7.5

为 *IBM WebSphere 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 953 页的『声明』中的信息。

此版本适用于 IBM® WebSphere MQ V 7 发行版 5 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。

当您向 IBM 发送信息时，授予 IBM 以它认为适当的任何方式使用或分发信息的非独占权利，而无需对您承担任何责任。

内容

开发应用程序 7
- 应用程序开发概念 7
  - 使用 MQI 的应用程序 8
  - IBM WebSphere MQ 消息 9
  - 准备和运行 Microsoft Transaction Server 应用程序 34
  - 将 IBM WebSphere MQ 与 WebSphere Application Server 配合使用 34
  - 业务支持方案 35
- 决定要使用的语言 64
  - IBM WebSphere MQ 数据定义文件 66
  - 采用 C 进行编码 68
  - 采用 COBOL 进行编码 70
  - 采用 pTAL 进行编码 70
  - 采用 Visual Basic 进行编码 71
  - IBM WebSphere MQ 对象模型 72
  - 使用 JMS 或 Java 73
- 设计 IBM WebSphere MQ 应用程序 73
  - 设计消息 75
  - 应用程序设计和性能 76
  - 高级 IBM WebSphere MQ 技术 77
- 样本 IBM WebSphere MQ 程序 78
  - 分布式平台的样本程序 79
- 编写排队应用程序 162
  - 消息队列接口概述 163
  - 连接到队列管理器和从队列管理器断开连接 173
  - 打开和关闭对象 179
  - 将消息放置到队列上 188
  - 从队列取出消息 201
  - 编写发布/预订应用程序 232
  - 查询和设置对象属性 268
  - 落实和回退工作单元 271
  - 使用触发器启动 IBM WebSphere MQ 应用程序 275
  - 与 MQI 和集群配合使用 289
- 编写客户机应用程序 293
  - 针对客户机应用程序使用消息队列接口 (MQI) 294
  - 为 IBM WebSphere MQ MQI 客户机构建应用程序 297
  - 在 IBM WebSphere MQ MQI 客户机环境中运行应用程序 299
  - 准备和运行 CICS 和 Tuxedo 应用程序 309
  - 准备和运行 Microsoft Transaction Server 应用程序 310
  - 准备和运行 IBM WebSphere MQ JMS 应用程序 311
- 用户出口、API 出口和可安装服务 311
  - 编写和编译出口和可安装服务 311
- 构建 IBM WebSphere MQ 应用程序 357
  - 在 AIX 上构建应用程序 357
  - 在 HP Integrity NonStop Server 上构建应用程序 363
  - 在 HP-UX 上构建应用程序 368
  - 在 Linux 上构建应用程序 373
  - 在 Solaris 上构建应用程序 379
  - 在 Windows 系统上构建应用程序 385
  - 将轻量级目录访问协议服务与 IBM WebSphere MQ for Windows 配合使用 392
- 开发 IBM WebSphere MQ Telemetry 应用程序 397
  - IBM WebSphere MQ Telemetry 样本程序 397
  - 使用 Java 创建第一个发布程序 400

使用 Java 创建异步发布程序 405
使用 Java 创建可恢复的异步发布程序 409
使用 Java 创建订户 414
使用 JAAS 认证 MQTT 客户机 419
使用自签名证书来认证 SSL 连接 425
使用证书链认证 SSL 连接 429
使用 C 语言创建第一个发布程序 433
使用 C 语言创建异步发布程序 436
使用 C 语言创建订户 439
客户机编程概念 444
C 客户机编程概念 459
处理程序错误 462
本地确定的错误 462
使用报告消息进行问题确定 463
远程确定的错误 463
多点广播编程 465
多点广播和消息队列接口 465
多点广播连接到队列管理器 467
多点广播消息传递的编程数据转换 468
多点广播异常报告 468
使用 .NET 470
IBM WebSphere MQ classes for .NET 入门 471
编写和部署 IBM WebSphere MQ.NET 程序 483
用于 Microsoft Windows Communication Foundation (WCF) 的 IBM WebSphere MQ 定制通道 500
使用 IBM WebSphere MQ 定制通道 for WCF with .NET 3 简介 501
将 IBM WebSphere MQ 定制通道用于 WCF 504
使用 WCF 样本 518
IBM WebSphere MQ 的 WCF 定制通道上的问题确定 523
使用 C++ 529
样本程序 532
C++ 语言概念 536
C++ 中的消息传递 539
构建 IBM WebSphere MQ C++ 程序 545
使用 IBM WebSphere MQ classes for Java 552
IBM WebSphere MQ Java 类入门 552
安装和配置 IBM WebSphere MQ classes for Java 554
面向程序员的简介 564
为 Java 应用程序编写 IBM WebSphere MQ 类 564
使用 IBM WebSphere MQ classes for JMS 603
IBM WebSphere MQ classes for JMS 入门 605
安装和配置 IBM WebSphere MQ classes for JMS 606
面向程序员的简介 667
为 JMS 应用程序编写 IBM WebSphere MQ 类 674
应用程序服务器工具 (ASF) 775
使用 IBM WebSphere MQ JMS 管理工具 781
使用 IBM WebSphere MQ Explorer for JMS 配置 788
使用 WebSphere MQ Headers 软件包 788
与 WebSphere MQ classes for Java 配合使用 789
与 WebSphere MQ classes for JMS 配合使用 789
在 IBM WebSphere MQ 中使用 Web Service 791
IBM WebSphere MQ Transport for SOAP 791
IBM WebSphere MQ Bridge for HTTP 858
使用组件对象模型接口 (IBM WebSphere MQ Automation Classes for ActiveX) 867
使用 IBM WebSphere MQ Automation Classes for ActiveX 进行设计和编程 868
IBM WebSphere MQ Automation Classes for ActiveX 参考 873
故障诊断 935
到 MQAI 的 ActiveX 接口 939
关于 IBM WebSphere MQ Automation Classes for ActiveX 入门模板样本 947

声明.....953
编程接口信息.....954
商标.....954

开发应用程序

IBM WebSphere MQ 提供了多种方法，您可以通过这些方法开发应用程序来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。

在为 IBM WebSphere MQ 开发应用程序之前，请确保您熟悉 IBM WebSphere MQ 技术概述 IBM WebSphere MQ 技术概述中的概念。

您可以使用多种不同的编程语言为 IBM WebSphere MQ 开发应用程序。 有关受支持的编程语言及其功能的信息，请参阅 第 64 页的『决定要使用的编程语言』。

请参阅以下部分，以了解可以在不同平台上为 IBM WebSphere MQ 编写的应用程序类型。

您可以为 IBM WebSphere MQ 编写的应用程序类型

此信息是关于可以在 IBM WebSphere MQ 编写的应用程序类型的信息。

IBM WebSphere MQ 产品是队列管理器和应用程序启用程序。 它们支持 IBM 消息队列接口 (MQI)，通过该接口，程序可以将消息放入队列中并从队列中获取消息。

通过 IBM WebSphere MQ for non-z/OS 平台，您可以编写以下应用程序:

- 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 WebSphere MQ 平台上运行的应用程序发送消息。
- 在 CICS for TXSeries for AIX， TXSeries for HP-UX， TXSeries for Solaris 和 TXSeries for Windows 系统应用程序中使用消息排队。
- 对于 AIX， HP-UX， Solaris 和 Windows 系统，使用 Encina 中的消息排队。
- Use message queuing from within Tuxedo for AIX, AT&T, HP-UX, Solaris, and 窗口 systems.
- 将 IBM WebSphere MQ 用作事务管理器，协调外部资源管理器在 IBM WebSphere 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。
- 从完整 IBM WebSphere MQ 环境运行，或者从以下平台上的 IBM WebSphere MQ MQI 客户机环境运行:
  - UNIX and Linux® 系统
  - Windows

相关概念

安全性

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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

在开始设计和编写 IBM WebSphere MQ 应用程序之前，请先熟悉基本 IBM WebSphere MQ 概念，请参阅 技术概述中的主题。 有关可以为 IBM WebSphere MQ 编写的应用程序类型的信息，请参阅 第 7 页的『开发应用程序』。

使用以下链接来了解特定于应用程序开发的 IBM WebSphere MQ 概念:

- 第 9 页的『IBM WebSphere MQ 消息』

- 点到点消息传递
- WebSphere MQ 发布/预订消息传递简介
- 第 294 页的『在客户机应用程序中使用消息队列接口 (MQI)』
- 第 791 页的『在 WebSphere MQ 中使用 Web Service』
- 第 330 页的『消息传递通道的通道出口程序』
- 第 35 页的『业务支持方案』

必须先创建某些 IBM WebSphere 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 第 8 页的『使用 MQI 的应用程序』。

相关概念

第 73 页的『设计 IBM WebSphere MQ 应用程序』
当您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 WebSphere MQ 提供的功能。

第 78 页的『样本 WebSphere MQ 程序』
使用此主题集合来了解不同平台上的样本 WebSphere MQ 程序。

第 162 页的『编写排队应用程序』
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 293 页的『编写客户机应用程序』
在 WebSphere MQ 上编写客户机应用程序所需的知识。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 603 页的『使用 WebSphere MQ classes for JMS』
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) 是随 WebSphere MQ 提供的 JMS 提供程序。 除了实现 javax.jms 包中定义的接口外， WebSphere MQ JMS 类还提供了两组 JMS API 扩展。

第 867 页的『使用组件对象模型接口 (WebSphere MQ Automation Classes for ActiveX)』
WebSphere MQ Automation Classes for ActiveX (MQAX) 是 ActiveX 组件，用于提供可在应用程序中用于访问 WebSphere MQ 的类。

第 552 页的『使用 WebSphere MQ classes for Java』
WebSphere MQ Java 类使您能够在 Java 环境中使用 WebSphere MQ 。 Java 应用程序可以使用 WebSphere MQ classes for Java 或 WebSphere MQ classes for JMS 来访问 WebSphere MQ 资源。

第 470 页的『使用 .NET』
WebSphere MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 WebSphere MQ MQI 客户机连接到 WebSphere MQ ，或者直接连接到 WebSphere MQ 服务器。

第 529 页的『使用 C++』
WebSphere MQ 提供相当于 WebSphere MQ 对象的 C++ 类以及相当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 357 页的『构建 IBM WebSphere MQ 应用程序』
使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

使用 MQI 的应用程序

IBM WebSphere MQ 应用程序需要特定对象才能成功运行。

第 9 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。

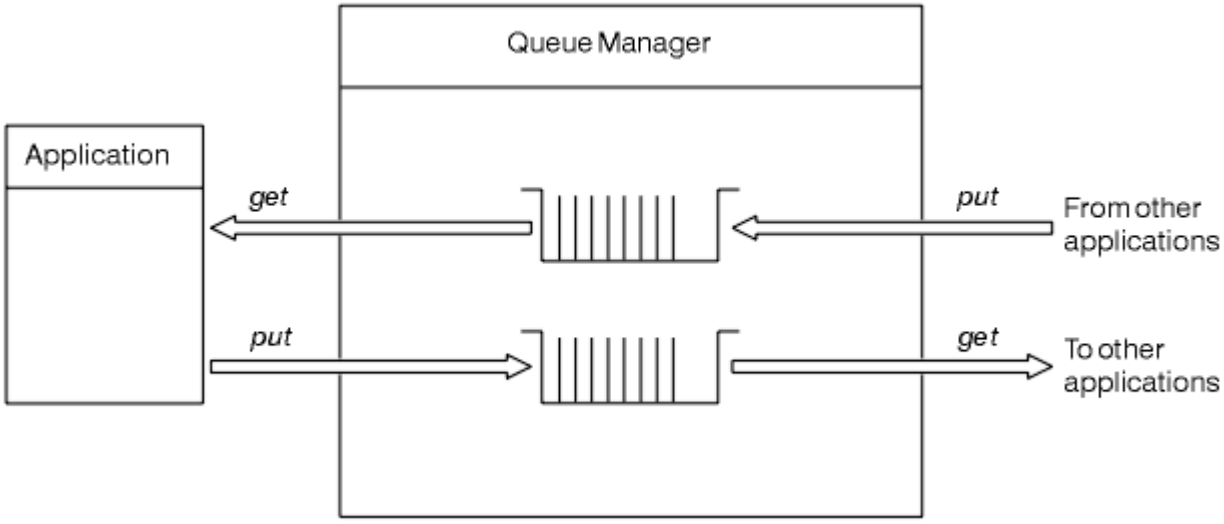


图 *1:* 队列、消息和应用程序

虽然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 (使用 MQGET) 获取消息。

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 WebSphere MQ。 请参阅第 357 页的『构建 IBM WebSphere MQ 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 9 页的图 1 中没有显示系统的这一部分。

IBM WebSphere MQ 消息

此信息介绍了 IBM WebSphere MQ 消息概念、消息组成部分和消息描述符。

IBM WebSphere MQ 消息由两部分组成：

- 消息属性
- 应用程序数据

第 9 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。

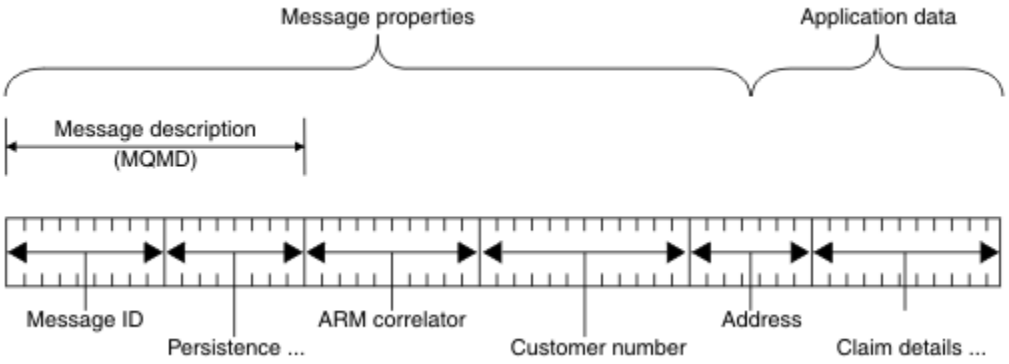


图 *2:* 消息说明

除非对其执行数据转换，否则队列管理器不会更改 WebSphere MQ 消息中携带的应用程序数据。 此外，WebSphere MQ 不会对此数据的内容施加任何限制。 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。

在 WebSphere MQ for AIX 上， WebSphere MQ for HP-UX， WebSphere MQ for Linux， WebSphere MQ for Solaris ， 和 WebSphere MQ for Windows， *MaxMsgLength* 缺省为 100 MB (104 857 600 字节)。

某些情况下，请让消息稍短于 *MaxMsgLength* 属性的值。 请参阅第 192 页的『消息中的数据』以获取更多信息。

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 结构来访问消息控制信息，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 32 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 30 页的『消息组』）的其他信息。 这与 V 1 消息描述符相同，但具有其他字段。 这些内容在 MQMDE-消息描述符扩展中进行了描述。

消息类型

IBM WebSphere 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 WebSphere MQ 定义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。）

报告和分段消息

在 WebSphere MQ for z/OS 上不受支持。

如果消息已分段（请参阅第 219 页的『消息分段』获取分短消息的说明），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。

对于 WebSphere 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

如果应用程序生成报告，请始终将原始消息数据开头的 WebSphere 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）。

您可以识别必须复制的 WebSphere MQ 头，方法是查看连续的格式名称，从 MQMD 开始，并继续处理存在的任何头。 以下 Format 名称指示这些 WebSphere 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 WebSphere MQ 头，它出现在同一位置。 头的长度包含在一个字段中，该字段也出现在 MQMDE， MQIMS 和所有 MQH* 头的相同位置。

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，不包括您找到的任何 WebSphere 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。

当队列上存在足够的报告消息 (例如， COA 和具有相同 *GroupId* 的报告消息) 来表示一条完整的原始消息时，将满足带有 MQGMO_COMPLETE_MSG 的 MQGET。 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* 关联，并使用 *Feedback* 字段来标识每个报告消息的类型。 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

如果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，那么 MQMDE 结构 (用于标识报告所表示的 *Offset* 和 *OriginalLength*) 将始终包括在报告数据中，此外还有零个， 100 个字节或消息中的所有原始数据。

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:

Format
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要，那么您可以定义自己的格式 (用户定义的格式) ，但不得对这些格式使用以 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）。

Encoding
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）
- WebSphere MQ 定义的消息，例如可编程命令格式

  WebSphere MQ 将可编程命令格式消息用于管理消息和事件 (在本例中使用的格式名称为 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 346 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

WebSphere 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。 如果这是混合代码页，那么 WebSphere MQ 使用混合代码页的 SBCS 部分作为队列管理器 CCSID。

对于常规数据转换，如果底层操作系统支持 DBCS 代码页，那么 WebSphere MQ 可以使用它。

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 346 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

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 181 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 WebSphere MQ 中，此属性的值为 9; 您可以创建优先级介于 0 (最低) 和 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。 它们还促进 WebSphere MQ 与 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。

在 WebSphere MQ 中使用消息属性会模拟在 JMS 中使用属性。 这意味着您可以在 JMS 应用程序中设置属性，并在过程 WebSphere MQ 应用程序中检索这些属性，或者以其他方式进行舍入。 要使某个属性可供 JMS 应用程序使用，请为其指定前缀 "usr"; 然后将其作为 JMS 消息用户属性提供 (不带前缀)。 例如， JMS 应用程序可以使用 JMS 调用 `message.getStringProperty('myproperty')` 来访问 WebSphere MQ 属性 *usr.myproperty* (字符串)。 请注意，如果 JMS 应用程序包含两个或更多 U+002E (".") ，那么这些应用程序无法访问前缀为 "usr" 的属性 字符。 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 WebSphere 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* 来控制可随 WebSphere MQ 队列管理器中的任何消息一起流动的属性的大小。

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。

当流向低于产品版本 7.0 的队列管理器时，消息的属性 (消息描述符中的属性除外) 将计入消息的长度。 因此，您应该根据需要提高转至版本低于 7.0 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。

由于 JMS 中没有最大属性名长度，因此 JMS 应用程序可以在存储在 MQRFH2 结构中时设置不是有效 WebSphere MQ 属性名的有效 JMS 属性名。

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。 当属性名被截断时， WebSphereMQ 会发出错误日志消息。

所有属性名称都必须遵循 Java 标识的 Java 语言规范定义的规则，但允许使用 Unicode 字符 U+002E (.) 作为名称的一部分，但不允许使用 start。 Java 标识的规则等同于属性名称的 JMS 规范中包含的规则。

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。

WebSphere MQ 属性名称将字符 U+002E (.) 用于属性的逻辑分组。 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，如果您正在使用域名 "`ourcompany.com`" 开发应用程序，那么可以使用前缀 "`com.ourcompany`" 来命名所有属性。 此命名约定还允许轻松选择属性; 例如，应用程序可以查询所有从 "`com.ourcompany.%`" 开始的消息属性。

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 WebSphere MQ JMS 类使用。
   - "usr.JMS"-无效。

   唯一的例外是提供 JMS 属性同义词的以下属性：

属性	同义词
JMSCorrelationID	根 。MQMD.CorrelId 或 jms.Cid
JMSDeliveryMode	根 。MQMD.Persistence 或 jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	根 。MQMD.Expiry 或 jms.Exp
JMSMessageID	根 。MQMD.MsgId
JMSPriority	根 。MQMD.Priority 或 jms.Pri

属性	同义词
JMSRedelivered	根 。MQMD.BackoutCount
JMSReplyTo（以 URI 编码的字符串）	根 。MQMD.ReplyToQ 或 Root 。MQMD.ReplyToQMgr 或 jms.Rto
JMSTimestamp	根 。MQMD.PutDate 或 Root 。MQMD.PutTime 或 jms.Tms
JMSType	mcd.Type 或 mcd.Set 或 mcd.Fmt
JMSXAppID	根 。MQMD.PutApplName
JMSXDeliveryCount	根 。MQMD.BackoutCount
JMSXGroupID	根 。MQMD.GroupId 或 jms.Gid
JMSXGroupSeq	根 。MQMD.MsgSeqNumber 或 jms.Seq
JMSXUserID	根 。MQMD.UserIdentifier

这些同义词允许 MQI 应用程序以类似于 WebSphere 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， WebSphere MQ JMS 类中提供的 JMS_IBM_ * 属性不可使用 MQI。 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 " 开头的属性名 不以 "mq_usr" 开头的任何小写或大写组合只能包含一个 "." 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 应用程序设置属性 "a.b" ，然后设置属性 "a.b.c" ，那么不清楚层次结构 "b" 中是包含值还是包含其他逻辑分组。 此类层次结构是 "混合内容" ， 此 不受支持。 不允许设置会产生混合内容的属性。

这些 限制由验证机制实施，如下所示:

- 如果在创建消息句柄时请求了验证，那么使用 MQSETMP - 设置消息属性 调用设置属性时将验证属性名称。 如果 尝试验证属性，但由于指定属性名称时发生错误而失败，那么完成代码为 MQCC_FAILED ，原因如下:
  - 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>	<Field>

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 概述中的 "MQPUT 和 MQPUT1 调用上指定的 MQMDE" 部分。

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。

字节字符串在 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 WebSphere MQ 支持的 Unicode 转换的 CCSID 会导致 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 20 页的图 3 显示了使用 MQSUB 调用的选择过程。

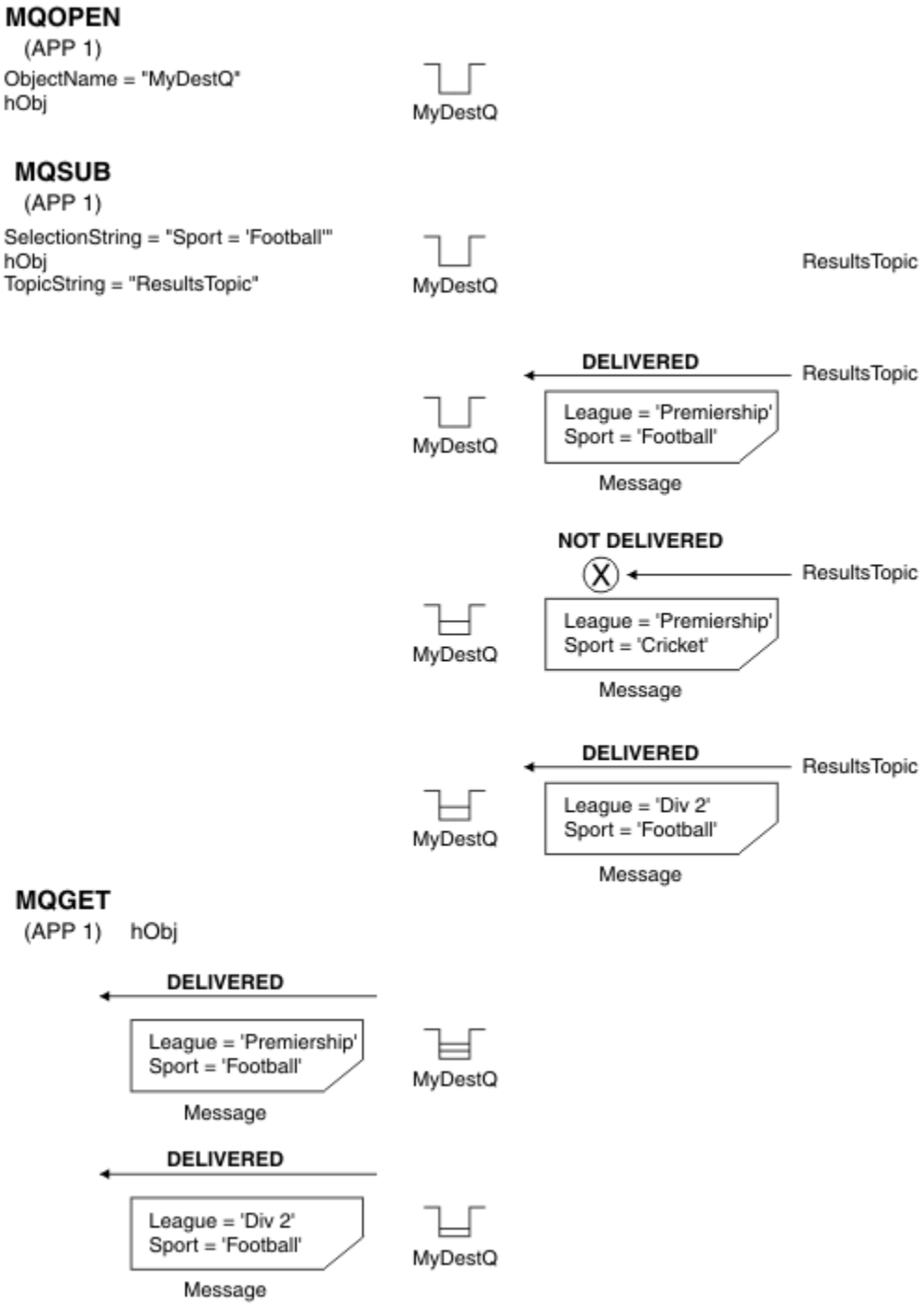


图 *3:* 使用 *MQSUB* 调用进行选择

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 21 页的图 4 显示了使用 MQOPEN 调用的选择过程。

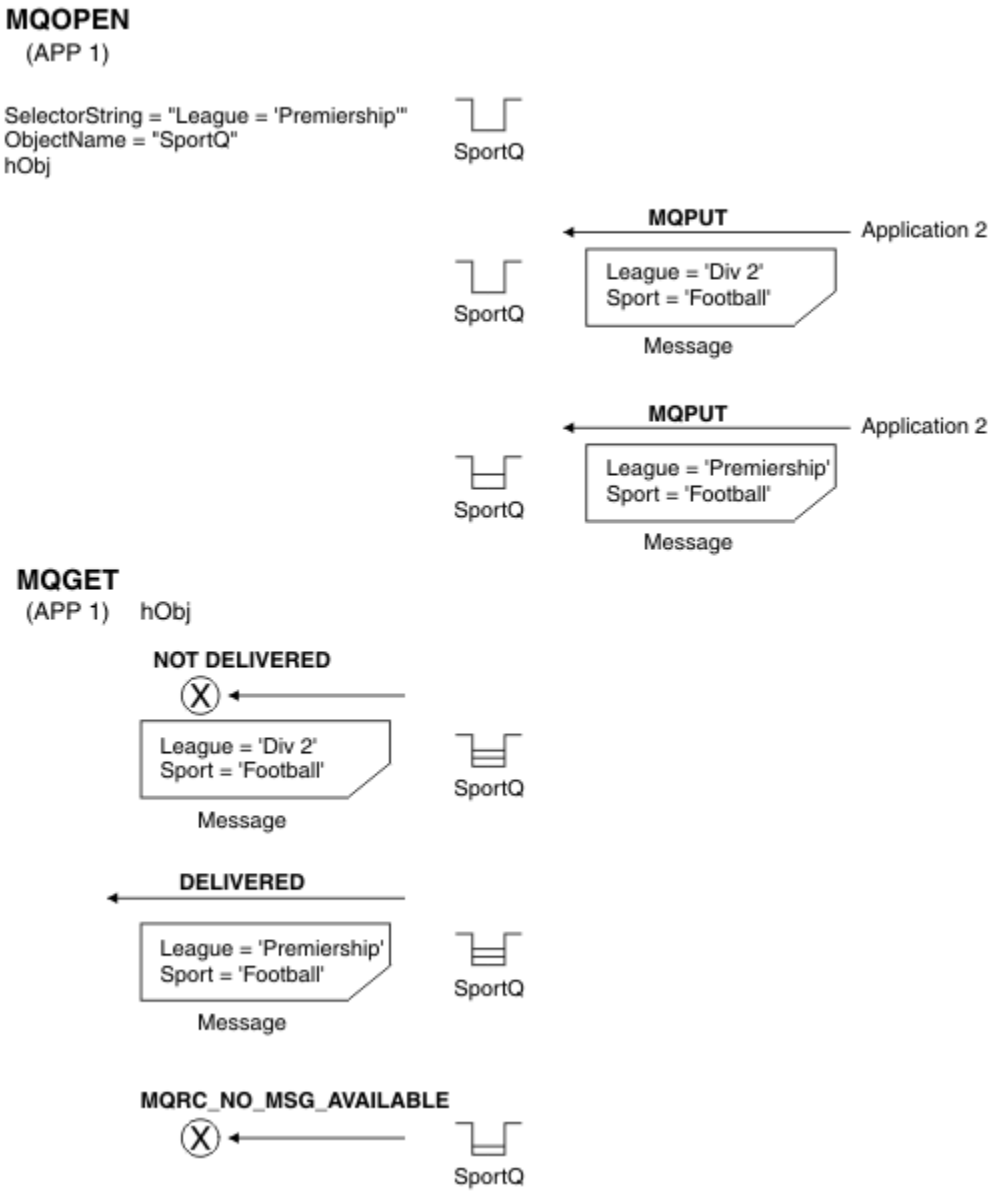


图 *4:* 使用 *MQOPEN* 调用进行选择

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。

选择行为
IBM WebSphere 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

在对消息属性进行任何匹配之前，可以将选择字符串解析为 TRUE 或 FALSE。 例如，如果选择字符串设置为 "TRUE <>FALSE"，那么可能是这种情况。 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，消息可能仍满足 `'Color IS NULL'`之类的选择字符串，其中 `'Color'` 不作为消息属性存在于消息中。

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 WebSphere 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。

要在选择基于 JMSCorrelationID 或 JMSMessageID 时优化深度队列上的消息选择，请使用格式为 `JMSCorrelationID = ...` 或 `JMSMessageID = ...` 的选择字符串并仅引用一个属性。

该方法能够大大提高基于 JMSCorrelationID 的选择的性能并提升 JMSMessageID 边际性能。

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 WebSphere 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。 在某些平台上进行了大约 100 次测试后，可能会发生保护，因此接近该数量的组件的选择器可能会出现故障。 建议在适当的平台上彻底尝试和测试使用具有许多组件的选择器，以确保未达到保护限制。

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

消息选择器语法

WebSphere MQ 消息选择器是一个语法基于 SQL92 条件表达式语法子集的字符串。

选择消息内容

可以根据消息有效内容的选择 (也称为内容过滤) 进行预订，但是 WebSphere MQ 无法直接执行将哪些消息传递到此类预订的决策; 而是需要扩展消息选择提供程序 (例如 IBM Integration Bus) 来处理消息。

消息选择器语法

WebSphere MQ 消息选择器是一个语法基于 SQL92 条件表达式语法子集的字符串。

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。

WebSphere MQ 在提供消息选择器时验证其语法正确性。 如果选择字符串的语法不正确或属性名称无效，且扩展的消息选择提供者不可用，将向应用程序返回 MQRC_SELECTION_NOT_AVAILABLE。 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。

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'。 与 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 (因为字节数不是 2 的倍数)
    - "myBytes = "0x0AFC2300"" = FALSE (因为尾部零在比较中很重要)
    - "myBytes = "0x000AFC23"" = FALSE (因为在比较中前导零很重要)
    - "myBytes = "0x23FC0A"" = FALSE (因为不考虑 endianness)
  - 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。
  - 精确数字文字是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。 WebSphere MQ 支持 -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807 范围内的精确数字。
  - 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。 WebSphere MQ 支持 -1.797693134862315E+308 到 1.797693134862315E+308 范围内的数字。

    有效字符应跟在可选的符号字符 (+ 或 -) 后面。 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 16 页的『属性名称』和第 17 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。

映射到属性名称或 MQMD 字段名称的 JMS 字段和属性名称也是选择字符串中的有效标识。 WebSphere MQ 将识别的 JMS 字段和属性名称映射到消息属性值。 请参阅第 676 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 >=" 在当前消息的 jms 文件夹中的 Pri 属性上选择。

- 溢出/下溢:

  对于十进制和近似数字，未定义以下内容:

  – 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: =， >， >=， <， <=和 <> (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。 一个例外是，比较精确数字值和近似数字值是有效的 (所需类型转换由 Java 数字提升规则定义)。 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` 等同于 `age >= 15 AND age <= 19`。
  - `Age NOT BETWEEN 15 and 19` 等同于 `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。 它等同于表达式 `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`。
  - `Country NOT IN ('UK', 'US', 'France')` 对于 `'UK'` 为 false，对于 `'Peru'` 为 true。 它等同于表达式 `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%'` 作为语法有效，但它求值为 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

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

表 *1:* 逻辑为 *A AND B* 时的布尔运算符结果 *(继续)*		
运算符 A	**运算符 B**	**结果 (A AND B)**
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		
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	
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

选择行为
IBM WebSphere MQ 选择行为概述。

选择消息内容
可以根据消息有效内容的选择 (也称为内容过滤) 进行预订，但是 WebSphere MQ 无法直接执行将哪些消息传递到此类预订的决策; 而是需要扩展消息选择提供程序 (例如 IBM Integration Bus) 来处理消息。

第 16 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。 它们还促进 WebSphere MQ 与 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。

- 使用单个等号字符检验等价; 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 `!=`。 此表示法不是 `<>`的有效同义词; 例如， `a <> b` 有效，而 `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 `||` 不是逻辑连接/分离的同义词; 例如，必须将 `a && b` 指定为 `a AND b`。
- 通配符 `*` 和 `?` 不是 `%` 和 `_`的同义词。
- 包含复合表达式 (例如 `20 < b < 30`) 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）。 而是必须将表达式写为 `(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` 关键字仅存在以支持 `IS NULL` 和 `IS NOT NULL` 案例。

相关概念

使用消息选择器时的 UTF-8 和 Unicode 注意事项

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+007F）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。

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。

相关概念

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。

选择消息内容

可以根据消息有效内容的选择 (也称为内容过滤) 进行预订，但是 WebSphere MQ 无法直接执行将哪些消息传递到此类预订的决策; 而是需要扩展消息选择提供程序 (例如 IBM Integration Bus) 来处理消息。

当应用程序在主题字符串上发布时 (其中一个或多个订户在消息内容上选择了选择字符串)， WebSphere MQ 将请求扩展消息选择提供程序解析该发布，并通知 WebSphere MQ 该发布是否与每个订户通过内容过滤器指定的选择标准相匹配。

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

选择行为
IBM WebSphere MQ 选择行为概述。

消息选择器语法
WebSphere MQ 消息选择器是一个语法基于 SQL92 条件表达式语法子集的字符串。

IBM WebSphere 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，但是在其他环境中， "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 29 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。

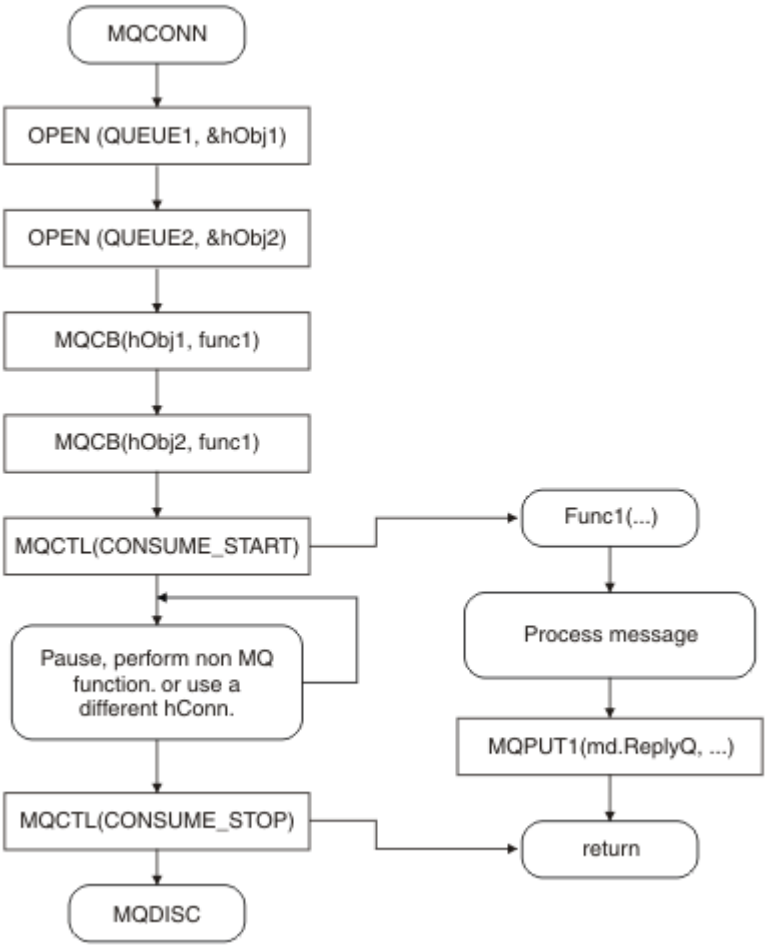


图 *5:* 使用来自两个队列的消息的标准消息驱动应用程序

第 30 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。

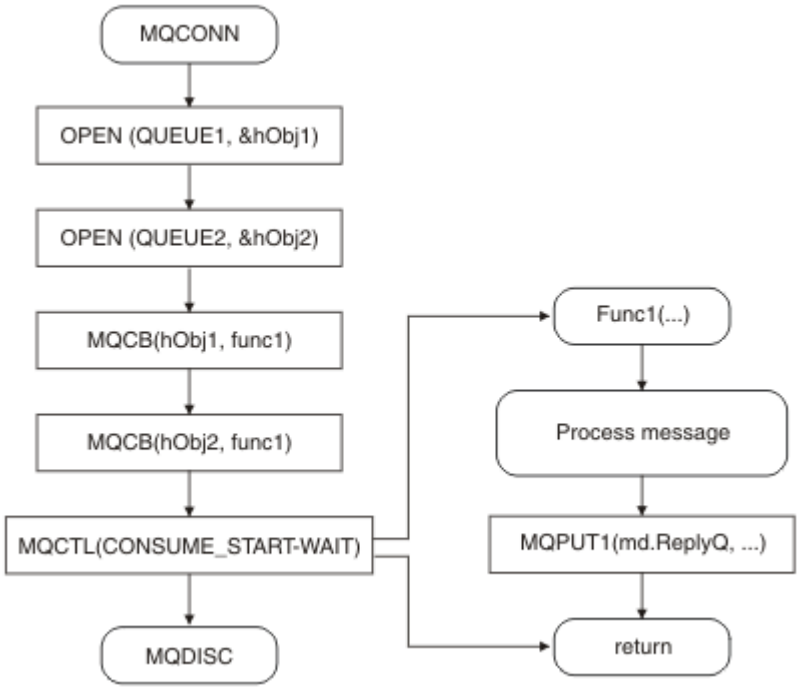


图 *6:* 使用来自两个队列的消息的单线程消息驱动应用程序

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 205 页的『逻辑与物理排序』）。 在非 z/OS(相关概念) 的平台上， 第 219 页的『消息分段』 支持将大型消息拆分为更小的段。 将消息放入主题时不能使用组合或分段消息。

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 30 页的图 7 显示了一组逻辑消息：

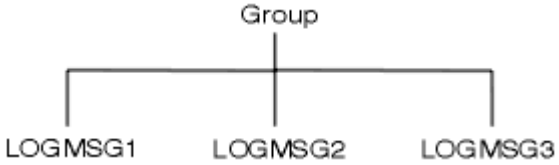


图 *7:* 逻辑消息组

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。 每条消息在逻辑上都是单独的消息，只有 MQMD 中的 *GroupId* 和 *MsgSeqNumber* 字段需要与组中的其他消息具有任何关系。 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。

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 219 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 31 页的图 8 显示了组中的消息示例）。 段在逻辑上是单个消息的一部分，因此只有 MQMD 中的 *MsgId*， *Offset* 和 *SegmentFlag* 字段应该在同一消息的不同段之间有所不同。 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 31 页的图 8 显示了一组逻辑消息，其中某些消息已分段：

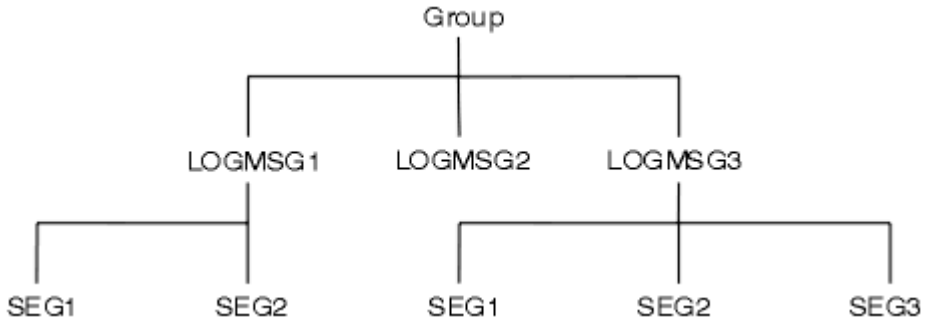


图 *8:* 分段消息

不能为发布/预订使用分段或组合消息。

有关逻辑消息和物理消息的描述，请参阅 第 205 页的『逻辑与物理排序』。 有关对消息进行分段的更多信息，请参阅 第 219 页的『消息分段』。

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。

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 271 页的『落实和回退工作单元』。
- 在 UNIX 系统上的 IBM WebSphere MQ， Linux 系统上的 IBM WebSphere MQ， 和 IBM WebSphere MQ for Windows 上的中，仅当落实工作单元 (并且工作单元可能包含许多队列操作) 时，才会记录当前工作单元中的持久消息。

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 462 页的『处理程序错误』以获取更多信息。

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。

在 WebSphere MQ for WebSphere MQ for Windows， WebSphere MQ on UNIX 系统， WebSphere MQ on Linux 系统 回退计数始终在队列管理器重新启动后。 对 *HardenGetBackout* 属性的任何更改将被忽略。

有关提交和回退消息的更多信息，请参阅第 271 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 464 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 193 页的『控制上下文信息』。

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 WebSphere 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

源上下文信息描述在队列上放置消息的应用程序，该队列指的是当前存储此消息的队列。 消息描述符包含源上下文信息的以下字段：

PutApplType	放置消息的应用程序的类型 (例如， CICS 事务)。
PutApplName	放置消息的应用程序的名称（例如，作业或事务的名称）。
PutDate	消息放入队列的日期。
PutTime	消息放入队列的时间。
ApplOriginData	应用程序希望包含的有关消息源的其他任何信息。 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。

WebSphere MQ 对象

此信息提供有关 WebSphere MQ 对象的详细信息，这些对象包括: 队列管理器，队列共享组，队列，管理主题对象，名称列表，进程定义，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）。 这些属性的值会影响 WebSphere 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。

使用 WebSphere MQ 命令来定义，改变或删除对象时，例如，队列管理器会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 193 页的『控制上下文信息』
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 186 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。

准备和运行 Microsoft Transaction Server 应用程序

要准备 MTS 应用程序以作为 WebSphere MQ MQI 客户机应用程序运行，请根据您的环境遵循以下指示信息。

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

要准备 MTS 应用程序以作为 WebSphere MQ MQI 客户机应用程序运行，请对该应用程序的每个组件执行下列其中一项操作:

- 如果组件为 MQI 使用 C 语言绑定，请遵循第 386 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 WebSphere MQ C++ 类，请遵循 第 551 页的『在 Windows 上构建 C++ 程序』 中的指示信息，但将组件与库 imqx23vn.lib 而不是 imqc23vn.lib 链接。
- 如果组件将 Visual Basic 语言绑定用于 MQI，请遵循第 390 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 WebSphere MQ Automation Classes for ActiveX (MQAX) ，请使用值 `mqic32xa.dll` 定义环境变量 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。

将 IBM WebSphere MQ 用于 WebSphere Application Server

使用本主题来了解 IBM WebSphere MQ 与 WebSphere Application Server 的用法。

在 WebSphere Application Server 下运行的以 Java 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。 此环境中的点到点消息传递可能由 IBM WebSphere MQ 队列管理器提供

使用 IBM WebSphere MQ 队列管理器提供点到点消息传递的好处是，连接 JMS 应用程序可以完全参与 IBM WebSphere MQ 网络的功能，这允许应用程序与在多个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的本地。 如果队列管理器不是应用程序的本地队列管理器，那么应安装 客户机连接 以允许应用程序连接到在另一台机器或映像上运行的队列管理器。

缺省情况下，保存在 IBM WebSphere MQ 队列上的 JMS 消息使用 MQRFH2 头来保存某些 JMS 消息头信息。 许多旧的 IBM WebSphere MQ 应用程序无法处理具有这些头的消息，并且需要它们自己的特征头，例

如，用于 CICS 网桥的 MQCIH 或用于 IBM WebSphere MQ Workflow 应用程序的 MQWIH。 有关这些特殊注意事项的更多详细信息，请参阅 第 678 页的『将 JMS 消息映射到 WebSphere MQ 消息』。

业务支持方案

通过使用事务支持，您可以使应用程序能够可靠地处理数据库。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

本部分介绍了事务支持。 使应用程序能够将 IBM WebSphere MQ 与数据库产品配合使用所需的工作跨应用程序编程和系统管理领域。 将此处的信息与 第 271 页的『落实和回退工作单元』一起使用。

我们首先介绍构成事务的工作单元，然后描述您启用 IBM WebSphere MQ 以将事务与数据库进行协调的方式。

相关概念

第 35 页的『介绍工作单元』
本主题介绍并定义工作单元，落实，回退和同步点的一般概念。 它还包含说明全局工作单元的两种方案。

IBM WebSphere MQ 和 HP NonStop TMF

介绍工作单元

本主题介绍并定义工作单元，落实，回退和同步点的一般概念。 它还包含说明全局工作单元的两种方案。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

当程序将消息放入工作单元中的队列时，仅当程序 落实 工作单元时，这些消息才会对其他程序可视。 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决定不使 put 操作永久存在，那么它可以 回退 工作单元。 当程序执行回退时，WebSphere MQ 通过除去该工作单元放在队列上的消息来复原队列。

同样，当程序从工作单元中的一个或多个队列获取消息时，这些消息将保留在队列上，直到程序落实工作单元为止，但这些消息不可供其他程序检索。 当程序落实工作单元时，将从队列中永久删除这些消息。 如果程序回退工作单元，那么 WebSphere MQ 将通过使消息可供其他程序检索来复原队列。

在最简单的情况下，将在任务结束时作出落实或回退更改的决定。 但是，应用程序在任务中的其他逻辑点同步数据更改可能更有用。 这些逻辑点称为同步点 (或同步点) ，在两个同步点之间处理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。

通过 WebSphere MQ，我们需要区分 本地 和 全局 工作单元:

本地工作单元

将唯一的操作放入并从 WebSphere MQ 队列中获取，并且使用 单阶段落实 进程在队列管理器中提供每个工作单元的协调。

当要更新的唯一资源是由单个 WebSphere MQ 队列管理器管理的队列时，请使用本地工作单元。 使用 MQCMIT 动词落实更新或使用 MQBACK 回退更新。

除了使用本地工作单元所涉及的日志管理以外，没有任何系统管理任务。 在将 MQPUT 和 MQGET 调用与 MQCMIT 和 MQBACK 配合使用的应用程序中，请尝试使用 MQPMO_SYNCPOINT 和 MQGMO_SYNCPOINT 选项。 (有关日志管理的信息，请参阅 管理日志文件 。)

全局工作单元

是否还会更新其他资源 (例如关系数据库中的表)。 如果涉及多个 资源管理器 ，那么需要使用 两阶段落实 进程来协调全局工作单元的 事务管理器 软件。

如果还需要包含对关系数据库管理器软件 (例如 Db2， Oracle， Sybase 和 Informix) 的更新，请使用全局工作单元。

有几种使用全局工作单元的可能方案。 此处记录了两种场景:

1. 首先，队列管理器本身充当事务管理器。 在此场景中， MQI 动词控制全局工作单元; 它们在应用程序中使用 MQBEGIN 动词启动，然后使用 MQCMIT 落实或使用 MQBACK 回退。

2. 在第二个角色中，事务管理员角色由其他软件 (例如 TXSeries， Encina 或 Tuxedo) 执行。 在此场景中，事务管理器软件提供的 API 用于控制工作单元 (例如， EXEC CICS SYNCPOINT for TXSeries)。

以下部分描述了使用全局工作单元所需的所有步骤 (按两种方案组织):

- 第 36 页的『方案 1: 队列管理器执行协调』
- 第 57 页的『方案 2: 其他软件提供协调』

方案 1: 队列管理器执行协调

在方案 1 中，队列管理器充当事务管理器。 在此场景中， MQI 动词控制全局工作单元; 它们在应用程序中使用 MQBEGIN 动词启动，然后使用 MQCMIT 落实或使用 MQBACK 回退。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

隔离级别

在 IBM WebSphere MQ 中，根据在数据库中实现的事务隔离设计，队列上的消息在数据库更新之前可能可视。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

当 IBM WebSphere MQ 队列管理器作为 XA 事务管理器工作时，要协调 XA 资源管理器的更新，请遵循以下落实协议:

1. 准备所有 XA 资源管理器。
2. 落实 IBM WebSphere MQ 队列管理器资源管理器。
3. 落实其他资源管理器。

在步骤 2 和 3 之间，应用程序可能会看到一条落实到队列的消息，但数据库中的相应行不会反映此消息。

如果配置了数据库以使应用程序的数据库 API 调用等待暂挂更新完成，那么这不是问题。

您可以通过以不同方式配置数据库来解决此问题。 所需配置的类型称为 "隔离级别"。 有关隔离级别的更多信息，请参阅数据库文档。 或者，您可以配置队列管理器以按以下反向顺序落实资源管理器:

1. 准备所有 XA 资源管理器。
2. 落实其他资源管理器。
3. 落实 IBM WebSphere MQ 队列管理器资源管理器。

更改协议时，将最后落实 IBM WebSphere MQ 队列管理器，因此从队列中读取消息的应用程序只有在完成相应的数据库更新后才会看到消息。

要配置队列管理器以使用此已更改的协议，请设置 **AMQ_REVERSE_COMMIT_ORDER** 环境变量。

在运行 **strmqm** 以启动队列管理器的环境中设置此环境变量。 例如，在启动队列管理器之前，请在 shell 中运行以下命令:

```
export AMQ_REVERSE_COMMIT_ORDER=1
```

注: 设置此环境变量可能会导致每个事务有额外的日志条目，因此这将对每个事务的性能产生较小的影响。

数据库协调

当队列管理器协调全局工作单元本身时，可以在工作单元内集成数据库更新。 即，可以编写混合 MQI 和 SQL 应用程序，并且可以使用 MQCMIT 和 MQBACK 动词来落实或回滚对队列和数据库的更改。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

队列管理器使用 *X/Open* 分布式事务处理*: XA* 规范中描述的两阶段落实协议来实现此目标。 要落实工作单元时，队列管理器首先会询问每个参与的数据库管理器是否准备落实其更新。 仅当所有参与者 (包括队列管理器本身) 准备落实时，才会落实所有队列和数据库更新。 如果任何参与者无法准备其更新，那么将改为回退工作单元。

通常，全局工作单元通过以下方法在应用程序中实现 (伪代码):

```
MQBEGIN
MQGET (在消息选项中包含标志 MQGMO_SYNCPOINT)
MQPUT (在消息选项中包含标志 MQPMO_SYNCPOINT)
SQL INSERT
MQCMIT
```

MQBEGIN 的目的是表示全局工作单元的开始。 MQCMIT 的目的是表示全局工作单元的结束，并使用两阶段落实协议在所有参与的资源管理器中完成该工作单元。

当使用 MQCMIT 成功完成工作单元 (也称为 事务) 时，在该工作单元中执行的所有操作都将永久或不可逆。 如果工作单元因任何原因而失败，那么将改为回退所有操作。 无法使工作单元中的一个操作成为永久操作，而另一个操作已回退。 这是工作单元的原则: 要么工作单元内的所有操作都是永久性的，要么都不是永久性的。

注:

1. 应用程序员可以通过调用 MQBACK 来强制回退工作单元。 如果在调用 MQCMIT 之前应用程序或数据库失败 ，那么队列管理器也会回退工作单元。
2. 如果应用程序在不调用 MQCMIT 的情况下调用 MQDISC ，那么队列管理器的行为就像调用了 MQCMIT 一样，并落实工作单元。

在 MQBEGIN 和 MQCMIT 之间，队列管理器不会对数据库进行任何调用以更新其资源。 即，更改数据库表的唯一方式是由您的代码 (例如，伪代码中的 SQL INSERT) 进行更改。

如果队列管理器在落实协议期间失去与任何数据库管理器的联系，那么将提供完全恢复支持。 如果数据库管理器在处于不确定状态时变为不可用，即，它已成功准备落实，但尚未收到落实或回退决策，那么队列管理器将记住工作单元的结果，直到该结果成功交付到数据库为止。 同样，如果队列管理器因未完成的落实操作而终止，那么将在队列管理器重新启动时记住这些操作。 如果应用程序意外终止，那么工作单元的完整性不会受到损害，但结果取决于应用程序在流程中终止的位置，如 第 37 页的表 5 中所述。

下表概述了在数据库或应用程序失败时发生的情况:

表 *4:* 数据库服务器发生故障时发生的情况

故障发生	**结果**
在应用程序调用 MQCMIT 之前。	工作单元已回退。
在应用程序调用 MQCMIT 期间， **之前** 所有数据库都指示已成功准备。	工作单元已回退，原因码为 MQRC_BACKED_OUT。
在应用程序调用 MQCMIT 期间， **之后** 所有数据库都指示已成功准备，但之前都指示已成功落实。	工作单元由队列管理器保持在可恢复状态，原因码为 MQRC_OUTCOME_PENDING。
在应用程序调用 MQCMIT 期间， **之后** 所有数据库都指示它们已成功落实。	已落实工作单元，原因码为 MQRC_NONE。
在应用程序调用 MQCMIT 之后。	已落实工作单元，原因码为 MQRC_NONE。

表 *5:* 应用程序失败时发生的情况

故障发生	**结果**
在应用程序调用 MQCMIT 之前。	工作单元已回退。
在应用程序调用 MQCMIT 期间， **之前** 队列管理器已接收到应用程序的 MQCMIT 请求。	工作单元已回退。
在应用程序调用 MQCMIT 期间， **在** 队列管理器接收到应用程序的 MQCMIT 请求之后。	队列管理器尝试使用两阶段落实 (取决于数据库产品成功执行并落实其部分工作单元) 进行落实。

如果从 MQCMIT 返回的原因码为 MQRC_OUTCOME_PENDING，那么队列管理器将记住工作单元，直到它能够重新建立与数据库服务器的联系，并告诉它落实其部分工作单元为止。 请参阅 第 51 页的『丢失与 XA 资源管理器的联系时的注意事项』，以获取有关如何以及何时执行恢复的信息。

队列管理器使用 XA 接口与数据库管理器通信，如 *X/Open* 分布式事务处理*: XA* 规范中所述。 这些函数调用的示例包括 xa_open， xa_start， xa_end， xa_prepare 和 xa_commit。 我们使用术语 事务管理器 和 资源管理器 与 XA 规范中使用的术语相同。

限制
数据库协调支持存在限制。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

具有以下限制:

- 在 MQI 客户机应用程序中 **不** 支持在 WebSphere MQ 工作单元内协调数据库更新的能力。 在客户机应用程序中使用 MQBEGIN 失败。 调用 MQBEGIN 的程序必须在与队列管理器相同的机器上作为 服务器 应用程序运行。

  注: 服务器 应用程序是已与必需的 WebSphere MQ 服务器库链接的程序; 客户机 应用程序是已与必需的 WebSphere MQ 客户机库链接的程序。 有关编译和链接程序的详细信息，请参阅 第 297 页的『为 WebSphere MQ MQI 客户机构建应用程序』 和 第 357 页的『构建 IBM WebSphere MQ 应用程序』。
- 数据库服务器可以驻留在与队列管理器服务器不同的机器上，只要数据库客户机与队列管理器安装在同一机器上，它就支持此功能。 请参阅数据库产品的文档以确定其客户机软件是否可用于两阶段落实系统。
- 虽然队列管理器充当资源管理器 (为了参与方案 2 全局工作单元)，但无法使一个队列管理器在其方案 1 全局工作单元中协调另一个队列管理器。

切换装入文件

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

切换装入文件是由 IBM WebSphere MQ 应用程序和队列管理器中的代码装入的共享库 (Windows 系统上的 DLL)。 其目的是简化数据库的客户机共享库的装入，并将指针返回到 XA 函数。

必须在启动队列管理器之前指定交换机装入文件的详细信息。 详细信息放在 Windows UNIX and Linux 系统上的 qm.ini 文件中。

- 在 Windows 和 Linux (x86 和 x86-64 平台) 系统上，使用 IBM WebSphere MQ Explorer 来更新 qm.ini 文件。
- 在所有其他系统上，直接编辑文件 qm.ini。

如果 IBM WebSphere MQ 安装支持方案 1 全局工作单元，那么会随该安装提供交换机装入文件的 C 源。 源包含名为 MQStart 的函数。 装入交换机装入文件时，队列管理器将调用此函数，这将返回称为 *XA* 交换机的结构的地址。

XA 切换结构存在于数据库客户机共享库中，并且包含许多函数指针，如 第 38 页的表 6 中所述:

表 *6: XA* 切换函数指针

函数指针名称	**XA 函数**	**用途**
xa_open_entry	xa_open	连接到数据库
xa_close_entry	xa_close	与数据库断开连接
xa_start_entry	xa_start	启动全局工作单元的分支
xa_end_entry	xa_end	暂挂全局工作单元的分支
xa_rollback_entry	XA_rollback	回滚全局工作单元的分支
xa_prepare_entry	xa_prepare	准备落实全局工作单元的分支
xa_commit_entry	xa_commit	落实全局工作单元的分支

表 6: XA 切换函数指针 (继续)		
函数指针名称	**XA 函数**	**用途**
xa_recover_entry	xa_recover	从数据库中发现它是否具有不确定的工作单元
xa_forget_entry	xa_算了	允许数据库忘记全局工作单元的分支
xa_complete_entry	xa_complete	完成全局工作单元的分支

在应用程序中的第一个 MQBEGIN 调用期间，作为 MQBEGIN 的一部分执行的 IBM WebSphere MQ 代码将装入切换装入文件，并在数据库共享库中调用 xa_open 函数。 类似地，在队列管理器启动期间以及在其他后续场合，某些队列管理器进程装入切换装入文件并调用 xa_open。

您可以使用 动态注册来减少 xa_ * 调用数。 有关此优化技术的完整描述，请参阅 第 55 页的『XA 动态注册』。

配置系统以进行数据库协调

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

在数据库管理器可以参与由队列管理器协调的全局工作单元之前，必须执行若干任务。 这些描述如下:

- 第 39 页的『安装和配置数据库产品』
- 第 39 页的『创建交换机装入文件』
- 第 40 页的『向队列管理器添加配置信息』
- 第 41 页的『编写和修改应用程序』
- 第 42 页的『测试系统』

安装和配置数据库产品
要安装和配置数据库产品，请参阅产品自己的文档。 本部分中的本主题描述了常规配置问题以及它们如何与 WebSphere MQ 与数据库之间的互操作相关。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

数据库连接

与队列管理器建立标准连接的应用程序与单独的本地队列管理器代理进程中的线程相关联。 (不是 *fastpath* 连接的连接是此上下文中的 标准 连接。 有关更多信息，请参阅 第 175 页的『使用 MQCONNX 调用连接到队列管理器』。)

当应用程序发出 MQBEGIN 时，它和代理进程都会在数据库客户机库中调用 xa_open 函数。 为此，数据库客户机库代码 将 连接到 来自应用程序和队列管理器进程的工作单元中涉及的数据库。 只要应用程序保持与队列管理器的连接，就会维护这些数据库连接。

如果数据库仅支持有限数量的用户或连接，那么这是一个重要的注意事项，因为正在与数据库建立两个连接以支持一个应用程序。

客户机/服务器配置

装入到 WebSphere MQ 队列管理器和应用程序进程中的数据库客户机库 **必须** 能够发送到其服务器并从其服务器接收。 请确保:

- 数据库的客户机/服务器配置文件具有正确的详细信息
- 相关环境变量是在队列管理器 **和** 应用程序进程的环境中设置的

创建交换机装入文件
WebSphere MQ 随附样本 makefile ， 用于为受支持的数据库管理器构建切换装入文件。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

样本 makefile 与构建交换机装入文件所需的所有关联 C 源文件一起安装在以下目录中:

- 对于 WebSphere MQ for Windows，请在 *MQ_INSTALLATION_PATH*\tools\c\samples\xatm\ 目录中
- 对于 WebSphere MQ for UNIX and Linux 系统，请在 *MQ_INSTALLATION_PATH*/samp/xatm/ 目录中

用于构建交换机装入文件的样本源模块包括:

- 对于 DB2， db2swit.c
- 对于 Oracle， oraswit.c
- 对于 Informix， infswit.c
- 对于 Sybase， sybswit.c

生成交换机装入文件时，请在 /var/mqm/exits 中安装 32 位交换机装入文件，并在 /var/mqm/exits64 中安装 64 位交换机装入文件。

如果您有 32 位队列管理器，那么样本 make 文件 xaswit.mak 将在 /var/mqm/exits 中安装 32 位交换机装入文件。

如果您有 64 位队列管理器，那么样本 make 文件 xaswit.mak 将在 /var/mqm/exits 中安装 32 位交换机装入文件，在 /var/mqm/exits64 中安装 64 位交换机装入文件。

文件安全性

由于 WebSphere MQ 无法控制的原因，操作系统可能无法通过 WebSphere MQ 装入交换机装入文件。 如果发生此情况，那么会将错误消息写入 WebSphere MQ 错误日志，并且可能导致 MQBEGIN 调用失败。 为了帮助确保操作系统不会使交换机装入文件的装入失败，您必须满足以下要求:

1. 切换装入文件必须在 qm.ini 文件中提供的位置中可用。
2. 交换机装入文件必须可供需要装入该文件的所有进程 (包括队列管理器进程和应用程序进程) 访问。
3. 交换机装入文件所依赖的所有库 (包括数据库产品提供的库) 都必须存在且可访问。

向队列管理器添加配置信息
为数据库管理器创建切换装入文件并将其放在安全位置时，必须向队列管理器指定该位置。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

要指定位置，请执行以下步骤:

- 在 Windows 和 Linux (x86 和 x86-64 平台) 系统上，使用 WebSphere 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。
- 在所有其他系统上，在队列管理器的 qm.ini 文件的 XAResourceManager 节中指定交换机装入文件的详细信息。

为队列管理器要协调的数据库添加 XAResourceManager 节。 最常见的情况是只有一个数据库，因此只有一个 XAResourceManager 节。 有关涉及多个数据库的更复杂的配置的详细信息，请参阅 第 50 页的『多个数据库配置』。 XAResourceManager 节的属性如下所示:

名称 = 名称
用户选择的用于标识资源管理器的字符串。 实际上，它为 XAResourceManager 节提供了名称。 该名称是必需的，长度最多可以为 31 个字符。

您选择的名称必须唯一; 此 qm.ini 文件中必须只有一个具有此名称的 XAResourceManager 节。 该名称也应该有意义，因为在使用 dspmqtrn 命令时，队列管理器使用它来在队列管理器错误日志消息和输出中引用此资源管理器。 (请参阅 第 52 页的『使用 dspmqtrn 命令显示未完成的工作单元』 以获取更多信息。)

选择名称并启动队列管理器后，请勿更改 "名称" 属性。 有关更改配置信息的更多详细信息，请参阅 第 54 页的『更改配置信息』。

SwitchFile= 名称
这是先前构建的 XA 切换装入文件的名称。 这是必需属性。 队列管理器和 WebSphere MQ 应用程序进程中的代码两次尝试装入交换机装入文件:

1. 在队列管理器启动时
2. 在 WebSphere MQ 应用程序进程中首次调用 MQBEGIN 时

交换机装入文件的安全性和许可权属性必须允许这些进程执行此操作。

XAOpenString= 字符串
这是 WebSphere MQ 代码在其对数据库管理器的 xa_open 函数的调用中传递的数据字符串。 这是可选属性; 如果省略此属性，那么将假定字符串长度为零。

队列管理器和 WebSphere MQ 应用程序进程中的代码两次调用 xa_open 函数:

1. 在队列管理器启动时
2. 在 WebSphere MQ 应用程序进程中首次调用 MQBEGIN 时

此字符串的格式特定于每个数据库产品，将在该产品的文档中进行描述。 通常， xa_open 字符串包含认证信息 (用户名和密码) ，以允许连接到队列管理器和应用程序进程中的数据库。

XACloseString= 字符串
这是 WebSphere MQ 代码在其对数据库管理器的 xa_close 函数的调用中传递的数据字符串。 这是可选属性; 如果省略此属性，那么将假定字符串长度为零。

队列管理器和 WebSphere MQ 应用程序进程中的代码两次调用 xa_close 函数:

1. 在队列管理器启动时
2. 在 WebSphere MQ 应用程序进程中调用 MQDISC 时，先前已调用 MQBEGIN

此字符串的格式特定于每个数据库产品，将在该产品的文档中进行描述。 通常，该字符串为空，并且通常会从 XAResourceManager 节中省略 XACloseString 属性。

ThreadOfControl=THREAD |PROCESS
ThreadOf 控制值可以是 THREAD 或 PROCESS。 队列管理器将其用于序列化目的。 这是可选属性; 如果省略此属性，那么将采用值 PROCESS。

如果数据库客户机代码允许线程在不序列化的情况下调用 XA 函数，那么 ThreadOfControl 的值可以是 THREAD。 队列管理器假定它可以在必要时同时从多个线程调用数据库客户机共享库中的 XA 函数。

如果数据库客户机代码不允许线程以此方式调用其 XA 函数，那么 ThreadOfControl 的值必须为 PROCESS。 在这种情况下，队列管理器会序列化对数据库客户机共享库的所有调用，以便一次仅从特定进程中发出一个调用。 如果应用程序使用多个线程运行，那么您可能还需要确保应用程序执行类似的序列化。

请注意，数据库产品以这种方式处理多线程进程的能力是该产品的供应商面临的问题。 有关是否可以将 ThreadOf 控制属性设置为 THREAD 或 PROCESS 的详细信息，请参阅数据库产品的文档。 如果可以，建议将 ThreadOf 控件设置为 THREAD。 如果不确定，那么 更安全 选项将其设置为 PROCESS ，尽管您将失去使用 THREAD 的潜在性能优势。

编写和修改应用程序

如何实现全局工作单元。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

第 35 页的『介绍工作单元』中描述了随 WebSphere MQ 安装提供的方案 1 全局工作单元的样本应用程序。

通常，全局工作单元通过以下方法在应用程序中实现 (伪代码):

```
MQBEGIN
MQGET
MQPUT
```

SQL INSERT
MQCMIT

MQBEGIN 的目的是表示全局工作单元的开始。 MQCMIT 的目的是表示全局工作单元的结束，并使用两阶段落实协议在所有参与的资源管理器中完成该工作单元。

在 MQBEGIN 和 MQCMIT 之间，队列管理器不会对数据库进行任何调用以更新其资源。 即，更改数据库表的唯一方式是由您的代码 (例如，伪代码中的 SQL INSERT) 进行更改。

就数据库而言，队列管理器的作用是告诉它全局工作单元何时开始，何时结束，以及全局工作单元是应该落实还是回滚。

就应用程序而言，队列管理器执行两个角色: 资源管理器 (其中资源是队列上的消息) 和全局工作单元的事务管理器。

从提供的样本程序开始，完成在这些程序中进行的各种 WebSphere MQ 和数据库 API 调用。 相关 API 调用在 第 78 页的『样本 WebSphere MQ 程序』， MQI 中使用的数据类型以及 (对于数据库自己的 API) 数据库自己的文档中都有完整记录。

测试系统
您知道是否仅通过在测试期间运行应用程序和系统来正确配置这些应用程序和系统。 您可以通过构建和运行提供的其中一个样本程序来测试系统的配置 (队列管理器与数据库之间的成功通信)。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

配置 *Db2*

DB2 支持和配置信息。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

Db2 的受支持级别是在 IBM WebSphere MQ 详细系统需求 页面上定义的。

注: 在队列管理器为 64 位的平台上，不支持 32 位 Db2 实例。

请执行以下操作:

1. 请检查环境变量设置。
2. 创建 Db2 交换机装入文件。
3. 添加资源管理器配置信息。
4. 必要时更改 Db2 配置参数。

请结合 第 39 页的『配置系统以进行数据库协调』中提供的一般信息阅读本信息。

警告: 如果在 UNIX and Linux 平台上运行 `db2profile` ，那么将设置环境变量 LIBPATH 和 LD_LIBRARY_PATH。 建议使用 `unset` 这些环境变量，请参阅相应的 快速入门 指南。

检查 Db2 环境变量设置

确保为队列管理器进程 **以及应用程序进程** 设置了 Db2 环境变量。 特别是，必须始终设置启动队列管理器的 DB2INSTANCE 环境变量 **前** 。 DB2INSTANCE 环境变量标识包含要更新的 Db2 数据库的 Db2 实例。 例如:

- 在 UNIX and Linux 系统上，使用:

```
export DB2INSTANCE=db2inst1
```

- 在 Windows 系统上，使用:

```
set DB2INSTANCE=DB2
```

在具有 Db2 数据库的 Windows 上，必须将用户 MUSR_MQADMIN 添加到 DB2USERS 组，以使队列管理器能够启动。

创建 Db2 交换机装入文件

创建 Db2 交换机装入文件的最简单方法是使用样本文件 xaswit.mak， WebSphere MQ 提供该样本文件来为各种数据库产品构建交换机装入文件。

在 Windows 系统上，可以在目录 *MQ_INSTALLATION_PATH*\tools\c\samples\xatm 中找到 xaswit.mak 。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。 要使用 Microsoft Visual C++ 创建 Db2 交换机装入文件，请使用:

```
nmake /f xaswit.mak db2swit.dll
```

生成的交换机文件放置在 c:\Program Files\IBM\WebSphere MQ\exits 中。

您可以在目录 *MQ_INSTALLATION_PATH*/samp/xatm 中找到 xaswit.mak 。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。

编辑 xaswit.mak 以 取消注释 适用于您正在使用的 Db2 版本的行。 然后使用以下命令执行 makefile:

```
make -f xaswit.mak db2swit
```

生成的 32 位交换机装入文件将放在 /var/mqm/exits 中。

生成的 64 位交换机装入文件放置在 /var/mqm/exits64 中。

为 Db2 添加资源管理器配置信息

您必须修改队列管理器的配置信息，以将 Db2 声明为全局工作单元的参与者。 在 第 40 页的『向队列管理器添加配置信息』 中的更多详细信息中描述了如何以此方式修改配置信息。

- 在 Windows 和 Linux (x86 和 x86-64 平台) 系统上，使用 WebSphere 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。
- 在所有其他系统上，在队列管理器的 qm.ini 文件的 XAResourceManager 节中指定交换机装入文件的详细信息。

第 43 页的图 9 是 UNIX 样本，显示一个 XAResourceManager 条目，其中要协调的数据库名为 mydbname，此名称在 XAOpenString 中指定:

```
XAResourceManager:
  Name=mydb2
  SwitchFile=db2swit
  XAOpenString=mydbname,myuser,mypasswd,toc=t
  ThreadOfControl=THREAD
```

图 *9: UNIX* 平台上 *Db2* 的样本 *XAResourceManager* 条目

注:

1. ThreadOfControl=THREAD 不能与低于 V 8 的 Db2 版本配合使用。 将 ThreadOfControl 和 XAOpenString 参数 toc 设置为下列其中一个组合:
   - ThreadOfControl=THREAD 和 toc=t
   - ThreadOfControl=PROCESS 和 toc=p

   如果要使用 jdbcdb2 XA 切换装入文件来启用 JDBC/JTA 协调，那么必须使用 ThreadOfControl=PROCESS 和 toc=p。

更改 Db2 配置参数

对于队列管理器正在协调的每个 Db2 数据库，必须设置数据库特权，更改 tp_mon_name 参数，并重置 maxappls 参数。 要进行此操作，请执行以下步骤:

设置数据库特权

队列管理器进程在 UNIX and Linux 系统上以有效用户和组 mqm 运行。 在 Windows 系统上，它们以启动队列管理器的用户身份运行。 这可以是以下之一：

1. 发出 `strmqm` 命令的用户，或者
2. 运行 IBM MQSeries 服务 COM 服务器的用户

缺省情况下，此用户称为 MUSR_MQADMIN。

如果未在 xa_open 字符串上指定用户名和密码，那么 Db2 将使用 **运行队列管理器的用户** 来认证 xa_open 调用。 如果此用户 (例如， UNIX and Linux 系统上的用户 mqm) 在数据库中没有最低特权，那么数据库将拒绝认证 xa_open 调用。

同样的注意事项也适用于您的应用程序进程。 如果未在 xa_open 字符串上指定用户名和密码，那么 Db2 将使用运行应用程序的用户来认证在第一个 MQBEGIN 期间发出的 xa_open 调用。 同样，此用户必须在数据库中具有最低特权才能工作。

例如，通过发出以下 Db2 命令，授予 mydbname 数据库中的 mqm 用户连接权限：

```
db2 connect to mydbname
db2 grant connect on database to user mqm
```

有关安全性的更多信息，请参阅 第 51 页的『安全性注意事项』 。

Windows 更改 TP_MON_NAME 参数

仅对于 Db2 for Windows 系统，请更改 TP_MON_NAME 配置参数以命名 Db2 用于调用队列管理器以进行动态注册的 DLL。

使用命令 `db2 update dbm cfg using TP_MON_NAME mqmax` 来命名 MQMAX.DLL 作为 Db2 用于调用队列管理器的库。 这必须存在于 PATH 中的目录中。

重置 maxappls 参数

您可能需要查看 *maxappls* 参数的设置，这将限制可以连接到数据库的最大应用程序数。 请参阅第 39 页的『安装和配置数据库产品』。

配置 *Oracle*

Oracle 支持和配置信息。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

完成以下步骤：

1. 检查环境变量设置。
2. 创建 Oracle 交换机装入文件。
3. 添加资源管理器配置信息。
4. 根据需要更改 Oracle 配置参数。

在 IBM WebSphere MQ 详细系统需求 页面上提供了 IBM WebSphere MQ 支持的 Oracle 级别的当前列表。

检查 Oracle 环境变量设置

确保为队列管理器进程以及应用程序进程设置了 Oracle 环境变量。 尤其是，在启动队列管理器之前，请始终设置以下环境变量：

ORACLE_HOME

Oracle 主目录。 例如，在 UNIX and Linux 系统上，使用：

```
export ORACLE_HOME=/opt/oracle/product/8.1.6
```

在 Windows 系统上，使用:

```
set ORACLE_HOME=c:\oracle\ora81
```

ORACLE_SID
正在使用的 Oracle SID。 如果要将 Net8 用于客户机/服务器连接，那么可能不必设置此环境变量。 请参阅 Oracle 文档。

后续示例是在 UNIX and Linux 系统上设置此环境变量的示例:

```
export ORACLE_SID=sid1
```

Windows 系统上的等效项为:

```
set ORACLE_SID=sid1
```

注: 必须将 PATH 环境变量设置为包含二进制文件目录 (例如 ORACLE_INSTALL_DIR/VERSION/32BIT_NAME/bin 或 ORACLE_INSTALL_DIR/VERSION/64BIT_NAME/bin)，否则您可能会看到一条消息，指出机器中缺少 oraclient 库。

如果在 Windows 64 位系统上运行队列管理器，那么必须同时安装 64 位和 32 位 Oracle 客户机。 必须安装这两个客户机，因为队列管理器作为使用 32 位交换机装入文件的 32 位进程运行，而该进程又必须启动 32 位 Oracle 客户机 dll。

由 64 位队列管理器装入的交换机装入文件必须访问 Oracle 64 位客户机库。 当 IBM WebSphere MQ 在 Windows 64 位系统上运行时， 32 位队列管理器必须访问 32 位 Oracle 客户机。

创建 Oracle 交换机装入文件

要创建 Oracle 交换机装入文件，请使用样本文件 xaswit.mak， IBM WebSphere MQ 提供此文件来为各种数据库产品构建交换机装入文件。 在 Windows 系统上，可以在目录 C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm 中找到 xaswit.mak 。 要使用 Microsoft Visual C++ 创建 Oracle 交换机装入文件，请使用: nmake /f xaswit.mak oraswit.dll

生成的交换机文件放置在 *MQ_INSTALLATION_PATH*\exits 中。*MQ_INSTALLATION_PATH* 表示安装了 IBM WebSphere MQ 的高级目录。

您可以在目录 *MQ_INSTALLATION_PATH*/samp/xatm 中找到 xaswit.mak 。 *MQ_INSTALLATION_PATH* 表示安装了 IBM WebSphere MQ 的高级目录。

编辑 xaswit.mak 以取消注释与您正在使用的 Oracle 版本相应的行。 然后使用以下命令执行 makefile:

```
make -f xaswit.mak oraswit
```

生成的 32 位交换机装入文件放置在 /var/mqm/exits 中。

生成的 64 位交换机装入文件放置在 /var/mqm/exits64 中。

为 Oracle 添加资源管理器配置信息

您必须修改队列管理器的配置信息，以将 Oracle 声明为全局工作单元中的参与者。 在 第 40 页的『向队列管理器添加配置信息』中更详细地描述了如何以此方式修改队列管理器的配置信息。

- 在 Windows 和 Linux (x86 和 x86-64 平台) 系统上，使用 IBM WebSphere 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。
- 在所有其他系统上，在队列管理器的 qm.ini 文件的 XAResourceManager 节中指定交换机装入文件的详细信息。

第 46 页的图 10 是显示 XAResourceManager 条目的 UNIX and Linux 系统样本。 必须将 LogDir 添加到 XA 打开字符串，以便将所有错误和跟踪信息记录到同一位置。

```
XAResourceManager:
  Name=myoracle
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypasswd+SesTm=35+LogDir=/tmp+threads=true
  ThreadOfControl=THREAD
```

图 *10: UNIX and Linux* 平台上 *Oracle* 的样本 *XAResourceManager* 条目

注:

1. 在 第 46 页的图 10 中， xa_open 字符串已与四个参数配合使用。 可以包括其他参数，如 Oracle 的文档中所述。
2. 使用 IBM WebSphere MQ 参数 `ThreadOfControl=THREAD` 时，必须使用 XAResourceManager 节中的 Oracle 参数 `+threads=true` 。

有关 xa_open 字符串的更多信息，请参阅 *Oracle8 Server Application Developer 's Guide* 。

更改 Oracle 配置参数

对于队列管理器正在协调的每个 Oracle 数据库，必须查看最大会话数并设置数据库特权。 为此，请完成以下步骤:

查看最大会话数
您可能必须查看 LICENSE_MAX_SESSIONS 和 PROCESSES 设置以考虑属于队列管理器的进程所需的其他连接。 请参阅第 39 页的『安装和配置数据库产品』以获取更多详细信息。

设置数据库特权
xa_open 字符串中指定的 Oracle 用户名必须具有访问 DBA_PENDING_TRANSACTIONS 视图的特权，如 Oracle 文档中所述。

可以使用以下示例命令来授予必需的特权:

```
grant select on DBA_PENDING_TRANSACTIONS to myuser;
```

配置 *Informix*

Informix 支持和配置信息。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

完成以下步骤:

1. 确保已安装相应的 Informix 客户端 SDK:
   - 32 位队列管理器和应用程序需要 32 位 Informix 客户端 SDK。
   - 64 位队列管理器和应用程序需要 64 位 Informix 客户端 SDK。
2. 确保正确创建了 Informix 数据库 。
3. 检查环境变量设置。
4. 构建 Informix 切换装入文件。
5. 添加资源管理器配置信息。

在 IBM WebSphere MQ 详细系统需求 页面上提供了 WebSphere MQ 支持的 Informix 的当前级别列表。

确保正确创建 Informix 数据库

必须创建由 WebSphere MQ 队列管理器协调的每个 Informix 数据库，并指定 `log` 参数。 例如:

```
create database mydbname with log;
```

WebSphere MQ 队列管理器无法协调在创建时未指定 log 参数的 Informix 数据库。 如果队列管理器尝试协调在创建时未指定 log 参数的 Informix 数据库，那么对 Informix 的 xa_open 调用将失败，并生成大量 FFST 错误。

检查 Informix 环境变量设置

确保为队列管理器进程 **以及应用程序进程** 设置了 Informix 环境变量。 尤其是，在启动队列管理器 **之前** ，请始终设置以下环境变量:

INFORMIXDIR
Informix 产品安装目录。

- 对于 32 位 UNIX and Linux 应用程序，请使用以下命令:

```
export INFORMIXDIR=/opt/informix/32-bit
```

- 对于 64 位 UNIX and Linux 应用程序，请使用以下命令:

```
export INFORMIXDIR=/opt/informix/64-bit
```

- 对于 Windows 应用程序，请使用以下命令:

```
set INFORMIXDIR=c:\informix
```

对于具有必须同时支持 32 位和 64 位应用程序的 64 位队列管理器的系统，您需要同时安装 Informix 32 位和 64 位客户机 SDK。 用于创建切换装入文件的样本 makefile xaswit.mak 还会设置两个产品安装目录。

INFORMIXSERVER
Informix 服务器的名称。 例如，在 UNIX and Linux 系统上，使用:

```
export INFORMIXSERVER=hostname_1
```

在 Windows 系统上，使用:

```
set INFORMIXSERVER=hostname_1
```

ONCONFIG
Informix 服务器配置文件的名称。 例如，在 UNIX and Linux 系统上，使用:

```
export ONCONFIG=onconfig.hostname_1
```

在 Windows 系统上，使用:

```
set ONCONFIG=onconfig.hostname_1
```

创建 Informix 交换机装入文件

要创建 Informix 切换装入文件，请使用样本文件 xaswit.mak， WebSphere MQ 提供该样本文件来构建各种数据库产品的切换装入文件。 在 Windows 系统上，可以在目录 *MQ_INSTALLATION_PATH*\tools\c\samples\xatm 中找到 xaswit.mak 。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。 要使用 Microsoft Visual C++ 创建 Informix 切换装入文件，请使用:

```
nmake /f xaswit.mak infswit.dll
```

生成的交换机文件放置在 c:\Program Files\IBM\WebSphere MQ\exits 中。

您可以在目录 *MQ_INSTALLATION_PATH*/samp/xatm 中找到 xaswit.mak 。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。

编辑 xaswit.mak 以 取消注释 与您正在使用的 Informix 版本相应的行。 然后使用以下命令执行 makefile:

```
make -f xaswit.mak infswit
```

生成的 32 位交换机装入文件放置在 /var/mqm/exits 中。

生成的 64 位交换机装入文件放置在 /var/mqm/exits64 中。

为 Informix 添加资源管理器配置信息

您必须修改队列管理器的配置信息，以将 Informix 声明为全局工作单元中的参与者。 在 第 40 页的『向队列管理器添加配置信息』中更详细地描述了如何以此方式修改队列管理器的配置信息。

- 在 Windows 和 Linux (x86 和 x86-64 平台) 系统上，使用 WebSphere 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。
- 在所有其他系统上，在队列管理器的 qm.ini 文件的 XAResourceManager 节中指定交换机装入文件的详细信息。

第 48 页的图 11 是 UNIX 样本，显示 qm.ini XAResourceManager 条目，其中要协调的数据库名为 mydbname，此名称在 XAOpenString 中指定:

```
XAResourceManager:
  Name=myinformix
  SwitchFile=infswit
  XAOpenString=DB=mydbname@myinformixserver\;USER=myuser\;PASSWD=mypasswd
  ThreadOfControl=THREAD
```

图 *11: UNIX* 平台上 *Informix* 的样本 *XAResourceManager* 条目

注: 缺省情况下， UNIX 平台上的样本 xaswit.mak 会创建一个使用线程化 Informix 库的交换机装入文件。 使用这些 Informix 库时，必须确保 ThreadOfControl 设置为 THREAD。 在 第 48 页的图 11 中， qm.ini 文件 XAResourceManager 节属性 ThreadOf 控件设置为 THREAD。 指定 THREAD 时，必须使用线程 Informix 库和 WebSphere MQ 线程 API 库来构建应用程序。

XAOpenString 属性必须包含数据库名称，后跟 @ 符号，然后后跟 Informix 服务器名称。

要使用非线程 Informix 库，必须确保 qm.ini 文件 XAResourceManager 节属性 ThreadOfControl 设置为 PROCESS。 您还必须对样本 xaswit.mak:

1. 取消注释非线程交换机装入文件的生成。
2. 注释掉线程切换装入文件的生成。

Sybase 配置

Sybase 支持和配置信息。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

完成以下步骤:

1. 确保已安装 Sybase XA 库，例如通过安装 XA DTM 选项。
2. 检查环境变量设置。
3. 启用 Sybase XA 支持。
4. 创建 Sybase 交换机装入文件。
5. 添加资源管理器配置信息。

在 IBM WebSphere MQ 详细系统需求 页面上提供了 WebSphere MQ 支持的 Sybase 级别的当前列表。

检查 Sybase 环境变量设置

确保为队列管理器进程 **以及** 应用程序进程设置了 Sybase 环境变量。 尤其是，在启动队列管理器 **之前** ，请始终设置以下环境变量:

SYBASE
Sybase 产品安装的位置。 例如，在 UNIX and Linux 系统上，使用:

```
export SYBASE=/sybase
```

在 Windows 系统上，使用:

```
set SYBASE=c:\sybase
```

SYBASE_OCS
已安装 Sybase 客户机文件的 SYBASE 下的目录。 例如，在 UNIX and Linux 系统上，使用:

```
export SYBASE_OCS=OCS-12_0
```

在 Windows 系统上，使用:

```
set SYBASE_OCS=OCS-12_0
```

启用 Sybase XA 支持

在 Sybase XA 配置文件 $SYBASE/$SYBASE_OCS/xa_config 中，为要更新的 Sybase 服务器的每个连接定义逻辑 Resource Manager (LRM)。 第 49 页的图 12 中显示了 $SYBASE/$SYBASE_OCS/xa_config 的内容示例。

```
# The first line must always be a comment
[xa]
  LRM=lrmname
  server=servername
```

图 *12: $SYBASE/ $SYBASE_OCS/xa_config* 的示例内容

创建 Sybase 交换机装入文件

要创建 Sybase 切换装入文件，请使用 WebSphere MQ 随附的样本文件。在 Windows 系统上，可以在目录 C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm 中找到 xaswit.mak 。 要使用 Microsoft Visual C++ 创建 Sybase 切换装入文件，请使用:

```
nmake /f xaswit.mak sybswit.dll
```

生成的交换机文件放置在 c:\Program Files\IBM\WebSphere MQ\exits 中。

您可以在目录 *MQ_INSTALLATION_PATH*/samp/xatm 中找到 xaswit.mak 。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。

编辑 xaswit.mak 以 取消注释 适合于您正在使用的 Sybase 版本的行。 然后使用以下命令执行 makefile:

```
make -f xaswit.mak sybswit
```

