柄 (*Hconn*)。 使用 MQCONN 或 MQCONNX 调用返回的句柄。

MQBACK 的输出是完成代码和原因码。

此输出指示队列管理器，应用程序已到达同步点，并且自从上一个同步点开始的所有获取和放置操作都已回退。

在 MQBACK 中提供了 MQBACK 调用的描述。

事务管理和可恢复的资源管理器服务

事务管理和可恢复的资源管理器服务 (RRS) 是 z/OS 工具，用于在参与的资源管理器之间提供两阶段同步点支持。

应用程序可以更新由各种 z/OS 资源管理器（如 IBM MQ 和 Db2）管理的可恢复资源，然后作为单个工作单元落实或回退这些更新。 RRS 可在正常执行期间提供必要的工作单元状态日志记录、协调同步点处理，并在子系统重新启动期间提供相应的工作单元状态信息。

IBM MQ for z/OS RRS 参与者支持使批处理， TSO 和 Db2 存储过程环境中的 IBM MQ 应用程序能够更新 IBM MQ 和非 IBM MQ 资源 (例如， Db2) 在单个逻辑工作单元中。 有关 RRS 参与者支持的信息，请参阅 *z/OS MVS Programming: Resource Recovery*。

您的 IBM MQ 应用程序可以使用 MQCMIT 和 MQBACK 或者同等 RRS 调用 SRRCMIT 和 SRRBACK。 有关更多信息，请参阅 第 817 页的『RRS 批处理适配器』。

RRS 可用性

如果在 z/OS 系统上未激活 RRS，那么从与任一 RRS 存根（CSQBRSTB 或 CSQBRRSI）链接的程序发出的任何 IBM MQ 调用都将返回 MQRC_ENVIRONMENT_ERROR。

Db2 存储过程

如果将 Db2 存储过程与 RRS 配合使用，请注意：

- 使用 RRS 的 Db2 存储过程必须由工作负载管理器管理（WLM 管理）。
- 如果 Db2 管理的存储过程包含 IBM MQ 调用，并且与任一 RRS 存根（CSQBRSTB 或 CSQBRRSI）链接，那么 MQCONN 或 MQCONNX 调用将返回 MQRC_ENVIRONMENT_ERROR。
- 如果 WLM 管理的存储过程包含 IBM MQ 调用，并且与非 RRS 存根链接，那么 MQCONN 或 MQCONNX 调用会返回 MQRC_ENVIRONMENT_ERROR，除非是从存储过程地址空间启动后执行的首个 IBM MQ 调用。
- 如果 Db2 存储过程包含 IBM MQ 调用，并且与非 RRS 存根链接，那么在存储过程地址空间终止后或者后续存储过程执行 MQCMIT（使用 IBM MQ 批处理/TSO 存根）后才会落实此存储过程中更新的 IBM MQ 资源。
- 在相同地址空间中可以并行执行相同存储过程的多个副本。 如果希望 Db2 使用存储过程的单一副本，请确保使用可重入方式对程序编程。 否则，在程序的任何 IBM MQ 调用上可能出现 MQRC_HCONN_ERROR。
- 请勿在 WLM 管理的 Db2 存储过程中编码 MQCMIT 或 MQBACK。
- 请将所有程序设计为在语言环境 (LE) 中运行。

IBM i *CICS for IBM i 应用程序中的同步点*

IBM MQ for IBM i 参与 CICS for IBM i 工作单元。 您可以使用 CICS for IBM i 应用程序中的 MQI 来放入和获取当前工作单元中的消息。

您可以使用 EXEC CICS SYNCPOINT 命令来建立包含 IBM MQ for IBM i 操作的同步点。 要将所有更改回退至上一个同步点，可以使用 EXEC CICS SYNCPOINT ROLLBACK 命令。

如果将 MQPUT， MQPUT1 或 MQGET 与 CICS for IBM i 应用程序中设置的 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT 选项配合使用，那么在 IBM MQ for IBM i 将其注册为 API 落实资源之前，无法注销 CICS for IBM i 。 请在从队列管理器断开连接之前落实或回退任何暂挂的 put 或 get 操作。 这将使您能够注销 CICS for IBM i。

Multi *IBM MQ for Multiplatforms 中的同步点*

同步点支持对两种类型的工作单元产生作用：本地和全局。

本地工作单元是其中更新的资源仅限于 IBM MQ 队列管理器资源的工作单元。 此处的同步点协调由队列管理器使用单阶段落实过程自行提供。

全局工作单元是其中属于其他资源管理器（如数据库）的资源同样也进行更新的工作单元。 IBM MQ 可以自行协调此类工作单元。 也可以使用外部落实控制器对其进行协调。 例如：

- 其他事务管理器
- IBM i IBM i 落实控制器

为确保完整性，请使用两阶段落实过程。 两阶段落实可由兼容 XA 的事务管理器和数据库提供。 例如：

- TXSeries
- UDB
- IBM i IBM i 落实控制器

ULW IBM MQ 产品可以使用两阶段落实过程来协调全局工作单元。

IBM i IBM MQ for IBM i 可以充当 WebSphere Application Server 环境内全局工作单元的资源管理器，但是不能充当事务管理器。

隐式同步点

V 9.1.0

在放置持久消息时，会对 IBM MQ 进行优化以将持久消息放置在同步点下。 多个应用程序将持久消息放置到相同队列时，使用同步点执行效果更佳。 这是因为使用同步点来放置持久消息可减少队列争用。

当应用程序将持久消息放置在同步点之外时，**ImplSyncOpenOutput** 会添加隐式同步点。 这会提升性能，而不会使应用程序感知到隐式同步点。

仅当有多个应用程序放置到队列中时，隐式同步点才会提升性能，这是因为它可减少队列争用。 因此，**ImplSyncOpenOutput** 指定使队列打开用于输出的最小应用程序数量，才会添加隐式同步点。 缺省值为 2。 这意味着如果您不显式指定 **ImplSyncOpenOutput**，那么仅当多个应用程序放置到队列中时才会添加隐式同步点。

请参阅调整参数以获取更多信息。

多平台上的本地工作单元

仅涉及队列管理器的工作单元称为本地工作单元。 队列管理器本身使用单相落实过程来协调同步点（内部协调）。

为启动本地工作单元，应用程序会发出 MQGET、MQPUT 或 MQPUT1 请求，同时指定相应的同步点选项。 此工作单元使用 MQCMIT 落实，或者使用 MQBACK 回滚。 但是，当应用程序与队列管理器之间的连接中断（有意或无意）时，工作单元也会终止。

如果应用程序从队列管理器断开连接 (MQDISC) 而由 IBM MQ 协调的全局工作单元仍处于活动状态，那么将尝试落实工作单元。 但是，如果应用程序在未断开连接的情况下终止，那么工作单元将回滚，因为应用程序被视为异常终止。

多平台上的全局工作单元

同时需要包含属于其他资源管理器的资源的更新时，请使用全局工作单元。

此处的协调可以是队列管理器的内部或外部协调：

内部同步点协调

IBM MQ for IBM i 或 IBM MQ for z/OS 不支持全局工作单元的队列管理器协调。 它在 IBM MQ MQI client 环境中不受支持。

在此处，IBM MQ 执行协调。 为启动全局工作单元，应用程序会发出 MQBEGIN 调用。

必须提供 MQCONN 或 MQCONNX 调用返回的连接句柄 (*Hconn*) 作为 MQBEGIN 调用的输入。 此句柄表示与 IBM MQ 队列管理器的连接。

应用程序会发出 MQGET、MQPUT 或 MQPUT1 请求，指定相应的同步点选项。 这意味着您可以使用 MQBEGIN 来启动全局工作单元以更新本地资源和/或属于其他资源管理器的资源。 使用其他资源管理器的 API 完成对属于这些资源管理器的资源的更新。 但是不能使用 MQI 来更新属于其他队列管理器的队列。 在启动其他工作单元（本地或全局）之前发出 MQCMIT 或 MQBACK。

全局工作单元是使用 MQCMIT 落实的；这将启动工作单元中所涉及的所有资源管理器的两阶段落实。 使用两阶段落实进程来要求所有资源管理器（例如，Db2、Oracle 和 Sybase 之类的兼容 XA 的数据库管理器）首先准备落实。 仅当全部准备完成后，才会要求进行落实。 如果任何资源管理器发出无法落实信号，那么会改为要求全部资源管理器回退。 或者，您可以使用 MQBACK 来回滚所有资源管理器的更新。

如果在全局工作单元仍处于活动状态时，应用程序断开连接 (MQDISC)，那么将落实工作单元。 但是，如果应用程序在未断开连接的情况下终止，那么工作单元将回滚，因为应用程序被视为异常终止。

MQBEGIN 的输出是完成代码和原因码。

使用 MQBEGIN 启动全局工作单元时，包含使用队列管理器配置的所有外部资源管理器。 但在以下情况下，调用可启动工作单元，但完成时会显示警告：

- 没有任何参与的资源管理器（即，未使用队列管理器配置任何资源管理器）。

或者

- 有一个或多个资源管理器不可用。

在上述情况下，工作单元必须包含在其启动时可用的资源管理器的更新。

如果某一个资源管理器未落实其更新，那么所有资源管理器将收到回滚更新的指示，MQCMIT 完成并显示警告。 在非正常环境中（通常有操作员干预时），如果部分资源管理器落实其更新但其余资源管理器回滚更新，那么 MQCMIT 调用可能失败；工作将被视为已完成并存在混合结果。 此类情况将在队列管理器的错误日志中进行诊断，可能采取补救操作。

如果所涉及的所有资源管理器都落实其更新，那么全局工作单元的 MQCMIT 即可成功。

要获取 MQBEGIN 调用的描述，请参阅 MQBEGIN。

外部同步点协调

选择除 IBM MQ 以外的其他同步点协调程序时，可能发生此情况；例如，选择 CICS、Encina 或 Tuxedo。

在此情况下， UNIX and Linux 系统上的 IBM MQ 和 IBM MQ for Windows 会向同步点协调程序注册他们对工作单元结果的兴趣，以便他们可以根据需要落实或回滚任何未落实的 get 或 put 操作。 外部同步点协调程序可确定提供一阶段还是两阶段落实协议。

使用外部协调程序时，无法发出 MQCMIT、MQBACK 和 MQBEGIN。 对于这些函数的调用将失败，并显示原因码 MQRC_ENVIRONMENT_ERROR。

外部协调的工作单元的启动方式取决于同步点协调程序提供的编程接口。 可能需要显式调用。 如果需要显式调用，那么工作单元未启动时，请发出 MQPUT 调用并指定 MQPMO_SYNCPOINT 选项，这样会返回完成代码 MQRC_SYNCPOINT_NOT_AVAILABLE。

工作单元的作用域由同步点协调程序确定。 影响应用程序发出的 MQI 调用成功还是失败的是应用程序与队列管理器之间的连接状态，而不是工作单元的状态。 例如，在工作单元处于活动状态期间，应用程序可能断开连接并重新连接到队列管理器，并在同一个工作单元内执行进一步的 MQGET 和 MQPUT 操作。 这称为暂挂的断开连接。

无论您是否选择使用 CICS 的 XA 功能，都可以在 CICS 程序中使用 IBM MQ API 调用。 如果不使用 XA，那么在 CICS 原子工作单元内将不管理队列上往来的 put 和 get 消息。 选择此方式的原因之一是因为工作单元的总体一致性对您而言不重要。

如果工作单元的完整性很重要，那么必须使用 XA。 使用 XA 时，CICS 会使用两阶段落实协议来确保工作单元内的所有资源都同时进行更新。

有关设置事务支持的更多信息，请参阅事务支持场景以及 TXSeries CICS 文档（例如，*TXSeries for Multiplatforms CICS* 开放式系统管理指南）。

V 9.1.0 Multi 多平台上的隐式同步点

隐式同步点支持使持久消息放置在同步点之外。

在放置持久消息时，会对 IBM MQ 进行优化以将持久消息放置在同步点下。 多个应用程序并行将持久消息放置到相同队列时，使用同步点通常执行效果更佳。 这是由于如果在放置持久消息时使用同步点，那么 IBM MQ 的锁定策略更高效。

qm.ini 文件中的 **ImplSyncOpenOutput** 参数可控制当应用程序将持久消息放置在同步点之外时，是否可添加隐式同步点。 这可以提升性能，而不会使应用程序感知到隐式同步点。

仅当有多个应用程序并行放置到队列中时，隐式同步点才会提升性能，这是因为它可减少锁争用。 **ImplSyncOpenOutput** 指定使队列打开用于输出的最小应用程序数量，方可添加隐式同步点。 缺省值为

2。 这意味着如果您不显式指定 **ImplSyncOpenOutput**，那么仅当多个应用程序放置到队列中时才会添加隐式同步点。

如果添加隐式同步点，那么统计信息会反映正在发生此状况，您可能会看到来自 **runmqsc display conn** 的事务输出。

如果永不希望添加隐式同步点，请设置 **ImplSyncOpenOutput**=*OFF*。

请参阅调整参数以获取更多信息。

Multiplatforms 上到外部同步点管理器的接口
IBM MQ for Multiplatforms 支持由使用 X/Open XA 接口的外部同步点管理器对事务进行协调。

部分 XA 事务管理器 (TXSeries) 要求每个 XA 资源管理器提供其自己的名称。 这在 XA 切换结构中称为 name 字符串。

- ULW UNIX, Linux, and Windows 上 IBM MQ 的资源管理器名为 MQSeries_XA_RMI。
- IBM i 对于 IBM i，资源管理器名称为 MQSeries XA RMI。

有关 XA 接口的更多详细信息，请参阅由 The Open Group 发布的 XA 文档 *CAE Specification Distributed Transaction Processing: The XA Specification*。

在 XA 配置中，IBM MQ for Multiplatforms 履行 XA 资源管理器的角色。 XA 同步点协调程序可以管理一组 XA 资源管理器，并在两个资源管理器中同步事务的落实或回退。 以下是针对静态注册的资源管理器的工作方式：

1. 应用程序通知同步点协调程序要启动事务。
2. 同步点协调程序对已知的所有资源管理器发出调用，将当前事务通知这些资源管理器。
3. 应用程序发出调用以更新由与当前事务关联的资源管理器管理的资源。
4. 应用程序请求同步点协调程序落实或回滚事务。
5. 同步点协调程序使用两阶段落实协议向每个资源管理器发出调用以完成请求的事务。

XA 规范要求每个资源管理器提供称为“XA 开关”的结构。 此结构声明资源管理器的功能以及由同步点协调程序调用的功能。

此结构存在两种版本：

表 *132: XA* 开关版本

版本	**描述**
MQRMIXASwitch	静态 XA 资源管理
MQRMIXASwitchDynamic	动态 XA 资源管理

要获取包含此结构的库的列表，请参阅 IBM MQ XA 切换结构。

将其链接到 XA 同步点协调程序时必须采用的方法是由协调程序定义的；请查阅此协调程序提供的文档以确定如何启用 IBM MQ 以配合 XA 同步点协调程序使用。

由同步点协调程序在任何 *xa_info* 调用上传递的 *xa_open* 结构可以是要管理的队列管理器的名称。 这与传递到 MQCONN 或 MQCONNX 的队列管理器名称采用相同形式，如果要使用缺省队列管理器，那么可以为空。 但可以使用两个额外的参数：TPM 和 AXLIB

TPM 允许您向 IBM MQ 指定事务管理器名称，例如，CICS。 AXLIB 允许您指定 XA AX 入口点所在的事务管理器中的实际库名。

如果使用其中任一参数或非缺省队列管理器，那么必须使用 QMNAME 参数指定队列管理器名称。 有关更多信息，请参阅 xa_open 字符串的 CHANNEL、TRPTYPE、CONNAME 和 QMNAME 参数。

限制

1. 不允许将全局工作单元与共享 Hconn 配合使用（如第 690 页的『使用 MQCONNX 的共享（独立于线程）连接』中所述）。
2. IBM i IBM MQ for IBM i 不支持 XA 资源管理器的动态注册。

   唯一受支持的事务管理器为 WebSphere Application Server。
3. Windows 在 Windows 系统上，XA 开关中声明的所有函数都声明为 _cdecl 函数。
4. 外部同步点协调程序每次只能管理一个队列管理器。 这是因为协调程序具有与每个队列管理器的有效连接，因此受到每次仅允许一个连接的规则的约束。

   注: 注: 在 JEE 服务器中运行的 JMS 客户机应用程序 (CLIENT JEE 应用程序) 没有此限制，因此单个 JEE 服务器管理的事务可以协调同一事务中的多个队列管理器。 但是，以绑定方式运行的 JMS 服务器应用程序仍受到每次仅允许一个连接的规则的约束。
5. 使用同步点协调程序运行的所有应用程序都只能连接到由此协调程序管理的队列管理器，因为这些应用程序已有效连接到此队列管理器。 必须发出 MQCONN 或 MQCONNX 以获取连接句柄，并且必须在退出之前发出 MQDISC。 或者，可以针对 TXSeries CICS 使用出口 UE014015。

IBM i 到 *IBM i* 外部同步点管理器的接口

IBM MQ for IBM i 可以将本机 IBM i 落实控制用作外部同步点协调器。

不允许将独立于线程的（共享）连接用于落实控件。 请参阅 *IBM i Programming: Backup and Recovery Guide, SC21-8079* 以获取有关 IBM i 的落实控件功能的更多信息。

要启动 IBM i 落实控件工具，请使用 STRCMTCTL 系统命令。 要终止落实控件，请使用 ENDCMTCTL 系统命令。

注: 落实定义作用域的缺省值为 *ACTGRP。 针对 IBM MQ for IBM i，该值必须定义为 *JOB。 例如：

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i 还可以执行仅包含 IBM MQ 资源更新的本地工作单元。 当应用程序调用 MQPUT、MQPUT1 或 MQGET 并指定 MQPMO_SYNCPOINT、MQGMO_SYNCPOINT 或 MQBEGIN 时，将在每个应用程序中选择由 IBM i 协调的本地工作单元或者参与全局工作单元。 如果发出首个此类调用时落实控件未处于活动状态，那么 IBM MQ 会启动本地工作单元，并且针对此连接到 IBM MQ 的所有其他工作单元也将使用本地工作单元，无论随后是否启动了落实控件都是如此。 要落实本地工作单元，请使用 MQCMIT。 要回退本地工作单元，请使用 MQBACK。 IBM i 落实和回滚调用（如 CL 命令 COMMIT）对 IBM MQ 本地工作单元没有影响。

如果要将 IBM MQ for IBM i 与本机 IBM i 落实控件配合使用，作为外部同步点协调程序，请确保具有落实控件任何作业均处于活动状态，并确保正在单线程作业中使用 IBM MQ。 如果在已启动落实控件的多线程作业中调用 MQPUT、MQPUT1 或 MQGET 并指定 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT，那么调用失败并显示原因码 MQRC_SYNCPOINT_NOT_AVAILABLE。

可以在多线程作业中使用本地工作单元和 MQCMIT 与 MQBACK 调用。

如果在启动落实控件之后调用 MQPUT、MQPUT1 或 MQGET 并指定 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT，IBM MQ for IBM i 会将其自身作为 API 落实资源添加到落实定义中。 这通常作业中首个此类调用。 虽然在特定落实定义已注册有 API 落实资源，但无法终止此定义的落实控件。

从队列管理器断开连接时，如果当前工作单元内没有暂挂的 MQI 操作，那么 IBM MQ for IBM i 将除去其作为 API 落实资源的注册。

如果当前工作单元中存在暂挂的 MQPUT、MQPUT1 或 MQGET 操作时从队列管理器断开连接，那么 IBM MQ for IBM i 仍注册为 API 落实资源，以便在下次落实或回滚时会收到通知。 到达下一个同步点时，IBM MQ for IBM i 会根据需要落实或回滚更改。 在工作单元处于活动状态期间，应用程序可能断开连接并重新连接到队列管理器，并在同一个工作单元内执行进一步的 MQGET 和 MQPUT 操作（这是暂挂的断开连接）。

如果尝试针对此落实定义发出 ENDCMTCTL 系统命令，那么将发出 CPF8355 消息，指示暂挂的更改处于活动状态。 当作业终止时也会在作业日志中显示此消息。 为避免此问题，请落实或回滚所有暂挂的 IBM MQ

for IBM i 操作，并从队列管理器断开连接。 因此请在执行 ENDCMTCTL 之前使用 COMMIT 或 ROLLBACK 命令，以使结束落实控件成功完成。

使用 IBM i 落实控件作为外部同步点协调程序时，不能发出 MQCMIT、MQBACK 和 MQBEGIN 调用。 对于这些函数的调用将失败，并显示原因码 MQRC_ENVIRONMENT_ERROR。

要落实或回滚（即回退）工作单元，请使用支持落实控件的编程语言之一。 例如：

- CL 命令：COMMIT 和 ROLLBACK
- ILE C 编程函数：_Rcommit 和 _Rrollback
- ILE RPG：COMMIT 和 ROLBK
- COBOL/400：COMMIT 和 ROLLBACK

将 IBM i 落实控件作为外部同步点协调程序用于 IBM MQ for IBM i 时，IBM i 会执行有 IBM MQ 参与其中的两阶段落实协议。 因为每个工作单元都分两阶段落实，在第一阶段内投票落实之后，队列管理器在第二阶段期间可能不可用。 例如，如果队列管理器内部作业终止，可能发生此情况。 在此情况下，执行落实的作业日志包含消息 CPF835F，指示落实或回滚操作失败。 此消息之前的消息可指示问题原因、问题是在落实操作还是回滚操作期间发生的，以及失败的工作单元的逻辑工作单元标识 (LUWID)。

如果问题是由于 IBM MQ API 落实资源在落实或回滚准备的工作单元期间失败而导致的，那么可以使用 WRKMQMTRN 命令来完成此操作并恢复事务的完整性。 此命令要求您知道要落实和回退的工作单元的 LUWID。

使用触发器启动 IBM MQ 应用程序

了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

某些提供队列的 IBM MQ 应用程序可连续运行，因此始终可用于检索到达队列的消息。 但如果无法预测到达队列的消息数量，那么可能不希望使用此功能。 在此情况下，即使没有可供检索的消息，应用程序可能仍消耗系统资源。

IBM MQ 提供了支持在有消息可供检索的情况下自动启动应用程序的工具。 此工具称为触发。

有关触发通道的信息，请参阅触发通道。

什么是触发?

队列管理器定义构成触发器事件的某些条件。

如果为队列启用了触发且发生触发器事件，那么队列管理器将一条触发器消息发送至称为启动队列的队列。 启动队列上出现触发器消息表示发生了触发器事件。

队列管理器生成的触发器消息不是持久的。 这将减少日志记录（从而提高性能），并最大限度减少重新启动期间的重复，从而缩短重新启动时间。

处理启动队列的程序称为触发器监视器应用程序，其功能是读取触发器消息并根据触发器消息中的信息执行适当的操作。 通常，此操作将启动某些其他应用程序以处理生成触发器消息的队列。 从队列管理器的角度来看，触发器监视器应用程序没有任何特殊的地方，它只是从队列（启动队列）读取消息的另一个应用程序。

如果针对队列启用触发，那么可以创建与之关联的进程定义对象。 此对象包含有关处理导致触发器事件的消息的应用程序的信息。 如果创建了进程定义对象，那么队列管理器会抽取此信息，并将其放置到触发器消息中以供触发器监视器应用程序使用。 与队列关联的进程定义的名称由 *ProcessName* local-queue 属性提供。 每个队列可指定不同的进程定义，或几个队列可共享同一个进程定义。

如果要触发启动通道，无需定义进程定义对象。 将改为使用传输队列定义。

UNIX, Linux, and Windows 上运行的 IBM MQ 客户机支持触发。 在客户机环境中运行的应用程序可与客户机库链接，除此之外它与在完整的 IBM MQ 环境中运行的应用程序相同。 但要启动的触发器监视器和应用程序必须位于相同环境内。

触发包括：

应用程序队列
应用程序队列是本地队列，在设置开启触发的情况下并且满足条件时，此队列要求写入触发器消息。

进程定义

应用程序队列可具有与之关联的进程定义对象，其中保存将从应用程序队列获取消息的应用程序的详细信息。（请参阅进程定义属性，以获取属性列表。）

请记住，如果要触发启动通道，无需定义进程定义对象。

传输队列

如果要触发启动通道，需要传输队列。

对于除 Linux 以外的任何平台上的传输队列，传输队列的 *TriggerData* 属性可以指定要启动的通道的名称。 这可替代触发通道的进程定义，但仅当未创建进程定义时才会使用。

触发器事件

触发器事件是导致队列管理器生成触发器消息的事件。 这通常是到达应用程序队列的消息，但也可能在其他时间发生。 例如，请参阅第 801 页的『触发器事件的条件』。

IBM MQ 包含各种选项，使您可以控制导致触发器事件的条件（请参阅第 804 页的『控制触发器事件』）。

触发器消息

队列管理器可在识别触发器消息时创建一条触发器消息。 它会将此消息复制到有关要启动的应用程序的触发器消息信息中。 此信息来自与应用程序队列有关的应用程序队列和进程定义对象。

触发器消息具有固定格式（请参阅第 810 页的『触发器消息的格式』）。

初始队列

启动队列是队列管理器在其中放置触发器消息的本地队列。 请注意，启动队列不能为别名队列或模型队列。

每个队列管理器可以拥有多个启动队列，每个启动队列都与一个或多个应用程序队列相关联。

z/OS 共享队列、可供队列共享组中的队列管理器访问的本地队列均可作为 IBM MQ for z/OS 上的启动队列。

触发器监视器

触发器监视器是一个连续运行的应用程序，它为一个或多个启动队列提供服务。 当触发器消息到达启动队列时，触发器监视器将检索该消息。 触发器监视器会使用触发器消息中的信息。 它会发出命令启动应用程序以检索到达应用程序队列的消息，并将触发器消息头中包含的信息（包括应用程序队列名称）传递到其中。

在所有平台上，称为通道启动程序的特殊触发器监视器负责启动通道。

z/OS 在 z/OS 上，通常是手动启动通道启动程序，也可以通过更改队列管理器启动 JCL 中的 CSQINP2 以在启动队列管理器时自动启动通道启动程序。

Multi 在 多平台 上，通道启动程序将在队列管理器启动时自动启动，或者可以使用 **runmqchi** 命令手动启动。

有关更多信息，请参阅第 807 页的『触发器监视器的启动队列处理』。

要了解触发的工作方式，请参考第 797 页的图 100，这是触发器类型 FIRST (MQTT_FIRST) 的示例。

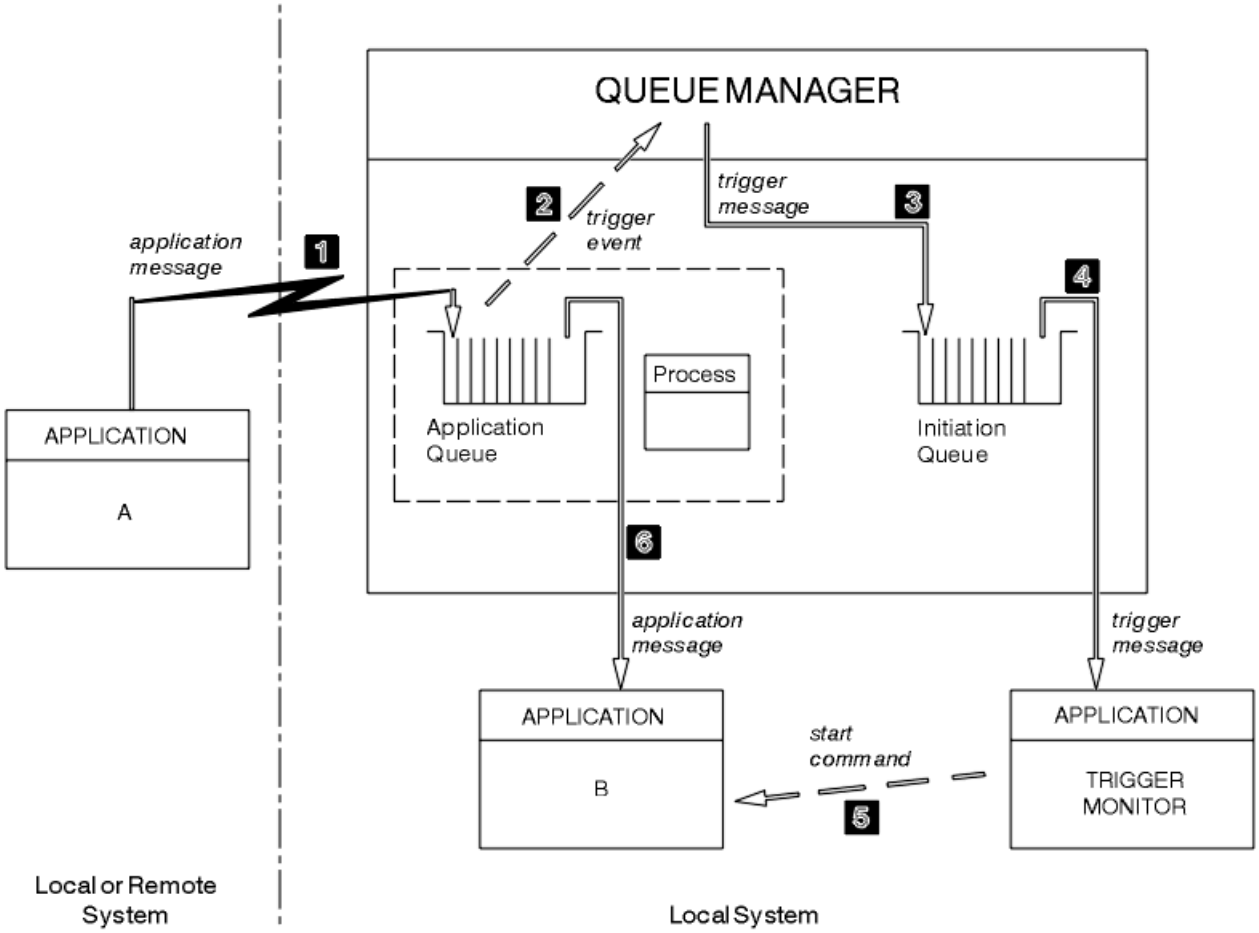


图 *100:* 应用程序和触发器消息流

在第 797 页的图 100 中，事件序列为：

1. 应用程序 A 可以时队列管理器的本地或远程系统，它将消息放置在应用程序队列上。 没有应用程序打开此队列进行输入。 但此事实仅与触发器类型 FIRST 和 DEPTH 有关。
2. 队列管理器会检查是否满足生成触发器事件的条件。 如果满足，那么将生成触发器事件。 创建触发器消息时会使用关联进程定义对象中保存的信息。
3. 队列管理器会创建触发器消息，并将其放置在与此应用程序队列关联的启动队列上，但前提是仅限应用程序（触发器监视器）已打开启动队列进行输入时。
4. 触发器监视器从启动队列中检索触发器消息。
5. 触发器监视器发出命令以启动应用程序 B（服务器应用程序）。
6. 应用程序 B 打开应用程序队列并且检索消息。

注：

1. 如果任何程序打开应用程序队列以进行输入，并且为 FIRST 和 DEPTH 设置触发，那么不会发生任何触发事件，因为已为队列提供服务。
2. 如果未打开启动队列以进行输入，那么队列管理器不会生成任何触发器消息；它会等待至应用程序打开启动队列进行输入为止。
3. 对通道使用触发时，使用触发器类型 FIRST 或 DEPTH。
4. 触发应用程序使用启动了触发器监视器的用户、CICS 用户或启动队列管理器的用户的用户标识和用户组来运行。

因此，触发内队列之间的关系仅为一对一关系。 请参见第 798 页的图 101。

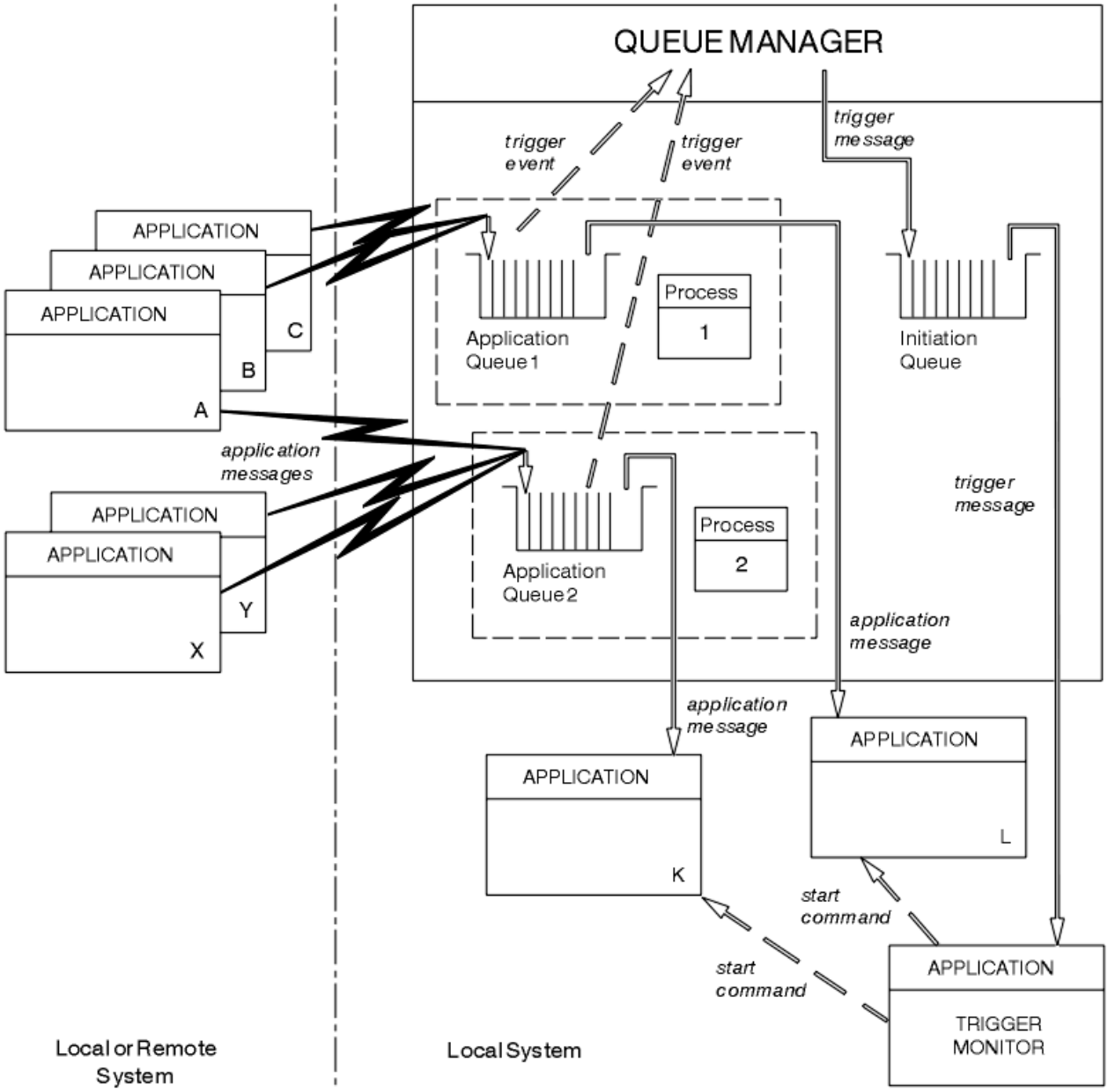


图 *101:* 触发内队列的关系

应用程序队列具有与之关联的进程定义对象，用于保存将处理消息的应用程序的详细信息。 队列管理器将信息放置在触发器消息中，因此只需一个启动队列。 触发器监视器从触发器消息抽取此信息，并启动相关应用程序以处理每个应用程序队列上的消息。

请记住，如果要触发启动通道，无需定义进程定义对象。 传输队列定义可确定要触发的通道。

使用以下链接来详细了解有关使用触发器启动 IBM MQ 应用程序：

- 第 799 页的『触发的先决条件』
- 第 801 页的『触发器事件的条件』
- 第 804 页的『控制触发器事件』
- 第 806 页的『设计使用触发队列的应用程序』
- 第 807 页的『触发器监视器的启动队列处理』
- 第 809 页的『触发器消息的属性』
- 第 811 页的『触发无效时』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

触发的先决条件

在使用触发之前使用此信息来了解有关要采取的步骤。

在应用程序使用触发之前，请完成以下步骤：

1. 请完成下面任意一项任务：
   a. 为应用程序队列创建启动队列。 例如：

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   或

   b. 确定存在并且可供应用程序使用的本地队列名称（通常此名称为 SYSTEM.DEFAULT.INITIATION.QUEUE，或者如果使用触发器启动通道，那么此名称为 SYSTEM.CHANNEL.INITQ），并在应用程序队列的 *InitiationQName* 字段中指定其名称。

2. 将启动队列与应用程序队列相关联。 每个队列管理器可拥有多个启动队列。 您可能希望由不同程序来服务部分应用程序队列，在此情况下，您可以针对每个服务程序使用一个启动队列，但这不是必需的。 以下是如何创建应用程序队列的示例：

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i 以下截取自创建启动队列的 IBM MQ for IBM i 的 CL 程序：

```
/*   Queue used by AMQSINQA                                        */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                        QTYPE(*LCL)  REPLACE(*YES)                  +
                        MQMNAME                                     +
                        TEXT('queue for AMQSINQA')                  +
                        SHARE(*YES)                 /* Shareable   */+
                        DFTMSGPST(*YES)/* Persistent messages OK   */+
                                                                    +
                        TRGENBL(*YES)  /* Trigger control on       */+
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. 如果触发应用程序，请创建一个进程定义对象以包含有关服务应用程序队列的应用程序的信息。 例如，要通过触发器启动称为 PAYR 的 CICS 薪资事务：

```
  DEFINE PROCESS (process.name) +
         REPLACE +
         DESCR ('process description') +
         APPLICID ('PAYR') +
         APPLTYPE (CICS) +
         USERDATA ('Payroll data')
```

IBM i 以下截取自创建进程定义对象的 IBM MQ for IBM i 的 CL 程序：

```
/*   Process definition                                            */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        REPLACE(*YES)                               +
                        MQMNAME                                     +
                        TEXT('trigger process for AMQSINQA')        +
                        ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                        APPID('AMQSINQA')    /* Program name      */
```

当队列管理器创建触发器消息时，它会将来自进程定义对象属性的信息复制到触发器消息中。

平台	要创建进程定义对象
UNIX, Linux, and Windows 系统	请使用 DEFINE PROCESS 或者使用 SYSTEM.DEFAULT.PROCESS 并使用 ALTER PROCESS 进行修改
z/OS z/OS z/OS	使用 DEFINE PROCESS（请参阅步骤 第 800 页的『3』 中的样本代码）或者使用操作和控制面板。
IBM i IBM i IBM i	使用包含代码的 CL 程序，如步骤 第 800 页的『3』 中所示。

4. 可选：创建传输队列定义，并针对 **ProcessName** 属性使用空白。

   TrigData 属性可包含要触发的通道的名称，或者也可以留空。 （在 IBM MQ for z/OS 上除外）如果留空，那么通道启动程序会搜索通道定义文件，直至找到与指定的传输队列相关联的通道为止。 当队列管理器创建触发器消息时，它会将来自传输队列定义的 **TrigData** 属性的信息复制到触发器队列中。

5. 如果已创建进程定义对象以指定服务应用程序队列的应用程序的属性，请通过在队列的 **ProcessName** 属性中对命名对象来将此进程对象与应用程序队列相关联。

平台	使用命令
UNIX, Linux, and Windows 系统	ALTER QLOCAL

平台	使用命令
z/OS z/OS	ALTER QLOCAL
IBM i IBM i	CHGMQMQ

6. 启动要用于服务您定义的启动队列的触发器监视器 IBM i （或 IBM MQ for IBM i 中的触发器服务器）的实例。 请参阅第 807 页的『触发器监视器的启动队列处理』，了解更多信息。

如果要注意任何未传递的触发器消息，请确保队列管理器已定义死信（未传递的消息）队列。 在 *DeadLetterQName* 队列管理器字段中指定队列名称。

然后，可以使用定义应用程序队列的队列对象属性来设置所需的触发条件。 有关更多信息，请参阅第 804 页的『控制触发器事件』。

触发器事件的条件

当满足本主题中所详述的条件时，队列管理器会创建一条触发器消息。

本主题中提到的共享队列表示队列共享组中的共享队列，这些共享队列仅在 IBM MQ for z/OS 上可用。

以下条件导致队列管理器创建触发器消息：

1. 消息放置在队列上。
2. 消息优先级大于或等于队列的阈值触发器优先级。 此优先级在 **TriggerMsgPriority** 本地队列属性中设置；如果设置为零，那么任何消息具备资格。
3. 根据 *TriggerType*，队列上优先级大于或等于 *TriggerMsgPriority* 的消息数量先前为：
   - 零（针对触发器类型 MQTT_FIRST）
   - 任何数字（针对触发器类型 MQTT_EVERY）
   - *TriggerDepth* 减 1（针对触发器类型 MQTT_DEPTH）

   注：

   a. 对于非共享的本地队列，队列管理器在评估触发器事件的条件是否存在时，会计算已落实的消息和未落实的消息的数量。 因此，如果不存在消息可供应用程序检索，那么可能启动应用程序，因为队列上的消息尚未落实。 在此情况下，请考虑使用含合适的 *WaitInterval* 的 wait 选项，以便使应用程序等待其消息到达。

   b. 对于本地共享队列，队列管理器仅计算已落实的消息数量。
4. 对于类型为 FIRST 或 DEPTH 的触发，没有任何程序会打开应用程序队列以除去消息（即，**OpenInputCount** 本地队列属性为零）。

   注：

   a. 对于共享队列，当有多个队列管理器针对某一个队列运行多个触发器监视器时，适用特殊条件。 在此情况下，如果有一个或多个队列管理器打开队列用于共享输入，那么其他队列管理器上的触发条件作为 *TriggerType* MQTT_FIRST 和 *TriggerMsgPriority* 零来处理。 当所有队列管理器关闭队列进行输入时，触发条件会还原为队列定义中指定的条件。

   受此条件影响的示例方案包含多个队列管理器 QM1、QM2 和 QM3 以及针对应用程序队列 A 运行的触发器监视器。 一条满足触发条件的消息到达队列 A，并在启动队列上生成一条触发器消息。 QM1 上的触发器监视器收到触发器消息并触发应用程序。 触发的应用程序会打开应用程序队列用于共享输入。 从此时开始，应用程序队列 A 的触发条件将求值为 *TriggerType* MQTT_FIRST，而在队列管理器 QM2 和 QM3 上求值为 *TriggerMsgPriority* 零，直至 QM1 关闭应用程序队列为止。

   b. 对于共享队列，此条件适用于每个队列管理器。 即，队列管理器的 *OpenInputCount* 针对队列必须为零，才能使该队列管理器为队列生成触发器消息。 但是，如果队列共享组中的任何队列管理器使用 MQOO_INPUT_EXCLUSIVE 选项打开队列，那么此队列共享组中的任何队列管理器都不会为此队列生成触发器消息。

当触发的应用程序打开队列以进行输入时，会导致触发条件的求值方式发生变化。 如果仅有一个触发器监视器在运行，那么其他应用程序可能受到相同的影响，因为其他应用程序同样打开应用程序队列以进行输入。 由触发器监视器启动的应用程序打开还是由其他应用程序打开应用程序队列无关紧要；重要的是已打开队列以在其他队列管理器上进行输入，而这导致改变触发条件。

5. 在 IBM MQ for z/OS 上，如果应用程序队列的 **Usage** 属性为 MQUS_NORMAL，那么将不会禁止此应用程序队列的 get 请求（即，**InhibitGet** 队列属性为 MQQA_GET_ALLOWED）。 并且，如果触发的应用程序队列的 **Usage** 属性为 MQUS_XMITQ，那么将不会禁止此应用程序队列的 get 请求。
6. 请完成下面任意一项任务：
   - 队列的 **ProcessName** 本地队列属性不为空，并且该属性标识的进程定义对象已创建，或者
   - 队列的 **ProcessName** 本地队列属性全部为空，但是此队列为传输队列。 由于进程定义为可选，**TriggerData** 属性也可能包含要启动的通道的名称。 在此情况下，触发器消息的属性包含以下值：
     - **QName**：队列名称
     - **ProcessName**：为空
     - **TriggerData**：触发器数据
     - **ApplType**：MQAT_UNKNOWN
     - **ApplId**：为空
     - **EnvData**：为空
     - **UserData**：为空
7. 已创建启动队列，并且已在 **InitiationQName** 本地队列属性中指定了此启动队列。 并且：
   - 针对启动队列未禁止 get 请求（即 **InhibitGet** 队列属性的值为 MQQA_GET_ALLOWED）。
   - 针对启动队列不得禁止 put 请求（即 **InhibitPut** 队列属性的值必须为 MQQA_PUT_ALLOWED）。
   - 启动队列的 **Usage** 属性的值必须为 MQUS_NORMAL。
   - 在支持动态队列的环境中，启动队列不得是已标记为逻辑删除的动态队列。
8. 触发器监视器当前已打开启动队列以除去消息（即，**OpenInputCount** 本地队列属性大于零）。
9. 应用程序队列的触发器控件（**TriggerControl** 本地队列属性）设置为 MQTC_ON。 为此，请在定义队列时设置 **trigger** 属性（或者使用 ALTER QLOCAL 命令）。
10. 触发器类型（**TriggerType** 本地队列属性）不是 MQTT_NONE。

    如果满足所有必需条件，并且已放置导致触发条件的消息作为工作单元的一部分，那么在工作单元完成后（已落实工作单元，或者针对触发器类型 MQTT_FIRST 或 MQTT_DEPTH，已回退工作单元），触发器消息才可供触发器监视器检索。
11. 针对 **TriggerType** 为 MQTT_FIRST 或 MQTT_DEPTH，已将适合的消息放置在队列上，并且队列：
    - 先前不为空 (MQTT_FIRST)，或
    - 具有 **TriggerDepth** 条或更多条消息 (MQTT_DEPTH)

    并满足条件 第 801 页的『2』 到 第 802 页的『10』（不包括 第 801 页的『3』），前提是对于 MQTT_FIRST，自从最近一次为此队列写入触发器消息后已经过足够长的时间（**TriggerInterval** 队列管理器属性）。

    这是考虑到未处理完队列中所有消息便已结束的队列服务器。 触发器时间间隔的目的是减少生成的重复触发器消息数量。

    注: 如果停止并重新启动队列管理器，那么 **TriggerInterval** 计时器会重置。 在一个短暂时间窗口内可能生成两条触发器消息。 当队列的触发器属性设置为启用的时间与消息到达的时间相同，并且此队列先前不为空 (MQTT_FIRST) 或具有 **TriggerDepth** 条或更多条消息 (MQTT_DEPTH) 时，便存在此时间窗口。
12. 服务队列的唯一应用程序针对 **TriggerType** 为 MQTT_FIRST 或 MQTT_DEPTH 的队列发出 MQCLOSE 调用，并且至少存在
    - 一条 (MQTT_FIRST) 或
    - **TriggerDepth** (MQTT_DEPTH)

条消息，位于具有充足优先级（条件 第 801 页的『2』）的队列上，并满足条件 第 802 页的『6』 到 第 802 页的『10』。

这是考虑到以下情况的队列服务器：发出 MQGET 调用、找到空队列然后结束；但是在 MQGET 与 MQCLOSE 调用之间的时间段内，有一条或多条消息到达。

注:

a. 如果服务应用程序队列的程序不检索所有消息，那么这可能导致闭合循环。 程序每次关闭队列时，队列管理器都会创建另一条触发器消息，导致触发器监视器重新启动服务器程序。

b. 如果服务应用程序队列的程序在关闭队列之前回退其 get 请求（或者如果程序异常终止），那么也会发生同样问题。 但是，如果程序在回退 get 请求之前关闭队列，那么此队列将为空，不创建触发器消息。

c. 为防止发生此类循环，请使用 MQMD 的 *BackoutCount* 字段来检测重复回退的消息。 有关更多信息，请参阅 第 38 页的『回退的消息』。

13. 使用 MQSET 或命令来满足以下条件：

a. • **TriggerControl** 更改为 MQTC_ON，或者

• **TriggerControl** 已设置为 MQTC_ON，并且 **TriggerType**、**TriggerMsgPriority** 或 **TriggerDepth**（如果相关）的值发生更改，

那么至少存在：

• 一条（MQTT_FIRST 或 MQTT_EVERY），或

• **TriggerDepth** (MQTT_DEPTH)

条消息，位于具有充足优先级（条件 第 801 页的『2』）的队列上，并满足条件 第 801 页的『4』 到 第 802 页的『10』（不包括 第 802 页的『8』）。

这是考虑到在已满足触发器触发条件时更改触发条件的应用程序或运算符。

b. 启动队列的 **InhibitPut** 队列属性的值从 MQQA_PUT_INHIBITED 更改为 MQQA_PUT_ALLOWED，并且至少存在：

• 一条（MQTT_FIRST 或 MQTT_EVERY），或

• **TriggerDepth** (MQTT_DEPTH)

条消息，位于具有充足优先级（条件 第 801 页的『2』）的任意队列上，相对于此消息所在队列，该队列为启动队列，并且满足条件 第 801 页的『4』 到 第 802 页的『10』。 （针对每个满足条件的此类队列会生成一条触发器消息。）

这是考虑到由于启动队列上的 MQQA_PUT_INHIBITED 条件而未生成的触发器消息，但是此条件现在已改变。

c. 应用程序队列的 **InhibitGet** 队列属性的值从 MQQA_GET_INHIBITED 更改为 MQQA_GET_ALLOWED，并且至少存在：

• 一条（MQTT_FIRST 或 MQTT_EVERY），或

• **TriggerDepth** (MQTT_DEPTH)

条消息，位于具有充足优先级（条件 第 801 页的『2』）的队列上，并且满足条件 第 801 页的『4』 到 第 802 页的『10』（不包括 第 802 页的『5』）。

由此仅当检索来自应用程序队列的消息时才触发应用程序。

d. 触发器监视器应用程序针对来自启动队列的输入发出 MQOPEN 调用，并且至少存在：

• 一条（MQTT_FIRST 或 MQTT_EVERY），或

• **TriggerDepth** (MQTT_DEPTH)

条消息，位于具有充足优先级（条件 第 801 页的『2』）的任意应用程序队列上，相对于此消息所在应用程序队列，该队列为启动队列，并且满足条件 第 801 页的『4』 到 第 802 页的『10』（不包括 第 802 页的『8』），没有任何其他应用程序打开启动队列用于进行输入（针对每个满足条件的此类队列会生成一条触发器消息。）

这是考虑到触发器监视器未在运行时到达队列的消息，也是考虑到重新启动的队列管理器和丢失的触发器消息（均为非持久性）。

14. MSGDLVSQ 设置正确。 如果设置 MSGDLVSQ=FIFO，那么按先进先出的顺序将消息传递到队列。 忽略消息优先级，将队列的缺省优先级分配给消息。 如果 **TriggerMsgPriority** 设置为高于队列缺省优先级的值，那么不触发任何消息。 如果 **TriggerMsgPriority** 设置为等于或低于队列缺省优先级，针对类型 FIRST、EVERY 和 DEPTH 将触发消息。 有关这些类型的信息，请参阅 第 804 页的『控制触发器事件』 下 **TriggerType** 字段的描述。

    如果设置 MSGDLVSQ=PRIORITY，并且消息优先级等于或大于 *TriggerMsgPriority* 字段，那么消息仅计入触发器事件。 在此情况下，针对类型 FIRST、EVERY 和 DEPTH 将发生触发。 例如，如果放置 100 条优先级低于 **TriggerMsgPriority** 的消息，那么触发目的的有效队列深度仍为零。 如果随后将另一条消息放入队列，但此次优先级大于或等于 **TriggerMsgPriority**，那么有效队列深度将从零增加至满足 **TriggerType** FIRST 条件的值。

注意:

1. 从步骤 第 802 页的『12』（因除消息到达应用程序队列以外的某些事件而生成触发器消息时）开始，触发器消息不作为工作单元的一部分来放置。 并且，如果 **TriggerType** 为 MQTT_EVERY，并且在应用程序队列上存在一条或多条消息，那么仅生成一条触发器消息。
2. 如果 IBM MQ 在 MQPUT 期间对某一条消息分段，那么在将所有分段成功放置到队列上之后才会处理触发器事件。 但当消息段位于队列上时，IBM MQ 会将其作为独立消息来处理，以便对其进行触发。 例如，一条逻辑消息拆分为三部分，导致首先对其进行 MQPUT 然后进行分段后，仅处理一起触发器事件。 但，每个分段都会导致在经过 IBM MQ 网络时处理自己的触发器事件。
3. 对于 IBM MQ for z/OS，如果设置了用于触发和连接到托管共享队列的耦合设施的共享队列，那么可能会生成触发器事件并将消息放入启动队列。 即使未将任何消息放入原始共享队列设置以进行触发，也会发生此情况。 这是由 IXLVECTR 宏 (如 The List Notification Vector 中所述) 过度指示位所导致的。

控制触发器事件

使用定义应用程序队列的部分属性来控制触发器事件。 此信息还提供了使用以下触发器类型的示例：EVERY、FIRST 和 DEPTH。

您可以启用和禁用触发，并且可以选择计入触发器事件的消息数量或优先级。 在对象属性中提供了这些属性的完整描述。

相关属性包括：

TriggerControl
使用此属性可为应用程序队列启用和禁用触发。

TriggerMsgPriority
消息计入触发器事件必须具备的最低优先级。 如果优先级小于 *TriggerMsgPriority* 的消息到达应用程序队列，那么队列管理器确定是否创建触发器消息时会忽略此消息。 如果 *TriggerMsgPriority* 设置为零，那么所有消息都计入触发器事件。

TriggerType
除触发器类型 NONE（禁用触发，与将 *TriggerControl* 设置为 OFF 相同）之外，您可以使用以下触发器类型来设置队列对触发器事件的敏感度：

EVERY
每次消息到达应用程序队列时都发生触发器事件。 如果希望启动应用程序的多个实例，请使用此类型的触发器。

第一个
仅当应用程序队列上的消息数量从零更改为一时，才发生触发器事件。 如果希望在首条消息到达队列时启动服务程序，并且持续直至没有消息可供处理后终止，那么请使用此类型的触发器。 必须始终处理队列直至队列为空。 另请参阅第 805 页的『触发器类型 FIRST 的特殊情况』。

DEPTH
仅当应用程序队列上的消息数量达到 **TriggerDepth** 属性的值时，才发生触发器事件。 一般当接收到对某一组请求的所有回复后启动程序时使用此类型的触发。

按深度触发: 按深度触发时，队列管理器在创建触发器消息后会禁用触发（使用 *TriggerControl* 属性）。 发生此情况后，应用程序必须自行重新启用触发（通过使用 MQSET 调用）。

禁用触发的操作不受同步点控制，因此无法通过回退工作单元来重新启用触发。 如果程序回退导致触发器事件的 put 请求，或者如果程序异常终止，那么必须通过使用 MQSET 调用或 ALTER QLOCAL 命令来重新启用触发。

TriggerDepth
使用按深度触发时导致触发器事件的队列上的消息数量。

在第 801 页的『触发器事件的条件』中描述了队列管理器创建触发器消息时必须满足的条件。

使用触发器类型 EVERY 的示例

考虑生成汽车保险请求的应用程序。 应用程序可能向多家保险公司发送请求消息，每次都指定相同的应答队列。 它可能在此应答队列上设置类型为 EVERY 的触发器，以便每次收到回复时，回复会触发服务器处理回复的实例。

使用触发器类型 FIRST 的示例

考虑具有多个分支机构办公地点的企业，每个分支机构各自将日常业务详细信息传输至总部。 所有分支机构都在每个工作日结束时同时执行此操作，在总部有一个应用程序负责处理来自所有分支机构的详细信息。 到达总部的第一条消息导致启动此应用程序的触发器事件。 此应用程序将继续处理直至队列上没有任何消息为止。

使用触发器类型 DEPTH 的示例

考虑旅行社应用程序，此应用程序会创建单一请求以确认机票预定情况、确认酒店房间预定情况、租车和订购某些旅行支票。 此应用程序可将这些项分为四条请求消息，将每条请求消息发送到不同的目标。 它可在其应答队列上设置类型为 DEPTH 的触发器（深度设置为值 4），这样仅当收到全部 4 条回复后才会重新启动此应用程序。

如果在四条回复中的最后一条到达队列之前有其他消息（可能来自不同请求）到达此队列，那么将提前触发请求应用程序。 为避免此情况，在使用 DEPTH 触发来收集某个请求的多个回复时，请始终针对每个请求使用新的应答队列。

触发器类型 FIRST 的特殊情况

对于触发器类型 FIRST，如果在另一条消息到达时应用程序队列上已有一条消息，那么队列管理器不会通常不会创建另一条触发器消息。

但是，服务队列的应用程序可能不会实际打开队列（例如，应用程序可能由于系统问题而终止）。 如果将错误的应用程序名称放入进程定义对象，那么服务队列的应用程序将不会选取任何消息。 在此类情况下，如果有其他消息到达应用程序队列，那么不会运行任何服务器以处理此消息（和队列上的任何其他消息）。

为处理此情况，队列管理器会在以下情况下创建更多触发器消息：

- 当其他消息到达应用程序队列时，但仅当自从队列管理器为此队列创建最后一条触发器消息起，预定义的时间间隔已耗尽时。 此时间间隔在队列管理器属性 *TriggerInterval* 中定义。 其缺省值为 999 999 999 毫秒。
- 在 IBM MQ for z/OS 上，将定期扫描指定打开启动队列的应用程序队列。 如果自从发出最后一条触发器事件后已经过 *TRIGINT* 毫秒，并且队列满足触发器事件条件，CURDEPTH 大于零，那么会生成触发器事件。 此过程称为逆止后备。

决定在应用程序中要使用的触发器时间间隔值时，请考虑以下几点：

- 如果将 *TriggerInterval* 设置为较低的值，并且没有应用程序来服务应用程序队列，那么触发器类型 FIRST 的行为可能与触发器类型 EVERY 类似。 这取决于将消息放入应用程序队列的速度，后者取决于其他系统活动。 这是因为，如果触发器时间间隔过短，每次将一条消息放入应用程序队列时都会生成另一条触发器消息，即使触发器类型为 FIRST 而不是 EVERY 也是如此。 （触发器时间间隔为零的触发器类型 FIRST 等同于触发器类型 EVERY。）
- 在 IBM MQ for z/OS 上，如果将 *TRIGINT* 设置为较低的值，并且没有应用程序来服务触发器类型 FIRST 的应用程序队列，那么每次指定打开启动队列的应用程序的定期扫描发生时，后备触发都会生成一条触发器消息。

- 如果工作单元回退（请参阅触发器消息和工作单元）并且触发器时间间隔已设置为较高的值（或缺省值），那么当工作单元回退时，会生成一条触发器消息。 但是如果将触发器时间间隔设置为较低的值或零（导致触发器类型 FIRST 行为与触发器类型 EVERY 类似），那么可生成许多触发器消息。 如果工作单元回退，那么所有触发器消息仍可用。 生成的触发器消息的数量取决于触发器时间间隔。 如果触发器时间间隔设置为零，那么将生成最大数量的消息。

设计使用触发队列的应用程序

您已了解如何设置和控制应用程序的触发。 以下是在设计应用程序时需要考虑的一些提示。

触发器消息和工作单元

由于不属于工作单元的触发器事件而创建的触发器消息将放置在启动队列上，位于任何工作单元之外，这些消息不从属于任何其他消息，并且可供触发器监视器立即检索。

解析 UOW 后，由于属于工作单元的触发器事件而创建的触发器消息在启动队列上可用，无论工作单元已落实还是已回退都是如此

如果队列管理器未能将触发器消息放置在启动队列上，那么会将其放置在死信（未送达的消息）队列上。

注:

1. 队列管理器在评估触发器事件的条件是否存在时，会计算已落实的消息和未落实的消息的数量。

   对于触发类型 FIRST 或 DEPTH，即使工作单元回退，触发器消息仍可用，因此只要满足所需条件，触发器消息始终可用。 例如，针对使用触发器类型 FIRST 触发的队列，请考虑工作单元内的 put 请求。 此请求导致队列管理器创建一条触发器消息。 如果从其他工作单元发生另一个 put 请求，这不会导致生成另一起触发器事件，因为应用程序队列上的消息数量已经从一条更改为两条，这并不满足触发器事件的条件。 如果第一个工作单元回退，但第二个工作单元已落实，那么仍将触发一条触发器消息。

   但这意味着有时不满足触发器事件的条件时，仍会创建触发器消息。 使用触发的应用程序随时随地准备处理此类情况。 建议针对 MQGET 调用使用等待选项，将 *WaitInterval* 设置为合适的值。

   创建的触发器消息始终可用，无论工作单元已回退还是已落实都是如此。
2. 对于本地共享队列（即，队列共享组中共享的队列），队列管理器仅计算已落实的消息数量。

从触发的队列获取消息

设计使用触发的应用程序时，请注意在触发器监视器启动程序和其他消息在应用程序队列上变为可用之间可能存在延迟。 如果在落实其他消息之前先落实导致触发器事件的消息，那么可能发生此情况。

考虑到消息到达的时间，请在使用 MQGET 调用始终使用 wait 选项，以从已设置触发条件的队列中除去消息。 *WaitInterval* 必须足以确保在放置消息与落实放置调用之间有最大限度的合理时间。 如果消息来自远程队列管理器，那么此时间受到以下因素的影响:

- 在落实之前放置的消息数量
- 通信链路的速度和可用性
- 消息的大小

要了解应将 MQGET 调用与 wait 选项配合使用的情况示例，请考虑描述工作单元时所使用的样本示例。 它是针对使用触发器类型 FIRST 触发队列时工作单元内的 put 请求。 此事件导致队列管理器创建一条触发器消息。 如果从其他工作单元发生另一个 put 请求，这不会导致生成另一起触发器事件，因为应用程序队列上的消息数量未从零更改为一。 如果第一个工作单元回退，但第二个工作单元已落实，那么仍将触发一条触发器消息。 因此在第一个工作单元回退时会创建触发器消息。 如果在第二条消息落实前存在显著延迟，那么触发的应用程序可能需要等待此消息。

对于触发类型 DEPTH，即使所有相关消息最终落实，仍可能发生延迟。 假设 **TriggerDepth** 队列属性的值为 2。 当有两条消息到达队列时，第二条消息将导致创建触发器消息。 但如果先落实第二条消息，那么此时触发器消息会变为可用。 触发器监视器会启动服务器程序，但是在落实第一条消息之前，程序只能检索第二条消息。 因此，程序可能需要等待第一条消息变为可用。

设计应用程序，以便在等待时间间隔过后没有消息可供检索时应用程序可终止。 如果有一条或多条消息稍后到达，请依靠重新触发的应用程序来处理这些消息。 此方式会阻止应用程序空闲以及不必要地使用资源。

触发器监视器的启动队列处理

对于队列管理器，触发器监视器与服务队列的任何其他应用程序相似。 但是，触发器监视器会为启动队列提供服务。

触发器监视器通常是持续运行的程序。 当触发器消息到达启动队列时，触发器监控器会检索该消息。 它使用消息中的信息来发出启动应用程序的命令，以便处理应用程序队列上的消息。

触发器监视器必须将充足的信息传递给要启动的程序，以便程序能够对正确的应用程序队列执行正确的操作。

用于消息通道代理程序的特殊类型触发器监视器的一个示例便是通道启动程序。 但在此情况下，必须使用触发器类型 FIRST 或 DEPTH。

Windows UNIX *UNIX* 和 *Windows* 系统上的触发器监视器

本主题包含有关 UNIX 和 Windows 系统上提供的触发器监视器的信息。

针对服务器环境提供了以下触发器监视器：

amqstrg0
这是样本触发器监视器，可提供一部分 **runmqtrm** 提供的功能。 请参阅第 964 页的『在 Multiplatforms 上使用样本程序』，以获取有关 amqstrg0 的更多信息。

runmqtrm
此命令的语法为 **runmqtrm** [*-m QMgrName*] [*-q InitQ*]，其中 QMgrName 是队列管理器， InitQ 是启动队列。 在缺省队列管理器上，缺省队列为 SYSTEM.DEFAULT.INITIATION.QUEUE。 它会为相应的触发器消息调用程序。 此触发器监视器支持缺省应用程序类型。

由触发器监视器传递给操作系统的命令字符串构建方式如下：

1. 来自相关进程定义（如果已创建）的 *ApplId*。
2. MQTMC2 结构，使用双引号括起
3. 来自相关进程定义（如果已创建）的 *EnvData*。

其中 *ApplId* 是要运行的程序的名称，将在命令行中输入此名称。

传递的参数为 MQTMC2 字符结构。 调用的包含此字符串的命令字符串使用双引号括起，以便系统命令将其作为一个参数来接受。

触发器监视器在其启动的应用程序完成之前，不会查看启动队列上是否存在其他消息。 如果应用程序需要进行许多处理工作，那么触发器监视器可能无法及时处理收到的大量触发器消息。 您有两个选择：

- 运行更多触发器监视器
- 在后台运行已经启动的应用程序

如果您运行更多的触发器监视器，您可以控制在任一时间可以运行的应用程序的最大数目。 如果您在后台运行应用程序，那么 IBM MQ 不会对可运行的应用程序数量加以限制。

要在 Windows 系统上在后台运行启动的应用程序，请在 *ApplId* 字段中，使用 START 命令给应用程序名称添加前缀。 例如：

```
START ?B AMQSECHA
```

UNIX To run the started application in the background on UNIX, put an & at the end of the *EnvData* of the PROCESS definition.

注: Windows 其中 Windows 路径包含空格作为路径名的一部分，这些空格应使用引号括起 (")，以确保将其作为单一自变量来处理。 例如，“`C:\Program Files\Application Directory\Application.exe`”。

以下是 APPLICID 字符串的示例，其中文件名包含空格作为路径的一部分：

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

示例中 Windows START 命令的语法包含使用双引号括起的空字符串。 START 指定引号中的第一个自变量将作为新命令的标题来处理。 为确保 Windows 不会将应用程序路径误作为“标题”变量来处理，请在命令中的应用程序名称之前添加一个标题字符串，并使用双引号将其括起。

为 IBM MQ 客户机提供了以下触发器监视器：

runmqtmc
此监视器与 runmqtrm 相同，但它与 IBM MQ MQI client 库链接。

Windows UNIX 适用于 *CICS* 的触发器监视器

为 CICS 提供了 amqltmc0 触发器监视器。 其工作方式与标准触发器监视器 runmqtrm 相同，但运行方式不同，并且它会触发 CICS 事务。

本主题仅适用于 Windows、UNIX 和 Linux x86-64 系统。

触发器监视器将作为 CICS 程序提供；可使用 4 个字符的事务名称来定义触发器监视器。 输入 4 个字符的名称以启动触发器监视器。 它使用缺省队列管理器 (在 qm.ini 文件中指定，或者在 IBM MQ for Windows 上使用注册表) 和 SYSTEM.CICS.INITIATION.QUEUE。

如果要使用其他队列管理器或队列，请构建触发器监视器 MQTMC2 结构：这要求您编写一个使用 EXEC CICS START 调用的程序，因为此结构过长，无法作为参数来添加。 然后，将 MQTMC2 结构作为数据传递到触发器监视器的 START 请求。

在使用 MQTMC2 结构时，只需向触发器监视器提供 *StrucId*、*Version*、*QName* 和 **QMgrName** 参数，因为它不引用任何其他字段。

从启动队列读取消息，并使用 EXEC CICS START 启动 CICS 事务，假定触发器消息中的 APPL_TYPE 为 MQAT_CICS。 在 CICS 同步点控制下执行从启动队列读取消息。

当监视器启动和停止时以及发生错误时，都会生成消息。 这些消息会发送到 CSMT 瞬时数据队列。

表 *133:* 触发器监视器的可用版本.
包含两列的表。 第一列列出触发器监视器的可用版本，第二列显示每个版本所适用的平台。

版本	适用平台
amqltmc0	TXSeries，适用于： • AIX AIX • Linux Linux x86-64 系统 • Solaris Oracle Solaris V5.1
amqltmc4	Windows TXSeries for Windows 5.1
amqltmcc	CICS 触发器监视器的客户机绑定版本

如果需要适用于其他环境的触发器监视器，请编写一个程序，以便可以处理队列管理器放入启动队列中的触发器消息。 此类程序应执行以下操作：

1. 使用 MQGET 调用等待消息到达启动队列。
2. 检查触发器消息的 MQTM 结构的字段，以查找要启动的应用程序的名称以及运行此应用程序的环境。
3. 发出特定于环境的启动命令。

   z/OS 例如，在 z/OS 批处理中，向内部阅读器提交一个作业。
4. 根据需要将 MQTM 结构转换为 MQTMC2 结构。

5. 将 MQTMC2 或 MQTM 结构传递到已启动的应用程序。 其中可包含用户数据。
6. 将应用程序队列与要为其提供服务的应用程序相关联。 可通过在队列的 **ProcessName** 属性中命名进程定义对象（如果已创建）来执行此操作。 要命名进程定义对象，可以使用 **DEFINE QLOCAL** 或 **ALTER QLOCAL** 命令。

   IBM i 在 IBM i 上，还可以使用 CRTMQMQ 或 CHGMQMQ。

有关触发器监视器接口的更多信息，请参阅 MQTMC2。

IBM i *IBM i 上的触发器监视器*

在 IBM i 上，使用 IBM MQ for IBM i CL 命令 **STRMQMTRM** 代替 **runmqtrm** 控制命令。

使用 STRMQMTRM 命令，如下所示：

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

详细信息与 runmqtrm 的详细信息相同。

还提供了以下样本程序，以便您用作为编写自己的触发器监视器的模型：

AMQSTRG4
这是触发器监视器，可为要启动的进程提交 IBM i 作业，但这意味着存在与每条触发器消息关联的额外处理。

AMQSERV4
这是触发器服务器。 对于每条触发器消息，此服务器会为其自己的作业中的进程运行命令，并且可以调用 CICS 事务。

触发器监视器和触发器服务器都会将 MQTMC2 结构传递至其启动的程序。 要获取此结构的描述，请参阅 MQTMC2。 这两个样本都以源格式和可执行文件格式提供。

由于这些触发器监视器只能调用本机 IBM i 程序，无法直接触发 Java 程序，因为 Java 类位于 IFS 中。 但是，可通过触发 CL 程序来间接触发 Java 程序，随后 CL 程序可调用 Java 程序并传递 TMC2 结构。 TMC2 的结构的最小大小是 732 个字节。

以下是样本 CLP 的源：

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

针对 IBM MQ MQI client 提供以下触发器监视器程序：RUNMQTMC

调用 RUNMQTMC，如下所示：

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

触发器消息的属性

以下主题描述了触发器消息的部分其他属性。

- 第 809 页的『触发器消息的持久性和优先级』
- 第 810 页的『队列管理器重新启动并触发消息』
- 第 810 页的『触发器消息以及针对对象属性的更改』
- 第 810 页的『触发器消息的格式』

触发器消息的持久性和优先级

因为不要求触发器消息保持持久性，所以触发器消息为非持久性消息。

但是，生成触发器事件的条件为持久性条件，因此只要满足这些条件，就会生成触发器消息。 如果触发器消息丢失，应用程序队列上应用程序消息持续存在可保证队列管理器在满足所有条件后立即生成触发器消息。

如果工作单元回滚，始终会传递其生成的所有触发器消息。

触发器消息采用启动队列的缺省优先级。

队列管理器重新启动并触发消息

在队列管理器重新启动后，下一步打开启动队列以进行输入时，如果与之关联的应用程序队列有消息在此启动队列上并已定义为触发，那么可将触发器消息放入此启动队列。

触发器消息以及针对对象属性的更改

根据发生触发器事件时生效的触发器属性值创建触发器消息。

如果触发器消息原先对触发器监视器不可用（因为导致生成此触发器消息的消息放置在工作单元内），那么与此同时对触发器属性进行的任何更改对触发器消息都无效。 尤其是在创建触发器消息后，禁用触发不会阻止触发器消息变为可用。 此外，当触发器消息变为可用时，应用程序队列可能已不存在。

触发器消息的格式

触发器消息的格式由 MQTM 结构来定义。

包括以下字段，队列管理器会在创建触发器消息时，使用应用程序队列的对象定义和与此队列关联的进程的对象定义中的信息来填充这些字段。

StrucId
结构标识符。

Version
结构版本。

QName
在其中发生触发器事件的应用程序队列的名称。 当队列管理器创建消息时，会使用应用程序队列的 **QName** 属性来填充该字段。

ProcessName
与应用程序队列关联的进程定义对象的名称。 当队列管理器创建消息时，会使用应用程序队列的 **ProcessName** 属性来填充该字段。

TriggerData
自由格式字段，可供触发器监视器使用。 当队列管理器创建消息时，会使用应用程序队列的 **TriggerData** 属性来填充该字段。 在任何 IBM MQ 产品（除 IBM MQ for z/OS 外）上，该字段可用于指定要触发的通道的名称。

ApplType
触发器监视器要启动的应用程序的类型。 当队列管理器创建触发器消息时，它将使用 *ProcessName* 中标识的进程定义对象的 **ApplType** 属性来填充此字段。

ApplId
识别触发器监视器要启动的应用程序的字符串。 当队列管理器创建触发器消息时，它将使用 *ProcessName* 中标识的进程定义对象的 **ApplId** 属性来填充此字段。

当使用 CICS 提供的触发器监视器 CKTI 时，进程定义对象的 **ApplId** 属性是 CICS 事务标识。

当使用 IBM MQ for z/OS 提供的 CSQQTRMN 时，进程定义对象的 **ApplId** 属性是 IMS 事务标识。

EnvData
字符字段，包含可供触发器监视器使用的环境相关数据。 当队列管理器创建触发器消息时，它将使用 *ProcessName* 中标识的进程定义对象的 **EnvData** 属性来填充此字段。 CICS 提供的触发器监视器 (CKTI) 或 IBM MQ for z/OS 提供的触发器监视器 (CSQQTRMN) 不使用此字段，但其他触发器监视器可能会选择使用此字段。

UserData
字符字段，包含可供触发器监视器使用的用户数据。 当队列管理器创建触发器消息时，它将使用 *ProcessName* 中标识的进程定义对象的 **UserData** 属性来填充此字段。 此字段可用于指定要触发的通道名称。

在 MQTM 中提供了触发器消息结构的完整描述。

触发无效时

如果触发器监视器无法启动程序，或者如果队列管理器无法交付触发器消息，那么无法触发程序。 例如，进程对象中的应用程序标识必须指定在后台启动程序；否则触发器监视器无法启动程序。

如果已创建触发器消息，但无法将其放置在启动队列上（例如，由于队列已满，或者触发器消息长度超过针对启动队列指定的最大消息长度），那么触发器消息将改为放置在死信（未传递的消息）队列上。

如果放置到死信队列的操作无法成功完成，那么将放弃触发器消息，并向 z/OS 控制台或系统操作员发送一条警告消息，或者将警告消息放入错误日志。

将触发器消息放置在死信队列上可能为此队列生成触发器消息。 如果此第二条触发器消息向死信队列添加一条消息，那么将放弃此条触发器消息。

如果成功触发程序，但在收到来自队列的消息之前异常终止，请使用跟踪实用程序（例如，如果程序在 CICS 下运行，那么可使用 CICS AUXTRACE）来查找故障原因。

与 MQI 和集群配合使用

对于与集群相关的调用和返回码存在特殊选项。

使用以下链接来了解有关可用于集群的调用和返回码的可用选项：

- 第 812 页的『MQOPEN 和集群』
- 第 813 页的『MQPUT、MQPUT1 和集群』
- 第 813 页的『MQINQ 和集群』
- 第 814 页的『MQSET 和集群』
- 第 814 页的『返回码』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

MQOPEN 和集群

打开集群队列时，要将消息放置到的队列或者要从中读取消息的队列取决于 MQOPEN 调用。

选择目标队列

如果在对象描述符 MQOD 中不提供队列管理器名称，那么队列管理器会选择要将消息发送到的队列管理器。如果在对象描述符中提供队列管理器名称，那么始终将消息发送到所选队列管理器。

如果队列管理器要选择目标队列管理器，那么选择取决于绑定选项 MQOO_BIND_*，以及本地队列是否存在。 如果存在队列的本地实例，那么始终会优先于远程实例打开此本地实例，除非 CLWLUSEQ 属性设置为 ANY。 否则，选择将取决于绑定选项。将 消息组 与集群配合使用时，必须指定 MQOO_BIND_ON_OPEN 或 MQOO_BIND_ON_GROUP，以确保在同一目标处处理组中的所有消息。

如果队列管理器要选择目标队列管理器，可使用工作负载管理算法以循环法方式来进行选择；请参阅集群中的工作负载均衡。

使用工作负载均衡算法的时机取决于打开集群队列的方式：

- MQOO_BIND_ON_OPEN - 应用程序打开集群时使用此算法。
- MQOO_BIND_NOT_FIXED - 将每条消息放入队列时使用此算法。
- MQOO_BIND_ON_GROUP - 在每个消息组开头使用此算法。

MQOO_BIND_ON_OPEN
MQOPEN 调用上的 MQOO_BIND_ON_OPEN 选项指定要修复目标队列管理器。 如果在集群内存在同一个队列的多个实例，那么请使用 MQOO_BIND_ON_OPEN 选项。 放入队列并且指定从 MQOPEN 调用返回的对象句柄的所有消息都将定向至相同的队列管理器。

- 当消息具有亲缘关系时，请使用 MQOO_BIND_ON_OPEN 选项。 例如，如果一批消息全部由同一个队列管理器来进行处理，那么打开队列时请指定 MQOO_BIND_ON_OPEN。 IBM MQ 会修复队列管理器以及放入此队列的所有消息所采用的路径。
- 如果指定了 MQOO_BIND_ON_OPEN 选项，那么必须重新打开此队列才能选择队列的新实例。

MQOO_BIND_NOT_FIXED
MQOPEN 调用上的 MQOO_BIND_NOT_FIXED 选项指定不修复目标队列管理器。 写入队列并指定从 MQOPEN 调用返回的对象句柄的消息将根据每条消息路由至执行 MQPUT 时的队列管理器。 如果不希望强制降所有消息写入相同目标，那么请使用 MQOO_BIND_NOT_FIXED 选项。

- 请勿同时指定 MQOO_BIND_NOT_FIXED 和 MQMF_SEGMENTATION_ALLOWED。 如果同时指定这两者，那么可能将消息段传递到分散于集群中的不同队列管理器。

MQOO_BIND_ON_GROUP
允许应用程序请求将一组消息分配给同一个目标实例。 此选项仅对于队列有效，并且仅影响集群队列。如果为不是集群队列的队列指定此选项，则会忽略此选项。

- 仅当在 MQPUT 上指定 MQPMO_LOGICAL_ORDER 时，才会将组路由至单个目标。 指定了 MQOO_BIND_ON_GROUP 但消息不属于某个逻辑组时，会改为使用 BIND_NOT_FIXED 行为。

MQOO_BIND_AS_Q_DEF
如果不指定 MQOO_BIND_ON_OPEN、MQOO_BIND_NOT_FIXED 或 MQOO_BIND_ON_GROUP，缺省选项为 MQOO_BIND_AS_Q_DEF。 使用 MQOO_BIND_AS_Q_DEF 会导致从 DefBind 查询属性中提取用于队列句柄的绑定。

MQOPEN 选项的相关性

MQOPEN 选项 MQOO_BROWSE, MQOO_INPUT_* 或 MQOO_SET 需要集群队列的本地实例才能使 MQOPEN 成功。

MQOPEN 选项 MQOO_OUTPUT、MQOO_BIND_* 或 MQOO_INQUIRE 无需集群队列的本地实例即可成功。

已解析队列管理器名称

在 MQOPEN 时间解析队列管理器名称时，解析的名称会返回到应用程序。 如果应用程序尝试在后续 MQOPEN 调用上使用此名称，可能会发现无权访问名称。

MQPUT、*MQPUT1* 和集群

如果在 MQOPEN 上指定了 MQOO_BIND_NOT_FIXED，那么工作负载管理例程会选择 MQPUT 或 MQPUT1 所选的目标。

如果在 MQOPEN 调用上指定了 MQOO_BIND_NOT_FIXED，那么后续每个 MQPUT 调用都会调用工作负载管理例程以确定要将消息发送到的队列管理器。 根据每条消息来选择要使用的目标和路径。 如果网络条件发生改变，那么放置消息后，目标和路径可能发生改变。 MQPUT1 调用始终按 MQOO_BIND_NOT_FIXED 生效的方式来运行，即始终调用工作负载管理例程。

当工作负载管理例程选择了队列管理器之后，本地队列管理器会完成放置操作。 可将消息放置在不同队列上：

1. 如果目标是队列的本地实例，那么会将消息放置在本地队列上。
2. 如果目标是集群中的队列管理器，那么会将消息放置在集群传输队列上。
3. 如果目标是位于集群外部的队列管理器，那么会将消息放置在与目标队列管理器同名的传输队列上。

如果在 MQOPEN 调用上指定了 MQOO_BIND_ON_OPEN，那么 MQPUT 调用不会调用工作负载管理例程，因为已选中了目标和路径。

MQINQ 和集群

所查询的集群队列取决于配合 MQOO_INQUIRE 使用的选项。

在查询队列之前，请使用 MQOPEN 调用将其打开并指定 MQOO_INQUIRE。

要查询集群队列，请使用 MQOPEN 调用，并将其他选项与 MQOO_INQUIRE 配合使用。 可查询的属性取决于是否存在集群队列的本地实例以及队列的打开方式：

- 将 MQOO_BROWSE、MQOO_INPUT_* 或 MQOO_SET 与 MQOO_INQUIRE 配合使用需要打开集群队列的本地实例才能成功。 在此情况下，可以查询针对本地队列有效的所有属性。
- 将 MQOO_OUTPUT 与 MQOO_INQUIRE 配合使用，并且不指定先前任何选项，那么打开的实例为以下任一实例：
  - 本地队列管理器上的实例（如果有）。 在此情况下，可以查询针对本地队列有效的所有属性。
  - 集群中的其他实例（如果不存在本地队列管理器实例）。 在此情况下，只能查询以下属性。 在此情况下，QType 属性的值为 MQQT_CLUSTER。
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

要查询集群队列的 DefBind 属性，请将 MQINQ 调用与选择器 MQIA_DEF_BIND 配合使用。 返回的值为 MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED 或 MQBND_BIND_ON_GROUP。在将组与集群配合使用时，必须指定 MQBND_BIND_ON_OPEN 或 MQBND_BIND_ON_GROUP 。

要查询队列的本地实例的 CLUSTER 和 CLUSNL 属性，请将 MQINQ 调用与选择器 MQCA_CLUSTER_NAME 或选择器 MQCA_CLUSTER_NAMELIST 配合使用。

注: 如果打开集群队列而不固定与 MQOPEN 绑定的队列，那么后续 MQINQ 可能查询集群队列的其他实例。

相关概念

第 697 页的『集群队列的 MQOPEN 选项』

用于队列句柄的绑定取自 **DefBind** 队列属性（可采用值 MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED 或 MQBND_BIND_ON_GROUP）。

MQSET 和集群

MQOPEN 选项的 MQOO_SET 选项要求存在集群队列的本地实例才能使 MQSET 成功。

您可以使用 MQSET 调用来设置集群中其他队列的属性。

您可以打开使用集群属性定义的本地别名和远程队列，并使用 MQSET 调用。 您可以设置本地别名或远程队列的属性。 目标队列是否是其他队列管理器上定义的集群队列无关紧要。

返回码

特定于集群的返回码

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

发出一个 MQOPEN、MQPUT 或 MQPUT1 调用以打开集群队列或者在其中放入一条消息。 由队列管理器的 ClusterWorkloadExit 属性定义的集群工作负载出口意外发生故障或者未及时响应。

将一条消息写入 IBM MQ for z/OS 上的系统日志，提供有关此错误的更多信息。

针对此队列句柄的后续 MQOPEN、MQPUT 和 MQPUT1 调用将按 ClusterWorkloadExit 属性为空的方式来处理。

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

在 z/OS 上，无法装入集群工作负载出口。

将一条消息写入系统日志，并按 ClusterWorkloadExit 属性为空的方式来处理。

Multi 在 多平台 上，发出 MQCONN 或 MQCONNX 调用以连接到队列管理器。 由于无法装入由队列管理器的 ClusterWorkloadExit 属性定义的集群工作负载出口，因此调用失败。

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

针对集群队列发出 MQOPEN 调用，其中 MQOO_OUTPUT 和 MQOO_BIND_ON_OPEN 选项生效。 通过将 InhibitPut 属性设置为 MQQA_PUT_INHIBITED 来禁止放置当前集群中的所有队列实例。 由于没有队列实例可接收消息，因此 MQOPEN 调用失败。

仅当以下两个语句同时为 true 时，才出现此原因码：

- 没有队列的本地实例。 如果存在本地实例，那么 MQOPEN 调用将成功，即使禁止放置本地实例也是如此。
- 对于队列没有集群工作负载出口，或者存在集群工作负载出口，但是未选择队列实例。 （如果集群工作负载出口选择队列实例，那么 MQOPEN 调用成功，即使此实例禁止放置也是如此。）

如果 MQOO_BIND_NOT_FIXED 选项在 MQOPEN 调用上指定了，那么即使集群中所有队列均为禁止放置，此调用仍可成功。 但是，如果在调用时所有队列仍处于禁止放置状态，那么后续 MQPUT 调用可能失败。

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. 发出一个 MQOPEN、MQPUT 或 MQPUT1 调用以打开集群队列或者在其中放入一条消息。 由于需要来自完整存储库队列管理器的响应，但是没有响应可用，因此无法正确解析队列定义。
2. 针对指定了 PUBSCOPE (ALL) 或 SUBSCOPE (ALL) 的主题对象发出 MQOPEN， MQPUT， MQPUT1 或 MQSUB 调用。 无法正确解析集群主题定义，因为需要来自完整存储库队列管理器的响应，但没有响应可用。

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

针对集群队列发出一个 MQOPEN、MQPUT 或 MQPUT1 调用。 尝试使用集群所需资源时发生错误。

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

发出一个 MQPUT 或 MQPUT1 调用以将消息放置在集群队列上。 在调用时，集群中不再有任何队列实例。 MQPUT 失败，不发送消息。

如果在打开队列的 MQOPEN 调用上指定了 MQOO_BIND_NOT_FIXED，或者使用 MQPUT1 来放置消息，那么可能发生此错误。

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

发出一个 MQOPEN、MQPUT 或 MQPUT1 调用以在集群队列上打开或放置消息。 集群工作负载出口拒绝调用。

z/OS 在 IBM MQ for z/OS 上使用和编写应用程序

IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

此信息说明了可供每个受支持的环境内运行的程序使用的 IBM MQ 工具。 此外，

- 有关使用 IBM MQ-CICS bridge 的信息，请参阅将 IBM MQ 与 CICS 配合使用。
- 有关使用 IMS 和 IMS 网桥的信息，请参阅第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』。

使用以下链接来查找有关在 IBM MQ for z/OS 上使用和编写应用程序的更多信息：

- 第 815 页的『依赖环境的 IBM MQ for z/OS 功能』
- 第 816 页的『调试工具、同步点支持和恢复支持』
- 第 817 页的『IBM MQ for z/OS 接口与应用程序环境』
- 第 818 页的『编写 z/OS UNIX 系统服务应用程序』
- 第 820 页的『使用共享队列的应用程序编程』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 56 页的『IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序』
此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

依赖环境的 *IBM MQ for z/OS* 功能

在考虑使用 IBM MQ for z/OS 功能时，请利用该信息。

在运行 IBM MQ for z/OS 的环境中，要考虑 IBM MQ 函数之间的主要差异：

- IBM MQ for z/OS 提供以下触发器监视器：

  - 在 CICS 环境中使用的 CKII
  - 在 IMS 环境中使用的 CSQQTRMN

  您必须自己编写模块以在其他环境中启动应用程序。
- 在 CICS 和 IMS 环境中，支持使用两阶段落实的指向同步点。 同时也在使用事务管理和可恢复资源管理器服务 (RRS) 的 z/OS 批处理环境中受到支持。 单阶段落实在 z/OS 环境中受 IBM MQ 本身支持。
- 对于批处理和 IMS 环境，MQI 提供调用以将程序连接到队列管理器，以及将其与队列管理器断开连接。 程序可以连接到多个队列管理器。
- 可以将 CICS 系统只连接到一个队列管理器。 如果在 CICS 系统启动作业中定义了子系统名称，那么在初始化 CICS 时可以执行此操作。 在 CICS 环境中，容许 MQI 连接和断开连接调用，但这不会产生任何影响。
- API 交叉出口允许程序干预所有 MQI 调用的处理。 该出口仅适用于 CICS 环境。
- 在多处理器系统上的 CICS 中，可以获得一些性能优势，这是因为可以在多个 z/OS TCB 下执行 MQI 调用。 有关更多信息，请参阅规划 z/OS *IBM MQ for z/OS Concepts and Planning Guide*。

第 816 页的表 134 中概述了这些功能。

表 *134: z/OS* 环境功能

	CICS	**IMS**	**Batch/TSO**
提供触发器监视器	Yes	Yes	否
两阶段落实	Yes	Yes	Yes
单阶段落实	Yes	否	Yes
连接/断开 MQI 调用	可容忍	Yes	Yes
API 交叉出口	Yes	否	否

注: 在使用 RRS 的 Batch/TSO 环境中支持两阶段落实。

调试工具、同步点支持和恢复支持

使用此信息来了解程序调试工具、同步点支持以及恢复支持。

程序调试工具

IBM MQ for z/OS 提供了跟踪功能，可用于在所有环境中调试您的程序。

此外，在 CICS 环境中还可以使用：

- CICS 执行诊断设施 (CEDF)
- CICS 跟踪控制事务 (CETR)
- IBM MQ for z/OS API 交叉出口

在 z/OS 平台上，可以使用所用编程语言支持的任何可用的交互式调试工具。

同步点支持

在事务处理环境中需要同步工作单元的开始和结束，以便可以安全地使用事务处理。

IBM MQ for z/OS 在 CICS 和 IMS 环境中完全支持此操作。 完全支持表示资源管理器之间相互合作，这样工作单元可以在 CICS 或 IMS 的控制下一致地落实或回退。 例如，资源管理器包括 Db2、CICS 文件控制、IMS 和 IBM MQ for z/OS。

z/OS 批处理应用程序可以使用 IBM MQ for z/OS 调用来提供单阶段落实功能。 这意味着可以落实或回退应用程序定义的一组队列操作，而无需引用其他资源管理器。

在使用事务管理和可恢复资源管理器服务 (RRS) 的 z/OS 批处理环境中，也支持两阶段落实。 有关更多信息，请参阅 z/OS 批处理应用程序中的同步点。

恢复支持

如果在事务期间队列管理器和 CICS 或 IMS 系统之间的连接中断，那么某些工作单元可能无法成功回退。

但是，在重新建立队列管理器（在同步点管理器的控制下）与 CICS 或 IMS 系统之间的连接时，该队列管理器会解析这些工作单元。

IBM MQ for z/OS 接口与应用程序环境

为了允许在不同环境中运行的应用程序通过消息排队网络发送和接收消息，IBM MQ for z/OS 为其支持的每个环境都提供一个适配器。

这些适配器是应用程序和 IBM MQ for z/OS 子系统之间的接口。 它们允许程序使用 MQI。

批处理适配器

利用该信息来了解批处理适配器和其支持的落实协议。

批处理适配器为采用以下方式运行的程序提供对 IBM MQ for z/OS 资源的访问：

- 任务 (TCB) 方式
- 问题或管理程序状态
- 主地址空间控制方式

程序不得采用跨内存方式。

应用程序和 IBM MQ for z/OS 之间的连接处于任务级别。 适配器提供从应用程序任务控制块 (TCB) 到 IBM MQ for z/OS 的单一连接线程。

对于对 IBM MQ for z/OS 拥有的资源所做的更改，适配器支持单阶段落实协议；它不支持多阶段落实协议。

RRS 批处理适配器

利用该信息来了解 RRS 批处理适配器和由 IBM MQ 提供的两个 RRS 批处理适配器。

事务管理和可恢复资源管理器服务 (RRS) 适配器：

- 将 z/OS RRS 用于落实控制。
- 支持从单个任务同时连接到在单个 z/OS 实例上运行的多个 IBM MQ 子系统。
- 为通过针对以下项的 z/OS RRS 兼容的可恢复管理器访问的可恢复资源，提供 z/OS 范围的协调落实控制（使用 z/OS RRS）：
  - 使用 RRS 批处理适配器连接到 IBM MQ 的应用程序。
  - Db2-在 z/OS 上由工作负载管理器 (WLM) 管理的 Db2 存储过程地址空间中执行的存储过程。
- 支持在 TCB 之间切换 IBM MQ 批处理线程的功能。

IBM MQ for z/OS 提供两个 RRS 批处理适配器：

CSQBRSTB
该适配器要求您在 IBM MQ 应用程序中将任何 MQCMIT 语句更改为 SRRCMIT，并将任何 MQBACK 语句更改为 SRRBACK。 （如果您在与 CSQBRSTB 链接的应用程序中编写 MQCMIT 和 MQBACK 的代码，那么会收到 MQRC_ENVIRONMENT_ERROR。）

CSQBRRSI
该适配器允许您的 IBM MQ 应用程序使用 MQCMIT 和 MQBACK，或者 SRRCMIT 和 SRRBACK。

注: CSQBRSTB 和 CSQBRRSI 附带链接属性 AMODE(31) RMODE(ANY)。 如果应用程序装入任一低于 16 MB 界限的存根，那么会首先重新链接存根与 RMODE(24)。

迁移

您可以迁移现有 Batch/TSO IBM MQ 应用程序，以便稍作更改或无需更改即可使用 RRS 协调。

如果使用 CSQBRRSI 适配器链接编辑您的 IBM MQ 应用程序，MQCMIT 和 MQBACK 会在 IBM MQ 和其他所有启用 RRS 的资源管理器之间对工作单元执行同步点操作。 如果使用 CSQBRSTB 适配器链接编辑您的 IBM MQ 应用程序，请将 MQCMIT 更改为 SRRCMIT，并将 MQBACK 更改为 SRRBACK。 后一种方法更为可取，它明确指出了同步点并不只限于 IBM MQ 资源。

IMS 适配器
如果您正在使用 IBM MQ for z/OS 系统中的 IMS 适配器，请确保 IMS 能够获取足够的存储空间来容纳长达 100 MB 的消息。

用户注意事项

IMS 适配器为以下项提供对 IBM MQ for z/OS 资源的访问：

- 在线消息处理程序 (MPP)
- 交互式快速路径程序 (IFP)
- 成批消息处理程序 (BMP)

要使用这些资源，必须在任务 (TCB) 方式和问题状态下运行这些程序，它们不能采用跨内存方式和访问注册方式。

适配器提供从应用程序任务控制块 (TCB) 到 IBM MQ 的连接线程。 对于对 IBM MQ for z/OS 拥有的资源所做的更改，适配器支持两阶段落实协议，IMS 充当同步点协调程序。

适配器还提供触发器监视器程序，可在满足队列上的某些触发条件时自动启动程序。 有关更多信息，请参阅第 795 页的『使用触发器启动 IBM MQ 应用程序』。

如果您正在编写批处理 DL/I 程序，请遵循本主题中为 z/OS 批处理程序提供的指导信息。

编写 *z/OS UNIX* 系统服务应用程序

批处理适配器支持来自批处理和 TSO 地址空间的队列管理器连接：

如果我们考虑批处理地址空间，那么适配器支持来自该地址空间中多个 TCB 的连接，如下所示：

- 每个 TCB 都可以使用 MQCONN 或 MQCONNX 调用连接到多个队列管理器（但一个 TCB 在任何时候都只有一个与特定队列管理器连接的实例）。
- 多个 TCB 可以连接到同一个队列管理器（但是在任何 MQCONN 或 MQCONN 调用上返回的队列管理器句柄都会绑定到发出的 TCB，并且不能由任何其他 TCB 使用）。

z/OS UNIX 系统服务支持两种类型的 pthread_create 调用：

1. 为每个 TCB 运行一个重量级线程，在线程开始和结束时通过 z/OS 执行 ATTACH 和 DETACH 操作。
2. 为每个 TCB 运行一个中量级线程，但 TCB 可以是大量长时间运行的 TCB 之一。 应用程序必须执行所有必要的应用程序清除操作，因为如果连接到服务器，并**不**总是会推动在任务 (TCB) 终止时可能由服务器提供的缺省线程终止。

不支持轻量级线程。 （如果应用程序创建了分派其工作请求的永久线程，该**应用程序**负责在启动下一个工作请求之前清除任何资源。）

IBM MQ for z/OS 使用批处理适配器支持 z/OS UNIX 系统服务线程，如下所示：

1. 完全支持将重量级线程作为批处理连接。 每个线程都在各自的 TCB 中运行，会在线程开始和结束时进行连接和分离。 如果线程在发出 MQDISC 调用之前结束，那么 IBM MQ for z/OS 会执行其标准任务清除操作，这包括在线程正常终止时提交任何未完成的工作单元，或者在线程异常终止时将其回退。
2. 完全支持中量级线程，但如果其他线程要复用 TCB，应用程序就必须确保在下一个线程启动之前发出 MQDISC 调用（之前先发出 MQCMIT 或 MQBACK）。 这意味着如果应用程序已经建立了程序中断处理程序，然后应用程序异常终止，中断处理程序必须在为另一个线程复用 TCB 之前发出 MQCMIT 和 MQDISC 调用。

注: 线程技术模型**不**支持从多个线程访问常用的 IBM MQ 资源。

针对 *z/OS* 的 *API* 交叉出口

本主题包含产品敏感的编程接口信息。

出口是 IBM 提供的代码中的一个点，您可以在其中运行自己的代码。 IBM MQ for z/OS 提供 *API* 交叉出口，您可以用来拦截对 MQI 的调用，以及监视或修改 MQI 调用的功能。 本章节描述了如何使用 API 交叉出口，并描述了 IBM MQ for z/OS 随附的样本出口程序。

本章节仅适用于 CICSTS V3.1 和更早版本的用户。 CICS TS V3.2 和更高版本的用户应参阅 CICS 产品文档中的 CICS Integration with IBM MQ 部分。

注

API 交叉出口仅由 IBM MQ for z/OS 的 CICS 适配器调用。 出口程序在 CICS 地址空间中运行。

编写自己的出口程序
您可以将 IBM MQ for z/OS 随附的 API 交叉出口程序 (CSQCAPX) 用作为自己程序的框架。

在第 819 页的『样本 API 交叉出口程序 CSQCAPX』中对此进行了描述。

在编写出口程序时，要查找由应用程序发出的 MQI 调用的名称，请检查 MQXP 结构的 *ExitCommand* 字段。 要查找调用上的参数数量，请检查 *ExitParmCount* 字段。 您可以使用 16 字节 *ExitUserArea* 字段来存储该应用程序获得的任何动态存储器的地址。 该字段保留在出口的调用中，并且其生命周期与 CICS 任务相同。

如果正在使用 CICS Transaction Server V3.2，那么必须将出口程序编写为线程安全的程序，并声明出口程序是线程安全的。 如果正在使用较早的 CICS 发行版，同样建议您编写并声明出口程序是线程安全的程序，以准备好迁移至 CICS Transaction Server V3.2。

您的出口程序可以通过在 *ExitResponse* 字段中返回 MQXCC_SUPPRESS_FUNCTION 或 MQXCC_SKIP_FUNCTION 来禁止执行 MQI 调用。 要允许执行该调用（且在调用完成之后重新调用该出口程序），出口程序必须返回 MQXCC_OK。

在 MQI 调用之后调用时，出口程序可以检查和修改该调用设置的完成代码和原因码。

使用说明

以下是编写出口程序时要考虑的一些常见要点：

- 出于性能原因，请使用汇编语言编写程序。 如果您使用 IBM MQ for z/OS 支持的其他任何语言编写程序，那么必须提供自己的数据定义文件。
- 将程序链接编辑为 AMODE(31) 和 RMODE(ANY)。
- 要定义程序的出口参数块，请使用汇编语言宏 CMQXPA。
- 在定义出口程序以及其调用的任何程序时，请指定 CONCURRENCY(THREADSAFE)。
- 如果您正在使用 CICS Transaction Server for z/OS 存储保护功能，那么您的程序必须在 CICS 执行密钥中运行。 也就是说，在定义出口程序及该程序向其传递控制的任何程序时，必须指定 EXECKEY(CICS)。 有关 CICS 出口程序和 CICS 存储保护功能的信息，请参阅 *CICS Customization Guide*。
- 您的程序可以使用所有 API (例如， IMS， Db2 和 CICS) CICS 任务相关的用户出口程序可以使用。 它也可以使用除 MQCONN、MQCONNX 和 MQDISC 之外的任何 MQI 调用。 但是，出口程序中的任何 MQI 调用都不会再次调用该出口程序。
- 您的应用程序可以发出 EXEC CICS SYNCPOINT 或 EXEC CICS SYNCPOINT ROLLBACK 命令。 但是，这些命令会落实或回滚由任务执行的**所有**更新，直到使用了该出口为止，所以不建议使用这些命令。
- 必须通过发出 EXEC CICS RETURN 命令来结束您的程序。 该程序不能使用 XCTL 命令传输控制。
- 将出口编写为 IBM MQ for z/OS 代码的扩展。 确保您的出口不会中断任何使用 MQI 的 IBM MQ for z/OS 程序或事务。 这些通常用前缀 CSQ 或 CK 来指示。
- 如果将 CSQCAPX 定义到 CICS，那么当 CICS 连接到 IBM MQ for z/OS 时， CICS 系统会尝试装入出口程序。 如果此尝试成功，会将消息 CSQC301I 发送至 CKQC 面板或系统控制台。 如果装入失败（例如，如果装入模块不存在于 DFHRPL 连接的任何库中），会将消息 CSQC315 发送至 CKQC 面板或系统控制台。
- 因为通信区中的参数是地址，所以必须将出口程序定义为 CICS 系统的本地程序（即，不能作为远程程序）。

样本 *API* 交叉出口程序 *CSQCAPX*
样本出口程序作为汇编语言程序提供。 源文件 (CSQCAPX) 在库 **thlqual**.SCSQASMS 中提供（其中 **thlqual** 是安装所使用的高级限定符）。 该源文件包含描述该程序逻辑的伪码。

样本程序包含您在编写自己的出口程序时可使用的初始化代码和布局。

样本显示如何：

- 设置出口参数块
- 处理调用和出口参数块
- 决定正在为其调用出口的 MQI 调用
- 决定是在处理 MQI 调用之前还是之后调用出口
- 将消息放置在 CICS 临时存储队列上
- 使用宏 DFHEIENT 以获取动态存储器，从而保持可重入性
- 将 DFHEIBLK 用于 CICS exec 接口控制块
- 捕获错误条件
- 将控制返回给调用者

样本出口程序的设计

样本出口程序将消息写入 CICS 临时存储队列 (CSQ1EXIT) 以显示出口的操作。

这些消息显示了是在 MQI 调用之前还是之后调用出口。 如果在调用之后调用了出口，那么消息包含调用返回的完成代码和原因码。 样本使用 CMQXPA 宏的命名常量来检查条目的类型（即，在调用之前或之后）。

该样本不执行任何监视功能，而只是将带有时间戳记的消息放入到指示其所处理调用类型的 CICS 队列中。 这指示了 MQI 的性能，表明出口程序正确运行。

注: 样本出口程序会对程序运行时生成的每个 MQI 调用发出 6 个 EXEC CICS 调用。 如果您使用此出口程序，IBM MQ for z/OS 性能会降低。

准备并使用 *API* 交叉出口
样本出口仅以源形式提供。

要使用样本出口或已经编写的出口程序，请创建装入库，与为任何其他 CICS 程序创建一样，如第 938 页的『在 z/OS 中构建 CICS 应用程序』中所述。

- 对于 CICS Transaction Server for z/OS 和 CICS for MVS/ESA，在更新 CICS 系统定义 (CSD) 数据集时，所需的定义位于成员 **thlqual**.SCSQPROC(CSQ4B100) 中。

  注: 这些定义使用 MQ 后缀。 如果已在企业中使用该后缀，那么在组合件登台之前必须更改此后缀。

如果您使用提供的缺省 CICS 程序定义，那么安装的出口程序 CSQCAPX 处于**已禁用**状态。 这是由于使用出口程序会显著降低性能。

要暂时激活 API 交叉出口：

1. 从 CICS 主终端发出命令 **CEMT S PROGRAM(CSQCAPX) ENABLED** 。
2. 运行 CKQC 事务，使用“连接”下拉菜单中的选项 3 将 API 交叉出口的状态更改为**已启用**。

如果想要在永久启用 API 交叉出口的情况下运行 IBM MQ for z/OS，请通过 CICS Transaction Server for z/OS 和 CICS for MVS/ESA 执行以下操作之一：

- 更改成员 CSQ4B100 中的 CSQCAPX 定义，将 STATUS(DISABLED) 更改为 STATUS(ENABLED)。 您可以使用 CICS 提供的批处理程序 DFHCSDUPY 来更新 CICS CSD 定义。
- 通过将状态从 DISABLED 更改为 ENABLED 来更改 CSQCAT1 组中的 CSQCAPX 定义。

在这两种情况下，您都必须重新安装该组。 您可以通过冷启动 CICS 系统或在 CICS 运行时使用 CICS CEDA 事务重新安装该组来执行此操作。

注: 如果该组中的任何条目当前正在使用，那么使用 CEDA 可能会导致错误。

产品敏感的编程接口信息的结束。

使用共享队列的应用程序编程

本主题提供了在设计新应用程序以使用共享队列以及将现有应用程序迁移到共享队列环境时，需要考虑的一些因素的相关信息。

实现应用程序序列化

某些类型的应用程序可能必须确保按照消息到达队列的顺序从队列中检索消息。

例如，如果正在使用 IBM MQ 将数据库更新映射到远程系统，必须在描述记录插入的消息之后处理描述该记录更新的消息。 在本地排队环境中，这通常由以下应用程序来实现：该应用程序正在获取消息，使用 MQOO_INPUT_EXCLUSIVE 选项打开队列，从而防止任何其他获取应用程序同时处理该队列。

IBM MQ 允许应用程序采用独占方式以相同的方法打开共享队列。 但是，如果应用程序正在通过队列的一个分区运行（例如，所有数据库更新都在同一队列上，但表 A 的更新具有相关标识 A，表 B 的更新具有相关标识 B），且应用程序想要同时获取表 A 更新和表 B 更新的消息，那么无法实现以独占方式打开队列这一简单机制。

如果该类型的应用程序将充分利用共享队列的高可用性，您可能会决定在主获取应用程序或队列管理器失败时，访问相同共享队列且运行在辅助队列管理器上的应用程序的另一个实例应接管工作。

如果主队列管理器失败，那么会发生以下两种情况：

- 共享队列对等恢复可确保完成或回退主应用程序的任何未完成的更新。
- 辅助应用程序会接管该队列的处理工作。

辅助应用程序可能会在处理完所有未完成的工作单元之前启动，这可能会使辅助应用程序不按顺序检索消息。 要解决这类问题，可以选择序列化的应用程序。

序列化的应用程序使用 MQCONNX 调用连接到队列管理器，指定在连接时该应用程序唯一的连接标记。 该应用程序执行的任何工作单元都标记有该连接标记。 IBM MQ 可确保将队列共享组中带有相同连接标记的工作单元序列化（根据 MQCONNX 调用上的序列化选项）。

这意味着，如果主应用程序使用带有连接标记 `Database shadow retriever` 的 MQCONNX 调用，并且辅助接管应用程序尝试使用带有相同连接标记的 MQCONNX 调用，那么在此情况下，辅助应用程序无法连接到第二个 IBM MQ ，直到已完成任何未完成的主工作单元 (通过对等恢复)。

对于依赖于队列上消息的准确序列的应用程序，考虑使用序列化的应用程序方法。 尤其是：

- 在应用程序或队列管理器失败之后，直到完成该应用程序先前执行的所有落实和回退操作，才能重新启动的应用程序。

  在这种情况下，仅当应用程序在同步点工作时才适用序列化的应用程序方法。
- 在同一应用程序的另一个实例已经运行时，不得启动的应用程序。

  在这种情况下，仅当应用程序无法打开用于独占输入的队列时，才会需要序列化的应用程序方法。

注: IBM MQ 只保证在满足特定条件时才会保留消息序列。 这些在 MQGET 的说明中进行了描述。

不适合与共享队列一起使用的应用程序

使用共享队列时，不支持 IBM MQ 的某些功能，因此使用这些功能的应用程序不适用于共享队列环境。

在设计共享队列应用程序时，请考虑以下几点：

- 为共享队列建立队列索引是受到限制的。 如果想要使用消息标识或相关标识来选择要从该队列获取的消息，那么应该使用正确的值来建立队列索引。 如果您只通过消息标识来选择消息，那么该队列需要索引类型 MQIT_MSG_ID（尽管您也可以使用 MQIT_NONE）。 如果您只通过相关标识来选择消息，那么队列必须具有索引类型 MQIT_CORREL_ID。
- 您无法将临时动态队列作为共享队列使用。 但是，您可以使用永久动态队列。 虽然共享动态队列模型的创建和销毁方式与 PERMDYN（永久动态）队列的方式相同，但它们的 DEFTYPE 为 SHAREDYN（共享动态）。

决定是否共享非应用程序队列

在考虑共享非应用程序队列时利用该信息。

这些是除应用程序队列之外您可能考虑共享的队列。

启动队列

如果您定义共享启动队列，那么不需要在队列共享组中的每个队列管理器上都运行触发器监视器，只要至少有一个触发器监视器在运行即可。（即使队列共享组中的每个队列管理器上都运行一个触发器监视器，您也可以使用共享启动队列。）

如果您拥有共享应用程序队列，并使用触发器类型 EVERY（或触发器类型 FIRST，具有很短的触发间隔，其行为类似于触发器类型 EVERY），那么您的启动队列必须始终为共享队列。 有关何时使用共享启动队列的更多信息，请参阅第 822 页的表 135。

SYSTEM.* 队列

您可以定义 SYSTEM.ADMIN.* 用于将事件消息作为共享队列保存的队列。 如果发生异常，这可用于检查负载均衡。 IBM MQ 创建的每条事件消息都包含相关标识，表明了哪个队列管理器生成了该消息。

必须定义 SYSTEM.QSG* 用于共享通道和组内排队 (作为共享队列) 的队列。

您也可以更改要共享的 SYSTEM.DEFAULT.LOCAL.QUEUE 的定义，或定义自己的缺省共享队列定义。 请参阅 为 IBM MQ for z/OS 定义系统对象 以获取更多信息。

不能定义任何其他 SYSTEM.* 队列作为共享队列。

迁移现有的应用程序以使用共享队列

原因码、触发和 MQINQ API 调用可以在共享队列环境中以不同的方式工作。

有关将现有队列迁移到共享队列的信息，请参阅 将非共享队列迁移到共享队列 。

在迁移现有应用程序时，请考虑以下几点，这可能会在共享队列环境中以不同的方式工作：

原因码

在迁移现有应用程序以使用共享队列时，请检查可发出的新原因码。

触发

如果您正在使用共享应用程序队列，触发只对已落实的消息生效（在非共享应用程序队列上，触发对所有消息生效）。

如果使用触发启动应用程序，您可以想要使用共享启动队列。 第 822 页的表 135 描述了在决定使用哪种类型的启动队列时需考虑的事项。

表 *135:* 何时使用共享启动队列

	非共享应用程序队列	**共享应用程序队列**
非共享启动队列	针对先前发行版。	如果您使用触发器类型 FIRST 或 DEPTH，那么可以将非共享启动队列与共享应用程序队列结合使用。 可能会生成额外的触发器消息，但该设置适用于触发长期运行的应用程序（如 CICS bridge），并提供了高可用性。 对于触发器类型 FIRST 或 DEPTH，针对正在运行触发器监视器且尚未打开应用程序队列以进行输入的每个队列管理器，触发器消息会触发一个应用程序实例。 将为每个队列管理器生成一条触发器消息；如果针对非共享本地启动队列运行的触发器监视器不止一个，那么在特定的队列管理器上，它们将争相处理该消息。

表 135: 何时使用共享启动队列 (继续)		
	非共享应用程序队列	**共享应用程序队列**
共享启动队列	请勿将共享启动队列与非共享应用程序队列结合使用。	对于触发器类型 EVERY，当应用程序将消息放入共享应用程序队列时，放入队列管理器将确定哪些队列管理器对触发器中的每个事件有兴趣，并向其中一个队列管理器发送通知。 在通知的队列管理器上，生成的操作是生成到启动队列的触发器消息。 **注:** 如果您具有触发器类型为 EVERY 的共享应用程序队列，请使用共享启动队列，否则您可能会在某些情况下丢失触发器消息，例如，队列管理器出现故障时。 对于触发器类型 FIRST 或 DEPTH，打开了指定启动队列以进行输入的每个队列管理器都会生成一条触发器消息。 **注:** 对于触发器类型 FIRST 或 DEPTH，如果一个触发器监视器实例正忙，这可能会使另一个不太忙的触发器监视器处理来自共享启动队列的多条触发器消息。 因此，可能会针对给定队列管理器启动服务器应用程序的多个实例。 请注意，这些实例是由于处理多条触发器消息而启动的。 通常情况下，对于触发器类型 FIRST 或 DEPTH，如果应用程序实例已经服务于应用程序队列，那么该应用程序连接的队列管理器将不会再生成一条触发器消息。

MQINQ
在使用 MQINQ 调用来显示共享队列的信息时，打开队列以进行输入和输出的 MQOPEN 调用数的值，仅与发出该调用的队列管理器相关。 在打开队列的队列共享组中，不会生成其他队列管理器的信息。

z/OS IBM MQ for z/OS 上的 IMS 和 IMS 网桥应用程序

此信息可帮助您使用 IBM MQ 编写 IMS 应用程序。

- 要在 IMS 应用程序中使用同步点和 MQI 调用，请参阅第 57 页的『使用 IBM MQ 编写 IMS 应用程序』。
- 要编写使用 IBM MQ - IMS 网桥的应用程序，请参阅第 60 页的『编写 IMS 桥接应用程序』。

使用以下链接了解有关 IMS 和 IBM MQ for z/OS 上的 IMS 桥接应用程序的更多信息：

- 第 57 页的『使用 IBM MQ 编写 IMS 应用程序』
- 第 60 页的『编写 IMS 桥接应用程序』

相关概念

第 673 页的『消息队列接口概述』
了解有关消息队列接口 (MQI) 组件的信息。

第 685 页的『连接到队列管理器和从队列管理器断开连接』
要使用 IBM MQ 编程服务，程序必须具有与队列管理器的连接。 使用此信息以了解如何连接到队列管理器以及从其断开连接。

第 692 页的『打开和关闭对象』
使用此信息，可深入了解打开和关闭 IBM MQ 对象。

第 700 页的『将消息放置到队列上』
使用此信息以了解如何将消息放置到队列上。

第 713 页的『从队列取出消息』
使用此信息以了解如何从队列取出消息。

第 783 页的『查询和设置对象属性』
属性是用于定义 IBM MQ 对象特征的特性。

第 786 页的『落实和回退工作单元』
此信息描述了如何落实和备份在工作单元中发生的任何可恢复的获取和放置操作。

第 795 页的『使用触发器启动 IBM MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM MQ 应用程序。

第 811 页的『与 MQI 和集群配合使用』
对于与集群相关的调用和返回码存在特殊选项。

第 815 页的『在 IBM MQ for z/OS 上使用和编写应用程序』
IBM MQ for z/OS 应用程序可由许多不同环境内运行的程序组成。 这意味着他们可以利用在多个环境中可用的设施。

使用 *IBM MQ* 编写 *IMS* 应用程序

在 IMS 应用程序中使用 IBM MQ 时还有其他注意事项。这些注意事项包括可以使用哪些 MQ API 调用以及用于同步点的机制。

使用以下链接可了解有关在 IBM MQ for z/OS 上编写 IMS 应用程序的更多信息:

- 第 57 页的『IMS 应用程序中的同步点』
- 第 58 页的『IMS 应用程序中的 MQI 调用』

限制

存在使用 IMS 适配器的应用程序可以使用 IBM MQ API 调用的限制。

以下 IBM MQ API 调用在使用 IMS 适配器的应用程序中不受支持:

- MQCB
- MQCB_FUNCTION
- MQCTL

相关概念

第 60 页的『编写 IMS 桥接应用程序』
本主题包含有关编写应用程序以使用 IBM MQ - IMS 网桥的应用程序的信息。

IMS 应用程序中的同步点
在 IMS 应用程序中，通过使用 IMS 调用（如 GU，获取唯一）IOPCB 和 CHKP（检查点）来建立同步点。

要回退从先前检查点开始的所有更改，可以使用 IMS ROLB（回滚）调用。 有关更多信息，请参阅 IMS 文档中的 ROLB 调用 。

队列管理器是两阶段落实协议的参与者；IMS 同步点管理器是协调者。

所有打开的句柄由 IMS 适配器在同步点时关闭（除了批处理或非消息驱动的 BMP 环境）。 这是因为执行 MQCONN、MQCONNX 和 MQOPEN 调用（而非 MQPUT 或 MQGET 调用）时不同用户可以启动下一个工作单元，并且执行 IBM MQ 安全性检查。

但在等待输入 (WFI) 或伪等待输入 (PWFI) 环境中，只有收到下一条消息或者将 QC 状态码返回至应用程序之后，IMS 才会通知 IBM MQ 关闭句柄。 如果应用程序在 IMS 区域内等待，并且所有句柄都属于已触发队列，那么由于队列处于打开状态，将不会发生触发。 因此，在为下一消息执行 GU 调用 IOPCB 之前，在 WFI 或 PWFI 环境中运行的应用程序应该以显式方式使用 MQCLOSE 关闭队列句柄。

如果 IMS 应用程序（BMP 或 MPP）发出 MQDISC 调用，那么将关闭打开的队列，但不会生成隐式同步点。 如果应用程序正常结束，所有打开的队列将关闭，并会发生隐式提交。 如果应用程序异常结束，所有打开的队列将关闭，并会发生隐式回退。

IMS 应用程序中的 *MQI* 调用
通过此信息了解 MQI 调用在服务器应用程序和查询应用程序上的使用。

本节介绍在以下 IMS 应用程序类型中使用 MQI 调用:

- 第 825 页的『服务器应用程序』
- 第 827 页的『查询应用程序』

服务器应用程序

此处为 MQI 服务器应用程序模型的概述：

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

样本程序 CSQ4ICB3 显示了在 C/370 中使用此型号实施 BMP。 程序首先与 IMS 建立通信，然后与 IBM MQ 建立通信：

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

Return
```

IMS 初始化确定是作为消息驱动的 BMP 调用程序还是作为面向批处理的 BMP 调用程序，同时相应地控制 IBM MQ 队列管理器连接和队列句柄：

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

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

```
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

IBM MQ 初始化连接到队列管理器并打开队列。 在消息驱动的 BMP 中，该调用发生在获取每个 IMS 同步点之后，在面向批处理的 BMP 中，仅在程序启动期间进行此调用。

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

Return to calling function
```

在 MPP 中实施服务器型号受到每次调用 MPP 只处理单个工作单元这一事实的影响。 这是因为在获取同步点 (GU) 时，连接和队列句柄会关闭并会发送下一条 IMS 消息。 以下方法可以在一定程度上克服此限制：

- **在一个工作单元内处理多条消息**

  这包括在循环中：

  – 正在读取消息
  – 处理所需的更新
  – 放置应答

  直到所有消息均处理完成，或者已处理最大消息数量，此时将获取同步点。

  该方法只适用于特定类型的应用程序（例如，简单的数据库更新或查询）。 尽管 MQI 应答消息可以通过正在处理的 MQI 消息的发起方权限放入，但任何 IMS 资源更新的安全含义都需要小心处理。

- **每个 MPP 调用处理一条消息并确保多个 MPP 调度处理所有可用消息**。

  使用 IBM MQ IMS 触发器监视器程序 (CSQQTRMN) 在 IBM MQ 队列上存在消息并且没有应用程序为其提供服务时调度 MPP 事务。

  如果触发器监视器启动 MPP，队列管理器名称和队列名称将传递到程序，如以下 COBOL 代码段所示：

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
```

```
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME        ='
MQTMC-QNAME  OF MQTMC '='.
```

尽管 BMP 不能使用 CSQQTRMN 触发，但此服务器型号（长时间运行的任务）最好在批处理区域内受支持。

查询应用程序

一般的 IBM MQ 应用程序按照以下过程启动查询或更新工作：

- 收集用户数据
- 放置一条或多条 IBM MQ 消息
- 获取应答消息（可能需要等待）
- 向用户提供响应

因为放入 IBM MQ 队列的消息在提交之前不可用于其他 IBM MQ 应用程序，所以这些消息必须放在同步点外，或者 IMS 应用程序必须分为两个事务。

如果查询涉及放置单条消息，您可以使用无同步点选项；但是，当查询更为复杂，或者涉及资源更新时，如果出现故障并且您不使用同步点，您可能会遇到一致性问题。

要解决此问题，您可以使用程序间消息开关通过 MQI 调用拆分 IMS MPP 事务; 请参阅 *IMS* 系统间通信 *(ISC)* 以获取有关此问题的信息。 这使得查询程序可以在 MPP 中实施：

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

编写 *IMS* 桥接应用程序

本主题包含有关编写应用程序以使用 IBM MQ - IMS 网桥的应用程序的信息。

有关 IBM MQ - IMS 网桥的信息，请参阅 IMS 网桥。

使用以下链接了解有关在 IBM MQ for z/OS 上编写 IMS 桥接应用程序的更多信息：

- 第 61 页的『IMS 网桥如何处理消息』
- 第 833 页的『通过 IBM MQ 编写 IMS 事务程序』

相关概念

第 57 页的『使用 IBM MQ 编写 IMS 应用程序』
在 IMS 应用程序中使用 IBM MQ 时还有其他注意事项。这些注意事项包括可以使用哪些 MQ API 调用以及用于同步点的机制。

IMS 网桥如何处理消息
使用 IBM MQ - IMS 网桥向 IMS 应用程序发送消息时，需要以特殊格式构造消息。

您还必须将消息放入定义了存储类的 IBM MQ 队列，存储类用于指定目标 IMS 系统的 XCF 组和成员名称。这些队列称为 MQ-IMS 网桥队列，或简称为**网桥**队列。

如果网桥队列定义了 QSGDISP(QMGR)，或者定义了 QSGDISP(SHARED) 和 NOSHARE 选项，IBM MQ-IMS 网桥需要对网桥队列具有独占输入访问权 (MQOO_INPUT_EXCLUSIVE)。

用户在向 IMS 应用程序发送消息之前不需要登录 IMS。 MQMD 结构的 *UserIdentifier* 字段中的用户标识用于安全性检查。 检查级别在 IBM MQ 连接到 IMS 时确定，并在 IMS 网桥的应用程序访问控制中进行了描述。 这使得伪登录可以实施。

IBM MQ - IMS 网桥接受以下消息类型：

- 包含 IMS 事务数据和 MQIIH 结构（MQIIH 中进行了描述）的消息：

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

注：

1. 方括号 [ ] 表示可选的多区段。
2. 将 MQMD 结构的 *Format* 字段设置为 MQFMT_IMS 以使用 MQIIH 结构。

- 包含 IMS 事务数据，但不包含 MQIIH 结构的消息：

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

IBM MQ 将对消息数据进行验证以确保 LL 字节数加上 MQIIH（如果存在）的长度总和等于消息长度。

IBM MQ - IMS 网桥收到来自网桥队列的消息时，会按照以下过程处理这些消息：

- 如果消息中包含 MQIIH 结构，网桥将验证 MQIIH（请参阅 MQIIH），构建 OTMA 头并将此消息发送到 IMS。 事务代码在输入消息中指定。 如果事务代码为 LTERM，IMS 将应答 DFS1288E 消息。 如果事务代码表示命令，IMS 将执行此命令；否则，消息将在事务的 IMS 中排队。
- 如果消息包含 IMS 事务数据，但不包含 MQIIH 结构，IMS 网桥将进行以下假设：
  - 事务代码是 5 - 12 个字节的用户数据
  - 事务处于非会话模式
  - 会话使用提交模式 0（提交然后发送）
  - MQMD 中的 *Format* 用作 *MFSMapName*（输入时）
  - 安全模式为 MQISS_CHECK

  还会在不使用 MQIIH 结构的情况下构建应答消息，从 IMS 输出的 *MFSMapName* 中获取 MQMD 的 *Format* 。

IBM MQ - IMS 网桥为每个 IBM MQ 队列使用一个或两个 Tpipe：

- 同步的 Tpipe 用于使用落实方式 0 (COMMIT_THEN_SEND) 的所有消息 (这些显示在 IMS /DIS TMEMBER 客户机 TPIPE xxxx 命令的状态字段中带有 SYN)
- 非同步 Tpipe 用于使用提交模式 1 (SEND_THEN_COMMIT) 的所有消息

Tpipe 由 IBM MQ 在第一次使用时创建。 非同步 Tpipe 在 IMS 重新启动前一直存在。 同步 Tpipe 在 IMS 冷启动之前一直存在。 您自己无法删除这些 Tpipe。

请参阅以下主题以了解有关 IBM MQ - IMS 网桥如何处理消息的更多信息：

- 第 62 页的『将 IBM MQ 消息映射到 IMS 事务类型』
- 第 62 页的『如果消息无法放到 IMS 队列』
- 第 63 页的『IMS 网桥反馈代码』
- 第 63 页的『IMS 网桥消息中的 MQMD 字段』
- 第 64 页的『IMS 网桥消息中的 MQIIH 字段』
- 第 65 页的『来自 IMS 的回复消息』
- 第 65 页的『在 IMS 事务中使用备用响应 PCB』
- 第 65 页的『发送来自 IMS 的自发消息』

- 第 65 页的『消息分段』
- 第 66 页的『消息与 IMS 网桥之间的数据转换』

相关概念

第 833 页的『通过 IBM MQ 编写 IMS 事务程序』

通过 IBM MQ 处理 IMS 事务所需的编码取决于 IMS 事务所需的消息格式及其可返回的响应范围。 但是，在应用程序处理 IMS 屏幕格式化信息时，应考虑以下几点。

将 *IBM MQ* 消息映射到 *IMS* 事务类型

下表描述 IBM MQ 消息到 IMS 事务类型的映射。

表 *136:* 如何将 *IBM MQ* 消息映射到 *IMS* 事务类型

IBM MQ 消息类型	**落实后发送（方式 0）- 使用同步 IMS Tpipe**	**发送后落实（方式 1）- 使用非同步 IMS Tpipe**
持久 IBM MQ 消息	• 可恢复完全功能事务 • IMS 拒绝不可恢复事务	• 快速路径事务 • 对话式事务 • 完全功能事务
非持久 IBM MQ 消息	• 不可恢复完全功能事务 • IMS V8 和 APAR PQ61404 以及 IMS 的所有更高版本允许可恢复事务	• 快速路径事务 • 对话式事务 • 完全功能事务

注: IMS 命令不能将持久 IBM MQ 消息用于落实方式 0。 请参阅 落实方式 (commitMode) 以获取更多信息。

如果消息无法放到 *IMS* 队列

了解在消息无法放到 IMS 队列时要采取的操作。

如果消息无法放到 IMS 队列，那么 IBM MQ 将采取以下操作：

- 如果消息由于无效而无法放到 IMS，那么会将该消息放到死信队列，并向系统控制台发送消息。
- 如果消息有效但遭到 IMS 拒绝，那么 IBM MQ 会向系统控制台发送错误消息，该消息包含 IMS 检测代码，并且 IBM MQ 消息放到死信队列。 如果 IMS 检测代码为 001A，那么 IMS 会发送包含应答队列故障原因的 IBM MQ 消息。

  注: 在先前所列的情况下，如果 IBM MQ 由于任何原因无法将消息放到死信队列，那么该消息会返回到始发 IBM MQ 队列。 将向系统控制台发送错误消息，并且不会从该队列发送任何其他消息。

  要重新发送消息，请执行以下其中一**项**操作：

  – 在与队列对应的 IMS 中停止并重新启动 Tpipe
  – 将队列变更为 GET(DISABLED)，然后再次变更为 GET(ENABLED)
  – 停止并重新启动 IMS 或 OTMA
  – 停止并重新启动 IBM MQ 子系统
- 如果消息由于除消息错误以外的任何其他原因遭到 IMS 拒绝，那么 IBM MQ 消息会返回到始发队列，IBM MQ 停止处理队列，并向系统控制台发送错误消息。

  如果需要异常报告消息，那么网桥会使用发起方的权限将其放到应答队列。 如果消息无法放到该队列，那么会使用网桥的权限将报告消息放到死信队列。 如果它无法放到 DLQ，那么会将其丢弃。

IMS 网桥反馈代码

IMS 检测代码通常是诸如 CSQ2001I 之类的 IBM MQ 控制台消息中十六进制格式的输出（例如，检测代码 0x001F）。 放到死信队列的消息的死信消息头中所显示的 IBM MQ 反馈代码是十进制数字。

IMS 网桥反馈代码在范围 301 到 399 内，对于 NACK 检测代码 0x001A 则在范围 600 到 855 内。 它们按如下所述从 IMS-OTMA 检测代码进行映射：

1. IMS-OTMA 检测代码从十六进制数字转换为十进制数字。
2. 将 300 添加到 1 中的计算所产生的数字中，并给出 IBM MQ *Feedback* 代码。
3. IMS-OTMA 检测代码 0x001A，十进制 26 是特殊情况。 将生成范围 600-855 内的反馈代码。
   a. IMS-OTMA 原因码从十六进制数字转换为十进制数字。
   b. 将 a 中的计算产生的数字加上 600，得出 IBM MQ 反馈代码。

有关 IMS-OTMA 检测代码的信息，请参阅用于 NAK 消息的 OTMA 检测代码。

IMS 网桥消息中的 *MQMD* 字段

了解有关来自 IMS 网桥的消息中 MQMD 字段的信息。

始发消息的 MQMD 由 IMS 在 OTMA 头的“用户数据”部分中携带。 如果消息源于 IMS，那么该消息由 IMS 目标解析出口构建。 从 IMS 接收的消息的 MQMD 构建如下：

StrucID
“MD”

版本
MQMD_VERSION_1

报告
MQRO_NONE

MsgType
MQMT_REPLY

到期
如果在 MQIIH 的“标志”字段中设置了 MQIIH_PASS_EXPIRATION，那么此字段包含剩余到期时间，否则它设置为 MQEI_UNLIMITED

Feedback
MQFB_NONE

编码
MQENC.Native（z/OS 系统的编码）

CodedCharSetId
MQCCSI_Q_MGR（z/OS 系统的 CodedCharSetID）

格式
如果输入消息的格式为 MQFMT_IMS，那么值为 MQFMT_IMS，否则值为 IOPCB.MODNAME

优先级
输入消息的 MQMD.Priority

持久
取决于落实方式：如果方式为 CM-1，那么值为输入消息的 MQMD.Persistence；如果方式为 CM-0，那么持久性与 IMS 消息的可恢复性相匹配

MsgId
如果是 MQRO_PASS_MSG_ID，那么值为 MQMD.MsgId，否则值为新 MsgId（缺省值）

CorrelId
如果是 MQRO_PASS_CORREL_ID，那么值为来自输入消息的 MQMD.CorrelId，否则值为来自输入消息的 MQMD.MsgId（缺省值）

BackoutCount
0

ReplyToQ
空白

ReplyToQMgr
空白（在 MQPUT 期间由队列管理器设置为本地 qmgr 名称）

UserIdentifier
输入消息的 MQMD.UserIdentifier

AccountingToken
输入消息的 MQMD.AccountingToken

ApplIdentityData
输入消息的 MQMD.ApplIdentityData

PutApplType
如果没有错误，那么值为 MQAT_XCF，否则值为 MQAT_BRIDGE

PutApplName
如果没有错误，那么值为 <XCFgroupName><XCFmemberName>，否则值为 QMGR 名称

PutDate
消息的放置日期

PutTime
消息放置的时间

ApplOriginData
空白

IMS 网桥消息中的 *MQIIH* 字段
了解有关来自 IMS 网桥的消息中 MQIIH 字段的信息。

从 IMS 接收的消息的 MQIIH 构建如下：

StrucId
“IIH”

版本
1

StrucLength
84

编码
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

格式
如果 MQIIH.ReplyToFormat 不是空白，那么值为输入消息的 MQIIH.ReplyToFormat，否则值为 IOPCB.MODNAME

标志
0

LTermOverride
来自 OMTA 头的 LTERM 名称 (Tpipe)

MFSMapName
来自 OMTA 头的映射名称

ReplyToFormat
空白

Authenticator
如果是将回复消息放到 MQ-IMS 网桥队列，那么值为输入消息的 MQIIH.Authenticator；否则值为空白。

TranInstanceId
如果在对话中，那么值为来自 OTMA 头的对话标识/服务器令牌。 在 V14 之前的 IMS 版本中，如果不在对话中，那么此字段始终为空。 从 IMS V14 开始，可以通过 IMS 设置该字段，即使不在对话中。

TranState
如果在对话中，那么值为“C”，否则值为空白

CommitMode
来自 OTMA 头的落实方式（“0”或“1”）

SecurityScope
Blank

已保留
Blank

来自 *IMS* 的回复消息

当 IMS 事务 ISRT 到其 IOPCB 时，会将消息路由回原始 LTERM 或 TPIPE。

这些消息在 IBM MQ 中被视为回复消息。 来自 IMS 的回复消息放到原始消息中指定的应答队列上。 如果消息无法放到应答队列上，那么会使用网桥的权限将其放到死信队列上。 如果消息无法放到死信队列，那么会向 IMS 发送否定确认以表明无法接收消息。 于是，消息是否重发的责任又回到 IMS 手里。 如果使用的是落实方式 0，那么来自该 Tpipe 的消息不会发送到网桥，并且保留在 IMS 队列上；即，在重新启动之前不会发送任何其他消息。 如果使用的是落实方式 1，那么其他工作可以继续。

如果回复具有 MQIIH 结构，那么其格式类型为 MQFMT_IMS；如果没有此结构，那么其格式类型由插入消息时所使用的 IMS MOD 名称指定。

在 *IMS* 事务中使用备用响应 *PCB*

当 IMS 事务使用备用响应 PCB（ISRT 到 ALTPCB，或对可修改 PCB 发出 CHNG 调用）时，会调用预路由出口 (DFSYPRX0) 以确定是否应该重新路由该消息。

如果要重新路由消息，那么会调用目标解析出口 (DFSYDRU0) 来确认目标并准备头信息。请参阅在 IMS 中使用 OMTA 出口和预路由出口 DFSYPRX0，以获取有关这些出口程序的信息。

除非在出口中执行操作，否则从 IBM MQ 队列管理器 (无论是 IOPCB 还是 ALTPCB) 启动的 IMS 事务的所有输出都将返回到同一队列管理器。

发送来自 *IMS* 的自发消息

要将消息从 IMS 发送到 IBM MQ 队列，您需要调用要 ISRT 到 ALTPCB 的 IMS 事务。

您需要编写预路由出口和目标解析出口以路由来自 IMS 的自发消息并构建 OTMA 用户数据，以便可以正确构建消息的 MQMD。 请参阅预路由出口 DFSYPRX0 和目标解析用户出口以获取有关这些出口程序的信息。

注: IBM MQ - IMS 网桥不知道它接收的消息是回复消息还是自发消息。 它在各种情况下以相同方式处理消息，从而根据随消息到达的 OTMA UserData 构建回复的 MQMD 和 MQIIH

自发消息可以创建新 Tpipe。 例如，如果现有 IMS 事务切换到新 LTERM（例如 PRINT01），但是实施要求通过 OTMA 传递输出，那么将会创建新 Tpipe（在本例中名为 PRINT01）。 缺省情况下，这是非同步 Tpipe。 如果实施要求消息可恢复，请设置目标解析出口输出标志。 请参阅《*IMS* 定制指南》以获取更多信息。

消息分段

您可以将 IMS 事务定义为期望单段或多段输入。

始发 IBM MQ 应用程序必须将遵循 MQIIH 结构的用户输入构造为一个或多个 LLZZ 数据段。 IMS 消息的所有段都必须包含在通过单个 MQPUT 发送的单条 IBM MQ 消息中。

LLZZ 数据段的最大长度由 IMS/OTMA 定义（32767 字节）。 IBM MQ 消息总长度是 LL 字节的总和，加上 MQIIH 结构的长度。

回复的所有段都包含在单条 IBM MQ 消息中。

对于格式为 MQFMT_IMS_VAR_STRING 的消息的 32 KB 限制存在进一步制约。 当 ASCII 混合 CCSID 消息中的数据转换为 EBCDIC 混合 CCSID 消息时，每次 SBCS 和 DBCS 字符之间存在转换时，将会添加移入或移出字节。 32 KB 限制适用于消息的最大大小。 即，由于消息中的 LL 字段不能超过 32 KB，因此消息不得超过 32 KB（包括所有移入和移出字符）。 构建消息的应用程序必须将此考虑在内。

消息与 *IMS* 网桥之间的数据转换

数据转换由分布式排队工具（可以调用任何必要的出口）或由组内排队代理程序（不支持使用出口）在将消息放到具有为其存储类定义的 XCF 信息的目标队列时执行。 通过发布/预订将消息传递到队列时，不会发生数据转换。

所需的任何出口都必须在 CSQXLIB DD 语句所引用的数据集中可供分布式排队工具使用。 这意味着您可以使用 IBM MQ - IMS 网桥从任何 IBM MQ 平台将消息发送到 IMS 应用程序。

如果有转换错误，那么消息会未经转换放到队列；这最终导致它被 IBM MQ - IMS 网桥视为错误，因为网桥无法识别头格式。 如果发生转换错误，那么将会向 z/OS 控制台发送错误消息。

请参阅第 893 页的『编写数据转换出口』以获取有关一般数据转换的详细信息。

将消息发送到 IBM MQ - IMS 网桥

要确保正确执行转换，必须告知队列管理器消息的格式。

如果消息具有 MQIIH 结构，那么 MQMD 中的 *Format* 必须设置为内置格式 MQFMT_IMS，并且 MQIIH 中的 *Format* 必须设置为用于描述消息数据的格式的名称。 如果没有 MQIIH，请将 MQMD 中的 *Format* 设置为您的格式名称。

如果数据（除 LLZZ 以外）全部都是字符数据 (MQCHAR)，请将内置格式 MQFMT_IMS_VAR_STRING 用作格式名称（根据情况，在 MQIIH 或 MQMD 中）。 否则，使用自己的格式名称，在此情况下还必须为格式提供数据转换出口。 除数据本身以外，出口还必须处理消息中 LLZZ 的转换（但是不必处理消息开头的任何 MQIIH）。

如果应用程序使用 *MFSMapName*，那么可以改用具有 MQFMT_IMS 的消息，并在 MQIIH 的 MFSMapName 字段中定义传递到 IMS 事务的映射名称。

从 IBM MQ - IMS 网桥接收消息

如果在发送到 IMS 的原始消息上存在 MQIIH 结构，那么在回复消息上也存在该结构。

要确保正确转换回复，请执行以下操作：

- 如果在原始消息上具有 MQIIH 结构，请在原始消息的 MQIIH *ReplytoFormat* 字段中指定希望回复消息具有的格式。 该值放入回复消息的 MQIIH *Format* 字段中。 这在所有输出数据都为 LLZZ<character data> 形式时特别有用。
- 如果在原始消息上没有 MQIIH 结构，请在 IMS 应用程序的 ISRT 到 IOPCB 中指定希望回复消息具有的格式作为 MFS MOD 名称。

通过 *IBM MQ* 编写 *IMS* 事务程序

通过 IBM MQ 处理 IMS 事务所需的编码取决于 IMS 事务所需的消息格式及其可返回的响应范围。 但是，在应用程序处理 IMS 屏幕格式化信息时，应考虑以下几点。

在从 3270 屏幕启动 IMS 事务时，消息会通过 IMS 消息格式化服务传递。 这会从事务所见的数据流中除去所有终端依赖性。 通过 OTMA 启动事务时，不会涉及 MFS。 如果在 MFS 中实现应用程序逻辑，那么必须在新应用程序中重新创建。

在一些 IMS 事务中，最终用户应用程序可以修改某些 3270 屏幕行为，例如，突出显示已经输入了无效数据的字段。 对于需要由该程序修改的每个屏幕字段，通过向 IMS 消息添加双字节属性字段来传达此类信息。

因此，如果您正在编码应用程序以模拟 3270，那么在构建或接收消息时需要考虑这些字段。

您可能需要在程序中编写信息代码以进行处理：

- 按下哪个键（例如，Enter 和 PF1）
- 将消息传递到应用程序时光标所在位置
- 是否已通过 IMS 应用程序设置了属性字段
  - 高、正常或零强度
  - 颜色
  - IMS 是否希望在下次按下 Enter 键时返回该字段
- IMS 应用程序是否在任何字段中使用空字符 (X'3F')。

如果 IMS 消息只包含字符数据（除 LLZZ 数据段之外），且您正在使用 MQIIH 结构，请将 MQMD 格式设置为 MQFMT_IMS，并将 MQIIH 格式设置为 MQFMT_IMS_VAR_STRING。

如果 IMS 消息只包含字符数据（除 LLZZ 数据段之外），且您**没有**使用 MQIIH 结构，请将 MQMD 格式设置为 MQFMT_IMS_VAR_STRING，并确保在回复时 IMS 应用程序指定了 MODname

MQFMT_IMS_VAR_STRING。 如果出现问题（例如，用户无权使用事务）并且 IMS 发送了一条错误消息，那么此消息的 MODname 格式为 DFSMOx，其中 x 是 1 到 5 范围内的数字。 这将放入 MQMD.Format 中。

如果 IMS 消息包含二进制、打包或浮点数据（除 LLZZ 数据段之外），请编写您自己的数据转换例程的代码。 请参阅 *IMS/ESA Application Programming: Transaction Manager*，以获取有关 IMS 屏幕格式化的信息。

编写代码以通过 IBM MQ 处理 IMS 事务时，请考虑以下主题。

- 第 834 页的『编写 IBM MQ 应用程序以调用 IMS 对话式事务』
- 第 834 页的『编写包含 IMS 命令的程序』
- 第 834 页的『触发』

编写 IBM MQ 应用程序以调用 IMS 对话式事务

使用该信息作为编写 IBM MQ 应用程序以调用 IMS 对话式事务的注意事项指南。

在编写调用 IMS 对话的应用程序时，请考虑以下几点：

- 在应用程序消息中包含 MQIIH 结构。
- 将 MQIIH 中的 *CommitMode* 设置为 MQICM_SEND_THEN_COMMIT。
- 要调用新对话，请将 MQIIH 中的 *TranState* 设置为 MQITS_NOT_IN_CONVERSATION。
- 要调用对话的第二个步骤和后续步骤，请将 *TranState* 设置为 MQITS_IN_CONVERSATION，并将 *TranInstanceId* 设置为该对话的上一步中所返回的此字段的值。
- 如果您丢失从 IMS 发送的原始消息，那么 IMS 中没有简单的方法来查找 *TranInstanceId* 的值。
- 应用程序必须检查来自 IMS 的消息的 *TranState* ，以检查 IMS 事务是否已终止对话。
- 您可以使用 /EXIT 来结束对话。 您还必须将 *TranInstanceId* 加上引号，将 *TranState* 设置为 MQITS_IN_CONVERSATION，并使用正在其上执行该对话的 IBM MQ 队列。
- 您不能使用 /HOLD 和 /REL 来保持和发布对话。
- 如果重新启动 IMS ，那么将终止通过 IBM MQ - IMS 网桥调用的对话。

编写包含 IMS 命令的程序

应用程序可以构建格式为 LLZZ 命令的 IBM MQ 消息，而不是事务，其中 *command* 的格式为 /DIS TRAN PART 或 /DIS POOL ALL。

大多数 IMS 命令都可以采用这种方式发布，请参阅 *IMS V11 Communications and Connections* 以获取详细信息。 命令输出在 IBM MQ 回复消息中以文本形式接收，将发送至 3270 终端以显示。

OTMA 实现了特殊格式的 IMS 显示事务命令，这将返回输出的架构形式。 *IMS V11 Communications and Connections* 中定义了准确的格式。 要从 IBM MQ 消息调用此格式，请如同之前一样构建消息数据（例如 /DIS TRAN PART），并将 MQIIH 中的 TranState 字段设置为 MQITS_ARCHITECTED。 IMS 会处理该命令，并以架构形式返回该回复。 架构响应包含能够在文本格式输出中找到的所有信息，以及一部分附加信息：事务是定义为可恢复还是定义为不可恢复。

触发

IBM MQ - IMS 网桥不支持触发器消息。

如果定义了使用具有 XCF 参数的存储类的启动队列，那么在消息到达网桥时，会拒绝将这些消息放入该队列。

编写客户机过程应用程序

使用过程语言在 IBM MQ 上编写客户机应用程序时需要知道的内容。

可以在 IBM MQ 客户机环境中构建和运行应用程序。 必须构建应用程序并链接到所使用的 IBM MQ MQI client。 构建和链接应用程序的方式因所使用的平台和编程语言而异。 有关如何构建客户机应用程序的信息，请参阅第 839 页的『为 IBM MQ MQI clients 构建应用程序』。

您可以在完整 IBM MQ 环境和 IBM MQ MQI client 环境中运行 IBM MQ 应用程序，而无需更改代码，前提是满足特定条件。 有关在 IBM MQ 客户机环境中运行应用程序的更多信息，请参阅第 841 页的『在 IBM MQ MQI client 环境中运行应用程序』。

如果您使用消息队列接口 (MQI) 编写要在 IBM MQ MQI client 环境中运行的应用程序，那么在 MQI 调用期间会施加一些其他控制以确保 IBM MQ 应用程序处理过程不会中断。 有关这些控制的更多信息，请参阅第 835 页的『在客户机应用程序中使用 MQI』。

有关准备其他应用程类型并将其作为客户机应用程序运行的信息，请参阅以下主题：

- 第 851 页的『准备和运行 CICS 和 Tuxedo 应用程序』
- 第 40 页的『准备和运行 Microsoft Transaction Server 应用程序』
- 第 853 页的『准备和运行 IBM MQ JMS 应用程序』

相关概念

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

第 40 页的『IBM MQ 应用程序的设计注意事项』
当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

第 672 页的『编写用于排队的过程应用程序』
使用此信息来了解有关编写排队应用程序、连接和断开队列管理器、发布/预订以及打开和关闭对象的信息。

第 746 页的『编写发布/预订应用程序』
开始编写发布/预订 IBM MQ 应用程序。

第 908 页的『构建过程应用程序』
您可以使用若干过程语言之一编写 IBM MQ 应用程序，并在几个不同平台上运行该应用程序。

第 947 页的『处理过程程序错误』
本信息说明与应用程序 MQI 调用关联的错误，这些错误可能是应用程序进行调用时产生的，也可能是将其消息传递给最终目标时产生的。

相关任务

第 963 页的『使用 IBM MQ 样本过程程序』
这些样本程序采用过程语言编写，并演示了消息队列接口 (MQI) 的典型用法。 IBM MQ 在不同平台上编程。

在客户机应用程序中使用 MQI

该系列主题考虑了编写 IBM MQ 应用程序以在消息队列接口 (MQI) 客户机环境中运行和在完整的 IBM MQ 队列管理器环境中运行之间的差异。

在设计应用程序时，请考虑在 MQI 调用期间需要施加哪些控制，以确保 IBM MQ 应用程序处理过程不会中断。

必须先创建某些 IBM MQ 对象，然后才可运行使用 MQI 的应用程序。 有关更多信息，请参阅使用 MQI 的应用程序。

限制客户机应用程序中消息的大小

队列管理器具有最大消息长度，但您可从客户机应用程序传输的最大消息大小受通道定义的限制。

队列管理器的最大消息长度 (MaxMsgLength) 属性是可由该队列管理器处理的消息的最大长度。

Multi 在 多平台 上，您可以增加队列管理器的最大消息长度属性。 有关更多信息，请参阅 ALTER QMGR。

您可以通过使用 MQINQ 调用查找出队列管理器的 MaxMsgLength 值。

如果更改了 MaxMsgLength 属性，即使已没有队列，甚至消息的长度大于新的值，也不会进行检查。 在更改该属性之后，请重新启动应用程序和通道，以确保此更改生效。 因此，不可能生成超过队列管理器或队列的 MaxMsgLength 的任何新消息（除非允许队列管理器分段）。

通道定义中的最大消息长度限制您在客户机连接期间可以发送的消息的大小。 如果 IBM MQ 应用程序尝试使用消息大于该值的 MQPUT 调用或 MQGET 调用，那么会向应用程序返回错误代码。 通道定义的最大消息大小参数不影响当对客户机连接使用 MQCB 时可使用的最大消息大小。

相关概念

第 839 页的『使用 MQCONNX』
您可以使用 MQCONNX 调用来指定 MQCNO 结构中的通道定义 (MQCD) 结构。

相关参考

最大消息长度 (MAXMSGL)

ALTER CHANNEL

2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

选择客户机或服务器 *CCSID*

针对客户机使用本地编码字符集标识 (CCSID)。 队列管理器会执行必要的转换。 使用 MQCCSID 环境变量覆盖 CCSID。 如果您的应用程序执行多个 PUT，那么可在完成第一个 PUT 之后覆盖 MQMD 的 CCSID 和编码字段。

通过消息队列接口 (MQI) 从应用程序传递到客户机存根的数据必须位于本地 CCSID 中，针对 IBM MQ MQI client 进行编码。 如果已连接的队列管理器需要转换数据，那么由队列管理器上的客户机支持代码执行该转换。

在 IBM WebSphere MQ 7.0 和更高版本中，如果队列管理器无法执行该转换，那么 Java 客户机可以执行转换。 请参阅第 302 页的『IBM MQ classes for Java 客户机连接』。

客户机代码假设客户机中通过 MQI 的字符数据位于为该工作站配置的 CCSID 中。 如果该 CCSID 是不受支持的 CCSID 或不是必需的 CCSID，那么可以通过使用以下命令之一，将其覆盖为 MQCCSID 环境变量：

- Windows

```
SET MQCCSID=850
```

- UNIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

如果在概要文件中设置该参数，那么假设所有 MQI 数据都位于代码页 850 中。

注: 关于代码页 850 的假设不适用于消息中的应用程序数据。

如果您的应用程序正在执行多个 PUT，其中包含消息描述符之后的 IBM MQ 头，请注意，在完成第一个 PUT 之后，会覆盖 MQMD 的 CCSID 和编码字段。

在第一个 PUT 之后，这些字段会包含连接队列管理器用于转换 IBM MQ 头的值。 确保您的应用程序将这些值重置为其所需的值。

在客户机应用程序中使用 *MQINQ*

使用 MQINQ 查询到的某些值由客户机代码进行了修改。

CCSID

将设置为客户机 CCSID，而不是队列管理器的 CCSID。

MaxMsgLength
当它受限于通道定义时会减少。 它将是以下两个值中的较小值：

- 在队列定义中定义的值，或
- 在通道定义中定义的值。

有关更多信息，请参阅 MQINQ。

在客户机应用程序中使用同步点协调

运行于基本客户机上的应用程序可以发出 MQCMIT 和 MQBACK，但同步点控制的范围局限于 MQI 资源。 您可以将外部事务管理器与扩展事务客户机配合使用。

在 IBM MQ 中，队列管理器的角色之一即是在应用程序中充当同步点控制角色。 如果应用程序在 IBM MQ 基本客户机上运行，那么它可以发出 MQCMIT 和 MQBACK，但同步点控制的范围局限于 MQI 资源。 IBM MQ 动词 MQBEGIN 在基本客户机环境中无效。

在服务器完全队列管理器环境中运行的应用程序可以通过一个事务监视器协调多个资源（例如数据库）。 在服务器上，您可以使用 IBM MQ 产品随附的事务监视器或其他事务监视器，例如 CICS。 您不能对基本客户机应用程序使用事务监视器。

您可以对 IBM MQ 扩展事务客户机使用外部事务管理器。 请参阅 什么是扩展事务客户机? 以获取详细信息。

在客户机应用程序中使用预读

可以在客户机上使用预读，以允许将非持久消息发送到客户机，而不必使客户机应用程序请求这些消息。

当客户机需要从服务器获取消息时，它会将请求发送到该服务器。 它为要使用的每条消息单独发送一个请求。 要通过避免必须发送这些请求消息来提高使用非持久消息的客户机的性能，可以将客户机配置为使用预读。 预读允许将消息发送到客户机，而不必使应用程序请求这些消息。

在通过客户机应用程序使用非持久消息时，使用预读可提高性能。 此性能提升适用于 MQI 和 JMS 应用程序。 在使用非持久消息时，使用 MQGET 或异步消费的客户机应用程序会从性能提高中获益。

在通过 MQOO_READ_AHEAD 调用 MQOPEN 时，IBM MQ 客户机仅在满足某些条件时才会启用预读。 这些条件包括：

- 客户机和远程队列管理器都必须处于 IBM WebSphere MQ 7.0 或更高版本。
- 必须针对线程化的 IBM MQ MQI 客户机库编译和链接客户机应用程序。
- 客户机通道必须使用的是 TCP/IP 协议
- 该通道必须在客户机和服务器通道定义中具有非零 SharingConversations (SHARECNV) 设置。

在启用预读时，会将消息发送到客户机上称为预读缓冲区的内存缓冲区。 在预读已启用的情况下，客户机将为它打开的每个队列都提供一个预读缓冲区。 预读缓冲区中的消息为非持久消息。 客户机会定期为服务器更新有关它已使用的数据量的信息。

并非所有客户机应用程序设计都适合使用预读，这是因为并非所有选项都支持使用。 当预读已启用时，有些选项被要求在 MQGET 调用之间保持一致。 如果客户机在 MQGET 调用之间改变其选择标准，那么存储在预读缓冲区中的消息将滞留在客户机预读缓冲区中。 有关更多信息，请参阅第 728 页的『提高非持久消息的性能』

预读配置由三个属性控制，即 MaximumSize、PurgeTime 和 UpdatePercentage，IBM MQ 客户机配置文件的 MessageBuffer 节中指定了这些属性。

在客户机应用程序中使用异步放入方法

使用异步放入方法时，应用程序可以将消息放入队列中而不必等待队列管理器的响应。 您可以使用此方法在某些情况下提高消息传递性能。

通常，当应用程序使用 MQPUT 或 MQPUT1 将一条或多条消息放入队列中时，该应用程序必须等待队列管理器确认它已处理该 MQI 请求。 通过改为选择异步放入消息，可以提高消息传递性能，特别是对于使用客户机绑定的应用程序以及将大量短消息放入队列中的应用程序。 应用程序异步放置消息时，队列管理器不会返回每个调用的成功或失败状态，但可以改为定期检查错误。

要将消息异步放入队列中，请使用 MQPMO 结构的 *Options* 字段中的 MQPMO_ASYNC_RESPONSE 选项。

如果消息不符合异步放入的条件，那么它会被同步放入队列中。

当针对 MQPUT 或 MQPUT1 请求异步放入响应时，CompCode 和 Reason 为 MQCC_OK 和 MQRC_NONE 不一定意味着消息已被成功放入队列中。 虽然可能不会立即返回每个 MQPUT 或 MQPUT1 调用的成功或失败情况，但在异步调用下发生的第一个错误可稍后通过调用 MQSTAT 来确定。

有关 MQPMO_ASYNC_RESPONSE 的更多详细信息，请参阅 MQPMO 选项。

“异步放入方法”样本程序演示了一些可用的功能。 有关该程序的功能和设计以及运行方法的详细信息，请参阅第 979 页的『Asynchronous Put 样本程序』。

在客户机应用程序中使用共享对话

在允许共享对话的环境中，对话可以共享 MQI 通道实例。

共享对话由两个字段（都称为 SharingConversations）控制，其中一个是通道定义 (MQCD) 结构的一部分，另一个是通道退出参数 (MQCXP) 结构的一部分。 MQCD 中的 SharingConversations 字段是整数值，用于确定可以共享通道关联的通道实例的最大对话数。 MQCXP 中的 SharingConversations 字段是布尔值，指示当前是否共享通道实例。

在不允许共享对话的环境中，指定相同 MQCD 的新客户机连接不会共享通道实例。

当满足以下条件时，新客户机应用程序连接将共享通道实例：

- 通道实例的客户机连接端和服务器连接端均配置了共享对话，并且这些值不会被通道退出覆盖。
- 在先确定现有通道实例的情况下，客户机连接 MQCD 值（在客户机 MQCONNX 调用或客户机通道定义表 (CCDT) 中提供）与客户机 MQCONNX 调用或 CCDT 中提供的客户机连接 MQCD 值完全匹配。 请注意，原始 MQCD 之后可能已被退出或通道协商变更，但这一匹配是在进行这些变更之前针对提供给客户机系统的值进行的。
- 不超过服务器端的共享对话限制。

如果新客户机应用程序连接符合与其他对话共享通道实例的条件，那么应在该对话上调用任何退出之前做出此决定。 此类对话上的退出可能会变更它与其他对话共享通道实例这一事实。 如果没有与新通道定义匹配的现有通道实例，那么将连接新通道实例。

仅针对通道实例上的首个对话进行通道协商；通道实例的协商值将在该阶段确定，且在后续对话启动时无法变更。 也只针对首个对话进行 TLS 认证。

如果在通道实例的客户机连接端或服务器连接端的套接字上，首个对话的任何安全、发送或接收退出的初始化期间变更了 MQCD SharingConversations 值，那么将使用该字段在所有这些退出初始化之后获得的新值来确定通道实例的共享对话值（最低值优先）。

如果共享对话的协商值为 0，那么永不共享通道实例。 同样，将此字段设置为 0 的其他退出程序也在自己的通道实例上运行。

如果共享对话的协商值大于 0，那么对于后续退出调用，MQCXP SharingConversations 将设置为 TRUE，表示必须同时进入当前退出程序和该通道实例上的其他退出程序。

在编写通道退出程序时，请考虑它是否将在可能涉及共享对话的通道实例上运行。 如果通道实例可能涉及共享对话，请考虑更改 MQCD 字段对该通道退出的其他实例的影响；所有 MQCD 字段在所有共享对话中具有通用值。 在确定通道实例之后，如果退出程序尝试变更 MQCD 字段，那么可能会遇到问题，因为该通道实例上运行的其他退出程序实例可能同时在尝试变更相同的字段。 如果您的退出程序可能会发生此情况，那么必须在退出代码中对 MQCD 的访问进行串行化。

如果使用的是已定义为共享对话的通道，但不想在某个特定通道实例上发生共享，那么在该通道实例上的首个对话上初始化通道退出时，请将 MQCD SharingConversations 值设置为 1 或 0。 请参阅 SharingConversations，以获取对 SharingConversations 值的说明。

示例

启用共享对话。

您使用的是指定了退出程序的客户机连接通道定义。

该通道首次启动时，退出程序在初始化时变更了某些 MQCD 参数。 这些变更按通道生效，因此通道运行所用的定义现在与原始提供的定义有所不同。 MQCXP SharingConversations 参数设置为 TRUE。

应用程序下次使用此通道进行连接时，对话将在先前启动的通道实例上运行，因为它具有相同的原始通道定义。 应用程序第二次连接到的通道实例与第一次连接到的实例相同。 因此，它使用已被退出程序变更的定义。 当针对第二个对话初始化出口程序时，虽然可能变更了 MQCD 字段，但不会按通道生效。 这些相同的特征也适用于共享通道实例的任何后续对话。

使用 *MQCONNX*

您可以使用 MQCONNX 调用来指定 MQCNO 结构中的通道定义 (MQCD) 结构。

这允许调用的客户机应用程序在运行时指定客户机连接通道的定义。 有关更多信息，请参阅在 MQCONNX 调用上使用 MQCNO 结构。 当您使用 MQCONNX 时，服务器发出的调用取决于服务器级别和侦听器配置。

当您在客户机上使用 MQCONNX 时，忽略以下选项：

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

您可以使用的 MQCD 结构取决于您正在使用的 MQCD 版本号。 有关 MQCD 版本 (MQCD_VERSION) 的信息，请参阅 MQCD 版本。 例如，您可以使用 MQCD 结构将通道出口程序传递到服务器。 如果您使用的是 MQCD 版本 3 或更新版本，那么您可以使用该结构将一组退出传递给服务器。 通过为每个操作添加一个出口，而不是修改一个现有的出口，您可以使用该功能在相同的消息上执行多个操作，例如加密和压缩。 如果您没有在 MQCD 结构中指定一个数组，那么将检查单个出口字段。 有关通道出口程序的更多信息，请参阅第 875 页的『消息传递通道的通道出口程序』。

MQCONNX 上的共享连接句柄

您可以使用共享连接句柄在同一进程中的不同线程之间共享句柄。

当您指定一个共享连接句柄时，从 MQCONNX 调用返回的连接句柄可以在进程中任何线程上的后继 MQI 调用中传递。

注: 您可以在 IBM MQ MQI client 上使用共享连接句柄，以便连接到不支持共享连接句柄的服务器队列管理器。

有关更多信息，请参阅第 839 页的『使用 MQCONNX』。

为 IBM MQ MQI clients 构建应用程序

可在 IBM MQ MQI client 环境中构建和运行应用程序。 必须构建应用程序并链接到所使用的 IBM MQ MQI client。 构建和链接应用程序的方式因所使用的平台和编程语言而异。

如果应用程序将在客户机环境中运行，您可以用下表显示的语言来编写：

表 *137:* 在客户机环境中支持的编程语言

客户机平台	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX AIX	Yes	Yes	Yes			
IBM i IBM i	Yes		Yes		Yes	
Linux Linux	Yes	Yes	Yes			
Solaris Solaris	Yes	Yes	Yes			
Windows Windows	Yes	Yes	Yes			Yes

Multi 将 *C* 应用程序与 *IBM MQ MQI client* 代码链接

编写要在 IBM MQ MQI client 上运行的 IBM MQ 应用程序后，必须将其链接到 IBM MQ MQI client 代码。

您可以通过两种方式将应用程序链接到 IBM MQ MQI client 代码：

1. 可以直接将应用程序连接到队列管理器，在这种情况下，队列管理器必须与应用程序在同一机器上。

2. 链接到客户机库文件，这样您可以访问相同或不同机器上的队列管理器。

IBM MQ 为每个环境都提供了客户机库文件：

AIX **AIX**
用于非线程应用程序的 libmqic.a 库或用于线程应用程序的 libmqic_r.a 库。

Linux **Linux**
用于非线程应用程序的 libmqic.so 库或用于线程应用程序的 libmqic_r.so 库。

IBM i **IBM i**
使用非线程应用程序的 LIBMQIC 客户机服务程序，或使用线程应用程序的 LIBMQIC_R 服务程序绑定客户机应用程序。

Solaris **Solaris**
libmqic.so.

如果想要在只安装了 IBM MQ MQI client for Solaris 的机器上使用程序，那么必须重新编译程序以将它们与客户机库链接：

```
$ /opt/SUNWspro/bin/cc -o prog_name prog_name c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

参数必须以正确顺序输入，如同显示的那样。

Windows **Windows**
MQIC32.LIB.

ULW 将 *C++ 应用程序与 IBM MQ MQI client* 代码链接

您可以用 C++ 编写要在客户机上运行的应用程序。构建方法因环境而异。

有关如何链接 C++ 应用程序的信息，请参阅构建 IBM MQ C++ 程序。

有关使用 C++ 的所有方面的完整详细信息，请参阅使用 C++

Multi 将 *COBOL 应用程序与 IBM MQ MQI client* 代码链接

在编写了想要在 IBM MQ MQI client 上运行的 COBOL 应用程序之后，您必须将其与相应的库相链接。

IBM MQ 为每个环境都提供了客户机库文件：

AIX **AIX**
将非线程 COBOL 应用程序与库 libmqicb.a 相链接，或将线程 COBOL 应用程序与 libmqicb_r.a 相链接。

IBM i **IBM i**
使用非线程应用程序的 AMQCSTUB 服务程序或线程应用程序的 AMQCSTUB_R 服务程序绑定 COBOL 客户机应用程序。

Solaris **Solaris**
将非线程 COBOL 应用程序与库 libmqicb.so 相链接，或将线程 COBOL 应用程序与 libmqicb_r.so 相链接。

Windows **Windows**
将应用程序代码与用于 32 位 COBOL 的 MQICCBB 库相链接。 IBM MQ MQI client for Windows 不支持 16 位 COBOL。

Windows 将 *Visual Basic 应用程序与 IBM MQ MQI client* 代码相链接

您可以将 Microsoft Visual Basic 应用程序与 Windows 上的 IBM MQ MQI client 代码链接在一起。

从 IBM MQ 9.0 开始，不推荐使用对 Microsoft Visual Basic 6.0 的 IBM MQ 支持。 IBM MQ classes for .NET 是建议使用的替代技术。 有关更多信息，请参阅开发 .NET 应用程序。

将 Visual Basic 应用程序与以下包含文件相链接：

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
PCF 命令

CMQXB.bas
通道

在 Visual Basic 编译器中为客户机设置 `mqtype=2` ，以确保正确自动选择客户机 dll:

MQIC32.dll
Windows 7、Windows 8、Windows 2008 和 Windows 2012

相关概念
第 959 页的『在 Visual Basic 中编码』
在 Microsoft Visual Basic 中对 IBM MQ 程序进行编码时要考虑的信息。 Visual Basic 仅在 Windows 上受支持。

第 932 页的『在 Windows 中准备 Visual Basic 程序』
在 Windows 上使用 Microsoft Visual Basic 程序时要考虑的信息。

在 IBM MQ MQI client 环境中运行应用程序

您可以在完整 IBM MQ 环境和 IBM MQ MQI client 环境中运行 IBM MQ 应用程序，而无需更改代码，前提是满足特定条件。

这些条件如下：

- 该应用程序不需要同时连接到多个队列管理器。
- 在 `MQCONN` 或 `MQCONNX` 调用中，该队列管理器的名称前缀并非是星号 (*)。
- 应用程序不需要使用在 IBM MQ MQI client 上运行哪些应用程序？ 中列出的任何异常

注: 在链接编辑时使用的库确定您的应用程序必须在其中运行的环境。

在 IBM MQ MQI client 环境中工作时，请记住：

- 在 IBM MQ MQI client 环境中运行的每个应用程序都有自己与服务器的连接。 每次应用程序发出 `MQCONN` 或 `MQCONNX` 调用时，它都会与服务器建立一个连接。
- 一个应用程序同步发送和获取消息。 这表示从客户机上发出调用到通过网络返回完成代码和原因码之间的等待时间。
- 所有数据转换均由服务器完成，但请同时参阅 MQCCSID，获取有关覆盖机器的已配置 CCSID 的信息。

将 *IBM MQ MQI client* 应用程序连接到队列管理器

在 IBM MQ MQI client 环境中运行的应用程序可以通过各种方式连接到队列管理器。 您可以使用环境变量、MQCNO 结构或客户机定义表。

当在 IBM MQ 客户机环境中运行的应用程序发出 MQCONN 或 MQCONNX 调用时，客户机会识别其如何进行连接。 在通过 IBM MQ 客户机上的应用程序发出 MQCONNX 调用时，MQI 客户机库会按照以下顺序搜索该客户机通道信息：

1. 使用 MQCNO 结构 (如果提供) 的 `ClientConnOffset` 或 `ClientConnPtr` 字段的内容。 这些字段可以识别用作客户机连接通道定义的通道定义结构 (MQCD)。 可使用预连接出口覆盖连接详细信息。 有关更多信息，请参阅第 902 页的『使用存储库的预连接出口引用连接定义』。
2. 如果设置了 MQSERVER 环境变量，将使用它定义的通道。
3. 如果定义了 `mqclient.ini` 文件并且此文件包含 ServerConnectionParms，那么将使用其定义的通道。有关更多信息，请参阅使用配置文件配置客户机和客户机配置文件的 CHANNELS 节。
4. 如果设置了 MQCHLLIB 和 MQCHLTAB 环境变量，将使用它们指向的客户机通道定义表。 或者，从 IBM MQ 9.0 起，MQCCDTURL 环境变量提供了用于设置 MQCHLLIB 和 MQCHLTAB 环境变量组合的同等功

能。 如果设置了 MQCCDTURL，将使用它指向的客户机通道定义表。 有关更多信息，请参阅对客户机通道定义表的 Web 可寻址访问。

5. 如果定义了 `mqclient.ini` 文件并且此文件包含 ChannelDefinitionDirectory 和 ChannelDefinitionFile 属性，那么可使用这些属性来查找客户机通道定义表。 有关更多信息，请参阅使用配置文件配置客户机和客户机配置文件的 CHANNELS 节。
6. 最后，如果未设置环境变量，客户机会通过从 `mqs.ini` 文件中的 `DefaultPrefix` 确定的路径和名称来搜索客户机通道定义表。 如果对客户机通道定义表的搜索失败，那么客户机将使用以下路径：
   - Linux UNIX 在 UNIX and Linux 上：`/var/mqm/AMQCLCHL.TAB`
   - Windows 在 Windows 上：`C:\Program Files\IBM\MQ\amqclchl.tab`
   - IBM i 在 IBM i 上：`/QIBM/UserData/mqm/@ipcc`
   - MQ Appliance 在 IBM MQ Appliance 上：*QMname*`_AMQCLCHL.TAB`. 它们出现在 `mqbackup:// URI` 下。

先前列表中所描述的第一个选项（使用 MQCNO 的 `ClientConnOffset` 或 `ClientConnPtr` 字段）仅受 MQCONNX 调用支持。 如果应用程序使用的是 MQCONN 而不是 MQCONNX，将按列表中所示顺序在其余五种方式中搜索通道信息。 如果客户机无法找到通道信息，那么 MQCONN 或 MQCONNX 调用会失败。

通道名称（用于客户机连接）必须匹配在服务器上定义的服务器连接通道名称，以使 MQCONN 或 MQCONNX 调用成功。

相关概念

客户机通道定义表

对客户机通道定义表的 Web 可寻址访问

相关任务

配置服务器与客户机之间的连接

相关参考

MQSERVER

MQCHLLIB

MQCHLTAB

MQCCDTURL

MQCNO - 连接选项

使用环境变量将客户机应用程序连接到队列管理器

可以通过环境变量将客户机通道信息提供给在客户机环境中运行的应用程序。

在 IBM MQ MQI client 环境中运行的应用程序可以使用以下环境变量连接到队列管理器：

MQSERVER

MQSERVER 环境变量用于定义最小通道。 MQSERVER 用于指定 IBM MQ 服务器位置以及要使用的通信方法。

MQCHLLIB

MQCHLLIB 环境变量用于指定包含客户机通道定义表 (CCDT) 的文件的目录路径。 此文件在服务器上创建，但可复制到 IBM MQ MQI client 工作站。

MQCHLTAB

MQCHLTAB 环境变量用于指定包含客户机通道定义表 (CCDT) 的文件的名称。

从 IBM MQ 9.0 开始，MQCCDTURL 环境变量将提供相当于设置 MQCHLLIB 和 MQCHLTAB 环境变量组合的功能。 MQCCDTURL 允许您提供文件、FTP 或 HTTP URL，作为可从中获取客户机通道定义表的单一值。 有关更多信息，请参阅对客户机通道定义表的 Web 可寻址访问。

使用 *MQCNO* 结构将客户机应用程序连接到队列管理器

您可以在一个通道定义结构 (MQCD) 中指定通道的定义，该结构是通过使用 MQCONNX 调用的 MQCNO 结构提供的。

有关更多信息，请参阅在 MQCONNX 调用上使用 MQCNO 结构。

使用客户机通道定义表将客户机应用程序连接到队列管理器
如果使用 MQSC DEFINE CHANNEL 命令，那么您提供的详细信息将放置在客户机通道定义表 (ccdt) 中。 MQCONN 或 MQCONNX 调用的 **QMgrName** 参数的内容确定客户机连接到哪个队列管理器。

该文件由客户机访问以确定应用程序将使用的通道。 对于存在多个合适的通道定义的情况，通道被选中的可能性受客户机通道权重 (CLNTWGHT) 和连接亲缘关系 (AFFINITY) 通道属性的影响。

使用客户机自动重新连接
通过配置多个组件，可以使客户机应用程序自动进行重新连接，而不必编写任何其他代码。

自动客户机重新连接是内联的。 将在客户机应用程序中的任意点自动复原连接，并且将全部复原用于打开对象的句柄。

相比之下，手动重新连接需要客户机应用程序使用 MQCONN 或 MQCONNX 重新创建连接，并重新打开对象。 自动客户机重新连接适合许多但并非所有客户机应用程序。

有关更多信息，请参阅客户机自动重新连接。

客户机通道定义表的角色
客户机通道定义表 (CCDT) 包含客户机连接通道的定义。 如果您的客户机应用程序可能需要连接到许多备选队列管理器，那么这将非常有用。

当您定义队列管理器时，将创建客户机通道定义表。 多个 IBM MQ 客户机可使用同一个文件。

客户机应用程序可通过多种方法使用 CCDT。 可以将 CCDT 复制到客户端计算机。 您可以将 CCDT 复制到多个客户机共享的位置。 您可以将 CCDT 作为共享文件供客户机访问，而它仍然位于服务器上。

从 IBM MQ 9.0 开始，CCDT 可以托管在可通过 URI 访问的中央位置，无需单独更新每个已部署客户机的 CCDT。

相关概念
客户机通道定义表
对客户机通道定义表的 Web 可寻址访问
相关任务
访问客户机连接通道定义

CCDT 中的队列管理器组
您可以将客户机通道定义表 (CCDT) 中的一组连接定义为队列管理器组。 您可以将应用程序连接到一个队列管理器上，该队列管理器是队列管理器组的一部分。 可以通过对 MQCONN 或 MQCONNX 调用上的队列管理器名称加上星号前缀来完成此操作。

您可能会选择定义到多个服务器的连接，因为：

- 您想要将客户机连接至一组队列管理器中任何一个正在运行的队列管理器，以便提高可用性。
- 您想要将客户机重新连接到上次成功连接的同一队列管理器，但如果连接失败，就连接到其他队列管理器。
- 如果连接失败，您想要再次在客户机程序中发出 MQCONN 来重试到其他队列管理器的客户机连接。
- 如果连接失败，您想要在不编写任何客户机代码的情况下，使客户机自动重新连接至另一队列管理器。
- 如果备用实例接管了工作，您想要在不编写任何客户机代码的情况下，使客户机自动重新连接到多实例队列管理器的其他实例。
- 您想要在许多队列管理器之间均衡客户机连接，使某些队列管理器连接的客户机多于其他队列管理器。
- 您想要逐渐将多个客户机连接的重新连接分布在多个队列管理器上，以防大量连接导致出现故障。
- 您想要能够在不更改任何客户机应用程序代码的情况下移动队列管理器。
- 您想要编写不需要知道队列管理器名称的客户机应用程序。

连接到不同队列管理器并不总是合适的。 例如， WebSphere Application Server 中的扩展事务客户机或 Java 客户机可能需要连接到可预测的队列管理器实例。 IBM MQ classes for Java 不支持客户机自动重新连接。

队列管理器组是在客户机通道定义表 (CCDT) 中定义的连接集合。 该集合由其成员定义，这些成员在其通道定义中具有相同的 **QMNAME** 属性值。

第 844 页的图 102 是客户机连接表的图形表示法，显示了三个队列管理器组：CCDT 中编写为 **QMNAME**（QM1）和 **QMNAME**（QMGrp1）的两个命名队列管理器组，以及一个编写为 **QMNAME**（' '）的空白或缺省组。

1. 队列管理器组 QM1 具有三个客户机连接通道，可将其连接到队列管理器 QM1 和 QM2。 QM1 可能是位于两个不同服务器上的多实例队列管理器。
2. 缺省队列管理器组具有六个客户机连接通道，将其连接至所有队列管理器。
3. QMGrp1 具有与两个队列管理器 QM4 和 QM5 的客户机连接通道。

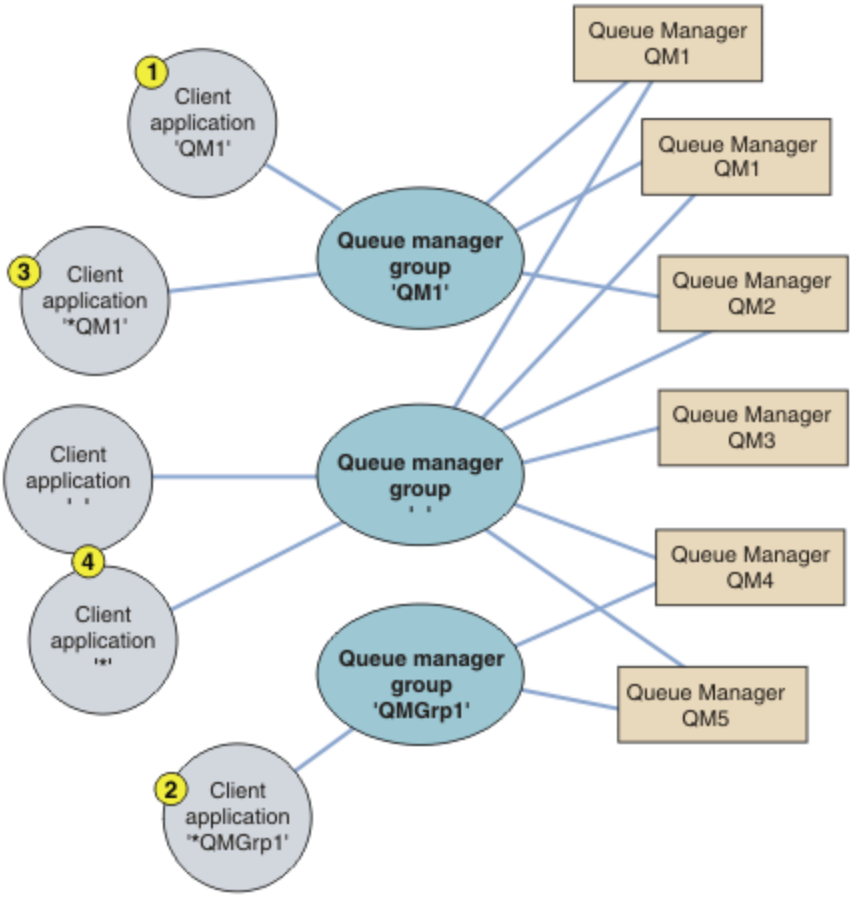


图 *102:* 队列管理器组

利用第 844 页的图 102 中已编号的客户机应用程序，描述了四个使用该客户机连接表的示例。

1. 在第一个示例中，客户机应用程序将队列管理器名称 QM1 作为 **QmgrName** 参数传递给 MQCONN 或 MQCONNX MQI 调用。 IBM MQ 客户机代码会选择匹配的队列管理器组 QM1。 该组包含三个连接通道，IBM MQ MQI client 尝试使用这些通道中的每个通道依次连接到 QM1 ，直到找到连接到名为 QM1 的正在运行的队列管理器的连接的 IBM MQ 侦听器为止。

   连接尝试的顺序取决于客户机连接 AFFINITY 属性值和客户机通道权重。 在这些约束范围内，在三个可能的连接上以及随着时间的推移，连接尝试顺序是随机的，以便分散进行连接的负载。

   在与 QM1 的正在运行的实例建立连接后，由客户机应用程序发出的 MQCONN 或 MQCONNX 调用成功执行。
2. 在第二个示例中，客户机应用程序将带有星号前缀的队列管理器名称 *QMGrp1 作为 **QmgrName** 参数传递给 MQCONN 或 MQCONNX MQI 调用。 IBM MQ 客户机会选择匹配的队列管理器组 QMGrp1。 该组包含两个客户机连接通道，IBM MQ MQI client 会依次使用每个通道尝试连接到任意队列管理器。 在此示例中，IBM MQ MQI client 需要成功进行连接；它连接的队列管理器的名称则无关紧要。

   适用于连接尝试顺序的规则与之前相同。 唯一不同之处在于，通过对队列管理器名称添加星号前缀，客户机指示该队列管理器的名称是不相关的。

当与通过 QMGrp1 队列管理器组中的通道连接的任何队列管理器的运行实例建立连接时，由客户机应用程序发出的 MQCONN 或 MQCONNX 调用成功执行。

3. 第三个示例基本上与第二个示例相同，原因是 **QmgrName** 参数以星号作为前缀 *QM1。 该示例说明，您无法通过单独检查一个通道定义中的 QMNAME 属性来决定客户机通道连接要连接到的队列管理器。 通道定义的 **QMNAME** 属性为 QM1 这一事实并不足以要求与名为 QM1 的队列管理器建立连接。 如果客户机应用程序的 **QmgrName** 参数带有星号前缀，那么任何队列管理器都有可能成为连接目标。

   在这种情况下，当与 QM1 或 QM2 的运行实例建立连接时，客户机应用程序发出的 MQCONN 或 MQCONNX 调用成功执行。

4. 第四个示例举例说明缺省组的使用。 在这种情况下，客户机应用程序会将星号 '*' 或空白 ' ' 作为 **QmgrName** 参数传递给其 MQCONN 或 MQCONNX MQI 调用。 按照客户机通道定义中的惯例，空白 **QMNAME** 属性表示缺省队列管理器组，空白或星号 **QmgrName** 参数与空白 **QMNAME** 属性匹配。

   在此示例中，缺省队列管理器组拥有与所有队列管理器的客户机通道连接。 通过选择缺省队列管理器组，应用程序可能已连接至组中的任何队列管理器。

   当与任何队列管理器的正在运行的实例建立连接后，由客户机应用程序发出的 MQCONN 或 MQCONNX 调用成功执行。

   注: 虽然应用程序使用空白 **QmgrName** 参数连接到缺省队列管理器组或缺省队列管理器，但缺省组不同于缺省队列管理器。 缺省队列管理器组的概念仅与客户机应用程序相关，而缺省队列管理器与服务器应用程序相关。

只在一个队列管理器上定义您的客户机连接通道，包含那些连接至第二或第三个队列管理器的通道。 不要在两个队列管理器上定义客户机连接通道，然后合并这两个客户机通道定义表。 只有一个客户机通道定义表可被客户机访问。

示例

在主题开始处再次查看使用队列管理器组的原因的列表。 如何使用队列管理器组来提供这些功能?

连接至队列管理器集合中的任何一个队列管理器。
定义一个队列管理器组，具有到集合中所有队列管理器的连接，并使用以星号为前缀的 **QmgrName** 参数连接该组。

重新连接至相同的队列管理器，但如果上次连接至的队列管理器不可用，那么连接至另一个队列管理器。
像之前一样定义队列管理器组，但在每个客户机通道定义上设置属性 **AFFINITY** (PREFERRED)。

如果连接失败，请重试连接到另一个队列管理器。
连接至队列管理器组，并在连接断开或队列管理器失败的情况下，重新发出 MQCONN 或 MQCONNX MQI 调用。

如果连接失败，那么自动重新连接至另一个队列管理器。
使用 MQCONNX **MQCNO** 选项 MQCNO_RECONNECT 连接至队列管理器组。

自动重新连接至多实例队列管理器的其他实例。
执行与上述示例相同的操作。 在这种情况下，如果想要限制队列管理器组连接到特定多实例队列管理器的实例，请定义仅连接到多实例队列管理器实例的组。

您也可以让客户机应用程序发出 MQCONN 或 MQCONNX MQI 调用，但 **QmgrName** 参数不添加星号前缀。 这样客户机应用程序只能连接至指定的队列管理器。 最后，您可以将 **MQCNO** 选项设置为 MQCNO_RECONNECT_Q_MGR。 此选项接受重新连接至先前连接的同一队列管理器。 您也可以使用此值来限制重新连接至正常队列管理器的同一实例。

在队列管理器之间均衡客户机连接，使某些队列管理器连接的客户机多于其他队列管理器。
定义队列管理器组，并在每个客户机通道定义上设置 **CLNTWGHT** 属性，以便不均匀地分发连接。

在连接或队列管理器出现故障之后，不均匀地分布客户机重新连接负载，并逐步分散负载。
执行与上述示例相同的操作。 IBM MQ MQI client 使队列管理器之间的重新连接随机化，并随时间推移分布重新连接。

在不更改任何客户机代码的情况下移动队列管理器。
CCDT 将客户机应用程序与队列管理器位置隔离。 CCDT 是可以在客户机定义、从共享位置读取或从 Web 服务器访存的数据文件。 有关更多信息，请参阅客户机通道定义表。

编写不知道队列管理器名称的客户机应用程序。
使用队列管理器组名，并为与组织中的客户机应用程序相关的队列管理器组名建立命名约定，然后反映解决方案的体系结构，而不是队列管理器的命名。

z/OS 连接到队列共享组

您可以将自己的应用程序连接到一个队列管理器，该队列管理器是队列共享组的一部分。 可以使用队列共享组名而不是 MQCONN 或 MQCONNX 调用上的队列管理器名称来完成此操作。

队列共享组具有最多为四个字符的名称。 该名称在网络中必须是唯一的，并且必须与所有队列管理器名称不同。

该客户机通道定义应该使用队列共享组通用接口来连接该组中可用的队列管理器。 有关更多信息，请参阅将客户机连接到队列共享组。 将进行检查，确保侦听器所连接的队列管理器是队列共享组的一个成员。

有关共享队列的更多信息，请参阅共享队列和队列共享组。

通道权重和亲缘关系的示例

这些示例证明，在使用非零 `ClientChannelWeights` 时，如何选中客户机连接通道。

`ClientChannelWeight` 和 `ConnectionAffinity` 通道属性控制当多个适当通道可供一个连接使用时如何选择客户机连接通道。 这些通道将配置为连接至不同队列管理器，以提供较高的可用性和/或工作负载均衡功能。可能导致连接到多个队列管理器之一的 MQCONN 调用必须使用星号作为队列管理器名称的前缀，如下所述：MQCONN 调用的示例：示例 1。 队列管理器名称包含星号 (*)。

适用于连接的候选通道是 QMNAME 属性与 MQCONN 调用中指定的队列管理器名称匹配的通道。如果某个连接的所有适用通道的 `ClientChannelWeight` 为 0（缺省值），那么将按字母顺序选择这些通道，如以下示例中所示：MQCONN 调用的示例：示例 1。 队列管理器名称包含星号 (*)。

下面的示例说明在使用非零 `ClientChannelWeight` 值时发生的情况。 请注意，因为此功能涉及伪随机通道选择，所以这些示例显示可能发生而不是一定会发生的一序列操作。

示例 *1.* 在 *ConnectionAffinity* 设置为 *PREFERRED* 时选择通道

该示例说明了在将 ConnectionAffinity 设置为 PREFERRED 时，IBM MQ MQI client 如何从 CCDT 选择通道。

在此示例中，大量客户端机器使用由队列管理器提供的客户机通道定义表 (CCDT)。 该 CCDT 包含具有以下属性的客户机连接通道（通过使用 DEFINE CHANNEL 命令语法来显示）：

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

应用程序发出 MQCONN(*CORE)

通道 A 不是此连接的候选通道，因为 QMNAME 属性不匹配。 通道 B、C 和 D 标识为候选通道，并以基于其权重的优先顺序放置。 在此示例中，顺序可能是 C、B 和 D。 客户机将尝试连接到 core2.ops.company.example 处的队列管理器。 未对该地址处的队列管理器名称进行检查，因为 MQCONN 调用在队列管理器名称中包含了星号。

要注意的一点是，在设置了 AFFINITY(PREFERRED) 的情况下，每次此特定客户机进行连接时，它都将以同一初始优先顺序放置这些通道。 甚至当连接是来自不同进程或是在不同时间进行时，此规律都适用。

在此示例中，无法与 core.2.ops.company.example 处的队列管理器进行连接。 客户机尝试连接到 core1.ops.company.example，因为通道 B 是优先顺序中的下一个。 此外，通道 C 降级成为尾选项。

同一应用程序发出第二个 MQCONN(*CORE) 调用。 先前的连接使通道 C 降级，因此最首选的通道现在是通道 B。 将与 core1.ops.company.example 建立此连接。

共享相同客户机通道定义表的第二台机器可能会以不同的初始优先顺序放置通道。 例如：D、B、C。 在正常情况下，如果所有通道都正常工作，那么此机器上的应用程序将连接到 core3.ops.company.example，而

第一台机器上的应用程序将连接到 core2.ops.company.example。 在允许各个客户机与同一队列管理器（如果可用）进行连接的同时，这允许在多个队列管理器之间平衡大量客户机的工作负载。

示例 *2*： 在 *ConnectionAffinity* 设置为 *NONE* 时选择通道
该示例说明了在将 ConnectionAffinity 设置为 NONE 时，IBM MQ MQI client 如何从 CCDT 选择通道。

在此示例中，大量客户机使用由队列管理器提供的客户机通道定义表 (CCDT)。 该 CCDT 包含具有以下属性的客户机连接通道（通过使用 DEFINE CHANNEL 命令语法来显示）：

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

该应用程序发出 MQCONN(*CORE)。 与上一个示例一样，因为 QMNAME 不匹配，所以未考虑通道 A。 基于通道 B、C 或 D 的权重选择通道，概率为 50%、30% 或 20%。 在此示例中，可能选择通道 B。 没有创建任何持久的优先顺序。

发出第二个 MQCONN(*CORE) 调用。 会再次选中这三个适用通道之一，选择概率相同。 在此示例中选择通道 C。 但是，core2.ops.company.example 未响应，所以在剩余的候选通道之间再次选择。 选中通道 B，并将应用程序连接到 core1.ops.company.example。

在设置了 AFFINITY(NONE) 的情况下，每个 MQCONN 调用都与任何其他调用无关。 因此，当此示例应用程序发出第三个 MQCONN(*CORE) 调用时，它可能再一次尝试通过中断的通道 C 进行连接，然后才选择 B 或 D 中之一。

MQCONN 调用的示例
使用 MQCONN 连接到特定队列管理器或一组队列管理器之一的示例。

在以下每个示例中，网络相同，且定义了一个从同一 IBM MQ MQI client 到两个服务器的连接。 （在这些示例中，可以使用 MQCONNX 调用代替 MQCONN 调用。）

两个队列管理器运行在服务器上，一个名叫 SALE，另一个名叫 SALE_BACKUP。

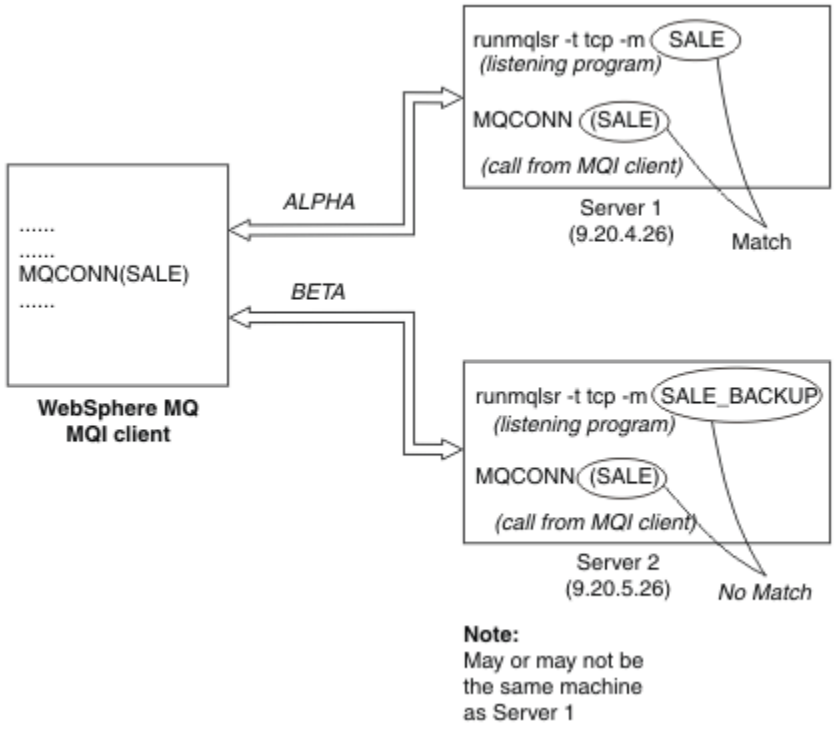


图 *103: MQCONN* 示例

在这些示例中，通道定义是：

SALE 定义：

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

SALE_BACKUP 定义：

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

客户机通道定义概述如下所示：

名称	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

MQCONN 示例演示什么
这些示例演示使用多个队列管理器作为备份系统。

假设至服务器 1 的通信链路暂时已断开。 演示使用多个队列管理器作为备份系统。

通过应用以下规则，每个示例涉及一个不同的 MQCONN 调用，并且对在所显示的特定示例中发生了什么作出说明：

1. 针对与 MQCONN 调用中的给定值相对应的队列管理器名称（QMNAME 字段），按通道名称字母顺序对客户机通道定义表 (CCDT) 进行扫描。
2. 如果查找到匹配，那么使用该通道定义。
3. 尝试启动通往由连接名称 (CONNAME) 标识的机器的通道。 如果成功，应用程序继续运行。 它要求：
   - 在服务器上运行一个侦听器。
   - 侦听器所要连接的与客户机要连接的（如果指定）是同一队列管理器。
4. 如果尝试启动通道失败，并且在客户机通道定义表中存在多个条目（在该示例中有两个条目），那么搜索文件用于进一步匹配。 如果查找到匹配，按步骤 1 继续处理。
5. 如果没有查找到匹配，或在客户机通道定义表中没有更多的条目，并且通道启动失败，那么应用程序就无法连接。 在 MQCONN 调用中返回相应的原因码和完成代码。 应用程序可以基于返回的原因码和完成代码进行操作。

示例 *1*：队列管理器名称包含星号 *(*)*
在此示例中，应用程序不关心它连接至哪个队列管理器。 应用程序针对包含星号的队列管理器名称发出 MQCONN 调用。 选择了合适的通道。

应用程序发出：

```
MQCONN (*SALE)
```

根据规则，以下为该实例中所发生的情况：

1. 针对与应用程序 MQCONN 调用匹配的队列管理器名称 SALE，扫描客户机通道定义表 (CCDT)。
2. 找到针对 ALPHA 和 BETA 的通道定义。

3. 如果其中一个通道的 CLNTWGHT 值为 0，那么此通道被选中。 如果两个通道的 CLNTWGHT 值都为 0，那么通道 ALPHA 被选中，因为按字母顺序，它排在第一位。 如果这两个通道的 CLNTWGHT 值都为非零值，那么根据权重随机地选中其中一个通道。
4. 进行了启动该通道的尝试。
5. 如果通道 BETA 被选中，那么启动它的尝试成功。
6. 如果通道 ALPHA 被选中，那么启动它的尝试失败，因为通信链路已中断。 然后应用以下步骤：
   a. 针对队列管理器名称 SALE 的唯一其他通道为 BETA。
   b. 尝试启动此通道 - 尝试成功。
7. 检查是否有侦听器正在运行，显示有一个侦听器正在运行。 它没有连接到 SALE 队列管理器上，但是因为 MQI 调用参数中有一个星号 (*)，因此没有检查。 应用程序连接到 SALE_BACKUP 队列管理器并继续处理。

示例 *2*： 指定的队列管理器名称
在此示例中，应用程序必须连接到特定队列管理器。 应用程序为该队列管理器名称发出 MQCONN 调用。 选择了合适的通道。

应用程序需要一个到特定队列管理器的连接，名为 SALE，就像在 MQI 调用中看到的一样：

```
MQCONN (SALE)
```

根据规则，以下为该实例中所发生的情况：

1. 针对与应用程序 MQCONN 调用匹配的队列管理器名称 SALE，按通道名称字母顺序扫描客户机通道定义表 (CCDT)。
2. 第一个查找到的匹配的通道定义是 ALPHA。
3. 尝试启动此通道 - 未成功，因为通信链路已中断。
4. 再次扫描客户机通道定义表寻找队列管理器名称 SALE，并查找到通道名称 BETA。
5. 尝试启动此通道 - 尝试成功。
6. 查看侦听器正在运行的检查显示有一个正在运行，但它没有连接到 SALE 队列管理器。
7. 在客户机通道定义表中没有更多的条目。 应用程序无法继续，并接收到返回码 MQRC_Q_MGR_NOT_AVAILABLE。

示例 *3.* 队列管理器名称为空白或星号 *(*)*
在此示例中，应用程序不关心它连接至哪个队列管理器。 应用程序发出指定空白队列管理器名称或星号的 MQCONN。 选择了合适的通道。

这以与第 848 页的『示例 1：队列管理器名称包含星号 (*)』中相同的方式进行处理。

注: 如果此应用程序曾在 IBM MQ MQI client 之外的环境中运行并且名称为空白，此应用程序将尝试连接到缺省队列管理器。 从客户机环境中运行时，则不是这种情形；访问的是与通道所连接的侦听器相关联的队列管理器。

应用程序发出：

```
MQCONN ("")
```

或者

```
MQCONN (*)
```

根据规则，以下为该实例中所发生的情况：

1. 将按照通道名称的字母顺序扫描客户机通道定义表 (CCDT) 以查找与应用程序 MQCONN 调用匹配的空白队列管理器名称。

2. 通道名称 ALPHA 的条目在 SALE 定义中有一个队列管理器名称。 这与 MQCONN 调用参数不匹配，该参数要求队列管理器名称为空白。
3. 下一个条目用于通道名称 BETA。
4. 定义中的 queue manager name 为 SALE。 再一次，这与 MQCONN 调用参数不匹配，该参数要求队列管理器名称为空白。
5. 在客户机通道定义表中没有更多的条目。 应用程序无法继续，并接收到返回码 MQRC_Q_MGR_NOT_AVAILABLE。

客户机环境中的触发

在 IBM MQ MQI clients 上运行的 IBM MQ 应用程序发送的消息将以与任何其他消息完全相同的方式触发，这些消息可用于在服务器和客户机上触发程序。

触发通道中对触发进行了详细说明。

触发器监视器和要启动的应用程序必须在同一个系统上。

已触发队列的缺省特征与服务器环境中的缺省特征相同。 尤其是，如果将消息放入已触发队列（位于 z/OS 队列管理器本地）的客户机应用程序中没有指定 MQPMO 同步点控制选项，消息将放入工作单元内。 如果此时满足触发条件，触发器消息将放在同一工作单元中的启动队列上，并且工作单元结束前，触发器监视器不能检索此消息。 工作单元结束后，要触发的进程才会启动。

进程定义

您必须在服务器上定义进程定义，因为这与已设置触发条件的队列相关联。

进程对象用于定义要触发的内容。 如果客户机和服务器不在同一平台上运行，那么任何由触发器监视器启动的进程必须定义 *ApplType*，否则服务器会采用其缺省定义（即，通常与服务器相关联的应用程序类型）并导致故障。

例如，如果触发器监视器在 Windows 客户机上运行并希望向另一个操作系统上的服务器发送请求，那么必须定义 MQAT_WINDOWS_NT，否则另一个操作系统将使用其缺省定义，而此进程将失败。

触发器监视器

非 z/OS IBM MQ 产品提供的触发器监视器在 IBM i IBM i 和 UNIX, Linux, and Windows 系统的客户机环境中运行。

要运行触发器监视器，请发出以下其中一个命令：

- IBM i 在 IBM i 上：

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ULW 在 Windows, UNIX and Linux 平台上：

```
runmqtmc [-m QMgrName] [-q InitQ]
```

在缺省队列管理器上，缺省启动队列为 SYSTEM.DEFAULT.INITIATION.QUEUE。 启动队列是触发器监视器查找触发器消息的位置。 然后，触发器监视器会调用程序以获取相应的触发器消息。 此触发器监视器支持缺省应用程序类型，并与 runmqtrm 相同（除了它与客户机库链接之外）。

由触发器监视器构建的命令字符串如下：

1. 来自相关进程定义的 *ApplicId*。 *ApplicId* 是将在命令行上输入的要运行的程序的名称。
2. 从启动队列获取的位于引号内的 MQTMC2 结构。 具有此字符串（与提供的字符串完全相同，位于引号内）的命令字符串将按照系统命令接受其作为一个参数的顺序启动。
3. 来自相关进程定义的 *EnvrData*。

触发器监视器启动的应用程序完成之前，它不会查看启动队列上是否有其他消息。 如果应用程序要执行很多处理，触发器监视器可能跟不上收到的触发器消息数量。 有两种方法可以处理这种情况：

1. 运行更多触发器监视器

如果您选择运行更多的触发器监视器，您可以控制任一时间可以运行的应用程序的最大数目。

2. 在后台运行已经启动的应用程序

   如果您选择在后台运行应用程序，IBM MQ 不会对可以运行的应用程序数量施加任何限制。

To run the started application in the background on UNIX and Linux systems, you must put an & (ampersand) at the end of the *EnvrData* of the process definition.

CICS 应用程序（非 *z/OS*）
必须将发出 MQCONN 或 MQCONNX 调用的非 z/OS CICS 应用程序定义为 RESDA。 如果作为客户机重新链接 CICS 服务器应用程序，会有丢失同步点支持的风险。

必须将发出 MQCONN 或 MQCONNX 调用的非 z/OS CICS 应用程序定义为 RESDA。 要使驻留代码尽可能小，您可以链接至一个独立程序以发出 MQCONN 或 MQCONNX 调用。

如果 MQSERVER 环境变量用于定义客户机连接，那么 CICSENV.CMD 文件中必须指定此变量。

IBM MQ 应用程序可以在 IBM MQ 服务器环境或 IBM MQ 客户机上运行，无需更改代码。 但是，在 IBM MQ 服务器环境中，CICS 可以充当同步点协调程序，您使用 EXEC CICS SYNCPOINT 和 EXEC CICS SYNCPOINT ROLLBACK，而不是 **MQCMIT** 和 **MQBACK**。 如果 CICS 应用程序只是作为客户机重新链接，将丢失同步点支持。 **MQCMIT** 和 **MQBACK** 必须用于 IBM MQ MQI client 上运行的应用程序。

Windows UNIX 准备和运行 CICS 和 Tuxedo 应用程序

要作为客户机应用程序运行 CICS 和 Tuxedo 应用程序，可使用与用于服务器应用程序的库不同的库。 运行应用程序的用户标识也不同。

要准备将 CICS 和 Tuxedo 应用程序作为 IBM MQ MQI client 应用程序运行，请遵循配置扩展事务客户机中的指示信息。

但请注意，专门用于准备 CICS 和 Tuxedo 应用程序的信息（包括随 IBM MQ 一起提供的样本程序）假设您要准备的是在 IBM MQ 服务器系统上运行的应用程序。 因此，此信息只引用旨在于服务器系统上使用的 IBM MQ 库。 在准备客户机应用程序时，您必须执行以下操作：

- 对您的应用程序使用的语言绑定使用适当的客户机系统库。 例如：
  - UNIX 对于使用 C 语言在 UNIX 上编写的应用程序，请使用库 libmqic 而不是 libmqm。
  - Windows 在 Windows 系统上，请使用库 mqic.lib 而不是 mqm.lib。
- 不使用第 851 页的表 138 和第 851 页的表 139 中显示的服务器系统库，而使用等效的客户机系统库。 如果服务器系统库未在这些表中列出，请使用客户机系统上的相同库。

表 *138: UNIX* 上的客户机系统库

用于 IBM MQ 服务器系统的库	**用在 IBM MQ 客户机系统上的等效库**
libmqmxa	libmqcxa

表 *139: Windows* 系统上的客户机系统库

用于 IBM MQ 服务器系统的库	**用在 IBM MQ 客户机系统上的等效库**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

客户机应用程序使用的用户标识。

在 CICS 下运行 IBM MQ 服务器应用程序时，它通常从 CICS 用户切换到事务的用户标识。 但是，在 CICS 下运行 IBM MQ MQI client 应用程序时，它将保留 CICS 特权权限。

Windows UNIX *CICS* 和 *Tuxedo* 样本程序

CICS 和 Tuxedo 样本程序，在 UNIX 和 Windows 系统上使用。

第 852 页的表 140 列出了为在 UNIX 客户机系统上使用而提供的 CICS 和 Tuxedo 样本程序。 第 852 页的表 141 列出了针对 Windows 客户机系统的等效信息。 该表也列出了用于准备和运行程序的文件。 有关这些样本程序的描述，请参阅 第 982 页的『CICS 样本事务』和 第 1022 页的『在 UNIX 和 Windows 上使用 TUXEDO 样本』。

表 *140: UNIX* 客户机系统的样本程序

描述	源	可执行文件模块
CICS 程序	amqscic0.ccs	amqscicc
CICS 程序的头文件	amqscih0.h	-
放置消息的 Tuxedo 客户机程序	amqstxpx.c	-
获取消息的 Tuxedo 客户机程序	amqstxgx.c	-
用于两个客户机程序的 Tuxedo 服务器程序	amqstxsx.c	-
Tuxedo 程序的 UBBCONFIG 文件	ubbstxcx.cfg	-
Tuxedo 程序的字段表文件	amqstxvx.flds	-
Tuxedo 程序的视图描述文件	amqstxvx.v	-

表 *141: Windows* 客户机系统的样本程序

描述	源	可执行文件模块
CICS 事务	amqscic0.ccs	amqscicc
CICS 事务的头文件	amqscih0.h	-
放置消息的 Tuxedo 客户机程序	amqstxpx.c	-
获取消息的 Tuxedo 客户机程序	amqstxgx.c	-
用于两个客户机程序的 Tuxedo 服务器程序	amqstxsx.c	-
Tuxedo 程序的 UBBCONFIG 文件	ubbstxcx.cfg	-
Tuxedo 程序的字段表文件	amqstxvx.fld	-
Tuxedo 程序的视图描述文件	amqstxvx.v	-
Tuxedo 程序的 Makefile	amqstxmc.mak	-
Tuxedo 程序的 ENVFILE 文件	amqstxen.env	-

Windows UNIX 错误消息 *AMQ5203*（针对 *CICS* 和 *Tuxedo* 应用程序进行过修改）

在您运行使用扩展事务客户机的 CICS 或 Tuxedo 应用程序时，您可能会看到标准诊断消息。 其中一条消息已经过修改，可以用于扩展事务客户机

您可能在 IBM MQ 错误日志文件中看到的消息记录在 诊断消息: AMQ4000-9999 中。 消息 AMQ5203 已经过修改，可用于扩展事务客户机。 下面是修改后的消息文本:

AMQ5203: 调用 XA 接口时发生错误。

说明

错误编号为 &2，值 1 表示提供的标志值 &1 无效，2 表示尝试了在同一进程中使用线程库和非线程库，3 表示提供的队列管理器名称“&3”存在错误，4 表示资源管理器标识 &1 无效，5 表示在已连接队列管理器的情况下尝试了使用名为“&3”的另一个队列管理器，6 表示在应用程序未连接到队列管理器时调用了事务管理器，7 表示在正在进行另一个调用时进行了 XA 调用，8 表示 xa_open 调用中的 xa_info 字符串

“&4”包含的参数值对于参数名称“&5”无效，9 表示 xa_open 调用中的 xa_info 字符串“&4”缺少名为“&5”的必需参数。

用户响应

请更正错误并重试该操作。

Windows 准备和运行 Microsoft Transaction Server 应用程序

要准备让 MTS 应用程序作为 IBM MQ MQI client 应用程序运行，请遵循适合您环境的指示说明。

有关如何开发访问 IBM MQ 资源的 Microsoft Transaction Server (MTS) 应用程序的常规信息，请参阅 IBM MQ 帮助中心中有关 MTS 的部分。

要准备让 MTS 应用程序作为 IBM MQ MQI client 应用程序运行，请对应用程序的各个组件执行下列操作：

- 如果组件为 MQI 使用 C 语言绑定，请遵循第 929 页的『在 Windows 中准备 C 程序』中的指示信息，但请将组件与库 mqicxa.lib（而不是 mqic.lib）链接在一起。
- 如果组件使用 IBM MQ C++ 类，请遵循第 459 页的『在 Windows 上构建 C++ 程序』中的指示信息，但请将组件与库 imqx23vn.lib（而不是 imqc23vn.lib）链接在一起。
- 如果组件将 Visual Basic 语言绑定用于 MQI，请遵循第 932 页的『在 Windows 中准备 Visual Basic 程序』中的指示说明，但在定义 Visual Basic 项目时，请在**条件编译自变量**字段中输入 MqType=3。
- 如果组件使用 IBM MQ Automation Classes for ActiveX (MQAX)，请定义一个值为 `mqic32xa.dll` 的环境变量 GMQ_MQ_LIB。

  您可以在您的应用程序中定义环境变量，或者将它定义为作用域是整个系统。 但是，将它的作用域定义为系统将导致任何现有的未在应用程序中定义环境变量的 MQAX 应用程序的行为不正常。

准备和运行 IBM MQ JMS 应用程序

您可以在客户机模式下运行 IBM MQ JMS 应用程序，使用 WebSphere Application Server 作为您的事务管理器。 您可能会看到某些警告消息。

要在客户机模式下准备并运行 IBM MQ JMS 应用程序，并使用 WebSphere Application Server 作为您的事务管理器，请遵循第 67 页的『使用 IBM MQ classes for JMS』中的指示信息。

在您运行 IBM MQ JMS 客户机应用程序时，您可能会看到以下警告消息：

MQJE080

许可证单元不足 - 运行 setmqcap

MQJE081

包含许可证单元信息的文件格式错误 - 运行 setmqcap

MQJE082

找不到包含许可证单元信息的文件 - 运行 setmqcap

用户出口、API 出口和 IBM MQ 可安装服务

本主题包含有关使用和开发这些程序的信息链接

有关如何使用用户出口、API 出口和可安装服务来扩展队列管理器设施的简介，请参阅扩展队列管理器设施。

有关编写和编译出口和可安装服务的信息，请参阅子主题。

相关概念

MQI 通道的通道出口程序

相关参考

API 出口参考

可安装服务接口参考信息

IBM i IBM i 上的可安装服务接口参考信息

ULW 在 UNIX、Linux 和 Windows 上编写出口和可安装服务

在没有链接到任何 IBM MQ 库的情况下，您可以在 UNIX、Linux 和 Windows 上编写和编译出口。

关于此任务

本主题仅适用于 UNIX, Linux, and Windows 系统。 有关为其他平台编写出口和可安装服务的详细信息，请参阅特定于有关平台的主题。

如果 IBM MQ 安装在非缺省位置，那么必须在不链接任何 IBM MQ 库的情况下编写和编译出口。

您可以在 UNIX, Linux, and Windows 系统上编写和编译出口，而无需链接以下任何 IBM MQ 库:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

链接到这些库的现有出口将继续运作，前提是在 UNIX and Linux 系统上，IBM MQ 安装在缺省位置。

过程

1. 包含 cmqec.h 头文件。
   包含此头文件可自动包含 cmqc.h、cmqxc.h 和 cmqzc.h 头文件。
2. 编写一个出口，以便通过 MQIEP 结构执行 MQI 和 DCI 调用。 有关 MQIEP 结构的更多信息，请参阅 MQIEP 结构。
   - 可安装服务
     - 使用 **Hconfig** 参数指向 MQZEP 调用。
     - 使用 **Hconfig** 参数之前，您必须检查 **Hconfig** 的前四个字节与 MQIEP 结构的 **StrucId** 是否匹配。
     - 有关编写可安装服务组件的更多信息，请参阅 MQIEP。
   - API 出口
     - 使用 **Hconfig** 参数指向 MQXEP 调用。
     - 使用 **Hconfig** 参数之前，您必须检查 **Hconfig** 的前四个字节与 MQIEP 结构的 **StrucId** 是否匹配。
     - 有关编写 API 出口的更多信息，请参阅第 868 页的『编写 API 出口』。
   - 通道出口
     - 使用 MQCXP 结构的 **pEntryPoints** 参数指向 MQI 和 DCI 调用。
     - 使用 **pEntryPoints** 之前，您必须检查 MQCXP 版本号是否为 8 或更高版本。
     - 有关编写通道出口的更多信息，请参阅第 877 页的『编写通道出口程序』。
   - 数据转换出口
     - 使用 MQDXP 结构的 **pEntryPoints** 参数指向 MQI 和 DCI 调用。
     - 使用 **pEntryPoints** 之前，您必须检查 MQDXP 版本号是否为 2 或更高版本。
     - 您可以使用 **crtmqcvx** 命令和 amqsvfc0.c 源文件创建使用 **pEntryPoints** 参数的数据转换代码。请参阅第 900 页的『为 IBM MQ for Windows 编写数据转换出口』和第 897 页的『在 UNIX and Linux 系统上为 IBM MQ 编写数据转换出口』。
     - 如果您拥有通过 **crtmqcvx** 命令生成的现有数据转换出口，您必须使用更新后的命令重新生成出口。
     - 有关编写数据转换出口的更多信息，请参阅第 893 页的『编写数据转换出口』。

- 预连接出口
  - 使用 MQNXP 结构的 **pEntryPoints** 参数指向 MQI 和 DCI 调用。
  - 使用 **pEntryPoints** 之前，您必须检查 MQNXP 版本号是否为 2 或更高版本。
  - 有关编写预连接出口的更多信息，请参阅第 902 页的『使用存储库的预连接出口引用连接定义』。
- 发布出口
  - 使用 MQPSXP 结构的 **pEntryPoints** 参数指向 MQI 和 DCI 调用。
  - 使用 **pEntryPoints** 之前，您必须检查 MQPSXP 版本号是否为 2 或更高版本。
  - 有关编写发布出口的更多信息，请参阅第 903 页的『编写和编译发布出口』。
- 集群工作负载出口
  - 使用 MQWXP 结构的 **pEntryPoints** 参数指向 MQXCLWLN 调用。
  - 使用 **pEntryPoints** 之前，您必须检查 MQWXP 版本号是否为 4 或更高版本。
  - 有关编写集群工作负载出口的更多信息，请参阅第 905 页的『编写和编译集群工作负载出口』。

例如，在调用 MQPUT 的通道出口中：

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

您可以在第 963 页的『使用 IBM MQ 样本过程程序』中查看其他示例。

3. 编译出口：
   - 不要链接到 IBM MQ 库。
   - 不要在出口中包含任何 IBM MQ 库的嵌入式 RPath。
   - 有关编译出口的更多信息，请参阅以下其中一个主题：
     - API 出口：第 869 页的『编译 API 出口』。
     - 通道出口、发布出口和集群工作负载出口：第 892 页的『在 Windows、UNIX and Linux 系统上编译通道出口程序』。
     - 数据转换出口：第 893 页的『编写数据转换出口』。
4. 将出口置于以下其中一个位置：
   - 配置出口时您完全限定选择的路径
   - 特定安装目录中的缺省出口路径。 例如，*MQ_DATA_PATH*/exits/installation2。
   - 缺省出口路径

     缺省出口路径为 *MQ_DATA_PATH*/exits（对于 32 位出口）和 *MQ_DATA_PATH*/exits64（对于 64 位出口）。 您可以在 qm.ini 或 mqclient.ini 文件中更改这些路径。 有关更多信息，请参阅出口路径。 在 Windows 和 Linux 上，您可以使用 IBM MQ 资源管理器更改路径：

     a. 右键单击队列管理器名称

     b. 单击**属性...**

     c. 单击**出口**

     d. 在出口缺省路径字段中，指定保存出口程序的目录的路径名称。

   如果同时将出口放入特定安装目录和缺省路径目录，路径中指定的 IBM MQ 安装将使用特定安装目录出口。 例如，出口位于 /exits/installation2 和 /exits 中，而不是位于 /exits/installation1 中。 IBM MQ 安装 installation2 将使用 /exits/installation2 中的出口。 IBM MQ 安装 installation1 将使用 /exits 目录中的出口。
5. 若有必要，请配置出口：

- 可安装服务：第 862 页的『配置服务和组件』。
- API 出口：第 872 页的『配置 API 出口』。
- 通道出口：第 893 页的『配置通道出口』。
- 发布出口：第 904 页的『配置发布出口』。
- 预连接出口：客户机配置文件的 PreConnect 节。

ULW *API 出口未与 MQI 库相链接*

在某些情况下，应将无法重新编码以使用 MQIEP 函数指针的现有 API 出口与 IBM MQ API 库相链接。

这是必需的，以便现有 API 出口可由系统的运行时链接程序成功装入到尚未装入函数指针的程序。

注: 此信息仅限于直接进行 MQI 调用的现有 API 出口。 即，不使用 MQIEP 的出口。 在可能的情况下，应计划对出口重新编码以改用 MQIEP 入口点。

从 IBM MQ 8.0 开始，**runmqsc** 是未直接与 MQI 库链接的程序的示例。

因此，未与其必需的 IBM MQ API 库链接或重新编码以使用 MQIEP 的 API 出口无法装入到 **runmqsc** 中。

您会在队列管理器错误日志中看到错误，例如，AMQ6175: **系统无法动态装入共享库**，以及诸如 `undefined symbol: MQCONN` 之类的限定性文本。

还有 AMQ7214: **无法装入 API 出口“myexitname”的模块**。

相关任务

第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』
在没有链接到任何 IBM MQ 库的情况下，您可以在 UNIX、Linux 和 Windows 上编写和编译出口。

ULW UNIX、Linux 和 Windows 的可安装服务和组件

本节介绍了可安装服务以及与之相关的功能和组件。 说明了这些功能的接口，这样您或者软件供应商可以提供这些组件。

为 IBM MQ 提供可安装服务的主要原因为:

- 为您选择是使用 IBM MQ 产品提供的组件，还是使用其他组件替换或扩充这些组件提供灵活性。
- 通过提供可以使用新技术的组件，无需对 IBM MQ 产品进行内部更改便可让供应商参与其中。
- 让 IBM MQ 更快速更经济的利用新技术，从而尽早以更低的价格提供产品。

可安装服务和服务组件是 IBM MQ 产品结构的一部分。 此结构的中心是实现与消息队列接口（MQI）关联的功能和规则的队列管理器的部件。 此中央部件需要许多称为可安装服务的服务功能，以便执行其工作。 可安装服务为:

- 授权服务
- 名称服务

每个可安装服务是使用一个或多个服务组件实现的一组相关功能。 每个组件都使用设计合理的公共接口来调用。 这使得独立软件供应商和其他第三方合作伙伴可以提供可安装组件，以扩充或替换 IBM MQ 产品提供的组件。 第 856 页的表 142 概述了可以使用的服务和组件。

表 *142:* 可安装服务组件摘要

可安装服务	提供的组件	函数	需求
授权服务	对象权限管理器 (object authority manager，OAM)	提供对命令和 MQI 调用的权限检查。 用户可以编写自己的组件来扩充或替换 OAM。 例如，检查某个用户标识是否有权限打开队列。	（采用适当的平台授权工具）

表 142: 可安装服务组件摘要 (继续)			
可安装服务	**提供的组件**	**函数**	**需求**
名称服务	None	为队列管理器提供支持，以查找拥有指定队列的队列管理器的名称。 • 用户定义	• 第三方或用户编写的名称管理器

可安装服务接口参考信息对可安装服务接口进行了描述。

相关任务
配置可安装服务

编写服务组件
本节描述服务、组件、入口点以及返回码之间的关系。

功能和组件

每个服务包含一组相关的功能。 例如，名称服务包含以下功能：

- 查找队列名称并返回定义此队列的队列管理器的名称。
- 将队列名称插入服务目录。
- 从服务目录中删除队列名称

还包含初始化和终止功能。

可安装服务由一个或多个服务组件提供。 每个组件可以执行为该服务定义的部分或所有功能。 例如，在 IBM MQ for AIX 中，提供的授权服务组件 OAM 可执行所有可用功能。 请参阅第 860 页的『授权服务接口』获取更多信息。 此组件还负责管理实现此服务所需的任何底层资源或软件（例如，LDAP 目录）。 配置文件提供一种装入组件及确定它提供的功能例程地址的标准方法。

第 857 页的图 104 显示服务与组件是如何进行关联的：

- 通过配置文件中的节，将服务定义到队列管理器。
- 通过队列管理器中提供的代码支持每个服务。 用户无法更改此代码，因此无法创建自己的服务。
- 每个服务由一个或多个组件实现；这些组件可以与产品一起提供或由用户进行编写。 可以调用一个服务的多个组件，每个组件支持服务中的不同工具。
- 入口点将服务组件连接到队列管理器中的支持代码。

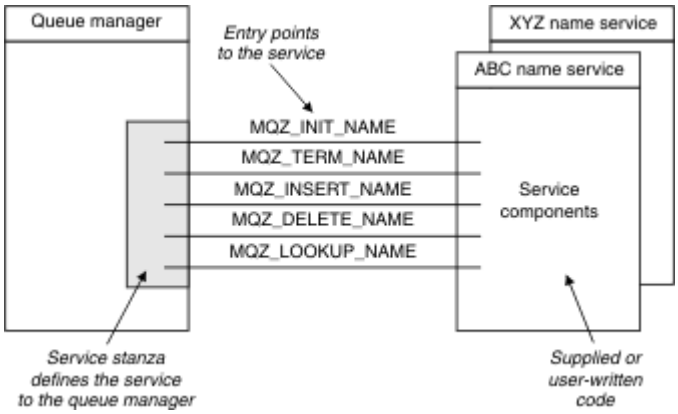


图 104: 理解服务、组件和入口点

入口点

每个服务组件通过一个支持特定可安装服务的例程的入口点地址列表表示。 可安装服务定义由每个例程执行的功能。

配置服务组件时的顺序定义了尝试满足服务请求时调用入口点的顺序。

在提供的头文件 cmqzc.h 中，对每个服务提供的入口点具有 MQZID_ 前缀。

如果存在这些服务，将按照预定义顺序加载这些服务。 以下列表显示了服务以及对其进行初始化的顺序。

1. NameService
2. AuthorizationService
3. UserIdentifierService

AuthorizationService 是唯一一个缺省配置的服务。 如果您希望使用 NameService 和 UserIdentifierService，请手动对其进行配置。

服务和服务组件有一对一或一对多映射。 可以为每个服务定义多个服务组件。 在 UNIX and Linux 系统上，ServiceComponent 节的服务值必须与 qm.ini 文件中服务节的名称值匹配。 在 Windows 上，ServiceComponent 的服务注册表键值必须与名称注册表键值匹配，并按照如下所示定义：HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation*MQ_INSTALLATION_NAME*\Configuration\QueueManager*qmname*\，其中 *qmname* 是队列管理器的名称。

对于 UNIX and Linux 系统，服务组件按照它们在 qm.ini 文件中的定义顺序启动。 在 Windows 上，因为使用 Windows 注册表，IBM MQ 将发出 **RegEnumKey** 调用，此调用将按照字母顺序返回值。 因此，在 Windows 上，将根据服务在注册表中的定义顺序按字母顺序调用这些服务。

ServiceComponent 定义的顺序非常重要。 此顺序决定为给定服务运行组件的顺序。 例如，AuthorizationService on Windows 是使用名为 MQSeries.WindowsNT.auth.service 的缺省 OAM 组件配置的。 可以为此服务定义其他组件来覆盖缺省 OAM。 除非指定 MQCACF_SERVICE_COMPONENT，否则将使用按字母顺序遇到的第一个组件来处理请求并使用此组件的名称。

返回码

服务组件为队列管理器提供返回码，以报告各种情况。 这些返回码用于报告操作成功或失败，并指示队列管理器是否继续至下一个服务组件。 单独的 *Continuation* 参数执行此指示。

组件数据

单个服务组件可能要求在它的各种功能之间共享数据。 可安装服务提供每次调用服务组件时传递的选用数据区。 此数据区供服务组件独占使用。 即使特定函数的所有调用来自不同地址空间或进程，它们也共享此数据区。 无论何时调用服务组件，都能保证可从服务组件设置地址。 您必须在 *ServiceComponent* 节中声明此区域的大小。

组件的初始化和终止

组件初始化和终止选项的使用。

调用组件初始化例程时，此例程必须为此组件支持的每个入口点调用队列管理器 **MQZEP** 函数。 **MQZEP** 将入口点定义到此服务。 所有未定义的出口点假设为 NULL。

在以任何其他方法调用组件之前，始终使用主初始化选项调用该组件一次。

可在特定平台上使用此辅助初始化选项调用的组件。 例如，可以对每个操作系统进程、线程或访问服务的任务调用一次该组件。

如果使用辅助初始化：

- 可以为辅助初始化多次调用此组件。 对于每个这样的调用，当不再需要服务时会发出辅助终止的匹配调用。

  对于命名服务，此调用为 MQZ_TERM_NAME。

  对于授权服务，此调用为 MQZ_TERM_AUTHORITY。
- 每次对主初始化和辅助初始化调用此组件时，必须重新指定入口点（通过调用 MQZEP）。
- 只有一个组件数据副本用于此组件；对于每个辅助初始化没有不同的副本。

- 在进行辅助初始化之前，不对此服务的任何其他调用（从相应地操作系统进程、线程或任务进行）调用此组件。
- 对于主初始化和辅助初始化，此组件必须将 **Version** 参数设置为相同的值。

当不再需要此组件时，始终会使用主终止选项调用一次此组件。 不对此组件进行其他调用。

如果已为辅助初始化调用此组件，将使用辅助终止选项调用此组件。

对象权限管理器 *(OAM)*
随 IBM MQ 产品一起提供的授权服务组件称为对象权限管理器 (OAM)。

缺省情况下，OAM 处于活动状态，并使用控制命令 **dspmqaut**（显示权限）、**dmpmqaut**（转储权限）和 **setmqaut**（设置或重置权限）。

使用控制命令管理对这些命令的语法及其使用方法进行了描述。

OAM 用于主体或组的实体：

- Linux UNIX 在 UNIX and Linux 系统上，主体为用户标识或与代表用户运行的应用程序关联的标识；组是系统定义的一组主体。
- Windows 在 Windows 系统上，主体为 Windows 用户标识或与代表用户运行的应用程序关联的标识；组是 Windows 组。

可在主体级别或组级别授予或撤销权限。

当进行 MQI 请求或发出命令时，OAM 将检查与此操作关联的实体是否有权执行请求的操作以及是否有权访问指定的队列管理器资源。

授权服务使您可以编写自己的授权服务组件，来扩充或替换为队列管理器提供的权限检查。

名称服务
名称服务是一个可安装服务，用于为队列管理器查找拥有指定队列的队列管理器名称提供支持。 不能从名称服务检索其他队列属性。

名称服务使应用程序能够打开远程队列以用于输出，就象它们是本地队列一样。 对于队列以外的其他对象，不调用名称服务。

注: 远程队列必须将其 **Scope** 属性设置为 CELL。

当应用程序打开队列时，首先会在队列管理器的目录中查找队列的名称。 如果在目录中没有找到队列名称，那么它会查看已配置的所有名称服务，直至找到识别此队列名称的名称服务。 如果无名称服务识别此名称，则此打开失败。

名称服务返回拥有该队列的队列管理器。 然后队列管理器继续处理 MQOPEN 请求，就像此命令已在原始请求中指定队列和队列管理器名称一样。

名称服务接口 (NSI) 是 IBM MQ 框架的一部分。

名称服务如何工作

如果队列定义指定 **Scope** 属性作为队列管理器（即 MQSC 中的 SCOPE(QMGR)），队列定义（与所有队列属性一起）将只存储在队列管理器的目录中。 这无法由可安装服务替换。

如果队列定义指定 **Scope** 属性作为一个单元（即 MQSC 中的 SCOPE(CELL），队列定义将随所有队列属性同样存储到队列管理器的目录中。 但是，队列和队列管理器名称也存储在名称服务中。 如果没有可用于存储此信息的服务，则无法定义具有 *Scope* 单元的队列。

要达到此目的，存储信息的目录可由该服务管理，或者该服务可以使用底层服务，例如，LDAP 目录。 无论是哪种情况，即使组件和队列管理器已终止，存储在目录中的定义也必须保留到显式删除它们的时候。

注:

1. 要将消息发送到远程主机的命名目录单元内其他队列管理器上的本地队列定义（作用域为 CELL），您需要定义通道。
2. 即使远程队列的作用域设置为 CELL，您也无法直接从其获取消息。

3. 发送到作用域为 CELL 的队列时不需要远程队列定义。
4. 命名服务主要定义目标队列，虽然您仍需要一个目标队列管理器的传输队列和一对通道定义。 此外，本地系统上的传输队列必须具有与远程系统上拥有目标队列（具有单元作用域）的队列管理器相同的名称。

   例如，如果远程队列管理器具有名称 QM01，则本地系统上的传输队列也必须具有名称 QM01。

授权服务接口
授权服务提供供队列管理器使用的入口点。

入口点如下所示：

MQZ_AUTHENTICATE_USER
认证用户标识和密码，并可以设置身份上下文字段。

MQZ_CHECK_AUTHORITY
检查实体是否有权限对指定的对象执行一个或多个操作。

MQZ_CHECK_PRIVILEGED
检查指定用户是否是特权用户。

MQZ_COPY_ALL_AUTHORITY
将引用的对象当前存在的所有权限复制到另一个对象。

MQZ_DELETE_AUTHORITY
删除与指定对象关联的所有权限。

MQZ_ENUMERATE_AUTHORITY_DATA
检索符合指定的选择标准的所有权限数据。

MQZ_FREE_USER
释放相关联的已分配资源。

MQZ_GET_AUTHORITY
获取实体访问指定的对象的权限。

MQZ_GET_EXPLICIT_AUTHORITY
获取指定的组访问指定对象的权限（但是不获取 **nobody** 组的附加权限）或获取指定主体的主组访问指定对象的权限。

MQZ_INIT_AUTHORITY
初始化授权服务组件。

MQZ_INQUIRE
查询授权服务的受支持的功能。

MQZ_REFRESH_CACHE
刷新所有权限。

MQZ_SET_AUTHORITY
设置实体对指定对象的权限。

MQZ_TERM_AUTHORITY
终止授权服务组件。

此外，在 IBM MQ for Windows 上，授权服务提供以下供队列管理器使用的入口点：

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

这些入口点支持使用 Windows 安全标识 (NT SID)。

这些名称在头文件 `cmqzc.h` 中定义为 **typedef**，可用于制作组件功能的原型。

初始化函数 (**MQZ_INIT_AUTHORITY**) 必须是组件的主入口点。 其他函数通过初始化函数添加到组件入口点向量的入口点地址来调用。

名称服务接口
名称服务提供供队列管理器使用的入口点。

提供了以下入口点：

MQZ_INIT_NAME
初始化名称服务组件。

MQZ_TERM_NAME
终止名称服务组件。

MQZ_LOOKUP_NAME
查找指定队列的队列管理器名。

MQZ_INSERT_NAME
将包含拥有指定队列的队列管理器名的条目插入服务所使用的目录。

MQZ_DELETE_NAME
从此服务所使用的目录删除指定队列的条目。

如果配置了多个名称服务：

- 对于查找，为列表中的每个服务调用 MQZ_LOOKUP_NAME 函数，直至解析出此队列名（除非任何组件指示搜索应停止）。
- 对于插入，为列表中支持 MQZ_INSERT_NAME 函数的第一个服务调用此函数。
- 对于删除，为列表中支持 MQZ_DELETE_NAME 函数的第一个服务调用此函数。

请不要使用多个支持插入和删除函数的组件。 但是，仅支持查找的组件是可行的，并且可使用（举例来说），因为列表中的最后一个组件将任何其它名称服务组件未知的名称解析到可以定义此名称的队列管理器。

在 C 编程语言中，可以使用 typedef 语句将名称定义为函数数据类型。 这些数据类型可用于制作服务函数的原型，以确保这些参数正确。

对于 C 语言，包含特定于可安装服务的所有资料的头文件是 `cmqzc.h`。

除了必须是组件主入口点的初始化函数（MQZ_INIT_NAME）以外，这些函数将由初始化函数添加的入口点地址使用 MQZEP 进行调用。

使用多个服务组件
您可以为一个服务安装多个组件。 这允许组件仅提供服务的部分实现功能，并依赖其他组件提供其余功能。

使用多个组件的示例

假设您创建两个名称服务组件，称为 `ABC_name_serv` 和 `XYZ_name_serv`。

ABC_name_serv
此组件支持在服务目录中插入名称或删除名称，但不支持查找队列名称。

XYZ_name_serv
此组件支持查找队列名，但不支持在服务目录中插入名称或删除名称。

组件 `ABC_name_serv` 持有一个队列名称的数据库，并使用两个简单算法在服务目录中插入名称或删除名称。

组件 `XYZ_name_serv` 使用一个简单算法，此算法返回使用任何队列名调用算法的固定队列管理器名称。此组件不持有队列名称数据库，因此不支持插入和删除功能。

组件安装在同一队列管理器上。 对 *ServiceComponent* 节进行了排序，因此将首先调用组件 `ABC_name_serv`。 任何在组件目录中插入或删除队列的调用均由组件 `ABC_name_serv` 处理；这是实现这些功能的唯一组件。 但组件 `ABC_name_serv` 不能解析的查询调用将传递给仅用于查找的组件 `XYZ_name_serv`。 此组件通过其简单算法提供队列管理器名称。

在使用多个组件时省略入口点

如果您决定要使用多个组件提供服务，您可以设计不实现特定功能的服务组件。 可安装服务框架对您可省略的功能没有任何限制。 但是，对于特定可安装服务，省略一个或多个功能可能在逻辑上与服务的目的不一致。

与多个组件一起使用入口点的示例

第 862 页的表 143 显示已安装两个组件的可安装名称服务示例。 每个组件都支持与此特定可安装服务关联的不同功能集。 对于插入功能，首先调用 ABC 组件入口点。 未定义到服务（使用 **MQZEP**）的入口点假设为 NULL。 表中提供了用于初始化的入口点，但这不是必需的，因为初始化由组件的主入口点实现。

当队列管理器必须使用可安装服务时，将使用为该服务（第 862 页的表 143 中的列）定义的入口点。 依次采用每个组件，队列管理器确定实现必需功能的例程地址。 如果例程存在，则调用该例程。 如果操作成功，那么队列管理器会使用任何结果和状态信息。

表 *143:* 可安装服务的入口点示例

功能号	**ABC 名称服务组件**	**XYZ 名称服务组件**
MQZID_INIT_NAME（初始化）	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME（终止）	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME（插入）	ABC_Insert()	NULL
MQZID_DELETE_NAME（删除）	ABC_Delete()	NULL
MQZID_LOOKUP_NAME（查找）	NULL	XYZ_Lookup()

如果例程不存在，则队列管理器为列表中的下一个组件重复此进程。 另外，如果例程确实存在但返回一个表明它无法执行此操作的代码，将继续尝试下一个可用的组件。 服务组件中的例程可能返回一个代码，指示不应该再尝试执行此操作。

配置服务和组件

除非在 Windows 系统上（每个队列管理器在注册表中都有自己的节），否则请使用队列管理器配置文件来配置服务组件。

过程

1. 将节添加到队列管理器配置文件，以将服务定义到队列管理器并指定模块的位置:
   - 使用的每个服务都必须有 `Service` 节，此节可将服务定义到队列管理器。 有关更多信息，请参阅 qm.ini 文件的服务节。
   - 对于服务中的每个组件，必须要有一个 `ServiceComponent` 节。 此节标识包含该组件的代码的模块的名称和路径。 有关更多信息，请参阅 qm.ini 文件的 ServiceComponent 节。

   称为对象权限管理器（OAM）的权限服务组件随此产品一起提供。 当您创建队列管理器时，队列管理器配置文件（或者 Windows 系统上的注册表）自动更新为包含授权服务和缺省组件（OAM）的相应节。 对于其他组件，您必须手动配置队列管理器配置文件。

   在启动队列管理器时，在平台上支持的位置使用动态绑定将每个服务组件的代码装入此队列管理器。
2. 停止并重新启动队列管理器以激活组件。

相关参考

qm.ini 文件的服务节

qm.ini 文件的 ServiceComponent 节

更改用户权限之后刷新 *OAM*

在 IBM MQ 中，您可以在更改用户的授权组成员资格后立即刷新 OAM 的授权组信息以反应在操作系统级别所做的更改，无须停止并重新启动队列管理器。 要执行此操作，请发出 **REFRESH SECURITY** 命令。

注: 使用 `setmqaut` 命令更改权限时，QAM 会立即实现这些更改。

队列管理器将授权数据存储在名为 SYSTEM.AUTH.DATA.QUEUE 的本地队列中。 此数据由 **amqzfuma.exe** 管理。

相关参考

REFRESH SECURITY

IBM i IBM i 上的可安装服务和组件

使用本信息了解可安装服务以及与之相关的功能和组件。 说明了这些功能的接口，这样您或者软件供应商可以提供这些组件。

为 IBM MQ 提供可安装服务的主要原因为：

- 为您选择是使用 IBM MQ for IBM i 提供的组件，还是使用其他组件替换或扩充这些组件提供灵活性。
- 通过提供可以使用新技术的组件，无需对 IBM MQ for IBM i 进行内部更改便可让供应商参与其中。
- 让 IBM MQ 更快速更经济的利用新技术，从而尽早以更低的价格提供产品。

可安装服务和服务组件是 IBM MQ 产品结构的一部分。 此结构的中心是实现与消息队列接口（MQI）关联的功能和规则的队列管理器的部件。 此中央部件需要许多称为可安装服务的服务功能，以便执行其工作。 IBM MQ for IBM i 中可用的可安装服务为授权服务。

每个可安装服务是使用一个或多个服务组件实现的一组相关功能。 每个组件都使用设计合理的公共接口来调用。 这使得独立软件供应商和其他第三方合作伙伴可以提供可安装组件，以扩充或替换 IBM MQ for IBM i 提供的组件。 第 863 页的表 144 概括了对授权服务的支持。

表 *144:* 授权服务组件摘要

提供的组件	**函数**	**需求**
对象权限管理器 (OAM)	提供对命令和 MQI 调用的权限检查。 用户可以编写自己的组件来扩充或替换 OAM。	（采用适当的平台授权工具）
DCE 名称服务组件 **注:** DCE 仅在低于 V6.0 的 IBM MQ 版本上受支持。	• 允许队列管理器共享队列，或 • 用户定义 **注:** 共享队列必须将其 **Scope** 属性设置为 CELL。	• 提供的组件必需使用 DCE，或 • 第三方或用户编写的名称管理器

IBM i *IBM i* 上的功能和组件

使用本信息了解您可以在 IBM MQ for IBM i 中使用的功能和组件、入口点、返回代码和组件数据。

每个服务包含一组相关的功能。 例如，名称服务包含以下功能：

- 查找队列名称并返回定义此队列的队列管理器的名称。
- 将队列名称插入服务目录。
- 从服务目录中删除队列名称

还包含初始化和终止功能。

可安装服务由一个或多个服务组件提供。 每个组件可以执行为该服务定义的部分或所有功能。 此组件还负责管理实施此服务所需的任何基础资源或软件。 配置文件提供一种装入组件及确定它提供的功能例程地址的标准方法。

服务和组件是按照如下所示相关的：

- 通过配置文件中的节，将服务定义到队列管理器。
- 通过队列管理器中提供的代码支持每个服务。 用户无法更改此代码，因此无法创建自己的服务。
- 每个服务由一个或多个组件实现；这些组件可以与产品一起提供或由用户进行编写。 可以调用一个服务的多个组件，每个组件支持服务中的不同工具。
- 入口点将服务组件连接到队列管理器中的支持代码。

入口点

每个服务组件通过一个支持特定可安装服务的例程的入口点地址列表表示。 可安装服务定义由每个例程执行的功能。 配置服务组件时的顺序定义了尝试满足服务请求时调用入口点的顺序。 在提供的头文件 cmqzc.h 中，对每个服务提供的入口点具有 MQZID_ 前缀。

返回码

服务组件为队列管理器提供返回码，以报告各种情况。 这些返回码用于报告操作成功或失败，并指示队列管理器是否继续至下一个服务组件。 单独的 *Continuation* 参数执行此指示。

组件数据

单个服务组件可能要求在它的各种功能之间共享数据。 可安装服务提供每次调用特定服务组件时传递的选用数据区。 此数据区供服务组件独占使用。 即使对给定功能的所有调用是从不同的地址空间或进程发出的，它们也可共享此数据区。 无论何时调用服务组件，都能保证可从服务组件设置地址。 您必须在 *ServiceComponent* 节中声明此区域的大小。

IBM i IBM i 上的初始化

调用组件初始化例程时，此例程必须为此组件支持的每个入口点调用队列管理器 MQZEP 函数。 MQZEP 将入口点定义到此服务。 所有未定义的出口点假设为 NULL。

主初始化
在以任何其他方法调用组件之前，总是用此选项调用该组件一次。

辅助初始化
可在特定平台上使用此选项调用组件。 例如，可以对每个操作系统进程、线程或访问服务的任务调用一次该组件。

如果使用辅助初始化：

- 可以为辅助初始化多次调用此组件。 对于每个这样的调用，当不再需要服务时会发出辅助终止的匹配调用。

  对于授权服务，此调用为 MQZ_TERM_AUTHORITY。
- 每次对主初始化和辅助初始化调用此组件时，必须重新指定入口点（通过调用 MQZEP）。
- 只有一个组件数据副本用于此组件；对于每个辅助初始化没有不同的副本。
- 在进行辅助初始化之前，不对此服务的任何其他调用（从相应地操作系统进程、线程或任务进行）调用此组件。
- 对于主初始化和辅助初始化，此组件必须将 **Version** 参数设置为相同的值。

主终止
当不再需要主终止组件时，总是使用此选项启动一次该组件。 不对此组件进行其他调用。

辅助终止
如果已为辅助初始化启动辅助终止组件，那么以此选项启动此组件。

IBM i 在 IBM i 上配置服务和组件

您可以使用队列管理器配置文件来配置服务组件。

过程

1. 将节添加到队列管理器配置文件 qm.ini，以将服务定义到队列管理器并指定模块的位置：
   - 使用的每个服务都必须有 Service 节，此节可将服务定义到队列管理器。 有关更多信息，请参阅 qm.ini 文件的服务节。
   - 对于服务中的每个组件，必须要有一个 ServiceComponent 节。 此节标识包含该组件的代码的模块的名称和路径。 有关更多信息，请参阅 qm.ini 文件的 ServiceComponent 节。

称为对象权限管理器（OAM）的权限服务组件随此产品一起提供。 当您创建队列管理器时，队列管理器配置文件自动更新为包含授权服务和缺省组件（OAM）的相应节。 对于其他组件，您必须手动配置队列管理器配置文件。

在启动队列管理器时，在平台上支持的位置使用动态绑定将每个服务组件的代码装入此队列管理器。

2.