生成的 32 位交换机装入文件将放在 /var/mqm/exits 中。

生成的 64 位交换机装入文件放置在 /var/mqm/exits64 中。

为 Sybase 添加资源管理器配置信息

您必须修改队列管理器的配置信息，以将 Sybase 声明为全局工作单元中的参与者。 第 40 页的『向队列管理器添加配置信息』中更详细地描述了如何修改配置信息。

- 在 Windows 和 Linux (x86 和 x86-64 平台) 系统上，使用 WebSphere 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。
- 在所有其他系统上，在队列管理器的 qm.ini 文件的 XAResourceManager 节中指定交换机装入文件的详细信息。

第 50 页的图 13 显示了 UNIX and Linux 样本，该样本使用与 Sybase XA 配置文件 $SYBASE/$SYBASE_OCS/xa_config 中的 *lrmname* LRM 定义相关联的数据库。 如果要记录 XA 函数调用，请包含日志文件名:

```
XAResourceManager:
  Name=mysybase
  SwitchFile=sybswit
  XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
  ThreadOfControl=THREAD
```

图 *13: UNIX and Linux* 平台上 *Sybase* 的样本 *XAResourceManager* 条目

将多线程程序与 Sybase 配合使用

如果将多线程程序与 WebSphere MQ 全局工作单元配合使用，并将更新合并到 Sybase，那么 **必须** 将值 THREAD 用于 ThreadOfControl 参数。 还要确保将程序 (和交换机装入文件) 与线程安全 Sybase 库 (_r 版本) 链接。 将值 THREAD 用于 ThreadOf 控制参数将在 第 50 页的图 13 中显示。

多个数据库配置

如果要配置队列管理器以便可以在全局工作单元中包含对多个数据库的更新，请为每个数据库添加 XAResourceManager 节。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

如果数据库全部由同一数据库管理器管理，那么每个节定义一个单独的数据库。 每个节指定相同的 *SwitchFile*，但 *XAOpenString* 的内容不同，因为它指定要更新的数据库的名称。 例如， 第 50 页的图 14 中显示的节通过 UNIX and Linux 系统上的 Db2 数据库 *MQBankDB* 和 *MQFeeDB* 来配置队列管理器。

要点: 不能有多个节指向同一数据库。 此配置在任何情况下都不起作用，如果您尝试此配置，那么它将失败。

您将收到格式为 when the MQ code makes its second xa_open call in any process in this environment, the database software fails the second xa_open with a -5 error, XAER_INVAL 的错误。

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=DB2 MQFeeDB
  SwitchFile=db2swit
  XAOpenString=MQFeeDB
```

图 *14:* 多个 *Db2* 数据库的样本 *XAResourceManager* 条目

如果要更新的数据库由不同的数据库管理器管理，请为每个数据库管理器添加 XAResourceManager 节。 在这种情况下，每个节指定不同的 *SwitchFile*。 例如，如果 *MQFeeDB* 由 Oracle 而不是 DB2 管理，请在 UNIX and Linux 系统上使用以下节:

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=Oracle MQFeeDB
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypassword+SesTm=35+LogDir=/tmp/ora.log+DB=MQFeeDB
```

图 *15: DB2* 和 *Oracle* 数据库的样本 *XAResourceManager* 条目

原则上，可以使用单个队列管理器配置的数据库实例数没有限制。

注: 有关支持在全局工作单元中的多个数据库更新中包含 Informix 数据库的信息，请检查产品自述文件。

安全性注意事项

在 XA 模型下运行数据库的注意事项。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

提供以下信息仅供参考。 在所有情况下，请参阅数据库管理器随附的文档，以确定在 XA 模型下运行数据库的安全性影响。

应用程序进程表示使用 MQBEGIN 动词启动全局工作单元。 第一个 MQBEGIN 调用是应用程序问题通过在 xa_open 入口点调用其客户机库代码来连接到所有参与数据库。 所有数据库管理器都提供了在其 XAOpenString 中提供用户标识和密码的机制。 这是认证信息流动的唯一时间。

请注意，在 UNIX and Linux 平台上，快速路径应用程序在进行 MQI 调用时必须使用有效用户标识 mqm 运行。

丢失与 *XA* 资源管理器的联系时的注意事项

队列管理器允许数据库管理器不可用。 这意味着您可以独立于数据库服务器启动和停止队列管理器。 恢复联系人后，队列管理器和数据库再同步。 您还可以使用 rsvmqtrn 命令来手动解析不确定的工作单元。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

在正常操作中，完成配置步骤后仅需要最少量的管理。 由于队列管理器允许数据库管理器不可用，因此使管理作业更容易执行。 这尤其意味着:

- 队列管理器可以随时启动，而无需首先启动每个数据库管理器。
- 如果其中一个数据库管理器变为不可用，那么队列管理器不需要停止并重新启动。

这允许您独立于数据库服务器启动和停止队列管理器。

每当队列管理器与数据库之间失去联系时，它们都需要在两者再次变为可用时再同步。 再同步是完成涉及该数据库的任何不确定工作单元的过程。 通常，这会自动发生，而无需用户干预。 队列管理器要求数据库提供不确定的工作单元列表。 然后，它指示数据库落实或回滚其中每个不确定的工作单元。

当队列管理器启动时，它将与每个数据库再同步。 当单个数据库变为不可用时，仅当队列管理器通知该数据库再次可用时，才需要对该数据库进行再同步。

当使用 MQBEGIN 启动新的全局工作单元时，队列管理器将自动恢复与先前不可用的数据库的联系。 它通过在数据库客户机库中调用 xa_open 函数来执行此操作。 如果此 xa_open 调用失败，那么 MQBEGIN 将返回完成代码 MQCC_WARNING 和原因码 MQRC_PARTICIPANT_NOT_AVAILABLE。 您可以稍后重试 MQBEGIN 调用。

请勿继续尝试涉及在 MQBEGIN 期间指示失败的数据库更新的全局工作单元。 将不存在与可通过其进行更新的数据库的连接。 唯一的选项是结束程序，或者定期重试 MQBEGIN ， 以希望数据库可能再次可用。

或者，可以使用 `rsvmqtrn` 命令来显式解析所有不确定的工作单元。

不确定的工作单元

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

如果在指示数据库管理器准备之后丢失了与队列管理器的联系，那么数据库可能会保留不确定的工作单元。 在数据库服务器从队列管理器接收结果 (落实或回滚) 之前，它需要保留与更新关联的数据库锁定。

由于这些锁定会阻止其他应用程序更新或读取数据库记录，因此需要尽快进行再同步。

如果由于某种原因，您无法等待队列管理器自动与数据库再同步，那么可以使用数据库管理器提供的工具来手动落实或回滚数据库更新。 在 *X/Open* 分布式事务处理*: XA* 规范中，这称为作出 启发式 决策。 仅将其用作最后手段，因为可能会损害数据完整性; 例如，当所有其他参与者已落实其更新时，您可能会错误地回滚数据库更新。

最好重新启动队列管理器，或者在重新启动数据库后使用 `rsvmqtrn` 命令来启动自动再同步。

使用 *dspmqtrn* 命令显示未完成的工作单元

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

当数据库管理器不可用时，可以使用 **dspmqtrn** 命令来检查涉及该数据库的未完成全局工作单元的状态。

dspmqtrn 命令仅显示其中一个或多个参与者处于不确定状态的工作单元。 参与者正在等待队列管理器的决策，以落实或回滚已准备的更新。

对于其中每个全局工作单元，将在 **dspmqtrn** 的输出中显示每个参与者的状态。 如果工作单元未更新特定资源管理器的资源，那么不会显示该工作单元。

关于不确定的工作单元，据说资源管理器已执行下列其中一项操作:

准备
资源管理器已准备好落实其更新。

已落实
资源管理器已落实其更新。

已回滚
资源管理器已回滚其更新。

参与
资源管理器是参与者，但尚未准备，落实或回滚其更新。

重新启动队列管理器时，它会要求每个具有 XAResourceManager 节的数据库获取其不确定全局工作单元的列表。 如果数据库尚未重新启动，或者在其他情况下不可用，那么队列管理器无法向数据库交付这些工作单元的最终结果。 当数据库再次可用时，会将不确定工作单元的结果第一次传递到数据库。

在这种情况下，报告数据库管理器处于 *prepared* 状态，直到发生再同步为止。

每当 `dspmqtrn` 命令显示不确定的工作单元时，它都会首先列出可能参与的所有可能的资源管理器。 为这些资源分配了唯一标识 *RMId*，在报告其与不确定工作单元相关的状态时，将使用该标识来代替资源管理器的 名称 。

样本 dspmqtrn 输出 显示发出以下命令的结果:

```
dspmqtrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB.
AMQ7107: Resource manager 2 is DB2 MQFeeDB.

AMQ7056: Transaction number 0,1.
    XID: formatID 5067085, gtrid_length 12, bqual_length 4
         gtrid [3291A50600002013746573 74]
         bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

其中， 事务号 是可与 rsvmqtrn 命令配合使用的事务的标识。 请参阅 AMQ7000-7999: WebSphere MQ 产品，以获取有关 AMQ7056 消息的更多信息。 *XID* 变量是 *X/Open XA* 规范的一部分; 有关此规范的最新信息，请参阅: https://publications.opengroup.org/c193。

图 *16:* 样本 *dspmqtrn* 输出

Sample dspmqtrn output 中的输出显示有三个资源管理器与队列管理器相关联。 第一个是资源管理器 0，它是队列管理器本身。 其他两个资源管理器实例是 MQBankDB 和 MQFeeDB Db2 数据库。

此示例仅显示单个不确定工作单元。 将针对所有三个资源管理器发出一条消息，这意味着已在工作单元中对队列管理器和两个 Db2 数据库进行了更新。

对队列管理器 **0** 进行的更新已 落实。 Db2 数据库的更新处于 已准备 状态，这意味着在调用 Db2 以落实对 *MQBankDB* 和 *MQFeeDB* 数据库的更新之前，它必须已变为不可用。

不确定工作单元具有称为 XID (事务标识) 的外部标识。 这是队列管理器提供给 Db2 的一部分数据，用于标识其在全局工作单元中的部分。

使用 *rsvmqtrn* 命令解析未完成的工作单元

队列管理器和 DB2 再同步时，未完成的工作单元完成。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

第 53 页的图 16 中显示的输出显示了一个不确定的工作单元，在该工作单元中，落实决策尚未交付到两个 DB2 数据库。

要完成此工作单元，队列管理器和 DB2 需要在 DB2 下次变为可用时再同步。 队列管理器将新工作单元的启动用作重新与 DB2 取得联系的机会。 或者，您可以指示队列管理器使用 **rsvmqtrn** 命令显式再同步。

在重新启动 DB2 之后立即执行此操作，以便尽快释放与不确定工作单元相关联的任何数据库锁定。 使用 -a 选项，它指示队列管理器解析所有不确定的工作单元。 在以下示例中， DB2 已重新启动，因此队列管理器可以解析不确定的工作单元:

```
> rsvmqtrn -m MY_QMGR -a
Any in-doubt transactions have been resolved.
```

混合结果和错误

虽然队列管理器使用两阶段落实协议，但这不会完全消除某些工作单元完成混合结果的可能性。 在此情况下，一些参与者会落实其更新，而一些参与者会回退其更新。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

以混合结果完成的工作单元会产生严重影响，因为本应更新为单个工作单元的共享资源不再处于一致状态。

主要在对工作单元做出启发式决策时导致混合结果，而不是允许队列管理器解决不确定的工作单元本身。 此类决策不在队列管理器的控制范围内。

每当队列管理器检测到混合结果时，它都会生成 FFST 信息，并在其错误日志中记录故障，并显示以下两条消息之一:

- 如果数据库管理器回滚而不是落实:

```
AMQ7606 A transaction has been committed but one or more resource
        managers have rolled back.
```

- 如果数据库管理器落实而不是回滚:

```
AMQ7607 A transaction has been rolled back but one or more resource
        managers have committed.
```

进一步的消息标识试探性损坏的数据库。 然后，您负责在本地恢复受影响数据库的一致性。 这是一个复杂的过程，您需要首先隔离错误落实或回滚的更新，然后手动撤销或重做数据库更改。

更改配置信息

在成功启动队列管理器以协调全局工作单元之后，请勿更改任何资源管理器配置信息。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

如果需要更改配置信息，您可以随时执行此操作，但这些更改直到重新启动队列管理器之后才会生效。

如果除去数据库的资源管理器配置信息，那么将有效除去队列管理器与该数据库管理器联系的能力。

从不 更改任何资源管理器配置信息中的 名称 属性。 此属性向队列管理器唯一地标识该数据库管理器实例。 如果更改此唯一标识，那么队列管理器假定已除去数据库并添加了全新的实例。 队列管理器仍将未完成的工作单元与旧的 名称相关联，这可能使数据库处于不确定状态。

除去数据库管理器实例

如果需要从配置中永久除去数据库，请确保在重新启动队列管理器之前该数据库未处于不确定状态。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

数据库产品提供用于列出不确定事务的命令。 如果存在任何不确定事务，请首先允许队列管理器与数据库再同步。 通过启动队列管理器来执行此操作。 您可以使用 **rsvmqtrn** 命令或数据库自己的命令来查看不确定的工作单元，以验证是否已执行再同步。 一旦您确信已进行再同步，请结束队列管理器并除去数据库的配置信息。

如果未能观察到此过程，那么队列管理器仍会记住涉及该数据库的所有不确定工作单元。 每次重新启动队列管理器时，都会发出一条警告消息 AMQ7623。 如果不再使用队列管理器配置此数据库，请使用 **rsvmqtrn** 命令的 -r 选项指示队列管理器忘记数据库参与其不确定事务。 仅当所有参与者都已完成不确定事务时，队列管理器才会忘记此类事务。

有时可能需要临时除去某些资源管理器配置信息。 在 UNIX and Linux 系统上，最好通过注释掉该节，以便可以在以后轻松恢复该节。 如果队列管理器每次与特定数据库或数据库管理器联系时都发生错误，那么您可能决定执行此操作。 临时除去相关的资源管理器配置信息允许队列管理器启动涉及所有其他参与者的全局工作单元。 以下是已注释掉的 XAResourceManager 节的示例:

```
# This database has been temporarily removed
#XAResourceManager:
#   Name=mydb2
#   SwitchFile=db2swit
#   XAOpenString=mydbname,myuser,mypassword,toc=t
#   ThreadOfControl=THREAD
```

图 *17:* 已注释掉 *UNIX and Linux* 系统上的 *XAResourceManager* 节

在 Windows 系统上，使用 WebSphere MQ Explorer 来删除有关数据库管理器实例的信息。 在恢复时，请非常小心地在 名称 字段中输入正确的名称。 如果错误输入名称，那么可能会迁到不确定问题，如 第 54 页的『更改配置信息』中所述。

XA 动态注册

XA 规范提供了一种减少事务管理器对资源管理器进行的 xa_* 调用数的方法。 此优化称为 动态注册。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

DB2 支持动态注册。 其他数据库可能支持此操作; 请参阅数据库产品的文档以获取详细信息。

为什么动态注册优化有用? 在应用程序中，某些全局工作单元可能包含对数据库表的更新; 其他工作单元可能不包含此类更新。 如果未对数据库的表进行持久更新，那么无需将该数据库包含在 MQCMIT 期间发生的落实协议中。

无论您的数据库是否支持动态注册，您的应用程序都会在 WebSphere MQ 连接上的第一个 MQBEGIN 调用期间调用 xa_open 。 它还会在后续 MQDISC 调用上调用 xa_close 。 后续 XA 调用的模式取决于数据库是否支持动态注册:

如果您的数据库不支持动态注册 ...

每个全局工作单元都涉及由 WebSphere MQ 代码对数据库客户机库进行的若干 XA 函数调用，而无论您是否在工作单元中对该数据库的表进行了持久更新。 其中包括:

- 应用程序进程中的 xa_start 和 xa_end 。 这些用于声明全局工作单元的开始和结束。
- 来自队列管理器代理进程 amqzlaa0 的 xa_prepare， xa_commit 和 xa_rollback 。 这些用于交付全局工作单元的结果: 落实或回滚决策。

此外，队列管理器代理进程还会在第一个 MQBEGIN 期间调用 xa_open 。

如果您的数据库支持动态注册 ...

WebSphere MQ 代码仅执行必需的 XA 函数调用。 对于 **未** 涉及数据库资源的持久更新的全局工作单元，**不会** 对数据库进行 XA 调用。 对于 **已** 涉及此类持久更新的全局工作单元，调用的目的是:

- 来自应用程序进程的 xa_end ，用于声明全局工作单元的结束。
- 来自队列管理器代理进程 amqzlaa0 的 xa_prepare， xa_commit 和 xa_rollback 。 这些用于交付全局工作单元的结果: 落实或回滚决策。

要使动态注册生效，数据库必须能够在执行持久更新时告知 WebSphere MQ 它希望包含在当前全局工作单元中，这一点至关重要。 WebSphere MQ 为此提供了 ax_reg 功能。

在应用程序进程中运行的数据库的客户机代码将找到 ax_reg 函数并对其进行调用，以 动态注册 它在当前全局工作单元中执行持久工作的事实。 作为对此 ax_reg 调用的响应， WebSphere MQ 记录了数据库已参与的记录。 如果这是此 WebSphere MQ 连接上的第一个 ax_reg 调用，那么队列管理器代理进程将调用 xa_open。

数据库客户机代码在进程中运行时 (例如，在 SQL UPDATE 调用期间或数据库的客户机 API 中的任何调用) 发出此 ax_reg 调用。

错误条件

在 XA 动态注册中，可能会在队列管理器中发生混淆故障。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

一个常见示例是，如果在启动队列管理器之前忘记正确设置数据库环境变量，那么队列管理器对 xa_open 的调用将失败。 不能使用全局工作单元。

要避免此情况，请确保在启动队列管理器之前设置了相关环境变量。 查看数据库产品的文档以及 第 42 页的『配置 Db2』， 第 44 页的『配置 Oracle』和 第 48 页的『Sybase 配置』中提供的建议。

对于所有数据库产品，队列管理器在队列管理器启动时调用 xa_open 一次，作为恢复会话的一部分 (如 第 51 页的『丢失与 XA 资源管理器的联系时的注意事项』中所述)。 如果未正确设置数据库环境变量，但这不会导致队列管理器无法启动，那么此 xa_open 调用将失败。 这是因为数据库客户机库使用相同的 xa_open 错误代码来指示数据库服务器不可用。 WebSphere MQ 不会将此视为严重错误，因为队列管理器必须能够开始在涉及该数据库的全局工作单元之外继续处理数据。

在 WebSphere MQ 连接上的第一个 MQBEGIN 期间 (如果未使用动态注册) 或在数据库客户机代码调用 WebSphere MQ 提供的 ax_reg 函数期间 (如果正在使用动态注册) ，将从队列管理器进行后续调用 xa_open 。

任何错误情况 (有时是 FFST 报告) 的 **计时** 取决于您是否正在使用动态注册:

- 如果您正在使用动态注册，那么 MQBEGIN 调用可能会成功，但 SQL UPDATE (或类似) 数据库调用将失败。
- 如果您未使用动态注册，那么 MQBEGIN 调用将失败。

确保在应用程序和队列管理器进程中正确设置环境变量。

汇总 *XA* 调用
以下是由于控制全局工作单元的各种 MQI 调用而对数据库客户机库中的 XA 函数进行的调用的列表。 这不是 XA 规范中描述的协议的完整描述; 它作为简要概述提供。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

请注意， `xa_start` 和 `xa_end` 调用始终由应用程序进程中的 WebSphere MQ 代码调用，而 `xa_prepare`， `xa_commit` 和 `xa_rollback` 始终从队列管理器代理进程 amqzlaa0 调用。

此表中显示的 `xa_open` 和 `xa_close` 调用都是从应用程序进程进行的。 队列管理器代理进程在 第 55 页的『错误条件』中描述的情况下调用 `xa_open` 。

表 *7: XA* 函数调用摘要

MQI 调用	**使用动态注册进行的 XA 调用**	**在未进行动态注册的情况下进行的 XA 调用**
第一个 MQBEGIN	`xa_open`	`xa_open` `xa_start`
后续 MQBEGIN	无 XA 调用	`xa_start`

MQCMIT (在当前全局工作单元期间调用**而不调用** `ax_reg`)	无 XA 调用	`xa_end`
		`xa_prepare`
		`xa_commit`
		`xa_rollback`

MQCMIT (，**在当前全局工作单元期间调用** `ax_reg`)	`xa_end`
	`xa_prepare`
	`xa_commit`
	`xa_rollback`
		不适用。 不会以非动态方式调用 `ax_reg`。

MQBACK (在当前全局工作单元期间调用**而不调用** `ax_reg`)	无 XA 调用	`xa_end` `xa_rollback`
MQBACK (，**在当前全局工作单元期间调用** `ax_reg`)	`xa_end` `xa_rollback`	不适用。 不会以非动态方式调用 `ax_reg`。
MQDISC ，其中首先调用了 MQCMIT 或 MQBACK。 如果不是，那么首先在 MQDISC 期间执行 MQCMIT 处理。	`xa_close`	`xa_close`

注意:

1. 对于 MQCMIT ，如果 `xa_prepare` 成功，那么将调用 `xa_commit` 。 否则，将调用 `xa_rollback` 。

方案 2: 其他软件提供协调

在方案 2 中，外部事务管理器协调全局工作单元，在事务管理器的 API 控制下启动并落实这些工作单元。 MQBEGIN、MQCMIT 和 MQBACK 动词均不可用。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

本部分描述了此场景，包括:

- 第 57 页的『外部同步点协调』
- 第 59 页的『使用 CICS』
- 第 63 页的『使用 Microsoft Transaction Server (COM +)』

HP Integrity NonStop Server 的 IBM WebSphere MQ 客户机可以使用 HP NonStop 事务管理设施 (TMF) 来协调全局工作单元。 有关更多信息，请参阅 使用 HP NonStop TMF。

外部同步点协调

全局工作单元还可以由符合 X/Open XA 的外部事务管理器进行协调。 此处， WebSphere MQ 队列管理器参与但不协调工作单元。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

由外部事务管理器协调的全局工作单元中的控制流如下所示:

1. 应用程序告知外部同步点协调程序 (例如， TXSeries) 要启动事务。
2. 同步点协调程序告知已知资源管理器 (例如 WebSphere MQ) 有关当前事务的信息。
3. 应用程序向与当前事务关联的资源管理器发出调用。 例如，应用程序可以向 WebSphere MQ 发出 MQGET 调用。
4. 应用程序向外部同步点协调程序发出落实或回退请求。
5. 同步点协调程序通过向每个资源管理器 (通常使用两阶段落实协议) 发出相应的调用来完成事务。

可以为 WebSphere MQ 参与的事务提供两阶段落实过程的受支持的外部同步点协调程序级别在 IBM WebSphere MQ 详细系统需求中定义。

本节的其余部分描述了如何启用外部工作单元。

IBM WebSphere MQ XA 开关结构

参与外部协调工作单元的每个资源管理器都必须提供 XA 切换结构。 此结构定义资源管理器的功能以及要由同步点协调程序调用的功能。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

IBM WebSphere MQ 提供了此结构的两个版本:

- 用于静态 XA 资源管理的 *MQRMIXASwitch*
- *MQRMIXASwitchDynamic* 用于动态 XA 资源管理

请参阅事务管理器文档以确定是使用静态还是动态资源管理接口。 只要事务管理器支持，我们就建议您使用动态 XA 资源管理。

某些 64 位事务管理器将 XA 规范中的 *long* 类型视为 64 位，而某些事务管理器将其视为 32 位。 WebSphere MQ 支持两种模型:

- 如果事务管理器为 32 位，或者事务管理器为 64 位，但将 *long* 类型视为 32 位，请使用 第 58 页的表 8 中列出的交换机装入文件。
- 如果事务管理器是 64 位，并且将 *long* 类型视为 64 位，请使用 第 58 页的表 9 中列出的交换机装入文件。

第 58 页的表 10 中提供了将 *long* 类型视为 64 位的已知 64 位事务管理器的列表。 如果您不确定事务管理器使用的模型，请查阅事务管理器文档。

表 *8: XA* 切换装入文件名		
平台	**交换机装入文件名 (服务器)**	**交换机装入文件名 (扩展事务客户机)**
Windows	*mqmxa.dll*	*mqcxa.dll*
AIX (非线程)	*libmqmxa.a*	*libmqcxa.a*
AIX (线程化)	*libmqmxa_r.a*	*libmqcxa_r.a*
HP-UX (非线程)	*libmqmxa.so*	*libmqcxa.so*
HP-UX (线程化)	*libmqmxa_r.so*	*libmqcxa_r.so*
Linux (非线程)	*libmqmxa.so*	*libmqcxa.so*
Linux (线程化)	*libmqmxa_r.so*	*libmqcxa_r.so*
Solaris	*libmqmxa.so*	*libmqcxa.so*

表 *9:* 备用 *64* 位 *XA* 切换装入文件名		
平台	**交换机装入文件名 (服务器)**	**交换机装入文件名 (扩展事务客户机)**
AIX (非线程)	*libmqmxa64.a*	*libmqcxa64.a*
AIX (线程化)	*libmqmxa64_r.a*	*libmqcxa64_r.a*
HP-UX (非线程)	*libmqmxa64.so*	*libmqcxa64.so*
HP-UX (线程化)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Linux (非线程)	*libmqmxa64.so*	*libmqcxa64.so*
Linux (线程化)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Solaris	*libmqmxa64.so*	*libmqcxa64.so*

表 *10:* 需要备用 *64* 位交换机装入文件的 *64* 位事务管理器
事务管理器
Tuxedo

某些外部同步点协调程序 (而不是 CICS) 要求参与工作单元的每个资源管理器在 XA 开关结构的名称字段中提供其名称。 WebSphere MQ 资源管理器名称为 MQSeries_XA_RMI。

同步点协调程序定义 WebSphere MQ XA 切换结构如何链接到它。 第 59 页的『使用 CICS』中提供了有关将 WebSphere MQ XA 切换结构与 CICS 链接的信息。 有关将 WebSphere MQ XA 切换结构与其他符合 XA 的同步点协调程序链接的信息，请参阅随这些产品提供的文档。

以下注意事项适用于将 WebSphere MQ 与所有符合 XA 的同步点协调程序配合使用:

- 同步点协调程序在任何 xa_open 调用上传递的 xa_info 结构包含 WebSphere MQ 队列管理器的名称。 该名称采用与传递到 MQCONN 调用的队列管理器名称相同的格式。 如果在 xa_open 调用上传递的名称为空，那么将使用缺省队列管理器。

  或者， xa_info 结构可以包含 *TPM* 和 *AXLIB* 参数的值。 *TPM* 参数指定正在使用的事务管理器。 有效值为 CICS， TUXEDO 和 ENCINA。 *AXLIB* 参数指定包含事务管理器的 ax_reg 和 ax_unreg 函数的库的名称。 有关这些参数的更多信息，请参阅 配置扩展事务客户机。 如果 xa_info 结构包含其中任一参数，那么除非正在使用缺省队列管理器，否则将在 *QMNAME* 参数中指定队列管理器名称。

- 一次只能有一个队列管理器参与由外部同步点协调程序实例协调的事务。 同步点协调程序有效连接到队列管理器，并且遵循一次仅支持一个连接的规则。
- 所有包含对外部同步点协调程序的调用的应用程序都只能连接到参与由外部协调程序管理的事务的队列管理器 (因为它们已有效地连接到该队列管理器)。 但是，此类应用程序必须发出 MQCONN 调用以获取连接句柄，并在它们退出之前发出 MQDISC 调用。
- 具有由外部同步点协调程序协调的资源更新的队列管理器必须在外部同步点协调程序之前启动。 同样，同步点协调程序必须在队列管理器之前结束。
- 如果外部同步点协调程序异常终止，请停止并重新启动队列管理器 **前** ，重新启动同步点协调程序，以确保正确解决在发生故障时未落实的任何消息传递操作。

使用 *CICS*

CICS 是 TXSeries 的其中一个元素。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

在以下位置定义了符合 XA 的 TXSeries 版本 (并使用两阶段落实过程): IBM WebSphere MQ 详细系统需求

WebSphere MQ 还支持其他事务管理器。 请参阅 IBM WebSphere MQ 详细系统需求 以获取受支持软件的当前列表。

两阶段落实过程的需求

将 CICS 两阶段落实过程与 WebSphere MQ 配合使用时，两阶段落实过程的需求。 这些要求不适用于 z/OS。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

请注意以下需求:

- WebSphere MQ 和 CICS 必须位于同一物理机器上。
- WebSphere MQ 在 WebSphere MQ MQI 客户机上不支持 CICS 。
- 在尝试启动 CICS 之前，必须先启动队列管理器，并在 XAD 资源定义节中指定其名称， 。 如果将 WebSphere MQ 的 XAD 资源定义节添加到 CICS 区域，那么执行此操作失败将阻止您启动 CICS 。
- 一次只能从单个 CICS 区域访问一个 WebSphere MQ 队列管理器。
- CICS 事务必须先发出 MQCONN 请求，然后才能访问 WebSphere MQ 资源。 MQCONN 调用必须指定在 CICS 区域的 XAD 资源定义节的 XAOpen 条目上指定的 WebSphere MQ 队列管理器的名称。 如果此条目为空，那么 MQCONN 请求必须指定缺省队列管理器。
- 访问 WebSphere MQ 资源的 CICS 事务必须在返回到 CICS 之前从该事务发出 MQDISC 调用。 未能执行此操作可能意味着 CICS 应用程序服务器仍处于连接状态，从而使队列处于打开状态。 此外，如果未安装任务终止出口 (请参阅 第 62 页的『样本任务终止出口』) ，那么 CICS 应用程序服务器可能会在稍后异常结束 (可能在后续事务期间)。
- 必须确保 CICS 用户标识 (cics) 是 mqm 组的成员，以便 CICS 代码有权调用 WebSphere MQ。

  对于在 CICS 环境中运行的事务，队列管理器会调整其授权方法并确定上下文，如下所示:

  – 队列管理器查询用于运行 CICS 事务的用户标识。 这是对象权限管理器检查的用户标识，用于上下文信息。
  – 在消息上下文中，应用程序类型为 MQAT_CICS。
  – 将从 CICS 事务名称复制上下文中的应用程序名称。

常规 *XA* 支持

常规 XA 在 IBM i 上不受支持。 提供了 XA 切换装入模块，使您能够将 CICS 与 UNIX and Linux 系统上的 WebSphere MQ 链接。 此外，还提供了样本源代码文件，使您能够为其他事务消息开发 XA 交换机。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

提供的交换机装入模块的名称为:

表 *11: CICS* 应用程序的基本代码 *:XA* 初始化例程	
C（源代码）	**C (exec)-将下列其中一项添加到您的 XAD.Stanza**
amqzscix.c	amqzsc - TXSeries 针对 AIXV 5.1 amqzsc - TXSeries for HP-UXV 5.1 amqzsc - TXSeries for Sun Solaris V 5.1
amqzscin.c	mqmc4swi - TXSeries 针对 WindowsV 5.1

构建用于 *TXSeries for Multiplatforms* 的库
构建用于 TXSeries for Multiplatforms 的库时，请使用此信息。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

预构建的交换机装入文件 是可与 CICS 程序配合使用的共享库 (在 Windows 系统上称为 *DLL*)，这些程序需要使用 XA 协议的 2 阶段落实事务。 这些预先构建的库的名称位于表 CICS 应用程序的基本代码 :XA 初始化例程中。 以下目录中还提供了样本源代码:

表 *12: Windows* 上的安装目录， *UNIX and Linux* 操作系统		
平台	**目录**	**源文件**
UNIX and Linux	*MQ_INSTALLATION_PATH*/samp/	amqzscix.c
Windows	*MQ_INSTALLATION_PATH*\Tools\c \ 样本	amqzscin.c

其中，*MQ_INSTALLATION_PATH* 是 IBM WebSphere MQ 的安装目录。

要从样本源构建交换机装入文件，请遵循适合于您的操作系统的指示信息:

AIX
发出以下命令:

```
export MQM_HOME=/usr/mqm
echo "amqzscix" > tmp.exp
xlc_r $MQM_HOME/samp/amqzscix.c -I/usr/lpp/cics/include -I$MQM_HOME/inc -e amqzscix -bE:tmp.exp -bM:SRE
-o amqzsc /usr/lpp/cics/lib/regxa_swxa.o -L$MQM_HOME/lib -L/usr/lpp/cics/lib -lcicsrt -lEncina
-lEncServer -lpthreads -lsarpc -lmqmcics_r -lmqmxa_r -lmqzi_r -lmqmcs_r
rm tmp.exp
```

Solaris
发出以下命令:

```
/opt/SUNWspro/bin/cc -s -l/opt/encina/include amqzscix.c -G -o amqzscix  -e
    CICS_XA_Init -LMQ_INSTALLATION_PATH/lib -L/opt/encina/lib
    -L/opt/dcelocal/lib /opt/cics/lib/reqxa_swxa.o
    -lmqmcics -lmqmxa -lmqzi -lmqmcs -lmqmzse -lcicsrt -lEncina -lEncSfs -ldce
```

HP-UX
发出以下命令:

```
cc -c -s -I/opt/encina/include MQ_INSTALLATION_PATH/samp/amqzscix.c -Aa +z -o  amqzscix.o ld -b
    -o amqzscix amqzscix.o /opt/cics/lib/regxa_swxa.o +e CICS_XA_Init \
    -LMQ_INSTALLATION_PATH/lib -L/opt/encina/lib -L/opt/cics/lib
    -lmqmxa_r -lmqzi_r -lmqmcs_r -lmqmzse -ldbm -lc -lm
```

Linux 平台

发出以下命令:

```
gcc -m32 -shared -fPIC -o amqzscix amqzscix.c
    \-IMQ_INSTALLATION_PATH/inc -I CICS_INSTALLATION_PATH/include
    \-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
    \-Wl,-rpath=/usr/lib -Wl,-rpath-link,/usr/lib -Wl,--no-undefined
    -Wl,--allow-shlib-undefined \-L CICS_LIB_PATH/regxa_swxa.o \-lpthread -ldl -lc
    -shared -lmqzi_r -lmqmxa_r -lmqmcics_r -ldl -lc
```

Windows

请按照以下步骤操作:

1. 使用 cl 命令通过至少编译以下变量来构建 amqzscin.obj :

   ```
   cl.exe -c -IEncinaPath\include -IMQ_INSTALLATION_PATH\include -Gz -LD amqzscin.c
   ```

2. 创建名为 `mqmc1415.def` 的模块定义文件，其中包含以下行:

   ```
   LIBRARY MQMC4SWI
   EXPORTS
   CICS_XA_Init
   ```

3. 使用 **lib** 命令通过至少使用以下选项来构建导出文件和导入库:

   ```
   lib -def:mqmc4swi.def -out:mqmc4swi.lib
   ```

   如果 lib 命令成功，那么还会构建 `mqmc4swi.exp` 文件。

4. 使用链接命令通过至少使用以下选项来构建 `mqmc4swi.dll` :

   ```
   link.exe -dll -nod -out:mqmc4swi.dll
       amqzscin.obj CicsPath\lib\regxa_swxa.obj
       mqmc4swi.exp mqmcics4.lib
      CicsPath\lib\libcicsrt.lib
      DcePath\lib\libdce.lib DcePath\lib\pthreads.lib
      EncinaPath\lib\libEncina.lib
      EncinaPath\lib\libEncServer.lib
      msvcrt.lib kernel32.lib
   ```

IBM WebSphere MQ XA 支持和 *Tuxedo*

IBM WebSphere MQ 在 Windows 上，UNIX and Linux 系统可以在 xa_start 中无限期地阻止 Tuxedo 协调的 XA 应用程序。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

仅当 Tuxedo 在单个全局事务中协调的两个或多个进程尝试使用同一事务分支标识 (XID) 访问 IBM WebSphere MQ 时，才会发生此情况。 如果 Tuxedo 为全局事务中的每个进程提供不同的 XID 以用于 IBM WebSphere MQ，那么不会发生此情况。

为避免此问题，请在 Tuxedo 服务器组中的单个全局事务标识 (gtrid) 下配置 Tuxedo 中访问 IBM WebSphere MQ 的每个应用程序。 同一服务器组中的进程在代表单个 gtrid 访问资源管理器时使用相同的 XID ，因此容易在 IBM WebSphere MQ 中的 xa_start 中受到阻止。 不同服务器组中的进程在访问资源管理器时使用单独的 XID ，因此不必在 IBM WebSphere MQ 中序列化其事务工作。

启用 *CICS* 两阶段落实过程

要使 CICS 能够使用两阶段落实过程来协调包含 MQI 调用的事务，请将 CICS XAD 资源定义节条目添加到 CICS 区域。 请注意，本主题不适用于 z/OS。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

以下是为 WebSphere MQ for Windows 添加 XAD 节条目的示例，其中 <Drive> 是安装了 WebSphere MQ 的驱动器 (例如 D:)。

```
cicsadd -cxad -r<cics_region> \
     ResourceDescription="MQM XA Product Description" \
     SwitchLoadFile="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc4swi.dll" \
     XAOpen=<queue_manager_name>
```

对于扩展事务客户机，请使用切换装入文件 mqcc4swi.dll。

以下是为 WebSphere MQ for UNIX and Linux 系统添加 XAD 节条目的示例，其中 *MQ_INSTALLATION_PATH* 表示安装 WebSphere MQ 的高级目录:

```
cicsadd -cxad -r<cics_region> \
      ResourceDescription="MQM XA Product Description" \
      SwitchLoadFile="MQ_INSTALLATION_PATH/lib/amqzsc" \
      XAOpen=<queue_manager_name>
```

对于扩展事务客户机，请使用切换装入文件 amqczsc。

有关使用 **cicsadd** 命令的信息，请参阅适用于您平台的 *CICS Administration Reference* 或 *CICS Administration Guide* 。

对 WebSphere MQ 的调用可以包含在 CICS 事务中，并且将根据 CICS 的指示落实或回滚 WebSphere MQ 资源。 此支持不可用于客户机应用程序。

必须 从 CICS 事务发出 MQCONN ， 以便访问 WebSphere MQ 资源，然后在退出时发出相应的 MQDISC 。

启用 *CICS* 用户出口

CICS 用户出口 点 (通常称为 用户出口) 是 CICS 模块中的一个位置，在此位置， CICS 可以将控制权转移到您已编写的程序 (用户出口 程序) ，在此位置， CICS 可以在出口程序完成其工作后恢复控制权。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

在使用 CICS 用户出口之前，请阅读适用于您的平台的 *CICS* 管理指南 。

样本任务终止出口

WebSphere MQ 为 CICS 任务终止出口提供样本源代码。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

样本源代码位于以下目录中:

表 *13: CICS* 任务终止出口

平台	**目录**	**源文件**
UNIX and Linux 系统	*MQ_INSTALLATION_PATH*/samp	amqzscgx.c
Windows	*MQ_INSTALLATION_PATH*\Tools \c \ 样本	amqzscgn.c

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

样本任务终止出口的构建指示信息包含在每个源文件顶部附近的注释中。

此出口由 CICS 在正常和异常任务终止时 (在执行任何同步点之后) 调用。 在出口程序中不允许可恢复的工作。

这些功能仅在 CICS 版本支持 XA 接口的 WebSphere MQ 和 CICS 上下文中使用。 CICS 将这些库称为 “程序” 或 “用户出口”。

CICS 具有多个用户出口，并且 amqzscgx (如果使用) 在 CICS 上定义并启用为 “任务终止用户出口 (UE014015)”，即出口号 15。

当 CICS 调用任务终止出口时， CICS 已通知 WebSphere MQ 任务的终止状态，并且 WebSphere MQ 已执行相应的操作 (落实或回滚)。 出口执行的所有操作都是发出 MQDISC 以进行清除。

安装和配置 CICS 系统以使用任务终止出口的一个目的是保护系统免受发生故障的应用程序代码的某些后果的影响。 例如，如果 CICS 事务在未首先调用 MQDISC 的情况下异常结束，并且未安装任务终止出口，那么您可能会看到 (在大约 10 秒内) CICS 区域的后续不可恢复故障。 这是因为不会发布在 cicsas 进程中运行的 WebSphere MQ 的运行状况线程，也不会为其提供清除和返回的时间。 症状可能是 cicsas 进程立即结束，已将 FFST 报告写入 /var/mqm/errors 或 Windows 上的等效位置。

使用 *Microsoft Transaction Server (COM +)*

COM + (Microsoft Transaction Server) 旨在帮助用户在典型的中间层服务器中运行业务逻辑应用程序。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

请参阅 在 Windows 上只能与主安装配合使用的功能部件 以获取重要信息。

COM + 将工作划分为 活动，这些活动通常是业务逻辑的简短独立块，例如 将资金从帐户 *A* 转移到帐户 *B*。 COM + 在很大程度上依赖于对象方向，尤其是 COM; 松散地， COM + 活动由 COM (业务) 对象表示。

COM + 是操作系统的集成部分。 要在 Windows 2000 和 Windows XP 上使用 COM + ， 您需要 Hotfix Q313582 (也称为 COM + Rollup Package 19.1)。

COM + 为业务对象管理员提供三种服务，消除了业务对象程序员的许多烦恼:

- 事务管理
- 安全性
- 资源池

通常，将 COM + 与前端代码 (对于 COM + 中保存的对象) 和后端服务 (例如数据库) 配合使用，并在 COM + 业务对象与后端之间进行 WebSphere MQ 桥接。

前端代码可以是独立程序，也可以是由 Microsoft Internet Information Server (IIS) 托管的活动服务器页面 (ASP)。 前端代码可以与 COM + 及其业务对象在同一台计算机上，通过 COM 进行连接。 或者，前端代码可以在不同的计算机上，通过 DCOM 进行连接。 您可以使用不同的客户机在不同的情况下访问同一个 COM + 业务对象。

后端代码可以与 COM + 及其业务对象位于同一台计算机上，也可以位于通过任何 WebSphere MQ 支持的协议进行连接的另一台计算机上。

即将到期的全球工作单元

可以将队列管理器配置为在预先配置的不活动时间间隔后使全局工作单元到期。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

要启用此行为，请设置以下环境变量:

- *AMQ_TRANSACTION_EXPIRY_RESCAN*=<rescan interval in milliseconds>
- *AMQ_XA_TRANSACTION_*到期=<timeout interval in milliseconds>

注意: 环境变量仅影响 XA 规范的表 6-4 中处于 空闲 状态的事务。 即，未在任何应用程序线程上关联但外部事务管理器软件尚未调用 **xa_prepare** 函数调用的事务。

外部事务管理器仅保留已准备，已落实或已回滚的事务的日志。 如果外部事务管理器由于任何原因而关闭，那么在返回时，它会将已准备，已落实和已回滚的事务驱动至完成，但尚未准备的任何活动事务将成为孤立事务。 要避免此情况，请设置 *AMQ_XA_TRANSACTION_*到期 ，以允许应用程序执行 MQI 事务性 API 调用与完成事务之间的预期时间间隔，在其他资源管理器上执行事务性工作。

要确保在 *AMQ_XA_TRANSACTION_* 到期 到期后及时清除，请将 *AMQ_TRANSACTION_EXPIRY_RESCAN* 值设置为低于 *AMQ_XA_TRANSACTION_* 到期 时间间隔的值，以便在 *AMQ_XA_TRANSACTION_* 到期 时间间隔内多次发生重新扫描。

恢复单元处置

WebSphere MQ for z/OS 提供了恢复单元处置。 此功能部件允许您配置在连接到同一队列共享组 (QSG) 中的另一个队列管理器时，是否可以驱动 2 阶段落实事务的第二阶段 (例如，在恢复期间)。

注: 此主题在 IBM MQ Version 8.0 和更高版本中也可用。 但是，您无法使用 “更改版本” 列表框切换到更高版本。 要转至更高版本中的主题，请在浏览器的 URL 框中编辑版本号。

WebSphere MQ for z/OS V7.0.1 和更高版本支持恢复单元处置。

恢复单元处置
恢复处置单元与应用程序的连接以及随后启动的任何事务相关。 有两个可能的恢复单元。

- GROUP 恢复单元处置标识事务应用程序在逻辑上已连接到队列共享组，并且与任何特定队列管理器都没有亲缘关系。 在连接到 QSG 中的任何队列管理器时，可以查询并解决它启动的任何 2 阶段落实事务，这些事务已完成落实进程的 phase-1 (即，它们处于不确定状态)。 在恢复方案中，这意味着事务协调程序不必重新连接到可能不可用的同一队列管理器。
- QMGR 恢复处置单元标识应用程序与它所连接的队列管理器具有直接亲缘关系，并且它启动的任何事务也具有此处置。

  在恢复方案中，事务协调程序必须重新连接到同一队列管理器以查询和解决任何不确定事务，而不考虑该队列管理器是否属于队列共享组。

决定要使用的编程语言

使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

IBM WebSphere MQ 提供了对以下编程过程语言的支持:

- C
- Visual Basic (仅限 Windows 系统)
- COBOL

这些语言使用消息队列接口 (MQI) 来访问消息队列服务。 有关对这些语言的支持的更多信息，请参阅 第 64 页的『将过程语言与 WebSphere MQ 配合使用』。

IBM WebSphere MQ 提供对以下内容的支持:

- .NET
- ActiveX
- C++
- Java
- JMS

这些语言使用 IBM WebSphere MQ 对象模型，该模型提供的类提供与 WebSphere MQ 调用和结构相同的功能，但这是在面向对象的环境中更自然的编程方式。 使用 IBM WebSphere MQ 对象模型的某些语言提供了消息队列接口 (MQI) 中不可用的其他函数。 有关对这些语言的支持的更多信息，请参阅 第 65 页的『使用 WebSphere MQ 进行面向对象的编程』。

将过程语言与 WebSphere MQ 配合使用

有关如何使用所选语言编写应用程序的详细信息，请参阅以下链接:

- 第 68 页的『采用 C 进行编码』
- 第 71 页的『采用 Visual Basic 进行编码』
- 第 70 页的『采用 COBOL 进行编码』

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。

WebSphere MQ 提供数据定义文件以帮助您编写应用程序。 有关完整描述，请参阅 第 66 页的『IBM WebSphere MQ 数据定义文件』。

如果可以选择要对程序进行编码的语言，请考虑程序将处理的消息的最大长度。 如果程序将仅处理已知最大长度的消息，那么可以使用任何受支持的编程语言对其进行编码。 但是，如果您不知道程序将必须处理的消息的最大长度，那么您选择的语言将取决于您是编写 CICS， IMS 还是批处理应用程序:

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
以 CICS 支持的任何语言对程序进行编码。 EXEC CICS 接口提供了用于管理内存的调用 (如果需要)。

使用 WebSphere MQ 进行面向对象的编程

使用 IBM WebSphere MQ 对象模型的某些语言和编程框架提供了消息队列接口 (MQI) 中不可用的其他函数。 有关 IBM WebSphere MQ 对象模型提供的类，方法和属性的详细信息，请参阅 第 72 页的『IBM WebSphere MQ 对象模型』。

.NET
请参阅 使用 .NET，以获取有关使用 WebSphere MQ .NET 类对 .NET 程序进行编码的信息。 Message Service Clients for C/C++ and .NET 提供名为 XMS 的应用程序编程接口 (API)，该接口具有与 Java 消息服务 (JMS) 相同的接口集 API。

C++
IBM WebSphere MQ 提供了相当于 WebSphere MQ 对象的 C++ 类以及相当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。 请参阅 使用 C++，以获取有关使用 C + + 中的 WebSphere MQ 对象模型对程序进行编码的信息。 Message Service Clients for C/C++ and .NET 提供了称为 XMS 的应用程序编程接口 (API)，该接口与 Java 消息服务 (JMS) 具有相同的接口集 API。

Java
请参阅 使用 Java，以获取有关使用 Java 中的 WebSphere MQ 对象模型对程序进行编码的信息。 有关 IBM WebSphere MQ classes for Java 和 IBM WebSphere MQ 类之间的差异的信息，请参阅 第 73 页的『我应该使用 IBM WebSphere MQ classes for Java 还是 IBM WebSphere MQ classes for JMS?』，以帮助您决定要使用的类。

JMS
Websphere MQ 还提供了实现 Java 消息服务 (JMS) 规范的类。 有关用于 JMS 的 Websphere MQ 类的详细信息，请参阅 使用 Java。 有关 IBM WebSphere MQ classes for Java 与 IBM WebSphere MQ 类之间的差异的信息，请参阅 第 73 页的『我应该使用 IBM WebSphere MQ classes for Java 还是 IBM WebSphere MQ classes for JMS?』，以帮助您决定要使用的类。

Message Service Clients for C/C++ and .NET 提供名为 XMS 的应用程序编程接口 (API)，该接口具有与 Java 消息服务 (JMS) 相同的接口集 API。

ActiveX
WebSphere MQ ActiveX 通常称为 MQAX。 MQAX 作为 WebSphere MQ 的一部分包含在 窗口中。 ActiveX 支持已稳定在 WebSphere MQ 版本 6.0 级别。 要利用引入到 WebSphere MQ 版本更高 6.0 的功能部件，请考虑改为使用 .NET。 有关使用 ActiveX 中的 WebSphere MQ 对象模型对程序进行编码的信息，请参阅 使用组件对象模型接口 (WebSphere MQ Automation Classes for ActiveX)。

相关概念
技术概述
第 7 页的『开发应用程序』
IBM WebSphere MQ 提供了多种方法，您可以通过这些方法开发应用程序来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。

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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

相关参考
应用程序开发参考

IBM WebSphere MQ 数据定义文件

IBM WebSphere 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

WebSphere MQ COPY，头，包含和模块文件中描述了用于帮助您编写通道出口的数据定义文件。

第 311 页的『用户出口， API 出口和 WebSphere 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++。

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 上安装库的高级限定符。

本主题在以下标题下介绍 WebSphere MQ 数据定义文件:

- 第 66 页的『C 语言包含文件』
- 第 67 页的『Visual Basic 模块文件』
- 第 67 页的『COBOL 副本文件』

C 语言包含文件

WebSphere MQ C 包含文件列示在 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
UNIX 平台	*MQ_INSTALLATION_PATH*/inc/
Windows 系统	*MQ_INSTALLATION_PATH*\Tools\c\include

其中 *MQ_INSTALLATION_PATH* 表示安装 WebSphere MQ 的高级目录。

注: 对于 UNIX 平台，包含文件以符号方式链接到 `/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

WebSphere MQ for Windows 提供了四个 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， WebSphere MQ 提供了包含指定常量的单独副本文件，以及每个结构的两个副本文件。

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。

WebSphere MQ COBOL 副本文件列示在 COBOL COPY 文件中。 它们安装在以下目录中：

平台	安装目录或库
其他 UNIX 平台	*MQ_INSTALLATION_PATH*/inc/
Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (针对 Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (针对 IBM VisualAge COBOL)

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 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。

采用 C 进行编码

在 C 中对 WebSphere MQ 程序进行编码时，请注意以下部分中的信息。

- 第 68 页的『MQI 调用的参数』
- 第 68 页的『具有未定义数据类型的参数』
- 第 68 页的『数据类型』
- 第 68 页的『处理二进制字符串』
- 第 69 页的『处理字符串』
- 第 69 页的『结构的初始值』
- 第 69 页的『动态结构的初始值』
- 第 70 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
```

采用 COBOL 进行编码

在 COBOL 中对 WebSphere 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 WebSphere 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 WebSphere MQ 结构放在一起，以便其全都边界对齐。

采用 pTAL 进行编码

采用 pTAL 编写 IBM WebSphere MQ 程序代码时，请注意以下部分中的信息。

HP Integrity NonStop Server

定义并初始化 IBM WebSphere MQ 结构

为 IBM WebSphere MQ 结构的 pTAL 结构定义提供以 ^DEF 结尾的名称。 例如，将对以下 pTAL 声明进行编码以创建 IBM WebSphere MQ 消息描述符 (MQMD) 和 IBM WebSphere MQ 放置消息选项 (MQPMO) 结构。

```
STRUCT MYMD(MQMD^DEF);        ! Declare an MQMD structure
STRUCT MYPMO(MQPMO^DEF);      ! Declare an MQPMO structure
```

IBM WebSphere MQ 为 pTAL DEFINE 提供以 ^DEFAULT 结尾的名称，以使用缺省值初始化 IBM WebSphere MQ 结构。 将对以下 pTAL 语句进行编码以向声明的 MQMD 和 MQPMO 结构分配缺省值：

```
MQMD^DEFAULT(MYMD);           ! Assign default values to an MQMD structure
MQPMO^DEFAULT(MYPMO);         ! Assign default values to an MQPMO structure
```

您可以使用类似的代码来声明和初始化其他 IBM WebSphere MQ 结构。

pTAL 和 CRE

pTAL 程序无法初始化公共运行时环境，因此它们必须与 C 语言或 COBOL 主例程一起使用。

IBM WebSphere MQ 随附的 pTAL 样本使用名为 AMQSPTM0.C 的 C 语言主线例程

具有 MQCHAR 数据类型的参数

MQGET， MQPUT 和 MQPUT1 过程各自具有具有 MQCHAR .EXT 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。

这种类型的参数以字符串数组的形式显示在 pTAL 样本中。 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。 将过程参数声明为 MQCHAR .EXT，但可以在过程调用时将任何数据的地址指定为参数。

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。

采用 Visual Basic 进行编码

在 Visual Basic 中对 WebSphere MQ 程序进行编码时，请注意以下部分中的信息。

注: 在 .NET 环境外， WebSphere MQ 中对 Visual Basic (VB) 的支持已稳定在 V6.0 级别。 添加到 WebSphere MQ 7.0 或更高版本的大多数新功能不可用于 VB 应用程序。 如果您正在 VB.NET，使用 WebSphere MQ .NET 类。 有关更多信息，请参阅使用 .NET。

Visual Basic 仅在 Windows 上受支持。

要避免在 Visual Basic 和 WebSphere MQ 之间传递二进制数据的意外转换，请使用 MQBYTE 定义而不是 MQSTRING。 CMQB.BAS 定义了几种与 C 字节定义等效的新 MQBYTE 类型，并在 WebSphere 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 WebSphere MQ 数据结构的引用是按偏移量而不是指针。 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。 WebSphere MQ Support for Visual Basic 提供 MQCONNXAny 调用以实现此目的，并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。

IBM WebSphere MQ 对象模型

IBM WebSphere MQ 对象模型由类、方法和属性组成。 使用此信息来了解其中每个概念的信息。

IBM WebSphere MQ 对象模型包括:

- 类 表示熟悉的 WebSphere 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 WebSphere MQ 对象的属性的每个类上的 属性 。

使用 WebSphere MQ 对象模型创建 WebSphere MQ 应用程序时，请在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。

以下主题详细描述了每个 WebSphere MQ 对象模型:

- 第 72 页的『类』
- 第 73 页的『对象引用』
- 第 73 页的『返回码』

类

WebSphere MQ 对象模型提供以下基本类集。

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 WebSphere MQ 程序中， WebSphere MQ 将连接句柄和对象句柄返回到程序。

这些句柄必须在后续 WebSphere MQ 调用上作为参数传递。 通过 WebSphere MQ 对象模型，这些句柄将在应用程序中隐藏。 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。

我应该使用 IBM WebSphere MQ classes for Java 还是 IBM WebSphere MQ classes for JMS?

Java 应用程序可以使用 IBM WebSphere MQ classes for Java 或 IBM WebSphere MQ classes for JMS 来访问 IBM WebSphere MQ 资源。 这些方法各有优势。

IBM WebSphere MQ Java 类封装了消息队列接口 (MQI) (本机 IBM WebSphere MQ API)，并使用与其他面向对象的接口相同的对象模型，而 Java 消息服务的 IBM WebSphere MQ 类实现了 Sun 的 Java 消息服务 (JMS) 接口。

如果您熟悉使用过程语言或面向对象的语言的 Java 以外环境中的 IBM WebSphere MQ，那么可以使用 IBM WebSphere MQ Java 类将现有知识传输到 Java 环境。 您还可以利用 IBM WebSphere MQ 的全部功能部件，但并非所有这些功能部件都在 IBM WebSphere MQ classes for JMS 中可用。

如果您不熟悉 IBM WebSphere MQ，或者已具有 JMS 经验，那么通过使用 IBM WebSphere MQ classes for JMS，您可能会发现使用熟悉的 JMS API 来访问 IBM WebSphere MQ 资源更容易。 JMS 也是 Java Platform, Enterprise Edition (Java EE) 平台的组成部分。 Java EE 应用程序可以使用消息驱动的 Bean (MDB) 异步处理消息， MDB 只能处理 JMS 消息。 JMS 也是 Java EE 与异步消息传递系统 (例如 IBM WebSphere MQ) 交互的标准机制。 符合 Java EE 的每个应用程序服务器都必须包含 JMS 提供程序，因此您可以使用 JMS 在不同的应用程序服务器之间进行通信，也可以将应用程序从一个 JMS 提供程序移植到另一个 JMS 提供程序，而无需对应用程序进行任何更改。

设计 IBM WebSphere MQ 应用程序

当您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 WebSphere MQ 提供的功能。

设计 IBM WebSphere 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 WebSphere MQ 的管理自动化。 有关更多信息，请参阅 WebSphere MQ 管理接口 (MQAI) 简介 和 自动化管理任务 。

编程语言

IBM WebSphere MQ 支持许多用于编写应用程序的过程语言和面向对象的编程语言。 有关更多信息，请参阅 第 64 页的『决定要使用的编程语言』。

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。

如果使用的是 C，请使用 ANSI 标准 C 中的代码。 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？ 要查看受支持的队列类型，请参阅 队列。

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。 有关更多信息，请参阅 队列管理器集群。

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 75 页的『设计消息』。

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 WebSphere MQ 消息中共享的信息发送到由 IBM WebSphere MQ 发布/预订管理的标准目标，并且让 IBM WebSphere 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 IBM WebSphere MQ 发布/预订消息传递简介。

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 WebSphere 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 WebSphere MQ 触发功能，请参阅第 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』）。 或者，当队列上的消息处理速度不够快 (使用 检测事件中描述的 IBM WebSphere MQ 检测事件 功能) 时，您可能想要启动应用程序的另一个实例。

在 IBM WebSphere MQ 客户机上运行应用程序

客户机环境中支持完整 MQI，这使几乎任何 IBM WebSphere MQ 应用程序都能够重新链接以在 IBM WebSphere MQ MQI 客户机上运行。 将 IBM WebSphere MQ MQI 客户机上的应用程序链接到 MQIC 库，而不是 MQI 库。

注: 在 IBM WebSphere 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 299 页的『在 IBM WebSphere MQ MQI 客户机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 WebSphere MQ 应用程序性能的建议，请参阅第 76 页的『应用程序设计和性能』。

高级 IBM WebSphere 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 WebSphere MQ 方法，例如关联回复以及生成和发送 IBM WebSphere MQ 上下文信息。 有关更多信息，请参阅第 77 页的『高级 IBM WebSphere MQ 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 WebSphere 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 271 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 WebSphere MQ 消息的 持久性 功能来确保重要消息的传递。

测试 IBM WebSphere MQ 应用程序

IBM WebSphere 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 WebSphere MQ 跟踪功能。

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 WebSphere MQ 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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

第 162 页的『编写排队应用程序』
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 293 页的『编写客户机应用程序』
在 WebSphere MQ 上编写客户机应用程序所需的知识。

第 470 页的『使用 .NET』
WebSphere MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 WebSphere MQ MQI 客户机连接到 WebSphere MQ ，或者直接连接到 WebSphere MQ 服务器。

第 529 页的『使用 C++』
WebSphere MQ 提供相当于 WebSphere MQ 对象的 C++ 类以及相当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 603 页的『使用 WebSphere MQ classes for JMS』
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) 是随 WebSphere MQ 提供的 JMS 提供程序。 除了实现 javax.jms 包中定义的接口外， WebSphere MQ JMS 类还提供了两组 JMS API 扩展。

第 552 页的『使用 WebSphere MQ classes for Java』
WebSphere MQ Java 类使您能够在 Java 环境中使用 WebSphere MQ 。 Java 应用程序可以使用 WebSphere MQ classes for Java 或 WebSphere MQ classes for JMS 来访问 WebSphere MQ 资源。

第 867 页的『使用组件对象模型接口 (WebSphere MQ Automation Classes for ActiveX)』
WebSphere MQ Automation Classes for ActiveX (MQAX) 是 ActiveX 组件，用于提供可在应用程序中用于访问 WebSphere MQ 的类。

设计消息

考虑本信息中给出的各方面以帮助设计消息。

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

队列管理器会保留所有持久消息，并在重新启动时根据需要从 WebSphere MQ 日志文件中恢复这些消息。 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 WebSphere MQ for Windows 或 WebSphere 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。

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

如果可能需要将消息放在多个队列上，那么可以使用分发列表或发布到主题。

应用程序设计和性能

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 WebSphere 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。
- 将您的消息保留在工作单元中，以便可以同时落实或回退这些消息。
- 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。

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。 可向您的应用程序添加代码以便缓冲区大小可以相应增加并重新发出 MQGET 调用。

注: 如果未显式地设置 *MaxMsgLength* 属性，它将缺省为 4 MB，这在将它用于影响应用程序的缓冲区大小时会非常不够。

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

对于 WebSphere 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。

高级 IBM WebSphere MQ 方法

对于简单 IBM WebSphere MQ 应用程序，您需要决定要在应用程序中使用的 WebSphere MQ 对象以及要使用的消息类型。 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 223 页的『等待消息』）。
- 建立要在消息到达时驱动的回调出口；请参阅 第 28 页的『IBM WebSphere 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 WebSphere MQ 应用程序中，当程序接收到请求它执行某些工作的消息时，该程序通常会向请求者发送一条或多条应答消息。

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 32 页的『消息上下文』。

自动启动 WebSphere MQ 程序

使用 WebSphere MQ 触发 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 275 页的『使用触发器启动 IBM WebSphere 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 WebSphere MQ 工具在计时器上自动启动程序。

WebSphere MQ 可以定义服务对象以在队列管理器启动时启动 WebSphere MQ 程序; 请参阅 服务对象。

生成 WebSphere 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

第 11 页的『报告消息』中有关于这些规则的描述。

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 。

样本 WebSphere MQ 程序

使用此主题集合来了解不同平台上的样本 WebSphere MQ 程序。

- 第 79 页的『分布式平台的样本程序』

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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 73 页的『设计 IBM WebSphere MQ 应用程序』
当您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 WebSphere MQ 提供的功能。

第 162 页的『编写排队应用程序』
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 293 页的『编写客户机应用程序』
在 WebSphere MQ 上编写客户机应用程序所需的知识。

第 791 页的『在 WebSphere MQ 中使用 Web Service』
您可以使用 IBM WebSphere MQ Transport for SOAP 或 IBM WebSphere MQ Bridge for HTTP 为 Web Service 开发 IBM WebSphere MQ 应用程序。

第 232 页的『编写发布/预订应用程序』
开始编写发布/预订 WebSphere MQ 应用程序。

第 357 页的『构建 IBM WebSphere MQ 应用程序』
使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

第 462 页的『处理程序错误』
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。

分布式平台的样本程序

本主题描述使用 C 和 COBOL 编写的 IBM WebSphere MQ 随附的样本程序。 这些样本演示了消息队列接口 (MQI) 的典型用法。

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。 但是，这些样本适合用作您自己的消息排队程序的基础。

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。

C++ 样本程序: 请参阅 使用 C++ 以获取 C++ 中提供的样本程序的描述。

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。 要设置队列管理器以接受客户机连接，请参阅第 88 页的『准备并运行样本程序』以获取详细信息。

使用以下链接了解有关样本程序的更多信息：

- 第 80 页的『样本程序中演示的功能』
- 第 111 页的『“发布/预订”样本程序』
- 第 115 页的『“放置”样本程序』
- 第 104 页的『“分发列表”样本程序』
- 第 94 页的『浏览样本程序』
- 第 95 页的『Browser 样本程序』
- 第 106 页的『“获取”样本程序』
- 第 116 页的『“参考消息”样本程序』
- 第 121 页的『“请求”样本程序』
- 第 110 页的『“查询”样本程序』
- 第 111 页的『“查询消息句柄属性”样本程序』
- 第 125 页的『“设置”样本程序』
- 第 105 页的『“回传”样本程序』
- 第 97 页的『“数据转换”样本程序』
- 第 128 页的『“触发”样本程序』
- 第 93 页的『Asynchronous Put 样本程序』
- 第 97 页的『“数据库协调”样本』

- 第 96 页的『CICS 事务样本』
- 第 129 页的『TUXEDO 样本』
- 第 104 页的『“死信队列处理程序”样本』
- 第 96 页的『“连接”样本程序』
- 第 92 页的『API 出口样本程序』
- 第 142 页的『在 Windows 系统上使用 SSPI 安全出口』
- 第 142 页的『使用远程队列运行样本』
- 第 143 页的『集群队列监视样本程序 (AMQSCLM)』
- 第 150 页的『连接端点查找 (CEPL) 的样本程序』

样本程序中演示的功能

一组表，用于显示 WebSphere MQ 样本程序所演示的方法。

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。

UNIX and Linux 系统的样本

本主题显示了 UNIX and Linux 系统上 WebSphere MQ 的样本程序所演示的方法。

请参阅 第 90 页的『在 UNIX 系统上准备和运行样本程序』 以了解 UNIX 和 Linux 系统上 WebSphere MQ 的样本程序的存储位置。

第 80 页的表 14 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 *14: WebSphere MQ on UNIX and Linux* 样本程序演示 *MQI (C* 和 *COBOL)* 的使用

方法	**C（源代码）(第 82 页的『1』)**	**COBOL（源代码）(第 82 页的『2』)**	**服务器（C 可执行文件）**	**客户机（C 可执行程序）(第 82 页的『3』)**
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	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
将消息放置到分发列表中 (第 82 页的『4』)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
回复请求消息	amqsinqa	amqminqx amqiinqx	amqsinq	无样本
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 82 页的『4』)	amqsprma	无样本	amqsprm	amqsprmc
从队列获取参考消息 (第 82 页的『4』)	amqsgrma	无样本	amqsgrm	amqsgrmc

表 14: *WebSphere MQ on UNIX and Linux* 样本程序演示 *MQI (C* 和 *COBOL)* 的使用 *(*继续*)*				
方法	**C（源代码）(第 82 页的『1』)**	**COBOL（源代码）(第 82 页的『2』)**	**服务器（C 可执行文件）**	**客户机（C 可执行程序）(第 82 页的『3』)**
参考消息通道出口 (第 82 页的『4』)	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 82 页的『5』)	无样本	无样本	无样本
WebSphere MQ (协调 XA 兼容的数据库管理器) 使用 SQL 访问单个数据库	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	无样本	无样本
WebSphere MQ (协调 XA 兼容的数据库管理器) 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 82 页的『6』)	amqscic0.ccs	无样本	amqscic0	无样本
Encina 事务 (第 82 页的『4』)	amqsxae0	无样本	amqsxae0	无样本
用于放置消息的 TUXEDO 事务 (第 82 页的『7』)	amqstxpx	无样本	无样本	无样本
用于获取消息的 TUXEDO 事务 (第 82 页的『7』)	amqstxgx	无样本	无样本	无样本
TUXEDO 的服务器 (第 82 页的『7』)	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 82 页的『8』)	无样本	amqsdlq	无样本

表 *14: WebSphere MQ on UNIX and Linux* 样本程序演示 *MQI (C* 和 *COBOL)* 的使用 *(*继续*)*

方法	C（源代码）(第 82 页的『1』)	COBOL（源代码）(第 82 页的『2』)	服务器（C 可执行文件）	客户机（C 可执行程序）(第 82 页的『3』)
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 SSL/TLS 连接信息	amqssslc	无样本	不适用	amqssslc

注意:

1. WebSphere MQ MQI 客户机样本的可执行版本与在服务器环境中运行的样本共享相同的源。
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，使用任一编译器编译以 "amq0" 开头的程序。
3. WebSphere MQ MQI 客户机样本的可执行版本在 WebSphere MQ for HP-UX 上不可用。
4. 仅在 WebSphere MQ for AIX， WebSphere MQ for HP-UX 和 WebSphere MQ for Solaris 上受支持。
5. 在 WebSphere MQ for AIX， WebSphere MQ for HP-UX 和 WebSphere MQ for Solaris 上，此程序称为 `amqsvfc0.c`
6. CICS 仅受 WebSphere MQ for AIX 和 WebSphere MQ for HP-UX 支持。
7. 对于 System p 上的 Linux， WebSphere MQ 不支持 TUXEDO。
8. 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 WebSphere MQ 系统需求页面 (IBM WebSphere MQ 的系统需求)。

HP Integrity NonStop Server 的 *IBM WebSphere MQ* 客户机样本

本主题显示 HP Integrity NonStop Server 系统上 IBM WebSphere MQ 客户机的样本程序所演示的方法。

第 82 页的表 15 此表列出了提供的 C， COBOL 和 pTAL 源样本程序。

表 *15: IBM WebSphere MQ on HP Integrity NonStop Server* 样本程序演示 *C*， *COBOL* 和 *pTAL* 的使用

方法	C				COBOL		pTAL	
	OSS（源）	OSS（可执行文件）	监护器（源）	监护器（可执行文件）	OSS（源）	监护器（源）	OSS（源）	监护器（源）

表 *15: IBM WebSphere MQ on HP Integrity NonStop Server* 样本程序演示 *C*， *COBOL* 和 *pTAL* 的使用 *(*继续*)*

方法	C				COBOL		pTAL	
使用发布/预订接口	amqspuba.c amqssbxa.c amqssuba.c amqspse0.c	amqspubc amqssbxc amqssubc	MQSPUBC MQSSBXC MQSSUBC	AMQSPUBC AMQSSBXC AMQSSUBC	amq0pub0.cbl amq0sub0.cbl	MQSPUBL MQSSUBL	amqtpub0.tal amqtsub0.tal	MQSPUBT MQSSUBT
使用 MQPUT 调用放置消息	amqsput0.c	amqsputc	MQSPUTC	AMQSPUTC	amq0put0.cbl	MQSPUTL	amqtput0.tal	MQSPUTT
使用 MQPUT1 调用放置单条消息	amqsecha.c	amqsechc	MQSECHC	AMQSECHC			amqtech0.tal	MQSECHT
将消息放到分发列表	amqsptl0.c	amqsptlc	MQSPTLC	AMQSPTLC	amq0ptl0.cbl	MQSPTLL		
回复请求消息	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
获取消息（无等待）	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
获取消息（具有时间限制的等待）	amqsget0.c	amqsgetc	MQSGETC	AMQSGETC	amq0get0.cbl	MQSGETL	amqtget0.tal	MQSGETT
获取消息（无限等待）	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
获取消息（具有数据转换）	amqsecha.c	amqsechc	MQSECHC	AMQSECHC				
将参考消息放入队列	amqsprma.c	amqsprmc	MQSPRMC	AMQSPRMC				
从队列获取参考消息	amqsgrma.c	amqsgrmc	MQSGRMC	AMQSGRMC				
参考消息通道出口	amqsqrma.c amqsxrma.c		MQSQRMC MQSXRMC					

表 *15: IBM WebSphere MQ on HP Integrity NonStop Server* 样本程序演示 *C*， *COBOL* 和 *pTAL* 的使用 *(*继续*)*

方法	C				COBOL		pTAL	
浏览消息的前 20 个字符	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
浏览完整消息	amqsbcg0.c	amqsbcgc	MQSBCGC	AMQSBCGC				
使用共享输入队列	amqsinqa.c	amqsinqc	MQSINQC	MQSINQC				
使用独占输入队列	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
使用 MQINQ 调用	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
使用 MQSET 调用	amqsseta.c	amqssetc	MQSSETC	AMQSSETC				
使用应答队列	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
请求消息异常	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
接受截断的消息	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
使用已解析的队列名称	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
触发进程	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
使用数据转换	amqsvfc0.c							
死信队列处理程序 (1)	目录 ./samp/dlq							
使用 MQCONNX 连接到队列管理器	amqscnxc.c	amqscnxc	MQSCNXC					
使用 API 出口	amqsaxe0.c amqsaem0.c							
集群工作负载均衡出口	amqswlm0.c		MQSWLMC					

表 *15: IBM WebSphere MQ on HP Integrity NonStop Server* 样本程序演示 *C*， *COBOL* 和 *pTAL* 的使用 *(*继续*)*

方法	C				COBOL		pTAL	
集群队列监视器	amqsclma.c							
使用 MQSTAT 调用异步放置消息并获取状态	amqsapt0.c	amqsaptc	MQSAPTC	MQSAPTC				
可重新连接的客户机	amqsghac.c amqsmhac.c amqsphac.c	amqsghac amqsmhac amqsphac	MQSGHAC MQSMHAC MQSPHAC MQSFHAC	AMQSGHAC AMQSMHAC AMQSPHAC AMQSFHAC				
使用消息使用者来异步使用来自多个队列的消息	amqscbf0.c	amqscbfc						
指定有关 MQCONNX 的 SSL/TLS 连接信息	amqsssl c.c	amqsssl c	MQSSSLC	AMQSSSLC				
活动跟踪	amqsact0.c	amqsactc	MQSACTC	AMQSACTC				
消息属性	amqsiqma.c amqsstma.c	amqsiqmc amqsstmc	MQSIQMC MQSSTMC	AMQSIQMC AMQSSTMC				
命令服务器	amqsstop.c		MQSSTOC					
日志事件	amqslog0.c	amqslogc	MQSLOGC	AMQSLOGC				
记帐	amqsmon0.c	amqsmonc	MQSMONC	AMQSMONC				
管理接口	amqsaicq.c amqsaiem.c amqsailq.c							

表 *15: IBM WebSphere MQ on HP Integrity NonStop Server* 样本程序演示 *C*， *COBOL* 和 *pTAL* 的使用 *(*继续*)*

方法	C				COBOL		pTAL	
用于调用 pTAL 的 C 语言主函数的示例			MQSPTM C					

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。
2. 有关在 HP Integrity NonStop Server 平台上为 IBM WebSphere MQ 客户机开发应用程序的信息，请参阅:
   - 第 363 页的『在 HP Integrity NonStop Server 上构建应用程序』
     - 第 365 页的『在 HP Integrity NonStop Server 中准备 C 程序』
     - 第 366 页的『准备 COBOL 程序』
     - 第 367 页的『准备 pTAL 程序』

IBM WebSphere MQ for Windows 的样本

本主题显示 IBM WebSphere MQ for Windows 的样本程序所演示的方法。

第 86 页的表 16 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 *16: IBM WebSphere MQ for Windows* 样本程序演示 *MQI (C* 和 *COBOL)* 的使用

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

表 *16: IBM WebSphere MQ for Windows* 样本程序演示 *MQI (C* 和 *COBOL)* 的使用 *(*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）
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
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
WebSphere MQ (协调 XA 兼容的数据库管理器) 使用 SQL 访问单个数据库	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	无样本	无样本
WebSphere MQ (协调 XA 兼容的数据库管理器) 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 88 页的『1』)	无样本	amqsdlq	无样本
从 WebSphere MQ MQI 客户机，放置消息	无样本	无样本	无样本	amqsputc
从 WebSphere MQ 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

表 16: *IBM WebSphere MQ for Windows* 样本程序演示 *MQI (C* 和 *COBOL)* 的使用 *(*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）**
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 SSL/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。				

IBM WebSphere MQ for Windows 的 *Visual Basic* 样本

本主题显示 IBM WebSphere MQ for Windows 的 Visual Basic 样本程序演示的方法。

第 88 页的表 17 显示了 IBM WebSphere 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 WebSphere MQ 服务器协作。 要了解如何将样本项目更改为与 IBM WebSphere MQ 客户机协作，请参阅第 390 页的『在 Windows 中准备 Visual Basic 程序』。

表 17: *IBM WebSphere MQ for Windows* 样本程序演示 *MQI* 的使用 *(Visual Basic)*	
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

准备并运行样本程序

将队列管理器配置为安全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。 确定是否已启用通道认证记录，如下所示:

```
DISPLAY QMGR CHLAUTH
```

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。 如果队列管理器未使用通道认证记录，那么可以将步骤 第 89 页的『4』 替换为备用认证方法 (例如，安全出口)，该方法将 MCAUSER 设置为您将在步骤 第 89 页的『1』 中获取的 非特权用户标识 。

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 是队列管理器的名称
   nnnn 是所选端口号

   a) 对于 UNIX 和 Windows 系统:

   ```
   runmqlsr -t tcp -m qmgr -p 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 89 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) 对于 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 89 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) 对于 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 89 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) 对于 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 89 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。

相关任务

在 UNIX 或 Linux 系统和 Windows 上授予对 WebSphere MQ 对象的访问权

相关参考

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

在 *UNIX* 系统上准备和运行样本程序

表 *18:* 在 *UNIX and Linux* 系统上查找 *WebSphere 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 表示安装 WebSphere MQ 的高级目录。	

如果在安装时使用了缺省值，那么 UNIX and Linux 系统上的 WebSphere MQ 样本文件位于 第 90 页的表 18 中列出的目录中。 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。 有关如何执行此操作的信息，请参阅第 91 页的『运行样本程序』。

在 *Windows* 系统上准备和运行样本程序

表 *19:* 在何处查找 *WebSphere MQ for Windows* 的样本

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）

表 19: 在何处查找 *WebSphere MQ for Windows* 的样本 (继续)	
内容	**目录**
样本 MQSC 文件	*MQ_INSTALLATION_PATH*\Tools\MQSC\Samples
Visual Basic 源代码	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET 样本	*MQ_INSTALLATION_PATH*\Tools\dotnet \ 样本

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

注:

1. 某些 C 可执行文件样本的 64 位版本可用。

The WebSphere MQ for 窗口 sample files are in the directories listed in 第 90 页的表 19 if the defaults were used at installation time; the installation drive defaults to <c:>. To run the samples, either use the executable versions supplied or compile the source versions as you would any other WebSphere MQ for 窗口 applications. 有关如何执行此操作的信息，请参阅第 91 页的『运行样本程序』。

运行样本程序

跨不同平台运行样本程序时，请考虑使用本主题。

请先创建队列管理器并设置缺省定义，然后才能运行任何样本程序。 这在管理中进行了说明。

在 Windows， UNIX 和 Linux 平台上

样本需要与队列集协作。 请使用自己的队列或者运行样本 MQSC 文件 amqscos0.tst 来创建集。

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。

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。

您现在可以运行样本应用程序。 输入样本应用程序的名称，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。

请参阅个别样本的描述以获取其中各样本期望的参数的信息。

队列名称的长度

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。

问询 (Inquire)、设置 (Set) 和回传 (Echo) 示例

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。

如果想要 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， UNIX and Linux 系统通过编辑 `amqscos0.tst` 文件并在使用 **runmqsc** 命令之前将 C 可执行文件名更改为 COBOL 可执行文件名来执行此操作，如前所述。

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 322 页的『编写和编译 API 出口』。

来源
amqsaxe0.c

二进制
amqsaxe

为样本出口进行配置

1. 向 qm.ini 文件中添加以下内容。

   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 WebSphere MQ 的安装目录。

   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 WebSphere MQ 的安装目录。

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: 重新连接类型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 WebSphere MQ Explorer 在 QL1 上放置测试消息。 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。 在异步使用的情况下，不需要轮询， WebSphere MQ 将执行线程和存储管理。 “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 80 页的『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 80 页的『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。

UNIX， *Linux* 和 *Windows* 系统

在 UNIX， Linux 和 Windows 系统上了解 "浏览" 样本程序时，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
```

仅显示每条消息的前 50 个字符，在此情况下，后跟 `- - - truncated` 。

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 80 页的『样本程序中演示的功能』，以了解这些程序的名称。

此程序采用以下参数:

1. 源队列的名称
2. 队列管理器的名称
3. 属性的可选参数。

此程序的前两个输入参数是必需的。 例如，通过下列其中一种方式启动程序:

- `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）

属性参数的允许的值为:

值	行为
0	缺省行为，与 V6 的行为相同。 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。

值	行为
4	指定了 MQGMO_PROPERTIES_COMPATIBILITY，因此可以返回所有属性，具体取决于是否包含 V 6 属性，否则将废弃这些属性。

程序仅限于列显消息的前 65535 个字符，如果读取更长的消息，那么将失败并显示原因已截断消息。

请参阅 管理 以获取此实用程序的输出示例。

CICS 事务样本

提供了样本 CICS 事务程序，名为 amqscic0.ccs (对于源代码) 和 amqscic0 (对于可执行版本)。 您可以使用标准 CICS 工具来构建事务。

请参阅第 357 页的『构建 IBM WebSphere MQ 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.WORKQUEUE，并将其放入本地队列中，该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]] [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

SvrconnChannelName
服务器连接通道的名称

QMgrName
目标队列管理器名称

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。 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。 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 80 页的『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

提供了两个样本，用于演示 WebSphere MQ 如何在同一工作单元中协调 WebSphere MQ 更新和数据库更新。

下面提供了这些样本：

1. AMQSXAS0 (以 C 为单位) 或 AMQ0XAS0 (以 COBOL 为单位) ，用于更新 WebSphere MQ 工作单元中的单个数据库。
2. AMQSXAG0 (在 C 中) 或 AMQ0XAG0 (在 COBOL 中) ， AMQSXAB0 (在 C 中) 或 AMQ0XAB0 (在 COBOL 中) ， AMQSXAF0 (在 C 中) 或 AMQ0XAF0 (在 COBOL 中) ，它们一起更新 WebSphere MQ 工作单元中的两个数据库，显示可以如何访问多个数据库。 提供这些样本是为了显示 MQBEGIN 调用，混合 SQL 和 WebSphere MQ 调用的使用情况以及连接到数据库的位置和时间。

第 98 页的图 18 说明了如何使用所提供的样本来更新数据库：

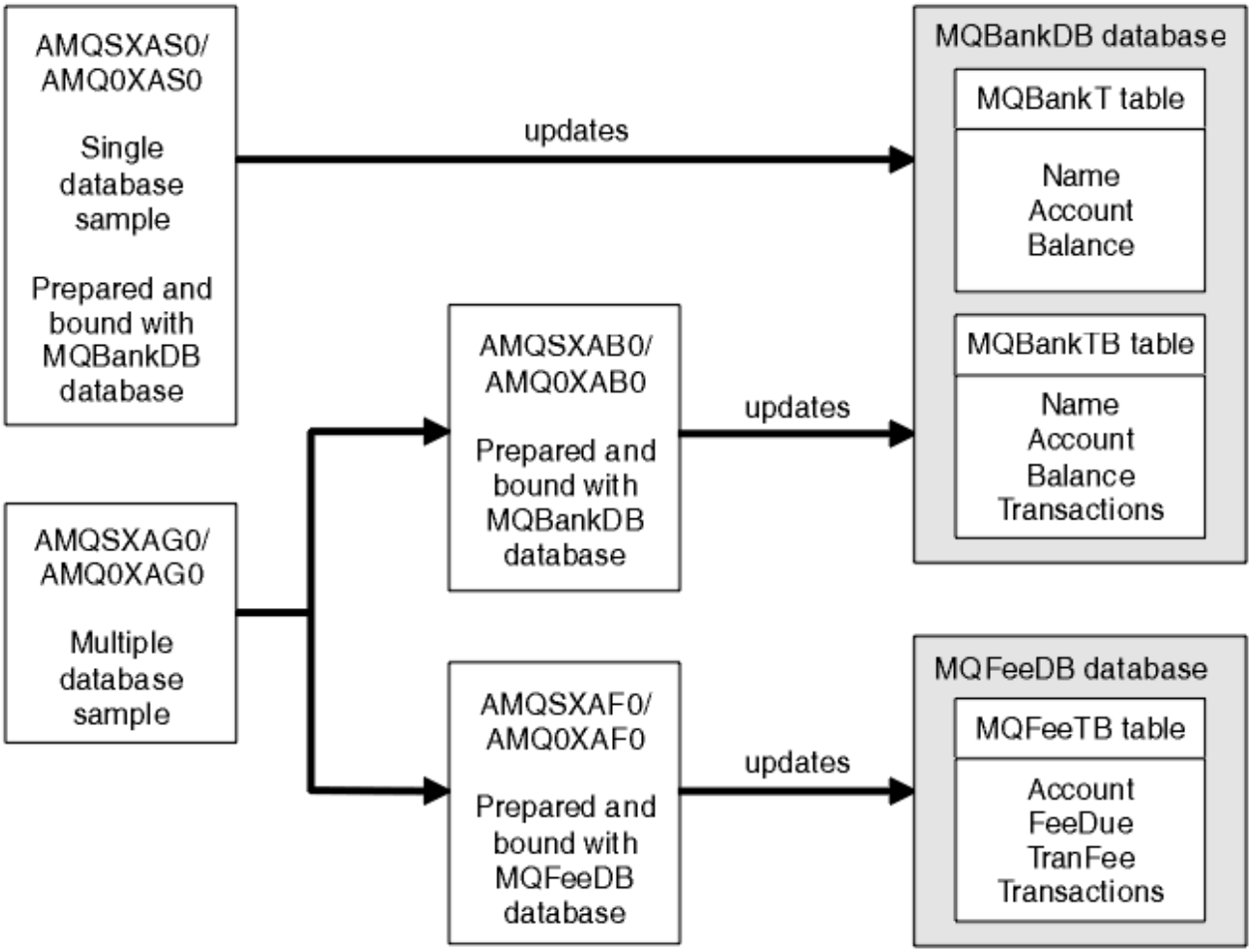


图 *18: "*数据库协调"样本

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。

单个数据库样本 (AMQSXAS0 或 AMQ0XAS0) 没有 SQL CONNECT 语句， WebSphere MQ 通过 MQBEGIN 调用隐式建立与数据库的连接。 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。

这些样本是使用 IBM DB2 数据库产品准备的，因此您可能需要对其进行修改以使用其他数据库产品。

SQL 错误检查使用 UTIL.C 和 CHECKERR.CBL 由 DB2 提供。 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（在 C 中）和 .SQB 文件（在 COBOL 中），并针对相应数据库绑定这些文件以生成 .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

以下样本命令使用符号 *<DB2TOP>* 和 *MQ_INSTALLATION_PATH*。 *<DB2TOP>* 表示 DB2 产品的安装目录。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。

- 在 AIX 上，目录路径为:

```
/usr/lpp/db2_05_00
```

- 在 HP-UX 和 Solaris 上，目录路径为:

```
/opt/IBMdb2/V5.0
```

- 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，请确保 LIB 环境变量包含 DB2 和 WebSphere MQ 库的路径。