IBM i 在 *IBM i* 上创建您自己的服务组件

使用本信息了解如何在 IBM MQ for IBM i 上创建服务组件。

要创建自己的服务组件，请执行下列操作：

- 确保头文件 `cmqzc.h` 包含在您的程序中。
- 通过编译程序并将其与共享库 `libmqm*` 和 `libmqmzf*` 链接来创建共享库。

  注: 因为代理程序可以在线程化环境中运行，所以您必须构建 OAM 以在线程化环境中运行。 这包含使用 `libmqm` 和 `libmqmzf` 的线程化版本。
- 将节添加到队列管理器配置文件，以将服务定义到队列管理器，并指定模块的位置。
- 停止并重新启动队列管理器以激活组件。

IBM i *IBM i* 上的授权服务

授权服务是一个可安装服务，它使队列管理器能调用授权工具，例如，检查某个用户标识是否有权限打开队列。

此服务是 IBM MQ 安全性启用接口 (SEI) 的组件，此接口是 IBM MQ 框架的一部分。 将讨论以下主题：

- 第 865 页的『对象权限管理器 (OAM)』
- 第 865 页的『对操作系统定义服务』
- 第 866 页的『配置授权服务节』
- 第 866 页的『IBM i 上的授权服务接口』

对象权限管理器 (OAM)

随 IBM MQ 产品一起提供的授权服务组件称为对象权限管理器 (OAM)。 缺省情况下，OAM 处于活动状态并与以下控制命令一起使用：

- **WRKMQMAUT** 处理权限
- **WRKMQMAUTD** 处理权限数据
- **DSPMQMAUT** 显示对象权限
- **GRTMQMAUT** 授予对象权限
- **RVKMQMAUT** 撤销对象权限
- **RFRMQMAUT** 刷新安全性

CL 命令帮助对这些命令的语法及其使用方法进行了描述。 OAM 使用主体或组的实体。

当进行 MQI 请求或发出命令时，OAM 将检查与此操作关联的实体的权限，以确认其是否可以执行下列操作：

- 执行请求的操作。
- 访问指定的队列管理器资源。

授权服务使您可以编写自己的授权服务组件，来扩充或替换为队列管理器提供的权限检查。

对操作系统定义服务

队列管理器配置文件 `qm.ini` 中的授权服务节定义了队列管理器的授权服务。 请参阅第 864 页的『在 IBM i 上配置服务和组件』以获取关于节的类型的信息。

配置授权服务节

在 IBM MQ for IBM i 上：

主体
为 IBM i 系统用户概要文件。

组
为 IBM i 系统组概要文件。

只能在组级别授予或撤销权限。 请求授予或撤销某个用户权限将更新该用户的主组。

每个队列管理器都有它自己的队列管理器配置文件。 例如，队列管理器 QMNAME 的队列管理器配置文件的缺省路径和文件名为 /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini。

会自动将缺省授权组件的 *Service* 节和 *ServiceComponent* 节添加到 qm.ini，但是可能会被 WRKENVVAR 覆盖。 任何其他的 *ServiceComponent* 节都必须手动添加。

例如，队列管理器配置文件中的以下节定义两个授权服务组件：

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

图 *105: IBM i* 上 *qm.ini* 中的授权服务节

第一个服务组件节 MQ.UNIX.authorization.service 定义缺省授权服务组件 OAM。 如果您移除此节并重新启动队列管理器，OAM 将被禁用并且不会进行权限检查。

IBM i *IBM i* 上的授权服务接口

授权服务接口提供了多个供队列管理器使用的入口点。

MQZ_AUTHENTICATE_USER
认证用户标识和密码，并可以设置身份上下文字段。

MQZ_CHECK_AUTHORITY
检查实体是否有权限对指定的对象执行一个或多个操作。

MQZ_COPY_ALL_AUTHORITY
将引用的对象当前存在的所有权限复制到另一个对象。

MQZ_DELETE_AUTHORITY
删除与指定对象关联的所有权限。

MQZ_ENUMERATE_AUTHORITY_DATA
检索符合指定的选择标准的所有权限数据。

MQZ_FREE_USER
释放相关联的已分配资源。

MQZ_GET_AUTHORITY
获取实体访问指定的对象的权限。

MQZ_GET_EXPLICIT_AUTHORITY
获取指定的组访问指定对象的权限（但是不获取 **nobody** 组的附加权限）或获取指定主体的主组访问指定对象的权限。

MQZ_INIT_AUTHORITY
初始化授权服务组件。

MQZ_INQUIRE
查询授权服务的受支持的功能。

MQZ_REFRESH_CACHE
刷新所有权限。

MQZ_SET_AUTHORITY
设置实体对指定对象的权限。

MQZ_TERM_AUTHORITY
终止授权服务组件。

这些入口点支持使用 Windows 安全标识 (NT SID)。

这些名称在头文件 `cmqzc.h` 中定义为 **typedef**，可用于制作组件功能的原型。

初始化函数 (**MQZ_INIT_AUTHORITY**) 必须是组件的主入口点。 其他函数通过初始化函数添加到组件入口点向量的入口点地址来调用。

请参阅第 865 页的『在 IBM i 上创建您自己的服务组件』以获取更多信息。

Multi 在多平台上编写和编译 API 出口

通过 API 出口，可以编写用于更改 IBM MQ API 调用（如 MQPUT 和 MQGET）行为的代码，然后直接在这些调用之前或之后插入该代码。

注: z/OS 在 IBM MQ for z/OS 上不受支持。

为什么要使用 API 出口?

每个应用程序都需要完成特定的工作，而它的代码应尽可能高效地完成这一任务。 在更高的级别上，您可能想要为所有使用某一特定队列管理器的应用程序，对该队列管理器应用一些标准或业务流程。 这样做比起在个别应用程序级别进行修改的效率高得多，而且不必更改每个受影响的应用程序的代码。

下面是针对 API 出口可能适用于的方面提出的几个建议：

安全性
为了实现安全性，您可以提供认证，检查应用程序是否有权访问队列或队列管理器。 您还可以监管应用程序对 API 的使用，认证个别 API 调用（甚至是其使用的参数）。

灵活性
为了实现灵活性，您可以响应业务环境中的快速变化，而无需更改依赖于该环境中的数据的应用程序。例如，可以使用 API 出口来响应利率、货币汇率或生产环境中组件价格的变化。

监视队列或队列管理器的使用情况
为了监视队列或队列管理器的使用情况，您可以跟踪应用程序和消息流、记录 API 调用中的错误、设置用于记帐目的的审计跟踪或者收集用于规划目的的使用情况统计信息。

在运行 API 出口时会发生什么情况?

在编写出口程序并向 IBM MQ 标识此出口程序后，队列管理器会自动在注册点处调用该出口代码。

在以下平台的节中会指出要运行的 API 出口例程：

- IBM i IBM i
- Linux UNIX UNIX and Linux
- Windows Windows

本主题涵盖了配置文件 `mqs.ini` 和 `qm.ini` 中的节。

例程定义可能出现在以下三个位置：

1. mqs.ini 文件中的 ApiExitCommon，用于标识整个 IBM MQ 的例程，在队列管理器启动时应用。 这些项可被针对个别队列管理器定义的例程覆盖（请参阅此列表中的第 第 868 页的『3』 项）。

2. ApiExit 模板 (在 mqs.ini 文件中) 标识在创建新队列管理器时复制到 ApiExit 本地集的整个 IBM MQ 的例程 (请参阅此列表中的项 第 868 页的『3』)。
3. qm.ini 文件中的 ApiExitLocal，用于标识适用于特定队列管理器的例程。

当创建新的队列管理器时，会将 mqs.ini 中的 ApiExitTemplate 定义复制到新队列管理器 qm.ini 中的 ApiExitLocal 定义。 当队列管理器启动时，会使用 ApiExitCommon 和 ApiExitLocal 定义。 如果这两个定义标识了同名的例程，那么 ApiExitLocal 定义将替换 ApiExitCommon 定义。 第 872 页的『配置 API 出口』中描述的 Sequence 属性确定在节中定义的例程的运行顺序。

跨多个 IBM MQ 安装使用 API 出口

确保为较低版本的 IBM MQ 所编写的 API 出口可用于所有版本，因为对 IBM WebSphere MQ 7.1 中的出口所做的更改可能不能用于较低版本。 有关对出口所作的更改的更多信息，请参阅 第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』。

为 API 出口 amqsaem 和 amqsaxe 提供的样本反映了编写出口时所需的更改。 在启动客户机应用程序之前，该应用程序必须确保已将与队列管理器（与该应用程序相关联）安装对应的正确 IBM MQ 库链接到该应用程序。

Multi 编写 *API* 出口

您可以使用 C 编程语言为每个 API 调用编写出口。

可用出口

出口可用于每个 API 调用，具体如下：

- MQCB，为指定对象句柄重新注册回调并控制对回调的激活和更改
- MQCTL，对为连接打开的对象句柄执行控制操作
- MQCONN/MQCONNX，提供后续 API 调用使用的队列管理器连接句柄
- MQDISC，断开与队列管理器的连接
- MQBEGIN，开始全局工作单元（UOW）
- MQBACK，回退工作单元
- MQCMIT，落实工作单元
- MQOPEN，打开 IBM MQ 资源以供后续访问
- MQCLOSE，关闭之前为了访问而打开的 IBM MQ 资源
- MQGET，检索先前为进行访问而打开的队列中的消息
- MQPUT1，将消息放入队列
- MQPUT，将消息放入先前为了访问而打开的队列
- MQINQ，查询先前为了访问而打开的 IBM MQ 资源的属性
- MQSET，设置先前为了访问而打开的队列的属性
- MQSTAT，检索状态信息
- MQSUB，向特定主题注册应用程序预订
- MQSUBRQ，发出预订请求

MQ_CALLBACK_EXIT 提供出口函数以在回调处理之前和之后执行。 有关更多信息，请参阅回调 - MQ_CALLBACK_EXIT。

编写 API 出口

在 API 出口内，调用采用一般形式：

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

其中 *call* 是不带 MQ 前缀的 MQI 调用名称；例如，PUT、GET。 *parameters* 控制出口的功能，主要提供出口与外部控制块之间的通信 MQAXP (API 出口参数结构) 和 MQAXC (API 出口上下文结构)。 *context* 描述调用 API 出口的上下文，*ApiCallParameters* 表示 MQI 调用的参数。

为了帮助您编写 API 出口，提供了样本出口 amqsaxe0.c，此出口可为您指定的文件生成跟踪条目。 编写出口时，可以使用此样本作为起点。 有关使用样本出口的更多信息，请参阅 第 977 页的『API 出口样本程序』。

有关 API 出口调用、外部控制块和关联主题的更多信息，请参阅 API 出口参考。

有关如何编写、编译和配置出口的常规信息，请参阅第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』。

在 API 出口中使用消息句柄

您可以控制 API 出口对哪些消息属性有访问权。 属性与 ExitMsgHandle 相关联。 PUT 出口中的属性集在放入消息时设置，但 GET 出口中检索到的属性不会返回到应用程序。

使用 MQXEP MQI 调用（**Function** 设置为 MQXF_INIT，**ExitReason** 设置为 MQXR_CONNECTION）注册 MQ_INIT_EXIT 出口函数时，作为 **ExitOpts** 参数传入 MQXEPO 结构。 MQXEPO 结构包含 ExitProperties 字段，该字段指定可设置为用于此出口的属性集。 该字段指定为表示属性前缀的字符串，此字符串与 MQRFH2 文件夹名称对应。

每个 API 出口都会收到包含 ExitMsgHandle 字段的 MQAXP 结构。 此字段设置为 IBM MQ 生成的值并特定于连接。 因此，句柄在同一连接上相同类型或不同类型的 API 出口间保持不变。

在 **ExitReason** 为 MQXR_BEFORE 的 MQ_PUT_EXIT 或 MQ_PUT1_EXIT 中（即放入消息之前执行的 API 出口中），出口完成时与 ExitMsgHandle 关联的任何属性（除消息描述符属性外）将在放入消息时进行设置。 为了防止出现此情况，请将 ExitMsgHandle 设置为 MQHM_NONE。 您也可以提供其他消息句柄。

在 MQ_GET_EXIT 和 MQ_CALLBACK_EXIT 中，将清除 ExitMsg 句柄的属性，并在注册 MQ_INIT_EXIT 时使用 ExitProperties 字段中指定的属性 (消息描述符属性除外) 进行填充。 这些属性不可用于获取应用程序。 如果获取应用程序在 MQGMO（获取消息选项）字段中指定了消息句柄，那么与该句柄关联的任何属性（包括消息描述符属性）都可用于 API 出口。 为了防止使用属性填充 ExitMsgHandle，请将其设置为 MQHM_NONE。

注: 对于要处理的出口消息属性:

- 在 MQ_GET_EXIT 函数之后，必须为出口定义 MQ_GET_EXIT 函数之前的函数。
- 在 MQ_CALLBACK_EXIT 函数之前，必须为该出口定义 MQ_CB_EXIT 函数之前的函数。

提供了样本程序 amqsaem0.c 来展示 API 出口中消息句柄的使用。

相关参考

用户出口、API 出口和可安装服务参考

Multi 编译 *API* 出口

在编写出口后，按照如下所示对其进行编译和链接。

以下示例显示了用于 第 977 页的『API 出口样本程序』中描述的样本程序的命令。 对于除 Windows 系统以外的平台，您可以在 *MQ_INSTALLATION_PATH*/samp 中找到样本 API 出口代码，并在 *MQ_INSTALLATION_PATH*/samp/bin 中找到已编译和链接的共享库。

Windows 对于 Windows 系统，您可以在 *MQ_INSTALLATION_PATH* \Tools\c\Samples 中找到样本 API 出口代码。 *MQ_INSTALLATION_PATH* 代表 IBM MQ 安装所在的目录。

注: 64 位平台上的编码标准中列出了 64 位应用程序的编程指南。

对于多点广播客户机，API 出口和数据转换出口必须能够在客户端上运行，这是因为某些消息可能无法通过队列管理器。 以下库属于客户机包和服务器包:

表 145: 位于客户机包和服务器包中的库	
操作系统	**库**
AIX	32 位及 64 位：libmqm.a 和 libmqm_r.a
IBM i	LIBMQM 和 LIBMQM_R
Linux	32 位及 64 位：libmqm.so 和 libmqm_r.so
Solaris	32 位及 64 位：libmqm.so
Windows	32 位及 64 位：mqm.dll 和 mqm.pdb

Linux UNIX 在 *UNIX* 和 *Linux* 系统上编译 *API* 出口

如何在 UNIX and Linux 系统上编译 API 出口的示例。

在所有平台上，模块的入口点都是 MQStart。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

在 AIX 上

AIX

通过发出以下其中一个命令编译 API 出口源代码：

32 位应用程序

非线程化

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

64 位应用程序

非线程化

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

在 Linux 上

Linux

通过发出以下其中一个命令编译 API 出口源代码：

31 位应用程序

非线程化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

32 位应用程序

非线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

64 位应用程序

非线程化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

在 Solaris 上

Solaris

通过发出以下其中一个命令编译 API 出口源代码：

32 位应用程序

SPARC 平台

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

x86-64 平台

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

64 位应用程序

SPARC 平台

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

x86-64 平台

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

Windows 在 *Windows* 系统上编译 *API* 出口
在 Windows 上编译和链接样本 API 出口程序 `amqsaxe0.c`

清单文件是可选的 XML 文档，包含可嵌入已编译应用程序或 DLL 的版本或其他任何信息。

如果您没有此类文本，请忽略 **mt** 命令中的 -manifest *manifest.file* 参数。

调整 第 872 页的图 106 或 第 872 页的图 107 中示例中的命令，以在 Windows 上编译和链接 `amqsaxe0.c`。 该命令可用于 Microsoft Visual Studio 2008、2010 或 2012。 这些示例假定 `C:\Program Files\IBM\MQ\tools\c\samples` 目录是当前目录。

32 位

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

图 *106:* 在 *32* 位 *Windows* 上编译并链接 *amqsaxe0.c*

64 位

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

图 *107:* 在 *64* 位 *Windows* 上编译并链接 *amqsaxe0.c*

相关概念
第 977 页的『API 出口样本程序』
样本 API 出口生成对具有 MQAPI_TRACE_LOGFILE 环境变量中所定义前缀的用户指定的文件的 MQI 跟踪。

IBM i 在 *IBM i* 上编译 *API* 出口
在 IBM i 上编译 API 出口。

按照如下所示创建出口（C 语言示例）：

1. 使用 CRTCMOD 创建一个模块。 对模块进行编译，通过包含 TERASPACE(*YES *TSIFC) 参数以使用太字节空间。
2. 使用 CRTSRVPGM 从模块中创建服务程序。 对于多线程 API 出口，您必须将其绑定到服务程序 QMQM/LIBMQMZF_R。

配置 *API* 出口
配置 IBM MQ 以通过更改配置信息启用 API 出口。

要更改配置信息，您必须更改定义了出口例程及其运行顺序的节。 本信息可以通过以下方式更改：

- Windows Linux 在 Windows 和 Linux (x86 和 x86-64 平台) 上使用 IBM MQ Explorer 。
- Windows 在 Windows 上使用 **amqmdain** 命令。
- Multi 直接在 Windows， IBM i IBM i， 和 UNIX and Linux 系统上使用 `mqs.ini` 和 `qm.ini` 文件。

  `mqs.ini` 文件包含与特定节点上所有队列管理器相关的信息。 可以在以下位置中找到该文件：

  – IBM i 在 IBM i 上的 `/QIBM/UserData/mqm` 目录中。
  – Linux UNIX 在 UNIX and Linux 上的 `/var/mqm` 目录中。
  – Windows 在 Windows 系统上的 `HKLM\SOFTWARE\IBM\WebSphere MQ` 键中指定的 WorkPath 中。

  `qm.ini` 文件包含与特定队列管理器相关的信息。 每个队列管理器都有一个队列管理器配置文件，该文件位于队列管理器所在目录树的根目录中。 例如，队列管理器 QMNAME 的配置文件的路径和名称是：

  IBM i 在 IBM i 系统上：

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  Linux UNIX 在 UNIX and Linux 系统上：

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows 在 Windows 系统上：

  ```
  C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
  ```

  在编辑配置文件之前首先进行备份，这样您就有了一个副本，可在需要时进行恢复。

  可以通过以下任一方式来编辑配置文件：

  – 自动，在节点上使用用于更改队列管理器配置的命令。
  – 手动，使用标准文本编辑器。

  如果您在配置文件属性上设置了不正确的值，此值将被忽略，并会发出操作员消息以指出问题。 这个效果和完全缺少属性是一样的。

要配置的节

下面是必须要更改的节：

ApiExitCommon

在 `mqs.ini` 中以及在 IBM MQ 属性页面的 IBM MQ Explorer 中的 Exits 下定义。

当任何队列管理器启动时，会读取此节中的属性，这些属性随后会被 `qm.ini` 中定义的 API 出口覆盖。

ApiExitTemplate

在 `mqs.ini` 中以及在 IBM MQ 属性页面的 IBM MQ Explorer 中的 Exits 下定义。

当创建任何队列管理器时，本节中的属性将复制到新建 qm.ini 文件的 ApiExitLocal 节下。

ApiExitLocal

在 `qm.ini` 中以及在队列管理器属性页面的 IBM MQ Explorer 中的 Exits 下定义。

当队列管理器启动时，此处定义的 API 出口将覆盖 mqs.ini 中定义的缺省值。

节的属性

- 使用以下属性指定 API 出口：

  Name=ApiExit_name
  传递到 MQAXP 结构中 ExitInfoName 字段的 API 出口的描述性名称。

  此名称必须唯一，长度不能超过 48 个字符，且只包含对于 IBM MQ 对象名称（例如，队列名称）有效的字符。

- 使用以下属性标识要运行的模块和 API 出口代码的入口点：

  Function=function_name
  指向包含 API 出口代码的模块的函数入口点名称。 此入口点是 MQ_INIT_EXIT 函数。

  此字段的长度受限于 MQ_EXIT_NAME_LENGTH。

  Module=module_name
  包含 API 出口代码的模块。

  如果此字段包含模块的完整路径名，那么将照原样使用它。

  如果此字段仅包含模块名，则使用 gm.ini 文件中 ExitPath 的 ExitsDefaultPath 属性来定位模块。

  在支持单独线程化库的平台上，您必须提供 API 出口模块的非线程化版本和线程化版本。 线程化的版本必须有 _r 后缀。 IBM MQ 应用程序存根的线程化版本会在加载给定模块之前将 _r 隐式附加到给定模块名称。

  此字段的长度限制为平台支持的最大路径长度。

- 可选择使用以下属性通过出口传递数据：

  Data=data_name
  要传递到 MQAXP 结构的 ExitData 字段中 API 出口的数据。

  如果您包含此属性，前导和结尾的空白将被移除，而剩余的字符串则被截断为 32 个字符，并且将结果传递给出口。 如果您省略此属性，则向出口传递 32 位空白的缺省值。

  此字段的最大长度是 32 个字符。

- 使用以下属性标识此出口相对于其他出口的顺序：

  Sequence=sequence_number
  调用此 API 出口的顺序与其他 API 出口有关。 先调用序号较低的出口，然后调用序号较高的出口。 出口的序列号不必连续。 1、2、3 顺序与 7、42、1096 顺序的结果相同。 如果两个出口具有相同的序列号，由队列管理器确定首先调用哪一个。 通过在 MQAXP 中 ExitChainAreaPtr 指示的 ExitChainArea 中放入时间或标记或写自己的日志文件，可以在事件后区分调用了哪个出口。

  此属性是无符号的数字值。

样本节

样本 mqs.ini 文件包含以下节：

ApiExitTemplate
此节定义了描述性名称为 OurPayrollQueueAuditor、模块名称为 auditor 且序号为 2 的出口。 将数据值 123 传递到出口。

ApiExitCommon
此节定义了描述性名称为 MQPoliceman、模块名称为 tmqp 且序号为 1 的出口。 传递的数据是一条指令 (CheckEverything)。

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
```

```
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

以下样本 qm.ini 文件包含带有描述性名称 `ClientApplicationAPIchecker`、模块名称 `ClientAppChecker` 和序列号 3 的出口的 ApiExitLocal 定义。

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

消息传递通道的通道出口程序

此主题系列包含有关消息传递通道的 IBM MQ 通道出口程序的信息。

消息通道代理程序 (MCA) 也可以称为数据转换出口。 有关编写数据转换出口的更多信息，请参阅第 893 页的『编写数据转换出口』。

其中某些信息还适用于 MQI 通道上的出口，此通道用于将 IBM MQ MQI clients 连接到队列管理器。 有关更多信息，请参阅 MQI 通道的通道出口程序。

通道出口程序由 MCA 程序在执行处理时在定义的位置调用。

某些用户出口程序以互补对的形式运作。 例如，如果用户出口程序通过发送 MCA 进行调用来加密要传输的消息，那么接收端必须运行互补进程才能逆转此过程。

第 875 页的表 146 中显示了可用于各个通道类型的通道出口类型。

表 *146:* 可用于各个通道类型的通道出口

通道类型	消息出口	消息重试出口	接收出口	安全出口	发送出口	自动定义出口
发送发通道	Yes		Yes	Yes	Yes	
服务器通道	Yes		Yes	Yes	Yes	
集群发送方通道	Yes		Yes	Yes	Yes	Yes
接收方通道	Yes	Yes	Yes	Yes	Yes	Yes
请求者通道	Yes	Yes	Yes	Yes	Yes	
集群接收方通道	Yes	Yes	Yes	Yes	Yes	Yes
客户机连接通道			Yes	Yes	Yes	
服务器连接通道			Yes	Yes	Yes	Yes

注意: z/OS

1. 在 z/OS 上，自动定义出口只适用于集群发送方和集群接收方通道。

如果您要在客户机上运行通道出口，不能使用 MQSERVER 环境变量。 而是要按照客户机通道定义表中所述创建和引用客户机通道定义表 (CCDT)。

处理概述

MCA 如何使用通道出口程序的概述。

在启动时，MCA 会交换启动对话框以同步处理。 然后切换到包含安全出口的数据交换。 这些出口必须成功结束才能完成启动阶段以允许传输消息。

如第 876 页的图 108 所示，安全性检查阶段是一个循环。

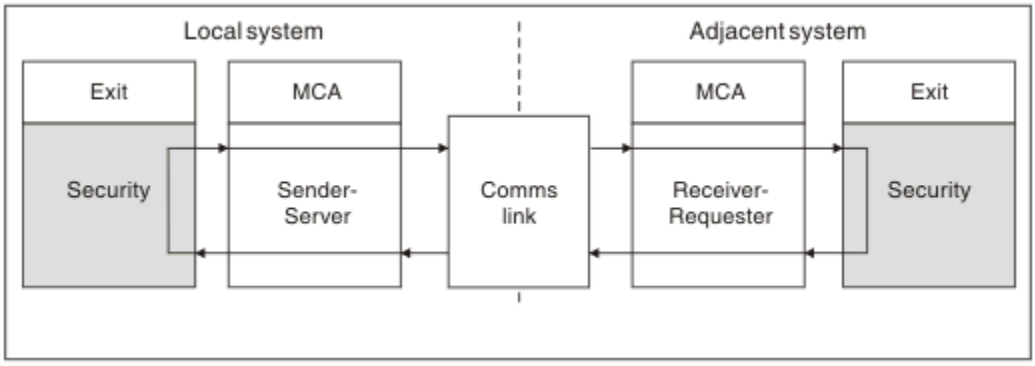


图 *108:* 安全出口循环

如第 876 页的图 109 所示，在消息传输阶段，发送 MCA 先从传输队列获取消息，之后调用消息出口，然后再调用发送出口，最后将消息发送到接收 MCA。

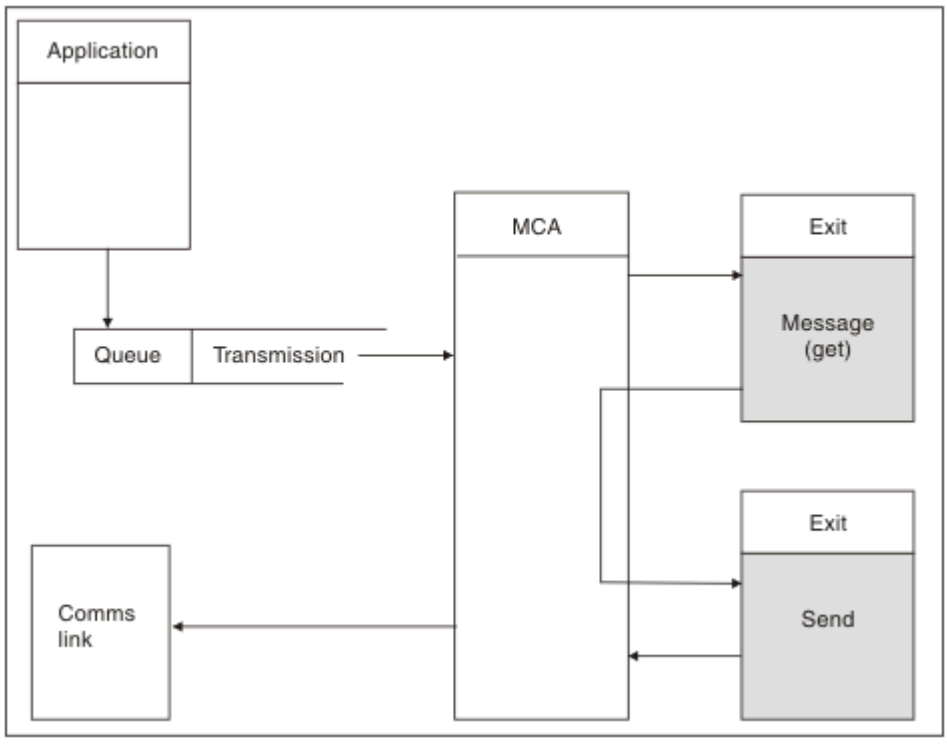


图 *109:* 消息通道的发送方端的发送出口示例

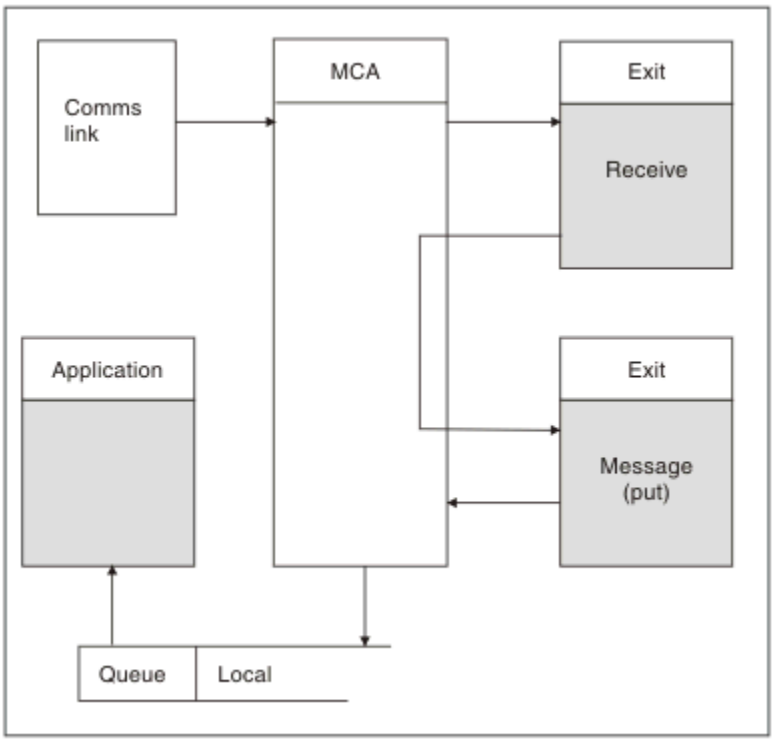


图 *110:* 消息通道的接收方端的接收出口示例

如第 877 页的图 110 所示，接收 MCA 先从通信链路接收消息，之后调用接收出口，然后再调用消息出口，最后将消息放入本地队列。（调用消息出口之前可以多次调用接收出口。）

编写通道出口程序

以下信息可帮助您编写通道出口程序。

除了后续部分特别注明的情况外，用户出口和通道出口程序可以使用所有 MQI 调用。 对于 MQ V7 和更高版本，MQCXP 结构 V7 和更高版本包含连接句柄 hConn，该句柄可用来代替发出 MQCONN。 对于较低版本，要获取连接句柄，必须发出 MQCONN，即使因为通道本身已连接到队列管理器而返回 MQRC_ALREADY_CONNECTED 警告也是如此。

请注意，通道出口必须保证线程安全。

对于客户机连接通道上的出口，该出口尝试连接的队列管理器取决于链接此出口的方式。 如果已将出口链接至 MQM.LIB （或 IBM i 上的 QMQM/LIBMQM）并且您没有在 MQCONN 调用上指定队列管理器名称，出口将尝试连接到系统上的缺省队列管理器。 如果已将出口链接至 MQM.LIB （或 IBM i 上的 QMQM/LIBMQM），并且您指定通过 MQCD 的 QMgrName 字段传递到出口的队列管理器名称，那么出口将尝试连接到该队列管理器。 如果已将出口链接至 MQIC.LIB 或其他任何库，那么无论您是否指定队列管理器名称，MQCONN 调用都会失败。

您应该避免更改与通道出口中传递的 hConn 关联的事务状态；不得对该通道 hConn 使用 MQCMIT、MQBACK 或 MQDISC 动词，并且不能使用 MQBEGIN 动词指定通道 hConn。

如果使用 MQCONNX 并指定 MQCNO_HANDLE_SHARE_BLOCK 或 MQCNO_HANDLE_SHARE_NO_BLOCK 来新建 IBM MQ 连接，那么您有责任保证正确管理此连接并正确地与队列管理器断开连接。 例如，在不断开连接的情况下，通道出口会在每次调用时新建与队列管理器的连接，从而导致连接句柄累积和代理程序线程数量增加。

出口会与 MCA 本身在同一线程内运行并使用相同的连接句柄。 因此，出口会与 MAC 在同一工作单元中运行，在同步点下进行的任何调用都由批处理端的通道落实或回退。

因此，当落实包含原始消息的批处理时，通道消息出口可能会只发送落实到该队列的通知消息。 因此，有可能从通道消息出口发出同步点 MQI 调用。

通道出口可以更改 MQCD 中的字段。 但除了列出的情况，不会实行这些更改。 如果通道出口程序更改 MQCD 数据结构中的字段，IBM MQ 通道进程将忽略新值。 但是，新值仍保留在 MQCD 中并传递到出口链中的所有剩余出口，同时会传递到任何共享此通道实例的对话中。 有关更多信息，请参阅更改通道出口中的 MQCD 字段

此外，对于以 C 语言编写的程序，通道出口程序中不得使用非重入 C 库函数。

Linux UNIX 如果您同时使用多个通道出口库，如果两个不同出口的代码包含名称相同的函数，某些 UNIX and Linux 平台上可能会出现问题。 装入通道出口时，动态装入程序会将出口库中的函数名解析到装入此库的地址。 如果两个出口库定义了正好具有相同名称的不同函数，此解析过程可能会将一个库的函数名错误地解析为使用另一个库的函数。 如果发生此问题，请指示链接程序只能导出需要的出口和 MQStart 函数，因为这些函数不受影响。 必须为其他函数提供本地可视性，这样这些函数才不会被自己的出口库之外的函数使用。 请查看链接程序文档以获取更多信息。

所有出口的调用都带有通道出口参数结构 (MQCXP)、通道定义结构 (MQCD)、准备的数据缓冲区、数据长度参数和缓冲区长度参数。 不得超过缓冲区长度：

- 对于消息出口，您必须考虑需要在通道间发送的最大消息，外加 MQXQH 结构的长度。
- 对于发送和接收出口，您必须考虑到的最大缓冲区如下：

  LU 6.2
  32 KB

  TCP：

  IBM i IBM i 16 KB

  IBM i 其他 32 KB

  注： 最大可用长度可能比此长度小 2 字节。 请查看 MaxSegmentLength 中返回的值以了解详细信息。 有关 MaxSegmentLength 的更多信息，请参阅 MaxSegmentLength。

  NetBIOS:
  64KB

  SPX：
  64KB

  注： 发送方通道上的接收出口和接收方通道上的发送方出口针对 TCP 使用 2 KB 缓冲区。
- 对于安全出口，分布式排队工具为缓冲区分配了 4000 个字节。

容许出口随相关参数一起返回替代缓冲区。 请参阅第 875 页的『消息传递通道的通道出口程序』获取调用详细信息。

z/OS 在 *z/OS* 上编写通道出口程序

以下信息可帮助您为 z/OS 编写和编译通道出口程序。

在以下情况下，出口就像通过 z/OS 链接启动一样进行启动：

- 未授权的问题程序状态
- 主地址空间控制方式
- 非交叉内存模式
- 非访问注册方式
- 31 位寻址方式

链接编辑的模块必须放在由通道启动程序地址空间过程的 CSQXLIB DD 语句指定的数据集中；装入模块的名称指定为通道定义中的出口名称。

为 z/OS 编写通道出口时，需循序以下规则：

- 必须使用汇编语言或 C 语言来编写出口；如果使用的是 C 语言，那么它必须符合系统出口的 C 系统编程环境，如 z/OS C/C++ 编程指南中所述。
- 出口从由 CSQXLIB DD 语句定义的非授权库中装入。 如果 CSQXLIB 具有 DISP=SHR，那么出口可以在通道启动程序运行时更新。 重新启动通道时将使用新版本。

- 出口必须是可重入的，并且能够在虚拟存储器中的任何地方运行。
- 返回时，出口必须将环境重置为入口处的环境。
- 出口必须释放获取的所有存储空间，或者确保由后续出口调用释放。

  对于调用之间持久存在的存储空间，请使用 z/OS STORAGE 服务或针对系统编程 C 的 4kmalc 库函数。

  有关此功能的更多信息，请参阅 4kmalc() -- 分配页面对齐的存储。
- 除了 MQCMIT 或 CSQBCMT 以及 MQBACK 或 CSQBBAK 外，所有 IBM MQ MQI 出口都可以使用。 这些出口必须包含在 MQCONN 后（具有空白的队列管理器名称）。 如果使用这些调用，出口必须经过链接编辑并带有存根 CSQXSTUB。

  此规则的例外情况是安全通道出口可以发出落实或回退 MQI 调用。 要发出此类调用，请对动词 CSQXCMT 和 CSQXBAK 编码，以取代 MQCMIT 或 CSQBCMT 以及 MQBACK 或 CSQBBAK。
- 所有使用来自 IBM WebSphere MQ 7.0 或更高版本的存根 CSQXSTUB 的出口都必须在 CSQXLIB 装入库中以 PDS-E 格式编辑链接。
- 出口不得使用任何会导致等待的系统服务，因为使用此类系统服务将严重影响部分或所有其他通道的处理。 很多通道通常在单个 TCB 下运行。 如果在出口中执行某些会导致等待的操作并且不使用 MQXWAIT，将导致这些所有通道都等待。 使通道等待不会导致任何功能问题，但可能会对性能造成负面影响。 除以下 SVC 外，多数 SVC 涉及等待，因此您必须避免使用这些 SVC：
  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

  因此，通常应避免使用 SVC、PC 和 I/O。 请改为使用 MQXWAIT 调用。
- 除了在出口连接的子任务中，出口不会发出 ESTAE 或 SPIE，因为它们的错误处理可能会干扰 IBM MQ 执行的错误处理。 这意味着 IBM MQ 可能无法从错误中恢复，或者您的出口程序可能收不到所有错误信息。
- MQXWAIT 调用（请参阅 MQXWAIT）提供了可等待 I/O 和其他事件的等待服务；如果使用此服务，那么出口不得使用链接堆栈。

  对于不提供要等待的非阻止性工具或 ECB 的 I/O 和其他工具，必须连接单独的子任务，且通过 MQXWAIT 等待其完成；由于此方法造成的处理的原因，此工具只能用于安全出口。

- MQDISC MQI 调用不会导致出口程序中出现隐式落实。 落实通道进程只会在通道协议控制下执行。

以下出口样本随 IBM MQ for z/OS 一起提供：

CSQ4BAX0
此样本使用汇编程序编写，并展示了 MQXWAIT 的使用。

CSQ4BCX1 和 CSQ4BCX2
这些样本以 C 语言编写，展示了如何访问参数。

CSQ4BCX3 和 CSQ4BAX3
这些样本分别使用 C 语言和汇编程序编写。

CSQ4BCX3 样本（已预编译到 SCSQAUTH LOADLIB）应该无需对出口本身进行任何更改便可运作。 您可以创建 LOADLIB（例如，称为 MY.TEST.LOADLIB）并将 SCSQAUTH(CSQ4BCX3) 成员复制到其中。

要在客户机连接上设置安全出口，请执行以下过程：

1. 为通道启动程序使用的用户标识建立有效的 OMVS 分段。

   这允许 IBM MQ for z/OS 通道启动程序对 UNIX 系统服务 (USS) 套接字接口使用 TCP/IP，进而推动出口处理。 请注意，不需要为任何连接客户机的用户标识定义 OMVS 分段。
2. 请确保出口代码本身只在程序控制的环境中运行。

   这意味着装入到 CHINIT 地址空间的所有内容都必须从程序控制的库（即 STEPLIB 中的所有库）中装入，以及从 CSQXLIB 上指定的任何库和下列库装入：

```
++hlq++.SCSQANLx
++hlq++.SCSQMVR1
++hlq++.SCSQAUTH
```

要将装入库设置为由程序控制，请使用类似于以下示例的命令：

```
RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
```

然后可以通过发出以下命令来激活或刷新程序控制的环境：

```
SETROPTS WHEN(PROGRAM) REFRESH
```

3. 通过发出以下命令将出口 LOADLIB 添加到 CSQXLIB DD（在 CHINIT 启动的过程中）：

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

此命令将为指定通道激活出口。

4. 您的外部安全性管理器 (ESM) 列出了要由程序控制的其他所有库，但请注意，没有任何 ESM 或 C 库需要受程序控制。

请参阅 IBM MQ for z/OS 服务器连接通道以获取有关使用样本 CSQ4BCX3 设置安全出口的更多信息。

CSQ4BCX4

此样本以 C 语言编写并展示了使用 MQCXP 中的 **RemoteProduct** 和 **RemoteVersion** 字段。

相关概念

IBM MQ for z/OS 服务器连接通道

第 880 页的『在 IBM i 上编写通道出口程序』
以下信息可帮助您为 IBM i 编写和编译通道出口程序。

第 881 页的『在 UNIX, Linux, and Windows 上编写通道出口程序』
以下信息可帮助您为 UNIX, Linux, and Windows 系统编写通道出口程序。

IBM i 在 *IBM i* 上编写通道出口程序

以下信息可帮助您为 IBM i 编写和编译通道出口程序。

出口是使用 ILE C、ILE RPG 或 ILE COBOL 语言编写的程序对象。 出口程序名称及其库在通道定义内指定。

创建和编译出口程序时请遵循以下条件：

- 程序必须线程安全且使用 ILE C、ILE RPG 或 ILE COBOL 编译器创建。 对于 ILE RPG，您必须指定 THREAD(*SERIALIZE) 控制规范，对于 ILE COBOL，您必须为 PROCESS 语句的 THREAD 选项指定 SERIALIZE。 这些程序还必须绑定到线程化 IBM MQ 库：如果是 ILE C 和 ILE RPG，则绑定到 QMQM/LIBMQM_R，如果是 ILE COBOL，则绑定到 AMQ0STUB_R。 有关使 RPG 或 COBOL 应用程序线程安全的其他信息，请参阅相应语言的程序员指南。
- IBM MQ for IBM i 要求启用出口程序以获取太字节空间支持。 （太字节空间是 OS/400 V4R4 中引入的一种形式的共享内存。） 对于 ILE RPG 和 COBOL 编译器，OS/400 V4R4 或更高版本上编译的任何程序都是这样启用。 对于 C 语言，程序必须使用 CRTCMOD 或 CRTBNDC 命令上指定的 TERASPACE(*YES *TSIFC) 选项编译。
- 将指针返回到其自己的缓冲区空间的出口必须确保：指向的对象的存在时间超出通道出口程序的时间范围。 指针不能是程序堆栈上的变量地址，也不能是程序堆中的变量。 相反，指针必须从系统获取。 例如用户出口中创建的用户空间。 为了确保通道出口程序分配的任何数据区域在程序结束时仍可用于 MCA，必须在调用者的激活组或指定激活组中运行通道出口。 通过将 CRTPGM 上的 ACTGRP 参数设置为用户定义的值或设置为 *CALLER 可以实现此目的。 如果使用这种方式创建程序，通道出口程序可以分配动态内存并将指向此内存的指针传递回 MCA。

相关概念

第 881 页的『在 UNIX, Linux, and Windows 上编写通道出口程序』
以下信息可帮助您为 UNIX, Linux, and Windows 系统编写通道出口程序。

第 878 页的『在 z/OS 上编写通道出口程序』
以下信息可帮助您为 z/OS 编写和编译通道出口程序。

ULW在 *UNIX, Linux, and Windows* 上编写通道出口程序
以下信息可帮助您为 UNIX, Linux, and Windows 系统编写通道出口程序。

请遵循第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』中概述的指示信息。 请在适用时使用以下特定通道出口信息：

出口必须以 C 语言编写，并且在 Windows 上为 DLL。

在出口中定义虚拟 MQStart() 例程并在库中指定 `MQStart` 作为入口点。 第 881 页的图 111 显示了如何设置程序的入口：

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

图 *111:* 通道出口的样本源代码

使用 Visual C++ 为 Windows 编写通道出口时，必须编写自己的 DEF 文件。 第 881 页的图 112 中显示了如何编写通道出口的示例。 有关编写通道出口程序的更多信息，请参阅第 877 页的『编写通道出口程序』。

```
EXPORTS
ChannelExit
```

图 *112: Windows* 的样本 *DEF* 文件

相关概念
第 880 页的『在 IBM i 上编写通道出口程序』
以下信息可帮助您为 IBM i 编写和编译通道出口程序。

第 878 页的『在 z/OS 上编写通道出口程序』
以下信息可帮助您为 z/OS 编写和编译通道出口程序。

通道安全出口程序
您可以使用安全出口程序来验证通道另一端的合作伙伴是否真实。 这称为认证。 要指定某个通道必须使用安全出口，请在通道定义的 SCYEXIT 字段中指定出口名称。

注: 认证也可以通过通道认证记录完成。 通道认证记录在阻止特定用户和通道访问队列管理器方面，以及将远程用户映射到 IBM MQ 用户标识方面提供了很大的灵活性。 IBM MQ 还提供了 TLS 支持来认证您的用户并为您的数据提供加密和数据完整性检查。 有关 TLS 的更多信息，请参阅 IBM MQ 中的 TLS 安全协议。 但是，如果您仍需要更复杂（或其他）形式的安全处理以及其他类型的检查和安全上下文建立，请考虑编写安全出口。

对于在 IBM WebSphere MQ 7.1 之前编写的安全出口，需要注意 IBM MQ 的较低版本会查询底层安全套接字提供者（例如，GSKit）以确定远程合作伙伴的证书主题专有名称 (SSLPEER) 和颁发者专有名称 (SSLCERTI)。 在 IBM WebSphere MQ 7.1 中，为一系列新的安全属性添加了支持。 为了访问这些属性，IBM WebSphere MQ 7.1 将获取证书的 DER 编码并使用此编码确定主题和颁发者 DN。 主题和颁发者 DN 属性显示在以下通道状态属性中：

- SSLPEER（PCF 选择器 MQCACH_SSL_SHORT_PEER_NAME）
- SSLCERTI（PCF 选择器 MQCACH_SSL_CERT_ISSUER_NAME）

这些值由通道状态命令返回，并且会列出传递到通道安全出口的数据，如下所示：

- MQCD SSLPeerNamePtr

- MQCXP SSLRemCertIssNamePtr

在 IBM WebSphere MQ 7.1 中，SERIALNUMBER 属性还包含在主题 DN 中并包含远程合作伙伴证书的序列号。 此外，某些 DN 属性的返回顺序与先前发行版的返回顺序不同。 因此，与先前发行版相比，SSLPEER 和 SSLCERTI 字段的构成在 IBM WebSphere MQ 7.1 中已发生变更，因此建议您检查所有安全出口或依赖于这些字段的应用程序并进行更新。

通过通道定义的 SSLPEER 字段指定的现有 IBM MQ 对等名称过滤器不受影响，将继续以与先前发行版相同的方式运作。 这是因为 IBM MQ 对等名称匹配算法已经过更新，无需变更通道定义便可处理现有 SSLPEER 过滤器。 此更改最可能会影响依赖于 PCF 编程接口返回的主题 DN 和颁发者 DN 值的安全出口和应用程序。

安全出口可以使用 C 语言或 Java 编写。

在 MCA 的处理周期中，通道安全出口程序在以下位置调用：

- MCA 启动和终止时。
- 通道启动时的初始数据协商完成后立即调用。 通过提供要发送到远端的安全出口的消息，通道的接收方端或服务器端可以启动与远端的安全消息交换。 此交换也可能会遭到拒绝。 出口程序将再次启动以处理从远端接收的任何安全消息。
- 通道启动时的初始数据协商完成后立即调用。 通道的发送方端或请求方端处理从远端接收的安全消息，或在远端无法发送时启动安全交换。 出口程序将再次启动以处理可能会收到的所有后续安全消息。

永不会使用 MQXR_INIT_SEC 调用请求者通道。 通道先通知服务器具有安全出口程序，然后服务器才有可能启动安全出口。 如果通道没有安全出口程序，它会通知请求者并向出口程序返回零长度流。

注: 避免发送零长度安全消息。

图 第 883 页的图 113 到 第 885 页的图 116 展示了由安全出口程序交换的数据示例。 这些示例显示了涉及接受方的安全出口以及发送方的安全出口时事件的发生顺序。 图中的连续行表示时间段。 在某些情况下，接收方和发送方的事件不相关，因此可能会同时发生，也可能在不同时间发生。 其他情况下，一个出口程序上的事件会导致另一个出口程序上稍后发生互补事件。 例如，在 第 883 页的图 113 中：

1. 使用 MQXR_INIT 分别调用接收方和发送方，但这些调用不相关，因此可以同时发生，也可以不同时发生。
2. 接下来使用 MQXR_INIT_SEC 调用接收方，但会返回不需要发送方出口发生互补事件的 MQXCC_OK。
3. 接下来使用 MQXR_INIT_SEC 调用发送方。 此调用与使用 MQXR_INIT_SEC 的接收方调用不相关。 发送方返回 MQXCC_SEND_SEC_MSG，将导致接收方出口上发生互补事件。
4. 然后，使用 MQXR_SEC_MSG 调用接收方，将返回导致发送方出口发生互补事件的 MQXCC_SEND_SEC_MSG。
5. 然后使用 MQXR_SEC_MSG 调用发送方，将返回不需要接收方出口发生互补事件的 MQXCC_OK。

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Message transfer begins	

图 *113:* 发送方发起的使用协议交换

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

图 *114:* 发送方发起的无协议交换

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

图 *115:* 接收方发起的协议交换

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

图 *116:* 接收方发起的无协议交换

将向通道安全出口程序传递包含安全数据的代理程序缓冲区，不包括安全出口生成的任何传输头。 此数据可以是任何合适的数据，以便通道的任何一端能执行安全验证。

消息通道的发送端和接收端上的安全出口程序可以将两个响应代码中的任意一个返回到任何调用：

- 安全交换无错误结束
- 禁止并关闭通道

注：

1. 通道安全出口通常成对工作。 在定义相应的通道时，请确保为通道两端指定兼容的出口程序。
2. IBM i 在 IBM i 中，使用 `Use adopted authority` (USEADPAUT=*YES) 编译过的安全出口程序可以采用 QMQM 或 QMQMADM 授权。 请注意，该出口不使用此功能，以免对您的系统构成安全风险。
3. 在 TLS 通道上（通道的另一端提供证书），安全出口接收通过 SSLPeerNamePtr 访问的 MQCD 字段中此证书主题的专有名称以及通过 SSLRemCertIssNamePtr 访问的 MQCXP 字段中颁发者的专有名称。 使用此名称可以：
   - 限制通过 TLS 通道进行访问。
   - 根据此名称更改 MQCD.MCAUserIdentifier。

相关概念

通道认证记录

传输层安全性 (TLS) 概念

编写安全出口

您可以使用安全出口框架代码编写安全出口。

第 886 页的图 117 展示了如何编写安全出口。

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

图 *117:* 安全出口框架代码

标准 IBM MQ 入口点 MQStart 必须存在，但对于执行任何函数并非必需项。 函数的名称（此示例中为 EntryPoint）可以更改，但编译和链接库时必须导出函数。 如以上示例所示，指针 pChannelExitParms 必须转换为 PMQCXP，而 pChannelDefinition 必须转换为 PMQCD。 有关调用通道出口和参数使用的常规信息，请参阅 MQ_CHANNEL_EXIT。 这些参数用于安全出口，具体如下所示：

PMQVOID pChannelExitParms
输入/输出

指向 MQCXP 结构的指针 - 转换为 PMQCXP 才能访问字段。 此结构用于出口和 MCA 之间的通信。 MQCXP 中的以下字段特别关注安全出口：

ExitReason
告知安全出口安全交换中的当前状态，并用于决定执行哪种操作。

ExitResponse
对 MAC 的响应，用于决定安全交换的下一个阶段。

ExitResponse2
额外的控制标志，用于管理 MCA 如何理解安全出口的响应。

ExitUserArea
16 字节（最大）的存储空间，可由安全出口用于维护调用之间的状态。

ExitData
包含通道定义的 SCYDATA 字段内指定的数据（填充到右侧空白处的 32 个字节）。

PMQVOID pChannelDefinition
输入/输出

指向 MQCD 结构的指针 - 转换为 PMQCD 才能访问字段。 此参数包含通道的定义。 MQCD 中的以下字段特别关注安全出口：

ChannelName
通道名称（填充到右侧空白处的 20 个字节）。

ChannelType
定义通道类型的代码。

MCA 用户标识
这三个字段组初始化为通道定义中指定的 MCAUSER 字段的值。 安全出口在这些字段中指定的任何用户标识均用于访问控制（不适用于 SDR、SVR、CLNTCONN 或 CLUSSDR 通道）。

MCAUserIdentifier
填充到右侧空白处的标识符的前 12 个字节。

LongMCAUserIdPtr
指向包含完整长度标识（不保证以空值结束）的缓冲区的指针，优先于 MCAUserIdentifier。

LongMCAUserIdLength
LongMCAUserIdPtr 指向的字符串的长度 - 如果设置了 LongMCAUserIdPtr，则必须设置此参数。

远程用户标识
仅适用于 CLNTCONN/SVRCONN 通道对。 如果没有定义 CLNTCONN 安全出口，那么这三个字段将由客户机 MCA 初始化，因此它们可能包含客户机环境中的用户标识，SVRCONN 安全出口可使用此用户标识进行认证，指定 MCA 用户标识时也可使用此用户标识。 如果定义了 CLNTCONN 安全出口，那么这些字段不会初始化，并可以通过 CLNTCONN 安全出口进行设置，或者安全消息可用于将用户标识从客户机传递到服务器。

远程用户标识
填充到右侧空白处的标识符的前 12 个字节。

LongRemoteUserIdPtr
指向包含完整长度标识（不保证以空值结束）的缓冲区的指针，优先于 RemoteUserIdentifier。

LongRemoteUserIdLength
LongRemoteUserIdPtr 指向的字符串的长度 - 如果设置了 LongRemoteUserIdPtr，则必须设置此参数。

PMQLONG pDataLength
输入/输出

指向 MQLONG 的指针。 包含调用安全出口时 AgentBuffer 内所含的任何安全出口的长度。 必须由安全出口设置为 AgentBuffer 或 ExitBuffer 中正在发送的任何消息的长度。

PMQLONG pAgentBufferLength
输入

指向 MQLONG 的指针。 调用安全出口时 AgentBuffer 中所含数据的长度。

PMQVOID pAgentBuffer
输入/输出

在调用安全出口时，此参数指向任何从合作伙伴出口发送的消息。 如果 MQCXP 结构中的 ExitResponse2 具有 MQXR2_USE_AGENT_BUFFER 标志设置（缺省值），那么安全出口需要将此参数设置为指向正在发送的任何消息数据。

PMQLONG pExitBufferLength
输入/输出

指向 MQLONG 的指针。 此参数在第一次调用安全出口时初始化为 0，并且返回的值会在安全交换期间的安全出口调用间保留。

PMQPTR pExitBufferAddr
输入/输出

此参数在第一次调用安全出口时初始化为空指针，并且返回的值会在安全交换期间的安全出口调用间保留。 如果 MQCXP 结构的 ExitResponse2 中设置了 MQXR2_USE_AGENT_BUFFER 标志，那么安全出口需要将此参数设置为指向正在发送的任何消息数据。

CLNTCONN/SVRCONN 通道对和其他通道对上定义的安全出口之间的行为差异
所有类型的通道上都可以定义安全出口。 但是，CLNTCONN/SVRCONN 通道对上定义的安全出口的行为与其他通道对上定义的安全出口的行为稍有不同。

CLNTCONN 通道上的安全出口可以设置通道定义中的远程用户标识以供合作伙伴 SVRCONN 出口处理，或者如果没有定义 SVRCONN 安全出口并且没有设置 SVRCONN 的 MCAUSER 字段，可以使用此远程用户标识进行 OAM 授权。

如果没有定义 CLNTCONN 安全出口，通道定义中的远程用户标识将由客户机 MCA 设置为客户机环境（可以为空）中的用户标识。

SVRCONN 安全出口返回 MQXCC_OK 的 ExitResponse 时，说明 CLNTCONN 和 SVRCONN 通道对上定义的安全出口之间的安全交换成功完成。 当启动交换的安全出口返回 MQXCC_OK 的 ExitResponse 时，说明其他通道对之间的安全交换成功完成。

但是，在以下情况下，MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse 代码可用于强制继续进行安全交换：如果 CLNTCONN 或 SVRCONN 安全出口返回 MQXCC_SEND_AND_REQUEST_SEC_MSG 的 ExitResponse，那么合作伙伴出口必须发送安全消息（不是 MQXCC_OK 或空响应）或通道终止来进行响应。 对于其他类型的通道上定义的安全出口，为响应合作伙伴安全出口的 MQXCC_SEND_AND_REQUEST_SEC_MSG 而返回的 MQXCC_OK 的 ExitResponse 将导致继续进行安全交换，如同返回了空响应且空响应不在通道终止中一样。

SSPI 安全出口
IBM MQ for Windows 提供了一个安全出口，此出口通过使用安全服务编程接口为 IBM MQ 通道提供认证。 SSPI 提供了 Windows 的集成安全工具。

此安全出口用于 IBM MQ 客户机和 IBM MQ 服务器。

安全数据包从 security.dll 或 secur32.dll 加载。 这些 DLL 随操作系统一起提供。

Windows 使用 NTLM 认证服务提供单向认证。 Windows 2000 使用 Kerberos 认证服务提供双向认证。

安全出口程序以源代码和对象格式提供。 您可以照原样使用对象代码，或者使用源代码作为起点来创建您自己的用户出口程序。 有关使用 SSPI 安全出口的对象或源代码的更多信息，请参阅第 1033 页的『在 Windows 上使用 SSPI 安全出口』

通道发送和接收出口程序
您可以使用发送和接收出口执行数据压缩和解压之类的任务。 您可以指定一列要连续运行的发送和接收出口程序。

在 MCA 的处理周期中，通道发送和接收出口程序在以下位置调用：

- 分别在 MCA 启动和终止时为初始化和终止调用发送和接收出口程序。
- 发送出口程序在通道一端或另一端调用，具体取决于在紧接通过链接发送传输前，在哪一端为一个消息传送发送传输。 注释 4 解释了为什么即使消息通道只在一个方向发送消息，而出口可用于两个方向。
- 接收出口程序在通道一端或另一端调用，具体取决于紧接着从链接获取传输后，从哪一端为一个消息传送接收传输。 注释 4 解释了为什么即使消息通道只在一个方向发送消息，而出口可用于两个方向。

一个消息传送可能有多个传输，并且消息到达接收端的消息出口前，发送和接收出口程序可能有多次迭代。

从通信链路发送或接收传输数据时，将向通道发送和接收出口程序传递包含传输数据的代理程序缓冲区。 对于发送出口程序，缓冲区的前 8 个字符保留为供 MCA 使用，不得进行修改。 如果程序返回其他缓冲区，那么这前 8 个字节必须存在于新缓冲区中。 未定义呈现至出口程序的数据格式。

正确的响应代码必须由发送和接收出口程序返回。 其他任何响应都会导致 MCA 异常结束（中止）。

注: 请不要在发送或接收出口的同步点内发出 MQGET、MQPUT 或 MQPUT1 调用。

注:

1. 发送和接收出口通常成对工作。 例如，发送出口可以压缩数据，而接收出口可以解压数据；或者发送出口可以加密数据，接收出口可以解密数据。 在定义相应的通道时，请确保为通道两端指定兼容的出口程序。
2. 如果为通道启用压缩，将向出口传递压缩数据。
3. 可能会为应用程序数据之外的消息分段（例如，状态消息）调用通道发送和接收出口。 启动对话框和安全检查阶段不会调用这些出口。
4. 尽管消息通道只在一个方向发送消息，但通道控制数据（例如，心跳和批处理结束）在两个方向流动，并且这些出口也在两个方向均可用。 但是，某些初始通道启动数据流会被任何出口免除处理。
5. 在某些情况下，发送和接收出口可以不按顺序调用；例如，如果您正在运行一系列出口程序或者您同时在运行安全出口。 那么，当第一次调用接收出口来处理数据时，该出口可能会收到未通过对应发送出口传递的数据。 如果接收出口在没有先检查是否需要某操作的情况下直接执行了该操作（例如，解压），可能会出现非预期的结果。

   您需要通过特定方式对您的发送和接收出口进行编码，以便接收出口可以检查它要接收的数据是否已由对应发送出口进行处理。 实现此目标的建议方式是对出口程序进行编码，以便:

   - 在执行操作前，发送出口将数据的第九个字节的值设置为 0 并每 1 字节依次变换所有数据。 （前 8 个字节保留为供 MCA 使用。）
   - 如果接收出口收到第 9 个字节为 0 的数据，便知道该数据来自发送出口。 接收出口将移除 0 并执行互补运算，然后每 1 字节依次变换回结果数据。
   - 如果接收出口收到第 9 个字节不是 0 的数据，该出口将假设发送出口未运行并将数据按照原样发回给调用者。

   在使用安全出口时，如果通道由安全出口结束，可能是在没有对应接收出口的情况下调用了发送出口。 下面的方法可防止此问题的发生：对安全出口进行编码，以在 MQCD.SecurityUserData 或 MQCD.SendUserData 中设置一个标志，例如，在出口决定结束通道时。 然后，发送出口需要检查此字段，并只在未设置此标志的情况下处理数据。 此检查可阻止发送出口不必要地变更数据，从而防止在安全出口收到变更数据时可能会出现的任何转换错误。

通道发送出口程序 - 保留空间
传输之前，您可以使用发送和接收出口转换数据。 通过在传输缓冲区内保留空间，通道发送出口程序可以添加自己的有关转换的数据。

此数据由接收出口程序处理，然后从缓冲区中移除。 例如，您可能希望对数据进行加密并添加用于解密的安全密钥。

如何保留和使用空间

为初始化调用发送出口程序时，请将 MQXCP 的 *ExitSpace* 字段设置为要保留的字节数。 有关详细信息，请参阅 MQCXP。 *ExitSpace* 只能在初始化期间设置，即 *ExitReason* 的值为 MQXR_INIT 时。 紧接着传输之前调用发送出口时，*ExitReason* 设置为 MQXR_XMIT，将在传输缓冲区中保留 *ExitSpace* 个字节。 *ExitSpace* 在 z/OS 上不受支持。

发送出口不需要使用所有保留空间。 该出口使用的空间可以少于 *ExitSpace* 字节，或者如果传输缓冲区未满，该出口使用的空间可以大于保留量。 在设置 *ExitSpace* 的值时，您必须至少为传输缓冲区中的消息数据保留 1KB。 如果保留空间用于大量数据，通道性能可能会受到影响。

传输缓冲区通常为 32KB 长。 但是，如果通道使用 TLS，那么传输缓冲区大小将减少到 15,352 字节以适应 RFC 6101 定义的最大记录长度和一系列相关的 TLS 标准。 另外 1024 字节保留为供 IBM MQ 使用，因此发送出口可以使用的最大传输缓冲区空间为 14,328 字节。

通道接收端会发生什么

必须对通道接收出口程序进行设置以与对应的发送出口兼容。 接收出口必须知道保留空间中的字节数并且必须移除该空间内的数据。

多个发送出口

您可以指定一列要连续运行的发送和接收出口程序。 IBM MQ 保留所有发送出口保留的空间总量。 此空间总量必须至少为传输缓冲区中的消息数据保留 1 KB。

以下示例显示了如何为三个连续调用的发送出口分配空间：

1. 为初始化调用时：
   - 发送出口 A 保留 1 KB。
   - 发送出口 B 保留 2 KB。
   - 发送出口 C 保留 3 KB。
2. 最大传输大小为 32 KB，用户数据长度为 5 KB。
3. 使用 5 KB 的数据调用出口 A；最多有 27 KB 的可用空间，因为要为出口 B 和 C 保留 5 KB 的空间。 出口 A 添加 1 KB 的空间（其保留的空间）。
4. 使用 6 KB 的数据调用出口 B；最多有 29 KB 的可用空间，因为要为出口 C 保留 3 KB 的空间。 出口 B 添加 1 KB 的空间（小于其保留的 2 KB 空间）。
5. 调用出口 C 使用 7 KB 数据；最多有 32 KB 可用。 出口 C 占用 10K，大于其保留的 3 KB。 此数量有效，因为数据总量 17 KB 小于 32 KB 最大值。

使用 TLS 的通道的最大传输缓冲区大小为 15,352 字节，而不是 32Kb。 这是因为底层安全套接字传输段限制为 16Kb，而其中某些空间必须用于 TLS 记录开销。 另外 1024 字节保留为供 IBM MQ 使用，因此发送出口可以使用的最大传输缓冲区空间为 14,328 字节。

通道消息出口程序
通道消息出口可用于执行诸如对链接加密、验证或替换入局用户标识、消息数据转换、日志记录和参考消息处理之类的任务。 您可以指定一列要连续运行的消息出口程序。

在 MCA 的处理周期中，通道消息出口程序在以下位置调用：

- MCA 启动和终止时
- 紧接着发送 MCA 发出 MQGET 调用后
- 接收 MCA 发出 MQPUT 调用之前

包含传输队列头 MQXQH 以及从队列中检索的应用程序消息文本的代理程序缓冲区将传递到消息出口。 MQXQH - 传输队列头中提供了 MQXQH 的格式。

如果您使用参考消息（此类型消息只包含指向要发送的其他某些对象的头），消息出口可识别头 MQRMH。 消息头将识别对象（使用相应的方法检索对象并将其附加到头）并将其传递到 MCA 以传输到接收 MCA。 在接收 MCA 上，另一个消息出口将识别此消息为参考消息，然后抽取对象并将头传递到目标队列。 请参阅第 735 页的『参考消息』和第 1005 页的『运行“参考消息”样本』以了解有关参考消息和某些处理这些消息的样本消息出口的信息。

消息出口可以返回以下响应：

- 发送消息（GET 出口）。 消息可能已被出口更改。 （将返回 MQXCC_OK。）
- 将消息放入队列（PUT 出口）。 消息可能已被出口更改。 （将返回 MQXCC_OK。）
- 请不要处理此消息。 该消息将由 MCA 放入死信队列（未发送的消息队列）。
- 关闭通道。
- 错误返回码，将导致 MCA 异常结束。

注：

1. 将为每个完成传输的消息调用一次消息出口，即使消息被拆分为多个部分。
2. Linux UNIX 如果在 UNIX 或 Linux 上提供消息出口，那么用户标识不会自动转换为小写字符（如此处所述）。

3. 出口会与 MCA 本身在同一线程内运行。 因为它使用相同的连接句柄，因此还会与 MCA 在同一工作单元 (UOW) 内运行。 因此，在同步点下进行的任何调用都由批处理端的通道落实或回退。 例如，当落实包含原始消息的批处理时，一个通道消息出口程序可以向另一个通道消息出口程序发送通知消息，并且这些消息只会落实到队列。

   因此，您可以从通道消息出口程序发出同步点 MQI 调用。

在消息出口外转换消息
在调用消息出口前，接收 MCA 将对消息执行某些转换。 本主题描述了用于执行这些转换的算法。

处理哪些头

调用消息出口前，转换例程在接收方的 MCA 中运行。 转换例程从消息开头的 MQXQH 头开始。 转换例程随后处理 MQXQH 后跟的连锁头，在需要的位置执行转换。 连锁头的扩展可以超出 MQCXP 数据（传递到接收方消息出口）的 HeaderLength 参数内包含的偏移量。 以下头将就地转换：

- MQXQH（格式名称“MQXMIT”）
- MQMD（此头是 MQXQH 的一部分，没有格式名称）
- MQMDE（格式名称“MQHMDE”）
- MQDH（格式名称“MQHDIST”）
- MQWIH（格式名称“MQHWIH”）

以下头不会转换，但因为 MCA 要继续处理连锁头，所以会跳过这些头：

- MQDLH（格式名称“MQDEAD”）
- 此处未提及的格式名称以“MQH”三个字符开头的任何头（例如，“MQHRF”）

如何处理头

每个 IBM MQ 头的 Format 参数由 MCA 读取。 Format 参数是头中的 8 个字节，是包含名称的 8 个单字节字符。

然后，MCA 将每个头后跟的数据理解为命名类型。 如果 Format 是符合 IBM MQ 数据转换资格的头类型的名称，将转换此名称。 如果是另一个指示非 MQ 数据（例如，MQFMT_NONE 或 MQFMT_STRING）的名称，MCA 将停止处理这些头。

什么是 MQCXP HeaderLength?

为消息出口提供的 MQCXP 数据中的 HeaderLength 参数是消息开头处 MQXQH（包含 MQMD）、MQMDE 和 MQDH 头的总长度。 这些头使用“Format”名称和长度连接。

MQWIH

连锁头的扩展可以超出 HeaderLength 到达用户数据区域。 MQWIH 头（如果存在）是显示时超过 HeaderLength 的那些头之一。

如果连锁头中存在 MQWIH 头，在调用接收方的消息出口前，MQWIH 头将就地转换。

通道消息重试出口程序
尝试打开目标队列不成功时会调用通道消息重试出口。 您可以使用此出口确定重试条件、重试次数和重试频率。

MCA 初始化或终止时，通道接收端也会调用此出口。

包含传输队列头 MQXQH 以及从队列中检索的应用程序消息文本的代理程序缓冲区将传递到通道消息重试出口。 MQXQH 概述中提供了 MQXQH 的格式。

将对所有原因码调用此出口；此出口确定 MCA 要针对哪些原因码执行重试以及执行重试的次数和时间间隔。 （定义通道时设置的消息重试计数值将传递到 MQCD 中的出口，但出口可以忽略此值。）

MCA 每调用一次出口，MQCXP 中的 MsgRetryCount 字段就会增长一次，同时出口会返回 MQXCC_OK，其中 MQCXP 的 MsgRetryInterval 字段中包含等待时间，或者返回 MQXCC_SUPPRESS_FUNCTION。 出口在 MQCXP 的 ExitResponse 字段中返回 MQXCC_SUPPRESS_FUNCTION 之前，重试将一直持续。 请参阅 MQCXP 获取有关 MCA 为这些完成代码采取的操作的信息。

如果所有重试都不成功，消息将写入死信队列。 如果没有可用的死信队列，通道将停止。

如果您没有为通道定义消息重试出口，同时又发生了临时性的故障（例如 MQRC_Q_FULL），MCA 将使用定义通道时设置的消息重试计数和消息重试时间间隔。 如果此故障具有更持久的性质并且您没有定义处理此故障的出口程序，消息将写入死信队列。

通道自动定义出口程序
当收到启动接收方或服务器连接通道的请求，但没有用于该通道的定义时，可以使用通道自动定义出口（不适用于 IBM MQ for z/OS）。 也可以在所有平台上为集群发送方和集群接收方通道调用此出口以允许修改通道实例的定义。

当收到启动接收方或服务器连接通道的请求，但没有通道定义时，可以在所有平台上（除 z/OS 外）调用通道自动定义出口。 您可以使用此出口修改为自动定义的接收方或服务器连接通道（SYSTEM.AUTO.RECEIVER 或 SYSTEM.AUTO.SVRCON）提供的缺省定义。 请参阅准备通道以获取如何自动创建通道定义的描述。

还可以在收到启动集群发送方通道的请求时调用通道自动定义出口。 可以为集群发送方和集群接收方通道调用此出口以允许修改此通道实例的定义。 这种情况下，此出口也适用于 IBM MQ for z/OS。 通道自动定义出口的一般用途是更改消息出口（MSGEXIT、RCVEXIT、SCYEXIT 和 SENDEXIT）的名称，因为出口名称在不同的平台上有不同的格式。 如果没有指定通道自动定义出口，z/OS 上的缺省行为是检查 *[path]/libraryname(function)* 格式的分布式出口名称并获取最多 8 个字符的函数（如果存在）或库名。 在 z/OS 上，通道自动定义出口程序必须变更由 MsgExitPtr、MsgUserDataPtr、SendExitPtr、SendUserDataPtr、ReceiveExitPtr 和 ReceiveUserDataPtr 处理的字段，而不是变更 MsgExit、MsgUserData、SendExit、SendUserData、ReceiveExit 和 ReceiveUserData 字段本身。

有关更多信息，请参阅使用自动定义通道。

与其他通道出口一样，参数列表为：

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

`ChannelExitParms` 在 MQCXP 中进行了描述。 `ChannelDefinition` 在 MQCD 中进行了描述。

MQCD 包含缺省通道定义内使用的值，前提是出口没有变更这些值。 此出口只能更改这些字段的子集；请参阅 MQ_CHANNEL_AUTO_DEF_EXIT。 但是，尝试更改其他字段不会导致错误发生。

通道自动定义出口返回 MQXCC_OK 或 MQXCC_SUPPRESS_FUNCTION 响应。 如果不返回以上任一响应，MCA 将如同返回 MQXCC_SUPPRESS_FUNCTION 了一样继续处理。 即放弃自动定义，不创建新的通道定义，通道将无法启动。

ULW 在 *Windows、UNIX and Linux* 系统上编译通道出口程序

以下示例用于帮助您为 Windows、UNIX and Linux 系统编译通道出口程序。

Windows

Windows

用于 Windows 上的通道出口程序的编译器和链接程序命令：

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

UNIX and Linux 系统

Linux UNIX

在以上示例中，exit 是库名，ChannelExit 是函数名。 在 AIX 上，导出文件称为 exit.exp。 这些名称由通道定义用于引用出口程序，这些出口程序使用 MQCD- 通道定义中描述的格式。 另请参阅 DEFINE CHANNEL 命令的 MSGEXIT 参数。

AIX 用于 AIX 上的通道出口的样本编译器和链接程序命令：

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux 用于 Linux（其中队列管理器为 32 位）上的通道出口的样本编译器和链接程序命令：

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux 用于 Linux（其中队列管理器为 64 位）上的通道出口的样本编译器和链接程序命令：

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Solaris 用于 Solaris 上的通道出口的样本编译器和链接程序命令：

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

在客户机上，可以使用 32 位或 64 位出口。 此出口必须链接到 mqic_r。

AIX 在 AIX 上，必须导出 IBM MQ 调用的所有函数。 下面是此 makefile 的样本导出文件：

```
#
!channelExit
MQStart
```

配置通道出口

要调用通道出口，必须在通道定义中对它进行命名。

必须在通道定义中对通道出口进行命名。 您可以在首次定义通道时进行此命名，也可以在以后添加此信息，例如，使用 MQSC 命令 ALTER CHANNEL。 您也可以在 MQCD 通道数据结构中给出通道出口名称。 出口名称格式取决于您的 IBM MQ 平台；请参阅 MQCD 或 MQSC 命令以获取信息。

如果通道定义不包含用户出口程序名，那么不会调用用户出口。

通道自动定义出口是队列管理器的属性，而不是单个通道。 为了调用此出口，必须在队列管理器定义中对其进行命名。 要更改队列管理器定义，请使用 MQSC 命令 ALTER QMGR。

编写数据转换出口

此主题集合包含了有关如何编写数据转换出口的信息。

注: 在 MQSeries for VSE/ESA 中不受支持。

执行 MQPUT 时，应用程序会创建消息的消息描述符 (MQMD)。 由于 IBM MQ 需要能理解 MQMD 的内容，无论它基于哪个平台而创建，系统都会自动将其转换。

但是，无法自动转换应用程序数据。 如果字符数据正在 CodedCharSetId 和 Encoding 字段不同的平台之间进行交换（例如，ASCII 和 EBCDIC 之间），应用程序必须安排消息的转换。 队列管理器自身或用户出口程序可以执行应用程序数据转换，并将其称为数据转换出口。 如果应用程序数据为某种内置格式（如 MQFMT_STRING），那么队列管理器可以使用某个内置的转换例程自行执行数据转换。 本主题包含了当应用程序数据不是内置格式时，IBM MQ 所提供的有关数据转换出口工具的信息。

在 MQGET 调用期间，可以将控制传递给数据转换出口。 这避免了在到达最终目标之前跨多个不同平台进行转换。 但是，如果最终目标是一个不支持在 MQGET 上进行数据转换的平台，那么您必须在将数据发送至其最终目标的发送方通道上指定 CONVERT(YES)。 这可以确保 IBM MQ 在传输期间转换数据。 在这种情况下，数据转换出口必须位于已定义发送方通道的系统上。

MQGET 调用由应用程序直接发布。 将 MQMD 中的 CodedCharSetId 和 Encoding 字段设置为所需的字符集和编码。 如果应用程序使用与队列管理器相同的字符集和编码，那么将 CodedCharSetId 设置为 MQCCSI_Q_MGR，并将 Encoding 设置为 MQENC_NATIVE。 在 MQGET 调用完成之后，这些字段便具有了与返回的消息数据相应的值。 如果转换不成功，这些值可能与所需的值不同。 应用程序应将这些字段重置为每次调用 MQGET 之前所需的值。

调用数据转换出口所需的条件在 MQGET 中针对 MQGET 调用进行定义。

有关传递到数据转换出口的参数描述以及详细使用说明，请参阅数据转换，以了解 MQ_DATA_CONV_EXIT 调用和 MQDXP 结构。

在不同机器编码和 CCSID 之间转换应用程序数据的程序必须符合 IBM MQ 数据转换接口 (DCI)。

对于多点广播客户机，API 出口和数据转换出口必须能够在客户端上运行，这是因为某些消息可能无法通过队列管理器。 以下库属于客户机包和服务器包：

表 *147:* 位于客户机包和服务器包中的库

操作系统	**库**
AIX	32 位及 64 位：libmqm.a 和 libmqm_r.a
IBM i	LIBMQM 和 LIBMQM_R
Linux	32 位及 64 位：libmqm.so 和 libmqm_r.so
Solaris	32 位及 64 位：libmqm.so
Windows	32 位及 64 位：mqm.dll 和 mqm.pdb

调用数据转换出口

数据转换出口是用户编写的出口，用于在处理 MQGET 调用期间接收控制。

如果以下语句为 true，那么会调用该出口：

- 在 MQGET 调用上指定 MQGMO_CONVERT 选项。
- 部分或全部消息数据不在所请求的字符集或编码中。
- 与消息关联的 MQMD 结构中的 *Format* 字段不是 MQFMT_NONE。
- MQGET 调用上指定的 *BufferLength* 不为零。
- 消息数据长度不为零。
- 消息包含了具有用户定义格式的数据。 用户定义的格式可以占用整个消息，或者前面有一个或者多个内置格式。 例如，用户定义的格式前面可以是 MQFMT_DEAD_LETTER_HEADER 格式。 调用出口来仅转换用户定义的格式；队列管理器会转换用户定义格式之前的任意内置格式。

  也可以调用用户编写的出口来转换内置格式，但这仅在内置转换例程无法成功转换内置格式时才会发生。

还有一些其他条件，MQ_DATA_CONV_EXIT 中 MQ_DATA_CONV_EXIT 调用的使用说明中有完整描述。

请参阅 MQGET，以获取有关 MQGET 调用的详细信息。 除 MQXCNVC 之外，数据转换出口无法使用 MQI 调用。

由于应用程序已连接到了队列管理器，因此当应用程序尝试检索使用该 *Format* 的第一条消息时，会装入出口的新副本。 如果队列管理器已丢弃先前装入的副本，可能也会在其他时间装入新副本。

数据转换出口在类似于发出 MQGET 调用的程序环境中运行。 除了用户应用程序，该程序可以是向不支持消息转换的目标队列管理器发送消息的 MCA（消息通道代理）。 该环境包括地址空间和用户配置文件（在适当的情况下）。 该出口不能损害队列管理器的完整性，因为它不会在队列管理器的环境中运行。

z/OS 上的数据转换

z/OS

在 z/OS 上，请注意以下事项：

- 只能以汇编语言编写出口程序。
- 出口程序必须是可重入的，并且能够在存储器中的任何地方运行。
- 出口程序必须将出口上的环境复原到入口上的环境，并且必须释放获取的所有存储器。
- 出口程序禁止 WAIT，或者发出 ESTAE 或 SPIE。
- 通常会像通过 z/OS LINK 一样采用以下方式调用出口程序：
  - 未授权的问题程序状态
  - 主地址空间控制方式
  - 非跨内存方式
  - 非访问注册方式
  - 31 位寻址方式
  - TCB-PRB 方式
- 当由 CICS 应用程序使用时，出口由 EXEC CICS LINK 调用，并且必须符合 CICS 编程约定。 参数由 CICS 通信区 (COMMAREA) 中的指针（地址）进行传递。

  虽然不推荐，但用户出口程序也可以使用 CICS API 调用，不过请注意以下事项：
  - 请勿发出同步点，因为结果可能会影响 MCA 声明的工作单元。
  - 请勿更新除了 IBM MQ for z/OS 之外的资源管理器所控制的任何资源，包括由 CICS Transaction Server 所控制的那些资源。

  对于具有 CONVERT=YES 的通道，将从 CSQXLIB DD 语句引用的数据集装入出口。 MQ 针对 IBM MQ CICS 网桥提供的出口 CSQCBDCI 和 CSQCBDCO 位于 SCSQAUTH 中。

为 *IBM i* 编写数据转换出口程序

有关为 IBM i 编写 MQ 数据转换出口程序时要考虑的步骤的信息。

请按照以下步骤操作：

1. 为您的消息格式命名。 该名称必须适合 MQMD 的 *Format* 字段。 *Format* 名称不得有前导嵌入空格，并且忽略尾部空格。 对象名称不能超过八个非空字符，因为 *Format* 只有八个字符长。 请记住每次发送消息时都使用此名称（我们的示例使用名称 Format）。
2. 创建结构以表示您的消息。 有关示例，请参阅有效语法。
3. 通过 CVTMQMDTA 命令运行此结构，以便为您的数据转换出口创建代码片段。

   CVTMQMDTA 命令生成的函数使用文件 QMQM/H(AMQSVMHA) 中提供的宏。 如果所有结构都已打包，那么会编写这些宏；否则，会对它们进行修改。
4. 获取已提供的框架源文件 QMQMSAMP/QCSRC(AMQSVFC4) 的副本并对其重命名。 （我们的示例使用名称 EXIT_MOD。）
5. 在源文件中找到以下注释框，并按照所述插入代码：
   a. 在接近源文件的末尾，注释框始于：

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   此处，插入在步骤 第 895 页的『3』 中生成的代码片段。
   b. 在接近源文件的中间，注释框始于：

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   这之后是注释掉对函数 ConverttagSTRUCT 的调用。

将函数的名称更改为在步骤 第 895 页的『5.a』 中所添加的函数名称。 移除注释字符以激活该函数。 如果有多个函数，请为每个函数创建调用。

c. 在接近源文件的开头，注释框始于：

```
/* Insert the function prototypes for the functions produced by */
```

此处，为步骤 第 895 页的『5.a』 中添加的函数插入函数原型语句。

如果消息包含字符数据，那么生成的代码会调用 MQXCNVC；这可以通过绑定服务程序 QMQM/LIBMQM 来解决。

6. 编译源模块 EXIT_MOD，如下所示：

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. 创建/链接程序。

有关非线程应用程序，请使用以下内容：

```
CRTPGM PGM(library/Format)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

除了为基本环境创建数据转换出口之外，还需要在线程环境中创建另一个出口。 此可装入对象后面必须跟有 _R。 请使用 LIBMQM_R 库来解决对 MQXCNVC 的调用。 这两种可装入对象对于线程环境都是必需的。

```
CRTPGM PGM(library/Format_R)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM_R)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

8. 将输出放入 IBM MQ 作业的库列表中。 对于生产，建议将数据转换出口程序存储在 QSYS 中。

注：

1. 如果 CVTMQMDTA 使用打包的结构，那么所有 IBM MQ 应用程序都必须使用 _Packed 限定符。
2. 数据转换出口程序必须可重入。
3. MQXCNVC 是可以从数据转换出口发出的唯一 MQI 调用。
4. 通过将用户概要文件编译器选项设置为 *USER 来编译出口程序，以便该出口以用户的权限运行。
5. IBM MQ for IBM i 的所有用户出口都需要启用太字节空间内存；在 CRTCMOD 和 CRTBNDC 命令中指定 TERASPACE(*YES *TSIFC)。

为 *IBM MQ for z/OS* 编写数据转换出口程序

有关为 IBM MQ for z/OS 编写数据转换出口程序时要考虑步骤的信息。

请按照以下步骤操作：

1. 使用提供的源框架 CSQ4BAX9（对于非 CICS 环境）或 CSQ4CAX9（对于 CICS）作为起点。
2. 运行 CSQUCVX 实用程序。
3. 按照 CSQ4BAX9 或 CSQ4CAX9 序言中的说明，合并由 CSQUCVX 实用程序生成的例程，采用这些结构在要转换的消息中出现的顺序。
4. 该实用程序假定数据结构未打包，采用数据的隐含对齐方式并且结构从全字边界开始，根据需要跳过字节（如有效语法示例中的 ID 和 VERSION 之间）。 如果结构已打包，那么忽略生成的 CMQXCALA 宏。 因此，请考虑以指定所有字段且不跳过任何字节的方式声明您的结构；在有效语法中的示例中，在 ID 和 VERSION 之间添加字段 “MQBYTE DUMMY;”。

5. 如果输入缓冲区比要转换的消息格式要短，那么提供的出口会返回错误。 尽管出口会尽可能多地转换完整字段，但错误会导致未转换的消息返回到应用程序。 如果希望尽可能地转换短输入缓冲区（包括部分字段），请将 CSQXCDFA 宏上的 TRUNC= 值更改为 YES：没有返回错误，因此应用程序会收到一条转换的消息。 应用程序必须处理截断情况。
6. 添加您需要的任何其他特殊处理代码。
7. 将程序重命名为数据格式名称。
8. 像批处理应用程序一样编译并链接编辑您的程序（除非它与 CICS 应用程序一起使用）。 实用程序生成的代码中的宏位于库 **thlqual**.SCSQMACS 中。

   如果消息包含字符数据，那么生成的代码会调用 MQXCNVC。 如果出口使用此调用，那么通过出口存根程序 CSQASTUB 对其进行链接编辑。 存根与语言和环境均无关。 或者，您可以使用动态调用名称 CSQCNBC 来动态装入存根。 有关更多信息，请参阅第 940 页的『动态调用 IBM MQ 存根』。

   将链接编辑的模块放入应用程序装入库中，以及由通道启动程序所启动任务过程的 CSQXLIB DD 语句引用的数据集中。
9. 如果出口由 CICS 应用程序所使用，请像 CICS 应用程序一样对其进行编译和链接编辑，包括 CSQASTUB（如果需要）。 将其放入 CICS 应用程序库中。 以典型的方式将应用程序定义到 CICS，在定义中指定 EXECKEY (CICS)。

注: 虽然运行 CSQUCVX 实用程序需要 LE/370 运行时库（请参阅步骤 第 896 页的『2』），但它们对于链接编辑或运行数据转换出口本身不是必需的（请参阅步骤 第 897 页的『8』 和 第 897 页的『9』）。

请参阅第 60 页的『编写 IMS 桥接应用程序』，了解有关 IBM MQ - IMS 网桥中的数据转换的信息。

Linux UNIX 在 *UNIX and Linux* 系统上为 *IBM MQ* 编写数据转换出口

有关在 UNIX and Linux 系统上为 IBM MQ 编写数据转换出口程序时要考虑的步骤的信息。

请按照以下步骤操作：

1. 为您的消息格式命名。 该名称必须适合 MQMD 的 *Format* 字段，并且必须是大写，例如 MYFORMAT。 *Format* 名称不得有前导空格。 尾部空格将被忽略。 对象名称不能超过八个非空字符，因为 *Format* 只有八个字符长。 请记住每次发送消息时都使用此名称。

   如果在线程环境中使用数据转换出口，可装入对象必须后跟 _r 以指示其为线程版本。
2. 创建结构以表示您的消息。 有关示例，请参阅有效语法。
3. 通过 `crtmqcvx` 命令运行此结构，以便为您的数据转换出口创建代码片段。

   由 `crtmqcvx` 命令生成的函数会使用宏，假设所有结构都已打包；否则，会对它们进行修改。
4. 复制提供的框架源文件，将其重命名为您在步骤 第 897 页的『1』 中所设置的消息格式的名称。 框架源文件和副本是只读的。

   框架源文件称为 amqsvfc0.c。
5. 在 IBM MQ for AIX 上，还提供了一个名为 amqsvfc.exp 的框架导出文件。 复制此文件，将其重命名为 MYFORMAT.EXP。
6. 该框架在目录 *MQ_INSTALLATION_PATH*/inc 中包含样本头文件 amqsvmha.h，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。 确保您的包含路径指向此目录以选取此文件。

   amqsvmha.h 文件包含宏，这些宏由 `crtmqcvx` 命令生成的代码所使用。 如果要转换的结构包含字符数据，那么这些宏会调用 MQXCNVC。
7. 在源文件中找到以下注释框，并按照所述插入代码：

   a. 在接近源文件的末尾，注释框始于：

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   此处，插入在步骤 第 897 页的『3』 中生成的代码片段。

   b. 在接近源文件的中间，注释框始于：

```
/* Insert calls to the code fragments to convert the format's */
```

这之后是注释掉对函数 ConverttagSTRUCT 的调用。

将函数的名称更改为在步骤 第 897 页的『7.a』 中所添加的函数名称。 移除注释字符以激活该函数。 如果有多个函数，请为每个函数创建调用。

c. 在接近源文件的开头，注释框始于：

```
/* Insert the function prototypes for the functions produced by */
```

此处，为步骤 第 897 页的『3』 中添加的函数插入函数原型语句。

8. 将出口编译为共享库，使用 MQStart 作为入口点。 要执行此操作，请参阅第 898 页的『在 UNIX and Linux 系统上编译数据转换出口』。
9. 将输出放入出口目录中。 缺省出口目录为 /var/mqm/exits（对于 32 位系统）和 /var/mqm/exits64（对于 64 位系统）。 您可以在 qm.ini 或 mqclient.ini 文件中更改这些目录。 可以为每个队列管理器设置此路径，并且只能在该路径中查找该出口。

注：

1. 如果 crtmqcvx 使用打包结构，所有 IBM MQ 应用程序都必须使用这种方式进行编译。
2. 数据转换出口程序必须可重入。
3. MQXCNVC 是可以从数据转换出口发出的唯一 MQI 调用。

Linux UNIX 在 *UNIX and Linux* 系统上编译数据转换出口

有关如何在 UNIX and Linux 系统上编译数据转换出口的示例。

在所有平台上，模块的入口点都是 MQStart。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

AIX

AIX

通过发出以下某个命令编译出口源代码：

32 位应用程序

非线程化

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

64 位应用程序

非线程化

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

通过发出以下某个命令编译出口源代码：

31 位应用程序

非线程化

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

32 位应用程序

非线程化

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

64 位应用程序

非线程化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

线程化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

Solaris

Solaris

通过发出以下某个命令编译出口源代码：

32 位应用程序

SPARC 平台

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

x86-64 平台

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

64 位应用程序

SPARC 平台

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

x86-64 平台

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

Windows 为 *IBM MQ for Windows* 编写数据转换出口

有关为 IBM MQ for Windows 编写数据转换出口程序时要考虑步骤的信息。

请按照以下步骤操作：

1. 为您的消息格式命名。 该名称必须适合 MQMD 的 *Format* 字段。 *Format* 名称不得有前导空格。 尾部空格将被忽略。 对象名称不能超过八个非空字符，因为 *Format* 只有八个字符长。

   在样本目录 *MQ_INSTALLATION_PATH*\Tools\C\Samples 中还提供了名为 amqsvfcn.def 的 .DEF 文件。 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。 获取此文件的副本并对其重命名，例如，重命名为 MYFORMAT.DEF。 确保正在创建的 DLL 名称与 MYFORMAT.DEF 中指定的名称相同。 使用新格式名称覆盖 MYFORMAT.DEF 中的名称 FORMAT1。

   请记住每次发送消息时都使用此名称。
2. 创建结构以表示您的消息。 有关示例，请参阅有效语法。
3. 通过 crtmqcvx 命令运行此结构，以便为您的数据转换出口创建代码片段。

   由 CRTMQCVX 命令生成的函数会使用在假定所有结构都已打包的情况下编写的宏；否则，会对它们进行修改。
4. 复制提供的框架源文件 amqsvfc0.c，将其重命名为您在步骤 第 900 页的『1』 中所设置的消息格式的名称。

   amqsvfc0.c 位于 *MQ_INSTALLATION_PATH*\Tools\C\Samples 中，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。 (缺省安装目录为 C:\Program Files\IBM\MQ。)

   该框架在 *MQ_INSTALLATION_PATH*\Tools\C\include 目录中包含样本头文件 amqsvmha.h 。 确保您的包含路径指向此目录以选取此文件。

   amqsvmha.h 文件包含宏，这些宏由 CRTMQCVX 命令生成的代码所使用。 如果要转换的结构包含字符数据，那么这些宏会调用 MQXCNVC。
5. 在源文件中找到以下注释框，并按照所述插入代码：

   a. 在接近源文件的末尾，注释框始于：

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   此处，插入在步骤 第 900 页的『3』 中生成的代码片段。

   b. 在接近源文件的中间，注释框始于：

```
/* Insert calls to the code fragments to convert the format's */
```

这之后是注释掉对函数 ConverttagSTRUCT 的调用。

将函数的名称更改为在步骤 第 900 页的『5.a』 中所添加的函数名称。 移除注释字符以激活该函数。 如果有多个函数，请为每个函数创建调用。

c. 在接近源文件的开头，注释框始于：

```
/* Insert the function prototypes for the functions produced by */
```

此处，为步骤 第 900 页的『3』 中添加的函数插入函数原型语句。

6. 创建以下命令文件：

```
cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp    \
MYFORMAT.C


MYFORMAT.DEF
```

其中 *MQ_INSTALLATION_PATH* 是安装 IBM MQ 的目录。

7. 发出命令文件，将您的出口编译为 DLL 文件。
8. 将输出放在 IBM MQ 数据目录下的出口子目录中。 用于安装出口的缺省目录是 *MQ_DATA_PATH*\Exits（对于 32 位系统）和 *MQ_DATA_PATH*\Exits64（对于 64 位系统）

用于查找数据转换出口的路径在注册表中提供。 注册表文件夹是：

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

注册表键是：ExitsDefaultPath。 可以为每个队列管理器设置此路径，并且只能在该路径中查找该出口。

注：

1. 如果 CRTMQCVX 使用打包的结构，那么所有 IBM MQ 应用程序都必须以这种方式进行编译。
2. 数据转换出口程序必须可重入。
3. MQXCNVC 是可以从数据转换出口发出的唯一 MQI 调用。

Windows *Windows 操作系统上的出口和切换装入文件*

IBM WebSphere MQ for Windows 7.5 队列管理器进程为 32 位。 因此，在使用 64 位应用程序时，某些类型的出口和 XA 切换装入文件也需要具有可供队列管理器使用的 32 位版本。 如果需要 32 位版本的出口或 XA 切换装入文件但其不可用，那么相关的 API 调用或命令将失败。

在 qm.ini file 中，*ExitPath* 支持两个属性。 这些是 ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits 和 ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64。 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。 使用这些可确保找到相应的库。 如果将出口用于 IBM MQ 集群，也可确保能找到远程系统上相应的库。

下表列出了不同类型的出口和切换装入文件，并根据是使用 32 位还是 64 位应用程序，记录了是否需要 32 位和/或 64 位版本：

文件类型	32 位应用程序	64 位应用程序
API 出口 (API exit)	32 位和 64 位	64 位
数据转换出口	32 位	64 位
服务器通道出口（所有类型）	64 位	64 位

文件类型	32 位应用程序	64 位应用程序
客户机通道出口（所有类型）	32 位	64 位
可安装服务出口	64 位	64 位
集群 WLM 出口	64 位	64 位
发布/预订路由出口	64 位	64 位
数据库切换装入文件	32 位和 64 位	64 位
外部事务管理器 AX 库	32 位	64 位
预连接出口	32 位	64 位

使用存储库的预连接出口引用连接定义

可以配置 IBM MQ MQI clients 来查找存储库以使用预连接出口库获取连接定义。

介绍

客户机应用程序可以使用客户机通道定义表 (CCDT) 连接到队列管理器。 通常，CCDT 文件位于中央网络文件服务器上，并具有引用它的客户机。 由于难以控制和管理引用 CCDT 文件的各种客户机应用程序，因此灵活的方法是将客户机定义存储在全局存储库中，如 LDAP 目录、WebSphere Registry and Repository 或任何其他存储库。 将客户机连接定义存储在存储库中会使得管理客户机连接定义更容易，并且应用程序可以访问正确且最新的客户机连接定义。

在执行 MQCONN/X 调用期间，IBM MQ MQI client 会装入应用程序指定的预连接出口库，并调用出口函数来检索连接定义。 然后，检索到的连接定义用来与队列管理器建立连接。 要调用的出口库和函数的详细信息在 mqclient.ini 配置文件中指定。

语法

```
void MQ_PRECONNECT_EXIT (pExitParms, pQMgrName, ppConnectOpts, pCompCode, pReason);
```

参数

pExitParms
类型：PMQNXP 输入/输出

PreConnection 出口参数结构。

结构由出口的调用者分配和维护。

pQMgrName
类型：PMQCHAR 输入/输出

队列管理器的名称。

在输入中，此参数是通过 **QMgrName** 参数提供给 MQCONN API 调用的过滤器字符串。 此字段可能为空，显式或包含特定通配符。 该字段由出口更改。 使用 MQXR_TERM 调用出口时参数为 NULL。

ppConnectOpts
类型：ppConnectOpts 输入/输出

控制 MQCONNX 操作的选项。

这是控制 MQCONN API 调用操作的 MQCNO 连接选项结构的指针。 使用 MQXR_TERM 调用出口时参数为 NULL。 MQI 客户机始终向出口提供 MQCNO 结构，即使它最初不是由应用程序提供的。 如果应用程序提供 MQCNO 结构，那么客户机会制作副本以将其传递给修改它的出口。 客户机保留 MQCNO 的所有权。

通过 MQCNO 引用的 MQCD 优先于通过数组提供的任何连接定义。 客户机使用 MQCNO 结构来连接到此队列管理器，而忽略其他队列管理器。

pCompCode
类型：PMQLONG 输入/输出

完成代码。

指向用于接收出口完成代码的 MQLONG 的指针。 它必须是下列其中一个值：

- MQCC_OK - 成功完成
- MQCC_WARNING - 警告（部分完成）
- MQCC_FAILED - 调用失败

pReason
类型：PMQLONG 输入/输出

限定 pCompCode 的原因。

指向用于接收出口原因码的 MQLONG 的指针。 如果完成代码是 MQCC_OK，那么唯一的有效值是：

- MQRC_NONE - (0, x'000') 没有要报告的原因。

如果完成代码是 MQCC_FAILED 或 MQCC_WARNING，出口函数可以将原因码字段设置为任何有效 MQRC_* 值。

C 调用

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

编写和编译发布出口

您可以在队列管理器上配置发布出口，以便在订户接收发布的消息之前更改该消息的内容。 另外，您还可以更改消息头或者不将该消息传递到预订。

注: 发布出口在 z/OS 上不受支持。

您可以使用发布出口来检查和更改发送给订户的消息：

- 对发布到每个订户的消息内容进行检查
- 对发布到每个订户的消息内容进行修改
- 更改要将消息放入的队列
- 停止向订户传送消息

编写发布出口

使用第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』中的步骤，帮助您编写和编译出口。

发布出口的提供者定义了出口的作用。 但是，此出口必须符合 MQPSXP 中定义的规则。

IBM MQ 不提供 MQ_PUBLISH_EXIT 入口点的实现。 它提供了 C 语言 typedef 声明。 使用 typedef 向用户编写的出口正确声明参数。 以下示例演示如何使用 typedef 声明：

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;
```

```
void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

发布出口作为下列操作的结果在队列管理器进程中运行：

- 用于将消息传递到一个或多个订户的“发布”操作。
- 用于传递一条或多条已保留的消息的“预订”操作
- 用于传递一条或多条已保留的消息的“预订请求”操作

如果为连接调用发布出口，那么会在首次调用时设置 MQXR_INIT 的 *ExitReason* 代码。 在使用发布出口之后、断开连接之前，使用 MQXR_TERM 的 *ExitReason* 代码调用出口。

如果已配置发布出口，但无法在启动队列管理器时装入该出口，那么该队列管理器将禁止执行发布/预订消息操作。 您必须先解决问题或重新启动队列管理器，然后才能重新启用发布/预订消息传递。

需要发布出口的每个 IBM MQ 连接可能无法装入或初始化出口。 如果此出口无法装入或初始化，那么会对该连接禁用需要发布出口的发布/预订操作。 这些操作将失败，并返回 IBM MQ 原因码 MQRC_PUBLISH_EXIT_ERROR。

从中调用发布出口的上下文是应用程序与队列管理器之间的连接。 队列管理器为正在执行发布操作的每个连接维护用户数据区域。 此出口可以在每个连接的用户数据区域保留信息。

发布出口可以使用一些 MQI 调用。 它只能使用处理消息属性的那些 MQI 调用。 调用如下：

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

如果发布出口更改了目标队列管理器或队列名，那么不会执行新的权限检查。

编译发布出口

发布出口是动态装入的库；您可以将其想像成通道出口。 有关编译出口的信息，请参阅第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』。

样本发布出口

样本出口程序称为 amqspse0.c。 它根据是否调用出口来初始化、发布或终止操作，向日志文件写入不同的消息。 它还展示了如何使用出口用户区域字段来适当地分配和释放存储器。

配置发布出口

必须定义某些属性才能配置发布出口。

您可以在 Windows 和 Linux 上使用 IBM MQ Explorer 来定义属性。 在队列管理器属性页面的“发布/预订”下定义属性。

要在 UNIX and Linux 系统上的 qm.ini 文件中配置发布出口，请创建名为 PublishSubscribe 的节。 PublishSubscribe 节具有以下属性：

PublishExitPath=[path]|*module_name*
发布出口代码所在的模块的名称和路径。 此字段的最大长度为 MQ_EXIT_NAME_LENGTH。 缺省情况是没有发布出口。

PublishExitFunction=函数名
发布出口代码所在的模块的函数入口点名称。 此字段的最大长度为 MQ_EXIT_NAME_LENGTH。

IBM i 在 IBM i 上使用程序时，请省略 PublishExitFunction。

PublishExitData=字符串
如果队列管理器正在调用发布出口，那么它将传递 MQPSXP 结构作为输入。 使用 **PublishExitData** 属性指定的数据在结构的 *ExitData* 字段中提供。 此字符串的长度可达 MQ_EXIT_DATA_LENGTH 个字符。 缺省值是 32 个空白字符。

编写和编译集群工作负载出口

编写集群工作负载出口程序来定制集群的工作负载管理。 在路由消息时，您可能会考虑在一天的不同时间使用通道或消息内容的成本。 但这些不是标准工作负载管理算法考虑的因素。

在大多数情况下，工作负载管理算法足以满足您的需求。 但是，为了可以提供自己的用户出口程序来定制工作负载管理，IBM MQ 包含了一个用户出口，即集群工作负载出口。

您可能有一些可用于影响工作负载平衡的网络或消息的相关特定信息。 您可能知道哪些是大容量通道或廉价的网络路由，也可能需要根据其内容来路由消息。 您可以决定编写集群工作负载出口程序，或使用第三方提供的程序。

在访问集群队列时，将调用集群工作负载出口。 它由 MQOPEN、MQPUT1 和 MQPUT 调用。

如果指定了 MQOO_BIND_ON_OPEN，那么在 MQOPEN 时选择的目标队列管理器是固定的。 在这种情况下，出口只运行一次。

如果目标队列管理器在 MQOPEN 时不是固定的，那么会在 MQPUT 调用时选中目标队列管理器。 如果目标队列管理器不可用，或者当消息仍在传输队列上时发生故障，那么会再次调用该出口。 选择新的目标队列管理器。 如果在传输消息时消息通道发生故障，并且消息已回退，那么会选择新的目标队列管理器。

Multi 在 多平台 上，队列管理器将在下一次启动时装入新的集群工作负载出口。

如果队列管理器定义不包含集群工作负载出口程序名称，那么不会调用集群工作负载出口。

将各种数据传递到出口参数结构 MQWXP 中的集群工作负载出口：

- 消息定义结构 MQMD。
- 消息长度参数。
- 消息的副本或消息的一部分。

在非 z/OS 平台上，如果使用 CLWLMode=FAST，每个操作系统进程都会装入各自的出口副本。 与队列管理器的不同连接可能会导致调用不同的出口副本。 如果出口在缺省安全模式 CLWLMode=SAFE 下运行，那么出口的单个副本会在其独立进程中运行。

编写集群工作负载出口

z/OS 有关为 z/OS 编写集群工作负载出口的信息，请参阅第 907 页的『针对 IBM MQ for z/OS 的集群工作负载出口编程』。

Multi 对于 Multiplatforms，集群工作负载出口不得使用 MQI 调用。 在其他方面，编写和编译集群工作负载出口程序的规则类似于通道出口程序所适用的规则。 按照第 854 页的『在 UNIX、Linux 和 Windows 上编写出口和可安装服务』中的步骤，使用样本程序第 906 页的『样本集群工作负载出口』帮助您编写和编译出口。

有关通道出口的更多信息，请参阅第 877 页的『编写通道出口程序』。

配置集群工作负载出口

通过在 ALTER QMGR 命令中指定集群工作负载出口属性，可以在队列管理器定义中对集群工作负载出口进行命名。 例如：

```
ALTER QMGR CLWLEXIT(myexit)
```

相关参考

集群工作负载出口调用和数据结构

样本集群工作负载出口

IBM MQ 包括样本集群工作负载出口程序。 您可以复制样本，并且使用它作为程序的基础。

z/OS **IBM MQ for z/OS**

样本集群工作负载出口程序在汇编程序和 C 中提供。 汇编程序版本称为 CSQ4BAF1 ，可在库 thlqual.SCSQASMS 中找到。 C 版本称为 CSQ4BCF1，可以在库 thlqual.SCSQC37S 中找到。 thlqual 是安装中 IBM MQ 数据集的目标库高级限定符。

Multi **IBM MQ for Multiplatforms**

提供了 C 语言版本的样本集群工作负载出口程序（称为 amqswlm0.c）。 可以在以下内容中找到：

表 *148:* 针对 *Multiplatforms* 的样本集群工作负载出口程序位置

平台	菲尔帕特
AIX AIX Solaris Solaris	*MQ_INSTALLATION_PATH*/samp
Windows Windows	*MQ_INSTALLATION_PATH* \Tools\c\Samples
IBM i IBM i	qmqm 库

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

该样本出口将所有消息路由到特定队列管理器，除非该队列管理器不可用。 它通过将消息路由到另一个队列管理器，对该队列管理器的故障作出反应。

指出要将消息发送到哪个队列管理器。 在队列管理器定义上的 CLWLDATA 属性中，提供集群接收方通道的名称。 例如：

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

要启用该出口，请在 CLWLEXIT 属性中提供其完整路径和名称：

Linux UNIX 在 UNIX and Linux 上：

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows 在 Windows 上：

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS 在 z/OS 上：

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

其中， x 是 'A' 或 'C'，具体取决于您正在使用的版本的编程语言。

IBM i 在 IBM i 上，使用以下命令之一：

- 使用 MQSC 命令：

```
ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
```

程序名称和库名称都占用 10 个字符，并在必要时在右侧用空格填充。

- 使用 CL 命令：

```
CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
```

现在，代替使用提供的工作负载管理算法，IBM MQ 会调用此出口将所有消息路由到所选的队列管理器。

z/OS 针对 *IBM MQ for z/OS* 的集群工作负载出口编程

如同使用 z/OS **LINK** 命令一样调用集群工作负载出口。 出口要受制于许多严格的编程规则。 避免使用涉及等待的大多数 SVC 命令，或在工作负载出口中使用 STATE 或 ESTATE。

如同使用 z/OS **LINK** 一样采用以下方式调用集群工作负载出口：

- 未授权的问题程序状态
- 主地址空间控制方式
- 非交叉内存模式
- 非访问注册方式
- 31 位寻址方式
- 存储键 8
- 程序键掩码 8
- TCB 键 8

V 9.1.0 将链接编辑的模块放入由通道启动程序的启动式任务过程的 CSQXLIB DD 语句指定的数据集中。 将装入模块的名称指定为队列管理器定义中的工作负载出口名称。

在为 IBM MQ for z/OS 编写工作负载出口时，应用以下规则：

- 您必须用汇编语言或 C 语言编写出口。 如果使用 C 语言，那么它必须符合系统出口的 C 系统编程环境，如 *z/OS C/C++* 编程指南 (SC09-4765) 中所述。
- 如果使用 MQXCLWLN 调用，请通过 *thlqual*.SCSQLOAD 中提供的 CSQMFCLW 进行链接编辑。
- 从 CSQXLIB DD 语句定义的非授权库装入出口。 如果 CSQXLIB 具有 DISP=SHR，那么可以在运行队列管理器时更新出口，在队列管理器启动的下一个 MQCONN 线程中使用新版本。
- 出口必须是可重入的，并且能够在虚拟存储器中的任何地方运行。
- 出口必须在返回时将该环境重置为入口环境。
- 出口必须释放所获得的所有存储器，或确保后续出口调用已释放存储器。
- 不允许 MQI 调用。
- 出口不得使用任何可能导致等待的系统服务，因为等待会严重降低队列管理器的性能。 因此，一般来说，避免使用 SVC、PC 或 I/O。
- 出口不得发出 ESTATE 或 SPIE，除了在其连接的任何子任务内部。

注: 对于可以在出口中执行哪些操作，没有绝对的限制。 不过，大多数 SVC 涉及等待，因此要避免使用，但以下命令除外：

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

请勿使用 ESTAE 和 ESPIE，因为它们的错误处理可能会干扰由 IBM MQ 执行的错误处理。 IBM MQ 可能无法从错误中恢复，或者出口程序无法收到所有错误信息。

系统参数 EXITLIM 会限制出口可能运行的时间量。 EXITLIM 的缺省值是 30 秒。 如果您看到返回码 MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA'，出口可能会循环。 如果您认为出口需要 30 秒以上才能完成，请增加 EXITLIM 的值。

构建过程应用程序

您可以使用若干过程语言之一编写 IBM MQ 应用程序，并在几个不同平台上运行该应用程序。

AIX 在 AIX 上构建过程化应用程序

AIX 出版物描述了如何从编写的程序中构建可执行的应用程序。

本主题描述了在构建 IBM MQ for AIX 应用程序以在 AIX 下运行时，必须执行的其他任务以及对标准任务的更改。 支持 C、C++ 和 COBOL。 有关准备 C++ 程序的信息，请参阅使用 C++。

为使用 IBM MQ for AIX 创建可执行应用程序而必须执行的任务随编写源代码所采用的编程语言而异。 除在源代码中对 MQI 调用进行编码以外，您还必须添加相应的语言语句来针对您使用的语言包含 IBM MQ for AIX 包含文件。 请熟悉这些文件的内容。 请参阅 第 670 页的『IBM MQ 数据定义文件』以获取完整描述。

在运行线程服务器或线程客户机应用程序时，请设置环境变量 AIXTHREAD_SCOPE=S。

AIX 在 *AIX* 中准备 *C* 程序

本主题包含在 AIX 上准备 C 程序所需的链接库的相关信息。

在 *MQ_INSTALLATION_PATH*/samp/bin 目录中提供了预编译的 C 程序。 使用 ANSI 编译器并运行以下命令。 有关 64 位应用程序编程的更多信息，请参阅 64 位平台上的编码标准。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

对于 32 位应用程序：

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

其中 amqsput0 是样本程序。

对于 64 位应用程序：

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

其中 amqsput0 是样本程序。

如果正在使用针对 C++ 程序的 VisualAge C/C++ 编译器，必须包含选项 -q namemangling=v5 才能获得链接库时解析的所有 IBM MQ 符号。

如果要在仅安装了 IBM MQ MQI client for AIX 的机器上使用程序，请重新编译程序以将它们与客户机库 (-lmqic) 链接起来。

链接库

您需要以下库：

- 将程序与 IBM MQ 提供的相应的库链接起来。

  在非线程环境中，链接到以下某个库：

库文件	程序/出口类型
libmqm.a	针对 C 的服务器
libmqic.a & libmqm.a	针对 C 的客户机

  在线程环境中，链接到以下某个库：

库文件	程序/出口类型
libmqm_r.a	针对 C 的服务器

库文件	程序/出口类型
libmqic_r.a & libmqm_r.a	针对 C 的客户机

例如，要从单个编译单元构建一个简单的线程 IBM MQ 应用程序，请运行以下命令。

对于 32 位应用程序：

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

其中 `amqsput0` 是样本程序。

对于 64 位应用程序：

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

其中 `amqsput0` 是样本程序。

如果要在仅安装了 IBM MQ MQI client for AIX 的机器上使用程序，请重新编译程序以将它们与客户机库 (`-lmqic`) 链接起来。

注：

1. 不能链接到多个库。 即，不能同时链接到线程库和非线程库。
2. 如果要编写可安装的服务（请参阅 管理 以获取更多信息），那么需要链接到非线程应用程序中的 `libmqmzf.a` 库以及线程应用程序中的 `libmqmzf_r.a` 库。
3. 如果要生成一个应用程序以通过符合 XA 的事务管理器（例如，IBM TXSeries、Encina 或 BEA Tuxedo）进行外部协调，那么需要链接到非线程应用程序中的 `libmqmxa.a`（如果事务管理器将 “long”类型视为 64 位，则为 `libmqmxa64.a`）和 `libmqz.a` 库，以及链接到线程应用程序中的 `libmqmxa_r.a`（或 `libmqmxa64_r.a`）和 `libmqz_r.a` 库。
4. 您需要将受信任的应用程序链接到线程 IBM MQ 库。 但是，一次只能连接 UNIX and Linux 系统上 IBM MQ 上可信应用程序中的一个线程。
5. 您必须在链接其他任何产品库之前链接 IBM MQ 库。

AIX 在 *AIX* 中准备 *COBOL* 程序

在 AIX 中使用 IBM COBOL Set 和 Micro Focus COBOL 准备 COBOL 程序时，请参照此信息。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

- 32 位 COBOL 副本安装在以下目录中：

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy32
  ```

  并在以下位置创建符号链接：

  ```
  MQ_INSTALLATION_PATH/inc
  ```

- 64 位 COBOL 副本安装在以下目录中：

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

在以下示例中，将 **COBCPY** 环境变量设置为：

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

（对于 32 位应用程序）以及：

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

（对于 64 位应用程序）。

您需要将程序与以下某个库文件链接起来：

库文件	程序/出口类型
libmqmcb.a	针对 COBOL 的服务器（非线程应用程序）
libmqmcb_r.a	针对 COBOL 的服务器（线程应用程序）
libmqicb.a	针对 COBOL 的客户机（非线程应用程序）
libmqicb_r.a	针对 COBOL 的客户机（线程应用程序）

您可以根据程序使用 IBM COBOL Set 编译器或 Micro Focus COBOL 编译器：

- 以 amqm 开头的程序适合于 Micro Focus COBOL 编译器，并且
- 以 amq0 开头的程序适合于任一编译器。

使用 IBM COBOL Set for AIX 准备 COBOL 程序

样本 COBOL 程序随 IBM MQ 提供。 要编译这样的程序，请输入以下列表中相应的命令：

32 位非线程服务器应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

32 位非线程客户机应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

32 位线程服务器应用程序

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

32 位线程客户机应用程序

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

64 位非线程服务器应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

64 位非线程客户机应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

64 位线程服务器应用程序

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

64 位线程客户机应用程序

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

使用 Micro Focus COBOL 准备 COBOL 程序

在编译程序之前，如下所示设置环境变量：

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

要使用 Micro Focus COBOL 编译 32 位 COBOL 程序，请输入：

- 针对 COBOL 的服务器

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- 针对 COBOL 的客户机

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- COBOL 的线程服务器

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- COBOL 的线程客户机

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

要使用 Micro Focus COBOL 编译 64 位 COBOL 程序，请输入：

- 针对 COBOL 的服务器

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- 针对 COBOL 的客户机

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- COBOL 的线程服务器

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- COBOL 的线程客户机

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

其中 amqminqx 是样本程序

有关需要设置的环境变量的说明，请参阅 Micro Focus COBOL 文档。

AIX 在 *AIX* 中准备 *CICS* 应用程序

在 AIX 中准备 CICS 程序时使用此信息。

使用 *XA* 切换模块将 CICS 与 IBM MQ 链接起来。 有关 XA 切换结构的更多信息，请参阅 XA 切换结构。

提供样本源代码文件，使您能够为其他事务消息开发 XA 切换。 所提供的切换装入模块的名称列在第 912 页的表 149 中。

表 149: *AIX* 上 *CICS* 应用程序的基本代码 *:XA* 初始化例程

描述	**C（源代码）**	**C (exec) - 添加到 XAD.Stanza**
XA 初始化例程	amqzscix.c	*amqzsc* - CICS 表示 AIX

使用产品随附的 IBM MQ 切换装入文件 *amqzsc* 的预构建版本。

始终将 C 事务与线程安全 IBM MQ 库 *libmqm_r.a* 链接。 以及具有 COBOL 库 *libmqmcb_r.a* 的 *COBOL* 事务。

You can find more information about supporting CICS transactions in the 管理 IBM MQ System Administration Guide.

AIX *TXSeries CICS* 支持

IBM MQ 上的 AIX 使用 XA 接口支持 TXSeries CICS。 确保将 CICS 应用程序链接到 IBM MQ 库的线程版本。

您可以使用 IBM COBOL Set for AIX 或 Micro Focus COBOL 来运行 CICS 程序。 以下部分描述了在 IBM COBOL Set for AIX 和 Micro Focus COBOL 上运行 CICS 程序之间的差别。

在 C 或 COBOL 中写入装入到同一 CICS 区域中的 IBM MQ 程序。 您不能将 C 和 COBOL MQI 调用组合到同一个 CICS 区域中。 大多数使用第二语言的 MQI 调用都会失败，原因码为 `MQRC_HOBJ_ERROR`。

使用 IBM COBOL Set for AIX 准备 CICS COBOL 程序

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

要使用 IBM COBOL，请按照以下步骤执行操作：

1. 导出以下环境变量：

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                -e _iwz_cobol_main  \
```

   其中 `LIB` 是编译器伪指令。

2. 通过输入以下内容来转换、编译和链接程序：

```
cicstcl -l IBMCOB yourprog.ccp
```

使用 Micro Focus COBOL 准备 CICS COBOL 程序

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

要使用 Micro Focus COBOL，请遵循以下步骤：

1. 使用以下命令将 IBM MQ COBOL 运行时库模块添加到运行时库：

```
cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

   注: 通过 `cicsmkcobol`，IBM MQ 不允许使用 C 编程语言从 COBOL 应用程序进行 MQI 调用。

   如果现有应用程序具有任何此类调用，建议将这些函数从 COBOL 应用程序移动到自己的库中，例如 `myMQ.so`。 移动函数后，在为 CICS 构建 COBOL 应用程序时，请勿包含 IBM MQ 库 `libmqmcbrt.o`。

此外，如果 COBOL 应用程序未进行任何 COBOL MQI 调用，请勿将 `libmqmz_r` 与 `cicsmkcobol` 链接起来。

这将创建 Micro Focus COBOL 语言方法文件，并使 CICS 运行时 COBOL 库能够在 UNIX and Linux 系统上调用 IBM MQ 。

注: 仅当安装以下产品之一时运行 `cicsmkcobol`：

- Micro Focus COBOL 的新版本或发行版
- CICS for AIX 的新版本或发行版
- 任何受支持的数据库产品的新版本或发行版（仅限 COBOL 事务）
- IBM MQ 的新版本或发行版

2. 导出以下环境变量：

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. 通过输入以下内容来转换、编译和链接程序：