将以下文件复制到临时目录中：

- WebSphere MQ 安装中的 amqsxag0.c 文件

  注: 可在下列目录中找到此文件：

  – 在 UNIX and Linux 系统上：

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – 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 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
  ```

使用以下适用于所用平台的编译器命令，为每个 .c 文件构建对象文件：

- AIX

  ```
  xlc_r -IMQ_INSTALLATION_PATH/inc -I
  <DB2TOP>/include -c -o
   <FILENAME>.o <FILENAME>.c
  ```

- HP-UX

  ```
  cc -Aa +z -IMQ_INSTALLATION_PATH/inc -I
  <DB2TOP>/include -c -o
   <FILENAME>.o <FILENAME>.c
  ```

- Solaris

  ```
  cc -Aa -KPIC -mt -IMQ_INSTALLATION_PATH
  /inc -I<DB2TOP>/include -c -o
   <FILENAME>.o <FILENAME>.c
  ```

- Windows 系统

  ```
  cl /c /IMQ_INSTALLATION_PATH\tools\c\include /I
  <DB2TOP>\include
   <FILENAME>.c
  ```

使用以下适用于所用平台的链接命令，构建 amqsxag0 可执行文件：

- AIX

  ```
  xlc_r -H512 -T512 -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib
   -lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
  ```

- HP-UX 修订版 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread -lcl
 /lib/crt0.o util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Solaris

```
cc -mt -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib
 -lmqm -lthread -lsocket -lc -lnsl -ldl util.o
  amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows 系统

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
 /out:amqsxag0.exe
```

使用适用于所用平台的以下编译和链接命令，构建 amqsxas0 可执行文件：

- AIX

```
xlc_r -H512 -T512 -L<DB2TOP>/lib -ldb2
 -LMQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- HP-UX 修订版 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread
 -lcl /lib/crt0.o util.o amqsxas0.o -o amqsxas0
```

- Solaris

```
cc -mt -L<DB2TOP>/lib -ldb2-LMQ_INSTALLATION_PATH/lib
 -lqm -lthread -lsocket -lc -lnsl -ldl util.o
  amqsxas0.o -o amqsxas0
```

- Windows 系统

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

其他信息

如果您正在 AIX 或 HP-UX 上工作，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，请参阅第 36 页的『方案 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 102 页的『C 样本』
- 第 103 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， **而不是** 放入 WebSphere MQ 错误日志文件。

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 80 页的『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 UNIX， Linux 和 Windows 系统上，它们的唯一输入是包含目标队列和队列管理器的名称的 MQTMC2 (触发器消息) 结构。 COBOL 版本使用缺省队列管理器。

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、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。

“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 80 页的『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

这些程序各采用两个参数:

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 amqsget 将连接到缺省队列管理器，而 amqsgetc 将连接到由环境变量或客户机通道定义文件标识的队列管理器。

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` 是该程序要从中获取消息的队列的名称，而 `qmanagername` 是拥有 `myqueue` 的队列管理器。

如果省略 `qmanagername`，那么程序将采用缺省值，或者在 MQI 客户机的情况下，采用由环境变量或客户机通道定义文件标识的队列管理器。

“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 108 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 IBM WebSphere 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 IBM WebSphere 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 IBM WebSphere 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 *QueueManager* 名称 *QueueName SideQueue* 名称 *InTransaction* 计数 *RepeatCount* (0|1|2)

- **amqsfhac** 是 IBM WebSphere 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 WebSphere MQ 多实例队列管理器是否维护数据完整性。 遵循 验证共享文件系统行为 中运行 **amqsfhac** 的步骤。
- 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。
- 它将 *InTransactionCount***RepeatCount* 持久消息放入 *QueueName* 中，在此期间，您会使队列管理器进行任何次数的故障转移。 **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。
- 通过将最后一个参数设置为 (0|1|2)，使输出跟踪量与 **amqsfhac** 不同。

  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。 请参阅验证客户机安装。

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。

消息由 **amqsphac** 放置在队列 SOURCE 上，由 **amqsmhac** 传输到 TARGET，并由 **amqsghac** 从 TARGET 检索; 请参阅 第 108 页的图 19。

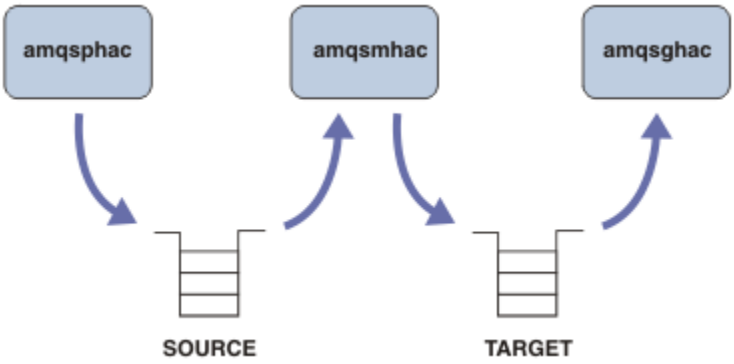


图 *19:* 可重新连接的客户机样本

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，在先前命令提示符处输入 **start** 三次，在其中一个窗口中启动每个程序。 请参阅第 108 页的『设置和控制队列管理器』中的步骤 第 109 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
WebSphere 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
WebSphere 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'.
WebSphere 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: WebSphere MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: WebSphere MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: WebSphere MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: WebSphere MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: WebSphere MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start WebSphere MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start WebSphere 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 108 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.
WebSphere MQ queue manager 'QM1' ending.
WebSphere MQ queue manager 'QM1' ended.

C:\>strmqm QM1
WebSphere 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'.
WebSphere 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 80 页的『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，对于 IBM i、Windows 和 UNIX and Linux 系统，这些程序的唯一输入是 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。 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：

- 对于 UNIX、Linux 和 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。

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。

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，程序将读取数据中包含的队列 (我们将调用 目标队列) 的名称，并使用带有 MQOO_INQ 选项的 MQOPEN 调用打开该队列。 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\WebSphere 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

发布/预订样本程序演示了 WebSphere MQ 中发布和预订功能的使用。

有三个 C 语言样本程序说明如何编程到 WebSphere MQ 发布/预订接口。 有一些 C 样本使用较旧的接口，还有 Java 样本。 Java 样本使用 com.ibm.mq.jar 中的 WebSphere 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程序样本 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。 此外还提供了一个名为 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 88 页的『准备并运行样本程序』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。 从现在开始进入此窗口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。

订户使用输出 Calling MQGET : 30 seconds wait time 进行响应。 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)。

此外还提供了另一个预订程序样本 amqssbx，它提供额外的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）。

这些附加样本位于 samples 文件夹中的 pubsub 子目录中。

第 112 页的表 20 中列出了六种样本类型。

表 *20:* 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. 需要在 amqsgama.tst 和 amqsresa.tst 中定义队列 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. 需要在 amqsgama.tst 和 amqsresa.tst 中定义队列 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

Java 样本 MQPubSubApiSample.java 在单个程序中组合了发布程序和订户。 可在 wmqjava 样本文件夹中找到其源代码和已编译的类文件。

如果您选择以客户机模式运行，请先查看第 88 页的『准备并运行样本程序』以获取详细信息。

如果已配置 Java 环境，请使用 Java 命令从命令行运行样本。 您还可以从已设置 Java 编程工作台的 WebSphere 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 112 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

如何使用 Eclipse 平台中的 Java 开发工具来运行 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 88 页的『准备并运行样本程序』中的步骤。

关于此任务

Java 发布/预订样本程序是 WebSphere MQ MQI 客户机 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。 此任务描述此简单案例，并在一般术语中指示如何提供参数和修改样本以适应不同的 WebSphere MQ 配置。 说明了在 Windows 上运行的示例。 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 WebSphere MQ 安装文件夹，然后选择 `com.ibm.mq.jar` 和 `com.ibm.mq.jmqi.jar`
   h) 单击**打开** > **完成**。
   i) 右键单击**“包资源管理器”**视图中的 `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 WebSphere MQ 安装目录。
   k) 在**“导入”**面板 (第 114 页的图 20) 上，单击 `samples`（请勿选中此复选框）。
   l) 选择 `MQPubSubApiSample.java`。 **Into folder** 字段应包含 `MQ Java Samples/src`。 单击**完成**。

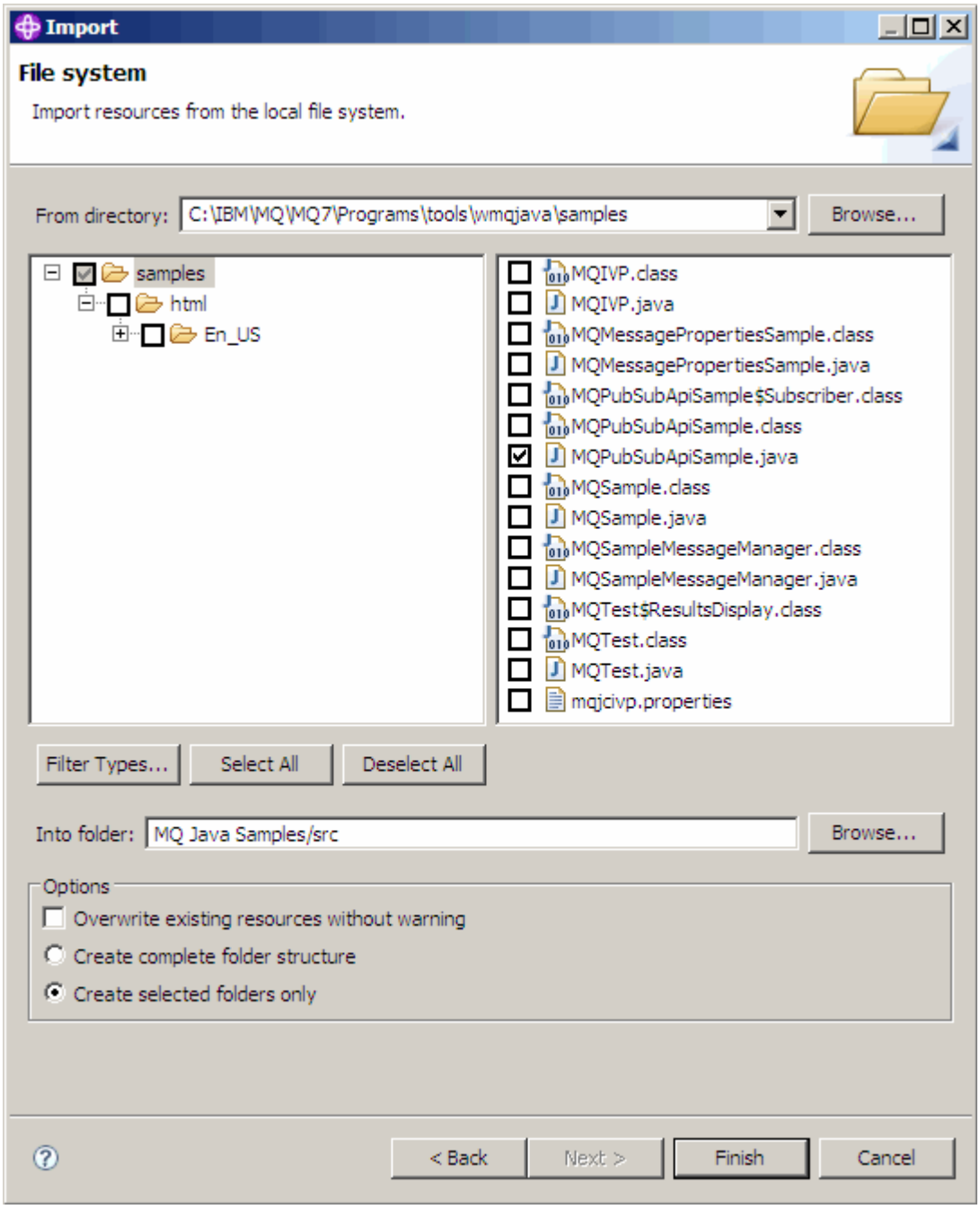


图 *20:* 文件系统导入

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。
       这些属性可使用 -D JVM 开关进行修改，或者通过编辑源代码为系统属性提供缺省值。

- 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

要设置系统属性，请在存取器中编码缺省值，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) = 参数** 选项卡。
c. 在 **VM 参数:** 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:
   - 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* 表示安装了 WebSphere MQ 的高级目录。 在 Windows 上，在注册表中设置等效属性。
2. 确保模块可供 WebSphere MQ 访问。
3. 重新启动队列管理器以选取配置。
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:
   - 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
     ```

   - 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 80 页的『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

这些程序各采用 2 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 将连接到环境变量或客户机通道定义文件所指定的队列管理器。 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

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。

“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) ，而不需要将该对象存储在源节点或目标节点上的 WebSphere 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。 以下部分描述了“参考消息”样本程序。

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。 如果接收方在任何 WebSphere MQ 产品 (WebSphere MQ for IBM i 下运行，请使用 amqsxrm4 。

运行“参考消息”样本

请使用以下信息来了解如何运行“参考消息”样本程序。

“参考消息”样本按如下方式运行:

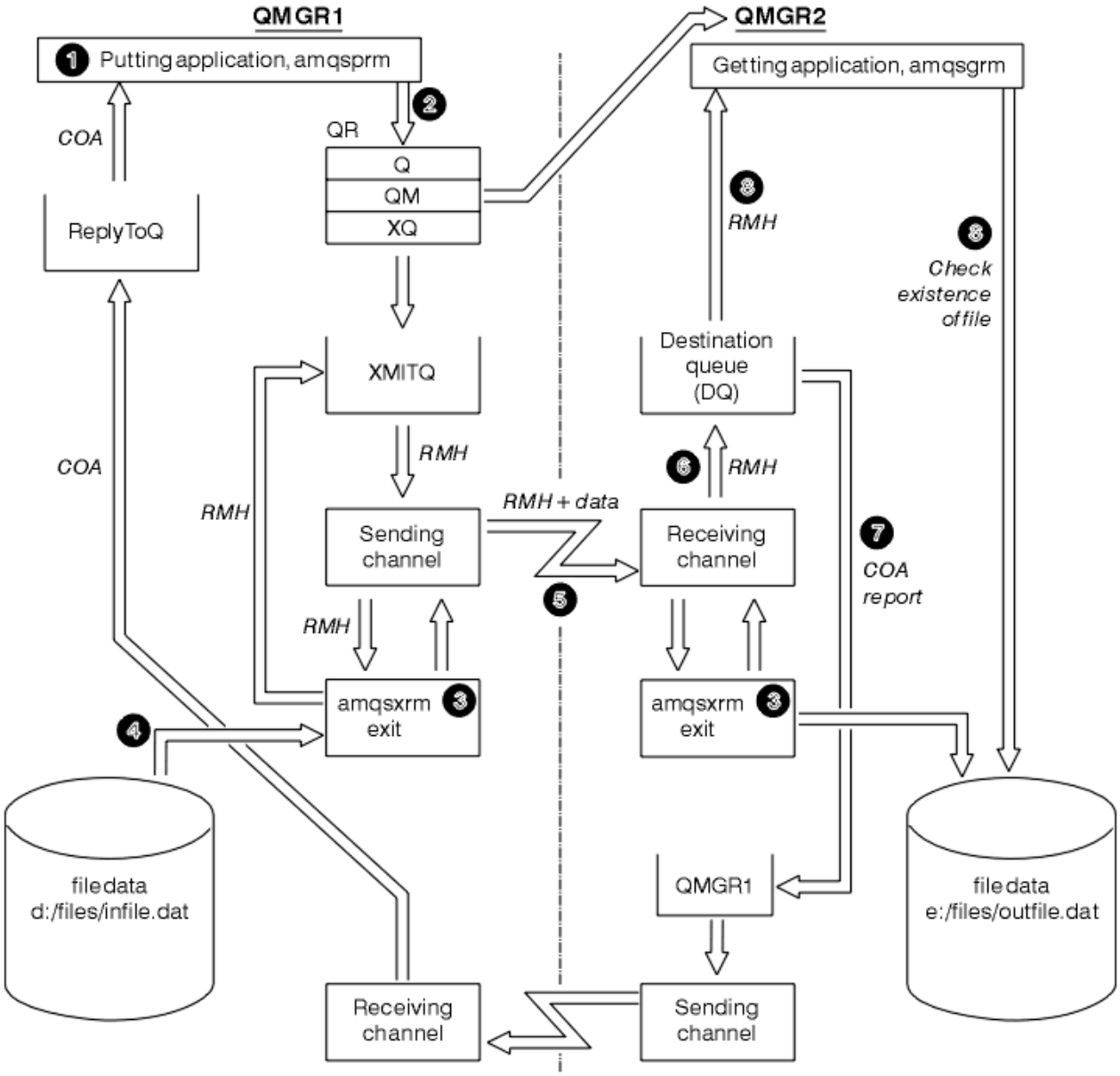


图 *21:* 运行“参考消息”样本

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”示例，这里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）的应用程序。 根据机器类型以及安装 WebSphere MQ 产品的位置，确保正确设置了 CONNAME 字段，并且 MSGEXIT 字段反映了您的目录结构。

   MQSC 定义已使用 AIX 样式来定义出口。 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
```

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)
```

2. 创建 WebSphere 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:

   -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 队列名称

   -g -q 参数定义的队列所在的队列管理器的名称。此选项缺省设置为 -m 参数指定的队列管理器

   -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
```

注: 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。 这是在 WebSphere MQ for Windows 上定义的，如下所示：

```
msgexit('pathname\amqsxrm.dll(MsgExit)')
```

这在 WebSphere MQ for AIX， WebSphere MQ for HP-UX 和 WebSphere MQ for Solaris 上定义，如下所示：

```
msgexit('pathname/amqsxrm(MsgExit)')
```

如果指定路径名，请指定完整名称。 如果省略路径名，那么假定程序在 `qm.ini` 文件中指定的路径中 (或者在 WebSphere MQ for Windows 上，在注册表中指定的路径)。

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。 在 WebSphere MQ for AIX, WebSphere MQ for HP-UX 和 WebSphere MQ for Solaris 上，将目标目录的组所有者更改为 "mqm" ，以便样本消息出口可以在该目录中创建文件。 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。 文件数据未存储在 WebSphere MQ 队列上。
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， WebSphere MQ 随附的样本构建参考消息，以将文件 d:\x\file.in 从 QMGR1 发送到 QMGR2 ，并使用以下参数以 d:\y\file.out 形式重新创建该文件:

   ```
   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，而不是继续发送至目标队列。 在 MQCXP 中返回一个反馈代码。*Feedback* 并返回到将消息放置在报告消息的消息描述符的 *Feedback* 字段中的应用程序。 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

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

要编译 amqsxrma，请使用以下命令:

在 AIX 上

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

在 HP-UX 上

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -IMQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -LMQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

打开 Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
```

在 Solaris 上

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm

 -lsocket
-lnsl -ldl
```

在 Windows 上

WebSphere MQ 现在提供了具有客户机软件包和服务器软件包的 mqm 库，因此以下示例使用 `mqm.lib` 而不是 `mqmvx.lib`:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

有关编写和编译通道出口的常规信息，请参阅第 333 页的『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 80 页的『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。

UNIX， Linux 和 *Windows* 系统

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 123 页的图 22 说明了如何结合使用“请求”和“查询”样本。

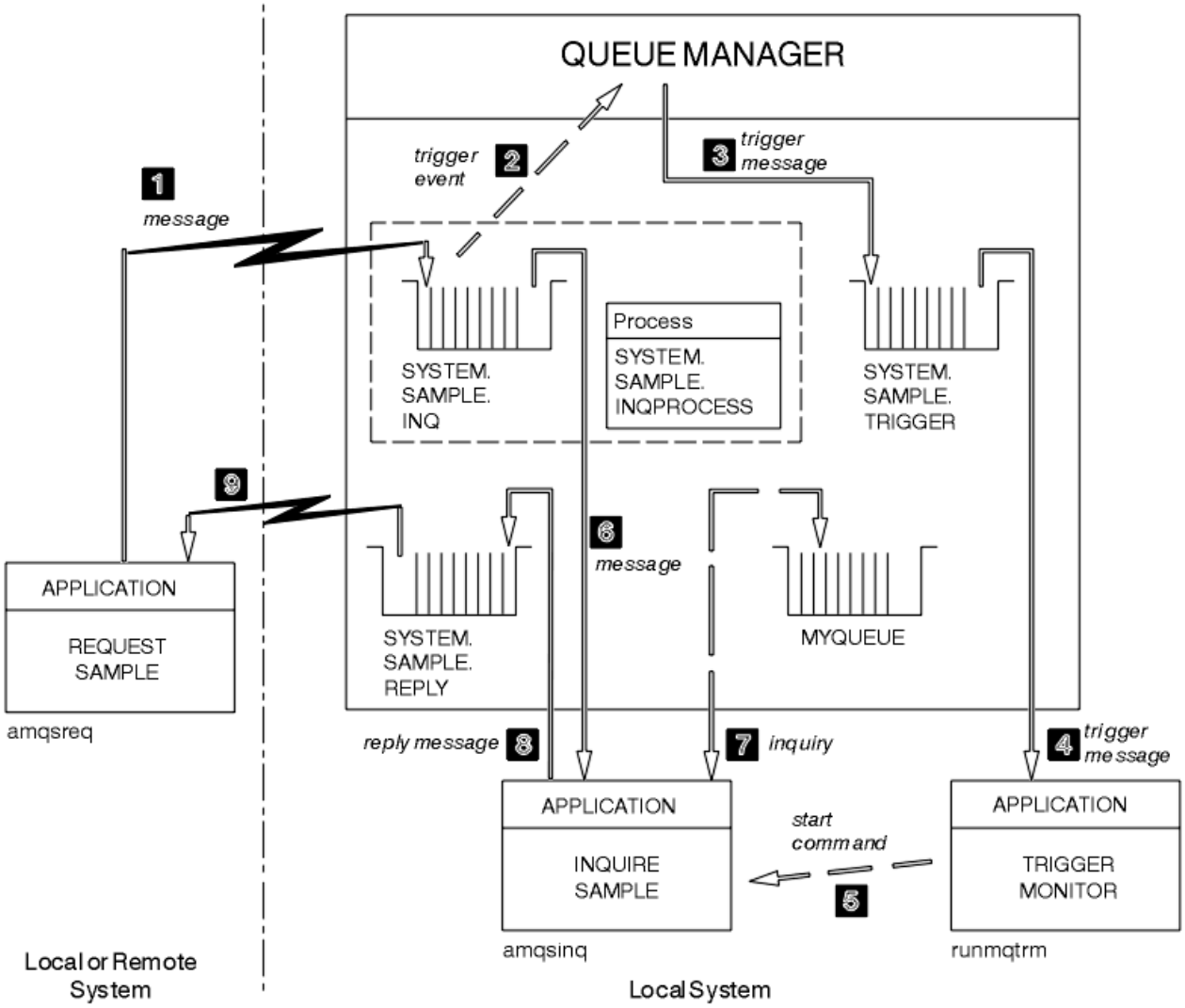


图 *22:* 使用触发机制的“请求”和“查询”样本

在第 123 页的图 22 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 123 页的图 22 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 qmanagerıname -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 124 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』。

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。

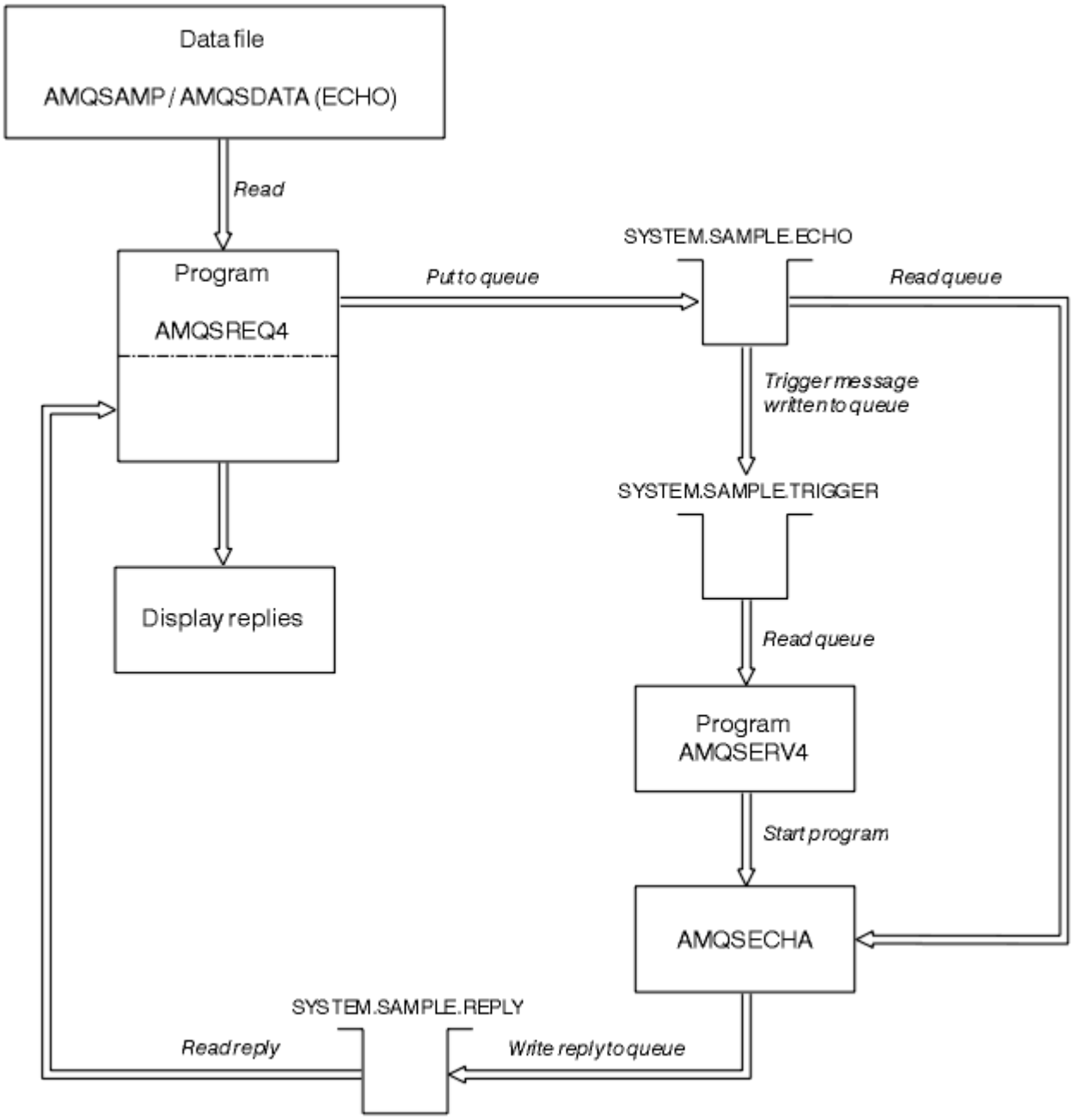


图 *23:* 样本 *IBM i* 客户机/服务器（回传）程序流程图

“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 80 页的『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：

- 对于 UNIX、Linux 和 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。
- 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，程序将读取数据中包含的队列 (我们将调用 目标队列) 的名称，并使用带有 MQOO_SET 选项的 MQOPEN 调用打开该队列。 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。

SSL/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 SSL/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 SSL/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

SSL/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 SSL/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。

OCSP 证书撤销参数：

-o URL
OCSP 响应程序 URL

运行 *SSL/TLS* 样本程序

要运行 SSL/TLS 样本程序，首先必须设置 SSL 或 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 系统上设置密钥存储库。
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 系统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 从 UNIX, Linux 和 Windows 系统上的密钥存储库中抽取自签名证书的公共部分 ，以及 将 CA 证书 (或自签名证书的公共部分) 添加到 UNIX， Linux 或 Windows 系统上的密钥存储库中。
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(NULL_SHA)
   ```

   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\WebSphere MQ\clientkey" -s NULL_SHA
-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 SSL CipherSpec NULL_SHA
Using SSL key repository stem c:\Program Files\IBM\WebSphere 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
```

要获取原因码的列表，请参阅 API 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 80 页的『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*。 这些字段中的最后两个字段是数字，因此程序将创建字符替换以在 UNIX， Linux 和 Windows 系统的 MQTMC2 结构中使用。

“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。 在 UNIX， Linux 和 Windows 系统中， *EnvData* 字段用作调用命令字符串的扩展。

最后，该程序将关闭启动队列。

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: 此样本中的函数是 **runmqtrm** 程序中提供的完整触发函数的子集。

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
- 对于 CICS 应用程序，从 STRCICSUSR 命令中的触发器消息替换 *EnvData*，例如，指定 CICS 区域
- 打开启动队列以获取共享输入，因此允许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。

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。

在运行这些样本之前，必须构建服务器环境。

注: 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。 在单独一行中输入每个命令。

构建服务器环境

有关为不同平台的 WebSphere MQ 构建服务器环境的信息。

假定您拥有正常运行的 TUXEDO 环境。

为 *WebSphere MQ for AIX (32 位)* 构建服务器环境

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， *MQ_INSTALLATION_PATH* 表示安装了 WebSphere 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. 将以下内容添加到 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
```

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。

为 *WebSphere MQ for AIX (64 位)* 构建服务器环境

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 表示 TUXEDO 的根目录， *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ installed.：

```
       $ 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. 将以下内容添加到 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
```

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。

为 *WebSphere MQ for Solaris (32 位)* 构建服务器环境

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

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：

```
$ 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。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a
```

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  \

$ buildclient -o dogets  -f amqstxgx.c \
```

```
        -f MQ_INSTALLATION_PATH/lib/libmqm.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
```

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。

为 *WebSphere MQ for Solaris (64 位)* 构建服务器环境

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

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。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a
```

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  \

$ buildclient -o dogets  -f amqstxgx.c \
```

```
        -f MQ_INSTALLATION_PATH/lib64/libmqm.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
```

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。

为 *WebSphere MQ for HP-UX (32 位)* 构建服务器环境

注: 32 位 TUXEDO 服务器环境只能在 Itanium 平台上构建。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

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="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. 将以下内容添加到 TUXEDO 文件 udataobj/RM

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.so MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

4. 运行命令：

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

在运行 mkfldhdr 和 viewc 命令之后，TUXEDO 应用程序目录中将创建 amqstxvx.h 头文件。 将此文件从 TUXEDO 应用程序目录复制到 TUXEDO include 目录，然后运行以下命令。

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
```

```
        -f MQ_INSTALLATION_PATH/lib/libmqm.so \
        -r MQSeries_XA_RMI -s MPUT2:MPUT \
        -s MGET2:MGET \
        -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
        -f MQ_INSTALLATION_PATH/lib/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
        -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. 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
```

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。

为 *WebSphere MQ for HP-UX (64 位)* 构建服务器环境

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

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="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib64:/lib64
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. 将以下内容添加到 TUXEDO 文件 udataobj/RM

在 HP-UX IA64 (IPF) 平台上：

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

注: HP-UX IA64 (IPF) 平台上提供的 WebSphere MQ 库具有 .so 文件扩展名。

4. 运行命令：

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

在运行 mkfldhdr 和 viewc 命令之后，TUXEDO 应用程序目录中将创建 amqstxvx.h 头文件。 将此文件从 TUXEDO 应用程序目录复制到 TUXEDO include 目录，然后运行以下命令。

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
```

在 HP-UX IA64 (IPF) 平台上:

```
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

5. 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
```

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。

为 *WebSphere MQ for Windows (32 位)* 构建服务器环境

注: 将以下 <> 标识的字段更改到以下目录路径:

<MQMDIR>	安装 WebSphere MQ 时指定的目录路径，例如 `g:\Program Files\IBM\WebSphere MQ`
<TUXDIR>	安装 TUXEDO 时指定的目录路径，例如 `f:\tuxedo`
<APPDIR>	用于样本应用程序的目录路径，例如 `f:\tuxedo\apps\mqapp`

要构建服务器环境和样本:

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 WebSphere 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 137 页的图 24）以添加机器名称和要连接到的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
```

where <MQMDIR> is replaced as shown in the previous example. 虽然此处显示为两行，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，请调用 `tmadmin -c`，然后输入以下命令:

```
crdl -z <APPDIR>\TLOG
```

其中 <*APPDIR*> 已替换。

8. 将当前目录设置为 <APPDIR>，并且将样本 makefile (amqstxmn.mak) 作为外部项目 makefile 进行调用。 例如，使用 Microsoft Visual C++ ，发出以下命令:

```
msvc amqstxmn.mak
```

选择 **build** 构建所有样本程序。

```
*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 *24: WebSphere MQ for Windows* 的 *ubbstxcn.cfg* 文件示例

注: 更改目录名称和目录路径以匹配您的安装。 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。 需要添加的其他信息由 <> 字符标识。

WebSphere MQ for Windows 的样本 ubbconfig 文件在 第 137 页的图 24 中列出。 它在 WebSphere MQ 样本目录中作为 ubbstxcn.cfg 提供。

为 WebSphere MQ for Windows 提供的样本 makefile (请参阅 第 138 页的图 25) 称为 ubbstxmn.mak，并保存在 WebSphere 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 *25: WebSphere MQ for Windows* 的样本 *TUXEDO makefile*

为 *WebSphere MQ for Windows (64* 位*)* 构建服务器环境

注: 将以下 <> 标识的字段更改到以下目录路径:

<MQMDIR>	安装 WebSphere MQ 时指定的目录路径，例如 `g:\Program Files\IBM\WebSphere MQ`
<TUXDIR>	安装 TUXEDO 时指定的目录路径，例如 `f:\tuxedo`
<APPDIR>	用于样本应用程序的目录路径，例如 `f:\tuxedo\apps\mqapp`

要构建服务器环境和样本:

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 WebSphere 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 139 页的图 26）以添加机器名称和要连接到的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
   ```

   其中 <MQMDIR> 将被替换。 虽然此处显示为两行，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，请调用 `tmadmin -c`，然后输入以下命令:

```
crdl -z <APPDIR>\TLOG
```

   其中 <*APPDIR*> 已替换，如上例中所示。

8. 将当前目录设置为 <APPDIR>，并且将样本 makefile (amqstxmn.mak) 作为外部项目 makefile 进行调用。 例如，使用 Microsoft Visual C++ ，发出以下命令:

```
msvc amqstxmn.mak
```

   选择 **build** 构建所有样本程序。

```
*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 *26: WebSphere MQ for Windows* 的 *ubbstxcn.cfg* 文件示例

注: 更改目录名称和目录路径以匹配您的安装。 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。 需要添加的其他信息由 <> 字符标识。

WebSphere MQ for Windows 的样本 ubbconfig 文件在 第 139 页的图 26 中列出。 它在 WebSphere MQ 样本目录中作为 ubbstxcn.cfg 提供。

为 WebSphere MQ for Windows 提供的样本 makefile (请参阅 第 140 页的图 27) 称为 ubbstxmn.mak，并保存在 WebSphere 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 *27: WebSphere MQ for Windows* 的样本 *TUXEDO makefile*

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。

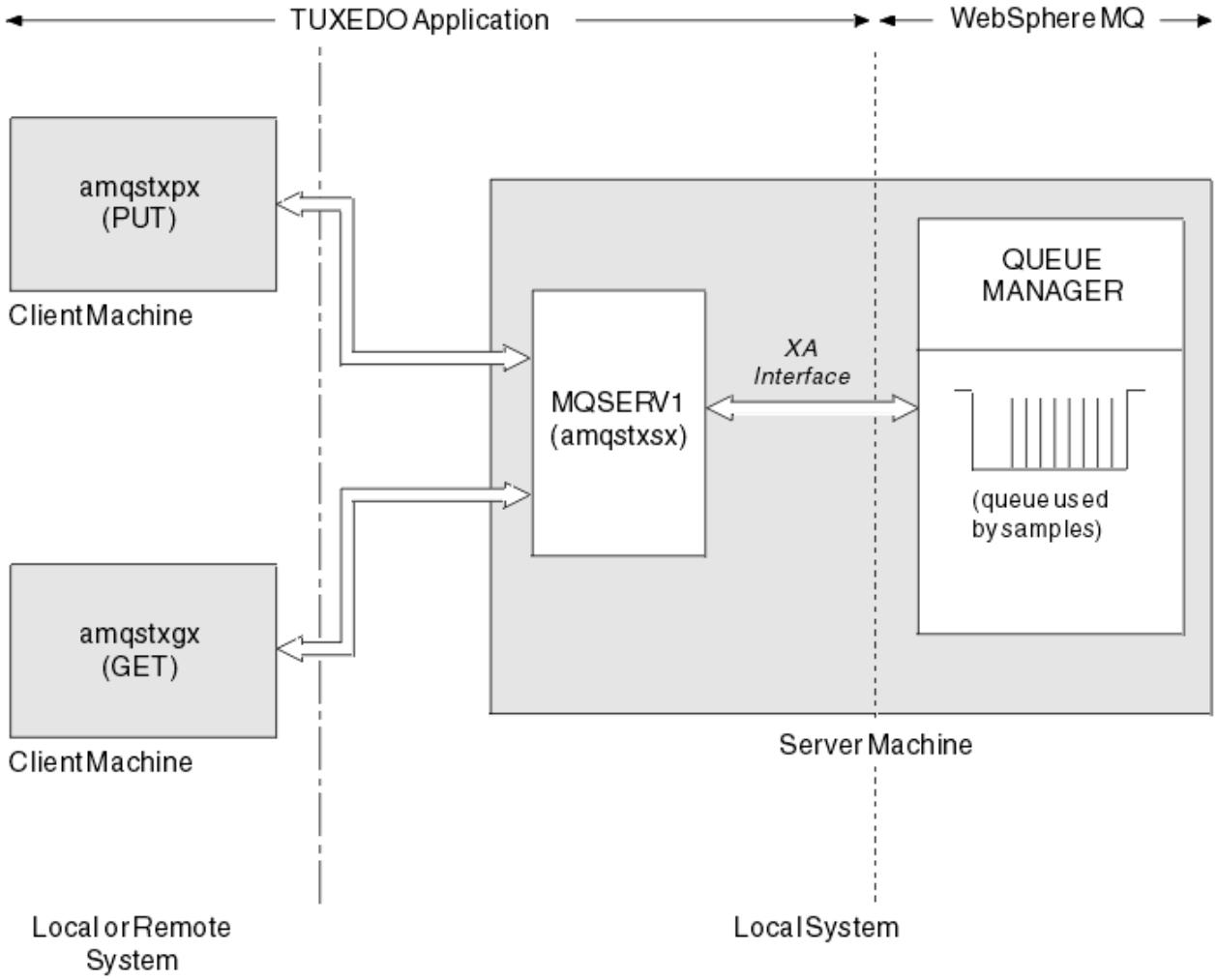


图 *28: TUXEDO* 样本如何协同工作

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。

TUXEDO 的 *GET* 样本

此样本允许您从队列中成批获取消息。

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：

- `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。

在 Windows 系统上使用 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

对象代码文件名为 amqrspin.dll。 对于客户机和服务器，它都作为 WebSphere 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\WebSphere 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* 在表单 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

出口源代码文件名为 amqsspin.c。 它位于 C:\Program Files\IBM\WebSphere 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。

在您执行以下命令之前，确保您的路径中提供 cl.exe，和 Visual C++ 库以及 include 文件夹。 例如：

```
cl /VERBOSE /LD /MT /I<path_to_Microsoft_platform_SDK\include>
/I<path_to_WebSphere 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 WebSphere 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 WebSphere 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， WebSphere MQ 使用每个队列实例的集群工作负载优先级属性 **CLWLPRTY** 来确定要向哪些实例发送消息。 WebSphere MQ 将消息发送到具有最高 **CLWLPRTY** 值的队列的可用实例。

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。

WebSphere MQ 集群技术将集群队列的已更新 **CLWLPRTY** 属性传播到集群中的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**) 并将消息 (使用 **MQPUT**) 放入同一个集群队列，将消息从不活动的本地队列传输到一个或多个活动的远程队列。 此传输会导致 WebSphere MQ 集群工作负载管理根据高于本地队列实例的 CLWLPRTY 值来选择其他目标实例。 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于底层 WebSphere 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。
- 必须在具有适当权限的情况下运行集群队列监视样本程序，才能访问所需的所有 WebSphere 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， WebSphere MQ 集群工作负载均衡将向位于放置应用程序所连接到的同一队列管理器上的集群队列实例发送消息。 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。
- WebSphere MQ 系统和应用程序不得将 **CLWLPRTY** 用于要监视的队列或正在使用的通道。 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

为了运行集群队列监视样本，您必须配置队列管理器以安全接受客户机模式下运行的应用程序的入站连接请求。

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，可以使用 **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，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 WebSphere 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 **RUNMQSC** 示例将集群队列监视样本定义为 IBM WebSphere MQ 服务。

   ```
   define service(AMQSCLM) +
       descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
   ```

```
    control(qmgr) +
    servtype(server) +
    startcmd('<Install Root>\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')
```

其中 <Install Root> 是安装位置。

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**	包含要监视的队列名称掩码的队列名称列表的文件的完整路径和文件名。 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**	The directory path in which to store logged information messages in a set of wrapping<fn>For each queue manager and queue combination a report file is generated that is capped at a certain size. The logger always writes into the same file, but it also keeps the two previous versions of the file.</fn> report files.
-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) 当受监视队列实例处于不活动状态时，自动将其 **CLWLUSEQ** 属性切换为 ANY，当其处于活动状态时，将其切换为 **ActiveVal** 属性。 **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）除报告文件外，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 WebSphere 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。

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 WebSphere 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 文件填满后，将创建新的 RPT01 文件。 旧的 RPT01 文件重命名为 RPT02。 RPT02 重命名为 RPT03。 将丢弃旧 RPT03。

在下列情况下，样本创建参考消息：

- 在启动时
- 在终止时
- 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： 检查消息标识 CLM0010E 的报告文件。

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。

可能的解释：队列处于 INACTIVE 状态， **CLWLPRTY**= 0 ，因为它没有使用者。 或者，它保持 ACTIVE **CLWLPRTY**> =1 ，因为它至少有 1 个使用者。

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

潜在说明：消息将直接放入拥有不活动队列的队列管理器，并且队列的 **CLWLUSEQ** 值不是 ANY，并且 **-u** 参数不用于 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

潜在说明：未启用消息传输 (**-t**)。

操作：确保启用消息传输 (**-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。

潜在解释：队列具有 **CLWLUSEQ**=LOCAL 或 **CLWLUSEQ**=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 或更大的队列实例。

可能的解释:远程队列实例具有使用者 (**IPPROCS**> = 1) ，但在这些队列管理器上处于不活动状态 (**CLWLPRTY**= 0) ，因为 AMQSCLM 未监视这些远程实例。

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: 远程队列实例处于活动状态 (**CLWLPRTY**= 1) ，但在本地队列管理器上被视为处于不活动状态 (**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: 此场景可能是由于 AMQSCLM 未以具有足够权限访问队列的用户身份运行。

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。

连接端点查找 (CEPL) 的样本程序

IBM WebSphere MQ "连接端点查找" 样本提供了一个简单而功能强大的出口模块，该模块为 WebSphere MQ 用户提供从 LDAP 存储库 (例如 Tivoli Directory Server) 检索连接定义的方法。

必须安装 Tivoli Directory Server v6.3 Client 才能使用 CEPL。

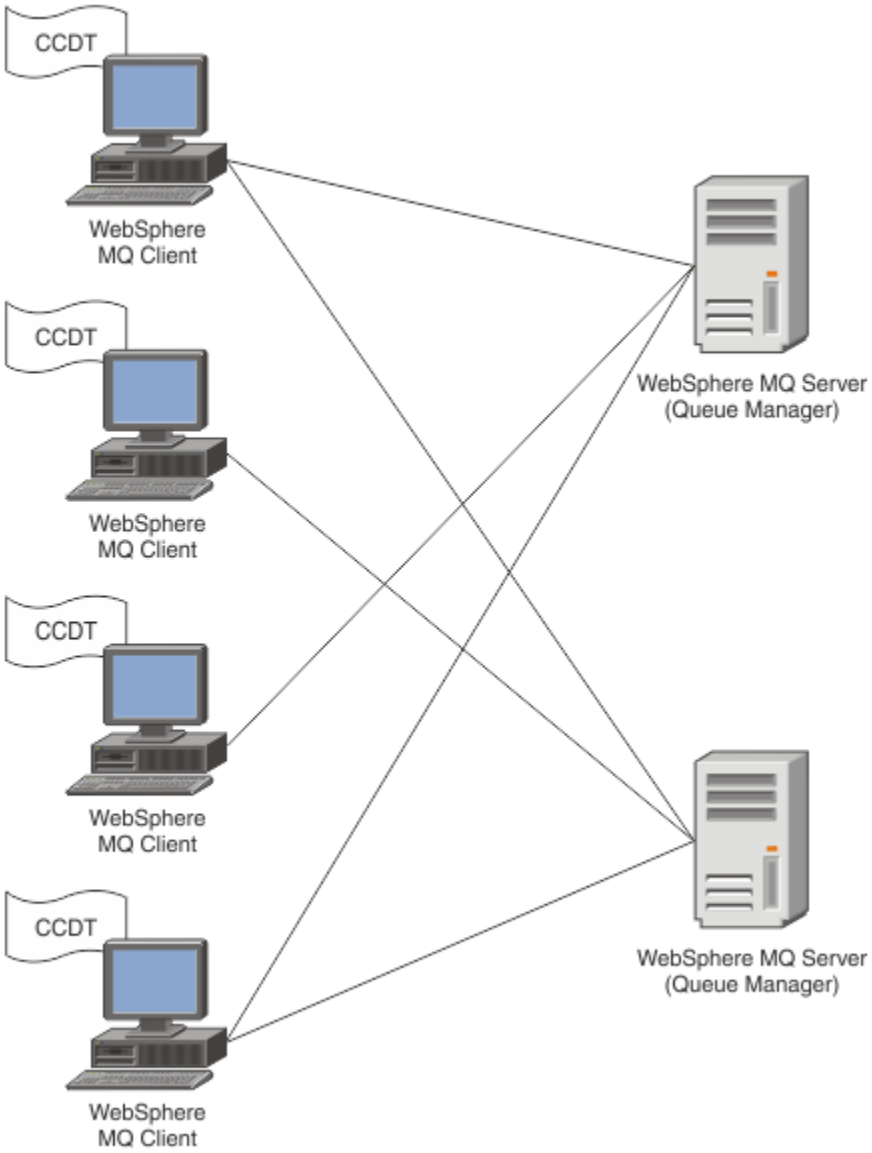
要使用此样本，需要具备受支持平台上的 WebSphere MQ 管理的工作知识。

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 WebSphere 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。

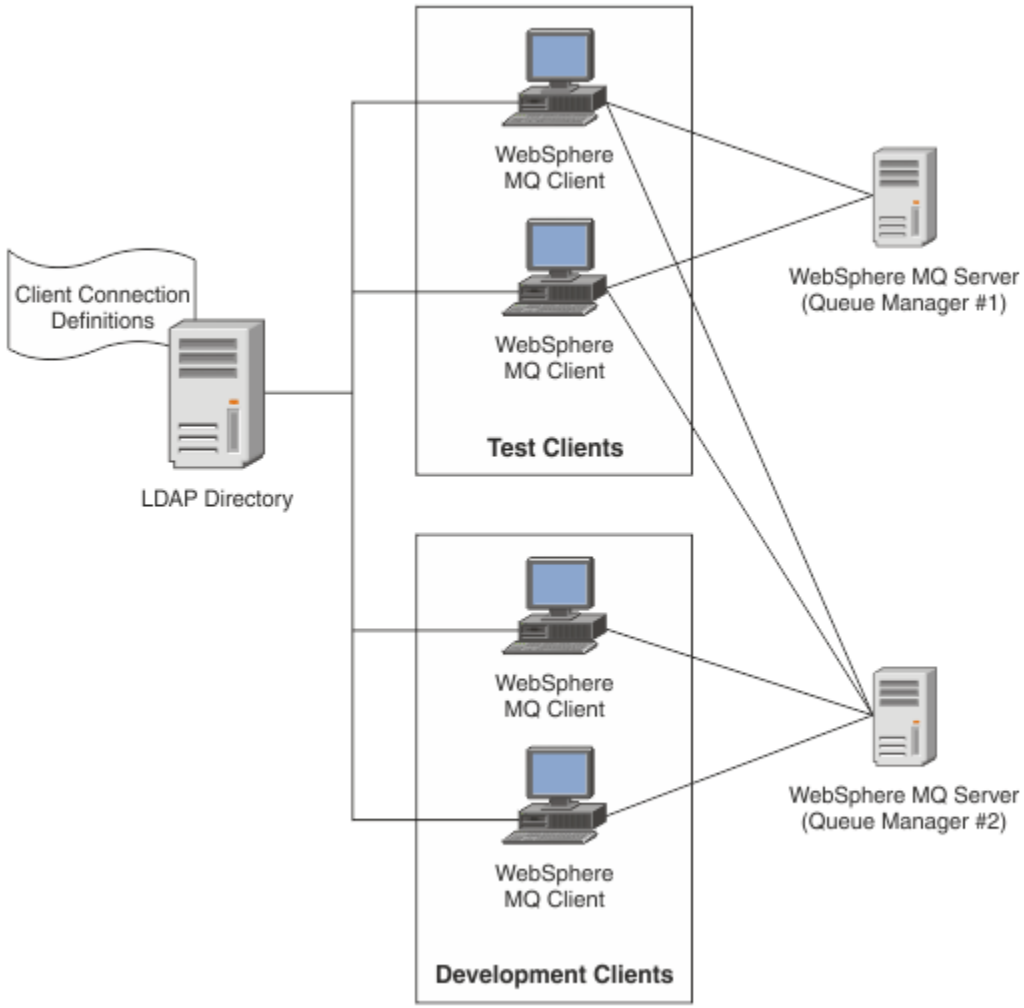
CCDT 是通过标准 WebSphere MQ MQSC 管理接口创建的。 用户必须连接到队列管理器才能创建客户机连接定义，即使定义中包含的数据没有限制为队列管理器。 生成的 CCDT 文件必须在客户机和应用程序之间手动分发。



CCDT 文件必须分发到每个 WebSphere MQ 客户机。 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。



出口模块可以查找 LDAP 存储库 (例如， IBM Tivoli Directory Server) 以检索通道定义。 通过使用这些连接定义， WebSphere MQ 客户机应用程序可以建立与队列管理器的连接。

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 WebSphere 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。

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 WebSphere MQ 连接端点查找需要以下软件：

- IBM WebSphere MQ V7.0 或更高版本
- Tivoli Directory Server V6.3 Client 或更高版本

受支持的操作系统：

1. Windows (XP/2003/2008)
2. Solaris (SPARC 和 x86-64)
3. AIX
4. Linux
   - System p 上的 RHEL v4 和 v5
   - System p 上的 SUSE v9 和 v10
   - RHEL v4 和 v5 System x32 位和 x64 位
   - SUSE v9 和 v10 System x32 位和 x64 位
5. HP IA64。

注: 样本程序不可用于 z/OS， i/5 和 HP PARISC 平台。

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 WebSphere MQ 期间，出口模块安装在 `tools/samples/c/preconnexit/bin` 下。 对于 32 位平台，必须先将出口模块复制到 `exit/<install 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

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。 此示例假定正在使用 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，其中 LDAP 标识和 LDAP 密码是 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** 的部分添加到客户机配置文件 *mqclient.ini*。 PreConnect 部分包含以下关键字:

Module： 包含 API 出口代码的模块名称。 如果此字段包含模块的完整路径，那么将搜索 WebSphere MQ 安装中的其他 *exit* 或 *exit64* 文件夹。

Function： 包含 PreConnect 出口代码的库的函数入口点名称。 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* 文件所需更改的示例。

```
PreConnect:
  Module=amqlcelp
  Function=PreConnectExit
  Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
  Sequence=1
```

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。

WebSphere MQ v7.5 为出口模块中的入口点定义以下语法。

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP  pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

在 MQCONN/X 调用执行期间， WebSphere MQ C Client 装入包含函数语法实现的出口模块。 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。

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 */
};
```

构建连接端点查找出口的样本代码

用于在 Windows 和分布式平台上编译源代码的代码片段。

编译源

您可以使用任何 LDAP 客户机库 (例如， IBM Tivoli Directory Server 6.3 客户机库) 来编译源。 本文档假定您正在使用 Tivoli Directory Server 6.3 客户机库。

注: 已使用以下 LDAP 服务器测试预连接出口库:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

以下代码片段描述如何在 Windows 和其他分布式平台上编译出口:

在 Windows 平台上编译出口

可以使用以下片段在 Windows 上编译出口源:

```
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: 如果您使用的是使用 Microsoft Visual Studio 2003 编译器编译的 IBM Tivoli Directory Server 6.3 客户机库，那么在使用 Microsoft Visual Studio 2005 或更高版本编译器编译 IBM Tivoli Directory Server 6.3 客户机库时可能会收到警告。

在其他分布式平台上编译出口

您可以使用以下片段在其他分布式平台上编译出口源，例如 Linux。 其他分布式平台上的某些编译器选项可能有所不同。

```
#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 同时提供静态和动态链接库，但只能使用其中一种形式的库。 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)
```

出口模块调用

可以使用三个不同的原因码来调用 PreConnect 出口模块。 本节更深入地描述了每个出口原因。

MQXR_INIT

使用 MQXR_INIT 原因码调用出口以初始化和建立与 LDAP 服务器的连接。

在 *MQXR_INIT* 调用之前，将使用 *mqclient.ini* 文件 (即 LDAP) 中 PreConnect 节中的数据属性填充 MQNXP 结构的 *pExitDataPtr* 字段。

LDAP URL 至少包含协议、主机名、端口号和搜索的基本 DN。 出口解析 *pExitDataPtr* 字段中包含的 LDAP URL ，分配 MQNLDAPCTX LDAP 查找上下文结构并相应地填充该结构。 此结构的地址存储在 *pExitUserAreaPtr* 字段中。 未能正确解析 LDAP URL 将导致错误 MQCC_FAILED。

此时，出口使用 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 原因码调用出口模块，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 *QMgrName* 参数包含特定队列管理器名称，那么搜索将返回其 *ibm-amqQueueManagerName* LDAP 属性值与给定队列管理器名称匹配的所有通道定义。

如果 *QMgrName* 参数为 "*" 或 " ' (空白) ，那么搜索将返回其 *ibm-amqIsClientDefault* 连接端点属性设置为 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

当要清除出口时，将使用此原因码来调用该出口。 在此期间，出口与 LDAP 服务器断开连接，释放出口分配和维护的所有内存。 这将包括 *MQNLDAPCTX* 结构，指针数组及其引用的每个 MQCD。 任何其他字段设置为缺省值。 *pQMgrName* 和 *ppConnectOpts* 出口参数在 *MQXR_TERM* 期间未使用，可能为 NULL。

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。 WebSphere MQ 客户机使用 LDAP 目录来获取连接定义。 LDAP 目录中 WebSphere MQ 客户机连接定义的结构称为 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。

LDAP 属性 中描述了每个属性的描述以及它们如何映射到正常 WebSphere MQ 属性。

LDAP 属性

定义的 LDAP 属性特定于 WebSphere MQ ，并直接映射到客户机连接属性。

WebSphere MQ 客户机通道目录字符串属性

下表列出了字符串属性及其到 WebSphere MQ 属性的映射。 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 *21: WebSphere MQ* 客户机通道目录字符串属性

LDAP 属性	描述	WebSphere 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	s bThe LU 6.2 方式名。	MODENAME
ibm-amqPassword	可以使用的密码。	密码
ibm-amqQueueManagerName	WebSphere MQ 客户机应用程序可以向其请求连接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	SSL 连接的单个 CipherSpec。	SSLCIPH
ibm-amqSslPeerName	从 WebSphere MQ 通道另一端的对等队列管理器或客户机检查证书的专有名称 (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

WebSphere 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 *22: WebSphere MQ* 客户机通道目录整数属性		
LDAP 属性	**描述**	**WebSphere 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

WebSphere MQ 客户机通道布尔值属性

此布尔属性未映射到任何 WebSphere MQ 属性。 此属性的语法表示布尔值。

表 *23: WebSphere 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。

WebSphere MQ 客户机通道列表属性

WebSphere MQ 属性作为单值逗号分隔列表属性存储在 LDAP 目录中。 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 WebSphere MQ 属性的映射。

表 *24: WebSphere MQ* 客户机通道列表属性		
LDAP 属性	**描述**	**WebSphere 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
ibm-amqReceiveExitName	通道接收用户出口运行的用户出口程序的名称。	RCVEXIT

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。

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。

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。

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。

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。

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。

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。

ibm-amqQueueManagerName
此属性指定 WebSphere MQ 客户机应用程序可以向其请求连接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。

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。

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。

ibm-amqSslCipherSpec
此 LDAP 属性为 SSL 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。

ibm-amqSslPeerName
此 LDAP 属性用于检查来自 WebSphere MQ 通道另一端的对等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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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** 指定的值是映射到 `NONE` 的 `0` 。 任何超出允许的最大限制的值都将被客户机忽略。 例如，ibm-amqHeaderCompression 在列表中包含最多 2 个整数。

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。

客户机将忽略任何超过允许的最大限制的值。 例如，ibm-amqMessageCompression 在列表中最多包含 16 个整数。

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。

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。

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。

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。

编写排队应用程序

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:

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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 73 页的『设计 IBM WebSphere MQ 应用程序』
当您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 WebSphere MQ 提供的功能。

第 78 页的『样本 WebSphere MQ 程序』
使用此主题集合来了解不同平台上的样本 WebSphere MQ 程序。

第 293 页的『编写客户机应用程序』
在 WebSphere MQ 上编写客户机应用程序所需的知识。

第 791 页的『在 WebSphere MQ 中使用 Web Service』
您可以使用 IBM WebSphere MQ Transport for SOAP 或 IBM WebSphere MQ Bridge for HTTP 为 Web Service 开发 IBM WebSphere MQ 应用程序。

第 357 页的『构建 IBM WebSphere MQ 应用程序』
使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

第 462 页的『处理程序错误』
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。

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

WebSphere MQ for Windows and WebSphere MQ on UNIX and Linux 系统还提供：

- 调用， WebSphere MQ for Windows 和 WebSphere 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 79 页的『分布式平台的样本程序』。
- 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 163 页的『MQI 调用』
- 第 164 页的『同步点调用』
- 第 164 页的『数据转换、数据类型、数据定义和结构』
- 第 165 页的『IBM WebSphere MQ 存根程序和库文件』
- 第 169 页的『所有调用的通用参数』
- 第 170 页的『指定缓冲区』
- 第 171 页的『UNIX and Linux 信号处理』

相关概念

第 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

MQI 调用

使用此信息可了解 MQI 中的调用。

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 WebSphere MQ 是工作单元的协调程序时，请使用这些调用。 MQBEGIN 启动工作单元。 MQCMIT 和 MQBACK 结束工作单元，落实或回滚在工作单元期间进行的更新。 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：

Windows ， UNIX 和 Linux 平台上的 IBM WebSphere MQ 调用

以下产品提供 MQCMIT 和 MQBACK 调用：

- IBM WebSphere MQ (对于 Windows)
- UNIX and Linux 系统上的 IBM WebSphere 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。 要落实和回退 CICS 环境中的更改，请使用 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

使用此信息来了解使用消息队列接口时的数据转换，基本数据类型， WebSphere MQ 数据定义和结构。

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。 除了在 WebSphere MQ for z/OS 上，此调用仅从数据转换出口使用。

请参阅 MQXCNVC - 转换字符以了解用于 MQXCNVC 调用的语法，并参阅第 346 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。

WebSphere MQ 数据定义

WebSphere MQ 随附的数据定义文件包含:

- 所有 WebSphere MQ 常量和返回码的定义
- WebSphere 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 WebSphere MQ 数据定义文件的完整描述，请参阅 第 66 页的『IBM WebSphere 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 163 页的『MQI 调用』中所列的 MQI 调用结合使用的结构。

请参阅结构数据类型摘要以获取结构的摘要。

IBM WebSphere 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 357 页的『构建 IBM WebSphere MQ 应用程序』。 有关链接到 C++ 库文件的信息，请参阅 使用 C++ *WebSphere MQ* 使用 *C++*。

IBM WebSphere MQ (对于 Windows)

在 IBM WebSphere MQ for Windows 上，除了操作系统提供的 MQI 库文件以外，还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:

表 *25: 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.lib	服务器 TXSeries CICS 支持 C (32 位)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib	客户机 TXSeries CICS C (32 位) 支持
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	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）

表 25: Windows 应用程序的库文件 (继续)	
库文件	**环境**
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Client for IBM COBOL (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	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	Client for IBM COBOL (64 位)
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 表示安装 WebSphere MQ 的高级目录。

使用 `amqmdnet.dll` 来编译 .NET 程序。 请参阅部分第 470 页的『使用 .NET』中的第 500 页的『编译 WebSphere 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

IBM WebSphere MQ 用于 AIX

在 IBM WebSphere MQ for AIX 上，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:

表 *26:* 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 & **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	WebSphere MQ Runtime Library for Micro Focus COBOL 支持
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:

表 *27:* 线程 *AIX* 应用程序的库文件	
库文件	**环境**
libmqm_r.a	针对 C 的服务器
libmqic_r.a & **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
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

IBM WebSphere MQ for HP-UX

在 IBM WebSphere MQ for HP-UX 上，除了操作系统提供的 MQI 库文件之外，还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。

IA64 (IPF) 平台

在非线程应用程序中:

表 *28:* 非线程 *HP-UX* 应用程序的库文件	
库文件	**环境**
libmqm.so	针对 C 的服务器
libmqic.so & **libmqm.so**	针对 C 的客户机

表 *28:* 非线程 *HP-UX* 应用程序的库文件 *(*继续*)*	
库文件	**环境**
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
libimqi23ah.so	C++
libmqmcbrt.o	WebSphere MQ Runtime Library for Micro Focus COBOL 支持
libmqmcb.so	针对 COBOL 的服务器
libmqicb.so	针对 COBOL 的客户机

在线程应用程序中：

表 *29:* 线程 *HP-UX* 应用程序的库文件	
库文件	**环境**
libmqm_r.so	针对 C 的服务器
libmqmzf_r.so & **libmqm_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
libimqi23ah_r.so	C++

IBM WebSphere MQ for Linux

在 IBM WebSphere MQ for Linux 上，除了操作系统提供的 MQI 库文件之外，还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。

在非线程应用程序中:

表 *30:* 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
libmqic.so & **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

表 *30:* 非线程 *Linux* 应用程序的库文件 *(*继续*)*	
库文件	**环境**
libimqs23gl.so	针对 C++ 的服务器

在线程应用程序中：

表 *31:* 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 & **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

IBM WebSphere MQ 适用于 *Solaris*

在 IBM WebSphere MQ for Solaris 上，除了操作系统提供的 MQI 库文件之外，还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。

表 *32: 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
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

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*。 对于 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。

要使程序能够使用 WebSphere MQ 对象，该程序必须具有用于识别该对象的唯一标识。 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：

- 针对所有其他 WebSphere MQ 平台的 原因码

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。

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：

UNIX and Linux 系统中的派生系统调用

在 IBM WebSphere 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 WebSphere MQ 调用 (例如，`MQCONN`) 或使用 **`ImqQueueManager`** 创建 IBM WebSphere MQ 对象之前调用 `fork` 。

如果应用程序希望在执行任何 IBM WebSphere 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 WebSphere MQ API 调用会返回 MQRC_ENVIRONMENT_ERROR。

UNIX and Linux 信号处理

这不适用于 WebSphere MQ for z/OS 或 WebSphere 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。

非线程应用程序
在 Solaris 上不适用，因为所有应用程序都被视为线程，即使它们只使用单个线程也是如此。

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）。

在 WebSphere MQ 处理同步信号 (SIGSEGV， SIGBUS， SIGFPE 和 SIGILL) 之后，它尝试在进行 MQI 函数调用之前将信号传递给任何已注册的信号处理程序。

线程应用程序
线程被视为从 MQCONN (或 MQCONNX) 连接到 WebSphere MQ，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，当任何线程连接到 WebSphere MQ 时， WebSphere MQ 会在应用程序进程中为以下信号设置自己的处理程序：

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。 当 WebSphere MQ 设置其自己的同步信号处理程序时，它会为每个信号保存任何先前注册的处理程序。 在 WebSphere MQ 处理列出的其中一个信号之后， WebSphere MQ 尝试调用在进程中的第一个 WebSphere MQ 连接时生效的信号处理程序。 当所有应用程序线程都已从 WebSphere MQ 断开连接时，将复原先前注册的处理程序。

由于信号处理程序由 WebSphere MQ 保存和复原，因此应用程序线程不得为这些信号建立信号处理程序，而同一进程的另一个线程也有可能连接到 WebSphere MQ。

注: 当应用程序或中间件库 (作为应用程序的一部分运行) 在线程连接到 WebSphere MQ 时建立信号处理程序时，应用程序的信号处理程序必须在处理该信号期间调用相应的 WebSphere 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：

- 当应用程序在连接到 WebSphere MQ 之后建立信号处理程序时，必须先复原先前的信号处理程序，然后应用程序才能断开与 WebSphere MQ 的连接。
- 当应用程序在连接到 WebSphere MQ 之前建立信号处理程序时，该应用程序必须先与 WebSphere MQ 断开连接，然后才能恢复其信号处理程序。

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

WebSphere MQ 不会在线程应用程序中使用任何异步信号，除非它们是客户机应用程序。

线程客户机应用程序的其他注意事项

WebSphere MQ 在对服务器进行 I/O 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）

其他考虑事项
使用 UNIX 信号处理时，请注意以下注意事项。

快速路径（可信）应用程序

Fastpath 应用程序在与 WebSphere MQ 相同的进程中运行，因此在多线程环境中运行。

在此环境中， WebSphere MQ 处理同步信号 SIGSEGV， SIGBUS， SIGFPE 和 SIGILL。 当快速路径应用程序连接到 WebSphere MQ 时，不得将所有其他信号传递到该应用程序。 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 175 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 WebSphere MQ 将像往常一样在同步点中回退未落实的消息，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。

在 AIX 环境中， WebSphere MQ 要求重新启动由信号中断的系统调用。 使用 sigaction (2) 建立自己的信号处理程序时，请在新操作结构的 sa_flags 字段中设置 SA_RESTART 标志，否则 WebSphere MQ 可能无法完成由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 WebSphere MQ 进程的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。 请考虑将这些消息永久连接到 WebSphere MQ，因此不使用信号或非线程安全的操作系统调用。

VMS 出口处理程序

用户可以使用 **SYS$DCLEXH** 系统服务为 WebSphere MQ 应用程序安装出口处理程序。

当图像退出时，出口处理程序会收到控制。 通常，调用 Exit ($EXIT) 或 Force Exit ($FORCEX) 服务时，会发生图像退出。 $FORCEX 在用户方式下中断目标进程。 然后，所有用户方式出口处理程序（由 $DCLEXH 建立）会开始按建立时采用的相反顺序执行。 有关出口处理程序和 $FORCEX 的更多详细信息，请参阅《VMS 编程概念手册》和《VMS 系统服务手册》。

如果在出口处理程序中调用 MQI 函数，那么函数的行为取决于终止图像的方式。 如果在其他 MQI 函数处于活动状态时终止了图像，那么会返回 MQRC_CALL_IN_PROGRESS。

如果没有其他 MQI 函数处于活动状态并且对 WebSphere MQ 应用程序禁用了上行调用，那么可以从出口处理程序中调用 MQI 函数。 如果对 WebSphere MQ 应用程序启用了调用，那么它将失败，原因码为 MQRC_HCONN_ERROR。

通常，MQCONN 或 MQCONNX 调用的作用域为发出此调用的线程。 如果启用了上行调用，那么出口处理程序会作为单独线程运行，且无法共享连接句柄。

将在目标进程的中断上下文中启动出口处理程序。 应用程序负责确保针对从其调用的异步中断上下文，处理程序所执行的操作安全且可靠。

连接到队列管理器和从队列管理器断开连接

要使用 WebSphere 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：

z/OS 批处理， WebSphere MQ for IBM i， WebSphere MQ on UNIX 系统， WebSphere MQ on Linux 系统以及 WebSphere MQ for Windows

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 批处理程序可以连续或并发地连接到同一 TCB 上的多个队列管理器。

IMS

IMS 控制区域在启动时连接到一个或多个队列管理器。 此连接由 IMS 命令控制。 但是，消息排队 IMS 程序的写程序必须使用 MQCONN MQI 调用来指定它们要连接到的队列管理器。 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。

IMS 程序可以连续或并发地连接到同一 TCB 上的多个队列管理器。

CICS Transaction Server for z/OS 和 CICS for MVS/ESA

CICS 程序不需要执行任何工作来连接到队列管理器，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 CKQC 事务随 WebSphere MQ for z/OS 提供。

CICS 任务只能连接到 CICS 区域本身所连接的队列管理器。

注: CICS 程序还可以使用 MQI 连接和断开连接调用 (MQCONN 和 MQDISC)。 您可能想要执行此操作，以便可以将这些应用程序移植到非 CICS 环境中，并进行最低限度的重新编码。 但是，这些调用 始终 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。

TXSeries for Windows 和开放式系统

这些程序不需要执行任何工作来连接到队列管理器，因为 CICS 系统本身已连接。 因此，一次仅支持一个连接。 CICS 应用程序在退出之前必须发出 MQCONN 调用以获取连接句柄和 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 174 页的『使用 MQCONN 调用连接到队列管理器』
- 第 175 页的『使用 MQCONNX 调用连接到队列管理器』
- 第 179 页的『使用 MQDISC 从队列管理器断开程序连接』

相关概念

第 163 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

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：

- 对于 IBM WebSphere 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。
- 对于 Windows， IBM i， UNIX 和 Linux 系统的 IBM WebSphere MQ ，将在 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，可以连接到队列共享组中的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 WebSphere 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 IBM WebSphere MQ for z/OS 时，将在 CSQQDEFV 模块中指定缺省队列管理器。

对于 TXSeries CICS 环境以及 Windows 和 AIX 的 TXSeries ，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，请参阅安全性。

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 181 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 WebSphere 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，一次只能从一个线程建立一个 WebSphere MQ 连接。 或者，请考虑 第 178 页的『使用 MQCONNX 的共享（独立于线程）连接』 以允许使用来自单个线程的多个 WebSphere MQ 连接以及来自任何线程的 WebSphere MQ 连接。[1]

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。

作为 MQCONNX 的输入，您可以在 z/OS 共享队列系统上提供队列管理器名称或队列共享组名称。 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。 *Options* 字段允许您为任何版本的 MQCNO 设置 STANDARD_BINDING， FASTPATH_BINDING， SHARED_BINDING 或 SOLATED_BINDING。 还可使用 MQCONNX 调用执行共享（独立于线程的）连接。 请参阅第 178 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 (如 MQCONN) 意味着两个逻辑线程，其中 WebSphere MQ 应用程序和本地队列管理器代理程序在不同的进程中运行。 WebSphere MQ 应用程序请求 WebSphere MQ 操作，本地队列管理器代理程序为该请求提供服务。 这由 MQCONNX 调用上的 MQCNO_STANDARD_BINDING 选项定义。

如果指定 MQCNO_STANDARD_BINDING，那么 MQCONNX 调用将使用 MQCNO_SHARED_BINDING 或 MQCNO_ISATED_BINDING，具体取决于在 qm.ini 或 Windows 注册表中定义的队列管理器的 DefaultBindType 属性的值。

这是缺省值。

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，那么可以提供下列其中一个选项来更改 MQCONN 或 MQCONNX 的行为 (如果指定了 MQCNO_STANDARD_BINDING)。 （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。

[1] 将多线程应用程序与 UNIX and Linux 系统上的 IBM WebSphere 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。

值	含义
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 WebSphere MQ 应用程序与本地队列管理器代理程序成为同一进程。 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。

您需要将可信应用程序链接到线程 WebSphere MQ 库。 有关如何设置 WebSphere MQ 应用程序以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：

- 在 z/OS 上，使用 MQRC_OPTIONS_ERROR 拒绝 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。
- V 7.5.0.7 在 IBM WebSphere MQ Version 7.5.0 修订包 7 之前，MQCNO_CLIENT_BINDING 不适用于 .NET，因为它具有其自己的选择绑定类型的机制。 从 Version 7.5.0, Fix Pack 7 开始，移除了对 MQCNO_CLIENT_BINDING 使用 .NET 的限制。
- 如果在调用 MQCONNX 时未设置 MQ_CONNECT_TYPE 环境变量，那么将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：

- 在 z/OS 上，将忽略 MQCNO_LOCAL_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。
- V 7.5.0.7 在 IBM WebSphere MQ Version 7.5.0 修订包 7 之前，MQCNO_LOCAL_BINDING 不适用于 .NET，因为它具有其自己的选择绑定类型的机制。 从 Version 7.5.0, Fix Pack 7 开始，将除去对 MQCNO_LOCAL_BINDING 使用 .NET 的限制。
- 如果在调用 MQCONNX 时未设置 MQ_CONNECT_TYPE 环境变量，那么将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。

在 z/OS 上，允许使用这些选项，但仅执行标准绑定连接。 针对 z/OS 的 MQCNO V 3 允许四个备用选项：

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。 这是与标记所有者位于同一队列共享组中的 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。
- 在 UNIX and Linux 系统上的 WebSphere MQ 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。
- 在 WebSphere MQ for HP-UX 上，多线程快速路径应用程序可能需要设置大于缺省值的堆栈大小。 使用 256 KB 大小。
- 在 WebSphere MQ 上，不支持 Windows 可信 64 位应用程序。 如果尝试运行可信 64 位应用程序，那么会降级为标准绑定连接。
- 在 UNIX and Linux 系统上的 WebSphere MQ 上，不支持可信的 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: 在 WebSphere MQ for z/OS 上不受支持。

在除 WebSphere MQ for z/OS 以外的 WebSphere MQ 平台上，使用 MQCONN 建立的连接仅对建立该连接的线程可用。 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。

除 MTS (Microsoft Transaction Server) 环境外，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 178 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 调用选项如何与 MQ_CONNECT_TYPE 配合使用。

在 WebSphere MQ for IBM i， WebSphere MQ for Windows 和 WebSphere MQ on UNIX and Linux 系统上，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 *33: MQ_CONNECT_TYPE* 环境变量		
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，除非使用 MQCONNX，否则调用是可选的，并且您希望在应用程序结束之前删除连接标记。
- 在 WebSphere MQ for IBM i 上，当您从操作系统注销时，将执行隐式 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，并且在再次调用 MQCONN 或 MQCONNX 之前，不能再发出任何 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。

如果使用 MQCONNX 在 WebSphere MQ for z/OS 上进行连接，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，如果存在与连接标记关联的活动恢复单元，那么将拒绝 MQDISC，原因码为 MQRC_CONN_TAG_NOT_RELEASED。

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 WebSphere MQ 对象的作业将关闭该对象或与该对象断开连接。 即使已连接或打开 WebSphere MQ 对象的作业的权限被撤销，也会接受 MQCLOSE 和 MQDISC 调用。

打开和关闭对象

此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

要执行以下任何操作，必须首先 打开 相关 WebSphere 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 195 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 173 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。

您可以打开四种类型的 WebSphere 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 WebSphere MQ 对象的更多信息，请参阅 对象。

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 196 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，当程序在 GU (获取唯一) IMS 调用之后开始处理新用户时，将强制断开连接。 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 181 页的『使用 MQOPEN 调用打开对象』
- 第 186 页的『创建动态队列』
- 第 187 页的『打开远程队列』
- 第 187 页的『使用 MQCLOSE 调用关闭对象』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 188 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 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

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 175 页的『MQCONN 或 MQCONNX 的作用域』和第 178 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，只能在执行 MQOPEN 调用的同一 CICS 任务中使用该句柄。

IMS 和 z/OS 批处理
在 IMS 和批处理环境中，可以在同一任务中使用句柄，但不能在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 196 页的『分发列表』下的第 197 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 WebSphere MQ 队列时， MQOPEN 调用会对您指定的队列名称执行名称解析功能。 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 182 页的表 34，以获取名称解析过程的概述。 您在 *ObjectQMgrName* 中提供的名称在 *ObjectName* 中解析为 前 。

第 182 页的表 34 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 *34:* 使用 *MQOPEN* 时解析队列名称

MQOD 的输入		**解析名称**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**传输队列**
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）
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）

表 *34:* 使用 *MQOPEN* 时解析队列名称 *(*继续*)*

MQOD 的输入		**解析名称**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**传输队列**
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）
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名称。
2. *RemoteQName* 是远程队列的本地定义中的远程队列名称。
3. *RemoteQMgrName* 是远程队列的本地定义中的远程队列管理器名称。
4. *XmitQName* 是远程队列的本地定义中的传输队列名称。
5. 使用属于队列共享组 (QSG) 的 WebSphere MQ for z/OS 队列管理器时，可以使用 QSG 的名称来代替 第 182 页的表 34 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将消息放入队列，那么将通过组内队列或 WebSphere MQ 通道将消息传输到指定的 ObjectQMgr 名称。

6. 在表的 *ObjectName* 列中，CLUSTER 同时指队列的 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: 规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 290 页的『MQOPEN 和集群』以获取更多信息。

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，集群队列是使用 MQOO_BIND_NOT_FIXED 而不是 MQOO_BIND_ON_OPEN 打开的。 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 227 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 196 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 185 页的表 35，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 *35: 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**		
Shareability	*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：

- 如果知道即使其他程序可同时从队列移除消息也能成功运行应用程序，请使用 MQOO_INPUT_SHARED 选项。 第 185 页的表 35 说明在某些情况下，将如何为您指定对队列的专用访问权，甚至在使用此选项时也是如此。
- 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 196 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 268 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 32 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 193 页的『控制上下文信息』
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。

用于停顿队列管理器的 *MQOPEN* 选项
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 选项时，当队列管理器停顿时，调用才会失败。

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 189 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。 您可以使用 WebSphere MQ 命令或操作和控制面板来创建模型队列。 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。

动态队列和模型队列 中有更多有关动态队列的信息。

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。

这些方法在 第 182 页的表 34 中进行了概述。

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 WebSphere MQ for z/OS 上，如果您有任何 MQGET 请求具有针对该队列的未完成的 MQGMO_SET_SIGNAL 选项，那么将取消这些请求。
- 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。 使用用于打开它的相同连接句柄，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 189 页的『使用 MQPUT 调用将消息放置到本地队列上』
- 第 193 页的『将消息放置到远程队列上』
- 第 193 页的『设置消息的属性』
- 第 193 页的『控制上下文信息』
- 第 195 页的『使用 MQPUT1 调用将一条消息放置到队列上』
- 第 196 页的『分发列表』
- 第 200 页的『放置调用失败的一些情况』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

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 189 页的『指定句柄』
- 第 189 页的『使用 MQMD 结构定义消息』
- 第 190 页的『使用 MQPMO 结构指定选项』
- 第 192 页的『消息中的数据』
- 第 193 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 9 页的『IBM WebSphere 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 196 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，同步点控制缺省值为 yes; 对于其他平台，缺省值为 no。

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 32 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 193 页的『控制上下文信息』。

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 196 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 199 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 199 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 198 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
- WebSphere MQ 添加的任何消息头的大小 (包括死信头， MQDLH 和分发列表头 MQDH)

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 WebSphere MQ 产品，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。

WebSphere MQ 在以下情况下向消息添加头信息：

- 将消息放入远程队列时， WebSphere 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 WebSphere MQ 无法将消息传递到远程队列，那么它会尝试将消息放在死信 (undelivered-message) 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 WebSphere MQ 会向消息添加 MQDH 头。 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。
- 如果消息是段或组中的消息，那么 WebSphere MQ 可能会添加 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。

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 187 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 WebSphere MQ 分布式消息传递技术 。

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。

如果要将消息放入低于 WebSphere MQ 版本 7.0 的队列管理器 (无法处理消息属性) 上的队列，那么可以在通道定义中设置 PropertyControl 参数以指定如何处理属性。

控制上下文信息

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。

要控制上下文信息，请在 MQPMO 结构中使用 *Options* 字段。

如果不存在，那么队列管理器将使用它为消息生成的身份和上下文信息来覆盖可能已存在于消息描述符中的上下文信息。 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 WebSphere MQ 执行的任何权限检查。 无法为空用户标识分配对 IBM WebSphere 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。

如果不需要与消息关联的上下文信息，请使用 MQPMO_NO_CONTEX 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 194 页的『传递身份上下文』
- 第 194 页的『传递用户上下文』
- 第 194 页的『传递所有上下文』
- 第 194 页的『设置身份上下文』
- 第 194 页的『设置用户上下文』
- 第 195 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 32 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 186 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 181 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 (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。

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 190 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 WebSphere 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。

V 7.5.0.8

注意: 分发列表不支持使用指向主题对象的别名队列。 从 Version 7.5.0, Fix Pack 8 开始，如果别名队列指向分发列表中的主题对象，那么 IBM WebSphere 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 197 页的图 29 显示分发列表如何工作。

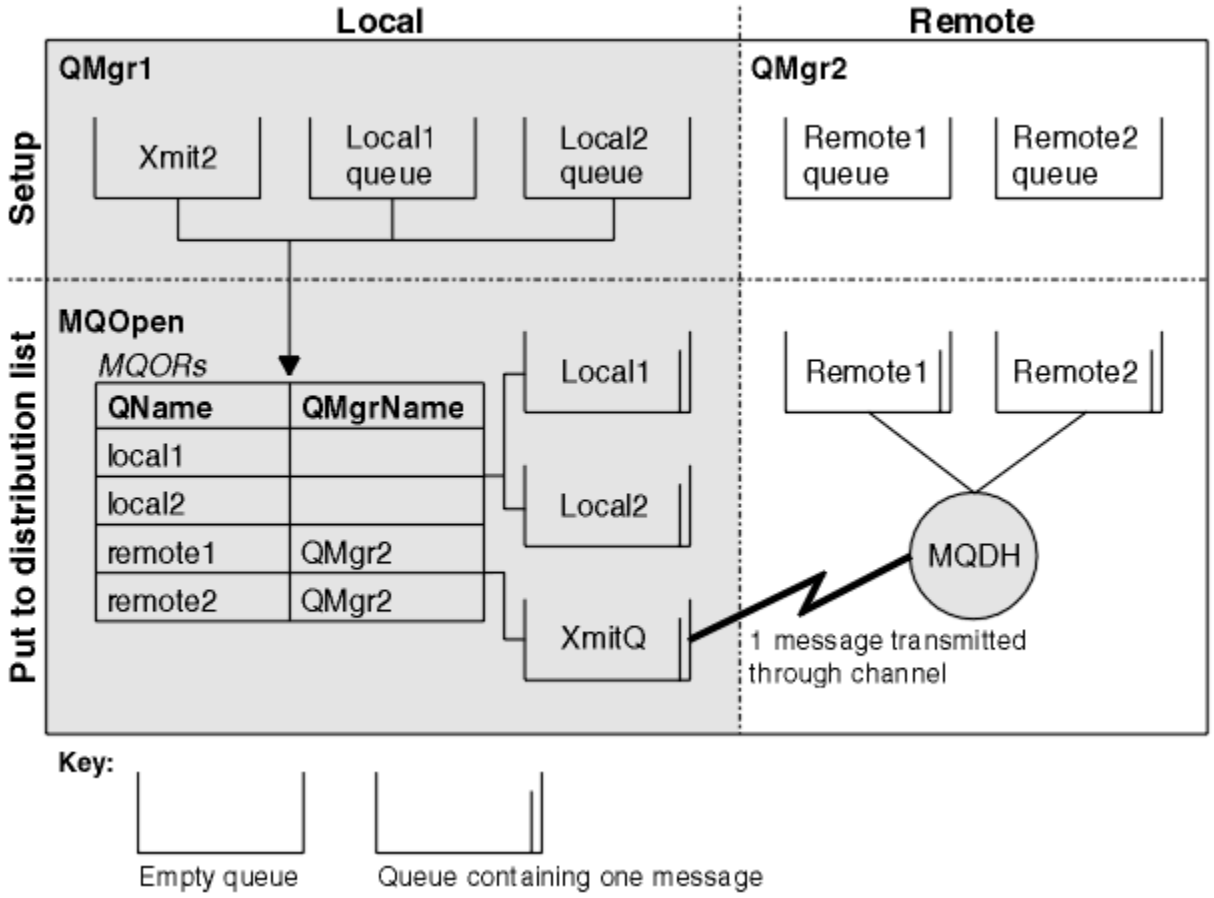


图 *29:* 分发列表如何工作

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 188 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*。

无论您选择哪种技术，都必须使用 *ObjectRecOffset* 和 *ObjectRecPtr* 之一; 如果两者都为零或两者都为非零，那么调用将失败，原因码为 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 199 页的图 30 说明您如何在 C 中打开分发列表。

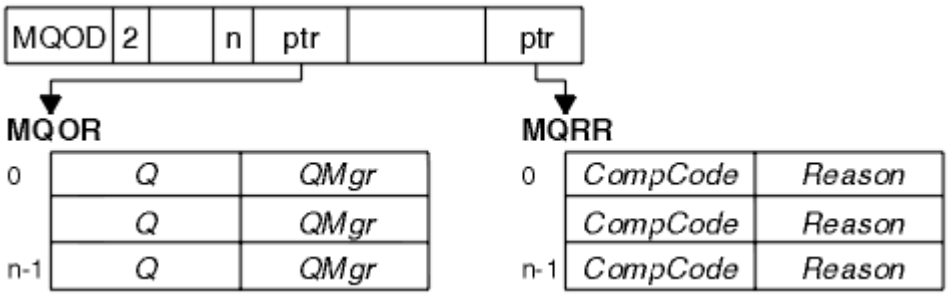


图 *30:* 在 *C* 中打开分发列表

第 199 页的图 31 说明您如何在 COBOL 中打开分发列表。

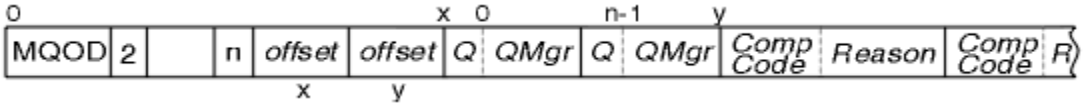


图 *31:* 在 *COBOL* 中打开分发列表

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 179 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 188 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 190 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 104 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 200 页的图 32 说明您如何在 C 中将消息放置到分发列表中。