```
cicstcl -l COBOL -e yourprog.ccp
```

准备 CICS C 程序

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

使用标准 CICS 工具来构建 CICS C 程序：

1. 导出以下环境变量之一：
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" 导出 USERLIB
2. 通过输入以下内容来转换、编译和链接程序：

```
cicstcl -l C amqscic0.ccs
```

CICS C 样本事务

AIX IBM MQ 事务的样本 C 源代码由 AMQSCIC0.CCS 提供。 事务从传输队列 SYSTEM.SAMPLE.CICS 中读取消息。WORKQUEUE 位于缺省队列管理器上，并使用消息的传输头中包含的队列名称将它们放置到本地队列上。 任何故障都将发送到队列 SYSTEM.SAMPLE。CICS。DLQ。 使用样本 MQSC 脚本 AMQSCIC0.TST 创建这些队列和样本输入队列。

IBM i 在 IBM i 上构建过程化应用程序

IBM i 出版物描述了如何从您编写的程序构建可执行应用程序，以便在 iSeries 或 System i 系统上与 IBM i 一起运行。

本主题描述了在构建 IBM MQ for IBM i 过程应用程序以在 IBM i 系统上运行时，必须执行的其他任务以及对标准任务的更改。 支持 COBOL、C、C++、Java 和 RPG 编程语言。 有关准备 C++ 程序的信息，请参阅使用 C++。 有关准备 Java 程序的信息，请参阅使用 IBM MQ classes for Java。

创建可执行 IBM MQ for IBM i 应用程序必须执行的任务取决于编写源代码所采用的编程语言。 除了在源代码中对 MQI 调用进行编码之外，必须添加相应语言语句以包含您正使用语言的 IBM MQ for IBM i 数据定义文件。 请熟悉这些文件的内容。 请参阅 第 670 页的『IBM MQ 数据定义文件』以获取完整描述。

IBM i 在 *IBM i* 中准备 *C* 程序

IBM MQ for IBM i 支持最大为 100 MB 大小的消息。 ILE C 中编写的应用程序支持大于 16 MB 的 IBM MQ 消息，需要使用 *Teraspace* 编译器选项为这些消息分配足够的内存。

有关 C 编译器选项的更多信息，请参阅 *WebSphere Development Studio ILE C/C++ Programmer's Guide*。

要编译 C 模块，可使用 IBM i 命令，CRTCMOD。 编译时，请确保含有包含文件 (QMQM) 的库位于库列表中。

然后，必须使用 CRTPGM 命令将编译器的输出与服务程序绑定。

表 *150:* 非线程环境中的 *CRTPGM* 示例	
命令	**程序/出口类型**
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM)`	针对 C 的服务器或客户机

其中 *pgmname* 是程序的名称。

线程环境的命令示例如下：

表 *151:* 线程环境中的 *CRTPGM* 示例	
命令	**程序/出口类型**
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM_R)`	针对 C 的服务器或客户机

其中 *pgmname* 是程序的名称。

下表列出了在非线程环境和线程环境中的 IBM i 上准备 C 程序时所需的库。

表 *152:* 非线程环境	
库文件	**程序/出口类型**
LIBMQM	针对 C 的服务器
LIBMQIC 和 LIBMQM	针对 C 的客户机

表 *153:* 线程环境	
库文件	**程序/出口类型**
LIBMQM_R	针对 C 的服务器
LIBMQIC_R & LIBMQM_R	针对 C 的客户机

IBM i 在 *IBM i* 中准备 *COBOL* 程序

了解如何在 IBM i 中准备 COBOL 程序以及从 COBOL 程序中访问 MQI 的方法。

关于此任务

要从 COBOL 程序中访问 MQI，IBM MQ for IBM i 提供了服务程序所提供的绑定过程调用接口。 此接口提供了对 IBM MQ for IBM i 中所有 MQI 函数的访问以及线程应用程序支持。 此接口只能与 ILE COBOL 编译器结合使用。

将使用标准 COBOL CALL 语法来访问 MQI 函数。

包含要用于 MQI 的命名常量和结构定义的 COBOL 副本文件都包含在源物理文件 QMQM/QCBLLESRC 中。

COBOL 副本文件使用单引号字符 (') 作为字符串定界符。 IBM i COBOL 编译器假定定界符是引号 (")。要防止编译器生成警告消息，请在命令 **CRTCBLPGM**、**CRTBNDCBL** 或 **CRTCBLMOD** 上指定 `OPTION(*APOST)`。

要使编译器接受单引号字符 (') 作为 COBOL 副本文件中的字符串定界符，请使用编译器选项 \APOST。

注: 在 IBM MQ 9.0 或更高版本中未提供动态调用接口。

要使用绑定过程调用接口，请完成以下步骤。

过程

1. 通过使用 **CRTCBLMOD** 编译器并指定以下参数来创建模块：

```
LINKLIT(*PRC)
```

2. 使用 **CRTPGM** 命令并指定相应参数来创建程序对象：

   对于非线程应用程序：

```
BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
```

   对于线程应用程序：

```
BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
```

   注： 除使用 V4R4 ILE COBOL 编译器创建且在 PROCESS 语句中包含 THREAD(SERIALIZE) 选项的程序以外，COBOL 程序不得使用 IBM MQ 线程库。 即使已通过这种方式使 COBOL 程序成为线程安全程序，在设计应用程序时也请小心谨慎，因为 THREAD(SERIALIZE) 会在模块级别强制序列化 COBOL 过程并可能影响整体性能。

   有关更多信息，请参阅 *WebSphere Development Studio: ILE COBOL Programmer's Guide* 和 *WebSphere Development Studio: ILE COBOL Reference*。

   有关编译 CICS 应用程序的更多信息，请参阅 *CICS for IBM i Application Programming Guide* (SC41-5454)。

IBM i 在 *IBM i* 中准备 *CICS* 程序

了解在 IBM i 中准备 CICS 程序时所需的步骤。

要创建包含 EXEC CICS 语句和 MQI 调用的程序，请执行以下步骤：

1. 如有必要，使用 CRTCICSMAP 命令准备映射。
2. 将 EXEC CICS 命令转换为本机语言语句。 针对 C 程序使用 CRTCICSC 命令。 针对 COBOL 程序使用 CRTCICSCBL 命令。

   在 CRTCICSC 或 CRTCICSCBL 命令中包含 CICSOPT(*NOGEN)。 这会停止处理，以使您能够包含相应的 CICS 和 IBM MQ 服务程序。 缺省情况下，此命令将代码放入 QTEMP/QACYCICS 中。
3. 使用 CRTCMOD 命令（针对 C 程序）或 CRTCBLMOD 命令（针对 COBOL 程序）编译源代码。
4. 使用 CRTPGM 将已编译的代码与相应的 CICS 和 IBM MQ 服务程序进行链接。 这将创建可执行程序。

以下是此类代码的示例（它编译随附的 CICS 样本程序）：

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i 在 *IBM i* 中准备 *RPG* 程序

如果您使用的是 IBM MQ for IBM i，那么可以采用 RPG 来编写应用程序。

有关更多信息，请参阅第 962 页的『采用 RPG 对 IBM MQ 程序进行编码（仅限 IBM i）』和 IBM i 应用程序编程参考 (ILE/RPG)。

IBM i IBM i 的 SQL 编程注意事项

了解使用 SQL 在 IBM i 上构建应用程序时所需的步骤。

如果程序包含 EXEC SQL 语句和 MQI 调用，请执行以下步骤：

1. 将 EXEC SQL 命令转换为本机语言语句。 针对 C 程序使用 CRTSQLCI 命令。 针对 COBOL 程序使用 CRTSQLCBLI 命令。

   在 CRTSQLCI 或 CRTSQLCBLI 命令中包含 OPTION(*NOGEN)。 这会停止处理，以使您能够包含相应的 IBM MQ 服务程序。 缺省情况下，此命令将代码放入 QTEMP/QSQLTEMP 中。
2. 使用 CRTCMOD 命令（针对 C 程序）或 CRTCBLMOD 命令（针对 COBOL 程序）编译源代码。
3. 使用 CRTPGM 将已编译的代码与相应的 IBM MQ 服务程序进行链接。 这将创建可执行程序。

以下是此类代码的示例（它编译库 SQLUSER 中的程序 SQLTEST）：

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux 在 Linux 上构建过程化应用程序

此信息描述了构建要运行的 IBM MQ for Linux 应用程序时，必须执行的其他任务以及对标准任务的更改。

支持 C 和 C++。 有关准备 C++ 程序的信息，请参阅使用 C++。

Linux 在 Linux 中准备 C 程序

在 *MQ_INSTALLATION_PATH*/samp/bin 目录中提供了预编译的 C 程序。 要从源代码构建样本，请使用 gcc 编译器。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

在正常环境中工作。 有关 64 位应用程序编程的更多信息，请参阅 64 位平台上的编码标准。

链接库

下表列出了在 Linux 上准备 C 程序时所需的库。

- 您需要将程序与 IBM MQ 提供的相应的库链接起来。

  在非线程环境中，链接到以下某个库：

库文件	程序/出口类型
libmqm.so	针对 C 的服务器
libmqic.so & libmqm.so	针对 C 的客户机

  在线程环境中，链接到以下某个库：

库文件	程序/出口类型
libmqm_r.so	针对 C 的服务器
libmqic_r.so & libmqm_r.so	针对 C 的客户机

注：

1. 不能链接到多个库。 即，不能同时链接到线程库和非线程库。

2. 如果要编写可安装的服务（请参阅 管理 以获取更多信息），那么需要链接到 `libmqmzf.so` 库。
3. 如果要生成一个应用程序以通过符合 XA 的事务管理器（例如，IBM TXSeries、Encina 或 BEA Tuxedo）进行外部协调，那么需要链接到非线程应用程序中的 `libmqmxa.so`（如果事务管理器将“long”类型视为 64 位，则为 `libmqmxa64.so`）和 `libmqz.so` 库，以及链接到线程应用程序中的 `libmqmxa_r.so`（或 `libmqmxa64_r.so`）和 `libmqz_r.so` 库。
4. 您必须在链接其他任何产品库之前链接 IBM MQ 库。

Linux 构建 *31* 位应用程序

本主题包含用于在各种环境中构建 31 位程序的命令示例。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

31 位非线程 C 客户机应用程序

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

31 位线程 C 客户机应用程序

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

31 位非线程 C 服务器应用程序

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

31 位线程 C 服务器应用程序

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

31 位非线程 C++ 客户机应用程序

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

31 位线程 C++ 客户机应用程序

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

31 位非线程 C++ 服务器应用程序

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

31 位线程 C++ 服务器应用程序

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

31 位非线程 C 客户机出口

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

31 位线程 C 客户机出口

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

31 位非线程 C 服务器出口

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

31 位线程 C 服务器出口

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux 构建 *32* 位应用程序

本主题包含用于在各种环境中构建 32 位程序的命令示例。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

32 位非线程 C 客户机应用程序

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

32 位线程 C 客户机应用程序

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

32 位非线程 C 服务器应用程序

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

32 位线程 C 服务器应用程序

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

32 位非线程 C++ 客户机应用程序

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32 位线程 C++ 客户机应用程序

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

32 位非线程 C++ 服务器应用程序

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

32 位线程 C++ 服务器应用程序

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

32 位非线程 C 客户机出口

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

32 位线程 C 客户机出口

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

32 位非线程 C 服务器出口

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

32 位线程 C 服务器出口

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux 构建 *64* 位应用程序

本主题包含用于在各种环境中构建 64 位程序的命令示例。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

64 位非线程 C 客户机应用程序

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

64 位线程 C 客户机应用程序

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

64 位非线程 C 服务器应用程序

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

64 位线程 C 服务器应用程序

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

64 位非线程 C++ 客户机应用程序

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 位线程 C++ 客户机应用程序

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

64 位非线程 C++ 服务器应用程序

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64 位线程 C++ 服务器应用程序

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64 位非线程 C 客户机出口

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

64 位线程 C 客户机出口

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

64 位非线程 C 服务器出口

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

64 位线程 C 服务器出口

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
```

```
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux 在 *Linux* 中准备 *COBOL* 程序

了解有关在 Linux 中准备 COBOL 程序以及使用 IBM COBOL for Linux on x86 和 Micro Focus COBOL 来准备 COBOL 程序的信息。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

1. 32 位 COBOL 副本安装在以下目录中:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   并在以下位置创建符号链接:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 在 64 位平台上， 64 位 COBOL 副本安装在以下目录中:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. 在以下示例中，将 COBCPY 设置为:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   针对 32 位应用程序，以及:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   (对于 64 位应用程序)。

您需要将程序与以下某项链接起来:

库文件	程序/出口类型
libmqmcb.so	针对 COBOL 的服务器
libmqicb.so	针对 COBOL 的客户机
libmqmcb_r.so	针对 COBOL 的服务器（线程应用程序）
libmqicb_r.so	针对 COBOL 的客户机（线程应用程序）

在 x86 上使用 IBM COBOL for Linux 来准备 COBOL 程序

样本 COBOL 程序随 IBM MQ 提供。 要编译这样的程序，请输入以下列表中相应的命令:

32 位非线程服务器应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

32 位非线程客户机应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

32 位线程服务器应用程序

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

32 位线程客户机应用程序

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

使用 Micro Focus COBOL 准备 COBOL 程序

在编译程序之前，如下所示设置环境变量：

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

要使用 Micro Focus COBOL 在受支持的情况下编译 32 位 COBOL 程序，请输入：

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

要使用 Micro Focus COBOL 编译 64 位 COBOL 程序，请输入：

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

其中 amqsput 是样本程序

有关所需环境变量的说明，请参阅 Micro Focus COBOL 文档。

Solaris 在 Solaris 上构建过程化应用程序

此信息描述了在构建 IBM MQ for Solaris 应用程序以在 Solaris 下运行时，必须执行的其他任务以及对标准任务的更改。

支持 COBOL、C 和 C++ 编程语言。 有关准备 C++ 程序的信息，请参阅使用 C++。

除在源代码中对 MQI 调用进行编码以外，您还必须添加相应的包含文件。 请熟悉这些文件的内容。 请参阅第 670 页的『IBM MQ 数据定义文件』以获取完整描述。

在本主题中，使用反斜杠 (\) 字符来拆分超过一行的长命令。 请勿输入此字符，将各命令以单行形式输入。

Solaris 在 *Solaris* 中准备 *C* 程序

在 *MQ_INSTALLATION_PATH*/samp/bin 目录中提供了预编译的 C 程序。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

有关 64 位应用程序编程的更多信息，请参阅 64 位平台上的编码标准。

如果要在仅安装了 IBM MQ MQI client for Solaris 的机器上使用程序，请编译这些程序以将其与客户机库 (-lmqic) 进行链接。

如果使用不受支持的编译器 /usr/ucb/cc，那么应用程序可能会成功编译并链接。 但是，在运行应用程序时，如果它尝试连接到队列管理器，那么会失败。

注: 在 Intel 系统上运行时，为符合 FIPS 140-2 操作配置的 32 位 Solaris x86 SSL 和 TLS 客户机将发生故障。 由于未在 Intel 芯片上装入符合 FIPS 140-2 的 GSKit-Crypto Solaris x86 32 位库文件，因此会发生此故障。 在受影响的系统上，将在客户机错误日志中报告错误 AMQ9655。 要解决此问题，请禁用 FIPS 140-2 合规性或重新编译客户机应用程序 64 位，因为 64 位代码不受影响。

链接库

您必须与适合于您的应用程序类型的 IBM MQ 库进行链接：

库文件	程序/出口类型
libmqm.so	针对 C 的服务器
libmqic.so & libmqm.so	针对 C 的客户机

注:

1. 如果编写的是可安装服务（有关进一步信息，请参阅管理），请链接到 `libmqmzf.so` 库。
2. 如果产生的是用于通过 XA 相容事务管理器（例如 IBM TXSeries Encina 或 BEA Tuxedo）进行外部协调的应用程序，那么必须链接到 `libmqmxa.so`（或 `libmqmxa64.so`，如果事务管理器将“long”类型视为 64 位）和 `libmqz.so` 库。
3. 您必须在链接其他任何产品库之前链接 IBM MQ 库。

Solaris 在 *Solaris x86-64* 上构建应用程序

本主题包含用于在 Solaris x86-64 平台上的各种环境中构建程序的命令示例。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

C 客户机应用程序，32 位

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C 客户机应用程序，64 位

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

C 服务器应用程序，32 位

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C 服务器应用程序，64 位

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

C++ 客户机应用程序，32 位

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

C++ 客户机应用程序，64 位

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

C++ 服务器应用程序，32 位

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
```

```
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

C++ 服务器应用程序，64 位

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

C 客户机出口，32 位

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

C 客户机出口，64 位

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

C 服务器出口，32 位

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

C 服务器出口，64 位

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

在 *Solaris SPARC* 上构建应用程序
本主题包含用于在 Solaris SPARC 平台上的各种环境中构建程序的命令示例。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

C 客户机应用程序，32 位

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C 客户机应用程序，64 位

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

C 服务器应用程序，32 位

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C 服务器应用程序，64 位

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

C++ 客户机应用程序，32 位

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic
-lsocket -lnsl -ldl
```

C++ 客户机应用程序，64 位

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

C++ 服务器应用程序，32 位

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

C++ 服务器应用程序，64 位

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

C 客户机出口，32 位

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

C 客户机出口，64 位

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

C 服务器出口，32 位

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

C 服务器出口，64 位

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

Solaris 在 *Solaris* 中准备 *COBOL* 程序

了解如何在 Solaris 中准备 COBOL 程序。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

1. 32 位 COBOL 副本安装在以下目录中：

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   并在以下位置创建符号链接：

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 64 位 COBOL 副本安装在以下目录中：

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. 在以下示例中，将 COBCPY 设置为：

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   （对于 32 位应用程序）以及：

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   （对于 64 位应用程序）。

使用 Micro Focus 编译器来编译程序。 用于声明结构的副本文件在 *MQ_INSTALLATION_PATH*/inc 中：

```
$ export LIB= MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="COBCPY_VALUE"
```

编译 32 位程序：

- $ cob32 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqmcb
  针对 COBOL 的服务器
- $ cob32 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqicb
  针对 COBOL 的客户机
- $ cob32 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqmcb_r
  COBOL 的线程服务器
- $ cob32 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqicb_r
  COBOL 的线程客户机

编译 64 位程序：

- $ cob64 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqmcb
  针对 COBOL 的服务器
- $ cob64 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqicb
  针对 COBOL 的客户机
- $ cob64 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqmcb_r
  COBOL 的线程服务器
- $ cob64 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqicb_r
  COBOL 的线程客户机

其中 *amqs0put0.cbl* 是样本程序。

您必须将程序与以下之一进行链接：

- libmqmcb.so

  针对 COBOL 的服务器
- libmqicb.so

  针对 COBOL 的客户机

Solaris 在 *Solaris* 中准备 *CICS* 程序

了解有关在 Solaris 中准备 CICS 程序的信息。

提供 XA 切换模块，使您能够将 CICS 与 IBM MQ 链接起来：

表 *154: CICS* 应用程序 *(Solaris)* 的基本代码		
描述	**C（源代码）**	**C（可执行代码）**
XA 初始化例程	amqzscix.c	amqzsc - TXSeries for Solaris

请始终将您的事务与线程安全 IBM MQ 库 libmqm.so 进行链接。

您可以在 管理中找到有关支持 CICS 事务的更多信息。

Solaris *TXSeries CICS* 支持

IBM MQ for Solaris 使用 XA 接口支持 TXSeries CICS。

在 C 或 COBOL 中写入装入到同一 CICS 区域中的 IBM MQ 程序。 您不能将 C 和 COBOL MQI 调用组合到同一个 CICS 区域中。 大多数使用第二语言的 MQI 调用都会失败，原因码为 MQRC_HOBJ_ERROR。

使用 Micro Focus COBOL 准备 CICS COBOL 程序

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

要使用 Micro Focus COBOL，请遵循以下步骤：

1. 使用以下命令将 IBM MQ COBOL 运行时库模块添加到运行时库：

   ```
   cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
   ```

   注： 通过 cicsmkcobol，IBM MQ 不允许使用 C 编程语言从 COBOL 应用程序进行 MQI 调用。

   如果现有应用程序具有任何此类调用，请将这些函数从 COBOL 应用程序移至您自己的库，例如 myMQ.so。 在移动这些函数之后，在为 CICS 构建 COBOL 应用程序时，请勿包含 IBM MQ 库 libmqmcbrt.o 。

   此外，如果 COBOL 应用程序未进行任何 COBOL MQI 调用，请勿将 libmqmz_r 与 cicsmkcobol 链接起来。

   这将创建 Micro Focus COBOL 语言方法文件，并使 CICS 运行时 COBOL 库能够在 UNIX and Linux 系统上调用 IBM MQ 。

   注： 仅当安装以下产品之一时运行 cicsmkcobol：

   - Micro Focus COBOL 的新版本或发行版
   - TXSeries for Solaris 的新版本或发行版
   - 任何受支持的数据库产品的新版本或发行版（仅限 COBOL 事务）
   - IBM MQ 的新版本或发行版

2. 导出以下环境变量：

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. 通过输入以下内容来转换、编译和链接程序：

```
cicstcl -l COBOL -e yourprog.ccp
```

准备 CICS C 程序

使用标准 CICS 工具来构建 CICS C 程序：

1. 导出以下环境变量之一：
   - LDFLAGS = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* 运行库 -lmqm_r" 导出 USERLIB
2. 通过输入以下内容来转换、编译和链接程序：

```
cicstcl -l C amqscic0.ccs
```

CICS C 样本事务

CICS IBM MQ 事务的样本 C 源代码由 AMQSCIC0.CCS 提供。 事务从传输队列 SYSTEM.SAMPLE.CICS 中读取消息。WORKQUEUE 位于缺省队列管理器上，并使用消息的传输头中包含的队列名称将它们放置到本地队列上。 任何故障都将发送到队列 SYSTEM.SAMPLE。CICS。DLQ。 使用样本 MQSC 脚本 AMQSCIC0.TST 创建这些队列和样本输入队列。

Windows 在 Windows 上构建过程化应用程序

Windows 系统出版物描述了如何通过编写的程序来构建可执行的应用程序。

本主题描述了在构建要在 Windows 系统下运行的 IBM MQ for Windows 应用程序时必须执行的其他任务以及对标准任务的更改。 支持 ActiveX、C、C++、COBOL 和 Visual Basic 编程语言。 有关准备 ActiveX 程序的信息，请参阅使用组件对象模型接口 (WebSphere MQ Automation Classes for ActiveX)。 有关准备 C++ 程序的信息，请参阅使用 C++。

为使用 IBM MQ for Windows 创建可执行应用程序而必须执行的任务随编写源代码所采用的编程语言而异。除在源代码中对 MQI 调用进行编码以外，您还必须添加相应的语言语句来针对您使用的语言包含 IBM MQ for Windows 包含文件。 请熟悉这些文件的内容。 请参阅 第 670 页的『IBM MQ 数据定义文件』以获取完整描述。

Windows 在 *Windows 64* 位应用程序

在 IBM MQ for Windows 上同时支持 32 位和 64 位应用程序。 IBM MQ 可执行文件和库文件同时以 32 位和 64 位形式提供，根据您处理的应用程序，使用相应的版本。

可执行文件和库

以下位置同时提供了 IBM MQ 库的 32 位和 64 位版本：

表 *155: IBM MQ* 库的位置

库版本	包含库文件的目录
32 位	*MQ_INSTALLATION_PATH* \Tools\Lib
64 位	*MQ_INSTALLATION_PATH* \Tools\Lib64

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

在迁移后，32 位应用程序通常继续工作。 32 位文件与产品的先前版本存在于同一目录中。

如果要创建 64 位版本，那么必须确保您的环境已配置为使用 *MQ_INSTALLATION_PATH* \Tools\Lib64 中的库文件。 确保 LIB 环境变量未设置为在包含 32 位库的文件夹中进行查找。

Windows 在 *Windows* 中准备 *C* 程序

在典型 Windows 环境中工作；IBM MQ for Windows 没有特殊需求。

有关 64 位应用程序编程的更多信息，请参阅 64 位平台上的编码标准。

- 将您的程序与 IBM MQ 提供的相应库进行链接：

库文件	程序/出口类型
MQ_INSTALLATION_PATH \Tools\Lib\mqm.lib	32 位 C 的服务器
MQ_INSTALLATION_PATH \Tools\Lib\mqic.lib	32 位 C 的客户机
MQ_INSTALLATION_PATH \Tools\Lib\mqicxa.lib	带有事务协调的 32 位 C 的客户机
MQ_INSTALLATION_PATH \Tools\Lib64\mqm.lib	64 位 C 的服务器
MQ_INSTALLATION_PATH \Tools\Lib64\mqic.lib	64 位 C 的客户机
MQ_INSTALLATION_PATH \Tools\Lib64\mqicxa.lib	带有事务协调的 64 位 C 的客户机

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

以下命令提供编译样本程序 amqsget0（使用 Microsoft Visual C++ 编译器）的示例。

对于 32 位应用程序：

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

对于 64 位应用程序：

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

注：

- 如果编写的是可安装服务（请参阅管理以获取进一步信息），那么需要链接到 mqmzf.lib 库。
- 如果产生的是用于通过 XA 相容事务管理器（例如 IBM TXSeries Encina 或 BEA Tuxedo）进行外部协调的应用程序，那么需要链接到 mqmxa.lib 或 mqmxa.lib 库。
- 如果编写的是 CICS 出口，请链接到 mqmcics4.lib 库。
- 您必须在链接其他任何产品库之前链接 IBM MQ 库。

- DLL 必须在您已指定的路径 (PATH) 中。

- 如果尽可能使用小写字符，那么可以在 UNIX and Linux 系统上从 IBM MQ for Windows 移至 IBM MQ，其中需要使用小写字符。

准备 CICS 和事务服务器程序

CICS IBM MQ 事务的样本 C 源代码由 AMQSCIC0.CCS 提供。 使用标准 CICS 设施对其进行构建。 例如，对于 TXSeries for Windows 2000：

1. 设置环境变量（在一行上输入以下代码）：

```
set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. 设置 USERLIB 环境变量：

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. 转换、编译和链接样本程序：

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

这在《*Transaction Server for Windows NT* 应用程序变成指南 *(CICS) V4*》中进行了描述。

您可以在 管理中找到有关支持 CICS 事务的更多信息。

Windows 在 *Windows* 中准备 *COBOL* 程序

使用此信息来了解如何在 Windows 中准备 COBOL 程序以及准备 CICS 和事务服务器程序。

1. 32 位 COBOL 副本安装在以下目录中: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook。
2. 64 位 COBOL 副本安装在以下目录中: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64
3. 在以下示例中，将 CopyBook 设置为：

```
CopyBook
```

   针对 32 位应用程序，以及：

```
CopyBook64
```

   (对于 64 位应用程序)。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

要在 Windows 系统上准备 COBOL 程序，请将您的程序链接到 IBM MQ 提供的以下库之一：

库文件	程序或出口类型
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	用于 Micro Focus COBOL 的 32 位服务器
MQ_INSTALLATION_PATH \Tools\Lib\mqiccb	Micro Focus COBOL 的 32 位客户机
MQ_INSTALLATION_PATH \Tools\Lib64\mqmcb	用于 Micro Focus COBOL 的 64 位服务器
MQ_INSTALLATION_PATH \Tools\Lib64\mqiccb	用于 Micro Focus COBOL 的 64 位客户机

在 MQI 客户机环境中运行程序时，请确保 DOSCALLS 库出现在任何 COBOL 或 IBM MQ 库之前。

使用 Micro Focus COBOL 准备 COBOL 程序

使用 mqmcb.lib 或 mqiccb.lib(而不是 mqmcbb 和 mqiccbb 库) 重新链接任何现有 32 位 IBM MQ Micro Focus COBOL 程序。

例如，要使用 Micro Focus COBOL 编译样本程序 amq0put0：

1. 将 COBCPY 环境变量设置为指向 IBM MQ COBOL 副本（在一行上输入以下代码）：

```
set COBCPY= MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook
```

2. 编译程序以提供对象文件：

```
cobol amq0put0 LITLINK
```

3. 将对象文件链接到运行时系统。
   - 将 LIB 环境变量设置为指向编译器 COBOL 库。
   - 链接对象文件以供在 IBM MQ 服务器上使用：

```
cbllink amq0put0.obj mqmcb.lib
```

   - 或者链接对象文件以供在 IBM MQ 客户机上使用：

```
cbllink amq0put0.obj mqiccb.lib
```

准备 CICS 和事务服务器程序

要使用 IBM VisualAge COBOL 编译和链接 TXSeries for Windows NT V5.1 程序:

1. 设置环境变量（在一行上输入以下代码）：

```
set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. 设置 USERLIB 环境变量：

```
set USERLIB=MQMCBB.LIB
```

3. 转换、编译和链接程序：

```
cicstcl -l IBMCOB myprog.ccp
```

Transaction Server for Windows NT， *V4 Application Programming* 指南对此进行了描述。

要使用 Micro Focus COBOL 来编译和链接 CICS for Windows V5 程序，请执行以下操作：

- 设置 INCLUDE 变量：

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
        drive:\opt\cics\include;%INCLUDE%
```

- 设置 COBCPY 环境变量：

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- 设置 COBOL 选项：

- set
- COBOPTS=/LITLINK /NOTRUNC

并运行以下代码：

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows 在 *Windows* 中准备 *Visual Basic* 程序

在 Windows 上使用 Microsoft Visual Basic 程序时要考虑的信息。

从 IBM MQ 9.0 开始，不推荐使用对 Microsoft Visual Basic 6.0 的 IBM MQ 支持。 IBM MQ classes for .NET 是建议使用的替代技术。 有关更多信息，请参阅开发 .NET 应用程序。

注: 未提供 Visual Basic 模块文件的 64 位版本。

要在 Windows 上准备 Visual Basic 程序：

1. 创建新项目。
2. 将所提供的模块文件 CMQB.BAS 添加到该项目。
3. 如果需要，请添加所提供的其他模块文件：
   - CMQBB.BAS：MQAI 支持
   - CMQCFB.BAS：PCF 支持
   - CMQXB.BAS：通道出口支持
   - CMQPSB.BAS：发布/预订

请参阅第 959 页的『在 Visual Basic 中编码』，以获取有关从 Visual Basic 内使用 MQCONNXAny 调用的信息。

在项目代码中进行任何 MQI 调用之前，请调用过程 MQ_SETDEFAULTS。 此过程设置 MQI 调用所需的缺省结构。

通过设置条件编译变量 *MqType*，指定在编译或运行项目之前是创建 IBM MQ 服务器还是客户机。 将 Visual Basic 项目中的 *MqType* 设置为 1 (对于服务器) 或 2 (对于客户机)，如下所示：

1. 选择“项目”菜单。
2. 选择 *Name* 属性（其中 *Name* 是当前项目的名称）。
3. 选择对话框中“处理”选项卡。
4. 在“条件编译自变量”字段中，针对服务器输入：

   ```
   MqType=1
   ```

   或者针对客户机输入：

   ```
   MqType=2
   ```

相关概念

第 959 页的『在 Visual Basic 中编码』
在 Microsoft Visual Basic 中对 IBM MQ 程序进行编码时要考虑的信息。 Visual Basic 仅在 Windows 上受支持。

相关参考

第 840 页的『将 Visual Basic 应用程序与 IBM MQ MQI client 代码相链接』
您可以将 Microsoft Visual Basic 应用程序与 Windows 上的 IBM MQ MQI client 代码链接在一起。

Windows *SSPI* 安全出口

IBM MQ for Windows 为 IBM MQ MQI client 和 IBM MQ 服务器均提供安全出口。 这是通过使用安全服务编程接口 (SSPI) 来为 IBM MQ 通道提供认证的通道出口程序。 SSPI 提供 Windows 系统的集成安全功能。

安全数据包从 security.dll 或 secur32.dll 加载。 这些 DLL 随操作系统一起提供。

使用 NTLM 认证服务提供单向认证。 使用 Kerberos 认证服务提供双向认证。

安全出口程序以源代码和对象格式提供。 您可以照原样使用对象代码，或者使用源代码作为起点来创建您自己的用户出口程序。

另请参阅第 1033 页的『在 Windows 上使用 SSPI 安全出口』。

安全出口简介

安全出口形成两个安全出口程序之间的安全连接，其中一个程序用于发送“消息通道代理程序”（MCA），另一个用于接收 MCA。

启动安全连接的程序（即，在建立 MCA 会话后获取控制的第一个程序）称为上下文发起方。 伙伴程序称为上下文接受方。

下表显示用于表示上下文发起方及其关联的上下文接受方的某些通道类型。

表 *156:* 上下文发起方及其关联的上下文接受方

上下文发起方	上下文接受方
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

安全出口程序具有两个入口点：

- **SCY_NTLM**

  这使用提供单向认证的 NTLM 认证服务。 NTLM 允许服务器验证其客户机的身份。 它不允许客户机验证服务器的身份，或者一个服务器验证另一个服务器的身份。 NTLM 认证旨在用于假设其中的服务器真实可靠的网络环境。

- **SCY_KERBEROS**

  这使用 Kerberos 相互认证服务。 Kerberos 协议不会假设网络环境中的服务器真实可靠。 网络连接两端的各方可以验证另一方的身份。 即，服务器可以验证客户机和其他服务器的身份，并且客户机可以验证服务器的身份。

安全出口的作用

本主题描述 SSPI 通道出口程序的作用。

所提供的通道出口程序在会话建立时提供伙伴程序的单向或双向（相互）认证。 对于特定通道，各出口程序具有关联的主体（类似于用户标识，请参阅第 934 页的『IBM MQ 访问控制和 Windows 主体』）。 两个出口程序之间的连接是两个主体之间的关联。

建立底层会话后，即已建立两个安全出口程序（一个用于发送 MCA，另一个用于接收 MCA）之间的安全连接。 操作顺序如下：

1. 各程序与特定主体相关联，例如，作为显式登录操作的结果。
2. 上下文发起方请求与安全包中的合作伙伴（对于 Kerberos，是指命名合作伙伴）的安全连接并接收令牌（称为 token1）。 令牌使用已经建立的底层会话发送到伙伴程序。

3. 伙伴程序（上下文接受方）将 token1 传递到安全包，后者验证上下文发起方是否真实。 对于 NTLM，现已建立连接。
4. 对于 Kerberos 提供的安全出口（即，用于相互认证），安全包还会生成第二个令牌（称为 token2），上下文接受方通过使用底层会话将该令牌返回到上下文发起方。
5. 上下文发起方使用 token2 来验证上下文接受方是否可信。
6. 在此阶段，如果两个应用程序对于合作伙伴的令牌的真实性均满意，那么将建立安全（已认证）连接。

IBM MQ 访问控制和 Windows 主体

IBM MQ 提供的访问控制基于用户和组。 Windows 提供的真实性基于主体，例如用户和 servicePrincipalName (SPN)。 在主体为 servicePrincipalName 的情况下，可能有许多与单个用户相关联的这些主体。

SSPI 安全出口使用相关的 Windows 主体进行认证。 如果 Windows 认证成功，那么出口将与 Windows 主体相关联的用户标识传递到 IBM MQ 以进行访问控制。

与认证相关的 Windows 主体根据所使用的认证类型而异。

- 对于 NTLM 认证，上下文发起方的 Windows 主体是与正在运行的进程相关联的用户标识。 由于此认证为单向，因此与上下文接受方相关联的主体无关。
- 对于 CLNTCONN 通道上的 Kerberos 认证，Windows 主体是与正在运行的进程相关联的用户标识。 否则，Windows 主体是通过向 QueueManagerName 添加前缀形成的 servicePrincipalName。

```
ibmMQSeries/
```

z/OS 在 z/OS 上构建过程化应用程序

CICS、IMS 和 z/OS 出版物描述了如何构建要在这些环境中运行的应用程序。

此主题集合描述其他任务以及对标准任务的更改，在为这些环境构建 IBM MQ for z/OS 应用程序时必须执行这些更改。 支持 COBOL、C、C++、汇编语言和 PL/I 编程语言。 （有关构建 C++ 应用程序的信息，请参阅使用 C++。）

为创建可执行 IBM MQ for z/OS 应用程序而必须执行的任务同时取决于编写程序所采用的编程语言和应用程序将运行所在的环境。

除在程序中对 MQI 调用进行编码以外，请添加相应的语言语句来针对您使用的语言包含 IBM MQ for z/OS 数据定义文件。 请熟悉这些文件的内容。 请参阅 第 670 页的『IBM MQ 数据定义文件』以获取完整描述。

注

名称 **thlqual** 是 z/OS 上的安装库的高级限定符。

z/OS 准备要运行的程序

在为 IBM MQ 应用程序编写程序以创建可执行应用程序之后，您必须编译或组合该程序，然后编辑产生的对象代码与 IBM MQ for z/OS 针对其支持的各环境提供的存根程序之间的链接。

您如何准备程序同时取决于应用程序运行所在的环境（批处理、CICS、IMS（BMP 或 MPP）、Linux 或 UNIX 系统服务）以及 z/OS 安装上数据集的结构。

第 940 页的『动态调用 IBM MQ 存根』描述在程序中进行 MQI 调用的替代方法，以便无需编辑 IBM MQ 存根的链接。 此方法并非对于所有语言和环境都适用。

请勿编辑比程序运行所在的 IBM MQ for z/OS 版本级别更高的存根程序的链接。 例如，不得编辑在 MQSeries for OS/390 V5.2 上运行的程序与 IBM MQ for z/OS V7 随附的存根程序之间的链接。

z/OS 构建 *64* 位 *C* 应用程序

在 z/OS 中，使用 LP64 编译器和绑定程序选项来构建 64 位 C 应用程序。 IBM MQ for z/OS *cmqc.h* 头文件识别何时将此选项提供给编译器，何时生成适用于 64 位操作的 IBM MQ 数据类型和结构。

必须对使用此选项构建的 C 代码进行构建以使用适于所需协调语义的动态链接库 (DLL)。 将已编译的代码与在每个协调语义所需要的备卡名称中定义的相应备卡绑定以显示所需要的特定 DLL。

表 *157:* 每个协调语义所需要的备卡名称

协调	**备卡名称**
单阶段落实 MQI	CSQBMQ2X
使用 RRS 协调的两阶段落实，使用 RRS 动词	CSQBRR2X
使用 RRS 协调的两阶段落实，使用 MQI 动词	CSQBRI2X

使用 *z/OS XL C/C++* 随附的 EDCQCB JCL 过程来构建单阶段落实 IBM MQ 程序作为批处理作业，如下所示：

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',             < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',             < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//         DD
//         DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

要在 z/OS Unix 系统服务中构建 RRS 协调的程序，请按照如下编译和链接：

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS 构建 *z/OS* 批处理应用程序

了解如何构建 z/OS 批处理应用程序以及在执行此操作时要考虑的步骤。

要为在 z/OS 批处理下运行的 IBM MQ for z/OS 构建应用程序，请创建执行以下任务的作业控制语言 (JCL)：

1. 编译（或组合）程序以产生对象代码。 供编译的 JCL 必须包含 SYSLIB 语句，这些语句使编译器能够使用产品数据定义文件。 数据定义在以下 IBM MQ for z/OS 库中提供：
   - 对于 COBOL：**thlqual**.SCSQCOBC
   - 对于汇编语言：**thlqual**.SCSQMACS
   - 对于 C：**thlqual**.SCSQC370
   - 对于 PL/I：**thlqual**.SCSQPLIC
2. 对于 C 应用程序，预链接在步骤 第 935 页的『1』 中创建的对象代码。
3. 对于 PL/I 应用程序，使用编译器选项 EXTRN(SHORT)。
4. 链接编辑在步骤 第 935 页的『1』（或步骤 第 935 页的『2』，针对 C 应用程序）中创建的对象代码以产生装入模块。 在编辑代码的链接时，必须包含其中一个 IBM MQ for z/OS 批处理存根程序（CSQBSTUB 或其中一个 RRS 存根程序：CSQBRRSI 或 CSQBRSTB）。

   CSQBSTUB
   由 IBM MQ for z/OS 提供的单阶段落实

   CSQBRRSI
   由 RRS 使用 MQI 提供的两阶段落实

   CSQBRSTB
   由 RRS 直接提供的两阶段落实

   注意：

a. 如果使用 CSQBRSTB，那么还必须编辑您的应用程序与 SYS1.CSSLIB 中的 ATRSCSS 之间的链接。 第 936 页的图 118 和第 936 页的图 119 显示用于执行此操作的 JCL 片段。 存根与语言无关，并且在库 **thlqual**.SCSQLOAD 中提供。

b. 如果您的应用程序在语言环境下运行，那么应确保改用语言环境 DLL 编辑链接，如第 936 页的『使用语言环境构建 z/OS 批处理应用程序』中所述。

5. 将装入模块存储在应用程序装入库中。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

图 *118:* 用于在批处理环境中编辑对象模块链接的 *JCL* 片段，使用单阶段落实

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

图 *119:* 用于在批处理环境中编辑对象模块链接的 *JCL* 片段，使用两阶段落实

要运行批处理或 RRS 程序，必须在 STEPLIB 或 JOBLIB 数据集并置中包含库 **thlqual**.SCSQAUTH 和 **thlqual**.SCSQLOAD。

要运行 TSO 程序，必须在 TSO 会话所使用的 STEPLIB 中包含库 **thlqual**.SCSQAUTH 和 **thlqual**.SCSQLOAD。

要从 UNIX 系统服务 shell 运行 UNIX 系统服务批处理程序，请将库 **thlqual**.SCSQAUTH 和 **thlqual**.SCSQLOAD 添加到 $HOME?.profile 中的 STEPLIB 规范，如下所示：

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS 使用语言环境构建 *z/OS* 批处理应用程序

IBM MQ for z/OS 提供在编辑应用程序链接时必须使用的一组动态链接库 (DLL)。

库有两个变体，应用程序通过这些变体可以使用以下调用接口之一：

- 31 位语言环境调用接口。
- 31 位 XPLINK 调用接口。 z/OS XPLINK 是可用于 C 应用程序的高性能调用约定。

要使用 DLL，应用程序针对所谓的侧板而不是较早版本随附的存根进行绑定或链接。 侧板位于 SCSQDEFS 库（而不是 SCSQLOAD 库）中。

表 *158:* 动态链接库变体			
Commit	**31 位语言环境 DLL**	**31 位 XPLINK DLL**	**等效存根名称**
1 阶段落实 MQI 库	CSQBMQ1	CSQBMQ1X	CSQBSTUB
2 阶段落实，使用 RSS 事务控件动词进行 RSS 协调	CSQBRR1	CSQBRR1X	CSQBRSTB
2 阶段落实，使用 MQI 事务控件动词进行 RRS 协调	CSQBRI1	CSQBRI1X	CSQBRRSI

注: 所有侧板都包含数据转换入口点 MQXCNVC（先前通过包含 CSQASTUB 进行解析）的定义。

常见问题:

- 如果应用程序使用异步消息消耗（MQCB、MQCTL 或 MQSUB 调用），并且未使用先前的 DLL 接口，那么在作业日志上会出现以下消息:

  CSQB001E 在 z/OS 批处理或 USS 中运行的语言环境程序必须使用 IBM MQ 的 DLL 接口

  解决方案：使用侧板而非存根重新构建应用程序，如先前详述。

- 在程序构建时，将会出现以下消息

  IEW2469E 对来自 *your-code* 部分的 *MQAPI-NAME* 的引用的属性与以下项的属性不匹配：
  目标符号

  原因：这意味着您已使用 cmqc.h V701（或更高版本）编译 XPLINK 程序，但是没有与侧板绑定。

  解决方案：更改程序的构建文件，以针对来自 SCSQDEFS 的相应侧板（而不是来自 SCSQLOAD 的存根）进行绑定。

以下样本 JCL 演示如何可以编译 C 程序和如何编辑 C 程序的链接，以使用 31 位语言环境 DLL 调用接口:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

注: 编译使用 **DLL** 选项。 链接编辑使用 **DYNAM=DLL** 选项并引用 **CSQBMQ1** 库。

以下样本 JCL 演示如何可以编译 C 程序和如何编辑 C 程序的链接，以使用 31 位 XPLINK DLL 调用接口:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
```

```
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

注: 编译使用 **XPLINK** 和 **DLL** 选项。 链接编辑使用 **DYNAM=DLL** 选项并引用 **CSQBMQ1X** 库。

确保将编译选项 DLL 添加到模块的每个程序中。 诸如 IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED 之类的消息指示需要检查是否所有程序都已使用 DLL 选项进行编译。

z/OS 在 *z/OS* 中构建 *CICS* 应用程序

在 z/OS 中构建 CICS 应用程序时使用此信息。

要为 IBM MQ for z/OS 构建在 CICS 下运行的应用程序，必须执行以下操作：

- 将程序中的 CICS 命令转换为编写程序的其余部分所采用的语言。
- 编译或组合来自转换程序的输出以产生对象代码。
  - 对于 PL/I 程序，使用编译器选项 EXTRN(SHORT)。
  - 对于 C 应用程序，如果应用程序使用的不是 XPLINK，请使用编译器选项 DEFINE(MQ_OS_LINKAGE=1)。
- 链接编辑对象代码以创建装入模块。

CICS 提供针对其支持的各编程语言按顺序执行这些步骤的过程。

- 对于 CICS Transaction Server for z/OS， *CICS Transaction Server for z/OS* 系统定义指南描述如何使用这些过程，*CICS/ESA* 应用程序编程指南提供了有关转换过程的更多信息。

必须包含以下内容：

- 在编译（或组合）阶段的 SYSLIB 语句中，用于使产品数据定义文件对编译器可用的语句。 数据定义在以下 IBM MQ for z/OS 库中提供：
  - 对于 COBOL：**thlqual**.SCSQCOBC
  - 对于汇编语言：**thlqual**.SCSQMACS
  - 对于 C：**thlqual**.SCSQC370
  - 对于 PL/I：**thlqual**.SCSQPLIC
- 在链接编辑 JCL 中，IBM MQ for z/OS CICS 存根程序 (CSQCSTUB)。 第 938 页的图 120 显示用于执行此操作的 JCL 代码片段。 存根与语言无关，并且在库 **thlqual**.SCSQLOAD 中提供。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

图 *120:* 用于在 *CICS* 环境中编辑对象模块链接的 *JCL* 片段

- 对于高于 CICS TS 3.2 的 CICS 版本，或者，如果要使用 IBM MQ 消息属性 API 或者 IBM MQ API MQCB、MQCTL、MQSTAT、MQSUB 或 MQSUBR，那么必须使用 CICS 提供的存根 DFHMQSTB（而不是 IBM MQ 提供的 CSQCSTUB）来链接编辑对象代码。 有关为 CICS 构建 IBM MQ 程序的更多信息，请参阅 CICS 产品文档中的 API 存根程序以访问 IBM MQ MQI 调用 。

完成这些步骤后，请按通常方式将装入模块存储在应用程序装入库中并将程序定义到 CICS。

在运行 CICS 程序之前，系统管理员必须将其定义为 CICS 作为 IBM MQ 程序和事务，然后可以典型方式运行该程序。

z/OS 构建 *IMS*（*BMP* 或 *MPP*）应用程序

在构建 IMS（BMP 或 MPP）应用程序时，请使用此信息。

如果您构建的是批处理 DL/I 程序，请参阅第 935 页的『构建 z/OS 批处理应用程序』。 要构建在 IMS 下运行的其他应用程序（作为 BMP 或 MPP），请创建执行以下任务的 JCL：

1. 编译（或组合）程序以产生对象代码。 供编译的 JCL 必须包含 SYSLIB 语句，这些语句使编译器能够使用产品数据定义文件。 数据定义在以下 IBM MQ for z/OS 库中提供：
   - 对于 COBOL：**thlqual**.SCSQCOBC
   - 对于汇编语言：**thlqual**.SCSQMACS
   - 对于 C：**thlqual**.SCSQC370
   - 对于 PL/I：**thlqual**.SCSQPLIC
2. 对于 C 应用程序，预链接在步骤第 939 页的『1』中创建的对象模块。
3. 对于 PL/I 程序，使用编译器选项 EXTRN(SHORT)。
4. 对于 C 应用程序，如果应用程序使用的不是 XPLINK，请使用编译器选项 DEFINE(MQ_OS_LINKAGE=1)。
5. 链接编辑在步骤第 939 页的『1』（或步骤第 939 页的『2』，针对 C/370 应用程序）中创建的对象代码以产生装入模块：
   a. 包含 IMS 语言接口模块 (DFSLI000)。
   b. 包含 IBM MQ for z/OS IMS 存根程序 (CSQQSTUB)。 第 939 页的图 121 显示用于执行此操作的 JCL 片段。 存根与语言无关，并且在库 **thlqual**.SCSQLOAD 中提供。

      注: 如果使用的是 COBOL，请选择 NODYNAM 编译器选项以使链接编辑器能够解析对 CSQQSTUB 的引用，除非计划按第 940 页的『动态调用 IBM MQ 存根』中所述使用动态链接。
6. 将装入模块存储在应用程序装入库中。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

图 *121:* 用于在 *IMS* 环境中编辑对象模块链接的 *JCL* 片段

在运行 IMS 程序之前，系统管理员必须将其定义为 IMS 作为 IBM MQ 程序和事务: 然后可以典型方式运行该程序。

z/OS 构建 *z/OS UNIX* 系统服务应用程序

在构建 z/OS UNIX 系统服务应用程序时，请使用此信息。

要为 IBM MQ for z/OS 构建在 UNIX 系统服务下运行的 C 应用程序，请按如下编译和链接您的应用程序：

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

其中 **thlqual** 是您的安装使用的高级限定符。

要运行 C 程序，需要向 .profile 文件中添加以下内容；此文件应位于根目录中：

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

请注意，您需要退出 UNIX 系统服务，然后再次进入 UNIX 系统服务，以便系统识别更改。

如果要运行多个 shell，请在行的开头添加单词 export，即：

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

成功完成此操作之后，即可链接 CSQBSTUB 并发出 IBM MQ 调用。

第 940 页的『动态调用 IBM MQ 存根』描述在程序中进行 MQI 调用的替代方法，以便无需编辑 IBM MQ 存根的链接。 此方法并非对于所有语言和环境都适用。

请勿编辑比程序运行所在的 IBM MQ for z/OS 版本级别更高的存根程序的链接。 例如，不得编辑在 IBM WebSphere MQ for z/OS 7.1 上运行的程序与 IBM MQ for z/OS 8.0 随附的存根程序之间的链接。

z/OS 动态调用 *IBM MQ* 存根

您可以在程序中动态调用存根，而不是使用对象代码来链接编辑 IBM MQ 存根程序。

您可以在批处理、IMS 和 CICS 环境中执行此操作。 RRS 环境中不支持此功能。 如果应用程序使用 RSS 来协调更新，请参阅 第 944 页的『RRS 注意事项』。

但是，此方法有以下缺点：

- 增加程序的复杂性
- 增加程序在运行时所需的存储容量
- 降低程序的性能
- 意味着无法在其他环境中使用相同程序

如果动态调用存根，那么相应的存根程序及其别名在执行时必须可用。 要确保此情况，请包含 IBM MQ for z/OS 数据集 SCSQLOAD：

- 对于批处理和 IMS，在 JCL 的 STEPLIB 并置中。
- 对于 CICS，在 CICS DFHRPL 并置中。

对于 IMS，确保包含动态存根的库 (如 设置 IMS 适配器 中有关安装 IMS 适配器的信息中所述构建) 位于区域 JCL 的 STEPLIB 并置中的数据集 SCSQLOAD 之前。

动态调用存根时，请使用第 940 页的表 159 中显示的名称。 在 PL/I 中，仅声明您的程序中使用的调用名称。

表 *159:* 用于动态链接的调用名称

MQI 调用	**批处理（非 RRS）动态调用名称**	**CICS 动态调用名称**	**IMS 动态调用名称**
MQBACK	CSQBBACK	不支持	不支持
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	不支持
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	不支持	不支持
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH

表 159: 用于动态链接的调用名称 (继续)			
MQI 调用	**批处理（非 RRS）动态调用名称**	**CICS 动态调用名称**	**IMS 动态调用名称**
MQCTL	CSQBCTL	CSQCCTL [1]	不支持
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGet	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

注: 1. 仅当使用 CICS TS 3.2 或更高版本时，这些 API 调用才可用，并且必须使用 CICS 随附的 CSQCSTUB。 对于 CICS TS 3.2，必须应用 APAR PK66866。 对于 CICS TS 4.1，必须应用 APAR PK89844。

有关如何使用此方法的示例，请参阅下图：

- 批处理和 COBOL：请参阅第 942 页的图 122
- CICS 和 COBOL：请参阅第 942 页的图 123
- IMS 和 COBOL：请参阅第 942 页的图 124
- 批处理和汇编程序：请参阅第 943 页的图 125
- CICS 和汇编程序：请参阅第 943 页的图 126
- IMS 和汇编程序：请参阅第 943 页的图 127
- 批处理和 C：第 943 页的图 128
- CICS 和 C：请参阅第 943 页的图 129
- IMS 和 C：请参阅第 944 页的图 130
- 批处理和 PL/I：请参阅第 944 页的图 131
- IMS 和 PL/I：请参阅第 944 页的图 132

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

图 *122:* 在批处理环境中使用 *COBOL* 进行动态链接

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

图 *123:* 在 *CICS* 环境中使用 *COBOL* 进行动态链接

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

图 *124:* 在 *IMS* 环境中使用 *COBOL* 进行动态链接

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

图 *125:* 在批处理环境中使用汇编语言进行动态链接

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

图 *126:* 在 *CICS* 环境中使用汇编语言进行动态链接

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

图 *127:* 在 *IMS* 环境中使用汇编语言进行动态链接

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

图 *128:* 在批处理环境中使用 *C* 语言进行动态链接

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

图 *129:* 在 *CICS* 环境中使用 *C* 语言进行动态链接

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

图 *130:* 在 *IMS* 环境中使用 *C* 语言进行动态链接

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

图 *131:* 在批处理环境中使用 *PL/I* 进行动态链接

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

图 *132:* 在 *IMS* 环境中使用 *PL/I* 进行动态链接

z/OS *RRS* 注意事项

如果应用程序使用 RSS 来协调更新，请考虑使用此信息。

IBM MQ 为需要 RRS 协调的批处理程序提供两个不同存根 - 请参阅 第 817 页的『RRS 批处理适配器』。 后续 API 调用的行为差异在 MQCONN 时间由批处理适配器根据 MQCONN 或 MQCONNX API 上的存根例程所传递的信息进行确定。 这意味着动态 API 调用可用于需要 RRS 协调的批处理程序，前提是通过使用相应的存根与 IBM MQ 进行初始连接。 以下示例对此进行了说明：

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
```

```
                              W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS 调试程序

使用此信息了解如何调试 TSO 和 CICS 程序，以及对 CICS 跟踪的洞察。

对调试 IBM MQ for z/OS 应用程序的主要辅助工具是各 API 调用返回的原因码。 有关这些原因码的列表，包括纠正操作的思路，请参阅：

- IBM MQ for z/OS 消息，完成和原因码 for IBM MQ for z/OS
- 消息和原因码（对于所有其他 IBM MQ 平台）

本主题还建议要在特定环境中使用的其他调试工具。

调试 TSO 程序

以下交互式调试工具可用于 TSO 程序：

- TEST 工具
- VS COBOL II 交互式调试工具
- INSPECT 交互式调试工具，用于 C 和 PL/I 程序

调试 CICS 程序

您可以使用 CICS 执行诊断设施 (CEDF) 来交互测试 CICS 程序，而不必修改程序或程序准备过程。

有关 EDF 的更多信息，请参阅 *CICS Transaction Server for z/OS CICS Application Programming Guide*。

CICS 跟踪

您可能还将发现使用 CICS 跟踪控制事务 (CETR) 来控制 CICS 跟踪活动有所帮助。

有关 CETR 的更多信息，请参阅 *CICS Transaction Server for z/OS CICS-Supplied Transactions* 手册。

要确定 CICS 跟踪是否处于活动状态，请使用 CKQC 面板显示连接状态。 此面板还显示跟踪号。

要解释 CICS 跟踪条目，请参阅第 945 页的表 160。

这些值的 CICS 跟踪条目为 AP0 *xxx*（其中 *xxx* 是启用 CICS 适配器之后指定的跟踪号）。 除 CSQCTEST 以外的所有跟踪条目都由 CSQCTRUE 发出。 CSQCTEST 由 CSQCRST 和 CSQCDSP 发出。

表 *160: CICS* 适配器跟踪条目

名称	描述	跟踪顺序	跟踪数据
CSQCABNT	异常终止	在向 IBM MQ 发出 END_THREAD ABNORMAL 之前。 这是由于任务结束且应用程序可能执行了隐式回退。 在此情况下，END_THREAD 调用中包含 ROLLBACK 请求。	工作单元信息。 在了解工作状态时，可以使用此信息。 （例如，可以根据 DISPLAY THREAD 命令产生的输出或 IBM MQ for z/OS 日志打印实用程序对其进行验证。）
CSQCBACK	同步点回退	在向 IBM MQ for z/OS 发出 BACKOUT 之前。 这是由于来自应用程序的显式回退请求。	工作单元信息。

表 160: CICS 适配器跟踪条目 (继续)

名称	描述	跟踪顺序	跟踪数据
CSQCCCRC	完成代码和原因码	在从 API 调用未成功返回之后。	完成代码和原因码。
CSQCCOMM	同步点落实	在向 IBM MQ for z/OS 发出 COMMIT 之前。 这可能是由于单阶段落实请求或两阶段落实请求的第二阶段。 该请求是由于来自应用程序的显式同步点请求。	工作单元信息。
CSQCEXER	执行解析	在向 IBM MQ for z/OS 发出 EXECUTE_RESOLVE 之前。	发出 EXECUTE_RESOLVE 的工作单元的工作单元信息。 这是再同步过程中的最后一个不确定的工作单元。
CSQCGETW	GET 等待	在发出 CICS 等待之前。	要在其上进行等待的 ECB 的地址。
CSQCGMGD	GET 消息数据	在从 MQGET 成功返回之后。	最多 40 字节的消息数据。
CSQCGMGH	GET 消息句柄	在向 IBM MQ for z/OS 发出 MQGET 之前。	对象句柄。
CSQCGMGI	获取消息标识	在从 MQGET 成功返回之后。	消息的消息标识和相关标识。
CSQCINDL	不确定的列表	在从第二个 INQUIRE_INDOUBT 成功返回之后。	不确定的工作单元列表。
CSQCINDO	仅供 IBM 使用		
CSQCINDS	不确定的列表大小	在从第一个 INQUIRE_INDOUBT 成功返回且不确定的列表不为空之后。	列表的长度。 除以 64 即得出不确定的工作单元的数量。
CSQCINQH	INQ 句柄	在向 IBM MQ for z/OS 发出 MQINQ 之前。	对象句柄。
CSQCLOSH	CLOSE 句柄	在向 IBM MQ for z/OS 发出 MQCLOSE 之前。	对象句柄。
CSQCLOST	处置丢失	在再同步过程中，CICS 通知适配器它已重新启动，因此没有有关进行再同步的工作单元的处置信息可用。	CICS 已知的进行再同步的工作单元的工作单元标识。
CSQCNIND	处置并非不确定	在再同步过程中，CICS 通知适配器进行再同步的工作单元不应处于不确定状态（即，它可能仍在运行）。	CICS 已知的进行再同步的工作单元的工作单元标识。
CSQCNORT	正常终止	在向 IBM MQ for z/OS 发出 END_THREAD NORMAL 之前。 这是由于任务结束，并且应用程序因此可能执行隐式同步点落实。 在此情况下，END_THREAD 调用中包含 COMMIT 请求。	工作单元信息。
CSQCOPNH	OPEN 句柄	在从 MQOPEN 成功返回之后。	对象句柄。
CSQCOPNO	OPEN 对象	在向 IBM MQ for z/OS 发出 MQOPEN 之前。	对象名称。

表 160: CICS 适配器跟踪条目 (继续)

名称	描述	跟踪顺序	跟踪数据
CSQCPMGD	PUT 消息数据	在向 IBM MQ for z/OS 发出 MQPUT 之前。	最多 40 字节的消息数据。
CSQCPMGH	PUT 消息句柄	在向 IBM MQ for z/OS 发出 MQPUT 之前。	对象句柄。
CSQCPMGI	PUT 消息标识	在从 IBM MQ for z/OS 成功执行 MQPUT 之后。	消息的消息标识和相关标识。
CSQCPREP	同步点准备	在两阶段落实处理的第一阶段中向 IBM MQ for z/OS 发出 PREPARE 之前。 也可以从分布式排队组件将此调用作为 API 调用发出。	工作单元信息。
CSQCP1MD	PUTONE 消息数据	在向 IBM MQ for z/OS 发出 MQPUT1 之前。	最多 40 字节的消息数据。
CSQCP1MI	PUTONE 消息标识	在从 MQPUT1 成功返回之后。	消息的消息标识和相关标识。
CSQCP1ON	PUTONE 对象名称	在向 IBM MQ for z/OS 发出 MQPUT1 之前。	对象名称。
CSQCRBAK	已解析的回退	在向 IBM MQ for z/OS 发出 RESOLVE_ROLLBACK 之前。	工作单元信息。
CSQCRCMT	已解析的落实	在向 IBM MQ for z/OS 发出 RESOLVE_COMMIT 之前。	工作单元信息。
CSQCRMIR	RMI 响应	在从特定调用返回到 CICS RMI（资源管理器接口）之前。	架构式 RMI 响应值。 其含义取决于调用的类型。 *CICS Transaction Server for z/OS Customization Guide* 中记录了这些值。 要确定调用的类型，请查看 CICS RMI 组件产生的先前跟踪条目。
CSQCRSYN	再同步	在对任务启动再同步过程之前。	CICS 已知的进行再同步的工作单元的工作单元标识。
CSQCSETH	SET 句柄	在向 IBM MQ for z/OS 发出 MQSET 之前。	对象句柄。
CSQCTASE	仅供 IBM 使用		
CSQCTEST	跟踪测试	在 EXEC CICS ENTER TRACE 调用中用于验证用户提供的跟踪号和连接的跟踪状态。	无数据。
CSQCDCFF	仅供 IBM 使用		

处理过程程序错误

本信息说明与应用程序 MQI 调用关联的错误，这些错误可能是应用程序进行调用时产生的，也可能是将其消息传递给最终目标时产生的。

只要可能，队列管理器会在进行 MQI 调用时立即返回所有错误。 这些属于本地确定的错误。

向远程队列发送消息时，进行 MQI 调用时错误可能不明显。 在此情况下，标识错误的队列管理器会通过向原始程序发送另一条消息来报告这些错误。 这些属于远程确定的错误。

本地确定的错误

有关本地确定的错误的信息，包括：MQI 调用失败、系统中断以及包含不正确数据的消息。

队列管理器可以立即报告的三种常见错误原因是：

- MQI 调用失败；例如，由于队列已满导致调用失败
- 应用程序所依赖的系统的某部分运行中断；例如，队列管理器中断
- 无法成功处理包含数据的消息

如果使用的是异步放入功能，那么将不会立即报告错误。 使用 MQSTAT 调用可以检索上一个异步放置操作的状态信息。

MQI 调用失败

队列管理器可以立即报告 MQI 调用的编码中的任何错误。 它使用预定义的返回码完成此操作。 返回码分为完成代码和原因码。

为显示调用是否成功，队列管理器会在调用完成时返回一个完成代码。 完成代码包含三种，指示调用成功、部分完成和调用失败。 队列管理器还会返回一个原因码，指示调用部分完成或失败的原因。

每个调用的完成代码和原因码会与该调用的描述一起列出在原因码中。 有关更多详细信息（包括更正操作建议），请参阅：

- z/OS IBM MQ for z/OS 消息，完成和原因码 for IBM MQ for z/OS
- 消息和原因码（对于所有其他 IBM MQ 平台）

设计您的程序以处理每个调用可能产生的所有返回码。

System i 中断

如果应用程序所连接的队列管理器必须从系统故障中恢复，那么应用程序可能感知不到任何中断。 但是，您必须设计自己的应用程序，以使其在发生此类中断时能够确保不丢失任何数据。

可用于确保数据保持不变的方法取决于运行队列管理器的平台：

z/OS z/OS

在 CICS 和 IMS 环境中，可以在 CICS 或 IMS 管理的工作单元中实施 MQPUT 和 MQGET 调用。 在批处理环境中，可以使用相同的方式实施 MQPUT 和 MQGET 调用，但是必须使用以下方式声明同步点：

- IBM MQ for z/OS MQCMIT 和 MQBACK 调用（请参阅第 786 页的『落实和回退工作单元』）或
- z/OS 事务管理和可恢复的资源管理器服务 (RRS)，提供两阶段同步点支持。 RRS 允许您在单个逻辑工作单元中同时更新 IBM MQ 和其他支持 RRS 的产品资源，如 Db2 存储的过程资源。 有关 RRS 同步点支持的信息，请参阅第 789 页的『事务管理和可恢复的资源管理器服务』。

IBM i IBM i

您可以在 IBM i 落实控制管理的全局工作单元中进行 MQPUT 和 MQGET 调用。 可以通过使用本机 IBM i COMMIT 和 ROLLBACK 命令或特定语言的命令来声明同步点。 局部工作单元由 IBM MQ 使用 MQCMIT 和 MQBACK 调用管理。

UNIX, Linux, and Windows 系统

在这些环境中，可通过常用方式实施 MQPUT 和 MQGET 调用，但必须使用 MQCMIT 和 MQBACK 调用声明同步点（请参阅第 786 页的『落实和回退工作单元』）。 在 CICS 环境中，禁用了 MQCMIT 和 MQBACK 命令，因为您可以在 CICS 管理的工作单元中实施 MQPUT 和 MQGET 调用。

对于不容丢失的所有数据，请使用持久消息来传递。 如果队列管理器必须从故障中恢复，那么可在队列中恢复持久消息。 ULW 使用 IBM MQ on UNIX, Linux, and Windows 时，应用程序中的 MQGET 或 MQPUT 调用将在填充所有日志文件时失败，并显示消息 MQRC_RESOURCE_PROBLEM。 有关 UNIX, Linux, and Windows 上日志文件的更多信息，请参阅管理。 z/OS 对于 z/OS，请参阅规划 z/OS。

如果操作员在应用程序运行期间停止队列管理器，那么通常会使用停顿选项。 队列管理器将进入停顿状态，在该状态下应用程序可以继续运行，但必须尽快终止。 小的快速应用程序可以忽略该停顿状态并继续，直到

其正常终止。 运行时间较长的应用程序或等待接收消息的应用程序，应当在使用 MQOPEN、MQPUT、MQPUT1 和 MQGET 调用时使用如果停顿则失败选项。 这些选项意味着当队列管理器停顿时调用将失败，但应用程序仍有时间通过发出忽略停顿状态的调用来彻底终止。 此类应用程序还可以落实或放弃所做的更改，然后再终止。

如果强制停止队列管理器（即不停顿，直接停止），那么当应用程序实施 MQI 调用时将收到 MQRC_CONNECTION_BROKEN 原因码。 退出应用程序或在 IBM i IBM MQ for IBM i、 UNIX, Linux, and Windows 系统上，发出 MQDISC 调用。

包含不正确数据的消息

在应用程序中使用工作单元时，如果程序无法成功处理从队列检索的消息，那么将回退 MQGET 调用。

队列管理器会在消息描述符的 *BackoutCount* 字段中保持发生回退的次数。 它会在受影响的每个消息的描述符中保持该计数。 该计算可以提供关于应用程序效率的有价值信息。 回退计数随时间增大的消息表明遭反复拒绝；请设计您的应用程序，以使其分析出现此情况的原因并相应地处理此类消息。

z/OS 在 IBM MQ for z/OS 上，要使回退计数在队列管理器重新启动后继续存在，请将 **HardenGetBackout** 属性设置为 MQQA_BACKOUT_HARDENED；否则，如果重新启动队列管理器，它将无法保留每条消息的准确回退计数。 使用这种方法设置属性会增加额外处理。

在 IBM MQ for IBM i IBM i， Windows 和 UNIX and Linux 系统上，回退计数始终在队列管理器重新启动之后。

z/OS 另外，在 IBM MQ for z/OS 上，从工作单元的队列中除去消息时，您可以标记一条消息以使其在应用程序回退该工作单元后不再可用。 对于标记的消息，会将其视为已在新的工作单元下检索。 您可以使用 MQGMO_MARK_SKIP_BACKOUT 选项标记要跳过回退的消息（在 MQGMO 结构中）当您使用 MQGET 调用时。 有关该方法的更多信息，请参阅第 738 页的『跳过回退』。

使用报告消息进行问题确定

在执行 MQI 调用时，远程队列管理器无法报告诸如“无法将消息放入队列”等错误，但是它可以发送一条报告消息，说明对消息做了怎样的处理。

在您的应用程序中，您可以创建 (MQPUT) 报告消息并选择接收消息的选项（在此情况下，消息由另一个应用程序或某个队列管理器发送）。

创建报告消息

报告消息能够使应用程序告知另一个应用程序：它无法处理发送的消息。

然而，最初必须分析 *Report* 字段，以确定发送消息的应用程序是否希望得到有关任何问题的通知。 确定需要报告消息后，您必须决定：

- 希望包含全部原始消息、仅数据的前 100 个字节还是不包含任何原始消息。
- 怎么处理原始消息。 可以将其丢弃，或者使其进入死信队列。
- 以及是否需要 *MsgId* 和 *CorrelId* 字段的内容。

使用 *Feedback* 字段指示生成报告消息的原因。 将报告消息放入应用程序的应答队列。 请参阅反馈以获取更多信息。

请求和接收 (MQGET) 报告消息

向另一个应用程序发送消息时，除非您完成了用于指示所需反馈的 *Report* 字段，否则不会得到任何问题通知。 请参阅报告字段的结构以了解可用选项。

队列管理器始终将报告消息放置在应用程序的应答队列中，建议您自己的应用程序也执行相同的操作。 在使用报告消息功能时，请在消息的消息描述符中指定应答队列的名称；否则 MQPUT 调用将失败。

您的应用程序中必须包含监视应答队列的过程和处理收到的任何消息的过程。 请谨记，报告消息可以包含全部原始消息、仅原始消息的前 100 个字节，或者不包含任何原始消息。

队列管理器设置报告消息的 *Feedback* 字段，用于指示导致错误的原因；例如，目标队列不存在。 您的程序应当执行相同操作。

有关报告消息的更多信息，请参阅第 16 页的『报告消息』。

远程确定的错误

向远程队列发送消息时，即使本地队列管理器已处理了您的 MQI 调用并且未发现任何错误，其他因素也会影响远程队列管理器处理您消息的方式。

例如，您作为目标的队列可能已满，或甚至可能不存在。 如果您的消息必须由到达目标队列的路由上的其他中间队列管理器处理，那么在此过程中可能会发现错误。

传递消息时出现问题

如果 MQPUT 调用失败，您可以尝试再次将消息放入队列、将其返回给发送方或将其放入死信队列。

每个选项都有各自的度量，但是如果是因为目标队列已满而导致 MQPUT 失败，那么您可能不希望尝试再次放入消息。 在这种情况下，将该消息放入死信队列能够使您在稍后将其传递到正确的目标队列。

重试消息传递

如果为通道设置了 *MsgRetryCount* 和 *MsgRetryInterval* 属性，或者提供了可使用的重试出口程序（其名称保留在通道属性 *MsgRetryExitId* 字段中），那么在将消息放入死信队列之前，远程队列管理器会尝试再次将消息放入队列。

如果 *MsgRetryExitId* 字段为空，那么将使用 *MsgRetryCount* 和 *MsgRetryInterval* 属性中的值。

如果 *MsgRetryExitId* 字段不为空，那么将运行该名称的出口程序。 有关使用您自己的出口程序的更多信息，请参阅第 875 页的『消息传递通道的通道出口程序』。

将消息返回给发送方

通过请求生成包含所有原始消息的报告消息，可以将消息返回给发送方。

有关报告消息选项的详细信息，请参阅第 16 页的『报告消息』。

使用死信（未送达消息）队列

当队列管理器无法传递消息时，它会尝试将消息放入其死信队列。 应该在安装队列管理器时定义此队列。

与队列管理器一样，您的程序也可以以同样的方式使用死信队列。 通过打开队列管理器对象（使用 MQOPEN 调用）并查询有关 **DeadLetterQName** 属性（使用 MQINQ 调用）的信息，您可以查找死信队列的名称。

当队列管理器将消息放入该队列时，它会向该消息添加一个头，死信消息头 (MQDLH) 结构中描述了该头的格式；请参阅 MQDLH - 死信消息头。 该头中包含目标队列的名称以及将消息放入死信队列的原因。 必须首先除去该头并解决存在的问题，然后才可以将消息放入期望的队列。 另外，队列管理器会更改消息描述符 (MQMD) 的 *Format* 字段来表面消息包含 MQDLH 结构。

MQDLH 结构

建议您向放入死信队列的所有消息都添加 MQDLH 结构；但是，如果您希望使用特定 IBM MQ 产品提供的死信处理程序，那么必须向您的消息添加 MQDLH 结构。

为消息添加头可能会导致消息对死信队列来说过长，因此请始终确保消息的长度不超过死信队列允许的最大大小，至少是 MQ_MSG_HEADER_LENGTH 常量的值。 队列允许的最大消息大小由队列的 **MaxMsgLength** 属性值确定。 对于死信队列，请确保将该属性设置为队列管理器允许的最大值。 如果您的应用程序无法传递消息，并且该消息过长，无法放入死信队列，请遵照 MQDLH 结构的描述中提供的建议操作。

请确保死信队列受监控，并且所有到达该队列的消息都能够得到处理。 死信队列处理程序作为批处理实用程序运行，可用于对死信队列中所选的消息执行各种操作。 要获取更多详细信息，请参阅第 951 页的『死信队列处理』。

如果需要进行数据转换，队列管理器会在您针对 MQGET 调用使用 MQGMO_CONVERT 选项时转换头信息。如果放置消息的进程是 MCA，那么会在头后追加原始消息的所有文本。

如果放入死信队列的消息对于该队列来说过长，那么可能会将消息截断。 此情况的可能迹象是：死信队列的消息长度等于队列的 **MaxMsgLength** 属性的值。

死信队列处理
此信息包含使用死信队列处理时的通用编程接口信息。

死信队列处理取决于本地系统需求，但在拟定规范时，请考虑以下因素：

- 由于 MQMD 中 format 字段的值为 MQFMT_DEAD_LETTER_HEADER，因此可以将消息标识为具有死信队列头。
- 在 IBM MQ for z/OS 使用 CICS 时，如果 MCA 将此消息放入死信队列，那么 *PutApplType* 字段为 MQAT_CICS， *PutApplName* 字段为 CICS 系统的 *ApplId* ，后跟 MCA 的事务名称。
- 将消息路由到死信队列的原因包含在死信队列头的 *Reason* 字段。
- 死信队列头中包含目标队列名称和队列管理器名称的详细信息。
- 死信队列头中包含必须在消息描述符中恢复才能将消息放入目标队列的字段。 这些字段为：
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- 消息描述符除显示三个字段（Encoding、CodedCharSetId 和 Format）外，与原始应用程序的 PUT 相同。

死信队列应用程序必须执行以下一个或多个操作：

- 检查 *Reason* 字段。 MCA 可能会因以下原因而放入消息：
  - 消息的长度超过通道的最大消息大小

    原因为 MQRC_MSG_TOO_BIG_FOR_CHANNEL
  - 消息无法放入其目标队列

    原因为 MQPUT 操作可能返回的任意 MQRC_* 原因
  - 用户出口已请求此操作

    用户出口提供的原因码，或者缺省值 MQRC_SUPPRESSED_BY_EXIT
- 尝试将消息转发给可能的预期目标。
- 在确定转移的原因，但无法立即纠正的情况下，在丢弃消息之前将其保留的时间长度。
- 为管理员提供指示信息，使其纠正已确定的问题。
- 丢弃已受损或者无法进行处理的消息。

您可以通过以下两种方式处理从死信队列恢复的消息：

1. 如果消息针对本地队列：
   - 执行抽取应用程序数据所需的任何代码转换
   - 如果是局部函数，那么针对该数据执行代码转换
   - 将最终消息及其恢复的所有消息描述符详细信息放入本地队列
2. 如果消息针对远程队列，那么将消息放入该队列。

有关如何在分布式队列环境中处理未传递消息的信息，请参阅无法传递消息时如何处理？。

多点广播编程

本信息用于了解 IBM MQ 多点广播编程任务，如连接到队列管理器和异常报告。

IBM MQ 多点广播旨在尽可能对用户透明，同时仍与现有的应用程序兼容。 定义 COMMINFO 对象和设置 TOPIC 对象的 **MCAST** 和 **COMMINFO** 参数，意味着现有的 IBM MQ 应用程序无需大量改写即可使用多点广

播。 但是，可能存在一些需要考虑的限制（请参阅第 952 页的『多点广播和 MQI』 以了解更多信息）和安全性问题（请参阅多点广播安全性以了解更多信息）。

多点广播和 MQI

本信息用于了解主要消息队列接口 (MQI) 概念以及它们与 IBM MQ 多点广播的关系。

多点广播预订是非持续的；因为未涉及任何物理队列，没有任何位置用于存储持续预订创建的脱机消息。

应用程序预订多点广播主题后，会得到一个返回的对象句柄，应用程序可以使用该对象句柄或从中执行 MQGET 操作，就像它是到队列的句柄那样。 这意味着，仅支持受管的多点广播预订（使用 MQSO_MANAGED 创建的预订）；换而言之，您不可能在一个队列中进行订阅和“指向”消息。 即，只能从预订调用返回的对象句柄使用消息。 在客户机上，在客户机使用之前消息一直存储在消息缓冲区；有关更多信息，请参阅客户机配置文件的 MessageBuffer 节。 如果客户机跟不上发布速率，那么会根据需要丢弃消息，且最早的消息被首先丢弃。

应用程序是否使用多点广播通常是一种管理决策，通过设置 TOPIC 对象的 MCAST 属性来指定。 如果发布应用程序必须确保不使用多点广播，那么可以使用 MQOO_NO_MULTICAST 选项。 同样，预订应用程序可以通过使用 MQSO_NO_MULTICAST 选项进行预订来确保不使用多点广播。

IBM MQ 多点广播支持使用消息选择器。 应用程序可使用该选择器注册感兴趣且属性满足选择字符串表示的 SQL92 查询的消息。 有关消息选择器的更多信息，请参阅第 25 页的『选择器』。

下表列出了所有主要 MQI 概念以及它们与多点广播的关系：

表 *161: MQI* 概念以及它们与多点广播的关系

MQI 概念	**尝试使用多点广播时的操作**	**原因码**
放入长度为零的消息	已拒绝	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
分组	已拒绝	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
分段	已拒绝	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
分发列表	已拒绝	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	针对主题句柄拒绝：不支持主题的 MQINQ 和 MQSET。	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	对于受管处理接受。 只能查询**当前深度**。	• 如果值为“当前深度”，那么将没有适用的原因码。 • 如果该值不是“当前深度”，那么原因码为 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR。
MQSET	对所有句柄拒绝。	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
事务（XA 或非 XA）	已拒绝	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
消息浏览	已拒绝	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
锁定消息	已拒绝	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
浏览标记项	已拒绝	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
传递上下文	已拒绝	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

表 161: MQI 概念以及它们与多点广播的关系 (继续)		
MQI 概念	**尝试使用多点广播时的操作**	**原因码**
MQPUT1	被拒绝。 对仅含多点广播的主题尝试 MQPUT1 无效。	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
持续预订	如果主题标记为“仅限多点广播”，那么将拒绝，否则将执行非多点广播预订。	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	被拒绝。 如果主题字符串大于 255 个字符，客户机会将其拒绝。	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
执行非受管预订	如果主题标记为“仅限多点广播”，那么将拒绝，否则将执行非多点广播预订。	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	已拒绝	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

以下项是上面表中部分 MQI 概念的扩展，并且提供该表中未包含的部分 MQI 概念的信息：

消息持久性
对于非持续多点广播预订，会以不可恢复的方式传递来自发布程序的持久消息。

消息截断
支持消息截断，这意味着应用程序可以：

1. 发出 MQGET。
2. 获取 MQRC_TRUNCATED_MSG_FAILED。
3. 分配更大的缓冲区。
4. 重新发出 MQGET 以检索消息。

预订到期时间
不支持预订到期时间。 将忽略设置到期时间的任何尝试。

多点广播的高可用性

本信息用于了解 IBM MQ 多点广播连续对等操作；虽然 IBM MQ 连接到 IBM MQ 队列管理器，但消息并不流过该队列管理器。

虽然必须建立到队列管理器的连接才能对多点广播主题对象进行 MQOPEN 或 MQSUB 操作，但消息本身并不流过该队列管理器。 因此，针对多点广播主题对象完成 MQOPEN 或 MQSUB 操作后，即使到队列管理器的连接中断，依旧可以继续传输多点广播消息。 有两种操作方式：

建立到队列管理器的正常连接
存在到队列管理器的连接时，即可实现多点广播通信。 如果连接失败，那么将应用正常的 MQI 规则，例如，多点广播对象句柄的 MQPUT 将返回 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN。

将重新建立到队列管理器的客户机连接

重新连接周期内依旧可以进行多点广播通信。 这意味着即使到队列管理器的连接已中断，放入和使用多点广播消息也不受影响。 客户机会尝试重新连接到队列管理器，如果重新连接失败，那么连接句柄将中断，并且包括多点广播在内的所有 MQI 调用都将失败。 有关更多信息，请参阅客户机自动重新连接

如果任何应用程序明确发出 MQDISC，那么所有多点广播预订和对象句柄都将关闭。

多点广播连续对等操作

客户机之间的对等通信优势之一是消息无需流经队列管理器；因为如果到队列管理器的连接中断，消息传输依旧会继续。 此方式的连续消息需求具有以下限制：

- 必须使用某个 MQCNO_RECONNECT_* 选项建立连接以进行连续操作。 该过程表示虽然通信会话可能中断，但实际的连接句柄并不会中断，而会处于重新连接状态。 如果重新连接失败，此时连接句柄将中断，以阻止所有进一步的 MQI 调用。
- 此方式下仅支持 MQPUT、MQGET、MQINQ 和异步使用。 任何 MQOPEN、MQCLOSE 或 MQDISC 动词都需要重新连接到队列管理器才可以完成。
- 到队列管理器的状态流将停止；因此，队列管理器中的任何状态都可能会失效或丢失。 这意味着客户机可能会发送和接收消息，但不会从队列管理器上了解到任何状态。 有关更多信息，请参阅：多点广播应用程序监视

MQI 中针对多点广播消息传递的数据转换

本信息用于了解针对 IBM MQ 多点广播消息传递的数据转换如何工作。

IBM MQ 多点广播是一种共享的无连接协议，因此对于每个客户机来说不可能针对数据转换发出特定的请求。 预订同一多点广播流的每个客户机都会收到相同的二进制数据；因此，如果需要进行 IBM MQ 数据转换，都会在每个客户机本地执行转换。

数据将在客户机上转换为 IBM MQ 多点广播流量。 如果指定了 **MQGMO_CONVERT** 选项，那么会根据请求完成数据转换。 用户定义格式需要在客户机上安装数据转换出口；有关客户机和服务器软件包中现有库的信息，请参阅第 893 页的『编写数据转换出口』。

有关管理数据转换的信息，请参阅针对多点广播消息传递启用数据转换。

有关数据转换的更多信息，请参阅数据转换。

有关数据转换出口和 `ClientExitPath` 的更多信息，请参阅客户机配置文件的 ClientExitPath 节。

多点广播异常报告

本信息用于了解 IBM MQ 多点广播事件处理程序和报告 IBM MQ 多点广播异常。

IBM MQ 多点广播有助于问题确定，方法是调用事件处理程序以报告使用标准 IBM MQ 事件处理程序机制报告的多点广播事件。

单个多点广播事件可能会导致调用多个 IBM MQ 事件，因为可能存在使用同一多点广播发送器或接收器的多个 MQHCONN 连接句柄。 然而，每个多点广播异常都只会导致每个 IBM MQ 连接调用一个事件处理程序。

IBM MQ MQCBDO_EVENT_CALL 常量能够使应用程序注册回调，以只接收 IBM MQ 事件，而 MQCBDO_MC_EVENT_CALL 能够使应用程序注册回调，以只接收多点广播事件。 如果同时使用这两个常量，那么将同时接收到这两种类型的事件。

请求多点广播事件

IBM MQ 多点广播事件在 `cbd.Options` 字段中使用 MQCBDO_MC_EVENT_CALL 常量。 以下示例演示如何请求多点广播事件：

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

为 cbd.Options 字段指定 MQCBDO_MC_EVENT_CALL 选项时，将仅向事件处理程序发送 IBM MQ 多点广播事件，而不是连接级别事件。 要请求向事件处理程序同时发送这两种类型的事件，应用程序必须在 cbd.Options 字段中指定 MQCBDO_EVENT_CALL 常量并且要指定 MQCBDO_MC_EVENT_CALL 常量，如以下示例所示：

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

如果不使用这些常量，那么只会向事件处理程序发送连接级别事件。

有关 Options 字段的值的更多信息，请参阅选项 (MQLONG)。

多点广播事件格式

IBM MQ 多点广播异常中包含回调函数的 **Buffer** 参数中返回的一些支持信息。 **Buffer** 指针指向指针数组，MQCBC.DataLength 字段指定数组的大小（字节数）。 数组的第一个元素始终指向事件的简短文本描述。 根据事件的类型，可能会提供更多参数。 下表列出了一些异常：

表 *162:* 多点广播事件代码描述

事件代码	描述	附加数据
MQMCEV_PACKET_LOSS	不可恢复的丢包	丢包数目
MQMCEV_HEARTBEAT_TIMEOUT	长时间没有脉动信号的控制分组	N/A
MQMCEV_VERSION_CONFLICT	接收更新协议版本包	N/A
MQMCEV_RELIABILITY	发送器和接收器的不同可靠性方式	N/A
MQMCEV_CLOSED_TRANS	主题传送由 1 个源关闭	N/A
MQMCEV_STREAM_ERROR	在流中检测到错误	N/A
MQMCEV_NEW_SOURCE	新源开始在主题上传输	源结构
MQMCEV_RECEIVE_QUEUE_TRIMMED	由于时间或空间到期而从 PacketQ 中移除的包	删除的包数目
MQMCEV_PACKET_LOSS_NACK_EXPIRE	由于 NACK 到期而导致的不可恢复丢包	丢包数目
MQMCEV_ACK_RETRIES_EXCEEDED	超出 **max_ack_retries** 后从历史记录中移除的包	已移除的包数目
MQMCEV_STREAM_SUSPEND_NACK	已针对该主题接受的流暂挂了 NACK	暂挂流标识 流暂挂的时间（毫秒）
MQMCEV_STREAM_RESUME_NACK	针对流暂挂后恢复的 NACK	流标识
MQMCEV_STREAM_EXPELLED	该主题接受的流由于驱逐请求已被拒绝	流标识
MQMCEV_FIRST_MESSAGE	来自源的第一条消息	消息编号
MQMCEV_LATE_JOIN_FAILURE	无法启动延迟连接会话	N/A
MQMCEV_MESSAGE_LOSS	不可恢复的消息丢失	丢失消息数
MQMCEV_SEND_PACKET_FAILURE	多点广播发送器无法发送多点广播信息包	N/A

表 162: 多点广播事件代码描述 (继续)		
事件代码	**描述**	**附加数据**
MQMCEV_REPAIR_DELAY	多点广播接收器未收到未完成 NAK 的修复包	N/A
MQMCEV_MEMORY_ALERT_ON	接收器接收缓冲区即将填满	缓冲池利用率百分比
MQMCEV_MEMORY_ALERT_OFF	接收器接收缓冲区下降到正常水平	缓冲池利用率百分比
MQMCEV_NACK_ALERT_ON	接收器修复包请求率达到高水位标记	包中当前每秒的修复请求率
MQMCEV_NACK_ALERT_OFF	接收器修复包请求率下降到正常水平	包中当前每秒的修复请求率
MQMCEV_REPAIR_ALERT_ON	发送器修复包发送率达到高水位标记	N/A
MQMCEV_REPAIR_ALERT_OFF	发送器修复包发送率下降到正常状态	N/A
MQMCEV_SHM_DEST_UNUSABLE	检测到发送器主题目标使用的共享内存区域不可用	N/A
MQMCEV_SHM_PORT_UNUSABLE	检测到接收器实例使用的共享内存端口不可用	N/A
MQMCEV_CCT_GETTIME_FAILED	从“协调集群时间”获取时间失败	N/A
MQMCEV_DEST_INTERFACE_FAILURE	发送器主题目标使用的网络接口出现故障，备用网络接口不可用	
MQMCEV_DEST_INTERFACE_FAILOVER	发送器主题目标使用的网络接口出现故障，已成功完成到另一个接口的故障转移	
MQMCEV_PORT_INTERFACE-FAILURE	接收器 rmmPort 使用的网络接口出现故障，备用网络接口不可用（或者也出现故障）	RMM 配置
MQMCEV_PORT_INTERFACE_FAILOVER	接收器 rmmPort 使用的网络接口出现故障，已成功完成到另一个接口的故障转移	RMM 配置

采用 C 进行编码

采用 C 对 IBM MQ 程序进行编码时，请注意以下部分中的信息。

- 第 957 页的『MQI 调用的参数』
- 第 957 页的『具有未定义数据类型的参数』
- 第 957 页的『数据类型』
- 第 957 页的『处理二进制字符串』
- 第 957 页的『处理字符串』
- 第 958 页的『结构的初始值』
- 第 958 页的『动态结构的初始值』
- 第 958 页的『从 C++ 使用』

MQI 调用的参数

只输入且类型为 MQHCONN、MQHOBJ、MQHMSG 或 MQLONG 的参数按值传递；对于所有其他参数，参数的地址按值传递。

每次调用函数时，并非所有按地址传递的参数都需要指定。 在无需特定参数的情况下，可以在调用函数时将空指针指定为参数，从而代替参数数据的地址。 可以进行此操作的参数在调用描述中被识别。

没有任何参数作为函数值返回；在 C 术语中，这意味着所有函数都返回空。

函数的属性由 MQENTRY 宏变量定义；此宏变量的值取决于环境。

具有未定义数据类型的参数

MQGET、MQPUT 和 MQPUT1 函数各自具有未定义数据类型的 **Buffer** 参数。 此参数用于发送和接收应用程序的消息数据。

此类参数在 C 示例中显示为 MQBYTE 的数组。 您可以通过此方式声明参数，但是将其声明为描述消息中数据布局的结构，通常更为方便。 函数参数声明为空指针，因此在调用函数时可以将任何数据的地址指定为参数。

数据类型

所有数据类型都使用 `typedef` 语句进行定义。

对于各数据类型，还会定义对应的指针数据类型。 指针数据类型的名称是以字母 P（表示指针）为前缀的基本或结构数据类型的名称。 指针的属性由 MQPOINTER 宏变量定义；此宏变量的值取决于环境。 以下代码说明如何声明指针数据类型：

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

处理二进制字符串

二进制数据的字符串声明为 MQBYTEn 数据类型之一。

只要复制、比较或设置此类型的字段，就请使用 C 函数 `memcpy`、`memcmp` 或 `memset`：

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

请勿使用字符串函数 strcpy、strcmp、strncpy 或 strncmp，因为这些函数在数据声明为 MQBYTE24 时无法正常工作。

处理字符串

当队列管理器将字符数据返回到应用程序时，队列管理器始终根据字段的已定义长度来使用空白填充字符数据。 队列管理器不返回以 null 结束的字符串，但是可以在输入中使用这些字符串。 因此，在复制、比较或并置此类字符串时，请使用字符串函数 strncpy、strncmp 或 strncat。

请勿使用要求字符串以 null 结束的字符串函数（strcpy、strcmp 和 strcat）。 此外，请勿使用函数 strlen 来确定字符串的长度；改用 sizeof 函数来确定字段的长度。

结构的初始值

包含文件 <cmqc.h> 定义可在声明结构的实例时用于提供这些结构的初始值的各种宏变量。 这些宏变量具有 MQxxx_DEFAULT 形式的名称，其中 MQxxx 表示结构的名称。 请按如下对其进行使用：

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

对于某些字符字段，MQI 定义有效的特定值（例如，对于 *StrucId* 字段或对于 MQMD 中的 *Format* 字段）。 对于各有效值，将会提供两个宏变量：

- 一个宏变量将值定义为具有某个长度的字符串（暗含的 null 除外），该长度与字段的已定义长度相匹配。 例如，符号 ¬ 表示空白字符：

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  将此形式用于 memcpy 和 memcmp 函数。
- 另一个宏变量将值定义为 char 的数组；此宏变量的名称是以 _ARRAY 为后缀的字符串形式的名称。 例如：

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  当使用与 MQMD_DEFAULT 宏变量所提供的值不同的值来声明结构的实例时，使用此形式来初始化字段。

动态结构的初始值

当需要可变数量的结构实例时，通常在使用 calloc 或 malloc 函数动态获取的主存储器中创建实例。

要初始化此类结构中的字段，建议使用以下方法：

1. 使用相应的 MQxxx_DEFAULT 宏变量声明结构的实例以初始化该结构。 此实例成为其他实例的模型：

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                     /* declare model instance */
   ```

   对声明中的 static 或 auto 关键字进行编码，以根据需要提供模型实例静态或动态生命周期。
2. 使用 calloc 或 malloc 函数获取结构的动态实例的存储器：

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                     /* get storage for dynamic instance */
   ```
3. 使用 memcpy 函数将模型实例复制到动态实例：

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                     /* initialize dynamic instance */
   ```

从 C++ 使用

对于 C++ 编程语言，头文件包含仅当使用 C++ 编译器时才含有的以下附加语句：

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
```

```
  }
#endif
```

Windows 在 Visual Basic 中编码

在 Microsoft Visual Basic 中对 IBM MQ 程序进行编码时要考虑的信息。 Visual Basic 仅在 Windows 上受支持。

注: 从 IBM WebSphere MQ 7.0 开始，在 .NET 环境外，对 Visual Basic (VB) 的支持已稳定在 IBM WebSphere MQ 6.0 级别。 向 IBM WebSphere MQ 7.0 或更高版本添加的大多数新函数对于 VB 应用程序不可用。 如果是采用 VB.NET 进行编程，请使用 IBM MQ classes for .NET。 有关更多信息，请参阅开发 .NET 应用程序。

从 IBM MQ 9.0 开始，不推荐使用对 Microsoft Visual Basic 6.0 的 IBM MQ 支持。 IBM MQ classes for .NET 是建议使用的替代技术。

要避免无意间转换在 Visual Basic 和 IBM MQ 之间传递的二进制数据，请使用 MQBYTE 定义而不是 MQSTRING。 CMQB.BAS 定义与 C 字节定义等效的若干新 MQBYTE 类型，并在 IBM MQ 结构中使用这些类型。 例如，对于 MQMD（消息描述符）结构，MsgId（消息标识）定义为 MQBYTE24。

Visual Basic 没有指针数据类型，因此对其他 IBM MQ 数据结构的引用是通过偏移而非指针进行引用的。 声明由两个组件结构组成的复合结构，并且指定调用中的复合结构。 IBM MQ 对 Visual Basic 的支持提供 MQCONNXAny 调用来使其成为可能，并且允许客户机应用程序在客户机连接上指定通道属性。 它接受非类型化结构 (MQCNOCD) 来代替典型 MQCNO 结构。

MQCNOCD 结构是由 MQCNO 后跟 MQCD 组成的复合结构。 此结构在出口头文件 CMQXB 中进行声明。 使用例程 MQCNOCD_DEFAULTS 来初始化 MQCNOCD 结构。 提供了进行 MQCONNX 调用的样本 (amqscnxb.vbp)。

MQCONNXAny 与 MQCONNX 具有相同的参数，不同在于 **ConnectOpts** 参数声明为 Any 数据类型而非 MQCNO 数据类型。 这使函数能够接受 MQCNO 或 MQCNOCD 结构。 此函数在主要头文件 CMQB 中进行声明。

相关概念

第 932 页的『在 Windows 中准备 Visual Basic 程序』
在 Windows 上使用 Microsoft Visual Basic 程序时要考虑的信息。

相关参考

第 840 页的『将 Visual Basic 应用程序与 IBM MQ MQI client 代码相链接』
您可以将 Microsoft Visual Basic 应用程序与 Windows 上的 IBM MQ MQI client 代码链接在一起。

采用 COBOL 进行编码

采用 COBOL 对 IBM MQ 程序进行编码时，请注意以下部分中的信息。

命名常量

常量的名称显示为将下划线字符 (_) 包含作为名称的一部分。 在 COBOL 中，必须使用连字符字符 (-) 来代替下划线。 具有字符串值的常量使用单引号字符 (') 作为字符串定界符。 要使编译器接受此字符，请使用编译器选项 APOST。

副本文件 CMQV 包含以命名常量作为 10 级项的声明。 要使用常量，请显式声明 01 级项，然后使用 COPY 语句在常量的声明中进行复制:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

但是，此方法导致常量即使在未引用的情况下也会占用程序中的存储器。 如果常量包含在同一运行单元内的许多单独程序中，那么将存在常量的多个副本；这可能导致使用大量主存储器空间。 可以通过向 01 级声明添加 GLOBAL 子句来避免此情况：

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

这仅为运行单元内的一个常量集分配存储器；但是，常量可由运行单元内的任意程序引用，而不只是由包含 01 级声明的程序引用。

确保结构对齐

请注意确保传递用于在 MQ 调用时启动的 IBM MQ 结构必须在字边界上对齐。 字边界对于 32 位进程为 4 字节，对于 64 位进程为 8 字节，对于 128 位进程为 16 字节 (IBM i)。

如有可能，请将所有 IBM MQ 结构放在一起，以便其全都边界对齐。

采用 System/390 汇编语言（消息队列接口）进行编码

采用汇编语言对 IBM MQ for z/OS 程序进行编码时，请注意以下部分中的信息。

- 第 960 页的『名称』
- 第 960 页的『使用 MQI 调用』
- 第 960 页的『声明常量』
- 第 961 页的『指定结构的名称』
- 第 961 页的『指定结构的形式』
- 第 961 页的『控制列表』
- 第 961 页的『指定字段的初始值』
- 第 962 页的『编写可重新输入的程序』
- 第 962 页的『使用 CEDF』

名称

调用描述中的参数名称和结构描述中的字段名称以混合大小写的形式显示。 在 IBM MQ 随附的汇编语言宏中，所有名称都为大写形式。

使用 MQI 调用

MQI 是调用接口，因此汇编语言程序必须遵守操作系统链接约定。

具体而言，汇编语言程序在发出 MQI 调用之前，必须将寄存器 R13 指向至少包含 18 个全字的保存区域。 此保存区域为被调用程序提供存储器。 它在调用者寄存器的内容销毁之前存储这些寄存器，并在返回时复原调用者寄存器的内容。

注: 这对于使用 DFHEIENT 宏来设置其动态存储器，但是选择将 R13 的缺省 DATAREG 覆盖其他寄存器的 CICS 汇编语言程序非常重要。 当 CICS 资源管理器接口从存根接收到控制时，它将寄存器的当前内容存储在 R13 指向的地址。 未能为此目的保留保存区域会产生不可预测的结果，并将可能在 CICS 中导致异常终止。

声明常量

大多数常量在宏 CMQA 中声明为等式。

但是，以下常量无法声明为等式，并且在使用缺省选项调用宏时不会包含这些常量：

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE

- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

要包含这些常量，请在调用宏时添加关键字 EQUONLY=NO。

CMQA 防范多重声明，因此可以多次将其包含在内。 但是，仅在首次包含宏时，关键字 EQUONLY 才会生效。

指定结构的名称

为允许声明结构的多个实例，生成该结构的宏使用用户可指定的字符串和下划线字符 (_) 作为各字段名称的前缀。

请在调用宏时指定字符串。 如果不指定字符串，那么宏使用结构的名称来构造前缀：

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

调用描述中的结构声明显示缺省前缀。

指定结构的形式

宏可以通过两种形式之一生成结构声明（由 DSECT 参数控制）：

DSECT=YES
汇编语言 DSECT 指令用于启动新数据段；结构定义紧跟在 DSECT 语句之后。 由于未分配存储器，因此无法初始化。 宏调用中的标签用作数据段的名称；如果未指定标签，那么将使用结构的名称。

DSECT=NO
汇编语言 DC 指令用于定义例程中当前位置的结构。 字段使用值进行初始化，可以通过对宏调用中的相关参数进行编码来指定这些值。 宏调用中未指定值的字段使用缺省值进行初始化。

如果未指定 DSECT 参数，那么采用 DSECT=NO。

控制列表

您可以使用 LIST 参数来控制汇编语言列表中结构声明的外观：

LIST=YES
结构声明出现在汇编语言列表中。

LIST=NO
结构声明不出现在汇编语言列表中。 如果未指定 LIST 参数，那么采用此形式。

指定字段的初始值

您可以指定要用于对结构中的字段初始化的值，方法是将该字段的名称（没有前缀）编码为宏调用中的参数，随附所需的值。

例如，要声明具有使用 MQMT_REQUEST 初始化的 *MsgType* 字段以及使用字符串 MY_REPLY_TO_QUEUE 初始化的 *ReplyToQ* 字段的消息描述符结构，请使用以下代码：

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

如果将命名常量（或等式）指定为宏调用中的值，请使用 CMQA 宏来定义命名常量。 不得将字符串值用单引号 (' ') 引起来。

编写可重新输入的程序

IBM MQ 将其结构同时用于输入和输出。 如果要使程序保持可重新输入，请执行以下操作：

1. 将结构的工作存储器版本定义为 DSECT，或者定义在已定义的 DSECT 内联的结构。 然后，将 DSECT 复制到使用以下方式获取的存储器：
   - 对于批处理和 TSO 程序，STORAGE 或 GETMAIN z/OS 汇编程序宏
   - 对于 CICS，工作存储器 DSECT (DFHEISTG) 或 EXEC CICS GETMAIN 命令

   要正确初始化这些工作存储器结构，请将对应结构的常量版本复制到工作存储器版本。

   注: MQMD 和 MQXQH 结构各自的长度超过 256 字节。 要将这些结构复制到存储器，请使用 MVCL 汇编程序指令。
2. 通过使用 LIST 形式 (MF=L) 的 CALL 宏保留存储器中的空间。 使用 CALL 宏进行 MQI 调用时，请使用 EXECUTE 形式 (MF=E) 的宏（使用先前保留的存储器），如第 962 页的『使用 CEDF』 下的示例中所示。 有关如何执行此操作的更多示例，请参阅 IBM MQ 随附的汇编语言样本程序。

使用汇编语言 RENT 选项来帮助确定程序是否可重新输入。

有关编写可重新输入程序的信息，请参阅 *z/OS MVS Application Development Guide: Assembler Language Program*。

使用 CEDF

如果要使用 CICS 提供的事务 CEDF（CICS 执行诊断设施）来帮助调试程序，请向各 CALL 语句添加 ,VL 关键字，例如：

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

先前示例是可重新输入的汇编语言代码，其中 PARMAREA 是所指定的工作存储器中的区域。

使用 MQI 调用

MQI 是调用接口，因此汇编语言程序必须遵守操作系统链接约定。 具体而言，汇编语言程序在发出 MQI 调用之前，必须将寄存器 R13 指向至少包含 18 个全字的保存区域。 此保存区域为被调用程序提供存储器。 它在调用者寄存器的内容销毁之前存储这些寄存器，并在返回时复原调用者寄存器的内容。

注: 这对于使用 DFHEIENT 宏来设置其动态存储器，但是选择将 R13 的缺省 DATAREG 覆盖其他寄存器的 CICS 汇编语言程序非常重要。 当 CICS 资源管理器接口从存根接收到控制时，它将寄存器的当前内容存储在 R13 指向的地址。 未能为此目的保留适当的保存区域会产生不可预测的结果，并将可能在 CICS 中导致异常终止。

IBM i 采用 RPG 对 IBM MQ 程序进行编码（仅限 IBM i）

在 IBM MQ 文档中，调用的参数、数据类型的名称、结构的字段和常量的名称全都使用其长名称进行描述。 在 RPG 中，这些名称缩写为六个或更少的大写字符。

例如，字段 *MsgType* 在 RPG 中变为 *MDMT*。 有关更多信息，请参阅 IBM i 应用程序编程参考 (ILE/RPG)。

采用 PL/I 编码（仅限 z/OS）

采用 PL/I 针对 IBM MQ 进行编码时的有用信息

结构

结构使用 BASED 属性进行声明，因此除非程序声明结构的一个或多个实例，否则请勿占用任何存储器。

可以使用 `like` 属性来声明结构的实例，例如：

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

结构字段使用 INITIAL 属性进行声明；当 `like` 属性用于声明结构的实例时，该实例继承为该结构定义的初始值。 只需设置所需值与初始值不同的字段。

PL/I 不分区大小写，因此可以采用小写、大写或混合大小写的形式对调用、结构字段和常量的名称进行编码。

命名常量

命名常量声明为宏变量；因此，未被程序引用的命名常量在已编译过程中不占用任何存储器。

但是，在编译程序时，必须指定导致源由宏预处理器处理的编译器选项。

所有宏变量都是字符变量，即使表示数字值的宏变量也如此。 虽然这可能看似难以想象，但在宏处理器替换宏变量后，不会导致任何数据类型冲突，例如：

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

使用 IBM MQ 样本过程程序

这些样本程序采用过程语言编写，并演示了消息队列接口 (MQI) 的典型用法。 IBM MQ 在不同平台上编程。

关于此任务

有两组样本：

- 分布式系统和 IBM i 的样本程序。
- z/OS 的样本程序。

过程

- 使用以下链接了解有关样本程序的更多信息：
  - 第 964 页的『在 Multiplatforms 上使用样本程序』
  - z/OS 第 1054 页的『针对 z/OS 使用样本程序』

相关概念

第 7 页的『应用程序开发概念』
您可以选择使用过程化语言或面向对象语言来编写 IBM MQ 应用程序。 在开始设计和编写 IBM MQ 应用程序之前，请先熟悉 IBM MQ 基本概念。

第 5 页的『为 IBM MQ 开发应用程序』
您可以开发应用程序以发送和接收消息，以及管理队列管理器和相关资源。 IBM MQ 支持使用多种不同语言和框架编写的应用程序。

第 40 页的『IBM MQ 应用程序的设计注意事项』

当您已决定应用程序如何利用可供您使用的平台和环境时，您需要决定如何使用 IBM MQ 提供的功能。

第 672 页的『编写用于排队的过程应用程序』
使用此信息来了解有关编写排队应用程序、连接和断开队列管理器、发布/预订以及打开和关闭对象的信息。

第 834 页的『编写客户机过程应用程序』
使用过程语言在 IBM MQ 上编写客户机应用程序时需要知道的内容。

第 746 页的『编写发布/预订应用程序』
开始编写发布/预订 IBM MQ 应用程序。

第 908 页的『构建过程应用程序』
您可以使用若干过程语言之一编写 IBM MQ 应用程序，并在几个不同平台上运行该应用程序。

第 947 页的『处理过程程序错误』
本信息说明与应用程序 MQI 调用关联的错误，这些错误可能是应用程序进行调用时产生的，也可能是将其消息传递给最终目标时产生的。

Multi 在 Multiplatforms 上使用样本程序

这些样本过程程序随附于产品。 样本采用 C 和 COBOL 编写，并且演示消息队列接口 (MQI) 的典型用法。

关于此任务

样本并非旨在演示常规编程方法，因此会省略您可能想要在生产程序中包含的一些错误检查。

所有样本的源代码都随附于产品；此源代码包含用于说明程序中演示的消息排队方法的注释。

IBM i 对于 RPG 编程，请参阅 IBM i 应用程序编程参考 (ILE/RPG)。

样本的名称以前缀 amq 开头。 第四个字符指示编程语言以及必要情况下的编译器：

- s：C 语言
- 0：COBOL 语言（在 IBM 和 Micro Focus 编译器上）
- i：COBOL 语言（仅在 IBM 编译器上）
- m：COBOL 语言（仅在 Micro Focus 编译器上）

可执行程序的第八个字符指示样本是以本地绑定方式还是客户机方式运行。 如果没有第八个字符，那么样本以本地绑定方式运行。 如果第八个字符为“c”，那么样本以客户机方式运行。

要运行样本应用程序，必须首先创建并配置队列管理器。 要设置队列管理器以接受客户机连接，请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』。

过程

- 使用以下链接了解有关样本程序的更多信息：
  - 第 965 页的『Multiplatforms 上的样本程序中演示的功能』
  - 第 998 页的『“发布/预订”样本程序』
  - 第 1003 页的『“放置”样本程序』
  - 第 990 页的『“分发列表”样本程序』
  - 第 980 页的『浏览样本程序』
  - 第 981 页的『Browser 样本程序』
  - 第 992 页的『“获取”样本程序』
  - 第 1004 页的『“参考消息”样本程序』
  - 第 1012 页的『“请求”样本程序』
  - 第 997 页的『“查询”样本程序』
  - 第 998 页的『“查询消息句柄属性”样本程序』
  - 第 1017 页的『“设置”样本程序』

- 第 991 页的『“回传”样本程序』
- 第 983 页的『“数据转换”样本程序』
- 第 1020 页的『“触发”样本程序』
- 第 979 页的『Asynchronous Put 样本程序』
- 第 984 页的『“数据库协调”样本』
- 第 982 页的『CICS 样本事务』
- 第 1022 页的『在 UNIX 和 Windows 上使用 TUXEDO 样本』
- 第 990 页的『“死信队列处理程序”样本』
- 第 982 页的『“连接”样本程序』
- 第 977 页的『API 出口样本程序』
- 第 1033 页的『在 Windows 上使用 SSPI 安全出口』
- 第 1034 页的『使用远程队列运行样本』
- 第 1034 页的『集群队列监视样本程序 (AMQSCLM)』
- 第 1041 页的『连接端点查找 (CEPL) 的样本程序』

相关概念

第 440 页的『C++ 样本程序』
提供四个样本程序以演示获取和放置消息。

相关任务

第 1054 页的『针对 z/OS 使用样本程序』
随附于 IBM MQ for z/OS 的样本过程应用程序演示消息队列接口 (MQI) 的典型用法。

Multi *Multiplatforms* 上的样本程序中演示的功能

用于显示 IBM MQ 样本程序所演示的方法的表的集合。

所有样本程序都使用 MQOPEN 和 MQCLOSE 调用来打开和关闭队列，因此这些方法未在表中单独列出。 请参阅包含您感兴趣的平台的标题。

z/OS 对于 z/OS 平台，请参阅第 1054 页的『针对 z/OS 使用样本程序』。

Linux UNIX *UNIX and Linux* 系统样本

UNIX and Linux 系统上 IBM MQ 的样本程序所演示的方法。

请参阅 第 975 页的『在 UNIX and Linux 上准备和运行样本程序』 以了解 UNIX and Linux 系统上 IBM MQ 的样本程序的存储位置。

第 965 页的表 163 该表列出提供哪些 C 和 COBOL 源文件以及是否包含服务器或客户机可执行文件。

表 *163:* 用于演示在 *UNIX and Linux* 上如何使用 *MQI*（*C* 和 *COBOL*）的样本程序.

一个包含四列的表。 第一列列出样本所演示的方法。 第二列列出 C 样本，第三列列出用于演示第一列中所列的每种方法的 COBOL 样本。 第四列显示是否包含服务器 C 可执行文件，第五列显示是否包含客户机 C 可执行文件。

方法	C（源代码）(第 967 页的『1』)	COBOL（源代码）(第 967 页的『2』)	服务器（C 可执行文件）	客户机（C 可执行文件）
使用发布/预订接口	amqspuba amqssuba amqssbxa	无样本	amqspub amqssub amqssbx	无样本
使用 MQPUT 调用放置消息	amqsput0	amq0put0	amqsput	amqsputc

表 *163:* 用于演示在 *UNIX and Linux* 上如何使用 *MQI*（*C* 和 *COBOL*）的样本程序.

一个包含四列的表。 第一列列出样本所演示的方法。 第二列列出 C 样本，第三列列出用于演示第一列中所列的每种方法的 COBOL 样本。 第四列显示是否包含服务器 C 可执行文件，第五列显示是否包含客户机 C 可执行文件。

(继续)

方法	C（源代码）(第 967 页的『1』)	COBOL（源代码）(第 967 页的『2』)	服务器（C 可执行文件）	客户机（C 可执行文件）
使用 MQPUT1 调用放置单条消息	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
将消息放置到分发列表中 (第 967 页的『3』)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
回复请求消息	amqsinqa	amqminqx amqiinqx	amqsinq	无样本
使用浏览获取消息（无等待）	amqsgbr0	amq0gbr0	amqsgbr	无样本
获取消息（具有时间限制的等待）	amqsget0	amq0get0	amqsget	amqsgetc
获取消息（无限等待）	amqstrg0	无样本	amqstrg	amqstrgc
获取消息（具有数据转换）	amqsecha	无样本	amqsech	无样本
将参考消息放入队列 (第 967 页的『3』)	amqsprma	无样本	amqsprm	amqsprmc
从队列获取参考消息 (第 967 页的『3』)	amqsgrma	无样本	amqsgrm	amqsgrmc
参考消息通道出口 (第 967 页的『3』)	amqsqrma amqsxrma	无样本	amqsxrm	无样本
浏览消息的前 20 个字符	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
浏览完整消息	amqsbcg0	无样本	amqsbcg	amqsbcgc
使用共享输入队列	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
使用独占输入队列	amqstrg0	amq0req0	amqstrg	amqstrgc
使用 MQINQ 调用	amqsinqa	amqminqx amqiinqx	amqsinq	无样本
使用 MQSET 调用	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
使用应答队列	amqsreq0	amq0req0	amqsreq	amqsreqc
请求消息异常	amqsreq0	amq0req0	amqsreq	无样本
接受截断的消息	amqsgbr0	amq0gbr0	amqsgbr	无样本
使用已解析的队列名称	amqsgbr0	amq0gbr0	amqsgbr	无样本
触发进程	amqstrg0	无样本	amqstrg	amqstrgc
使用数据转换	(第 967 页的『4』)	无样本	无样本	无样本
IBM MQ（协调 XA 兼容数据库管理器）使用 SQL 访问单个数据库	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	无样本	无样本

表 163: 用于演示在 *UNIX and Linux* 上如何使用 *MQI*（*C* 和 *COBOL*）的样本程序.

一个包含四列的表。 第一列列出样本所演示的方法。 第二列列出 C 样本，第三列列出用于演示第一列中所列的每种方法的 COBOL 样本。 第四列显示是否包含服务器 C 可执行文件，第五列显示是否包含客户机 C 可执行文件。

(继续)

方法	C（源代码）(第 967 页的『1』)	COBOL（源代码）(第 967 页的『2』)	服务器（C 可执行文件）	客户机（C 可执行文件）
IBM MQ（协调 XA 兼容数据库管理器）使用 SQL 访问两个数据库	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	无样本	无样本
CICS 事务 (第 968 页的『5』)	amqscic0.ccs	无样本	amqscic0	无样本
Encina 事务 (第 967 页的『3』)	amqsxae0	无样本	amqsxae0	无样本
用于放置消息的 TUXEDO 事务 (第 968 页的『6』)	amqstxpx	无样本	无样本	无样本
用于获取消息的 TUXEDO 事务 (第 968 页的『6』)	amqstxgx	无样本	无样本	无样本
TUXEDO 的服务器 (第 968 页的『6』)	amqstxsx	无样本	无样本	无样本
死信队列处理程序	目录 `./tools/c/Samples/dlq` (第 968 页的『7』)	无样本	amqsdlq	无样本
从 MQI 客户机放置消息	无样本	无样本	无样本	amqsputc
从 MQI 客户机获取消息	无样本	无样本	无样本	amqsgetc
使用 MQCONNX 连接到队列管理器	amqscnxc	无样本	无样本	amqscnxc
使用 API 出口	amqsaxe0	无样本	amqsaxe	无样本
集群工作负载均衡出口	amqswlm0	无样本	amqswlm	无样本
使用 MQSTAT 调用异步放置消息并获取状态	amqsapt0	无样本	amqsapt	amqsaptc
可重新连接的客户机	amqsphac amqsghac amqsmhac	无样本	不适用	amqsphac amqsghac amqsmhac
使用消息使用者异步消耗来自多个队列的消息	amqscbf0	无样本	amqscbf	amqscbfc
指定有关 MQCONNX 的 TLS 连接信息	amqssslc	无样本	不适用	amqssslc

注意:

1. IBM MQ MQI client 样本的可执行文件版本与在服务器环境中运行的样本共享相同的源代码。
2. 使用 Micro Focus COBOL 编译器编译以“amqm”开头的程序，使用 IBM COBOL 编译器编译以“amqi”开头的程序，以及使用其中任一编译器编译以“amq0”开头的程序。
3. UNIX 仅在 IBM MQ for AIX 和 IBM MQ for Solaris 上受支持。
4. UNIX 在 IBM MQ for AIX 和 IBM MQ for Solaris 上，此程序名为 `amqsvfc0.c`

5. AIX CICS 仅受 IBM MQ for AIX 支持。
6. Linux IBM MQ for Linux on System p 不支持 TUXEDO。
7. 死信队列处理程序的源代码由若干文件组成并在单独的目录中提供。

有关 UNIX and Linux 系统支持的详细信息，请参阅 IBM MQ 的系统需求。

Windows *IBM MQ for Windows* 的样本

IBM MQ for Windows 的样本程序所演示的方法。

第 968 页的表 164 列出提供了哪些 C 和 COBOL 源文件以及是否包含服务器或客户机可执行文件。

表 *164: IBM MQ for Windows* 样本程序演示 *MQI*（*C* 和 *COBOL*）的用法

方法	C（源代码）	COBOL（源代码）	服务器（C 可执行文件）	客户机（C 可执行文件）
使用发布/预订接口	amqspuba amqssuba amqssbxa	无样本	amqspub amqssub amqssbx	无样本
使用 MQPUT 调用放置消息	amqsput0	amq0put0	amqsput	amqsputc
使用 MQPUT1 调用放置单条消息	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
将消息放到分发列表	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
回复请求消息	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
获取消息（无等待）	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
获取消息（具有时间限制的等待）	amqsget0	amq0get0	amqsget	amqsgetc
获取消息（无限等待）	amqstrg0	无样本	amqstrg	amqstrgc
获取消息（具有数据转换）	amqsecha	无样本	amqsech	amqsechc
将参考消息放入队列	amqsprma	无样本	amqsprm	amqsprmc
从队列获取参考消息	amqsgrma	无样本	amqsgrm	amqsgrmc
参考消息通道出口	amqsqrma amqsxrma	无样本	amqsxrm	无样本
浏览消息的前 20 个字符	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
浏览完整消息	amqsbcg0	无样本	amqsbcg	amqsbcgc
使用共享输入队列	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
使用独占输入队列	amqstrg0	amq0req0	amqstrg	amqstrgc
使用 MQINQ 调用	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
使用 MQSET 调用	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
使用 MQINQMP 调用	amqsiqma	无样本	无样本	无样本
使用应答队列	amqsreq0	amq0req0	amqsreq	amqsreqc

表 *164: IBM MQ for Windows* 样本程序演示 *MQI*（*C* 和 *COBOL*）的用法 *(*继续*)*				
方法	**C（源代码）**	**COBOL（源代码）**	**服务器（C 可执行文件）**	**客户机（C 可执行文件）**
请求消息异常	amqsreq0	amq0req0	amqsreq	amqsreqc
接受截断的消息	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
使用已解析的队列名称	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
触发进程	amqstrg0	无样本	amqstrg	amqstrgc
使用数据转换	amqsvfc0	无样本	无样本	无样本
IBM MQ（协调 XA 兼容数据库管理器）使用 SQL 访问单个数据库	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	无样本	无样本
IBM MQ（协调 XA 兼容数据库管理器）使用 SQL 访问两个数据库	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	无样本	无样本
用于放置消息的 TUXEDO 事务	amqstxpx	无样本	无样本	无样本
用于获取消息的 TUXEDO 事务	amqstxgx	无样本	无样本	无样本
TUXEDO 的服务器	amqstxsx	无样本	无样本	无样本
死信队列处理程序	目录 `./tools/c/Samples/dlq`（第 970 页的『1』）	无样本	amqsdlq	无样本
从 IBM MQ MQI client 放置消息	无样本	无样本	无样本	amqsputc
从 IBM MQ MQI client 获取消息	无样本	无样本	无样本	amqsgetc
使用 MQCONNX 连接到队列管理器	amqscnxc	无样本	无样本	amqscnxc
使用 API 出口	amqsaxe0	无样本	amqsaxe	无样本
集群工作负载均衡	amqswlm0	无样本	amqswlm	无样本
SSPI 安全例程	amqsspin	无样本	amqrspin.dll	amqrspin.dll
使用 MQSTAT 调用异步放置消息并获取状态	amqsapt0	无样本	amqsapt	amqsaptc
可重新连接的客户机	amqsphac amqsghac amqsmhac	无样本	不适用	amqsphac amqsghac amqsmhac
使用消息使用者异步消耗来自多个队列的消息	amqscbf0	无样本	amqscbf	amqscbfc
指定有关 MQCONNX 的 TLS 连接信息	amqssslc	无样本	不适用	amqssslc

注意:

1. 死信队列处理程序的源代码由若干文件组成并在单独的目录中提供。

Windows *IBM MQ for Windows* 的 *Visual Basic* 样本

Windows 系统上 IBM MQ 的样本程序所演示的方法。

第 970 页的表 165 显示 IBM MQ for Windows 样本程序所演示的方法。

项目可以包含若干文件。 在 Visual Basic 内打开项目时，将会自动装入其他文件。 未提供可执行程序。

所有样本项目（mqtrivc.vbp 除外）都设置为与 IBM MQ 服务器协作。 要了解如何将样本项目更改为与 IBM MQ 客户机协作，请参阅第 932 页的『在 Windows 中准备 Visual Basic 程序』。

表 *165:* 演示 *MQI (Visual Basic)* 用法的 *IBM MQ for Windows* 样本程序

方法	**项目文件名**
使用 MQPUT 调用放置消息	amqsputb.vbp
使用 MQGET 调用获取消息	amqsgetb.vbp
使用 MQGET 调用浏览队列	amqsbcgb.vbp
简单的 MQGET 和 MQPUT 样本（客户机）	mqtrivc.vbp
简单的 MQGET 和 MQPUT 样本（服务器）	mqtrivs.vbp
使用 MQPUT 和 MQGET 放置和获取字符串及用户定义的结构	strings.vbp
使用 PCF 结构启动和停止通道	pcfsamp.vbp
使用 MQAI 创建队列	amqsaicq.vbp
使用 MQAI 列出队列管理器的队列	amqsailq.vbp
使用 MQAI 监视事件	amqsaiem.vbp

IBM i *IBM i* 的样本

IBM i 系统上 IBM MQ 的样本程序所演示的方法。

第 970 页的表 166 显示 IBM MQ for IBM i 样本程序所演示的方法。 某些方法出现在多个样本程序中，但是表中仅列出了一个程序。

表 *166: IBM i* 上演示 *MQI*（*C* 和 *COBOL*）用法的样本程序

方法	**C（源代码）(第 971 页的『1』)**	**COBOL（源代码）(第 971 页的『2』)**	**RPG（源代码）(第 972 页的『3』)**	**客户机（C 可执行文件）(4)**
使用 MQPUT 调用放置消息	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
使用 MQPUT 调用放置来自数据文件的消息	AMQSPUT4	无样本	无样本	无样本
使用 MQPUT1 调用放置单条消息	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC、AMQSECHC
将消息放到分发列表	AMQSPTL4	无样本	无样本	AMQSPTLC
回复请求消息	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
获取消息（无等待）	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
获取消息（具有时间限制的等待）	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
获取消息（无限等待）	AMQSTRG4	无样本	AMQ3TRG4	AMQSTRGC
获取消息（具有数据转换）	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
将参考消息放入队列	AMQSPRM4	无样本	无样本	AMQSPRMC

表 166: *IBM i* 上演示 *MQI*（*C* 和 *COBOL*）用法的样本程序 *(继续)*

方法	C（源代码）(第 971 页的『1』)	COBOL（源代码）(第 971 页的『2』)	RPG（源代码）(第 972 页的『3』)	客户机（C 可执行文件）(4)
从队列获取参考消息	AMQSGRM4	无样本	无样本	AMQSGRMC
参考消息通道出口	AMQSQRM4, AMQSXRM4	无样本	无样本	无样本
消息出口	AMQSCMX4	无样本	无样本	无样本
浏览消息的前 49 个字符	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
浏览完整消息	AMQSBCG4	无样本	无样本	AMQSBCGC
使用共享输入队列	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
使用独占输入队列	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
使用 MQINQ 调用	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
使用 MQSET 调用	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
使用应答队列	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
请求消息异常	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
接受截断的消息	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
使用已解析的队列名称	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
触发进程	AMQSTRG4	无样本	AMQ3TRG4	AMQSTRGC
触发器服务器	AMQSERV4	无样本	AMQ3SRV4	无样本
使用触发器服务器（包括 CICS 事务）	AMQSERV4	无样本	AMQ3SRV4	无样本
使用数据转换	AMQSVFC4	无样本	无样本	无样本
使用 API 出口	AMQSAXE0	无样本	无样本	无样本
集群工作负载均衡	AMQSWLM0	无样本	无样本	无样本
使用 MQSTAT 调用异步放置消息并获取状态	AMQSAPT0	无样本	无样本	AMQSAPTC
使用发布/预订接口	AMQSPUBA、AMQSSUBA、AMQSSBXA	无样本	无样本	AMQSPUBC、AMQSSUBC、AMQSSBXC
可重新连接的客户机 (5)	AMQSPHAC、AMQSGHAC、AMQSMHAC	无样本	无样本	无样本
使用消息使用者异步消耗来自多个队列的消息 (5)	AMQSCBFO	无样本	无样本	无样本
指定有关 MQCONNX 的 TLS 连接信息	AMQSSSLC	无样本	无样本	AMQSSSLC
使用 MQCONNX 连接到队列管理器	AMQSCNXC	无样本	无样本	AMQSCNXC

注意:

1. C 样本的源代码位于文件 QMQMSAMP/QCSRC 中。 包含文件作为文件 QMQM/H 中的成员存在。
2. COBOL 样本的源代码位于文件 QMQMSAMP/QCBLLESRC 中。 成员命名为 AMQ0 xxx 4，其中 xxx 指示样本函数。

3. RPG 样本的源代码位于 QMQMSAMP/QRPGLESRC 中。 成员命名为 AMQ3 xxx 4，其中 xxx 指示样本函数。 副本成员存在于 QMQM/QRPGLESRC 中。 每个成员名具有后缀 G。
4. IBM MQ MQI client 样本的可执行文件版本与在服务器环境中运行的样本共享相同的源代码。 客户机环境中的样本的源代码与服务器相同。 IBM MQ MQI client 样本与客户机库 LIBMQIC 进行链接，IBM MQ 服务器样本与服务器库 LIBMQM 进行链接。
5. 如果必须运行可重新连接的客户机和异步使用者的样本应用程序的客户机可执行文件，那么必须对其进行编译并将其与线程库 LIBMQIC_R 进行链接。 因此，它必须在线程环境中运行。 将环境变量 QIBM_MULTI_THREADED 设置为“Y”并从 qsh 运行应用程序。

   有关更多信息，请参阅使用 Java 和 JMS 设置 IBM MQ。

除这些以外，IBM MQ for IBM i 样本选项还包含样本数据文件，该文件用作用于演示管理任务的样本程序、AMQSDATA 和样本 CL 程序的输入。 CL 样本在管理 IBM i 中进行了描述。 您可以使用样本 CL 程序 amqsamp4 来创建队列以与本主题中描述的样本程序结合使用。

Windows IBM i UNIX 准备并运行样本程序

完成一些初始准备后，您可以运行样本程序。

关于此任务

在运行样本程序之前，必须首先创建队列管理器，同时创建所需的队列。 如果要运行 COBOL 样本（举例来说），您还可能需要进行其他一些准备。 完成必要的准备后，您可以运行样本程序。

过程

有关如何准备和运行样本程序的信息，请参阅以下主题：

- 第 974 页的『在 IBM i 上准备和运行样本程序』
- 第 975 页的『在 UNIX and Linux 上准备和运行样本程序』
- 第 976 页的『在 Windows 上准备和运行样本程序』

Windows IBM i UNIX 配置队列管理器以接受多平台上的客户机连接

要运行样本应用程序，必须首先创建队列管理器。 然后，可将队列管理器配置为以安全方式接受来自以客户机方式运行的应用程序的入局连接请求。

开始之前

确保队列管理器已经存在并已启动。 通过发出 MQSC 命令确定是否已经启用通道认证记录：