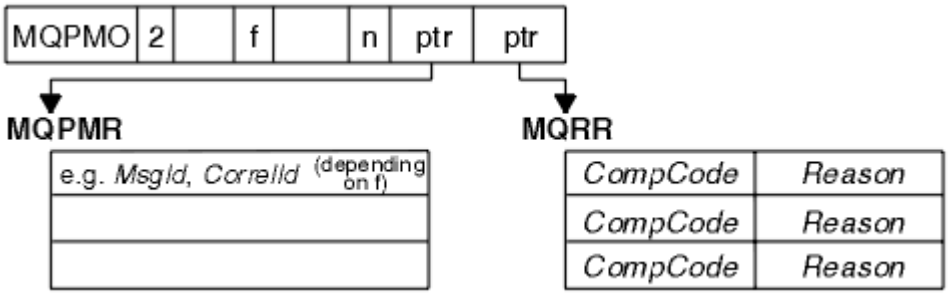


图 *32:* 在 *C* 中将消息放置到分发列表中

第 200 页的图 33 说明您如何在 COBOL 中将消息放置到分发列表中。

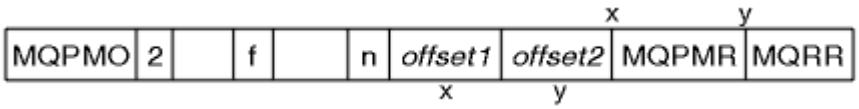


图 *33:* 在 *COBOL* 中将消息放置到分发列表中

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 173 页的『连接到队列管理器和从队列管理器断开连接』和第 179 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 202 页的『使用 MQGET 调用从队列取出消息』
- 第 205 页的『从队列检索消息的顺序』
- 第 214 页的『获取特定消息』
- 第 216 页的『提高非持久消息的性能』
- 第 219 页的『处理消息长度大于 4 MB 的消息』
- 第 223 页的『等待消息』
-
- 第 224 页的『跳过回退』
- 第 226 页的『应用程序数据转换』
- 第 227 页的『浏览队列中的消息』
- 第 231 页的『MQGET 调用失败的一些情况』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

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 202 页的『指定连接句柄』
- 第 202 页的『使用 MQMD 结构和 MQGET 调用描述消息』
- 第 203 页的『使用 MQGMO 结构指定 MQGET 选项』
- 第 204 页的『指定缓冲区的大小』

指定连接句柄

对于 z/OS 应用程序上的 CICS ，您可以指定常量 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 9 页的『IBM WebSphere 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 214 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 205 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 223 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
- 在 WebSphere MQ for z/OS 上，是否将检索到的消息标记为跳过回退 (请参阅 第 224 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
- 在 WebSphere MQ for z/OS 上，是否允许调用完成。 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
- 在 WebSphere MQ for z/OS 上，如果连接处于停顿状态，那么调用是否失败
- 是否需要转换应用程序消息数据（请参阅第 226 页的『应用程序数据转换』）
- 从队列中检索消息和 (WebSphere MQ for z/OS 除外) 段的顺序
- 除非在 WebSphere MQ for z/OS 上，否则仅可检索完整的逻辑消息
- 是否只能在组中的所有消息均可用时检索该组中的消息。
- 除了在 WebSphere MQ for z/OS 上，仅当逻辑消息中的 所有 段都可用时，才能检索逻辑消息中的段

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。 此外，在 WebSphere MQ for z/OS 中，调用不会在此类消息到达时设置请求通知的信号。
- 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	Yes
Windows 系统	否

- 在 WebSphere MQ for z/OS 上，未将检索到的消息标记为跳过回退。
- 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* 中指定的时间内无消息到达，那么将完成调用并返回原因码，显示无消息与队列上的选择标准匹配。

在 WebSphere 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 223 页的『等待消息』 。

Signal1
Signal1 在 WebSphere MQ for z/OS 和 MQSeries for HP Integrity NonStop Server only 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 WebSphere MQ 中， *Signal1* 字段是保留字段，其值不重要。

Signal2
Signal2 字段在所有平台上均为保留字段，其值并不重要。

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 219 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 15 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 214 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。

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 30 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 206 页的图 34：

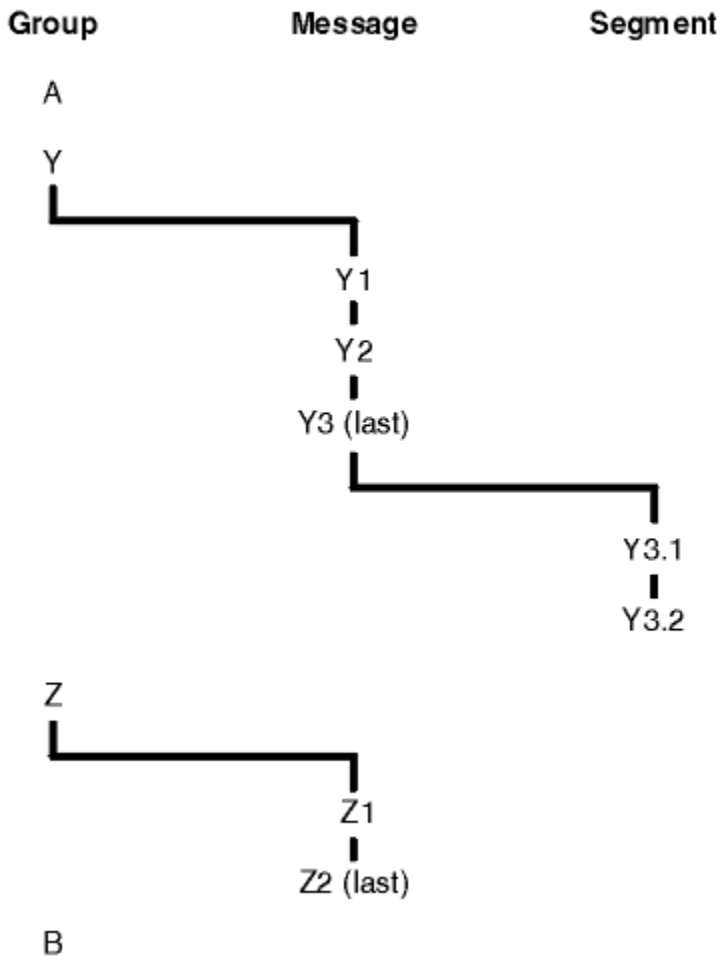


图 *34:* 队列上的逻辑顺序

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 207 页的图 35:

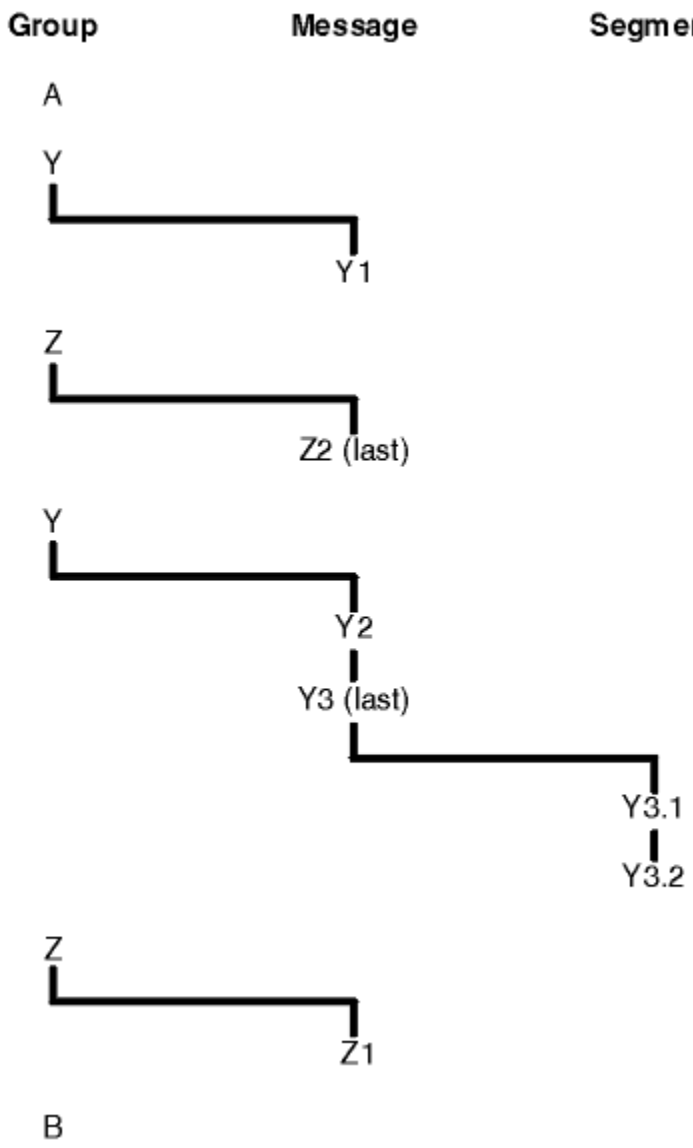


图 *35:* 队列上的物理顺序

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 WebSphere 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 WebSphere MQ 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 207 页的图 35 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 *36:* 与逻辑消息的组和段中的消息相关的 *MQPUT* 选项

指定的选项				调用前的组和 log-msg 状态		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

表 *36:* 与逻辑消息的组和段中的消息相关的 *MQPUT* 选项 *(*继续*)*

指定的选项				调用前的组和 log-msg 状态		队列管理器使用的值		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	***GroupId***	***MsgSeqNumber***	***Offset***

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 211 页的表 37 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 *37: 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
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，与 MQPMO_LOGICAL_ORDER 选项提供的控制相比，专用应用程序可能需要更多的控制，可以通过不指定该选项来实现; 如果执行此操作，那么必须确保在每个 MQPUT 或 MQPUT1 调用之前正确设置 MQMD 中的 *GroupId*， *MsgSeqNumber*， *Offset* 和 *MsgFlags* 字段。

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 30 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 221 页的『逻辑消息的应用程序分段』和第 212 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。 如果涉及分段，那么适用相同的原则 (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 215 页的表 38 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 *38:* 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
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 215 页的表 39 显示针对这些字段的可能设置检索的消息。

表 *39:* 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 205 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。

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 19 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。

使用 MQOO_READ_AHEAD 调用 MQOPEN 时，仅当满足特定条件时， WebSphere MQ 客户机才会启用预读。 这些条件包括：

- 客户机和远程队列管理器都必须处于 WebSphere MQ V7 或更高版本。
- 必须针对线程 WebSphere 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。 此性能改进可用于 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。

当预订应用程序请求 WebSphere MQ 管理要向其发送发布的目标时，会根据预定义的模型队列将受管队列创建为动态队列。 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 218 页的『在 AIX 上调整非持久消息的性能』
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 217 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 216 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。

使用 MQOO_READ_AHEAD 调用 MQOPEN 时，仅当满足特定条件时， WebSphere MQ 客户机才会启用预读。 这些条件包括：

- 客户机和远程队列管理器都必须处于 WebSphere MQ V7 或更高版本。
- 必须针对线程 WebSphere 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 *40: MQGET* 选项和预读

	在启用预读时允许，并且可以在 MQGET 调用之间进行更改 [5]	**在启用预读但无法在 MQGET 调用之间进行更改时允许** [1]	**启用预读时不允许的 MQGET 选项** [2]
MQGET MQMD 值	消息标识 [3] CorrelId[3]	编码 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_MSG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR[4] • MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

注意：

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

使用 MQOO_READ_AHEAD 调用 MQOPEN 时，仅当满足特定条件时， WebSphere MQ 客户机才会启用预读。 这些条件包括：

- 客户机和远程队列管理器都必须处于 WebSphere MQ V7 或更高版本。

- 必须针对线程 WebSphere 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。

在 *AIX* 上调整非持久消息的性能

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，在使用多线程应用程序时（尤其在具有多个处理器的机器上运行时），强烈建议在启动应用程序前在 mqm 标识 `.profile` 中设置 AIXTHREAD_SCOPE=S，或在环境中设置 AIXTHREAD_SCOPE=S，以实现更高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。 此限制为您提供 IBM WebSphere MQ 原因码 MQRC_UNEXPECTED_ERROR （2195），如果有足够的文件描述符，那么还提供 IBM WebSphere 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
```

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。 根据环境， WebSphere MQ 提供了多种处理长度超过 4 MB 的消息的方法。

在 V6 或更高版本的所有 WebSphere MQ 系统上，可以将 *MaxMsgLength* 属性增大到 100 MB。 请设置此值，以反映使用队列的消息的大小。 在除 WebSphere MQ for z/OS 以外的 WebSphere MQ 系统上，您还可以：

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；可设置在 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。

消息分段

使用此信息来了解消息分段。

注: 在 IBM WebSphere MQ for z/OS 或使用 IBM WebSphere MQ classes for JMS 的应用程序中不受支持。

按第 219 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 30 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 212 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。

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
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。

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
```

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: 在 WebSphere MQ for z/OS 中不受支持。

此方法允许将大对象从一个节点传输到另一个节点，而不将该对象存储在源节点或目标节点上的 WebSphere MQ 队列上。 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 343 页的『通道消息出口程序』。

WebSphere MQ 定义参考消息头 (MQRMH) 的格式。 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 WebSphere 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 是 在中间节点的 WebSphere MQ 队列上存在批量数据。 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。 如果是这样，那么消息出口必须保留已应用更新的日志; 这可能需要使用 WebSphere 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了字段 (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。 在 MQGET 上请求时，WebSphere MQ 会自动转换 MQHREF 格式以及后续的任何批量数据。

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 79 页的『分布式平台的样本程序』。

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制的等待时间间隔，因为这将阻止 IMS 系统终止。 (当 IMS 终止时，它需要所有从属区域结束。) 相反， IMS 应用程序可以指定有限等待时间间隔; 然后，如果调用完成而未在该时间间隔之后检索消息，请发出另一个带有等待选项的 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。
- 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。

有关发生这些操作的情况的更多信息，请参阅 MQGMO。

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 WebSphere 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 WebSphere 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. 在参与由 RRS 控制的工作单元的 DB2 存储过程中，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 225 页的图 36 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。

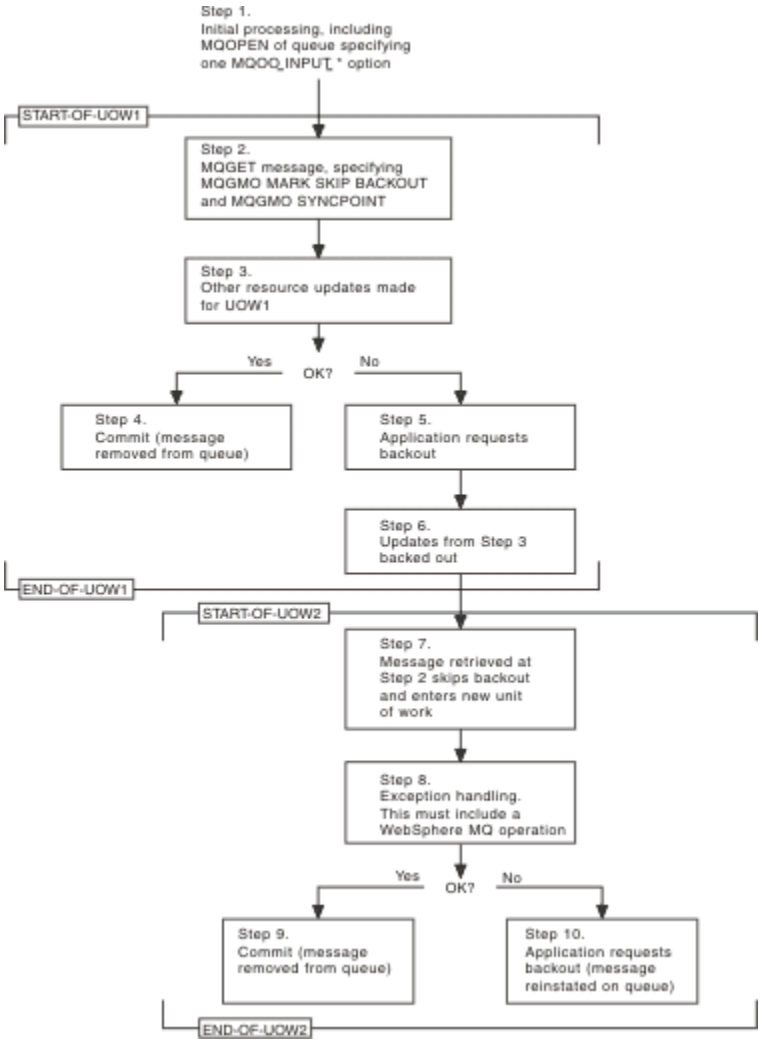


图 *36:* 使用 *MQGMO_MARK_SKIP_BACKOUT* 跳过回退

第 225 页的图 36 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 225 页的图 36 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，并且以下所有条件均为真时，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 346 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。

您可以使用依赖于平台的交换机来执行此操作，该交换机强制 WebSphere MQ 使用未修改的转换表，但您必须了解可能导致的不一致行为。

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 205 页的『从队列检索消息的顺序』。）

- 第 227 页的『FIFO（先入先出）序列中的队列』
- 第 227 页的『优先序列中的队列』
- 第 228 页的『未落实的消息数』
- 第 228 页的『队列序列的更改』
- 第 228 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 WebSphere 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。）

要移除消息，请再次调用 MQGET，但应在 MQGMO 结构的 *Options* 字段中指定 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 205 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 229 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 WebSphere MQ 应用程序。

有关发布/预订概念的概述，请参阅 WebSphere MQ 发布/预订消息传递简介。

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 232 页的『编写发布者应用程序』
- 第 239 页的『编写订户应用程序』
- 第 255 页的『发布/预订生命周期』
- 第 258 页的『发布/预订消息属性』
- 第 260 页的『消息排序』
- 第 260 页的『拦截发布内容』
- 第 267 页的『发布选项』
- 第 267 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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 73 页的『设计 IBM WebSphere MQ 应用程序』
当您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 WebSphere MQ 提供的功能。

第 78 页的『样本 WebSphere MQ 程序』
使用此主题集合来了解不同平台上的样本 WebSphere MQ 程序。

第 162 页的『编写排队应用程序』
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 293 页的『编写客户机应用程序』
在 WebSphere MQ 上编写客户机应用程序所需的知识。

第 791 页的『在 WebSphere MQ 中使用 Web Service』
您可以使用 IBM WebSphere MQ Transport for SOAP 或 IBM WebSphere MQ Bridge for HTTP 为 Web Service 开发 IBM WebSphere MQ 应用程序。

第 357 页的『构建 IBM WebSphere MQ 应用程序』
使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

第 462 页的『处理程序错误』
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。

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。

编写简单的 WebSphere MQ 发布程序应用程序就像编写将消息放入队列 (第 233 页的表 41) 的 WebSphere MQ 点到点应用程序一样。 不同之处在于您将 MQPUT 消息发送到主题，而不是发送到队列。

表 *41:* 点到点与发布/预订 *WebSphere 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 233 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 236 页的『示例 2：可变主题的发布者』) 中，程序与 WebSphere MQ 的交互模式类似。 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 239 页的『编写订户应用程序』
通过研究三个示例来开始编写订户应用程序: 一个使用来自队列的消息的 WebSphere MQ 应用程序，一个创建预订且不需要排队知识的应用程序，最后一个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

WebSphere MQ 程序，用于说明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 234 页的图 38 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 *37:* 固定主题的简单 *WebSphere MQ* 发布程序。

```
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 *38:* 第一个发布者示例的样本输出

以下所选代码行说明了为 WebSphere MQ 编写发布程序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 WebSphere MQ Explorer 或此 MQSC 命令预先创建主题对象。

```
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。 设置 WebSphere MQ 的 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);
将 1 添加到传递到 MQPUT 的字符串的长度，以将空终止字符作为消息缓冲区的一部分传递到 WebSphere MQ 。

第一个示例演示什么？ 该示例尽可能紧密地模仿用于编写点到点 WebSphere MQ 程序的已尝试和已测试的传统模式。 WebSphere MQ 编程模式的一个重要功能是程序员不关心发送消息的位置。 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。 内置到 WebSphere MQ 发布/预订模型的体系结构中是提供主题的根部分或根子树的管理主题对象的概念。 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 236 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 236 页的『示例 2：可变主题的发布者』
这个 WebSphere MQ 程序演示如何发布到通过程序定义的主题。

第 239 页的『编写订户应用程序』
通过研究三个示例来开始编写订户应用程序: 一个使用来自队列的消息的 WebSphere MQ 应用程序，一个创建预订且不需要排队知识的应用程序，最后一个同时使用排队和预订的示例。

示例 2：可变主题的发布者

这个 WebSphere MQ 程序演示如何发布到通过程序定义的主题。

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 238 页的图 40 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 *39:* 变量主题的简单 *WebSphere MQ* 发布程序。

```
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 *40:* 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。 WebSphere MQ 使用已解析主题字符串中隐式的树来确定哪个管理主题对象定义了与发布相关联的属性。

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 238 页的图 41)。 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 238 页的图 40 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 WebSphere 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。

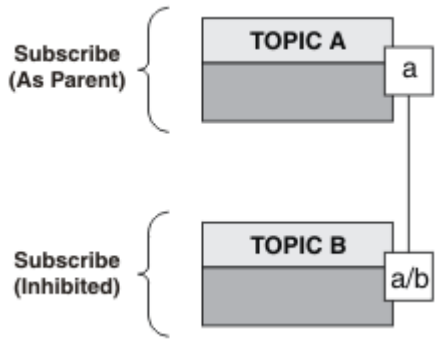


图 *41:* 主题对象关联

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，虽然 WebSphere MQ 使用其底层排队系统来实现发布/预订，但它们并不存在; 与消息传递和排队相关联的保证传递，事务性和松耦合的优点由发布/预订应用程序继承。

设计者必须决定发布者和订户程序是否知道底层主题树，以及订户程序是否知道排队。 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 233 页的『示例 1：固定主题的发布者』
WebSphere MQ 程序，用于说明如何发布到以管理方式定义的主题。

第 239 页的『编写订户应用程序』
通过研究三个示例来开始编写订户应用程序: 一个使用来自队列的消息的 WebSphere MQ 应用程序，一个创建预订且不需要排队知识的应用程序，最后一个同时使用排队和预订的示例。

编写订户应用程序

通过研究三个示例来开始编写订户应用程序: 一个使用来自队列的消息的 WebSphere MQ 应用程序，一个创建预订且不需要排队知识的应用程序，最后一个同时使用排队和预订的示例。

在 第 239 页的表 42 中，列出了三种样式的使用者或订户，以及描述其特征的 WebSphere MQ 函数调用的序列。

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。 导致发布路由到队列的预订是使用 WebSphere MQ Explorer 或命令以管理方式创建的。
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。

了解这些样式的一种方法是研究 第 239 页的表 42 中针对每个样式列出的示例 C 程序。 这些示例设计成与 第 232 页的『编写发布者应用程序』中的发布者示例配合运行。

表 *42:* 点到点与预订 *WebSphere MQ* 程序模式。

步骤	MQ 消息使用者	第 239 页的『示例 1：MQ 发布内容使用者』	第 242 页的『示例 2: 受管 MQ 订户』	第 247 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。 由于在受管 MQ 订户案例中关闭预订队列时不太可能需要指定 MQCLOSE 选项，并且对称性自变量不相关，因此在 示例 2: 受管 MQ 订户 中未显式关闭预订队列。

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 255 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 WebSphere 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队列，请运行以下命令，或者使用 WebSphere MQ Explorer 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 247 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 241 页的图 43 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 *42: 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 *43: MQ* 发布内容使用者的输出

有几个标准 WebSphere MQ *C* 语言编程提示需要注意:

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 WebSphere 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空值，以确保返回的消息在 "if (messlen == strlen(publication));" 为 true 的情况下具有尾部空值。 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 242 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。 此示例需要队列， 主题 或 预订的 *no* 管理定义。

第 247 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 232 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。 此示例需要队列， 主题 或 预订的 *no* 管理定义。

受管订户的这种最简单的 类型 通常使用 非持久 预订。 此示例侧重于非持久预订。 只要来自 MQSUB 的预订句柄的生存期，预订 仅 持续。 在预订生存期内与主题字符串匹配的任何 发布都将发送到预订队列 (如果未设置或缺省设置了标志 MQSO_NEW_PUBLICATIONS_ONLY，保留了与主题字符串匹配的较早发布，并且该发布是持久发布，或者自创建发布以来队列管理器未终止，那么可能是保留发布)。

另外，还可以将持久预订与此模式配合使用。 通常，如果使用了受管持久预订，那么执行此操作是出于可靠性原因，而不是为了建立不会发生任何错误的预订，从而使订户处于活动状态。 有关与受管，非受管 持久 和 非持久预订相关联的不同生命周期的更多信息，请参阅相关主题部分。

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. 在 上，需求预订: 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 预订仅在订户应用程序的持续时间内存在。
   - 如果定义 持久 受管预订，那么会导致永久预订队列和发布继续存储在该队列上，而没有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。
   - 对于持久预订，请考虑 设置 SubExpiry，以便停止将发布发送到队列，并且订户可以在除去预订之前使用任何剩余的发布，并使队列管理器删除该队列及其上的任何剩余发布。
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，可以部署应用程序，而无需应用程序无意中创建与另一个实例或另一个应用程序创建的另一个主题树重叠的主题树。
5. 受管主题: 通过 使用第一部分与管理定义的主题对象匹配的主题字符串，将根据主题 对象的属性来管理发布。
   - 例如，对于，如果主题字符串的第一部分与与集群主题对象相关联的主题字符串相匹配，那么预订可以从集群的其他成员接收发布

- 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。管理员为主题提供属性，程序员动态定义 "sub-主题" 而不关心主题管理。
- 它 是生成的主题字符串，用于匹配提供与主题关联的属性的主题对象，而不一定是 sd.Objectname 中指定的主题对象，尽管它们通常是同一。 请参阅 第 236 页的『示例 2：可变主题的发布者』。

通过在示例中 使预订持久，在订户使用 MQCO_KEEP_SUB 选项关闭预订之后，将继续向预订队列发送发布内容。 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。 您 可以通过按主题提供自己的持久和非持久模型队列作为与预订关联的主题对象的属性来覆盖缺省值。

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。 该队列的行为就像已针对互斥 输入打开该队列一样。

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（第 247 页的『示例 3：非受管 MQ 订户』）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 245 页的图 46 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 *44:* 受管 *MQ* 订户部分 *1:* 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;
无法显式打开非持久受管预订队列以接收发布内容，但您需要为队列管理器在为您打开队列时返回的对象句柄分配存储器。 将句柄初始化为 MQHO_OBJECT 很重要。 此 向队列管理器指示它需要将队列句柄返回到预订队列。

MQSD sd = {MQSD_DEFAULT};
MQSUB 中使用的新预订描述符。

MQCHAR48 qName;
虽然 示例 需要 n't 了解预订队列，但该示例会查询预订队列的名称- MQINQ 绑定在 C 语言中有点尴尬，因此您可能会发现示例的此部分对研究很有用。

```
    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 *45:* 受管 *MQ* 订户-部件 *2:* 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 *46:* 来自受管 *MQ* 订户的输出

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。 有许多选项， 示例仅指定最常用的选项; 其他选项保留为 缺省值 。

1. 由于预订是 非持久的，即，它在应用程序中具有开放式预订的生存期，请设置 MQSO_CREATE 标志。 您还可以设置 *(缺省值)* MQSO_NON_DURABLE 标志以实现可读性。
2. MQSO_RESUME 是对 MQSO_CREATE 的补充。 可以同时设置这两个标志; 队列管理器 创建 新 预订或恢复现有预订 (以适当者为准)。 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。 给定正确的许可权，已恢复的预订的属性将更改为与 MQSUB 中指定的其他属性相匹配。

   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 247 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 WebSphere MQ 设置 MQCHARV ResObjectString 以将解析后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。 此 防止覆盖所提供的空终止符，以防解析的主题字符串填充整个缓冲区。

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。 如果它是非持久的，那么您可能对它的名称不感兴趣，尽管您可以在 WebSphere MQ Explorer 中检查它的状态。 您可以提供 sd.SubName 参数作为 输入，因此您知道要查找的名称; 显然必须避免与其他预订发生名称冲突。

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 239 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 WebSphere MQ 消息使用者，它不预订主题本身。

第 247 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 232 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。 通过使预订持久并让发布者创建持久发布内容，可以避免在队列管理器或订户过早终止并需要恢复时丢失消息。

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 252 页的图 49 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。 所有预订都显示在 WebSphere MQ Explorer 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。
  - 或者，通过对特定选项 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER 进行编码，如果作为 MQSUB 的输入提供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 248 页的表 43)。 第 254 页的图 55 显示针对具有 sd.Options 标志的不同单独设置的 Subscription X 发出 MQSUB 的结果，并将其传递给三个不同的对象句柄。

  请探查第 251 页的图 48 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。 通常是由于使用 MQSO_RESUME 或 MQSO_ALTER 传递空或无效的预订名称而导致的。
- 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。 请谨慎使用此功能，因为这可能会导致 "重叠" 预订，并多次接收同一发布内容。 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。 它是非受管 MQ 订户的变体，但一个程序预订发布，另一个程序使用发布，而不是在同一程序中发出 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。

了解 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER 的行为很重要，尤其是当您计划通过使用这些选项的组合来简化代码时。 请研究第 248 页的表 43，此表显示了将不同队列句柄传递到 MQSUB 的结果以及运行第 253 页的图 50 到第 254 页的图 55 所示示例程序的结果。

用于构造表的方案有一个预订 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 248 页的表 43 中， MQSUB 将预订 X 和队列句柄传递到队列 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，如果队列 A 上不存在预订 X ，那么调用将失败。
- 在打开预订和队列方面， MQSO_ALTER 的行为方式与 MQSO_RESUME 类似。 但是，如果传递到 MQSUB 的预订描述符中包含的属性与预订的属性不同，那么 MQSO_RESUME 将失败，而 MQSO_ALTER 将成功，只要程序实例有权更改这些属性。 请注意，您永远无法更改预订中的主题字符串; 但是， MQSUB 将忽略预订描述符中的主题名称和主题字符串值，并使用现有预订中的值，而不是返回错误。

接下来，查看 第 248 页的表 43，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 成功并在队列 B 上创建预订 X ，因为这是队列 B 上的新预订。
- MQSO_RESUME 将失败。 MQSUB 在队列 B 上查找预订 X ，但不返回 *RC = 2428*-预订 *X* 不存在 ，而是返回 *RC = 2019*-预订队列与队列对象句柄不匹配。 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。 该调用失败，返回 *RC = 2428*-队列 *A* 上不存在预订 X。

表 *43:* 使用不同的队列句柄与预订组合时 *MQSUB* 返回的错误

	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 *43:* 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 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 *47:* 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 *48:* 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 Alter | C reate | Resume，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为 MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE。

注: 设置 MQSO_ALTER 或 MQSO_RESUME 而不设置 MQSO_DURABLE 是错误，必须设置 sd.SubName 并引用可恢复或更改的预订。

***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;
如果将缺省预订 IBMSTOCKPRICESUB 替换为空字符串，那么此示例将除去 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 *49:* 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称，那么示例将使用 MQHO_NONE 作为 Hobj 的值。

MQOPEN(...);
预订队列已打开，并且队列句柄保存在 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。 根据预订是否为持久预订，将使用隐式 MQCO_KEEP_SUB 或 MQCO_REMOVE_SUB 来关闭预订。 您可以使用 MQCO_REMOVE_SUB 关闭持久预订，但 不能 使用 MQCO_KEEP_SUB 关闭非持久预订。 MQCO_REMOVE_SUB 的操作是除去预订，这将停止向预订队列发送任何其他发布。

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 253 页的图 50 中，示例通过在 NYSE/IBM/PRICE 主题上发布 130 开始。

```
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 *50:* 将 *130* 发布到 *NYSE/IBM/PRICE*

在第 253 页的图 51 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 254 页的图 55 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。 示例的实际行为取决于 MQSO_CREATE， MQSO_RESUME 和 MQSO_ALTER 的选择或组合。 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 *51:* 接收保留式发布内容

在 (第 253 页的图 52) 中，未收到任何发布，因为持久预订已接收到保留的发布。 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 中的 2038 错误是由于 MQSUB 不包含 MQOO_INQUIRE 选项而导致 STOCKTICKER 的隐式 MQOPEN 。 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 *52:* 恢复预订

在第 254 页的图 53 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 *53:* 使用新的非受管非持久预订来接收保留式发布内容

在第 254 页的图 54 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 *54:* 重叠预订

在第 254 页的图 55 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 *55:* 无法更改预订主题

相关概念

第 239 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 WebSphere MQ 消息使用者，它不预订主题本身。

第 242 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。 此示例需要队列， 主题 或 预订的 *no* 管理定义。

第 232 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， WebSphere MQ 对象 (例如队列) 是通过管理方式创建的，或者由使用可编程命令格式 (PCF) 的管理程序创建的。 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 [2] 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 256 页的图 56 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。

[2] 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。

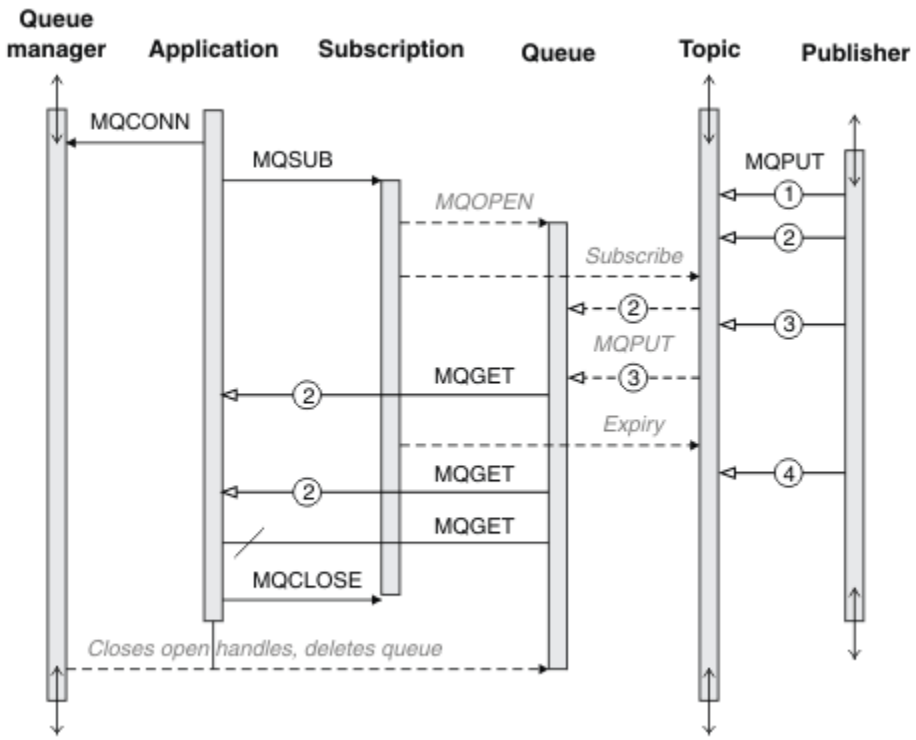


图 *56:* 受管非持久订户生命线

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 WebSphere MQ 接口的一部分。 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。

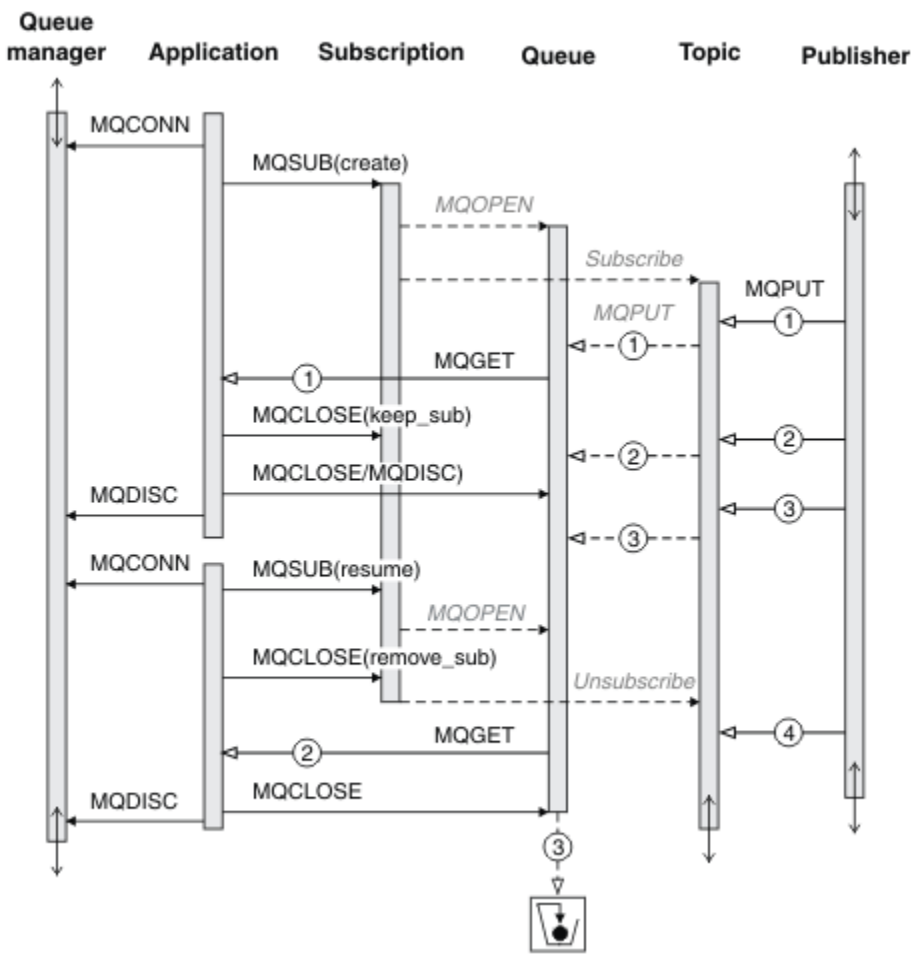


图 *57:* 受管持久订户生命线

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。

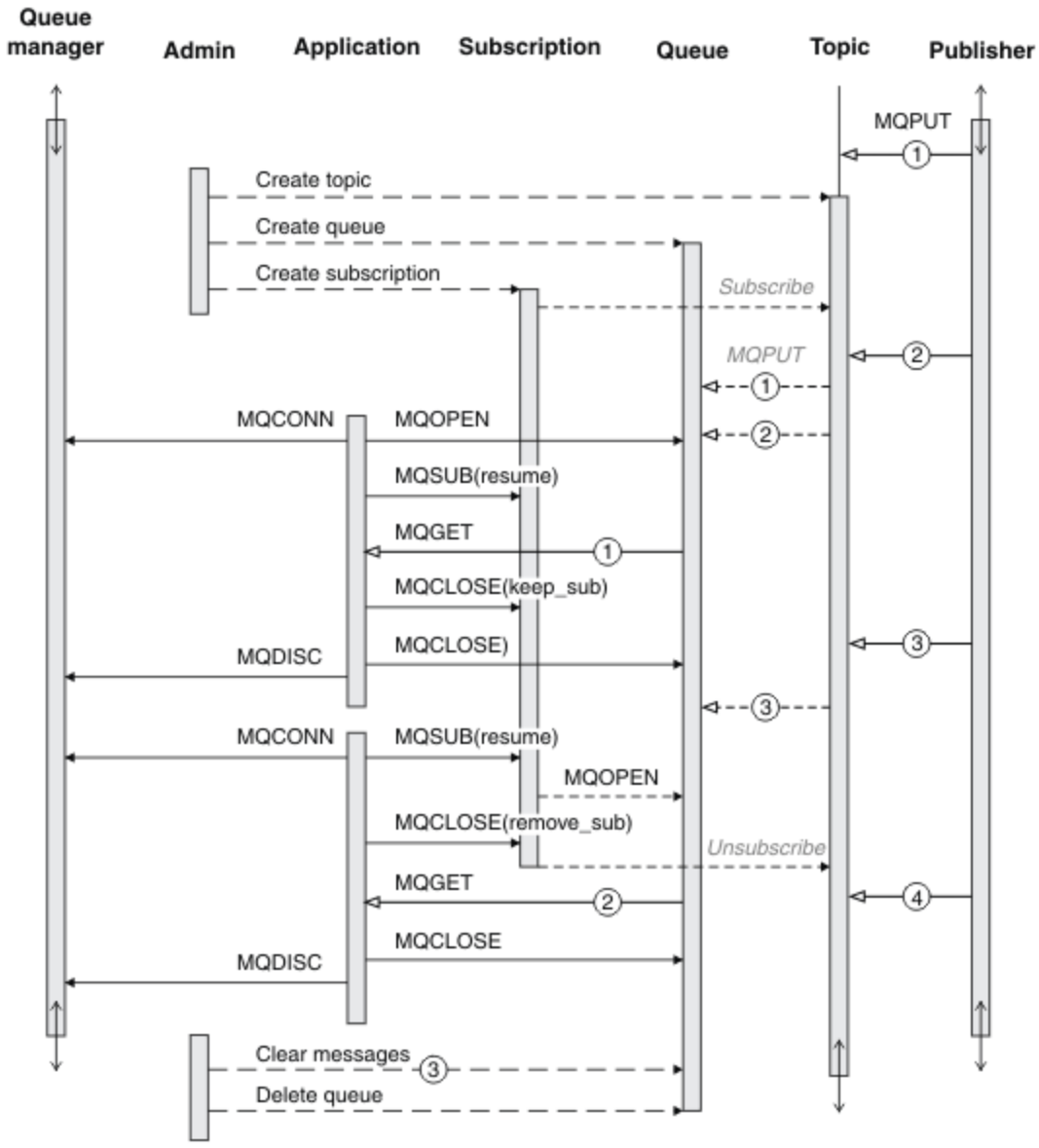


图 *58:* 非受管持久订户生命线

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 WebSphere 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 32 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 32 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 260 页的表 44 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 44: 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，与所有 WebSphere MQ 消息一样，有时可能会将消息按顺序传递。 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。

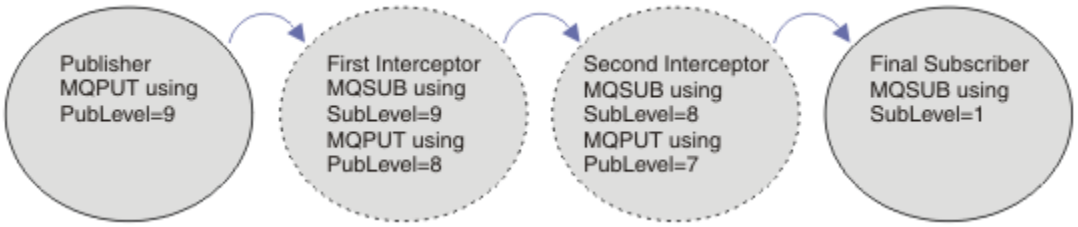


图 *59:* 拦截订户的顺序

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 261 页的表 45 中所述。

表 *45:* 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 261 页的表 45 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 262 页的表 46 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 运算符向 **MQPMO**.**选项** 字段添加其他值，以组合 put 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 262 页的『示例』中列出的代码片段。

表 *46:* 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 262 页的表 46 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 262 页的图 60 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 *60:* 预处理器伪指令

第 263 页的图 61 列出了代码片段中使用的声明。 除突出显示的术语外，声明是 WebSphere MQ 应用程序的标准声明。

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 *61:* 声明

在第 264 页的图 62 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
使用大多数 WebSphere MQ 结构的当前版本很重要。 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。

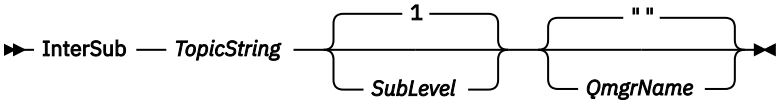
MQPRI_PRIORITY_AS_PUBLISHED
发布内容置于与其发布时处于相同消息优先级的发布内容队列中。

```
    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 *62:* 初始化

第 264 页的图 63 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 *63:* 准备拦截发布内容

主要代码片段第 265 页的图 64 会从发布内容队列获取消息。 它会查询消息属性，并使用主题字符串和发布的原始 **MQPMO**.**选项** 属性来重新发布消息。

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 *64:* 拦截发布内容并重新发布

最终代码片段如第 265 页的图 65 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 *65:* 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 266 页的图 66 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。

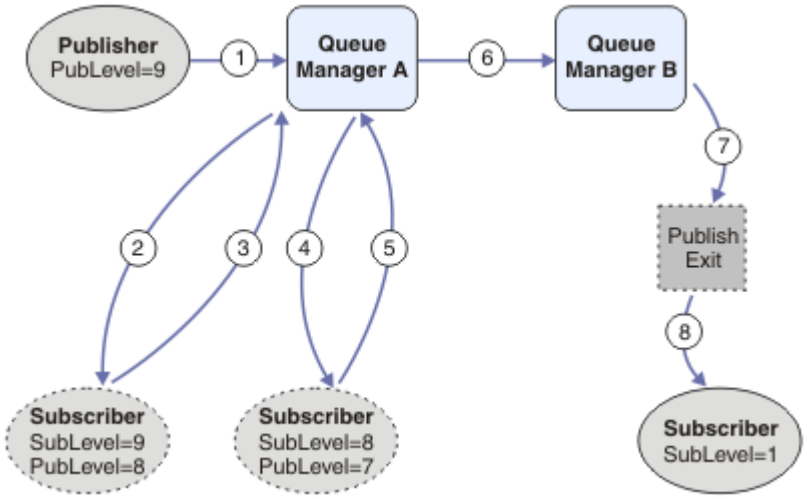


图 66: 集群中的拦截和发布出口

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 266 页的图 67 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 266 页的图 67 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。

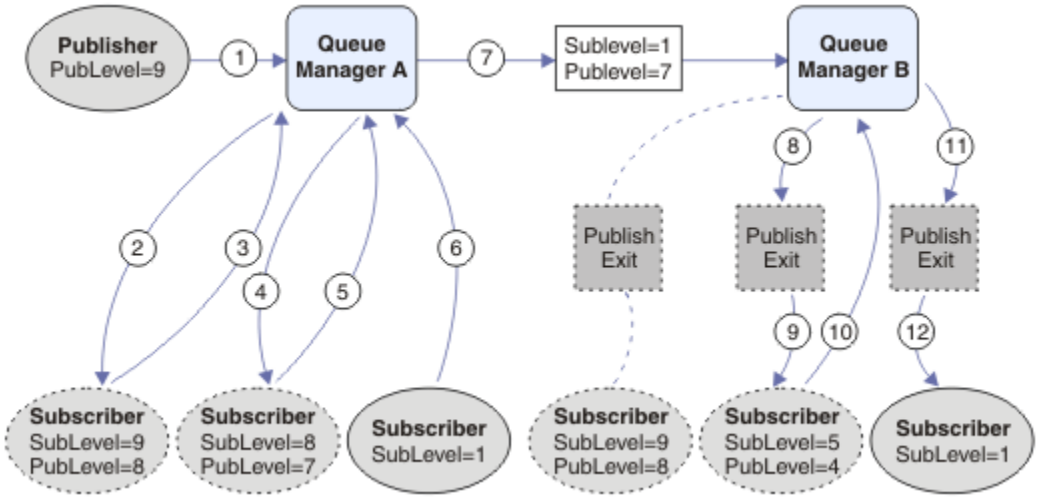


图 67: 拦截订户的复杂部署

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 WebSphere MQ 对象的特征的属性。

属性会影响队列管理器处理对象的方式。 对象属性中详细描述了每种类型的 WebSphere MQ 对象的属性。

某些属性是在定义对象时设置的，只能使用 WebSphere MQ 命令进行更改; 此类属性的示例是放入队列中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：

Windows Linux UNIX **对于 WebSphere MQ for Windows， UNIX and Linux 平台**

MQSC 工具，在 MQSC 引用中对此进行了描述。

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 WebSphere MQ 命令来定义，改变或显示属性时，必须使用主题链接中的命令描述中显示的关键字来标识属性。

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。

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 173 页的『连接到队列管理器和从队列管理器断开连接』和第 179 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 269 页的『查询对象的属性』
- 第 270 页的『MQINQ 调用失败的一些情况』
- 第 270 页的『设置队列属性』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 201 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 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

查询对象的属性

使用 MQINQ 调用来查询有关任何类型的 IBM WebSphere 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 290 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 (用于访问另一个队列的 WebSphere MQ 对象) 的属性，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* (不在 z/OS 上)
- *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 WebSphere 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 272 页的『IBM WebSphere 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 WebSphere 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 WebSphere 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)，在两个同步点之间处理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 属性在其他平台 (z/OS 除外) 上控制。 有关这些命令的详细信息，请参阅 MQSC 引用 *WebSphere MQ Script (MQSC) Command Reference* 。

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 过程是一个过程，在此过程中，程序对 IBM WebSphere MQ 队列进行的更新可以与对其他资源 (例如，受 DB2 控制的数据库) 的更新进行协调。 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 WebSphere 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 272 页的『IBM WebSphere MQ 应用程序中同步点注意事项』
- 第 273 页的『UNIX, Linux, and Windows 系统中的 IBM WebSphere MQ 中的同步点』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

第 289 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。

IBM WebSphere 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 WebSphere MQ 应用程序中的同步点。

在以下系统中支持两阶段落实：

- WebSphere MQ for AIX
- WebSphere MQ for HP-UX
- WebSphere MQ for Linux
- WebSphere MQ for Solaris
- WebSphere MQ for Windows
- CICS for MVS/ESA 4.1
- CICS Transaction Server for z/OS
- TXSeries
- IMS/ESA
- 使用 X/Open XA 接口的其他外部协调程序

在以下系统中支持单阶段落实：

- UNIX 系统上的 WebSphere MQ
- WebSphere MQ for Windows

注: 有关外部接口的更多详细信息，请参阅第 274 页的『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 (例如 CICS， IMS， Encina 和 Tuxedo) 可以参与与其他可恢复资源协调的两阶段落实。 这意味着可以将 WebSphere MQ 提供的排队功能引入到由事务管理器管理的工作单元的作用域内。

WebSphere MQ 随附的样本显示了协调 XA 兼容数据库的 WebSphere MQ 。 有关这些样本的更多信息，请参阅第 79 页的『分布式平台的样本程序』。

在 WebSphere MQ 应用程序中，您可以在每个 put 和 get 调用上指定是否希望该调用处于同步点控制下。 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。 同步点控制缺省值 为 no。

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。

如果程序在存在未落实的请求时发出 MQDISC 调用，那么会出现隐式同步点。 如果此程序异常终止，那么将发生隐式回退。

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。

UNIX, Linux, and Windows 系统中的 IBM WebSphere MQ 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 工作单元是其中唯一更新的资源是 WebSphere MQ 队列管理器的资源。 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。 WebSphere MQ 可以协调此类工作单元本身。 也可以使用外部落实控制器（如其他事务管理器或 IBM i 落实控制器）来对其进行协调。

为确保完整性，请使用两阶段落实过程。 两阶段落实可以由符合 XA 的事务管理器和数据库 (例如 IBM 的 TXSeries 和 UDB 提供。 WebSphere MQ 产品 (WebSphere MQ for IBM i 和 WebSphere MQ for z/OS 除外) 可以使用两阶段落实过程来协调全局工作单元。

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。

如果应用程序在 WebSphere MQ 协调的全局工作单元仍处于活动状态时与队列管理器断开连接 (MQDISC) ，那么将尝试落实该工作单元。 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。

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

全局工作单元的队列管理器协调不受 WebSphere MQ for IBM i 或 WebSphere MQ for z/OS 支持。 它在 WebSphere MQ MQI 客户机环境中不受支持。

在此， WebSphere 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 WebSphere 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。 使用两阶段落实过程，首先要求资源管理器 (例如，符合 XA 的数据库管理器，例如 DB2， Oracle 和 Sybase) 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

当选择了除 WebSphere MQ 以外的同步点协调程序时，会发生此情况; 例如， CICS， Encina 或 Tuxedo。

在此情况下， UNIX and Linux 系统和 WebSphere MQ for Windows 上的 WebSphere MQ 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 WebSphere MQ API 调用。 如果不使用 XA ，那么将不会在 CICS 原子工作单元中管理消息与队列之间的放入和获取。 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 使用两阶段落实协议来确保同时更新工作单元中的所有资源。

有关设置事务支持的更多信息，请参阅 第 35 页的『业务支持方案』以及 TXSeries CICS 文档，例如 *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*。

到外部同步点管理器的接口

UNIX and Linux 系统上的 WebSphere MQ ， 和 WebSphere MQ for Windows 支持使用 X/Open XA 接口的外部同步点管理器协调事务。

某些 XA 事务管理器 (TXSeries) 要求每个 XA 资源管理器提供其名称。 这在 XA 切换结构中称为 name 字符串。 UNIX， Linux 和 Windows 系统上 WebSphere MQ 的资源管理器名为 MQSeries_XA_RMI。 有关 XA 接口的更多详细信息，请参阅由 Open Group 发布的 XA 文档 *CAE Specification Distributed Transaction Processing: The XA Specification*。

在 XA 配置中， UNIX， Linux 和 Windows 系统上的 WebSphere MQ 可充当 XA Resource Manager 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：

MQRMIXASwitch	静态 XA 资源管理
MQRMIXASwitchDynamic	动态 XA 资源管理

有关包含此结构的库的列表，请参阅 第 57 页的『IBM WebSphere MQ XA 开关结构』。

必须用于将它们链接到 XA 同步点协调程序的方法由协调程序定义; 请参阅该协调程序提供的文档以确定如何使 WebSphere MQ 与 XA 同步点协调程序进行合作。

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 WebSphere 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 178 页的『使用 MQCONNX 的共享（独立于线程）连接』中所述）。
2. 在 Windows 系统上， XA 开关中声明的所有函数都声明为 _cdecl 函数。
3. 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 :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 服务器应用程序仍遵循一次只允许一个连接的规则。
4. 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，他们可以将出口 UE014015 用于 TXSeries CICS。

使用触发器启动 IBM WebSphere MQ 应用程序

了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 应用程序。

某些为队列提供服务的 WebSphere MQ 应用程序持续运行，因此它们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。

WebSphere MQ 提供了一个工具，使应用程序能够在有消息可供检索时自动启动。 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* 本地队列属性提供。 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。

在以下环境中运行的 WebSphere MQ 客户机支持触发:

- UNIX and Linux 系统
- Windows 系统

在客户机环境中运行的应用程序与在完整 WebSphere MQ 环境中运行的应用程序相同，除非您将其与客户机库链接。 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 AIX， HP-UX， IBM i， Solaris ， z/OS 或 Windows 系统上的传输队列，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出现 (请参阅 第 280 页的『触发器事件的条件』)。 WebSphere MQ 具有一系列选项，允许您控制导致触发器事件的条件 (请参阅 第 283 页的『控制触发器事件』)。

触发器消息
队列管理器在识别触发器事件时创建 触发器消息 (请参阅 第 280 页的『触发器事件的条件』)。 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 288 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。 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) 可以是 WebSphere 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。 在 z/OS 上，通常手动启动通道启动程序，或者在队列管理器启动时，可以通过在队列管理器启动 JCL 中更改 CSQINP2 来自动启动通道启动程序。 在其他平台上，它会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 286 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 277 页的图 68，这是触发器类型 FIRST (MQTT_FIRST) 的示例。

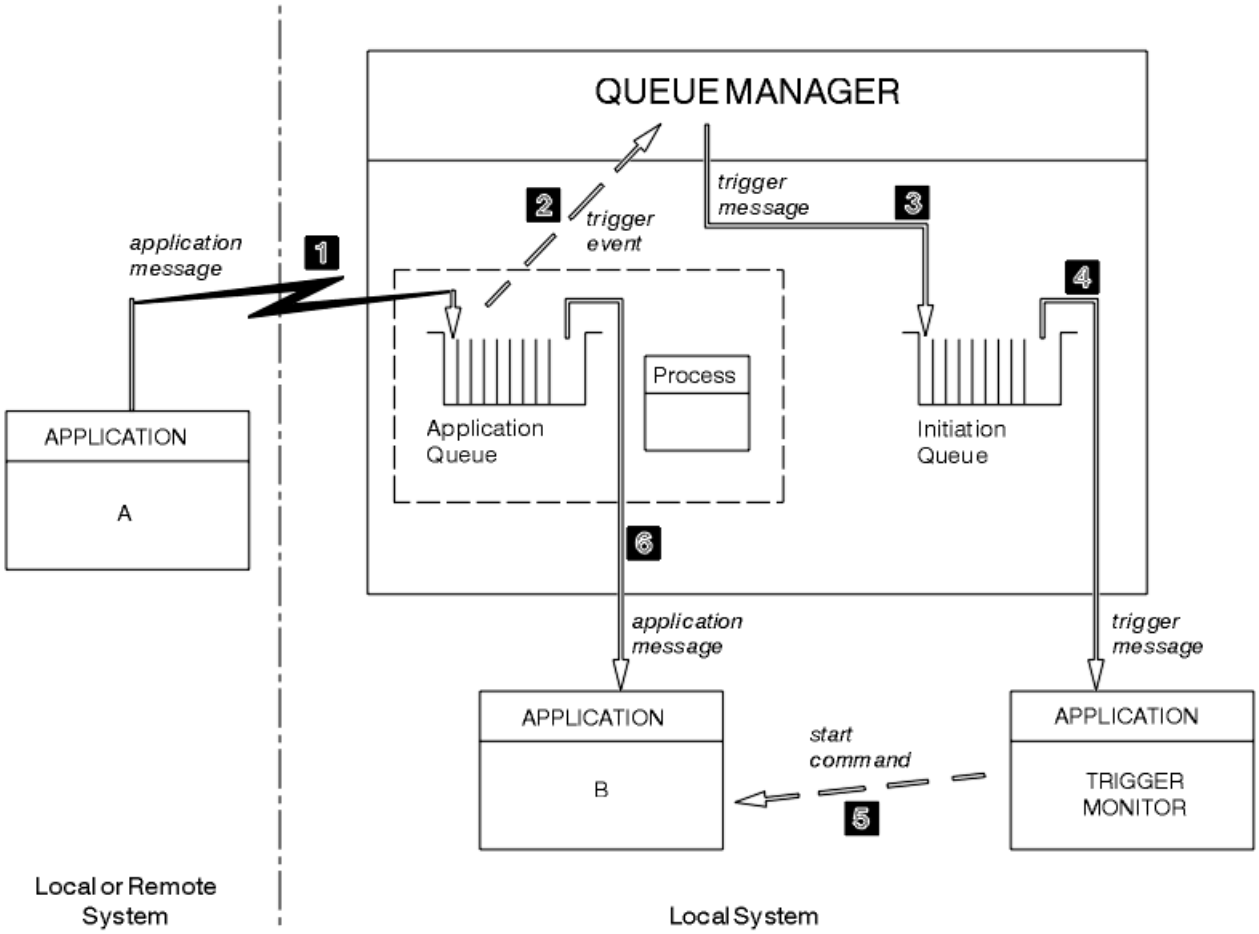


图 *68:* 应用程序和触发器消息流

在第 277 页的图 68 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 278 页的图 69。

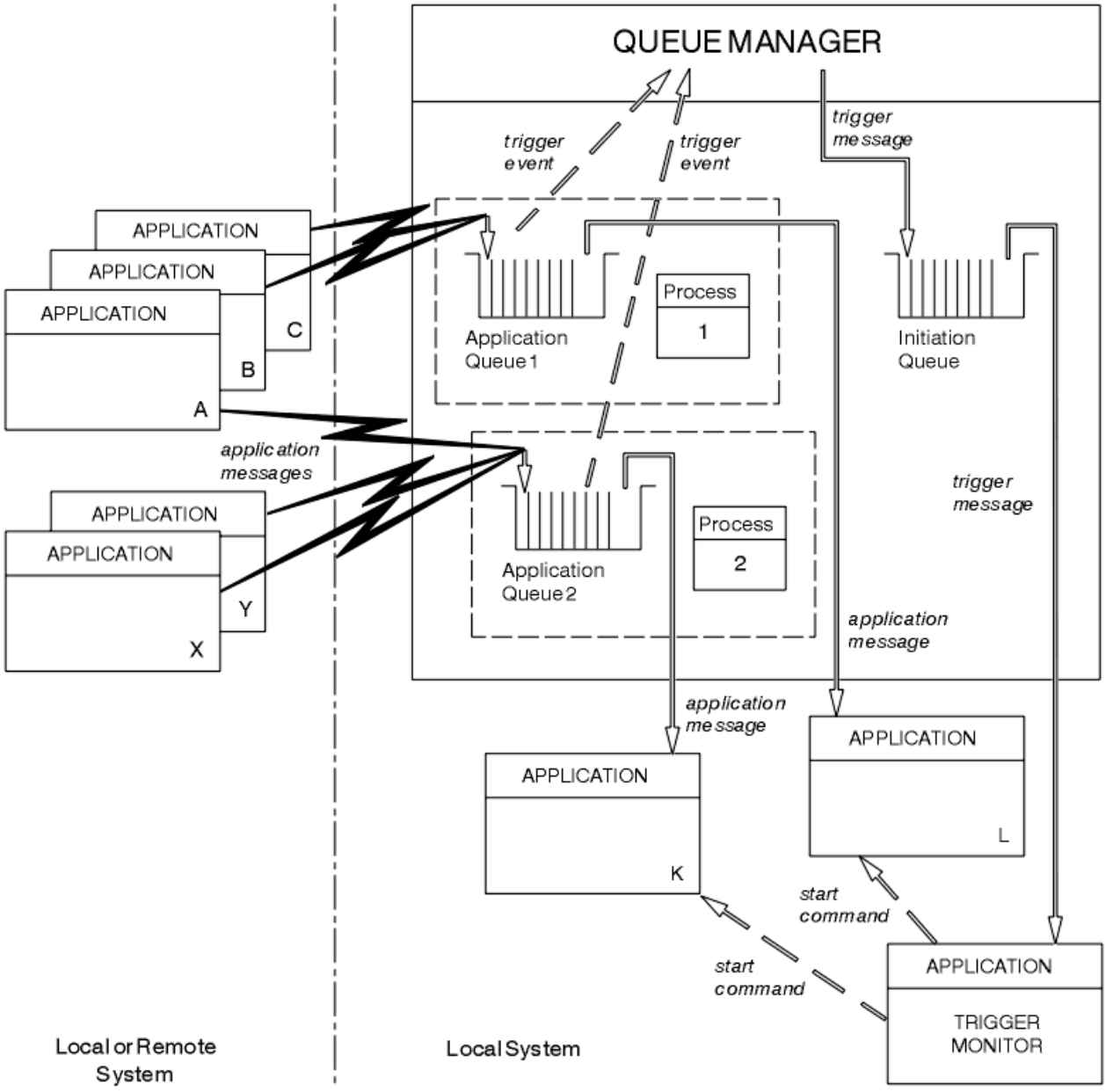


图 *69:* 触发内队列的关系

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。

使用以下链接来了解有关使用触发器启动 WebSphere MQ 应用程序的更多信息:

- 第 279 页的『触发的先决条件』
- 第 280 页的『触发器事件的条件』
- 第 283 页的『控制触发器事件』
- 第 285 页的『设计使用触发队列的应用程序』
- 第 286 页的『触发器监视器的启动队列处理』
- 第 288 页的『触发器消息的属性』
- 第 289 页的『触发无效时』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 289 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。

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)
```

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')
```

   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 和 Windows 系统	请使用 DEFINE PROCESS 或者使用 SYSTEM.DEFAULT.PROCESS 并使用 ALTER PROCESS 进行修改

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 WebSphere 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 和 Windows 系统	ALTER QLOCAL

6. 启动要用于服务您定义的启动队列的触发器监视器的实例。 请参阅第 286 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 283 页的『控制触发器事件』。

触发器事件的条件

本主题中对共享队列的引用表示队列共享组中的共享队列，仅在 WebSphere MQ for z/OS 上可用。

满足以下条件时，队列管理器将创建触发器消息：

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 WebSphere MQ for z/OS 上，如果应用程序队列的 *Usage* 属性为 MQUS_NORMAL ，那么不会禁止针对它的获取请求 (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* 队列属性为 MQQA_GET_ALLOWED）。
   - 针对启动队列不得禁止 put 请求（即 *InhibitPut* 队列属性必须为 MQQA_PUT_ALLOWED）。
   - 启动队列的 *Usage* 属性必须为 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 280 页的『2』 到 第 282 页的『10』（不包括 第 280 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 timer* 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 280 页的『2』）的队列上，并满足条件 第 281 页的『6』 到 第 282 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 32 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 280 页的『2』）的队列上，并满足条件 第 281 页的『4』 到 第 282 页的『10』（不包括 第 281 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 280 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 281 页的『4』 到 第 282 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 280 页的『2』）的队列上，并且满足条件 第 281 页的『4』 到 第 282 页的『10』（不包括 第 281 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 280 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 281 页的『4』 到 第 282 页的『10』（不包括 第 281 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 283 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：

1. 从步骤 第 282 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 WebSphere MQ 在 MQPUT 期间对消息进行分段，那么在将所有分段成功放入队列之前，将不会处理触发器事件。 但是，一旦消息段位于队列中， WebSphere MQ 就会将它们视为单独的消息以用于触发目的。 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。 但是，三个段中的每个段都会导致在通过 WebSphere MQ 网络移动时处理它们自己的触发器事件。

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。 如果优先级低于 *TriggerMsgPriority* 的消息到达应用程序队列，那么队列管理器在确定是否创建触发器消息时将忽略该消息。 如果 *TriggerMsgPriority* 设置为零，那么所有消息都将计入触发器事件。

TriggerType
除了触发器类型 NONE (就像将 *TriggerControl* 设置为 OFF 一样禁用触发) 之外，您还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 285 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达到 *TriggerDepth* 属性的值时，才会发生触发器事件。 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:** With triggering by depth, the queue manager disables triggering (using the <xph><pv>TriggerControl</pv></xph> attribute) after it creates a trigger message. 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 280 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 WebSphere MQ for z/OS 上，将定期扫描用于命名开放式启动队列的应用程序队列。 如果自上次发送触发器消息以来已经过 *TRIGINT* 毫秒，并且队列满足触发器事件的条件并且 CURDEPTH 大于零，那么将生成触发器消息。 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 WebSphere MQ for z/OS 上，如果将 *TRIGINT* 设置为较小的值，并且没有应用程序为触发器类型 FIRST 应用程序队列提供服务，那么每次对命名开放式启动队列的应用程序队列进行定期扫描时，回退触发都会生成一条触发器消息。
- 如果工作单元已回退 (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高值 (或缺省值)，那么在回退工作单元时将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* 队列属性具有值 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。

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 79 页的『分布式平台的样本程序』，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。 如果在后台运行应用程序，那么 WebSphere MQ 不会对可运行的应用程序数施加限制。

要在 Windows 系统上的后台运行已启动的应用程序，请在 *ApplId* 字段中以 START 命令作为应用程序名称的前缀。 例如：

```
START ?B AMQSECHA
```

To run the started application in the background on UNIX systems, put an & at the end of the *EnvData* of the PROCESS definition.

注: 如果 Windows 路径将空格作为路径名的一部分，那么应将这些空格括在引号 (") 中 以确保将其作为单个自变量进行处理。 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 命令的语法包含一个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。 要确保 Windows 不会将应用程序路径误认为是 "title" 自变量，请将用双引号括起的标题字符串添加到命令中的应用程序名称之前。

为 WebSphere MQ 客户机提供了以下触发器监视器:

runmqtmc
这与 runmqtrm 相同，只是它与 WebSphere MQ MQI 客户机库链接。

对于 *CICS*
为 CICS 提供了 amqltmc0 触发器监视器。 它的工作方式与标准触发器监视器 runmqtrm 相同，但您以不同的方式运行它，并且它会触发 CICS 事务。

本主题仅适用于 Windows， UNIX 和 Linux 系统。

它作为 CICS 程序提供; 使用 4 字符的事务名称定义它。 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 WebSphere 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: 这要求您使用 EXEC CICS START 调用来编写程序，因为该结构太长，无法添加为参数。 然后，将 MQTMC2 结构作为数据传递到触发器监视器的 START 请求。

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。 从启动队列读取消息是在 CICS 同步点控制下执行的。

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。

以下是触发器监视器的可用版本：

版本	适用平台
amqltmc0	TXSeries for AIX， HP-UX 和 Sun Solaris V 5.1
amqltmc4	TXSeries for WindowsV 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。
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）来完成关联。

有关触发器监视器接口的更多信息，请参阅 MQTMC2。

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 288 页的『触发器消息的持久性和优先级』
- 第 288 页的『队列管理器重新启动并触发消息』
- 第 288 页的『触发器消息以及针对对象属性的更改』
- 第 288 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。 在除 WebSphere MQ for z/OS 以外的任何 WebSphere MQ 产品上，此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，它会使用 *ProcessName* 中标识的进程定义对象的 *ApplType* 属性来填充该字段。

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，它会使用 *ProcessName* 中标识的进程定义对象的 *ApplId* 属性来填充该字段。 使用 WebSphere MQ for z/OS 提供的触发器监视器 CKTI 或 CSQQTRMN 时，进程定义对象的 *ApplId* 属性是 CICS 或 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，它会使用 *ProcessName* 中标识的进程定义对象的 *EnvData* 属性来填充该字段。 WebSphere MQ for z/OS 提供的触发器监视器 (CKTI 或 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，它会使用 *ProcessName* 中标识的进程定义对象的 *UserData* 属性来填充该字段。 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。

如果无法成功完成对死信队列的放置操作，那么将废弃触发器消息，并将一条警告消息发送到 z/OS 控制台或系统操作员，或者将其放入错误日志中。

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。

如果程序成功触发但在接收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，那么使用 CICS AUXTRACE) 来查找失败原因。

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 290 页的『MQOPEN 和集群』
- 第 291 页的『MQPUT、MQPUT1 和集群』
- 第 291 页的『MQINQ 和集群』

- 第 292 页的『MQSET 和集群』
- 第 292 页的『返回码』

相关概念

第 163 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 173 页的『连接到队列管理器和从队列管理器断开连接』
要使用 WebSphere 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 179 页的『打开和关闭对象』
此信息提供了有关打开和关闭 WebSphere MQ 对象的洞察。

第 188 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 201 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 268 页的『查询和设置对象属性』
属性是用于定义 WebSphere MQ 对象的特征的属性。

第 271 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 275 页的『使用触发器启动 IBM WebSphere MQ 应用程序』
了解有关触发器以及如何使用触发器启动 IBM WebSphere MQ 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。

如果队列管理器正在选择目标队列管理器，那么它将使用工作负载管理算法以循环方式执行此操作; 请参阅 工作负载均衡。

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 WebSphere 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。 如果指定了 MQOO_BIND_ON_GROUP ，但消息不是组的一部分，那么将改为使用 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 184 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 WebSphere 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。

在 z/OS 以外的平台上，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。

仅当同时符合以下两种情况时，才会发生此原因码:

- 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。

编写客户机应用程序

在 WebSphere MQ 上编写客户机应用程序所需的知识。

可以在 WebSphere 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将其链接到所使用的 WebSphere MQ MQI 客户机。 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 297 页的『为 WebSphere MQ MQI 客户机构建应用程序』。

您可以在完整的 WebSphere MQ 环境和 WebSphere MQ MQI 客户机环境中运行 WebSphere MQ 应用程序，而无需更改代码，前提是满足特定条件。 有关在 WebSphere MQ 客户机环境中运行应用程序的更多信息，请参阅第 299 页的『在 IBM WebSphere MQ MQI 客户机环境中运行应用程序』。

如果使用消息队列接口 (MQI) 来编写要在 WebSphere MQ MQI 客户机环境中运行的应用程序，那么在 MQI 调用期间需要施加一些额外的控制，以确保 WebSphere MQ 应用程序处理不会中断。 有关这些控制的更多信息，请参阅第 294 页的『在客户机应用程序中使用消息队列接口 (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 309 页的『准备和运行 CICS 和 Tuxedo 应用程序』
- 第 34 页的『准备和运行 Microsoft Transaction Server 应用程序』
- 第 311 页的『准备和运行 WebSphere 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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 73 页的『设计 IBM WebSphere MQ 应用程序』
当您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 WebSphere MQ 提供的功能。

第 78 页的『样本 WebSphere MQ 程序』
使用此主题集合来了解不同平台上的样本 WebSphere MQ 程序。

第 162 页的『编写排队应用程序』
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 791 页的『在 WebSphere MQ 中使用 Web Service』
您可以使用 IBM WebSphere MQ Transport for SOAP 或 IBM WebSphere MQ Bridge for HTTP 为 Web Service 开发 IBM WebSphere MQ 应用程序。

第 232 页的『编写发布/预订应用程序』
开始编写发布/预订 WebSphere MQ 应用程序。

第 357 页的『构建 IBM WebSphere MQ 应用程序』
使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

第 462 页的『处理程序错误』
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。

在客户机应用程序中使用消息队列接口 (MQI)

此主题集合考虑编写 WebSphere MQ 应用程序以在 WebSphere MQ MQI 客户机环境中运行和在完整的 WebSphere MQ 队列管理器环境中运行之间的差异。

设计应用程序时，请考虑在 MQI 调用期间需要实施哪些控件，以确保 WebSphere MQ 应用程序处理不会中断。

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。

在除 z/OS 之外的平台上，您可以增加队列管理器的最大消息长度属性。 ALTER QMGR 中提供了详细信息。

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 WebSphere MQ 应用程序尝试将 MQPUT 调用或 MQGET 调用与大于此消息的消息配合使用，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。

选择客户机或服务器编码字符集标识 (CCSID)

使用客户机的本地 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 MQI 从应用程序传递到客户机存根的数据必须使用本地 CCSID (针对 WebSphere MQ MQI 客户机进行编码)。 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。

但是，如果队列管理器无法执行转换，那么 V7 中的 Java 客户机可以执行转换。 请参阅 第 565 页的『用于 Java 客户机连接的 WebSphere MQ 类』

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：

- 在 Windows 上：

```
SET MQCCSID=850
```

- 在 UNIX 系统上：

```
export MQCCSID=850
```

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。

如果应用程序在消息描述符 (MQMD) 之后执行包含 WebSphere MQ 头的多个 PUT，请注意在完成第一个 PUT 之后将覆盖 MQMD 的 CCSID 和编码字段。

在第一个 PUT 之后，这些字段包含已连接的队列管理器用于转换 WebSphere 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 WebSphere MQ 中，队列管理器的其中一个角色是应用程序中的同步点控制。 如果应用程序在 WebSphere MQ 基本客户机上运行，那么它可以发出 MQCMIT 和 MQBACK，但同步点控制的作用域仅限于 MQI 资源。 WebSphere 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，您可以使用随 WebSphere MQ 产品提供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。

您可以将外部事务管理器与 WebSphere MQ 扩展事务客户机配合使用。 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。 此性能改进可用于 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。

使用 MQOO_READ_AHEAD 调用 MQOPEN 时，仅当满足特定条件时， WebSphere MQ 客户机才会启用预读。 这些条件包括:

- 客户机和远程队列管理器都必须处于 WebSphere MQ V7 或更高版本。
- 必须针对线程 WebSphere 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 216 页的『提高非持久消息的性能』。

预读配置由三个属性 (MaximumSize， PurgeTime 和 UpdatePercentage) 控制，这些属性在 WebSphere MQ 客户机配置文件的 MessageBuffer 节中指定。

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 93 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/SSL 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退出程序时，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 330 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 WebSphere MQ MQI 客户机上使用共享连接句柄来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 297 页的『使用 MQCONNX』。

为 WebSphere MQ MQI 客户机构建应用程序

可以在 WebSphere MQ MQI 客户机环境中构建和运行应用程序。 必须构建应用程序并将其链接到所使用的 WebSphere MQ MQI 客户机。 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 *47:* 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	Yes	Yes	Yes			
HP Integrity NonStop Server	Yes		Yes	Yes		
HP-UX	Yes	Yes	Yes			
Linux	Yes	Yes	Yes			
Solaris	Yes	Yes	Yes			
Windows	Yes	Yes	Yes			Yes

请参阅相关主题，以获取有关以这些语言链接或构建客户机应用程序的指示信息。

将 C 应用程序与 WebSphere MQ MQI 客户机代码链接

编写要在 WebSphere MQ MQI 客户机上运行的 WebSphere MQ 应用程序后，必须将其链接到队列管理器。

您可以通过两种方式将应用程序链接到队列管理器:

1. 直接，在这种情况下，队列管理器必须与应用程序位于同一工作站上
2. 客户机库文件，允许您访问相同或不同工作站上的队列管理器

WebSphere MQ 为每个环境提供一个客户机库文件:

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。

HP-UX
用于非线程应用程序的 libmqic.sl 库或用于线程应用程序的 libmqic_r.sl 库。

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。

Solaris
libmqic.so.

如果要在仅安装了 WebSphere MQ MQI Client for Solaris 的工作站上使用这些程序，那么必须重新编译这些程序以将它们与客户机库链接:

```
$ /opt/SUNWspro/bin/cc -o <prog> <prog> 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
MQIC32.LIB.

将 C++ 应用程序与 WebSphere MQ MQI 客户机代码链接

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 WebSphere 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++

将 COBOL 应用程序与 IBM WebSphere MQ MQI 客户机代码链接

编写了要在 IBM WebSphere MQ MQI 客户机上运行的 COBOL 应用程序后，必须将其与相应的库链接。

IBM WebSphere MQ 为每个环境都提供了客户机库文件:

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。

HP-UX
将非线程 COBOL 应用程序与库 libmqicb.sl 相链接，或将线程 COBOL 应用程序与 libmqicb_r.sl 相链接。

Linux
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。

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
将应用程序代码与用于 32 位 COBOL 的 MQICCBB 库相链接。 Windows 的 IBM WebSphere MQ MQI 客户机不支持 16 位 COBOL。

将 Visual Basic 应用程序与 WebSphere MQ MQI 客户机代码链接

您可以在 Windows 上将 Visual Basic 应用程序与 WebSphere MQ MQI 客户机代码链接。

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`，以确保能正确地自动选择客户机 dll：

MQIC32.dll
Windows 2000， Windows XP 和 Windows 2003

在 IBM WebSphere MQ MQI 客户机环境中运行应用程序

您可以在完整 IBM WebSphere MQ 环境和 IBM WebSphere MQ MQI 客户机环境中运行 IBM WebSphere 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 哪些应用程序在 IBM WebSphere MQ MQI 客户机上运行? 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 WebSphere MQ MQI 客户机环境中工作时，请记住：

- 在 IBM WebSphere MQ MQI 客户机环境中运行的每个应用程序都有自己的服务器连接。 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 WebSphere MQ MQI 客户机应用程序连接到队列管理器

在 IBM WebSphere MQ MQI 客户机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 WebSphere 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 WebSphere 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 353 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，就使用它定义的通道。
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，就使用它们指向的客户机通道定义表。
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。 如果对客户机定义表的搜索失败，那么客户机将使用以下路径：
   - UNIX and Linux 系统：`/var/mqm/AMQCLCHL.TAB`
   - Windows: `C:\Program Files\IBM\Websphere MQ\amqclchl.tab`

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。

如果从应用程序收到 MQRC_Q_MGR_NOT_AVAILABLE 返回码并且错误日志文件中记录有错误消息 `AMQ9517 - File damaged`，请参阅迁移和客户机通道定义表 (CCDT)。

相关概念

客户机通道定义表

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

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 MQSERVER、MQCHLLIB 和 MQCHLTAB 环境变量将客户机通道信息提供给一个运行在客户机环境中的应用程序。

请参阅 MQSERVER、MQCHLLIB 和 MQCHLTAB，获取这些变量的详细信息。

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。

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。

注: 多个 IBM WebSphere MQ 客户机可使用同一个文件。 您可以使用 MQCHLLIB 和 MQCHLTAB IBM WebSphere MQ 环境变量来访问此文件的不同版本。 有关环境变量的信息，请参阅 使用 WebSphere MQ 环境变量 。

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。 WebSphere 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 302 页的图 70 是客户机连接表的图形表示，显示三个队列管理器组，两个在 CCDT 中作为 **QMNAME**(QM1) 和 **QMNAME**(QMGrp1)写入的指定队列管理器组，一个作为 **QMNAME**(')写入的空白组或缺省组。

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。

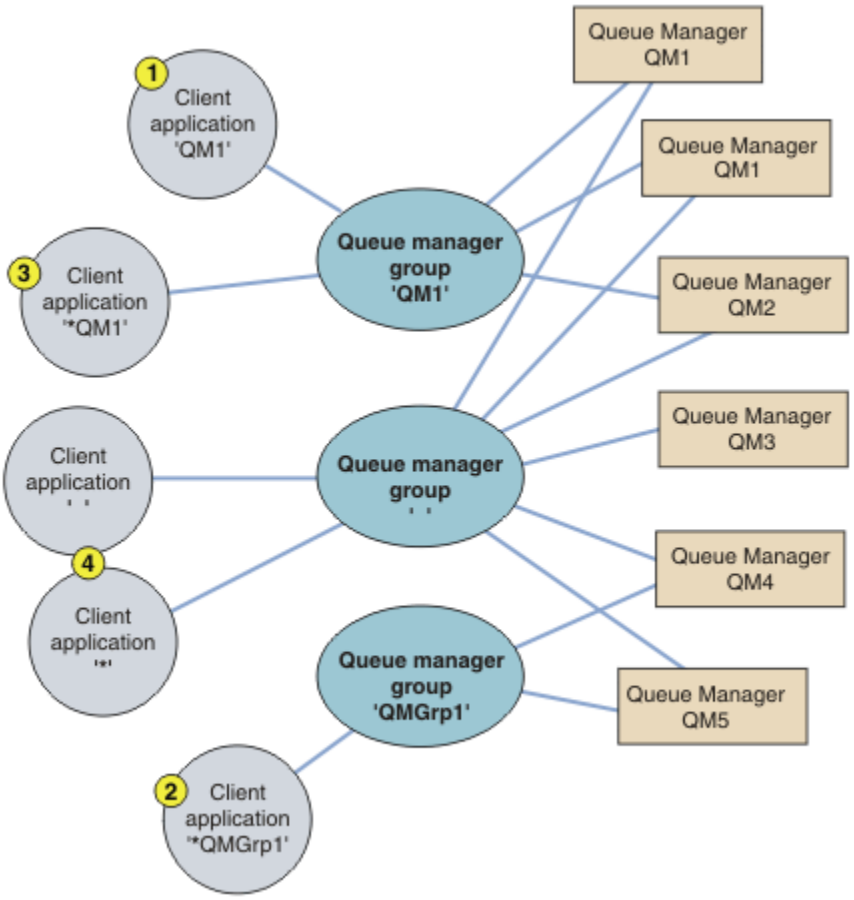


图 *70:* 队列管理器组

利用第 302 页的图 70 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。 WebSphere MQ 客户机代码选择匹配的队列管理器组 QM1。 该组包含三个连接通道， WebSphere MQ MQI 客户机尝试依次使用其中每个通道连接到 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 WebSphere MQ 侦听器。

   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。 WebSphere MQ 客户机选择匹配的队列管理器组 QMGrp1。 此组包含两个客户机连接通道， WebSphere MQ MQI 客户机尝试依次使用每个通道连接到 任何 队列管理器。 在此示例中， WebSphere MQ MQI 客户机需要成功建立连接; 与其连接的队列管理器的名称无关紧要。

   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，连接到某些队列管理器的客户机多于其他队列管理器的客户机。
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。 WebSphere MQ MQI 客户机将队列管理器之间的重新连接随机化，并随着时间推移扩展重新连接。

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。

您可以选择将客户机连接表分发到每个客户机，也可以将 CCDT 放置在共享文件系统上，以供每个客户机引用。 或者。使用 MQCONNX MQI 调用中支持的 CCDT 的程序版本，并调用服务以将 CCDT 传递至客户机应用程序。

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。

连接到队列共享组
您可以将您的应用程序连接到一个队列管理器上，该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。

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 调用必须以星号作为队列管理器名称的前缀，如 MQCONN 调用的示例: 示例 1 中所述。 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
此示例说明 WebSphere MQ MQI 客户机如何从 CCDT 中选择通道，其中 ConnectionAffinity 设置为 PREFERRED。

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
此示例说明 WebSphere MQ MQI 客户机如何从 CCDT 中选择通道，其中 ConnectionAffinity 设置为 NONE。

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; 定义了与来自同一 WebSphere MQ MQI 客户机的两个服务器的连接。 （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。

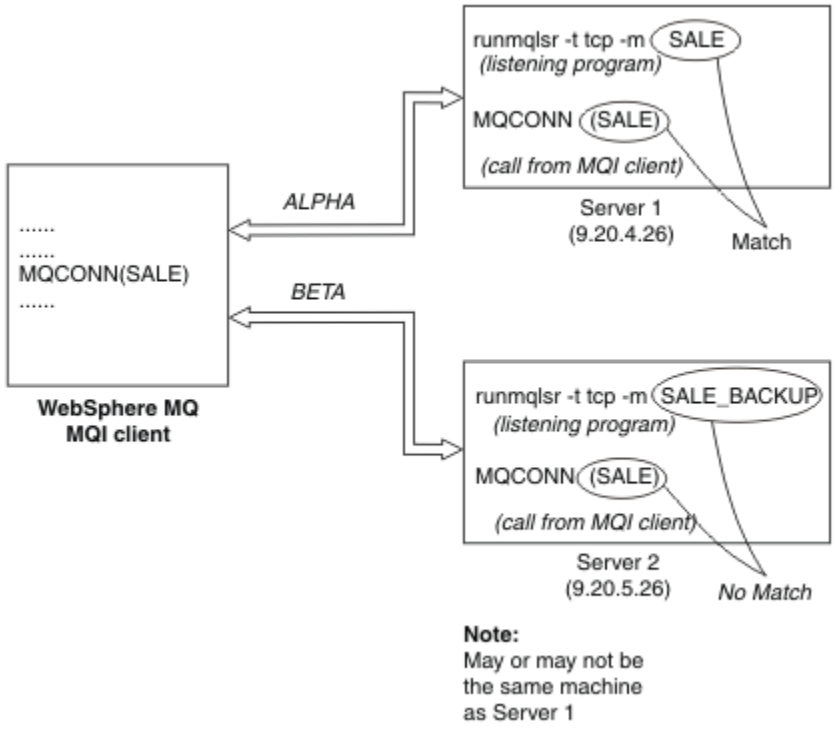


图 *71: MQCONN* 示例

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 WebSphere MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('WebSphere MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('WebSphere MQ MQI client connection to server 2') +
QMNAME(SALE)
```

SALE_BACKUP 定义：

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere 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. 尝试启动该此通道 - 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 306 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: 如果此应用程序在 WebSphere MQ MQI 客户机以外的环境中运行，并且名称为空，那么它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

```
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。 这同样与 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

由在 WebSphere MQ MQI 客户机上运行的 WebSphere MQ 应用程序发送的消息有助于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，如果在将消息放入 z/OS 队列管理器本地的触发队列的客户机应用程序中未指定 MQPMO 同步点控制选项，那么这些消息将放入工作单元中。 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，如果触发器监视器正在 Windows 客户机上运行，并且想要将请求发送到另一个操作系统上的服务器，那么必须定义 MQAT_WINDOWS_NT ，否则其他操作系统将使用其缺省定义，并且进程将失败。

触发器监视器

由非 z/OS WebSphere MQ 产品提供的触发器监视器在 UNIX， Linux 和 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：

- Windows Linux UNIX 在 Windows， UNIX 和 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

   如果选择在后台运行应用程序，那么 WebSphere MQ 不会对可运行的应用程序数施加任何限制。

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。 如果将 CICS 服务器应用程序作为客户机重新链接，那么可能会失去同步点支持。

必须将发出 MQCONN 或 MQCONNX 调用的非 z/OS CICS 应用程序定义为 RESDA。 要使常驻代码尽可能小，您可以链接到单独的程序以发出 MQCONN 或 MQCONNX 调用。

如果 MQSERVER 环境变量用于定义客户机连接，那么必须在 CICSENV.CMD 文件。

WebSphere MQ 应用程序可以在 WebSphere MQ 服务器环境中运行，也可以在 WebSphere MQ 客户机上运行，而无需更改代码。 但是，在 WebSphere MQ 服务器环境中， CICS 可以充当同步点协调程序，您可以使用 EXEC CICS SYNCPOINT 和 EXEC CICS SYNCPOINT ROLLBACK 而不是 MQCMIT 和 MQBACK。 如果仅将 CICS 应用程序作为客户机重新链接，那么将失去同步点支持。 MQCMIT 和 MQBACK 必须用于在 WebSphere MQ MQI 客户机上运行的应用程序。

准备和运行 CICS 和 Tuxedo 应用程序

要将 CICS 和 Tuxedo 应用程序作为客户机应用程序运行，请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 CICS 和 Tuxedo 应用程序以作为 WebSphere MQ MQI 客户机应用程序运行，请遵循 配置扩展事务客户机中的指示信息。

但是，请注意，专门处理准备 CICS 和 Tuxedo 应用程序的信息 (包括随 WebSphere MQ 提供的样本程序) 假定您正在准备要在 WebSphere MQ 服务器系统上运行的应用程序。 因此，该信息仅指打算在服务器系统上使用的 WebSphere 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，对于在 AIX， HP-UX 或 Solaris 上以 C 编写的应用程序，请使用库 libmqic 而不是 libmqm。 在 Windows 系统上，使用库 mqic.lib 而不是 mqm.lib。
- 对于 AIX， HP-UX 和 Solaris 以及 第 309 页的表 49(对于 Windows 系统) ，请使用等效的客户机系统库，而不是 第 309 页的表 48 中显示的服务器系统库。 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 *48: AIX， HP-UX* 和 *Solaris* 上的客户机系统库

WebSphere MQ 服务器系统的库	**要在 WebSphere MQ 客户机系统上使用的等效库**
libmqmxa	libmqcxa

表 *49: Windows* 系统上的客户机系统库

WebSphere MQ 服务器系统的库	**要在 WebSphere MQ 客户机系统上使用的等效库**
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 WebSphere MQ 服务器应用程序时，通常会从 CICS 用户切换到事务的用户标识。 但是，在 CICS 下运行 WebSphere MQ MQI 客户机应用程序时，它会保留 CICS 特权权限。

CICS 和 Tuxedo 样本程序

CICS 和 Tuxedo 样本程序，在 AIX， HP-UX， Solaris 和 Windows 系统上使用。

第 309 页的表 50 列出了提供用于 AIX， HP-UX 和 Solaris 客户机系统的 CICS 和 Tuxedo 样本程序。 第 310 页的表 51 列出了 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 96 页的『CICS 事务样本』和 第 129 页的『TUXEDO 样本』。

表 *50: AIX， HP-UX* 和 *Solaris* 客户机系统的样本程序

描述	**来源**	**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	-

表 *50: AIX， HP-UX* 和 *Solaris* 客户机系统的样本程序 *(*继续*)*		
描述	**来源**	**可执行文件模块**
Tuxedo 程序的字段表文件	amqstxvx.flds	-
Tuxedo 程序的视图描述文件	amqstxvx.v	-

表 *51: Windows* 客户机系统的样本程序		
描述	**来源**	**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	-

针对 CICS 和 Tuxedo 应用程序修改的错误消息 AMQ5203

运行使用扩展事务客户机的 CICS 或 Tuxedo 应用程序时，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 WebSphere 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，值为 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”有错，4 表示资源管理器标识 &1 无效，5 表示在已连接另一个队列管理器的情况下尝试了使用名为“&3”的次队列管理器，6 表示在应用程序未连接至队列管理器时调用了事务管理器，7 表示另一个调用正在进行时执行了 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”的参数。

用户响应

请更正错误并重试该操作。

准备和运行 Microsoft Transaction Server 应用程序

要准备 MTS 应用程序以作为 WebSphere MQ MQI 客户机应用程序运行，请根据您的环境遵循以下指示信息。

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

要准备 MTS 应用程序以作为 WebSphere MQ MQI 客户机应用程序运行，请对该应用程序的每个组件执行下列其中一项操作:

- 如果组件为 MQI 使用 C 语言绑定，请遵循第 386 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 WebSphere MQ C++ 类，请遵循 第 551 页的『在 Windows 上构建 C++ 程序』 中的指示信息，但将组件与库 imqx23vn.lib 而不是 imqc23vn.lib 链接。
- 如果组件将 Visual Basic 语言绑定用于 MQI，请遵循第 390 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 WebSphere MQ Automation Classes for ActiveX (MQAX)，请使用值 `mqic32xa.dll` 定义环境变量 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 WebSphere MQ JMS 应用程序

您可以使用 WebSphere Application Server 作为事务管理器，以客户机方式运行 WebSphere MQ JMS 应用程序。 您可能会看到某些警告消息。

要以客户机方式在 WebSphere Application Server 作为事务管理器的情况下准备和运行 WebSphere MQ JMS 应用程序，请遵循 第 603 页的『使用 WebSphere MQ classes for JMS』中的指示信息。

运行 WebSphere MQ JMS 客户机应用程序时，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 WebSphere MQ 可安装服务

您可以使用用户出口， API 出口或可安装服务来扩展队列管理器设施。 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 第 311 页的『编写和编译出口和可安装服务』。

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

编写和编译出口和可安装服务

您可以编写和编译出口，而无需链接到 UNIX， Linux 和 Windows 上的任何 IBM WebSphere MQ 库。

关于此任务

本主题仅适用于 Windows UNIX and Linux 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 WebSphere MQ 安装在非缺省位置，那么必须在不链接任何 IBM WebSphere MQ 库的情况下编写和编译出口。

您可以在 Windows 和 UNIX and Linux 系统上编写和编译出口，而无需链接以下任何 IBM WebSphere 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 WebSphere 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 323 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 333 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 351 页的『针对 WebSphere MQ for Windows 编写数据转换出口』和第 347 页的『在 UNIX and Linux 系统上为 WebSphere 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 346 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 353 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 354 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 356 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 78 页的『样本 WebSphere MQ 程序』中查看其他示例。

3. 编译出口：
   - 不要链接到 IBM WebSphere MQ 库。
   - 不要在出口中包含任何 IBM WebSphere 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 324 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 345 页的『在 Windows 上编译通道出口程序， UNIX and Linux 系统』。
     - 数据转换出口：第 346 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

     对于 32 位出口，缺省出口路径为 *MQ_DATA_PATH*/exits，对于 64 位出口，缺省出口路径为 *MQ_DATA_PATH*/exits64。 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，可以使用 WebSphere MQ Explorer 来更改路径：

     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。

   如果出口同时位于特定安装目录和缺省路径目录中，那么路径中指定的 WebSphere MQ 安装将使用特定安装目录出口。 例如，出口放在 /exits/installation2 和 /exits 中，但没有放在 /exits/installation1 中。 WebSphere MQ 安装 installation2 使用 /exits/installation2 的出口。 WebSphere MQ 安装 installation1 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 319 页的『配置服务和组件』。
   - API 出口：第 328 页的『配置 API 出口』。
   - 通道出口：第 346 页的『配置通道出口』。
   - 发布出口：第 356 页的『配置发布出口』。
   - 预连接出口：第 354 页的『客户机配置文件的 PreConnect 节』。

适用于 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。

提供 WebSphere MQ 可安装服务的主要原因如下：

- 使您能够灵活地选择是使用 WebSphere MQ 产品提供的组件，还是用其他产品替换或扩充这些组件。
- 要允许供应商参与，请提供可能使用新技术的组件，而不必对 WebSphere MQ 产品进行内部更改。
- 允许 WebSphere MQ 更快，更便宜地利用新技术，从而以更低的价格更早地提供产品。

可安装服务 和 服务组件 是 WebSphere 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。 这使独立软件供应商和其他第三方能够提供可安装组件，以扩充或替换 WebSphere MQ 产品提供的组件。 第 314 页的表 52 概述了可以使用的服务和组件。

表 *52:* 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）
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 **注:** 共享队列必须将其 *Scope* 属性设置为 CELL。	• 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。

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 WebSphere MQ for AIX 中，提供的授权服务组件 OAM 执行所有可用功能。 请参阅第 317 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 315 页的图 72 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。

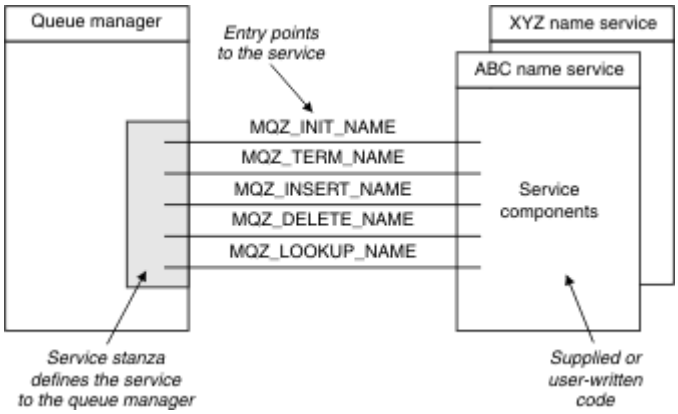


图 *72:* 理解服务、组件和入口点

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，并且定义为: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\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，由于使用了 Windows 注册表，因此 WebSphere MQ 发出 **RegEnumKey** 调用，该调用将按字母顺序返回值。 因此，在 Windows 上，将按字母顺序调用服务，因为它们是在注册表中定义的。

`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， Windows 上的 `AuthorizationService` 配置了名为 `MQSeries.WindowsNT.auth.service` 的缺省 OAM 组件。 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 WebSphere MQ 产品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)。

控制命令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。

- 在 UNIX and Linux 系统上：
  - 主体是用户标识或与代表用户运行的应用程序相关联的标识。
  - 组是 UNIX 或 Linux 系统定义的主体集合。
  - 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。
- 在 Windows 系统上：
  - 主体是 Windows 用户标识或与代表用户运行的应用程序相关联的标识。
  - 该组是 Windows 组。
  - 可在主体级别或组级别授予或撤销权限。

当发出 MQI 请求或发出命令时， OAM 会检查与该操作关联的实体的授权，以查看它是否可以:

- 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。

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 WebSphere 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 WebSphere MQ for Windows 上，授权服务提供以下入口点供队列管理器使用:

- **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**，可用于对组件函数进行原型设计。

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。 在组件目录中插入或删除队列的任何调用都由组件 ABC_name_serv 处理; 这是实现这些函数的唯一调用。 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 319 页的表 53 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。 假定尚未对服务定义的入口点 (使用 **MQZEP**) 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 319 页的表 53 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 *53:* 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

使用队列管理器配置文件配置服务组件，但在 Windows 系统上除外，其中每个队列管理器在注册表中都有自己的节。

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。

对于服务中的每个组件，必须要有一个 *ServiceComponent* 节。 此节用于标识包含该组件代码的模块名称和路径。

有关更多信息，请参阅第 320 页的『服务节格式』和第 320 页的『服务组件节格式』

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。 创建队列管理器时，将自动更新队列管理器配置文件 (或 Windows 系统上的注册表) ，以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。

服务节格式

服务节包含服务的名称和为服务定义的入口点数。

节的格式如下所示:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

其中:

<service_name>
服务名称。 由服务定义。

<entries>
为服务定义的入口点数量。 其中包含初始化和终止入口点。

Windows 系统的服务节格式

在 Windows 系统上， *Service* 节包含 SecurityPolicy 属性。

节的格式为:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
  SecurityPolicy=<policy>
```

其中:

<service_name>
服务名称。 由服务定义。

<entries>
为服务定义的入口点数量。 其中包含初始化和终止入口点。

<policy>
NTSIDsRequired (Windows 安全标识) 或 Default。 如果未指定 NTSIDsRequired，那么将使用 Default 值。 仅当 Name 具有 AuthorizationService 值时，此属性才有效。

另请参阅第 321 页的『配置授权服务节: Windows 系统』。

服务组件节格式

服务组件节的格式如下:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

其中:

<service_name>
服务名称。 必须与服务节中指定的 Name 匹配。

<component_name>
服务组件的描述性名称。 这必须唯一，并且仅包含对 WebSphere MQ 对象的名称有效的字符 (例如，队列名称)。 此名称出现在由服务生成的操作员消息中。 建议您使用以公司商标或类似的有辨识度的字符串开始的名称。

<module_name>
将包含这个组件代码的模块的名称。

<size>
在每个调用中传递给组件的组件数据区的大小（以字节计）。 如果不需要组件数据，请指定零。

这两节可以按任意顺序出现，并且这些节下的节键也可以按任何顺序出现。 对于这两个节中的任何一个节，所有节键都必须存在。 如果某个节键重复，将使用后一个节键。

启动时，队列管理器依次处理配置文件中的每个服务组件条目。 然后装入指定的组件模块，调用组件的入口点（它必须是组件初始化的入口点），传递给它一个配置句柄。

配置授权服务节：*UNIX and Linux* 系统
在 UNIX and Linux 系统上，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与文件名是 `/var/mqm/qmgrs/QMNAME/qm.ini`。

缺省授权组件的 *Service* 节和 *ServiceComponent* 节将自动添加到 `qm.ini` ，但可以被 `mqsnoaut` 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了 WebSphere MQ for AIX 上的两个授权服务组件。
MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

```
Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=MQ_INSTALLATION_PATH/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

图 *73: qm.ini* 中的 *UNIX and Linux* 授权服务节

服务组件节 (`MQSeries.UNIX.auth.service`) 定义缺省权限服务组件 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。

配置授权服务节*: Windows* 系统
在 WebSphere MQ for Windows 上，每个队列管理器在注册表中都有自己的节。

缺省授权组件的 *Service* 节和 *ServiceComponent* 节将自动添加到注册表，但可以使用 `mqsnoaut` 进行覆盖。 任何其他的 *ServiceComponent* 节都必须手动添加。

您还可以使用 WebSphere MQ 服务来添加 `SecurityPolicy` 属性。 仅当 *Service* 节上指定的服务是授权服务 (即缺省 OAM) 时， `SsecurityPolicy` 属性才适用。 `SecurityPolicy` 属性允许您为每个队列管理器指定安全策略。 可能的值为:

Default
如果您希望缺省安全策略生效，请指定 `Default`。 如果未将特定用户标识的 Windows 安全标识 (NT SID) 传递到 OAM ，那么将尝试通过搜索相关安全数据库来获取相应的 SID。

NTSIDsRequired
要求在执行安全检查时将 NT SID 传递给 OAM。

有关服务节格式的信息，请参阅第 320 页的『 Windows 系统的服务节格式』。 有关安全性的更多常规信息，请参阅 在 Windows UNIX and Linux 系统上设置安全性。

服务组件节 `MQSeries.WindowsNT.auth.service` 定义了缺省授权服务组件 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。

配置名称服务节：*Unix* 和 *Linux* 系统
在此处放置简短描述；用于第一段和摘要。

以下名称服务的 UNIX and Linux 配置文件节示例指定了由 ABC 公司（虚拟）提供的名称服务组件。

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

图 *74: qm.ini* 中的名称服务节（对于 *UNIX and Linux* 系统）

注: 在 Windows 系统上，名称服务节信息存储在注册表中。

更改用户权限之后刷新 *OAM*
在 WebSphere 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，以反映在操作系统级别所作的更改，而无需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

编写和编译 API 出口

API 出口允许您编写用于更改 WebSphere MQ API 调用 (例如 MQPUT 和 MQGET) 的行为的代码，然后在这些调用之前或之后立即插入该代码。

注: 在 WebSphere 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**应用程序都使用的特定队列管理器应用各种标准或业务流程。 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：

- 在安全性方面，您可以提供认证，检查应用程序是否经过授权可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）。
- 在灵活性方面，您可对业务环境中的快速更改作出反应，而不必更改环境中依赖于数据的应用程序。 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。
- 在监视队列或队列管理器的使用情况时，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用法统计信息。

在运行 API 出口时会发生什么情况?

编写出口程序并将其标识到 WebSphere MQ 后，队列管理器会在已注册的点自动调用您的出口代码。

IBM i、Windows、UNIX and Linux 系统上的节中会识别要运行的 API 出口例程。 本主题介绍了配置文件 mqs.ini 和 qm.ini 中的节。

例程定义可能出现在以下三个位置:

1. ApiExit 公共，在 mqs.ini 文件中，标识在队列管理器启动时应用的整个 WebSphere MQ 的例程。 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 323 页的『3』 项）。
2. ApiExit 模板 (在 mqs.ini 文件中) 标识在创建新的队列管理器时复制到 ApiExit 本地集 (请参阅此列表中的项 第 323 页的『3』) 的整个 WebSphere MQ 的例程。
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 328 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。

在 WebSphere MQ 的多个安装中使用 API 出口

确保为较早版本的 WebSphere MQ 编写的 API 出口用于处理所有版本，因为对版本 7.1 中的出口所作的更改可能无法用于较早版本。 有关对出口所作的更改的更多信息，请参阅 第 311 页的『编写和编译出口和可安装服务』。

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。 在启动应用程序之前，客户机应用程序必须确保与与该应用程序相关联的队列管理器的安装相对应的正确 WebSphere MQ 库已链接到该应用程序。

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。

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，用于打开 WebSphere MQ 资源以进行后续访问
- MQCLOSE，用于关闭先前已打开用于访问的 WebSphere 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，用于查询先前已打开以进行访问的 WebSphere 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。

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* 控制出口的函数，主要提供出口与外部控制块 MQAXP（API 出口参数结构）和 MQAXC（API 出口上下文结构）之间的通信。 *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 92 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 311 页的『编写和编译出口和可安装服务』。

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 WebSphere MQ 生成的值，并且特定于连接。 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 中，ExitMsgHandle 会清除属性并填充注册 MQ_INIT_EXIT 时 ExitProperties 字段内指定的属性，不包含消息描述符属性。 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。

提供了样本程序 amqsaem0.c 来展示 API 出口中消息句柄的使用。

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 92 页的『API 出口样本程序』中描述的样本程序的命令。 对于 Windows 系统以外的平台，可以在 *MQ_INSTALLATION_PATH*/samp 中找到样本 API 出口代码，在 *MQ_INSTALLATION_PATH*/samp/bin 中找到已编译和链接的共享库。 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* 表示 WebSphere MQ 的安装目录。

用户注意事项：

1. 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。

随着多点广播客户机的引入，API 出口和数据转换出口需要能够在客户端上运行，这是因为一些消息可能无法通过队列管理器。 以下库现在属于客户机包和服务器包：

表 *54:* 现在位于客户机和服务器包中的库

操作系统	**库**
Windows	32 位及 64 位：mqm.dll 和 mqm.pdb
Linux & HP-UX	32 位及 64 位：libmqm.so 和 libmqm_r.so
AIX	32 位及 64 位：libmqm.a 和 libmqm_r.a

表 *54:* 现在位于客户机和服务器包中的库 *(*继续*)*	
操作系统	**库**
Solaris	32 位及 64 位：libmqm.so

在 *Unix* 和 *Linux* 系统上编译 *API* 出口
如何在 UNIX 和 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 表示安装 WebSphere MQ 的高级目录。

在 AIX 上

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 -IMQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
     amqsaxe0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

在 HP-UX Itanium 平台上

32 位应用程序

非线程化

编译 API 出口源代码：

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

链接 API 出口源代码

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe
rm amqsaxe.o
```

线程化

编译 API 出口源代码：

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

链接 API 出口源代码

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe_r
rm amqsaxe.o
```

64 位应用程序

非线程化

编译 API 出口源代码：

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

链接 API 出口源代码

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe
rm amqsaxe.o
```

线程化

编译 API 出口源代码：

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

链接 API 出口源代码

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe_r
rm amqsaxe.o
```

打开 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 \
     -IMQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

32 位应用程序

非线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

64 位应用程序

非线程化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

线程化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

在 Solaris 上

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 -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

在 *Windows* 系统上

在 Windows 上编译并链接样本 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。

在 Windows 上调整 第 327 页的图 75 或 第 328 页的图 76 中的示例中的命令以编译和链接 amqsaxe0.c 。 这些命令与 Microsoft Visual Studio 2005,2008 或 2010 配合使用。 这些示例假定 WebSphere MQ C:\Program Files\IBM\WebSphere 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 *75:* 在 *32* 位 *Windows* 上编译和链接 *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 *76:* 在 *64* 位 *Windows* 上编译和链接 *amqsaxe0.c*

相关概念

第 92 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。

配置 *API* 出口

配置 IBM WebSphere 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：

- 使用 IBM WebSphere MQ Explorer (在 Windows 和 Linux (x86 和 x86-64 平台) 上)
- 使用 **amqmdain** 命令（在 Windows 上）
- 直接使用 mqs.ini 和 qm.ini 文件（在 Windows、 UNIX and Linux 系统上）。

  mqs.ini 文件包含与特定节点上的所有队列管理器相关的信息。 您可以在 UNIX and Linux 上的 /var/mqm 目录以及 Windows 系统上的 HKLM\SOFTWARE\IBM\WebSphere MQ 键中指定的 WorkPath 中找到该文件。

  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：

  在 UNIX and Linux 系统上：

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  在 Windows 系统上：

  ```
  C:\Program Files\IBM\WebSphere 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。

  您可以用以下方法之一编辑配置文件：

  – 自动，在节点上使用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 和 IBM WebSphere MQ 属性页面上 "出口" 下的 IBM WebSphere MQ Explorer 中定义。

当任何队列管理器启动时，将读取此节中的属性，然后由 qm.ini 中定义的 API 出口覆盖。

ApiExitTemplate

在 mqs.ini 和 IBM WebSphere MQ 属性页面上 "出口" 下的 IBM WebSphere MQ Explorer 中定义。

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

定义在 qm.ini 中和队列管理器属性页的 IBM WebSphere MQ Explorer 中，位于出口下。

当队列管理器启动时，此处定义的 API 出口将覆盖 mqs.ini 中定义的缺省出口。

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 WebSphere 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 WebSphere 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

此主题集合包含有关消息传递通道的 WebSphere 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 346 页的『编写数据转换出口』。

其中一些信息也适用于 MQI 通道上的出口，这些通道将 WebSphere MQ MQI 客户机连接到队列管理器。 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 330 页的表 55 中显示了可用于各个通道类型的通道出口类型。

表 *55:* 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	

表 55: 可用于各个通道类型的通道出口 (继续)						
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**
服务器连接通道			Yes	Yes	Yes	Yes

如果您要在客户机上运行通道出口，不能使用 MQSERVER 环境变量。 而是创建并引用客户机通道定义表 (CCDT)，如 客户机通道定义表 中所述。

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 331 页的图 77 所示，安全性检查阶段是一个循环。

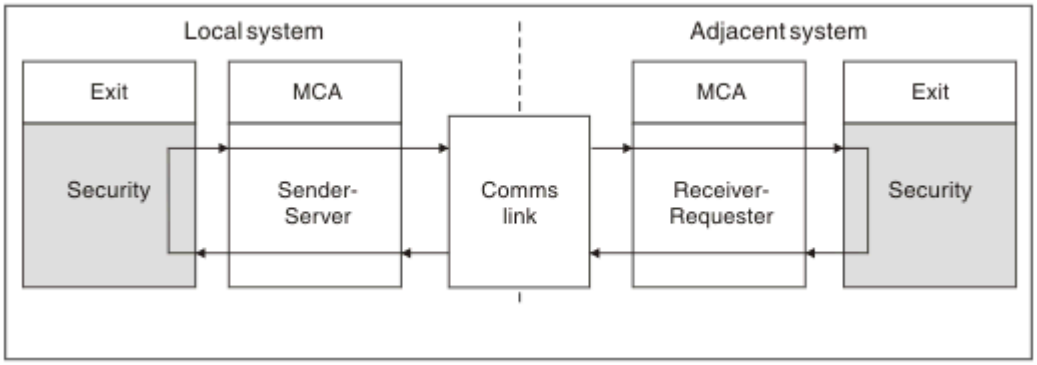


图 77: 安全出口循环

如第 332 页的图 78 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。

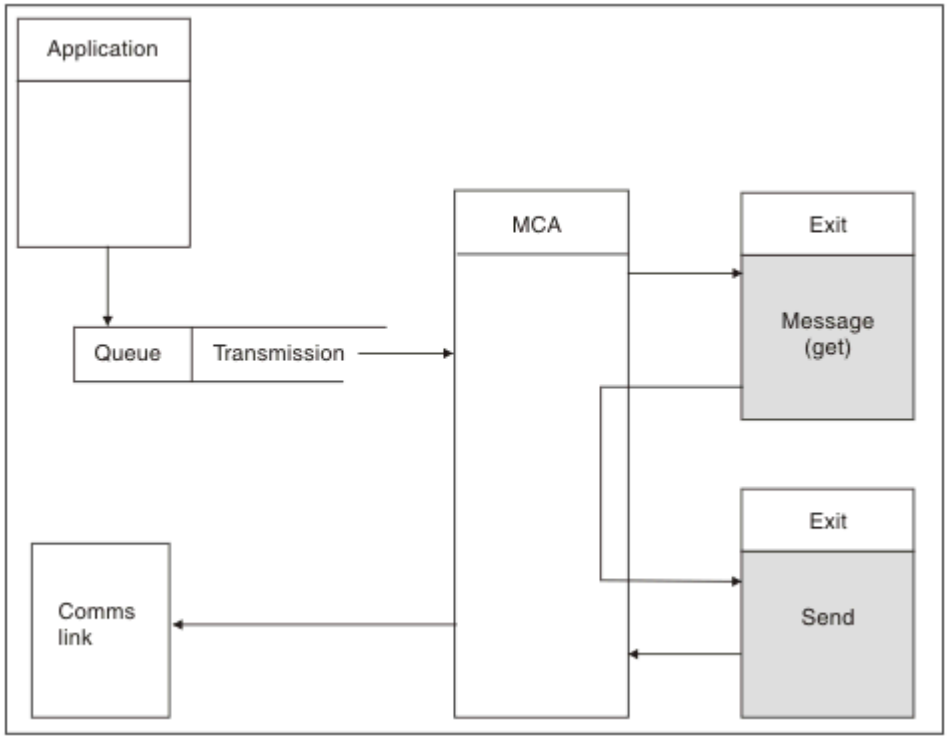


图 *78:* 消息通道的发送方端的发送出口示例

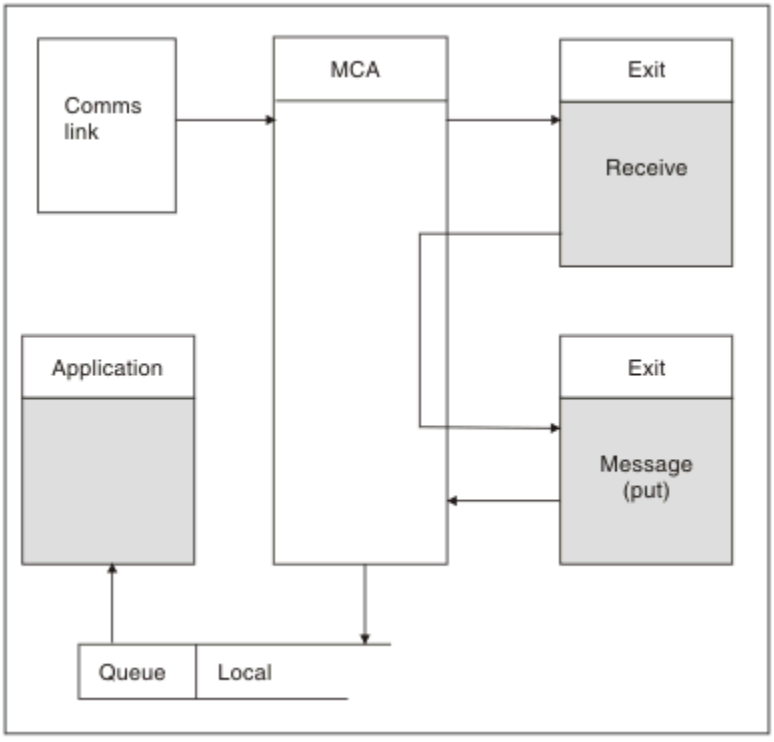


图 *79:* 消息通道的接收方端的接收出口示例

如第 332 页的图 79 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。 如果出口与 MQM.LIB，并且您未在 MQCONN 调用上指定队列管理器名称，出口会尝试连接到系统上的缺省队列管理器。 如果出口与 MQM.LIB，并指定通过 MQCD 的 QMgrName 字段传递到出口的队列管理器的名称，该出口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 WebSphere 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，那么 IBM WebSphere 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。

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：
  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 330 页的『消息传递通道的通道出口程序』获取调用详细信息。

在 *Windows UNIX and Linux* 系统上编写通道出口程序

您可以使用以下信息来帮助您为 Windows UNIX and Linux 系统编写通道出口程序。

请遵循第 311 页的『编写和编译出口和可安装服务』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，并且是 Windows 上的 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 334 页的图 80 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 *80:* 通道出口的样本源代码

使用 Visual C++ 为 Windows 编写通道出口时，必须编写您自己的 DEF 文件。 第 334 页的图 81 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 333 页的『编写通道出口程序』。

```
EXPORTS
ChannelExit
```

图 *81: Windows* 的样本 *DEF* 文件

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 WebSphere MQ 用户标识方面提供了很大的灵活性。 IBM WebSphere MQ 还提供了 SSL 和 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 SSL 和 TLS 的更多信息，请参阅 WebSphere MQ 支持 SSL 和 TLS。 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 Version 7.1 之前编写的安全出口，需要注意 IBM WebSphere 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 Version 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 Version 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 Version 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 Version 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 WebSphere MQ 对等名称过滤器不受影响，并且将继续以与先前发行版相同的方式运行。 这是因为 WebSphere MQ 对等名称匹配算法已更新为处理现有 SSLPEER 过滤器，而无需更改通道定义。 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 336 页的图 82 到 第 338 页的图 85 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 336 页的图 82 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 *82:* 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 *83:* 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 *84:* 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 *85:* 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. 在 IBM i 中，使用 "使用沿用权限" (USEADPAUT = *YES) 编译的安全出口程序可以采用 QMQM 或 QMQMADM 权限。 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 SSL 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 SSL 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

安全套接字层 (SSL) 和传输层安全性 (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 339 页的图 86 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 *86:* 安全出口框架代码

标准 WebSphere MQ 入口点 MQStart 必须存在，但不需要执行任何功能。 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
WebSphere MQ for Windows 提供了一个安全出口，该出口通过使用安全服务编程接口 (SSPI) 为 WebSphere MQ 通道提供认证。 SPI 提供 Windows 的集成安全设施。

此安全出口适用于 WebSphere MQ 客户机和 WebSphere 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。

在 Windows 上使用 NTLM 认证服务提供单向认证。 在 Windows 2000 上，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 142 页的『在 Windows 系统上使用 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。

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。 WebSphere MQ 为所有发送出口保留的空间维护总计。 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。

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 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 222 页的『参考消息』和第 116 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. 在 UNIX 系统中，如果出于任何原因提供消息出口，那么用户标识到小写字符的自动转换不会运行。 请参阅 UNIX and Linux 系统上的对象安全。
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，有可能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 "MQXMIT ")
- MQMD（此头是 MQXQH 的一部分，没有格式名称）
- MQMDE (格式名 "MQHMDE ")
- MQDH (格式名 "MQHDIST ")
- MQWIH (格式名 "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 "MQDEAD ")
- 格式名称以三个字符 "MQH" 开头的任何头 (例如 "MQHRF ") 其他未提及的

如何处理头

每个 WebSphere 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 "格式" 是适合于 WebSphere MQ 数据转换的头类型的名称，那么将对其进行转换。 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
当接收到启动接收方或服务器连接通道的请求，但该通道的定义不存在 (不适用于 WebSphere MQ for z/OS) 时，可以使用通道自动定义出口。 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。

当接收到启动接收方或服务器连接通道的请求但不存在通道定义时，可以在除 z/OS 以外的所有平台上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。 在这种情况下，该出口也适用于 WebSphere 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。 如果未指定通道自动定义出口，那么 z/OS 上的缺省行为是检查格式为 *[path]/libraryname(function)* 的分布式出口名称，并采用最多 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更改 MsgExitPtr， MsgUserDataPtr， SendExitPtr，SendUserDataPtr， ReceiveExitPtr 和 ReceiveUserDataPtr(而不是 MsgExit， MsgUserData， SendExit) 所寻址的字段。 SendUser 数据， ReceiveExit 和 ReceiveUser 数据字段本身。

有关更多信息，请参阅 通道的自动定义。

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。

在 *Windows* 上编译通道出口程序， *UNIX and Linux* 系统

使用以下示例来帮助您为 Windows UNIX and Linux 系统编译通道出口程序。

Windows

Windows

Windows 上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 和 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 上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
```

HP-UX 上通道出口的样本编译器和链接程序命令

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b exit.o +ee MQStart +ee ChannelExit -o
/var/mqm/exits64/exit -L/usr/lib/pa20_64 -lpthread
$ rm exit.o
```

用于 Linux 平台（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
```

用于 Linux 平台（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 上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。

在 AIX 上，必须导出 IBM WebSphere 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。 出口名称的格式取决于 IBM WebSphere MQ 平台; 请参阅 MQCD 或 脚本 (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 VSE/ESA 的 MQSeries 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 WebSphere MQ 需要能够了解 MQMD 的内容，而不考虑它是在哪个平台上创建的，因此系统会自动对其进行转换。

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。 本主题包含有关 WebSphere MQ 在应用程序数据未采用内置格式时提供的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)。 这将确保 WebSphere 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 WebSphere MQ 数据转换接口 (DCI)。

随着多点广播客户机的引入，API 出口和数据转换出口需要能够在客户端上运行，这是因为一些消息可能无法通过队列管理器。 以下库现在属于客户机包和服务器包：

表 *56:* 现在位于客户机和服务器包中的库

操作系统	**库**
Windows	32 位及 64 位：mqm.dll 和 mqm.pdb
Linux & HP-UX	32 位及 64 位：libmqm.so 和 libmqm_r.so
AIX	32 位及 64 位：libmqm.a 和 libmqm_r.a
Solaris	32 位及 64 位：libmqm.so

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。

如果出现以下情况，那么会调用出口：

- 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。

在 *UNIX and Linux* 系统上为 *WebSphere MQ* 编写数据转换出口

有关在 UNIX and Linux 系统上为 WebSphere 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 347 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 WebSphere MQ for AIX 上，还提供了名为 amqsvfc.exp 的框架导出文件。 复制此文件，将其重命名为 MYFORMAT.EXP。
6. 框架在目录 *MQ_INSTALLATION_PATH*/inc 中包含样本头文件 amqsvmha.h，其中 *MQ_INSTALLATION_PATH* 表示安装 WebSphere MQ 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 348 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 348 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 */
   ```

   在此，插入上面步骤 第 348 页的『3』 中添加的函数的函数原型语句。
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 348 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，那么必须以此方式编译所有 WebSphere MQ 应用程序。
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。

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 表示安装 WebSphere MQ 的高级目录。

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 -IMQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
    MYFORMAT.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

线程化

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
    MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

HP-UX Itanium 平台

通过发出以下某个命令集来编译和链接出口源代码：

32 位应用程序

非线程化

编译出口源代码：

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

链接出口对象：

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
   /var/mqm/exits/MYFORMAT -L/usr/lib/hpux32
rm MYFORMAT.o
```

线程化

编译出口源代码：

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

链接出口对象：

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
   /var/mqm/exits/MYFORMAT_r -L/usr/lib/hpux32 \
   -lpthread
rm MYFORMAT.o
```

64 位应用程序

非线程化

编译出口源代码：

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

链接出口对象：

```
ld -b MYFORMAT.o +ee MQStart \
    -o  /var/mqm/exits64/MYFORMAT \
    -L/usr/lib/hpux64
rm MYFORMAT.o
```

线程化

编译出口源代码：

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

链接出口对象：

```
ld -b MYFORMAT.o +ee MQStart  \
    -o  /var/mqm/exits64/MYFORMAT_r  \
    -L/usr/lib/hpux64 -lpthread
rm MYFORMAT.o
```

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 \
     -IMQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

32 位应用程序

非线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

64 位应用程序

非线程化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

线程化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

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 -IMQ_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 -IMQ_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 -IMQ_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 -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64  -lsocket -lnsl -ldl
```

针对 *WebSphere MQ for Windows* 编写数据转换出口

有关为 WebSphere MQ for Windows 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。 *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。

   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 WebSphere 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 351 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 WebSphere MQ 的安装目录。 (缺省安装目录为 C:\Program Files\IBM\WebSphere 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 351 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 351 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 351 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 WebSphere MQ 的安装目录。

7. 发出命令文件，将您的出口编译为 DLL 文件。
8. 将输出放在 WebSphere MQ 数据目录下的 exit 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\MQSeries\CurrentVersion\Configuration\ClientExitPa
th\
```

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 使用打包结构，那么必须以此方式编译所有 WebSphere MQ 应用程序。
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。

在 *Windows* 操作系统上退出和切换装入文件

IBM WebSphere MQ for Windows Version 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* 表示安装了 WebSphere MQ 的高级目录。 使用这些可确保找到相应的库。 如果在 WebSphere MQ 集群中使用了出口，那么这也会确保可以在远程系统上找到相应的库。

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 交叉出口	32 位	32 位和 64 位
数据转换出口	32 位	64 位
服务器通道出口（所有类型）	32 位	32 位
客户机通道出口（所有类型）	32 位	64 位

文件类型	32 位应用程序	64 位应用程序
可安装服务出口	32 位	32 位
服务跟踪模块	32 位	32 位和 64 位
集群 WLM 出口	32 位	32 位
发布/预订路由出口	32 位	32 位
数据库切换装入文件	32 位	32 位和 64 位
外部事务管理器 AX 库	32 位	64 位

使用存储库的预连接出口引用连接定义

可以将 WebSphere MQ MQI 客户机配置为使用预连接出口库来查找存储库以获取连接定义。

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。 由于很难管理引用 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 (例如 LDAP 目录， WebSphere Registry and Repository 或任何其他存储库) 中。 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 WebSphere 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*/
```

客户机配置文件的 *PreConnect* 节

使用 PreConnect 节在 `mqclient.ini` 文件中配置 PreConnect 出口。

可以将以下属性包含在 PreConnect 节中：

Data=*<URL>*
存储连接定义的存储库的 URL。 例如，在使用 LDAP 服务器时：

Data = ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com

Function=*<myFunc>*
包含 PreConnect 出口代码的库中的函数入口点名称。

函数定义遵循 PreConnect 出口原型 MQ_PRECONNECT_EXIT。

此字段的最大长度是 MQ_EXIT_NAME_LENGTH。

Module=*<amqldapi>*
包含 API 出口代码的模块名称。

如果此字段包含模块的完整路径名，那么将照原样使用该名称。

Sequence=*<sequence_number>*
相对于其他出口调用此出口的顺序。 先调用序号较低的出口，然后调用序号较高的出口。 出口的序列编号不需要连续；1、2、3 序列与 7、42、1096 序列的结果相同。 此属性是无符号的数字值。

可以在 `mqclient.ini` 文件中定义多个 PreConnect 节。 每个出口的处理顺序由节的 Sequence 属性确定。

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。

z/OS 上不支持发布出口。

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 311 页的『编写和编译出口和可安装服务』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。

WebSphere MQ 未提供 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 WebSphere MQ 连接都可能无法装入或初始化该出口。 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。 操作失败， WebSphere MQ 原因码为 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 311 页的『编写和编译出口和可安装服务』。

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。

在 Windows 和 Linux 上，可以使用 WebSphere MQ 资源管理器来定义属性。 在队列管理器属性页面的“发布/预订”下定义属性。

要在 UNIX 和 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=*function_name*
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。

PublishExitData=*string*
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，为了提供您自己的用户出口程序来定制工作负载管理， WebSphere MQ 包含一个用户出口，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，那么将选择新的目标队列管理器。

在 z/OS 以外的平台上，队列管理器将在下次启动队列管理器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，那么每个操作系统进程都将装入其自己的出口副本。 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

对于除 z/OS 以外的平台，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 311 页的『编写和编译出口和可安装服务』中的步骤，使用样本程序第 357 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 333 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 命令上指定集群工作负载出口属性，可以在队列管理器定义中命名集群工作负载出口。 例如：

```
ALTER QMGR CLWLEXIT(myexit)
```

样本集群工作负载出口

WebSphere MQ 包含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。

在 z/OS 以外的平台上

样本集群工作负载出口程序使用 C 语言提供，并称为 amqswlm0.c。 可以在以下内容中找到：

表 57: 样本集群工作负载出口程序位置 (非 z/OS)	
平台	**菲尔帕特**
AIX， HP-UX 和 Sun Solaris	*MQ_INSTALLATION_PATH*/samp
Windows	*MQ_INSTALLATION_PATH*\Tools\c\Samples

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 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：

在 UNIX and Linux 系统上：

```
ALTER QMGR CLWLEXIT('path/amqswlm(clwlFunction)')
```

在 Windows 上：

```
ALTER QMGR CLWLEXIT('path\amqswlm(clwlFunction)')
```

现在， WebSphere MQ 调用此出口以将所有消息路由到所选队列管理器，而不是使用提供的工作负载管理算法。

构建 IBM WebSphere MQ 应用程序

使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

在 AIX 上构建应用程序

AIX 出版物描述了如何从您编写的程序构建可执行应用程序。

本主题描述了在构建 WebSphere 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++。

使用 WebSphere MQ for AIX 创建可执行应用程序时必须执行的任务因编写源代码所使用的编程语言而异。除了对源代码中的 MQI 调用进行编码外，还必须添加相应的语言语句，以包含您正在使用的语言的 WebSphere MQ for AIX 包含文件。 请熟悉这些文件的内容。 请参阅 第 66 页的『IBM WebSphere 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。

在 AIX 中准备 C 程序

本主题包含有关在 AIX 上准备 C 程序所需的链接库的信息。

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 表示安装 WebSphere MQ 的高级目录。

对于 32 位应用程序：

```
  $ xlc_r -o amqsput_32 amqsput0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

其中 amqsput0 是样本程序。

如果要将 VisualAge C/C++ 编译器用于 C++ 程序，那么必须包含选项 -q namemangling=v5 以在链接库时解析所有 WebSphere MQ 符号。

如果要在仅安装了 WebSphere MQ MQI Client for AIX 的机器上使用这些程序，请重新编译这些程序以将它们与客户机库 (-lmqic) 链接。

链接库

您需要以下库：

- 将程序与 WebSphere MQ 提供的相应库链接。

  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
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简单线程 WebSphere 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 -IMQ_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 -IMQ_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 WebSphere MQ MQI Client for AIX 的机器上使用这些程序，请重新编译这些程序以将它们与客户机库 (-lmqic) 链接。

注：

1. 如果您正在编写可安装服务（请参阅管理，获取更多信息），那么需要链接到非线程应用程序中的 libmqmzf.a 库和线程应用程序中的 libmqmzf_r.a 库。
2. If you are producing an application for external coordination by an XA-compliant transaction manager such as IBM TXSeries, 恩奇纳, or BEA Tuxedo, you need to link to the libmqmxa.a (or libmqmxa64.a if your transaction manager treats the 'long' type as 64 bit) and libmqz.a libraries in a non-threaded application and to the libmqmxa_r.a (or libmqmxa64_r.a) and libmqz_r.a libraries in a threaded application.
3. 您需要将可信应用程序链接到线程 WebSphere MQ 库。 但是，一次只能连接 UNIX and Linux 系统上 WebSphere MQ 上可信应用程序中的一个线程。
4. 必须先链接 WebSphere MQ 库，然后再链接任何其他产品库。

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 表示安装 IBM WebSphere MQ 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）

库文件	程序/出口类型
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 来准备 COBOL 程序

样本 COBOL 程序随 IBM WebSphere 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 \
-I<COBCPY>
```

32 位非线程客户机应用程序

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-I<COBCPY>
```

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 -I<COBCPY>
```

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 -I<COBCPY>
```

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 -I<COBCPY>
```

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 -I<COBCPY>
```

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 -I<COBCPY>
```

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 -I<COBCPY>
```

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>
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 是一个样本程序

有关需要设置的环境变量的说明，请参阅 Micro Focus COBOL 文档。

在 AIX 中准备 CICS 应用程序

在 AIX 中准备 CICS 程序时使用此信息。

提供了 XA 切换模块以使您能够将 CICS 与 IBM WebSphere MQ 链接：

表 58: 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 WebSphere MQ switch load file *amqzsc* 的预构建版本。

始终将 C 事务与线程安全 IBM WebSphere MQ 库 *libmqm_r.a* 链接。 以及具有 COBOL 库 *libmqmcb_r.a* 的 *COBOL* 事务。

您可以在 管理中找到有关支持 CICS 事务的更多信息。

TXSeries CICS 支持

AIX 上的 IBM WebSphere MQ 支持使用 XA 接口的 TXSeries CICS 。 确保将 CICS 应用程序链接到 IBM WebSphere 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。

编写装入到 C 或 COBOL 中同一 CICS 区域的 WebSphere MQ 程序。 不能将 C 和 COBOL MQI 调用组合到同一 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 表示安装 IBM WebSphere MQ 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  \
                -IMQ_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 表示安装 IBM WebSphere MQ 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 WebSphere MQ COBOL 运行时库模块添加到运行时库：

```
cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

   注: 通过 cicsmkcobol，IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 表示安装 IBM WebSphere MQ 的高级目录。

使用标准 CICS 工具构建 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 WebSphere 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.WORKQUEUE ，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。

在 HP Integrity NonStop Server 上构建应用程序

此信息描述了在构建 IBM WebSphere MQ 客户机以使 HP Integrity NonStop Server 应用程序在 HP Integrity NonStop Server 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、COBOL 和 pTAL。

OSS 以及监护器头和公共库

提供 OSS 以及监护器头和公共库的列表。 列出了 OSS 头、OSS 公共可执行文件和公共导入库、监护器头以及监护器公共可执行文件和公共导入库。

第 364 页的『OSS 头』

第 364 页的『OSS 公共可执行文件和公共导入库』

第 365 页的『监护器头』

第 365 页的『监护器公共可执行文件和公共导入库』

OSS 头

表 *59: OSS* 头

Object	位置	描述
cmqbc.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqc.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqfc.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqec.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqpsc.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqxc.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqzc.h	<mqinstall>/inc	IBM WebSphere MQ C 语言头 (OSS)
cmqcobol.cpy	<mqinstall>/inc	IBM WebSphere MQ COBOL 副本 (OSS)
cmqbt.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL 头 (OSS)
cmqcft.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL 头 (OSS)
cmqpst.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL 头 (OSS)
cmqt.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL 头 (OSS)
cmqxt.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL 头 (OSS)

OSS 公共可执行文件和公共导入库

表 *60: OSS* 公共可执行文件和公共导入库

Object	位置	描述
libmqic.so	<mqinstall>/bin	IBM WebSphere MQ 公共可执行文件库（OSS 非线程化）
libmqic_r.so	<mqinstall>/bin	IBM WebSphere MQ 公共可执行文件库（OSS 多线程）
libmqic.so	<mqinstall>/lib	IBM WebSphere MQ 公共导入库（OSS 非线程化）
libmqic_r.so	<mqinstall>/lib	IBM WebSphere MQ 公共导入库（OSS 多线程）
mqicb	<mqinstall>/lib	针对 COBOL 的 IBM WebSphere MQ 公共导入库 (OSS)

监护器头

表 *61:* 监护器头

Object	位置	描述
cmqbch	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqch	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqfch	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqech	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqpsch	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqxch	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqzch	<mqinstall>/inc/G	IBM WebSphere MQ C 语言头（监护器）
cmqcobol	<mqinstall>/inc/G	IBM WebSphere MQ COBOL 副本（监护器）
cmqbt	<mqinstall>/inc/G	IBM WebSphere MQ pTAL 头（监护器）
cmqcft	<mqinstall>/inc/G	IBM WebSphere MQ pTAL 头（监护器）
cmqpst	<mqinstall>/inc/G	IBM WebSphere MQ pTAL 头（监护器）
cmqt	<mqinstall>/inc/G	IBM WebSphere MQ pTAL 头（监护器）
cmqxt	<mqinstall>/inc/G	IBM WebSphere MQ pTAL 头（监护器）

监护器公共可执行文件和公共导入库

表 *62:* 监护器公共可执行文件和公共导入库

Object	位置	描述
mqic	<mqinstall>/bin/G	IBM WebSphere MQ 公共可执行文件库（监护器）
mqicb	<mqinstall>/lib/G	针对 COBOL 的 IBM WebSphere MQ 公共导入库（监护器）

在 HP Integrity NonStop Server 中准备 C 程序

本主题包含在 HP Integrity NonStop Server 中准备 C 程序时需考虑的信息，以及构建应用程序时、使用 OSS C 编译器时和使用监护器 C 编译器时可使用的命令示例。

在 MQ_INSTALLATION_PATH/opt/mqm/samp/bin 目录中提供预编译的 C 程序。 要通过源代码构建样本，请使用 c89 编译器。

您必须将自己的程序与 IBM WebSphere MQ 所提供的相应库链接。 下表列示了在 HP Integrity NonStop Server 上准备 C 程序时必须链接到的库。

表 *63:* . HP Integrity NonStop Server 链接库	
库	**描述**
libmqic.so	非线程化 OSS
libmqic_r.so	多线程的 OSS
mqic	监护器

多线程的本机 IBM WebSphere MQ 应用程序必须使用 Posix 用户线程 (PUT) 功能。 本产品不支持标准 Posix 线程 (SPT)。

使用 OSS C 编译器构建应用程序

本部分包含使用 OSS 编译器时用于构建以 OSS 或监护器为目标的程序的命令示例。

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

以下示例将构建非线程化的 C 客户机 OSS 应用程序：

```
c89 -Wsystype=oss -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
```

以下示例将构建多线程的 C 客户机 OSS 应用程序：

```
c89 -Wsystype=oss -D_PUT_MODEL_ -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic_r -lput
```

以下示例将构建监护器 C 客户机应用程序：

```
c89 -Wsystype=guardian -o /G/vol/subvol/amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
```

使用监护器 C 编译器构建应用程序

本部分包含使用监护器编译器时用于构建以监护器为目标的程序的命令示例。

MQ_INSTALLATION_PATH 表示安装了 IBM WebSphere MQ 的监护器卷和子卷。

以下示例将构建监护器 C 客户机应用程序：

```
CCOMP /in AMQSPUT0/ AMQSPUTC;&
       runnable,systype guardian,nolist,&
       ssv0 "$system.system",&
       ssv1 "MQINSTALLATION_SUBVOL",&
       ld(-LMQINSTALLATION_SUBVOL -lmqic)
```

准备 COBOL 程序

本主题包含为 IBM WebSphere MQ Client for HP Integrity NonStop Server 准备 C 程序时要考虑的信息。 其中包含构建应用程序时、使用 OSS ECOBOL 编译器时以及使用监护器 ECOBOL 编译器时可使用的命令示例。

要使用源代码来构建 COBOL 样本，请使用 ECOBOL 编译器。

下表列示了在 HP Integrity NonStop Server 上准备 COBOL 程序时需要的库。 您必须将自己的程序与 IBM WebSphere MQ 所提供的相应库链接。

表 *64:* . HP Integrity NonStop Server 链接库	
库	**描述**
libmqic.so	非线程化 OSS
mqic	监护器

在运行连接到队列管理器的 COBOL 应用程序时，您必须先将 *SAVE-ENVIRONMENT* 变量设置为 ON。 要将 *SAVE-ENVIRONMENT* 变量设置为 ON：

- 对于 OSS，请输入以下命令：

```
export SAVE-ENVIRONMENT=ON
```

- 对于监护器，请输入以下命令：

```
param SAVE-ENVIRONMENT ON
```

如果未将 *SAVE-ENVIRONMENT* 变量设置为 ON，那么当应用程序尝试连接到队列管理器时，它会失败，原因码为 2058 (080A) (RC2058):MQRC_Q_MGR_NAME_ERROR。

使用 OSS ECOBOL 编译器构建应用程序

本部分包含使用 OSS ECOBOL 编译器时用于构建以 OSS 或监护器为目标的程序的命令示例。

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

以下示例将构建 COBOL 客户机 OSS 应用程序：

```
ecobol -Wsystype=oss
       -Wcobol="ansi;port"
       -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
       -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
       -LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
       -o amq0put0
       MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

以下示例将构建 COBOL 客户机监护器应用程序：

```
ecobol -Wsystype=guardian
       -Wcobol="ansi;port;save all"
       -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
       -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
       -LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
       -o amq0put0
       MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

使用监护器 ECOBOL 编译器构建应用程序

本部分包含使用监护器 ECOBOL 编译器时用于构建以监护器为目标的程序的命令示例。

MQ_INSTALLATION_SUBVOL 表示安装了 IBM WebSphere MQ 的监护器卷和子卷。

以下示例将构建 COBOL 客户机监护器应用程序：

```
ECOBOL /in MQSPUTL/ MQSPUT,MQINSTALLATION_SUBVOL.cmqcobol;
       call-shared;ansi;port;save all;nolist;runnable;
       consult MQINSTALLATION_SUBVOL.mqicb;
       eld(-LMQINSTALLATION_SUBVOL -lmqic)
```

准备 pTAL 程序

了解如何在 HP Integrity NonStop Server 平台上为 IBM WebSphere MQ 客户机构建 pTAL 程序。

要通过源代码构建 pTAL 样本，请使用 EPTAL 编译器。

注:

- pTAL IBM WebSphere MQ 应用程序必须使用采用 C 或 COBOL 语言编写的主例程。
- 只可以在监护器中构建 pTAL 应用程序。

下表列示了在 HP Integrity NonStop Server 上准备 pTAL 程序时需要的库。 您必须将自己的程序与 IBM WebSphere MQ 所提供的相应库链接。

表 *65:* . HP Integrity NonStop Server 链接库	
库	**描述**
mqic	监护器

使用监护器 EPTAL 编译器构建应用程序

本部分包含使用监护器 EPTAL 编译器时用于构建以监护器为目标的程序的命令示例。

MQINSTALLATION_SUBVOL 表示 IBM WebSphere MQ 所安装在的监护器卷和子卷。

pTAL IBM WebSphere MQ 应用程序必须使用采用 C 或 COBOL 语言编写的主例程。

以下示例将构建 pTAL 客户机监护器应用程序:

```
ASSIGN SSV0, $SYSTEM.SYSTEM
ASSIGN SSV1, MQINSTALLATION_SUBVOL

EPTAL /in MQINSTALLATION_SUBVOL.MQSPUTT/ MQSPUTO;nolist

CCOMP /in MQINSTALLATION_SUBVOL.MQSPTMC/ MQSPUT;
       runnable,systype guardian,extensions,nolist,
       ssv0 "$system.system",
       ssv1 "MQINSTALLATION_SUBVOL",
       eld(MQSPUTO -LMQINSTALLATION_SUBVOL -lmqic)
```

在 HP-UX 上构建应用程序

本信息描述了在构建 WebSphere MQ for HP-UX 应用程序以在 HP-UX 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++。

要使用 WebSphere MQ for HP-UX 来创建可执行应用程序，必须执行的任务随编写源代码所用的编程语言而异。 除了在源代码中对 MQI 调用进行编码外，还必须添加相应的语言语句，以包含您正在使用的语言的 WebSphere MQ for HP-UX 包含文件。 请熟悉这些文件的内容。 请参阅 第 66 页的『IBM WebSphere MQ 数据定义文件』以获取完整描述。

在本主题中，我们使用反斜杠 (\) 字符将长命令拆分成多行。 请勿输入此字符；请将每个命令作为一行输入。

在 HP-UX 中准备 C 程序

本主题包含在 HP-UX 中准备 C 程序时要考虑的信息; 以及 IA64 (IPF) 平台的示例。

MQ_INSTALLATION_PATH 表示安装了 WebSphere MQ 的高级目录。

在正常环境中工作。 在 *MQ_INSTALLATION_PATH*/samp/bin 目录中提供了预编译的 C 程序。

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。

要使用 SSL，必须使用 POSIX 线程来构建 HP-UX 上的 WebSphere MQ MQI 客户机。

要考虑的一些示例是:

- 第 369 页的『IA64 (IPF) 平台』
- 第 370 页的『链接库』

IA64 (IPF) 平台

在 IA64(IPF) 平台上构建 amqsput0、cliexit 和 srvexit 的示例。

以下示例在 32 位非线程环境中将样本程序 amqsput0 构建为客户机应用程序：

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic
```

以下示例在 32 位线程环境中将样本程序 amqsput0 构建为客户机应用程序：

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic_r -lpthread
```

以下示例在 64 位非线程环境中将样本程序 amqsput0 构建为客户机应用程序：

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

以下示例在 64 位线程环境中将样本程序 amqsput0 构建为客户机应用程序：

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

以下示例在 32 位非线程环境中将样本程序 amqsput0 构建为服务器应用程序：

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm
```

以下示例在 32 位线程环境中将样本程序 amqsput0 构建为服务器应用程序：

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm_r -lpthread
```

以下示例在 64 位非线程环境中将样本程序 amqsput0 构建为服务器应用程序：

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

以下示例在 64 位线程环境中将样本程序 amqsput0 构建为服务器应用程序：

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

以下示例在 32 位非线程环境中构建了客户机出口 cliexit：

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic
```

以下示例在 32 位线程环境中构建了客户机出口 cliexit：

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

以下示例在 64 位非线程环境中构建了客户机出口 cliexit：

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

以下示例在 64 位线程环境中构建了客户机出口 cliexit：

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

以下示例在 32 位非线程环境中构建了服务器出口 srvexit：

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm
```

以下示例在 32 位线程环境中构建了服务器出口 srvexit：

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

以下示例在 64 位非线程环境中构建了服务器出口 srvexit：

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c
-IMQ_INSTALLATION_PATHMQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

以下示例在 64 位线程环境中构建了服务器出口 srvexit：

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

链接库

您需要将程序与 WebSphere MQ 提供的相应库链接。

下表显示了在不同环境中使用的库。

硬件平台	线程或非线程环境	程序/出口类型	库文件
IA64 (IPF)	线程化	针对 C 的服务器及客户机	libmqm_r.so
IA64 (IPF)	线程化	针对 C 的客户机	libmqic_r.so
IA64 (IPF)	非线程	针对 C 的服务器及客户机	libmqm.so
IA64 (IPF)	非线程	针对 C 的客户机	libmqic.so

注:

1. 如果您正在编写可安装服务（请参阅管理，获取更多信息），您需要链接到 `libmqmzf.sl` 库。
2. If you are producing an application for external coordination by an XA-compliant transaction manager such as IBM TXSeries 恩奇纳, or BEA Tuxedo, you need to link to the `libmqmxa.sl` (or `libmqmxa64.sl` if your transaction manager treats the 'long' type as 64 bit) and `libmqz.sl` libraries in a non-threaded application and to the `libmqmxa_r.sl` (or `libmqmxa64_r.sl`) and `libmqz_r.sl` libraries in a threaded application.
3. 必须先链接 WebSphere MQ 库，然后再链接任何其他产品库。

在 HP-UX 中准备 COBOL 程序

了解如何在 HP-UX 中准备 COBOL 程序，如何在 IA64 (IPF) 平台上将 Micro Focus Server Express 与 WebSphere MQ 配合使用，以及如何在 WebSphere MQ MQI 客户机环境中运行程序。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

用户备注

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。 声明结构的副本文件位于 *MQ_INSTALLATION_PATH*/inc 中：

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

编译 32 位程序：

```
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

编译 64 位程序：

```
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

其中 `amqsput` 是样本程序

确保您已指定了足够的运行时堆栈大小；建议的最小值为 16 KB。

您需要将程序与 WebSphere MQ 提供的相应库链接。 下表显示了在不同环境中使用的库。

硬件平台	程序/出口类型	库文件
IA64 (IPF)	针对 COBOL 的服务器	libmqmcb.so

硬件平台	程序/出口类型	库文件
IA64 (IPF)	针对 COBOL 的客户机	libmqicb.so
IA64 (IPF)	线程应用程序	libmqmcb_r.so

在 IA64 (IPF) 平台上将 Micro Focus Server Express 与 WebSphere MQ 配合使用

有关将 Micro Focus Server Express 与 HP/IPF 平台上的 WebSphere MQ 结合使用的详细信息，请参阅 第 373 页的『 WebSphere MQ for HP-UX on IA64 (IPF) 支持的地址空间模型』 。

要在 WebSphere MQ MQI 客户机环境中运行的程序

如果使用 LU 6.2 将 MQI 客户机连接到服务器，请将应用程序链接到 libsna.a（SNAplusAPI 产品的一部分）。 在编译和链接命令中使用 `-lV3` 和 `-lstr` 选项。

- The `-lV3` option gives your program access to the AT&T signaling library (the SNAplusAPI uses AT&T signals)
- `-lstr` 选项可将程序链接到数据流组件

在 HP-UX 中准备 CICS 程序

了解如何在 HP-UX 中构建 CICS 事务程序。

要构建样本 CICS 事务 amqscic0.ccs，请运行以下命令:

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

提供了 XA 切换模块，使您能够将 CICS 与 WebSphere MQ 链接:

表 *66: CICS* 应用程序的基本代码 *(HP-UX)*		
描述	**C（源代码）**	**C（可执行代码）**
XA 初始化例程	amqzscix.c	amqzsc

您可以在 管理中找到有关支持 CICS 事务的更多信息。

TXSeries CICS 支持

HP-UX 上的 WebSphere MQ 支持使用 XA 接口的 TXSeries CICS 。 确保 CICS 应用程序链接到 MQ 库的线程版本。

编写装入到 C 或 COBOL 中同一 CICS 区域的 WebSphere MQ 程序。 不能将 C 和 COBOL MQI 调用组合到同一 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`。

CICS C 样本事务

CICS WebSphere MQ 事务的样本 C 源由 AMQSCIC0.CCS。 事务从传输队列 SYSTEM.SAMPLE.CICS.WORKQUEUE ，并使用消息的传输头中包含的队列名称将它们放在本地队列上。 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。

使用 Micro Focus COBOL 准备 CICS COBOL 程序

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 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 WebSphere MQ COBOL 运行时库模块添加到运行时库:

```
cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

注: 通过 `cicsmkcobol`， WebSphere MQ 不允许您从 COBOL 应用程序以 C 编程语言进行 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`。 在移动这些函数之后，在为 CICS 构建 COBOL 应用程序时，请勿包含 WebSphere 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 WebSphere 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 HP-UX 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）
- WebSphere 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
```

WebSphere MQ for HP-UX on IA64 (IPF) 支持的地址空间模型

HP-UX 提供了多个可供 WebSphere MQ 应用程序利用的地址空间模型。

HP-UX 支持两个地址空间模型:

- MGAS-大部分全局地址空间 (这是缺省值，由 WebSphere MQ 使用)
- MPAS - 主要专用地址空间

连接到 WebSphere MQ 的应用程序可以使用 MGAS 或 MPAS 地址空间模型。 使用 MPAS 模型构建的应用程序使用共享内存连接到 WebSphere MQ ，由于将 WebSphere MQ 使用的共享内存页面映射到 MPAS 程序的虚拟地址空间的效率低下，这些应用程序可能会产生较小的性能成本。

通过 Micro Focus Server Express 构建的 COBOL 应用程序在缺省情况下使用 MPAS 模型。

您可以使用 **chatr** 程序来检查和更改程序使用的寻址模型。

如果从 32 位 MPAS 程序连接到 WebSphere MQ 时迁到问题，请考虑使用 MGAS 寻址模型，或者将应用程序构建为 64 位 MPAS 应用程序而不是 32 位 MPAS 应用程序。

有关 MGAS 和 MPAS 地址空间模型的更多详细信息可在 HP-UX 文档中找到。

在 Linux 上构建应用程序

此信息描述了在为 Linux 应用程序构建要运行的 WebSphere MQ 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++。

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 表示安装 WebSphere MQ 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 WebSphere MQ 提供的相应库链接。

  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. 如果您正在编写可安装服务（请参阅管理，获取更多信息），您需要链接到 libmqmzf.so 库。
2. If you are producing an application for external coordination by an XA-compliant transaction manager such as IBM TXSeries 恩奇纳, or BEA Tuxedo, you need to link to the libmqmxa.so (or libmqmxa64.so if your transaction manager treats the 'long' type as 64 bit) and libmqz.so libraries in a non-threaded application and to the libmqmxa_r.so (or libmqmxa64_r.so) and libmqz_r.so libraries in a threaded application.
3. 必须先链接 WebSphere MQ 库，然后再链接任何其他产品库。

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 表示安装 WebSphere MQ 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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm_r -lpthread
```

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 表示安装 WebSphere MQ 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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
lmqm_r -lpthread
```

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 表示安装 WebSphere MQ 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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_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
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

在 Linux 中准备 COBOL 程序

了解如何在 Linux 中准备 COBOL 程序，以及如何使用 Micro Focus COBOL 准备 COBOL 程序。

MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 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
```

（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）

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>
export LIB=MQ_INSTALLATION_PATHlib:$LIB
```

要使用 Micro Focus COBOL 编译 32 位 COBOL 程序（如果获得支持），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。

在 Solaris 上构建应用程序

本信息描述了在为 Solaris 应用程序构建 WebSphere MQ 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 66 页的『IBM WebSphere 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。

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 表示安装 WebSphere MQ 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 WebSphere MQ MQI Client for Solaris 的机器上使用这些程序，请编译这些程序以将它们与客户机库 (-lmqic) 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和链接。 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 标准的操作配置的 32 位 Solaris x86 SSL 和 TLS 客户机将失败。 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

必须与适合于您的应用程序类型的 WebSphere MQ 库链接:

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. 如果要通过 XA 兼容的事务管理器 (例如 IBM TXSeries 恩奇纳或 BEA Tuxedo) 生成用于外部协调的应用程序，那么必须链接到 libmqmxa.so (或者 libmqmxa64.so 如果事务管理器将 "long" 类型视为 64 位) 和 libmqz.so 库。
3. 必须先链接 WebSphere MQ 库，然后再链接任何其他产品库。

在 *x86-64* 上构建应用程序

本主题包含用于在 x86-64 平台上的各种环境中构建程序的命令示例。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

C 客户机应用程序，32 位

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

在 *SPARC* 上构建应用程序

本主题包含用于在 SPARC 平台上的各种环境中构建程序的命令示例。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

C 客户机应用程序，32 位

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
```

```
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_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
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

在 Solaris 中准备 COBOL 程序

了解有关在 Solaris 中准备 COBOL 程序的信息。

MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 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>"
```

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

在 Solaris 中准备 CICS 程序

了解有关在 Solaris 中准备 CICS 程序的信息。

提供了 XA 切换模块，使您能够将 CICS 与 WebSphere MQ 链接：

表 *67: CICS* 应用程序的基本代码 *(Solaris)*

描述	**C（源代码）**	**C（可执行代码）**
XA 初始化例程	amqzscix.c	amqzsc - TXSeries for Solaris

始终将事务与线程安全 WebSphere MQ 库 libmqm.so 链接。

您可以在 管理中找到有关支持 CICS 事务的更多信息。

TXSeries CICS 支持

WebSphere MQ for Solaris 支持使用 XA 接口的 TXSeries CICS 。

编写装入到 C 或 COBOL 中同一 CICS 区域的 WebSphere MQ 程序。 不能将 C 和 COBOL MQI 调用组合到同一 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 表示安装 WebSphere MQ 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 WebSphere MQ COBOL 运行时库模块添加到运行时库：

```
cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
```

注: 通过 `cicsmkcobol`， WebSphere MQ 不允许您从 COBOL 应用程序以 C 编程语言进行 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 WebSphere 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 WebSphere 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）
- WebSphere 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 WebSphere MQ 事务的样本 C 源由 AMQSCIC0.CCS。 事务从传输队列 SYSTEM.SAMPLE.CICS.WORKQUEUE ，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。

在 Windows 系统上构建应用程序

Windows 系统出版物描述了如何从您编写的程序构建可执行应用程序。

本主题描述在构建 WebSphere MQ for Windows 应用程序以在 Windows 系统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 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++。

使用 WebSphere MQ for Windows 创建可执行应用程序时必须执行的任务因编写源代码时使用的编程语言而异。 除了在源代码中对 MQI 调用进行编码外，还必须添加相应的语言语句，以包含您正在使用的语言的 WebSphere MQ for Windows 包含文件。 请熟悉这些文件的内容。 请参阅 第 66 页的『IBM WebSphere MQ 数据定义文件』以获取完整描述。

在 Windows 上构建 64 位应用程序

在 IBM WebSphere MQ for Windows Version 7.5 上同时支持 32 位和 64 位应用程序。 IBM WebSphere 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 WebSphere MQ 库的 32 位和 64 位版本：

表 *68: IBM WebSphere 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 表示安装 WebSphere MQ 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，那么必须确保环境配置为使用 *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。

在 Windows 中准备 C 程序

在典型 Windows 环境中工作; WebSphere MQ for Windows 不需要任何特殊内容。

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。

- 将程序与 WebSphere MQ 提供的相应库链接：

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 表示安装 WebSphere MQ 的高级目录。

以下命令提供了编译样本程序 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。
- 如果要通过符合 XA 的事务管理器 (例如 IBM TXSeries Encina 或 BEA Tuxedo) 生成用于外部协调的应用程序，那么需要链接到 mqmxa.lib 或 mqmxa.lib 库。
- 如果要编写 CICS 出口，请链接到 mqmcics4.lib 库。
- 必须先链接 WebSphere MQ 库，然后再链接任何其他产品库。

- DLL 必须在您已指定的路径 (PATH) 中。
- 如果尽可能使用小写字符，那么可以从 WebSphere MQ for Windows 移至 UNIX and Linux 系统上的 WebSphere MQ，其中需要使用小写。

准备 CICS 和 Transaction Server 程序

CICS WebSphere MQ 事务的样本 C 源由 AMQSCIC0.CCS。 您可以使用标准 CICS 工具进行构建。 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=-IMQ_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 表示安装 WebSphere MQ 的高级目录。

这在 *Transaction Server for Windows NT Application Programming Guide (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。

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
   ```

   （对于 32 位应用程序）以及：

   ```
   CopyBook64
   ```

   （对于 64 位应用程序）。

MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 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 WebSphere MQ 提供的以下库之一：

库文件	程序或出口类型
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb	IBM COBOL 的 32 位服务器
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb	Micro Focus COBOL 的 32 位服务器
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb	IBM COBOL 的 32 位客户机
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb	Micro Focus COBOL 的 32 位客户机
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb	IBM COBOL 的 64 位服务器
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb	Micro Focus COBOL 的 64 位服务器
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb	IBM COBOL 的 64 位客户机
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb	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 WebSphere MQ 库之前。

您可以根据程序使用 IBM COBOL Set 编译器或 Micro Focus COBOL 编译器：

- 以 amqi 开头的程序适合于 IBM COBOL Set 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。

IBM 和 Micro Focus COBOL

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 WebSphere MQ Micro Focus COBOL 程序。

例如，要使用 IBM VisualAge COBOL 编译样本程序 amq0put0:

1. 设置 SYSLIB 环境变量以包含 IBM WebSphere MQ VisualAge COBOL 副本的路径 (在一行上输入以下代码):

   ```
   set SYSLIB=MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook\VAcobol;%SYSLIB%
   ```

2. 要在 IBM WebSphere MQ 服务器上使用:

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqmcbb.lib"
   ```

3. 要在 IBM WebSphere MQ 客户机上使用:

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqiccbb.lib"
   ```

   注: 虽然必须使用编译器选项 CALLINT(SYSTEM)，但是此选项对于 cob2 为缺省选项。

例如，要使用 Micro Focus COBOL 编译样本程序 amq0put0，请执行以下操作：

1. 将 COBCPY 环境变量设置为指向 IBM WebSphere 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 WebSphere MQ 服务器上使用：

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - 或者链接对象文件以在 IBM WebSphere 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
   ```

这在 *Transaction Server for Windows NT V4 Application Programming* 指南中进行了描述。

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
```

在 Windows 中准备 Visual Basic 程序

考虑在 Windows 上使用 Visual Basic 程序时，请使用此信息。

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 71 页的『采用 Visual Basic 进行编码』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 WebSphere MQ 服务器还是客户机。 在 Visual Basic 项目中将 *MqType* 设置为 1（表示服务器）或 2（表示客户机），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
```

SSPI 安全出口

WebSphere MQ for Windows 为 WebSphere MQ MQI 客户机和 WebSphere MQ 服务器提供了安全出口。 这是一个通道出口程序，它使用安全服务编程接口 (SSPI) 为 WebSphere MQ 通道提供认证。 SPI 提供 Windows 系统的集成安全设施。

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 142 页的『在 Windows 系统上使用 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 69: 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 391 页的『WebSphere 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。

WebSphere MQ 访问控制和 Windows 主体

WebSphere MQ 提供的访问控制基于用户和组。 Windows 提供的认证基于主体，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。

SPI 安全出口使用相关 Windows 主体进行认证。 如果 Windows 认证成功，那么出口会将与 Windows 主体相关联的用户标识传递给 WebSphere MQ 以进行访问控制。

与认证相关的 Windows 主体因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 Kerberos 认证，在 CLNTCONN 通道上， Windows 主体是与正在运行的进程相关联的用户标识。 否则， Windows 主体是通过将以下前缀添加到 QueueManager 名称而形成的 servicePrincipal 名称。

```
ibmMQSeries/
```

将轻量级目录访问协议服务与 WebSphere MQ for Windows 配合使用

本主题说明了什么是目录服务以及目录访问协议 (DAP) 所扮演的角色。 它还说明了 WebSphere MQ 应用程序如何使用轻量级目录访问协议 (LDAP) 目录 (使用样本程序作为指南)。

注: 样本程序是为已经熟悉 LDAP 的人设计的。

以下主题提供了有关目录服务， LDAP 以及将 LDAP 与 WebSphere MQ 配合使用的更多信息。

- 第 392 页的『目录服务』
- 第 392 页的『轻量级目录访问协议 (LDAP)』
- 第 392 页的『将 LDAP 与 WebSphere MQ 配合使用』

目录服务

目录是对象相关信息的存储库，其组织方式使您能够轻松地找到有关特定对象的信息。

常见示例是电话簿，其中存储有关人员和公司的信息 (地址和电话号码)。 另一个示例是电子邮件系统的地址簿，其中为人员存储了电子邮件地址和可选的其他信息 (例如电话号码)。

在计算机系统上，目录可以存储有关计算机资源的信息，例如打印机或共享磁盘。 例如，您可以使用目录来查找最近的彩色打印机所在的位置。 在 WebSphere MQ 应用程序中，可以使用目录来提供应用程序服务 (例如，应收账款处理) 与用于需要该服务的消息 (可能通过队列名称及其主机队列管理器名称标识) 的队列之间的关联。

目录实现为客户机/服务器系统，其中目录服务器保存来自客户机的所有信息和应答请求。 客户机可以是直接向用户提供信息的用户界面程序，也可以是需要查找资源以完成其工作的应用程序。 目录服务由目录服务器，管理程序以及配置，更新和读取目录所需的客户机库和程序组成。

轻量级目录访问协议 (LDAP)

存在许多目录服务，例如 Novell Directory Services， DCE Cell Directory Service， Banyan StreetTalk, Windows Directory Services， X.500 以及与电子邮件产品关联的地址簿服务。 国际标准组织 (ISO) 建议将 X.500 作为全球目录服务的标准。 它需要一个 OSI 协议栈来进行通信，很大程度上因为这样，它的使用已经被限制在大型组织和学术机构。 X.500 目录服务器使用目录访问协议 (DAP) 与其客户机进行通信。

LDAP (轻量级目录访问协议) 已创建为 DAP 的简化版本。 它更容易实现，省略了 DAP 的一些较少使用的功能，并在 TCP/IP 上运行。 由于这些变化，它很快被采用为大多数目的的目录访问协议，取代了以前使用的多种专有协议。 LDAP 客户机仍然可以通过网关访问 X.500 服务器 (X.500 仍然需要 OSI 协议堆栈)，或者越来越多的 X.500 实现通常包含对 LDAP 的本机支持以及 DAP 访问。

可以分发 LDAP 目录，并且可以使用复制来实现对其内容的高效访问。

有关 LDAP 的更完整描述，请参阅 IBM Redbooks 出版物 了解 *LDAP*。

将 LDAP 与 WebSphere MQ 配合使用

在 WebSphere MQ 配置中，定义消息和传输队列的信息存储在本地。 这意味着在 WebSphere MQ 网络中分发了各种定义，并且没有可供浏览的此信息的中央目录。 WebSphere MQ 应用程序之间的远程消息传递通常是通过使用远程队列的本地定义来实现的。 应用程序首先使用远程队列的本地定义中指定的名称发出 MQOPEN 调用。 要将消息放在远程队列上，应用程序将发出 MQPUT，指定从 MQOPEN 调用返回的句柄。 远程队列定义提供目标队列，目标队列管理器和 (可选) 传输队列的名称。 在此技术中，应用程序必须在运行时知道本地队列定义中指定的名称。

先前的变体避免使用远程队列的本地定义。 应用程序可以指定完整目标队列名称，其中包括作为 MQOPEN 一部分的远程队列管理器名称。 因此，应用程序必须在运行时知道这两个名称。 必须使用本地队列定义正确配置本地队列管理器，并使用适当命名 (或缺省) 的传输队列和传递到目标的关联通道。

在将源队列管理器和目标队列管理器都定义为同一集群的成员的情况下，可以忽略前两种方案的传输队列和通道方面。 如果目标传输队列是集群队列，那么也不需要远程队列的本地定义。 但是，与先前描述的情况类似，应用程序仍必须知道目标队列的名称。

目录服务可用于除去此应用程序对队列名称 (或队列和队列管理器名称的组合) 的依赖关系。 应用程序条件与 WebSphere MQ 对象名之间的映射可以保存在目录中，并且可以动态更新，并且独立于应用程序。 在运行时，要发送消息的 WebSphere MQ 应用程序首先使用基于应用程序的条件 (例如 :service_name = "应收账款") 来查询目录，检索相关的 WebSphere MQ 对象名，然后在 MQOPEN 调用中使用这些返回的值。

另一个使用目录的示例是对于具有许多小型库或办公室的公司，可以使用 WebSphere MQ MQI 客户机将消息发送到位于较大办公室中的 WebSphere MQ 服务器。 客户机需要知道主机的名称， MQI 通道以及它们将消息发送到的每个服务器的队列名称。 有时，可能需要将 WebSphere MQ 服务器移至另一台机器; 与该服务器通信的每个客户机都需要了解更改。 可以使用 LDAP 目录服务来存储主机的名称 (以及通道和队列名称) ，只要客户机程序想要将消息发送到服务器，就可以从目录中检索信息。 在这种情况下，仅当主机名 (或通道或队列名称) 发生更改时，才需要更新目录。

应用程序消息的多个目标可以存储在一个目录中，而选择的目标取决于可用性或负载共享注意事项。

WebSphere MQ 还可以使用 LDAP 目录来存储用于安全套接字层 (SSL) 的认证信息。 WebSphere MQ Java 类还可以将信息存储在 LDAP 目录中。

LDAP 样本程序

样本程序是为熟悉 LDAP 且可能已使用 LDAP 的人员设计的。 它旨在显示 WebSphere MQ 应用程序如何使用 LDAP 目录。

构建样本程序

此程序仅在使用 TCP/IP 的 Windows 上进行了构建和测试。 以及 第 386 页的『在 Windows 中准备 C 程序』中提到的一般注意事项，请注意以下几点:

- 此程序设计为作为客户机程序运行，因此应与 MQIC.LIB 库。
- 除了 WebSphere MQ 头文件和库外，必须使用 LDAP 客户机头文件和库来构建此程序。

  例如，使用 IBM eNetwork 客户机，将程序与 LIBLDAPSTATICE.LIB 和 LIBLBERSTATICSSL.LIB 库。

配置目录

必须先使用样本数据配置 LDAP 目录服务器，然后才能运行样本程序。

`tools\c\samples` 目录中的文件 MQuser.ldif 包含 LDIF (LDAP 数据交换格式) 中的一些样本数据。 您可以编辑此文件以满足您的需要。 它包含一家名为 MQuser 的虚构公司的数据，该公司有一个由三个办公室组成的运输部。 其中每个办公室都有一台运行 WebSphere MQ 服务器的机器。

至少，您必须编辑包含运行 WebSphere MQ 服务器的机器的主机名的三行: 第 18,27 和 36 行:

```
host: LondonHost
  ...
host: SydneyHost
  ...
host: WashingtonHost
```

必须将 `LondonHost`， `SydneyHost` 和 `WashingtonHost` 更改为运行 WebSphere MQ 服务器的三台机器的名称。 如果需要，还可以更改通道和队列名称 (样本使用系统缺省值的名称)。 您可能还希望增加或减少样本数据中的办公室数。

配置 *IBM Tivoli* 目录服务器

Refer to the IBM Tivoli Directory Server (ITDS) Administrator's Guide for information about installing the directory. 在主题 `Installing and Configuring Server` 中，完成部分 `Installing Server` 和

Basic Server Configuration。 如有必要，请阅读主题 Administrator Interface 以熟悉界面的工作方式。

在主题 Configuring - How Do I 中，遵循指示信息以启动管理员，然后完成 Configure Database 部分并创建缺省数据库。 跳过部分 Configure replica 并使用部分 Work with Suffixes，添加后缀 o=MQuser。

在向数据库添加任何条目之前，必须通过添加一些属性定义和对象类定义来扩展目录模式。 This is described in the IBM Tivoli Directory Server Administrator's Guide in the chapter Reference Information under the section Directory Schema. 包含两个样本文件以帮助您执行此操作。 文件 mq.at.conf 包含必须添加到文件 ?etc?slapd.at.conf 的属性定义。 要执行此操作，请通过编辑 slapd.at.conf 并添加行来包含样本文件:

```
include <pathname>/mq.at.conf
```

或者，您可以编辑文件 slapd.at.conf，并将样本文件的内容直接添加到其中，即，添加以下行:

```
# MQ attribute definitions
attribute mqChannel           ces   mqChannel        1000   normal
attribute mqQueueManager      ces   mqQueueManager   1000   normal
attribute mqQueue             ces   mqQueue          1000   normal
attribute mqPort              cis   mqPort           64     normal
```

同样，对于对象类定义，您可以通过编辑 etc?slapd.oc.conf 来包含样本文件，并添加以下行:

```
include <pathname>/mq.oc.conf
```

或者，您可以将样本文件的内容直接添加到 slapd.oc.conf,，即，添加以下行:

```
# MQ object classdefinition
objectclass mqApplication
      requires
              objectClass,
              cn,
              host,
              mqChannel,
              mqQueue
      allows
              mqQueueManager,
              mqPort,
              description,
              l,
              ou,
              seeAlso
```

现在，您可以启动目录服务器 ("管理"， "服务器" 和 "启动") 并向其添加样本条目。 要添加样本条目，请转至 "管理"， "添加管理员的条目" 页面，输入样本文件 MQuser.ldif 的完整路径名，然后单击 "提交"。

目录服务器现在正在运行，并装入了适合运行样本程序的数据。

配置 *Netscape* 目录服务器

使用 "Netscape Server 管理" 页面，单击 **创建新的 Netscape Directory Server**。

现在应该向您显示包含配置信息的表单。 将 "目录后缀" 更改为 **o = MQuser**，并为不受限制的用户添加密码。 您还可以更改任何其他信息以适合您的安装。 单击 **确定**，应成功创建目录。 单击 **返回到服务器管理** 并启动目录服务器。 单击目录名称以启动新目录的 Directory Server 管理服务器。

在将任何条目添加到数据库之前，请通过添加一些属性定义和对象类定义来扩展目录模式。 单击 "目录服务器" 页面的 **模式** 选项卡。 现在，将向您显示一个允许您添加新属性的表单。 添加以下属性 (将所有属性的属性 OID 留空):

```
Attribute Name              Syntax
--------------              ------
mqChannel                   Case Exact String
```

```
mqQueueManager              Case Exact String
mqQueue                     Case Exact String
mqPort                      Integer
```

通过单击侧面板中的 **创建 ObjectClass** 来添加新的 objectClass 。 输入 **mqApplication** 作为 ObjectClass 名称，选择 **applicationProcess** 作为父 ObjectClass ，并将 **ObjectClass OID** 留空。 现在将一些属性添加到 objectClass。 选择 **host**， **mqChannel** 和 **mqQueue** 作为必需属性，并选择 **mqQueueManager** 和 **mqPort** 作为允许的属性。 按 **新建 ObjectClass** 按钮以创建 objectClass。

要添加样本数据，请单击 **数据库管理** 选项卡，然后从侧面板中选择 **添加条目** 。 输入样本数据文件 `<pathname>\MQuser.ldif` 的路径名，输入密码，然后单击 **确定**。

样本程序以未经授权的用户身份运行，缺省情况下， Netscape Directory 不允许未经授权的用户搜索目录。 通过单击 **访问控制** 选项卡来更改此值。 输入无限制用户的密码，然后单击 **确定** 以装入目录的访问控制条目。 这些当前应该为空。 按 **新建 ACI** 按钮以创建新的访问控制条目。 在显示的输入框中，单击 **拒绝** (带有下划线) ，然后在生成的对话框中，将其更改为 **允许**。 添加名称 (例如， **MQuser-access**) ，然后单击 **选择后缀** 以选择 **o = MQuser**。 输入 **o = MQuser** 作为目标，输入无限制用户的密码，然后单击 **提交**。

目录服务器现在正在运行，并装入了适合运行样本程序的数据。

运行样本程序

现在，您应该正在运行 LDAP Directory Server 并使用样本数据进行填充。 该数据指定三个主机，所有这些主机都应运行 WebSphere MQ 服务器。 确保缺省队列管理器正在每台机器上运行 (除非您更改了样本数据以指定不同的队列管理器)。

此外，在每台机器上启动 WebSphere MQ 侦听器程序; 样本使用带有缺省 WebSphere MQ 端口号的 TCP/IP ，因此您可以使用以下命令来启动侦听器:

```
runmqlsr -t tcp
```

要测试样本，您可能还希望运行一个程序来读取到达每个 WebSphere MQ 服务器的消息，例如，可以使用 amqstrg 样本程序:

```
amqstrg SYSTEM.DEFAULT.LOCAL.QUEUE
```

样本程序使用三个环境变量，一个是必需的，两个是可选的。 必需变量为 LDAP_BASEDN ，用于指定目录搜索的基本专有名称。 要处理样本数据，请将其设置为 `ou=Transport, o=MQuser`，例如，在 Windows 系统上的命令提示符处输入:

```
set LDAP_BASEDN=ou=Transport, o=MQuser
```

可选变量为 LDAP_HOST 和 LDAP_VERSION。 LDAP_HOST 变量指定运行 LDAP 服务器的主机的名称; 如果未指定，那么缺省为本地主机。 LDAP_VERSION 变量指定要使用的 LDAP 协议的版本，可以是 2 或 3。 现在，大多数 LDAP 服务器都支持版本 3 的协议; 它们都支持旧版本 2。 此样本与协议的任一版本同样有效，如果未指定此样本，那么缺省为 V 2。

现在，您可以通过输入程序名后跟要向其发送消息的 WebSphere MQ 应用程序的名称来运行样本，对于样本数据，应用程序名称为 London ，悉尼和 Washington。 例如，要将消息发送到伦敦应用程序:

```
amqsldpc London
```

如果程序无法连接到 WebSphere MQ 服务器，那么将显示相应的错误消息。 If it connects successfully you can start typing messages, each line that you type (terminated by <return> or <enter>) is sent as a separate message, an empty line ends the program.