```
DISPLAY QMGR CHLAUTH
```

要点： 此任务期望启用通道认证记录。 如果这是由其他用户和应用程序使用的队列管理器，那么更改此设置将影响所有其他用户和应用程序。 如果队列管理器不利用通道认证记录，那么步骤 4 可以替换为备用认证方法（例如，安全出口），此方法将 MCAUSER 设置为您将在步骤 第 972 页的『1』 中获取的 *non-privileged-user-id*。

您必须知道应用程序期望使用的通道名称，以便可以允许应用程序使用该通道。 您还必须知道应用程序期望使用哪些对象（例如队列或主题），以便可以允许应用程序使用这些对象。

关于此任务

此任务创建要用于连接到队列管理器的客户机应用程序的非特权用户标识。 将会授予使客户机应用程序只能通过使用此用户标识来使用其所需的通道和队列的访问权。

过程

1. 在队列管理器运行所在的系统上获取用户标识。 对于此任务，此用户标识不得是特权管理用户。 此用户标识将是客户机连接将在队列管理器上运行所使用的权限。

2. 使用以下命令启动侦听器程序，其中：

   qmgr-name 是队列管理器的名称
   nnnn 是所选端口号

   a) Windows UNIX
   对于 UNIX 和 Windows 系统：

   ```
   runmqlsr -t tcp -m qmgr-name -p nnnn
   ```

   b) IBM i
   对于 IBM i：

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. 如果应用程序使用 SYSTEM.DEF.SVRCONN，那么表明已经定义此通道。 如果应用程序使用其他通道，请通过发出 MQSC 命令创建该通道：

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name 是通道的名称。

4. 通过发出 MQSC 命令创建通道认证规则，从而仅允许客户机系统的 IP 地址使用通道：

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   channel-name 是通道的名称。
   client-machine-IP-address 是客户机系统的 IP 地址。
   如果样本客户机应用程序与队列管理器在同一机器上运行，那么在应用程序即将使用“localhost”进行连接时，使用 IP 地址“127.0.0.1”。 如果即将接入若干不同客户端机器，那么可以使用模式或范围而不是单个 IP 地址。 请参阅类属 IP 地址以获取详细信息。
   non-privileged-user-id 是在步骤 第 972 页的『1』 中获取的用户标识

5. 如果应用程序使用 SYSTEM.DEFAULT.LOCAL.QUEUE，那么表明已经定义此队列。 如果应用程序使用其他队列，请通过发出 MQSC 命令创建该队列：

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   queue-name 是队列的名称。

6. 授予用于连接并查询队列管理器的访问权：

   a) Windows IBM i UNIX
   对于 IBM i IBM i， UNIX 和 Windows 系统发出 MQSC 命令：

   ```
   SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
   AUTHADD(CONNECT, INQ)
   ```

   non-privileged-user-id 是在步骤 第 972 页的『1』 中获取的用户标识

7. 如果应用程序是点到点应用程序，那么表明它利用队列，通过发出 MQSC 命令来授予访问权，以允许按用户标识使用队列查询以及放置和获取消息：

   a) Windows IBM i UNIX
   对于 IBM i IBM i， UNIX 和 Windows 系统发出 MQSC 命令：

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

queue-name 是队列的名称。
non-privileged-user-id 是在步骤 第 972 页的『1』 中获取的用户标识

8. 如果应用程序是发布/预订应用程序，那么表明它利用主题，通过发出 MQSC 命令来授予访问权，以允许按用户标识使用主题进行发布和预订：

a) Windows IBM i UNIX

对于 IBM i IBM i， UNIX 和 Windows 系统发出 MQSC 命令：

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

non-privileged-user-id 是在步骤 第 972 页的『1』 中获取的用户标识
这将给予对主题树中任何主题的 *non-privileged-user-id* 访问权，或者，可以使用 **DEFINE TOPIC** 来定义主题对象并仅授予对该主题对象引用的主题树部分的访问权。 请参阅控制对主题的用户访问权以获取详细信息。

下一步做什么

客户机应用程序现在可以连接到队列管理器并使用队列放置或获取消息。

相关概念

IBM i IBM i 上的 IBM MQ 权限

相关任务

授予对 UNIX 或 Linux 系统和 Windows 上的 IBM MQ 对象的访问权

相关参考

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC

IBM i 在 *IBM i* 上准备和运行样本程序

在 IBM i 上运行样本程序之前，必须首先创建队列管理器，同时创建所需的队列。 如果要运行 COBOL 样本，您可能需要进行其他一些准备。

关于此任务

IBM MQ for IBM i 样本程序的源代码作为 QCSRC、QCLSRC、QCBLLESRC 和 QRPGLESRC 的成员在库 QMQMSAMP 中提供。

您可以在运行样本时使用自己的队列，也可以运行样本程序 AMQSAMP4 来创建某些样本队列。 AMQSAMP4 程序的源代码包含在库 QMQMSAMP 中的文件 QCLSRC 中。 您可以使用 CRTCLPGM 命令进行编译。

要运行样本，请使用库 QMQM 中提供的 C 可执行文件版本，或者以类似于任何其他 IBM MQ 应用程序的方式对其进行编译。

过程

1. 创建队列管理器并设置缺省定义。

必须完成此操作，然后才能运行任何样本程序。 有关创建队列管理器的更多信息，请参阅管理 IBM MQ。 有关将队列管理器配置为以安全方式接受来自以客户机方式运行的应用程序的入局连接请求的信息，请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』。

2. 要使用来自库 QMQMSAMP 的文件 AMQSDATA 中成员 PUT 的数据调用其中一个样本程序，请使用如下命令：

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

注： 为使已编译的模块使用 IFS 文件系统，请在 CRTCMOD 上指定选项 SYSIFCOPT(*IFSIO)，然后必须按以下格式指定作为参数传递的文件名：

```
home/me/myfile
```

3. 如果想要使用问询、设置和回传示例的 COBOL 版本，请在运行这些样本之前更改进程定义。

   对于问询、设置和回传示例，样本定义触发这些样本的 C 版本。 如果需要 COBOL 版本，那么必须更改进程定义：

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   在 IBM i 上，可以使用 **CHGMQMPRC** 命令（有关详细信息，请参阅更改 MQ 进程 (CHGMQMPRC)），或者使用备用定义编辑和运行 **AMQSAMP4** 命令。
4. 运行样本程序。

   有关每个样本期望的参数的更多信息，请参阅各样本的描述。

   注： 对于 COBOL 样本程序，在将队列名称作为参数传递时，必须提供 48 个字符，如有必要使用空白字符进行填充。 除 48 个字符以外的任何其他内容都会导致程序失败并显示原因码 2085。

相关参考

第 970 页的『IBM i 的样本』
IBM i 系统上 IBM MQ 的样本程序所演示的方法。

UNIX 在 *UNIX and Linux* 上准备和运行样本程序

在 UNIX 上运行样本程序之前，必须首先创建队列管理器，同时创建所需的队列。 如果要运行 COBOL 样本，您可能需要进行其他一些准备。

关于此任务

如果在安装时使用了缺省值，那么 IBM MQ on UNIX and Linux 系统样本文件位于 第 975 页的表 167 中列出的目录中。

表 *167:* 在 *UNIX and Linux* 系统上查找 *IBM MQ* 的样本的位置

内容	**目录**
源文件	*MQ_INSTALLATION_PATH*/samp
死信队列处理程序源文件	*MQ_INSTALLATION_PATH*/samp/dlq
可执行文件	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

样本需要与队列集协作。 您可以使用自己的队列或者运行样本 MQSC 文件 amqscos0.tst 来创建一个集合。要运行样本，请使用所提供的可执行程序版本，或者使用 ANSI 编译器按照编译任何其他应用程序的方式编译源版本。

过程

1. 创建队列管理器并设置缺省定义。

必须完成此操作，然后才能运行任何样本程序。 有关创建队列管理器的更多信息，请参阅管理 IBM MQ。 有关将队列管理器配置为以安全方式接受来自以客户机方式运行的应用程序的入局连接请求的信息，请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』。

2. 如果您未使用自己的队列，请运行样本 MQSC 文件 amqscos0.tst 来创建队列集。

   要在 UNIX and Linux 系统上执行此操作，请输入：

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   检查 `sampobj.out` 文件以确保没有错误。

3. 如果想要使用问询、设置和回传示例的 COBOL 版本，请在运行这些样本之前更改进程定义。

   对于问询、设置和回传示例，样本定义触发这些样本的 C 版本。 如果需要 COBOL 版本，那么必须更改进程定义：

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   在 Windows 上，通过编辑 `amqscos0.tst` 文件并将 C 可执行文件名更改为 COBOL 可执行文件名，然后使用 **runmqsc** 命令运行这些样本来执行此操作。

4. 运行样本程序。

   要运行样本，请输入样本名称，后跟任何参数，例如：

   ```
   amqsput myqueue qmanagername
   ```

   其中 *myqueue* 是消息即将放到的队列的名称，*qmanagername* 是拥有 *myqueue* 的队列管理器。

   有关每个样本期望的参数的更多信息，请参阅各样本的描述。

   注: 对于 COBOL 样本程序，在将队列名称作为参数传递时，必须提供 48 个字符，如有必要使用空白字符进行填充。 除 48 个字符以外的任何其他内容都会导致程序失败并显示原因码 2085。

相关参考

第 965 页的『UNIX and Linux 系统样本』
UNIX and Linux 系统上 IBM MQ 的样本程序所演示的方法。

Windows 在 *Windows* 上准备和运行样本程序

在 Windows 上运行样本程序之前，必须首先创建队列管理器，同时创建所需的队列。 如果要运行 COBOL 样本，您可能需要进行其他一些准备。

关于此任务

IBM MQ for Windows 样本文件位于第 976 页的表 168 中列出的目录内（如果在安装时使用了缺省值）。 缺省情况下，安装驱动器为 <c:>。

表 *168:* 在何处查找 *IBM MQ for Windows* 样本

内容	目录
C 源代码	*MQ_INSTALLATION_PATH*\Tools\C\Samples
死信处理程序样本的源代码	*MQ_INSTALLATION_PATH*\Tools\C\Samples\DLQ
COBOL 源代码	*MQ_INSTALLATION_PATH*\Tools\Cobol \ 样本
C 可执行文件 [1]	*MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin（32 位版本） *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64（64 位版本）
样本 MQSC 文件	*MQ_INSTALLATION_PATH*\Tools\MQSC\Samples
Visual Basic 源代码	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6

表 168: 在何处查找 IBM MQ for Windows 样本 (继续)	
内容	**目录**
.NET 样本	*MQ_INSTALLATION_PATH*\Tools\dotnet \ 样本

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

注: 某些 C 可执行文件样本的 64 位版本可用。

样本需要与队列集协作。 您可以使用自己的队列或者运行样本 MQSC 文件 amqscos0.tst 来创建队列集。 要运行样本，请使用所提供的可执行文件版本，或者按照编译任何其他 IBM MQ for Windows 应用程序的方式编译源版本。

过程

1. 创建队列管理器并设置缺省定义。

   必须完成此操作，然后才能运行任何样本程序。 有关创建队列管理器的更多信息，请参阅管理 IBM MQ。 有关将队列管理器配置为以安全方式接受来自以客户机方式运行的应用程序的入局连接请求的信息，请参阅第 972 页的『配置队列管理器以接受多平台上的客户机连接』。
2. 如果您未使用自己的队列，请运行样本 MQSC 文件 amqscos0.tst 来创建队列集。

   要在 Windows 系统上执行此操作，请输入：

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   检查 `sampobj.out` 文件以确保没有错误。 此文件位于当前目录中。

   注:
3. 如果想要使用问询、设置和回传示例的 COBOL 版本，请在运行这些样本之前更改进程定义。

   对于问询、设置和回传示例，样本定义触发这些样本的 C 版本。 如果需要 COBOL 版本，那么必须更改进程定义：

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   在 Windows 上，通过编辑 `amqscos0.tst` 文件并将 C 可执行文件名更改为 COBOL 可执行文件名，然后使用 **runmqsc** 命令运行这些样本来执行此操作。
4. 运行样本程序。

   要运行样本，请输入样本名称，后跟任何参数，例如：

   ```
   amqsput myqueue qmanagername
   ```

   其中 *myqueue* 是消息即将放到的队列的名称，*qmanagername* 是拥有 *myqueue* 的队列管理器。

   有关每个样本期望的参数的更多信息，请参阅各样本的描述。

   注: 对于 COBOL 样本程序，在将队列名称作为参数传递时，必须提供 48 个字符，如有必要使用空白字符进行填充。 除 48 个字符以外的任何其他内容都会导致程序失败并显示原因码 2085。

相关参考

第 968 页的『IBM MQ for Windows 的样本』
IBM MQ for Windows 的样本程序所演示的方法。

第 970 页的『IBM MQ for Windows 的 Visual Basic 样本』
Windows 系统上 IBM MQ 的样本程序所演示的方法。

API 出口样本程序

样本 API 出口生成对具有 MQAPI_TRACE_LOGFILE 环境变量中所定义前缀的用户指定的文件的 MQI 跟踪。

有关 API 出口的更多信息，请参阅第 867 页的『在多平台上编写和编译 API 出口』。

源
amqsaxe0.c

Binary
amqsaxe

为样本出口进行配置

1. 向 qm.ini 文件中添加以下内容。

   除 Windows 以外的平台

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   其中，*MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的目录。

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   其中，*MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的目录。

2. 设置环境变量

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. 运行应用程序。

   在 /tmp 目录中创建如下名称的输出文件：MqiTrace.*pid*.*tid*.log

异步消耗样本程序

amqscbf 样本程序演示使用 MQCB 和 MQCTL 异步消耗来自多个队列的消息。

amqscbf 在 Windows、UNIX and Linux 平台上提供作为 C 源代码以及二进制客户机和服务器可执行程序。

程序从命令行启动并采用以下可选参数：

```
Usage: [Options] Queue Name {queue_name}
  where Options are:
    -m Queue Manager Name
    -o Open options
    -r Reconnect Type
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

提供多个队列名称以读取来自多个队列的消息（样本最多支持 10 个队列。）

注: Reconnect type 仅对客户机程序有效。

示例

示例显示 amqscbf 运行为服务器程序，从 QL1 读取一条消息，然后停止。

使用 IBM MQ 资源管理器将测试消息放在 QL1 上。 通过按 Enter 键停止程序。

```
C:\>amqscbf QL1
Sample AMQSCBF0 start
```

```
Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

amqscbf 的演示内容

样本显示如何按照消息的到达顺序读取来自多个队列的消息。 这将要求更多的代码使用同步 MQGET。 在异步消耗情况下，无需轮询，并且线程和存储管理由 IBM MQ 执行。 “现实世界”示例将需要处理错误；在样本中，错误写出到控制台。

样本代码具有以下步骤，

1. 定义单消息消耗回调函数，

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
                     MQCBC   * pContext)
{ ... }
```

2. 连接到队列管理器，

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. 打开输入队列，并将各个队列与 `MessageConsumer` 回调函数相关联，

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   无需为各队列设置 `cbd.CallbackFunction`；它是只输入字段。 但是，可以将其他回调函数与各队列相关联。

4. 启动消息的消耗，

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. 等待直至用户按 Enter 键，然后停止消息的消耗，

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 最后，与队列管理器断开连接，

```
MQDISC(&Hcon,&CompCode,&Reason);
```

Asynchronous Put 样本程序

了解有关运行 amqsapt 样本以及 Asynchronous Put 样本程序的设计的信息。

Asynchronous Put 样本程序使用异步 MQPUT 调用将消息放在队列上，然后使用 MQSTAT 调用检索状态信息。 请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』以获取此程序在不同平台上的名称。

运行 amqsapt 样本

此程序最多采用 6 个参数：

1. 目标队列的名称（必需）
2. 队列管理器名称（可选）
3. 打开选项（可选）

4. 关闭选项（可选）
5. 目标队列管理器的名称（可选）
6. 动态队列名称（可选）

如果未指定队列管理器，那么 amqsapt 连接到缺省队列管理器。

Asynchronous Put 样本程序的设计

程序将 MQOPEN 调用与所提供的输出选项或者与 MQOO_OUTPUT 和 MQOO_FAIL_IF_QUIESCING 选项结合使用以打开用于放置消息的目标队列。

如果无法打开队列，那么该程序将输出错误消息，其中包含 MQOPEN 调用返回的原因码。 为保持程序简单，程序将在此和后续的 MQI 调用中使用许多选项的缺省值。

对于各行输入，程序将文本读取到缓冲区中，并且使用带有 MQPMO_ASYNC_RESPONSE 的 MQPUT 调用以创建包含该行的文本的数据报消息，然后异步将其放到目标队列。 程序继续运行，直至到达输入的结尾或者 MQPUT 调用失败。 如果程序到达输入的结尾，那么它使用 MQCLOSE 调用来关闭队列。

然后，程序发出 MQSTAT 调用，返回 MQSTS 结构，并且显示包含成功放置的消息数、放置的带有警告的消息数以及失败数的消息。

浏览样本程序

“浏览”样本程序使用 MQGET 调用浏览队列上的消息。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

浏览样本程序的设计

程序使用带有 MQOO_BROWSE 选项的 MQOPEN 调用来打开目标队列。 如果无法打开队列，那么该程序将输出错误消息，其中包含 MQOPEN 调用返回的原因码。

对于队列上的各消息，程序使用 MQGET 调用从队列复制消息，然后显示消息中包含的数据。 MQGET 调用使用以下选项：

MQGMO_BROWSE_NEXT
在 MQOPEN 调用之后，浏览光标在逻辑上位于队列中的第一条消息之前，因此该选项导致在首次进行调用时返回**第一条**消息。

MQGMO_NO_WAIT
如果队列上没有消息，那么程序不等待。

MQGMO_ACCEPT_TRUNCATED_MSG
MQGET 调用指定固定大小的缓冲区。 如果消息长度超过此缓冲区，那么程序显示已截断的消息，连同表明消息已截断的警告。

程序演示在各 MQGET 调用之后必须如何清除 MQMD 结构的 *MsgId* 和 *CorrelId* 字段，因为调用将这些字段设置为其检索的消息中包含的值。 清除这些字段意味着连续 MQGET 调用按照消息在队列中的存放顺序检索消息。

程序继续运行至队列的结尾；MQGET 调用返回 MQRC_NO_MSG_AVAILABLE 原因码，并且程序显示警告消息。 如果 MQGET 调用失败，那么程序将显示包含原因码的错误消息。

然后，程序使用 MQCLOSE 调用来关闭队列。

针对 *UNIX, Linux, and Windows* 的 *Browse* 样本程序

在了解有关 UNIX, Linux, and Windows 上 Browse 样本程序的信息时，请考虑使用本主题。

程序的 C 版本采用两个参数

1. 源队列的名称（必需）
2. 队列管理器名称（可选）

如果未指定队列管理器，那么它连接到缺省队列管理器。 例如，输入以下内容之一：

- `amqsgbr myqueue qmanagername`

- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

其中，`myqueue` 是将从中查看消息的队列的名称，`qmanagername` 是拥有 `myqueue` 的队列管理器。

如果省略 `qmanagername`，那么在运行 C 样本时，它假设缺省队列管理器拥有队列。

COBOL 版本没有任何参数。 它连接到缺省队列管理器，并且在运行它时会出现以下提示：

```
Please enter the name of the target queue
```

仅会显示各消息的前 50 个字符，后跟 `- - - truncated`（当情况如此时）。

IBM i 上的 *Browse* 样本程序
各程序检索在调用程序时指定的队列上的所有消息的副本；消息保留在队列上。

您可以使用提供的队列 SYSTEM.SAMPLE.LOCAL；首先运行 Put 样本程序以将某些消息放在队列上。 可以使用队列 SYSTEM.SAMPLE.ALIAS，它是同一本地队列的别名。 程序继续运行，直至到达队列的结尾或者 MQI 调用失败。

通过 C 样本，可以通过与 Windows 系统样本类似的方式，指定队列管理器名称（通常作为第二个参数）。 例如：

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

如果未指定队列管理器，那么它连接到缺省队列管理器。 这也与 RPG 样本相关。 但是，对于 RPG 样本，必须提供队列管理器名称，而不是允许其使用缺省值。

ULW *Browser* 样本程序

Browser 样本程序对于队列上所有消息的消息描述符和消息内容字段均进行读取和写入。

样本程序编写为实用程序，而不只是演示方法。 请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

此程序采用以下位置参数：

1. 源队列的名称（必需）
2. 队列管理器的名称（必需）
3. 属性的可选参数（可选）

这些程序还使用名为 **MQSAMP_USER_ID** 的环境变量，该变量应设置为要用于连接认证的用户标识。 如果设置了此环境变量，那么该程序会提示输入该用户标识及相应密码。

要运行这些程序，请输入以下命令之一：

- `amqsbcg` *`myqueue qmanagername`*
- `amqsbcgc` *`myqueue qmanagername`*

其中，*`myqueue`* 是即将浏览其中消息的队列的名称，*`qmanagername`* 是拥有 *`myqueue`* 的队列管理器。

它从队列读取各消息并将以下内容写入到 stdout：

- 格式化消息描述符字段
- 消息数据（以十六进制转储，如有可能，采用字符格式）

表 *169:* 属性参数允许的值

值	**行为**
0	缺省行为，如同针对 IBM WebSphere MQ 6 的行为。 传送到应用程序的属性取决于从中检索消息的 **PropertyControl** 队列属性。

表 169: 属性参数允许的值 (继续)	
值	**行为**

1	创建消息句柄并将其用于 MQGET。 通过与消息描述符类似的方式显示消息的属性，消息描述符（或扩展名）中包含的属性除外。 例如：
	`****Message properties****`
	`  property name: property value`
	或者，如果没有属性可用：
	`****Message properties****`
	`  None`
	数字值使用 `printf` 进行显示，字符串值用单引号引起来，字节字符串使用 X 和单引号引起来，如同针对消息描述符一样。

2	指定 MQGMO_NO_PROPERTIES，以便将仅返回消息描述符属性。
3	指定 MQGMO_PROPERTIES_FORCE_MQRFH2，以便在消息数据中返回所有属性。
4	指定 MQGMO_PROPERTIES_COMPATIBILITY，以便可以返回所有属性（具体取决于是否包含 IBM WebSphere MQ 6 属性），否则废弃属性。

该程序被限制为打印消息的前 65535 个字符，如果读取了较长的消息，那么该程序将失败，原因为 `truncated msg`。

有关来自此实用程序的输出的示例，请参阅浏览队列。

CICS 样本事务

针对源代码，提供了一个名为 amqscic0.ccs 的样本 CICS 事务程序；针对可执行版本，提供了 amqscic0。您可以使用标准 CICS 设施来构建事务。

请参阅第 908 页的『构建过程应用程序』，以获取有关平台所需命令的详细信息。

事务从传输队列 SYSTEM.SAMPLE.CICS 中读取消息。缺省队列管理器上的 WORKQUEUE ，并将它们放在本地队列上，本地队列的名称包含在消息的传输头中。 任何故障都将发送到队列 SYSTEM.SAMPLE。CICS。DLQ。

注: 您可以使用样本 MQSC 脚本 AMQSCIC0.TST 创建这些队列和样本输入队列。

“连接”样本程序

通过“连接”样本程序，您可以从客户机浏览 MQCONNX 调用及其选项。 此样本使用 MQCONNX 调用来连接到队列管理器，使用 MQINQ 调用来查询队列管理器名称并显示该名称。 此外，还介绍了如何运行 amqscnxc 样本。

注: “连接”样本程序是一个客户机样本。 虽然可以在服务器上编译和运行此样本程序，但它仅在客户机上才有意义，而且只提供了客户机可执行文件。

运行 amqscnxc 样本

“连接”样本程序的命令行语法如下：

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

这些参数都是可选参数，其顺序并不重要，但 QMgrName 除外，如果指定了 QMgrName，那么它必须是最后一个参数。 参数如下：

ConnName

服务器队列管理器的 TCP/IP 连接名称

如果未指定 TCP/IP 连接名称，那么将在 *ClientConnPtr* 设置为 NULL 的情况下发出 MQCONNX。

SvrconnChannelName
服务器连接通道的名称

如果指定了 TCP/IP 连接名称但未指定服务器连接通道（相反则不允许），那么该样本使用名称 SYSTEM.DEF.SVRCONN。

用户
要用于连接认证的用户名

如果指定了此选项，那么该程序会提示输入该用户标识及相应密码。

QMgrName
目标队列管理器名称

如果未指定目标队列管理器，那么该样本将连接到正在侦听指定 TCP/IP 连接名称的队列管理器。

注: 如果输入问号作为唯一参数，或者输入了不正确的参数，那么您会收到一条说明如何使用该程序的消息。

如果运行不带任何命令行选项的该样本，那么将使用 MQSERVER 环境变量的内容来确定连接信息。（在此示例中，MQSERVER 设置为 SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com。） 您将看到类似于以下内容的输出：

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

如果运行该样本，并提供 TCP/IP 连接名称和服务器连接通道名称但未提供目标队列管理器名称，那么类似于：

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

将使用缺省队列管理器名称，并且您将看到类似于以下内容的输出：

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

如果运行该样本并提供 TCP/IP 连接名称和目标队列管理器名称，那么类似于：

```
amqscnxc -x machine.site.company.com MACHINE
```

您将看到类似于以下内容的输出：

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

“数据转换”样本程序

“数据转换”样本程序是数据转换出口例程的框架。 了解数据转换样本的设计。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

设计“数据转换”样本

每个数据转换出口例程都可转换一种指定的消息格式。 此框架将用作数据转换出口生成实用程序所生成的代码片段的包装程序。

该实用程序针对每个数据结构生成一个代码片段；一种格式由多个这样的结构组成，因此会向此框架添加多个代码片段来生成例程，以用于执行整个格式的数据转换。

然后，该程序将检查转换是成功还是失败，并将所需值返回给调用者。

“数据库协调”样本

系统提供了两个样本，用于说明 IBM MQ 如何在同一工作单元中协调 IBM MQ 更新和数据库更新。

下面提供了这些样本：

1. AMQSXAS0（在 C 中）或 AMQ0XAS0（在 COBOL 中），用于在 IBM MQ 工作单元中更新单个数据库。
2. AMQSXAG0（在 C 中）或 AMQ0XAG0（在 COBOL 中）、AMQSXAB0（在 C 中）或 AMQ0XAB0（在 COBOL 中），以及 AMQSXAF0（在 C 中）或 AMQ0XAF0（在 COBOL 中），它们一起用于在 IBM MQ 工作单元中更新两个数据库并说明如何访问多个数据库。 提供的这些样本用于说明如何使用 MQBEGIN 调用、如何使用混合的 SQL 和 IBM MQ 调用，以及从何处和在何时连接到数据库。

第 984 页的图 133 说明了如何使用所提供的样本来更新数据库：

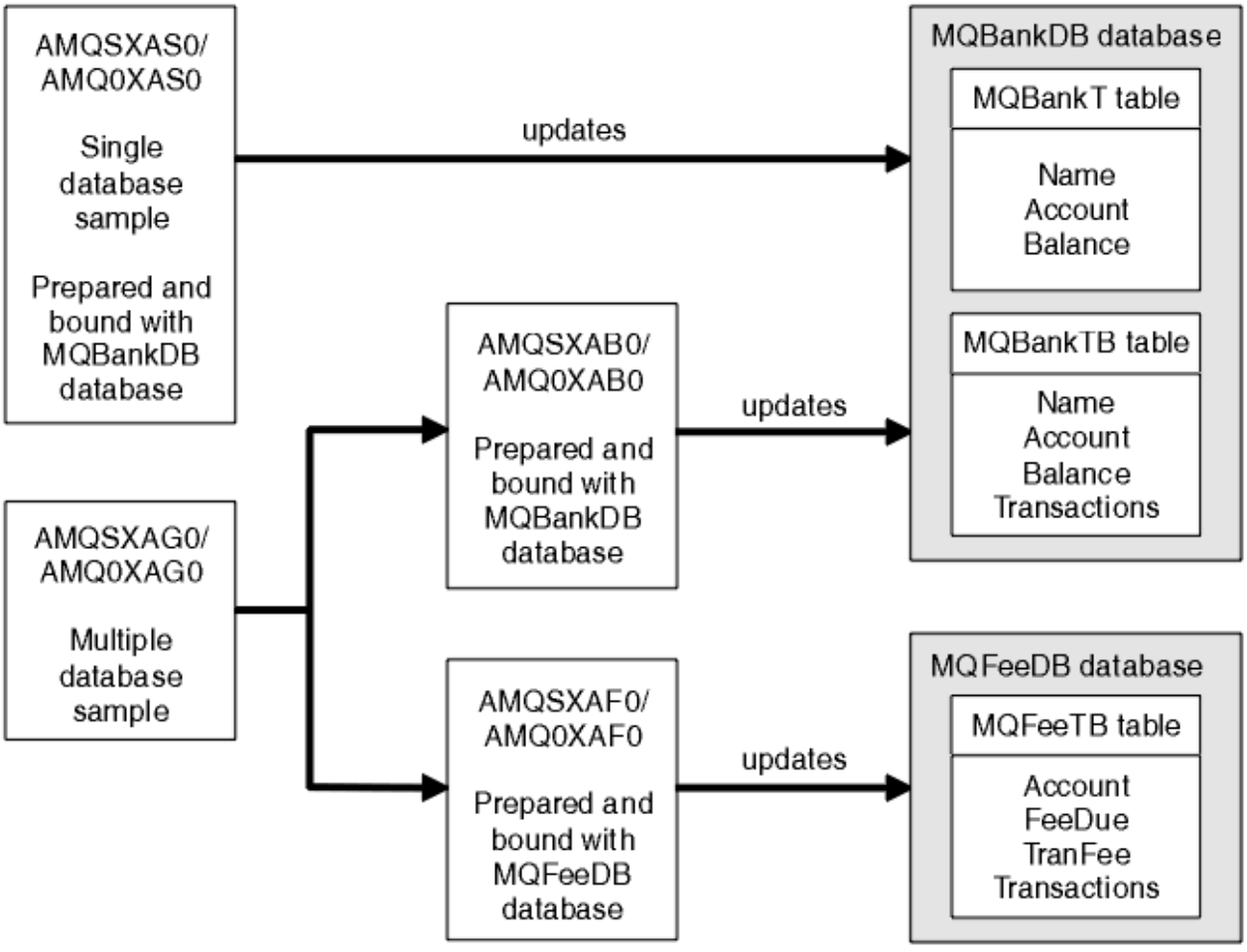


图 *133: “*数据库协调*”*样本

这些程序从队列中读取消息（在同步点下），然后使用该消息中的信息从数据库获取相关信息并进行更新。随后打印数据库的新状态。

程序逻辑如下：

1. 使用程序自变量中的输入队列名称

2. 使用 MQCONN 连接到缺省队列管理器（在 C 中，可选择连接到所提供的名称）
3. 在无故障的情况下打开输入队列（使用 MQOPEN）
4. 使用 MQBEGIN 启动工作单元
5. 在同步点下从队列中获取下一条消息（使用 MQGET）
6. 从数据库中获取信息
7. 更新数据库信息
8. 使用 MQCMIT 落实更改
9. 打印已更新的信息（未出现消息时计为失败，并且循环结束）
10. 使用 MQCLOSE 关闭该队列
11. 使用 MQDISC 断开与该队列的连接

这些样本中使用了 SQL 游标，因此在处理消息时将锁定对数据库的读操作（即多个实例），并允许同时运行这些程序的多个实例。 系统会显式打开游标，但通过 MQCMIT 调用隐式关闭游标。

单数据库样本（AMQSXAS0 或 AMQ0XAS0）不包含 SQL CONNECT 语句，并且 IBM MQ 通过 MQBEGIN 调用隐式地与数据库建立连接。 多数据库样本（AMQSXAG0 或 AMQ0XAG0、AMQSXAB0 或 AMQ0XAB0，以及 AMQSXAF0 或 AMQ0XAF0）包含 SQL CONNECT 语句，因为某些数据库产品只允许有一个活动连接。 如果您的数据库产品并非如此，或者您正在访问多数据库产品中的单个数据库，那么可除去 SQL CONNECT 语句。

这些样本由 IBM Db2 数据库产品预先准备，因此您可能需要修改这些样本以将其用于其他数据库产品。

SQL 错误检查会使用 Db2 提供的 UTIL.C 和 CHECKERR.CBL 中的例程。 必须先编译或替换这些例程，然后才能对其进行编译和链接。

注: 如果使用 Micro Focus COBOL 源 CHECKERR.MFC 来执行 SQL 错误检查，那么必须将程序标识更改为大写形式（即 CHECKERR），以便 AMQ0XAS0 正确进行链接。

创建数据库和表

在编译样本之前创建数据库和表。

要创建数据库，请使用适用于您数据库产品的常用方法，例如：

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

使用 SQL 语句创建表，如下所示：

在 C 中：

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

在 COBOL 中：

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
```

```
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

使用 SQL 语句在表中输入数据，如下所示：

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
⋮
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
⋮
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
⋮
```

注: 在 COBOL 中，请使用相同的 SQL 语句，但需要在每行末尾添加 END_EXEC。

预编译、编译和链接样本

下面介绍了如何在 C 和 COBOL 语言中预编译、编译和链接样本。

预编译 .SQC 文件（使用 C 语言）和 .SQB 文件（使用 COBOL 语言），并将它们与相应的数据库进行绑定以生成 .C 或 .CBL 文件。 要执行此操作，请使用适用于您数据库产品的典型方法。

在 C 中预编译

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

在 COBOL 中预编译

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
```

```
db2 bind AMQ0XAF0.BND
db2 connect reset
```

编译和链接

以下样本命令使用了符号 *DB2TOP* 和 *MQ_INSTALLATION_PATH*。 *DB2TOP* 表示 Db2 产品的安装目录。 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。

- AIX 在 AIX 上，目录路径为：

  ```
  /usr/lpp/db2_05_00
  ```

- Solaris 在 Solaris 上，目录路径为：

  ```
  /opt/IBMdb2/V5.0
  ```

- Windows 在 Windows 系统上，目录路径取决于安装产品时选择的路径。 如果选择缺省设置，那么路径为：

  ```
  c:\sqllib
  ```

注: 在 Windows 系统上发出链接命令之前，确保 LIB 环境变量包含 Db2 和 IBM MQ 库的路径。

将以下文件复制到临时目录中：

- IBM MQ 安装中的 `amqsxag0.c` 文件

  注: 可在下列目录中找到此文件：

  - Linux UNIX 在 UNIX and Linux 系统上：

    ```
    MQ_INSTALLATION_PATH/samp/xatm
    ```

  - Windows 在 Windows 系统上：

    ```
    MQ_INSTALLATION_PATH\tools\c\samples\xatm
    ```

- 通过预编译 `.sqc` 源文件（`amqsxas0.sqc`、`amqsxaf0.sqc` 和 `amqsxab0.sqc`）而获取的 `.c` 文件。
- Db2 安装中的文件 `util.c` 和 `util.h`。

  注: 可在以下目录中找到这些文件：

  ```
  DB2TOP/samples/c
  ```

使用以下编译器命令为您使用的平台构建每个 `.c` 文件的对象文件：

- AIX AIX

  ```
  xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
  FILENAME.o FILENAME.c
  ```

- Solaris Solaris

  ```
  cc -Aa -KPIC -mt -I MQ_INSTALLATION_PATH
  /inc -I DB2TOP/include -c -o
  FILENAME.o FILENAME.c
  ```

- Windows Windows 系统

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

使用以下链接命令为您使用的平台构建 amqsxag0 可执行文件：

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Solaris Solaris

```
cc -mt -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm -lthread -lsocket -lc -lnsl -ldl util.o
amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows Windows 系统

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

使用以下编译和链接命令为您使用的平台构建 amqsxas0 可执行文件：

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Solaris Solaris

```
cc -mt -L DB2TOP/lib -ldb2-L MQ_INSTALLATION_PATH/lib
-lqm -lthread -lsocket -lc -lnsl -ldl util.o
amqsxas0.o -o amqsxas0
```

- Windows Windows 系统

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

其他信息

AIX 如果您正在使用 AIX 并且想要访问 Oracle，请使用 xlc_r 编译器并链接到 libmqm_r.a。

运行样本
可使用以下信息来了解如何在 C 和 COBOL 中运行“数据库协调”样本之前配置队列管理器。

在运行样本之前，使用所用数据库产品配置队列管理器。 有关如何执行此操作的信息，请参阅场景 1：队列管理器执行协调。

以下标题提供了有关如何在 C 和 COBOL 中运行样本的信息：

- 第 988 页的『C 样本』
- 第 989 页的『COBOL 样本』

C 样本

必须从队列中读取以下格式的消息：

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT 可用于将消息放入队列中。

“数据库协调”样本采用以下两个参数：

1. 队列名称（必需）
2. 队列管理器名称（可选）

假定您已使用队列 singDBQ 为单数据库样本 singDBQM 创建和配置了队列管理器，您计划将 Mr Fred Bloggs 的帐户金额增加 50，如下所示：

```
AMQSPUT singDBQ singDBQM
```

然后，输入以下消息：

```
UPDATE Balance change=50 WHERE Account=1
```

您可以将多条消息放入队列中。

```
AMQSXAS0 singDBQ singDBQM
```

然后，将打印 Mr Fred Bloggs 帐户的更新状态。

假定您已使用队列 multDBQ 为多数据库样本 multDBQM 创建和配置了队列管理器，您计划将 Ms Mary Brown 的帐户金额减少 75，如下所示：

```
AMQSPUT multDBQ multDBQM
```

然后，输入以下消息：

```
UPDATE Balance change=-75 WHERE Account=3
```

您可以将多条消息放入队列中。

```
AMQSXAG0 multDBQ multDBQM
```

然后，将打印 Ms Mary Brown 帐户的更新状态。

COBOL 样本

必须从队列中读取以下格式的消息：

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

为简便起见，`Balance change` 必须是带符号的八字符数字，而 `Account` 必须是八字符数字。

样本 AMQSPUT 可用于将消息放入队列中。

这些样本不使用任何参数，而是使用缺省队列管理器。 可配置为在任何时候都只运行一个样本。 假定您已使用队列 singDBQ 为单数据库样本配置了缺省队列管理器，您计划将 Mr Fred Bloggs 的帐户金额增加 50，如下所示：

```
AMQSPUT singDBQ
```

然后，输入以下消息：

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

您可以将多条消息放入队列中：

```
AMQ0XAS0
```

输入队列的名称：

```
singDBQ
```

然后，将打印 Mr Fred Bloggs 帐户的更新状态。

假定您已使用队列 multDBQ 为多数据库样本配置了缺省队列管理器，您计划将 Ms Mary Brown 的帐户金额减少 75，如下所示：

```
AMQSPUT multDBQ
```

然后，输入以下消息：

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

您可以将多条消息放入队列中：

```
AMQ0XAG0
```

输入队列的名称：

```
multDBQ
```

然后，将打印 Ms Mary Brown 帐户的更新状态。

“死信队列处理程序”样本

已提供了一个样本死信队列处理程序，可执行版本的名称为 amqsdlq。 如果您希望使用与 RUNMQDLQ 不同的死信队列处理程序，那么该样本的来源可供您用作基础。

该样本类似于本产品中提供的死信处理程序，但是跟踪和错误报告有所不同。 您可以使用以下两个环境变量：

ODQ_TRACE
设置为 YES 或 yes 可开启跟踪

ODQ_MSG
设置为包含错误和参考消息的文件的名称。 提供的文件是 amqsdlq.msg。

根据您的平台，您需要使用 **export** 或 **set** 命令向您的环境标识这些变量；可使用 **unset** 命令关闭跟踪。

您可以根据自己的需求来修改错误消息文件 amqsdlq.msg。 该样本将消息放入 stdout 中，而**不是**放入 IBM MQ 错误日志文件中。

针对您的平台的 管理 或《系统管理指南》阐述了死信处理程序的工作方式以及运行方式。

“分发列表”样本程序

“分发列表”样本 amqsptl0 提供了有关将消息放入多个消息队列的示例。 其基于 MQPUT 样本 amqsput0。

运行“分发列表”样本 amqsptl0

“分发列表”样本与“放置”样本的运行方式相似。

它采用以下参数：

- 队列名称
- 队列管理器名称

应成对输入这些值。 例如：

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

使用 MQOPEN 打开队列，并使用 MQPUT 将消息放入队列中。 如果系统未检测到队列名称或队列管理器名称，那么将返回原因码。

请记住在队列管理器之间定义通道，以便能够在它们之间传递消息。 该样本程序不会为您执行此操作。

设计“分发列表”样本

“放置消息记录”(MQPMR) 指定每个目标的消息属性。 该样本提供了 *MsgId* 和 *CorrelId* 的值，而这些值将覆盖 MQMD 结构中指定的值。

MQPMO 结构中的 *PutMsgRecFields* 字段指示 MQPMR 中会包含哪些字段：

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

接下来，该样本会分配响应记录和对象记录。 “对象记录”(MQOR) 需要偶数个名称，且至少需要一对名称（即 *ObjectName* 和 *ObjectQMgrName*）。

下一个阶段包括使用 MQCONN 连接到队列管理器。 该样本会尝试连接到与 MQOR 中的第一个队列相关联的队列管理器；如果失败，那么将依次遍历各个对象记录。 如果无法连接到任何队列管理器，那么您会收到通知，并且该程序会退出。

使用 MQOPEN 打开目标队列，并使用 MQPUT 将消息放入这些队列中。 将在“响应记录”(MQRR) 中报告任何问题和故障。

最后，使用 MQCLOSE 关闭目标队列，并且该程序使用 MQDISC 断开与队列管理器的连接。 对于声明了 *CompCode* 和 *Reason* 的每个调用，将使用相同的响应记录。

“回传”样本程序

“回传”样本程序将消息从消息队列回传至应答队列。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

这些程序旨在作为触发程序运行。

在 IBM i、 UNIX, Linux, and Windows 系统上，它们的唯一输入是包含目标队列名称和队列管理器名称的 MQTMC2（触发器消息）结构。 COBOL 版本使用缺省队列管理器。

IBM i 在 IBM i 上，要使触发过程起作用，请确保要使用的 Echo 样本程序由到达队列 SYSTEM.SAMPLE.ECHO。 要执行此操作，请在进程定义 SYSTEM.SAMPLE.ECHOPROCESS 的 *ApplId* 字段中指定您要使用的“回传”样本程序的名称。 （为实现此目的，您可以使用 CHGMQMPRC 命令；要获取详细信息，请参阅更改 MQ 进程 (CHGMQMPRC)。） 样本队列具有触发器类型 FIRST，因此，如果在运行“请求”样本之前队列中已存在消息，那么您发送的消息将不会触发“回传”样本。

在正确设置了该定义后，首先在一个作业中启动 AMQSERV4，然后在另一个作业中启动 AMQSREQ4。 您可以使用 AMQSTRG4 代替 AMQSERV4，但作业提交可能出现延迟，因而导致不太容易关注到当前发生的情况。

使用“请求”样本程序向队列 SYSTEM.SAMPLE.ECHO 发送消息。 “回传”样本程序将向请求消息中指定的应答队列发送包含请求消息数据的应答消息。

设计“回传”样本程序

该程序将打开在启动时传递的触发器消息结构中指定的队列。 （为清楚起见，我们将其称为请求队列。） 该程序将使用 MQOPEN 调用打开该队列以获取共享输入。

该程序将使用 MQGET 调用从此队列中除去消息。 此调用将使用 MQGMO_ACCEPT_TRUNCATED_MSG、MQGMO_CONVERT 和 MQGMO_WAIT 选项，并将等待时间间隔设置为 5 秒。 该程序将测试每条消息的描述符以判断其是否为请求消息；如果不是，那么该程序会丢弃该消息并显示一条警告消息。

对于每行输入，该程序随后将文本读入缓冲区中，并使用 MQPUT1 调用将包含此行文本的请求消息放入应答队列中。

如果 MQGET 调用失败，那么该程序会将报告消息放入应答队列中，并将消息描述符的 *Feedback* 字段设置为 MQGET 返回的原因码。

如果请求队列中未剩下任何消息，那么该程序将关闭此队列并断开与队列管理器的连接。

IBM i 在 IBM i 上，该程序还可以响应从 IBM MQ for IBM i 以外的平台发送到该队列的消息，即使未提供适用于此情况的样本也是如此。 要使 ECHO 程序正常运行：

- 编写一个程序，并正确指定 **Format**、**Encoding** 和 **CCSID** 参数以发送文本请求消息。

  “回传”程序将请求队列管理器执行消息数据转换（如果需要）。
- 如果您编写的程序未针对应答提供类似转换，请在 IBM MQ for IBM i 发送通道上指定 CONVERT(*YES)。

“获取”样本程序

Get 样本程序使用 MQGET 调用从队列中获取消息。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

设计“获取”样本程序

该程序使用带有 MQOO_INPUT_AS_Q_DEF 选项的 MQOPEN 调用来打开目标队列。 如果无法打开该队列，那么该程序将显示包含 MQOPEN 调用返回的原因码的错误消息。

对于队列中的每条消息，该程序使用 MQGET 调用从队列中除去该消息，并显示该消息包含的数据。 MQGET 调用使用 MQGMO_WAIT 选项并将 *WaitInterval* 指定为 15 秒，因此，如果队列中不存在消息，那么该程序将等待此时间段。 如果在该时间间隔到期前没有消息到达该队列，那么该调用将失败并返回 MQRC_NO_MSG_AVAILABLE 原因码。

该程序说明在每个 MQGET 调用之后如何清除 MQMD 结构的 *MsgId* 和 *CorrelId* 字段，因为该调用将这些字段设置为其检索的消息中包含的值。 清除这些字段意味着连续 MQGET 调用按照消息在队列中的存放顺序检索消息。

MQGET 调用指定固定大小的缓冲区。 如果消息大小超出此缓冲区的大小，那么该调用将失败且该程序会停止。

该程序将继续运行，直至 MQGET 调用返回 MQRC_NO_MSG_AVAILABLE 原因码或者 MQGET 调用失败。 如果该调用失败，那么该程序将显示包含原因码的错误消息。

然后，程序使用 MQCLOSE 调用来关闭队列。

运行 *amqsget* 和 *amqsgetc* 样本

这些程序采用以下位置参数：

1. 源队列的名称（必需）
2. 队列管理器名称（可选）

   如果未指定队列管理器，那么 amqsget 将连接到缺省队列管理器，而 amqsgetc 将连接到环境变量或客户机通道定义文件所指定的队列管理器。
3. 打开选项（可选）

   如果未指定打开选项，那么该样本将使用值 8193，这是以下两个选项的组合：

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. 关闭选项（可选）

   如果未指定关闭选项，那么该样本将使用值 0，即 MQCO_NONE。

这些程序还使用名为 **MQSAMP_USER_ID** 的环境变量，该变量应设置为要用于连接认证的用户标识。 如果设置了此环境变量，那么该程序会提示输入该用户标识及相应密码。

要运行这些程序，请输入以下内容之一：

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

其中，`myqueue` 是程序将从中获取消息的队列的名称，而 `qmanagername` 是拥有 `myqueue` 的队列管理器。

使用 amqsget 和 amqsgetc

请注意，**amqsget** 通过使用共享内存连接到队列管理器来执行到队列管理器的本地连接，因此只能在队列管理器所在的系统上运行，而 **amqsgetc** 执行客户机样式连接（即使连接到同一系统上的队列管理器也是如此）。

使用 **amqsgetc** 时，您需要提供有关如何实际连接到队列管理器的应用程序详细信息， 即队列管理器主机或 IP 地址和队列管理器侦听器端口。

通常，这可以使用 MQSERVER 环境变量或通过使用客户机通道定义表定义连接详细信息完成，也可以使用环境变量将信息提供给 **amqsgetc**；有关示例，请参阅 MQCCDTURL。

使用 MQSERVER 连接到本地队列管理器的示例，该队列管理器的侦听器在端口 1414 上运行，并使用缺省服务器连接通道：

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

“高可用性”样本程序

amqsghac、**amqsphac** 和 **amqsmhac** 高可用性样本程序使用客户机自动重新连接来说明在发生队列管理器故障后的恢复过程。 **amqsfhac** 将检查使用联网存储器的队列管理器在发生故障后是否维护数据完整性。

amqsghac、**amqsphac** 和 **amqsmhac** 程序都通过命令行来启动，并且可组合使用以说明在多实例队列管理器的一个实例发生故障后的重新连接过程。

也可以使用 **amqsghac**、**amqsphac** 和 **amqsmhac** 样本来说明客户机重新连接到单实例队列管理器（通常配置为队列管理器组）的过程。

在此示例中，为简便起见并便于配置，将说明样本程序如何重新连接到已启动、停止然后重新启动的单实例队列管理器；请参阅第 995 页的『设置和控制队列管理器』。

同时使用 **amqsfhac** 和 **amqmfsck** 以检查文件系统完整性。 请参阅 **amqmfsck** (文件系统检查) 和 验证共享文件系统行为 以获取更多信息。

amqsphac *queueName* [*qMgrName*]

- **amqsphac** 是一个 IBM MQ MQI client 应用程序。 它将消息序列放入队列中（各条消息之间存在两秒延迟），并显示已发送到相应事件处理程序的事件。
- 在未使用同步点的情况下将消息放入队列中。
- 可重新连接到同一队列管理器组中的任何队列管理器。

amqsghac *queueName* [*qMgrName*]

- **amqsghac** 是一个 IBM MQ MQI client 应用程序。 它从队列中获取消息，并显示已发送到相应事件处理程序的事件。
- 在未使用同步点的情况下从队列中获取消息。
- 可重新连接到同一队列管理器组中的任何队列管理器。

amqsmhac -s *sourceQueueName* -t *targetQueueName* [-m *qMgrName*] [-w *waitInterval*]

- **amqsmhac** 是一个 IBM MQ MQI client 应用程序。 它将消息从一个队列复制到另一个队列中，在程序完成前收到最后一条消息后的缺省等待时间间隔为 15 分钟。
- 在同步点内复制这些消息。

- 只能重新连接到相同的队列管理器。

amqsfhac *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* (0 | 1 | 2)

- **amqsfhac** 是一个 IBM MQ MQI client 应用程序。 它检查使用联网存储器（例如，NAS 或集群文件系统）的 IBM MQ 多实例队列管理器是否维护数据完整性。 请执行这些步骤以在验证共享文件系统行为中运行 **amqsfhac**。
- 在连接到 *QueueManagerName* 时，它使用 MQCNO_RECONNECT_Q_MGR 选项。 在队列管理器执行故障转移时，它会自动重新连接。
- 在使队列管理器执行任意次数的故障转移期间，它将 *InTransactionCount*RepeatCount* 持久消息放入 *QueueName* 中。 **amqsfhac** 每次都会重新连接到队列管理器，并继续运行。 此测试可确保不丢失任何消息。
- *InTransactionCount* 消息将放入每个事务中。 该事务将重复 *RepeatCount* 次。 如果某个事务中发生故障，那么 **amqsfhac** 将回滚并在 **amqsfhac** 重新连接到队列管理器时重新提交该事务。
- 它还会将消息放入 *SideQueueName* 中。 它使用 *SideQueueName* 来检查是从 *QueueName* 中成功落实还是回滚所有消息。 如果检测到不一致情况，那么它将返回一条错误消息。
- 可通过将最后一个参数设置为 (0|1|2) 来更改 **amqsfhac** 的输出跟踪量。

  0
  最小输出量。

  1
  中等输出量。

  2
  最大输出量。

配置客户机连接

您需要配置客户机和服务器连接通道，才能运行这些样本。 客户机验证过程将阐述如何设置客户机测试环境。

也可以使用以下示例中提供的配置。

有关使用 amqsghac、amqsphac 和 amqsmhac 的示例

此示例说明了使用单实例队列管理器的可重新连接的客户机。

amqsphac 用于将消息放入队列 SOURCE 中，**amqsmhac** 用于将消息传输到 TARGET，而 **amqsghac** 用于从 TARGET 中检索消息；请参阅第 994 页的图 134。

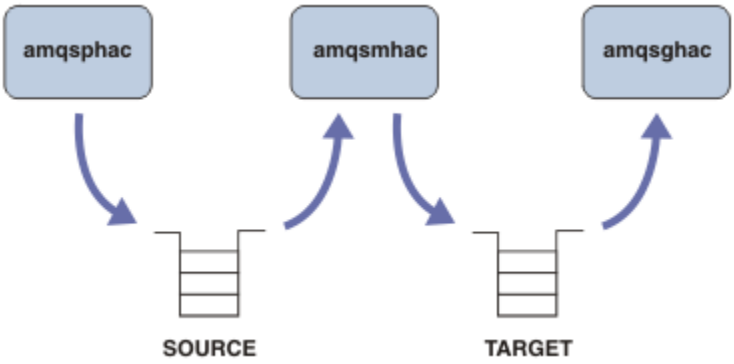


图 *134:* 可重新连接的客户机样本

请执行以下步骤来运行这些样本。

1. 创建包含以下命令的文件 hasamples.tst:

```
DEFINE QLOCAL(SOURCE) REPLACE
```

```
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. 在命令提示符处输入以下命令：
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. 将环境变量 **MQCHLLIB** 设置为 AMQCLCHL.TAB 客户机通道定义文件的路径；例如，SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc。
4. 在设置了 **MQCHLLIB** 的情况下打开三个新窗口；例如，在 Windows 上，在其中一个窗口中用于启动每个程序的先前命令提示符处输入 **start** 三次。 请参阅第 995 页的『设置和控制队列管理器』中的步骤 第 996 页的『5』。
5. 输入命令 `endmqm -r -p QM1` 以停止该队列管理器，然后允许客户机重新连接。
6. 输入命令 `strmqm QM1` 以重新启动该队列管理器。

在 Windows 上运行 **amqsghac**， **amqsphac** 和 **amqsmhac** 样本的结果显示在以下示例中。

设置和控制队列管理器

1. 创建队列管理器。

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   请记住稍后要设置 **MQCHLLIB** 变量的数据目录。
2. 启动队列管理器。

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. 创建队列和通道，修改侦听器端口，然后启动侦听器和通道。

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
```

```
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. 向客户机公开客户机通道表。

   使用从步骤 第 995 页的『1』 中的 **crtmqm** 命令返回的数据目录，并向其中添加目录 `@ipcc` 以设置 **MQCHLLIB** 变量。

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. 在其他窗口中启动样本程序

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. 终止队列管理器并重新启动。

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
```

```
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

相关任务

验证共享文件系统行为

相关参考

amqmfsck（文件系统检查）

“查询”样本程序

“查询”样本程序使用 MQINQ 调用来查询队列的某些属性。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

这些程序旨在作为触发的程序运行，因此，这些程序的唯一输入是 IBM i、Windows 和 UNIX and Linux 系统的 MQTMC2（触发器消息）结构。 此结构包含要查询其属性的目标队列的名称。 C 版本还使用队列管理器名称。 COBOL 版本使用缺省队列管理器。

要使触发进程正常运行，请确保到达队列 SYSTEM.SAMPLE.INQ 的消息能触发您要使用的“查询”样本程序。 要执行此操作，请在进程定义 SYSTEM.SAMPLE.INQPROCESS 的 *ApplicId* 字段中指定您要使用的“查询”样本程序的名称。 IBM i 对于 IBM i，您可以使用 CHGMQMPRC 命令来实现此目的；有关详细信息，请参阅更改 MQ 进程 (CHGMQMPRC)。 样本队列具有触发器类型 FIRST；如果在运行“请求”样本之前队列中已存在消息，那么您发送的消息将不会触发“查询”样本。

在正确设置了该定义后，请执行以下操作：

- ULW 对于 UNIX, Linux, and Windows，在一个会话中启动 **runmqtrm** 程序，然后在另一个会话中启动 amqsreq 程序。
- IBM i 对于 IBM i，在一个会话中启动 AMQSERV4 程序，然后在另一个会话中启动 AMQSREQ4 程序。 您可以使用 AMQSTRG4 代替 AMQSERV4，但作业提交可能出现延迟，因而导致不太容易关注到当前发生的情况。

使用“请求”样本程序来将请求消息（每条消息仅包含一个队列名称）发送到队列 SYSTEM.SAMPLE.INQ。 对于每条请求消息，“查询”样本程序都会发送应答消息，其中包含有关在请求消息中指定的队列的信息。 将向请求消息中指定的应答队列发送应答。

IBM i 在 IBM i 上，如果使用了样本输入文件成员 QMQMSAMP.AMQSDATA(INQ)，那么指定的最后一个队列不存在，因此该样本返回包含故障原因码的报告消息。

设计“查询”样本程序

该程序将打开在启动时传递的触发器消息结构中指定的队列。 （为清晰起见，我们将此队列称为请求队列。） 该程序将使用 MQOPEN 调用打开该队列以获取共享输入。

该程序将使用 MQGET 调用从此队列中除去消息。 此调用将使用 MQGMO_ACCEPT_TRUNCATED_MSG 和 MQGMO_WAIT 选项，并将等待时间间隔设置为 5 秒。 该程序将测试每条消息的描述符以判断其是否为请求消息；如果不是，那么该程序会丢弃该消息并显示一条警告消息。

对于从请求队列中除去的每条请求消息，该程序都会读取数据中包含的队列名称（称为目标队列），然后使用带有 MQOO_INQ 选项的 MQOPEN 调用来打开该队列。 然后，该程序使用 MQINQ 调用来查询目标队列的 *InhibitGet*、**CurrentQDepth** 和 **OpenInputCount** 属性的值。

如果 MQINQ 调用成功，那么该程序使用 MQPUT1 调用来将应答消息放入应答队列中。 此消息包含这三个属性的值。

如果 MQOPEN 或 MQINQ 调用不成功，那么该程序使用 MQPUT1 调用来将报告消息放入应答队列中。 此报告消息的消息描述符的 *Feedback* 字段值是 MQOPEN 或 MQINQ 调用（具体取决于失败的调用）返回的原因码。

在 MQINQ 调用完成后，该程序使用 MQCLOSE 调用来关闭目标队列。

如果请求队列中未剩下任何消息，那么该程序将关闭此队列并断开与队列管理器的连接。

“查询消息句柄属性”样本程序

AMQSIQMA 是一个样本 C 程序，用于查询消息队列中消息句柄的属性；它还是一个使用 MQINQMP API 调用的示例。

此样本创建一个消息句柄，并将其放入 MQGMO 结构的 MsgHandle 字段中。 然后，此样本获取一条消息，并查询和打印用于填充消息句柄的所有属性。

```
C:\Program Files\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

“发布/预订”样本程序

Publish/Subscribe 样本程序演示在 IBM MQ 中使用发布和预订功能。

提供了三个 C 语言样本程序，用于说明如何通过编程来设定 IBM MQ 发布/预订接口。 提供了一些使用旧接口的 C 样本，还提供了一些 Java 样本。 Java 样本使用 com.ibm.mq.jar 中的 IBM MQ 发布/预订接口以及 com.ibm.mqjms 中的 JMS 发布/预订接口。 本主题中未涵盖 JMS 样本。

C

在 C 样本文件夹中查找发布者样本 amqspub。 通过将所需的主题名称作为第一个参数并后跟可选的队列管理器名称来运行该样本。 例如，amqspub mytopic QM3。 还有一个名为 amqspubc 的客户机版本。 如果您选择运行该客户机版本，请先查看第 972 页的『配置队列管理器以接受多平台上的客户机连接』以获取详细信息。

发布程序连接到缺省队列管理器，并通过输出 target topic is mytopic 来作出响应。 从现在开始，在此窗口中输入的每一行都将发布到 mytopic。

在同一目录中打开另一个命令窗口，运行订户程序 amqssub 并为其提供相同的主题名称和可选的队列管理器名称。 例如，amqssub mytopic QM3。

预订程序通过输出 Calling MQGET : 30 seconds wait time 来作出响应。 从现在开始，在发布程序中输入的行都会出现在预订程序的输出中。

在另一个命令窗口中启动另一个预订程序，并观察两个预订程序是否都收到发布内容。

要获取有关这些参数的完整文档（包括设置选项），请参阅样本源代码。 以下主题中描述了预订程序选项字段的值：选项 (MQLONG)。

还有一个订户样本 amqssbx，此样本提供了其他预订选项作为命令行开关。

输入 amqssbx -d mysub -t mytopic -k，以使用终止预订程序后保留的持久预订来调用该预订程序。

通过使用发布程序发布另一个项来测试预订情况。 等待 30 秒，以使预订程序彻底终止。 在同一主题下发布更多的项。 重新启动预订程序。 在重新启动后，预订程序会立即显示预订程序未在运行时发布的最后一个项。

C 传统

同时还提供了另一组用于说明排队命令的 C 样本。 其中一些样本最初随 MQ0C Supportpac 一起提供。 鉴于兼容性原因，完全支持这些样本所说明的功能。

建议不要使用排队命令接口。 这比发布/预订 API 要复杂得多，而且在功能方面也没有强制要求编写复杂的排队命令。 但是，您可能发现排队方法更合适一些，这可能是因为已经使用了这种接口，或是因为编程环境能够简化构建复杂消息和调用常规 MQPUT 的过程（而不是构造不同的 MQSUB 调用）。

其他样本位于 samples 文件夹中的 pubsub 子目录中。

第 999 页的表 170 中列出了六种样本类型。

表 *170:* 传统发布/预订样本 *C* 程序的类别

类别	程序	注释
RFH1	amqssr1a.c amqspr1a.c	使用 RFH1 格式消息构建的简单发布/预订示例。
RFH2	amqssr2a.c amqspr2a.c	使用 RFH2 格式消息构建的简单发布/预订示例。
MQAI 样本	amqsppca.c amqsspca.c	使用 PCF 命令和 MQAI 命令接口构建的简单发布/预订示例。
使用 RFH1 的 MA0C 结果服务	amqsgama.c amqsresa.c	使用 RFH1 头构建的结果服务 1. 需要 amqsgama.tst 和 amqsresa.tst 中定义的队列 2. amqsresa 必须在 amqsgama 之前启动
使用 RFH2 的 MA0C 结果服务	amqsgr2a.c amqsrr2a.c	使用 RFH2 头构建的结果服务 1. 需要 amqsgama.tst 和 amqsresa.tst 中定义的队列 2. amqsresa 必须在 amqsgama 之前启动
路由出口发布/预订样本	amqspsra.c	说明如何更改路由出口发布/预订消息的队列或队列管理器目标。

Java 的样本程序

Java 样本 MQPubSubApiSample.java 将发布者和订户合并到单个程序中。 其源和编译类文件位于 wmqjava 样本文件夹中。

如果您选择以客户机模式运行，请先查看第 972 页的『配置队列管理器以接受多平台上的客户机连接』以获取详细信息。

如果已配置 Java 环境，请从命令行中使用 Java 命令运行该样本。 您还可以从已设置 Java 编程工作台的 IBM MQ Explorer Eclipse 工作空间运行样本。

您可能需要更改该样本程序的某些属性，然后再运行该样本程序。 可通过向 JVM 提供参数或者编辑源代码来执行此操作。

第 999 页的『运行 MQPubSubApiSample Java 样本』中的指示信息说明了如何从 Eclipse 工作空间中运行该样本。

运行 *MQPubSubApiSample Java* 样本
如何在 Eclipse 平台上使用 Java 开发工具运行 MQPubSubApiSample。

开始之前

打开 Eclipse 工作台。 创建新的工作空间目录并将其选中。 关闭欢迎窗口。

在作为客户机运行之前，请执行第 972 页的『配置队列管理器以接受多平台上的客户机连接』中的步骤。

关于此任务

Java 发布/预订样本程序是一个 IBM MQ MQI client Java 程序。 使用缺省队列管理器（侦听端口 1414）在不做修改的情况下运行该样本。 此任务描述了这种简单的情况，笼统地说明如何提供参数和修改该样本以满足不同的 IBM MQ 配置需求。 所阐述的示例在 Windows 上运行。 在其他平台上，文件路径会有所不同。

过程

1. 导入 Java 样本程序
   a) 在工作台中，单击**窗口** > **打开透视图** > **其他** > **Java**，然后单击**确定**。
   b) 切换到**"包资源管理器"**视图。
   c) 在**"包资源管理器"**视图的空白区域中单击鼠标右键。 单击**新建** > **Java 项目**。
   d) 在 **Project name** 字段中，输入 MQ Java Samples。 单击**下一步**。
   e) 在 **Java Settings** 面板中，切换到 **库** 选项卡。
   f) 单击**添加外部 JAR**。
   g) 浏览至 *MQ_INSTALLATION_PATH* \java\lib（其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 安装文件夹）并选择 com.ibm.mq.jar 和 com.ibm.mq.jmqi.jar
   h) 单击**打开** > **完成**。
   i) 在**包资源管理器**视图中右键单击 src。
   j) 选择**导入...** > **常规** > **文件系统** > **下一步** > **浏览**... 并浏览至路径 *MQ_INSTALLATION_PATH* \tools\wmqjava\samples，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 安装目录。
   k) 在**导入**面板第 1001 页的图 135 上，单击 samples（不选中复选框）。
   l) 选择 MQPubSubApiSample.java。 **Into folder** 字段应包含 MQ Java Samples/src。 单击**完成**。

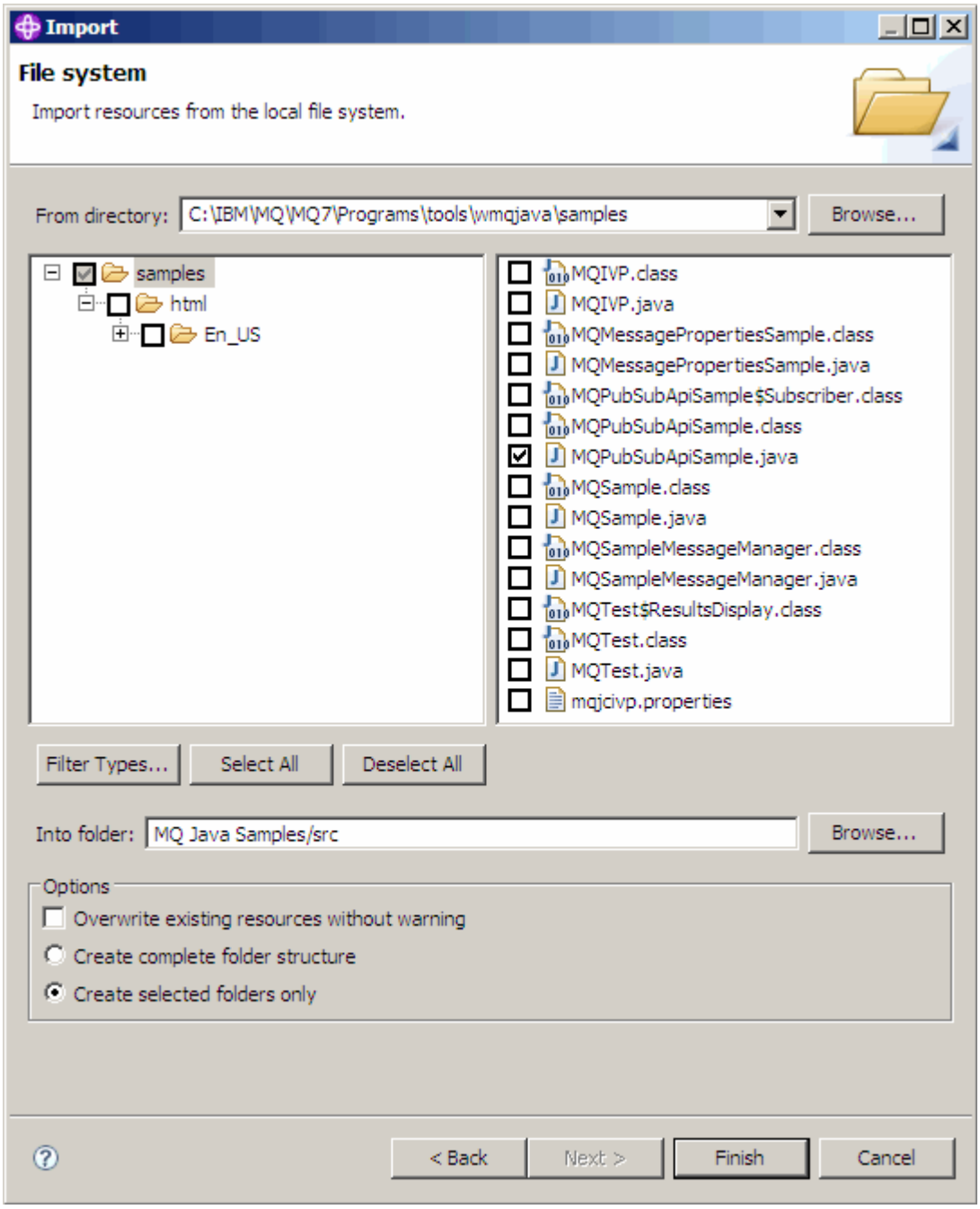


图 *135:* 文件系统导入

2. 运行发布/预订样本程序。

   可通过两种方式来运行该程序，具体取决于是否需要更改缺省参数。

   - 第一个选项是在不做任何更改的情况下运行该程序：
     - 在工作空间主菜单中，展开 `src` 文件夹。 右键单击 **MQPubSubApiSample.java 运行方式 > 1. Java 应用程序**
   - 第二个选项是在使用参数或针对环境修改的源代码的情况下运行该程序：
     - 打开 `MQPubSubApiSample.java` 并研究 `MQPubSubApiSample` 构造函数。
     - 修改该程序的属性。

       可以通过 -D JVM 开关或者通过编辑源代码来提供 System p 属性缺省值以修改以下属性。

- topicObject
- queueManagerName
- subscriberCount

只能通过编辑构造函数中的源代码来更改以下属性。

- hostname
- port
- 通道

要设置 System p 属性，请在存取器中通过编码来指定缺省值，例如：

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

或者，通过 -D 选项向 JVM 提供参数，如以下步骤中所示：

a. 复制要设置的 System.Property 的全名，例如：`com.ibm.mq.pubSubSample.queueManagerName`。
b. 在工作空间中，右键单击**运行 > 打开运行对话框**。 在 **创建，管理和运行应用程序** 中双击 Java 应用程序，然后单击 **(x) = Arguments** 选项卡。
c. 在**“VM 自变量：”**窗格中，输入 -D 并粘贴 System.property 名称 `com.ibm.mq.pubSubSample.queueManagerName`，后跟 =QM3。 单击**应用 > 运行**。
d. 以逗号分隔列表形式，或者以窗格中的附加行（不使用逗号分隔符）形式添加更多的自变量。

例如：`-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3,`
`-Dcom.ibm.mq.pubSubSample.subscriberCount=6`。

“发布出口”样本程序

AMQSPSE0 是用于在将发布内容交付到预订程序之前拦截发布内容的出口的样本 C 程序。 例如，出口可以更改消息头、有效内容或目标，或阻止将消息发布给订户。

要运行该样本，请执行以下任务：

1. 配置队列管理器：
   - Linux UNIX 在 UNIX and Linux 系统上，将与下面类似的节添加到 `qm.ini` 文件中：

     ```
     PublishSubscribe:
     PublishExitPath=Module
     PublishExitFunction=EntryPoint
     ```

     其中模块为 *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`。 *MQ_INSTALLATION_PATH* 表示 IBM MQ 安装所在的高级目录。
   - Windows 在 Windows 上，在注册表中设置同等属性。
2. 确保 IBM MQ 可访问该模块。
3. 重新启动队列管理器以使配置生效。
4. 在要跟踪的应用程序进程中，描述要用于写入跟踪文件的位置。 例如：
   - Linux UNIX 在 UNIX and Linux 系统上，确保目录 `/var/mqm/trace` 存在并导出以下环境变量：

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

- Windows 在 Windows 上，确保目录 C:\temp 存在并设置以下环境变量：

```
set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
```

“放置”样本程序

Put 样本程序使用 MQPUT 调用将消息放在队列上。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

设计“放置”样本程序

该程序使用具有 MQOO_OUTPUT 选项的 MQOPEN 调用来打开目标队列以便放入消息。

如果无法打开队列，那么该程序将输出错误消息，其中包含 MQOPEN 调用返回的原因码。 为保持程序简单，程序将在此和后续的 MQI 调用中使用许多选项的缺省值。

对于每行输入，该程序将文本读入缓冲区并使用 MQPUT 调用来创建包含此行文本的数据报消息。 该程序继续运行，直至到达输入的结尾或者 MQPUT 调用失败。 如果程序到达输入的结尾，那么它使用 MQCLOSE 调用来关闭队列。

运行“放置”样本程序

运行 amqsput 和 amqsputc 样本

ULW

amqqsput 样本是用于使用本地绑定放置消息的程序，amqsputc 样本是用于使用客户机绑定放置消息的程序。 这些程序采用以下位置参数：

1. 目标队列的名称（必需）
2. 队列管理器名称（可选）

   如果未指定队列管理器，那么 amqsput 将连接到缺省队列管理器，而 amqsputc 将连接到 MQSERVER 环境变量或客户机通道定义文件所指定的队列管理器。
3. 打开选项（可选）

   如果未指定打开选项，那么该样本将使用值 8208，这是以下两个选项的组合：

   - MQOO_OUTPUT
   - MQOO_FAIL_IF_QUIESCING
4. 关闭选项（可选）

   如果未指定关闭选项，那么该样本将使用值 0，即 MQCO_NONE。
5. 目标队列管理器的名称（可选）

   如果未指定目标队列管理器，那么 MQOD 中的 ObjectQMgrName 字段将保留为空。
6. 动态队列名称（可选）

   如果未指定动态队列名称，那么 MQOD 中的 DynamicQName 字段将保留为空。

这些程序还使用名为 **MQSAMP_USER_ID** 的环境变量，该变量应设置为要用于连接认证的用户标识。 如果设置了此环境变量，那么该程序会提示输入该用户标识及相应密码。

要运行这些程序，请输入以下内容之一：

- `amqsput myqueue qmanagername`
- `amqsputc myqueue qmanagername`

其中 myqueue 是消息即将放到的队列的名称，qmanagername 是拥有 myqueue 的队列管理器。

运行 amq0put 样本

ULW

COBOL 版本没有任何参数。 它连接到缺省队列管理器，并且在运行它时会出现以下提示：

```
Please enter the name of the target queue
```

它从 StdIn 获取输入，并将每一行输入添加到目标队列中。 空行指示无更多数据。

运行 AMQSPUT4 C 样本 (IBM i)

IBM i

C 程序 AMQSPUT4 仅可用于 IBM i 平台，并通过从大量源文件中读取数据来创建消息。

在启动该程序时，必须指定该文件名作为参数。 该文件的结构必须为：

```
queue name
text of message 1
text of message 2
⋮
text of message n
blank line
```

库 QMQMSAMP 文件 AMQSDATA 成员 PUT 中提供了“放置”样本程序的输入样本。

注: 请记住队列名称区分大小写。 样本文件创建程序 AMQSAMP4 所创建的所有队列的名称均采用大写形式。

C 程序将消息放入该文件第一行中指定的队列中；您可以使用所提供的队列 SYSTEM.SAMPLE.LOCAL。 该程序将该文件中以下各行文本放入单独的数据报消息中，并且在读取到文件末尾的空行时停止。

如果使用示例数据文件，那么命令为：

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

运行 AMQ0PUT4 COBOL 样本 (IBM i)

IBM i

COBOL 程序 AMQ0PUT4 仅可用于 IBM i 平台，并通过接受来自键盘的数据以创建消息。

要启动该程序，请在指定目标队列名称作为程序参数的情况下调用该程序。 该程序接受来自键盘的输入并将其放入缓冲区中，然后针对每行文本创建一条数据报消息。 当键盘上输入空行时，该程序将停止。

“参考消息”样本程序

“参考消息”样本允许将大对象从一个节点传输到另一个节点（通常在不同的系统上），而不需要将对象存储在源节点或目标节点上的 IBM MQ 队列上。

提供了一组样本程序，用于说明如何将参考消息放入队列中、消息出口如何接收参考消息以及如何从队列中获取参考消息。 样本程序使用参考消息来移动文件。 如果您想要移动其他对象（如数据库）或者想要执行安全性检查，请基于样本 amqsxrm 来定义自己的出口。

要使用的“参考消息”出口样本程序的版本取决于用于运行通道的平台：

- 在所有平台上，在发送端使用 amqsxrma。
- 如果接收器在 IBM i 以外的任何平台下运行，请在接收端使用 amqsxrma。
- IBM i 如果接收器在 IBM i 下运行，请使用 amqsxrm4。

.

IBM i *IBM i 用户的注意事项*

要使用样本消息出口接收参考消息，请在 IFS 或任何子目录的根文件系统中指定文件，以便可以创建流文件。

IBM i 上的样本消息出口将创建该文件，将数据转换为 EBCDIC，并将代码页设置为系统代码页。 然后，您可以使用 CPYFRMSTMF 命令将此文件复制到 QSYS.LIB 文件系统。 例如：

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

CPYFRMSTMF 命令并不会创建该文件。 在运行此命令之前，您必须创建该文件。

如果从 QSYS.LIB 发送文件，那么无需对样本进行任何更改。 对于任何其他文件系统，确保在 MQRMH 结构的 CodedCharSetId 字段中指定的 CCSID 与正在发送的批量数据相匹配。

在使用集成文件系统时，使用 SYSIFCOPT(*IFSIO) 选项集创建程序模块。 如果您想要移动数据库或定长记录文件，请基于所提供的样本 AMQSXRM4 定义自己的出口。

传输数据库文件的推荐方法是使用 CPYTOSTMF 命令将其转换为 IFS 结构，然后发送附加了 IFS 文件的参考消息。 如果选择通过从 IFS 引用数据库文件来传输该文件，但不将其转换为 IFS 结构，那么必须指定成员名。 如果选择此方法，那么不能保证数据完整性。

Multi 运行“参考消息”样本

使用此示例来了解如何在 UNIX、Linux 和 Windows 上运行“参考消息”样本应用程序 AMQSPRM，或者如何在 IBM i 上运行 AMQSPRMA。 此示例显示“参考消息”如何放入队列中、由消息出口接收以及从队列中取出。

“参考消息”样本按如下方式运行：

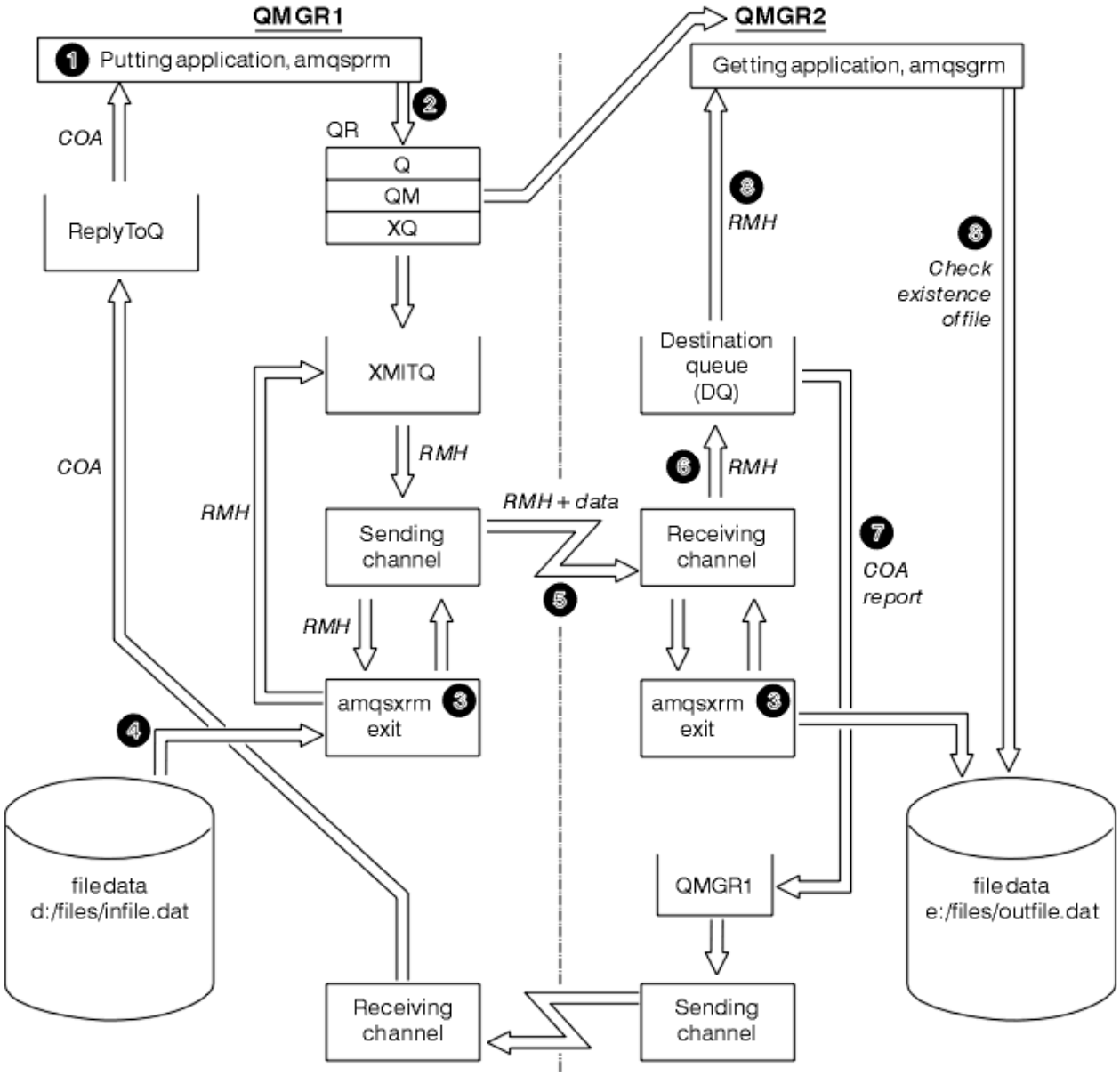


图 *136:* 运行“参考消息”样本

1. 设置环境以启动侦听器、通道和触发器监视器，并定义通道和队列。

   为了说明如何设置“参考消息”，此示例将发送机器指定为具有队列管理器 QMGR1 的 MACHINE1，将接收机器指定为具有队列管理器 QMGR2 的 MACHINE2。

   注: 以下定义允许构建参考消息，以便将对象类型为 FLATFILE 的文件从队列管理器 QMGR1 发送到 QMGR2，并按照 AMQSPRM（或者在 IBM i 上为 AMQSPRMA）调用中的定义来重新创建该文件。 使用通道 CHL1 和传输队列 XMITQ 来发送参考消息（包括文件数据），然后将该消息放入队列 DQ 中。 使用通道 REPORT 和传输队列 QMGR1 将异常和 COA 报告发回给 QMGR1。

   将使用启动队列 INITQ 和进程 PROC 来触发用于接收参考消息（AMQSGRM 或者 IBM i 上的 AMQSGRM）的应用程序。 确保正确设置 CONNAME 字段并且 MSGEXIT 字段能够基于机器类型和 IBM MQ 产品安装位置来反映目录结构。

   IBM i MQSC 定义已使用 AIX 样式定义了出口，因此如果在 IBM i 上使用 MQSC，那么需要相应地修改这些项。 值得注意的是，消息数据 FLATFILE 区分大小写，并且该样本仅在使用大写形式时才有效。

   在机器 MACHINE1 上，队列管理器 QMGR1

MQSC 语法

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i 命令语法

注: 如果未指定队列管理器名称，那么系统使用缺省队列管理器。

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
            REPLACE(*YES) TRPTYPE(*TCP) +
            CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
            MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ   QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
            REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
            MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ   QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
            REPLACE(*YES) RMTQNAME(DQ) +
            RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

在机器 MACHINE2 上，队列管理器 QMGR2

MQSC 语法

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i 命令语法

注: 在 IBM i 上，如果您没有指定队列管理器的名称，那么系统使用缺省队列管理器。

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
```

```
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. 在创建 IBM MQ 对象后：
   a. 在适用于平台的情况下，针对发送和接收队列管理器启动侦听器
   b. 启动通道 CHL1 和 REPORT
   c. 在接收队列管理器上，针对启动队列 INITQ 启动触发器监视器
3. 在命令行中，使用以下参数调用“放置参考消息”样本程序 AMQSPRM（在 IBM i 上为 AMQSPRMA）：

-m
本地队列管理器的名称；此选项缺省设置为缺省队列管理器

-i
源文件的名称和位置

-o
目标文件的名称和位置

-q
队列的名称

-g
-q 参数中所定义队列所在的队列管理器的名称。 此选项缺省设置为 -m 参数中指定的队列管理器。

-t
对象类型

-w
等待时间间隔，即等待来自接收队列管理器的异常和 COA 报告所用的时间

例如，要在该样本中使用先前定义的对象，应使用以下参数：

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

增加等待时间，使程序在消息超时前有足够时间通过网络发送大文件。

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

IBM i 用户： IBM i 在 IBM i 上，完成以下步骤：

a. 使用以下命令：

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

这假定原始文件 `rmsg1` 位于 IFS 目录 `/refmsgs` 中，并希望目标文件是目标系统的 IFS 目录 `/refmsgs` 中的 `rmsgx`。

b. 使用 CRTDIR 命令创建自己的目录，而不是使用根目录。
c. 在调用用于放置数据的程序时，请记住输出文件名需要反映 IFS 命名约定；例如，/TEST/FILENAME 将在目录 TEST 中创建名为 FILENAME 的文件。

注：

IBM i 在 IBM i 上，在指定参数时，可以使用正斜杠 (/) 或短划线 (-)。 例如：

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux UNIX 对于 UNIX and Linux 平台，必须使用两个反斜杠 (\\)（而不是一个）来表示目标文件目录。 因此，**amqsprm** 命令与下列命令类似：

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

运行“放置参考消息”程序将执行以下操作：

- 将参考消息放入队列管理器 QMGR1 上的队列 QR 中。
- 源文件和路径为 `d:\files\infile.dat` 并且位于发出示例命令的系统上。
- 如果队列 QR 是远程队列，那么会将参考消息发送到其他系统（在此系统上，将使用名称和路径 `e:\files\outfile.dat` 创建文件）上的另一个队列管理器。 此文件的内容与源文件相同。
- 对于来自目标队列管理器的 COA 报告，amqsprm 等待 30 秒。
- 对象类型为 `flatfile`，因此用于从队列 QR 移动消息的通道必须在 *MsgData* 字段中指定此项。

4. 在定义通道时，在接收端和发送端，将消息出口选为 amqsxrm。

   Windows 它在 Windows 上定义如下：

   ```
   msgexit(' pathname\amqsxrm.dll(MsgExit)')
   ```

   Solaris AIX 它在 AIX 和 Solaris 上定义如下：

   ```
   msgexit(' pathname/amqsxrm(MsgExit)')
   ```

   如果指定路径名，请指定完整名称。 如果省略了路径名，那么假定程序位于 `qm.ini` 文件中指定的路径（或在 IBM MQ for Windows 上，注册表中指定的路径）中。
5. 通道出口读取参考消息头并查找其引用的文件。
6. 在通过通道将该文件与头一起发送之前，通道出口会对该文件分段。

   Solaris AIX 在 AIX 和 Solaris 上，将目标目录的组所有者更改为“mqm”，从而使样本消息出口可在此目录中创建文件。 另外，更改目标目录的许可权以允许 mqm 组成员对其执行写操作。 该文件的数据不存储在 IBM MQ 队列中。
7. 在接收消息出口处理该文件的最后一个分段后，会将参考消息放入 amqsprm 指定的目标队列中。 如果触发此队列（也就是，定义指定 **Trigger**、**InitQ** 和 **Process** 队列属性），那么将触发目标队列的 PROC 参数指定的程序。 必须在 **Process** 属性的 `ApplId` 字段中定义要触发的程序。
8. 在参考消息到达目标队列 (DQ) 时，会将 COA 报告发回给“放置”应用程序 (amqsprm)。
9. “获取参考消息”样本 amqsgrm 将从输入触发器消息所指定的队列中获取消息，并检查该文件是否存在。

设计“放置参考消息”样本（*amqsprma.c* 和 *AMQSPRM4*）

本主题提供了“放置参考消息”样本的详细描述。

此样本将创建一条参考消息来引用文件并将该文件放入指定的队列中：

1. 该样本将使用 MQCONN 连接到本地队列管理器。
2. 随后打开 (MQOPEN) 用于接收报告消息的模型队列。
3. 该样本将构建一条包含移动文件时所需的值（例如，源和目标文件名以及对象类型）的参考消息。 例如，IBM MQ 随附的样本将构建一条参考消息以将文件 `d:\x\file.in` 从 QMGR1 发送到 QMGR2，并使用以下参数重新创建文件 `d:\y\file.out`：

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   其中，QR 是引用 QMGR2 上的目标队列的远程队列定义。

注: 对于 UNIX and Linux 平台，使用两个反斜杠 (\\)（而不是一个）来表示目标文件目录。 因此，**amqsprm** 命令与下列命令类似：

```
amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
```

4. 这会将参考消息放入 /q 参数指定的队列中（不含任何文件数据）。 如果这是远程队列，那么会将消息放入相应的传输队列中。
5. 对于 COA 报告，该样本将等待 /w 参数指定的时间段（缺省设置为 15 秒），此报告将与异常报告一起发回给本地队列管理器 (QMGR1) 上创建的动态队列。

设计“参考消息出口”样本（*amqsxrma.c* 和 *AMQSXRM4*）
此样本识别具有对象类型的参考消息，该对象类型与通道定义的消息出口用户数据字段中的对象类型相匹配。

对于这些消息，将执行以下操作：

- 在发送方或服务器通道上，会将指定长度的数据从指定文件的指定偏移量处复制到代理程序缓冲区中参考消息之后保留的空间。 如果未到达该文件末尾，那么在更新 *DataLogicalOffset* 字段后会将参考消息放回到传输队列中。
- 在请求方或接收方通道上，如果 *DataLogicalOffset* 字段为零并且指定的文件不存在，那么将创建此文件。 系统会将参考消息之后的数据添加到指定文件的末尾处。 如果参考消息不是指定文件的最后一项，那么将丢弃该消息。 否则，将其返回到通道出口以放入目标队列中，并且不带附加数据。

对于发送方和服务器通道，如果输入参考消息中的 *DataLogicalLength* 字段为零，那么会通过通道发送该文件的剩余部分（从 *DataLogicalOffset* 到文件末尾处）。 如果该字段不为零，那么仅发送指定长度的数据。

如果发生错误（例如，如果样本无法打开文件），MQCXP。 *ExitResponse* 设置为 MQXCC_SUPPRESS_FUNCTION，以便将正在处理的消息放入死信队列中，而不是继续发送到目标队列。 在 MQCXP 中返回一个反馈代码。 *Feedback* 并返回到应用程序，该应用程序将消息放到报告消息的消息描述符的 *Feedback* 字段中。 这是因为放置应用程序通过在 MQMD 的 *Report* 字段中设置 MQRO_EXCEPTION 来请求了异常报告。

如果参考消息的编码或 *CodedCharacterSetId* (CCSID) 与队列管理器的不同，那么会将该参考消息转换为本地编码和 CCSID。 在样本 amqsprm 中，对象格式为 MQFMT_STRING，因此在将数据写入文件之前，amqsxrm 在接收端将对象数据转换为本地 CCSID。

如果文件包含多字节字符（例如，DBCS 或 Unicode），那么请勿将要传输的文件格式指定为 MQFMT_STRING。 这是因为在发送端对文件进行分段时，可能会拆分多字节字符。 要传输并转换此类文件，请指定 MQFMT_STRING 以外的其他格式，从而使参考消息出口不转换该文件，而是在传输完成后在接收端转换该文件。

编译“参考消息出口”样本
要编译“参考消息出口”样本，请将命令用于安装 IBM MQ 的平台。

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

要编译 amqsxrma，请使用以下命令：

开启 AIX

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

开启 IBM i

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

注:

1. 要创建模块以便其使用 IFS 文件系统，请添加选项 SYSIFCOPT(*IFSIO)
2. 要创建程序以用于非线程通道，请使用以下命令：CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)
3. 要创建程序以用于线程通道，请使用以下命令：CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)

开启 Linux

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

开启 Solaris

Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -I  MQ_INSTALLATION_PATH/inc
-L  MQ_INSTALLATION_PATH/lib64 -R  MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm


-lsocket
-lnsl -ldl
```

开启 Windows

Windows IBM MQ 现在为 mqm 库提供客户机软件包以及服务器软件包，因此以下示例使用 mqm.lib 代替 mqmvx.lib:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

相关概念

第 877 页的『编写通道出口程序』
以下信息可帮助您编写通道出口程序。

设计“获取参考消息”样本（*amqsgrma.c* 和 *AMQSGRM4*）
本主题说明了如何设计“获取参考消息”样本。

程序逻辑如下:

1. 首先触发该样本，接着该样本从输入触发器消息中抽取队列名称和队列管理器名称。
2. 然后，它使用 MQCONN 连接到指定的队列管理器并使用 MQOPEN 打开指定的队列。
3. 该样本在循环中发出 MQGET（等待时间间隔为 15 秒）以从队列中获取消息。
4. 如果消息是参考消息，那么该样本将检查已传输的文件是否存在。
5. 随后，它关闭队列并断开与队列管理器的连接。

“请求”样本程序

“请求”样本程序说明了客户机/服务器处理。 这些样本是用于将请求消息放入由服务器程序处理的目标服务器队列的客户机。 它们会等待服务器程序将应答消息放入应答队列中。

“请求”样本使用 MQPUT 调用将一系列请求消息放入目标服务器队列中。 这些消息指定本地队列 SYSTEM.SAMPLE.REPLY 作为应答队列（这可以是本地队列或远程队列）。 这些程序将等待应答消息，然后显示这些消息。 仅在服务器应用程序处理目标服务器队列或者出于此目的（旨在触发“查询”、“设置”和“回传”样本程序）而触发应用程序时才会发送应答。 C 样本将等待 1 分钟（COBOL 样本将等待 5 分钟）以便第一个应答到达（提供足够时间以触发服务器应用程序），然后等待 15 秒以便后续应答到达，但两个样本都可在未收到任何应答的情况下结束。 请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解“请求”样本程序的名称。

运行“请求”样本程序

运行 amqsreq0.c、amqsreq 和 amqsreqc 样本

该程序的 C 版本采用以下三个参数：

1. 目标服务器队列名称（必需）
2. 队列管理器名称（可选）
3. 应答队列（可选）

例如，输入以下内容之一：

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

其中，`myqueue` 是目标服务器队列的名称，`qmanagername` 是拥有 `myqueue` 的队列管理器的名称，而 `replyqueue` 是应答队列的名称。

如果省略队列管理器的名称，那么假定缺省队列管理器拥有该队列。 如果省略应答队列的名称，那么将提供缺省应答队列。

运行 amq0req0.cbl 样本

COBOL 版本没有任何参数。 它连接到缺省队列管理器，并且在运行它时会出现以下提示：

```
Please enter the name of the target server queue
```

该程序采用来自 StdIn 的输入，并将每一行添加到目标管理器队列中，使每一行文本成为请求消息的内容。 在读到空行时，该程序将结束。

运行 AMQSREQ4 样本

C 程序通过接收来自 stdin（键盘）的数据（空白时终止输入）以创建消息。 该程序最多采用三个参数：目标队列名称（必需）、队列管理器名称（可选）和应答队列名称（可选）。 如果未指定队列管理器名称，那么将使用缺省队列管理器。 如果未指定应答队列，那么将使用 SYSTEM.SAMPLE.REPLY 队列。

以下示例说明了如何在指定了应答队列但保留缺省队列管理器的情况下调用 C 样本程序：

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

注: 请记住队列名称区分大小写。 样本文件创建程序 AMQSAMP4 所创建的所有队列的名称均采用大写形式。

运行 AMQ0REQ4 样本

COBOL 程序通过接受来自键盘的数据以创建消息。 要启动该程序，请在指定目标队列名称作为参数的情况下调用该程序。 该程序接受来自键盘的输入并将其放入缓冲区中，然后针对每行文本创建一条请求消息。 当键盘上输入空行时，该程序将停止。

运行使用触发机制的“请求”样本

如果将该样本与触发机制以及“查询”、“设置”或“回传”样本程序中的一个结合使用，那么输入行必须是您希望触发程序访问的队列的队列名称。

ULW 在 *UNIX, Linux, and Windows* 上使用触发运行“请求”样本

在 UNIX, Linux, and Windows 上，在一个会话中启动触发器监视器程序 RUNMQTRM，然后在另一个会话中启动 amqsreq 程序。

要运行使用触发机制的样本：

1. 在一个会话中启动触发器监视器程序 RUNMQTRM（启动队列 SYSTEM.SAMPLE.TRIGGER 可供使用）。
2. 在另一个会话中启动 amqsreq 程序。
3. 确保已定义目标服务器队列。

   可用作“请求”样本的目标服务器队列以用于存放消息的样本队列为：

   - SYSTEM.SAMPLE.INQ - 用于“查询”样本程序
   - SYSTEM.SAMPLE.SET - 用于“设置”样本程序
   - SYSTEM.SAMPLE.ECHO - 用于“回传”样本程序

   这些队列具有触发器类型 FIRST，因此，如果在运行“请求”样本之前这些队列中已存在消息，那么您发送的消息将不会触发服务器应用程序。
4. 确保已经为要使用的“查询”、“设置”或“回传”样本程序定义了队列。

这意味着在“请求”样本发送消息时触发器监视器已准备就绪。

注: 使用 RUNMQSC 和 amqscos0.tst 文件创建的样本进程定义将触发 C 样本。 更改 amqscos0.tst 中的进程定义，并将 RUNMQSC 与此更新文件结合使用以便使用 COBOL 版本。

第 1014 页的图 137 说明了如何结合使用“请求”和“查询”样本。

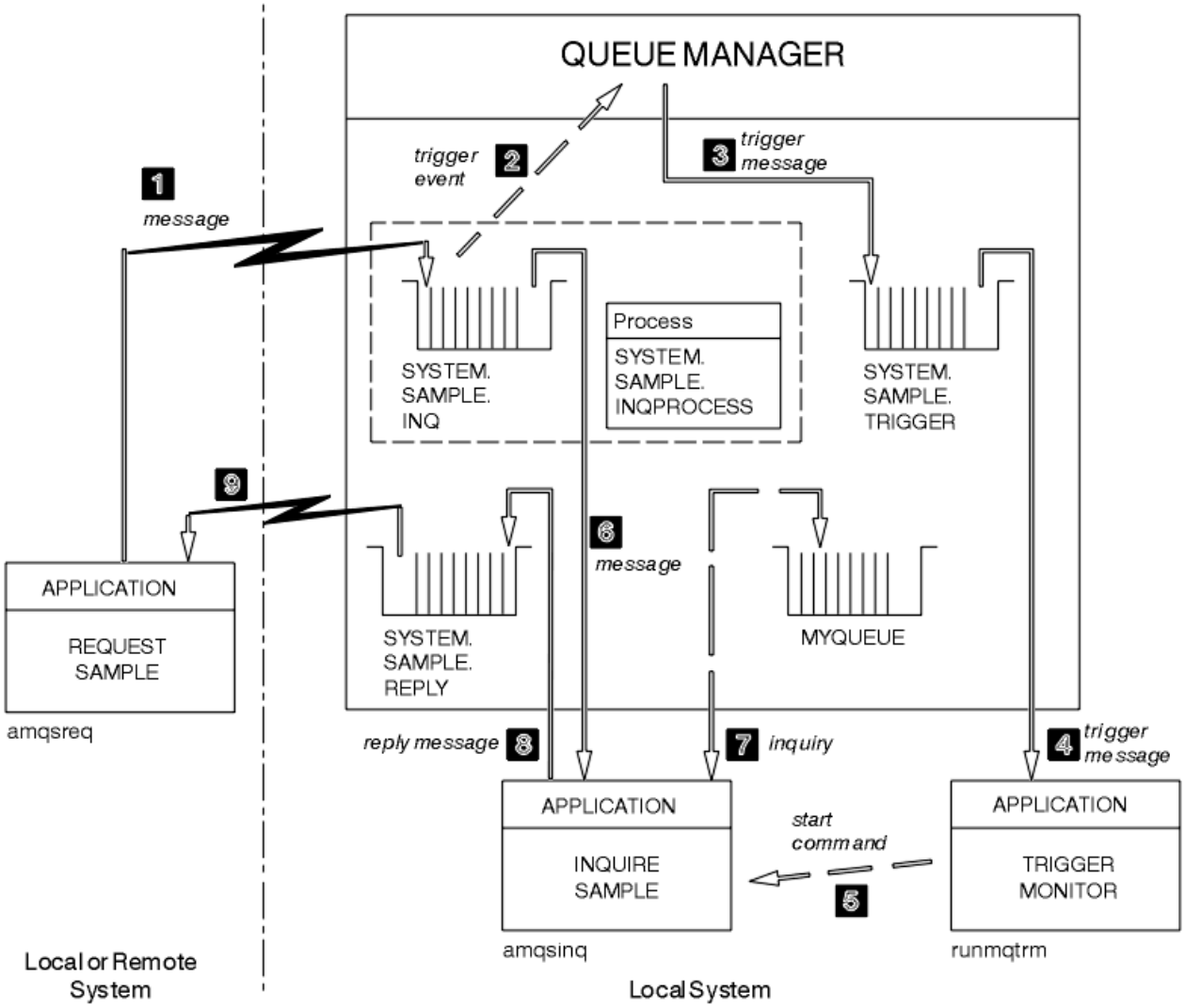


图 *137:* 使用触发机制的“请求”和“查询”样本

在第 1014 页的图 137 中，“请求”样本将消息放入目标服务器队列 SYSTEM.SAMPLE.INQ 中，而“查询”样本会查询队列 MYQUEUE。 或者，您可以使用在运行 amqscos0.tst 时定义的某个样本队列或者针对“查询”样本定义的任何其他队列。

注: 第 1014 页的图 137 中的数字表示事件序号。

要运行使用触发机制的“请求”和“查询”样本：

1. 检查是否定义了您想要使用的队列。 运行 amqscos0.tst 以定义样本队列，然后定义队列 MYQUEUE。
2. 运行触发器监视器命令 RUNMQTRM：

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. 运行“请求”样本

```
amqsreq SYSTEM.SAMPLE.INQ
```

注: 进程对象用于定义要触发的内容。 如果客户机和服务器不在同一平台上运行，那么任何由触发器监视器启动的进程必须定义 *ApplType*，否则服务器会采用其缺省定义（即，通常与服务器相关联的应用程序类型）并导致故障。

要获取应用程序类型的列表，请参阅 ApplType。

4. 输入您希望“查询”样本使用的队列的名称：

```
MYQUEUE
```

5. 输入空行（以结束“请求”程序）。
6. 然后，“请求”样本将显示一条消息，其中包含“查询”程序从 MYQUEUE 获取的数据。

您可以使用多个队列；在这种情况下，输入步骤 第 1015 页的『4』 中其他队列的名称。

有关触发机制的更多信息，请参阅第 795 页的『使用触发器启动 IBM MQ 应用程序』。

IBM i 在 *IBM i* 上使用触发运行“请求”样本

在 IBM i 上，请在一个作业中启动样本触发器服务器 AMQSERV4，然后在另一个作业中启动 AMQSREQ4。这意味着在“请求”样本程序发送消息时触发器服务器已准备就绪。

注：

1. AMQSAMP4 创建的样本定义将触发这些样本的 C 版本。 如果您想要触发 COBOL 版本，请更改进程定义 SYSTEM.SAMPLE.ECHOPROCESS、SYSTEM.SAMPLE.INQPROCESS 和 SYSTEM.SAMPLE.SETPROCESS。 您可以使用 CHGMQMPRC 命令（有关详细信息，请参阅更改 MQ 进程 (CHGMQMPRC)）来执行此操作，或者编辑并运行自己版本的 AMQSAMP4。
2. 仅针对 C 语言提供 AMQSERV4 的源代码。 但是，在库 QMQM 中提供可用于 COBOL 样本的已编译版本。

您可以将请求消息放入这些样本服务器队列中：

- SYSTEM.SAMPLE.ECHO（用于“回传”样本程序）
- SYSTEM.SAMPLE.INQ（用于“查询”样本程序）
- SYSTEM.SAMPLE.SET（用于“设置”样本程序）

第 1017 页的图 138 中显示了 SYSTEM.SAMPLE.ECHO 程序的流程图。 如果使用示例数据文件，那么用于向此服务器发出 C 程序请求的命令为：

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

注： 此样本队列具有触发器类型 FIRST，因此，如果在运行“请求”样本之前该队列中已存在消息，那么您发送的消息将不会触发服务器应用程序。

如果您想要尝试更多示例，那么可尝试以下变体：

- 使用 AMQSTRG4（或其命令行等效项 STRMQMTRM，有关详细信息，请参阅启动 MQ 触发器监视器 (STRMQMTRM)）代替 AMQSERV4 来提交作业，但作业提交可能出现延迟，因而导致不太容易关注到当前发生的情况。
- 运行 SYSTEM.SAMPLE.INQUIRE 和 SYSTEM.SAMPLE.SET 样本程序。 如果使用样本数据文件，那么用于向这些服务器发出 C 程序请求的命令分别为：

  ```
  CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
  CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
  ```

  这些样本队列也具有触发器类型 FIRST。

设计“请求”样本程序

该程序将打开目标服务器队列，以便在其中放入消息。 它使用具有 MQOO_OUTPUT 选项的 MQOPEN 调用。 如果它无法打开队列，那么程序将显示一条错误消息，其中包含 MQOPEN 调用返回的原因码。

然后，该程序打开名为 SYSTEM.SAMPLE.REPLY 的应答队列，以便可以获取应答消息。 因此，该程序使用具有 MQOO_INPUT_EXCLUSIVE 选项的 MQOPEN 调用。 如果无法打开该队列，那么该程序将显示包含 MQOPEN 调用返回的原因码的错误消息。

对于每行输入，该程序将文本读入缓冲区并使用 MQPUT 调用来创建包含此行文本的请求消息。 执行此调用时，该程序使用 MQRO_EXCEPTION_WITH_DATA 报告选项，以请求针对请求消息发送的任何报告消息都包含消息数据的前 100 个字节。 该程序继续运行，直至到达输入的结尾或者 MQPUT 调用失败。

然后，该程序使用 MQGET 调用从队列中除去应答消息，并显示应答中包含的数据。 MQGET 调用使用 MQGMO_WAIT、MQGMO_CONVERT 和 MQGMO_ACCEPT_TRUNCATED 选项。 对于第一个应答，在 COBOL 版本中，*WaitInterval* 为 5 分钟，而在 C 版本中为 1 分钟（提供足够时间以触发服务器应用程序），对于后续应答，该项为 15 秒。 如果队列中不存在消息，那么该程序将等待这些时间段。 如果在该时间间隔到期前没有消息到达该队列，那么该调用将失败并返回 MQRC_NO_MSG_AVAILABLE 原因码。 该调用还使用 MQGMO_ACCEPT_TRUNCATED_MSG 选项，因此将截断长度超过声明的缓冲区大小的消息。

该程序演示在每个 MQGET 调用之后如何清除 MQMD 结构的 *MsgId* 和 *CorrelId* 字段，因为该调用将这些字段设置为其检索的消息中包含的值。 清除这些字段意味着连续 MQGET 调用按照消息在队列中的存放顺序检索消息。

该程序将继续运行，直至 MQGET 调用返回 MQRC_NO_MSG_AVAILABLE 原因码或者 MQGET 调用失败。 如果该调用失败，那么该程序将显示包含原因码的错误消息。

然后，该程序使用 MQCLOSE 调用关闭目标服务器队列和应答队列。

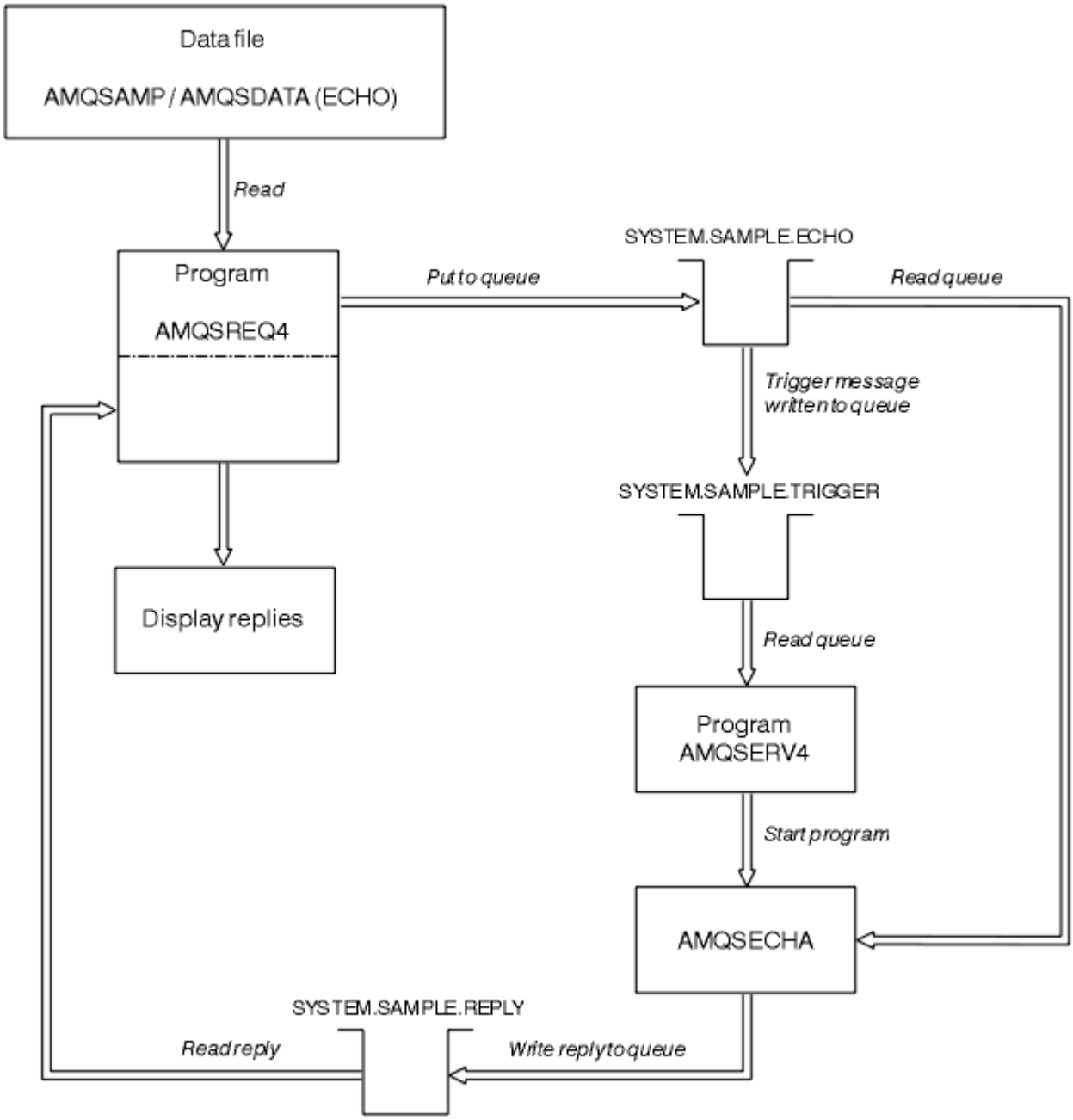


图 *138:* 样本 *IBM i* 客户机/服务器（回传）程序流程图

“设置”样本程序

“设置”样本程序使用 MQSET 调用来更改队列的 **InhibitPut** 属性，从而禁止队列中的放置操作。 此外，还介绍了如何设计“设置”样本程序。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

这些程序旨在作为触发程序运行，因此，这些程序的唯一输入是 MQTMC2（触发器消息）结构，其中包含具有待查询属性的目标队列的名称。 C 版本还使用队列管理器名称。 COBOL 版本使用缺省队列管理器。

要使触发进程正常运行，请确保到达队列 SYSTEM.SAMPLE.SET 的消息能触发您要使用的“设置”样本程序。 要执行此操作，请在进程定义 SYSTEM.SAMPLE.SETPROCESS 的 *ApplicId* 字段中指定您要使用的“设置”样本程序的名称。 样本队列具有触发器类型 FIRST；如果在运行“请求”样本之前队列中已存在消息，那么您发送的消息将不会触发“设置”样本。

在正确设置了该定义后，请执行以下操作：

- ULW 对于 UNIX, Linux, and Windows 系统，在一个会话中启动 **runmqtrm** 程序，然后在另一个会话中启动 amqsreq 程序。
- IBM i 对于 IBM i，在一个会话中启动 AMQSERV4 程序，然后在另一个会话中启动 AMQSREQ4 程序。 您可以使用 AMQSTRG4 代替 AMQSERV4，但作业提交可能出现延迟，因而导致不太容易关注到当前发生的情况。

使用“请求”样本程序来将请求消息（每条消息仅包含一个队列名称）发送到队列 SYSTEM.SAMPLE.SET。 对于每条请求消息，“设置”样本程序都会发送包含确认的应答消息，以确认已在指定的队列上禁止放置操作。 将向请求消息中指定的应答队列发送应答。

设计“设置”样本程序

该程序将打开在启动时传递的触发器消息结构中指定的队列。 （为清晰起见，我们将此队列称为请求队列。） 该程序将使用 MQOPEN 调用打开该队列以获取共享输入。

该程序将使用 MQGET 调用从此队列中除去消息。 此调用将使用 MQGMO_ACCEPT_TRUNCATED_MSG 和 MQGMO_WAIT 选项，并将等待时间间隔设置为 5 秒。 该程序将测试每条消息的描述符以判断其是否为请求消息；如果不是，那么该程序会丢弃该消息并显示一条警告消息。

对于从请求队列中除去的每条请求消息，该程序都会读取数据中包含的队列名称（称为目标队列），然后使用带有 MQOO_SET 选项的 MQOPEN 调用来打开该队列。 然后，该程序使用 MQSET 调用来将目标队列的 **InhibitPut** 属性值设置为 MQQA_PUT_INHIBITED。

如果 MQSET 调用成功，那么该程序使用 MQPUT1 调用来将应答消息放入应答队列中。 此消息包含字符串 `PUT inhibited`。

如果 MQOPEN 或 MQSET 调用不成功，那么该程序使用 MQPUT1 调用来将 `report` 消息放入应答队列中。 此报告消息的消息描述符的 *Feedback* 字段值是 MQOPEN 或 MQSET 调用（具体取决于失败的调用）返回的原因码。

在 MQSET 调用完成后，该程序使用 MQCLOSE 调用来关闭目标队列。

如果请求队列中未剩下任何消息，那么该程序将关闭此队列并断开与队列管理器的连接。

TLS 样本程序

AMQSSLC 是一个样本 C 程序，用于演示如何使用 MQCNO 和 MQSCO 结构在 MQCONNX 调用上提供 TLS 客户机连接信息。 这允许客户机 MQI 应用程序在运行时提供其客户机连接通道的定义和 TLS 设置，而无需客户机通道定义表 (CCDT)。

如果提供连接名称，那么该程序将采用 MQCD 结构构造客户机连接通道定义。

如果提供密钥存储库文件的主干名称，那么该程序将构造 MQSCO 结构；如果还提供了 OCSP 响应程序 URL，那么该程序将构造认证信息记录 MQAIR 结构。

然后，该程序使用 MQCONNX 连接到队列管理器。 它查询并打印其连接到的队列管理器的名称。

该程序旨在作为 MQI 客户机应用程序进行链接。 但它可以作为常规 MQI 应用程序进行链接。 然后，可简单连接到本地队列管理器，并忽略客户机连接信息

AMQSSLC 接受以下参数，所有这些参数都是可选的:

-m QmgrName
要连接到的队列管理器的名称

-c ChannelName
要使用的通道的名称

-x ConnName
服务器连接名称

TLS 参数:

-k KeyReposStem
密钥存储库文件的主干名称。 这是到该文件的完整路径，不含 .kdb 后缀。 例如:

```
/home/user/client
C:\User\client
```

-s CipherSpec
与队列管理器上 SVRCONN 通道定义中的 SSLCIPH 对应的 TLS 通道 CipherSpec 字符串。

-f
指定必须仅使用 FIPS 140-2 认证的算法。

-b VALUE1[,VALUE2...]
指定必须仅使用符合 Suite B 要求的算法。 此参数是以下一个或多个值的逗号分隔列表：NONE,128_BIT,192_BIT。 这些值具有与 MQSUITEB 环境变量以及客户机配置文件 SSL 节中等效 EncryptionPolicySuiteB 设置相同的含义。

-p Policy
指定要使用的证书验证策略。 这可以为以下值之一：

ANY
应用安全套接字库支持的所有证书验证策略并接受证书链（如果有任何策略认为该证书链有效）。 可使用此设置来确保与不符合最新证书标准的旧数字证书的最大向后兼容性。

RFC5280
仅应用符合 RFC 5280 的证书验证策略。 此设置提供比 ANY 设置更严格的验证，但是会拒绝一些较旧的数字证书。

缺省值为 ANY。

-l CertLabel
要用于安全连接的证书标签。

注: 您必须使用小写字符指定值。

OCSP 证书撤销参数：

-o URL
OCSP 响应程序 URL

运行 *TLS* 样本程序
要运行 TLS 样本程序，首先必须设置 TLS 环境。 随后，可通过提供多个参数来从命令行中运行该样本。

关于此任务
遵循以下指示信息来运行使用个人证书的样本程序。 例如，通过改变可使用的命令，可以使用 CA 证书并通过 OCSP 响应程序来检查其状态。 请参阅该样本中的指示信息。

过程
1. 创建名为 QM1 的队列管理器。 有关更多信息，请参阅 crtmqm。
2. 为此队列管理器创建密钥存储库。 有关更多信息，请参阅在 UNIX, Linux, and Windows 上设置密钥存储库。
3. 为客户机创建密钥存储库。 将其命名为 *clientkey.kdb*。
4. 为队列管理器创建个人证书。 有关更多信息，请参阅在 UNIX, Linux, and Windows 上创建自签名个人证书。
5. 为客户机创建个人证书。
6. 从服务器密钥存储库中抽取个人证书，并将其添加到客户机存储库。 有关更多信息，请参阅在 UNIX, Linux, and Windows 上的密钥存储库中抽取自签名证书的公共部分以及在 UNIX、Linux 或 Windows 系统上向密钥存储库添加 CA 证书（或自签名证书的公共部分）。
7. 从客户机密钥存储库中抽取个人证书，并将其添加到服务器密钥存储库。
8. 使用 MQSC 命令创建服务器连接通道：

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

有关更多信息，请参阅服务器连接通道

9. 在队列管理器上定义并启动通道侦听器。 有关更多信息，请参阅 DEFINE LISTENER 和 START LISTENER。
10. 使用以下命令运行该样本程序：

```
AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
-k "C:\Program Files\IBM\MQ\clientkey" -s TLS_RSA_WITH_AES_128_CBC_SHA256
-o http://dummy.OCSP.responder
```

结果

该样本程序将执行以下操作：

1. 使用指定的选项连接到指定的队列管理器或者缺省队列管理器。
2. 打开队列管理器并查询其名称。
3. 关闭队列管理器。
4. 断开与队列管理器的连接。

如果该样本程序运行成功，那么将显示与以下示例类似的输出：

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Program Files\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

如果该样本程序遇到问题，那么将显示相应的错误消息，例如，如果指定了无效的 OCSP 响应程序 URL，那么您会收到以下消息：

```
MQCONNX ended with reason code 2553
```

有关原因码的列表，请参阅 API 完成代码和原因码。

“触发”样本程序

“触发”样本中提供的函数是 **runmqtrm** 程序触发器监视器中提供的函数的子集。

请参阅第 965 页的『Multiplatforms 上的样本程序中演示的功能』，以了解这些程序的名称。

设计“触发”样本

“触发”样本程序使用带有 MQOO_INPUT_AS_Q_DEF 选项的 MQOPEN 调用来打开启动队列。 它使用带有 MQGMO_ACCEPT_TRUNCATED_MSG 和 MQGMO_WAIT 选项的 MQGET 调用（指定无限等待时间间隔）从启动队列中获取消息。 在每个 MQGET 调用按顺序获取消息之前，该程序会清除 *MsgId* 和 *CorrelId* 字段。

在从启动队列中检索消息时，该程序通过检查消息大小以确保其与 MQTM 结构大小相同，从而测试该消息。如果此测试失败，那么该程序将显示一条警告。

对于有效的触发器消息，“触发”样本将从以下字段复制数据：*ApplicId*、*EnvrData*、*Version* 和 *ApplType*。 其中最后两个字段是数字字段，因此，该程序将创建要在 IBM i、 UNIX, Linux, and Windows 系统的 MQTMC2 结构中使用的字符替换项。

“触发”样本将针对触发器消息的 *ApplicId* 字段中指定的应用程序发出启动命令，然后传递 MQTMC2 或 MQTMC（触发器消息的字符版本）结构。

- ULW 在 UNIX, Linux, and Windows 系统中，*EnvData* 字段用作调用命令字符串的扩展。
- IBM i 在 IBM i 中，它用作作业提交参数（例如，作业优先级或作业描述）。

最后，该程序将关闭启动队列。

结束 IBM i 上的“触发”样本程序

IBM i

可通过系统请求选项 2 (ENDRQS) 或禁止从触发器队列获取来结束触发器监视器程序。

如果使用样本触发器队列，那么命令为：

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

要点: 在此队列中重新启动触发之前，必须输入以下命令：

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

运行“触发”样本程序
本主题介绍了如何运行“触发”样本程序。

运行 amqstrg0.c、amqstrg 和 amqstrgc 样本

该程序采用以下 2 个参数：

1. 启动队列名称（必需）
2. 队列管理器名称（可选）

如果未指定队列管理器，那么它连接到缺省队列管理器。 在运行 amqscos0.tst 时将定义样本启动队列；此队列的名称为 SYSTEM.SAMPLE.TRIGGER，并且可在运行此程序时使用。

注: 该样本中的函数是 runmqtrm 程序中提供的完全触发函数的子集。

运行 AMQSTRG4 样本

IBM i

这是 IBM i 环境的触发器监视器。 它针对每个要启动的应用程序提交一个 IBM i 作业。 这意味着每个触发器消息都会进行相关的额外处理。

AMQSTRG4（在 QCSRC 中）采用以下两个参数：要服务的启动队列名称，以及队列管理器名称（可选）。 AMQSAMP4（在 QCLSRC 中）定义一个样本启动队列 SYSTEM.SAMPLE.TRIGGER，您可在尝试样本程序时使用此队列。

如果使用该示例触发器队列，那么要发出的命令为：

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

也可以使用 CL 等效项 STRMQMTRM；有关详细信息，请参阅启动 MQ 触发器监视器 (STRMQMTRM)。

运行 AMQSERV4 样本

IBM i

这是 IBM i 环境的触发器服务器。 对于每条触发器消息，此服务器在自己的作业中运行启动命令以启动指定的应用程序。 触发器服务器可调用 CICS 事务。

AMQSERV4 采用以下两个参数：要服务的启动队列名称，以及队列管理器名称（可选）。 AMQSAMP4 定义一个样本启动队列 SYSTEM.SAMPLE.TRIGGER，您可在尝试样本程序时使用此队列。

如果使用示例触发器队列，那么要发出的命令为：

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

触发器服务器的设计
触发器服务器的设计类似于触发器监视器的设计，但存在一些例外情况

触发器服务器的设计类似于触发器监视器的设计，但触发器服务器在以下方面有所不同：

- 允许 MQAT_CICS 以及 MQAT_OS400 应用程序。
- IBM i 在自己的作业中调用 IBM i 应用程序（或者使用 STRCICSUSR 启动 CICS 应用程序），而不是提交 IBM i 作业。
- 例如，对于 CICS 应用程序，替换 *EnvData* 以指定来自 STRCICSUSR 命令中触发器消息的 CICS 区域。
- 打开启动队列以获取共享输入，以便可以同时运行多个触发器服务器。

注: AMQSERV4 启动的程序不得使用 MQDISC 调用，因为这会停止触发器服务器。 如果 AMQSERV4 启动的程序使用 MQCONN 调用，那么它们将收到 MQRC_ALREADY_CONNECTED 原因码。

Windows UNIX 在 *UNIX* 和 *Windows* 上使用 *TUXEDO* 样本

下面介绍了 TUXEDO 的“放置”和“获取”样本程序，还介绍了如何在 TUXEDO 中构建服务器环境。

开始之前

在运行这些样本之前，必须构建服务器环境。

关于此任务

注: 在本部分中，使用反斜杠 (\) 字符来拆分超过一行的长命令。 请勿输入此字符。 在单独一行中输入每个命令。

Windows UNIX 构建服务器环境

下面介绍了如何在不同平台上为 IBM MQ 构建服务器环境。

开始之前

假定您拥有正常运行的 TUXEDO 环境。

AIX 构建 *AIX*（*32* 位）的服务器环境

如何构建 IBM MQ for AIX（32 位）的服务器环境。

过程

1. 创建构建服务器环境的目录（例如，APPDIR），并在此目录执行所有命令。
2. 导出以下环境变量，其中 TUXDIR 是 TUXEDO 的根目录，*MQ_INSTALLATION_PATH* 表示安装 IBM MQ 的高级目录：

   ```
   $ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
   $ export LDOPTS="-lmqm"
   $ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
   $ export VIEWFILES=/APPDIR/amqstxvx.V
   $ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
   ```

3. 将以下行添加到 TUXEDO 文件 udataobj/RM：

   ```
   MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
   ```

4. 运行命令：

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. 编辑 ubbstxcx.cfg 并在必要时添加机器名称、工作目录和队列管理器的详细信息：

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. 创建 TLOGDEVICE：

```
$tmadmin -c
```

将显示一个提示。 在显示提示时，输入：

```
> crdl -z /APPDIR/TLOG1
```

7. 启动队列管理器：

```
$ strmqm
```

8. 启动 Tuxedo：

```
$ tmboot -y
```

下一步做什么

您现在可以使用 doputs 和 dogets 程序将消息放到队列，以及从队列检索消息。

AIX 构建 *AIX*（*64* 位）的服务器环境
如何构建 IBM MQ for AIX（64 位）的服务器环境。

过程

1. 创建构建服务器环境的目录（例如，APPDIR），并在此目录执行所有命令。
2. 导出以下环境变量，其中 TUXDIR 表示 TUXEDO 的根目录，*MQ_INSTALLATION_PATH* 表示安装 IBM MQ 的高级目录：

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. 将以下行添加到 TUXEDO 文件 udataobj/RM：

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. 运行命令：

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. 编辑 ubbstxcx.cfg 并在必要时添加机器名称、工作目录和队列管理器的详细信息：

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. 创建 TLOGDEVICE：

```
$tmadmin -c
```

将显示一个提示。 在显示提示时，输入：

```
> crdl -z /APPDIR/TLOG1
```

7. 启动队列管理器：

```
$ strmqm
```

8. 启动 Tuxedo：

```
$ tmboot -y
```

下一步做什么

您现在可以使用 doputs 和 dogets 程序将消息放到队列，以及从队列检索消息。

Solaris 构建 *Solaris*（*32* 位）的服务器环境

如何构建 IBM MQ for Solaris（32 位）的服务器环境。

关于此任务

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

过程

1. 创建用于构建服务器环境的目录（例如，APPDIR），并执行此目录中的所有命令。
2. 导出以下环境变量，其中 TUXDIR 是 TUXEDO 的根目录：

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. 将以下内容添加到 TUXEDO 文件 udataobj/RM（RM 必须包含 *MQ_INSTALLATION_PATH*/lib/libmqmcs 和 *MQ_INSTALLATION_PATH*/lib/libmqmzse）。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a MQ_INSTALLATION_PATH/lib/libmqmcs.so \
MQ_INSTALLATION_PATH/lib/libmqmzse.so
```

4. 运行命令：

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
```

5. 编辑 ubbstxcx.cfg 并在必要时添加机器名称、工作目录和队列管理器的详细信息：

```
$ tmloadcf  -y ubbstxcx.cfg
```

6. 创建 TLOGDEVICE：

```
$tmadmin -c
```

将显示一个提示。 在显示提示时，输入：

```
> crdl -z /APPDIR/TLOG1
```

7. 启动队列管理器：

```
$ strmqm
```

8. 启动 Tuxedo：

```
$ tmboot -y
```

下一步做什么

您现在可以使用 doputs 和 dogets 程序将消息放到队列，以及从队列检索消息。

Solaris 构建 *Solaris*（*64* 位）的服务器环境
如何构建 IBM MQ for Solaris（64 位）的服务器环境。

关于此任务

MQ_INSTALLATION_PATH 表示 IBM MQ 安装所在的高级目录。

过程

1. 创建用于构建服务器环境的目录（例如，APPDIR），并执行此目录中的所有命令。
2. 导出以下环境变量，其中 TUXDIR 是 TUXEDO 的根目录：

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib64
$ export LD_LIBRARY_PATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. 将以下内容添加到 TUXEDO 文件 udataobj/RM（RM 必须包含 *MQ_INSTALLATION_PATH*/lib/libmqmcs 和 *MQ_INSTALLATION_PATH*/lib/libmqmzse）。

```
MQSERIES_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a MQ_INSTALLATION_PATH/lib64/libmqmcs.so \
MQ_INSTALLATION_PATH/lib64/libmqmzse.so
```

4. 运行命令：

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
```

5. 编辑 ubbstxcx.cfg 并在必要时添加机器名称、工作目录和队列管理器的详细信息：

```
$ tmloadcf  -y ubbstxcx.cfg
```

6. 创建 TLOGDEVICE：

```
$tmadmin -c
```

将显示一个提示。 在显示提示时，输入：

```
> crdl -z /APPDIR/TLOG1
```

7. 启动队列管理器：

```
$ strmqm
```

8. 启动 Tuxedo：

```
$ tmboot -y
```

下一步做什么

您现在可以使用 doputs 和 dogets 程序将消息放到队列，以及从队列检索消息。

Windows 构建 *Windows*（*32* 位）的服务器环境
构建 IBM MQ for Windows（32 位）的服务器环境。

关于此任务

注: 将以下标识为 *VARIABLES* 的字段更改为目录路径：

表 *171:* 要更改到目录路径的字段

字段	目录路径
MQMDIR	安装 IBM MQ 时指定的目录路径，例如 `g:\Program Files\IBM\MQ`。
TUXDIR	安装 TUXEDO 时指定的目录路径，例如 `f:\tuxedo`。
APPDIR	要用于样本应用程序的目录路径，例如 `f:\tuxedo\apps\mqapp`。

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

图 *139: IBM MQ for Windows* 的 *ubbstxcn.cfg* 文件示例

注: 更改机器名 *MachineName* 和目录路径以匹配您的安装。 同时还将队列管理器名称 *MYQUEUEMANAGER* 更改为您要连接的队列管理器的名称。

IBM MQ for Windows 的样本 `ubbconfig` 文件列示在 第 1027 页的图 139 中。 它在 IBM MQ 样本目录中提供为 `ubbstxcn.cfg`。

为 IBM MQ for Windows 提供的样本 Makefile（请参阅 第 1028 页的图 140）名为 ubbstxmn.mak，并保存在 IBM MQ 样本目录中。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

图 *140: IBM MQ for Windows* 的 *TUXEDO* 样本 *makefile*

要构建服务器环境和样本，请完成以下步骤。

过程

1. 创建构建样本应用程序的应用程序目录，例如：

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 将以下样本文件从 IBM MQ 样本目录复制到应用程序目录：
   - amqstxmn.mak
   - amqstxen.env
   - ubbstxcn.cfg
3. 编辑其中每一个文件，以设置安装使用的目录名称和目录路径。
4. 编辑 ubbstxcn.cfg（请参阅 第 1027 页的图 139）以添加要连接到的机器名和队列管理器的详细信息。
5. 将以下行添加到 TUXEDO 文件 *TUXDIR*udataobj\rm：

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   新条目在文件中必须为一行。
6. 设置以下环境变量：

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. 为 TUXEDO 创建 TLOG 设备。

要执行此操作，调用 `tmadmin -c`，并输入以下命令：

```
crdl -z APPDIR\TLOG
```

8. 将当前目录设置为 *APPDIR*，并将样本 Makefile `amqstxmn.mak` 作为外部项目 Makefile 调用。 例如，对于 Microsoft Visual C++，请发出以下命令：

```
msvc amqstxmn.mak
```

选择 **build** 构建所有样本程序。

Windows 构建 *Windows*（*64* 位）的服务器环境

如何构建 IBM MQ for Windows（64 位）的服务器环境。

关于此任务

注: 将以下标识为 *VARIABLES* 的字段更改为目录路径：

表 *172:* 要更改到目录路径的字段

字段	**目录路径**
MQMDIR	安装 IBM MQ 时指定的目录路径，例如 `g:\Program Files\IBM\MQ`。
TUXDIR	安装 TUXEDO 时指定的目录路径，例如 `f:\tuxedo`。
APPDIR	要用于样本应用程序的目录路径，例如 `f:\tuxedo\apps\mqapp`。

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

图 *141: IBM MQ for Windows* 的 *ubbstxcn.cfg* 文件示例

注: 更改机器名 *MachineName* 和目录路径以匹配您的安装。 同时还将队列管理器名称 *MYQUEUEMANAGER* 更改为您要连接的队列管理器的名称。

IBM MQ for Windows 的样本 ubbconfig 文件在第 1030 页的图 141 中列示。 它位于 IBM MQ 样本目录中，显示为 `ubbstxcn.cfg`。

针对 IBM MQ for Windows 提供的样本 Makefile（请参阅 第 1031 页的图 142）称为 `ubbstxmn.mak`，并保存在 IBM MQ 样本目录中。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

图 *142: IBM MQ for Windows* 的 *TUXEDO* 样本 *makefile*

要构建服务器环境和样本，请完成以下步骤。

过程

1. 创建构建样本应用程序的应用程序目录，例如：

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 将以下样本文件从 IBM MQ 样本目录复制到应用程序目录：
   - amqstxmn.mak
   - amqstxen.env
   - ubbstxcn.cfg
3. 编辑其中每一个文件，以设置安装使用的目录名称和目录路径。
4. 编辑 ubbstxcn.cfg（请参阅 第 1030 页的图 141）以添加要连接到的机器名和队列管理器的详细信息。
5. 将以下行添加到 TUXEDO 文件 *TUXDIR*udataobj\rm

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   新条目在文件中必须为一行。
6. 设置以下环境变量：

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. 为 TUXEDO 创建 TLOG 设备。 要执行此操作，调用 tmadmin -c 并输入命令：

   ```
   crdl -z APPDIR\TLOG
   ```

8. 将当前目录设置为 *APPDIR*，并将样本 Makefile amqstxmn.mak 作为外部项目 Makefile 调用。 例如，对于 Microsoft Visual C++，请发出以下命令：

```
msvc amqstxmn.mak
```

选择 **build** 构建所有样本程序。

Windows UNIX *TUXEDO* 的样本服务器程序

样本服务器程序 (amqstxsx) 旨在与 PUT (amqstxpx.c) 和 GET (amqstxgx.c) 样本程序一起运行。 在启动 TUXEDO 时，样本服务器程序将自动运行。

注: 在启动 TUXEDO 之前，您必须启动队列管理器。

样本服务器提供两项 TUXEDO 服务 MPUT1 和 MGET1：

- MPUT1 服务由 PUT 样本驱动，并且在同步点使用 MQPUT1 将消息放置在 TUXEDO 控制的工作单元中。它采用参数 QName 和消息文本，这些均由 PUT 样本提供。
- MGET1 服务每次获取消息时打开和关闭队列。 它采用参数 QName 和消息文本，这些均由 GET 样本提供。

任何错误消息、原因代码和状态消息都写入 TUXEDO 日志文件。

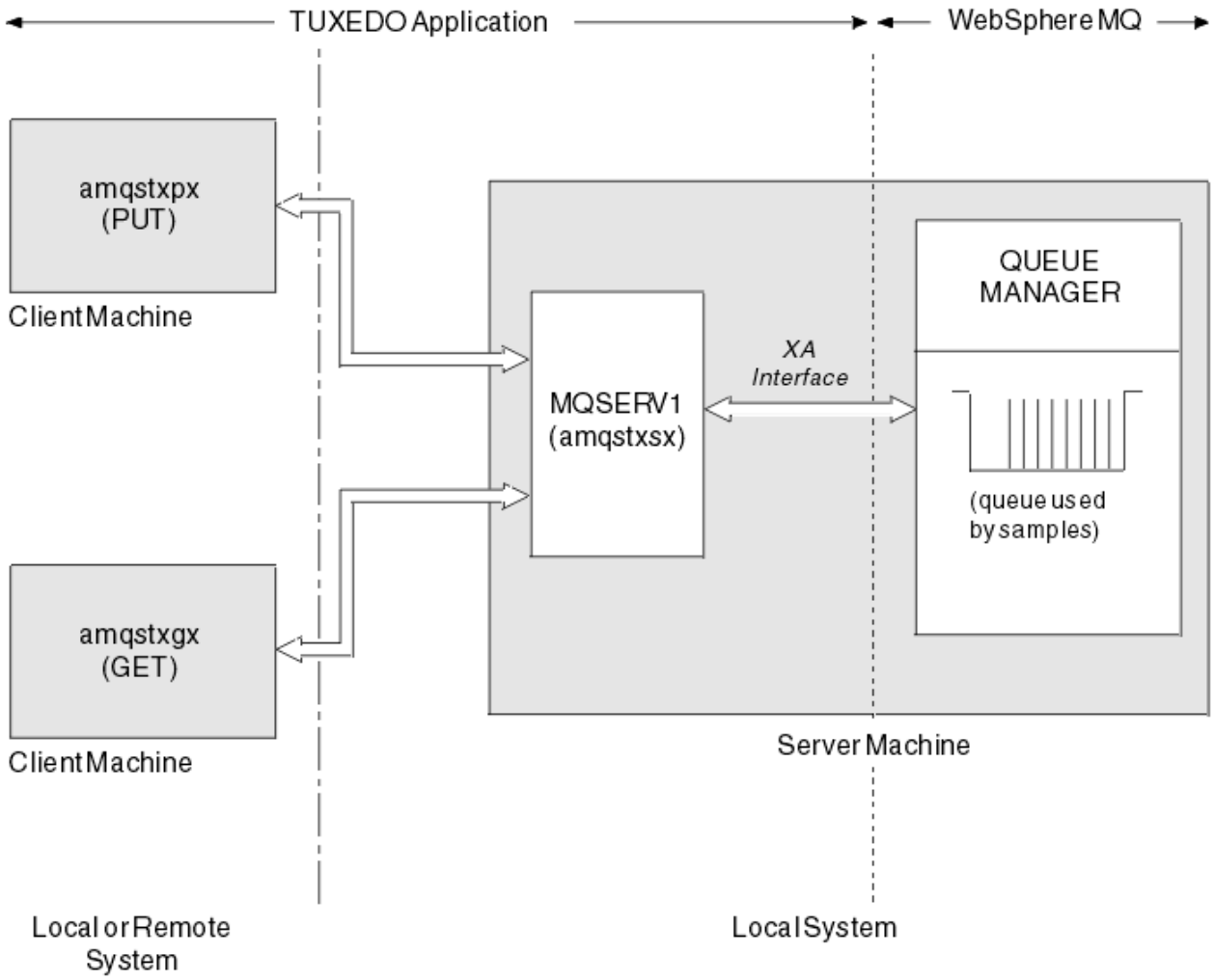


图 *143: TUXEDO* 样本如何协同工作

Windows UNIX *TUXEDO* 的 *PUT* 样本程序

此样本允许您将消息多次分批放入队列中，其演示了使用 TUXEDO 作为资源管理器的同步点。

要让 PUT 样本成功，必须运行样本服务器程序 `amqstxsx`；服务器样本程序连接到队列管理器并使用 XA 接口。 要运行样本，请输入：

- `doputs -n queuename -b batchsize -c trancount -t message`

例如：

- `doputs -n myqueue -b 5 -c 6 -t “Hello World”`

这样将 30 条消息放到名为 myqueue 的队列中，分六批，每批 5 条消息。 如果有任何问题，它返回一批消息，否则提交消息。

任何错误消息将写入 TUXEDO 日志文件和 stderr。 任何原因代码将写入 stderr。

Windows UNIX *TUXEDO* 的 *GET* 样本

此样本允许您从队列中成批获取消息。

要让 Get 样本成功，必须运行样本服务器程序 amqstxsx；样本服务器程序连接到队列管理器并使用 XA 接口。 要运行样本，请输入以下命令：

- `dogets -n queuename -b batchsize -c trancount`

例如：

- `dogets -n myqueue -b 6 -c 4`

这样将 24 条消息从名为 myqueue 的队列撤回，分六批，每批 4 条消息。 如果在 put 示例后运行此命令，PUT 示例将 30 条消息放在 myqueue 上，那么您在 myqueue 上只有 6 条消息。 PUT 和 GET 消息之间的批数量和批大小可能有所不同。

任何错误消息将写入 TUXEDO 日志文件和 stderr。 任何原因代码将写入 stderr。

Windows 在 *Windows* 上使用 *SSPI* 安全出口

本主题描述如何在 Windows 系统上使用 SSPI 通道出口程序。 提供的出口代码有两种格式：对象和源。

对象代码

对象代码文件称为 amqrspin.dll。 对于客户机和服务器，它将作为 IBM MQ for Windows 的标准部件安装在 *MQ_INSTALLATION_PATH*\exits*INSTALLATION_NAME* 文件夹中。 例如，C:\Program Files\IBM\MQ\exits\installation2。 它将作为标准用户出口装入。 您可以运行提供的安全通道出口，并在通道定义中使用认证服务。

要完成此任务，指定以下任一项：

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

要为受限通道提供支持，请在 SVRCONN 通道上指定以下内容：

```
SCYDATA('remote_principal_name')
```

其中 *remote_principal_name* 位于表单 *DOMAIN**user* 中。 只有远程主体名称与 *remote_principal_name* 相符时，才建立安全通道。

要在 Kerberos 安全域内运行的系统之间使用所提供的通道出口程序，请为队列管理器创建 **servicePrincipalName**。

源代码

出口源代码文件称为 amqsspin.c。 它位于 C:\Program Files\IBM\MQ\Tools\c\Samples 中。

如果修改源代码，必须重新编译修改过的源。

针对相关平台，采用与任何其他通道出口相同的方式对其进行编译并链接，不同之处是在编译时需要访问 SSPI 头，在链接时需要访问 SSPI 安全库和任何建议的关联库。

在执行以下命令之前，请确保 cl.exe、Visual C++ 库和 include 文件夹都位于您的路径中。 例如：

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

注: 源代码不提供任何跟踪或错误处理。 如果您修改和使用源代码，添加自己的追踪和错误处理例程。

使用远程队列运行样本

您可以通过在已连接的队列管理器上运行样本来演示远程排队。

程序 amqscos0.tst 提供远程队列 (SYSTEM.SAMPLE.REMOTE) 的本地定义，该远程队列使用名称为 OTHER 的远程队列管理器。 要使用此样本定义，将 OTHER 更改为您要使用的第二个队列管理器的名称。 您还必须在两个队列管理器之间设置消息通道；有关如何执行此操作的信息，请参阅定义通道。

请求样本程序将他们自己的本地队列管理器名称放在他们发送的消息的 *ReplyToQMgr* 字段中。 Inquire 和 Set 样本将应答消息发送给它们处理的请求消息的 *ReplyToQ* 和 *ReplyToQMgr* 字段中指定的队列和消息队列管理器。

集群队列监视样本程序 *(AMQSCLM)*

此样本使用内置 IBM MQ 集群工作负载均衡功能将消息定向到连接了使用应用程序的队列实例。 这种自动定向可防止消息堆积在未连接使用应用程序的集群队列实例上。

概述

您可以设置一个集群，使其对不同队列管理器上的相同队列具有多个定义。 此配置的好处是能够增加可用性和工作负载均衡。 但是 IBM MQ 没有内置根据连接的应用程序的状态跨集群动态修改消息分发的功能。 鉴于此，使用应用程序必须始终连接到队列的每个实例，以便确保消息被处理。

集群队列监视样本程序监视连接的应用程序的状态。 该程序动态调整内置工作负载均衡配置，以将消息定向到连接的使用应用程序的集群队列实例。 在特定情况下，此程序可让使用应用程序无需始终与队列的每个实例相连接。 并且还将重新发送在未连接使用应用程序的队列实例上排队的消息。 重新发送消息可以让消息绕过临时关闭的使用应用程序进行路由。

该程序在使用应用程序长期运行而不是频繁连接和断开应用程序时使用。

集群队列监视样本程序是 C 样本文件 amqsclma.c 的已编译的可执行程序。

有关集群和工作负载的更多信息可以在使用工作负载管理的集群中找到

AMQSCLM：设计和规划使用样本
有关集群队列监视样本程序如何运作的信息，为了运行样本程序设置系统时需要考虑的几个要点，以及可以对样本源代码进行的修改。

设计

集群队列监视样本程序监视连接使用应用程序的本地集群队列。 程序监视用户指定的队列。 队列的名称可能是具体名称，例如 APP.TEST01，也可能是通用名称。 通用名称必须是符合 PCF（可编程命令格式）的格式。 例如，通用名称 APP.TEST* 或 APP*。

拥有要监视的本地队列实例的集群中的每一个队列管理器都需要连接到集群队列监视样本程序的一个实例。

动态消息路由

集群队列监视样本程序使用队列的 **IPPROCS**（为输入过程计数打开）值确定队列是否有任何使用者。 大于 0 的值表示队列至少连接了一个使用应用程序。 此类队列处于活动状态。 值 0 指示队列没有连接任何使用程序。 此类队列处于非活动状态。

对于集群中具有多个实例的集群队列，IBM MQ 使用每个队列实例的集群工作负载优先级属性 **CLWLPRTY** 来确定将消息发送到哪个实例。 IBM MQ 将消息发送到 **CLWLPRTY** 值最高的可用队列实例。

集群队列监视样本程序通过将本地 **CLWLPRTY** 值设置为 1 来激活集群队列。 该程序通过将集群队列的 **CLWLPRTY** 值设置为 0 来取消激活该集群队列。

IBM MQ 集群技术将集群队列的已更新 **CLWLPRTY** 属性传播到集群中所有相关队列管理器。 例如，

- 其连接的应用程序将消息放入队列的队列管理器。
- 在同一集群中拥有同一名称本地队列的队列管理器。

传播使用集群的完整存储库队列管理器进行。 集群队列的新消息引导至集群中 **CLWLPRTY** 值最高的实例。

队列消息传输

CLWLPRTY 值的动态修改影响新消息的路由。 此动态修改不影响已经在没有连接使用者的队列实例中排队的消息，或者在修改过的 **CLWLPRTY** 值传播到整个集群之前已经通过工作负载均衡机制的消息。 结果，消息保留在任何非活动队列，没有经使用应用程序处理。 为解决这个问题，集群队列监视样本程序可以从没有使用者的本地队列获取消息，然后将这些消息发送到连接使用者的同一个队列的远程实例。

集群队列监视样本程序通过获取消息（使用 **MQGET**）并放入消息（使用 **MQPUT**）至同一个集群队列，将非活动本地队列的消息传输到一个或多个活动远程队列。 此传输导致 IBM MQ 集群工作负载管理根据高于本地队列实例的 CLWLPRTY 值选择不同的目标实例。 在消息传输期间保存消息持久性和上下文。 不保存消息顺序和任何绑定选项。

规划

在使用应用程序的连接有所变更时，集群队列监视样本程序将修改集群配置。 修改从集群队列监视样本程序监视队列的队列管理器传送到集群中的完整存储库队列管理器。 完整存储库队列管理器处理配置更新，并将其重新发送至集群中所有相关队列管理器。 相关队列管理器包括那些拥有同一名称的集群队列的队列管理器（运行集群队列监视样本程序实例的队列管理器） 和应用程序在最近 30 天内打开集群队列放入消息的任何队列管理器。

更改跨集群异步处理。 因此，在每次更改之后，集群内的不同队列管理器在一段时间内可能有不同的配置视图。

集群队列监视样本程序仅适合使用应用程序偶尔连接或断开连接的系统；例如，长期运行的使用应用程序。 如果用于监视使用应用程序仅短时间连接的系统，分发配置更新时产生的延迟将导致集群中的队列管理器具有错误的队列（连接使用者的队列）视图。 此延迟可能导致消息路由错误。

监视许多队列时，所有队列中连接的使用者的更改速度相对较低，可能增加整个集群的集群配置流量。 集群配置流量增加，将导致以下一个或多个队列管理器过载。

- 运行集群队列监视样本程序的队列管理器
- 完整存储库队列管理器
- 所连接的应用程序将消息放入队列的队列管理器
- 在同一集群中拥有同一名称本地队列的队列管理器

必须评估完整存储库队列管理器上处理器使用情况。 其他处理器使用情况作为消息流量在完整存储库队列 SYSTEM.CLUSTER.COMMAND.QUEUE 上是可视的。 如果该队列消息堆积，那么表示完整存储库队列管理器无法跟上系统中集群配置更改的速度。

集群队列监视样本程序监视许多队列时，有一些工作由样本程序和队列管理器执行。 即使所连接的使用者没有更改，也执行此工作。 可以修改 **-i** 参数，通过减少监视循环的频率，来降低本地系统上样本程序的处理器使用率。

为了帮助检测过量活动，集群队列监视样本程序报告每个轮询时间间隔的平均处理时间、花费的处理时间和配置更改次数。 报告以参考消息 **CLM0045I** 形式提供，频率为每 30 分钟或每 600 个轮询时间间隔中较快者。

集群队列监视使用要求

集群队列监视样本程序具有要求和限制。 可以通过修改提供的样本源代码来更改如何使用它的一些限制。 此部分列出的示例详细描述了可进行哪些修改。

- 集群队列监视样本程序旨在用于监视连接或未连接使用应用程序的队列。 如果系统的使用应用程序经常连接和断开连接，那么样本程序可能在整个集群内生成过量的集群配置活动。 这可能对集群中队列管理器的性能有影响。

- 集群队列监视样本程序依赖底层的 IBM MQ 系统和集群技术。 监视的队列数量、监视频率和每个队列状态更改的频率都影响整个系统上的负载。 选择要监视的队列和监视的轮询时间间隔时，必须考虑这些因素。
- 集群队列监视样本程序的实例必须连接到集群中拥有受监视队列实例的每个队列管理器。 不需要将样本程序连接到集群中没有队列的队列管理器。
- 必须使用合适的权限运行集群队列监视样本程序，以访问所需的所有 IBM MQ 资源。 例如，
  - 要连接的队列管理器
  - SYSTEM.ADMIN.COMMAND.QUEUE
  - 执行消息传输时要监视的所有队列
- 必须针对连接了集群队列监视样本程序的每个队列管理器运行命令服务器。
- 集群队列监视样本程序的每个实例需要独占使用其所连接的队列管理器上的本地（非集群）队列。 此本地队列用于控制样本程序，并且接收对队列管理器的命令服务器进行查询的应答消息。
- 集群队列监视样本程序的单个实例监视的所有队列必须位于同一集群中。 如果队列管理器在需要监视的多个集群中具有队列，那么需要样本程序的多个实例。 每个实例都需要一个控制和应答消息的本地队列。
- 要监视的所有队列必须在单个集群中。 不监视配置为使用集群名称列表的队列。
- 可以选择启用非活动队列的消息传输。 它适用于集群队列监视样本程序实例监视的所有队列。 如果只有监视队列的子集需要启用消息传输，那么需要集群队列监视样本程序的两个实例。 一个样本程序启用消息传输，另一个禁用消息传输。 样本程序的每个实例都需要一个控制和应答消息的本地队列。
- 缺省情况下，IBM MQ 集群工作负载均衡将消息发送至集群队列实例，该实例驻留在放入应用程序连接的同一队列管理器。 以下情况中，当本地队列处于非活动状态时必须禁用此项：
  - 放入应用程序连接到拥有受监视的非活动队列实例的队列管理器。
  - 队列消息正在从非活动队列传输到活动队列。

  可通过将 CLWLUSEQ 值设置为 ANY，来静态禁用队列上的本地工作负载均衡首选项。 在此配置中，本地队列的消息分发到本地和远程队列实例以均衡工作负载，即使存在本地使用应用程序。 或者，在队列没有连接使用者，从而导致仅本地消息进入队列的本地实例（该队列为活动状态）时，可以配置集群队列监视样本程序以将 **CLWLUSEQ** 值临时设置为 ANY。
- IBM MQ 系统和应用程序对于要监视的队列或正在使用的通道不得使用 **CLWLPRTY**。 否则，集群队列监视样本程序对 **CLWLPRTY** 队列属性进行的操作可能不会取得预期的效果。
- 集群队列监视样本程序将运行时信息记录到一套报告文件中。 需要存储这些报告的目录，且集群队列监视样本程序必须具有写入报告的权限。

AMQSCLM：准备和运行样本
集群队列监视样本可连接到队列管理器以在本地运行，也可以作为通过通道连接的客户机来运行。 每当运行队列管理器时，都应运行此样本。在本地运行时，可将其配置为队列管理器服务，以使用队列管理器自动启动和停止样本。

开始之前

运行集群队列监视样本之前必须完成下列步骤。

1. 在每个队列管理器上为内部使用样本创建工作队列。

   对于独占的内部使用，样本的每个实例需要本地非集群队列。 您可以选择队列名称。 本示例使用名称 AMQSCLM.CONTROL.QUEUE。 例如，在 Windows 上，您可以使用以下 **MQSC** 命令创建此队列：

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   您可以将 **MAXDEPTH** 和 **MAXMSGL** 的值用作缺省值。
2. 创建错误和参考消息日志的目录。

样本将诊断消息写入报告文件。 您必须选择储存文件的目录。 例如，在 Windows 上，您可以使用以下命令创建目录：

```
mkdir C:\AMQSCLM\rpts
```

样本创建的报告文件具有以下命名约定：

```
 QmgrName.ClusterName.RPT0n.LOG
```

3. （可选）将集群队列监视样本定义为 IBM MQ 服务。

要监视队列，样本必须始终运行。 要确保集群队列监视样本始终运行，您可以将样本定义为队列管理器服务。 将样本定义为服务意味着，在队列管理器启动时，AMQSCLM 也启动。 您可以使用以下示例将集群队列监视样本定义为 IBM MQ 服务。

```
define service(AMQSCLM) +
      descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
      control(qmgr) +
      servtype(server) +
      startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
      startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
c:\AMQSCLM\rpts') +
      stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
      stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
```

定义	描述
service	定义服务名称。 您可以选择服务名称。
descr	指定服务的文本描述。
control	指示服务与队列管理器同时启动和停止。
servtype	指示服务器服务对象，意味着一次只能为此队列管理器执行一个实例。
startcmd	指定程序的位置和名称。
startarg	指定样本的参数。 请注意使用 *+QMNAME+*。将自动替换队列管理器的名称。
stdout	将标准输出重定向到的标准文件名。 样本仅将确认样本终止的消息写入此文件。 样本之所以这样做，是因为标准错误文件在样本终止过程的前期已经关闭。
stderr	标准错误输出重定向到的标准文件名。 样本在终止之前向标准错误文件中写入任何错误消息。

关于此任务

此任务使您能够以不同方式启动或停止集群队列监视样本。 同时使您在运行样本时生成包含受监视队列相关统计信息的报告文件。

可以使用以下命令运行样本程序。

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

此表列出可用于集成队列监视样本的参数，以及每个参数的其他信息。

参数	变量	更多信息
-m	**QMgrName**	要监视的队列管理器。
-c	**ClusterName**	包含要监视队列的集群。
-q	**QNameMask**	要监视的一个或多个队列。 尾部的 * 监视其名称与零个或更多尾部字符匹配的所有队列。
-f	**QListFile**	文件的完整路径和文件名，其中包含一系列要监视的队列名称或队列名称掩码。 文件的每一行必须包含一个队列名称/掩码。 可以指定 **-q** 或 **-f**，但不能同时指定两者。
-r	**MonitorQName**	被样本独占使用的本地队列。
-l	**ReportDir**	在一组包装中存储所记录的信息消息的目录路径 [9] 封装报告文件的目录路径。
-t		（可选）支持将队列消息从非活动本地队列转移到活动队列。 如果不启用，仅输入集群的新消息动态路由到队列的活动实例。
-u	**ActiveVal**	（可选）自动将受监视队列实例的 **CLWLUSEQ** 属性切换到 ANY（在受监视队列实例不活动时），或切换到 **ActiveVal** 值（在受监视队列实例活动时）。 **ActiveVal** 可以为 LOCAL 或 QMGR。 如果在放入应用程序连接到同一个队列管理器，或者启用消息传输时，系统中未设置此参数，那么受监视队列的 **CLWLUSEQ** 值必须设置为 ANY，或者队列管理器 QMGR 具有 ANY 值。
-i	**Interval**	（可选）监视器检查队列的时间间隔（秒）。 缺省值为 300 秒（5 分钟）。
-d		（可选）启用其他诊断输出。 初始配置系统或者使用样本代码时调试输出非常有用。
-s		（可选）启用每个时间间隔的最小统计输出。
-v		（可选）除了报告文件外，还会将报告信息记录到 standard out。

参数列表示例：

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

队列列表文件示例：

```
Q1
QUEUE.*
ABC
ABD
```

过程

1. 启动集群队列监视样本。 您可以通过以下方式之一启动样本：
   - 通过相应的用户权限使用命令提示符。
   - 如果样本配置为 IBM MQ 服务，请使用 MQSC **START SERVICE** 命令。

   在这两种情况下参数列表相同。

   初始化程序后 10 秒内，样本未开始监视队列。 此延迟允许使用应用程序先连接到受监视队列，防止对队列的活动状态进行不必要的更改。

[9] 对于每个队列管理器和队列组合，都会生成一个固定大小的日志文件；当该文件填满时，系统会覆盖该文件。 记录器将始终写入到同一个文件中，并保留该文件的两个先前版本。

2. 停止集群队列监视样本。 队列管理器已停止、正在停止、正在停顿，或者与队列管理器的连接中断时，样本自动停止。 以下方式将停止样本而不结束队列管理器：
   - 配置样本独占使用的本地队列，以禁用 GET 功能。
   - 将 **CorrelId** 为 "STOP CLUSTER MONITOR\0\0\0\0" 的消息发送到样本独占使用的本地队列中。
   - 终止样本流程。 这可能导致传输到活动队列的非持久性消息丢失。 还可能导致样本使用的本地队列在终止后持续开放几秒钟。 此情况防止立即启动集群队列监视样本的新实例。

   如果样本已经作为 IBM MQ 服务启动，那么 **STOP SERVICE** 没有效果。 可以在队列管理器中使用描述为 **STOP SERVICE** 配置机制的一种终止方法。

下一步做什么

检查样本状态。

如果启用报告，您可以查看报告文件获取样本状态。 使用以下命令来查看最新的报告文件：

```
QMgrName.ClusterName.RPT01.LOG
```

要查看较早的报告文件，请使用以下命令：

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

报告文件最大大小大约 1 MB。 当 RPT01 文件填满时，将创建新的 RPT01 文件。 旧的 RPT01 文件重命名为 RPT02。 RPT02 重命名为 RPT03。 将废弃旧的 RPT03。

在下列情况下，样本创建参考消息：

- 启动时
- 终止时
- 在将队列标记为 **ACTIVE** 或 **INACTIVE** 时
- 将非活动队列的消息重新排队到活动实例时

样本创建错误消息 *CLMnnnnE* 以报告需要注意的问题。

每 30 分钟，样本在轮询时间间隔报告平均处理时间和花费的处理时间。 此信息保存在消息 CLM0045I 中。

统计消息在 **-s** 下启用时，样本报告有关每个队列检查的以下统计信息。

- 处理队列花费的时间（毫秒）
- 检查的队列数
- 进行的活动/非活动更改数
- 传输的消息数

此信息在消息 CLM0048I 中报告。

报告文件可能在调试方式下快速增长，或快速分层。 在这种情况下，可能会超出单个文件 1 MB 的大小限制。

AMQSCLM：故障诊断
以下部分包含使用样本时可能遇到的场景的信息。 提供有关场景的可能解释信息以及如何解决的选项。

场景：AMQSCLM 未启动

可能解释：语法不正确。

操作：查看正确语法的标准错误输出

可能解释：队列管理器不可用。

操作：在报告文件中查找消息标识 CLM0010E。

可能解释：无法打开或创建一个或多个报告文件。

操作：检查初始化期间错误消息的标准错误输出。

场景：AMQSCLM 未将队列更改为 ACTIVE 或 INACTIVE

可能解释： 队列不在受监视的队列列表中

操作：检查 **-q** 和 **-f** 参数值。

可能解释：队列不是正确集群的本地队列。

操作：检查队列是否为本地队列且位于正确集群中。

可能解释： AMQSCLM 未针对此队列管理器和集群运行。

操作：针对相关队列管理器和集群启动 AMQSCLM。

可能解释：因为没有使用者，队列处于非活动状态，**CLWLPRTY** =0。 或者，因为至少一个使用者，队列保持活动状态，**CLWLPRTY** >=1。

操作：检查使用应用程序是否连接到队列。

可能解释：队列管理器的命令服务器未运行。

操作：检查报告文件以查看错误。

场景：消息未路由至非活动队列

可能解释： 消息直接放到拥有非活动队列的队列管理器，队列的 **CLWLUSEQ** 值不是 ANY，并且 **u** 参数未用于 AMQSCLM。

操作：检查相关队列管理器的 **CLWLUSEQ** 值，或确保 **-u** 参数用于 AMQSCLM。

可能解释：在任何队列管理器上没有活动队列。 消息跨所有非活动队列进行工作负载均衡，直到队列处于活动状态。

操作：检查所有队列管理器上队列的状态。

可能的解释：消息放在集群中不同于拥有非活动队列的集群管理器的其他队列管理器中，已更新 **CLWLPRTY** 值 0 不会传播到放入应用程序的队列管理器。

操作：检查受监视的队列管理器和完整存储库队列管理器之间的集群通道是否正在运行。 检查放入队列管理器和完整存储库队列管理器之间的通道是否正在运行。 检查受监视、放入和完整存储库队列管理器的错误日志。

可能的解释：远程队列实例是活动的 (CLWLPRTY=1)，但是消息无法路由到那些队列实例中，因为本地队列管理器的集群发送方通道未在运行。

操作：检查从本地队列管理器到一个或多个远程队列管理器（具有队列的活动实例）的集群发送方通道的状态。

场景：AMQSCLM 未从非活动队列传输消息

可能解释：未启用消息传输 （**t**）。

操作：确保消息传输已启用 (**-t**)。

可能解释：队列不在受监视的队列列表中。

操作：检查 **-q** 和 **-f** 参数值。

可能解释：AMQSCLM 未针对集群中拥有相同队列实例的此队列管理器或其他队列管理器运行。

操作：启动 AMQSCLM。

可能解释：队列的 **CLWLUSEQ** 为 LOCAL 或 **CLWLUSEQ** 为 QMGR，并且未设置 **-u** 参数。

操作：设置 **-u** 参数，或将队列或队列管理器配置更改为 ANY。

可能解释：集群中没有队列的活动实例。

操作：检查 **CLWLPRTY** 值为 1 或更大值的队列实例。

可能的解释：远程队列实例具有使用者 (**IPPROCS** >=1)，但使用者在那些队列管理器上为非活动 (**CLWLPRTY** =0)，原因是 AMQSCLM 未在监视那些远程实例。

操作：确保 AMQSCLM 在那些队列管理器上运行，以及/或者队列在受监视队列列表中，方式是选中 **-q** 和 **-f** 参数值。

可能的解释：远程队列实例是活动的 (**CLWLPRTY** =1)，但是在本地队列管理器上为认为是非活动的 (**CLWLPRTY** =0)。 发生此情况是由于更新的 **CLWLPRTY** 值未传播到此队列管理器。

操作：确保远程队列管理器连接到集群中至少一个完整存储库队列管理器。 确保完整存储库队列管理器正确运行。 请检查完整存储库队列管理器和受监视的队列管理器之间的通道是否正在运行。

可能解释：消息未提交，因此检索不到。

操作：检查发送应用程序是否正常运行。

可能解释：AMQSCLM 无法访问消息排队的本地队列。

操作：检查 AMQSCLM 是否正在作为具有足够授权以访问队列的用户在运行。

可能解释：队列管理器的命令服务器未运行。

操作：启动队列管理器的命令服务器。

可能解释：AMQSCLM 遇到错误。

操作：检查报告文件以查看错误。

可能解释： 远程队列实例是活动的 (CLWLPRTY=1)，但是消息无法传输到那些队列实例，因为来自本地队列管理器的集群发送方通道未运行。 通常在 amqsclm 报告日志中伴有 CLM0030W 警告。

操作：检查从本地队列管理器到一个或多个远程队列管理器（具有队列的活动实例）的集群发送方通道的状态。

ULW 连接端点查找 *(CEPL)* 的样本程序

IBM MQ 连接端点查找样本提供简单但功能强大的退出模块，该模块向 IBM MQ 用户提供了一种方式来从 LDAP 存储库（例如 Tivoli Directory Server）检索连接定义。

必须安装 Tivoli Directory Server V6.3 客户机才能使用 CEPL。

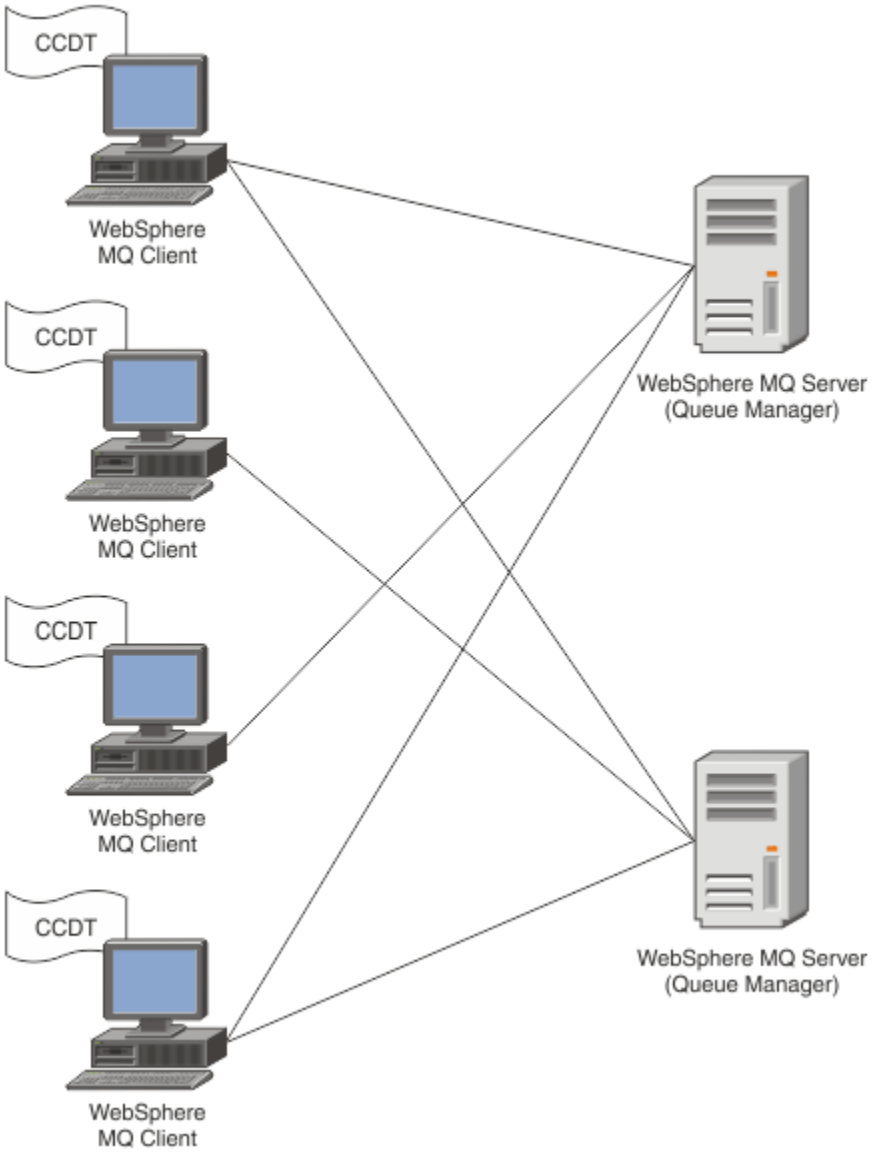
使用此样本需要在受支持的平台上管理 IBM MQ 的实际经验。

Windows Solaris Linux AIX 介绍

配置全局存储库，例如，LDAP（轻量级目录访问协议）目录，以存储客户机连接定义，协助维护和管理。

使用 IBM MQ Client 应用程序，通过客户机连接定义表 (CCDT) 建立至队列管理器的连接。

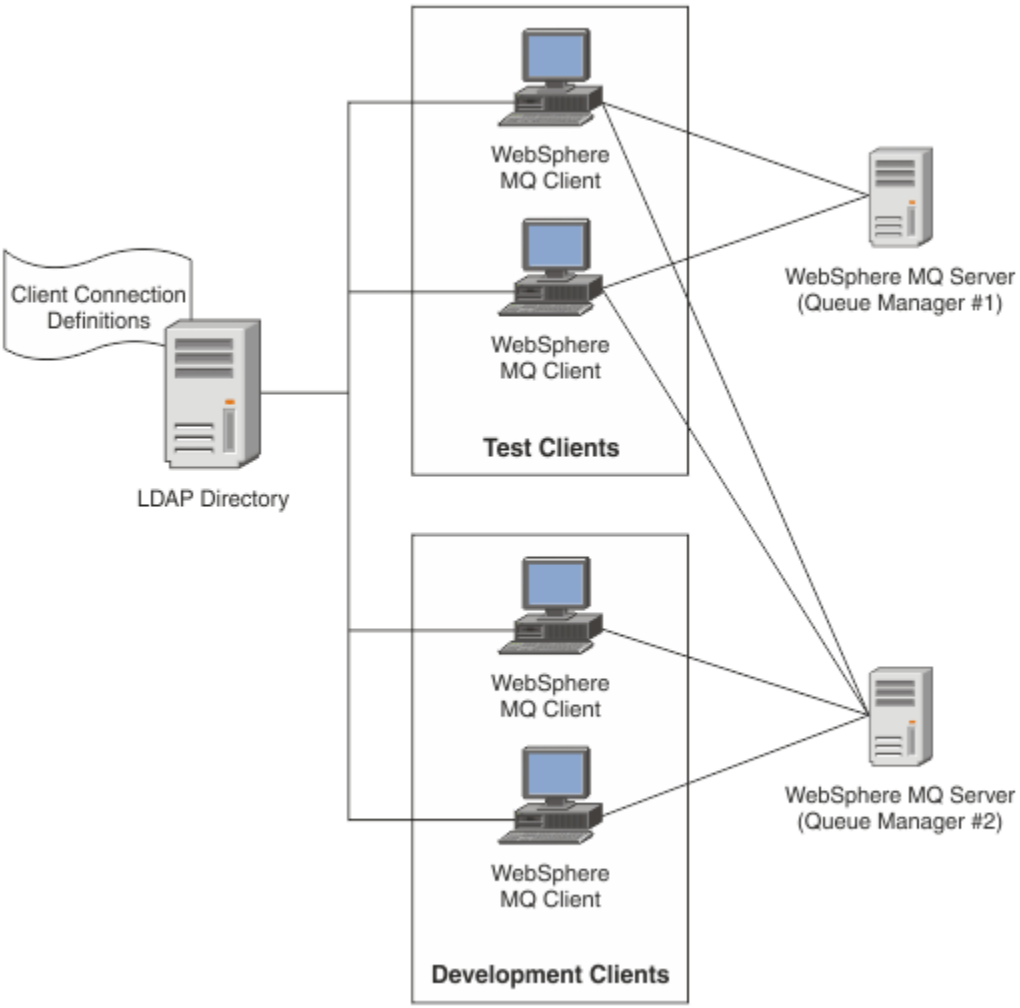
CCDT 通过标准 IBM MQ MQSC 管理界面创建。 用户必须连接到队列管理器才能创建客户机连接定义，即使定义中包含的数据没有限制为队列管理器。 生成的 CCDT 文件必须在客户机和应用程序之间手动分发。



必须向每个 IBM MQ 客户机分发 CCDT 文件。 当本地或全局存在成千上万个客户机时，将很难维护和管理。这就需要更灵活的方法来确保每个客户机都具有正确的客户机定义可用。

其中一个方法是在全局存储库（例如， LDAP（轻量级目录访问协议）目录）中存储客户机连接定义。 LDAP 目录也能提供额外的安全性、索引和搜索设施，从而允许每个客户机仅访问与他们相关的连接定义。

可以配置 LDAP 目录，使得仅特定定义可用于特定用户组。 例如，测试客户机可以访问队列管理器 #1 和 #2，而开发客户机仅可以访问队列管理器 #2。



退出模块可以查找 LDAP 目录，例如，IBM Tivoli Directory Server，以检索通道定义。 使用那些连接定义，IBM MQ 客户机应用程序可以建立到队列管理器的连接。

退出模块是预连接的退出模块，支持在 MQCONN/MQCONNX 调用期间从 LDAP 存储库获取通道定义。

退出模块和模式可能由以下对象实施：

- 已经使用基于现有 CCDT 文件的技术构建技能库并且想要降低管理和分发成本的客户。
- 已利用他们自己的专有技术分发客户机连接定义的现有客户。
- 当前没有利用任何类型客户机连接解决方案并且想要使用 IBM MQ 提供的功能部件的新客户或现有客户。
- 想要直接使用消息传递模型或微调消息传递模型以与任何当前 LDAP 业务体系结构保持一致的新客户或现有客户。

ULW 支持的环境

在运行“连接端点查找”样本之前验证您是否具有受支持的操作系统和相关软件。

IBM MQ 连接端点查找需要以下软件：

- IBM WebSphere MQ 7.0 或更高版本
- Tivoli Directory Server V6.3 客户机或更高版本

受支持的操作系统：

1. Windows Windows (7/8/2008/2012)

2. Solaris Solaris（SPARC 和 x86-64）
3. AIX AIX
4. Linux Linux
   - System p 上的 RHEL V4 和 V5
   - System p 上的 SUSE V9 和 V10
   - x86-64 32 位和 64 位上的 RHEL V4 和 V5
   - x86-64 32 位和 64 位上的 RHEL V4 和 V5

注: 该样本不可用于以下平台：

- z/OS z/OS
- IBM i IBM i

ULW 安装和配置
安装和配置退出模块和连接端点模式。

安装退出模块

在安装 IBM MQ 期间，出口模块安装在 `tools/samples/c/preconnexit/bin` 下。 对于 32 位平台，必须将出口模块复制到 `exit/`*`installation_name`*`/`，然后才能使用该模块。 对于 64 位平台，必须将退出模块复制到 exit64/*installation_name*/ 才可以使用。

安装连接端点模式

该出口使用连接端点模式 `ibm-amq.schema`。 必须将模式文件导入任何 LDAP 服务器才可以使用出口。 在导入模式之后，必须添加属性的值。

以下是导入连接端点模式的示例。 该示例假定正在使用 IBM Tivoli Directory Server (ITDS)。

- 确保 IBM Tivoli Directory Server 正在运行，然后将 `ibm-amq.schema` 文件复制或 FTP 到 ITDS 服务器。
- 在 ITDS 服务器上，输入以下命令以将模式安装到 ITDS 存储中，其中 *LDAP ID* 和 *LDAP password* 是 LDAP 服务器的根 DN 和密码：

  ```
  ldapadd -D "LDAP ID" -w "LDAP password" -f ibm-amq.schema
  ```
- 在命令窗口中，输入以下命令，或使用第三方工具浏览模式以进行验证：

  ```
  ldapsearch objectclass=ibm-amqClientConnection
  ```

请参阅 LDAP 服务器文档，获取有关导入模式文件的更多详细信息。

配置

必须将名为 PreConnect 的新部分添加到客户机配置文件，例如 `mqclient.ini`。 PreConnect 部分包含以下关键字：

Module：包含 API 退出代码的模块名称。 如果此字段包含模块的完整路径，那么将按原样使用该名称。 否则，将搜索 IBM MQ 安装中的 `exit` 或 `exit64` 文件夹。

Function：包含 PreConnect 退出代码的库中的函数入口点名称。 函数定义遵循 MQ_PRECONNECT_EXIT 原型。

Data：包含通道定义的 LDAP 存储库的 URI。

以下片段是 `mqclient.ini` 文件中所需的更改的示例。

```
PreConnect:
```

```
Module=amqlcelp
Function=PreConnectExit
Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ULW 出口和模式的概述

用于建立至队列管理器连接的语法和参数。

IBM MQ 9.1 定义出口模块中入口点的以下语法。

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

在 MQCONN/X 调用执行期间，IBM MQ C Client 装入包含函数语法实施的退出模块。 然后调用退出函数以检索通道定义。 然后将检索到的通道定义用于建立至队列管理器的连接。

参数

pExitParms

类型：PMQNXP 输入/输出

PreConnection 退出参数结构。 结构由出口的调用者分配和维护。

```
struct tagMQNXP
{
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     ExitId;              /* Type of exit */
  MQLONG     ExitReason;          /* Reason for invoking exit */
  MQLONG     ExitResponse;        /* Response from exit */
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

pQMgrName

类型：PMQCHAR 输入/输出

队列管理器的名称。 在输入时，该参数是通过 **QMgrName** 参数提供给 MQCONN API 调用的过滤器字符串。 此字段可能为空，显式或包含特定通配符。 该字段由出口更改。 使用 MQXR_TERM 调用出口时参数为 NULL。

ppConnectOpts

类型：ppConnectOpts 输入/输出

控制 MQCONNX 操作的选项。 这是控制 MQCONN API 调用操作的 MQCNO 连接选项结构的指针。 使用 MQXR_TERM 调用出口时参数为 NULL。 MQI 客户机始终向出口提供 MQCNO 结构，即使它最初不是由应用程序提供的。 如果应用程序提供 MQCNO 结构，那么客户机会制作副本以将其传递给修改它的出口。 客户机保留 MQCNO 的所有权。 通过 MQCNO 引用的 MQCD 优先于通过数组提供的任何连接定义。 客户机使用 MQCNO 结构来连接到此队列管理器，而忽略其他队列管理器。

pCompCode

类型：PMQLONG 输入/输出

完成代码。 指向用于接收出口完成代码的 MQLONG 的指针。 它必须是下列其中一个值：

- MQCC_OK - 成功完成
- MQCC_WARNING - 警告（部分完成）
- MQCC_FAILED - 调用失败

pReason
类型：PMQLONG 输入/输出

限定 pCompCode 的原因。 指向用于接收出口原因码的 MQLONG 的指针。 如果完成代码是 MQCC_OK，那么唯一的有效值是：MQRC_NONE - (0, x'000') 没有要报告的原因。

如果完成代码是 MQCC_FAILED 或 MQCC_WARNING，出口函数可以将原因码字段设置为任何有效 MQRC_* 值。

ULW *MQ LDAP* 上下文信息
出口使用以下数据结构获取上下文信息。

MQNLDAPCTX
MQNLDAPCTX 结构具有以下 C 原型。

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Solaris Linux AIX 构建连接端点查找出口的样本代码
您可以使用样本代码片段在 AIX、Linux、Solaris 和 Windows 上编译源代码。

编译源

您可以使用任何 LDAP 客户机库（例如 IBM Tivoli Directory Server V6.3 客户机库）编译源。 此文档假定您使用的是 Tivoli Directory Server V6.3 客户机库。

注: 以下 LDAP 服务器支持预连接的出口库。

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

以下代码片段描述如何编译出口：

Windows **在 Windows 平台上编译出口**
您可以使用以下片段来编译出口源：

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib
```

```
# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

注: 如果您正在使用通过 Microsoft Visual Studio 2003 编译器编译的 IBM Tivoli Directory Server V6.3 客户机库，那么在使用 Microsoft Visual Studio 2012 或更高版本的编译器编译 IBM Tivoli Directory Server V6.3 客户机库时，可能会收到警告。

Solaris Linux AIX **在 AIX、Linux 或 Solaris 上编译出口**

以下代码片段用于在 Linux 上编译出口源。 在 AIX 或 Solaris 上，某些编译器选项可能不同。

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server 随附了静态和动态链接库，但是您仅可以使用一种类型的库。 此脚本假定您使用的是静态库。

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl

amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ULW 调用 *PreConnect* 出口模块

可以使用三个不同的原因码调用 PreConnect 退出模块： MQXR_INIT 原因码用于初始化和建立至 LDAP 服务器的连接， MQXR_PRECONNECT 原因码用于从 LDAP 服务器检索通道定义，或者 在清除出口时使用 MQXR_TERM 原因码。

MQXR_INIT

使用 MQXR_INIT 原因码调用出口，以初始化和建立至 LDAP 服务器的连接。

在 MQXR_INIT 调用之前，将使用 `mqclient.ini` 文件（即 LDAP）中的 PreConnect 节中的 Data 属性来填充 MQNXP 结构的 pExitDataPtr 字段。

LDAP URL 至少包含协议、主机名、端口号以及搜索的基本 DN。 出口解析 pExitDataPtr 字段中包含的 LDAP URL，分配 MQNLDAPCTX LDAP Lookup Context 结构并进行相应的填充。 此结构的地址存储在 pExitUserAreaPtr 字段中。 解析 LDAP URL 失败导致错误 MQCC_FAILED。

此时，该出口将使用 **MQNLDAPCTX** 参数连接并绑定到 LDAP 服务器。 所生成的 LDAP API 手柄还存储在此结构中。

MQXR_PRECONNECT

使用 MQXR_PRECONNECT 原因码调用退出模块，以从 LDAP 服务器检索通道定义。

出口搜索 LDAP 服务器以获取匹配给定过滤器的通道定义。 如果 **QMgrNameparameter** 包含特定队列管理器名称，那么搜索将返回 **ibm-amqQueueManagerName** LDAP 属性值与给定队列管理器名称相匹配的所有通道定义。

如果 **QMgrName** 参数为 "*" 或 " "（空白），搜索会返回 **ibm-amqIsClientDefault Connection** 端点属性设置为 TRUE 的所有通道定义。

成功搜索之后，出口准备一个或一系列 MQCD 定义，并返回至调用者。

MQXR_TERM

要清除出口，请通过此原因码调用出口。 在此清除期间，出口从 LDAP 服务器断开连接，释放出口分配和维护的所有内存，包括 MQNLDAPCTX 结构、指针数组和其引用的每个 MQCD。 任何其他字段设置为

缺省值。 **pQMgrName** 和 **ppConnectOpts** 退出参数在使用 MQXR_TERM 原因码退出期间未使用，并且可能为 NULL。

相关概念

客户机配置文件的 PreConnect 节

ULW *LDAP* 模式

客户机连接数据存储在名为 LDAP（轻量级目录访问协议）目录的全局存储库中。 IBM MQ 客户机使用 LDAP 目录获取连接定义。 LDAP 目录中的 IBM MQ 客户机连接定义的结构被称为 LDAP 模式。 LDAP 模式是属性类型定义、对象类定义和服务器用来确定过滤器或属性值断言是否匹配条目属性以及是否允许、添加和修改操作的其他信息的集合。

在 LDAP 目录中存储数据

客户机连接定义位于作为连接点的目录树中的特定分支下。 像 LDAP 目录中的所有其他节点一样，连接点有专有名称 (DN) 与其关联。 您可以将此节点用作在目录上进行任何查询的起始点。 查询 LDAP 目录以返回客户机连接定义的子集时使用过滤。 您可以根据目录树其他部分中授予的权限限制对子树的访问，例如，限制用户、部门或组。

定义自己的属性和类

通过修改 LDAP 模式存储客户机通道定义。 所有 LDAP 数据定义都需要对象和属性。 对象和属性由唯一标识对象或属性的对象标识 (OID) 号识别。 LDAP 模式中的所有类直接或间接继承最高对象。 客户机通道定义对象包含最高对象的属性。 所有 LDAP 数据定义都需要对象和属性：

- 对象定义是 LDAP 属性的集合。
- 属性是 LDAP 数据类型。

每个属性的描述和他们如何映射到正常的 IBM MQ 属性在 LDAP 属性中进行了描述。

ULW *LDAP* 属性

定义的 LDAP 属性特定于 IBM MQ 并且直接映射到客户机连接属性。

IBM MQ 客户机通道目录字符串属性

字符串属性及其到 IBM MQ 属性的映射在下表中列出。 属性可以含有 directoryString（UTF-8 编码的 Unicode，即包含 IA5/ASCII 作为子集的可变字节编码系统）语法的值。 语法用其目标标识编号 (OID) 来表示。

表 *173: IBM MQ* 客户机通道目录字符串属性

LDAP 属性	**描述**	**IBM MQ 属性**
CN	通用名包括渠道名称和定义的队列管理器名称。	
ibm-amqChannelName	通道定义的名称。	通道
ibm-amqConnectionName	通信连接标识。	CONNAME
ibm-amqDescription	通道描述。	DESCR
ibm-amqLocalAddress	通道的本地通信地址。	LOCLADDR
ibm-amqModeName	LU 6.2 方式名。	MODENAME
ibm-amqPassword	可以使用的密码。	密码
ibm-amqQueueManagerName	IBM MQ 客户机应用程序可以请求连接到的队列管理器或队列管理器组的名称。	QMNAME
ibm-amqSecurityExitUserData	传递到安全出口的用户数据。	SCYDATA
ibm-amqSecurityExitName	通道安全出口运行的出口程序的名称。	SCYEXIT
ibm-amqSslCipherSpec	TLS 连接的单个 CipherSpec。	SSLCIPH

表 173: *IBM MQ* 客户机通道目录字符串属性 (继续)		
LDAP 属性	**描述**	**IBM MQ 属性**
ibm-amqSslPeerName	查看 IBM MQ 通道另一端同级队列管理器或客户机的证书的专有名称 (DN)。	SSLPEER
ibm-amqTransactionProgramName	事务程序名。	TPNAME
ibm-amqUserID	尝试使用远程 MCA 初始化安全 SNA 会话时 MCA 使用的用户标识。	USERID

IBM MQ 客户机连接整数属性

带有预定义值（例如，枚举类型）的属性作为标准整数存储。 这些值作为整数值存储在 LDAP 目录中，不使用关联的常量名称来存储。

表 174: *IBM MQ* 客户机通道目录整数属性		
LDAP 属性	**描述**	**IBM MQ 属性**
ibm-amqConnectionAffinity	确定通过同一个队列管理器名称多次连接的客户机应用程序是否使用相同的客户机通道。	AFFINITY
ibm-amqClientChannelWeight	影响使用哪个客户机连接通道定义的加权。	CLNTWGHT
ibm-amqHeartBeatInterval	传输队列上没有消息时，从发送 MCA 传递的脉动信号流量之间的大约间隔时间。	HBINT
ibm-amqKeepAliveInterval	通道的超时值。	KAINT
ibm-amqMaximumMessageLength	可在通道上传输的消息最大长度。	MAXMSGL
ibm-amqSharingConversations	共享每个 TCP/IP 通道实例的对话的最大数量。	SHARECNV
ibm-amqTransportType	要使用的传输类型。	TRPTYPE

IBM MQ 客户机通道布尔值属性

此布尔值属性不映射到任何 IBM MQ 属性。 此属性的语法表示布尔值。

表 175: *IBM MQ* 客户机通道布尔值属性	
LDAP 属性	**描述**
ibm-amqIsClientDefault	定义此布尔值属性以解决搜索其 ibm-amqQueueManagerName 属性未定义的条目的问题。

IBM MQ 客户机通道列表属性

IBM MQ 属性存储为 LDAP 目录中以逗号分隔的单值列表属性。 该属性和其他目录字符串属性采取同样的方法定义。 下表描述了列表属性及其到 IBM MQ 属性的映射。

表 176: *IBM MQ* 客户机通道列表属性		
LDAP 属性	**描述**	**IBM MQ 属性**
ibm-amqHeaderCompression	通道支持的头数据压缩技术列表。	COMPHDR
ibm-amqMessageCompression	通道支持的消息数据压缩技术列表。	COMPMSG
ibm-amqSendExitUserData	传递到发送出口的用户数据。	SENDDATA
ibm-amqSendExitUserName	通道发送出口运行的出口程序的名称。	SENDEXIT
ibm-amqReceiveExitUserData	传递到接收出口的用户数据。	RCVDATA

表 176: IBM MQ 客户机通道列表属性 (继续)		
LDAP 属性	**描述**	**IBM MQ 属性**
ibm-amqReceiveExitName	通道接收用户出口运行的用户出口程序的名称。	RCVEXIT

ULW 公共名称

通用名包括渠道名称和定义的队列管理器名称。

它是预先存在的属性。

CN 的格式是:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

例如:

```
CN=TC1(QM_T1)
```

您只能为此属性指定一个值。

此属性是字符串属性，值是不区分大小写的。 已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，其使用子字符串（例如，CN=jim*，其中 CN 是属性）指定搜索过滤器中属性的行为，并包含一个或多个通配符。

ULW *ibm-amqChannelName*

此属性指定通道定义的名称。

此属性具有不区分大小写、最大 20 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，其使用子字符串指定搜索过滤器中属性的行为，并包含一个或多个通配符。

ULW *ibm-amqDescription*

此 LDAP 属性提供通道描述。

此属性具有不区分大小写、最大 64 个字节的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqConnectionName*

此 LDAP 属性是通信连接标识。 它指定此通道使用的特别通信链路。

此属性具有不区分大小写、最大 264 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqLocalAddress*

此属性指定通道的本地通信地址。

此属性具有不区分大小写、最大 48 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqModeName*

此属性用于 LU 6.2 连接。 执行通信会话分配时，此属性为连接的会话特征提供额外定义。

此属性具有不区分大小写、恰好 8 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqPassword*
试图使用远程 MCA 初始化安全 LU 6.2 会话时，此 LDAP 属性指定 MCA 可以使用的密码。

此属性具有最大 12 个位的单个整数值。 它不是预先存在的属性。

ULW *ibm-amqQueueManagerName*
此属性指定 IBM MQ 客户机应用程序可以请求连接到的队列管理器或队列管理器组的名称。

此属性具有不区分大小写、最大 48 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

相关参考
第 1052 页的『ibm-amqIsClientDefault』
此布尔值属性解决在未定义 ibm-amqQueueManagerName 属性时搜索条目的问题。

ULW *ibm-amqSecurityExitUserData*
此 LDAP 属性指定传递到安全出口的用户数据。

此属性具有不区分大小写、最大 999 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqSecurityExitName*
此 LDAP 属性指定通道安全出口要运行的出口程序的名称。

如果没有有效的通道安全出口，就将其保留为空白。

此属性具有不区分大小写、最大 999 个字符的单个字符串值。 该属性不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqSslCipherSpec*
此 LDAP 属性为 TLS 连接指定单个 CipherSpec。

此属性具有不区分大小写、最大 32 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqSslPeerName*
此 LDAP 属性用于查看 IBM MQ 通道另一端同级队列管理器或客户机的证书的专有名称 (DN)。

此 LDAP 属性具有不区分大小写、最大 1024 个字节的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqTransactionProgramName*
此 LDAP 属性指定事务程序名。 可用于 LU 6.2 连接。

此属性具有不区分大小写、最大 64 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqUserID*
尝试使用远程 MCA 初始化安全 SNA 会话时此 LDAP 属性指定 MCA 使用的用户标识。

此属性具有不区分大小写、恰好 12 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

ULW *ibm-amqConnectionAffinity*
此 LDAP 属性指定使用同一个队列管理器名称多次连接的客户机应用程序是否使用相同的客户机通道。

此属性有单个整数值。 它不是预先存在的属性。

ULW *ibm-amqClientChannelWeight*
此 LDAP 属性指定影响使用哪个客户机连接通道定义的加权。

在多个合适定义可用时，可根据客户机通道加权属性来选择客户机通道定义。

此属性有单个整数值。 它不是预先存在的属性。

ULW *ibm-amqHeartBeatInterval*
此 LDAP 属性指定在传输队列上没有消息时从发送 MCA 传递的脉动信号流量之间的大约间隔时间。

此属性有单个整数值。 它不是预先存在的属性。 缺省值为 1。 在当前 MQSERVER 环境变量操作中设置了缺省值。

ULW *ibm-amqKeepAliveInterval*
此 LDAP 属性用于指定通道的超时值。

此属性的值传递到指定通道的保持活动计时的通信堆栈。 您可以使用它为每个通道指定不同的保持活动值。

此属性有单个整数值。 它不是预先存在的属性。

ULW *ibm-amqMaximumMessageLength*
此 LDAP 属性指定可以在通道上传输的消息的最大长度。

对于当前每个 MQSERVER 环境变量操作，此属性的缺省值是 104857600。 此属性有单个整数值，并且不是预先存在的属性。

ULW *ibm-amqSharingConversations*
此 LDAP 属性指定共享每个 TCP/IP 通道实例的最大对话数。

此属性有单个整数值。 它不是预先存在的属性。

ULW *ibm-amqTransportType*
此 LDAP 属性指定要使用的传输类型。

此属性有单个整数值。 它不是预先存在的属性。

ULW *ibm-amqIsClientDefault*
此布尔值属性解决在未定义 ibm-amqQueueManagerName 属性时搜索条目的问题。

预连接出口模式通常使用 ibm-amqQueueManagerName 属性的值作为搜索条件来搜索 LDAP 服务器。 此类查询将返回 ibm-amqQueueManagerName 属性值与 MQCONN/X 调用中指定的队列管理器名称匹配时的所有条目。 但是，当使用客户机通道定义表 (CCDT) 时，您可以将 MQCONN/X 调用上的队列管理器名称设置为空白或以星号 (*) 为前缀。如果队列管理器的名称为空白，那么客户机将连接到缺省队列管理器。 如果队列管理器名称添加星号 (*) 作前缀，那么客户机可连接任何队列管理器。

同样，可以不定义条目中的 ibm-amqQueueManagerName 属性。 在这种情况下，预计使用此端点信息的客户机可以连接到任何队列管理器。 例如，条目包含以下行：

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

在此示例中，客户机尝试连接到 myhost 上运行的指定的队列管理器。

但是，在 LDAP 服务器中，不对未定义的属性值进行搜索。 例如，如果条目包含 ibm-amqQueueManagerName 之外的连接信息，那么搜索结果不包含此条目。 要克服此问题，您可以设置 ibm-amqIsClientDefault。 这是布尔值属性，如果不定义，假定值为 FALSE。

对于未定义 ibm-amqQueueManagerName 且预计为搜索一部分的条目，将 ibm-amqIsClientDefault 设置为 TRUE。 在 MQCONN/X 调用的队列管理器名称指定为空白或星号 (*) 时，预连接出口将搜索 LDAP 服务器以获取所有 ibm-amqIsClientDefault 属性值设置为 TRUE 的条目。

注: 如果 ibm-amqIsClientDefault 设置为 TRUE，不要设置或定义 ibm-amqQueueManagerName 属性。

相关参考

第 1051 页的『ibm-amqQueueManagerName』
此属性指定 IBM MQ 客户机应用程序可以请求连接到的队列管理器或队列管理器组的名称。

ULW *ibm-amqHeaderCompression*
此 LDAP 属性是通道支持的头数据压缩技术的列表。

此属性的最大大小是 48 个字符。 它不是预先存在的属性。

您只能为此属性指定一个值。

使用以逗号分隔的格式将此列表属性指定为目录字符串。 例如，为 **ibm-amqHeaderCompression** 指定的值是 0，其映射到 NONE。 客户机将忽略超过最高许可限制的任何值。 例如，ibm-amqHeaderCompression 在列表中包含最多 2 个整数。

ULW *ibm-amqMessageCompression*
此 LDAP 属性是通道支持的消息数据压缩技术的列表。

此属性的最大大小是 48 个字符。 它不是预先存在的属性。

该属性不支持多个值。

使用以逗号分隔的格式将此列表属性指定为目录字符串。 例如，为此属性指定的值是 1、2、4，其映射到底层的压缩序列 RLE、ZLIBFAST 和 ZLIBHIGH。

客户机将忽略超过最高许可限制的任何值。 例如，ibm-amqMessageCompression 在列表中最多包含 16 个整数。

ULW *ibm-amqSendExitUserData*
此 LDAP 属性指定传递到发送出口的用户数据。

此 LDAP 属性具有不区分大小写、最大 999 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

注: 需要成对同步 **ibm-amqSendExitName** 和 **ibm-amqSendExitUserData** 。 用户数据应当与出口名称同步。 因此，如果指定一个，也必须对称地指定另一个，即使其中不包含数据。

ULW *ibm-amqSendExitName*
此 LDAP 属性指定通道发送出口要运行的出口程序的名称。

此属性具有不区分大小写、最大 999 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

注: ibm-amqSendExitName 和 **ibm-amqSendExitUserData** 必须成对同步。 用户数据必须与出口名称同步。 因此如果指定一个，也必须对称地指定另一个，即使其中不包含数据。

ULW *ibm-amqReceiveExitUserData*
此 LDAP 属性指定传递到接收出口的用户数据。

您可以运行一系列接收出口。 一系列出口的用户数据字符串使用逗号和/或空格分开。

此属性具有不区分大小写、最大 999 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

注: ibm-amqReceiveExitName 和 **ibm-amqReceiveExitUserData** 必须成对同步。 用户数据必须与出口名称同步。 因此如果指定一个，也必须对称地指定另一个，即使其中不包含数据。

ULW *ibm-amqReceiveExitName*
此 LDAP 属性指定通道接收用户出口要运行的用户出口程序的名称。

此属性是要连续运行的程序名称的列表。 如果没有有效的通道接收用户出口，那么将其保留为空白。

此属性具有不区分大小写、最大 999 个字符的单个字符串值。 它不是预先存在的属性。

已忽略子串匹配。 子串匹配是子模式中使用的匹配规则，指定搜索过滤器中属性的行为。

注: ibm-amqReceiveExitName 和 **ibm-amqReceiveExitUserData** 必须成对同步。 用户数据必须与出口名称同步。 因此如果指定一个，也必须对称地指定另一个，即使其中不包含数据。

z/OS 针对 z/OS 使用样本程序

随附于 IBM MQ for z/OS 的样本过程应用程序演示消息队列接口 (MQI) 的典型用法。

关于此任务

IBM MQ for z/OS 还提供样本数据转换出口，如第 893 页的『编写数据转换出口』中所述。

所有样本应用程序以源形式提供，几个样本应用程序还可以以可执行文件形式提供。 源模块包含描述程序逻辑的伪码。

注: 虽然一些样本应用程序有面板驱动的基本界面，但它们并非旨在演示如何设计您应用程序的外观。 有关如何设计不可编程终端的面板驱动界面的更多信息，请参阅 *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) 及其附录 (GG22-9508)。 上述指南可帮助您设计在应用程序内及跨其他应用程序保持一致的应用程序。

过程

- 使用以下链接了解有关样本程序的更多信息:
  - 第 1054 页的『在针对 z/OS 的样本应用程序中演示的功能』
  - 第 1060 页的『在 z/OS 上为批处理环境准备和运行样本应用程序』
  - 第 1063 页的『在 z/OS 上为 TSO 环境准备样本应用程序』
  - 第 1064 页的『在 z/OS 上为 CICS 环境准备样本应用程序』
  - 第 1067 页的『在 z/OS 上为 IMS 环境准备样本应用程序』
  - 第 1068 页的『z/OS 上的 Put 样本』
  - 第 1070 页的『z/OS 上的 Get 样本』
  - 第 1072 页的『z/OS 上的 Browse 样本』
  - 第 1073 页的『z/OS 上的 Print Message 样本』
  - 第 1076 页的『z/OS 上的 Queue Attributes 样本』
  - 第 1078 页的『z/OS 上的 Mail Manager 样本』
  - 第 1083 页的『z/OS 上的 Credit Check 样本』
  - 第 1093 页的『z/OS 上的 Message Handler 样本』
  - 第 1096 页的『z/OS 上的 Asynchronous Put 样本』
  - 第 1097 页的『z/OS 上的 Batch Asynchronous Consumption 样本』
  - 第 1099 页的『z/OS 上的 CICS 异步使用和发布/预订样本』
  - 第 1101 页的『z/OS 上的 Publish/Subscribe 样本』
  - 第 1103 页的『z/OS 上的 Set 和 Inquire 消息属性样本』

相关任务

第 964 页的『在 Multiplatforms 上使用样本程序』
这些样本过程程序随附于产品。 样本采用 C 和 COBOL 编写，并且演示消息队列接口 (MQI) 的典型用法。

z/OS 在针对 *z/OS* 的样本应用程序中演示的功能

本节总结了每个样本应用程序中演示的 MQI 功能，并显示了用于编写每个样本的编程语言以及用于运行每个样本的环境。

z/OS *z/OS 上的 Put 样本*

Put 样本演示了如何使用 MQPUT 调用将消息放入队列。

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

程序通过 COBOL 和 C 提供，在批处理环境和 CICS 环境中运行。 请参阅 第 1061 页的表 179 以获取批处理应用程序，并参阅 第 1065 页的表 186 以获取 CICS 应用程序。

z/OS *z/OS 上的 Get 样本*

Get 样本演示了如何使用 MQGET 调用从队列中获取消息。

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

程序通过 COBOL 和 C 提供，在批处理环境和 CICS 环境中运行。 请参阅 第 1061 页的表 179 以获取批处理应用程序，并参阅 第 1065 页的表 186 以获取 CICS 应用程序。

z/OS *z/OS 上的 Browse 样本*

Browse 样本显示如何使用“浏览”选项查找和打印消息，然后逐步浏览队列的消息。

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQGET（浏览消息）
- MQCLOSE
- MQDISC

程序以 COBOL、汇编程序、PL/I 和 C 语言提供。 应用程序在批处理环境下运行。 有关批处理应用程序，请参阅第 1061 页的表 180。

z/OS *z/OS 上的“打印消息”样本*

“打印消息”样本显示如何从队列移除消息，打印消息中的数据以及消息描述符的所有字段。 它可以选择性地显示与每条消息相关联的所有消息属性。

通过移除源模块两行中的注释字符，您可以更改程序，这样便可以浏览而不是移除队列上的消息。 此程序用于诊断在队列上放置消息的应用程序的问题，且效果很好。

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQGET（通过浏览选项移除队列中的消息）
- MQCLOSE
- MQDISC

- MQCRTMH
- MQDLTMH
- MQINQMP

程序通过 C 语言提供。 应用程序在批处理环境下运行。 有关批处理应用程序，请参阅第 1062 页的表 181。

z/OS *z/OS 上的“队列属性”样本*
“队列属性”样本演示了如何查询和设置 IBM MQ for z/OS 对象属性的值。

应用程序使用以下 MQI 调用：

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

程序通过 COBOL、汇编程序和 C 语言提供。 应用程序在 CICS 环境下运行。 请参阅 第 1065 页的表 187 以了解 CICS 应用程序。

z/OS *z/OS 上的“邮件管理器”样本*
使用“邮件管理器”样本时需要注意的事项。

“邮件管理器”样本显示以下技术：

- 使用别名队列
- 使用模型队列创建临时动态队列
- 使用应答队列
- 在 CICS 和批处理环境中使用同步点
- 发送命令至系统命令输入队列
- 测试返回码
- 通过使用远程队列的本地定义和直接将消息放入远程队列管理器的命名队列来将消息发送至远程队列管理器

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

提供了该应用程序的三个版本：

- 采用 COBOL 编写的 CICS 应用程序
- 采用 COBOL 编写的 TSO 应用程序
- 采用 C 编写的 TSO 应用程序

TSO 应用程序使用 IBM MQ for z/OS 批处理适配器，并且包含一些 ISPF 面板。

请参阅 第 1063 页的表 184 以了解 TSO 应用程序，并参阅 第 1066 页的表 188 以了解 CICS 应用程序。

z/OS *z/OS* 上的“信用检查”样本

此信息包含使用“信用检查”样本时要考虑的几点内容。

“信用检查”样本是展示这些技术的一套程序：

- 开发在多个环境中运行的应用程序
- 使用模型队列创建临时动态队列
- 使用相关标识
- 设置和传递上下文信息
- 使用消息优先级和持久性
- 通过触发启动程序
- 使用应答队列
- 使用别名队列
- 使用死信队列
- 使用名称列表
- 测试返回码

应用程序使用以下 MQI 调用：

- MQOPEN
- MQPUT
- MQPUT1
- MQGET（使用等待和信号选项浏览和获取消息，以及获取特定消息）
- MQINQ
- MQSET
- MQCLOSE

样本可以作为独立的 CICS 应用程序运行。 但是，为演示如何设计使用 CICS 和 IMS 环境均提供的设施的消息排队应用程序，还提供一个模块作为 IMS 成批消息处理程序。

CICS 程序以 C 和 COBOL 形式交付。 单个 IMS 程序通过 C 提供。

请参阅 第 1066 页的表 189 以了解 CICS 应用程序，并参阅 第 1068 页的表 191 以了解 IMS 应用程序。

z/OS *z/OS* 上的 *Message Handler* 样本

Message Handler 样本允许您浏览、转发和删除队列中的消息。

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

程序通过 C 和 COBOL 编程语言提供。 应用程序在 TSO 下运行。 有关 TSO 应用程序，请参阅第 1064 页的表 185。

z/OS *z/OS* 上的分布式排队的出口样本

分布式排队的出口样本的源程序表。

下表列出分布式排队的出口样本的源程序名称。

表 177: 分布式排队的出口样本的源			
成员名	**适用语言**	**描述**	**在库中提供**
CSQ4BAX0	汇编程序	源程序	SCSQASMS
CSQ4BCX1	C	源程序	SCSQC37S
CSQ4BCX2	C	源程序	SCSQC37S
CSQ4BCX4	C	源程序	SCSQC37S

注: 源程序是使用 CSQXSTUB 链接编辑的。

z/OS *z/OS* 上的数据转换出口样本

提供一个框架用于数据转换出口例程，且 IBM MQ 随附一个样本来演示 MQXCNVC 调用。

数据转换出口样本的源程序名称在以下表格中显示。

表 178: 数据转换出口样本的源（仅汇编语言）		
成员名	**描述**	**在库中提供**
CSQ4BAX8	源程序	SCSQASMS
CSQ4BAX9	源程序	SCSQASMS
CSQ4CAX9	源程序	SCSQASMS

注: 源程序是使用 CSQASTUB 链接编辑的。

请参阅 第 893 页的『编写数据转换出口』 以获取更多信息。

z/OS *z/OS* 上的 *Publish/Subscribe* 样本

Publish/Subscribe 样本程序演示在 IBM MQ 中使用发布和预订功能。

有四个 C 和两个 COBOL 编程语言样本程序用于演示如何对 IBM MQ 发布/预订接口进行编程。

应用程序使用以下 MQI 调用:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

“发布/预订”样本程序通过 C 和 COBOL 编程语言提供。 样本应用程序在批处理环境下运行。 有关批处理应用程序，请参阅发布/预订样本。

z/OS 配置队列管理器以接受 *z/OS* 上的客户机连接

要运行样本应用程序，必须首先创建队列管理器。 然后，可将队列管理器配置为以安全方式接受来自以客户机方式运行的应用程序的入局连接请求。

开始之前

确保队列管理器已经存在并已启动。 通过发出 MQSC 命令确定是否已经启用通道认证记录：

```
DISPLAY QMGR CHLAUTH
```

要点： 此任务期望启用通道认证记录。 如果这是由其他用户和应用程序使用的队列管理器，那么更改此设置将影响所有其他用户和应用程序。 如果队列管理器不利用通道认证记录，那么步骤 4 可以替换为备用认证方法（例如，安全出口），它将 MCAUSER 设置为将在步骤 第 1059 页的『1』 中获取的 *non-privileged-user-id*。

您必须知道应用程序期望使用的通道名称，以便可以允许应用程序使用该通道。 您还必须知道应用程序期望使用哪些对象（例如队列或主题），以便可以允许应用程序使用这些对象。

关于此任务

此任务创建要用于连接到队列管理器的客户机应用程序的非特权用户标识。 将会授予使客户机应用程序只能通过使用此用户标识来使用其所需的通道和队列的访问权。

过程

1. 在队列管理器运行所在的系统上获取用户标识。

   对于此任务，此用户标识不得是特权管理用户。 此用户标识是客户机连接在队列管理器上运行将使用的权限。
2. 启动侦听器程序。

   a) 确保已启动通道启动程序。 如果没有，请通过发出 **START CHINIT** 命令将其启动。

   b) 通过发出以下命令来启动侦听器程序：

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   其中，*nnnn* 是所选端口号。
3. 如果应用程序使用 SYSTEM.DEF.SVRCONN，那么表明已经定义此通道。 如果应用程序使用其他通道，请通过发出 MQSC 命令创建该通道：

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name 是通道的名称。
4. 通过发出 MQSC 命令创建通道认证规则，从而仅允许客户机系统的 IP 地址使用通道：

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   其中：

   channel-name 是通道的名称。

   client-machine-IP-address 是客户机系统的 IP 地址。 如果样本客户机应用程序与队列管理器在同一机器上运行，那么在应用程序即将使用“localhost”进行连接时，使用 IP 地址“127.0.0.1”。 如果即将接入若干不同客户端机器，那么可以使用模式或范围而不是单个 IP 地址。 请参阅类属 IP 地址以获取详细信息。

   non-privileged-user-id 是在步骤 第 1059 页的『1』 中获取的用户标识
5. 如果应用程序使用 SYSTEM.DEFAULT.LOCAL.QUEUE，那么表明已经定义此队列。 如果应用程序使用其他队列，请通过发出 MQSC 命令创建该队列：

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   其中，*queue-name* 是队列的名称。

6. 授予用于连接并查询队列管理器的访问权：
   a) 确保已启动通道启动程序。 如果未启动，请通过发出 START CHINIT 命令启动通道启动程序。
   b) 启动 TCP 侦听器，例如，发出以下命令：

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

   其中，*nnnn* 是所选端口号。
7. 如果应用程序是点到点应用程序，那么表明它利用队列，通过发出 MQSC 命令来授予访问权，以允许按用户标识使用队列查询以及放置和获取消息：
   发出 RACF 命令：

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

   其中：

   qmgr-name 是队列管理器的名称
   queue-name 是队列的名称。
   non-privileged-user-id 是在步骤 第 1059 页的『1』 中获取的用户标识
8. 如果应用程序是发布/预订应用程序（即，它利用主题），请通过发出以下 RACF 命令，按要使用的用户标识授予访问权以允许使用主题进行发布和预订：

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

   其中：

   qmgr-name 是队列管理器的名称
   non-privileged-user-id 是在步骤 第 1059 页的『1』 中获取的用户标识
   这将给予对主题树中任何主题的 *non-privileged-user-id* 访问权，或者，可以使用 **DEFINE TOPIC** 来定义主题对象并仅授予对该主题对象引用的主题树部分的访问权。 有关更多信息，请参阅控制用户对主题的访问权。

下一步做什么

客户机应用程序现在可以连接到队列管理器并使用队列放置或获取消息。

相关概念

z/OS 在 z/OS 上使用 IBM MQ 对象的权限

相关参考

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC

z/OS 在 z/OS 上为批处理环境准备和运行样本应用程序

要准备在批处理环境中运行的样本应用程序，请执行在构建任何批处理 IBM MQ for z/OS 应用程序时要执行的步骤。

这些步骤列出在第 935 页的『构建 z/OS 批处理应用程序』中。

或者，在我们提供了可执行形式的样本时，您可以从 thlqual.SCSQLOAD 装入库运行该样本。

注: “浏览”样本的汇编语言版本使用数据控制块 (DCB)，因此您必须使用 RMODE(24) 进行链接编辑。

要使用的库成员在第 1061 页的表 179、第 1061 页的表 180、第 1062 页的表 181 和 第 1062 页的表 182 中列示。

您必须编辑为您要使用的样本提供的运行 JCL（请参阅第 1061 页的表 179、第 1061 页的表 180、第 1062 页的表 181 和第 1062 页的表 182）。

所提供的 JCL 中的 PARM 语句包含一些您需要修改的参数。 要运行 C 样本程序，通过空格分隔参数；要运行汇编程序、COBOL 和 PL/I 样本程序，通过逗号分隔参数。 例如，如果队列管理器的名称为 CSQ1，并且您想要使用名称为 LOCALQ1 的队列运行应用程序，在 COBOL、PL/I 和汇编语言 JCL 中，您的 PARM 语句应如下所示:

```
PARM=(CSQ1,LOCALQ1)
```

在 C 语言 JCL 中，您的 PARM 语句应如下所示:

```
PARM=('CSQ1 LOCALQ1')
```

现在您已准备好提交作业。

z/OS *z/OS* 上样本批处理应用程序的名称

为样本批处理应用程序提供的程序的摘要。

下表概述了批处理应用程序:

- 第 1061 页的表 179 Put 和 Get 样本
- 第 1061 页的表 180 Browse 样本
- 第 1062 页的表 181 Print Message 样本
- 第 1062 页的表 182 Publish/Subscribe 样本
- 第 1062 页的表 183 其他样本

表 *179:* 批处理 *Put* 和 *Get* 样本

成员名	适用语言	描述	库中提供的源文件	库中提供的可执行文件
CSQ4BCJ1	C	Get 源程序	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Put 源程序	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	CSQ4BCJ1 和 CSQBCK1 的样本运行 JCL	SCSQPROC	None
CSQ4BVJ1	COBOL	Get 源程序	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Put 源程序	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	CSQBVJ1 和 CSQBVK1 的样本运行 JCL	SCSQPROC	None

表 *180:* 批处理 *Browse* 样本

成员名	适用语言	描述	库中提供的源文件	库中提供的可执行文件
CSQ4BVA1	COBOL	源程序	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	CSQ4BVA1 的样本运行 JCL	SCSQPROC	None

表 *180:* 批处理 *Browse* 样本 *(*继续*)*

成员名	适用语言	描述	库中提供的源文件	库中提供的可执行文件
CSQ4BAA1	汇编程序	源程序	SCSQASMS	SCSQLOAD
CSQ4BAAR	汇编程序	CSQ4BAA1 的样本运行 JCL	SCSQPROC	None
CSQ4BCA1	C	源程序	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	CSQ4BCA1 的样本运行 JCL	SCSQPROC	None
CSQ4BPA1	PL/I	源程序	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	CSQ4BPA1 的样本运行 JCL	SCSQPROC	None

表 *181:* 批处理 *Print Message* 样本（仅 *C* 语言）

成员名	描述	库中提供的源文件	库中提供的可执行文件
CSQ4BCG1	源程序	SCSQC37S	SCSQLOAD
CSQ4BCGR	CSQ4BCG1 的样本运行 JCL	SCSQPROC	None
CSQ4BCL1	Browse 源程序	SCSQC37S	SCSQLOAD
CSQ4BCLR	CSQ4BCL1 的样本运行 JCL	SCSQPROC	None

表 *182: Publish/Subscribe* 样本

成员名	适用语言	描述	库中提供的源文件	在 SCSQPROC 中的 JCL	库中提供的可执行文件
CSQ4BCP1	C	发布到主题源程序	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	预订主题并获取消息源程序	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	使用用户提供的目标订阅主题并获取消息源程序	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	使用扩展选项预订主题并获取消息源程序	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	发布到主题源程序	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	预订主题并获取消息源程序	SCSQCOBS	CSQ4BVPS	SCSQLOAD

表 *183:* 其他样本

成员名	适用语言	描述	库中提供的源文件	在 SCSQPROC 中的 JCL	库中提供的可执行文件
CSQ4BCS1	C	异步使用源程序	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	异步 Put 和 Check 状态源程序	SCSQC37S	CSQ4BCSP	SCSQLOAD

表 183: 其他样本 (继续)					
成员名	**适用语言**	**描述**	**库中提供的源文件**	**在 SCSQPROC 中的 JCL**	**库中提供的可执行文件**
CSQ4BCM1	C	询问消息属性源程序	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	设置消息属性源程序	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS 在 z/OS 上为 TSO 环境准备样本应用程序

要准备在 TSO 环境中运行的样本应用程序，请执行在构建任何批处理 IBM MQ for z/OS 应用程序时要执行的步骤。

这些步骤列出在第 935 页的『构建 z/OS 批处理应用程序』中。 要使用的库成员列出在第 1063 页的表 184中。

或者，在我们提供了可执行形式的样本时，您可以从 thlqual.SCSQLOAD 装入库运行该样本。

对于“邮件管理器”样本应用程序，确保其使用的队列在您的系统上可用。 他们在成员 **thlqual**.SCSQPROC(CSQ4CVD) 中定义。 要确保这些队列始终可用，您可以将这些成员添加到 CSQINP2 初始化输入数据集，或者使用 CSQUTIL 程序装入这些队列定义。

z/OS z/OS 上样本 TSO 应用程序的名称

有关为每个样本 TSO 应用程序提供的程序的名称，以及源、JCL 和可执行文件（仅针对消息处理程序样本）驻留的库的信息。

以下各表中概述了 TSO 应用程序：

- 第 1063 页的表 184 邮件管理器样本
- 第 1064 页的表 185 消息处理程序样本

这些样本使用 ISPF 面板。 所以，在链接编辑程序时必须包含 ISPF 存根 ISPLINK。

表 184: TSO 邮件管理器样本			
成员名	**适用语言**	**描述**	**库中提供的源文件**
CSQ4CVD	独立	IBM MQ for z/OS 对象定义	SCSQPROC
CSQ40	独立	ISPF 消息	SCSQMSGE
CSQ4RVD1	COBOL	启动 CSQ4TVD1 的 CLIST	SCSQCLST
CSQ4TVD1	COBOL	“菜单”程序的源程序	SCSQCOBS
CSQ4TVD2	COBOL	“获取邮件”程序的源程序	SCSQCOBS
CSQ4TVD4	COBOL	“发送邮件”程序的源程序	SCSQCOBS
CSQ4TVD5	COBOL	“昵称”程序的源程序	SCSQCOBS
CSQ4VDP1-6	COBOL	面板定义	SCSQPNLA
CSQ4VD0	COBOL	数据定义	SCSQCOBC
CSQ4VD1	COBOL	数据定义	SCSQCOBC
CSQ4VD2	COBOL	数据定义	SCSQCOBC
CSQ4VD4	COBOL	数据定义	SCSQCOBC
CSQ4RCD1	C	启动 CSQ4TCD1 的 CLIST	SCSQCLST

表 184: TSO 邮件管理器样本 (继续)			
成员名	**适用语言**	**描述**	**库中提供的源文件**
CSQ4TCD1	C	“菜单”程序的源程序	SCSQC37S
CSQ4TCD2	C	“获取邮件”程序的源程序	SCSQC37S
CSQ4TCD4	C	“发送邮件”程序的源程序	SCSQC37S
CSQ4TCD5	C	“昵称”程序的源程序	SCSQC37S
CSQ4CDP1-6	C	面板定义	SCSQPNLA
CSQ4TC0	C	包含文件	SCSQC370

表 185: TSO 消息处理程序样本				
成员名	**适用语言**	**描述**	**库中提供的源文件**	**库中提供的可执行文件**
CSQ4TCH0	C	数据定义	SCSQC370	None
CSQ4TCH1	C	源程序	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	源程序	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	源程序	SCSQC37S	SCSQLOAD
CSQ4RCH1	C 和 COBOL	启动 CSQ4TCH1 或 CSQ4TVH1 的 CLIST	SCSQCLST	None
CSQ4CHP1	C 和 COBOL	面板定义	SCSQPNLA	None
CSQ4CHP2	C 和 COBOL	面板定义	SCSQPNLA	None
CSQ4CHP3	C 和 COBOL	面板定义	SCSQPNLA	None
CSQ4CHP9	C 和 COBOL	面板定义	SCSQPNLA	None
CSQ4TVH0	COBOL	数据定义	SCSQCOBC	None
CSQ4TVH1	COBOL	源程序	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	源程序	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	源程序	SCSQCOBS	SCSQLOAD

z/OS 在 z/OS 上为 CICS 环境准备样本应用程序

在您运行 CICS 样本程序之前，使用日志方式 32702 登录到 CICS。 这是因为已编写样本程序来使用 3270 方式 2 屏幕。

要准备在 CICS 环境中运行的样本应用程序，请执行以下步骤：

1. 通过组合 BMS 屏幕定义源（在库 **thlqual**.SCSQMAPS 中提供，其中 **thlqual** 是您的安装所使用的高级别限定词）为样本创建符号描述映射和物理屏幕映射。 命名映射时，使用 BMS 屏幕定义源的名称（对于 Put 和 Get 样本程序不可用），但是省略该名称的最后一个字符。
2. 执行在构建任何 CICS IBM MQ for z/OS 应用程序时将会执行的相同步骤。 这些步骤列出在第 938 页的『在 z/OS 中构建 CICS 应用程序』中。 要使用的库成员在第 1065 页的表 186、第 1065 页的表 187、第 1066 页的表 188 和 第 1066 页的表 189 中列示。

   或者，在我们提供了可执行形式的样本时，您可以从 thlqual.SCSQCICS 装入库运行该样本。
3. 通过更新 CICS 系统定义 (CSD) 数据集识别 CICS 的映射集、程序和事务。 您需要的定义在成员 **thlqual**SCSQPROC (CSQ4S100) 中。 有关如何执行此操作的指导信息，请参阅 CICS Transaction Server

for z/OS 4.1 产品文档中的 *CICS-IBM MQ Adapter* 部分: CICS Transaction Server for z/OS 4.1， The CICS-IBM MQ adapter。

注: 对于“信用检查”样本应用程序，如果您没有创建该样本使用的 VSAM 数据集，将在此阶段获得错误消息。

4. 对于“信用检查”和“邮件管理器”样本应用程序，确保您的系统上提供它们使用的队列。 对于“信用检查”样本，它们是在成员 **thlqual**.SCSQPROC(CSQ4CVB)（对于 COBOL 语言）和 **thlqual**.SCSQPROC(CSQ4CCB)（对于 C 语言）中定义的。 对于“邮件管理器”样本，它们是在成员 **thlqual**.SCSQPROC(CSQ4CVD) 中定义的。 要确保这些队列始终可用，您可以将这些成员添加到 CSQINP2 初始化输入数据集，或者使用 CSQUTIL 程序装入这些队列定义。

   对于“队列属性”样本应用程序，您可以使用为其他样本应用程序提供的一个或多个队列。 或者，您可以使用自己的队列。 但是，在提供它的表单中，此样本仅与名称前八个字符为 CSQ4SAMP 的队列一起运行。

z/OS *z/OS 上样本 CICS 应用程序的名称*

本主题提供为样本 CICS 应用程序提供的程序摘要。

下表对 CICS 应用程序进行了概述:

- 第 1065 页的表 186 Put 和 Get 样本
- 第 1065 页的表 187 队列属性样本
- 第 1066 页的表 188 Mail Manager 样本（仅 COBOL）
- 第 1066 页的表 189 Credit Check 样本
- 第 1067 页的表 190 异步 Consumption 和 Publish/Subscribe 样本

表 *186: CICS Put* 和 *Get* 样本

成员名	适用语言	描述	库中提供的源文件	库中提供的可执行文件
CSQ4CCK1	C	Put 源程序	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Get 源程序	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Get 源程序	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Put 源程序	SCSQCOBS	SCSQCICS
CSQ4S100	独立	CICS 系统定义数据集	SCSQPROC	None

表 *187: CICS* 队列属性样本

成员名	适用语言	描述	库中提供的源文件	库中提供的可执行文件
CSQ4CVC1	COBOL	源程序	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	消息定义	SCSQCOBC	None
CSQ4VCMS	COBOL	BMS 屏幕定义	SCSQMAPS	SCSQCICS（命名为 CSQ4ACM）
CSQ4CAC1	汇编程序	源程序	SCSQASMS	SCSQCICS
CSQ4AMSG	汇编程序	消息定义	SCSQMACS	None
CSQ4ACMS	汇编程序	BMS 屏幕定义	SCSQMAPS	SCSQCICS（命名为 CSQ4ACM）
CSQ4CCC1	C	源程序	SCSQC37S	SCSQCICS

表 *187: CICS* 队列属性样本 *(*继续*)*				
成员名	**适用语言**	**描述**	**库中提供的源文件**	**库中提供的可执行文件**
CSQ4CMSG	C	消息定义	SCSQC370	None
CSQ4CCMS	C	BMS 屏幕定义	SCSQMAPS	SCSQCICS（命名为 CSQ4ACM）
CSQ4S100	独立	CICS 系统定义数据集	SCSQPROC	None

表 *188: CICS Mail Manager* 样本（仅 *COBOL*）		
成员名	**描述**	**库中提供的源文件**
CSQ4CVD	IBM MQ for z/OS 对象定义	SCSQPROC
CSQ4CVD1	菜单程序的源	SCSQCOBS
CSQ4CVD2	获取邮件程序的源	SCSQCOBS
CSQ4CVD3	显示消息程序的源	SCSQCOBS
CSQ4CVD4	发送邮件程序的源	SCSQCOBS
CSQ4CVD5	昵称程序的源	SCSQCOBS
CSQ4VDMS	BMS 屏幕定义源	SCSQMAPS
CSQ4S100	CICS 系统定义数据集	SCSQPROC
CSQ4VD0	数据定义	SCSQCOBC
CSQ4VD3	数据定义	SCSQCOBC
CSQ4VD4	数据定义	SCSQCOBC

表 *189: CICS Credit Check* 样本			
成员名	**适用语言**	**描述**	**库中提供的源文件**
CSQ4CVB	独立	IBM MQ 对象定义	SCSQPROC
CSQ4CCB	独立	IBM MQ 对象定义	SCSQPROC
CSQ4CVB1	COBOL	用户界面程序的源	SCSQCOBS
CSQ4CVB2	COBOL	信用应用程序管理器的源	SCSQCOBS
CSQ4CVB3	COBOL	支票帐户程序的源	SCSQCOBS
CSQ4CVB4	COBOL	分发程序的源	SCSQCOBS
CSQ4CVB5	COBOL	机构查询程序的源	SCSQCOBS
CSQ4CCB1	C	用户界面程序的源	SCSQC37S
CSQ4CCB2	C	信用应用程序管理器的源	SCSQC37S
CSQ4CCB3	C	支票帐户程序的源	SCSQC37S
CSQ4CCB4	C	分发程序的源	SCSQC37S
CSQ4CCB5	C	机构查询程序的源	SCSQC37S
CSQ4CB0	C	包含文件	SCSQC370

表 189: CICS Credit Check 样本 (继续)			
成员名	**适用语言**	**描述**	**库中提供的源文件**
CSQ4CBMS	C	BMS 屏幕定义源	SCSQMAPS
CSQ4VBMS	COBOL	BMS 屏幕定义源	SCSQMAPS
CSQ4VB0	COBOL	数据定义	SCSQCOBC
CSQ4VB1	COBOL	数据定义	SCSQCOBC
CSQ4VB2	COBOL	数据定义	SCSQCOBC
CSQ4VB3	COBOL	数据定义	SCSQCOBC
CSQ4VB4	COBOL	数据定义	SCSQCOBC
CSQ4VB5	COBOL	数据定义	SCSQCOBC
CSQ4VB6	COBOL	数据定义	SCSQCOBC
CSQ4VB7	COBOL	数据定义	SCSQCOBC
CSQ4VB8	COBOL	数据定义	SCSQCOBC
CSQ4BAQ	独立	VSAM 数据集的源	SCSQPROC
CSQ4FILE	独立	构建 CSQ4CVB3 使用的 VSAM 数据集的 JCL	SCSQPROC
CSQ4S100	独立	CICS 系统定义数据集	SCSQPROC

表 190: CICS 异步 Consumption 和 Publish/Subscribe 样本		
成员名	**描述**	**库中提供的源文件**
CSQ4CVCN	简单消息使用程序的源	SCSQCOBS
CSQ4CVCT	控制消息使用程序的源	SCSQCOBS
CSQ4CVEV	事件处理程序的源	SCSQCOBS
CSQ4CVPT	消息放入客户机程序的源	SCSQCOBS
CSQ4CVRG	注册客户机程序的源	SCSQCOBS
CSQ4S100	CICS 系统定义数据集	SCSQPROC

z/OS 在 z/OS 上为 IMS 环境准备样本应用程序

Credit Check 样本应用程序的一部分可以在 IMS 环境中运行。

要准备应用程序的此部分以通过 CICS 样本来运行，请首先执行第 1064 页的『在 z/OS 上为 CICS 环境准备样本应用程序』中描述的步骤。

然后，执行以下步骤：

1. 执行在构建任何 IMS IBM MQ for z/OS 应用程序时将会执行的相同步骤。 这些步骤列出在第 939 页的『构建 IMS（BMP 或 MPP）应用程序』中。 要使用的库成员列出在第 1068 页的表 191 中。
2. 向 IMS 识别应用程序和数据库。 样本随附于 PSBGEN、DBDGEN、ACB 定义、IMSGEN 和 IMSDALOC 语句以支持此识别。
3. 通过定制和运行为此目的提供的样本 JCL (CSQ4ILDB) 来装入数据库 CSQ4CA。 此 JCL 使用文件 CSQ4BAQ 中的数据来装入数据库。 使用针对数据库 CSQ4CA 的 DD 语句更新 IMS 控制区域。
4. 通过定制和运行为此目的提供的样本 JCL 将支票帐户程序作为成批消息处理 (BMP) 程序启动。 此 JCL 启动面向批处理的 BMP 程序。 要将程序作为面向消息的 BMP 程序运行，请从包含 IN= 语句的 JCL 中的行移除注释字符。

z/OS *z/OS 上样本 IMS 应用程序的名称*

此信息提供包含为 Credit Check 样本 IMS 应用程序提供的源和 JCL 列表的表。

表 *191: Credit Check IMS* 样本（仅 *C*）的源和 *JCL*		
成员名	**描述**	**在库中提供**
CSQ4CVB	IBM MQ 对象定义	SCSQPROC
CSQ4ICB3	支票帐户程序的源	SCSQC37S
CSQ4ICBL	用于装入支票帐户数据库的源	SCSQC37S
CSQ4CBI	数据定义	SCSQC370
CSQ4PSBL	数据库装入程序的 PSBGEN JCL	SCSQPROC
CSQ4PSB3	支票帐户程序的 PSBGEN JCL	SCSQPROC
CSQ4DBDS	数据库 CSQ4CA 的 DBDGEN JCL	SCSQPROC
CSQ4GIMS	CSQ4IVB3 和 CSQ4CA 的 IMSGEN 宏定义	SCSQPROC
CSQ4ACBG	CSQ4IVB3 的应用程序控制块 (ACB) 定义	SCSQPROC
CSQ4BAQ	数据库的源	SCSQPROC
CSQ4ILDB	数据库装入作业的样本运行 JCL	SCSQPROC
CSQ4ICBR	支票帐户程序的样本运行 JCL	SCSQPROC
CSQ4DYNA	数据库的 IMSDALOC 宏定义	SCSQPROC

z/OS *z/OS 上的 Put 样本*

Put 样本程序使用 MQPUT 调用将消息放在队列上。

源程序在批处理和 CICS 环境中采用 C 和 COBOL 形式提供（请参阅第 1061 页的表 179 和第 1065 页的表 186）。

Put 样本的设计

通过程序逻辑的流程为：

1. 使用 MQCONN 调用连接到队列管理器。 如果此调用失败，那么打印完成代码和原因码并停止处理。

   注: 如果您是在 CICS 环境中运行样本，那么无需发出 MQCONN 调用；如果发出调用，那么将返回 DEF_HCONN。 可以对后续的 MQI 调用使用连接句柄 MQHC_DEF_HCONN。
2. 使用带有 MQOO_OUTPUT 选项的 MQOPEN 调用打开队列。 在此调用的输入中，程序使用步骤 第 1070 页的『1』 中返回的连接句柄。 对于对象描述符结构 (MQOD)，它对所有字段使用缺省值，队列名称字段除外（该字段作为参数传递到程序）。 如果 MQOPEN 调用失败，那么打印完成代码和原因码并停止处理。
3. 在发出 MQPUT 调用的程序内创建循环，直至将所需数量的消息放在队列上。 如果 MQPUT 调用失败，那么会提前放弃循环，不再尝试其他 MQPUT 调用，并且返回完成代码和原因码。
4. 使用带有步骤 第 1070 页的『2』 中返回的对象句柄的 MQCLOSE 调用关闭队列。 如果此调用失败，那么打印完成代码和原因码。
5. 使用带有步骤 第 1070 页的『1』 中返回的连接句柄的 MQDISC 调用与队列管理器断开连接。 如果此调用失败，那么打印完成代码和原因码。

   注: 如果您是在 CICS 环境中运行样本，那么无需发出 MQDISC 调用。

z/OS *z/OS 上批处理环境的 Put 样本*

在考虑批处理环境的 Put 样本时，请使用本主题。

要运行样本，请编辑和运行样本 JCL，如第 1060 页的『在 z/OS 上为批处理环境准备和运行样本应用程序』中所述。

程序在 EXEC PARM 中采用以下参数（在 C 中以空格分隔，在 COBOL 中以逗号分隔）：

1. 队列管理器的名称（4 个字符）
2. 目标队列的名称（48 个字符）
3. 消息数（最多 4 位数）
4. 要在消息中写入的填充字符（1 个字符）
5. 要在消息中写入的字符数（最多 4 位数）
6. 消息的持久性（1 个字符：P 表示持久，N 表示非持久）

如果错误地输入其中任何参数，那么您将收到相应的错误消息。

来自样本的任何消息都写入到 SYSPRINT 数据集。

使用说明

- 为保持样本简单，在语言版本之间存在一些细小功能差异。 但是，如果使用样本运行 JCL、CSQ4BCJR 和 CSQ4BVJR 中显示的参数的布局，那么这些差异最小。 没有任何差异与 MQI 相关。
- CSQ4BCK1 允许针对发送的消息数和消息长度输入四位以上的数字。
- 对于两个数字字段，请输入范围在 1 到 9999 内的任何数字。 输入的值应为正数。 例如，要放置单条消息，可以输入 1、01、001 或 0001 作为值。 如果输入非数字值或负值，那么您可能会收到错误。 例如，如果输入 -1，那么 COBOL 程序发送单字节消息，但是 C 程序收到错误。
- 对于程序 CSQ4BCK1 和 CSQ4BVK1，如果希望消息为持久性消息，那么必须在持久性参数 ++PER++ 中输入 P。 如果未能这些做，那么消息将是非持久性消息。

z/OS *z/OS 上 CICS 环境的 Put 样本*

在考虑 CICS 环境的 Put 样本时，请使用本主题。

事务采用以下参数，以逗号分隔：

1. 消息数（最多 4 位数）
2. 要在消息中写入的填充字符（1 个字符）
3. 要在消息中写入的字符数（最多 4 位数）
4. 消息的持久性（1 个字符：P 表示持久，N 表示非持久）
5. 目标队列的名称（48 个字符）

如果错误地输入其中任何参数，那么您将收到相应的错误消息。

对于 COBOL 样本，通过输入以下内容来调用 CICS 环境中的 Put 样本：

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

对于 C 样本，通过输入以下内容来调用 CICS 环境中的 Put 样本：

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

来自样本的任何消息都会显示在屏幕上。

使用说明

- 为保持样本简单，在语言版本之间存在一些细小功能差异。 没有任何差异与 MQI 相关。

- 如果输入长度超过 48 个字符的队列名称，那么其长度会截断为最多 48 个字符，而不返回任何错误消息。
- 在输入事务之前，请按 CLEAR 键。
- 对于两个数字字段，请输入范围在 1 到 9999 内的任何数字。 输入的值应为正数。 例如，要放置单条消息，可以输入值 1、01、001 或 0001。 如果输入非数字值或负值，那么您可能会收到错误。 例如，如果输入 -1，那么 COBOL 程序发送单字节消息，但是 C 程序由于 malloc() 出错而异常终止。
- 对于程序 CSQ4CCK1 和 CSQ4CVK1，如果希望消息为持久性消息，请在持久性参数中输入 P。 对于非持久性消息，在持久性参数中输入 N。 如果输入任何其他值，那么您将收到错误消息。
- 消息放入同步点中，因为缺省值用于所有参数，程序调用期间设置的参数除外。

z/OS *z/OS* 上的 *Get* 样本

Get 样本程序使用 MQGET 调用从队列中获取消息。

源程序在批处理和 CICS 环境中采用 C 和 COBOL 形式提供（请参阅第 1061 页的表 179 和第 1065 页的表 186）。

z/OS *z/OS* 上 *Get* 样本的设计

了解有关 Get 样本的设计以及要考虑的一些使用说明。

通过程序逻辑的流程为：

1. 使用 MQCONN 调用连接到队列管理器。 如果此调用失败，那么打印完成代码和原因码并停止处理。

   注： 如果您是在 CICS 环境中运行样本，那么无需发出 MQCONN 调用；如果发出调用，那么将返回 DEF_HCONN。 可以对后续的 MQI 调用使用连接句柄 MQHC_DEF_HCONN。
2. 使用带有 MQOO_INPUT_SHARED 和 MQOO_BROWSE 选项的 MQOPEN 调用打开队列。 在此调用的输入中，程序使用步骤 第 1070 页的『1』 中返回的连接句柄。 对于对象描述符结构 (MQOD)，它对所有字段使用缺省值，队列名称字段除外（该字段作为参数传递到程序）。 如果 MQOPEN 调用失败，那么打印完成代码和原因码并停止处理。
3. 在发出 MQGET 调用的程序内创建循环，直至从队列检索所需数量的消息。 如果 MQGET 调用失败，那么会提前放弃循环，不再尝试其他 MQGET 调用，并且返回完成代码和原因码。 在 MQGET 调用中指定了以下选项：

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT 或 MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST 和 MQGMO_BROWSE_NEXT

   有关这些选项的描述，请参阅 MQGET。 对于各消息，将会打印消息号，后跟消息长度和消息数据。
4. 使用带有步骤 第 1070 页的『2』 中返回的对象句柄的 MQCLOSE 调用关闭队列。 如果此调用失败，那么打印完成代码和原因码。
5. 使用带有步骤 第 1070 页的『1』 中返回的连接句柄的 MQDISC 调用与队列管理器断开连接。 如果此调用失败，那么打印完成代码和原因码。

   注： 如果您是在 CICS 环境中运行样本，那么无需发出 MQDISC 调用。

使用说明

- 为保持样本简单，在语言版本之间存在一些细小功能差异。 但是，如果使用样本运行 JCL、CSQ4BCJR 和 CSQ4BVJR 中显示的参数的布局，那么这些差异最小。 没有任何差异与 MQI 相关。
- CSQ4BCJ1 允许针对检索的消息数输入四位以上的数字。
- 长度超过 64 KB 的消息会截断。
- CSQ4BCJ1 只能正确显示字符消息，因为它仅显示直至出现第一个 NULL (\0) 字符。
- 对于数字型消息数字段，请输入范围在 1 到 9999 内的任何数字。 输入的值应为正数。 例如，要获取单条消息，可以输入 1、01、001 或 0001 作为值。 如果输入非数字值或负值，那么您可能会收到错误。 例如，如果输入 -1，那么 COBOL 程序检索一条消息，但是 C 程序不检索任何消息。

- 对于程序 CSQ4BCJ1 和 CSQ4BVJ1，如果要浏览消息，请在获取参数 ++GET++ 中输入 B。
- 对于程序 CSQ4BCJ1 和 CSQ4BVJ1，请在同步点参数 ++SYNC++ 中输入 S 以在同步点中检索消息。

z/OS *z/OS 上针对批处理环境的 Get 样本*

要运行样本，请编辑和运行样本 JCL，如第 1060 页的『在 z/OS 上为批处理环境准备和运行样本应用程序』中所述。

程序在 EXEC PARM 中采用以下参数（在 C 中以空格分隔，在 COBOL 中以逗号分隔）：

1. 队列管理器的名称（4 个字符）
2. 目标队列的名称（48 个字符）
3. 要获取的消息数（最多 4 位数）
4. 浏览/获取消息选项（1 个字符：B 表示浏览，D 表示以中断性方式获取消息）
5. 同步点控制（1 个字符：S 表示同步点，N 表示无同步点）

如果错误地输入其中任何参数，那么您将收到相应的错误消息。

来自样本的输出写入到 SYSPRINT 数据集：

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *z/OS 上 CICS 环境的 Get 样本*

针对 CICS 环境的 Get 样本的特殊注意事项。

事务在 EXEC PARM 中采用以下参数，以逗号分隔：

1. 要获取的消息数（最多四位数）
2. 浏览/获取消息选项（一个字符： B 表示浏览，D 表示以中断性方式获取消息）
3. 同步点控制（一个字符：S 表示同步点，N 表示无同步点）
4. 目标队列的名称（48 个字符）

如果错误地输入其中任何参数，那么您将收到相应的错误消息。

对于 COBOL 样本，通过输入以下内容来调用 CICS 环境中的 Get 样本：