程序设计

该程序有两个不同的部分: 第一部分使用环境变量和命令行值来查询 LDAP 目录服务器; 第二部分使用从目录返回的信息建立 WebSphere MQ 连接并发送消息。

根据使用的是 LDAP 版本 2 还是 3，程序第一部分中使用的 LDAP 调用略有不同， LDAP 客户机库随附的文档详细描述了这些调用。 本部分提供了简要描述。

程序的第一部分检查它是否已正确调用并读取环境变量。 然后，它与指定主机上的 LDAP 目录服务器建立连接:

```
if (ldapVersion == LDAP_VERSION3)
{
  if ((ld = ldap_init(ldapHost, LDAP_PORT)) == NULL)
     ...
}
else
{
  if ((ld = ldap_open(ldapHost, LDAP_PORT)) == NULL )
     ...
}
```

当已建立连接时，程序使用 "ldap_set_option" 调用在服务器上设置一些选项，然后通过绑定到它来向服务器认证自身:

```
if (ldapVersion == LDAP_VERSION3)
{
  if (ldap_simple_bind_s(ld, bindDN, password) != LDAP_SUCCESS)
     ...
}
else
{
  if (ldap_bind_s(ld, bindDN, password, LDAP_AUTH_SIMPLE) !=
      LDAP_SUCCESS)
     ...
}
```

在样本程序 bindDN 和 password 中设置为 NULL，这意味着程序将自己认证为匿名用户，即，它没有任何特殊访问权，只能访问公开可用的信息。 在实践中，大多数组织会限制对其存储在目录中的信息的访问权，以便只有授权用户才能访问该信息。

绑定调用 ld 的第一个参数是一个句柄，用于在整个程序的其余部分中标识此特定 LDAP 会话。 认证后，程序将在目录中搜索与应用程序名称匹配的条目:

```
rc = ldap_search_s(ld,                 /* LDAP Handle             */
                   baseDN,             /* base distinguished name */
                   LDAP_SCOPE_ONELEVEL, /* one-level search        */
                   filterPattern,      /* filter search pattern   */
                   attrs,              /* attributes required     */
                   FALSE,              /* NOT attributes only     */
                   &ldapResult);       /* search result           */
```

这是对直接返回结果的服务器的简单同步调用。 还有其他类型的搜索更适合复杂查询或期望大量结果的情况。 搜索的第一个参数是标识会话的句柄 ld 。 第二个参数是基本专有名称，它指定在目录中搜索的开始位置，而第三个参数是搜索的范围，即搜索相对于起始点的条目。 这两个参数一起定义搜索目录中的哪些条目。 下一个参数 filterPattern 指定要搜索的内容。 attrs 参数列出了在找到对象时要从该对象返回的属性。 下一个属性表示我们是否只需要属性或它们的值; 将此属性设置为 FALSE 表示我们需要属性值。 最终参数用于返回结果。

结果可能包含许多目录条目，每个目录条目都具有指定的属性及其值。 我们必须从结果中提取我们想要的值。 在这个样本程序中，我们只期望找到一个条目，所以我们只看结果中的第一个条目:

```
ldapEntry = ldap_first_entry(ld, ldapResult);
```

此调用返回表示第一个条目的句柄，我们设置了一个 for 循环以从条目中抽取所有属性:

```
for (attribute = ldap_first_attribute(ld, ldapEntry, &ber);
     attribute != NULL;
     attribute = ldap_next_attribute(ld, ldapEntry, ber ))
{
```

对于这些属性中的每个属性，我们都会抽取与其关联的值。 同样，我们只期望每个属性有一个值，因此我们只使用第一个值; 我们确定我们有哪个属性，并将该值存储在相应的程序变量中:

```
values = ldap_get_values(ld, ldapEntry, attribute);
if (values != NULL && values[0] != NULL)
{
  if (stricmp(attribute, MQ_HOST_ATTR) == 0)
  {
    mqHost = strdup(values[0]);
    ...
```

最后，我们通过释放内存 (ldap_value_free， ldap_memfree， ldap_msgfree) 进行整理，并通过从服务器取消绑定 来关闭会话:

```
ldap_unbind(ld);
```

我们检查是否从目录中找到了所需的所有 WebSphere MQ 值，如果找到，请调用 sendMessages() 以连接到 WebSphere MQ 服务器并发送 WebSphere MQ 消息。

样本程序的第二部分是包含所有 WebSphere MQ 调用的 sendMessages() 例程。 这是在 amqsput0 样本程序上建模的，差异在于程序的参数已扩展，并且使用 MQCONNX 来代替 MQCONN 调用。

为 IBM WebSphere 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 IBM WebSphere MQ Telemetry 开发应用程序。 使用设备的 IBM WebSphere MQ Telemetry 守护程序简化将许多小型设备连接到 IBM WebSphere MQ 的过程。

相关概念

WebSphere MQ Telemetry

用于监视和控制目的的遥测概念与方案

相关任务

安装 WebSphere MQ Telemetry

管理 WebSphere MQ Telemetry

WebSphere MQ Telemetry 故障诊断

相关参考

WebSphere MQ Telemetry 参考

IBM WebSphere MQ Telemetry 样本程序

提供了样本脚本来演示 MQ Telemetry Transport v3 客户机应用程序的基本用法。 使用这些脚本来发布消息并预订主题。

开始之前

启动遥测 (MQXR) 服务以运行样本程序。

用户标识必须是 mqm 用户组的成员。

首先运行 SampleMQM 脚本，然后运行 MQTTV3Sample 脚本以执行发布和预订。 运行 CleanupMQM 样本脚本以删除由 SampleMQM 脚本创建的队列管理器。

由于 SampleMQM 脚本会创建并使用名为 QM1 的队列管理器，因此请勿在具有 QM1 队列管理器的系统上未更改地运行。 所做的任何更改都可能影响现有队列管理器的配置。

关于此任务

- SampleMQM 应用程序创建并启动名为 QM1 的支持遥测的队列管理器。 该脚本还为 QM1 设置缺省传输队列，并创建和启动在端口 1883 上侦听的缺省通道。 此通道不会对与其连接的客户机执行认证。 该通道具

有消息通道代理程序用户标识 (MCAUSER) 属性，在 Windows 系统上设置为 "访客" 或在 Linux 系统上设置为 "无人"。 连接到通道的客户机将被视为用户 "访客" 或用户 "无人"，这取决于运行该通道的系统。 该脚本授权 Windows 系统上的 "访客" 和 Linux 系统上的 "无人" 能够发布和预订 QM1 上的任何主题

- MQTTV3Sample 应用程序保存在以下位置中;
  - 在 Windows *MQ_INSTALLATION_PATH*\mqxr\samples 上

    其中 *MQ_INSTALLATION_PATH* 是安装 IBM WebSphere MQ 的位置。
  - 在 Linux *MQ_INSTALLATION_PATH*/mqxr/samples 上

  MQTTV3Sample 应用程序充当发布者，将单一消息发送到服务器上的主题。 它还充当订户，侦听来自服务器的消息。
- CleanupMQM 样本脚本结束并删除由 SampleMQM 脚本创建的 QM1 。 如果要重新运行 SampleMQM 脚本或除去 QM1，请使用 CleanupMQM 样本脚本。

过程

1. 在命令行上输入以下命令以运行 SampleMQM 脚本
   - 在 Windows 上，用于运行 SampleMQM 脚本的命令如下所示:

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\SampleMQM.bat
     ```
   - 在 AIX 和 Linux 上，用于运行 SampleMQM 脚本的命令如下所示:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

   其中 *MQ_INSTALLATION_PATH* 是安装 IBM WebSphere MQ 的位置。

   将创建名为 MQXR_SAMPLE_QM 的队列管理器。
2. 输入以下命令以运行 MQTTV3Sample 脚本的第一部分;
   - 在 Windows 上的一个命令行上，输入以下命令;

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -a subscribe
     ```
   - 在 AIX 和 Linux 上的一个 shell 窗口中，输入以下命令;

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscribe
     ```
3. 输入以下命令以运行 MQTTV3Sample 脚本的第二部分;
   - 在 Windows 上的另一个命令行上，输入以下命令;

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -m "Hello from an MQTT v3
     application"
     ```
   - 在 AIX 和 Linux 上的另一个 shell 窗口中，输入以下命令;

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -m "Hello from an MQTT v3 application"
     ```
4. 要除去由 SampleMQM 脚本创建的队列管理器，可以使用以下命令运行 CleanupMQM 脚本;
   - 在 Windows 上，输入以下命令;

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat
     ```
   - 在另一个 shell 窗口中的 AIX 和 Linux 上，输入以下命令;

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh
     ```

结果

您在第二个窗口中输入的 `Hello from an MQTT v3 application` 消息将由该应用程序发布，并由该应用程序在第一个窗口中接收。 第一个窗口中的应用程序将显示在屏幕上。

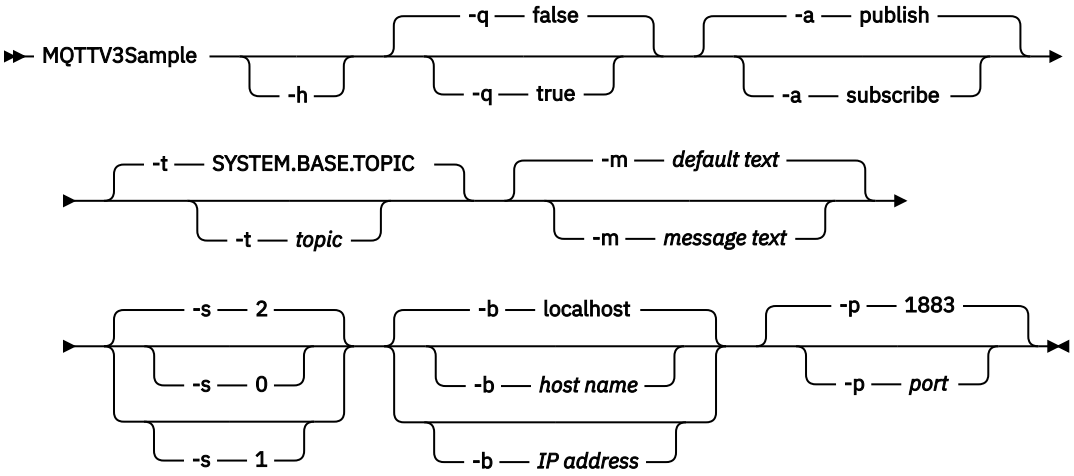
MQTTV3Sample 程序

有关 MQTTV3Sample 程序的样本语法和参数的参考信息。

用途

MQTTV3Sample 程序可用于发布消息和预订主题。

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:

```
runMQTTV3Sample -a subscribe
```

要在 Windows 上发布消息，请使用以下命令:

```
runMQTTV3Sample
```

有关运行所提供的样本脚本的更多信息，请参阅 第 397 页的『IBM WebSphere MQ Telemetry 样本程序』。

使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序

以教程方式描述了创建 MQTT 客户机应用程序的步骤。 对各代码行进行了解释。 在任务结束时，您将创建一个 MQTT 发布程序。 您可以使用 WebSphere MQ Explorer 来浏览出版物。

开始之前

在安装了 IBM WebSphere MQ Version 7.1 或更高版本的服务器上安装 WebSphere MQ Telemetry 功能部件。

客户机应用程序使用 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) 中的 `com.ibm.mq.micro.client.mqttv3` 软件包。 SDK 是 IBM WebSphere MQ Telemetry 安装的一部分。 客户机连接到 IBM WebSphere MQ Telemetry 功能部件以与 IBM WebSphere MQ 交换消息。

您还必须安装 IBM WebSphere MQ Explorer Version 7.1 的遥测更新以管理 IBM WebSphere MQ Telemetry。 更新是 IBM WebSphere MQ Telemetry 安装的一部分。

在 Java SE 上运行的 MQTT 客户机需要 Java SE V 6.0 或更高版本。 IBM Java SE v6.0 是 IBM WebSphere MQ Version 7.1 安装的一部分。 它位于 *WebSphere MQ installation directory*`\java\jre`

关于此任务

示例是发布应用程序 `PubSync`。 `PubSync` 在主题 `MQTT Examples` 上发布 `Hello World`，并等待确认发布已传递到队列管理器。

通过设置 `MQTT Examples` 的持久预订，可以检查应用程序是否正常工作。

此过程使用 Eclipse 来开发、构建和运行客户机。 您可以从位于以下网址的 Eclipse 项目 Web 站点下载 Eclipse：www.eclipse.org。

要创建应用程序，您可以创建 Java 文件，并使用命令行来编译和运行这些文件。

在新目录中，创建目录路径 `.\com\ibm\mq\id`。 创建两个 Java 文件 `Example.java` 和 `PubSync.java`。 将代码从 第 403 页的『示例代码』 复制到 Java 文件中。

使用以下命令编译 Java 代码:

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubSync.java com.ibm.mq.id.Example.java
```

使用以下命令运行 `PubSync`:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubSync
```

过程

1. 在 Eclipse 中创建 Java 项目。
   a) **文件> 新建> Java 项目** 并输入项目名称。 单击**下一步**。

      检查 JRE 是正确版本还是更高版本。 Java SE 必须为 6.0 或更高版本。

   b) 在 "Java 设置" 页面上，单击 **库> 添加外部 JAR ...**
   c) 浏览至 WebSphere MQ Telemetry SDK 文件夹的安装目录。 找到 `SDK\clients\java` 文件夹并选择所有 `.jar` 文件 > **> 打开 > 完成**。
2. 安装 MQTT 客户机 Javadoc。

安装了 MQTT 客户机 Javadoc 后， Java 编辑器将提供有关 MQTT v3 类的帮助。

a) 在 Java 项目中，打开 **包资源管理器> 引用的库**。 右键单击 `com.ibm.micro.client.mqttv3.jar` **> 属性**。

b) 在“属性导航器”中，单击 **Javadoc 位置**。

c) 在 "Javadoc 位置" 页面中，单击 **Javadoc URL> 浏览 ...** 并找到 *WMQ Installation directory*`\mqxr\SDK\clients\java\doc\javadoc` folder **> OK**。

d) 单击**验证... > 确定**

将提示您打开浏览器以查看文档。

3. 使用 Java 类向导创建类 PubSync。

a) 右键单击您创建的 Java 项目> **新建** > **类**。

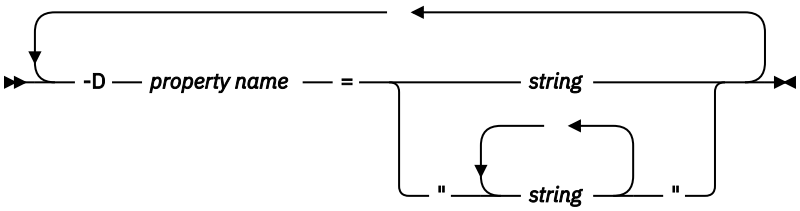
b) 输入包名 `com.ibm.mq.id`

c) 输入类名 PubSync

d) 选中方法存根框 **public static void main(String [] args)**

4. 在包 `com.ibm.mq.id` 中创建文件 `Example.java`。 将第 405 页的图 89 中的代码复制到该文件中。

示例中使用的所有参数都设置为属性。 您可以通过更改 `Example.java` 中的缺省值或通过使用 `-D` 参数在 Java 命令行上提供属性作为选项来覆盖值:



此示例以及第 405 页的『使用 Java 为 MQ Telemetry Transport 创建异步发布程序』示例中使用的客户机标识是以随机字符串为后缀的用户名。

5. 遵循步骤创建代码，或者从第 404 页的图 88 复制代码。

后续步骤解释 `Pubsync.java` 中的代码。

6. 创建 `try-catch` 块。

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

MQTT 客户机抛出 `MqttException`， `MqttPersistenceException` 或 `MqttSecurityException`。 `MqttPersistenceException` 和 `MqttSecurityException` 是 `MqttException` 的子类。

使用 `MqttException.getReasonCode` 方法来找出发生异常的原因。 如果抛出了 `MqttPersistenceException` 或 `MqttSecurityException`，请使用 `getCause` 方法返回底层可抛出异常。

7. 创建新的 `MqttClient` 实例。

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

为客户机提供服务器地址，稍后用于连接到 WebSphere MQ。 设置客户机标识以对客户机命名。

- （可选）可以提供 `MqttClientPersistence` 接口的实施以替换缺省实施。 缺省 `MqttPersistence` 实现将 QoS 1 和等待传递的 2 消息存储为文件; 请参阅第 451 页的『MQTT 客户机中的消息持久性』。
- MQTT 的缺省 IBM WebSphere MQ TCP/IP 端口为 1883。 对于 SSL，缺省端口为 8883。 在此示例中，缺省地址设置为 `tcp://localhost:1883`。

- 通常，能够使用客户机标识来识别特定物理客户机很重要。 在连接到服务器的所有客户机中，客户机标识必须唯一；请参阅第 448 页的『客户机标识』。 如果与先前实例使用同一客户机标识，那么指示目前实例是同一客户机的实例。 如果您在两个正在运行的客户机中重复使用同一个客户机标识，那么这两个客户机中都会抛出异常，并且一个客户机会终止。
- 客户机标识的长度限制为 23 字节。 如果超过此长度，那么将抛出异常。 客户机标识必须仅包含队列管理器名称中允许的字符；例如，无连字符或空格。
- 在调用 `MqttClient.connect` 方法之前，不进行任何消息处理。

使用客户机对象来发布和预订主题以及恢复有关尚未传送的发布的信息。

8. 创建要发布的主题。

```
MqttTopic topic = client.getTopic(Example.topicString);
```

主题字符串限制为 64 K 字节，该大小超过 IBM WebSphere MQ 主题字符串的最大长度。 否则，主题字符串遵循与 WebSphere MQ 主题字符串相同的规则; 请参阅 主题字符串。 此示例将设置主题字符串 `MQTT Examples`。

9. 创建发布消息。

```
MqttMessage message = new MqttMessage(Example.publication.getBytes());
```

字符串 `"Hello World"` 将转换为字节数组并用于创建 `MqttMessage`。

- MQTT 消息有效内容始终是字节数组。 `getBytes` 方法将字符串对象转换为 UTF-8。 `MqttMessage` 具有一个便利的 `toString` 方法，用于将消息有效内容作为字符串返回。 此方法等价于 `new string(message.getPayload)`
- 发布消息与 RFH2 头一起发送到队列管理器，并且消息数据作为 `jms-bytes` 消息进行发送。
- 消息对象具有服务质量和保留属性。 服务质量 (QoS) 确定在 MQTT 客户机与队列管理器之间传输消息的可靠性; 请参阅 第 454 页的『MQTT 客户机提供的服务质量』。 保留属性控制发布是否由队列管理器存储以供未来订户使用。 如果未保留发布，那么它仅发送到当前订户；请参阅第 456 页的『保留的发布和 MQTT 客户机』。 缺省 `MqttMessage` 设置为 “消息至少传递一次，并且不会保留。”

10. 连接到服务器。

```
client.connect();
```

此示例使用缺省连接选项连接到服务器。 一旦连接，即可开始发布。 缺省连接选项为:

- 每 15 秒发送一次简短的“保持活动”消息，以防止 TCP/IP 连接关闭。
- 启动会话，而不检查是否已完成先前的发布。
- 重试发送消息的时间间隔为 15 秒。
- 不为连接创建最终消息。
- 使用标准 SocketFactory 来创建连接。

通过创建 `ConnectionOptions` 对象并将其作为附加参数传递到 `client.connect` 来更改连接选项。

11. 发布。

```
MqttDeliveryToken token = topic.publish(message);
```

此示例将主题为 "MQTT 示例" 的 "Hello World" 出版物发送到队列管理器。

- 当 `publish` 方法返回时，消息将安全地传输到 MQTT 客户机，但尚未传输到服务器。 如果消息具有 QoS 1 或 2，那么在传送完成之前客户机发生故障的情况下，消息以本地方式存储。
- `publish` 返回传送标记，用于检查是否已从服务器接收到确认。

12. 等待来自服务器的确认。

```
token.waitForCompletion(Example.timeout);
```

PubSync 示例等待来自服务器的确认，它将确认是否已传送消息。

- 如果没有超时，那么客户机将无限等待。 任务第 405 页的『使用 Java 为 MQ Telemetry Transport 创建异步发布程序』说明如何通过使用回调对象来接收确认而不等待。

13. 将客户机与服务器断开连接。

```
client.disconnect();
```

客户机与服务器断开连接，并等待完成任何正在运行的 MqttCallback 方法。 然后，它将等待长达 30 秒以完成任何剩余工作。 可以将停顿超时指定为附加参数。

14. 保存对 PubSync.java 和 Example.Java 的更改

Eclipse 会自动编译 Java。 现在，即可通过运行程序来查看结果。

结果

要使用 WebSphere MQ 来查看出版物，请使用 第 403 页的图 87 中的脚本创建主题，队列和持久预订 (全部称为 "MQTTExampleTopic")。 运行客户机以发布 MQTT Examples 主题，然后运行样本程序 **amqsbcg** 以浏览 MQTTExamples 队列上的发布。

1. 启动队列管理器，并且启动其正在运行的遥测 (MQXR) 服务。 确保为遥测通道配置的 TCP/IP 地址和端口与您在 MQTT 应用程序中使用的值相匹配。
2. 通过创建 mqttexamples.txt 命令脚本并使用 **runmqsc:** 运行此脚本来配置持久预订

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

图 *87: mqttExampleTopic.txt*

要在 Windows 上运行脚本，请输入以下命令:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. 从 Eclipse 中作为 Java 应用程序运行客户机，或者通过在命令窗口中运行 Java 来运行客户机:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

注: 必须在包含 com\ibm\mq\id 路径的目录中打开命令窗口。

4. 使用 WebSphere MQ Explorer 浏览结果，或者运行以下命令:

```
amqsbcg MQTTExampleQueue queue manager name
```

示例代码

PubSync.java 是过程中描述的代码的完整列表。 修改 第 405 页的图 89 中的 Example 类以覆盖 PubSync.java 中使用的缺省参数。

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      client.connect();
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
          + " seconds for publication of \"" + message.toString()
          + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName()
          + "\" for client instance: \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *88: PubSync.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean          retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

图 *89: Example.java*

相关概念

MQTT 发布/预订应用程序

使用 Java 为 MQ Telemetry Transport 创建异步发布程序

在此任务中，您将遵循教程来修改第一个发布者应用程序。 通过修改，应用程序可以发送发布内容而不等待传送确认。 传送确认由您创建的回调类来接收。

开始之前

在安装了 IBM WebSphere MQ Version 7.1 或更高版本的服务器上安装 WebSphere MQ Telemetry 功能部件。

客户机应用程序使用 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) 中的 `com.ibm.mq.micro.client.mqttv3` 软件包。 SDK 是 IBM WebSphere MQ Telemetry 安装的一部分。 客户机连接到 IBM WebSphere MQ Telemetry 功能部件以与 IBM WebSphere MQ 交换消息。

您还必须安装 IBM WebSphere MQ Explorer Version 7.1 的遥测更新以管理 IBM WebSphere MQ Telemetry。 更新是 IBM WebSphere MQ Telemetry 安装的一部分。

在 Java SE 上运行的 MQTT 客户机需要 Java SE V 6.0 或更高版本。 IBM Java SE v6.0 是 IBM WebSphere MQ Version 7.1 安装的一部分。 它位于 *WebSphere MQ installation directory*\java\jre

关于此任务

示例是发布应用程序 PubAsync。 PubAsync 在主题 MQTT Examples 上发布 Hello World，而不等待确认发布已传递到队列管理器。 将在回调类 CallBack 中接收传送确认。

通过设置 MQTT Examples 的持久预订，可以检查应用程序是否正常工作。

此过程使用 Eclipse 来开发、构建和运行客户机。 您可以从位于以下网址的 Eclipse 项目 Web 站点下载 Eclipse：www.eclipse.org。

过程中的步骤修改第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』中的 PubSync.java 应用程序。

或者，可以将第 408 页的『示例代码』中的代码复制到新目录 .\com\ibm\mq\id 中。 创建三个 Java 文件 Example.java， CallBack.java 和 PubAsync.java。 使用以下命令来编译示例：

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsync.java com.ibm.mq.id.CallBack.java com.ibm.mq.id.Example.java
```

使用以下命令运行 PubAsync：

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsync.class
```

过程

1. 在 com.ibm.mq.id 包中，创建文件 CallBack.java。 将第 408 页的图 92 中的代码复制到该文件中。

   Callback.java 实施 MqttCallBack 接口。 在此示例中，附加构造函数使用某些实例数据来初始化回调。
2. 在 com.ibm.mq.id 包中，右键单击 PubSync.java 并复制此文件。 将其粘贴在同一个包中，但是将其重命名为 PubAsync。
3. 就在代码行 client.connect(); 的前面，将 CallBack 类实例化，从而传递客户机标识。

   ```
   CallBack callback = new CallBack(Example.clientId);
   client.setCallback(callback);
   ```

   - CallBack 类实施 MqttCallBack。 每个客户机标识需要一个回调实例。 在此示例中，构造函数传递客户机标识以另存为实例数据。 它在回调中用于识别已启动哪个回调实例。
   - 必须在回调类中实施三个方法：

     public void messageArrived(MqttTopic topic, MqttMessage message)
     接收已预订的发布。

     public void connectionLost(Throwable cause)
     在连接丢失时调用。

     public void deliveryComplete(MqttDeliveryToken token))
     在接收到已发布的 QoS 1 或 2 消息的传送标记时调用。
   - 回调由 MqttClient.connect 激活。
4. 使客户机断开连接

   a) 除去其中包含 token.waitForCompletion 表达式的语句。

   主线程将继续执行，而不等待传送发布。

   b) 测试客户机是否已断开连接。

   MQTT 客户机在返回到 MqttCallback 中的 lostConnection 方法的错误后断开连接，或者客户机应用程序可能断开连接。 测试以查看是否有打开的连接。

   c) 使用常量 Example.quiesceTimeout 设置客户机停顿的最长时间。

```
if (client.isConnected())
    client.disconnect(Example.quiesceTimeout);
```

当以下三个条件的组合成立时，客户机即完成：

a. 对于在此会话中（如果此会话已重新启动，则在先前会话中）已发布的所有消息，已调用回调。

b. 消息未完成，而停顿时间间隔已到期。 缺省情况下，停顿时间间隔为 30 秒。 可以通过将要等待的毫秒数作为 `client.disconnect` 的参数进行传递来更改停顿超时。

c. 在某些消息已发布并由客户机进行排队之后，但在发送消息之前，调用了 `client.disconnect`。 已排队的消息尚未处于未完成状态。 如果会话可重新启动，那么在会话启动重新启动时将重新发送消息。

结果

要使用 WebSphere MQ 来查看出版物，请使用 第 407 页的图 90 中的脚本创建主题，队列和持久预订 (全部称为 "MQTTExampleTopic")。 运行客户机以发布 `MQTT Examples` 主题，然后运行样本程序 **amqsbcg** 以浏览 `MQTTExamples` 队列上的发布。

1. 启动队列管理器，并且启动其正在运行的遥测 (MQXR) 服务。 确保为遥测通道配置的 TCP/IP 地址和端口与您在 MQTT 应用程序中使用的值相匹配。
2. 通过创建 `mqttexamples.txt` 命令脚本并使用 **runmqsc:** 运行此脚本来配置持久预订

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

图 *90: mqttExampleTopic.txt*

要在 Windows 上运行脚本，请输入以下命令：

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. 从 Eclipse 中作为 Java 应用程序运行客户机，或者通过在命令窗口中运行 Java 来运行客户机：

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

注: 必须在包含 `com\ibm\mq\id` 路径的目录中打开命令窗口。

4. 使用 WebSphere MQ Explorer 浏览结果，或者运行以下命令：

```
amqsbcg MQTTExampleQueue queue manager name
```

示例代码

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubAsync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      client.connect();
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *91: PubAsync.java*

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *92: CallBack.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean          retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

图 *93: Example.java*

使用 Java 为 MQ Telemetry Transport 创建可恢复异步发布程序

在此任务中，您将遵循教程来修改异步发布者应用程序。 通过修改，应用程序可以完成上次客户机运行时未确认的发布的传送。

开始之前

在安装了 IBM WebSphere MQ Version 7.1 或更高版本的服务器上安装 WebSphere MQ Telemetry 功能部件。

客户机应用程序使用 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) 中的 `com.ibm.mq.micro.client.mqttv3` 软件包。 SDK 是 IBM WebSphere MQ Telemetry 安装的一部分。 客户机连接到 IBM WebSphere MQ Telemetry 功能部件以与 IBM WebSphere MQ 交换消息。

您还必须安装 IBM WebSphere MQ Explorer Version 7.1 的遥测更新以管理 IBM WebSphere MQ Telemetry。 更新是 IBM WebSphere MQ Telemetry 安装的一部分。

在 Java SE 上运行的 MQTT 客户机需要 Java SE V 6.0 或更高版本。 IBM Java SE v6.0 是 IBM WebSphere MQ Version 7.1 安装的一部分。 它位于 *WebSphere MQ installation directory*\java\jre

关于此任务

示例是发布应用程序 PubAsyncRestartable。 PubAsyncRestartable 在主题 MQTT Examples 上发布 Hello World，而不等待确认是否已将发布传送到队列管理器。 将在回调类 CallBack 中接收传送确认。可以检查先前实例中未完成的发布的任何传送标记。 它们也由回调类进行处理。

通过设置 MQTT Examples 的持久预订，可以检查应用程序是否正常工作。

此过程使用 Eclipse 来开发、构建和运行客户机。 您可以从位于以下网址的 Eclipse 项目 Web 站点下载 Eclipse：www.eclipse.org。

过程 中的步骤将修改 第 405 页的『使用 Java 为 MQ Telemetry Transport 创建异步发布程序』中的 PubAsync.java 应用程序。

或者，可以将第 412 页的『示例代码』中的代码复制到新目录 .\com\ibm\mq\id 中。 创建三个 Java 文件 Example.java， CallBack.java 和 PubAsyncRestartable.java。 使用以下命令来编译示例：

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsyncRestartable.java com.ibm.mq.id.CallBack.java
com.ibm.mq.id.Example.java
```

使用以下命令运行 PubAsyncRestartable：

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsyncRestartable.class
```

过程

1. 在 com.ibm.mq.id 包中，右键单击 PubAsync.java 并复制此文件。 将其粘贴在同一个包中，但是将其重命名为 PubAsyncRestartable。
2. 创建可复用客户机标识。

   ```
   Example.clientId = String.format(
       "%-23.23s",
       (System.getProperty("user.name") + "_" + (System.getProperty(
           "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
   ```

   图 *94:* 可复用客户机标识

   第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』和第 405 页的『使用 Java 为 MQ Telemetry Transport 创建异步发布程序』中的应用程序对每个客户机连接使用新的客户机标识。 对于可重新启动的发布者或订户，每次连接客户机时，必须使用同一客户机标识，但是不同客户及必须使用不同标识符；请参阅第 448 页的『客户机标识』。 可复用客户机标识根据用户名和类名进行构建。 其长度限制为 23 字节。 它必须仅具有在队列管理器对象名称中有效的字符。 代码将除去可能已经插入的任何连字符。
3. 消息的 QoS 设置为 2，而不是缺省值 1，以避免消息重复。

   ```
   message.setQos(Example.QoS);
   ```

   您需要将 Example.QoS 的值更改为 2，或者使用 Java 命令行上的 -DQoS=2 选项将 QoS 属性作为自变量传递。
4. 创建 MqttConnectOptions 对象，并且将其 cleanSession 属性设置为 false。

   a) 创建 MqttConnectOptions 对象。

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   ```

   conOptions 是 MqttClient 构造函数上的选项参数。

   b) 设置 clearSession 属性。

   ```
   conOptions.setCleanSession(Example.cleanSession);
   ```

缺省情况下，参数 Example.cleanSession 设置为 true，从而与 MqttConnectionOptions.cleanSession 的缺省设置相匹配。

当 PubAsyncRestartable 重新启动时，重新启动时，它可以从“清除会话”开始，并清除 QoS 为 1 或 2 的消息的任何暂挂传送标记。

将 Example.cleanSession 设置为 false，以保留所有暂挂传送标记。 再次连接客户机时，标记由 MqttCallBack 类处理。

5. 如果正在重新启动会话，那么检索任何暂挂传送标记并打印其内容。

```
if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance "
            + client.getClientId());
```

6. 将 conOptions 参数传递到 MqttClient 构造函数。

```
client.connect(conOptions);
```

7. 在断开连接时，设置最大断开连接时间间隔。

```
client.disconnect(Example.timeout);
```

为能够显示正在处理的暂挂传送标记，先前实例必须在没有完成传送的情况下完成。 要在存在以下可能性的情况下运行此示例，请将 Example.timeout 设置为 0：在完成 PubAsyncRestartable 之前未确认发布。

结果

要使用 WebSphere MQ 来查看出版物，请使用 第 411 页的图 95 中的脚本创建主题，队列和持久预订 (全部称为 "MQTTExampleTopic")。 运行客户机以发布 MQTT Examples 主题，然后运行样本程序 **amqsbcg** 以浏览 MQTTExamples 队列上的发布。

1. 启动队列管理器，并且启动其正在运行的遥测 (MQXR) 服务。 确保为遥测通道配置的 TCP/IP 地址和端口与您在 MQTT 应用程序中使用的值相匹配。
2. 通过创建 mqttexamples.txt 命令脚本并使用 **runmqsc:** 运行此脚本来配置持久预订

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

图 *95: mqttExampleTopic.txt*

要在 Windows 上运行脚本，请输入以下命令:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. 从 Eclipse 中作为 Java 应用程序运行客户机，或者通过在命令窗口中运行 Java 来运行客户机:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

注: 必须在包含 com\ibm\mq\id 路径的目录中打开命令窗口。

4. 使用 WebSphere MQ Explorer 浏览结果，或者运行以下命令:

```
amqsbcg MQTTExampleQueue queue manager name
```

示例代码

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
import com.ibm.micro.client.mqttv3.MqttDeliveryToken;
import com.ibm.micro.client.mqttv3.MqttMessage;
import com.ibm.micro.client.mqttv3.MqttTopic;
public class PubAsyncRestartable {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + (System.getProperty(
            "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance \""
            + client.getClientId() + "\"");
      client.connect(conOptions);
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *96: PubAsyncRestartable.java*

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *97: CallBack.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

图 *98: Example.java*

使用 Java 为 MQ Telemetry Transport 创建订户

在此任务中，您将遵循教程来创建订户应用程序。 订户创建对主题的预订并接收该预订的发布。

开始之前

在安装了 IBM WebSphere MQ Version 7.1 或更高版本的服务器上安装 WebSphere MQ Telemetry 功能部件。

客户机应用程序使用 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) 中的 `com.ibm.mq.micro.client.mqttv3` 软件包。 SDK 是 IBM WebSphere MQ Telemetry 安装的一部分。 客户机连接到 IBM WebSphere MQ Telemetry 功能部件以与 IBM WebSphere MQ 交换消息。

您还必须安装 IBM WebSphere MQ Explorer Version 7.1 的遥测更新以管理 IBM WebSphere MQ Telemetry。 更新是 IBM WebSphere MQ Telemetry 安装的一部分。

在 Java SE 上运行的 MQTT 客户机需要 Java SE V 6.0 或更高版本。 IBM Java SE v6.0 是 IBM WebSphere MQ Version 7.1 安装的一部分。 它位于 *WebSphere MQ installation directory*`\java\jre`

关于此任务

示例是订户应用程序 Subscribe。 Subscribe 创建预订主题 MQTT Examples，并等待获得该预订的发布，等待时间为 30 秒。

订户可以创建预订并等待获得发布。 它还可以接收发送到先前为同一客户机标识创建的预订的发布。 MqttConnectionOptions.cleanSession 布尔属性控制是否接收到先前发送的发布；请参阅第 457 页的『预订』。

您可以使用发布示例程序来创建发布，或者使用 WebSphere MQ 资源管理器在 MQTT Examples 主题上创建测试发布。

此过程使用 Eclipse 来开发、构建和运行客户机。 您可以从位于以下网址的 Eclipse 项目 Web 站点下载 Eclipse：www.eclipse.org。

过程中的指示信息假设您已在先前其中一项任务中创建 com.ibm.mq.id 包，并在 Example.java 和 Callback.java 类中进行复制。

过程

1. 在包 com.ibm.mq.id 中创建类 Subscribe。
2. 创建可复用客户机标识。

```
Example.clientId = String.format(
    "%-23.23s",
    (System.getProperty("user.name") + "_" + (System.getProperty(
        "clientId", "Subscribe."))).trim()).replace('-', '_');
```

图 *99:* 可复用客户机标识

第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』和第 405 页的『使用 Java 为 MQ Telemetry Transport 创建异步发布程序』中的应用程序对每个客户机连接使用新的客户机标识。 对于可重新启动的发布者或订户，每次连接客户机时，必须使用同一客户机标识，但是不同客户及必须使用不同标识符；请参阅第 448 页的『客户机标识』。 可复用客户机标识根据用户名和类名进行构建。 其长度限制为 23 字节。 它必须仅具有在队列管理器对象名称中有效的字符。 代码将除去可能已经插入的任何连字符。

3. 创建 try-catch 块。

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

MQTT 客户机抛出 MqttException， MqttPersistenceException 或 MqttSecurityException。 MqttPersistenceException 和 MqttSecurityException 是 MqttException 的子类。

使用 MqttException.getReasonCode 方法来找出发生异常的原因。 如果抛出了 MqttPersistenceException 或 MqttSecurityException，请使用 getCause 方法返回底层可抛出异常。

4. 创建新的 MqttClient 实例。

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

为客户机提供服务器地址，稍后用于连接到 WebSphere MQ。 设置客户机标识以对客户机命名。

- （可选）可以提供 MqttClientPersistence 接口的实施以替换缺省实施。 缺省 MqttPersistence 实现将 QoS 1 和等待传递的 2 消息存储为文件; 请参阅 第 451 页的『MQTT 客户机中的消息持久性』。
- MQTT 的缺省 IBM WebSphere MQ TCP/IP 端口为 1883。 对于 SSL，缺省端口为 8883。 在此示例中，缺省地址设置为 tcp://localhost:1883。
- 通常，能够使用客户机标识来识别特定物理客户机很重要。 在连接到服务器的所有客户机中，客户机标识必须唯一；请参阅第 448 页的『客户机标识』。 如果与先前实例使用同一客户机标识，那么指示

目前实例是同一客户机的实例。 如果您在两个正在运行的客户机中重复使用同一个客户机标识，那么这两个客户机中都会抛出异常，并且一个客户机会终止。

- 客户机标识的长度限制为 23 字节。 如果超过此长度，那么将抛出异常。 客户机标识必须仅包含队列管理器名称中允许的字符；例如，无连字符或空格。
- 在调用 MqttClient.connect 方法之前，不进行任何消息处理。

使用客户机对象来发布和预订主题以及恢复有关尚未传送的发布的信息。

5. 就在代码行 client.connect(); 的前面，将 CallBack 类实例化，从而传递客户机标识。

```
CallBack callback = new CallBack(Example.clientId);
client.setCallback(callback);
```

- CallBack 类实施 MqttCallBack。 每个客户机标识需要一个回调实例。 在此示例中，构造函数传递客户机标识以另存为实例数据。 它在回调中用于识别已启动哪个回调实例。
- 必须在回调类中实施三个方法：

  public void messageArrived(MqttTopic topic, MqttMessage message)
  接收已预订的发布。

  public void connectionLost(Throwable cause)
  在连接丢失时调用。

  public void deliveryComplete(MqttDeliveryToken token))
  在接收到已发布的 QoS 1 或 2 消息的传送标记时调用。

- 回调由 MqttClient.connect 激活。

6. 创建 MqttConnectOptions 对象，并且设置其 cleanSession 属性。

   a) 创建 MqttConnectOptions 对象。

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   ```

   conOptions 是 MqttClient 构造函数上的选项参数。

   b) 设置 clearSession 属性。

   ```
   conOptions.setCleanSession(Example.cleanSession);
   ```

   缺省情况下，参数 Example.cleanSession 设置为 true，从而与 MqttConnectionOptions.cleanSession 的缺省设置相匹配。

   如果使用缺省 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，将移除客户机的所有旧预订。 在会话断开连接时，将除去客户机在会话期间进行的任何新预订。

   如果您在连接之前将 MqttConnectOptions.cleanSession 设置为 false，那么客户机创建的任何预订都会被添加至客户机在连接之前就已存在的所有预订中。 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。

   在连接之前，您必须设置 cleanSession 方式；在整个会话期间都将保持此方式。 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。 如果您将方式从使用 cleanSession=false 更改为 cleanSession=true，那么此客户机先前的所有预订以及尚未收到的任何发布都将被废弃。

7. 将 conOptions 参数传递到 MqttClient 构造函数。

```
client.connect(conOptions);
```

8. 创建预订。

```
      client.subscribe(Example.topicString, Example.QoS);
```

此示例使用 MqttClient.subscribe 方法，它通过 QoS 选项传递一个主题过滤器。 MqttClient.subscribe 方法有四个签名，您可以传递预订过滤器的数组，也可以传递单个过滤器。

该示例将发布示例所使用的主题字符串用作主题过滤器，因此它将接收这些发布示例创建的任何发布。

每次运行示例 subscribe.java 时，它将创建预订。 如果未更改 Example.topicString，那么它将再次重新创建同一预订。 如果重新创建了预订，那么不会产生两个完全相同的预订。 客户机不会接收与完全相同的预订相匹配的发布的重复副本。

在第 457 页的『预订』中描述了预订，并在第 458 页的『MQTT 客户机中的主题字符串和主题过滤器』中描述了过滤器。

9. 等待某些发布到达，然后断开客户机连接。

```
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
```

通过实施 MqttCallback.messageArrived 方法来接收发布。

预订应用程序尚未发布任何消息，因此它不等待任何传送标记。 client.disconnect 立即执行，而没有任何延迟。

示例代码

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
public class Subscribe {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + System.getProperty("clientId",
            "Subscribe."))).trim();
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      client.connect(conOptions);
      System.out.println("Subscribing to topic \"" + Example.topicString
          + "\" for client instance \"" + client.getClientId()
          + "\" using QoS " + Example.QoS + ". Clean session is "
          + Example.cleanSession);
      client.subscribe(Example.topicString, Example.QoS);
      System.out.println("Going to sleep for " + Example.sleepTimeout / 1000
          + " seconds");
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
      System.out.println("Finished");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *100: Subscribe.java*

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

图 *101: CallBack.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

图 *102: Example.java*

相关概念

MQTT 发布/预订应用程序

使用 JAAS 认证 MQTT Java 客户机

了解如何使用 JAAS 来认证客户机。 修改样本程序 JAASLoginModule.java 和示例 Java 程序 PubSync.java。 配置遥测通道以要求进行 JAAS 认证，并且运行已修改的发布者，从而使用 JAAS 检查其**用户名**和**密码**。

开始之前

在执行此任务之前，假定您已安装 MQTT v3 客户机 jar 文件， Javadoc， Eclipse，已配置遥测通道并编码和运行 PubSync.java 。 您具有包含 PubSync.java 的运行版本的 Eclipse 工作空间。

此任务是针对 Windows 编写的。 请针对 Linux 更改目录路径。

关于此任务

此任务基于修改 *WMQ Installation directory*\mqxr\samples\JAASLoginModule.java 中的样本 JAASLoginModule 类以创建 MyLogin.java。 在此任务中，还将修改第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』中的示例代码 PubSync.java 以设置**用户名**和**密码**。 作为测试，MyLogin.java 随机接受或拒绝**用户名**和**密码**。

此任务中的步骤编写为编程练习。 您必须改写过程以在生产环境中执行实际认证。

在如何对 JAAS 认证进行编程的典型说明中，假设登录模块正在认证已装入 JAAS 的上下文. 当遥测 (MQXR) 服务调用 JAAS 时，已装入 JAAS 的上下文是遥测 (MQXR) 服务。 认证遥测 (MQXR) 服务上下文没有意义；它始终为 mqm。 相反，遥测 (MQXR) 服务将客户机**用户名**和**密码**设置为对于登录模块类可用。 通过使用两个回调将**用户名**和**密码**传递到登录模块。

```
javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
callbacks[0] =
      new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] =
      new javax.security.auth.callback.PasswordCallback("PasswordCallback", false);
callbackHandler.handle(callbacks);
String username =
      ((javax.security.auth.callback.NameCallback) callbacks[0]).getName();
char[] password =
      ((javax.security.auth.callback.PasswordCallback) callbacks[1]).getPassword();
```

客户机的**用户名**和**密码**是有关对于登录模块可用的客户机的唯一信息。

过程

1. 在与 PubSync.java 相同的 Java 项目中创建两个包 samples 和 security.jaas 。

   samples 包仅供参考。 在 security.jaas 包中进行代码更改。
2. 在两个包中均导入 JAASLoginModule.java 和 JAASPrincipal.java。

   如果需要，请重构 Java 源中的包语句以消除编译错误。
3. 将 security.jaas 包中的类名 JAASLoginModule 重构为 MyLogin
4. 在 MyLogin.java 中，替换 login 方法中的某些代码以显示模块工作。

   a) 将以下代码：

   ```
   // Accept everything.
   if (true)
     loggedIn = true;
   else
     throw new javax.security.auth.login.FailedLoginException("Login failed");
   ```

   b) 替换为以下代码：

   ```
   // login half the users randomly
   PrintWriter pw = new PrintWriter(new FileWriter(System.getProperty("user.dir")
           + "\\MyLogin.log", true));
   pw.println("Called JAASLogin.login at "
           + System.getProperty("publication", "Hello World "
           + String.format("%tc", System.currentTimeMillis())));
   if (Math.random() < 0.5)
     loggedIn = true;
   pw.println("Username: \"" + username + "\", Password: \""
       + String.valueOf(password) + "\" loggedIn: " + loggedIn);
   pw.close();
   if (!loggedIn)
     throw new javax.security.auth.login.FailedLoginException("Login failed");
   principal= new JAASPrincipal(username);
   ```

   第 423 页的图 105 中提供了 MyLogin.java 的完整源代码。 第 424 页的图 106 中提供了 JAASPrincipal.java 的源代码，其中包名重构为 security.jaas。
5. 将 service.env 中的类路径设置为指向包含 security/jaas/MyLogin.class 和 security/jaas/JAASPrincipal.class 的路径的目录。

```
CLASSPATH=C:\WMQTelemetryApps\MQTTSecureExamples\bin
```

请参阅 遥测通道 JAAS 配置 ， 以获取有关使用 `service.env` 将类路径传递到 WebSphere MQ 服务的信息。

6. 向 `jaas.config` 添加登录模块节。

```
MyLoginExample {
  security.jaas.MyLogin required debug=true;
  };
```

请参阅遥测通道 JAAS 配置以获取有关使用 `jaas.config` 定义 JAAS 登录模块的信息。

7. 使用 WebSphere MQ Explorer 中的 " **新建遥测通道** " 向导添加遥测通道，将该通道配置为需要 JAAS 认证。 请将其送交到 `MyLoginExample` 节。

例如，从 `mqxr_win.properties` 文件中的此节来改写输入到向导中的信息。 如果您是在 Linux 中工作，那么文件称为 `mqxr_unix.properties`。 不要直接编辑遥测属性；请使用向导。

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=MyLoginExample;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

注: 如果修改任何遥测通道参数，或者修改 `security.jaas.Mylogin` 类，那么必须停止并重新启动遥测 (MQXR) 服务。 仅当重新启动该服务时，更改才会生效。

8. 在 `com.ibm.mq.id` 包中生成 `PubSync.java` 的副本，并将该副本命名为 `PubSyncJAAS.java`。

请参阅第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』以了解在 `com.ibm.mq.id` 包中创建 PubSync.java 的步骤。

9. 在 `PubSyncJAAS.java` 程序中设置 `MqttConnectOptions.username` 和 `MqttConnectOptions.password`，并将 `MqttConnectOptions` 作为 `MqttClient.connect` 的参数进行传递。

```
MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setUserName(Example.username);
conOptions.setPassword(Example.password);
client.connect(conOptions);
```

请参阅 PubSyncJAAS.Java 中使用 Example.java 中设置的常量的斜体代码。

10. 将 `Example.TCPAddress` 设置为已配置为使用 JAAS 配置 `MyLoginExample` 的遥测通道的套接字地址。 例如，使用 1884 作为端口号。
11. 多次运行 `PubSyncJAAS` 以了解客户机的登录是被接受还是拒绝。

每次拒绝登录尝试时，就会抛出异常。

结果

第 422 页的图 103 显示了运行 PubSyncJAAS.Java 两次。 第 422 页的图 104 中显示了日志记录。

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:05 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_61c57a18_4bf7_40d" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Client exception caught
Client is not connected (32104)
    at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:88)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:105)
    at com.ibm.micro.client.mqttv3.MqttTopic.publish(MqttTopic.java:68)
    at com.ibm.mq.id.PubSync.main(PubSync.java:24)
```

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:40 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_1d1599a0_50f5_4ea" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Delivery token "1731749688" has been received: true
```

图 *103:* 来自 *PubSyncJAAS.java* 的控制台输出

日志文件 MyLogin.log 存储在 *WMQ Data directory* 中; 例如， C:\IBM\MQ\Data\MyLogin.log:

```
Called JAASLogin.login at Hello World Fri Jun 04 08:31:05 BST 2010
Username: "Admin", Password: "Password" loggedIn: false
Called JAASLogin.login at Hello World Fri Jun 04 08:31:40 BST 2010
Username: "Admin", Password: "Password" loggedIn: true
```

图 *104: MyLogin.log*

示例

第 423 页的图 105 中的斜体代码是对样本 JAASLoginModule.java 的修改。

```
package security.jaas;
import java.io.FileWriter;
import java.io.PrintWriter;

public class JAASLogin implements javax.security.auth.spi.LoginModule {
  private javax.security.auth.Subject                    subject;
  private javax.security.auth.callback.CallbackHandler callbackHandler;
  JAASPrincipal                                          principal;
  boolean                                                loggedIn = false;
  public void initialize(javax.security.auth.Subject subject,
      javax.security.auth.callback.CallbackHandler callbackHandler,
      java.util.Map<String, ?> sharedState, java.util.Map<String, ?> options) {
    this.subject = subject;
    this.callbackHandler = callbackHandler;
  }
  public boolean login() throws javax.security.auth.login.LoginException {
    try {
      javax.security.auth.callback.Callback[] callbacks = new
javax.security.auth.callback.Callback[2];
      callbacks[0] = new javax.security.auth.callback.NameCallback(
          "NameCallback");
      callbacks[1] = new javax.security.auth.callback.PasswordCallback(
          "PasswordCallback", false);

      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // login half the users randomly
      PrintWriter pw = new PrintWriter(new FileWriter(System
          .getProperty("user.dir")
          + "\\mylogin.log", true));
      pw.println("Called JAASLogin.login at "
          + System.getProperty("publication", "Hello World "
              + String.format("%tc", System.currentTimeMillis())));
      if (Math.random() < 0.5)
        loggedIn = true;
      pw.println("Username: \"" + username + "\", Password: \""
          + String.valueOf(password) + "\" loggedIn: " + loggedIn);
      pw.close();
      if (!loggedIn)
        throw new javax.security.auth.login.FailedLoginException("Login failed");
      principal = new JAASPrincipal(username);
    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }
    return loggedIn;
  }
  public boolean abort() throws javax.security.auth.login.LoginException {
    logout();
    return true;
  }
  public boolean commit() throws javax.security.auth.login.LoginException {
    if (loggedIn) {
      if (!subject.getPrincipals().contains(principal))
        subject.getPrincipals().add(principal);
    }
    return true;
  }
  public boolean logout() throws javax.security.auth.login.LoginException {
    subject.getPrincipals().remove(principal);
    principal = null;
    loggedIn = false;
    return true;
  }
}
```

图 *105: MyLogin.java*

第 424 页的图 106 是复制到包 security.jaas 中的样本代码 JAASLoginPrincipal.java。JAASLoginPrincipal 的用途是实施 java.security.Principal 接口来记录已通过 MyLogin 成功登录的用户。

```
package security.jaas;
public class JAASPrincipal implements java.security.Principal,
    java.io.Serializable {
  private static final long serialVersionUID = 1L;
  String                    name;
  public JAASPrincipal(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public String toString() {
    return (name);
  }
  public boolean equals(Object object) {
    if (object != null && object instanceof JAASPrincipal
        && name.equals(((JAASPrincipal) object).getName()))
      return true;
    else
      return false;
  }
  public int hashCode() {
    return name.hashCode();
  }
}
```

图 *106: JAASLoginPrincipal.java*

PubSync.java 中修改为添加**用户名**和**密码**的代码在第 424 页的图 107 中以斜体显示。

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setUserName(Example.username);
      conOptions.setPassword(Example.password);
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("With username \"" + conOptions.getUserName()
              + "\" and password \"" + String.valueOf(conOptions.getPassword()) + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

图 *107: PubSyncJAAS.Java*

将 Example.java 中的常量修改为与配置相匹配。 请忽略此示例的 SSL 设置。

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean          retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

图 *108: Example.java*

使用自签名证书来认证 SSL Telemetry 连接

使用通过使用 **Keytool** 生成的自签名证书来认证 SSL 连接。 您可以选择认证遥测通道，也可以选择认证遥测通道及其相连的客户机。 在连接上流动的消息已加密。

开始之前

在开始之前，执行任务第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』，以使 PubSync.java 适用于不安全的 TCP/IP 连接。 在此任务中，修改 PubSync.java 以使用 SSL 连接。

关于此任务

此任务中的步骤编写为编程练习。 您必须改写过程以在生产环境中执行实际认证。

此任务是针对 Windows 编写的。 请针对 Linux 更改目录路径。

过程

1. 请执行第 426 页的『将 PubSync.java 修改为使用 SSL』任务以将 PubSync.java 修改为使用 SSL。

2. 配置遥测通道，并创建密钥库以使用 SSL。

   仅认证遥测通道，或者认证遥测通道以及与其相连的客户机：

   - 执行第 427 页的『认证遥测通道』任务，以使用 SSL 进行连接，并认证遥测通道。
   - 执行第 428 页的『认证遥测通道和客户机』任务，以使用 SSL 进行连接，并认证遥测通道以及与它相连的客户机。
3. 停止并重新启动遥测 (MQXR) 服务以拾取对遥测通道配置的更改。
4. 运行客户机程序以查看配置是否工作。

将 PubSync.java 修改为使用 SSL

修改第一个发布程序示例以使用 SSL 连接至遥测通道。 设置已修改的程序所使用的 SSL 属性。

开始之前

在执行此任务之前，假定您已安装 MQTT v3 客户机 jar 文件， Javadoc， Eclipse，已配置遥测通道并编码和运行 PubSync.java 。 您具有包含 PubSync.java 的运行版本的 Eclipse 工作空间。

关于此任务

此任务使用您在第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』中创建的发布者客户机 PubSync.java 作为基础。 只需稍作修改即可使用 SSL；请参阅第 427 页的图 109 和第 427 页的图 110。

过程

1. 在 `com.ibm.mq.id` 包中生成 `PubSync.java` 的副本，并将该副本命名为 `PubSyncSSL.java`。

   请参阅第 400 页的『使用 Java 创建第一个 MQ Telemetry Transport 发布程序应用程序』以了解在 `com.ibm.mq.id` 包中创建 PubSync.java 的步骤。
2. 将 `Example.SSLAddress` 设置为您已经配置为用于 SSL 配置的遥测通道的套接字地址。
3. 更改客户机构造函数的套接字地址参数以使用 `Example.SSLAddress`。

   ```
   MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
   ```
4. 在 `PubSyncSSL.java` 中设置 `MqttConnectOptions.SSLProperties`，并将 `MqttConnectOptions` 作为 `MqttClient.connect` 的参数进行传递。

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setSSLProperties(Example.getSSLSettings());
   client.connect(conOptions);
   ```

   请参阅使用 `Example.java` 中设置的常量的斜体代码 `PubSyncSSL.Java` 。

示例

在第 427 页的图 109 中，用斜体显示了为了添加 SSL 而对 PubSync.java 所作的修改。

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setSSLProperties(Example.getSSLSettings());
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("SSL Properties" + conOptions.getSSLProperties());
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

图 *109: PubSyncSSL.Java*

对 Example.java 的修改显示在第 427 页的图 110 中。

```
 public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");

 public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore", "C:\\Certificates\\SSClientKey.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword", "password");
    properties.setProperty("com.ibm.ssl.trustStore", "C:\\Certificates\\SSClientTrust.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword", "password");
    return properties;
```

图 *110:* 对 *Example.java* 的修改

认证遥测通道

客户机验证遥测通道以加密在通道上流动的消息的内容，并且确保客户机连接到正确的遥测通道。 服务器不会对客户机进行认证。

关于此任务

可以使用许多不同的密钥库编辑器来创建和管理自签名证书。 此任务使用命令行 **keytool** 命令，该命令是 JRE 的一部分。 您可以使用 GUI 工具 **iKeyman**(随 WebSphere MQ 一起提供) 来浏览密钥库并生成密钥。 使用命令 **strmqikm** 启动 **iKeyman**。

过程

1. 使用**新建遥测通道**向导来创建需要 SSL 连接的遥测通道 SSLSSOptClients。 此通道接受匿名客户机。

   从以下配置节来改写通道配置。 不要直接编辑遥测属性；请使用向导。

```
com.ibm.mq.MQXR.channel/SSLSSOptClients: \
com.ibm.mq.MQXR.Port=8883;\
com.ibm.mq.MQXR.Backlog=4096;\
com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerOptKey.jks;\
com.ibm.mq.MQXR.PassPhrase=password;\
com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

2. 为客户机生成密钥，以对遥测通道进行认证。

   a) 在新密钥库 SSServerOptKey.jks 中为遥测通道生成自签名密钥对：

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerOptKey.jks -storepass password -keypass password
   ```

   b) 使用 -rfc 选项将其公用证书导出为 ASCII 文件：

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerOptKey.jks -storepass password -rfc
   ```

   如果是在 Windows 上运行此任务，请双击 SSServerPublic.cer 以检查其内容。

   c) 将公用证书导入到新客户机信任库 SSClientTrust.jks 中：

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

   d) 创建空客户机密钥库 SSClientKey.jks。

   Keytool 没有用于创建空密钥库的命令。 您有两种选择：

   i) 运行 **strmqikm** 并创建密钥库 SSClientKey.jks，但是不添加任何密钥。

   ii) 执行第 428 页的『认证遥测通道和客户机』中的步骤 3a，但是尚不使用密钥。

认证遥测通道和客户机

客户机认证遥测通道，而遥测通道认证与其相连的客户机。 在通道上流动的消息已加密。

关于此任务

可以使用许多不同的密钥库编辑器来创建和管理自签名证书。 此任务使用命令行 **keytool** 命令，该命令是 JRE 的一部分。 您可以使用 GUI 工具 **iKeyman**(随 WebSphere MQ 一起提供) 来浏览密钥库并生成密钥。 使用命令 **strmqikm** 启动 **iKeyman**。

对于任务第 427 页的『认证遥测通道』，使用其他密钥库对遥测通道进行了配置。 可以使用同一密钥库，并省略步骤 第 428 页的『2』 以向密钥库中添加密钥。

过程

1. 使用**新建遥测通道**向导来创建需要 SSL 连接的遥测通道 SSLSSReqClients。 此通道仅接受已认证的客户机。

   从以下配置节来改写通道配置：

   ```
   com.ibm.mq.MQXR.channel/SSLSSReqClients: \
   com.ibm.mq.MQXR.Port=8884;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. 为客户机生成密钥，以对遥测通道进行认证。

   a) 在新密钥库 SSServerReqKey.jks 中为遥测通道生成自签名密钥对：

```
Keytool -genkey -noprompt -alias SSServerPrivate
        -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore SSServerReqKey.jks -storepass password -keypass password
```

b) 使用 `-rfc` 选项将其公用证书导出为 ASCII 文件：

```
Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
        -keystore SSServerReqKey.jks -storepass password -rfc
```

如果是在 Windows 上运行此任务，请双击 `SSServerPublic.cer` 以检查其内容。

c) 将公用证书导入到新客户机信任库 `SSClientTrust.jks` 中：

```
Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
        -keystore SSClientTrust.jks -storepass password
```

3. 为遥测通道生成密钥，以对客户机进行认证。

a) 在新密钥库 `SSClientKey.jks` 中为客户机生成自签名密钥对：

```
Keytool -genkey -noprompt -alias SSClientPrivate
        -dname "CN=mqttclient.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore SSClientKey.jks -storepass password -keypass password
```

b) 使用 `-rfc` 选项将其公用证书导出为 ASCII 文件：

```
Keytool -export -noprompt -alias SSClientPrivate -file SSClientPublic.cer
        -keystore SSClientKey.jks -storepass password -rfc
```

如果是在 Windows 上运行此任务，请双击 `SSClientPublic.cer` 以检查其内容。

c) 将公用证书导入到服务器密钥库 `SSServerReqKey.jks` 中：

```
Keytool -import -noprompt -alias SSClientPublic -file SSClientPublic.cer
        -keystore SSServerReqKey.jks -storepass password
```

遥测通道对专用密钥和可信证书均使用同一个库。

使用证书链认证 SSL 遥测连接

使用从认证中心获得的已签名证书或者通过实现您自己的认证过程而获得的证书来认证 SSL 连接。 您可以选择认证遥测通道，也可以选择认证遥测通道及其相连的客户机。 在连接上流动的消息已加密。

开始之前

在开始之前，请执行 第 425 页的『使用自签名证书来认证 SSL Telemetry 连接』 任务，以使 `PubSyncSSL.Java` 使用自签名证书处理受保护的 TCP/IP 连接。

关于此任务

在此任务中，修改第 425 页的『使用自签名证书来认证 SSL Telemetry 连接』中的任务第 427 页的『认证遥测通道』和第 428 页的『认证遥测通道和客户机』，以使用经过证书链认证的密钥。

您可以从认证中心获取此任务的证书，也可以使用 Web 站点（例如，http://www.openca.org/）来获取证书。 商业认证中心通常会在短期内免费提供试用证书。 已使用通过商业途径获得的证书测试了此任务。

另一个选项是构建您自己的认证过程，并使用 Web 站点（例如，https://www.openssl.org/）中的工具在您自己的计算机上运行此证书过程。

此任务中未使用 JRE `cacerts` 信任库。 您可以在任务第 430 页的『认证遥测通道』中的客户机上使用 JRE `cacerts` 信任库，而不是使用指定的信任库。 证书链可能由已在客户机上的 `cacerts` 库中具有其根证书的知名认证中心进行签名。 在此情况下，请勿在客户机上指定信任库。 确保如果在客户机上安装有多个 JRE，那么您管理正确的 `cacerts` 库。

过程

1. 如果您尚未这样做，那么请执行第 426 页的『将 PubSync.java 修改为使用 SSL』任务以将 PubSync.java 修改为使用 SSL。
2. 配置遥测通道，并创建密钥库以使用 SSL。

   仅认证遥测通道，或者认证遥测通道以及与其相连的客户机：

   - 执行第 430 页的『认证遥测通道』任务，以使用 SSL 进行连接，并认证遥测通道。
   - 执行第 431 页的『认证遥测通道和客户机』任务，以使用 SSL 进行连接，并认证遥测通道以及与它相连的客户机。
3. 停止并重新启动遥测 (MQXR) 服务以拾取对遥测通道配置的更改。
4. 运行客户机程序以查看配置是否工作。

认证遥测通道

客户机验证遥测通道以加密在通道上流动的消息的内容，并且确保客户机连接到正确的遥测通道。 服务器不会对客户机进行认证。

关于此任务

可以使用许多不同的密钥库编辑器来创建和管理证书。 此任务使用命令行 **keytool** 命令，该命令是 JRE 的一部分。 您可以使用 GUI 工具 **iKeyman**(随 WebSphere MQ 一起提供) 来浏览密钥库并生成密钥。 使用命令 **strmqikm** 启动 **iKeyman**。

过程

1. 使用**新建遥测通道**向导来创建需要 SSL 连接的遥测通道 SSLCAOptClients。 此通道接受匿名客户机。

   从以下配置节来改写通道配置。 不要直接编辑遥测属性；请使用向导。

   ```
   com.ibm.mq.MQXR.channel/SSLCAOptClients: \
   com.ibm.mq.MQXR.Port=8885;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```
2. 为客户机生成经过 CA 签名的密钥，以对遥测通道进行认证。

   a) 在新密钥库 SSServerOptKey.jks 中为遥测通道生成自签名密钥对：

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverOpt.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   密钥算法设置为 RSA，因为某些认证中心需要此密钥算法。 证书的公共名必须是唯一的，某些认证中心不会发出公共名完全相同的密钥。

   b) 以 ASCII 文件形式创建证书签名请求 (CSR)

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerOptKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   c) 运行认证中心软件，或者登录其 Web 站点。 当要求提供 CSR 文件时，请粘贴 CAServerOptKey.csr 的内容。

   d) 认证中心将以 ASCII 文件形式返回一个或两个证书以及一个已签名的响应文件。 将内容粘贴到两个或三个文件中：

   根证书
   粘贴到 CARoot.cer 中

中间证书
粘贴到 CAInter.cer 中

经过服务器签名的响应文件
粘贴到 CAServerOpt.rsp 中

此任务中未使用 JRE 证书库。 如果您接收到 CA 提供的一个根证书和一个已签名的响应，那么在以下步骤中使用该根证书和已签名的响应。 如果您接收到一个根证书和一个中间证书，那么使用该中间证书和已签名的响应。

e) 将已签名的服务器响应接收到从中发出了证书请求的服务器密钥库中。

接收响应时，将修改自签名证书，以便由 CA 对其进行签名。 如果您在接收响应前后在密钥库中查看此证书，那么签署者会发生更改。 如果未更改，那么密钥管理工具会报告错误。 在使用证书之前，请检查此证书，并验证签署者现在是否为 CA。

```
Keytool -import -noprompt -alias CAServer -file CAServerOpt.rsp
        -keystore CAServerOptKey.jks -storepass password
```

在某些密钥管理软件（例如，**iKeyman**）中，您将接收而不是导入响应文件。

f) 将 CA 证书导入客户机信任库中。

如果您接收到 CA 提供的两个证书，请导入中间证书；如果您只接收到一个证书，请导入根证书。

请完成下面任意一项任务：

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

或者：

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

认证遥测通道和客户机

客户机认证遥测通道，而遥测通道认证与其相连的客户机。 在通道上流动的消息已加密。

关于此任务

可以使用许多不同的密钥库编辑器来创建和管理证书。 此任务使用命令行 **keytool** 命令，该命令是 JRE 的一部分。 您可以使用 GUI 工具 **iKeyman**(随 WebSphere MQ 一起提供) 来浏览密钥库并生成密钥。 使用命令 **strmqikm** 启动 **iKeyman**。

对于任务第 430 页的『认证遥测通道』中的遥测通道，使用其他密钥库对遥测通道进行了配置。 可以使用同一密钥库，并省略步骤 第 431 页的『2』 以向密钥库中添加密钥。

过程

1. 使用**新建遥测通道**向导来创建需要 SSL 连接的遥测通道 SSLCAReqClients。 此通道仅接受已认证的客户机。

   从以下配置节来改写通道配置。 不要直接编辑遥测属性；请使用向导。

   ```
   com.ibm.mq.MQXR.channel/SSLCAReqClients: \
   com.ibm.mq.MQXR.Port=8886;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. 为客户机生成经过 CA 签名的密钥，以对遥测通道进行认证。

   a) 在新密钥库 CAServerReqKey.jks 中为遥测通道生成自签名密钥对：

```
Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAServerReqKey.jks -storepass password -keypass password
```

密钥算法设置为 RSA，因为某些认证中心需要此密钥算法。 证书的公共名必须是唯一的，某些认证中心不会发出公共名完全相同的密钥。

b) 以 ASCII 文件形式创建证书签名请求 (CSR)

```
Keytool -certreq -noprompt -alias CAServer -file CAServerReqKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAServerReqKey.jks -storepass password -keypass password
```

c) 运行认证中心软件，或者登录其 Web 站点。 当要求提供 CSR 文件时，请粘贴 `CAServerReqKey.csr` 的内容。

d) 认证中心将以 ASCII 文件形式返回一个或两个证书以及一个已签名的响应文件。 将内容粘贴到两个或三个文件中：

根证书
粘贴到 `CARoot.cer` 中

中间证书
粘贴到 `CAInter.cer` 中

经过服务器签名的响应文件
粘贴到 `CAServerReq.rsp` 中

此任务中未使用 JRE 证书库。 如果您接收到 CA 提供的一个根证书和一个已签名的响应，那么在以下步骤中使用该根证书和已签名的响应。 如果您接收到一个根证书和一个中间证书，那么使用该中间证书和已签名的响应。

e) 将已签名的服务器响应接收到从中发出了证书请求的服务器密钥库中。

接收响应时，将修改自签名证书，以便由 CA 对其进行签名。 如果您在接收响应前后在密钥库中查看此证书，那么签署者会发生更改。 如果未更改，那么密钥管理工具会报告错误。 在使用证书之前，请检查此证书，并验证签署者现在是否为 CA。

```
Keytool -import -noprompt -alias CAServer -file CAServerReq.rsp
        -keystore CAServerReqKey.jks -storepass password
```

在某些密钥管理软件（例如，**iKeyman**）中，您将接收而不是导入响应文件。

f) 将 CA 证书导入客户机信任库中。

如果您接收到 CA 提供的两个证书，请导入中间证书；如果您只接收到一个证书，请导入根证书。

请完成下面任意一项任务：

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

或者：

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

3. 为遥测通道生成密钥，以对客户机进行认证。

a) 在新密钥库 `CAClientKey.jks` 中为客户机生成自签名密钥对：

```
Keytool -genkey -noprompt -alias CAClientPrivate -keyalg RSA
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

密钥算法设置为 RSA，因为某些认证中心需要此密钥算法。 证书的公共名必须是唯一的，某些认证中心不会发出公共名完全相同的密钥。

b) 以 ASCII 文件形式创建证书签名请求 (CSR)

```
Keytool -certreq -noprompt -alias CAClient -file CAClientKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

c) 运行认证中心软件，或者登录其 Web 站点。 当要求提供 CSR 文件时，请粘贴 `CAClientKey.csr` 的内容。

d) 认证中心将以 ASCII 文件形式返回一个或两个证书以及一个已签名的响应文件。 将内容粘贴到两个或三个文件中：

根证书
粘贴到 `CARoot.cer` 中

中间证书
粘贴到 `CAInter.cer` 中

经过客户机签名的响应文件
粘贴到 `CAClient.rsp` 中

此任务中未使用 JRE 证书库。 如果您接收到 CA 提供的一个根证书和一个已签名的响应，那么在以下步骤中使用该根证书和已签名的响应。 如果您接收到一个根证书和一个中间证书，那么使用该中间证书和已签名的响应。

e) 将已签名的客户机响应接收到从中发出了证书请求的客户机密钥库中。

接收响应时，将修改自签名证书，以便由 CA 对其进行签名。 如果您在接收响应前后在密钥库中查看此证书，那么签署者会发生更改。 如果未更改，那么密钥管理工具会报告错误。 在使用证书之前，请检查此证书，并验证签署者现在是否为 CA。

```
Keytool -import -noprompt -alias CAClient -file CAClient.rsp
        -keystore CAClientKey.jks -storepass password
```

在某些密钥管理软件（例如，**iKeyman**）中，您将接收而不是导入响应文件。

f) 将 CA 证书导入服务器密钥库中。

如果您接收到 CA 提供的两个证书，请导入中间证书；如果您只接收到一个证书，请导入根证书。

请完成下面任意一项任务：

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAServerReqKey.jks -storepass password
```

或者：

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAServerReqKey.jks -storepass password
```

使用 C 创建第一个 MQ Telemetry Transport 发布程序应用程序

以教程方式描述了创建 MQTT 客户机发布程序应用程序的步骤。 对每行 C 语言代码都进行了说明。 在任务结束时，您将创建一个 MQTT 发布程序。

开始之前

开发的客户机应用程序使用客户机 MQTT v3 C 客户机库。 应用程序连接到 WebSphere MQ Telemetry 守护程序，以便设备发布消息。 请参阅 创建第一个发布程序 ，以获取客户机与 WebSphere MQ Telemetry 通信的示例。

关于此任务

示例是发布应用程序 `pubsync.c`。 程序 `pubsync.c` 将有效内容为 `Hello World!` 的消息发布到主题 `MQTT Example`，并等待确认发布已传递到守护程序。

为简单起见，不会测试所使用的某些函数的返回码来了解是否已正确完成。 在生产代码中，可以检查返回码以确保程序按预期运行。 如果发生意外错误，那么必须采取相应的操作。

通过设置 MQTT Example 的订户，您可以检查应用程序是否正常工作。

使用所选择的 C 语言开发环境来开发、构建和运行客户机。 如果您愿意，可以直接从示例中复制代码。

过程

1. 创建新的空源文件 pubsync.c
2. 创建文件 settings.h。 将图 2 中的代码复制到该文件中。

   程序中使用的所有参数都在 settings.h 中进行定义。 可以通过更改此文件中的值来覆盖值。
3. 后续步骤将解释代码。 请遵循相应的步骤，或者将代码从图 1 复制到 pubsync.c 中。
4. 为必需的标准库以及 MQTTClient.h 和 settings.h 文件添加头文件 include 语句。

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "MQTTClient.h"
#include "settings.h"
```

5. 开始定义 main() 函数。

```
int main(int argc, char* argv[])
{
```

6. 定义程序中使用的局部变量。

```
MQTTClient client;
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
MQTTClient_message pubmsg;
MQTTClient_deliveryToken token;
int rc;
```

   注: MQTTClient_connect 函数需要连接选项。 MQTTClient_connectOptions_initializer 包含缺省选项。
7. 创建客户机。

```
MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
```

   - &client 是指向新创建的客户机的句柄的指针。 当此函数返回的返回码为 0 时，此指针将包含新客户机的句柄。 此示例假设成功。 测试错误代码，以了解在生产代码中是否已正确完成。
   - ADDRESS 是守护程序监视入局客户机连接请求的 MQTT 端口的 URI。
   - CLIENTID 是用于向守护程序标识客户机的名称。 每个处于活动状态的客户机都必须具有唯一的名称。 如果您在两个正在运行的客户机中重复使用同一个客户机标识，那么这两个客户机中都会抛出异常，并且一个客户机会终止。 守护程序使用该名称来识别在断开连接后正在重新连接的客户机 tjhat，请参阅 客户机标识。
   - MQTTCLIENT_PERSISTENCE_NONE 指定客户机状态保存在内存中，并且如果发生系统故障，那么客户机状态会丢失。 MQTTCLIENT_PERSISTENCE_DEFAULT 指定基于文件系统的持久性，从而提供对某些故障的防护。 对于更专业化的应用程序，可以使用 MQTTCLIENT_PERSISTENCE_USER，它提供了一个接口供您实现自己的持久性机制。 有关更多详细信息，请参阅 MQTTClientPersistence.h 的 API 文档。 是否需要持久性是进行应用程序设计时应考虑的一个问题。 有关更多详细信息，请参阅 消息持久性。
   - MQTT 的缺省守护程序 TCP/IP 端口为 1883。 在此示例中，缺省地址设置为 tcp://localhost:1883。
   - 在您调用 MQTTClient_connect 函数之前，不会处理消息。
8. 将客户机连接到守护程序。

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
     printf("Failed to connect, return code %d\n", rc);
     exit(-1);
}
```

- 调用了 MQTTClient_connect 函数，从而将客户机句柄和指向连接选项的指针作为自变量进行传递。
- 测试了 MQTTClient_connect 调用产生的返回码，以确保连接请求成功。
- 如果 MQTTClient_connect 失败，那么程序将结束并产生错误代码 -1。
- 在应用程序连接之后，即可开始发布和预订。
- 每 20 秒发送一次简短的“保持活动”消息，以防止 TCP/IP 连接关闭。 此选项通过 conn_opts.keepAliveInterval 进行设置。
- 启动会话，而不检查是否已完成前一个连接中的其余未完成的消息，这是因为 conn_opts.cleansession 设置为 true。 有关更多详细信息，请参阅 清除会话。
- 不为连接创建最终消息。 有关更多详细信息，请参阅 最后遗嘱和遗嘱。

9. 使用数据填充 *MQTTClient_message* 结构，以定义消息有效内容及其属性。

```
pubmsg.payload = PAYLOAD;
pubmsg.payloadlen = strlen(PAYLOAD);
pubmsg.qos = QOS;
pubmsg.retained = 0;
```

- PAYLOAD 即是消息内容。
- 此示例使用字符串有效内容，但 MQTT 有效内容是字节数组。 需要字符串长度来指定有效内容大小。
- 示例发布 QoS=1 消息，因此请相应地设置该值
- 由于守护程序将不保留消息，因此保留属性设置为 false (0)。 有关更多详细信息，请参阅 保留发布。

10. 发布消息。

```
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
```

- 发布函数指定客户机、主题以及要发送到守护程序的有效内容。
- TOPIC 在 settings.h 中定义为 MQTT Example。
- 此外，还会为此函数传递指向 *MQTTClient_deliveryToken* 的指针。 当此函数返回时，将使用表示消息的令牌填充此指针。
- 该消息现在安全地传输到 MQTT 客户机，但尚未传输到守护程序。 如果消息具有 QoS=1 或 2，那么消息以本地方式存储，以防在传输完成之前客户机发生故障。
- 此函数将返回一个错误代码，可以测试此错误代码，以了解在生产代码中是否已正确完成。

11. 等待来自服务器的确认。

```
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
```

- pubsync.c 示例等待来自服务器的确认，它将确认是否已传送消息。
- client 和 token 自变量标识程序正在等待完成的特定消息。
- TIMEOUT 限制程序等待消息完成传送的时长。 任务 使用 C 创建 MQ Telemetry Transport 的异步发布程序 显示了如何使用回调函数接收应答而无需等待。
- 此函数将返回一个错误代码，可以测试此错误代码，以了解在生产代码中是否已正确完成。

12. 断开客户机与守护程序的连接。

```
MQTTClient_disconnect(client, 10000);
```

- 客户机将与服务器断开连接，并等待完成尚未完成的消息的回调函数（此示例未使用）。
- 第二个自变量指定停顿超时（以毫秒为单位）。 此示例将等待长达 10 秒钟，以便在断开连接之前完成其必须执行的任何其他工作。
- 此函数将返回一个错误代码，必须测试此错误代码，以了解在生产代码中是否已正确完成。

13. 释放客户机所使用的内存并结束该程序。

```
MQTTClient_destroy(&client);
}
```

结果

要查看由此客户机发送的发布，请创建 MQTT Example 主题订户。 有关更多详细信息，请参阅 使用 C 创建 MQ Telemetry Transport 订户

示例

图 1 是过程中描述的代码的完整列表。 通过图 2 中的 settings.h 文件，可以更改 pubsync.c 中使用的缺省参数。

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"

int pubsync_main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID,  MQTTCLIENT_PERSISTENCE_NONE, NULL);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
          printf("Failed to connect, return code %d\n", rc);
         exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for up to %d seconds for publication of %s\n"
          "on topic %s for client with ClientID: %s\n",
          TIMEOUT/1000, PAYLOAD, TOPIC, CLIENTID);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    printf("Message with delivery token %d delivered\n", token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

图 *111: pubsync.c*

```
#define ADDRESS   "tcp://localhost:1883"
#define CLIENTID  "ExampleClientPub"
#define TOPIC     "MQTT Example"
#define PAYLOAD   "Hello World!"
#define QOS      1
#define TIMEOUT   10000L
```

图 *112: settings.h*

使用 C 为 MQ Telemetry Transport 创建异步发布程序

以教程方式描述了创建 MQTT 客户机异步发布程序应用程序的步骤。 对每行 C 语言代码都进行了说明。 在任务结束时，您将创建一个 MQTT 异步发布程序。

在此任务中，您将遵循教程来修改第一个发布者应用程序。 通过修改，应用程序可以发送发布内容而不等待传送确认。 传送确认由您创建的回调函数来接收。

开始之前

开发的客户机应用程序使用客户机 MQTT v3 C 客户机库。 应用程序连接到 WebSphere MQ Telemetry 守护程序，以便设备发布消息。 请参阅 创建第一个发布程序 ，以获取客户机与 WebSphere MQ Telemetry 通信的示例。

关于此任务

示例是发布应用程序 pubasync.c。 程序 pubasync.c 将有效内容为 Hello World! 的消息发布到主题 MQTT Example，而不等待发布已传递到守护程序的确认。 将在回调函数 MQTTClient_deliveryComplete 中接收传送确认。

为简单起见，不会测试所使用的某些函数的返回码来了解是否已正确完成。 在生产代码中，可以检查返回码以确保程序按预期运行。 如果发生意外错误，那么必须采取相应的操作。

通过设置 MQTT Example 的订户，可以检查应用程序是否工作。

使用所选择的 C 语言开发环境来开发、构建和运行客户机。

过程 中的步骤从 第 433 页的『使用 C 创建第一个 MQ Telemetry Transport 发布程序应用程序』修改 pubsync.c 应用程序。 如果您愿意，可以直接从示例中复制代码。

过程

1. 创建新的空源文件 callback.h
2. 将图 2 中的代码复制到该文件中。
   - callback.h 声明异步客户机操作所需的三个回调方法。
   - 此外，还会声明变量 *deliveredtoken*。 此变量由不同执行线程上的主程序和回调进行访问。 因此，它被声明为具有“易失性”。 使用回调时，请注意确保以线程安全方式访问相关变量。
3. 创建新的空源文件 callback.c
4. 将图 3 中的代码复制到该文件中。
   - callback.c 实施由客户机用于异步操作的三个回调方法：delivered、msgarrvd 和 connlost。
5. 在 pubasync.c 中的其他 include 语句后面，为 callback.h 添加一个 include 语句。

   ```
   #include "callback.h"
   ```

6. 将 pubsync.c 的内容复制到新文件 pubasync.c. 中
7. 就在 pubasync.c 中的 MQTTClient_connect 函数调用前面，设置客户机的回调方法。

   ```
   MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
   ```

   - 您必须指定三个回调函数。 这些函数在 callback.c 中进行实施。
   - 当由于预订匹配而将消息发送到客户机时，将会调用 MQTTClient_messageArrived。 当客户机应用程序已经成功地接收到所接收到的消息时，此函数调用必须返回 true。 如果返回 false，那么向客户机表明应用程序在接收消息时发生问题。
   - 当客户机失去其与服务器的连接时，将会调用 MQTTClient_connectionLost。
   - 当 QoS1 或 QoS2 消息已到达并已由服务器确认时，将会调用 MQTTClient_deliveryComplete。 不会对 QoS0 消息调用此函数。 在此示例中，此函数将已传送的消息中的令牌保存在 *deliveredtoken* 中来指示消息已到达。
   - 当客户机与服务器断开连接时，必须调用 MQTTClient_setCallbacks。
   - 通过第二个自变量，可以将上下文信息传递到回调函数。 该自变量未在示例中使用，因此设置为 NULL。
8. 就在调用 MQTTClient_publishMessage 之前，清除 *deliveredtoken*。 当接收到令牌时，调用 MQTTClient_deliveryComplete 以设置 *deliveredtoken*。

   ```
   deliveredtoken = 0;
   ```

9. 移除 MQTTClient_waitForCompletion 调用以及跟在其后面的 printf 语句，并且替换为一个等待原始令牌和回调中所接收到的令牌的匹配项的循环。

   ```
   while(deliveredtoken != token);
   ```

这只是一个示例，并不用于处理在生产代码设计中必须囊括的许多情况。 这些情况包括：

- 在传送未完成的情况下，可以实施超时
- 多条消息可能未完成。 样本程序仅允许一个检查一个传送标记。

10. 断开客户机与守护程序的连接。

```
MQTTClient_disconnect(client, 10000);
```

- 客户机将与服务器断开连接，并等待完成未完成的消息的任何回调函数。
- 第二个自变量指定停顿超时（以毫秒为单位）。 此示例将等待长达 10 秒钟，以便在断开连接之前完成其必须执行的任何其他工作。
- 此函数将返回一个错误代码，必须测试此错误代码，以了解在生产代码中是否已正确完成。

11. 释放客户机所使用的内存并结束该程序。

```
MQTTClient_destroy(&client);
}
```

结果

要查看由此客户机发送的发布，请创建 MQTT Example 主题订户。 有关更多详细信息，请参阅 为 MQ Telemetry Transport 创建订户

示例

pubasync.c、callbacks.c 和 callbacks.h 是过程中所描述的代码的完整列表。

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    deliveredtoken = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for publication of %s\n"
           "on topic %s for client with ClientID: %s\n", PAYLOAD, TOPIC, CLIENTID);
    while(deliveredtoken != token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

图 *113: pubasync.c*

```
MQTTClient_deliveryComplete delivered;
MQTTClient_messageArrived msgarrvd;
MQTTClient_connectionLost connlost;

extern volatile MQTTClient_deliveryToken deliveredtoken;
```

图 *114: callback.h*

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt)
{
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
    }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    free(topicName);
    return 1;
}

void connlost(void *context, char *cause)
{
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

图 *115: callback.c*

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientPub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

图 *116: settings.h*

使用 C 为 MQ Telemetry Transport 创建订户

以教程方式描述了创建 MQTT 客户机订户应用程序的步骤。 对每行 C 语言代码都进行了说明。 在任务结束时，您将创建一个 MQTT 订户。

开始之前

开发的客户机应用程序使用客户机 MQTT v3 C 客户机库。 应用程序连接到 WebSphere MQ Telemetry 守护程序，以便设备发布消息。 请参阅 创建第一个发布程序，以获取客户机与 WebSphere MQ Telemetry 通信的示例。

关于此任务

提供了一个示例订户应用程序 `subscribe.c`。 程序 `subscribe.c` 预订主题 `MQTT Example`，并等待与预订匹配的发布，直到用户结束该程序。

订户将针对主题创建预订，并等待与此预订主题相匹配的消息。 当客户机重新连接时，就可以接收到客户机断开连接时发布的、与客户机先前所创建的预订相匹配的消息。 设备的 WebSphere MQ 遥测 (MQXR) 服务或守护程序可识别先前已通过客户机标识连接的客户机。 有关更多信息，请参阅 客户机标识。 `MQTTClient_connectOptions.cleansession` boolean 属性将控制是否接收到先前所发送的发布。 有关更多详细信息，请参阅 第 446 页的『清除会话』。

为简单起见，不会测试所使用的某些函数的返回码来了解是否已正确完成。 在生产代码中，可以检查返回码以确保程序按预期运行。 如果发生了意外错误，那么可以执行适当的操作。

您可以使用先前描述的发布示例程序将匹配的发布发送到设备的 WebSphere MQ Telemetry 守护程序。 或者，如果要将客户机连接到 WebSphere MQ Telemetry 通道，请使用 WebSphere MQ 资源管理器在 MQTT Example 主题上创建测试发布。

过程 中的指示信息假定您已在其中一个较早的任务中创建了 callback.c, callback.h 和 settings.h 文件。

使用所选择的 C 语言开发环境来开发、构建和运行客户机。 如果您愿意，可以直接从示例中复制代码。

过程

1. 对于该示例，创建 settings.h 的副本，将 *CLIENTID* 定义语句更改为以下内容：

   ```
   #define CLIENTID "ExampleClientSub"
   ```

   - 如果两个具有同一标识的客户机尝试连接至单个服务器，那么将强制使其中一个客户机断开连接。 通常，尝试建立新连接将成功，而旧连接将断开。
   - 如果更改 ClientID，那么将允许您使用先前开发的发布示例将消息发送至此订户。
2. 创建一个新的空白源文件 subscribe.c。
3. 后续步骤将解释代码。 请遵循相应的步骤，或者将代码从第 443 页的图 117 复制到文件 subscribe.c 中。
4. 为必需的标准库以及 MQTTClient.h 和 settings.h 文件添加头文件 include 语句。

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```

5. 开始定义 main() 函数。

   ```
   int main(int argc, char* argv[]) {
   ```

6. 定义程序中使用的局部变量。

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   MQTTClient_connect 函数需要连接选项。 MQTTClient_connectOptions_initializer 包含缺省选项。
7. 创建客户机。

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

   - &client 是指向新创建的客户机的句柄的指针。 当此函数返回的返回码为 0 时，此指针将包含新客户机的句柄。 此示例假设成功。 可以测试错误代码，以了解在生产代码中是否已正确完成。
   - ADDRESS 是守护程序监视入局客户机连接请求的 MQTT 端口的 URI。
   - CLIENTID 是用于向守护程序标识客户机的名称。 每个处于活动状态的客户机都必须具有唯一的名称。 如果您在两个正在运行的客户机中重复使用同一个客户机标识，那么这两个客户机中都会抛出异常，并且一个客户机会终止。 守护程序使用该名称来识别在断开连接后正在重新连接的客户机，请参阅 客户机标识。
   - MQTTCLIENT_PERSISTENCE_NONE 指定客户机状态保存在内存中，并且如果发生系统故障，那么客户机状态会丢失。 MQTTCLIENT_PERSISTENCE#_DEFAULT 指定基于文件系统的持久性，并提供某些保护以防失败。 对于更专业化的应用程序，可以使用 MQTTCLIENT_PERSISTENCE_USER，它提供了一个接口供您实现自己的持久性机制。 是否需要持久性是进行应用程序设计时应考虑的一个问题。 有关更多详细信息，请参阅 消息持久性。
   - MQTT 的缺省守护程序 TCP/IP 端口为 1883。 在此示例中，缺省地址设置为 tcp://localhost:1883。
   - 在您调用 MQTTClient_connect 函数之前，不会处理消息。

8. 将客户机连接至守护程序

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
    printf("Failed to connect, return code %d\n", rc);
    exit(-1);
}
```

- 调用了 MQTTClient_connect 函数，从而将客户机句柄和指向连接选项的指针作为自变量进行传递。
- 测试了 MQTTClient_connect 调用产生的返回码，以确保连接请求成功。
- 如果连接调用失败，那么程序将结束并产生错误代码 -1。
- 在应用程序连接之后，它就可以开始发布和预订。
- 每 20 秒发送一次简短的“保持活动”消息，以防止 TCP/IP 连接关闭。 此选项通过 conn_opts.keepAliveInterval 进行设置。
- 启动会话，而不检查是否已完成前一个连接中的其余未完成的消息，这是因为 *conn_opts.cleansession* 设置为 true。 有关更多详细信息，请参阅 清除会话。
- 不为连接创建最终消息。 有关更多详细信息，请参阅 最后的遗嘱和遗嘱

9. 预订主题。

```
MQTTClient_subscribe(client, TOPIC, QOS);
```

- 使用 MQTTClient_subscribe 函数向客户机应用程序预订所选主题。 主题名称中可以包括通配符。 有关更多详细信息，请参阅 第 458 页的『MQTT 客户机中的主题字符串和主题过滤器』。
- QoS 设置确定应用于发送至此订户的消息的最高服务质量。 服务器采用此设置与原始消息的 QoS 设置这两者当中的较小值来发送消息。
- 此函数将返回一个错误代码，可以测试此错误代码，以了解在生产代码中是否已正确完成。

10. 等待循环，直到用户通过键盘输入“Q”字符。

```
do {
    ch = getchar();
    } while(ch!='Q' && ch != 'q');
```

程序现在将等待消息到达。 在此示例中，所有消息处理操作都是在回调函数 MQTTClient_messageArrived 中进行的。 有关更多详细信息，请参阅 第 441 页的『接收消息』。

11. 断开客户机与守护程序的连接。

```
MQTTClient_disconnect(client, 10000);
```

- 客户机将与服务器断开连接，并等待完成尚未完成的消息的回调函数（此示例未使用）。
- 第二个自变量指定停顿超时（以毫秒为单位）。 此示例将等待长达 10 秒钟，以便在断开连接之前完成它必须执行的任何其他工作。
- 此函数将返回一个错误代码，可以测试此错误代码，以了解在生产代码中是否已正确完成。

12. 释放客户机所使用的内存并结束该程序。

```
MQTTClient_destroy(&client);
}
```

接收消息

关于此任务

当来自服务器的消息到达时，就会启动 MQTTClient_messageArrived 函数。 后续步骤将解释代码。

过程

1. 开始定义回调函数。 此定义必须与 MQTTClient_messageArrived 函数模板相匹配。

```
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
```

- 当调用了 MQTTClient_setCallbacks 函数时，context 提供对于传递给客户机库的上下文的访问权。 此示例中未使用此函数。
- topicName 是一个指针，指向将所接收到的消息发布至的主题。 如果您已经使用通配符进行预订，那么此参数标识用于此消息的特定主题。
- topicLen 是主题字符串的长度。 此选项是为必须在主题字符串中嵌入 NULL 字符的用户提供的。
- message 是一个指针，指向其中包含消息有效内容和属性的 MQTTClient_message 结构。

2. 定义使用的局部变量。

```
int i;
char* payloadptr;
```

此示例中使用这些变量以通过迭代有效内容来打印有效内容。

3. 打印一条消息，显示消息的主题和有效内容

```
printf("Message arrived\n");
printf("     topic: %s\n",topicName);
printf("   message: ");
payloadptr = message->payload;
for(i=0; i<message->payloadlen; i++){
    putchar(*payloadptr++);
    }
putchar('\n');
```

- 本示例假定收到的有效内容是一串可打印的字符。
- MQTT 有效内容是一个字节数组。 应用程序负责解释它们的含义。

4. 释放存储消息所用的内存空间。

```
MQTTClient_freeMessage(&message);
MQTTClient_free(topicName);
```

- 在此示例中，所有消息处理操作都是在回调函数中进行的。
- 请确保回调函数简短，并且尽快将控制权返回给它的调用线程。
- 传递了消息指针以在程序的主要部分进行处理。
- 完成处理时，主程序必须释放消息所使用的内存。 MQTTClient_freeMessage() 是一个便利函数，可将用于保存 *MQTTClient_message* 结构和消息有效内容的两个内存块返还给系统。 分配给 topicName 的内存必须按所显示那样单独释放。

5. 回调成功处理完消息后返回值 true

```
     return 1;
}
```

- 如果返回 true 值，那么表示客户机库可以认为已成功传递此消息。
- 如果回调函数无法正确处理此消息，那么将返回 false 值。 例如，如果回调函数正在将消息放入队列中供主程序进行处理，然而队列已满，那么将返回 false。
- 对于 QoS1 和 QoS2 消息，如果返回 false 值，那么表示未传递消息，并且已进一步尝试传递此消息。

示例代码

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc;
    int ch;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);

    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
        }
    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);

    MQTTClient_subscribe(client, TOPIC, QOS);
    do {
        ch = getchar();
        } while(ch!='Q' && ch != 'q');
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

图 *117: subscriber.c*

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt) {
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
        }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
void connlost(void *context, char *cause) {
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

图 *118: callback.h*

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientSub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

图 *119: settings.h*

MQTT 客户机编程概念

本节中描述的概念可帮助您了解 MQTT protocolV 3.1 的 Java， JavaScript 和 C 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。

com.ibm.micro.client.mqttv3 包含为 MQTT V 3.1 协议的客户机库提供公用方法的类。 随 IBM WebSphere MQ Telemetry 的安装一起提供了 com.ibm.micro.client.mqttv3 软件包的版本以及实现 Java SE 和 ME 协议的随附软件包。 要获取最新版本的 MQTT 客户机库 (Java 和 JavaScript) 以及查看或下载 API 文档，请参阅 “MQTT 客户机编程参考”。

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包。 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 protocolV 3.1 的参考实现。 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 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 WebSphere 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

MqttCallback 接口有三种回调方法；请参阅 Callback.java 中的示例实现。

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(MqttDeliveryToken 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 449 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，将完成交付。 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。

在已经发布消息之后，可以获取 MqttDeliveryToken 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 446 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(MqttTopic topic, MqttMessage message)

当客户机的与预订主题相匹配的预订到达时，就会调用 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。

如果使用值为 1 或 2 的 QoS 发送消息，那么该消息将由 MqttClientPersistence 类存储，之后 MQTT 客户机将调用 messageArrived。 messageArrived 的运行方式类似 deliveryComplete：它只能针对发布调用一次，当 messageArrived 返回到 MQTT 客户机时，此发布的本地副本将由 MqttClientPersistence.remove 除去。 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。 可以使用 token.waitForCompletion 并通过两种简单的方法来一次处理一个发布：

1. 暂停应用程序客户机，直到完成了发布的传递；请参阅第 404 页的图 88。
2. 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 446 页的表 70 描述了 MQTT Java 客户机中用于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 *70:* 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。
`MqttDeliveryToken.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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

“最后的消息”发布

如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

出版物

出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量

MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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

如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere MQ 中的主题字符串相同。

清除会话

MQTT 客户机和遥测 (MQXR) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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: WebSphere MQ 管理员可以强制关闭打开的会话，并删除所有会话信息。 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，将移除客户机的所有旧预订。 在会话断开连接时，将除去客户机在会话期间进行的任何新预订。

如果您在连接之前将 MqttConnectOptions.cleanSession 设置为 false，那么客户机创建的任何预订都会被添加至客户机在连接之前就已存在的所有预订中。 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。

在连接之前，您必须设置 cleanSession 方式；在整个会话期间都将保持此方式。 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。 如果您将方式从使用 cleanSession=false 更改为 cleanSession=true，那么此客户机先前的所有预订以及尚未收到的任何发布都将被废弃。

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

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere MQ 中的主题字符串相同。

客户机标识

客户机标识是一个 23 字节字符串，用于标识 MQTT 客户机。 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，假定某个设备发生了故障，也许客户通过呼叫服务台来通知您。 客户如何标识此设备？ 您如何使该标识与通常跟客户机相连的服务器相关联？ 您必须查阅将每台设备映射至客户机标识和服务器的数据库吗？ 设备的名称会标识它连接至哪个服务器吗？ 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。 您需要查找一个表以将客户机标识映射至实际设备吗？

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更换为一个新设备，此新设备与旧设备具有相同标识吗？ 您将分配新的标识吗？ 如果您更改实际设备，但是保持使用相同标识，那么会将未完成的发布和活动预订自动传递至新设备。

您如何确保客户机标识是唯一的？ 除了用于生成唯一标识的系统以外，您还必须通过执行可靠过程在客户机上设置标识。 也许客户机设备是一个“黑匣”，没有用户界面。 您在制造这种设备时会使用客户机标识（例如，使用它的 MAC 地址）吗？ 或者，在激活设备之前，您是否会执行软件安装以及用于配置此设备的配置过程？

您可以使用 48 位设备 MAC 地址来创建客户机标识，以使此标识较短并且是唯一的。 如果传输大小并不是关键问题，那么可以使用其余 17 个字节以使地址更容易管理。

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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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，可配置 WebSphere 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
出版物
出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere 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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

出版物
出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere 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，可配置 WebSphere 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)` 方法设置 "last will and testament"。

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，以使消息在 WebSphere MQ 中具有持久性，并保证传递。 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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

MQTT 客户机中的消息持久性

出版物

出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量

MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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

如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere MQ 中的主题字符串相同。

MQTT 客户机中的消息持久性

如果发布消息是使用“至少一次”或者“刚好一次”服务质量来发送的，那么这些消息具有持久性。 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 WebSphere 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“至多一次”(`QoS=0`)，那么客户机将在发布消息的同时废弃此消息。 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 WebSphere 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。 此消息存储在本地，仅当不再需要保证 "至少一次"， `QoS=1` 或 "正好一次"， `QoS=2` 和交付时，才会从客户机废弃此消息。
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 WebSphere MQ 中排队。 如果标记为 `QoS=0`，则作为非持久消息在 IBM MessageSight 和 IBM WebSphere MQ 中排队。 在 IBM WebSphere MQ 中，除非消息通道将 `NPMSPEED` 属性设置为 `FAST`，否则将在队列管理器之间 "正好传输一次" 非持久消息。

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 Java 的 MQTT 客户机 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。 客户机还具有 Java ME `MqttDefaultMIDPPersistence` 的缺省实现。

持久性类

`MqttClientPersistence`
将 `MqttClientPersistence` 实现实例作为 `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` 或 `MqttDefaultMIDPPersistence` 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` 或 `MqttDefaultMIDPPersistence`，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 客户机可实例化 `MqttDefaultFilePersistence`，并将文件放在缺省目录中。 目录的名称为 *`client identifier`*`-tcp` *`hostname portnumber`*。 将从目录名称字符串中除去 `"\", "\\", "/", ":"` 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。

`rcp.data` 是与 OSGi 或 Eclipse 富客户机平台 (RCP) 安装相关联的属性。

`usr.data` 是启动应用程序的 Java 命令在其中启动的目录。

`MqttDefaultMIDPPersistence`
`MqttDefaultMIDPPersistence` 有一个缺省构造函数，但没有参数。 它使用 `javax.microedition.rms.RecordStore` 包来存储消息。

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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere 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 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

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 协议允许的最大消息长度为 250 MB。

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 WebSphere 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> 文件夹中包含 clientId 和 qos，但是将来可能会更改此内容。

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 客户机可以在创建发布时从 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。 如果要查看是否已传递发布并且未受到阻止，请向 MQTT 客户机注册实施 MqttCallback 的回调类实例。 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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere 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以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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 协议不要求服务器将 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，发送方将告知接收方 - 它可以完成处理 "PUBREL" 消息。 如果发送方未接收到 "PUBREL" 消息的确认信息，那么会再次发送 "PUBREL" 消息，直到接收到确认信息为止。 当发送方接收到 "PUBREL" 消息的确认信息时，它就会删除它保存的消息。

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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

“最后的消息”发布

如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

保留的发布和 MQTT 客户机
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere MQ 中的主题字符串相同。

保留的发布和 MQTT 客户机

如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

使用 MqttMessage.setRetained 方法来指定是否保留了有关某个主题的发布。

要在 IBM WebSphere MQ 中删除保留发布，请运行 **CLEAR TOPICSTR** MQSC 命令。

如果创建了有效内容为空的发布，那么会将这个空白发布转发给订户。 其他 MQTT 代理程序可能不会将空白发布转发给订户。

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。

创建或更新保留发布时，请使用 QoS， 1 或 2 发送该发布。 如果发送 QoS 为 0 的发布，那么 IBM WebSphere 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。

使用保留的发布来记录测量结果的最新值。 保留的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。

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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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 WebSphere 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。

WebSphere MQ 命令可以除去预订。 使用 WebSphere 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，将移除客户机的所有旧预订。 在会话断开连接时，将除去客户机在会话期间进行的任何新预订。

如果您在连接之前将 `MqttConnectOptions.cleanSession` 设置为 `false`，那么客户机创建的任何预订都会被添加至客户机在连接之前就已存在的所有预订中。 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。

在连接之前，您必须设置 `cleanSession` 方式；在整个会话期间都将保持此方式。 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。 如果您将方式从使用 `cleanSession=false` 更改为 `cleanSession=true`，那么此客户机先前的所有预订以及尚未收到的任何发布都将被废弃。

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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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
传递令牌
“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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
出版物
出版物是与主题字符串关联的 MqttMessage 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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 WebSphere 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 WebSphere 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。 使用方法 `MqttClient.getTopic(java.lang.String topicString)`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。 使用预订方法创建主题过滤器; 例如， `MqttClient.subscribe(java.lang.String topicFilter)`。

主题字符串

主题字符串中描述了 IBM WebSphere MQ 主题字符串的语法。 在 Java 的 MQTT 客户机的 API 文档中的 `MqttClient` 类中描述了 MQTT 主题字符串的语法。 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 WebSphere MQ 的主题字符串必须遵循队列管理器名称的约定。 尤其是主题字符串中不能包含连字符。
2. 最大长度不同。 IBM WebSphere 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 WebSphere Message Broker 中，空的主题级别 `'...//...'` 无效。 然而，IBM WebSphere MQ 支持空的主题级。

与 IBM WebSphere MQ 发布/预订不同，`mqttv3` 协议没有“管理主题对象”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 WebSphere 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

基于主题的通配符方案中描述了 IBM WebSphere 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 Java 的 MQTT 客户机的 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。

每种类型的主题过滤器的语法几乎完全相同。 唯一的差别是不同的 MQTT 代理程序解释主题过滤器的方式有所不同。 在 WebSphere Message Broker V6 中，只能在主题过滤器末尾使用多点传送通配符。 在 IBM WebSphere MQ 中，可以在主题树中的任何级别使用多级通配符; 例如 `USA/#/Dutchess County`。

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) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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

传递令牌

“最后的消息”发布
如果 MQTT 客户机连接意外终止，可配置 WebSphere 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

出版物
出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT 客户机可创建发布内容以发送至 IBM WebSphere MQ，并预订 IBM WebSphere MQ MQ 上的主题以接收发布内容。

MQTT 客户机提供的服务质量
MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere 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
如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

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。

C 客户机编程概念

本主题描述了 MQ Telemetry Transport V 3.1 的 C 和 Java 客户机之间的差异。 本主题是对客户机概念和 C 参考信息的补充。

本主题的组织方式与第 444 页的『MQTT 客户机编程概念』相同。 每个标题都对应于 *WebSphere(r) MQ Telemetry Transport* 客户机编程概念中的主题。 这些部分描述了 C 客户机与 Java 客户机之间的差异。 未描述 Java 方法与 C 函数之间的特征符之间的微小差异。

C 客户机最常用于在遥测设备与设备的 WebSphere MQ Telemetry 守护程序之间实现轻量级适配器。 守护程序通常作为超轻量级遥测设备与遥测 (MQXR) 服务之间的网络集中器。

设备的 WebSphere MQ Telemetry 守护程序也是 C 客户机，描述了其行为与遥测 (MQXR) 服务的差异。 守护程序对连接到它的客户机不提供 JAAS 或 SSL 实施。

`mqttclient.dll` 和 `mqttclient.lib` 是 32 位 Windows 库，其中包含用于 MQ Telemetry Transport V 3.1 协议的 C 实现的客户机函数。 32 位 Linux 库是 `libmqttclient.so` 和 `libmqttclient.a`。 `MQTTClient.h` 和 `MQTTClientPersistence.h` 两个头文件中包含了客户机应用程序所需的函数和其他声明。 这些文件随 WebSphere MQ Telemetry 的安装一起提供。

要开发和运行 MQ Telemetry Transport 客户机，需要将这些文件复制到客户机设备上。 与 WebSphere MQ 客户机不同，您不需要安装单独的客户机运行时。

请参阅与 WebSphere MQ Telemetry 功能部件关联的许可条件，该功能部件用于管理将 MQ Telemetry Transport 客户机连接到设备的 WebSphere MQ 和 WebSphere MQ Telemetry 守护程序。

C 客户机是 MQ Telemetry Transport V 3.1 的参考实现。 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。 有关详细信息，请参阅 MQ 遥测传输格式与协议。

MQTT 客户机标识

第 448 页的『客户机标识』	客户机标识是一个 23 字节字符串，用于标识 MQTT 客户机。 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。

- 无差别。

出版物

第 453 页的『出版物』	出版物是与主题字符串关联的 `MqttMessage` 实例。 MQTT

- 针对服务质量为“发出消息之后无需等待应答”`QoS=0` 的发布未调用回调函数。

传递令牌

第 449 页的『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。

- 传送标记为 `int`。 它的 typedef 为 `MQTTClient_deliveryToken`
- 针对服务质量为“发出消息之后无需等待应答”`QoS=0` 的发布未调用回调函数。

保留的发布内容

第 456 页的『保留的发布和 MQTT 客户机』	如果您针对具有保留发布的主题创建预订，那么将立即转发给您有关该主题的最新保留发布。

- 仅当配置了持久性时，保留的消息才会保存在守护程序中; 请参阅 保存保留的消息和预订。

  对于 WebSphere MQ，服务质量会影响是否永久保存保留的消息。 当客户机连接到遥测服务时，如果队列管理器关闭，那么将丢失服务质量为“发出消息之后无需等待应答”`QoS=0` 的保留消息。

预订

第 457 页的『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。

- 仅当配置了持久性时，持久预订才会保存在守护程序中; 请参阅 保存保留的消息和预订。
- 可以同步接收发布。 调用 `MQTTClient_receive` 函数。

回调和同步

第 444 页的『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。

- C 客户机中的同步操作时模态的。 调用 MQTTClient_setCallback 会将客户机转变为异步方式。
- 在同步方式下，应用程序客户机必须主动生成控制，以便 MQTT 客户机可以处理应答并发出 MQTT ping 以保持网络处于活动状态。 通过调用 MQTTClient_receive 或 MQTTClient_yield 获得控件。

主题字符串和过滤器

第 458 页的『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 WebSphere MQ 中的主题字符串相同。

- 设备的 WebSphere MQ Telemetry 守护程序 # 以不同于 WebSphere MQ v7 的方式处理多级通配符。 /# 必须为过滤器字符串中的最后两个字符，才能使 # 表现为通配符。 在 WebSphere MQ v7 中， ../#/.. 是多级通配符的有效用法。 设备的 WebSphere MQ Telemetry 守护程序将多级通配符视为与 WebSphere MQ Broker v6 相同。

服务质量

第 454 页的『MQTT 客户机提供的服务质量』	MQTT 客户机提供了三种服务质量以向 WebSphere MQ 和 MQTT 客户机提供发布：“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 WebSphere MQ 以创建预订时，将使用“至少一次”服务质量来发送请求。

- 无差别。

消息持久性

第 451 页的『MQTT 客户机中的消息持久性』	如果发布消息是使用“至少一次”或者“刚好一次”服务质量来发送的，那么这些消息具有持久性。 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。

- 由于语言绑定差异，在 C 客户机中设置消息持久性机制如下。 使用设置为 MQTTClient_create 的第四个参数的三个选项之一来调用 MQTT C 客户机:

  MQTTCLIENT_PERSISTENCE_DEFAULT
  基于文件的持久性，其详细信息特定于客户机平台。

  MQTTCLIENT_PERSISTENCE_NONE
  数据仅保留在内存中，客户机停止后数据将丢失。 设备的 WebSphere MQ Telemetry 守护程序仅支持此选项。

  MQTTCLIENT_PERSISTENCE_USER
  您可以开发函数以实施自己的持久性机制。 在 MQTTClient_create 调用中，传递包含指向您函数的指针的结构 MQTTClient_persistence。 有关详细信息，请参阅 MQTT C 客户机参考信息。

清除会话

第 446 页的『清除会话』	MQTT 客户机和遥测 (MQXR) 服务都将维护会话状态信息。 状态信息用来确保进行“至少一次”和“刚好一次”传递，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。

- 无差别。

最后的消息

第 450 页的『“最后的消息”发布』	如果 MQTT 客户机连接意外终止，可配置 WebSphere 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。

- 无差别。

处理程序错误

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：

- 针对所有其他 WebSphere MQ 平台的 原因码

设计您的程序以处理每个调用可能产生的所有返回码。

系统中断

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：

UNIX， Linux 和 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 271 页的『落实和回退工作单元』）。 在 CICS 环境中，已禁用 MQCMIT 和 MQBACK 命令，因为您可以在由 CICS 管理的工作单元中进行 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。 通过 UNIX， Linux 和 Windows 系统上的 WebSphere MQ ，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 AIX， HP-UX， Linux， Solaris 和 Windows 系统上的日志文件的更多信息，请参阅 管理。

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，或者在 UNIX， Linux 和 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。

在 WebSphere MQ for Windows， UNIX 和 Linux 系统上，回退计数始终在队列管理器重新启动之后。

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 11 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 330 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 11 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此队列时，它会向消息添加一个头，其格式由死信头 (MQDLH) 结构描述; 请参阅 MQDLH-死信头 。 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

建议您将 MQDLH 结构添加到放入死信队列的所有消息中; 但是，如果您打算使用某些 WebSphere MQ 产品提供的死信处理程序，那么 **必须** 将 MQDLH 结构添加到消息中。

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 464 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。
- 在使用 CICS 的 WebSphere MQ for z/OS 上，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

使用此信息来了解 WebSphere MQ 多点广播编程任务 (例如，连接到队列管理器和异常报告)。

WebSphere MQ 多点广播设计为对用户尽可能透明，但仍与现有应用程序兼容。 定义 COMMINFO 对象并设置 TOPIC 对象的 **MCAST** 和 **COMMINFO** 参数，意味着现有 WebSphere MQ 应用程序不需要大量重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 465 页的『多点广播和消息队列接口』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）。

多点广播和消息队列接口

使用此信息来了解主要 MQI 概念以及它们如何与 WebSphere MQ 多点广播相关。

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。

WebSphere 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 19 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 *71: 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
	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
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

表 71: 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**
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

使用此信息来了解 WebSphere MQ 多点广播连续对等操作; 虽然 WebSphere MQ 连接到 WebSphere MQ 队列管理器，但消息不会流经该队列管理器。

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: 自动客户机重新连接

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

使用此信息可了解 WebSphere MQ 多点广播消息传递的数据转换工作方式。

WebSphere MQ 多点广播是一种共享的无连接协议，因此每个客户机都无法针对数据转换发出特定请求。 预订同一多点广播流的每个客户机都会接收相同的二进制数据; 因此，如果需要进行 WebSphere MQ 数据转换，那么将在每个客户机本地执行转换。

在客户机上针对 WebSphere MQ 多点广播流量转换数据。 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 346 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

使用此信息可了解 WebSphere MQ 多点广播事件处理程序和报告 WebSphere MQ 多点广播异常。

WebSphere MQ 多点广播通过调用事件处理程序以报告使用标准 WebSphere MQ 事件处理程序机制报告的多点广播事件来帮助确定问题。

单个多点广播事件可能会导致调用多个 WebSphere MQ 事件，因为可能有多个 MQHCONN 连接句柄使用同一个多点广播发送器或接收器。 但是，每个多点广播异常仅导致每个 WebSphere MQ 连接调用一个事件处理程序。

WebSphere MQ MQCBDO_EVENT_CALL 常量使应用程序能够注册回调以仅接收 WebSphere MQ 事件，而 MQCBDO_MC_EVENT_CALL 使应用程序能够注册回调以仅接收多点广播事件。 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

WebSphere 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，将仅发送事件处理程序 WebSphere 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

WebSphere MQ 多点广播异常包含回调函数的 **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 *72:* 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	不适用
MQMCEV_VERSION_CONFLICT	接收更新协议版本包	不适用
MQMCEV_RELIABILITY	发送器和接收器的不同可靠性方式	不适用
MQMCEV_CLOSED_TRANS	主题传送由 1 个源关闭	不适用
MQMCEV_STREAM_ERROR	在流中检测到错误	不适用
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	不适用
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	不适用
MQMCEV_REPAIR_DELAY	多点广播接收器未收到未完成 NAK 的修复包	不适用
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

表 72: 多点广播事件代码描述 (继续)

事件代码	描述	附加数据
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	不适用
MQMCEV_REPAIR_ALERT_OFF	发送器修复包发送率下降到正常状态	不适用
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	不适用
MQMCEV_SHM_PORT_UNUSABLE	检测到接收器实例使用的共享内存端口不可用	不适用
MQMCEV_CCT_GETTIME_FAILED	从“协调集群时间”获取时间失败	不适用
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

使用 .NET

WebSphere MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 WebSphere MQ MQI 客户机连接到 WebSphere MQ ，或者直接连接到 WebSphere MQ 服务器。

如果您有使用 Microsoft .NET Framework 的应用程序，并且想要利用 WebSphere MQ 的工具，那么必须使用 WebSphere MQ classes for .NET。

面向对象的 WebSphere MQ .NET 接口与 MQI 接口不同，因为它使用对象的方法，而不是使用 MQI 动词。

过程 WebSphere MQ 应用程序编程接口是围绕动词构建的，例如以下列表中的动词:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

这些动词都将它们要对其进行操作的 WebSphere MQ 对象的句柄作为参数。 因为 .NET 是面向对象的，所以 .NET 编程接口会轮到这一轮。 您的程序由一组 WebSphere MQ 对象组成，您可以通过对这些对象调用方法来对这些对象执行操作。 您可以使用 .NET 支持的任何语言编写程序。

使用过程接口时，通过使用调用 MQDISC (*Hconn*， CompCode， Reason) 从队列管理器断开连接，其中 *Hconn* 是队列管理器的句柄。

在 .NET 接口中，队列管理器由类 MQQueueManager 的对象表示。 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();
```

WebSphere MQ classes for .NET 是一组使 .NET 应用程序能够与 WebSphere MQ 交互的类。 它们表示应用程序使用的 WebSphere MQ 的各种组件，例如队列管理器，队列，通道和消息。 有关这些类的详细信息，请参阅 WebSphere MQ .NET 类和接口。

必须先安装 .NET Framework ，然后才能编译您编写的任何应用程序。 有关安装 WebSphere MQ classes for .NET 和 .NET Framework 的指示信息，请参阅 第 472 页的『安装 WebSphere MQ classes for .NET』。

相关概念

技术概述

第 471 页的『连接选项』
有三种方式可将 WebSphere MQ classes for .NET 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。

第 480 页的『使用 WebSphere MQ classes for .NET』
此主题集合描述了如何配置系统以运行样本程序来验证 WebSphere MQ classes for .NET 安装，以及如何运行您自己的程序。

第 483 页的『解决 WebSphere MQ .NET 问题』
如果程序未成功完成，请运行其中一个样本应用程序，并遵循诊断消息中给出的建议。

第 483 页的『编写和部署 WebSphere MQ .NET 程序』
要使用 WebSphere MQ classes for .NET 来访问 WebSphere MQ 队列，请以 .NET 支持的任何语言编写程序，其中包含将消息放入 WebSphere MQ 队列并从中获取消息的调用。

第 500 页的『Microsoft Windows Communication Foundation (WCF) 的 IBM WebSphere MQ 定制通道』
IBM WebSphere 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。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 7 页的『开发应用程序』
IBM WebSphere MQ 提供了多种方法，您可以通过这些方法开发应用程序来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。

WebSphere MQ classes for .NET 入门

WebSphere MQ classes for .NET 允许在 .NET 编程框架中编写的程序作为 WebSphere MQ MQI 客户机连接到 WebSphere MQ ，或者直接连接到 WebSphere MQ 服务器。

连接选项

有三种方式可将 WebSphere MQ classes for .NET 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。

客户机绑定连接

要将 WebSphere MQ classes for .NET 用作 WebSphere MQ MQI 客户机，可以将其与 WebSphere MQ MQI 客户机一起安装在 WebSphere MQ 服务器上或单独的机器上。 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， WebSphere MQ classes for .NET 将使用队列管理器 API ，而不是通过网络进行通信。 这为 WebSphere MQ 应用程序提供了比使用网络连接更好的性能。

要使用绑定连接，必须在 WebSphere MQ 服务器上安装 WebSphere MQ classes for .NET。

受管客户机连接

在此方式下建立的连接作为 WebSphere MQ 客户机连接到在本地或远程机器上运行的 WebSphere MQ 服务器。

用于以此方式连接的 .NET 的 WebSphere MQ 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。 有关这些信息的更多信息，请参阅 第 483 页的『受管客户机连接』。

安装 WebSphere MQ classes for .NET

WebSphere MQ classes for .NET (包括样本) 随 WebSphere MQ 一起安装。 Microsoft .NET Framework 有一个先决条件。

缺省情况下， WebSphere MQ classes for .NET 的最新版本作为标准 WebSphere MQ 安装的一部分安装在 *Java* 和 *.NET Messaging and Web Services* 功能部件中。 有关安装指示信息，请参阅 在 Windows 上安装 IBM WebSphere MQ 服务器 或 在 Windows 系统上安装 IBM WebSphere MQ 客户机 。

在多安装环境中，如果先前已将 WebSphere MQ classes for .NET 作为支持包安装，那么除非首先卸载支持包，否则无法安装 WebSphere MQ 。 随 WebSphere MQ 一起安装的 WebSphere 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 481 页的『样本应用程序』。

要在 32 位或 64 位平台上运行 WebSphere MQ classes for .NET ，必须已安装 Microsoft .NET Framework V2.0 或更高版本。

注: 如果在安装 WebSphere MQ V7.0.1 之前未安装 Microsoft .NET Framework v2.0 或更高版本，那么 WebSphere MQ 产品安装将继续且不会发生错误，但 WebSphere MQ classes for .NET 将不可用。 如果在安装 WebSphere MQ 7.0.1 之后安装 .NET Framework ，那么必须通过运行 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd 脚本来注册 WebSphere .NET 组合件，其中 *WMQInstallDir* 是安装 WebSphere MQ 7.0.1 的目录。 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% 中创建一组 amqi*.log 文件，用于记录执行的操作。

有关将 WebSphere MQ 定制通道用于具有 .NET 的 Microsoft WCF 3 的信息，请参阅: 第 500 页的『Microsoft Windows Communication Foundation (WCF) 的 IBM WebSphere MQ 定制通道』

.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。

WebSphere MQ .NET 类已支持非受管和服务器绑定方式连接中的分布式事务。 在这些方式下， WebSphere MQ .NET 类将其所有调用委派给 C 扩展事务客户机，后者代表 .NET 管理事务处理。

WebSphere MQ.NET 类现在以受管方式支持分布式事务，其中 WebSphere MQ .NET Classes 将 System.Transactions 名称空间用于分布式事务支持。 System.Transactions 基础结构通过支持在所有资源管理器 (包括 WebSphere MQ) 中启动的事务，使事务编程变得简单且高效。 WebSphere MQ .NET 应用程序可以使用 .NET 隐式事务编程或显式事务编程模型来放置和获取消息。 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。

WebSphere 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之间的事务。 WebSphere MQ 用作资源管理器。

WebSphere 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『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 WebSphere 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 名称空间的受管客户机事务支持与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

- WebSphere MQ .NET 完全受管客户机支持嵌套的 TransactionScope
- WebSphere MQ .NET 非受管客户机不支持嵌套的 TransactionScope

System.Transactions 中的从属事务

- WebSphere MQ .NET 完全受管客户机支持 System.Transactions 提供的从属事务工具。
- WebSphere MQ .NET 非受管客户机不支持 System.Transactions 提供的从属事务工具。

产品样本

在 `WebSphere MQ\tools\dotnet\samples\cs\base` 下提供了新产品样本 `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』

受管方式下的分布式事务

WebSphere MQ .NET 类将 System.Transactions 名称空间用于受管方式下的分布式事务支持。 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。

WebSphere MQ .NET 类提供基于 System.Transactions.Transaction 类的显式编程模型和使用 System.Transactions.TransactionScope 的隐式编程模型，其中事务由基础结构自动管理。

隐式事务

以下代码段描述了 WebSphere MQ .NET 应用程序如何使用 .NET 隐式事务编程来放置消息。

```
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

以下代码描述了 WebSphere MQ .NET 应用程序如何使用 .NET 显式事务编程模型来放置消息。

```
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*。

非受管方式下的分布式事务

WebSphere MQ.NET 类支持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) 的非受管连接 (客户机)。 在非受管方式下， WebSphere MQ .NET 类将其所有调用委派给代表 .NET 管理事务处理的 C 扩展事务客户机。

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。 WebSphere MQ .NET 类通过其非受管传输方式 (C 客户机) 公开此支持。 请参阅配置 XA 兼容的事务管理器

MTS 演变为事务处理 (TP) 系统，以在 Windows NT 上提供与在 CICS， Tuxedo 和其他平台上可用的功能相同的功能。 安装 MTS 时，会向称为 Microsoft 分布式事务协调程序 (MSDTC) 的 Windows NT 添加单独的服务。 MSDTC 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。

WebSphere MQ 通过实现一个接口 (专有资源管理器接口) ，使其能够将 DTC XA 调用映射到 WebSphere MQ(X/Open) 调用，从而与 MSDTC 兼容。 WebSphere MQ 充当资源管理器的角色。

当组件 (例如 COM +) 请求访问 WebSphere MQ 时，如果需要事务， COM 通常会使用相应的 MTS 上下文对象进行检查。 如果需要事务， COM 将通知 DTC 并自动为此操作启动完整的 WebSphere 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。 然后， DTC 发出对 MQMTS 的调用，而 MQMTS 又发出对 WebSphere MQ 的调用以落实或回滚事务。

使用非受管客户机编写 WebSphere MQ .NET 应用程序

要在 COM + 的上下文中运行， .NET 类必须继承自 System.EnterpriseServices.ServicedComponent。 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。
- 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 Framework 会在首次运行代码后处理注册。

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

WebSphere 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 服务必须正在运行并针对 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 WebSphere MQ 中可用的 C# 应用程序。 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 WebSphere 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

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
```

恢复事务

本部分描述了使用受管方式在 WebSphere 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。 来自其 System.Transactions 类库的 Microsoft .NET 提供了用于恢复不完整/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 的事务恢复模型中，事务管理器 (System.Transactions 和/或 Microsoft 分布式事务协调程序 (MS DTC)) 启动，协调和控制事务恢复。 基于 OLE Tx 协议 (Microsoft 的 XA 协议) 的资源管理器提供了用于配置 DTC 以驱动，协调和控制其恢复的选项。 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) 进行恢复：

Microsoft Distributed Transaction 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。

WebSphere MQ .NET 的事务恢复过程

本节描述如何使用 WebSphere MQ .NET 类恢复分布式事务。

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。 WebSphere 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。

WebSphere MQ .NET 中的事务恢复是一个两阶段进程。

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 是一个 .NET 受管应用程序，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

WebSphere MQ .NET XA Monitor 应用程序监视给定的队列管理器并恢复不完整的事务 (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。
- 如果事务已准备就绪，但 WebSphere 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。

必须从运行 WebSphere MQ .NET 客户机应用程序的系统运行监视应用程序。 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。

WebSphere MQ .NET 的事务恢复用例

以下是不同的使用方案:

- **WebSphere MQ 应用程序 (使用单个 DTC 和单个队列管理器实例):** 在此场景中，当您连接到队列管理器并在事务下运行工作单元 (UoW) 时，如果事务失败并且变得不完整，那么监视器应用程序将恢复并完成该事务。

  在此场景中，将有单个监视器应用程序实例在运行，因为单个队列管理器与事务相关联。

- **多个使用单个 DTC 和单个队列管理器实例的 WebSphere MQ 应用程序:** 在此场景中，单个 DTC 下存在多个 WMQ 应用程序，并且所有这些应用程序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。

  在此场景中，单个监视器应用程序将运行，因为在事务中使用一个队列管理器。

- **多个 WebSphere MQ 应用程序，多个 DTC 和不同的队列管理器实例:** 在此场景中，在不同的 DTC 下有多个 WMQ 应用程序 (即，每个应用程序都在不同的机器上运行) 并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。

  在此场景中，每个客户机 (用户或计算机) 将只有一个监视器应用程序实例在运行，因为每个客户机都在事务中使用了自己的队列管理器。

- **多个 WebSphere MQ 应用程序，多个 DTC 和多个相同的队列管理器实例:** 在此场景中，在不同的 DTC 下存在多个 WMQ 应用程序 (即，每个应用程序都在不同的机器上运行) ，并且所有这些应用程序都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。

  在此场景中，每个客户机 (用户或计算机) 将只有一个监视器应用程序实例在运行，因为每个客户机都有自己的队列管理器关联用于事务。

- **多个 WebSphere MQ 应用程序，单个 DTC 和不同的队列管理器实例:** 在此场景中，在单个 DTC 下有多个 WMQ 应用程序 (即，在计算机上，有多个 WMQ 应用程序正在运行) 并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。

  在此场景中，监视器应用程序的实例将与连接到的队列管理器一样多，因为每个应用程序都在事务中使用了自己的队列管理器，并且每个应用程序都必须恢复。

  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

XA 监视器应用程序必须手动运行。 可以随时将其启动。 当您在 SYSTEM.DOTNET.XARECOVERY.QUEUE 或您可以在对使用 WebSphere 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。

用于启动监视器应用程序的命令

```
WmqDotnetXAMonitor.exe -m <QueueManagerName> -n <ConnectionName> -c <Channel> -i
```

其中:

- **n** -主机 (端口) 格式的连接名称。 Connection Name 可以包含多个连接名称。 必须以逗号分隔列表形式提供多个连接名称，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** -通道名称。
- **m** -队列管理器名称。 可选
- **i** -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。 WebSphere MQ .NET 随附了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/Rollback
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/Rollback
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

使用 WebSphere MQ classes for .NET

此主题集合描述了如何配置系统以运行样本程序来验证 WebSphere MQ classes for .NET 安装，以及如何运行您自己的程序。

将队列管理器配置为接受 *TCP/IP* 客户机连接

要配置队列管理器以接受来自客户机的入局连接请求，请执行以下操作:

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[3]:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for WebSphere MQ classes for .NET')
```

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。

样本应用程序

要运行您自己的 .NET 应用程序，请使用验证程序的指示信息，将应用程序名称替换为样本应用程序。

提供了以下 5 个样本应用程序:

- 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 支持的任何语言编写应用程序。

“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。

[3] 在此样本中，我们不考虑安全影响。 对于生产系统，请考虑使用 SSL 或安全出口。 有关更多信息，请参阅 安全性 。

“Publish/subscribe”程序 (MQPubSubSample.cs)
此程序显示如何使用 WebSphere 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。

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* 表示安装了 WebSphere MQ 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

为构建样本应用程序，已为每种语言提供了批处理文件。

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，替换编译命令：

```
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
```

解决 *WebSphere MQ .NET* 问题

如果程序未成功完成，请运行其中一个样本应用程序，并遵循诊断消息中给出的建议。

第 480 页的『使用 WebSphere MQ classes for .NET』中描述了这些样本应用程序。

如果问题继续存在，并且您需要联系 IBM 服务团队，那么可能会要求您开启跟踪设施。

跟踪样本应用程序

有关使用跟踪工具的指示信息，请参阅第 500 页的『跟踪 WebSphere MQ .NET 程序』。

错误消息

您可能会看到以下常见错误消息：

类型为 "System.IO.FileNotFoundException

如果 amqmdnet.dll 或 amqmdxcs.dll 发生此错误，请确保两者都已在 "全局组合件高速缓存" 中注册，或者创建指向 amqmdnet.dll 和 amqmdxcs.dll 组合件的配置文件。 您可以使用作为 .NET Framework 一部分提供的 mscorcfg.msc 来检查和更改组合件高速缓存的内容。

如果安装 WebSphere MQ 时 .NET Framework 不可用，那么可能不会在全局组合件高速缓存中注册这些类。 您可以使用此命令手动重新运行注册过程

```
amqidnet -c MQ_INSTALLATION_PATH\bin\amqidotn.txt -l logfile.txt
```

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

有关此安装的信息将写入指定的日志文件 (在此示例中为 `logfile.txt`)。

编写和部署 WebSphere MQ .NET 程序

要使用 WebSphere MQ classes for .NET 来访问 WebSphere MQ 队列，请以 .NET 支持的任何语言编写程序，其中包含将消息放入 WebSphere MQ 队列并从中获取消息的调用。

WebSphere MQ 文档仅包含有关 C#， C++ 和 Visual Basic 语言的信息。

此主题集合提供了帮助编写应用程序以与 WebSphere MQ 系统交互的信息。 有关各个类的详细信息，请参阅 WebSphere MQ .NET 类和接口。

连接差异

您为 WebSphere MQ .NET 编程的方式与您要使用的连接方式有一些依赖关系。

受管客户机连接

当 WebSphere MQ classes for .NET 用作受管客户机时，与标准 WebSphere MQ MQI 客户机存在许多差异。

以下功能对受管客户机不可用：

- 通道压缩
- SSL 支持
- 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，与 .NET 受管客户机的服务器连接通道关闭时间可能比与其他客户机的服务器连接通道更长。

如果将 *NMQ_MQ_LIB* 设置为 managed 以使用受管 WebSphere MQ 问题诊断，那么不支持 **strmqtrc** 命令的参数 -i，-p，-s，-b 或 -c。

使用 XA 事务的受管 .NET 应用程序将无法使用 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 应用程序将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 以显式选择连接类型，如下表中所示

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，从 V 7.1 开始， mqm.dll 和 mqm.pdb 仅是客户机软件包的一部分。	

如果选择在环境中不可用的连接类型，例如，指定 mqic32xa.dll 并且不具有 XA 支持，那么 WebSphere 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 WebSphere MQ 问题诊断测试， .NET 数据转换和其他受管低级 WebSphere MQ 函数。

NMQ_MQ_LIB 的所有其他值导致 .NET 进程使用非受管 WebSphere MQ 问题诊断测试和数据转换以及其他非受管低级 WebSphere MQ 函数 (假定系统上安装了 WebSphere MQ MQI 客户机或服务器)。

WebSphere MQ .NET 选择连接类型，如下所示：

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" ，那么不会完全管理问题诊断测试，数据转换和其他低级别功能 (假定系统上安装了 WebSphere MQ MQI 客户机或服务器)。
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 WebSphere MQ 服务器，那么它将作为服务器进行连接。
6. 如果安装了 WebSphere MQ MQI 客户机，那么它将作为非 XA 客户机进行连接。
7. 否则，将作为受管客户机连接。

WebSphere MQ classes for .NET 的配置文件

.NET 客户机应用程序可以使用 WebSphere MQ MQI 客户机配置文件，如果您使用的是受管连接类型，那么还可以使用 .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

用于 .NET 客户机应用程序的 WebSphere MQ 类可以使用与任何其他 WebSphere MQ MQI 客户机相同的客户机配置文件。 该文件通常名为 mqclient.ini，但您也可以指定其他文件名称。 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 WebSphere MQ MQI 客户机配置文件。

只有 WebSphere MQ MQI 客户机配置文件中的以下属性与 WebSphere MQ classes for .NET 相关。 如果指定其他属性，那么将没有任何效果。

节	属性
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ServerConnectionParms
ClientExitPath	ExitsDefaultPath
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
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 WebSphere 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，才会对 .NET 应用程序配置文件设置执行操作，对于其他连接类型，将忽略这些设置。

.NET 应用程序配置文件及其格式由 Microsoft 定义，供在 .NET 框架中常规使用，但本文档中提到的特定节名称，键和值特定于 Websphere MQ。

.NET 应用程序配置文件的格式为多个 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 属性的部分，键和值:

```
<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 Stanzas 和 Attributes 的关键字完全匹配。

请参阅 Microsoft 文档以获取更多信息。

代码片段示例

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 a WebSphere 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 a WebSphere 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 a WebSphere MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("A WebSphere 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
```

队列管理器上的操作

本部分描述如何使用 WebSphere MQ classes for .NET 连接到队列管理器并与之断开连接。

设置 *WebSphere MQ* 环境

在使用客户机连接连接到队列管理器之前，必须设置 WebSphere MQ 环境。

注: 在服务器绑定方式下使用 WebSphere MQ classes for .NET 时，不需要执行此步骤。

.NET 编程接口允许您使用 NMQ_MQ_LIB 定制值，但还包含类 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，客户机尝试在端口 1414 上连接到 WebSphere 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

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，必须对队列管理器具有查询 (`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();
```

通过 WebSphere MQ .NET，您还可以使用 MQQueue 构造函数来创建队列。 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。 有许多主题的构造函数; 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 MQPUT 和 MQGET 放置和获取字节数组的过程接口不同， WebSphere MQ classes for .NET 放置和获取 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。

创建新消息实例时，所有 MQMD 参数都会自动设置为其缺省值，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。

请参阅 MQMessage .NET 类 以获取更多详细信息。

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。 当您的消息将由另一个 .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。 它们还有助于 WebSphere MQ 与 JMS 应用程序之间的通信。 有关 WebSphere 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
- 长
- Object
- 短
- 字符串

例如，要获取 WebSphere MQ 属性 myproperty (字符串)，请使用调用 `message.GetStringProperty('myproperty')`。 您可以选择传递属性描述符， WebSphere MQ 将完成该属性描述符。

处理错误

处理使用 `try` 和 `catch` 块的 .NET 的 WebSphere MQ 类所产生的错误。

.NET 接口中的方法不会返回完成代码和原因码。 相反，只要由 WebSphere MQ 调用生成的完成代码和原因码都不为零，它们就会抛出异常。 这将简化程序逻辑，使您不必在每次调用 WebSphere MQ 之后检查返回

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。 WebSphere MQ classes for .NET 允许在多个线程之间共享队列管理器对象，但确保对目标队列管理器的所有访问都同步。

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。

WebSphere MQ .NET 的实现可确保对于特定连接 (MQQueueManager 对象实例) ，对目标 WebSphere MQ 队列管理器的所有访问都是同步的。 缺省行为是要对队列管理器发出调用的线程被阻塞，直到该连接的所有其他正在进行的调用完成为止。 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。）

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与 .NET 配合使用

您可以将客户机通道定义表 (CCDT) 与 WebSphere 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。

如果以多种方式指定位置，那么环境变量将优先于客户机配置文件，而 .NET 应用程序配置文件将优先于其他两种方法。 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 WebSphere MQ .NET 客户机环境中，可以通过多种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，节名称 CHANNELS ，键 ServerConnection 参数 (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 的位置在 .NET 应用程序配置文件中指定 (仅适用于受管连接)
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 WebSphere MQ 中使用通道出口。NET

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，请定义用于实现相应接口的新 .NET 类。 WebSphere MQ 软件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* 给出可用于 *dataBuffer* 中的出口的最大长度 (来自 *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。

返回时， *dataOffset* 和 *dataLength* 的值应设置为指向 .NET 类随后应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

V 7.5.0.6 从 Version 7.5.0, Fix Pack 6 开始，通过受管 .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 WebSphere MQ .NET MQChannelDefinition 类中设置。 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 339 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 493 页的『将客户机通道定义表与 .NET 配合使用』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 (根据需要)。 *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。 必要时， .NET 客户机代码将通过解析字符串规范来装入并创建指定类的实例。

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 493 页的『将客户机通道定义表与 .NET 配合使用』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 中的自动客户机重新连接

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 WebSphere MQ .NET 客户机，那么当您使用 MQQueueManager 类来创建队列管理器时，可以通过为 CONNECT_OPTIONS_PROPERTY 指定适当的值来选择获取自动客户机重新连接。 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）。

安全套接字层（SSL）支持

以下部分不适用于受管客户机。

WebSphere MQ classes for .NET 客户机应用程序支持安全套接字层 (SSL) 加密。 SSL 提供了通信加密、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。

启用 *SSL*

只有客户机连接支持 SSL。 要启用 SSL，您必须指定在与队列管理器通信时要使用的 CipherSpec，并且这必须与目标通道设置的 CipherSpec 匹配。

要启用 SSL，请使用 MQEnvironment 的 SSLCipherSpec 静态成员变量指定 CipherSpec。 以下示例连接到名为 SECURE.SVRCONN.CHANNEL 的 SVRCONN 通道，已经将它设为要求 SSL 的 CipherSpec 为 NULL_MD5：

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "NULL_MD5";
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 SSL 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 SSL 证书标识它自己，该 SSL 证书包含一个专有名称（DN）。

WebSphere MQ .NET 客户机应用程序可以使用此 DN 来确保它与正确的队列管理器进行通信。 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 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 安全性。

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 *SSL* 时的错误处理

当使用 SSL 连接到队列管理器时， WebSphere MQ classes for .NET 可以发出以下原因码:

MQRC_SSL_NOT_ALLOWED
已设置 SSLCipherSpec 属性，但使用了绑定连接。 只有客户机连接支持 SSL。

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。

使用 .NET 监控器

请参阅 在 Windows 上只能与主安装配合使用的功能部件 以获取重要信息。

.NET 监视器是类似于 WebSphere MQ 触发器监视器的应用程序。 您可以创建 .NET 组件，只要在受监视队列上接收到消息，就会将这些组件实例化，然后处理该消息。 .NET 监视器由 runmqdnm 命令启动，并由 endmqdnm 命令停止。 有关这些命令的详细信息，请参阅 runmqdnm 和 endmqdnm。

要使用 .NET 监视器，请编写用于实现 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。 不能将其注册为 RequiresNew ，因为 .NET 监视器已启动事务。

组件从 runmqdnm 接收 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，则在 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，该组件必须根据情况处理消息并将控制权返回给 .NET 监视器。

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，因此它具有该消息的完整上下文信息，例如，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 WebSphere MQ。

代码片段示例

本主题包含从 .NET 监视器获取消息并将其打印的两个组件示例， 一个组件使用事务处理，另一个组件使用非事务处理。 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 WebSphere MQ .NET 程序

用于编译以各种语言编写的 .NET 应用程序的样本命令。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

要使用 WebSphere MQ classes for .NET 来构建 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 WebSphere MQ classes for .NET 来构建 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 WebSphere MQ classes for .NET 来构建受管 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。

跟踪 WebSphere MQ .NET 程序

在 WebSphere MQ .NET 中，您可以像在使用 MQI 的 WebSphere MQ 程序中一样启动和控制跟踪工具。

但是，strmqtrc 命令的 -i 和 -p 参数使您可以指定进程和线程标识符，而已命名的进程不会产生任何效果。

通常只需要在 IBM 服务人员的请求下使用跟踪工具。

有关跟踪命令的信息，请参阅 在 Windows 上使用跟踪 。

Microsoft Windows Communication Foundation (WCF) 的 IBM WebSphere MQ 定制通道

IBM WebSphere 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 501 页的『有关将 WebSphere MQ 定制通道用于具有 .NET 的 WCF 的简介 3』
使用 WebSphere MQ 定制通道 for Windows Communication Foundation (WCF) with .NET 3 的程序员可用的信息概述。

第 504 页的『将 WebSphere MQ 定制通道用于 WCF』
使用 Windows Communication Foundation (WCF) 的 WebSphere MQ V7 定制通道为程序员提供的信息概述。

第 518 页的『使用 WCF 样本』
Windows Communication Foundation (WCF) 样本提供了有关如何使用 WebSphere MQ 定制通道的一些简单示例。

第 523 页的『 WebSphere MQ 的 WCF 定制通道上的问题确定』
您可以使用 WebSphere MQ 跟踪来收集有关 WebSphere MQ 代码的各个部分的详细信息。 使用 Windows Communication Foundation (WCF) 时，将为与 Microsoft WCF 基础结构跟踪集成的 WCF 定制通道跟踪生成单独的跟踪输出。

有关将 WebSphere MQ 定制通道用于具有 .NET 的 WCF 的简介 3

使用 WebSphere MQ 定制通道 for Windows Communication Foundation (WCF) with .NET 3 的程序员可用的信息概述。

什么是 WCF 的 WebSphere MQ 定制通道?

WebSphere 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。

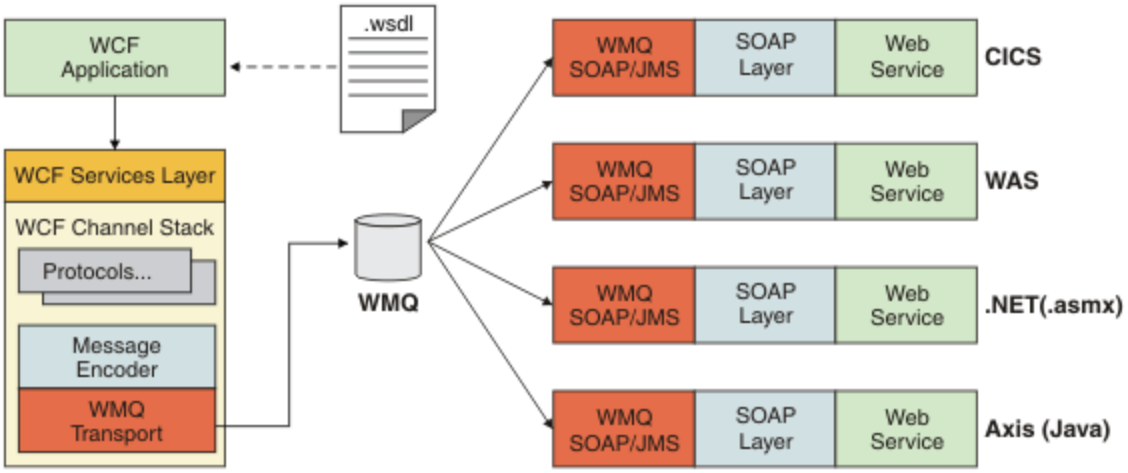
WebSphere MQ 的定制通道是传输通道，因此必须与使用 WCF 定制绑定的应用程序所需的消息编码器和可选协议配对。 通过这种方式，已开发为使用 WCF 的应用程序可以使用 WebSphere MQ 的定制通道，以与使用 Microsoft 提供的内置传输相同的方式发送和接收数据，从而实现与 WebSphere MQ 的异步，可扩展和可靠消息传递功能的简单集成。 有关受支持功能的完整列表，请参阅：第 504 页的『WCF 定制通道特性和功能』。

何时以及为何将 WebSphere MQ 定制通道用于 WCF?

WebSphere MQ 定制通道可用于以与 Microsoft 提供的内置传输相同的方式在 WCF 客户机和服务之间发送和接收消息，从而使应用程序能够访问 WCF 统一编程模型中 WebSphere MQ 的功能。

WCF 的 WebSphere MQ 定制通道的典型使用模式场景是作为通过 WebSphere MQ (SOAP/JMS) 托管的 Web Service 的接口

使用 WebSphere MQ 的 SOAP over JMS 消息格式传输消息，使 WCF 客户机和服务也能够调用或由其他与此格式兼容的 WebSphere MQ 应用程序或托管环境 (包括 Web Service 和在 WebSphere Application Server , CICS， Axis v1 (Java) 中运行的客户机) 调用。 和 .asmx (.NET) , 如下图所示:



有关 SOAP over JMS 的详细信息，请参阅: 第 791 页的『WebSphere MQ Transport for SOAP』

图 中典型方案的示例如下:

1. 在 WebSphere Application Server 中托管并通过 WebSphere MQ (使用对 WebSphere Application Server 中的 SOAP over JMS 的支持) 公开的 Web Service

2. 描述服务的 WSDL 文档然后可以由 WCF 工具用来生成客户机代理和配置，然后将创建一个适当的 WCF 通道栈，包括定制通道。
3. 然后，客户机应用程序可以使用该代理以与任何其他 Web 服务相同的方式启动 Web 服务。

该通道通常与 WCF 文本/SOAP 消息编码器一起使用，但是如果需要，该通道也可与其他 WCF 消息编码器配对。 使用备用编码器还可以提供与不支持 SOAP over JMS 的本机 WebSphere MQ 应用程序的有限集成，但这不是通道的主要角色。

在 WCF 环境中使用定制通道的主要优点有：

- 异步调用：支持触发并忽略客户机操作，其客户机与服务和功能的可用性（例如响应和多次反射的重新路由）减少。
- 可靠的扩展特性：基于队列的消息传递允许以可预测方式在系统中增加容量。
- 服务质量：消息是有形的和可追踪的，并且可以轻易掌控和管理。

针对 WCF 的 WebSphere MQ 定制通道的软件需求和安装指示信息

本主题概述了针对 WCF 的 WebSphere MQ 定制通道的软件需求和安装信息。

适用于 WCF 的 WebSphere MQ 定制通道只能连接到 WebSphere MQ V7 或更高版本的队列管理器。

WebSphere MQ 的 WCF 定制通道的软件需求

此信息列出了 WebSphere MQ 的 WCF 定制通道的软件需求。

运行时环境

- Microsoft .NET Framework v3.0 或更高版本必须安装在主机上。
- 缺省情况下， *Java* 和 *.NET Messaging and Web Services* 作为 WebSphere MQ 7.0.1 安装程序的一部分进行安装。 将定制通道所需的 .NET 组合件安装到全局组合件高速缓存中。

注: 如果在安装 WebSphere MQ V7.0.1 之前未安装 Microsoft .NET Framework v2.0 或更高版本，那么 WebSphere MQ 产品安装将继续且不会发生错误，但 WebSphere MQ 定制通道不可用。 如果在安装 WebSphere MQ 7.0.1 之后安装了 .NET Framework ，那么必须通过运行 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd 脚本来激活 WebSphere MQ 定制通道，其中 *WMQInstallDir* 是 WebSphere MQ 7.0.1 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% 中创建一组 amqi*.log 文件，用于记录执行的操作。 如果 .NET 从较早版本 (例如，从 .NET v2.0) 升级到 v3.0 或更高版本，那么无需重新运行 amqiRegisterdotNet.cmd 脚本。

开发环境

- Microsoft Visual Studio 2008 或 Windows Software Development Kit for .NET 3.0 或更高版本。
- 必须在主机上安装 Microsoft .NET Framework V3.5 或更高版本，才能构建样本解决方案文件。

注: 如果在安装 WebSphere MQ V7.0.1 之前未安装 Microsoft .NET Framework v2.0 或更高版本，那么 WebSphere MQ 产品安装将继续且不会发生错误，但 WebSphere MQ 定制通道不可用。 如果在安装 WebSphere MQ 7.0.1 之后安装了 .NET Framework ，那么必须通过运行 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd 脚本来激活 WebSphere MQ 定制通道，其中 *WMQInstallDir* 是 WebSphere MQ 7.0.1 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% 中创建一组 amqi*.log 文件，用于记录执行的操作。 如果 .NET 从较早版本 (例如，从 .NET v2.0) 升级到 v3.0 或更高版本，那么无需重新运行 amqiRegisterdotNet.cmd 脚本。

适用于 WCF 的 WebSphere MQ 定制通道: 安装了哪些内容?

WebSphere 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 WebSphere MQ 7.0.1 安装的一部分进行安装。

面向 WCF 的 WebSphere MQ 定制通道

缺省情况下， WCF 的 WebSphere MQ 定制通道作为 WebSphere MQ 7.0.1 安装的一部分进行安装; 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。从较低版本升级到 WebSphere MQ 7.0.1 时，如果先前在较低版本中安装了 Java and .NET Messaging and Web Services 组件，那么缺省情况下更新将安装 WCF 的 WebSphere MQ 定制通道。

Java and .NET Messaging and Web Services 组件包含 IBM.XMS.WCF.dll 文件，而 IBM.XMS.WCF.dll 文件是包含 WCF 接口类的主定制通道组合件。 此文件安装在全局组合件高速缓存 (GAC) 中，并且还在以下目录中提供: *MQ_INSTALLATION_PATH*\bin，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 7.0.1 安装所在的目录。

使用定制通道所需的密钥类位于 名称空间*: IBM.XMS.WCF* 和:

传输绑定名称	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement
传输绑定导入器	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter
传输绑定配置	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig

WebSphere MQ 定制通道样本

这些样本提供了一些简单示例，说明如何使用 WCF 的 WebSphere MQ 定制通道。 样本及其关联文件位于 *MQ_INSTALLATION_PATH*\tools\wcf\samples\ 目录中，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。 有关 WebSphere MQ 定制通道样本的更多信息，请参阅: 第 518 页的『使用 WCF 样本』

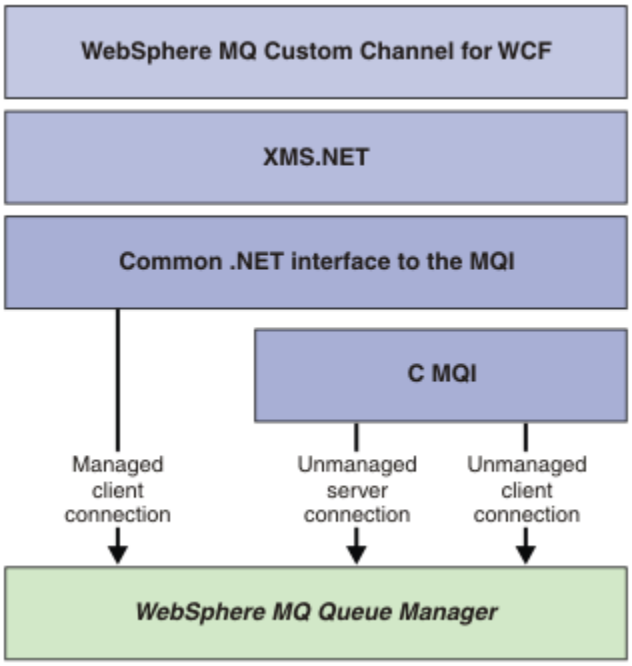
svcutil.exe.config

svcutil.exe.config 是使 Microsoft WCF svcutil 客户机代理生成工具能够识别定制通道所需的配置设置的示例。 svcutil.exe.config 文件位于 *MQ_INSTALLATION_PATH*\tools\wcf\docs\examples\ 目录中，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。 有关使用 svcutil.exe.config 的更多信息，请参阅: 第 516 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

针对 WCF 的 WebSphere MQ 定制通道集成在 IBM Message Service Client for .NET (XMS .NET) API 之上。

WCF 体系结构如下图所示:



缺省情况下，所有必需组件都随 WebSphere MQ V7.0.1 安装一起安装。

这三个连接是: 受管客户机连接，非受管服务器连接和非受管客户机连接。 有关这些连接的更多信息，请参阅第 508 页的『WCF 连接选项』。

将 WebSphere MQ 定制通道用于 WCF

使用 Windows Communication Foundation (WCF) 的 WebSphere MQ V7 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。 WebSphere MQ V7 现在可以在 .NET Framework 3 中以与 Microsoft 提供的内置通道相同的方式用作 WCF 中的定制通道。

通过定制通道传输的消息将根据 WebSphere MQ V7 的 SOAP over JMS 实现进行格式化。 然后，应用程序可以与 WCF 或 WebSphere SOAP over JMS 服务基础结构所托管的服务进行通信。 有关 SOAP over JMS 的详细信息，请参阅: 第 791 页的『WebSphere MQ Transport for SOAP』

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

WebSphere 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 WebSphere 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

例如，使用 WCF 的 WebSphere MQ 单向定制通道通过单向通道形状从 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

例如，使用针对 WCF 的 WebSphere MQ 请求/应答定制通道来异步发送两个方向的消息; 必须将同一客户机实例用于异步消息传递。 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

connectionFactory
需要 connectionFactory 参数。 有关此参数的语法，请参阅 用于 Web Service 部署的 URI 语法和参数 。

initialContextFactory
initialContextFactory 参数是必需的，必须设置为 "com.ibm.mq.jms.Nojndi" 才能与 WebSphere Application Server 和其他产品兼容 (请参阅 第 842 页的『将服务部署到 WebSphere Application Server 以使用 WebSphere Transport for SOAP』)。

WCF 定制通道保证传递

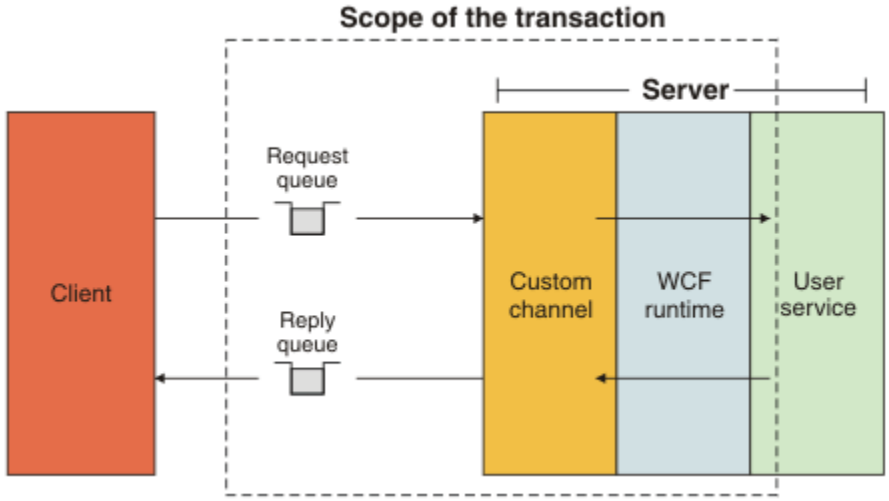
保证传递可保证服务请求或回复会被采取操作且不会丢失。

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。 有关设置 URI 属性的更多信息，请参阅用于 Web 服务部署的 URI 语法和参数。

不支持分布式事务，并且事务的作用域不会超出 WebSphere MQ 执行的请求和应答消息处理。 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 507 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 513 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 的 WebSphere MQ 定制通道仅支持将 SSL 用于与队列管理器的非受管客户机连接。

可使用以下两种方式之一来指定 SSL：

- 直接在 SOAP over JMS URI 上指定 SSL。 有关 SSL 选项的完整描述，请参阅 SSL 和 WebSphere MQ Transport for SOAP
- 使用客户机通道定义表 (CCDT) 中的项来指定 SSL。 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 WebSphere 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。

不能直接在 SOAP over JMS URI 中指定通道定义表。 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 505 页的『WCF 定制通道保证传递』
- 消息格式不正确，无法解释为 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 ，缺省情况下对请求消息设置了以下报告选项，这些选项不可配置:

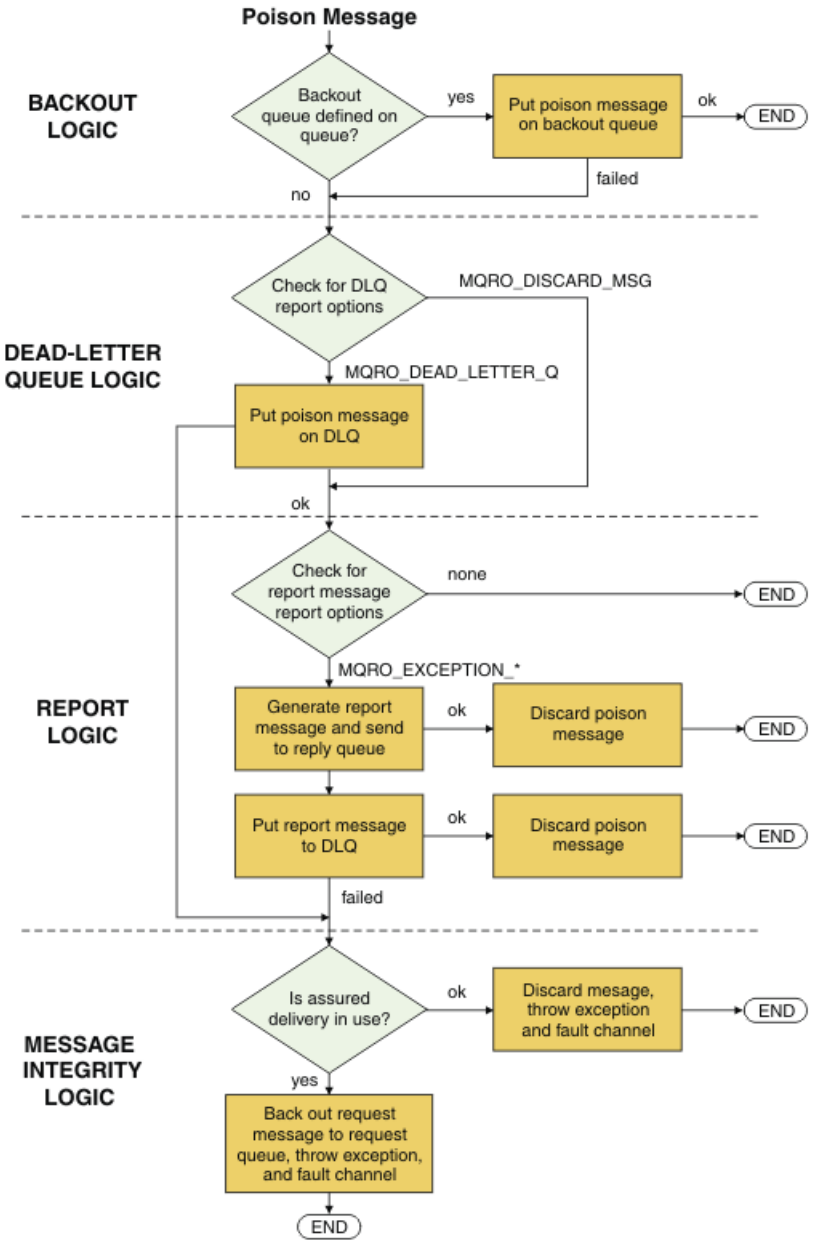
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

对于 SOAP over JMS ，缺省情况下在应答消息上设置了以下报告选项，并且不可配置:

- 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:



WCF 连接选项

有三种方式可以将 WCF 的 WebSphere 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 483 页的『连接差异』

有关 WCF 结构体系的更多信息，请参阅第 503 页的『WCF 体系结构』

非受管客户机连接

在此方式下建立的连接作为 WebSphere MQ 客户机连接到在本地机器或远程机器上运行的 WebSphere MQ 服务器。

要将 WCF 的 WebSphere MQ 定制通道用作 WebSphere MQ 客户机，可以将其与 WebSphere MQ MQI 客户机一起安装在 WebSphere MQ 服务器上或单独的机器上。

非受管服务器连接

在服务器绑定方式下使用时，用于 WCF 的 WebSphere MQ 定制通道使用队列管理器 API，而不是通过网络进行通信。 与使用网络连接相比，使用绑定连接为 WebSphere MQ 应用程序提供更好的性能。

要使用绑定连接，必须在 WebSphere MQ 服务器上安装 WCF 的 WebSphere MQ 定制通道。

受管客户机连接

在此方式下建立的连接作为 WebSphere MQ 客户机连接到在本地机器或远程机器上运行的 WebSphere MQ 服务器。

以此方式连接的 WebSphere MQ 定制 .NET 通道类 3 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 483 页的『受管客户机连接』。

为 WCF 创建和配置 WebSphere MQ 定制通道

用于 WCF 的 WebSphere MQ V7 定制通道的工作方式与 Microsoft 提供的传输 WCF 通道的工作方式相同。 可以通过两种方法之一来创建 WCF 的 WebSphere MQ 定制通道。

关于此任务

WebSphere MQ 定制通道作为 WCF 传输通道与 WCF 集成，因此必须与消息编码器和可选协议通道配对，以便可以创建可供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 形式。

可通过两种不同方式之一创建这些定义：

- 以管理方式创建定义，方法是在应用程序配置文件中提供详细信息 (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 WebSphere MQ 定制通道是传输级别 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。

要配置并使用 WCF 的 WebSphere MQ 定制通道 (这是传输级别 WCF 通道)，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 511 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* 是您应用程序的名称。 通过以下方式使用名为 `SvcConfigEditor.exe` 的 Microsoft 服务配置编辑器工具来最轻松地修改应用程序配置文件:

- 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 *73:* 新建绑定元素字段	
字段	**值**
名称	`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 *74:* 新建绑定配置字段	
字段	**值**
名称	`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 510 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 513 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 510 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 *75:* 新建客户机端点字段	
字段	**值**
名称	`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 512 页的『适用于 WCF 端点 URI 地址格式的 WebSphere MQ 定制通道』
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`

表 *75:* 新建客户机端点字段 *(继续)*	
字段	**值**
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 *76:* 新建服务端点字段	
字段	**值**
名称	`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 512 页的『适用于 WCF 端点 URI 地址格式的 WebSphere 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 WebSphere MQ 定制通道是传输级别 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。

要配置并使用 WCF 的 WebSphere MQ 定制通道 (这是传输级别 WCF 通道)，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 518 页的『使用 WCF 样本』

可通过两种方式创建这些定义：

- 在管理上，通过在应用程序配置文件中提供详细信息，如下所述: 第 509 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』
- 直接从应用程序代码编程，如以下示例中所述。

步骤 1: 创建通道的传输绑定元素的实例

将以下代码添加到应用程序：

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

步骤 2: 设置绑定属性

设置任何必需的绑定属性，例如，通过向应用程序添加以下代码来设置 `ClientConnectionMode`。

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

步骤 3: 创建用于将传输通道与消息编码器配对的定制绑定

通过向应用程序添加以下代码来创建定制绑定:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

步骤 4: 创建 SOAP/JMS URI

必须提供描述访问服务所需的 WebSphere MQ 连接详细信息的 SOAP/JMS URI 作为端点地址。 这取决于通道是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 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");
```

对于服务应用程序，必须创建 SOAP/JMS URI 作为 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 512 页的『适用于 WCF 端点 URI 地址格式的 WebSphere MQ 定制通道』

适用于 *WCF* 端点 *URI* 地址格式的 *WebSphere MQ* 定制通道

统一资源标识 (URI) 提供用于指定 Web Service 的位置和连接详细信息。 此 URI 格式允许在访问目标服务时对特定于 SOAP/ WebSphere MQ 的参数和选项进行全面控制。

使用统一资源标识 (URI) 指定 Web Service。 此部分指定 WebSphere MQ Transport for SOAP 中支持的 URI 格式。 此 URI 格式允许在访问目标服务时对特定于 SOAP/WebSphere MQ 的参数和选项进行全面控制。 此格式与 WebSphere Application Server (WAS) 和 CICS 兼容，有助于将 WebSphere MQ 与这两个产品进行集成。

URI 语法如下:

```
jms:/queue?name=value&name=value...
```

where name is a parameter name and 值 is an appropriate value, and the name=值 element can be repeated any number of times with the second and subsequent occurrences being preceded by an ampersand (&).

有关设置 URI 属性的更多信息，请参阅: 用于 Web Service 部署的 URI 语法和参数

参数名称区分大小写， WebSphere MQ 对象的名称也区分大小写。 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。 如果包含任何其他无法识别的参数，那么将忽略这些参数。

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 .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` 包含许多选项:

```
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，还指定了 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
```

WCF 绑定配置选项

本主题描述如何将配置选项应用于定制通道绑定信息，并列出可用的选项。

可通过以下两种不同的方式之一设置绑定配置属性：

1. 以管理方式: 必须在应用程序配置文件的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

还可以在应用程序配置文件中指定绑定属性设置，例如: app.config。 配置文件由 **svcutil** 生成，例如:

```
<customBinding>

...

     <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>

...

</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

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。

属性名	客户机或服务应用程序	管理值	编程值	描述
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 505 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，说明如何创建和配置 WCF 服务。

WCF 的 IBM WebSphere 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 WebSphere 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 509 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 WebSphere MQ 7 安装所在的目录。
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 511 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

有关公开配置为使用 WCF 的 WebSphere MQ 定制通道的服务的元数据的指示信息。

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。

可以为 WCF 服务创建客户机应用程序; 通常通过使用 Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) 来生成客户机应用程序，以创建可由应用程序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
有关使用 Microsoft svcutil.exe 工具为配置为使用 WCF 的 WebSphere MQ 定制通道的服务生成客户机的指示信息。

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 WebSphere MQ 定制通道端点引用之外， WCF 服务还必须使用 HTTP 端口公开其元数据，以直接从正在运行的服务生成客户机。
- 必须在 svcutil 的配置数据中注册定制通道。

关于此任务
以下步骤说明如何为配置为使用 WebSphere MQ 定制通道但在运行时通过单独的 HTTP 端口公开其元数据的服务生成客户机：

1. 启动 WCF 服务（在启动 svcutil 工具之前，必须先运行该服务）。
2. 将 `svcutil.exe` 配置文件中来自安装根目录的详细信息添加到活动 svcutil 配置文件 (通常为 `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config`)，以便 svcutil 识别 WebSphere 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 516 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 WebSphere 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
```

其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。

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 509 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 WebSphere MQ 7 安装所在的目录。
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 511 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* 是在步骤 第 518 页的『1』 中配置并传入的绑定和端点地址：

```
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) 样本提供了有关如何使用 WebSphere MQ 定制通道的一些简单示例。

要构建样本项目，需要 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 WebSphere 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。 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 516 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。 如果您必须更改资源名称，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上的相应值，并在 *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上的相应值，其中 *MQ_INSTALLATION_PATH* 是 IBM WebSphere MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 WebSphere MQ 产品文档中的 *WebSphere MQ Transport for SOAP* 。 如果需要修改样本解决方案和源，那么需要 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 WebSphere 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 WebSphere 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

此样本演示了使用请求/应答通道形状从 WCF 客户机启动 Windows 通信基础 (WCF) 服务的 WebSphere 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。 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 516 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，那么还需要更改 *MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\client\app.config 文件中的客户机应用程序以及 *MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs 文件中的服务应用程序上的相应值，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 WebSphere MQ 产品文档中的 *WebSphere MQ Transport for SOAP* 。 如果需要修改样本解决方案和源，那么需要 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，以便侦听器正在等待消息: 运行 *MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe 文件，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。
5. 运行客户机一次: 运行 *MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe 文件，其中 *MQ_INSTALLATION_PATH* 是 WebSphere 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 客户机到由 WebSphere MQ 样本托管的 .NET 服务

为 .NET 和 Java 提供了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 WebSphere 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。 有关安装和配置环境的信息，请参阅：第 799 页的『安装 WebSphere MQ Web Transport for SOAP』

当 .NET SOAP over JMS 服务托管环境在 WebSphere 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 WebSphere MQ 安装目录，例如: `C:\Program Files\IBM\WebSphere MQ`
2. 确保 Java 编译器 `javac` 可用并且在 PATH 上。
3. 将文件 `axis.jar` 从 WebSphere 安装 CD 的 `prereqs/axis` 目录复制到 WebSphere MQ 生产目录，例如: `C:\Program Files\IBM\WebSphere MQ\java\lib\soap`
4. 添加到 PATH: *MQ_INSTALLATION_PATH*`\Java\lib`，其中 *MQ_INSTALLATION_PATH* 表示 WebSphere MQ 的安装目录，例如: `C:\Program Files\IBM\WebSphere 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 WebSphere MQ 的安装目录，例如: `C:\Program Files\IBM\WebSphere 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 WebSphere 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 WebSphere 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

此样本演示了从 WCF 客户机到使用单向通道形状的 WebSphere MQ 中提供的 .NET SOAP over JMS 样本服务的连接。 该服务实现了一个简单的 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 516 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，那么还必须更改 *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQNET\default\client\app.config` 文件中的客户机应用程序以及 *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` 文件中的服务应用程序上的相应值，其中 *MQ_INSTALLATION_PATH* 表示 WebSphere MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 WebSphere MQ 产品文档中的 *WebSphere MQ Transport for SOAP* 。

过程

运行客户机一次: 运行 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe 文件，其中 *MQ_INSTALLATION_PATH* 表示 WebSphere 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 WebSphere MQ 样本托管的 Axis Java 服务

为 Java 和 .NET 提供了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 WebSphere 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。 有关安装和配置环境的信息，请参阅：第 799 页的『安装 WebSphere MQ Web Transport for SOAP』

当 .NET SOAP over JMS 服务托管环境在 WebSphere 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 WebSphere MQ 安装目录，例如: C:\Program Files\IBM\WebSphere MQ
2. 确保 Java 编译器 javac 可用并且在 PATH 上。
3. 将文件 axis.jar 从 WebSphere 安装 CD 的 prereqs/axis 目录复制到 WebSphere MQ 安装目录。
4. 添加到 PATH: *MQ_INSTALLATION_PATH*\Java\lib，其中 *MQ_INSTALLATION_PATH* 表示 WebSphere MQ 的安装目录，例如: C:\Program Files\IBM\WebSphere 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 WebSphere MQ 的安装目录，例如: C:\Program Files\IBM\WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ 中提供的 Axis Java SOAP over JMS 样本服务的连接。 该服务实现了一个简单的 StockQuote 示例，用于将文本字符串输出到保存在当前目录的文件中。

已通过使用 WSDL 生成客户机文件来生成了客户机，如第 516 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，那么还必须更改 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQAxis\default\client\app.config 文件中的客户机应用程序以及 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl 文件中的服务应用程序上的相应值，其中 *MQ_INSTALLATION_PATH* 表示 WebSphere MQ 的安装目录。

过程

运行客户机一次: 运行 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe 文件，其中 *MQ_INSTALLATION_PATH* 表示 WebSphere 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。

相关参考

第 529 页的『处理不同的 SOAP 响应元素名称』

缺省情况下，WCF 期望返回值的名称采用特定格式，但服务可能不会返回名称为期望格式的元素。

WCF 客户机到由 WebSphere Application Server 样本托管的 Java 服务

为 WebSphere Application Server (WAS) 6 提供了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

此样本要求使用以下 WebSphere MQ 配置:

表 *77: WebSphere MQ* 必需配置

Object	**必需名称**
队列管理器	QM1
本地队列	HelloWorld
本地队列	HelloWorldReply

此样本还要求正确安装和配置 WebSphere Application Server V6 托管环境。 缺省情况下， WebSphere Application Server V6 使用绑定方式连接来连接到 WebSphere MQ 。 因此， WebSphere Application Server V6 必须与队列管理器安装在同一机器上。

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 WebSphere 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. 输入以下命令以运行样本:
   MQ_INSTALLATION_PATH`\tools\wcf\samples\WAS\TestClient.exe`，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。

关于此任务

该样本演示了使用请求/响应通道形状从 WCF 客户机到 WebSphere MQ V7 中包含的 WCF 样本中提供的 WebSphere Application Server SOAP over JMS 样本服务的连接。 消息使用 WebSphere MQ 队列在 WCF 和 WebSphere Application Server 之间流动。 该服务实现 `HelloWorld(...)` 方法，该方法采用字符串并向客户机返回问候语。

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 516 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` 文件中的客户机应用程序以及 *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` 中的服务应用程序上的相应值，其中 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。 有关格式化 JMS 端点 URI 的更多信息，请参阅 Web Service 部署的 URI 语法和参数。

服务和客户机基于 IBM Developer 文章 使用 *SOAP over JMS* 和 *WebSphere Studio* 构建 JMS Web Service 中概述的服务和客户机。 如果要了解有关开发与 WebSphere MQ WCF 定制通道兼容的 SOAP over JMS Web Service 的更多信息，可以在以下位置找到相关文章: https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html。

过程

运行客户机一次: 运行 *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe` 文件，其中 *MQ_INSTALLATION_PATH* 是 WebSphere 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

服务应用程序会从样本队列获取消息，并向 `HelloWorld(...)` 方法调用提供客户机应用程序输出到控制台的响应。

WebSphere MQ 的 WCF 定制通道上的问题确定

您可以使用 WebSphere MQ 跟踪来收集有关 WebSphere MQ 代码的各个部分的详细信息。 使用 Windows Communication Foundation (WCF) 时，将为与 Microsoft WCF 基础结构跟踪集成的 WCF 定制通道跟踪生成单独的跟踪输出。

为 WCF 定制通道完全启用跟踪会生成两个输出文件:

1. 与 Microsoft WCF 基础结构跟踪集成的 WCF 定制通道跟踪。
2. 与 XMS .NET 集成的 WCF 定制通道跟踪。

通过具有两个跟踪输出，可以使用适当的工具跟踪每个接口的问题，例如:

- 使用合适的 Microsoft 工具确定 WCF 问题。
- 使用 XMS 跟踪格式的 WebSphere MQ MQI 客户机问题。

为了简化跟踪启用， .NET 3 TraceSource 和 XMS .NET 跟踪堆栈都使用单一接口进行控制，如 第 524 页的『WCF 跟踪配置和跟踪文件名』中所述。

WCF 定制通道异常层次结构

定制通道抛出的异常类型与 WCF 一致，且通常是 TimeoutException 或 CommunicationException（或 CommunicationException 的子类）。

使用链接的或内部的异常提供错误条件的更多详细信息（如果可用）。 以下异常是典型示例，通道体系结构中的每个层都提供了额外的链接异常，例如: CommunicationsException 具有链接的 XMSException ， 该异常具有链接的 MQException:

1. System.serviceModel.CommunicationsExceptions
2. IBM.XMS.XMSException
3. IBM.WMQ.MQException

在层次结构中最高的 CommunicationException 数据集合中捕获并提供关键信息。 这种数据捕获和供应防止应用程序需要链接到通道体系结构中的每个层，以便询问链接异常以及这些异常可能包含的任何其他信息。 定义了以下键名称:

- IBM。XMS.WCF.ErrorCode: 当前定制通道异常的错误消息代码。
- IBM。XMS.ErrorCode: 堆栈中第一个 XMS 异常的错误消息。
- IBM。WMQ.ReasonCode: 底层 WebSphere MQ 原因码。
- IBM。WMQ.CompletionCode: 底层 WebSphere MQ 完成代码。

WCF 跟踪配置和跟踪文件名

在完全启用跟踪时，它会生成两个输出文件，一个用于诊断 WCF 问题，另一个详细文件用于内部跟踪诊断材料。 为了简化跟踪启用， .NET 3 TraceSource 和 XMS .NET 跟踪堆栈都使用单个接口。

有两种不同的跟踪方法可用于 WCF 定制通道，这两种跟踪方法独立或一起激活。 每种方法都会生成各自的跟踪文件，所以当两种跟踪方法都激活时，会生成两个输出文件。

为了使配置和启用过程尽可能简单，请使用同一接口控制这两种跟踪方法。 必须编辑 `app.config` 文件，以包含相关跟踪配置，如以下部分中所述。 用户随后还可以添加各自的等效部分，以将输出与自有应用程序的跟踪结合起来。

缺省情况下，不会启用 WCF 定制通道跟踪。 您必须先创建跟踪侦听器，然后在 `app.config` 文件中为选中的跟踪源设置所需的跟踪级别。

使用 WCF 基础结构跟踪配置 WCF 定制通道

将以下代码部分添加到 `app.config` 文件中的 `<system.diagnostics><sources>` 部分:

```
<source name="IBM.XMS.WCF" switchValue="Verbose,ActivityTracing">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

前面的代码段使用 .NET 3 TraceSource 进行通道跟踪。 与可执行文件关联的配置文件的所有调用都由这部分代码控制。

使用 XMS .NET 跟踪配置 WCF 定制通道

配置 XMS .NET 跟踪需要将一部分代码添加到 `app.config` 文件中的 `<system.diagnostics><sources>` 部分。 但是，也会将这部分代码添加到使用 WCF 基础结构跟踪配置 WCF 定制通道部分中所显示的可扩展 `<source>` 元素。 因此，虽然 WCF 基础结构跟踪代码必须存在才

能使 XMS .NET 跟踪工作，但如果不需要 WCF 基础结构跟踪，那么可以禁用 WCF 基础结构跟踪，如 启用 WCF 跟踪 部分中所述。

```
<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing"
 xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path"
 xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

WCF 跟踪配置变量

表 *78: WCF* 跟踪配置变量

变量	**描述**
名	将名称指定为：`IBM.XMS.WCF`
switchValue	switchValue 用于控制跟踪级别。 在将 switchValue 设置为 `Off` 时，不会生成 WCF 基础结构 TraceSource。 任何其他值（例如 `Verbose`）都会生成 TraceSource。 有关 Microsoft 的详细跟踪级别信息，请参阅 WCF 文档或转至 Microsoft WCF 跟踪 Web 页面: https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx

xmsTrace 规范 =*ComponentName*=*type*=*state*	*ComponentName* 是想要跟踪的类的名称。 您可以在该名称中使用 * 通配符。 例如：
	`*=all=enabled`
	指定要跟踪所有类，以及
	`IBM.XMS.impl.*=all=enabled`
	指定只需要 API 跟踪。
	type 可以是以下任何跟踪类型：
	• all
	• debug
	• 事件
	• EntryExit
	state 可以启用或禁用。

xmsTraceFilePath="*filename*"	如果没有指定 xmsTraceFilePath，或提供了 xmsTraceFilePath（但包含空字符串），那么跟踪文件将放置在当前目录中。 要将跟踪文件存储在已命名的目录中，请在 xmsTraceFilePath 中指定目录名称，例如： `xmsTraceFilePath="c:\somepath"`
xmsTraceFileSize="*size*"	允许的跟踪文件的最大大小。 当文件达到此大小时，会将其归档并重命名。 缺省最大值为 20 KB，指定为： `xmsTraceFileSize="20000000".`
xmsTraceFileNumber="*number*"	要保留的跟踪文件的数量。 缺省值为 4（1 个活动文件和 3 个归档文件）。 允许的最小数量为 2。

表 78: WCF 跟踪配置变量 (继续)	
变量	**描述**

xmsTraceFormat="*format*"	msTraceFormat 有两个级别：basic 和 advanced。如果未指定 xmsTraceFormat，或提供了 xmsTraceFormat（但包含空字符串），那么缺省跟踪格式为 basic。 如果指定了以下内容，那么会以该格式生成跟踪文件。
	`xmsTraceFormat="basic"`
	如果要求跟踪与跟踪分析器工具兼容，那么必须指定：
	`traceFormat="advanced"`

启用 WCF 跟踪

可以使用四种组合来启用和禁用 2 种不同的跟踪方法。 这四种组合需要编辑前面部分中所述的编码部分的值。

还有一个可以设置的环境变量; 有关更多信息，请参阅 第 527 页的『使用 WCF_TRACE_ON 环境变量启用 WCF 跟踪』。

该表和显示的值取决于是否已将先前展示的代码部分添加到了 app.config 文件中。

表 79: WCF 跟踪启用组合。		
跟踪类型	**更改的值**	**示例**

已启用 XMS 跟踪。 已启用 WCF TraceSource	switchValue 未设置为 Off	`<source name="IBM.XMS.WCF"`
		`switchValue="Verbose, ActivityTracing"`
		`xmsTraceSpecification="*=all=enabled"`
		`xmsTraceFilePath="path"`
		`xmsTraceFileSize="2000000"`
		`xmsTraceFileNumber="4"`
		`xmsTraceFormat="advanced">`
		`<listeners>`
		`<remove name="Default"/>`
		`<add name="NewListener"/>`
		`</listeners>`
		`</source>`

已启用 XMS 跟踪。 已禁用 WCF TraceSource	switchValue 设置为 Off，并给出了 xmsTraceSpecification	`<source name="IBM.XMS.WCF"`
		`switchValue="Off, ActivityTracing"`
		`xmsTraceSpecification="*=all=enabled"`
		`xmsTraceFilePath="path"`
		`xmsTraceFileSize="2000000"`
		`xmsTraceFileNumber="4"`
		`xmsTraceFormat="advanced">`
		`<listeners>`
		`<remove name="Default"/>`
		`<add name="NewListener"/>`
		`</listeners>`
		`</source>`

表 79: *WCF* 跟踪启用组合。 *(继续)*

跟踪类型	更改的值	示例
已禁用 XMS 跟踪。 已启用 WCF TraceSource	有两种方式实现此结果: • switchValue 变量未设置为 Off，并且尚未添加 xmsTraceSpecification • switchValue 变量未设置为 Off，xmsTraceSpecification 已设置为 disabled	<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>
已禁用 XMS 跟踪。 已禁用 WCF TraceSource	有三种方式实现此结果: • app.config 文件中没有 <source> 元素 • switchValue 变量设置为 Off，并且尚未添加 xmsTraceSpecification • switchValue 变量设置为 Off，xmsTraceSpecification 已设置为 disabled	<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>

使用 WCF_TRACE_ON 环境变量启用 WCF 跟踪

除了前面描述的用于启用 WCF 跟踪的方法外，还可以使用 WCF_TRACE_ON 环境变量来启用 XMS .NET 跟踪。

将 WCF_TRACE_ON 环境变量设置为任何非空值等同于将 xmstraceSpecification 设置为 *=all=enabled，例如: "set WCF_TRACE_ON=true"

但是，如果在 app.config 文件中显式设置了 xmstraceSpecification，那么将会覆盖 WCF_TRACE_ON 环境变量。

WCF 跟踪输出文件和文件名

传统上使用基本名称和进程标识格式 xms_trace_*pid*.log 来命名 XMS 跟踪文件，其中 *pid* 是进程标识。

由于 XMS 跟踪文件仍可与 WCF 定制通道跟踪文件并行生成，因此与 XMS .NET 跟踪输出文件集成的 WCF 定制通道跟踪具有以下格式以避免混淆: wcfxms_trace_*pid*.log，其中 *pid* 是进程标识。

缺省情况下，会在当前工作目录中创建跟踪输出文件，但如果需要，也可以重新定义该目标。

WCF XMS 首次故障支持技术 (FFST)

您可以使用 WebSphere MQ 跟踪来收集有关 WebSphere MQ 代码的各个部分的详细信息。 XMS FFST 具有自己的 WCF 定制通道的配置和输出文件。

XMS FFST 跟踪文件通常使用基本名称和进程标识格式 xmsffdc*pid_date*.txt 进行命名，其中 *pid* 是进程标识， *date* 是时间和日期。

由于 XMS FFST 跟踪文件仍可与 WCF 定制通道 XMS FFST 文件并行生成，因此 WCF 定制通道 XMS FFST 输出文件具有以下格式以避免混淆: wcfffdc*pid_date*.txt，其中 *pid* 是进程标识， *date* 是时间和日期。

缺省情况下，会在当前工作目录中创建此跟踪输出文件，但如果需要，也可以重新定义该目标。

具有 XMS .NET 跟踪头的 WCF 定制通道类似于以下示例:

```
   ************* Start Display XMS WCF Environment *************
     Product Name  :- value
     WCF Version   :- value
     Level    :- value
   *************  End Display XMS WCF Environment  *************
```

FFST 跟踪文件以标准方式进行格式化，而没有任何特定于定制通道的格式化。

WCF 版本信息

WCF 版本信息有助于确定问题，并且包含在定制通道的组合件元数据中。

可以通过以下三种方法之一检索 WCF 版本元数据的 WebSphere MQ 定制通道:

- 使用 WebSphere MQ 实用程序 dspmqver。 有关如何使用 dspmqver 的信息，请参阅：dspmqver
- 使用 Windows 资源管理器属性对话框: 在 Windows 资源管理器中，右键单击 **IBM.XMS.WCF.dll** > **属性** > **V**。
- 从任何通道 FFST 或跟踪文件的头信息。 有关 FFST 头信息的更多信息，请参阅: 第 527 页的『WCF XMS 首次故障支持技术 (FFST)』

WCF 提示和技巧

以下提示和技巧并非按重要性排序，并且可能会在发布新版本的文档时添加。 如果这些主题与您正在做的工作相关，那么可能会节省您的时间。

外部化 *WCF* 服务主机的异常

对于使用 WCF 服务主机托管的服务，缺省情况下，不会外部化服务、WCF 内部或通道堆栈抛出的任何未处理的异常。 要获悉这些异常，必须注册错误处理程序。

以下代码提供了定义错误处理程序服务行为的示例，该行为可作为服务的属性加以应用：

```
using System.ServiceModel.Dispatcher;
using System.Collections.ObjectModel;
....
    public class ErrorHandlerBehaviorAttribute : Attribute, IServiceBehavior, IErrorHandler
    {
        //
        // IServiceBehavior Interface
        //
        public void AddBindingParameters(ServiceDescription serviceDescription,
           ServiceHostBase serviceHostBase, Collection<ServiceEndpoint> endpoints,
           BindingParameterCollection bindingParameters)
        {
        }
        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
            ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher channelDispatcher in serviceHostBase.ChannelDispatchers)
{
                channelDispatcher.ErrorHandlers.Add(this);
            }
        }
        public void Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
        {
        }

        //
        // IErrorHandler Interface
        //
        public bool HandleError(Exception e)
        {
            // Process the exception in the required way, in this case just outputting to the
console
            Console.Out.WriteLine(e);

            // Always return false to allow any other error handlers to run
            return false;
        }
        public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
        {
```

```
        }
    }
```

处理不同的 *SOAP* 响应元素名称

缺省情况下，WCF 期望返回值的名称采用特定格式，但服务可能不会返回名称为期望格式的元素。

WCF 的约定是期望返回的值以以下格式命名: *methodName*Result，其中 *methodName* 是服务操作的名称。 例如，对于名为 getQuote 的服务， WCF 期望调用响应: getQuoteResult。

但是，该服务可能会返回名称不符合该格式的元素。

在运行 scvutil 工具生成代理客户机时，如果 WSDL 指定了不同的名称，那么代理接口会添加参数来告知 WCF 要查找的名称。 例如:

```
[System.ServiceModel.OperationContractAttribute(Action = "", ReplyAction = "*")]
[System.ServiceModel.XmlSerializerFormatAttribute(Style =
System.ServiceModel.OperationFormatStyle.Rpc,
                                                  Use =
System.ServiceModel.OperationFormatUse.Encoded)]
[return: System.ServiceModel.MessageParameterAttribute(Name = "getQuoteReturn")]
float getQuote(string in0);
```

如果您创建了自己的接口（例如，通过将请求/应答方法添加到现有的代理接口），那么如果服务返回不同的名称，您就必须确保将相同的参数添加到该接口。 如果不执行此操作，那么最常见的问题是，对 service 方法的调用始终会返回一个 Null 值，如果返回一个对象，那么该方法会返回 Null，但如果返回一个数字值（如整数），那么该方法返回 0。

使用 C++

WebSphere MQ 提供相当于 WebSphere MQ 对象的 C++ 类以及相当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。

从 WebSphere MQ V 7.0 开始， WebSphere MQ 编程接口的增强功能将不会应用于 C++ 类。

WebSphere MQ C++ 提供以下功能:

- 自动初始化 WebSphere 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 网桥头传输和接收。
- 工作头传输和接收。
- 客户机通道定义。

以下 Booch 类图显示了所有类与过程 MQI (例如，使用 C) 中具有句柄或数据结构的 WebSphere MQ 实体大致平行。 所有类都继承自 ImqError 类 (请参阅 ImqError C++ 类)，这允许将错误条件与每个对象相关联。

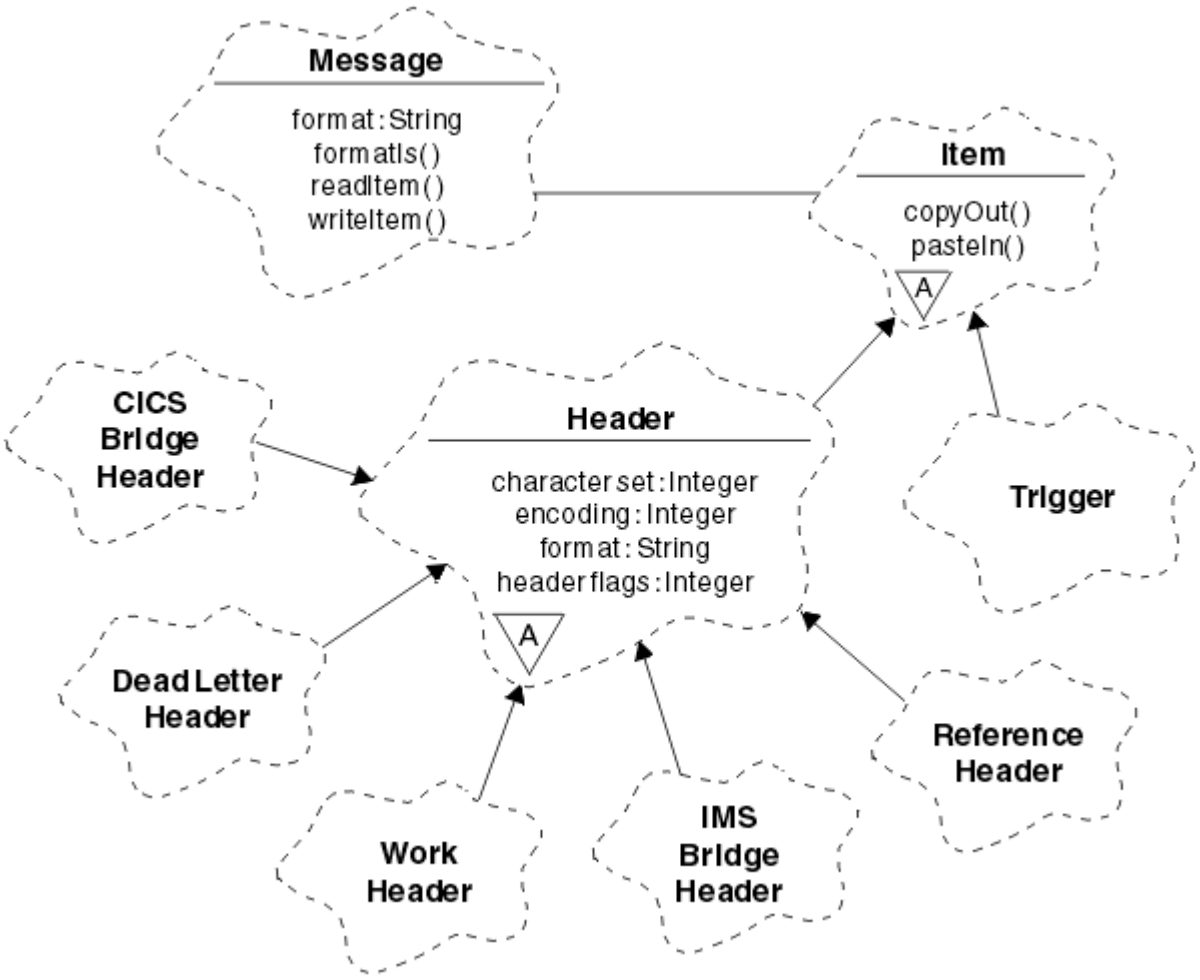


图 *120: WebSphere MQ C++* 类 *(*项处理*)*

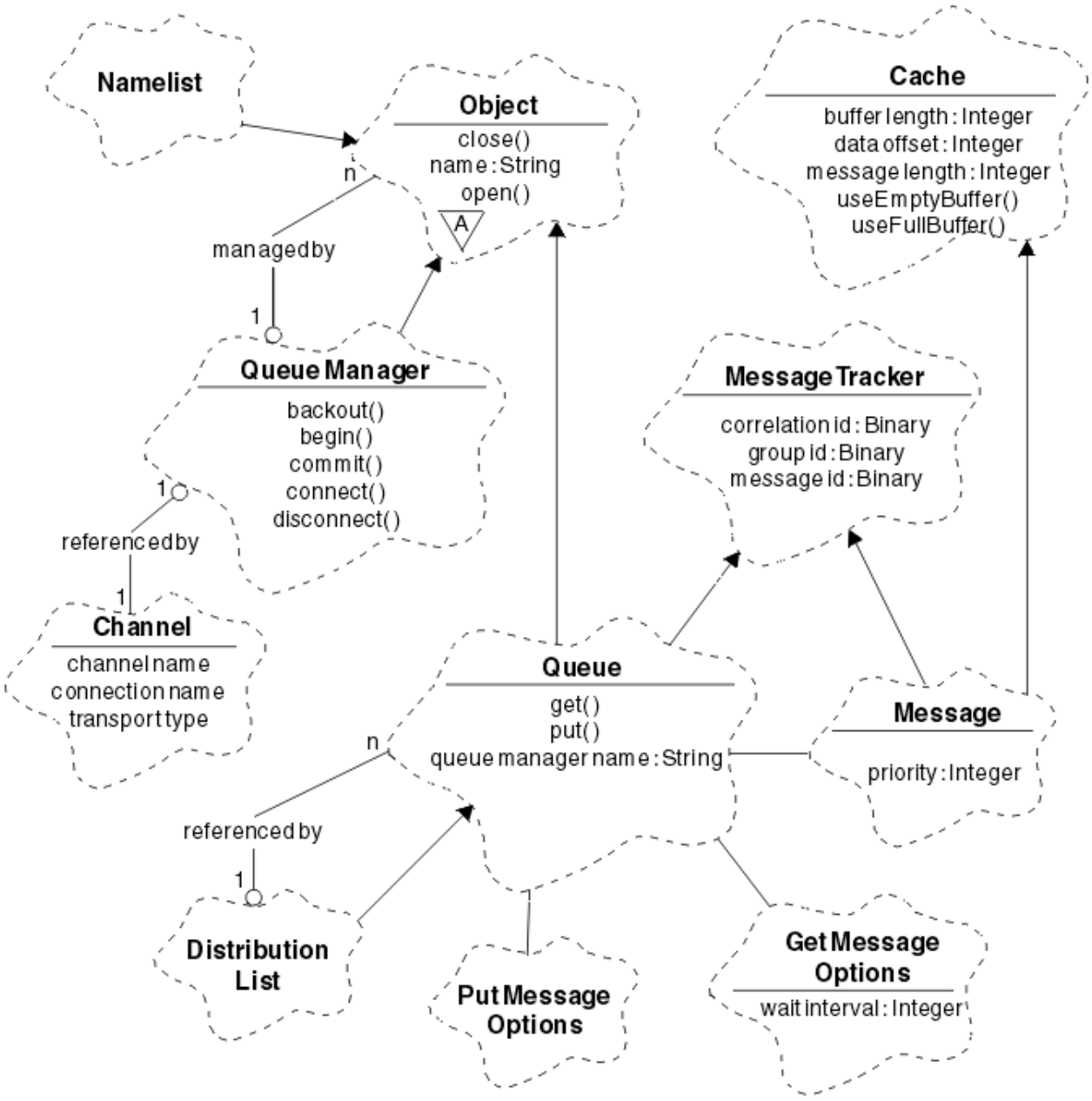


图 *121: WebSphere MQ C++* 类 *(*队列管理*)*

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。
- 云中的小三角形表示 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 531 页的图 121）和项处理类（请参阅第 530 页的图 120）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) ， 用于封装字节数组，例如 MQBYTE24。
- ImqBoolean 数据类型，定义为 **typedef unsigned char ImqBoolean**。
- ImqString 类 (请参阅 ImqString C++ 类) ， 它封装了诸如 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 类层次结构下 (请参阅 ImqObject C++ 类) ，并提供 MQI 的封装接口。 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) 封装 MQMD 数据结构，并通过提供高速缓存的缓冲区工具充当用户数据和 项 的保存点 (请参阅 第 540 页的『使用 C++ 读取消息』)。 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 532 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 536 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 539 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 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 7 页的『开发应用程序』
IBM WebSphere MQ 提供了多种方法，您可以通过这些方法开发应用程序来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。

相关参考

第 545 页的『构建 WebSphere MQ C++ 程序』
列出了受支持编译器的 URL 以及用于在 WebSphere 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

WebSphere 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 532 页的表 80 所示的目录中。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

表 *80:* 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

表 *80:* 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
HP-UX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ah（请参阅注释 第 533 页的『2』）
Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/as
Linux	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\samples	*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn（请参阅注释 第 533 页的『3』）

注意:

1. 使用面向 IBM i 的 ILE C++ 编译器构建的程序位于库 QMQM 中。 包含文件位于 /QIBM/ProdData/mqm/inc 中。
2. 使用 HP ANSI C++ 编译器构建的程序位于目录 *MQ_INSTALLATION_PATH*/samp/bin/ah 中。 有关更多信息，请参阅第 546 页的『在 HP-UX 上构建 C++ 程序』。
3. 使用 Microsoft Visual Studio 构建的程序可在 *MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn 中找到。 有关这些编译器的更多信息，请参阅第 551 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 以外的所有平台上

如果您正在使用与 WebSphere MQ 的服务器连接，请遵循下列其中一个过程:

- 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。

如果您正在使用与 WebSphere MQ 的客户机连接，请遵循下列其中一个过程:

- 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)

样本代码

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // WebSphere 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 以外的所有平台上

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。

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）。 此样本在 z/OS 上不受支持。

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，以帮助您以 C++ 语言编写 WebSphere MQ 应用程序。

下表中汇总了这些头文件。

表 *81: 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，WebSphere MQ 施加了某些限制，这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，但有一些 **ImqString** 类的特殊方法是可取的; **operator length**() 相当于 **strlen** 和 **storage**() 指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**::**获取**(**ImqMessage &** 消息)

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

对于导致对 MQI 进行任何调用的任何方法，将自动连接 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。

设置字符 (**char ***) 数据的方法始终采用数据的副本，但某些方法可能会截断副本，因为某些限制是由 WebSphere MQ 施加的。

ImqString 类 (请参阅 ImqString C++ 类) 封装了传统的 **char ***，并支持：

- 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) 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：

- ImqCICSBridge 头 (请参阅 ImqCICSBridge 头 C++ 类)
- ImqGetMessageOptions (请参阅 ImqGetMessageOptions C++ 类)
- ImqIMSBridge 头 (请参阅 ImqIMSBridge 头 C++ 类)
- ImqMessageTracker (请参阅 ImqMessageTracker C++ 类)
- ImqQueueManager (请参阅 ImqQueueManager C++ 类)
- ImqReference 头 (请参阅 ImqReference 头 C++ 类)
- ImqWork 头 (请参阅 ImqWork 头 C++ 类)

ImqBinary 类同样也支持比较和复制。

C++ 中不支持的函数

WebSphere MQ C++ 类和方法独立于 WebSphere MQ 平台。 可能会因此提供一些在某些平台上不支持的函数。

如果尝试在不受支持的平台上使用函数，那么 WebSphere MQ 会检测到该函数，但 C++ 语言绑定不会检测到该函数。 WebSphere MQ 向程序报告该错误，与任何其他 MQI 错误一样。

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。

发送消息时，将首先在由 ImqCache 对象管理的缓冲区中准备消息数据 (请参阅 ImqCache C++ 类)。 缓冲区 (通过继承) 与每个 ImqMessage 对象 (请参阅 ImqMessage C++ 类) 相关联: 它可以由应用程序 (使用 **useEmptyBuffer** 或 **useFullBuffer** 方法) 提供，也可以由系统自动提供。 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。

每个项都有一个对象类，对应于可识别的 WebSphere MQ 消息格式。 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 派生自 ImqItem 类 (请参阅 ImqItem C++ 类)，并从该类实现虚拟 **copyOut** 和 **pasteIn** 方法。

接下来的两个示例显示 ImqDeadLetterHeader 类中的代码 (请参阅 ImqDeadLetterHeader C++ 类)。 第一个示例显示了定制封装的消息-正在编写 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的消息-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

用于将消息写入 IMS 网桥的示例程序代码。

发送到 WebSphere MQ-IMS 网桥的消息可能使用特殊头。 IMS 网桥头以常规消息数据为前缀。

```
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 网桥

用于将消息写入 CICS 网桥的示例程序代码。

使用 CICS 网桥发送到 WebSphere MQ for z/OS 的消息需要特殊头。 CICS 网桥头以常规消息数据为前缀。

```
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

用于编写以 z/OS 工作负载管理器管理的队列为目标的消息的示例程序代码。

发送到 WebSphere MQ for z/OS 的消息 (发往由 z/OS 工作负载管理器管理的队列) 需要特殊头。 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 WebSphere MQ C++ 程序

列出了受支持编译器的 URL 以及用于在 WebSphere MQ 平台上编译，链接和运行 C++ 程序和样本的命令。

每个受支持的 WebSphere MQ 平台和版本的编译器列示在 WebSphere MQ 系统需求页面 (IBM WebSphere MQ 的系统需求) 中。

编译和链接 WebSphere MQ C++ 程序所需的命令取决于您的安装和需求。 以下示例显示了在许多平台上使用 WebSphere MQ 缺省安装的某些编译器的典型编译和链接命令。

在 AIX 上构建 C++ 程序

在 AIX 上使用 XL C Enterprise Edition 编译器构建 WebSphere MQ C++ 程序。

客户机

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

32 位非线程应用程序

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32 位线程应用程序

```
xlC_r -o imqsputc_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

服务器

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

32 位非线程应用程序

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32 位线程应用程序

```
xlC_r -o imqsput_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

在 HP-UX 上构建 C++ 程序

在 HP-UX 上使用 aC+ + 或 aCC 编译器构建 WebSphere MQ C++ 程序。

在 HP-UX Itanium 上， WebSphere MQ 仅支持标准运行时。 使用 aCC 编译器。

- libimqi23bh.sl 为标准运行时提供 WebSphere MQ C++ 类。
- 为了与早期发行版兼容，提供了从 libimqi23ah.sl 到 libimqi23bh.sl 的符号链接。

IA64 (IPF)

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

客户机：IA64 (IPF)

32 位非线程应用程序

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqic
```

32 位线程应用程序

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqic_r -lpthread
```

64 位非线程应用程序

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqic
```

64 位线程应用程序

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqic_r
-lpthread
```

服务器：IA64 (IPF)

32 位非线程应用程序

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqm
```

32 位线程应用程序

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqm_r -lpthread
```

64 位非线程应用程序

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqm
```

64 位线程应用程序

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqm_r
-lpthread
```

在 Linux 上构建 C++ 程序

使用 GNU g + + 编译器在 Linux 上构建 WebSphere MQ C++ 程序。

System p

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

System z

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

客户机: System z

32 位非线程应用程序

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64  -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

服务器 :System z

32 位非线程应用程序

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

System x (32 位)

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

客户机: System x (32 位)

32 位非线程应用程序

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

服务器: System x (32 位)

32 位非线程应用程序

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATHlib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

在 Solaris 上构建 C++ 程序

在 Solaris 上使用 Sun ONE 编译器构建 WebSphere MQ C++ 程序。

SPARC

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

客户机：SPARC

32 位应用程序

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_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 -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

在 Windows 上构建 C++ 程序

在 Windows 上使用 Microsoft Visual Studio C++ 编译器构建 WebSphere MQ C++ 程序。

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 位应用程序的文件安装在 *MQ_INSTALLATION_PATH*/Tools/Lib64 中。*MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 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
```

使用 WebSphere MQ classes for Java

WebSphere MQ Java 类使您能够在 Java 环境中使用 WebSphere MQ 。 Java 应用程序可以使用 WebSphere MQ classes for Java 或 WebSphere MQ classes for JMS 来访问 WebSphere MQ 资源。

WebSphere MQ classes for Java 允许 Java 应用程序执行以下操作:

- 作为 WebSphere MQ 客户机连接到 WebSphere MQ
- 直接连接到 WebSphere MQ 队列管理器

WebSphere MQ Java 类封装了消息队列接口 (MQI) ，即本机 WebSphere MQ API。

WebSphere MQ Java 类使用与 WebSphere MQ 的 C++ 和 .NET 接口类似的对象模型。

为什么应该使用 WebSphere MQ Java 类?

如果以下几点在安装中很重要，请考虑使用 WebSphere MQ classes for Java:

- WebSphere MQ Java 类封装了消息队列接口 (MQI) ，即本机 WebSphere MQ API。
  - 如果您熟悉在过程语言中使用 MQI ，那么可以将此知识传输到 Java 环境。
  - 您可以利用 WebSphere MQ 的所有功能部件，而不仅仅是通过 JMS 提供的功能部件。
- WebSphere MQ Java 类使用与 WebSphere MQ 的 C++ 和 .NET 接口类似的对象模型。 如果您熟悉这些接口，那么可以将这些知识传输到 Java 环境。

注: WebSphere MQ Java 类不支持自动客户机重新连接。

WebSphere MQ Java 类入门

此主题集合概述了 WebSphere MQ Java 类及其使用情况。

WebSphere MQ Java 类是什么?

WebSphere MQ Java 类允许您在 Java 环境中使用 WebSphere MQ 。

WebSphere MQ classes for Java 允许 Java 应用程序执行以下操作:

- 作为 WebSphere MQ 客户机连接到 WebSphere MQ
- 直接连接到 WebSphere MQ 队列管理器

WebSphere MQ Java 类封装了消息队列接口 (MQI) ，即本机 WebSphere MQ API。

WebSphere MQ Java 类使用与 WebSphere MQ 的 C++ 和 .NET 接口类似的对象模型。

为什么应该使用 WebSphere MQ Java 类?

Java 应用程序可以使用 WebSphere MQ classes for Java 或 WebSphere MQ classes for JMS 来访问 WebSphere MQ 资源。 使用 WebSphere MQ Java 类有许多优点。

如果以下几点在安装中很重要，请考虑使用 Websphere MQ Java 类:

- WebSphere MQ Java 类封装了消息队列接口 (MQI) ，即本机 WebSphere MQ API。

  - 如果您熟悉在过程语言中使用 MQI，那么可以将此知识传输到 Java 环境。
  - 您可以利用 WebSphere MQ 的所有功能部件，而不仅仅是通过 JMS 提供的功能部件。
- WebSphere MQ Java 类使用与 WebSphere MQ 的 C++ 和 .NET 接口类似的对象模型。 如果您熟悉这些接口，那么可以将这些知识传输到 Java 环境。

WebSphere MQ Java 类的连接选项

WebSphere MQ Java 类可以在客户机或绑定方式下进行连接。

可编程选项允许 WebSphere MQ Java 类通过以下任一方式连接到 WebSphere MQ：

- 作为使用传输控制协议/Internet Protocol (TCP/IP) 的 WebSphere MQ MQI 客户机
- 在绑定方式下，使用 Java 本机接口 (JNI) 直接连接到 WebSphere MQ

无法在 z/OS 上运行客户机，但是如果安装了客户机连接设施，那么其他平台上的客户机可以连接到 WebSphere MQ for z/OS 队列管理器。

以下部分更详细地描述了客户机方式和绑定方式连接选项。

客户机连接

要以客户机方式连接到队列管理器， WebSphere MQ classes for Java 应用程序可以在运行队列管理器的同一系统上运行，也可以在其他系统上运行。 在每种情况下， WebSphere MQ Java 类都通过 TCP/IP 连接到队列管理器。

用于 Java 应用程序的 WebSphere MQ 类可以使用客户机方式连接到任何受支持的队列管理器。

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 564 页的『针对 Java 连接方式的 WebSphere MQ 类』。

绑定连接

在绑定方式下使用时， WebSphere MQ classes for Java 使用 Java 本机接口 (JNI) 直接调用现有队列管理器 API，而不是通过网络进行通信。 在大多数环境中，以绑定方式连接通过避免 TCP/IP 通信成本，为 WebSphere MQ classes for Java 应用程序提供比以客户机方式连接更好的性能。

使用 WebSphere MQ classes for Java 以绑定方式连接的应用程序必须在它们所连接的队列管理器所在的系统上运行。

必须将用于运行 WebSphere MQ classes for Java 应用程序的 Java 运行时环境配置为装入 WebSphere MQ classes for Java 库; 请参阅 WebSphere 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 564 页的『针对 Java 连接方式的 WebSphere MQ 类』。

WebSphere MQ Java 类的先决条件

要使用 WebSphere MQ Java 类，您需要某些其他软件产品。

有关 WebSphere MQ classes for Java 的先决条件的最新信息，请参阅 WebSphere MQ 自述文件。

要为 Java 应用程序开发 WebSphere MQ 类，您需要 Java Development Kit (JDK)。 可在位于 IBM WebSphere MQ 的系统需求的 WebSphere MQ 系统需求页面上找到操作系统支持的 JDK 的详细信息。

要针对 Java 应用程序运行 WebSphere MQ 类，您需要以下软件组件:

- WebSphere MQ 队列管理器，用于连接到队列管理器的应用程序
- 针对运行应用程序的每个系统的 Java 运行时环境 (JRE)。 WebSphere MQ 随附了合适的 JRE。

如果需要 SSL 连接以使用经 FIPS 140-$tag1 认证的加密模块，那么需要 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS)。 每个 IBM JDK 和 JRE V 1.4.2 或更高版本都包含 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 WebSphere MQ classes for Java 应用程序 如果 IPv6 为 中使用 Internet Protocol V 6 (IPv6) 地址。

安装和配置 WebSphere MQ classes for Java

本节描述安装 WebSphere MQ classes for Java 时创建的目录和文件，并说明在安装后如何配置 WebSphere MQ classes for Java。

为 WebSphere MQ classes for Java 安装的内容

最新版本的 WebSphere MQ classes for Java 随 WebSphere 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 WebSphere MQ 的更多信息，请参阅:

安装 WebSphere MQ 服务器
安装 IBM WebSphere MQ 客户机

WebSphere MQ Java 类包含在 Java 归档 (JAR) 文件 com.ibm.mq.jar 和 com.ibm.mq.jmqi.jar 中。

对标准消息头 (例如可编程命令格式 (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。

安装和升级 WebSphere MQ Java JAR 文件类

唯一受支持的方法是将 Java JAR 文件的 WebSphere MQ 类安装到系统上，或者安装 WebSphere MQ 产品或 WebSphere MQ MQI 客户机 SupportPac，或者使用诸如 Apache Maven 之类的软件管理工具，有关更多信息，请参阅 第 561 页的『IBM WebSphere MQ classes for Java 和软件管理工具』。

请勿从其他机器移动或复制 Java JAR 文件的 WebSphere MQ 类，除非您正在使用软件管理工具。

- 修订包无法应用于从另一台机器复制了 JAR 文件的 "安装"，这使确保所有 JAR 文件保持一致并处于兼容级别变得更加困难。
- 在机器之间复制 WebSphere MQ JMS JAR 文件类还会导致同一机器上存在多个文件副本，这可能会导致处理代码和调试问题时出现问题。

请勿在应用程序归档中包含 Java JAR 文件的 WebSphere MQ 类。

- 无法使用 WebSphere MQ 修订包应用对 WebSphere MQ classes for Java 的更新。
- IBM 支持人员无法轻松确定应用程序正在使用的 WebSphere MQ Java 类的版本。
- 如果在同一 Java 运行时环境中运行的多个应用程序包含不同版本的 WebSphere MQ classes for Java，那么可能会出现问题，因为多个版本的 WebSphere MQ classes for Java 会同时装入到 Java 运行时环境中。
- 如果应用程序使用 BINDINGS 传输来连接到队列管理器，那么对队列管理器的任何主要升级还需要更新该应用程序以包含相应级别的 WebSphere MQ Java 类。

  例如，如果队列管理器升级到 WebSphere MQ Version 7.1 级别，那么使用 BINDINGS 传输连接到队列管理器的任何应用程序也需要更新以包含 WebSphere MQ Version 7.1 classes for Java。

以下 Java 库随 WebSphere MQ Java 类一起分发:

- connector.jar (V 1.0)

名为 Postcard 的样本应用程序位于 JAR 文件 com.ibm.mq.postcard.jar 中。

Javadoc 工具已用于生成 HTML 页面，其中包含 WebSphere MQ classes for Java 和 WebSphere MQ classes for JMS API 的规范。 HTML 页面位于 WebSphere MQ classes for JMS 安装目录的 doc 子目录中。在 UNIX， Linux 和 Windows 系统上， doc 子目录包含各个 HTML 页面的文件中。

安装完成后，将在 第 554 页的『 WebSphere MQ Java 类的安装目录』中显示的位置安装文件和样本。

安装后，必须在除 Windows 以外的任何平台上更新环境变量，如 第 555 页的『与 WebSphere MQ Java 类相关的环境变量』中所述。

WebSphere MQ Java 类的安装目录

WebSphere MQ Java 文件类根据平台安装在不同的位置。

第 555 页的表 82 显示了用于 Java 文件的 WebSphere MQ 类的安装位置。

表 *82: WebSphere MQ Java* 安装目录类	
平台	**目录**
AIX	*MQ_INSTALLATION_PATH*/java/lib
HP-UX， Linux 和 Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。	

WebSphere MQ 随附了一些样本应用程序，例如安装验证程序 (IVP)。 第 555 页的表 83 显示样本应用程序的安装位置。 WebSphere 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 *83:* 样本目录	
平台	**目录**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
HP-UX， Linux 和 Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。	

与 *WebSphere MQ Java* 类相关的环境变量

如果要运行 WebSphere MQ classes for Java 应用程序，那么其类路径必须包含 WebSphere MQ classes for Java 和样本目录。

要使 WebSphere MQ classes for Java 应用程序运行，其类路径必须包含相应的 WebSphere 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。 此信息可以在 Java 调用命令或 CLASSPATH 环境变量中提供。

第 555 页的表 84 显示了要在每个平台上使用的相应 CLASSPATH 设置，以针对 Java 应用程序 (包括样本应用程序) 运行 WebSphere MQ 类。

表 *84:* 用于针对 *Java* 应用程序运行 *WebSphere MQ* 类的 *CLASSPATH* 设置，包括针对 *Java* 样本应用程序的 *WebSphere MQ* 类	
平台	**CLASSPATH 设置**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
HP-UX， Linux 和 Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 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。 仅当已安装 IBM WebSphere MQ Advanced Message Security 时，此文件才存在。

WebSphere MQ classes for Java 随附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 WebSphere MQ classes for Java 的目录，如 WebSphere MQ classes for Java 安装目录中所示。

MQ_JAVA_LIB_PATH
此环境变量指定存储 WebSphere MQ classes for Java 库的目录，如 每个平台的 WebSphere MQ classes for Java 库的位置中所示。 WebSphere MQ Java 类随附的某些脚本 (例如 IVTRun) 使用此环境变量。

在 Windows 上，安装期间将自动设置所有环境变量。 在任何其他平台上，您必须自己设置这些环境变量。 在 UNIX 系统上，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。 在 AIX， HP-UX， Linux 和 Solaris 上，这些脚本位于 *MQ_INSTALLATION_PATH*/java/bin 目录中。

Java 库的 *IBM WebSphere MQ* 类

Java 库的 IBM WebSphere MQ 类的位置因平台而异。 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，请使用以下格式的 **java** 命令启动应用程序:

```
java -Djava.library.path=library_path application_name
```

其中 *library_path* 是 WebSphere MQ Java 库类 (包括 JNI 库) 的路径。 第 556 页的表 85 显示每个平台的 WebSphere MQ Java 库类的位置。

表 *85:* 每个平台的 *WebSphere MQ Java* 库类的位置

平台	包含 WebSphere MQ Java 库类的目录
AIX	*MQ_INSTALLATION_PATH*/java/lib（32 位库） *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
HP-UX 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）
Linux (x86 平台)	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib（32 位库） *MQ_INSTALLATION_PATH*\java\lib64（64 位库）
MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。	

注:

1. 在 AIX， HP-UX， 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. 在 Windows 上，可以使用 PATH 环境变量来指定 WebSphere MQ classes for Java 库的位置，而不是在 **java** 命令上指定它们的位置。
3. 要在 IBM i 上以绑定方式使用 WebSphere MQ Java 类，请确保库 QMQMJAVA 位于库列表中。

相关任务

使用 WebSphere MQ classes for Java

对 *IBM WebSphere MQ classes for Java* 上 *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。 一个 OSGi 捆绑软件作为 IBM WebSphere MQ classes for Java 的一部分提供。

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 WebSphere MQ classes for Java。 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 WebSphere MQ classes for Java。

where <version number> is the version number of WebSphere MQ that has been installed.

捆绑软件将安装到 IBM WebSphere MQ 安装的 `java/lib/OSGi` 子目录中，或者安装到 Windows 上的 `java\lib\OSGi` 文件夹中。

另外还有 9 个捆绑软件安装到 IBM WebSphere MQ 安装的 `java/lib/OSGi` 子目录或 Windows 上的 `java\lib\OSGi` 文件夹中。 这些捆绑软件是 IBM WebSphere MQ classes for JMS 的一部分，不得装入到已装入了 IBM WebSphere MQ classes for Java 捆绑软件的 OSGi 运行时环境中。 如果将 IBM WebSphere MQ classes for Java OSGi 捆绑软件装入到同时装入了 IBM WebSphere MQ classes for JMS 捆绑软件的 OSGi 运行时环境中，那么会发生以下错误:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

在运行使用 IBM WebSphere MQ classes for Java 捆绑软件或 IBM WebSphere MQ classes for JMS 捆绑软件的应用程序时发生。

IBM WebSphere 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 WebSphere 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 WebSphere MQ classes for Java 捆绑软件，但 IBM WebSphere MQ classes for Java 捆绑软件不会感知到任何用户捆绑软件。 因此， IBM WebSphere 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，请访问 OSGi Alliance Web 站点。

IBM WebSphere MQ classes for Java 配置文件

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

IBM WebSphere MQ classes for Java 配置文件的格式是标准 Java 属性文件的格式。

V 7.5.0.9 从 IBM WebSphere MQ Version 7.5.0 修订包 9 开始，在 IBM WebSphere MQ classes for Java 安装目录的 `bin` 子目录中提供了名为 `mqjava.config` 的样本配置文件。 该文件记录了所有受支持的属性及其缺省值。

注: 在将 IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java 配置文件标识为本地 Windows 系统上的文件 `D:\mydir\mqjava.config`。

IBM WebSphere 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 WebSphere MQ classes for Java 配置文件中指定的属性

IBM WebSphere MQ MQI client 配置文件还可以指定用于配置 IBM WebSphere 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 WebSphere 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 WebSphere MQ MQI client 配置文件中的任何属性 (attribute) 指定为 IBM WebSphere MQ classes for Java 配置文件中的特性 (property) 来覆盖该属性。 要覆盖 IBM WebSphere MQ MQI client 配置文件中的属性，请在 IBM WebSphere 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 WebSphere MQ MQI client 配置文件中包含该属性的节的名称。

propName
IBM WebSphere MQ MQI client 配置文件中指定的属性的名称。

propValue
用于覆盖 IBM WebSphere MQ MQI client 配置文件中指定属性值的特性的值。

或者，可以通过在 **java** 命令中将属性指定为系统属性来覆盖 IBM WebSphere 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 WebSphere MQ MQI client 配置文件中只有以下属性与 IBM WebSphere 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 568 页的『将客户机通道定义表用于 IBM WebSphere MQ classes for Java』，以获取有关如何将 CCDT 与 IBM WebSphere MQ classes for Java 一起使用的详细信息。

表 *86:* 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 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

有关 IBM WebSphere 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 WebSphere MQ classes for Java 应用程序

Java 标准环境跟踪节

您可以使用 Java "标准环境跟踪设置" 节来配置 IBM WebSphere MQ classes for Java 跟踪工具。

com.ibm.msg.client.commonservices.trace.outputName = traceOutputName

traceOutputName 是要将跟踪输出发送到的目录和文件名。

跟踪文件的缺省名称取决于应用程序正在使用的 IBM WebSphere MQ classes for Java 版本:

- 对于 IBM WebSphere MQ classes for Java for Version 7.5.0, Fix Pack 8 或更低版本，*traceOutputName* 缺省为当前工作目录中名为 `mqjms_%PID%.trc` 的文件。
- V 7.5.0.9 从 Version 7.5.0, Fix Pack 9 的 IBM WebSphere MQ classes for Java 开始，*traceOutputName* 缺省为当前工作目录中名为 `mqjava_%PID%.trc` 的文件。

其中 *%PID%* 是当前进程标识。 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）。 要在指定的文件名中包含进程标识，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 是跟踪的包和类的列表，或者是特殊值 `ALL` 或 `NONE`。

使用分号 (`;`) 分隔包或类名。 **includeList** 缺省为 `ALL`，并跟踪 IBM WebSphere MQ classes for Java 中的所有包和类。

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 是未跟踪的包和类的列表，或者是特殊值 `ALL` 或 `NONE`。

使用分号 (`;`) 分隔包或类名。 **excludeList** 缺省为 `NONE`，因此不会跟踪 IBM WebSphere MQ classes for Java 中的任何包和类。

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 **maxArrayBytes** 设置为正整数，那么它将限制字节数组中写入跟踪文件的字节数。 它在将 *maxArrayBytes* 写出后截断字节数组。 设置 **maxArrayBytes** 将减小生成的跟踪文件的大小，并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 与 **traceCycles** 配合使用。 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 是要循环使用的跟踪输出文件数。

如果当前跟踪输出文件达到 **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。 每个跟踪输出文件通过附加到文件名的数字后缀进行区分。 当前或最新的跟踪输出文件具有后缀 .trc.0，下一个最新的跟踪输出文件以 .trc.1 结尾，依此类推。 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** 是 1，那么当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 控制方法参数和返回值是否包含在跟踪中。

traceParameters 缺省为 TRUE。 如果 **traceParameters** 设置为 FALSE，那么仅跟踪方法特征符。

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 **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，那么将改为使用该名称，并且不会将任何后缀应用于该文件。

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。 与未压缩跟踪相比，压缩跟踪对 IBM WebSphere MQ classes for Java 性能的影响较小。

如果设置了 **maxTraceBytes** 和 **traceCycles**，那么将创建多个压缩跟踪文件来代替多个平面文件。

如果 IBM WebSphere MQ classes for Java 以不受控制的方式结束，那么压缩跟踪文件可能无效。 因此，仅当 IBM WebSphere MQ classes for Java 以受控方式关闭时，才必须使用跟踪压缩。 仅当正在调查的问题不会导致 JVM 本身意外停止时，才使用跟踪压缩。 在诊断可能导致 System.Halt() 关闭或异常，不受控制的 JVM 终止的问题时，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：

表 *87:* 针对每个跟踪级别跟踪的内容

值	**跟踪的内容**
0	已关闭跟踪
1	\u5f02\u5e38
3	\u5f02\u5e38 警告
6	\u5f02\u5e38 警告 参考跟踪点
8	\u5f02\u5e38 警告 参考跟踪点 方法入口和出口

表 *87:* 针对每个跟踪级别跟踪的内容 *(*继续*)*	
值	**跟踪的内容**
9	\u5f02\u5e38 警告 参考跟踪点 方法入口和出口 在 IBM WebSphere MQ classes for Java 与队列管理器之间发送的数据。

注: 除非 IBM 支持人员另有指示，否则请始终使用值 9 。

IBM WebSphere MQ classes for Java 和软件管理工具

软件管理工具（如 Apache Maven）可以与 IBM WebSphere 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 WebSphere MQ classes for Java 包含大量 JAR 文件。 使用此 API 开发 Java 语言应用程序时，需要在开发应用程序的机器上安装 IBM WebSphere MQ Server ， IBM WebSphere MQ Client 或 IBM WebSphere MQ Client SupportPac 。

如果要使用软件管理工具并将构成 IBM WebSphere 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 WebSphere MQ 发行版或修订包的一组完整且一致的 JAR 文件。
- 您负责使用 IBM 支持人员提供的任何维护来更新存储库。

对于 IBM WebSphere MQ Version 7.5，需要将以下 JAR 文件安装到存储库中：

- com.ibm.mq.commonservices.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- connector.jar

IBM WebSphere MQ 应用程序的后安装设置

在安装 IBM WebSphere MQ 之后，您可以配置安装，以便运行自己的应用程序。

请记住检查 IBM WebSphere MQ 自述文件以获取有关您的环境的稍后信息或更具体的信息。

在尝试以绑定方式运行 Java 应用程序的 IBM WebSphere MQ 类之前，请确保已按 配置 中所述配置 IBM WebSphere MQ 。

配置队列管理器以接受来自 *WebSphere MQ Java* 类的客户机连接

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 88 页的『准备并运行样本程序』，以了解详细信息。

在 *Java* 安全管理器下运行针对 *Java* 应用程序的 *WebSphere MQ* 类

WebSphere MQ Java 类可以在启用 Java 安全管理器的情况下运行。 要在启用安全管理器的情况下成功运行应用程序，必须使用合适的策略定义文件来配置 Java 虚拟机 (JVM)。

执行此操作的最简单方法是更改随 JRE 提供的策略文件。 在大部分系统上，此文件存储在 lib/security/java.policy 路径（相对于 JRE 目录）中。 可以使用您喜爱的编辑器或与 JRE 一起提供的 policytool 程序来编辑策略文件。

您必须授予对 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 安全管理器下运行时，系统属性 os.name 必须可用于 WebSphere MQ Java 类。

以下是允许 WebSphere MQ Java 类在缺省安全管理器下成功运行的策略文件条目示例:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //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
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase
"file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.commonservices.jar" {
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.mq.commonservices", "read";
  permission java.util.logging.LoggingPermission "control";
  //For access to the trace properties file.
  permission java.io.FilePermission "/tmp/trace.properties", "read";
  //For access to the trace output files.
  permission java.io.FilePermission "/tmp/*", "read,write";
};
```

注意:

- *MQ_INSTALLATION_PATH* 表示安装 WebSphere MQ 的高级目录。
- 此策略文件示例使 WebSphere MQ Java 类能够在安全管理器下正确工作，但您可能仍需要使自己的代码能够在应用程序工作之前正确运行。
- 要允许 WebSphere MQ Java 类访问应用程序的 Java 归档 (JAR) 文件，请向第一个 grant 语句添加以下许可权:

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- 要在策略配置文件中使用这些 grant 语句，您可能需要根据安装 WebSphere MQ Java 类的位置以及存储应用程序的位置来修改路径名。
- WebSphere MQ Java 类随附的样本代码未专门支持与安全管理器配合使用; 但是， IVT 测试在此策略文件和缺省安全管理器到位的情况下运行。

验证 Java 安装的 IBM WebSphere MQ 类

IBM WebSphere MQ classes for Java 随附了安装验证程序 MQIVP。 您可以使用此程序来测试 Java 的 IBM WebSphere MQ 类的所有连接方式。

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 88 页的『准备并运行样本程序』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 552 页的『使用 WebSphere 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 555 页的『与 WebSphere MQ Java 类相关的环境变量』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/VRM/java/samples/wmqjava，其中 MQ_INSTALLATION_PATH 是 IBM WebSphere MQ 安装的路径， VRM 是产品的版本，发行版和修改号。 然后，在命令提示符下，输入：

   ```
   java -Djava.library.path=library_path MQIVP
   ```

   其中 *library_path* 是 Java 库的 IBM WebSphere MQ 类的路径 (请参阅 Java 库的 WebSphere MQ 类)。

   在出现带有 (1) 标记的提示时：

   - 要使用 TCP/IP 连接，请输入 IBM WebSphere 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 WebSphere 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. 如果您选择服务器连接，将不会看到标记为 (2) 的提示。

解决 IBM WebSphere MQ 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 562 页的『验证 Java 安装的 IBM WebSphere MQ 类』中描述了此过程。

如果问题继续存在，并且您需要联系 IBM 服务团队，那么可能会要求您开启跟踪设施。 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 WebSphere MQ classes for Java 库的路径 (请参阅 WebSphere MQ classes for Java 库)。

有关如何使用跟踪的更多信息，请参阅跟踪 IBM WebSphere MQ classes for Java 应用程序。

面向程序员的简介

此主题集合包含程序员的常规信息。

有关编写程序的更多详细信息，请参阅 第 564 页的『为 Java 应用程序编写 WebSphere MQ 类』。

用于 Java 接口的 WebSphere MQ 类

过程 WebSphere MQ 应用程序编程接口使用对对象起作用的动词。 Java 编程接口使用对象，您可以通过调用方法对这些对象执行操作。

过程 WebSphere 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
```

这些动词都将它们要对其进行操作的 WebSphere MQ 对象的句柄作为参数。 因为 Java 是面向对象的，所以 Java 编程接口会轮到这一轮。 您的程序由一组 WebSphere MQ 对象组成，您可以通过对这些对象调用方法来对这些对象执行操作。

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，队列管理器由类 MQQueueManager 的对象表示。 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();
```

为 Java 应用程序编写 WebSphere MQ 类

此主题集合提供了帮助编写 Java 应用程序以与 WebSphere MQ 系统交互的信息。

要使用 WebSphere MQ classes for Java 来访问 WebSphere MQ 队列，请编写 Java 应用程序，其中包含将消息放入 WebSphere MQ 队列并从中获取消息的调用。 有关各个类的详细信息，请参阅 WebSphere MQ classes for Java。

注: WebSphere MQ Java 类不支持自动客户机重新连接。

针对 Java 连接方式的 WebSphere MQ 类

您为 WebSphere MQ Java 类编程的方式与您要使用的连接方式有一些依赖关系。

如果您使用客户机连接，虽然与 IBM WebSphere 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。

用于 *Java* 客户机连接的 *WebSphere MQ* 类

当 WebSphere MQ classes for Java 用作客户机时，它类似于 IBM WebSphere MQ MQI client，但具有许多差异。

如果要对 *WebSphere MQ classes for Java* 进行编程以用作客户机，请注意以下差异:

- 它只支持 TCP/IP。
- 它在启动时不会读取任何 WebSphere 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。
- 只有 WebSphere MQ 客户机配置文件中的以下属性与 WebSphere MQ Java 类相关。 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	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

- 如果连接到需要转换字符数据的队列管理器，那么 V7 Java 客户机现在能够执行转换 (如果队列管理器无法执行转换)。 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。
- WebSphere MQ classes for Java 不支持客户机自动重新连接。

在客户机方式下使用时， *WebSphere MQ classes for Java* 不支持 MQBEGIN 调用。

请参阅 第 553 页的『 WebSphere MQ Java 类的连接选项』 ，以获取有关受支持环境的更多信息。

WebSphere MQ classes for Java 绑定方式

WebSphere MQ Java 类的绑定方式与客户机方式主要有三种不同。

在绑定方式下使用时， WebSphere MQ classes for Java 使用 Java 本机接口 (JNI) 直接调用现有队列管理器 API ，而不是通过网络进行通信。

缺省情况下，在绑定方式下使用 WebSphere MQ Java 类的应用程序使用 *ConnectOption*MQCNO_STANDARD_BINDINGS 连接到队列管理器。

WebSphere 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 175 页的『使用 MQCONNX 调用连接到队列管理器』 。

绑定方式支持 MQBEGIN 调用以在除 WebSphere MQ for IBM i 和 WebSphere MQ for z/OS 之外的所有平台上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。

请参阅 第 553 页的『 WebSphere MQ Java 类的连接选项』，以获取有关受支持环境的更多信息。

定义要用于 *Java* 连接的 *WebSphere MQ* 类

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 WebSphere 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

此主题集合描述如何使用 WebSphere 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 *WebSphere MQ Java* 类设置 *WebSphere 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 647 页的『 WebSphere MQ JMS 类的连接方式』

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 客户机尝试在端口 1414 上连接到 WebSphere 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 (其中该字段称为 **APPLTAG**) 或 WebSphere MQ Explorer **Application Connections** 屏幕 (其中该字段称为 **App name**) 显示。

应用程序名称限制为 28 个字符，超过 28 个字符会被截断。 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（主要）类，但如果此信息不可用，就会使用 `WebSphere 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`。

注: 在 z/OS 平台上运行的队列管理器不支持设置应用程序名称。

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 WebSphere MQ 客户机配置文件中指定的属性

WebSphere MQ 客户机配置文件还可以指定用于配置 WebSphere MQ Java 类的属性。 但是，仅当应用程序以客户机方式连接到队列管理器时， WebSphere MQ MQI 客户机配置文件中指定的属性才适用。

如果需要，可以通过以下任何方式覆盖 WebSphere MQ 配置文件中的任何属性。 这些选项按照优先顺序显示。

- 设置配置属性的 Java 系统属性。
- 在 MQEnvironment.properties 映射中设置此属性。
- 在 Java5 和更高发行版上，设置系统环境变量。

只有 WebSphere MQ 客户机配置文件中的以下属性与 WebSphere MQ Java 类相关。 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

在 *WebSphere MQ Java* 类中连接到队列管理器

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 WebSphere MQ classes for Java*

IBM WebSphere 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。

作为通过在 `MQEnvironment` 类中设置某些字段和环境属性或者将这些字段和环境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的替代方法， IBM WebSphere 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 WebSphere MQ Script (MQSC) 命令或 IBM WebSphere MQ 可编程命令格式 (PCF) 命令，或者使用 IBM WebSphere MQ Explorer 来创建的。

当应用程序创建 `MQQueueManager` 对象时， IBM WebSphere MQ classes for Java 客户机将在客户机通道定义表中搜索合适的客户机连接通道定义，并使用该通道定义来启动 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` 中包含一个客户机通道定义表，并且该文件存储在运行此应用程序的同一系统上，那么该应用程序可以通过以下方式创建一个 URL 对象：

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

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);
```

此语句使 IBM WebSphere MQ classes for Java 客户机访问由 URL 对象 chanTab2 标识的客户机通道定义表，在该表中搜索合适的客户机连接通道定义，然后使用该通道定义启动到名为 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。 如果设置了通道名称，那么 IBM WebSphere MQ classes for Java 客户机将抛出 `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 163 页的『消息队列接口概述』。

  如果应用程序使用连接池，请参阅 第 585 页的『控制 WebSphere MQ Java 类中的缺省连接池』。
- 当 IBM WebSphere 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，它仅使用从此通道定义中抽取的信息来启动 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。

  如果您正在使用安全套接字层（SSL），请特别注意以下几点：
  - 仅当从客户机通道定义表中抽取的通道定义指定 IBM WebSphere MQ classes for Java 客户机支持的 CipherSpec 的名称时， MQI 通道才会使用 SSL。
  - 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 WebSphere MQ classes for Java 客户机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联机证书状态协议 (OCSP) 响应程序的位置。 IBM WebSphere MQ classes for Java 无法在客户机通道定义表文件中使用 OCSP 信息。 但是，您可以按照使用联机证书协议部分中所述来配置 OCSP。

  有关将 SSL 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 SSL。

  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以 Java， C 或 C++ 编写的通道出口。有关如何在 Java 中编写通道出口的更多信息，请参阅 第 580 页的『在 WebSphere MQ Java 类中创建通道出口』。 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 583 页的『将未使用 Java 编写的通道出口与 WebSphere MQ Java 类配合使用』。

指定 *IBM WebSphere 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 WebSphere MQ classes for Java 应用程序尝试以客户机方式连接到 IBM WebSphere MQ 队列管理器时，防火墙可能仅允许源自指定端口或端口范围的那些连接。 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 WebSphere 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 WebSphere MQ Java 类中的队列，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。 有关这些内容的详细信息，请参阅 类 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 WebSphere MQ Java 类中的消息

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。 与过程接口 (其中 MQPUT 和 MQGET 放置和获取字节数组) 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。

创建新消息实例时，所有 MQMD 参数都会自动设置为其缺省值，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 902 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。 当您的消息将由另一个 Java 程序 (使用 readUTF()) 读取时，这是发送字符串信息的最简单方法。

提高 *WebSphere MQ Java* 类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 WebSphere MQ Java 类中，使用 CMQC.MQSO_READ_AHEAD 和 CMQC.MQSO_NO_READ_AHEAD 属性，用于确定是否允许消息使用者和队列浏览器对该对象使用预读。

使用 *WebSphere 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 WebSphere MQ classes for Java 中发布/预订

在 WebSphere MQ classes for Java 中，该主题由 MQTopic 类表示，您使用 MQTopic.put() 方法向其发布。

有关 WebSphere MQ 发布/预订的常规信息，请参阅 WebSphere MQ 发布/预订消息传递简介 。

使用 WebSphere MQ Java 类处理 WebSphere MQ 消息头

提供了表示不同类型的消息头的 Java 类。 同时还提供了两个助手类。

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。

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，它解析头内容，但也允许您搜索特定头，添加新头，除去现有头，更新头字段，然后将头内容写回消息。 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 WebSphere MQ 头类与特定消息类型和格式相关联。 配置了 MQHeaderRegistry 以了解所有当前 WebSphere MQ 格式和头类型及其实现类，并且您还可以注册自己的头类型。

为以下常用的 Websphere MQ 头提供了支持

- MQRFH - 规则和格式化头
- MQRFH2 -类似于 MQRFH ，用于向属于 WebSphere Message Broker 的消息代理传递消息。 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 *WebSphere MQ Java* 类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 *WebSphere 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 *WebSphere MQ Java* 类在死信消息中查找原因码

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 *WebSphere MQ classes for Java* 从死信消息中读取和除去 *MQDLH*

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 *WebSphere MQ Java* 类打印消息内容

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 *WebSphere 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 *WebSphere MQ classes for Java* 来分析 *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。 此示例显示如何访问指定文件夹中的已知字段值-在此实例中，这是 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 *WebSphere MQ Java* 类来读写除 *MQMessage* 对象以外的字节流

当数据源不是 MQMessage 对象时，这些示例使用头类来解析和处理 WebSphere 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 WebSphere MQ 头内容。 每个头类实现的 MQHeader 接口都提供了方法 `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。

您还可以从其他数据流 (例如，文件或套接字流或 JMS 消息中携带的字节数组) 获取 DataInput 和 DataOutput 对象。 java.io.DataInputStream 类实现了 DataInput，java.io.DataOutputStream 类实现了 DataOutput。 此示例从字节数组中读取 WebSphere 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 *WebSphere MQ classes for Java* 为新头类型创建类

您可以为未随 WebSphere MQ Java 类提供的头类型创建 Java 类。

要添加表示新头类型的 Java 类，您可以采用与 WebSphere MQ Java 类随附的任何头类相同的方式使用该 Java 类，请创建用于实现 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 WebSphere MQ Java 类处理 PCF 消息

提供了 Java 类以创建和解析 PCF 结构化消息，并促进发送 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 WebSphere MQ Java 类中的消息属性

用于处理消息句柄的函数调用在 WebSphere MQ Java 类中没有等效的函数调用。 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 16 页的『属性名称』。

在 WebSphere MQ 中，用于 Java 的类通过 MQMessage 类访问消息。 因此，在 Java 环境中未提供消息句柄，并且没有等效于 WebSphere MQ 函数调用 MQCRTMH， MQDLTMH， MQMHBUF 和 MQBUFMH

要在过程接口中设置消息句柄属性，您可以使用调用 MQSETMP。 在 WebSphere MQ Java 类中，使用 MQMessage 类的相应方法：

- 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 WebSphere MQ Java 类中，使用 MQMessage 类的相应方法：

- 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 WebSphere MQ Java 类中，使用 MQMessage 类的 deleteProperty 方法。

处理 WebSphere MQ Java 类中的错误

处理使用 Java `try` 和 `catch` 块的 WebSphere MQ Java 类所产生的错误。

Java 接口中的方法不会返回完成代码和原因码。 相反，只要由 WebSphere MQ 调用生成的完成代码和原因码都不为零，它们就会抛出异常。 这将简化程序逻辑，使您不必在每次调用 WebSphere MQ 之后检查返回码。 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() );
}
```

在所有其他平台的 z/OS 的原因码 和 原因码 中记录了 z/OS 的 Java 异常中返回的 WebSphere MQ 调用原因码。

在运行 WebSphere MQ Java 应用程序类时抛出的异常也会写入日志。 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。

缺省情况下， MQException 类会自动将异常流式传输到 System.err，这通常会定向到控制台。 如果要停止在控制台上出现的异常，请在应用程序中包含一行以设置 MQException.log= null。

在 WebSphere MQ Java 类中获取和设置属性值

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 中数组的长度始终已知。 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。 WebSphere MQ classes for Java 允许队列管理器对象由多个线程共享，但确保对目标队列管理器的所有访问都同步。

多线程程序在 Java 中很难避免。 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。

对于基于 C 的 WebSphere MQ MQI 客户机，这将导致问题，因为存在多个线程共享句柄的限制。 WebSphere MQ Java 类会放宽此约束，允许多个线程共享队列管理器对象 (及其关联的队列，主题和进程对象)。

WebSphere MQ classes for Java 的实现确保对于特定连接 (MQQueueManager 对象实例) ，同步对目标 WebSphere MQ 队列管理器的所有访问权。 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 WebSphere MQ Java 类中使用通道出口

有关如何使用 WebSphere 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，如何对其进行分配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 *WebSphere MQ Java* 类中创建通道出口

您可以通过定义实现相应接口的 Java 类来提供自己的通道出口。

要实现出口，请定义用于实现相应接口的新 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 WebSphere 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 SSL 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，您的出口代码必须返回您希望 WebSphere MQ 解释的已修改数据。

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

在低于 7.0 的 WebSphere MQ 版本中，您将使用接口 MQSendExit， MQReceiveExit 和 MQSecurityExit 来实现这些出口，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 WebSphere MQ classes for Java* 中分配通道出口

您可以使用 IBM WebSphere MQ classes for Java 分配通道出口。

在 IBM WebSphere MQ classes for Java 中没有与 IBM WebSphere 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 (只能用于 IBM WebSphere MQ 版本低于 7.0 的版本中使用的出口类型) 和 channelSend 出口字段 (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 WebSphere 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 582 页的表 88 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 582 页的表 88 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 *88:* 通道出口程序的目录

平台	目录
AIX， HP-UX， Linux 和 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 WebSphere MQ 数据文件选择的目录。 缺省目录为 C:\Program Files\IBM\WebSphere MQ。	

将数据传递到 *WebSphere MQ Java* 类中的通道出口

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，您的出口代码必须返回您希望 WebSphere MQ 解释的已修改数据。

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 568 页的『将客户机通道定义表用于 IBM WebSphere MQ classes for Java』。

将未使用 *Java* 编写的通道出口与 *WebSphere MQ Java* 类配合使用

如何从 Java 应用程序使用以 C 编写的通道出口程序。

在 WebSphere MQ Version 7.0 中，可以将以 C 编写的通道出口程序的名称指定为传递到 MQEnvironment 对象或属性 Hashtable 中的 channelSecurityExit ， channelSendExit 或 channelReceiveExit 字段的字符串。 但是，不能在使用其他语言编写的应用程序中使用以 Java 编写的通道出口。

指定格式为 `library(function)` 的出口程序名，并确保出口程序的位置包含在路径环境变量中。

有关如何使用 C 语言编写通道出口的信息，请参阅第 330 页的『消息传递通道的通道出口程序』。

使用外部出口类

在低于 V 7.0 的 WebSphere MQ 版本中，提供了三个类，使您能够使用以 Java 以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。

要使用未使用 Java 编写的安全出口，应用程序首先必须创建 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 582 页的表 88 中显示的目录中。

在 *WebSphere MQ Java* 类中使用一系列通道发送或接收出口

WebSphere MQ Java 应用程序类可以使用连续运行的一系列通道发送或接收出口。

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 类的实例 (对于未使用 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。

如果使用字符串，那么它必须由一个或多个逗号分隔的出口定义组成，每个出口定义可以是 Java 类的名称，也可以是格式为 `library(function)` 的 C 程序。

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，如果链接了 Java 出口和 C 出口，那么 Java 出口的存在对 C 出口没有影响。

使用出口链类

在低于 V 7.0 的 WebSphere MQ 版本中，提供了两个类以允许出口序列:

- 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 WebSphere MQ Classes for Java 接口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 类的实例 (对于未使用 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。

WebSphere MQ Java 类中的通道压缩

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 WebSphere MQ classes for Java 使用内置到 IBM WebSphere MQ 中的压缩功能。

使用随 IBM WebSphere 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 WebSphere MQ 中的实现方式，请参阅数据压缩 (COMPMSG) 和头压缩 (COMPHDR)。

IBM WebSphere MQ classes for Java 应用程序指定可用于通过创建 java.util.Collection 对象在客户机连接上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的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 WebSphere 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 WebSphere 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 WebSphere 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。

WebSphere MQ Java 类中的连接池

WebSphere MQ Java 类允许将备用连接合用以复用。

WebSphere MQ Java 类为处理与 WebSphere MQ 队列管理器的多个连接的应用程序提供额外支持。 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。

WebSphere MQ 提供了缺省连接池。 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。 如果池在 WebSphere MQ classes for Java 构造 MQQueueManager 对象时处于活动状态，那么它将搜索此缺省池并复用任何合适的连接。 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。

控制 *WebSphere MQ Java* 类中的缺省连接池

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。

WebSphere MQ classes for Java 提供了可用于执行此操作的缺省连接池。 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 587 页的『在 WebSphere MQ Java 类中提供其他连接池』）。

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。

WebSphere MQ Java 类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 *WebSphere MQ Java* 类中提供其他连接池

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，应用程序调用 MQApp3.main()。 这将运行多个线程，其中每个线程独立使用 WebSphere MQ 。 这些线程会在伪造连接时使用 myConnMan。

为 *WebSphere MQ Java* 类提供您自己的 *ConnectionManager*

WebSphere MQ classes for Java 提供 Java EE 连接器体系结构的部分实现，允许使用 javax.resource.spi.ConnectionManager 的实现。

应用程序和中间件提供程序可以提供连接池的备用实现。 WebSphere MQ Java 类提供了 Java EE 连接器体系结构的部分实现。 **javax.resource.spi.ConnectionManager** 的实现可以用作缺省 Connection Manager ， 也可以在 MQQueueManager 构造函数上指定。

WebSphere MQ Java 类符合 Java EE 连接器体系结构的 "连接管理" 合同。 请结合 Java EE 连接器体系结构的 "连接管理" 合同阅读本部分 (请参阅 Sun 的 Java Web 站点 (https://java.sun.com))。

ConnectionManager 接口仅定义一个方法：

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

MQQueueManager 构造函数在相应的 ConnectionManager 上调用 allocateConnection 。 它将 ManagedConnectionFactory 和 ConnectionRequestInfo 的相应实现作为描述所需连接的参数传递。

ConnectionManager 在其池中搜索已使用相同的 ManagedConnectionFactory 和 ConnectionRequestInfo 对象创建的 javax.resource.spi.ManagedConnection 对象。 如果 ConnectionManager 找到任何合适的 ManagedConnection 对象，那么它将创建包含候选 ManagedConnections 的 java.util.Set 。 然后，ConnectionManager 调用以下内容:

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

ManagedConnectionFactory 的 WebSphere MQ 实现将忽略主体参数。 此方法从集合中选择并返回合适的 ManagedConnection ，或者如果找不到合适的 ManagedConnection，那么返回 null。 如果池中没有合适的 ManagedConnection ，那么 ConnectionManager 可以使用以下命令创建一个:

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

同样，将忽略主体参数。 此方法连接到 WebSphere MQ 队列管理器，并返回表示新伪造连接的 javax.resource.spi.ManagedConnection 的实现。 一旦 ConnectionManager 获取了 ManagedConnection (从池或刚创建) ，它就会使用以下命令创建连接句柄:

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

可以从 allocateConnection() 返回此连接句柄。

ConnectionManager 必须通过以下方法在 ManagedConnection 中注册兴趣:

```
mc.addConnectionEventListener()
```

如果在连接上发生严重错误，或者调用 MQQueueManager.disconnect () 时，将通知 ConnectionEvent 侦听器。 调用 MQQueueManager.disconnect () 时， ConnectionEvent 侦听器可以执行以下任一操作:

- 使用 mc.cleanup() 调用重置 ManagedConnection ，然后将 ManagedConnection 返回到池
- 使用 mc.destroy() 调用销毁 ManagedConnection

如果 ConnectionManager 是缺省 ConnectionManager，那么它还可以注册对 MQPoolTokens 的 MQEnvironment 管理的集合的状态感兴趣。 要执行此操作，请首先构造 MQPoolServices 对象，然后向 MQPoolServices 对象注册 MQPoolServicesEventListener 对象:

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

当从集合中添加或除去 MQPoolToken 时，或者当缺省 ConnectionManager 更改时，将通知侦听器。 MQPoolServices 对象还提供了一种方法来查询 MQPoolTokens 集合的当前大小。

使用 WebSphere MQ Java 类进行 JTA/JDBC 协调

WebSphere MQ classes for Java 支持 MQQueueManager.begin () 方法，此方法允许 WebSphere MQ 充当提供 JDBC 类型 2 或 JDBC 类型 4 兼容驱动程序的数据库的协调程序。

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 https://www.ibm.com/software/integration/wmq/requirements/。

要使用 XA-JTA 支持，您必须使用特殊 JTA 切换库。 使用此库的方法根据您使用的是 Windows 还是其他某个平台而有所不同。

在 *Windows* 上配置 *JTA/JDBC* 协调

XA 库作为 DLL 提供，名称格式为 `jdbcxxx.dll`。

V 7.5.0.7 提供的 `jdbcora12.dll` 与 Oracle 12C 兼容，适用于 IBM WebSphere MQ 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 Java 安装的 IBM WebSphere MQ 类的 `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 WebSphere 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`

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， WebSphere 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。

必须使用 WebSphere MQ 随附的相应 Makefile 来链接每个对象文件。 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 位交换机库，请在 WebSphere 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 (对于 DB2) 和 `oracle` (对于 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，每个交换机库的名称为 jdbcdb2 ，对于 Oracle，每个交换机库的名称为 jdbcora。 Makefile 确保 32 位和 64 位交换机库存储在不同的 WebSphere MQ 目录中。 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。

在为 DB2 和/或 Oracle 创建交换机库之后，必须向队列管理器声明这些库。 如果队列管理器配置文件 (qm.ini) 已包含 DB2 或 Oracle 数据库的 XAResourceManager 节，那么必须将每个节中的 SwitchFile 条目替换为下列其中一项:

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未包含 DB2 或 Oracle 数据库的 XAResourceManager 节，或者如果要添加其他 XAResourceManager 节，请参阅 管理 以获取有关如何构造 XAResourceManager 节的信息。 但是，新的 XAResourceManager 节中的每个 SwitchFile 条目必须与先前针对 DB2 或 Oracle 数据库描述的完全相同。 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 支持存在某些问题和限制，某些问题和限制取决于正在使用的数据库管理系统。

因为此支持会对 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。 **始终** 避免在有应用程序保持与数据库的打开连接时突然关闭数据库。

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
```

请参阅 DB2 文档以获取更多信息。

必须将 XAResourceManager 节配置为使用 ThreadOfControl=PROCESS。 对于 DB2 V 8.1 及更高版本，这与 DB2 的缺省控制线程设置不匹配，因此必须在 XA Open String 中指定 toc=p。 具有 JTA/JDBC 协调的 DB2 的 XAResourceManager 节示例如下所示：

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
```

```
XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
ThreadOfControl=PROCESS
```

这不会阻止使用 JTA/JDBC 协调的 Java 应用程序成为多线程应用程序本身。

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。

WebSphere MQ Java 类中的安全套接字层 (SSL) 支持

WebSphere MQ classes for Java 客户机应用程序支持安全套接字层 (SSL) 加密。 您需要 JSSE 提供程序才能使用 SSL 加密。

WebSphere MQ 类 (针对使用传输 (CLIENT) 的 Java 客户机应用程序) 支持安全套接字层 (SSL) 加密。 SSL 提供了通信加密、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。

WebSphere MQ Java 类使用 Java 安全套接字扩展 (JSSE) 来处理 SSL 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 WebSphere MQ classes for Java 客户机应用程序使用客户机通道定义表 (CCDT) 来连接到队列管理器，请参阅 第 568 页的『将客户机通道定义表用于 IBM WebSphere MQ classes for Java』。

在 *IBM WebSphere MQ classes for Java* 中启用 *SSL*

指定 CipherSuite 以启用 SSL。 可以通过两种方法来指定 CipherSuite。

只有客户机连接支持 SSL。 要启用 SSL，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 595 页的『WebSphere MQ Java 类中的 SSL CipherSpecs 和 CipherSuites』 包含一个表，将 IBM WebSphere MQ 支持的 CipherSpec 映射到 JSSE 已知的对等 CipherSuite。

要启用 SSL，使用 MQEnvironment 的 sslCipherSuite 静态成员变量指定 CipherSuite。 以下示例连接到名为 SECURE.SVRCONN.CHANNEL，已设置为需要 CipherSpec 为 RC4_MD5_EXPORT:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

虽然通道的 CipherSpec 为 RC4_MD5_EXPORT，但 Java 应用程序必须将 CipherSuite 指定为 SSL_RSA_EXPORT_WITH_RC4_40_MD5。 请参阅第 595 页的『WebSphere MQ Java 类中的 SSL CipherSpecs 和 CipherSuites』，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 568 页的『将客户机通道定义表用于 IBM WebSphere MQ classes for Java』。

如果需要客户机连接以使用 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 sslFips 必需字段设置为 `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 WebSphere 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 SSL 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

适用于 UNIX， Linux 和 Windows 的联邦信息处理标准 (FIPS)

在 *IBM WebSphere 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 SSL 证书识别自身。 IBM WebSphere MQ classes for Java 客户机应用程序可以使用此 DN 来确保它正在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 WebSphere 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 WebSphere MQ classes for Java 然后根据指定的 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 WebSphere MQ 的更多信息，请参阅 使用证书撤销列表和权限撤销列表 和 使用 WebSphere MQ classes for Java 和 WebSphere 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 *WebSphere MQ Java* 类中重新协商密钥

WebSphere MQ classes for Java 客户机应用程序可以根据发送和接收的总字节数来控制何时重新协商用于客户机连接上加密的密钥。

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 字段或环境属性 MQC.SSL_RESET_COUNT_PROPERTY 的值表示 WebSphere MQ classes for Java 客户机代码在重新协商密钥之前发送和接收的总字节数。 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。 字节数还包括 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java 使用此 SSLSocketFactory 来连接到 IBM WebSphere MQ 队列管理器。 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 *WebSphere MQ Java* 类中对 *JSSE* 密钥库或信任库进行更改

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，那么针对当时正在运行的 Java 应用程序的 WebSphere MQ 类不会自动执行更改。 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 *SSL* 与 *WebSphere MQ Java* 类配合使用时的错误处理

当使用 SSL 连接到队列管理器时， WebSphere MQ Java 类可以发出许多原因码。

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。 只有客户机连接支持 SSL。

MQRC_JSSE_ERROR
JSSE 提供程序报告了 WebSphere 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
MQCONN 或 MQCONNX 调用已与所指定的 SSL 配置选项一起发出，但是在初始化 SSL 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 595 页的『 WebSphere MQ Java 类中的 SSL CipherSpecs 和 CipherSuites』中找到可用于与 WebSphere 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。

WebSphere MQ Java 类中的 *SSL CipherSpecs* 和 *CipherSuites*

IBM WebSphere MQ classes for Java 应用程序是否可以与队列管理器建立连接取决于在 MQI 通道的服务器端指定的 CipherSpec 和在客户机端指定的 CipherSuite 。

对于 CipherSpec 和 CipherSuite 的每个组合， IBM WebSphere MQ classes for Java 应用程序是否可以连接到队列管理器取决于 MQEnvironment 类中 sslFips 必需字段的值，或者取决于环境属性 CMQC.SSL_FIPS_REQUIRED_PROPERTY 的值。

在 MQI 通道的服务器端，可以在 DEFINE CHANNEL CHLTYPE (SVRCONN) 命令中将 CipherSpec 的名称指定为 SSLCIPH 参数的值。 在 MQI 通道的客户机端， IBM WebSphere MQ classes for Java 应用程序可以在 MQEnvironment 类中设置 sslCipherSuite 字段，或者设置环境属性 CMQC.SSL_CIPHER_SUITE_PROPERTY。

配置应用程序以使用 IBM Java 或 Oracle Java CipherSuite 映射

从 IBM WebSphere MQ Version 7.5.0 修订包 5 开始，您可以配置应用程序是使用缺省 IBM Java CipherSuite 到 WebSphere MQ CipherSpec 映射，还是使用 Oracle CipherSuite 到 WebSphere MQ CipherSpec 映射。 另外，无论应用程序是使用 IBM JRE 还是 Oracle JRE ，都可以使用 TLS CipherSuites 。 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 WebSphere MQ CipherSpec 映射。

该值为缺省值。

false
使用 Oracle CipherSuite 到 WebSphere MQ CipherSpec 映射。

下表列出了 IBM WebSphere MQ 支持的 CipherSpec 及其对应的 CipherSuite。 此表还指示如果在 MQI 通道的服务器端指定了 CipherSpec ，并且在客户机端指定了等效的 CipherSuite ，那么 IBM WebSphere MQ classes for Java 应用程序是否可以与队列管理器建立连接。

表 *89: WebSphere MQ* 及其等效 *CipherSuites* 支持的 *