```
MVGT,9999,B,S,QUEUE.NAME
```

对于 C 样本，通过输入以下内容来调用 CICS 环境中的 Get 样本：

```
MCGT,9999,B,S,QUEUE.NAME
```

从队列检索消息时，消息放在与 CICS 事务具有相同名称（例如，对于 C 样本为 MCGT）的 CICS 临时存储队列上。

以下是 Get 样本的示例输出：

```
*************************** TOP OF QUEUE ************************
```

```
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE ***********************
```

使用说明

- 为保持样本简单，在语言版本之间存在一些细小功能差异。 没有任何差异与 MQI 相关。
- 如果输入长度超过 48 个字符的队列名称，那么其长度会截断为最多 48 个字符，而不返回任何错误消息。
- 在输入事务之前，请按 CLEAR 键。
- CSQ4CCJ1 只能正确显示字符消息，因为它仅显示直至出现第一个 NULL (\0) 字符。
- 对于数字字段，请输入范围在 1 到 9999 内的任何数字。 输入的值应为正数。 例如，要获取单条消息，可以输入值 1、01、001 或 0001。 如果输入非数字值或负值，那么您可能会收到错误。
- 长度超过 24,526 字节（在 C 中）和 9,950 字节（在 COBOL 中）的消息会截断。 这是由于 CICS 临时存储队列的使用方式造成。
- 对于程序 CSQ4CCK1 和 CSQ4CVK1，如果要浏览消息，请在 get 参数中输入 B，否则请输入 D。 这将执行中断性 MQGET 调用。 如果输入任何其他值，那么您将收到错误消息。
- 对于程序 CSQ4CCJ1 和 CSQ4CVJ1，请在同步点参数中输入 S 以在同步点中检索消息。 如果在同步点参数中输入 N，那么将在同步点外发出 MQGET 调用。 如果输入任何其他值，那么您将收到错误消息。

z/OS *z/OS 上的 Browse 样本*

Browse 样本是一个批处理应用程序，其演示了如何使用 MQGET 调用来浏览队列中的消息。

应用程序分步处理队列中的所有消息，打印各消息的前 80 个字节。 可以使用此应用程序来查看队列上的消息而不对其进行更改。

源程序和样本运行 JCL 以 COBOL、汇编程序、PL/I 和 C 语言形式提供（请参阅第 1061 页的表 180）。

要启动应用程序，请编辑和运行样本运行 JCL，如第 1060 页的『在 z/OS 上为批处理环境准备和运行样本应用程序』中所述。 您可以通过在运行 JCL 中指定队列的名称来查看自己其中一个队列上的消息。

运行应用程序（并且队列上有一些消息）时，输出数据集如下所示：

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

如果队列上没有消息，那么数据集仅包含标题和 `End of report` 消息。 如果任何 MQI 调用发生错误，那么会将完成代码和原因码添加到输出数据集。

z/OS *z/OS 上 Browse 样本的设计*

Browse 样本应用程序使用单个程序模块；针对每种受支持的编程语言都提供了一个模块。

通过程序逻辑的流程为：

1. 打开打印数据集并打印报告的标题行。 检查队列管理器和队列的名称是否已从运行 JCL 进行传递。 如果两个名称均已传递，那么打印包含名称的报告行。 如果未传递，那么打印错误消息，关闭打印数据集并停止处理。

程序测试从 JCL 传递的参数的方式取决于程序的编写语言；有关更多信息，请参阅第 1073 页的『z/OS 上与语言相关的设计注意事项』。

2. 使用 MQCONN 调用连接到队列管理器。 如果此调用不成功，那么打印完成代码和原因码，关闭打印数据集并停止处理。
3. 使用带有 MQOO_BROWSE 选项的 MQOPEN 调用打开队列。 在此调用的输入中，程序使用步骤第 1073 页的『2』中返回的连接句柄。 对于对象描述符结构 (MQOD)，它对所有字段使用缺省值，队列名称字段除外（该字段在步骤第 1072 页的『1』中传递）。 如果此调用不成功，那么打印完成代码和原因码，关闭打印数据集并停止处理。
4. 使用 MQGET 调用浏览队列上的第一条消息。 在此调用的输入中，程序指定：
   - 来自步骤第 1073 页的『2』和第 1073 页的『3』的连接句柄和队列句柄
   - 所有字段都设置为其初始值的 MQMD 结构
   - 两个选项：
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - 大小为 80 字节的缓冲区，用于存放从消息复制的数据

   MQGMO_ACCEPT_TRUNCATED_MSG 选项允许调用完成，即使消息长度超过调用中指定的 80 字节缓冲区也如此。 如果消息长度超过缓冲区，那么消息会截断以适合缓冲区大小，并且完成代码和原因码设置为显示此大小。 已设计样本，以便消息截断为 80 个字符，以使报告易读。 缓冲区大小由 DEFINE 语句设置，因此在需要的情况下可以轻松对其进行更改。
5. 执行以下循环，直至 MQGET 调用失败：
   a. 打印显示以下内容的报告行：
      - 消息的序列号（这是浏览操作的计数）。
      - 消息的实际长度（不是已截断的长度）。 该值在 MQGET 调用的 DataLength 字段中返回。
      - 消息数据的前 80 个字节。

   b. 将 MQMD 结构的 MsqId 和 CorrelId 字段重置为空值
   c. 使用带有以下两个选项的 MQGET 调用浏览下一条消息：
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. 如果 MQGET 调用失败，请测试原因码以查看调用是否失败，因为浏览光标已到达队列结尾。 在此情况下，打印 End of report 消息并转至步骤第 1073 页的『7』；否则，打印完成代码和原因码，关闭打印数据集并停止处理。
7. 使用带有步骤第 1073 页的『3』中返回的对象句柄的 MQCLOSE 调用关闭队列。
8. 使用带有步骤第 1073 页的『2』中返回的连接句柄的 MQDISC 调用与队列管理器断开连接。
9. 关闭打印数据集并停止处理。

z/OS *z/OS 上与语言相关的设计注意事项*

为“浏览”样本提供了四个编程语言版本的源模块。

在源模块之间主要有两个差异：

- 在测试从运行 JCL 传递的参数时，COBOL、PL/I 和汇编语言模块会搜索逗号字符 (,)。如果 JCL 传递 PARM=(,LOCALQ1)，应用程序会尝试在缺省队列管理器上打开队列 LOCALQ1。 如果在逗号（或无逗号）之后没有名称，那么应用程序会返回错误。 C 模块不会搜索逗号字符。 如果 JCL 传递单个参数（例如，PARM=('LOCALQ1')），那么 C 模块将使用此参数作为缺省队列管理器上的队列名称。
- 为保持汇编语言模块简单明了，它在创建打印报告时使用以下日期格式：*yy/ddd*（例如，05/116）。 其他模块则使用 *mm/dd/yy* 格式的日历日期。

z/OS *z/OS 上的 Print Message 样本*

Print Message 样本是一个批处理应用程序，其演示了如何使用 MQGET 调用从队列中移除所有消息。

Print Message 样本使用三个参数：

1. 队列管理器的名称
2. 源队列的名称
3. 属性的可选参数

它还针对各消息打印消息描述符的字段，后跟消息数据。 程序以十六进制和字符形式（如果这些字符可打印）打印数据。 如果字符不可打印，那么程序会将其替换为句点字符 (.)。您可以在诊断用于将消息放入队列的应用程序的问题时使用该程序。

属性参数的允许的值为：

表 *192:* 属性参数允许的值

值	**行为**
0	缺省行为，如同针对 IBM WebSphere MQ 6 的行为。 传送到应用程序的属性取决于从中检索消息的 **PropertyControl** 队列属性。

1	创建消息句柄并将其用于 MQGET。 通过与消息描述符类似的方式显示消息的属性，消息描述符（或扩展名）中包含的属性除外。 例如：
	`****Message properties****`
	`  property name: property value`
	或者，如果没有属性可用：
	`****Message properties****`
	`  None`
	数字值使用 `printf` 进行显示，字符串值用单引号引起来，字节字符串使用 X 和单引号引起来，如同针对消息描述符一样。

2	指定 MQGMO_NO_PROPERTIES，以便将仅返回消息描述符属性。
3	指定 MQGMO_PROPERTIES_FORCE_MQRFH2，以便在消息数据中返回所有属性。
4	指定 MQGMO_PROPERTIES_COMPATIBILITY，以便可以返回所有属性（具体取决于是否包含 IBM WebSphere MQ 6 属性），否则废弃属性。

您可以更改应用程序，以便其浏览消息，而不是从队列中移除这些消息。 要执行此操作，请使用选项 -DBROWSE 进行编译以定义 BROWSE 宏，如第 1075 页的『z/OS 上 Print Message 样本的设计』中所指示。 在 SCSQLOAD 库中为您提供了可执行代码。 模块 CSQ4BCG0 使用 -DBROWSE 进行构建；模块 CSQ4BCG1 以中断性方式读取队列。

应用程序具有采用 C 语言编写的单个源程序。 此外，还提供了样本运行 JCL 代码（请参阅第 1062 页的表 181）。

要启动应用程序，请编辑和运行样本运行 JCL，如第 1060 页的『在 z/OS 上为批处理环境准备和运行样本应用程序』中所述。 运行应用程序（并且队列上有一些消息）时，输出数据集如第 1075 页的图 144 中所示。

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR   '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                 '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '   '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

图 *144:* 来自 *Print Message* 样本应用程序的报告示例

z/OS *z/OS 上 Print Message 样本的设计*

Print Message 样本应用程序使用用 C 语言编写的单个程序。

通过程序逻辑的流程为：

1. 检查队列管理器和队列的名称是否已从运行 JCL 进行传递。 如果尚未传递，那么打印错误消息并停止处理。
2. 使用 MQCONN 调用连接到队列管理器。 如果此调用不成功，那么打印完成代码和原因码并停止处理；否则打印队列管理器的名称。
3. 使用带有 MQOO_INPUT_SHARED 选项的 MQOPEN 调用打开队列。

   注: 如果想要应用程序浏览消息而不是从队列中移除这些消息，请使用 -DBROWSE 来编译样本，或者在源代码的开头添加 `#define BROWSE`。 执行此操作时，宏预处理器在程序中添加用于选择编译中的 MQOO_BROWSE 选项的行。

在此调用的输入中，程序使用步骤 第 1075 页的『2』 中返回的连接句柄。 对于对象描述符结构 (MQOD)，它对所有字段使用缺省值，队列名称字段除外（该字段在步骤 第 1075 页的『1』 中传递）。 如果此调用不成功，那么打印完成代码和原因码并停止处理；否则，打印队列的名称。

4. 如果使用消息句柄来获取消息属性，那么使用 MQCRTMH 创建此类句柄以用于后续 MQGET 调用。 如果此调用不成功，那么打印完成代码和原因码并停止处理。
5. 将获取消息选项设置为反映任何消息属性的请求操作。
6. 执行以下循环，直至 MQGET 调用失败：
   a. 将缓冲区初始化为空白，以便消息数据不会被缓冲区中已有的任何数据损坏。
   b. 将 MQMD 结构的 `MsgId` 和 `CorrelId` 字段设置为空值，以便 MQGET 调用选择队列中的第一条消息。
   c. 使用 MQGET 调用从队列取出消息。 在此调用的输入中，程序指定：
      - 来自步骤 第 1075 页的『2』 和 第 1075 页的『3』 的连接句柄和对象句柄。
      - 将所有字段设置为其初始值的 MQMD 结构。 (对于每个 MQGET 调用，将 `MsgId` 和 `CorrelId` 重置为 null。)
      - 选项 MQGMO_NO_WAIT。

        注: 如果想要应用程序浏览消息而不是从队列中移除这些消息，请使用 -DBROWSE 来编译样本，或者在源代码的开头添加 `#define BROWSE`。 执行此操作时，宏预处理器在程序中添加用于选择即将编译的 MQGMO_BROWSE_NEXT 选项的行。 在针对队列（先前没有为其将浏览光标与当前对象句柄结合使用）的调用中使用此选项时，浏览光标在逻辑上位于第一条消息之前。
      - 大小为 64KB 的缓冲区，用于存放从消息复制的数据。
   d. 调用 printMD 子例程。 这将打印消息描述符中各字段的名称，后跟其内容。
   e. 如果在步骤 第 1076 页的『4』 中创建了消息句柄，那么调用 printProperties 子例程以显示任何消息属性。
   f. 打印消息的长度，后跟消息数据。 各行消息数据采用以下格式：
      - 此部分的数据的相对位置（十六进制形式）
      - 16 字节的十六进制数据
      - 同为 16 字节的字符格式数据，前提是它可打印（不可打印字符替换为句点）
7. 如果 MQGET 调用失败，那么测试原因码以查看调用是否失败，因为队列上再没有更多消息。 在此情况下，打印以下消息：No more messages；否则，打印完成代码和原因码。 在两种情况下，均转至步骤 第 1076 页的『9』。

   注: 如果 MQGET 调用找到具有超过 64KB 数据的消息，那么该调用失败。 要将程序更改为处理更大的消息，那么可以执行以下操作之一：
   - 将 MQGMO_ACCEPT_TRUNCATED_MSG 选项添加到 MQGET 调用，以便该调用获取前 64KB 数据并废弃其余部分
   - 在程序找到具有此数据量的消息时，使其将消息保留在队列上
   - 增大缓冲区的大小
8. 如果在步骤 第 1076 页的『4』 中创建了消息句柄，那么调用 MQDLTMH 以将其删除。
9. 使用带有步骤 第 1075 页的『3』 中返回的对象句柄的 MQCLOSE 调用关闭队列。
10. 使用带有步骤 第 1075 页的『2』 中返回的连接句柄的 MQDISC 调用与队列管理器断开连接。

z/OS *z/OS 上的 Queue Attributes 样本*

Queue Attributes 样本是演示 MQINQ 和 MQSET 调用的用法的会话式 CICS 应用程序。

它显示如何查询队列的 **InhibitPut** 和 **InhibitGet** 属性的值，以及如何对其进行更改，以便程序无法将消息放在队列上或从中获取消息。 在测试程序时，您可能想要以此方式锁定队列。

为防止意外干预您自己的队列，此样本仅在其名称的前 8 个字节中具有字符 CSQ4SAMP 的队列对象上适用。 但是，源代码包括用于说明如何移除此限制的注释。

源程序以 COBOL、汇编程序和 C 语言形式提供（请参阅第 1065 页的表 187）。

样本的汇编语言版本使用可重新输入的代码。 为此，您将注意到该版本的样本中各 MQI 调用的代码包含 MF 关键字；例如：

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

（VL 关键字意味着可以使用 CICS 执行诊断设施 (CEDF) 提供的事务来调试程序。） 有关编写可重新输入的程序的更多信息，请参阅采用 System/390 汇编语言进行编码。

要启动应用程序，请启动 CICS 系统并使用以下 CICS 事务：

- 对于 COBOL，MVC1
- 对于汇编语言，MAC1
- 对于 C，MCC1

您可以通过更改步骤 3 中提及的 CSD 数据集来更改其中任何事务的名称。

样本的设计

在启动样本时，它显示具有以下各项的字段的屏幕图：

- 队列的名称
- 用户请求（有效操作为：inquire、allow 或 inhibit）
- 队列的放置操作的当前状态
- 队列的获取操作的当前状态

前两个字段用于用户输入。 最后两个字段由应用程序填充：它们显示字 INHIBITED 或 ALLOWED。

应用程序验证在前两个字段中输入的值。 它检查以字符 CSQ4SAMP 开头的队列名称以及是否在 Action 字段中输入了三个有效请求之一。 应用程序将所有输入都转换为大写，因此无法使用名称中包含小写字符的任何队列。

如果在 **Action** 字段中输入 `inquire`，那么通过程序逻辑的流程为：

1. 使用带有 MQOO_INQUIRE 选项的 MQOPEN 调用打开队列
2. 使用选择器 MQIA_INHIBIT_GET 和 MQIA_INHIBIT_PUT 调用 MQINQ
3. 使用 MQCLOSE 调用关闭队列
4. 分析 MQINQ 调用的 **IntAttrs** 参数中返回的属性，并且根据情况将字 INHIBITED 或 ALLOWED 移至相关屏幕字段

如果在 **Action** 字段中输入 `inhibit`，那么通过程序逻辑的流程为：

1. 使用带有 MQOO_SET 选项的 MQOPEN 调用打开队列
2. 使用选择器 MQIA_INHIBIT_GET 和 MQIA_INHIBIT_PUT 以及 **IntAttrs** 参数中的值 MQQA_GET_INHIBITED 和 MQQA_PUT_INHIBITED 调用 MQSET
3. 使用 MQCLOSE 调用关闭队列
4. 将字 INHIBITED 移至相关屏幕字段

如果在 **Action** 字段中输入 `allow`，那么应用程序执行类似于 inhibit 请求的处理。 唯一差异在于屏幕上显示的属性和字的设置。

当应用程序打开队列时，它会将缺省连接句柄用于队列管理器。（CICS 在启动 CICS 系统时会与队列管理器建立连接。） 应用程序可以在此阶段捕获以下错误：

- 应用程序未连接到队列管理器
- 队列不存在
- 用户未经授权访问队列
- 应用程序未经授权打开队列

对于其他 MQI 错误，应用程序显示完成代码和原因码。

z/OS z/OS 上的 *Mail Manager* 样本

Mail Manager 样本应用程序是演示在单个环境内和跨不同环境发送和接收消息的程序套件。 该应用程序是一个简单的电子邮件系统，允许用户交换消息，即使他们使用不同的队列管理器也是如此。

该应用程序演示如何使用 MQOPEN 调用以及通过将 IBM MQ for z/OS 命令放在系统命令输入队列上来创建队列。

提供了该应用程序的三个版本：

- 采用 COBOL 编写的 CICS 应用程序
- 采用 COBOL 编写的 TSO 应用程序
- 采用 C 编写的 TSO 应用程序

z/OS 在 *z/OS* 上准备“邮件管理器”样本

在两个环境内运行的版本中提供了邮件管理器。 在运行应用程序之前必须执行的准备取决于要使用的环境。

用户从 TSO 和 CICS 均可访问邮件队列和昵称队列，只要其登录用户标识在各系统上相同即可。

必须先设置与另一个队列管理器的消息通道，然后才能将消息发送到该队列管理器。 要执行此操作，请使用 IBM MQ 的通道控制功能，如通道控制功能中所述。

为 TSO 环境准备样本

请按照以下步骤操作：

1. 按第 1063 页的『在 z/OS 上为 TSO 环境准备样本应用程序』中所述准备样本。
2. 定制为样本提供的 CLIST 以定义下列各项：
   - 面板的位置
   - 消息文件的位置
   - 装入模块的位置
   - 要与应用程序结合使用的队列管理器的名称

   针对样本的各语言版本提供了单独的 CLIST：
   - 对于 COBOL 版本：CSQ4RVD1
   - 对于 C 版本：CSQ4RCD1
3. 确保应用程序所使用的队列在队列管理器上可用。 （队列定义在 CSQ4CVD 中。）

注: VS COBOL II 不支持通过 ISPF 实现多任务。 这意味着不能在拆分屏幕两侧均使用 Mail Manager 样本应用程序。 如果执行此操作，那么结果不可预测。

z/OS 在 *z/OS* 上运行“邮件管理器”样本

要在 CICS Transaction Server for z/OS 环境中启动样本，请运行事务 MAIL。 如果您尚未登录到 CICS，那么应用程序会提示输入可以向其发送邮件的用户标识。

在启动应用程序时，它会打开邮件队列。 如果此队列不存在，那么应用程序将为您创建一个队列。 邮件队列具有 CSQ4SAMP.MAILMGR. *userid*，其中 *userid* 取决于环境：

在 TSO 中
用户的 TSO 标识

在 CICS 中
用户的 CICS 登录标识或在启动邮件管理器时提示用户输入的用户标识

邮件管理器使用的队列名称的所有部分都必须为大写。

然后，应用程序提供具有下列各项的选项的菜单面板：

- 读取待收邮件

- 发送邮件
- 创建 昵称

菜单面板还显示邮件队列上正在等待的消息数。 各菜单选项显示进一步的面板：

读取待收邮件
邮件管理器显示位于邮件队列上的消息的列表。 （仅显示队列上的前 99 条消息。） 有关此面板的示例，请参阅第 1082 页的图 147。 从此列表选择消息时，将会显示消息的内容（请参阅第 1082 页的图 148）。

发送邮件
面板提示输入以下内容：

- 要向其发送消息的用户的名称
- 拥有邮件队列的队列管理器的名称
- 消息的文本

在用户名字段中，可以输入使用邮件管理器创建的用户标识或昵称。 如果用户的邮件队列由您使用的同一队列管理器拥有，那么可以将队列管理器名称字段保留为空白；如果在用户名字段中输入了昵称，那么必须将其保留为空白：

- 如果仅指定用户名，那么程序首先假设名称为昵称，并且将消息发送到由该名称定义的对象。 如果没有此类昵称，那么程序尝试将消息发送到具有该名称的本地队列。
- 如果指定用户名和队列管理器名称，那么程序将消息发送到由这两个名称定义的邮件队列。

例如，如果要将消息发送到远程队列管理器 QM12 上的用户 JONESM，那么可以通过以下任一方式向其发送消息：

- 使用两个字段在队列管理器 QM12 上指定用户 JONESM。
- 定义该用户的昵称（例如，MARY），并且通过将 MARY 放在用户名字段中而不在队列管理器名称字段中放入任何内容来向其发送消息。

创建 昵称
您可以定义在向频繁联系的其他用户发送消息时可以使用的容易记住的名称。 系统会提示您输入其他用户的用户标识以及拥有它们的邮件队列的队列管理器的名称。

昵称是具有 CSQ4SAMP.MAILMGR. *userid.nickname* 形式的名称的队列，其中 *userid* 是您自己的用户标识，*nickname* 是要使用的昵称。 利用以此方式构造的名称，用户可以各自具有一组自己的昵称。

程序创建的队列的类型取决于如何完成“创建昵称”面板的字段：

- 如果仅指定用户名，或者队列管理器名称与邮件管理器连接到的队列管理器的名称相同，那么程序将创建别名队列。
- 如果同时指定用户名和队列管理器名称（并且队列管理器不是邮件管理器连接到的队列管理器），那么程序将创建远程队列的本地定义。 程序不检查此定义解析为的队列存在与否，甚至远程队列管理器是否存在。

例如，如果您自己的用户标识为 SMITHK 并为用户 JONESM（使用远程队列管理器 QM12）创建名为 MARY 的昵称，那么昵称程序将创建名为 CSQ4SAMP.MAILMGR.SMITHK.MARY 的远程队列的本地定义。 此定义解析为 Mary 的邮件队列，在队列管理器 QM12 上为 CSQ4SAMP.MAILMGR.JONESM。 如果您是自行使用队列管理器 QM12，那么程序会改为创建具有相同名称 (CSQ4SAMP.MAILMGR.SMITHK.MARY) 的别名队列。

与 COBOL 版本相比，TSO 应用程序的 C 版本能够更充分地利用 ISPF 的消息处理功能。 您可能会注意到 C 和 COBOL 版本显示的错误消息不同。

z/OS *z/OS* 上“邮件管理器”样本的设计

以下部分描述了构成 Mail Manager 样本应用程序的各个程序。

应用程序与应用程序使用的面板之间的关系在 第 1080 页的图 145 (对于 TSO 版本) 和 第 1081 页的图 146 (对于 CICS Transaction Server for z/OS 版本) 中显示。

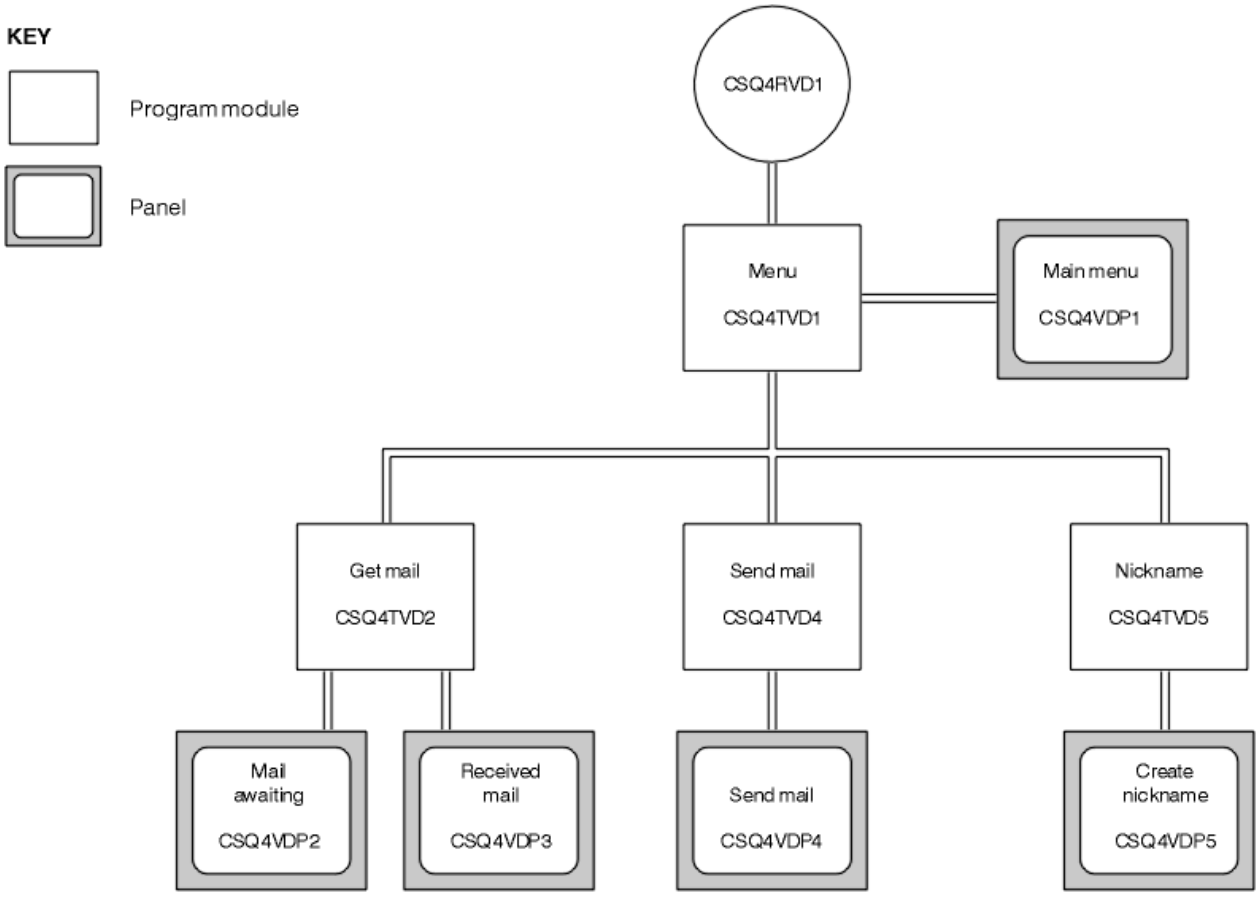


图 *145:* 邮件管理器的 *TSO* 版本的程序和面板

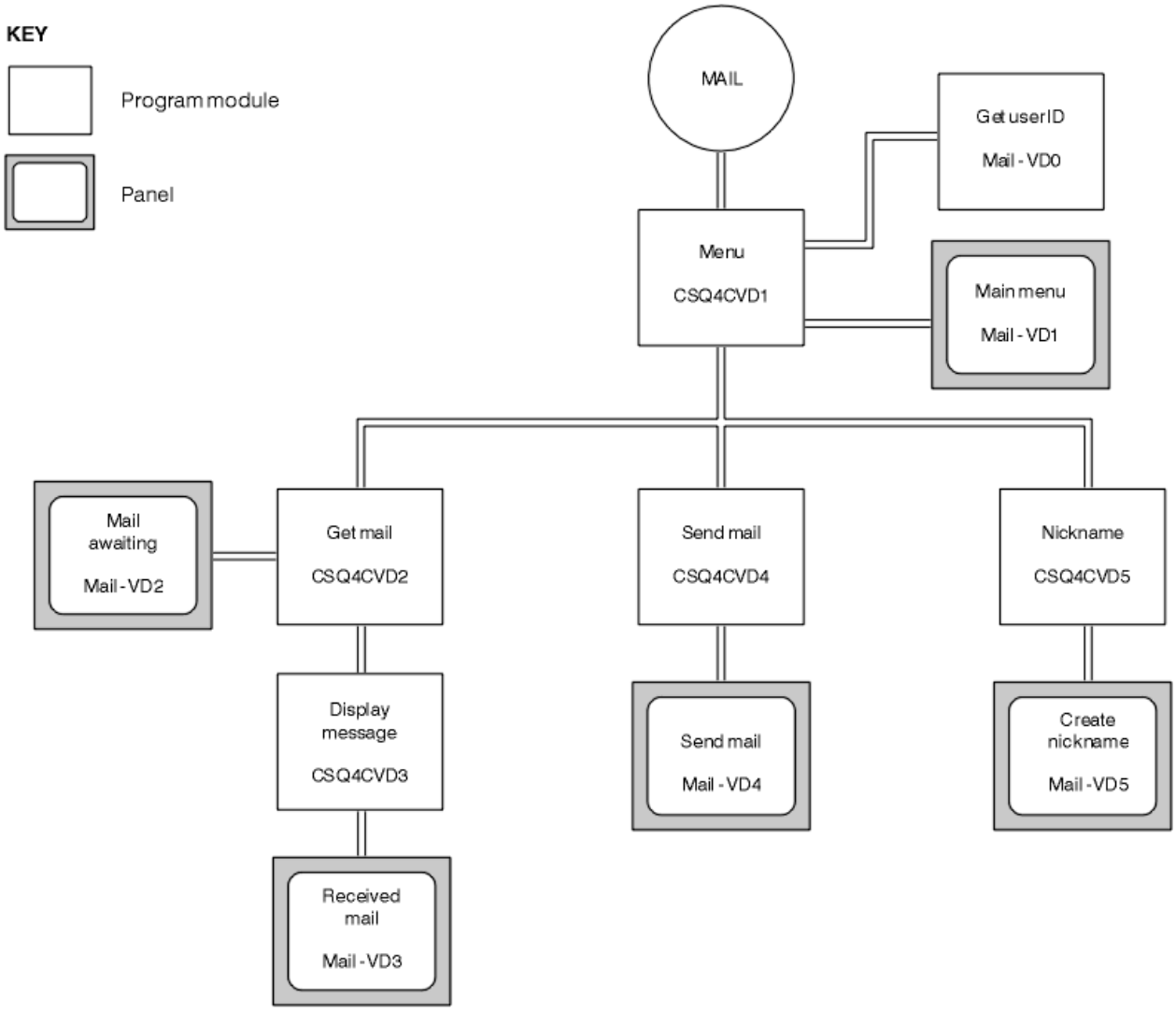


图 *146:* 邮件管理器的 *CICS* 版本的程序和面板

z/OS *z/OS* 上的菜单程序

在 TSO 环境中，菜单程序由 CLIST 调用。 在 CICS 环境中，此程序由事务 MAIL 调用。

菜单程序（对于 TSO 为 CSQ4TVD1，对于 CICS 为 CSQ4CVD1）是套件中的初始程序。 它显示菜单（对于 TSO 为 CSQ4VDP1，对于 CICS 为 VD1）并调用其他程序（当从菜单选择这些程序时）。

程序首先获取用户的标识：

- 在程序的 CICS 版本中，如果用户已登录到 CICS，那么通过使用 CICS 命令 ASSIGN USERID 来获取用户标识。 如果用户尚未登录，那么程序显示登录面板 (CSQ4VD0) 以提示用户输入用户标识。 在此程序内没有任何安全处理；用户可以给定任何用户标识。
- 在 TSO 版本中，从 CLIST 中的 TSO 获取用户的标识。 它作为 ISPF 共享池中的变量传递到菜单程序。

在程序获取用户标识后，它会检查以确保用户具有邮件队列 (CSQ4SAMP.MAILMGR. *userid*)。 如果邮件队列不存在，那么程序通过将消息放在系统命令输入队列上来创建邮件队列。 该消息包含 IBM MQ for z/OS 命令 DEFINE QLOCAL。 此命令使用的对象定义将队列的最大深度设置为 9999 条消息。

程序还创建临时动态队列以处理来自系统命令输入队列的回复。 为执行此操作，程序使用 MQOPEN 调用，将 SYSTEM.DEFAULT.MODEL.QUEUE 指定为动态队列的模板。 队列管理器使用具有前缀 CSQ4SAMP 的名称创建临时动态队列；名称的其余部分由队列管理器生成。

然后，程序打开用户的邮件队列并通过查询队列的当前深度来查找队列上的消息数。 为执行此操作，程序使用 MQINQ 调用，指定 MQIA_CURRENT_Q_DEPTH 选择器。

然后，程序执行显示菜单并处理用户进行的选择的循环。 当用户按 PF3 键时，将停止循环。 当进行了有效选择时，将启动相应的程序；否则显示错误消息。

z/OS *z/OS* 上的获取邮件程序和显示消息程序

在应用程序的 TSO 版本中，获取邮件和显示消息功能由同一程序 (CSQ4TVD2) 执行。 在应用程序的 CICS 版本中，这些函数由不同的程序（CSQ4CVD2 和 CSQ4CVD3）执行。

Mail Awaiting 面板（对于 TSO 为 CSQ4VDP2，对于 CICS 为 VD2；请参阅第 1082 页的图 147 以获取示例）显示位于用户的邮件队列上的所有消息。 为创建此列表，程序使用 MQGET 调用来浏览队列上的所有消息，保存有关各消息的信息。 除显示的信息以外，程序还记录各消息的 `MsgId` 和 `CorrelId`。

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

图 *147:* 显示等待消息列表的面板示例

从 Mail Awaiting 面板中，用户可以选择一条消息并显示该消息的内容（请参阅第 1082 页的图 148 以获取示例）。 程序使用 MQGET 调用通过在程序浏览所有消息时注明的 `MsgId` 和 `CorrelId` 来从队列中移除此消息。 使用 MQGMO_SYNCPOINT 选项来执行此 MQGET 调用。 程序显示消息的内容，然后声明同步点；这将落实 MQGET 调用，因此消息现在不再存在。

```
---------------------- IBM MQ for z/OS Sample Programs ----------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
---------------------------------- Message -------------------------------
| HELLO FROM JOHNJ                          |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
|                                           |
'-------------------------------------------------------------------------'
```

图 *148:* 显示消息内容的面板示例

对邮件管理器所提供的功能的一项明显扩展是，为用户提供在查看消息内容后将该消息保留在队列上的选项。 要执行此操作，在显示消息后必须撤回用于从队列中移除该消息的 MQGET 调用。

z/OS *z/OS* 上的“发送邮件”程序

当用户完成“发送邮件”面板（CSQ4VDP4（对于 TSO）或 VD4（对于 CICS））时，发送邮件程序（CSQ4TVD4（对于 TSO）或 CSQ4CVD4（对于 CICS））会将消息放入接收方的邮件队列中。

为执行此操作，程序使用 MQPUT1 调用。 消息的目标取决于用户如何填充“发送邮件”面板中的字段：

- 如果用户仅指定了用户名，那么程序首先假设名称为昵称，并且将消息发送到由该名称定义的对象。 如果没有此类昵称，那么程序尝试将消息发送到具有该名称的本地队列。
- 如果用户同时指定了用户名和队列管理器名称，那么程序将消息发送到由这两个名称定义的邮件队列。

该程序不接受空白消息，并且它从消息文本的各行中移除前导空白。

如果 MQPUT1 调用成功，那么程序会显示一条消息，其中显示用户名以及该消息放到的队列管理器名称。 如果该调用不成功，那么程序专门检查指示队列或队列管理器不存在的原因码；这些是 MQRC_UNKNOWN_OBJECT_NAME 和 MQRC_UNKNOWN_OBJECT_Q_MGR。 程序针对其中各错误显示其自己的错误消息；对于其他错误，程序显示该调用返回的完成代码和原因码。

z/OS *z/OS* 上的昵称程序

用户定义昵称时，昵称程序（CSQ4TVD5 for TSO、CSQ4CVD5 for CICS）将创建一个队列（将昵称作为其名称的一部分）。

程序通过将消息放在系统命令输入队列上来执行此操作。 消息包含 IBM MQ for z/OS 命令 DEFINE QALIAS 或 DEFINE QREMOTE。 程序创建的队列的类型取决于用户如何填充“创建昵称”面板（对于 TSO 为 CSQ4VDP5，对于 CICS 为 VD5）的字段：

- 如果用户仅指定了用户名，或者队列管理器名称与邮件管理器连接到的队列管理器的名称相同，那么程序将创建别名队列。
- 如果用户同时指定了用户名和队列管理器名称（并且队列管理器不是邮件管理器连接到的队列管理器），那么程序将创建远程队列的本地定义。 程序不检查此定义解析为的队列存在与否，甚至远程队列管理器是否存在。

程序还创建临时动态队列以处理来自系统命令输入队列的回复。

如果队列管理器由于程序预期的原因（例如，队列已经存在）无法创建昵称队列，那么程序显示其自己的错误消息。 如果队列管理器由于程序未预期的原因无法创建队列，那么程序最多显示两条由命令服务器返回到程序的错误消息。

注: 对于各昵称，昵称程序仅创建别名队列或远程队列的本地定义。 仅当昵称中包含的用户标识用于启动邮件管理器应用程序时，才会创建这些队列名称解析为的本地队列。

z/OS *z/OS* 上的 *Credit Check* 样本

Credit Check 样本应用程序是演示如何使用 IBM MQ for z/OS 提供的许多功能部件的程序套件。 它显示了应用程序的许多组件程序如何使用消息排队技术互相传递消息。

样本可以作为独立的 CICS 应用程序运行。 但是，为演示如何设计使用 CICS 和 IMS 环境均提供的设施的消息排队应用程序，还提供一个模块作为 IMS 成批消息处理程序。 第 1092 页的『z/OS 上 Credit Check 样本的 IMS 扩展』中描述了对样本的此扩展。

您也可以在多个队列管理器上运行样本，并在应用程序的各实例之间发送消息。 要执行此操作，请参阅第 1092 页的『z/OS 上具有多个队列管理器的 Credit Check 样本』。

CICS 程序以 C 和 COBOL 形式交付。 仅提供了一个 C 语言版本的 IMS 程序。 在 第 1066 页的表 189 和 第 1068 页的表 191 中显示了所提供的数据集。

该应用程序演示在银行客户请求贷款时评估风险的方法。 该应用程序说明银行如何通过两种方式处理贷款请求：

- 在直接与客户打交道时，银行员工想要立即访问帐户和信用风险信息。
- 在处理书面申请时，银行员工可以提交对帐户和信用风险信息的一系列请求，并且稍后处理回复。

该应用程序中的财务和安全性详细信息已保持简单，以便消息排队方法明确。

z/OS 在 *z/OS* 上准备和运行 *Credit Check* 样本

要准备并运行 Credit Check 样本，请执行以下步骤：

1. 创建用于存放有关某些示例帐户的信息的 VSAM 数据集。 通过编辑和运行数据集 CSQ4FILE 中提供的 JCL 来执行此操作。
2. 执行第 1064 页的『在 z/OS 上为 CICS 环境准备样本应用程序』中的步骤。 (如果要使用样本的 IMS 扩展，那么必须执行的其他步骤在第 1092 页的『z/OS 上 Credit Check 样本的 IMS 扩展』中进行了描述。)
3. 启动 CKTI 触发器监视器 (随 IBM MQ for z/OS 一起提供) 针对队列 CSQ4SAMP.INITIATION.QUEUE，使用 CICS 事务 CKQC。
4. 要启动应用程序，请启动 CICS 系统并使用事务 MVB1。
5. 从第一个面板中选择**立即**或**批量**查询。

   立即查询和批量查询面板类似；第 1084 页的图 149 显示“立即查询”面板。

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . . . . . _____________________
Social security number ___ __ ____
Bank account name . . _______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : _____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

   图 *149: Credit Check* 样本应用程序的“立即查询”面板

6. 在相应的字段中输入帐号和贷款金额。 请参阅第 1084 页的『在查询面板中输入信息』以获取有关要在这些字段中输入的信息的指南。

在查询面板中输入信息

Credit Check 样本应用程序检查在查询面板的 **Amount requested** 字段中输入的数据是否为整数形式。

如果输入以下帐号之一，那么应用程序在 VSAM 数据集 CSQ4BAQ 中查找相应的帐户名称、平均帐户余额和信誉指数：

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444

- 5555555555
- 6666666666
- 7777777777

您可以在其他字段中输入任何信息或不输入信息。 应用程序保留您输入的任何信息，并在其生成的报告中返回相同信息。

z/OS *z/OS 上 Credit Check 样本的设计*

本节描述了构成 Credit Check 样本应用程序的每个程序的设计。

有关在应用程序设计期间考虑的某些方法的更多信息，请参阅第 1090 页的『z/OS 上 Credit Check 样本的设计注意事项』。

第 1086 页的图 150 显示组成应用程序的程序，以及这些程序服务于的队列。 在此图中，已从所有队列名称中省略前缀 CSQ4SAMP 来使该图更易于理解。

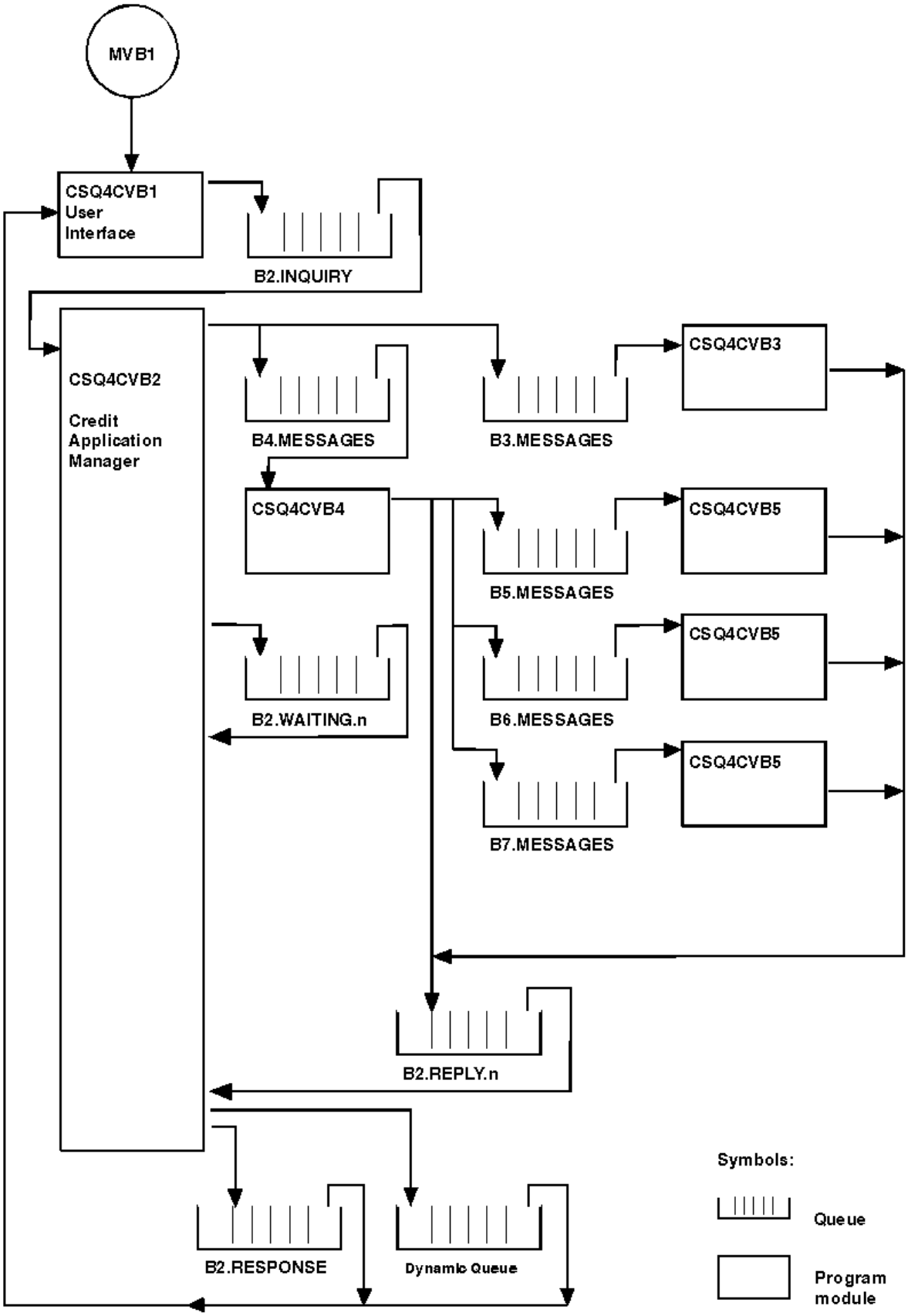


图 *150: Credit Check* 样本应用程序（仅 *COBOL* 程序）的程序和队列

z/OS *z/OS 上的用户接口程序 (CSQ4CVB1)*

启动会话方式 CICS 事务 MVB1 时，这将启动应用程序的用户接口程序。

此程序将查询消息放在队列 CSQ4SAMP.B2.INQUIRY 上，并且从它在进行查询时指定的应答队列获取对这些查询的回复。 从用户接口，可以提交立即查询或批量查询：

- 对于立即查询，程序创建它用作应答队列的临时动态队列。 这意味着各查询具有其自己的应答队列。
- 对于批量查询，用户接口程序从队列 CSQ4SAMP.B2.RESPONSE 获取回复。 为简便起见，程序从这一个应答队列获取其所有查询的回复。 很容易理解银行想要对 MVB1 的各用户使用单独的应答队列，以便他们各自只能查看对其已启动的那些查询的回复。

在批量方式和立即方式下，应用程序中使用的消息属性之间的重要差异为：

- 对于批量工作，消息具有低优先级，因此它们在以立即方式输入的任何贷款请求之后进行处理。 此外，消息是持久性的，因此如果应用程序或队列管理器必须启动，那么会恢复消息。
- 对于立即工作，消息具有高优先级，因此它们在以批量方式输入的任何贷款请求之前进行处理。 此外，消息不是持久性的，因此如果应用程序或队列管理器必须启动，那么会废弃消息。

但是，在所有情况下，贷款请求消息的属性都在整个应用程序内传播。 因此，例如，由高优先级请求产生的所有消息也将具有高优先级。

z/OS *z/OS 上的信用申请管理器 (CSQ4CVB2)*

信用申请管理器 (CAM) 程序执行对信用检查应用程序的大部分处理。

当在队列 CSQ4SAMP.B2.INQUIRY 或队列 CSQ4SAMP.B2.REPLY 上发生触发器事件时，CAM 由 CKTI 触发器监视器（随附于 IBM MQ for z/OS）启动。 *n*，其中 *n* 是用于标识一组应答队列中的某个队列的整数。 触发器消息含有包括发生触发器事件的队列的名称的数据。

CAM 使用具有 CSQ4SAMP.B2.WAITING.n 形式的名称的队列来存储有关其正在处理的查询的信息。 队列进行了命名，以便其各自与应答队列配对；例如，队列 CSQ4SAMP.B2.WAITING.3 包含特定查询的输入数据，队列 CSQ4SAMP.B2.REPLY.3 包含全都与该相同查询相关的一组回复消息（来自用于查询数据库的程序）。 要理解此设计背后的原因，请参阅第 1091 页的『分隔 CAM 中的查询队列和应答队列』。

启动逻辑

如果触发器事件发生在队列 CSQ4SAMP.B2.INQUIRY 上，那么 CAM 将打开该队列以进行共享访问。 然后，它会尝试打开各应答队列，直至找到可用应答队列。 如果找不到可用应答队列，那么 CAM 会记录事实并正常终止。

如果触发器事件发生在队列 CSQ4SAMP.B2.REPLY.n 上，那么 CAM 将打开该队列以进行独占访问。 如果返回码报告对象已在使用中，那么 CAM 正常终止。 如果发生任何其他错误，那么 CAM 会记录错误并终止。 CAM 打开对应的等待队列和查询队列，然后开始获取和处理消息。 CAM 从等待队列恢复部分完成的查询的详细信息。

为在本样本中简便起见，所使用的队列的名称存放在程序中。 在业务环境中，队列名称将可能存放在程序所访问的文件中。

从查询队列获取消息

CAM 首先使用带有 MQGMO_SET_SIGNAL 选项的 MQGET 调用尝试从查询队列获取消息。 如果消息立即可用，那么将处理该消息；如果没有消息可用，那么将设置信号。

然后，CAM 再次使用具有同一选项的 MQGET 调用尝试从应答队列获取消息。 如果消息立即可用，那么将处理该消息；否则设置信号。

当两个信号均已设置时，程序等待直至发布其中一个信号。 如果发布了信号以指示消息可用，那么会检索并处理该消息。 如果信号到期或队列管理器正在终止，那么程序将终止。

处理 CAM 检索的消息

CAM 检索的消息可以是四种类型之一：

- 查询消息
- 回复消息
- 传播消息
- 意外或不需要的消息

CAM 按照第 1088 页的『在 z/OS 上处理 CAM 检索的消息』中所述处理这些消息。

发送答案

在 CAM 针对查询收到其期望的所有回复后，将会处理回复并创建单条响应消息。 它将来自具有同一 CorrelId 的所有回复消息的所有数据合并成一条消息。 此响应放在原始贷款请求中指定的应答队列上。 响应消息放在包含最终回复消息的检索的同一工作单元内。 这旨在通过确保队列 CSQ4SAMP.B2.WAITING.n 上绝没有已完成的消息来简化恢复。

恢复部分完成的查询

CAM 将其收到的所有消息复制到队列 CSQ4SAMP.B2.WAITING.n 上。 它按如下设置消息描述符的字段：

- *Priority* 由消息的类型确定：
  - 对于请求消息，priority = 3
  - 对于数据报，priority = 2
  - 对于回复消息，priority = 1
- *CorrelId* 设置为贷款请求消息的 *MsgId*
- 其他 MQMD 字段从收到的消息的对应字段进行复制

完成查询后，将在回答处理期间从等待队列移除特定查询的消息。 因此，在任何时候，等待队列都包含与进行中的查询相关的所有消息。 如果程序必须重新启动，那么这些消息用于恢复进行中的查询的详细信息。 系统设置了不同的优先级，以便查询消息在传播消息或回复消息之前进行恢复。

z/OS 在 *z/OS* 上处理 *CAM* 检索的消息

信用申请管理器 (CAM) 检索的消息可以是四种类型之一。 CAM 处理消息的方式取决于其类型。

CAM 检索的消息可以是四种类型之一：

- 查询消息
- 回复消息
- 传播消息
- 意外或不需要的消息

CAM 按如下处理这些消息：

查询消息

查询消息来自用户接口程序。 它为各贷款请求创建查询消息。

对于所有贷款请求，CAM 请求客户的支票帐户的平均余额。 它通过将请求消息放在别名队列 CSQ4SAMP.B2.OUTPUT.ALIAS 上来执行此操作。 此队列名称解析为由支票帐户程序 CSQ4CVB3 处理的队列 CSQ4SAMP.B3.MESSAGES。 当 CAM 将消息放在此别名队列上时，它为应答队列指定相应的 CSQ4SAMP.B2.REPLY.n 队列。 此处使用了别名队列，以便程序 CSQ4CVB3 可以轻松替换为用于处理名称不同的基本队列的其他程序。 要执行此操作，需要重新定义别名队列，以便其名称解析为新队列。 此外，可以向别名队列和基本队列分配不同的访问权限。

如果用户请求的贷款大于 10000 个单位，那么 CAM 也会启动对其他数据库的检查。 它通过将请求消息放在由分发程序 CSQ4CVB4 处理的队列 CSQ4SAMP.B4.MESSAGES 上来执行此操作。 为此队列服务的进程将消息传播到由对其他记录（例如信用卡历史记录、储蓄帐户和抵押贷款）具有访问权的程序提供服务的队列。 来自这些程序的数据返回到放置操作中指定的应答队列。 此外，此程序会向应答队列发送传播消息来指定已发送的传播消息数。

在业务环境中，分发程序将可能重新格式化所提供的数据以匹配其他各类型的银行帐户所需的格式。

引用的任何队列都可位于远程系统上。

对于各查询消息，CAM 启动内存常驻查询记录表 (IRT) 中的条目。 此记录包含：

- 查询消息的 `MsgId`
- 在 ReplyExp 字段中，期望的响应数（等于已发送的消息数）
- 在 ReplyRec 字段中，收到的回复数（在此阶段为零）
- 在 PropsOut 字段中，用于表明是否期望传播消息的指示

CAM 将查询消息复制到具有以下设置的等待队列上：

- `Priority` 设置为 3
- `CorrelId` 设置为查询消息的 `MsgId`
- 其他消息描述符字段设置为查询消息的对应字段

传播消息

传播消息包含分发程序已向其转发查询的队列的数量。 消息按如下处理：

1. 向 IRT 中的相应记录的 ReplyExp 字段添加已发送的消息数。 此信息包含在消息中。
2. 将 IRT 中的记录的 ReplyRec 字段按 1 递增。
3. 将 IRT 中的记录的 PropsOut 字段按 1 递减。
4. 将消息复制到等待队列上。 CAM 将 `Priority` 设置为 2 并将消息描述符的其他字段设置为传播消息的对应字段。

回复消息

回复消息包含对支票帐户程序的请求之一或对机构查询程序之一的响应。 回复消息按如下处理：

1. 将 IRT 中的记录的 ReplyRec 字段按 1 递增。
2. 将消息复制到等待队列，其中 `Priority` 设置为 1 且消息描述符的其他字段设置为回复消息的对应字段。
3. 如果 ReplyRec = ReplyExp 且 PropsOut = 0，请设置 MsgComplete 标志。

其他消息

应用程序并不期望其他消息。 但是，应用程序可能会收到系统广播的消息或具有未知 `CorrelIds` 的回复消息。

CAM 将这些消息放在可以对其进行检查的队列 CSQ4SAMP.DEAD.QUEUE 上。 如果此放置操作失败，那么消息丢失并且程序继续。 有关程序的此部分的设计的更多信息，请参阅第 1091 页的『样本如何处理意外消息』。

z/OS *z/OS 上的支票帐户程序 (CSQ4CVB3)*

支票帐户程序由队列 CSQ4SAMP.B3.MESSAGES 上的触发器事件启动。 打开队列后，此程序会使用具有 wait 选项且将等待时间间隔设置为 30 秒的 MQGET 调用从队列中获取消息。

程序搜索 VSAM 数据集 CSQ4BAQ 以查找贷款请求消息中的帐号。 它检索对应的帐户名称、平均余额和信誉指数，或者注明帐户不在数据集中。

然后，程序将回复消息（使用 MQPUT 调用）放在贷款请求消息中命名的应答队列上。 对于此回复消息，程序执行以下操作：

- 复制贷款请求消息的 `CorrelId`
- 使用 MQPMO_PASS_IDENTITY_CONTEXT 选项

程序继续从队列取出消息，直至等待时间间隔到期。

z/OS *z/OS 上的分发程序 (CSQ4CVB4)*

分发程序由队列 CSQ4SAMP.B4.MESSAGES 上的触发器事件启动。

为模拟将贷款请求分发到对信用卡历史记录、储蓄帐户和抵押贷款等记录具有访问权的其他机构，该程序将同一消息的副本放在名称列表 CSQ4SAMP.B4.NAMELIST 中的所有队列上。 其中有三个队列的名称格式为 CSQ4SAMP.B *n*.MESSAGES（其中 *n* 为 5、6 或 7）。 在业务应用程序中，机构可能位于不同的位置，因此

这些队列可能是远程队列。 如果要修改样本应用程序以显示此内容，请参阅第 1092 页的『z/OS 上具有多个队列管理器的 Credit Check 样本』。

分发程序执行以下步骤：

1. 从名称列表中，获取程序将要使用的队列的名称。 程序通过使用 MQINQ 调用查询名称列表对象的属性来执行此操作。
2. 打开这些队列以及 CSQ4SAMP.B4.MESSAGES。
3. 执行以下循环，直至队列 CSQ4SAMP.B4.MESSAGES 上不再有其他消息：
   a. 使用带有等待选项且等待时间间隔设置为 30 秒的 MQGET 调用获取消息。
   b. 将消息放在名称列表中列出的各队列上，为应答队列指定相应的 CSQ4SAMP.B2.REPLY.n 队列的名称。 程序将贷款请求消息的 *CorrelId* 复制到这些副本消息，并且在 MQPUT 调用中使用 MQPMO_PASS_IDENTITY_CONTEXT 选项。
   c. 向队列 CSQ4SAMP.B2.REPLY.n 发送数据报消息来显示其已成功放置的消息数。
   d. 声明同步点。

z/OS *z/OS 上的机构查询程序 (CSQ4CVB5/CSQ4CCB5)*

机构查询程序以 COBOL 程序和 C 程序形式提供。 两种程序具有相同设计。 这表明不同类型的程序可以轻松地在一个 IBM MQ 应用程序中共存，并且可以轻松替换这种应用程序所包含的程序模块。

程序的实例由以下任何队列上的触发器事件启动：

- 对于 COBOL 程序 (CSQ4CVB5)：
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- 对于 C 程序 (CSQ4CCB5)，队列 CSQ4SAMP.B8.MESSAGES

注: 如果要使用 C 程序，那么必须修改名称列表 CSQ4SAMP.B4.NAMELIST 的定义以将队列 CSQ4SAMP.B7.MESSAGES 替换为 CSQ4SAMP.B8.MESSAGES。 要执行此操作，可以使用以下任一方式：

- IBM MQ for z/OS 操作和控制面板
- ALTER NAMELIST 命令
- CSQUTIL 实用程序

在它打开相应的队列后，此程序使用带有等待选项且等待时间间隔设置为 30 秒的 MQGET 调用从队列取出消息。

程序通过搜索 VSAM 数据集 CSQ4BAQ 以查找贷款请求消息中传递的帐号来模拟机构数据库的搜索。 然后，它将构建包含其正在服务的队列的名称的回复以及信誉指数。 为简化处理，将随机选择信誉指数。

放置回复消息时，程序使用 MQPUT1 调用并执行以下操作：

- 复制贷款请求消息的 `CorrelId`
- 使用 MQPMO_PASS_IDENTITY_CONTEXT 选项

程序将回复消息发送到贷款请求消息中命名的应答队列。 （贷款请求消息中还指定了拥有该应答队列的队列管理器的名称。）

z/OS *z/OS 上 Credit Check 样本的设计注意事项*

Credit Check 样本的设计注意事项。

本主题包含有关以下各项的信息：

- 第 1091 页的『分隔 CAM 中的查询队列和应答队列』
- 第 1091 页的『样本如何处理错误』
- 第 1091 页的『样本如何处理意外消息』
- 第 1091 页的『样本如何使用同步点』

- 第 1092 页的『样本如何使用消息上下文信息』
- 第 1092 页的『在 CAM 中使用消息和相关标识』

分隔 CAM 中的查询队列和应答队列

应用程序可能对查询和回复使用单个队列，但是由于以下原因，它可以使用不同队列：

- 当程序正在处理最大数量的查询时，进一步查询可以保留在队列上。 如果使用的是单个队列，那么必须从该队列移除这些查询并将其存储在其他位置。
- 其他 CAM 实例可以自动启动以服务于同一查询队列，前提是消息流量高到足以保证此服务。 但是程序必须跟踪进行中的查询，要执行此操作，它必须获取对其已启动的查询的所有回复。 如果仅使用了一个队列，那么程序将必须浏览消息，以查看这些消息是针对此程序还是针对其他程序。 这将大幅降低操作效率。

  应用程序可以支持多个 CAM，并且可以通过使用配对的应答队列和等待队列来有效恢复进行中的查询。
- 程序可以通过使用发信号在多个队列上有效等待。

样本如何处理错误

用户接口程序通过直接向用户报告错误来处理这些错误。

其他程序没有用户接口，因此它们必须以其他方式处理错误。 此外，在许多情况下（例如，如果 MQGET 调用失败），这些其他程序不知道应用程序的用户身份。

其他程序将错误消息放在名为 CSQ4SAMP 的 CICS 临时存储队列上。 您可以使用 CICS 提供的事务 CEBR 来浏览此队列。 程序还会将错误消息写入到 CICS CSML 日志。

样本如何处理意外消息

在设计消息排队应用程序时，您必须决定如何处理意外到达队列的消息。

两个基本选项为：

- 应用程序在处理意外消息之前，不再执行更多工作。 这可能意味着应用程序通知操作员，终止其自身并确保其不会自动重新启动（它可以通过将触发设置为关闭来实现此目的）。 此选项意味着对应用程序的所有处理都可通过单条意外消息来停止，并且需要操作员的干预来重新启动应用程序。
- 应用程序从它正在服务的队列中移除消息，将消息放在其他位置并继续处理。 放置此消息的最佳位置是在系统死信队列上。

如果选择第二个选项：

- 操作员或其他程序应检查放在死信队列上的消息以确定消息的来源。
- 如果意外消息无法放在死信队列上，那么它会丢失。
- 如果意外消息的长度超过死信队列上的消息限制或者超过程序中的缓冲区大小，那么会将其截断。

为确保应用程序平稳处理所有查询且受外部活动的影响最小，Credit Check 样本应用程序使用第二个选项。 为使您能够将样本与其他使用同一队列管理器的应用程序分隔，Credit Check 样本不使用系统死信队列，而是使用其自己的死信队列。 此队列命名为 CSQ4SAMP.DEAD.QUEUE。 样本会截断任何长度超过为样本程序提供的缓冲区的消息。 您可以使用 Browse 样本应用程序浏览此队列上的消息，或者使用 Print Message 样本应用程序打印消息及其消息描述符。

但是，如果将样本扩展为跨多个队列管理器运行，那么意外消息或无法传送的消息可能被队列管理器放在系统死信队列上。

样本如何使用同步点

Credit Check 样本应用程序中的程序会声明同步点以确保：

- 仅发送一条回复消息以响应各预期消息
- 意外消息的多个副本绝不会放在样本的死信队列上
- CAM 可以通过从其等待队列获取持久消息来恢复所有部分完成的查询的状态

要实现此目的，将使用单个工作单元来恢复消息的获取、该消息的处理以及任何后续放置操作。

样本如何使用消息上下文信息

当用户接口程序 (CSQ4CVB1) 发送消息时，它使用 MQPMO_DEFAULT_CONTEXT 选项。 这意味着队列管理器同时生成身份和源上下文信息。 队列管理器从已启动程序 (MVB1) 的事务和已启动该事务的用户标识获取此信息。

当 CAM 发送查询消息时，它使用 MQPMO_PASS_IDENTITY_CONTEXT 选项。 这意味着从原始查询消息的身份上下文复制进行放置的消息的身份上下文信息。 通过此选项，由队列管理器生成源上下文信息。

当 CAM 发送回复消息时，它使用 MQPMO_ALTERNATE_USER_AUTHORITY 选项。 这导致队列管理器在 CAM 打开应答队列时使用备用用户标识进行其安全性检查。 CAM 使用原始查询消息的提交者的用户标识。 这意味着仅允许用户查看对其已启动的查询的回复。 备用用户标识获取自原始查询消息的消息描述符中的身份上下文信息。

当查询程序 (CSQ4CVB3/4/5) 发送回复消息时，它们使用 MQPMO_PASS_IDENTITY_CONTEXT 选项。 这意味着从原始查询消息的身份上下文复制进行放置的消息的身份上下文信息。 通过此选项，由队列管理器生成源上下文信息。

注: 与 MVB3/4/5 事务关联的用户标识需要访问 B2.REPLY.n 队列。 这些用户标识可能与进行处理的请求的关联用户标识不同。 要解决此安全隐患，查询程序可以在放置其回复时使用 MQPMO_ALTERNATE_USER_AUTHORITY 选项。 这意味着 MVB1 的每个单独的用户需要权限来打开 B2.REPLY.n 队列。

在 CAM 中使用消息和相关标识

应用程序必须时刻监视其正在处理的所有实时查询的进度。 为实现此目的，它使用各贷款请求消息的唯一消息标识来关联它具有的有关各查询的所有信息。

CAM 将查询消息的 `MsgId` 复制到它针对该查询发送的所有请求消息的 `CorrelId` 中。 样本 (CSQ4CVB3 - 5) 中的其他程序将其收到的各消息的 `CorrelId` 复制到其回复消息的 `CorrelId` 中。

z/OS *z/OS* 上具有多个队列管理器的 *Credit Check* 样本

您可以使用 Credit Check 样本应用程序来演示分布式排队，方法是在两个队列管理器和 CICS 系统上安装样本（每个队列管理器都连接到不同的 CICS 系统）。

当安装了样本程序且触发器监视器 (CKTI) 正在各系统上运行时，需要执行以下操作：

1. 设置两个队列管理器之间的通信链路。 有关如何执行此操作的信息，请参阅配置分布式排队。
2. 在一个队列管理器上，为要使用的各远程队列（在其他队列管理器上）创建本地定义。 这些队列可以是任何 CSQ4SAMP.B*n*.MESSAGES，其中 *n* 为 3、5、6 或 7。 （这些是由支票帐户程序和机构查询程序提供服务的队列。） 有关如何执行此操作的信息，请参阅 DEFINE QREMOTE 和 DEFINE 队列。
3. 更改名称列表 (CSQ4SAMP.B4.NAMELIST) 的定义，以便其包含要使用的远程队列的名称。 有关如何执行此操作的信息，请参阅 DEFINE NAMELIST。

z/OS *z/OS* 上 *Credit Check* 样本的 *IMS* 扩展

支票帐户程序的版本以 IMS 批量消息处理 (BMP) 程序形式提供。 它采用 C 语言进行编写。

该程序执行与 CICS 版本相同的功能，不同在于如要获取帐户信息，程序读取 IMS 数据库而非 VSAM 文件。 如果将支票帐户程序的 CICS 版本替换为 IMS 版本，那么您将看到使用应用程序的方法没有差异。

要准备并运行 IMS 版本，必须执行以下操作：

1. 遵循第 1084 页的『在 z/OS 上准备和运行 Credit Check 样本』中的步骤。
2. 遵循第 1067 页的『在 z/OS 上为 IMS 环境准备样本应用程序』中的步骤。
3. 修改别名队列 CSQ4SAMP.B2.OUTPUT.ALIAS 的定义，以解析为队列 CSQ4SAMP.B3.IMS.MESSAGES（而非 CSQ4SAMP.B3.MESSAGES）。 要执行此操作，可以使用以下方法之一：
   - IBM MQ for z/OS 操作和控制面板
   - ALTER QALIAS 命令。

使用 IMS 支票帐户程序的另一种方法是使其服务于从分发程序接收消息的队列之一。 在 Credit Check 样本应用程序的已交付形式中，有三个这样的队列 (B5/6/7.MESSAGES)，全都由机构查询程序提供服务。 此程序搜索 VSAM 数据集。 要比较 VSAM 数据集和 IMS 数据库的使用，可以使 IMS 支票帐户程序改为服务于这些队列之一。 要执行此操作，必须修改名称列表 CSQ4SAMP.B4.NAMELIST 的定义，以将 CSQ4SAMP.B*n*.MESSAGES 队列之一替换为 CSQ4SAMP.B3.IMS.MESSAGES 队列。 可以使用以下方法之一：

- IBM MQ for z/OS 操作和控制面板
- ALTER NAMELIST 命令。

然后，可以从 CICS 事务 MVB1 运行样本。 用户看到操作或响应没有差异。 IMS BMP 在收到停止消息或处于不活动状态达到 5 分钟后将停止。

IMS 支票帐户程序 (CSQ4ICB3) 的设计

此程序作为 BMP 运行。 在向程序发送任何 IBM MQ 消息之前，请使用程序的 JCL 启动此程序。

程序搜索 IMS 数据库以查找贷款请求消息中的帐号。 它检索对应的帐户名称、平均余额和信誉指数。

程序将数据库搜索的结果发送到进行处理的 IBM MQ 消息中指定的应答队列。 返回的消息将帐号类型和搜索结果附加到所收到的消息，以便构建响应的事务可以确认处理的是正确的查询。 消息采用三个具有 79 个字符的组的形式，如下所示：

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

当作为面向消息的 BPM 运行时，程序将放弃 IMS 消息队列，然后从 IBM MQ for z/OS 队列读取消息并处理这些消息。 未从 IMS 消息队列收到任何信息。 由于已关闭句柄，因此程序会在各检查点之后重新连接到队列管理器。

在面向批量的 BMP 中运行时，由于句柄未关闭，因此程序在各检查点之后继续连接到队列管理器。

z/OS *z/OS 上的 Message Handler 样本*

通过 Message Handler 样本 TSO 应用程序，可以浏览、转发和删除队列上的消息。 该样本以 C 和 COBOL 形式提供。

准备并运行样本

请按照以下步骤操作：

1. 按第 1063 页的『在 z/OS 上为 TSO 环境准备样本应用程序』中所述准备样本。
2. 定制为样本提供的 CLIST (CSQ4RCH1) 以定义面板的位置、消息文件的位置和装入模块的位置。

可以使用 CLIST CSQ4RCH1 来运行 C 和 COBOL 版本的样本。 所提供的 CSQ4RCH1 版本运行 C 版本，并且包含有关 COBOL 版本的必需定制的指示信息。

注:

1. 样本没有随附样本队列定义。
2. VS COBOL II 不支持通过 ISPF 实现多任务，因此请勿在拆分屏幕两侧使用 Message Handler 样本应用程序。 如果执行此操作，那么结果不可预测。

z/OS 在 *z/OS* 上使用 *Message Handler* 样本

安装样本并从定制的 CLIST CSQ4RCH1 对其进行调用后，将会显示第 1094 页的图 151 中所示的屏幕。

```
---------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :


Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

图 *151: Message Handler* 样本的初始屏幕

输入要查看的队列管理器和队列名称（区分大小写），然后将显示消息列表屏幕（请参阅第 1094 页的图 152）。

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

图 *152: Message Handler* 样本的消息列表屏幕

此屏幕显示队列上的前 99 条消息，并且针对各消息显示以下字段:

Msg No
消息编号

Put Date MM/DD/YYYY
消息放在队列上的日期 (GMT)

Put Time HH:MM:SS
消息放在队列上的时间 (GMT)

Format Name
MQMD.Format 字段

UserIdentifier
MQMD.UserIdentifier 字段

Put Application Type
MQMD.PutApplType 字段

Put Application Name
MQMD.PutApplName 字段

此外还会显示队列上的消息总数。

从此屏幕可以按编号而不是按光标位置选择消息，然后显示该消息。 要获取示例，请参阅第 1095 页的图 153。

```
----------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE               :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
--------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId       : `C3E2D840E5D4F0F340404040404040AF6B30F0A89B7605`X
CorrelId     : `000000000000000000000000000000000000000000000000`X
BackoutCount    : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                 `
ReplyToQMgr    : `VM03                       `
UserIdentifier  : `NTSFV02    `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                        `
PutApplType    : 000000002
PutApplName    : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .....T......`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

图 *153:* 显示所选消息

显示消息后，即可将其删除，保留在队列上或转发到其他队列。 `Forward to Q Mgr` 和 `Forward to Queue` 字段使用来自 MQMD 的值进行初始化，在转发消息之前可以更改这些值。

样本设计仅允许选择和显示具有唯一 `MsgId / CorrelId` 组合的消息，因为将使用 `MsgId` 和 `CorrelId` 作为键来检索消息。 如果键不唯一，那么样本无法确切地检索所选消息。

注: 使用 SCSQCLST(CSQ4RCH1) 样本来浏览消息时，各调用导致消息的回退计数增大。 如果要更改此样本的行为，请复制样本并在必要时修改内容。 您应当意识到，依赖此回退计数的其他应用程序会受此不断增大的计数的影响。

z/OS *z/OS 上 Message Handler 样本的设计*

本主题描述组成 Message Handler 样本应用程序的各程序的设计。

对象验证程序

此程序请求有效的队列和队列管理器名称。

如果不指定队列管理器名称，那么将使用缺省队列管理器（如果可用）。 只能使用本地队列；系统将发出 MQINQ 来检查队列类型，如果队列不是本地队列，那么将报告错误。 如果队列未成功打开，或者在队列上禁止 MQGET 调用，那么将返回指示 CompCode 和 Reason 返回码的错误消息。

消息列表程序

此程序显示队列上的消息列表及其相关信息（例如 putdate、puttime 和消息格式）。

存储在列表中的最大消息数为 99。 如果队列上的消息数超过此数字，那么还会显示当前队列深度。 要选择消息以进行显示，请在输入字段中输入消息号（缺省值为 01）。 如果输入无效，那么您将收到相应的错误消息。

消息内容程序

此程序显示消息内容。

内容会进行格式化并拆分为两部分：

1. 消息描述符
2. 消息缓冲区

消息描述符在单独的行上显示各字段的内容。

消息缓冲区根据其内容进行格式化。 如果缓冲区包含死信消息头 (MQDLH) 或传输队列头 (MQXQH)，那么将在缓冲区自身之前格式化并显示这些内容。

在格式化缓冲区数据之前，标题行显示消息的缓冲区长度（以字节为单位）。 最大缓冲区大小为 32768 字节，并且长度超过此大小的任何消息都会截断。 缓冲区的完全大小随消息一起显示，指示仅显示该消息的前 32768 个字节。

缓冲区数据以两种方式格式化：

1. 在打印缓冲区偏移后，以十六进制形式显示缓冲区数据。
2. 然后，缓冲区数据再次显示为 EBCDIC 值。 如果无法打印任何 EBCDIC 值，那么将改为打印句点 (.)。

可以在操作字段中输入 D（表示删除）或 F（表示转发）。 如果选择转发消息，那么必须正确设置 `forward-to queue` 和 `queue manager name`。 这些字段的缺省值读取自消息描述符 ReplyToQ 和 ReplyToQMgr 字段。

如果转发消息，那么会删除存储在缓冲区中的任何头块。 如果成功转发消息，那么将从原始队列中移除该消息。 如果输入无效操作，那么将显示错误消息。

此外，还提供名为 CSQ4CHP9 的示例帮助面板。

z/OS *z/OS* 上的 *Asynchronous Put* 样本

Asynchronous Put 样本程序使用异步 MQPUT 调用将消息放在队列上。 该样本还使用 MQSTAT 调用检索状态信息。

Asynchronous Put 应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

样本程序以 C 编程语言形式交付。

Asynchronous Put 应用程序在批处理环境中运行。 请参阅其他样本以获取批处理应用程序。

本主题还提供有关 Asynchronous Consumption 程序的设计以及运行 CSQ4BCS2 样本的信息。

- 第 1097 页的『运行 CSQ4BCS2 样本』
- 第 1097 页的『Asynchronous Put 样本程序的设计』

运行 CSQ4BCS2 样本

此样本程序最多采用六个参数:

1. 目标队列的名称（必需）。
2. 队列管理器名称（可选）。
3. 打开选项（可选）。
4. 关闭选项（可选）。
5. 目标队列管理器的名称（可选）。
6. 动态队列名称（可选）。

如果未指定队列管理器，那么 CSQ4BCS2 连接到缺省队列管理器。 通过标准输入 (**SYSIN DD**) 提供消息内容。

存在用于运行程序的样本 JCL，它驻留在 CSQ4BCSP 中。

Asynchronous Put 样本程序的设计

程序将 MQOPEN 调用与所提供的输出选项或与 MQOO_OUTPUT 和 MQOO_FAIL_IF_QUIESCING 选项结合使用，以打开用于放置消息的目标队列。

如果程序无法打开队列，那么程序将输出错误消息，其中包含 MQOPEN 调用返回的原因码。 为在此调用和后续 MQI 调用中保持程序简单，将对许多选项使用缺省值。

对于各行输入，程序将文本读取到缓冲区中，并且使用带有 MQPMO_ASYNC_RESPONSE 的 MQPUT 调用创建包含该行的文本的数据报消息，然后以异步方式将消息放在目标队列上。 程序继续运行，直至到达输入的结尾或者 MQPUT 调用失败。 如果程序到达输入的结尾，那么它使用 MQCLOSE 调用来关闭队列。

然后，程序发出用于返回 MQSTS 结构的 MQSTAT 调用，并且显示包含成功放置的消息数、放置的带有警告的消息数以及失败数的消息。

注: 要观察在 MQSTAT 调用检测到 MQPUT 错误时发生的情况，请将目标队列上的 MAXDEPTH 设置为低值。

z/OS *z/OS* 上的 *Batch Asynchronous Consumption* 样本

CSQ4BCS1 样本程序以 C 形式交付，它演示使用 MQCB 和 MQCTL 异步消耗来自多个队列的消息。

Asynchronous Consumption 样本在批处理环境中运行。 请参阅其他样本以获取批处理应用程序。

此外还有在 CICS 环境中运行的 COBOL 样本，请参阅 第 1099 页的『z/OS 上的 CICS 异步使用和发布/预订样本』。

应用程序使用以下 MQI 调用:

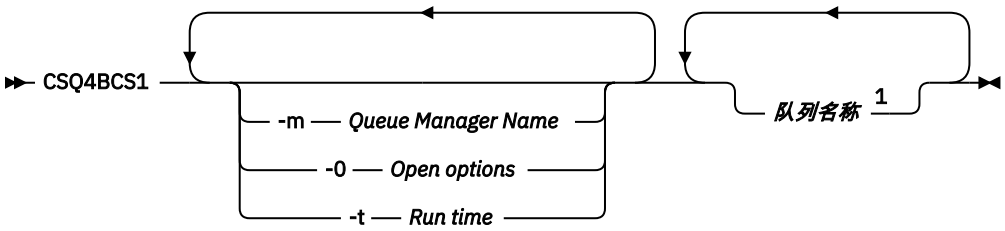
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

本主题还提供有关以下标题的信息:

- 第 1098 页的『运行 CSQ4BCS1 样本』
- 第 1098 页的『Batch Asynchronous Consumption 样本程序的设计』

运行 CSQ4BCS1 样本

此样本程序遵循以下语法:



注:

[1] 此样本最多支持 10 个队列。 请提供多个队列名称，以便读取来自多个队列的消息。

存在用于运行此程序的样本 JCL，它驻留在 CSQ4BCSC 中。

Batch Asynchronous Consumption 样本程序的设计

样本显示如何按照消息的到达顺序读取来自多个队列的消息。 这将要求更多的代码使用同步 MQGET。 在异步消耗情况下，无需轮询，并且线程和存储管理由 IBM MQ 执行。 在样本程序中，错误写入到控制台。

样本代码具有以下步骤:

1. 定义单个消息消耗回调函数。

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. 连接到队列管理器。

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. 打开输入队列，并将各队列与 `MessageConsumer` 回调函数相关联。

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   无需为各队列设置 `cbd.CallbackFunction`；它是只输入字段。 可以将其他回调函数与各队列相关联。

4. 启动消息的消耗。

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. 等待用户按 Enter 键，然后停止消息的消耗。

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 最后，与队列管理器断开连接。

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS z/OS 上的 CICS 异步使用和发布/预订样本

Asynchronous Consumption 和 Publish/Subscribe 样本程序演示在 CICS 内使用异步消耗以及发布和预订功能。

注册客户机程序注册三个回调处理程序（一个事件处理程序和两个消息使用者）并启动异步消耗。 消息传递客户机程序将消息放到队列，或者从 CICS 控制台发布合适的消息以供两个消息使用者（CSQ4CVCN 和 CSQ4CVCT）使用。

要提供对样本行为的运行时控制，可以使用其中一个消息使用者收到的消息指示其暂挂 (SUSPEND)、恢复 (RESUME) 或注销 (DEREGISTER) 任何回调处理程序。 它还可用于发出 MQCTL STOP 以结束受控制的异步消耗。 系统会注册另一个消息使用者以预订主题。

各程序在相应的点发出 COBOL DISPLAY 语句以显示样本的行为。

应用程序使用以下 MQI 调用：

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

程序以 COBOL 语言形式交付。 请参阅 CICS Asynchronous Consumption 和 Publish/Subscribe 样本以获取 CICS 应用程序。

本主题还提供以下信息：

- 第 1099 页的『设置』
- 第 1099 页的『注册客户机 CSQ4CVRG』
- 第 1100 页的『事件处理程序 CSQ4CVEV』
- 第 1100 页的『简单消息使用者 CSQ4CVCN』
- 第 1100 页的『控制消息使用者 CSQ4CVCT』
- 第 1100 页的『消息传递客户机 CSQ4CVPT』

设置

消息使用者使用的队列和主题的名称在注册客户机和消息传递客户机程序中进行硬编码。

运行样本之前，应将队列 **SAMPLE.CONTROL.QUEUE** 定义到与 CICS 区域相关联的队列管理器。 如果需要，可以定义主题 **News/Media/Movies**；或者，如果该主题不存在，那么可以在运行时将其创建在缺省管理对象下。

可以通过安装组 CSQ4SAMP 来安装 CICS 程序和事务定义。

注册客户机 CSQ4CVRG

必须在 CICS 事务 MVRG 下启动注册客户机程序。 它不采用任何输入。

在启动时，注册客户机使用 MQCB 来注册以下回调处理程序：

- CSQ4CVEV 作为事件处理程序。
- CSQ4CVCN 作为主题 **News/Media/Movies** 上的消息使用者。
- CSQ4CVCT 作为队列 **SAMPLE.CONTROL.QUEUE** 上的消息使用者。

注册客户机将包含全部三个已注册回调处理程序的名称的数据结构连同与两个消息使用者相关联的对象句柄一起传递到 CSQ4CVCT。

注册回调处理程序后，注册客户机发出 MQCTL START_WAIT 以启动异步消耗，并且暂挂直至向其返回控制（例如，通过其中一个回调处理程序发出 MQCTL STOP）。

事件处理程序 CSQ4CVEV

在驱动时，事件处理程序显示指示调用类型（例如 START）的消息。 在针对 IBM MQ 原因码 CONNECTION_QUIESCING 驱动时，事件处理程序发出 MQCTL STOP 以结束异步消耗并将控制返回到注册客户机。

简单消息使用者 CSQ4CVCN

在驱动时，消息使用者显示指示调用类型（例如 REGISTER）的消息。 当针对 MSG_REMOVED 调用类型进行驱动时，消息使用者将检索入站消息并将其输出到 CICS 作业日志。

控制消息使用者 CSQ4CVCT

在驱动时，消息使用者显示指示调用类型（例如 START）的消息。 在针对 MSG_REMOVED 调用类型驱动时，消息使用者检索由注册客户机传递的入站消息和数据结构。 根据消息内容，它发出相应的 MQCB 或 MQCTL 命令以执行以下操作之一：

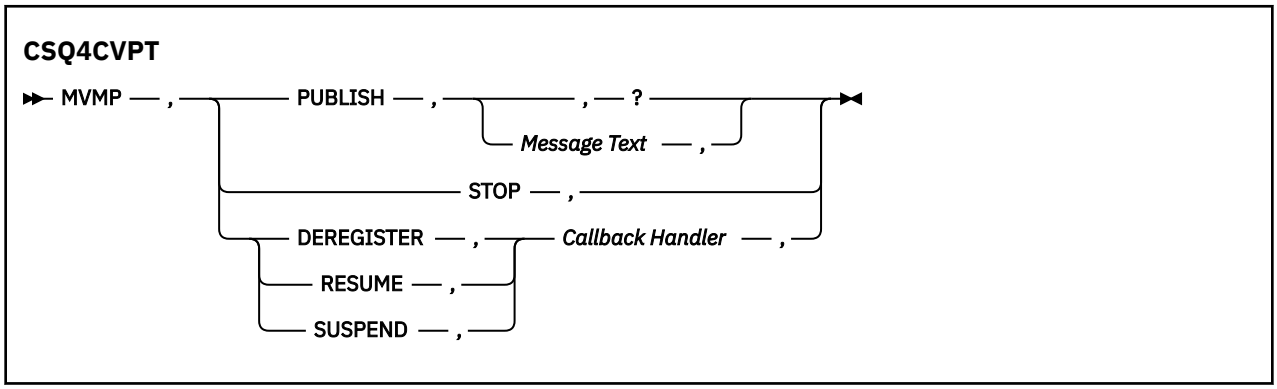
- 停止 (STOP) 异步消耗（将控制返回到注册客户机）。
- 暂挂 (SUSPEND)、恢复 (RESUME) 或注销 (DEREGISTER) 已命名的回调处理程序（包括其自身）。

消息传递客户机 CSQ4CVPT

消息传递客户机具有两个功能：

- 它将消息发布到主题以供消息使用者 CSQ4CVCN 使用。
- 它将控制消息放到队列以供控制消息使用者 CSQ4CVCT 使用，从而导致样本行为的潜在更改。

必须从 CICS 控制台下的 CICS 事务启动消息传递客户机程序，并且它采用具有以下语法的命令行输入：



PUBLISH
将消息文本（或缺省消息）作为保留消息发布以供简单消息使用者使用。

结束
停止异步消耗。

DEREGISTER
注销已命名的回调处理程序。

RESUME
恢复已命名的回调处理程序。

SUSPEND
暂挂已命名的回调处理程序。

输入字段已定位并以逗号分隔。 关键字和回调处理程序名称不区分大小写。

示例：

表 193: 输入示例	
示例	**描述**
MVMP,PUBLISH,,	发布缺省消息
MVMP,publish, A short message,	发布给定文本
MVMP,STOP,	停止异步消耗
MVMP,DEREGISTER,CSQ4CVEV,	注销事件处理程序
MVMP,resume,csq4cvcn,	恢复简单消息使用者
MVMP,SUSPEND,CSQ4CVEV,	暂挂事件处理程序

其中 MVMP 是与消息传递客户机程序 CSQ4CVPT 相关联的 CICS 事务。

注:

- 暂挂或注销所有回调处理程序将终止由注册客户机发出的 START_WAIT，向其返回控制并结束任务。
- 暂挂或注销控制回调处理程序尚未有意受到阻止，但是它会移除进一步控制样本行为的能力。

z/OS z/OS 上的 *Publish/Subscribe* 样本

Publish/Subscribe 样本程序演示在 IBM MQ 中使用发布和预订功能。

有四个 C 和两个 COBOL 编程语言样本程序用于演示如何对 IBM MQ 发布/预订接口进行编程。 这些程序以 C 和 COBOL 语言形式交付。 应用程序在批处理环境中运行；请参阅 Publish/Subscribe 样本以获取批处理应用程序。

还有在 CICS 环境中运行的 COBOL 样本；请参阅 第 1099 页的『z/OS 上的 CICS 异步使用和发布/预订样本』。

本主题还提供有关如何运行 Publish/Subscribe 样本程序的信息。 这些样本程序包括：

- 第 1101 页的『运行 CSQ4BCP1 样本』
- 第 1101 页的『运行 CSQ4BCP2 样本』
- 第 1102 页的『运行 CSQ4BCP3 样本』
- 第 1102 页的『运行 CSQ4BCP4 样本』
- 第 1102 页的『运行 CSQ4BVP1 样本』
- 第 1102 页的『运行 CSQ4BVP2 样本』

运行 CSQ4BCP1 样本

此程序采用 C 进行编写，它向主题发布消息。 在运行此程序之前，请启动订户样本之一。

此程序最多采用四个参数：

1. 目标主题字符串的名称（必需）。
2. 队列管理器名称（可选）。
3. 打开选项（可选）。
4. 关闭选项（可选）。

如果未指定队列管理器，那么 CSQ4BCP1 连接到缺省队列管理器。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BCPP 中。

通过标准输入 (**SYSIN DD**) 提供消息内容。

运行 CSQ4BCP2 样本

此程序采用 C 进行编写，它预订主题并打印收到的消息。

此程序最多采用三个参数：

1. 目标主题字符串的名称（必需）。
2. 队列管理器名称（可选）。
3. MQSD 预订选项（可选）。

如果未指定队列管理器，那么 CSQ4BCP2 连接到缺省队列管理器。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BCPS 中。

运行 CSQ4BCP3 样本

此程序采用 C 进行编写，它使用用户指定的目标队列预订主题并打印收到的消息。

此程序最多采用四个参数：

1. 目标主题字符串的名称（必需）。
2. 目标的名称（必需）。
3. 队列管理器名称（可选）。
4. MQSD 预订选项（可选）。

如果未指定队列管理器，那么 CSQ4BCP3 连接到缺省队列管理器。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BCPD 中。

运行 CSQ4BCP4 样本

此程序采用 C 进行编写，它通过允许在 MQSUB 调用中使用扩展选项并扩展更简单的 MQSUB 样本 CSQ4BCP2 上可用的选项，从主题预订和获取消息。 除消息有效内容以外，还会接收并显示各消息的消息属性。

此程序采用参数的变量集：

- **-t** *Topic string*（必填）。
- **-o** *Topic object name*（必填）。
- **-m** *Queue manager name*（可选）。
- **-q** *Destination queue name*（可选）。
- **-w** *Wait interval on MQGET in seconds* (可选)，其中 *seconds* 可以具有以下任何值:
  - unlimited：MQWI_UNLIMITED
  - none：无等待
  - *n*：等待时间间隔（以秒为单位）
  - 未指定值：在未指定值时，缺省值为 30 秒
- **-d** *Subscription name*（可选）。 创建或恢复指定的持久预订。
- **-k**（可选）。 在 MQCLOSE 上保留持久预订。

如果未指定队列管理器，那么 CSQ4BCP4 连接到缺省队列管理器。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BCPE 中。

运行 CSQ4BVP1 样本

此程序采用 COBOL 进行编写，它向主题发布消息。 在运行此程序之前，请启动订户样本之一。

此程序不采用任何参数。 **SYSIN DD** 提供输入主题名称、队列管理器名称和消息内容。

如果未指定队列管理器，那么 CSQ4BVP1 连接到缺省队列管理器。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BVPP 中。

运行 CSQ4BVP2 样本

此程序采用 COBOL 进行编写，它预订主题并打印收到的消息。

此程序不采用任何参数。 **SYSIN DD** 为主题名称和队列管理器名称提供输入。

如果未指定队列管理器，那么 CSQ4BVP1 连接到缺省队列管理器。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BVPP 中。

z/OS z/OS 上的 *Set* 和 *Inquire* 消息属性样本

消息属性样本程序演示向消息句柄添加用户定义的属性以及查询与该消息相关联的属性。

应用程序使用以下 MQI 调用：

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

程序以 C 语言形式交付。 应用程序在批处理环境中运行。 请参阅其他样本以获取批处理应用程序。

CSQ4BCM1 程序用于从消息队列查询消息句柄的属性，并且它是 MQINQMP API 调用的使用示例。 样本从队列获取一条消息，然后打印所有消息句柄属性。

CSQ4BCM2 程序用于在消息队列上设置消息句柄的属性，并且它是 MQSETMP API 调用的使用示例。 样本创建消息句柄并将其放入 MQGMO 结构的 `MsgHandle` 字段中。 然后，它将消息放到队列。

查询和打印消息属性的其他示例包含在 CSQ4BCG1 和 CSQ4BCP4 样本程序中。

本主题还提供有关在以下标题下运行 Set 和 Inquire 消息属性样本的信息：

- 第 1103 页的『运行 CSQ4BCM1 样本』
- 第 1103 页的『运行 CSQ4BCM2 样本』

运行 CSQ4BCM1 样本

此程序最多采用四个参数：

1. 目标队列的名称（必需）。
2. 队列管理器名称（可选）。
3. 打开选项（可选）。
4. 关闭选项（可选）。

运行 CSQ4BCM2 样本

此程序最多采用六个参数：

1. 目标队列的名称（必需）。
2. 队列管理器名称（可选）。
3. 打开选项（可选）。
4. 关闭选项（可选）。
5. 目标队列管理器的名称（可选）。
6. 动态队列名称（可选）。

通过标准输入 (**SYSIN DD**) 提供属性名称、值和消息内容。 存在用于运行程序的样本 JCL，它驻留在 CSQ4BCMP 中。

为 Managed File Transfer 开发应用程序

指定要随 Managed File Transfer 一起运行的程序，将 Apache Ant 用于 Managed File Transfer，使用用户出口定制 Managed File Transfer，并通过在代理命令队列上放置消息来控制 Managed File Transfer。

指定要使用 MFT 运行的程序

您可在正在运行 Managed File Transfer Agent 的系统上运行程序。 作为文件传输请求的一部分，您可以指定在传输启动前或者完成后运行的程序。 此外，您可以通过提交受管呼叫请求来启动不属于文件传输请求的程序。

您可在以下五种场景下指定要运行的程序：

- 作为传输请求的一部分，在传输启动前从源代理处
- 作为传输请求的一部分，在传输启动前从目标代理处
- 作为传输请求的一部分，在传输完成后从源代理处
- 作为传输请求的一部分，在传输完成后从目标代理处
- 不作为传输请求的一部分。 您可向代理提交请求以运行程序。 本场景有时候称为受管呼叫。

用户出口和程序调用按以下顺序进行调用：

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

注意：

1. 仅当传输成功或部分成功完成时，才会运行 **DestinationTransferEndExits**。
2. 仅当传输成功或部分成功完成时，才会运行 **postDestinationCall**。
3. 针对成功、部分成功或失败的传输运行 **SourceTransferEndExits**。
4. 仅在以下情况下调用 **postSourceCall**：
   - 未取消该传输。
   - 存在成功或部分成功的结果。
   - 任何目标后传输程序已成功运行。

有多种方式来指定要运行的程序。 这些选项如下：

使用 Apache Ant 任务

使用 `fte:filecopy`、`fte:filemove` 和 `fte:call` Ant 任务之一来启动程序。 通过使用 Ant 任务，您可以使用 `fte:presrc`、`fte:predst`、`fte:postdst`、`fte:postsrc` 和 `fte:command` 嵌套元素以在五个场景中的任何一个场景中指定程序。 有关更多信息，请参阅程序调用嵌套元素。

编辑文件传输请求消息

您可以编辑传输请求生成的 XML。 通过使用此方法，您可以通过将 **preSourceCall**、**postSourceCall**、**preDestinationCall**、**postDestinationCall** 和 **managedCall** 元素添加到 XML 文件以在五个场景中的任何一个场景中运行程序。 然后，使用这一经过修改的 XML 文件作为新文件传输请求的传输定义，例如，使用 **fteCreateTransfer -td** 参数。 有关更多信息，请参阅 MFT 代理调用请求消息示例。

使用 fteCreateTransfer 命令

您可以使用 **fteCreateTransfer** 命令来指定要启动的程序。 您可以使用该命令来指定在前四个场景中作为传输请求的一部分运行的程序，但是您无法启动受管呼叫。 有关要使用的参数的信息，请参阅 **fteCreateTransfer**: 启动新的文件传输。 要获取使用此命令的示例，请参阅使用 fteCreateTransfer 来启动程序的示例。

相关参考

commandPath MFT 属性

受管调用

Managed File Transfer (MFT) 代理程序通常用来传输文件或消息。 它们称为受管传输。 代理程序也可用于运行命令、脚本或 JCL，而无需传输文件或消息。 此功能称为受管调用。

受管调用请求可以通过多种方式提交给代理程序：

- 使用 fte:call Ant 任务。
- 使用用于运行命令或脚本的任务 XML 配置资源监视器。 请参阅配置监视器任务以启动命令和脚本来获取更多信息。
- 直接将 XML 消息放入代理的命令队列中。 有关受管调用 XML 模式的更多详细信息，请参阅文件传输请求消息格式。

对于受管调用，必须在代理程序属性 **commandPath** 中指定包含要运行的命令或脚本的目录。

受管调用无法运行位于代理程序的 **commandPath** 中未指定的目录中的命令或脚本。 这是为了确保该代理程序不会运行任何恶意代码。

此外，您还可以对代理程序启用权限检查，以确保仅允许授权用户提交受管调用请求。 有关更多信息，请参阅限制针对 MFT 代理程序操作的用户权限。

作为受管调用的一部分调用的命令、脚本或 JCL 将作为外部进程运行，该进程由代理程序监视。 当进程退出时，受管调用完成，并且来自进程的返回码可供调用 **fte:call** Ant 任务的代理程序或 Ant 脚本使用。

如果受管调用是由 **fte:call** Ant 任务启动的，那么 Ant 脚本可以检查返回码的值以确定受管调用是否成功。

对于所有其他类型的受管调用，您可以指定应该使用哪些返回码值来指示受管调用已成功完成。 外部进程完成时，代理程序会将来自进程的返回码与这些返回码进行比较。

注: 由于受管调用作为外部进程运行，因此一旦启动，就无法取消这些调用。

受管调用和源传输插槽

代理程序包含多个源传输槽（由代理程序属性 **maxSourceTransfers** 指定）如高级代理程序属性：传输限制中所述。

每当运行受管调用或受管传输时，它们会占用源传输插槽。 该插槽在受管调用或受管传输完成时释放。

如果在代理程序收到新的受管调用或受管转移请求时所有源传输插槽都在使用中，则该请求将由代理程序排队，直到有可用插槽。

如果受管调用启动受管传输（例如，如果受管调用运行 Ant 脚本，并且 Ant 脚本使用 fte: filecopy 或 fte: filemove 任务来传输文件），那么需要两个源传输插槽：

- 一个用于受管传输
- 一个用于受管调用

在此情况下，请务必注意，如果受管传输需要很长时间才能完成或进入恢复状态，那么两个源传输插槽将被占用，直到受管传输完成、取消或由于 **transferRecoveryTimeout** 而超时（请参阅传输恢复超时概念以获取有关 **transferRecoveryTimeout** 的详细信息）。 这可能会限制代理程序可以处理的其他受管传输或受管调用的数量。

因此，您应该考虑受管调用的设计，以确保它在很长一段时间内不会占用源传输插槽。

将 Apache Ant 与 MFT 结合使用

Managed File Transfer 提供了一些任务，您可以使用这些任务将文件传输功能集成到 Apache Ant 工具中。

您可以使用 **fteAnt** 命令在已配置的 Managed File Transfer 环境中运行 Ant 任务。 您可以使用 Ant 脚本中的文件传输 Ant 任务，通过解释后的脚本语言来协调复杂的文件传输操作。

要了解有关 Apache Ant 的更多信息，请参阅 Apache Ant 项目 Web 页面：https://ant.apache.org/

相关概念

第 1106 页的『入门：将 Ant 脚本用于 MFT』
将 Ant 脚本用于 Managed File Transfer 能够使您通过已解释的脚本语言协调复杂的文件传输操作。

相关参考

fteAnt: 在 MFT 中运行 Ant 任务

第 1107 页的『MFT 的样本 Ant 任务』
安装 Managed File Transfer 时提供了许多样本 Ant 脚本。 这些样本位于目录 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 中。 每个样本脚本都包含一个 `init` 目标，请编辑 `init` 目标中设置的属性以使用您的配置运行这些脚本。

入门：将 Ant 脚本用于 MFT

将 Ant 脚本用于 Managed File Transfer 能够使您通过已解释的脚本语言协调复杂的文件传输操作。

Ant 脚本

Ant 脚本（或构建文件）是定义一个或多个目标的 XML 文档。 这些目标中包含要运行的元素。 Managed File Transfer 提供可用于将文件传输功能集成到 Apache Ant 中的任务。 要了解有关 Ant 脚本的信息，请参阅 Apache Ant 项目 Web 页面：https://ant.apache.org/

在目录 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 中随产品安装一起提供了使用 Managed File Transfer 任务的 Ant 脚本示例

在协议网桥代理上，Ant 脚本运行在协议网桥代理系统上。 这些 Ant 脚本不直接访问 FTP 或 SFTP 服务器上的文件。

名称空间

名称空间用于将文件传输 Ant 任务与其他可能享有同一名称的 Ant 任务区分开来。 您可以在自己的 Ant 脚本的 project 标记中定义名称空间。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

属性 `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` 指示 Ant 在 `com.ibm.wmqfte.ant.taskdefs` 库中查找前缀为 `fte` 的任务定义。

您不必将 `fte` 用作名称空间前缀；您可以使用任意值。 名称空间前缀 `fte` 用于所有示例和 Ant 脚本样本。

运行 Ant 脚本

要运行包含文件传输 Ant 任务的 Ant 脚本，请使用 **fteAnt** 命令。 例如：

```
fteAnt -file ant_script_location/ant_script_name
```

有关更多信息，请参阅 **fteAnt**: 在 MFT 中运行 Ant 任务。

返回码

文件传输 Ant 任务返回与 Managed File Transfer 命令相同的返回码。 有关更多信息，请参阅 MFT 的返回码。

相关参考

fteAnt: 在 MFT 中运行 Ant 任务

第 1107 页的『MFT 的样本 Ant 任务』

安装 Managed File Transfer 时提供了许多样本 Ant 脚本。 这些样本位于目录 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 中。 每个样本脚本都包含一个 init 目标，请编辑 init 目标中设置的属性以使用您的配置运行这些脚本。

MFT 的样本 Ant 任务

安装 Managed File Transfer 时提供了许多样本 Ant 脚本。 这些样本位于目录 *MQ_INSTALLATION_PATH*/mqft/samples/fteant 中。 每个样本脚本都包含一个 init 目标，请编辑 init 目标中设置的属性以使用您的配置运行这些脚本。

email

email 样本演示如何使用 Ant 任务传输文件以及在传输失败时如何将电子邮件发送到指定的电子邮件地址。 脚本将检查源和目标代理是否处于活动状态以及是否能够使用 Managed File Transfer ping 任务来处理传输。 如果两个代理都处于活动状态，那么脚本将使用 Managed File Transfer fte:filecopy 任务在源和目标代理之间传输文件，而不删除原始文件。 如果传输失败，那么脚本将使用标准的 Ant email 任务发送一封包含失败相关信息的电子邮件。

hub

hub 样本由以下两个脚本组成：hubcopy.xml 和 hubprocess.xml.。hubcopy.xml 脚本显示了如何使用 Ant 脚本编制来构建“中心和辐射”样式的拓扑。 在该样本中，将两个文件从辐条机器上运行的代理传输到集线器机器上运行的代理。 将同时传输这两个文件，在传输完成后，将在中心机器上运行 hubprocess.xml Ant 脚本来处理这些文件。 如果两个文件传输正确，那么 Ant 脚本将合并这些文件的内容。 如果文件未正确传输，那么 Ant 脚本将通过删除传输的所有文件数据来进行清理。 要使此示例正常工作，必须将 hubprocess.xml 脚本放在中心代理的命令路径上。 要获取有关设置代理的命令路径的更多信息，请参阅 commandPath MFT 属性。

librarytransfer（仅限 IBM i 平台）

IBM i

IBM i librarytransfer 样本演示如何使用 Ant 任务将一个 IBM i 系统上的 IBM i 库传输到第二个 IBM i 系统。

IBM i IBM i 上的 IBM WebSphere MQ File Transfer Edition 7.0.2 不包含对本机 IBM i 库对象传输的直接支持。 librarytransfer 样本将 IBM i 上的本机保存文件支持与 Managed File Transfer 中提供的预定义 Ant 任务配合使用，以在两个 IBM i 系统之间传输本机库对象。 此样本在 Managed File Transfer 文件复制任务中使用 <presrc> 嵌套元素来调用可执行脚本 librarysave.sh，然后将源代理系统上请求的库保存到临时保存文件中。 此保存文件由文件复制 Ant 任务移动到目标代理系统，其中使用 <postdst> 嵌套元素来调用可执行脚本 libraryrestore.sh，以将保存在保存文件中的库复原到目标系统。

IBM i 在运行此样本之前，需要完成 librarytransfer.xml 文件中描述的某些配置。 您还必须在两台 IBM i 机器上具有有效的 Managed File Transfer 环境。 该设置必须包含在第一个 IBM i 机器上运行的源代理和在第二个 IBM i 机器上运行的目标代理。 两个代理必须能够彼此通信。

IBM i librarytransfer 样本包含以下三个文件：

- librarytransfer.xml
- librarysave.sh（<presrc> 可执行脚本）
- libraryrestore.sh（<postdst> 可执行脚本）

这些样本文件位于以下目录中：/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer

IBM i 要运行该样本，用户必须完成以下步骤：

1. 启动 Qshell 会话。 在 IBM i 命令窗口中，输入：STRQSH
2. 将目录切换到 bin 目录，如下所示：

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. 在完成所需配置后，通过使用以下命令运行该样本：

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer（仅限 IBM i 平台）

IBM i physicalfiletransfer 样本演示如何使用 Ant 任务将源物理或数据库文件从一个 IBM i 系统上的库传输到另一个 IBM i 系统上的库。

IBM i IBM i 上的 IBM WebSphere MQ File Transfer Edition 7.0.2 不包含对 IBM i 上本机源物理文件或数据库文件传输的直接支持。 physicalfiletransfer 样本将 IBM i 上的本机保存文件支持与 Managed File Transfer 中可用的预定义 Ant 任务配合使用，以在两个 IBM i 系统之间传输完整的源物理文件和数据库文件。 此样本在 Managed File Transfer 文件复制任务中使用 <presrc> 嵌套元素来调用可执行脚本 physicalfilesave.sh，以将请求的源物理或数据库文件从源代理系统上的库保存到临时保存文件中。 此保存文件由文件复制 Ant 任务移动到目标代理系统，其中使用 <postdst> 嵌套元素来调用可执行脚本 physicalfilerestore.sh，然后将保存文件中的文件对象复原到目标系统上的指定库中。

IBM i 在运行此样本之前，必须完成 physicalfiletransfer.xml 文件中描述的某些配置。 您还必须在两个 IBM i 系统上具有有效的 Managed File Transfer 环境。 该设置必须包含在第一个 IBM i 系统上运行的源代理和在第二个 IBM i 系统上运行的目标代理。 两个代理必须能够彼此通信。

IBM i physicalfiletransfer 样本包含以下三个文件：

- physicalfiletransfer.xml
- physicalfilesave.sh（<presrc> 可执行脚本）
- physicalfilerestore.sh（<postdst> 可执行脚本）

这些样本文件位于以下目录中：/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer

IBM i 要运行该样本，用户必须完成以下步骤：

1. 启动 Qshell 会话。 在 IBM i 命令窗口中，输入：STRQSH
2. 将目录切换到 bin 目录，如下所示：

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. 在完成所需配置后，通过使用以下命令运行该样本：

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

timeout

timeout 样本演示如何使用 Ant 任务尝试文件传输，并在花费时间长于指定的超时值时取消传输。 该脚本通过使用 Managed File Transfer fte:filecopy 任务启动文件传输。 将延迟该传输的结果。 该脚本使用

Managed File Transfer fte:awaitoutcome Ant 任务等待指定的秒数使传输完成。 如果传输在指定的时间内没有完成，那么 Managed File Transfer fte:cancel Ant 任务将用于取消文件传输。

vsamtransfer

z/OS

z/OS vsamtransfer 样本演示如何使用 Ant 任务以通过使用 Managed File Transfer 从一个 VSAM 数据集传输到另一个 VSAM 数据集。 Managed File Transfer 当前不支持传输 VSAM 数据集。 通过使用 `presrc` 程序调用嵌套元素调用可执行文件 `datasetcopy.sh`，样本脚本会将 VSAM 数据记录卸载到连续数据集。 该脚本使用 Managed File Transfer fte:filemove 任务将连续数据集从源代理传输到目标代理。 然后，该脚本使用 `postdst` 程序调用嵌套元素来调用 `loadvsam.jcl` 脚本。 该 JCL 脚本会将传输的数据集记录装入目标 VSAM 数据集。 该样本对目标调用使用 JCL 以演示该语言选项。 通过使用第二个 shell 脚本也可以获得相同的结果。

z/OS 该样本不需要源和目标数据集是 VSAM。 如果源和目标数据集的类型相同，那么该样本对任何数据集都有效。

z/OS 要使此样本正常工作，必须将 `datasetcopy.sh` 脚本放在源代理的命令路径上，并将 `loadvsam.jcl` 脚本放在目标代理的命令路径上。 要获取有关设置代理的命令路径的更多信息，请参阅 commandPath MFT 属性。

zip

zip 样本由以下两个脚本组成：`zip.xml` 和 `zipfiles.xml`。 该样本演示在执行文件传输移动操作之前如何使用 Managed File Transfer fte: filemove 任务中的 `presrc` 嵌套元素来运行 Ant 脚本。 由 `zip.xml` 脚本中的 `presrc` 嵌套元素调用的 `zipfiles.xml` 脚本会压缩目录的内容。 `zip.xml` 脚本将传输压缩文件。 此样本要求 `zipfiles.xml` Ant 脚本存在于源代理的命令路径上。 这是因为 `zipfiles.xml` Ant 脚本包含用于压缩源代理上的目录内容的目标。 要获取有关设置代理的命令路径的更多信息，请参阅 commandPath MFT 属性。

使用用户出口定制 MFT

您可以通过使用称为用户出口例程的个人程序来定制 Managed File Transfer 的功能。

要点: IBM 不支持用户出口中的任何代码，该代码的任何问题都需要由您的企业或提供该出口的供应商进行初步调查。

Managed File Transfer 在代码中提供了一些“点”，Managed File Transfer 可以通过这些点向您已编写的程序（用户出口例程）传递控件。 这些“点”被称为用户出口点。 程序完成其工作后，Managed File Transfer 便可恢复控制。 您不一定要使用任何用户出口，但是，如果您希望扩展和定制 Managed File Transfer 系统的功能以满足您的特定需求，那么用户出口会很有用。

在文件传输处理期间，有两个“点”可用于调用源系统的用户出口，还有两个“点”可用于调用目标系统的用户出口。 下表概述了以下各个用户出口点以及要使用出口点而必须实现的 Java 接口。

表 *194:* 源端和目标端出口点和 *Java* 接口摘要

出口点	**要实现的 Java 接口**
源端出口点：	
整个文件传输开始前	SourceTransferStartExit.java 接口
整个文件传输完成后	SourceTransferEndExit.java 接口
目标端出口点：	
整个文件传输开始前	DestinationTransferStartExit.java 接口
整个文件传输完成后	DestinationTransferEndExit.java 接口

用户出口的调用顺序如下：

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

SourceTransferStartExit 出口和 DestinationTransferStartExit 出口产生的更改会作为对后续出口的输入而传播。 例如，如果 SourceTransferStartExit 出口修改了传输元数据，这些更改会反映在对其他出口的输入传输元数据中。

用户出口和程序调用按以下顺序进行调用：

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

注意：

1. 仅当传输成功或部分成功完成时，才会运行 **DestinationTransferEndExits**。
2. 仅当传输成功或部分成功完成时，才会运行 **postDestinationCall**。
3. 针对成功、部分成功或失败的传输运行 **SourceTransferEndExits**。
4. 仅在以下情况下调用 **postSourceCall**：
   - 未取消该传输。
   - 存在成功或部分成功的结果。
   - 任何目标后传输程序已成功运行。

构建用户出口

用于构建用户出口的接口包含在 *MQ_INSTALL_DIRECTORY*/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar 中。 构建出口时，您必须在类路径中包含该 .jar 文件。 要运行出口，请作为 .jar 文件抽取出口，然后将该 .jar 文件置于后面部分中描述的目录中。

用户出口位置

您可以在以下两个可能位置中存储用户出口例程：

- exits 目录。 每个代理目录下存在一个 exits 目录。 例如：
  var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits
- 您可以设置 exitClassPath 属性来指定一个备用位置。 如果 exits 目录和 exitClassPath 设置的类路径中存在出口类，那么 exits 目录中的类优先，这意味着如果两个位置中有同名的类，那么 exits 目录中的类优先。

配置代理以使用用户出口

有四种代理属性可进行设置用以指定代理调用的用户出口。 这些代理程序属性为 sourceTransferStartExitClasses、sourceTransferEndExitClasses、destinationTransferStartExitClasses 和 destinationTransferEndExitClasses。 有关如何使用这些属性的信息，请参阅用户出口的 MFT 代理属性。

在协议网桥代理上运行用户出口

当源代理调用出口时，它会向出口传递要传输的源项目列表。 对于一般代理，这是标准文件名列表。 因为这些文件应为本地文件（或可通过安装进行访问），所以出口能够对其进行访问和加密。

但是，对于协议网桥代理，列表中的条目为以下格式：

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

对于列表中的每个条目，出口需要首先连接到文件服务器（使用 FTP、 FTPS 或 SFTP 协议）、下载文件、在本地对其加密，然后将加密后的文件重新上载到文件服务器。

在 Connect:Direct 网桥代理上运行用户出口

不能在 Connect:Direct 网桥代理上运行用户出口。

MFT 源和目标用户出口

目录分隔符

无论在 **fteCreateTransfer** 命令或 IBM MQ Explorer 中如何指定目录分隔符， 源文件规范中的目录分隔符始终使用正斜杠 (/) 字符表示。 编写出口时必须将此情况考虑在内。 例如，如果要检查是否存在源文件 `c:\a\b.txt`，并且已使用 **fteCreateTransfer** 命令或 IBM MQ Explorer 指定该源文件，那么请注意，该文件名实际存储为 `c:/a/b.txt`。因此，如果您搜索原始字符串 `c:\a\b.txt`，那么将找不到匹配项。

源端出口点

整个文件传输开始前

如果暂挂传输列表的下一步是传输请求，并且传输即将启动，那么由源代理调用该出口。

该出口点的示例用途是，使用外部命令分阶段将文件发送至该代理具有读/写访问权的目录，或者是对目标系统上的文件进行重命名。

将以下自变量传递到该出口：

- 源代理名称
- 目标代理名称
- 环境元数据
- 传输元数据
- 文件规范（包含文件元数据）

从该出口返回的数据如下：

- 已更新的传输元数据。 可添加、修改和删除条目。
- 已更新的文件规范列表，其中包含源文件和目标文件名称对。 可添加、修改和删除条目
- 指示符，指示是否继续传输
- 字符串，用于插入“传输日志”。

实现 SourceTransferStartExit.java 接口，以在该出口点调用用户出口代码。

整个文件传输完成后

整个文件传输完成后，会由源代理调用该出口。

下面是该出口点的使用示例：执行某些完成任务，例如发送电子邮件或 IBM MQ 消息以标记传输已完成。

将以下自变量传递到该出口：

- 传输出口结果
- 源代理名称
- 目标代理名称
- 环境元数据
- 传输元数据
- 文件结果

从该出口返回的数据如下：

- 已更新的字符串，以用于插入“传输日志”。

实现 SourceTransferEndExit.java 接口，以在该出口点调用用户出口代码。

目标端出口点

整个文件传输开始前

使用该出口点的示例为：验证位于目标的许可权。

将以下自变量传递到该出口：

- 源代理名称
- 目标代理名称
- 环境元数据
- 传输元数据
- 文件规范

从该出口返回的数据如下：

- 已更新的目标文件名集。 可修改条目，但是不可添加或删除条目。
- 指示符，指示是否继续传输
- 字符串，用于插入“传输日志”。

实现 DestinationTransferStartExit.java 接口，以在该出口点调用用户出口代码。

整个文件传输完成后

使用该用户出口的示例为：启动使用已传输的文件的批处理过程，或者传输失败时发送电子邮件。

将以下自变量传递到该出口：

- 传输出口结果
- 源代理名称
- 目标代理名称
- 环境元数据
- 传输元数据
- 文件结果

从该出口返回的数据如下：

- 已更新的字符串，以用于插入“传输日志”。

实现 DestinationTransferEndExit.java 接口，以在该出口点调用用户出口代码。

相关概念

MFT 用户出口的 Java 接口

相关参考

第 1115 页的『为 MFT 用户出口启用远程调试』
在开发用户出口时，您可能希望使用调试器来帮助找到代码中的问题。

第 1115 页的『MFT 源传输用户出口样本』

MFT 资源监视器用户出口

使用 MFT 传输 I/O 用户出口

您可以使用 Managed File Transfer 传输 I/O 用户出口来配置定制代码，以执行 Managed File Transfer 传输的的底层文件系统 I/O 工作。

通常对于 MFT 传输，代理会选择某个内置的 I/O 提供程序，以便与用于传输的相应文件系统进行交互。 内置 I/O 提供程序支持以下类型的文件系统：

- 常规 UNIX 类型和 Windows 类型文件系统
- z/OS z/OS 连续的或分区数据集（仅限 z/OS 上）
- IBM i IBM i 本机保存文件（仅限 IBM i 上）
- IBM MQ 队列
- 远程 FTP 和 SFTP 协议服务器（仅用于协议网桥代理）
- 远程 Connect:Direct 节点（仅限 Connect:Direct 网桥代理）

对于不受支持的文件系统或者其中需要定制 I/O 行为的文件系统，那么您可以编写一个传输 I/O 用户出口。

传输 I/O 用户出口将现有基础结构用于用户出口。 但是，这些传输 I/O 用户出口与其他用户出口不同，因为在整个传输中，将对每个文件多次访问这些传输 I/O 用户出口的功能。

使用代理属性 IOExitClasses（在 `agent.properties` 文件中）指定要装入的 I/O 出口类。 用逗号来分隔每个出口类，例如：

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

传输 I/O 用户出口的 Java 接口如下所示：

IOExit
主入口点，用以确定是否使用 I/O 出口。 此实例负责生成 IOExitPath 实例。

您只需为代理属性 IOExitClasses 指定 IOExit I/O 出口接口。

IOExitPath
表示一个抽象接口；例如，某个数据容器或者表示一组数据容器的通配符。 您无法创建实现此接口的类实例。 该接口允许对路径进行检查并列出派生的路径。 IOExitResourcePath 和 IOExitWildcardPath 接口扩展了 IOExitPath。

IOExitChannel
使数据可以在 IOExitPath 资源上读取和写入。

IOExitRecordChannel
为面向记录的 IOExitPath 资源扩展 IOExitChannel 接口，这使得能够在多个记录中的 IOExitPath 资源上读取和写入数据。

IOExitLock
表示对 IOExitPath 资源的锁定以进行共享或互斥存取。

z/OS **IOExitRecordResourcePath**
扩展 IOExitResourcePath 接口以表示面向记录的文件的数据容器；例如，z/OS 数据集。 您可以使用该接口来查找数据，从而为读/写操作创建 IOExitRecordChannel 实例。

IOExitResourcePath
扩展 IOExitPath 接口以表示数据容器；例如，文件或目录。 您可以使用此接口来查找数据。 如果此接口表示目录，那么您可以使用 listPaths 方法来返回路径的列表。

IOExitWildcardPath
扩展 IOExitPath 接口以表示指示通配符的路径。 您可以使用此接口来匹配多个 IOExitResourcePaths。

IOExitProperties
指定用于确定 Managed File Transfer 如何处理某些 I/O 方面的 IOExitPath 的属性。 例如，是否使用中间文件，或是否在重新启动传输时从开始重新读取资源。

IBM i IBM i 用户出口上的样本 MFT

Managed File Transfer 随安装文件一起提供了特定于 IBM i 的样本用户出口。 样本位于目录 *MQMFT_install_dir*/samples/ioexit-IBMi 和 *MQMFT_install_dir*/samples/userexit-IBMi 中。

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit 样本用户出口传输 IBM i 上 QDLS 文件系统中的文件。 在安装出口后，对以 /QDLS 开头的文件的任何传输都会自动使用该出口。

要安装该出口，请完成以下步骤：

1. 将 com.ibm.wmqfte.samples.ibmi.ioexits.jar 文件从 *WMQFTE_install_dir*/samples/ioexit-IBMi 目录复制到代理程序的 exits 目录。
2. 将 com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit 添加到 IOExitClasses 属性。
3. 重新启动代理。

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit 样本用户出口的行为方式类似于 MFT 文件监视器，它自动传输 IBM i 库中的物理文件成员。

要运行此出口，请指定“library.qsys.monitor”元数据字段的值（例如，使用 **-md** 参数）。 该参数采用文件成员的 IFS 样式路径，并且可以包含文件和成员通配符。 例如，/QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR、/QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR、/QSYS.LIB/FOO.LIB/*.FILE/*.MBR。

该样本出口还具有一个可选元数据字段“naming.scheme.qsys.monitor”，此字段可用于确定传输期间使用的命名方案。 缺省情况下，此字段设置为“unix”，这会导致将目标文件称为 F00.MBR。 您还可以指定值 "ibmi" 以使用 IBM i FTP FILE.MEMBER 方案，例如 /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR 作为 BAR.BAZ。

要安装该出口，请完成以下步骤：

1. 将 com.ibm.wmqfte.samples.ibmi.userexits.jar 文件从 *WMQFTE_install_dir*/samples/userexit-IBMi 目录复制到代理程序的 exits 目录。
2. 将 com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit 添加到 agent.properties 文件中的 sourceTransferStartExitClasses 属性。
3. 重新启动代理。

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

当在传输期间删除源文件成员时，com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit 样本用户出口将删除空文件对象。 因为 IBM i 文件对象可能保存许多成员，所以 MFT 将文件对象视为目录处理。 所以，无法使用 MFT 对文件对象执行移动操作；只有在成员级别才支持移动操作。 因此，在对成员执行移动操作时，当前的空文件会遗留下来。 如果希望在传输请求期间删除这些空文件，请使用该样本出口。

当为“empty.file.delete”元数据字段指定“true”并传输 FTEFileMember 时，如果父文件为空，该样本出口将删除该文件。

要安装该出口，请完成以下步骤：

1. 将 com.ibm.wmqfte.samples.ibmi.userexits.jar 文件从 *WMQFTE_install_dir*/samples/userexit-IBMi 复制到代理程序的 exits 目录。
2. 将 com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit 添加到 agent.properties 文件中的 sourceTransferStartExitClasses 属性。
3. 重新启动代理。

相关参考

第 1112 页的『使用 MFT 传输 I/O 用户出口』
您可以使用 Managed File Transfer 传输 I/O 用户出口来配置定制代码，以执行 Managed File Transfer 传输的的底层文件系统 I/O 工作。

用户出口的 MFT 代理属性

为 MFT 用户出口启用远程调试

在开发用户出口时，您可能希望使用调试器来帮助找到代码中的问题。

由于出口在运行代理的 Java 虚拟机中运行，因此您无法直接使用调试支持（通常包含在集成开发环境中）。 但是，您可以启用 JVM 的远程调试，然后连接一个合适的远程调试器。

要启用远程调试，请使用标准 JVM 参数 **-Xdebug** 和 **-Xrunjdwp**。 这些属性将通过 **BFG_JVM_PROPERTIES** 环境变量传递给运行代理程序的 JVM。 例如，在 UNIX 上，以下命令将启动代理，并使 JVM 在 TCP 端口 8765 上侦听调试器连接。

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

在调试器连接之前，代理不会启动。 在 Windows 上使用 **set** 命令，而不是 **export** 命令。

您也可以在调试器和 JVM 之间使用其他通信方法。 例如，JVM 可以打开与调试器的连接，反之则不可，或者您也可以使用共享内存来代替 TCP。 请参阅 Java 平台调试器体系结构文档获取更多详细信息。

在以远程调试方式启动代理程序时，必须使用 **-F**（前台）参数。

使用 Eclipse 调试器

以下步骤适用于 Eclipse 开发环境中的远程调试功能。 您还可以使用与 JPDA 兼容的其他远程调试器。

1. 单击**运行 > 打开调试对话框**（或**运行 > 调试配置**或**运行 > 调试对话框**，具体取决于 Eclipse 的版本）。
2. 在配置类型列表中双击**远程 Java 应用程序**以创建一个调试配置。
3. 填写配置字段，然后保存调试配置。 如果已通过调试方式启动了代理 JVM，那么现在可以连接到 JVM。

MFT 源传输用户出口样本

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {
```

```
    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

协议网桥凭证用户出口样本

有关如何使用该样本用户出口的信息，请参阅使用出口类映射文件服务器的凭证。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
```

```
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
                 inputStream = new FileInputStream(propertiesFile);

                 // Load the properties from the file
                 mappingProperties.load(inputStream);
             }
             catch (FileNotFoundException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                 initialisationResult = false;
             }
             catch (IOException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                 initialisationResult = false;
             }
             finally {
                 // Close the inputStream
                 if (inputStream != null) {
                     try {
                         inputStream.close();
                     }
                     catch (IOException ex) {
                         System.err.println("Error initializing SampleCredentialExit.");
                          System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                         initialisationResult = false;
                     }
                 }
             }

             if (initialisationResult) {
                 // Populate the map of mqUserId to server credentials from the properties
                 final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                  while ( propertyNames.hasMoreElements()) {
                     final Object name = propertyNames.nextElement();
```

```
                    if (name instanceof String ) {
                        final String mqUserId = ((String)name).trim();
                         // Get the value and split into serverUserId and serverPassword
                       final String value = mappingProperties.getProperty(mqUserId);
                       final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                        String serverUserId = "";
                        String serverPassword = "";
                        if (valueTokenizer.hasMoreTokens()) {
                            serverUserId = valueTokenizer.nextToken().trim();
                        }
                        if (valueTokenizer.hasMoreTokens()) {
                            serverPassword = valueTokenizer.nextToken().trim();
                        }
                        // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                        // Insert the credentials into the map
                        credentialsMap.put(mqUserId, credentials);
                    }
                }

            }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

协议网桥属性用户出口样本

有关如何使用该样本用户出口的信息，请参阅 ProtocolBridgePropertiesExit2：查找协议文件服务器属性

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
```

```
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
```

```
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
```

```
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

通过将消息放置在代理命令队列中来控制 MFT

您可编写一个应用程序，通过将消息放置在代理命令队列中来控制 Managed File Transfer。

您可以将一条消息放置在代理的命令队列中，以请求代理执行以下某项操作：

- 创建一次文件传输
- 创建一次计划的文件传输
- 取消一次文件传输
- 取消一次计划的文件传输
- 调用一个命令
- 创建一个监视器
- 删除一个监视器
- 返回 ping 以表明代理处于活动状态

要请求代理执行这些操作之一，消息必须是采用以下某种模式编译的 XML 格式：

FileTransfer.xsd
这种格式的消息可用于创建文件传输或计划的文件传输，调用命令，或者取消文件传输或计划的文件传输。 有关更多信息，请参阅文件传输请求消息格式。

Monitor.xsd
这种格式的消息可用于创建或删除资源监视器。 有关更多信息，请参阅 MFT 监视器请求消息格式。

PingAgent.xsd
这种格式的消息可用于 ping 代理，以检查其是否处于活动状态。 有关更多信息，请参阅 Ping MFT 代理请求消息格式。

该代理返回对请求消息的应答。 应答消息会放入在请求消息中定义的应答队列。 应答消息采用通过以下模式定义的 XML 格式：

Reply.xsd
有关更多信息，请参阅 MFT 代理回复消息格式。

为 MQ Telemetry 开发应用程序

遥测应用程序将检测设备和控制设备与因特网上和企业内提供的其他信息源集成。

使用设计模式、工作示例、样本程序、编程概念以及参考信息为 MQ Telemetry 开发应用程序。

相关概念
MQ Telemetry
遥测用例
相关任务
安装 MQ Telemetry
管理 MQ Telemetry
MQ Telemetry 故障诊断
相关参考
MQ Telemetry 参考

IBM MQ Telemetry Transport 样本程序

提供的样本脚本用于处理样本 IBM MQ Telemetry Transport v3 客户机应用程序 (`mqttv3app.jar`)。对于 IBM MQ 8.0.0 和更高版本，样本客户机应用程序不再包含在 MQ Telemetry 中。 它是（不再可用）IBM Messaging Telemetry Clients SupportPac 的一部分。 可以继续从 Eclipse Paho 和 MQTT.org 免费获取类似的样本应用程序。

有关最新信息和下载，请参阅以下资源：

- Eclipse Paho 项目和 MQTT.org 具有可免费下载的最新遥测客户机和一系列编程语言的样本。 使用这些站点可帮助您开发样本程序，以便发布和预订 IBM MQ Telemetry Transport，以及添加安全性功能。
- IBM Messaging Telemetry Clients SupportPac 不再可供下载。 如果具有先前下载的副本，那么其中包含以下内容：
  - IBM Messaging Telemetry Clients SupportPac 的 MA9B 版本包含了已编译的样本应用程序 (`mqttv3app.jar`) 和关联的客户机库 (`mqttv3.jar`)。它们已在以下目录中提供：
    - `ma9c/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9c/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - 在此 SupportPac 的 MA9C 版本中，已除去 `/SDK/` 目录和内容：
    - 仅提供了样本应用程序 (`mqttv3app.jar`) 的源。 它位于以下目录中：

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - 仍然提供了已编译的客户机库。 它位于以下目录中：

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

如果您仍有（不再可用的）IBM Messaging Telemetry Clients SupportPac 的副本，那么在使用命令行验证 MQ Telemetry 安装中提供了有关安装和运行样本应用程序的信息。

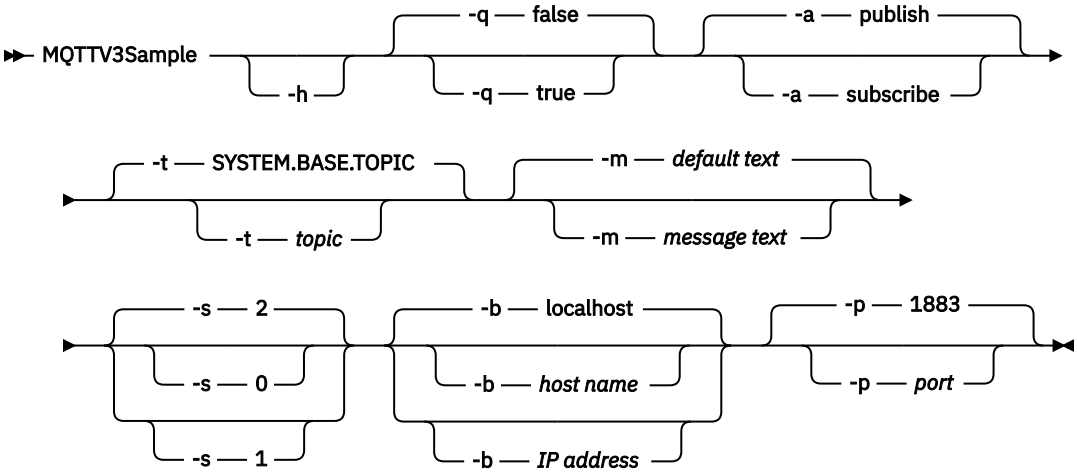
MQTTV3Sample 程序

提供了有关 `MQTTV3Sample` 程序的样本语法和参数的参考信息。

用途

`MQTTV3Sample` 程序可以用来发布消息和预订主题。 有关如何获取此样本程序的信息，请参阅第 1122 页的『IBM MQ Telemetry Transport 样本程序』。

MQTTV3Sample syntax



参数

-h
打印此帮助文本并退出

-q
设置静默方式，而不是使用缺省方式 false。

-a
设置“发布”或“预订”，而不是使用缺省操作“发布”。

-t
发布或预订主题，而不是发布或预订缺省主题

-m
发布消息文本，而不是发送缺省发布文本“Hello from an MQTT v3 application”。

-s
设置 QoS，而不是使用缺省 QoS（即 2）。

-b
连接到此主机名或 IP 地址，而不是连接到缺省主机名 localhost。

-p
使用此端口，而不是使用缺省端口 1883。

运行 MQTTV3Sample 程序

要在 Windows 上预订主题，请使用以下命令：

```
run MQTTV3Sample -a subscribe
```

要在 Windows 上发布消息，请使用以下命令：

```
run MQTTV3Sample
```

MQTT 客户机编程概念

本部分中所描述的概念可帮助您了解 MQTT protocol 的客户机库。 这些概念是对随客户机库一起提供的 API 文档的补充。

有关最新信息和下载，请参阅以下资源：

- Eclipse Paho 项目和 MQTT.org 具有可免费下载的最新遥测客户机和一系列编程语言的样本。 使用这些站点可帮助您开发样本程序，以便发布和预订 IBM MQ Telemetry Transport，以及添加安全性功能。

- IBM Messaging Telemetry Clients SupportPac 不再可供下载。 如果具有先前下载的副本，那么其中包含以下内容：
  - IBM Messaging Telemetry Clients SupportPac 的 MA9B 版本包含了已编译的样本应用程序 (mqttv3app.jar) 和关联的客户机库 (mqttv3.jar)。它们已在以下目录中提供：
    - ma9c/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar
    - ma9c/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar
  - 在此 SupportPac 的 MA9C 版本中，已除去 /SDK/ 目录和内容：
    - 仅提供了样本应用程序 (mqttv3app.jar) 的源。 它位于以下目录中：

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - 仍然提供了已编译的客户机库。 它位于以下目录中：

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

要开发和运行 MQTT 客户机，需要在客户机设备上复制或安装这些资源。 您无需安装独立的客户机运行时。

客户机的许可条件与其连接的服务器相关。

MQTT 客户机库是 MQTT protocol 的参考实施。 可以使用不同的语言来实现您自己的适合于不同设备平台的客户机。 请参阅 IBM MQ Telemetry Transport 格式和协议。

API 文档对客户机连接到哪个 MQTT 服务器不做任何假定。 连接到不同的服务器时，客户机的行为可能略有不同。 以下描述介绍了客户机在连接 IBM MQ 遥测服务时的行为。

MQTT 客户机应用程序中的回调和同步

MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

回调

注: 请访问 Eclipse Paho Web 站点，以了解 MqttCallback 的最新更改。 例如，MqttCallback 在 Paho 版本的客户机中定义为接口，并且异步方法由 Paho MqttAsyncClient 类提供。

MqttCallback 接口有三种回调方法：

connectionLost(java.lang.Throwable cause)

当通信错误导致连接断开时，就会调用 connectionLost。 在已经建立连接之后，如果由于服务器上发生错误而导致服务器断开连接，那么也会调用此方法。 服务器错误被记录到队列管理器错误日志中。 服务器将断开与客户机的连接，并且客户机将调用 MqttCallback.connectionLost。

在客户机应用程序所在的同一线程上作为异常抛出的唯一远程错误就是 MqttClient.connect 产生的异常。 建立连接后服务器检测到的错误将作为 throwables 报告回 MqttCallback.connectionLost 回调方法。

导致 connectionLost 的典型服务器错误是权限错误。 例如，遥测服务器试图代表一个未被授权发布主题的客户机来发布主题。 导致将 MQCC_FAIL 条件代码返回到遥测服务器的任何情况都会导致断开连接。

deliveryComplete(IMqttDeliveryToken token)

deliveryComplete 由 MQTT 客户机调用以将传递令牌传回客户机应用程序；请参阅第 1129 页的『传递令牌』。 通过使用传递令牌，回调可以使用 token.getMessage 方法来访问已发布的消息。

当应用程序回调在被 deliveryComplete 方法调用后将控制权返回给 MQTT 客户机时，将完成传递。 在完成传递之前，持久类将保留服务质量为 QoS 1 或 QoS 2 的消息。

调用 deliveryComplete 是应用程序与持久性类之间的同步点。 决不会对同一条消息两次调用 deliveryComplete 方法。

当应用程序回调从 deliveryComplete 返回到 MQTT 客户机时，客户机会调用 MqttClientPersistence.remove 以获取 QoS 为 1 或 2 的消息。 MqttClientPersistence.remove 删除已发布消息的本地存储副本。

从事务处理角度来说，调用 deliveryComplete 是用于落实传递的单阶段事务。 如果在回调期间处理失败，那么在重新启动客户机时，将再次调用 MqttClientPersistence.remove，以删除已发布的消息的本地副本。 不会再次调用此回调。 如果要使用回调来存储已传递的消息的日志，那么将无法使该日志与 MQTT 客户机同步。 如果您想可靠地存储日志，请更新 MqttClientPersistence 类中的日志。

传递令牌和消息由主应用程序线程和 MQTT 客户机引用。 完成传递后，MQTT 客户机将取消引用 MqttMessage 对象；客户机断开连接时，它将取消引用传递令牌对象。 在完成传递之后，如果客户机应用程序取消对 MqttMessage 对象的引用，那么可以对此对象进行垃圾回收。 在会话断开连接之后，可以对传递令牌进行垃圾回收。

在发布消息后，您可以获取 IMqttDeliveryToken 和 MqttMessage 属性。 如果您在已经发布消息之后尝试设置任何 MqttMessage 属性，那么结果是不确定的。

如果客户机重新连接到具有相同 ClientIdentifier 的先前会话，那么 MQTT 客户机将继续处理传递应答; 请参阅 第 1127 页的『清除会话』。 MQTT 客户机应用程序必须针对先前的会话将 MqttClient.CleanSession 设置为 false，并在新会话中将其设置为 false。 MQTT 客户机可在新会话中创建新的传递令牌和消息对象，以供暂挂的传递使用。 它将使用 MqttClientPersistence 类来恢复对象。 如果应用程序客户机仍然引用了旧的传递令牌和消息，请取消对它们的引用。 对于在先前会话中启动的以及在此会话中完成的任何传递，会在这些传递的新会话中调用应用程序回调。

在应用程序客户机连接之后，当完成暂挂的传递时，就会调用应用程序回调。 在应用程序客户机连接之前，它可以使用 MqttClient.getPendingDeliveryTokens 方法来检索暂挂的传递。

请注意，客户机应用程序最初创建了已发布的消息对象及其有效内容字节数组。 MQTT 客户机将引用这些对象。 由 token.getMessage 方法中的传递令牌返回的消息对象不一定是客户机所创建的同一消息对象。 如果新的 MQTT 客户机实例重新创建了传递令牌，那么 MqttClientPersistence 类将重新创建 MqttMessage 对象。 为了保持一致性，无论消息对象是由应用程序客户机还是由 MqttClientPersistence 类创建的，如果 token.isCompleted 为 true，那么 token.getMessage 将返回 null。

messageArrived(String topic, MqttMessage message)

当针对与预订主题匹配的客户机的发布到达时，将调用 messageArrived 。 topic 是发布主题，而不是预订过滤器。 如果过滤器中包含通配符，那么这两者可能不同。

如果此主题与客户机所创建的多个预订相匹配，那么客户机将收到此发布的多个副本。 如果客户机发布至它也预订了的某个主题，那么它将收到它自己的发布的副本。

如果使用 QoS 1 或 2 发送消息，那么在 MQTT 客户机调用 messageArrived 之前，消息由 MqttClientPersistence 类存储。 messageArrived 行为类似于 deliveryComplete: 仅对发布调用一次，当 messageArrived 返回到 MQTT 客户机时， MqttClientPersistence.remove 将除去该发布的本地副本。 当 messageArrived 返回到 MQTT 客户机时， MQTT 客户机会删除其对主题和消息的引用。 如果应用程序客户机尚不具备对对象的引用，那么会对主题和消息对象进行垃圾回收。

回调、线程技术和客户机应用程序同步

MQTT 客户机会将独立线程上的回调方法调用到主应用程序线程。 客户机应用程序不会创建回调线程，它由 MQTT 客户机创建。

MQTT 客户机将同步回调方法。 一次只有回调方法的一个实例运行。 通过同步，很容易更新一个用于清点已经传递了哪些发布的对象。 一次只运行 MqttCallback.deliveryComplete 的一个实例，因此，更新记录而不进一步进行同步是安全的。 一次只有一个发布到达也是这种情况。 messageArrived 方法中的代码可以更新对象而不使此对象同步。 如果您正在另一个线程中引用记录或者要更新的对象，那么使此记录或对象同步。

传递令牌提供了主应用程序线程与发布的传递之间的同步机制。 token.waitForCompletion 方法将一直等到完成了传递特定发布，或者等到可选超时已到期。 可以通过以下方式使用 token.waitForCompletion 来一次处理一个发布。

与 MqttCallback.deliveryComplete 方法同步。 只有当 MqttCallback.deliveryComplete 返回到 MQTT 客户机时，token.waitForCompletion 才会恢复。 通过使用此机制，在代码运行于主应用程序线程中之前，可以使 MqttCallback.deliveryComplete 中正在运行的代码同步。

如果您想进行发布而不等待传递每个发布，但您想确认何时已经传递了所有发布，情况会怎样呢？ 如果您在单个线程上发布，那么要发送的最后一个发布也是要传递的最后一个发布。

使发送至服务器的请求同步

第 1126 页的表 195 描述 MQTT Java 客户机中向服务器发送请求的方法。 除非应用程序客户机设置了无限期超时，否则客户机决不会无限期等待服务器的确认信息。 如果客户机暂停，那么可能是由于应用程序编程出现问题或者 MQTT 客户机存在缺陷。

表 *195:* 导致将请求发送至服务器的方法的同步行为

方法	同步	超时时间间隔
MqttClient.Connect	等待与服务器建立连接。	缺省设置为 30 秒，或者由参数进行设置，然后抛出异常。
MqttClient.Disconnect	等待 MQTT 客户机完成所有必需的工作，并等待 TCP/IP 会话断开连接。	
MqttClient.Subscribe	等待完成 Subscribe 或 UnSubscribe 方法。	
MqttClient.UnSubscribe		
MqttClient.Publish	将请求传递到 MQTT 客户机后，立即返回到应用程序线程。	无。
IMqttDeliveryToken.waitForCompletion	等待返回传递令牌。	无限期，或者作为参数来设置。

相关概念

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 MqttConnectOptions.cleanSession 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量

MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

清除会话

MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 `MqttConnectOptions.cleanSession` 来更改清除会话方式。

使用 `MqttClient.connect` 方法连接 MQTT 客户机应用程序时，客户机使用客户机标识和服务器地址来标识连接。 服务器将检查是否保存了先前与服务器建立连接时使用的会话信息。 如果先前的会话仍然存在，并且 `cleanSession=true`，那么将清除客户机和服务器中先前的会话信息。 如果 `cleanSession=false`，那么先前的会话将继续。 如果不存在先前的会话，那么将启动新的会话。

注: IBM MQ 管理员可以强制关闭打开的会话，并可删除所有会话信息。 如果客户机重新打开会话并且 `cleanSession=false`，那么将启动新的会话。

出版物

如果您使用缺省 `MqttConnectOptions`，或者在连接客户机之前将 `MqttConnectOptions.cleanSession` 设置为 `true`，那么在客户机建立连接时，将除去为客户机传递的所有暂挂的发布。

“清除会话”设置对于使用 `QoS=0` 发送的发布不起作用。 对于 `QoS=1` 和 `QoS=2`，使用 `cleanSession=true` 可能会导致丢失发布。

预订

如果在连接客户机之前使用缺省 `MqttConnectOptions` 或将 `MqttConnectOptions.cleanSession` 设置为 `true` ，那么在客户机连接时将除去客户机的任何旧预订。 当客户机断开连接时，将移除客户机在会话期间创建的所有新预订。

如果在连接之前将 `MqttConnectOptions.cleanSession` 设置为 `false` ，那么客户机创建的任何预订都将添加到客户机在连接之前已存在的所有预订。 当客户机断开连接时，所有预订仍保持活动状态。

要了解 `cleanSession` 属性影响预订的方式，另一种方法就是将它视作模态属性。 在其缺省方式 `cleanSession=true` 下，客户机仅在会话的作用域内创建预订和接收发布。 在另一种方式 `cleanSession=false` 下，预订是持久预订。 客户机可以连接和断开连接，而其预订保持活动状态。 当客户机重新连接时，它将接收任何未传递的发布。 在它连接之后，它可以自己修改处于活动状态的预订集。

必须先设置 `cleanSession` 方式，然后再进行连接; 该方式将持续整个会话。 要更改此属性的设置，必须将客户机断开连接，然后再重新连接客户机。 如果将方式从使用 `cleanSession=false` 更改为 `cleanSession=true`，那么将废弃客户机的所有先前预订以及尚未接收的任何发布。

相关概念
MQTT 客户机应用程序中的回调和同步

MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

客户机标识

客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

客户机标识在 MQTT 系统管理过程中使用。 由于可能要管理许多客户机，因此您需要能够快速标识特定客户机。 例如，如果某个设备发生故障，并且您已收到通知，那么可能是客户联系了帮助热线。 客户需要能够识别设备，并且您需要能够将该标识与通常连接到客户机的服务器相关联。

当您浏览 MQTT 客户机连接时，每个连接都会使用客户机标识进行标记。 要帮助确定将此标识映射到设备和服务器的最佳方式，请自我询问以下问题：

- 维护和使用用于将每个设备映射到客户机标识和服务器的数据库是否方便?
- 设备名称能否识别它所连接的服务器?
- 是否需要用于将客户机标识映射到物理设备的查找表?
- 客户机标识将标识特定设备、用户或者在客户机中运行的应用程序吗?
- 如果客户将故障设备替换为新设备，那么新设备是否具有与旧设备相同的标识，或者是否分配了新标识?（如果更改了物理设备并保留相同的标识，那么会自动将未完成的发布和活动预订传输到新设备。）

您还需要一个系统以确保客户机标识是唯一的，并且您必须使用一个可靠的进程来设置客户机的标识。 如果客户机设备是一个没有用户界面的“黑匣”，那么可以使用客户机标识来制造设备，也可以在激活设备之前使用软件安装和配置过程来配置设备。

要使标识保持简短且唯一，可以使用 48 位设备 MAC 地址来创建客户机标识。 如果传输大小不是关键问题，那么可以使用剩余的 17 字节来简化地址管理。

相关概念

MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 `MqttCallback` 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 `MqttConnectOptions.cleanSession` 来更改清除会话方式。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

传递令牌

当客户机在主题上发布内容时，将创建新的传递令牌。 使用传递令牌来监控发布的传递，或者阻止客户机应用程序，直到完成传递为止。

令牌是一个 `MqttDeliveryToken` 对象。 它通过调用 MqttTopic.publish() 方法来创建并由 MQTT 客户机保留，直到客户机会话断开连接且完成传递为止。

令牌的常规用法是检查是否已完成传递。 通过使用所返回的令牌来调用 `token.waitForCompletion`，从而阻塞客户机应用程序直到已完成传递为止。 或者，提供 `MqttCallBack` 处理程序。 当 MQTT 客户机接收到其期望作为传递发布的一部分的所有应答后，它会调用 `MqttCallBack.deliveryComplete`，将传递令牌作为参数进行传递。

在完成传递之前，可以通过调用 `token.getMessage` 从而使用所返回的传递令牌来检查发布。

已完成传递

完成传递的过程为异步，取决于与发布关联的服务质量。

至多一次

QoS=0
从 MqttTopic.publish 返回时，立即完成传递。 会立即调用 MqttCallback.deliveryComplete。

至少一次

QoS=1
从队列管理器接收到有关发布的确认信息时就完成了传递。 接收到确认信息时就会调用 MqttCallback.deliveryComplete。 如果通信速度很慢或者不可靠，那么在调用 MqttCallback.deliveryComplete 之前可能会多次传递消息。

刚好一次

QoS=2
当客户机接收到有关已完成将发布消息发布至订户的消息时就完成了传递。 一旦接收到发布消息，就会调用 MqttCallback.deliveryComplete。 它并不会等待完成消息。

在极少数情况下，客户机应用程序可能不会正常从 MqttCallback.deliveryComplete 返回到 MQTT 客户机。 但是，您知道已完成传递，因为已经调用了 MqttCallback.deliveryComplete。 如果客户机重新启动同一会话，那么不会再次调用 MqttCallback.deliveryComplete。

未完成的传递

如果在客户机会话断开连接之后未完成传递，那么您可以再次连接客户机并完成传递。 仅当通过 MqttConnectionOptions 属性设置为 false 的会话发布了消息时，才能完成传递此消息。

使用同一客户机标识和服务器地址来创建客户机，然后在将 cleanSession MqttConnectionOptions 属性设置为 false 的情况下再次连接。 如果您将 cleanSession 设置为 true，那么会抛弃暂挂的传递令牌。

可以通过调用 MqttClient.getPendingDeliveryTokens 来检查是否有任何暂挂的传递。 在连接客户机之前，可以调用 MqttClient.getPendingDeliveryTokens。

相关概念

MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 MqttConnectOptions.cleanSession 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物

出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

“最后的消息”发布

如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

为最后的消息创建主题。 您可以创建主题，例如，MQTTManagement/Connections/*server URI*/*client identifer*/Lost。

使用 MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained) 方法来设置“最后的消息”。

考虑在 lastWillPayload 消息中创建一个时间戳记。 请包括用于帮助标识客户机以及连接环境的其他客户机信息。 将 MqttConnectionOptions 对象传递至 MqttClient 构造函数。

将 lastWillQos 设置为 1 或 2，以使消息持久保存在 IBM MQ 中并且保证传递。 要保留最后的断开连接信息，请将 lastWillRetained 设置为 true。

如果连接意外结束，那么会将“最后的消息”发布发送至订户。 如果连接结束，而客户机未调用 MqttClient.disconnect 方法，就会发送此发布。

要监控连接，请使用其他发布来补充“最后的消息”发布，以记录连接和程序化断开连接。

相关概念
MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 MqttConnectOptions.cleanSession 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌
MQTT 客户机中的消息持久性

如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

MQTT 客户机中的消息持久性

如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

在 MQTT 中，消息持久性具备两个方面：如何传输消息以及其是否作为持久消息在 IBM MessageSight 和 IBM MQ 中排队。

1. MQTT 客户机将消息持久性与服务质量进行了耦合。 根据您为消息选择的服务质量，使消息成为持久消息。 要实现必需的服务质量，必须具备消息持久性。

   如果指定了“最多一次”（QoS=0），那么客户机会在发布消息后立即丢弃该消息。 如果在消息的上游处理过程中发生了任何故障，将不会再次发送此消息。 即使客户机保持活动状态，也不会再次发送此消息。 QoS=0 消息的行为与 IBM MQ 快速非持久消息相同。
   如果客户机采用 QoS 1 或者 QoS 2 来发布消息，那么此消息为持久消息。 消息存储在本地，仅当不再需要保证“至少一次”（QoS=1）或“恰好一次”（QoS=2）交付时，才会从客户机中丢弃该消息。
2. 如果某消息标记为 QoS 1 或 2，那么它会作为持久消息在 IBM MessageSight 和 IBM MQ 中排队。 如果标记为 QoS=0，则作为非持久消息在 IBM MessageSight 和 IBM MQ 中排队。 在 IBM MQ 非持久消息在队列管理器之间传输 “正好一次”，除非消息通道将 NPMSPEED 属性设置为 FAST。

持久发布将存储在客户机上，直到客户机应用程序接收到此发布为止。 对于 QoS=2，当应用程序回调返回控制时，将从客户机上废弃发布。 对于 QoS=1，如果发生了故障，那么应用程序可能会再次接收到发布。 对于 QoS=0，回调接收到发布不会超过一次。 如果发生了故障，或者在发布时客户机断开连接，那么回调可能不会接收到发布。

当您预订主题时，可以降低订户用来接收消息的 QoS，使它与其持久性功能相匹配。 将使用订户请求的最高的 QoS 发送在更高的 QoS 的情况下创建的发布。

存储消息

在不同的小型设备上，数据存储器的实现的变化很大。 受 MQTT 客户机管理的存储器中临时保存的持久消息模型可能太慢，或需要太大的存储空间。 在移动设备中，移动操作系统可提供充分适用于 MQTT 消息的存储服务。

要在符合小型设备限制的同时提供灵活性，MQTT 客户机有两个持久性接口。 该接口可定义存储持久消息的过程中涉及的操作。 在 MQTT client for Java 的 API 文档中描述了这些接口。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。 可以根据设备需求来实现这些接口。 在 Java SE 上运行的 MQTT 客户机具有用于在文件系统中存储持久消息的接口的缺省实现。 它使用 `java.io` 包。

持久性类

`MqttClientPersistence`
将 `MqttClientPersistence` 实现的实例作为 `MqttClient` 构造函数的参数传递到 MQTT 客户机。 如果在 `MqttClient` 构造函数中省略了 `MqttClientPersistence` 参数，那么 MQTT 客户机将使用类 `MqttDefaultFilePersistence` 存储持久消息。

`MqttPersistable`
`MqttClientPersistence` 使用存储关键字来获取和放置 `MqttPersistable` 对象。 如果您未使用 `MqttDefaultFilePersistence`，那么必须提供 `MqttPersistable` 的实现以及 `MqttClientPersistence` 的实现。

`MqttDefaultFilePersistence`

MQTT 客户机提供了 `MqttDefaultFilePersistence` 类。 如果您将客户机应用程序中的 `MqttDefaultFilePersistence` 实例化，那么可以作为 `MqttDefaultFilePersistence` 构造函数的一个参数来提供用于存储持久消息的目录。

或者，MQTT 客户机可以实例化 `MqttDefaultFilePersistence` 并将文件放置在以下缺省目录中：

`client identifier` `-tcp` *`hostname portnumber`*

将从目录名称字符串中移除以下字符：

“\”，“\\”，“/”，“:”和“ ”

目录的路径是系统属性 `rcp.data` 的值；如果未设置 `rcp.data`，那么路径是系统属性 `usr.data` 的值，其中

- `rcp.data` 是与安装 OSGi 或 Eclipse 富客户机平台 (RCP) 相关联的属性。
- `usr.data` 是用于启动应用程序的 Java 命令的启动目录。

相关概念
MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 `MqttCallback` 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 `MqttConnectOptions.cleanSession` 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌
“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

出版物
出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量

MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

出版物

出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MqttMessage 将字节数组作为其有效内容。 其目的在于使消息尽可能小。 MQTT protocol 允许的最大消息长度为 250MB。

通常，MQTT 客户机程序使用 java.lang.String 或 java.lang.StringBuffer 处理消息内容。 为了方便起见，MqttMessage 类使用 toString 方法将其有效内容转换为字符串。 要从 java.lang.String 或 java.lang.StringBuffer 来创建字节数组有效内容，可使用 getBytes 方法。

getBytes 方法将字符串转换为平台的缺省字符集。 缺省字符集通常为 UTF-8。 仅包含文本的 MQTT 发布通常使用 UTF-8 编码。 使用 getBytes("UTF8") 方法来覆盖缺省字符集。

在 IBM MQ 中，会接收 MQTT 发布作为 jms-bytes 消息。 该消息包含一个 MQRFH2 文件夹（包含 <mqtt>）和一个 <mqps> 文件夹。 <mqtt> 文件夹包含 clientId、msgId 和 qos，但此内容在未来可能发生更改。

MqttMessage 还有另外三个属性：服务质量、是否保留了此消息以及此消息是否重复。 仅当服务质量为“至少一次”或者“刚好一次”时才设置重复标志。 如果先前已发送消息且 MQTT 客户机未作足够快速的应答，将会把重复属性设置为 true 来重新发送消息。

正在发布

要在 MQTT 客户机应用程序中创建发布，请创建 MqttMessage。 设置其有效内容、服务质量以及是否保留，并调用 MqttTopic.publish(MqttMessage message) 方法；将返回 MqttDeliveryToken 且完成发布的过程为异步。

或者，MQTT 客户机可在创建发布时，根据 MqttTopic.publish(byte [] payload, int qos, boolean retained) 方法上的参数，为您创建临时消息对象。

如果发布具有“至少一次”或“刚好一次”服务质量（QoS=1 或 QoS=2），那么 MQTT 客户机将调用 MqttClientPersistence 接口。 在将传递令牌返回给应用程序之前，它将调用 MqttClientPersistence 以存储消息。

应用程序可以选择使用 MqttDeliveryToken.waitForCompletion 方法来一直阻塞到将消息传递至服务器为止。 或者，应用程序可以继续运行而不进行阻塞。 如果要检查是否已交付发布而未阻止，请向 MQTT 客户机注册实现 MqttCallback 的回调类的实例。 成功发布传递之后，MQTT 客户机会调用 MqttCallback.deliveryComplete 方法。 根据服务质量不同，对于 QoS=0，几乎可以立即进行传递；而对于 QoS=2，可能要花一些时间。

使用 MqttDeliveryToken.isComplete 方法来轮询是否完成了传递。 当 MqttDeliveryToken.isComplete 的值为 false 时，可以调用 MqttDeliveryToken.getMessage 以获取消息内容。 如果调用 MqttDeliveryToken.isComplete 获得的结果为 true，说明已废弃此消

息，调用 MqttDeliveryToken.getMessage 将抛出空指针异常。 MqttDeliveryToken.getMessage 与 MqttDeliveryToken.isComplete 之间没有内置同步。

如果客户机在接收所有暂挂的传递令牌之前断开连接，那么客户机的新实例在连接之前可以查询暂挂的传递令牌。 在客户机连接之前，没有完成新的传递，并且调用 MqttDeliveryToken.getMessage 很安全。 使用 MqttDeliveryToken.getMessage 方法来弄清楚尚未传递哪些发布。 如果您在 MqttConnectOptions.cleanSession 设置为缺省值 true 的情况下进行连接，那么会废弃暂挂的传递令牌。

预订

队列管理器或 IBM MessageSight 负责创建要发送给 MQTT 订户的发布内容。 队列管理器可检查 MQTT 客户机在预订中创建的主题过滤器是否与发布中的主题字符串匹配。 匹配可以是精确匹配，也可以包括通配符。 在由队列管理器将发布转发至订户之前，队列管理器将检查与此发布相关联的主题属性。 它按照使用包含了通配符的主题字符串来进行预订中所描述的搜索过程来确定管理主题对象是否授权用户进行预订。

当 MQTT 客户机接收到具有“至少一次”服务质量的发布时，它将调用 MqttCallback.messageArrived 方法来处理此发布。 如果发布的服务质量是“刚好一次”，QoS=2，那么 MQTT 客户机将调用 MqttClientPersistence 接口以存储接收到的消息。 然后，它将调用 MqttCallback.messageArrived。

相关概念

MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 MqttConnectOptions.cleanSession 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

MQTT 客户机提供的服务质量

MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

发布的服务质量是 MqttMessage 的一种属性。 它是由 MqttMessage.setQos 方法设置的。

MqttClient.subscribe 方法可以降低应用于向客户机发送的、针对某个主题的发布的服务质量。 转发至订户的发布的服务质量可能不同于该发布的服务质量。 转发发布时，将使用这两个值当中的较小者。

至多一次
QoS=0

至多传递一次消息，或者根本就不传递消息。 不会确认通过网络来传递此消息。
不存储此消息。 如果客户机断开连接或者服务器失败，那么消息可能会丢失。
QoS=0 是最快传递方式。 有时候将它称为“发出消息之后无需等待应答”。
MQTT protocol 不要求服务器将 QoS=0 时的发布转发至客户机。 如果在服务器接收发布时客户机已断开连接，那么根据服务器，可能会废弃此发布。 遥测 (MQXR) 服务不会废弃使用 QoS=0 发送的消息。 它们存储为非持久消息，且只有当队列管理器停止运行时才被废弃。

至少一次
QoS=1

QoS=1 是缺省传递方式。
始终会至少传递一次消息。 如果发送方未接收到确认信息，那么会在设置了 DUP 标志的情况下再次发送此消息，直到接收到确认信息为止。 因此，接收方可以多次发送同一消息，并且可以多次处理此消息。
必须将消息存储在发送方和接收方本地，直到已处理此消息为止。
在接收方已处理消息之后，就会从接收方删除此消息。 如果接收方是一个代理，那么会将消息发布至它的订户。 如果接收方是客户机，那么会将消息传递至订户应用程序。 删除消息之后，接收方会向发送方发送确认信息。
在发送方接收到来自接收方的确认信息之后，就会从发送方删除此消息。

刚好一次
QoS=2

始终刚好传递一次消息。
必须将消息存储在发送方和接收方本地，直到已处理此消息为止。
QoS=2 是最安全、但是最慢的传递方式。 从发送方删除消息之前，此方式在发送方与接收方之间至少采用两对传输。 在第一次传输之后，可以在接收方处理消息。
在第一对传输中，发送方将传输消息，并从它已将消息存储于的接收方获取确认信息。 如果发送方未接收到确认信息，那么会在设置了 DUP 标志的情况下再次发送此消息，直到接收到确认信息为止。
在第二对传输中，发送方告诉接收方它可以完成对消息 “PUBREL” 的处理。 如果发送方未收到对 “PUBREL” 消息的确认，那么将再次发送 “PUBREL” 消息，直到接收到确认为止。 发送方在收到对 “PUBREL” 消息的确认时删除它保存的消息
如果接收方不重新处理消息，那么接收方可以在第一阶段或者第二阶段处理此消息。 如果接收方是一个代理，那么它会将消息发布至订户。 如果接收方是客户机，那么它会将消息传递至订户应用程序。 接收方会将一条完成消息发送回发送方，指出它已经完成了处理此消息。

相关概念
MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

清除会话

MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 `MqttConnectOptions.cleanSession` 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

保留的发布和 MQTT 客户机

主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

使用 `MqttMessage.setRetained` 方法来指定是否保留了有关某个主题的发布。

当创建或更新保留发布时，请发送 QoS 为 1 或 2 的发布。 如果发送 QoS 为 0 的发布，那么 IBM MQ 将创建非持久性保留发布。 如果队列管理器停止，就不会保留此发布。

如果您针对具有保留发布的主题发布非保留发布，那么保留的发布不会受到影响。 当前订户将接收到新的发布。 新订户将首先接收到保留的发布，然后接收所有新的发布。

您可以使用保留的发布来记录测量结果的最新值。 主题的新订户将立即接收到测量结果的最新值。 如果自订户上一次预订发布主题后未进行任何新的测量，且如果订户重新预订，那么该订户将再次接收到有关该主题的最新的保留发布。

要删除保留的发布，您有两个选项：

- 运行 **CLEAR TOPICSTR** MQSC 命令。
- 创建长度为零的保留发布。 如 MQTT 3.1.1 规范中所指定，如果将长度为零的保留消息发布至主题，那么将清除针对该主题的任何保留的消息。

相关概念
MQTT 客户机应用程序中的回调和同步

MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 MqttConnectOptions.cleanSession 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

MQTT 客户机中的主题字符串和主题过滤器
主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

预订

使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

使用 MqttClient.subscribe 方法并传递一个或多个主题过滤器和服务质量参数来创建预订。 服务质量参数设置订户准备用于接收消息的最高服务质量。 无法使用更高的服务质量来传递发送至此客户机的消息。 服务质量被设置为发布消息时的原始值与为预订指定的级别这两者之间的较小者。 接收消息的缺省服务质量为 QoS=1（至少一次）。

预订请求本身是使用 QoS=1 发送的。

当 MQTT 客户机调用 MqttCallback.messageArrived 方法时，订户会接收发布。 messageArrived 方法还会传递主题字符串，消息与此主题字符串一起被发布至订户。

可以使用 MqttClient.unsubscribe 方法来除去某个预订或者一组预订。

IBM MQ 命令可以移除预订。使用 IBM MQ Explorer 或使用 **runmqsc** 或 PCF 命令列出预订。 已命名全部 MQTT 客户机预订。 将为其提供以下格式的名称: *ClientIdentifier*:*Topic name*

如果在连接客户机之前使用缺省 MqttConnectOptions 或将 MqttConnectOptions.cleanSession 设置为 true，那么在客户机连接时将除去客户机的任何旧预订。 当客户机断开连接时，将移除客户机在会话期间创建的所有新预订。

如果在连接之前将 MqttConnectOptions.cleanSession 设置为 false，那么客户机创建的任何预订都将添加到客户机在连接之前已存在的所有预订。 当客户机断开连接时，所有预订仍保持活动状态。

要了解 cleanSession 属性影响预订的方式，另一种方法就是将它视作模态属性。 在其缺省方式 cleanSession=true 下，客户机仅在会话的作用域内创建预订和接收发布。 在另一种方式 cleanSession=false 下，预订是持久预订。 客户机可以连接和断开连接，而其预订保持活动状态。 当客户机重新连接时，它将接收任何未传递的发布。 在它连接之后，它可以自己修改处于活动状态的预订集。

必须先设置 cleanSession 方式，然后再进行连接; 该方式将持续整个会话。 要更改此属性的设置，必须将客户机断开连接，然后再重新连接客户机。 如果将方式从使用 cleanSession=false 更改为 cleanSession=true，那么将废弃客户机的所有先前预订以及尚未接收的任何发布。

一旦发布与活动预订相匹配的发布，就会将它们发送至客户机。 如果客户机已断开连接，那么如果它使用同一客户机标识来重新连接至同一服务器，并且将 MqttConnectOptions.cleanSession 设置为 false，就会将它们发送至客户机。

由客户机标识来标识特定客户机的预订。 可以将客户机从不同的客户机设备重新连接至同一服务器，并继续处理相同的预订和接收未传递的发布。

相关概念

MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 MqttCallback 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 MqttConnectOptions.cleanSession 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

MQTT 客户机中的主题字符串和主题过滤器

主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

MQTT 客户机中的主题字符串和主题过滤器

主题字符串和主题过滤器用于发布和预订。 MQTT 客户机中的主题字符串和过滤器的语法大部分与 IBM MQ 中的主题字符串相同。

主题字符串用来将发布发送至订户。 使用 `MqttClient.getTopic(java.lang.String topicString)` 方法来创建主题字符串。

主题过滤器用来预订主题和接收发布。 主题过滤器中可以包含通配符。 借助通配符，可以预订多个主题。 通过使用预订方法（例如，`MqttClient.subscribe(java.lang.String topicFilter)`）来创建主题过滤器。

主题字符串

在主题字符串中描述了 IBM MQ 主题字符串的语法。 MQTT 主题字符串的语法在 MQTT client for Java 的 API 文档中的 `MqttClient` 类中进行了描述。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。

每种类型的主题字符串的语法几乎完全相同。 它们之间有四项次要的差别：

1. MQTT 客户机发送到 IBM MQ 的主题字符串必须遵循队列管理器名称的约定。
2. 最大长度不同。 IBM MQ 主题字符串不能超过 10,240 个字符。 MQTT 客户机最多可创建 65535 字节的主题字符串。
3. MQTT 客户机创建的主题字符串不能包含空字符。
4. 在 IBM Integration Bus 中，空的主题级 `'...//...'` 无效。 然而，IBM MQ 支持空的主题级。

与 IBM MQ 发布/预订不同，`mqttv3` 协议没有管理主题对象的概念。 不能根据主题对象和主题字符串来构造主题字符串。 但是，在 IBM MQ 中，主题字符串将映射至管理主题。 与管理主题相关联的访问控制会确定是废弃发布，还是将它发布至主题。 将发布转发至订户时，应用于此发布的属性会受到管理主题的属性的影响。

主题过滤器

在基于主题的通配符方案中描述了 IBM MQ 主题过滤器的语法。 可以使用 MQTT 客户机构造的主题过滤器的语法在 MQTT client for Java 的 API 文档中的 `MqttClient` 类中进行了描述。 有关 MQTT 客户机库的客户机 API 文档的链接，请参阅 MQTT 客户机编程参考。

相关概念

MQTT 客户机应用程序中的回调和同步
MQTT 客户机编程模型广泛地使用了线程。 线程将尽可能降低 MQTT 客户机应用程序与服务器之间传输消息时的延迟。 发布，传递令牌和连接丢失事件将传递到实现 `MqttCallback` 的回调类中的方法。

清除会话
MQTT 客户机和遥测 (MQXR) 服务都会维护会话状态信息。 状态信息用于确保“至少一次”和“刚好一次”交付以及“刚好一次”接收发布。 会话状态还包括由 MQTT 客户机创建的预订。 运行 MQTT 客户机时，您可以选择维护会话之间的状态信息，也可以选择不维护。 通过在连接之前设置 `MqttConnectOptions.cleanSession` 来更改清除会话方式。

客户机标识
客户机标识是用于标识 MQTT 客户机的 23 字节的字符串。 每个标识都必须唯一，以便一次只有一台已连接的客户机。 客户机标识必须只包含在队列管理器名称中有效的字符。 在满足这些约束的条件下，您可以使用任何标识字符串。 必须具备分配客户机标识的过程以及使用所选标识配置客户机的方法，这一点非常重要。

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，那么可以配置 MQ Telemetry 以发送“最后的消息”发布。 预定义发布的内容以及要将它发送至的主题。 “最后的消息”是一种连接属性。 请在连接客户机之前创建此消息。

MQTT 客户机中的消息持久性
如果以“至少一次”或“恰好一次”的服务质量发送发布消息，那么这些消息将成为持久消息。 您可以在客户机上实施自己的持久性机制，也可以使用随客户机一起提供的缺省持久性机制。 将发布发送至客户机或者从客户机发送发布时都存在持久性。

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布以发送至 IBM MQ，并预订 IBM MQ 上的主题以接收发布。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量，可用于向 IBM MQ 和 MQTT 客户机提供发布内容：“至多一次”、“至少一次”以及“刚好一次”。 当 MQTT 客户机将请求发送至 IBM MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

保留的发布和 MQTT 客户机
主题可以有一个且仅有一个保留发布。 如果您针对具有保留发布的主题创建预订，那么会立即将该发布转发给您。

预订
使用主题过滤器创建一些预订以注册对发布主题的兴趣。 客户机可以创建多个预订，或者创建一个包含使用通配符的主题过滤器的预订，以注册对于多个主题的兴趣。 关于主题的、与过滤器相匹配的发布被发送至客户机。 当客户机断开连接时，预订可以保持活动状态。 当客户机重新连接时，就会将这些发布发送至客户机。

使用 IBM MQ 开发 Microsoft Windows Communication Foundation 应用程序

IBM MQ 的 Microsoft Windows Communication Foundation (WCF) 定制通道在 WCF 客户机和服务之间发送和接收消息。

相关概念

第 1141 页的『将适用于 WCF 的 IBM MQ 定制通道与 .NET 相结合简介』
IBM MQ 的定制通道是使用 Microsoft Windows Communication Foundation (WCF) 统一编程模型的传输通道。

第 1145 页的『使用适用于 WCF 的 IBM MQ 定制通道』
使用 Windows Communication Foundation (WCF) 的 IBM MQ 定制通道为程序员提供的信息概述。

第 1160 页的『使用 WCF 样本』
Windows Communication Foundation (WCF) 样本提供了一些关于如何使用 IBM MQ 定制通道的简单示例。

FFST: WCF XMS 首次故障支持技术

相关任务

跟踪 IBM MQ 的 WCF 定制通道

对 IBM MQ 问题的 WCF 定制通道进行故障诊断

将适用于 WCF 的 IBM MQ 定制通道与 .NET 相结合简介

IBM MQ 的定制通道是使用 Microsoft Windows Communication Foundation (WCF) 统一编程模型的传输通道。

Microsoft.NET 3 中引入的 Microsoft Windows Communication Foundation 框架使 .NET 应用程序和服务能够独立于用于连接它们的传输和协议进行开发，从而允许根据部署服务或应用程序的环境使用备用传输或配置。

连接是由 WCF 在运行时管理，方法是构建一个一个堆栈通道，其中包含以下项的必需组合：

- 协议元素：一组可选的元素，其中可以不添加任何元素，也可以添加一个或多个元素以支持不同协议，如 WS- * 标准。
- 消息编码器：堆栈中的一个必需元素，用于控制将消息序列化为其有线格式。
- 传输通道：堆栈中的一个强制性元素，负责将序列化消息传输到其端点。

IBM MQ 的定制通道是传输通道，因此必须与使用 WCF 定制绑定的应用程序所需的消息编码器和可选协议配对。 以这种方式，已开发为使用 WCF 的应用程序可以使用 IBM MQ 定制通道以与使用 Microsoft 提供的内置传输相同的方式发送和接收数据，从而实现与 IBM MQ 的异步、可扩展以及可靠消息传递功能的简单集成。 有关受支持功能的完整列表，请参阅：第 1145 页的『WCF 定制通道特性和功能』。

何时以及为何使用针对 WCF 的 IBM MQ 定制通道?

您可以使用 IBM MQ 定制通道以与 Microsoft 所提供的内置传输相同的方式在 WCF 客户机与服务之间发送和接收消息，使应用程序能够在 WCF 统一编程模型中访问 IBM MQ 的功能。

用于 WCF 的 IBM MQ 定制通道的典型使用模式方案是用于传输本机 IBM MQ 消息的非 SOAP 接口。

使用非 SOAP/非 JMS 消息（纯 MQMessage）格式传送的消息

当您将 WCF 的 IBM MQ 定制通道用作用于传输本机 IBM MQ 消息的非 SOAP 接口时，将使用 IBM MQ 的非 SOAP/Non-JMS 消息 (纯 MQMessage) 格式来传递消息。

WCF 用户能够启动服务，或者换句话说，服务用户能够使用 MQMessage 将消息发送至 IBM MQ 队列。 应用程序可以获取和设置 MQMD 字段和有效内容。 当消息在 IBM MQ MQ 队列中可用时，可以由任何 WCF 服务或非 WCF 应用程序 (例如，在 Windows， UNIX 或 z/OS 上运行的 C 或 Java 应用程序) 处理此消息。

适用于 WCF 的 IBM MQ 定制通道的软件需求

本主题概括了适用于 WCF 的 IBM MQ 定制通道的软件需求。 适用于 WCF 的 IBM MQ 定制通道只能连接到 IBM WebSphere MQ 7.0 或更高版本的队列管理器。

运行时环境需求

- 必须在主机上安装 Microsoft.NET Framework V4.5.1 或更高版本。
- 缺省情况下，*Java and .NET Messaging and Web Services* 作为 IBM MQ 安装程序的一部分进行安装。 该组件会将定制通道所需的 .NET 程序集安装到全局程序集缓存中。

注: 如果在安装 IBM MQ 之前未安装 Microsoft .NET Framework V4.5.1 或更高版本，那么 IBM MQ 产品安装将继续进行而不会发生错误，但 IBM MQ classes for .NET 不可用。 如果在安装 IBM MQ 之后安装了 .NET Framework，那么必须通过运行 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd 脚本来注册 IBM MQ.NET 组合件，其中 *WMQInstallDir* 是 IBM MQ 的安装目录。 该脚本将在全局程序集缓存 (GAC) 中安装必需的程序集。 将在 %TEMP% 目录中创建用于记录所执行操作的一组 amqi*.log 文件。 如果 .NET 已从较低版本（例如，从 .NET v3.5）升级到 v4.5.1 或更高版本，那么不必重新运行 amqiRegisterdotNet.cmd 脚本。

开发环境需求

- Microsoft Visual Studio 2008 或者 Windows Software Development Kit for .NET 4.5.1 或更高版本。
- 必须在主机上安装 Microsoft.NET Framework V4.5.1 或更高版本，才能构建样本解决方案文件。

适用于 WCF 的 IBM MQ 定制通道：安装哪些内容?

IBM MQ 的定制通道是使用 Microsoft Windows Communication Foundation (WCF) 统一编程模型的传输通道。 缺省情况下，定制通道作为安装的一部分安装。

面向 WCF 的 IBM MQ 定制通道

定制通道及其依赖关系包含在缺省情况下安装的 Java and .NET Messaging and Web Services 组件中。 从低于 IBM MQ 8.0 的版本中升级 IBM MQ 时，如果之前在较低版本的安装中安装了 Java and .NET Messaging and Web Services 组件，那么缺省情况下，更新将安装 WCF 的 IBM MQ 定制通道。

.NET Messaging and Web Services 组件包含 IBM.XMS.WCF.dll 文件和 IBM.WMQ.WCF.dll 文件，这些文件是主定制通道组合件，其中包含 WCF 接口类。 这些文件安装在全局组合件高速缓存 (GAC)

中，也位于以下目录中：*MQ_INSTALLATION_PATH* \bin（其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录）。

下表总结了使用定制通道所需的关键类。

表 *196:* 使用定制通道所需的关键类

	SOAP/JMS 接口（现有接口）	**非 SOAP/非 JMS 接口（来自 IBM MQ 8.0）**
定制通道组合件	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
传输绑定名称	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
传输绑定导入器	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
传输绑定配置	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Samples(Oneway)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client,MQMessaging_OneWay_Service
Samples(RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	MQMessaging_RequestReply_Client,MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll 支持 SOAP/JMS 和非 SOAP/非 JMS 接口。 新开发的应用程序建议使用 IBM.WMQ.WCF 组合件，因为其支持两个接口。

发送 MQSTR 格式化消息

V 9.1.0

如果请求消息类型为 MQSTR，那么您可以选择以 MQSTR 格式发送回复消息。

您必须使用额外的 URI 参数 **replyMessageFormat** 来更改回复消息的格式。 受支持的值包括：

""

" " 是缺省值。

回复消息采用字节 (MQMFT_NONE) 格式。 例如：

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR

回复消息采用 MQSTR (MQMFT_STRING) 格式。 例如：

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

注意：

1. **replyMessageFormat** 的值不区分大小写。
2. 使用除 " " 或 *MQSTR* 以外的任何值都会导致无效参数值异常。

IBM MQ 定制通道样本

这些样本提供一些简单示例，说明如何使用适用于 WCF 的 IBM MQ 定制通道。 这些样本及其关联文件位于 *MQ_INSTALLATION_PATH* \tools\dotnet\samples\cs\wcf 目录中，其中

MQ_INSTALLATION_PATH 是 IBM MQ 的安装目录。 有关 IBM MQ 定制通道样本的更多信息，请参阅第 1160 页的『使用 WCF 样本』。

svcutil.exe.config

`svcutil.exe.config` 是启用 Microsoft WCF `svcutil` 客户机代理生成工具以识别定制通道所需的配置设置示例。 `svcutil.exe.config` 文件位于 *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\` 目录中，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。 有关使用 `svcutil.exe.config` 的更多信息，请参阅第 1158 页的『通过将 svcutil 工具与运行服务中的元数据配合使用来生成 WCF 客户机代理和应用程序配置文件』。

WCF 体系结构

适用于 WCF 的 IBM MQ 定制通道集成在 `IBM Message Service Client for .NET (XMS .NET)` API 上。

SOAP/JMS 接口

WCF 体系结构如下图所示：

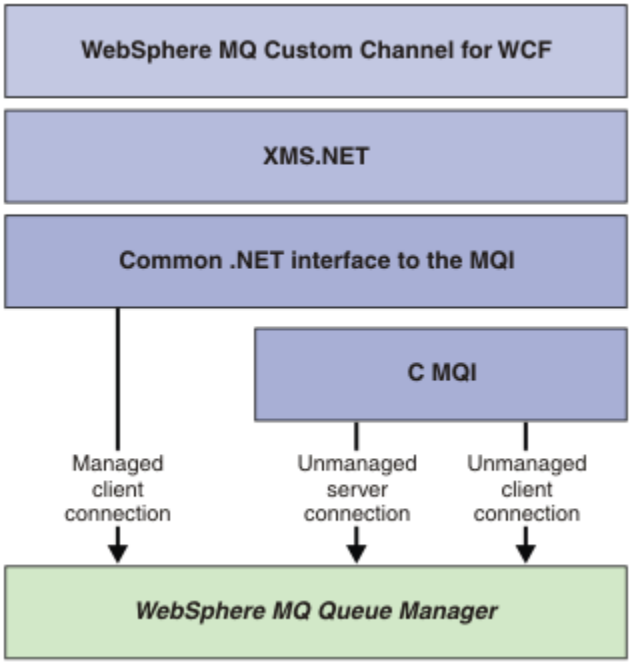


图 *154:* 面向 *SOAP/JMS* 接口的 *WCF* 体系结构

缺省情况下会在安装产品时安装所有必需组件。

三个连接为：

- 受管客户机连接
- 非受管服务器连接
- 非受管客户机连接

有关这些连接的更多信息，请参阅第 1150 页的『WCF 连接选项』。

非 SOAP/非 JMS 接口

适用于 WCF 的 IBM MQ 定制通道支持 SOAP/JMS 接口（可从 IBM WebSphere MQ 7.0.1 获得）和非 SOAP/非 JMS 接口。

WCF 体系结构如下图所示：

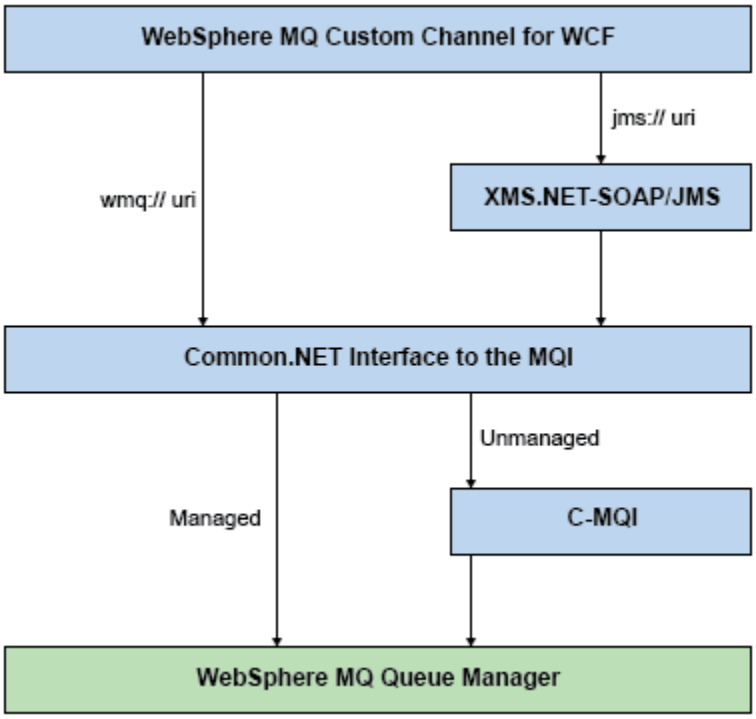


图 *155:* 面向非 *SOAP*/非 *JMS* 接口的 *WCF* 体系结构

使用适用于 WCF 的 IBM MQ 定制通道

使用 Windows Communication Foundation (WCF) 的 IBM MQ 定制通道为程序员提供的信息概述。

Microsoft Windows Communication Foundation 支持 Microsoft.NET Framework 3 中的 Web Service 和消息传递支持。 IBM WebSphere MQ 7.0 或更高版本可以通过与 Microsoft 提供的内置通道相同的方式在 .NET 框架 3 中用作 WCF 中的定制通道。

通过定制通道传输的消息将根据 IBM WebSphere MQ 7.0 或更高版本的 SOAP over JMS 实现进行格式化。然后，应用程序可以通过 JMS 服务基础结构与 WCF 或 WebSphere SOAP 托管的服务进行通信。

WCF 定制通道特性和功能

请通过以下主题获取有关 WCF 定制通道特性和功能的信息。

WCF 定制通道形状

IBM MQ 可以在 Microsoft Windows Communication Foundation (WCF) 定制通道中使用的定制通道形状的概述。

适用于 WCF 的 IBM MQ 定制通道支持两种通道形状：

- 单向
- 请求/应答

WCF 会根据正在托管的服务合同自动选项通道形状

包含仅使用 **IsOneWay** 参数的方法的合同由单向通道形状提供服务，例如：

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

包含单向和请求/应答方法混合型，或所有请求/应答方法的合同由请求/应答通道形状提供服务器。 例如：

```
[OperationContract]
int subtract(int a, int b);

[OperationContract(IsOneWay = true)]
void printString(string text);
```

注: 在混合的单向和请求/应答方法存在于同一合同中时，必须确保行为按您预期的进行（尤其是在混合环境中工作时），因为单向方法会一直等待，直到从服务接收到 null 应答。

单向通道

使用适用于 WCF 的 IBM MQ 单向定制通道，例如，使用单向通道形状从 WCF 客户机发送消息。 通道只能在一个方向发送消息，例如，从客户机队列管理器到 WCF 服务上的队列。

请求/应答通道

使用适用于 WCF 的 IBM MQ 请求/应答定制通道，例如，要在两个方向异步发送消息，必须使用相同的客户机实例以进行异步消息传递。 通道可以在一个方向上发送消息，例如，从客户机队列管理器到 WCF 服务器上的队列，然后将应答消息从 WCF 发送到客户机队列管理器上的队列。

WCF URI 参数名称和值

SOAP/JMS 接口和非 SOAP/非 JMS 接口的 URI 参数名称和值。

SOAP/JMS 接口

connectionFactory
需要 connectionFactory 参数。

initialContextFactory
initialContextFactory 参数是必需的，并且必须设置为“com.ibm.mq.jms.Nojndi”以便与 WebSphere Application Server 和其他产品兼容。

非 SOAP/非 JMS 接口

URI 格式遵循 MA93 规范。 请参阅 SupportPac - MA93 以获取 IBM MQ IRI 规范的更多详细信息。

IBM MQ URI 语法

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

IBM MQ IRI 示例
以下 IRI 示例告知服务请求者，可以使用端口 1414 上名为 example.com 的机器的 IBM MQ TCP 客户机绑定连接，并将持久性请求消息放入队列管理器 QM1 上的名为 SampleQ 的队列中。 IRI 指定了服务提供者将回复放入名为 SampleReplyQ 的队列中。

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

对于启用 TLS 的连接
要使用 WCF 客户机/服务进行安全 (TLS) 连接，请为 URI 中的以下属性设置合适的值。 前缀为“*”的所有属性都是强制要求进行安全连接。

- **sslKeyRepository**：*SYSTEM 或 *USER
- * **sslCipherSpec**：一个有效 CipherSpec，例如 TLS_RSA_WITH_AES_128_CBC_SHA256。
- **sslCertRevocationCheck**：true 或 false。
- **sslKeyResetCount**：大于 32kb 的值。
- **sslPeerName:**：服务器证书的专有名称

例如：

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

WCF 定制通道保证传递

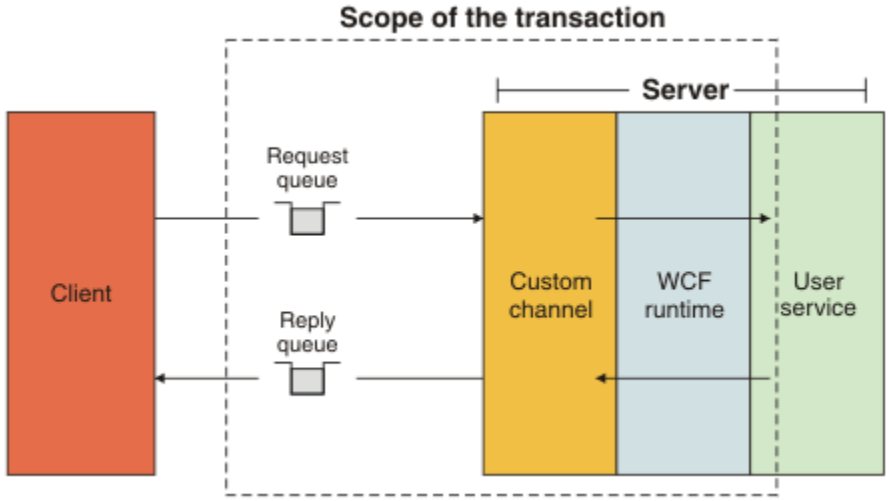
保证传递可保证服务请求或回复会被采取操作且不会丢失。

收到请求消息并且在本地事务同步点下发送任何回复消息，其可以出现运行故障的情况下回滚。 关于这些故障的示例有：服务抛出的未处理异常，无法向服务分派消息或无法传递回复消息。

AssuredDelivery 是保证传递属性，可以在服务合同上指定，以保证服务接收的任何请求消息以及从服务发送的任何回复消息在出现运行故障的情况下不会丢失。

为了确保在系统故障或断电的情况下也能保留消息，必须将消息作为持久性消息发送。 要使用持久性消息，客户机应用程序必须在其端点 URI 上指定该选项。

不支持分布式事务，且事务的作用域不能超出由 IBM MQ 执行的请求和回复消息处理的作用域。 服务中执行的任何工作可能会由于失败而重新运行，这导致再次收到该消息。 下图显示了事务的作用域：



通过对服务类应用 AssuredDelivery 属性来启用保证传递，如以下示例所示：

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

在使用 AssuredDelivery 属性时，您必须注意到以下几点：

- 在通道确定故障可能再次发生时，如果消息回滚并再次接收，那么该消息将被视为有害消息，并且不会返回到请求队列进行重新处理。 例如：如果接收的消息格式不正确或无法分派到服务。 总会重新发送从服务

操作抛出的未处理异常，直到重新给消息传递了由请求队列的 backout threshold 属性指定的最大次数。 有关更多信息，请参阅：第 1148 页的『WCF 定制通道有害消息』

- 通道通过执行单个线程来执行读取，处理以及作为原子操作的每个请求消息的应答，以强制事务完整性。 要使服务操作能够并发运行，通道使 WCF 能够创建多个通道实例。 绑定属性 `MaxConcurrentCalls` 控制可用于处理请求的通道实例数量。 有关更多信息，请参阅：第 1155 页的『WCF 绑定配置选项』
- 保证传递功能使用 IOperationInvoker 和 IErrorHandler WCF 扩展点。 如果应用程序在外部使用这些扩展点，那么应用程序必须确保调用任何先前注册的扩展点。 如果不执行此操作，IErrorHandler 可能会导致不报告错误。 如果不执行此操作，IOperationInvoker 可能导致 WCF 停止响应。

WCF 定制通道安全性

适用于 WCF IBM MQ 定制通道仅支持对非受管客户机连接到队列管理器使用 TSL。

使用客户机通道定义表 (CCDT) 中的项指定 TLS。 有关 CCDT 的更多信息，请参阅客户机通道定义表。

WCF 客户机通道定义表 *(CCDT)*

适用于 WCF 的 IBM MQ 定制通道支持使用客户机通道定义表 (CCDT) 来配置客户机连接的连接信息。

通过以下两个环境变量控制 CCDT：

- *MQCHLLIB* 指定该表所在的目录。
- *MQCHLTAB* 指定该表的文件名。

如果已经定义了这些环境变量，那么其将优先于 URI 中所指定的任何客户机连接详细信息。

有关客户机通道定义表的更多信息，请参阅客户机通道定义表。

相关概念

客户机通道定义表

WCF 定制通道有害消息

在服务无法处理请求消息时，或无法将应答消息传递到应答队列时，那么将该消息视为有害消息。

有害请求消息

如果无法处理请求消息，那么将视其为有害消息。 此操作阻止服务再次接收相同的不可处理的消息。 要将不可处理的请求消息视为有害消息，以下情况之一必须为真：

- 消息回退计数超过在请求队列上指定的回退阈值，仅当为服务指定了有保证的传递时，才会发生这种情况。 有关保证传递的更多信息，请参阅：第 1147 页的『WCF 定制通道保证传递』
- 消息格式不正确，且不能解释为 SOAP over JMS 消息。

有害应答回复

如果服务无法将回复消息递送到回复队列，那么将该回复消息视为有害消息。 对于回复消息，此操作可以稍后检索应答消息以帮助确定问题。

有害消息处理

为有害消息采取的操作取决于队列管理器配置和消息的报告选项中设置的值。 对于 SOAP over JMS，缺省情况下，会在请求消息上设置以下报告选项且不能配置这些选项：

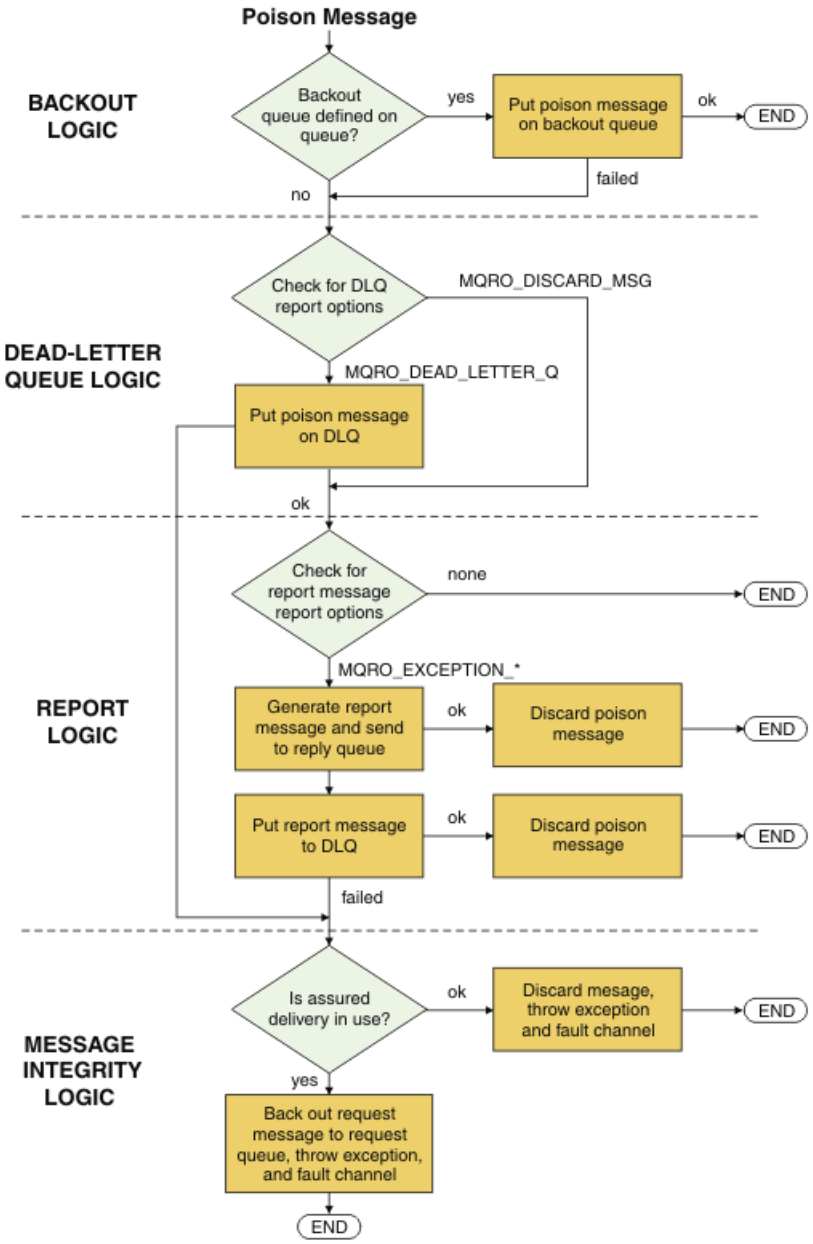
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

对于 SOAP over JMS，缺省情况下，会在回复消息上设置下面的报告选项且不能配置该选项：

- MQRO_DEAD_LETTER_Q

如果消息来自非 WCF 源，请参阅文档以获取该源。

下图显示了有害消息处理失败时可采取操作和步骤：



针对 *WCF* 应用程序的 *IBM MQ* 消息功能

针对 WCF 应用程序的非 SOAP/非 JMS（即，IBM MQ）消息功能。

对于非 SOAP/非 JMS 接口，针对 WCF 应用程序的 IBM MQ 消息功能如下所示：

- WCF 应用程序可以发送和接收可由任何 IBM MQ 应用程序处理的基本 IBM MQ 消息。
- WCF 应用程序可以完全控制更新 MQMD 和有效内容。
- WCF 客户机可以发送可由任何 IBM MQ 客户机 (例如 C， Java， JMS 和 .NET 客户机) 使用的 IBM MQ 消息。

针对非 SOAP/非 JMS 接口的 WCF 必须使用以下类为消息设置消息有效内容和 MQMD：

- WmqStringMessage，针对类型为 String 的有效内容
- WmqBytesMessage，针对类型为 Bytes 的有效内容
- WmqXmlMessage，针对类型为 XML 的有效内容

要设置消息的有效内容，请根据有效内容类型对 WmqStringMessage、WmqBytesMessage 或 WmqXmlMessage 类使用 **Data** 属性。 例如，请使用以下代码来设置类型为 String 的有效内容：

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

WCF 连接选项

有三种方式将 WCF 的 IBM MQ 定制通道连接到队列管理器。 请考虑哪种类型的连接最适合您的需求。

有关连接选项的更多信息，请参阅第 481 页的『连接差异』

有关 WCF 结构体系的更多信息，请参阅第 1144 页的『WCF 体系结构』

非受管客户机连接

在此模式中进行的连接将作为 IBM MQ 客户机连接到在本地计算机或远程计算机上运行的 IBM MQ 服务器。

要将适用于 WCF 的 IBM MQ 定制通道作为 IBM MQ 客户机使用，可以将该通道与 IBM MQ MQI client 一起安装在 IBM MQ 服务器上或独立的计算机上。

非受管服务器连接

在服务器绑定方式下使用时，适用于 WCF 的 IBM MQ 定制通道会使用队列管理器 API，而不是通过网络进行通信。 与使用网络连接相比，使用绑定连接能够为 IBM MQ 应用程序提供更好的性能。

要使用绑定连接，您必须在 IBM MQ 服务器上安装适用于 WCF 的 IBM MQ 定制通道。

受管客户机连接

在此模式中进行的连接将作为 IBM MQ 客户机连接到在本地计算机或远程计算机上运行的 IBM MQ 服务器。

在此方式下连接的 .NET 3 的 IBM MQ 定制通道类将保留在 .NET 受管代码中，并且不会调用本机服务。 有关受管代码的更多信息，请参阅 Microsoft 文档。

使用受管客户机时有若干限制。 有关这些限制的更多信息，请参阅 第 481 页的『受管客户机连接』。

创建和配置 WCF 的 IBM MQ 定制通道

WCF 的 IBM MQ 定制通道以与 Microsoft 提供的传输 WCF 通道相同的方式工作。 可通过以下方式之一创建 WCF 的 IBM MQ 定制通道。

关于此任务

IBM MQ 定制通道与 WCF 集成为 WCF 传输通道，因此必须与消息编码器和可选协议通道配对，这才能创建可由应用程序使用的完整通道堆栈。 要成功创建完整的通道堆栈，需要两个元素：

1. 绑定定义：指定在构建应用程序通堆栈时需要哪些元素，包括传输通道、消息编码器、任何协议以及常规配置设置。 对于定制通道，必须以 WCF 定制绑定的形式创建绑定定义。
2. 端点定义：将服务器合同与绑定定义链接，并提供描述应用程序可以连接的实际连接 URI。 对于定制通道，URI 采用 SOAP over JMS URI 的形式。

可通过两种不同方式之一创建这些定义：

- 管理方式：通过在应用程序配置文件中提供详细信息来创建定义（例如，`app.config`）。
- 编程方式：从应用程序代码直接创建定义。

决定使用哪种方法创建定义必须基于应用程序的要求，如下所示：

- 配置的管理方法可以灵活地更改服务和客户机部署后的详细信息，而无需重新构建应用程序
- 配置的编程方法提供了对配置错误的更大保护，以及在运行时动态生成配置的能力。

通过在应用程序配置文件中提供绑定和端点信息，以管理方式创建 *WCF* 定制通道

适用于 WCF 的 IBM MQ 定制通道是传输层 WCF 通道。 必须定义端点和绑定才能使用定制通道，可通过在应用程序配置文件中提供绑定和端点信息来完成这些定义。

要配置和使用适用于 WCF 的 IBM MQ 定制通道，该通道是传输层 WCF 通道，所以必须定义绑定和端点定义。 绑定保存通道的配置信息，端点定义保存连接详细信息。 可通过两种方式创建这些定义：

- 编程方式直接从应用代码定义，如第 1152 页的『通过以编程方式提供绑定和端点信息来创建 WCF 定制通道』中所述
- 管理方式，通过在应用程序配置文件中提供详细信息，如以下过程中所述。

客户机或服务应用程序配置文件通常命名为 *yourappname*.exe.config，其中 *yourappname* 是应用程序的名称。 修改应用程序配置文件的最简单方法是通过以下方式使用名为 SvcConfigEditor.exe 的 Microsoft 服务配置编辑器工具：

- 启动 SvcConfigEditor.exe 配置编辑器工具。 该工具的缺省安装位置为 *Drive:*\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe（其中 *Drive:* 是安装驱动器的名称）。

步骤 1：添加绑定元素扩展以使 WCF 能够找到定制通道

1. 右键单击**高级** > **扩展** > **绑定元素**以打开菜单，然后选择**新建**
2. 填写如下表所示的字段：

表 *197:* 新建绑定元素字段

字段	值
Name	IBM.XMS.WCF.SoapJmsIbmTransportChannel
Type	浏览至全局组合件高速缓存 (GAC) 的 IBM.XMS.WCF.dll 中，并选择 IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig

步骤 2：创建定制绑定定义，用于将定制通道与 WCF 消息编码器配对

1. 右键单击**绑定**以打开菜单，然后选择**新建绑定配置**
2. 填写如下表所示的字段：

表 *198:* 新建绑定配置字段

字段	值
Name	CustomBinding_WMQ
BindingElement 1	textMessageEncoding (MessageVersion: Soap11)
BindingElement 2	IBM.XMS.WCF.SoapJmsIbmTransportChannel

步骤 3：指定绑定属性

1. 从您在第 1151 页的『步骤 2：创建定制绑定定义，用于将定制通道与 WCF 消息编码器配对』创建的绑定中选择 *IBM.XMS.WCF.SoapJmsIbmTransportChannel* 传输绑定
2. 按第 1155 页的『WCF 绑定配置选项』中所述，对属性的缺省值进行任何所需的更改

步骤 4：创建端点定义

创建一个端点定义，该定义引用了您在第 1151 页的『步骤 2：创建定制绑定定义，用于将定制通道与 WCF 消息编码器配对』中创建的定制绑定并提供服务的连接详细信息。 指定此信息的方式取决于定义是针对客户机应用程序还是服务应用程序。

对于客户机应用程序，将端点定义添加到客户机部分，如下所示：

1. 右键单击**客户机** > **端点**以打开菜单，然后选择**新建客户机端点**
2. 填写如下表所示的字段：

表 *199:* 新建客户机端点字段	
字段	**值**
Name	`Endpoint_WMQ`
Address	*SOAP/JMS URI*，描述了访问服务所需的 *WMQ* 连接详细信息。 有关更多详细信息，请参阅第 1154 页的『适用于 WCF 端点 URI 地址格式的 IBM MQ 定制通道』
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract	服务合同接口的名称

对于服务应用程序，请将服务定义添加到服务部分，如下所示：

1. 右键单击**服务**以打开菜单，并选择**新建服务**，然后选择要托管的服务类。
2. 将端点定义添加到新建服务的“**端点**”部分，并完成如下表中所示的字段：

表 *200:* 新建服务端点字段	
字段	**值**
Name	`Endpoint_WMQ`
Address	*SOAP/JMS URI*，描述了访问服务所需的 *WMQ* 连接详细信息。 有关更多详细信息，请参阅第 1154 页的『适用于 WCF 端点 URI 地址格式的 IBM MQ 定制通道』
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract	服务实现类的名称

通过以编程方式提供绑定和端点信息来创建 *WCF* 定制通道

适用于 WCF 的 IBM MQ 定制通道是传输层 WCF 通道。 必须定义端点和绑定才能使用定制通道，并且这些定义可以直接从应用程序代码以编程方式完成。

要配置和使用适用于 WCF 的 IBM MQ 定制通道，该通道是传输层 WCF 通道，所以必须定义绑定和端点定义。 绑定保存通道的配置信息，端点定义保存连接详细信息。 有关更多信息，请参阅第 1160 页的『使用 WCF 样本』。

可通过两种方式创建这些定义：

- 管理方式，通过应用程序配置文件中提供详细信息，如第 1151 页的『通过在应用程序配置文件中提供绑定和端点信息，以管理方式创建 WCF 定制通道』中所述。
- 以编程方式直接从应用代码定义，如以下子主题中所述。

以编程方式定义绑定和端点信息：*SOAP/JMS* 接口
对于 SOAP/JMS 接口，您可以直接通过应用程序代码以编程方式定义端点和绑定。

关于此任务

要以编程方式提供绑定和端点信息，请通过完成以下步骤，将所需代码添加到您的应用程序。

过程

1. 通过将以下代码添加到您的应用程序来创建通道的传输绑定元素的实例：

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. 设置任何所需的绑定属性，例如，通过将以下代码添加到您的应用程序来设置 ClientConnectionMode：

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. 通过将以下代码添加到您的应用程序来创建定制绑定以将传输通道与消息编码器配对：

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. 创建 SOAP/JMS URI。

   描述访问服务所需的 IBM MQ 连接详细信息的 SOAP/JMS URI 必须作为端点地址提供。 您指定的地址取决于通道是用于服务应用程序还是用于客户机应用程序。

   - 对于客户机应用程序，必须将 SOAP/JMS URI 创建为 EndpointAddress，如下所示：

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?
     destination=SampleQ@QM1&connectionFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - 对于服务器应用程序，必须将 SOAP/JMS URI 创建为 URI，如下所示：

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     有关端点地址的更多信息，请参阅第 1154 页的『适用于 WCF 端点 URI 地址格式的 IBM MQ 定制通道』。

以编程方式定义绑定和端点信息：非 *SOAP*/非 *JMS* 接口

对于非 SOAP/非 JMS 接口，您可以直接通过应用程序代码以编程方式定义端点和绑定。

关于此任务

要以编程方式提供绑定和端点信息，请通过完成以下步骤，将所需代码添加到您的应用程序。

过程

1. 通过将以下代码添加到应用程序来创建 WmqBinding：

```
WmqBinding binding = new WmqBinding();
```

   该代码将为非 SOAP/非 JMS 接口创建所需的将 WmqMsgEncodingElement 和 WmqIbmTransportBindingElement 配对的绑定。

2. 提供描述访问服务时所需要的 IBM MQ 连接详细信息的 wmq:// URI。

   您提供 wmq:// URI 的方式取决于通道将用于服务应用程序还是客户机应用程序。

   - 对于客户机应用程序，必须将 wmq:// URI 创建为 EndpointAddress，如下所示：

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - 对于服务应用程序，必须将 wmq:// URI 创建为 URI，如下所示：

```
Uri sampleAddress = new Uri(
"wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

适用于 *WCF* 端点 *URI* 地址格式的 *IBM MQ* 定制通道

Web 服务是使用可以提供位置和连接详细信息的全局资源标识符 (URI) 来指定。 URI 格式取决于您使用的是 SOAP/JMS 接口，还是使用的是非 SOAP/非 JMS 接口。

SOAP/JMS 接口

IBM MQ Transport for SOAP 中所支持的 URI 格式允许在访问目标服务时对 SOAP IBM MQ 的特定参数和选项进行全面控制。 此格式与 WebSphere Application Server 和 CICS 兼容，有助于将 IBM MQ 与这两个产品集成在一起。

URI 语法如下：

```
jms:/queue? name=value&name=value...
```

其中 name 是参数名称，*value* 是相应的值；name = *value* 元素可以重复任意次数，第二次和后续出现的该元素前面有一个 & 符号。

参数名称区分大小写，IBM MQ 对也是如此。 如果任何参数多次指定，那么最后一次指定的参数生效，这表示着客户机应用程序可以通过附加到 URI 来覆盖参数值。 如果包括任何其他无法识别的参数，那么忽略这些参数

如果您在 XML 字符串中存储 URI，那么将必须用“& 字符表示为“&”来表示与字符。 同样，如果在脚本中编码 URI，请注意转义 **&** 等字符，否则需要通过 shell 来解释这些字符。

以下是一个针对 Axis 服务的简单 URI 示例：

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

以下是一个针对 .NET 服务的简单 URI 示例：

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

仅提供了必需参数 (仅 .NET 服务需要 `targetService`)，并且没有为 `connectionFactory` 提供任何选项。

在此 Axis 示例中，`connectionFactory` 包含多个选项：

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

在此 Axis 示例中，已指定 `connectionFactory` 的 `sslPeerName` 选项。 sslPeerName 的值本身包含名称值对和重要的嵌入空白：

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

非 SOAP/非 JMS 接口

非 SOAP/非 JMS 接口的 URI 格式允许在访问目标服务时对 IBM MQ 的特定参数和选项进行全面控制。

URI 语法如下：

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

该 IRI 告知服务器请求者，可以使用端口 1415 上名为 example.com 的机器的 IBM MQ TCP 客户机绑定连接，并可将持久性请求消息放到队列管理器 MOTOR.INS 上的名为 INS.QUOTE.REQUEST 的队列。 IRI 指定服务提供者将回复放入队列管理器 BRANCH452 上名为 INS.QUOTE.REPLY 的队列。 URI 格式与 SupportPac MA93 中指定的格式相同。 有关 IRI 规范的更多详细信息，请参阅 IBM MQSupportPac MA93：IBM MQ - 服务定义。

WCF 绑定配置选项

有两种方式将配置选项应用于定制通道绑定信息。 您可以以管理方式设置属性，也可以以编程方式设置属性。

可通过以下两种不同的方式之一设置绑定配置属性：

1. 管理方式：必须在应用程序配置文件中定制绑定定义的传输部分中指定绑定属性设置，例如：`app.config`。
2. 编程方式：必须修改应用程序代码以在定制绑定初始化期间指定属性。

以管理方式设置绑定属性

可以在应用程序配置文件中指定绑定属性设置，例如：`app.config`。 由 **svcutil** 生成配置文件，如以下示例所示。

SOAP/JMS 接口

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

非 SOAP/非 JMS 接口

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

以编程方式设置绑定属性

要添加 WCF 绑定属性以指定客户机连接方式，您必须修改服务代码以在定制绑定初始化期间指定属性。

使用以下示例以指定非受管的客户机连接方式：

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

WCF 绑定属性

表 *201:* 在通过管理方式或编程方式设置属性时绑定属性的值

属性名	客户机或服务应用程序	管理值	编程值	描述
maxBufferPoolSize	两者	0 到 64 位带符号整数	0 到 64 位带符号整数	指定可以用于存储通道实例的 WCF 消息缓冲的最大内存大小。
maxMessageSize	两者	1 到 32 位带符号整数	1 到 32 位带符号整数	指定可用于单个 WCF 消息的最大内存。
clientConnectionMode	两者	0（缺省值） 1	AS_URI（缺省值） CLIENT_UNMANAGED	指定传输通道的客户机连接方式。 0 表示客户机连接方式是在 URI 中指定。 只有使用客户机时才指定。 在 URI 中指定客户机连接方式。 如果未设置客户机连接模式，那么缺省值为 0。 1 表示客户机连接模式是非受管客户机。 只有使用客户机时才指定。
MaxConcurrentCalls	客户机	范围是 0 - 2 147 483 647 缺省值为 16	范围是 0 - 2 147 483 647 缺省值为 16	该属性定义了在任意时刻单个客户机代理上可能发生的最大并发操作数。 如果启动更多的操作，这些操作将会入队，直到正在处理的操作完成或超时。 该设置可用于控制单个代理可以使用的最大线程数和最大资源数。 0 会除去此限制，从而可以同时尝试启用所有操作。
MaxConcurrentCalls	服务	范围是 1 - 2 147 483 647 缺省值为 16	范围是 1 - 2 147 483 647 缺省值为 16	只要在启用保证传递时，才会使用该属性（有关保证传递的更多信息，请参阅第 1147 页的『WCF 定制通道保证传递』）。 该属性指定给定端点可同时处理的最大并发操作数。 更改此设置时需谨慎。 每个并发操作都需要其他资源，尤其是定制通道的新实例以及用于处理请求的线程池中的关联线程。 过度分配可能会产生反效果，并严重影响性能。 必须对线程池进行适当的配置以支持该属性。

构建并托管 WCF 的服务

Microsoft Windows Communication Foundation (WCF) 服务概述，介绍如何创建和配置 WCF 服务。

WCF 的 IBM MQ 定制通道以及使用该通道 WCF 服务可通过以下方法托管：

- 自托管
- Windows 服务

无法在 Windows Process Activation Service 中托管 WCF 的 IBM MQ 定制通道。

以下主题包含一些简单自托管示例以演示所涉及的步骤。 可在 Microsoft MSDN Web 站点 https://msdn.microsoft.com 上找到包含更多信息和最新详细信息的 Microsoft WCF 联机文档。

使用方法 *1* 构建 *WCF* 服务应用程序：使用应用程序配置文件以管理方式进行自托管

创建应用程序配置文件后，请打开服务的实例，然后将指定代码添加到您的应用程序。

开始之前

创建或编辑服务的应用程序配置文件，如第 1151 页的『通过在应用程序配置文件中提供绑定和端点信息，以管理方式创建 WCF 定制通道』中所述

关于此任务

1. 在服务主机中实例化并打开服务的实例。 服务类型必须与服务配置文件中指定的服务类型相同。
2. 将以下代码添加到应用程序：

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

使用方法 *2* 构建 *WCF* 服务应用程序：以编程方式直接从应用程序进行自托管

添加绑定属性，使用所需服务类的实例创建服务主机并打开服务。

开始之前

1. 将引用添加到项目的定制通道 `IBM.XMS.WCF.dll` 文件。 `IBM.XMS.WCF.dll` 位于 *WMQInstallDir*`\bin` 中，其中 *WMQInstallDir* 是安装 IBM MQ 的目录。
2. 将 *using* 语句添加到 `IBM.XMS.WCF` 名称空间，例如：`using IBM.XMS.WCF`
3. 创建通道绑定元素和端点的实例，如第 1152 页的『通过以编程方式提供绑定和端点信息来创建 WCF 定制通道』中所述

关于此任务

如果需要更改通道的绑定属性，请完成以下步骤：

1. 如以下示例所示，将绑定属性添加至 `transportBindingElement`：

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 使用所需服务类的实例创建服务主机：

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. 打开服务：

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

使用 *HTTP* 端点公开元数据

以下指示信息说明如何公开一项服务的元数据，该服务配置为对 WCF 使用 IBM MQ 定制通道。

关于此任务

如果必须公开服务元数据（使诸如 svcutil 的工具可直接从运行的服务器访问该元数据，而不需要从其他来源（例如脱机 WSDL 文件）访问），那么必须通过 HTTP 端点公开服务元数据来完成。 以下步骤可用于添加此附加端点。

1. 添加元数据必须公开给 ServiceHost 的基地址，例如：

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. 在打开服务之前，请将以下代码添加到 ServiceHost：

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

结果

现在可以从以下地址获取元数据：http://localhost:8000/MyService

为 WCF 构建客户机应用程序

生成和构建 Microsoft Windows Communication Foundation (WCF) 客户机应用程序的概述。

可为 WCF 服务创建客户机应用程序，通常可使用 Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) 生成客户机应用程序，以创建应用程序可直接使用的必需配置和代理文件。

通过将 *svcutil* 工具与运行服务中的元数据配合使用来生成 *WCF* 客户机代理和应用程序配置文件

以下指示信息介绍了如何使用 Microsoft svcutil.exe 工具来生成服务的客户机，这项服务配置为使用 WCF 的 IBM MQ 定制通道的服务。

开始之前

在使用 svcutil 工具创建可由应用程序直接使用的所需配置和代理文件时有三个先决条件：

- 在启动 svcutil 工具之前，必须先运行 WCF 服务。
- 除了 IBM MQ 定制通道端口引用之外，WCF 还必须使用 HTTP 端口公开其元数据以从运行服务直接生成客户机。
- 必须在 svcutil 的配置数据中注册定制通道。

关于此任务

以下步骤阐述了如何为配置为使用 IBM MQ 定制通道的服务生成客户机，以及通过独立的 HTTP 端口在运行时公开其元数据：

1. 启动 WCF 服务（在启动 svcutil 工具之前，必须先运行该服务）。
2. 将安装根目录中 svcutil.exe 配置文件中的详细信息添加到活动的 svcutil 配置文件（通常为 C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config）中，因此 svcutil 将识别 IBM MQ 定制通道。
3. 从命令提示符运行 svcutil，例如：

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. 将生成的 app.config 和 YourService.cs 文件复制到 Microsoft Visual studio 客户机项目。

下一步做什么

如果无法直接检索服务元数据，那么 svcutil 可生成 wsdl 的客户机文件。 有关更多信息，请参阅：第 1159 页的『使用 svcutil 工具和 WSDL 来生成 WCF 客户机代理和应用程序配置文件』

使用 *svcutil* 工具和 *WSDL* 来生成 *WCF* 客户机代理和应用程序配置文件

下面提供了通过 WSDL 生成 WCF 客户机（如果服务元数据不可用）的指示信息。

如果无法直接从正在运行的服务中检索服务元数据以通过元数据生成客户机，那么可以改为使用 svcutil 以通过 WSDL 生成客户机文件。 必须对 WSDL 进行以下修改以指定要使用的 IBM MQ 定制通道：

1. 添加以下名称空间定义和策略信息：

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. 修改绑定部分以引用新的策略部分，并从底层绑定元素中移除任何 `transport` 定义：

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. 从命令提示符运行 svcutil，例如：

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

使用具有应用程序配置文件的客户机代理构建 *WCF* 客户机应用程序

开始之前

为每个客户机创建或编辑应用程序配置文件，如 第 1151 页的『通过在应用程序配置文件中提供绑定和端点信息，以管理方式创建 WCF 定制通道』 中所述：

关于此任务

实例化并打开客户机代理的实例。 传递到生成代理的参数必须与客户机配置文件中指定的端点名称相同，例如 `Endpoint_WMQ`：

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
```

```
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

使用具有程序配置的客户机代理构建 *WCF* 客户机应用程序

开始之前

1. 将引用添加到项目的定制通道 IBM.XMS.WCF.dll 文件。 IBM.XMS.WCF.dll 位于 *WMQInstallDir*\bin 目录中，其中 *WMQInstallDir* 是安装 IBM MQ 的目录。
2. 将 *using* 语句添加到 IBM.XMS.WCF 名称空间，例如：`using IBM.XMS.WCF`
3. 创建绑定元素的实例和通道的端点，如第 1152 页的『通过以编程方式提供绑定和端点信息来创建 WCF 定制通道』中所述

关于此任务

如果需要更改通道的绑定属性，请完成以下步骤。

1. 如下图所示，将绑定属性添加至 `transportBindingElement`：

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 如下图所示，创建客户机代理，其中 *binding* 和 *endpoint address* 是步骤 1 中配置并传递的绑定和端点地址：

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

使用 WCF 样本

Windows Communication Foundation (WCF) 样本提供了一些关于如何使用 IBM MQ 定制通道的简单示例。

要构建样本项目，需要使用 Microsoft.NET 3.5 SDK 或 Microsoft Visual Studio 2008。

简单的单向客户机和服务器 WCF 样本

此样本演示了用于使用单向通道形状从 WCF 客户机启动 Windows 通信基础 (WCF) 服务的 IBM MQ 定制通道。

关于此任务

该服务实现了单个方法，用于将字符串输出到控制台。 已使用 `svcutil` 工具生成客户机以从单独公开的 HTTP 端点检索服务元数据，如 第 1158 页的『通过将 svcutil 工具与运行服务中的元数据配合使用来生成 WCF 客户机代理和应用程序配置文件』 中所述

已使用特定资源名称配置了样本，如以下过程中所述。 如果必须更改资源名称，那么还必须在 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` 文件中更改客户机应用程序的相应值以及在 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` 文件中更改服务应用程序的相应值，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 IBM MQ 产品文档中的 *IBM MQ Transport for SOAP* 。 如果需要修改样本解决方案和源，那么您需要 IDE，例如，Microsoft Visual Studio 8 或更高版本。

过程

1. 创建名为 *QM1* 的队列管理器
2. 创建名为 *SampleQ* 的队列目标
3. 启动服务以便侦听器等待消息：运行 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` 文件，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。
4. 运行一次客户机：运行 *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` 文件，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。

   客户机应用程序会循环 5 次，向 *SampleQ* 发送 5 条消息

结果

服务应用程序会从 *SampleQ* 获取消息，并在屏幕上显示 5 次 `Hello World`。

下一步做什么

简单的请求/应答客户机和服务器 WCF 样本

此样本演示用于使用请求/应答通道形状从 WCF 客户机启动 Windows 通信基础 (WCF) 服务的 IBM MQ 定制通道。

关于此任务

此服务提供了一些简单的计算器方法以添加和减去两个数字，然后返回结果。 已使用 `svcutil` 工具生成客户机以从单独公开的 HTTP 端点检索服务元数据，如 第 1158 页的『通过将 svcutil 工具与运行服务中的元数据配合使用来生成 WCF 客户机代理和应用程序配置文件』 中所述

已使用特定资源名称配置了样本，如以下过程中所述。 如果需要更改资源名称，那么还需要在 *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config` 文件中更改客户机应用程序的相应值以及在 *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` 文件中更改服务应用程序的相应值，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 IBM MQ 产品文档中的 *IBM MQ Transport for SOAP* 。 如果需要修改样本解决方案和源，那么您需要 IDE，例如，Microsoft Visual Studio 8 或更高版本。

过程

1. 创建名为 *QM1* 的队列管理器
2. 创建名为 *SampleQ* 的队列目标
3. 创建名为 *SampleReplyQ* 的队列目标

4. 启动服务以便侦听器等待消息：运行 *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe 文件，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。
5. 运行一次客户机：运行 *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe 文件，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。

结果

当客户机已运行时，将启动以下进程并重复四次，因此会向每个方向总共发送五条消息：

1. 客户机将请求消息放置在 *SampleQ* 上并等待响应。
2. 服务从 *SampleQ* 获取请求消息。
3. 服务使用消息的内容添加和减少一些值。
4. 然后，服务会将结果放入到 *SampleReplyQ* 上的消息中，并等待客户机放置新消息。
5. 客户机从 *SampleReplyQ* 获取消息，并将结果显示在屏幕上。

下一步做什么

WCF 客户机到 IBM MQ 样本托管的 .NET 服务

为 .NET 和 Java 提供样本客户机应用程序和样本服务代理应用程序。 样本基于股票报价服务，用于获取股票报价请求，然后提供股票报价。

开始之前

该样本要求在 IBM MQ 中正确安装和配置 .NET SOAP over JMS 服务托管环境，并且可从本地队列管理器进行访问。

当 .NET SOAP over JMS 服务托管环境在 IBM MQ 中正确安装和配置并且可从本地队列管理器访问时，必须完成其他配置步骤。

1. 将 WMQSOAP_HOME 环境变量设置为 IBM MQ 安装目录，例如：`C:\Program Files\IBM\MQ`
2. 确保 Java 编译器 `javac` 可用且位于 PATH 上。
3. 将文件 `axis.jar` 从 WebSphere 安装 CD 的 `prereqs/axis` 目录复制到 IBM MQ 生产目录，例如：`C:\Program Files\IBM\MQ\java\lib\soap`
4. 添加到 PATH：*MQ_INSTALLATION_PATH*\Java\lib，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录，例如：`C:\Program Files\IBM\MQ`
5. 确保在 *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd 中正确指定了 .NET 的位置，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装位置，例如：`C:\Program Files\IBM\MQ`。 可以指定 .NET 的位置，例如: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

完成以上步骤之后，测试并运行服务：

1. 浏览至 Soap over JMS 工作目录。
2. 输入以下命令之一运行验证测试，并使服务侦听器保持运行：
   - 对于 .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。
   - 对于 AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。

`hold` 自变量会使侦听器在完成测试之后继续运行。

如果在此配置期间报告了错误，那么可以移除所有更改，以便通过以下方式重新启动过程。

1. 删除生成的 SOAP over JMS 目录。
2. 删除队列管理器。

关于此任务

此样本演示了使用单向通道形状从 WCF 客户机到 IBM MQ 中提供的 .NET SOAP over JMS 样本服务的连接。该服务实现了简单的 StockQuote 示例，用于将文本字符串输出到控制台。

已通过使用 WSDL 生成客户机文件来生成了客户机，如第 1159 页的『使用 svcutil 工具和 WSDL 来生成 WCF 客户机代理和应用程序配置文件』中所述

已使用特定资源名称配置了样本，如以下过程中所述。 如果需要更改资源名称，那么还必须在 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\app.config 文件中更改客户机应用程序的相应值以及在 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl 文件中更改服务应用程序的相应值，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 IBM MQ 产品文档中的 *IBM MQ Transport for SOAP*。

过程

运行一次客户机：运行 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe 文件，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。

客户机应用程序会循环 5 次，向样本队列发送 5 条消息。

结果

服务应用程序会从样本队列获取消息，并在屏幕上显示 5 次 `Hello World`。

WCF 客户机到由 IBM MQ 样本托管的 Axis Java 服务

为 Java 和 .NET 提供样本客户机应用程序和样本服务代理应用程序。 样本基于股票报价服务，用于获取股票报价请求，然后提供股票报价。

开始之前

此样本要求在 IBM MQ 中正确安装和配置 .NET SOAP over JMS 服务托管环境，并且可从本地队列管理器进行访问。

当 .NET SOAP over JMS 服务托管环境在 IBM MQ 中正确安装和配置并且可从本地队列管理器访问时，必须完成其他配置步骤。

1. 将 WMQSOAP_HOME 环境变量设置为 IBM MQ 安装目录，例如：`C:\Program Files\IBM\MQ`
2. 确保 Java 编译器 `javac` 可用且位于 PATH 上。
3. 将文件 `axis.jar` 从 WebSphere 安装 CD 的 `prereqs/axis` 目录复制到 IBM MQ 安装目录。
4. 添加到 PATH：*MQ_INSTALLATION_PATH*\Java\lib，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录，例如：`C:\Program Files\IBM\MQ`
5. 确保在 *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd 中正确指定了 .NET 的位置，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装位置，例如：`C:\Program Files\IBM\MQ`。 可以指定 .NET 的位置，例如: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

完成以上步骤之后，测试并运行服务：

1. 浏览至 Soap over JMS 工作目录。
2. 输入以下命令之一运行验证测试，并使服务侦听器保持运行：
   - 对于 .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。
   - 对于 AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。

`hold` 自变量会使侦听器在完成测试之后继续运行。

如果在此配置期间报告了错误，那么可以移除所有更改，以便通过以下方式重新启动过程。

1. 删除生成的 SOAP over JMS 目录。
2. 删除队列管理器。

关于此任务

样本演示了从 WCF 客户机到使用单向通道形状的 IBM MQ 中提供的 Axis Java SOAP over JMS 样本服务的连接。 该服务实现一个简单的 StockQuote 示例，它将文本字符串输出到文件（已保存在当前目录中）。

已通过使用 WSDL 生成客户机文件来生成了客户机，如第 1159 页的『使用 svcutil 工具和 WSDL 来生成 WCF 客户机代理和应用程序配置文件』中所述

已使用特定的资源名称配置了样本，如该段中所述。 如果需要更改资源名称，那么还必须在 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\app.config 文件中更改客户机应用程序的相应值以及在 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl 文件中更改服务应用程序的相应值，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。

过程

运行一次客户机：运行 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe 文件，其中 *MQ_INSTALLATION_PATH* 表示 IBM MQ 的安装目录。

客户机应用程序会循环 5 次，向样本队列发送 5 条消息。

结果

服务应用程序会从样本队列获取消息，并向当前目录中的文件添加 5 次 Hello World。

WCF 客户机到 WebSphere Application Server 样本托管的 Java 服务

为 WebSphere Application Server 6 提供样本客户机应用程序和样本服务代理应用程序。 同时还提供请求/响应服务。

开始之前

该样本要求使用以下 IBM MQ 配置：

表 *202: IBM MQ* 所需的配置

Object	**必需名称**
队列管理器	QM1
本地队列	HelloWorld
本地队列	HelloWorldReply

该样本还要求正确安装和配置了 WebSphere Application Server 6 托管环境。 缺省情况下，WebSphere Application Server 6 使用绑定方式连接来连接到 IBM MQ。 因此，WebSphere Application Server 6 必须与队列管理器安装在同一台机器上。

在配置 WAS 环境之后，必须完成以下额外的配置步骤：

1. 在 WebSphere Application Server JNDI 存储库中创建以下 JNDI 对象：
   a. 名为 HelloWorld 的 JMS 队列目标
      - 将 JNDI 名称设置为 jms/HelloWorld
      - 将队列名称设置为 HelloWorld
   b. 名为 HelloWorldQCF 的 JMS 队列连接工厂
      - 将 JNDI 名称设置为 jms/HelloWorldQCF
      - 将队列管理器名称设置为 QM1

c. 名为 WebServicesReplyQCF 的 JMS 队列连接工厂

- 将 JNDI 名称设置为 jms/WebServicesReplyQCF
- 将队列管理器名称设置为 QM1

2. 使用以下配置在 WebSphere Application Server 中创建名为 HelloWorldPort 的消息侦听器端口:
   - 将连接工厂 JNDI 名称设置为 jms/HelloWorldQCF
   - 将目标 JNDI 名称设置为 jms/HelloWorld
3. 将 Web Service HelloWorldEJBEAR.ear 应用程序安装到 WebSphere Application Server，如下所示:
   a. 单击**应用程序 > 新建应用程序 > 新建企业应用程序**。
   b. 浏览至 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear（其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录）。
   c. 请勿更改向导中的任何缺省选项，并在安装应用程序之后重新启动应用程序服务器。

在完成 WAS 配置后，请运行一次来测试该服务:

1. 浏览至 Soap over JMS 工作目录。
2. 输入以下命令来运行样本：*MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\TestClient.exe，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。

关于此任务

该样本演示了从 WCF 客户机到 IBM MQ 中包含的 WCF 样本中提供的 WebSphere Application Server SOAP over JMS 样本服务的连接，使用请求/响应通道形状。 消息在 WCF 与使用 IBM MQ 队列的 WebSphere Application Server 之间流动。 此服务实现了 HelloWorld(...) 方法，该方法接受一个字符串并向客户机返回问候。

通过使用 svcutil 工具从单独公开的 HTTP 端点检索服务元数据生成了客户机，如第 1158 页的『通过将 svcutil 工具与运行服务中的元数据配合使用来生成 WCF 客户机代理和应用程序配置文件』中所述

已使用特定资源名称配置了样本，如以下过程中所述。 如果需要更改资源名称，那么还必须更改 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\default\client\app.config 文件中客户机应用程序上的相应值，以及 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear 中服务应用程序上的相应值（其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录）。

服务和客户机都基于 IBM Developer 文章“*Building a JMS Web Service using SOAP over JMS and WebSphere Studio*”中列出的服务和客户机。 有关开发与 IBM MQ WCF 定制通道兼容的 SOAP over JMS Web Service 的更多信息，请参阅 https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html。

过程

运行一次客户机：运行 *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe 文件，其中 *MQ_INSTALLATION_PATH* 是 IBM MQ 的安装目录。

客户机应用程序会同时启动两个服务方法，向样本队列发送两条消息。

结果

服务应用程序从样本队列获取消息，并提供对客户机应用程序输出到控制台的 HelloWorld(...) 方法调用的响应。

声明

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能。 有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。 任何对 IBM 产品、程序或服务的引用并非意在明示或默示只能使用 IBM 的产品、程序或服务。 只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务都可以代替 IBM 产品、程序或服务。 但是， 评估和验证任何非 IBM 产品、程序或服务的操作，由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。 提供本文档并未授予用户使用这些专利的任何许可。 您可以以书面形式将许可查询寄往：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往：

知识产权许可
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 063-8506 Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation “按现状” 提供本出版物，不附有任何种类的 (无论是明示的还是暗含的) 保证，包括但不限于暗含的有关非侵权，适销和适用于某种特定用途的保证。 某些国家或地区在某些交易中不允许免除明示或暗含的保证。 因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。 此处的信息将定期更改；这些更改将编入本资料的新版本中。 IBM 可以随时对本出版物中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式 充当对那些 Web 站点的保证。 那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：(i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系：

IBM Corporation
软件互操作性协调员，部门 49XA
北纬 3605 号公路
罗切斯特，明尼苏达州 55901
U.S.A.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。 因此，在其他操作环境中获得的数据可能会有明显的不同。 有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。 此外，有些测量是通过推算而估计的， 实际结果可能会有差异。 本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中 获取。 IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。 有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息包含日常商业运作所使用的数据和报表的示例。 为了尽可能全面地说明这些数据和报表，这些示例包括个人、公司、品牌和产品的名称。 所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址有任何雷同，纯属巧合。

版权许可：

本信息包含源语言形式的样本应用程序，用以阐明在不同操作平台上的编程技术。 如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。 这些示例并未在所有条件下作全面测试。 因此，IBM 不能担保或默示这些程序的可靠性、可维护性或功能。

如果您正在查看本信息的软拷贝，图片和彩色图例可能无法显示。

编程接口信息

编程接口信息 (如果提供) 旨在帮助您创建用于此程序的应用软件。

本书包含有关允许客户编写程序以获取 WebSphere MQ 服务的预期编程接口的信息。

但是，该信息还可能包含诊断、修改和调优信息。 提供诊断、修改和调优信息是为了帮助您调试您的应用程序软件。

要点: 请勿将此诊断，修改和调整信息用作编程接口，因为它可能会发生更改。

商标

IBM IBM 徽标 ibm.com 是 IBM Corporation 在全球许多管辖区域的商标。 当前的 IBM 商标列表可从 Web 上的 “Copyright and trademark information”www.ibm.com/legal/copytrade.shtml 获取。 其他产品和服务名称可能是 IBM 或其他公司的商标。

Microsoft 和 Windows 是 Microsoft Corporation 在美国和/或其他国家或地区的商标。

UNIX 是 Open Group 在美国和其他国家或地区的注册商标。

Linux 是 Linus Torvalds 在美国和/或其他国家或地区的商标。

此产品包含由 Eclipse 项目 (http://www.eclipse.org/) 开发的软件。

Java 和所有基于 Java 的商标和徽标是 Oracle 和/或其附属公司的商标或注册商标。

部件号:

(1P) P/N: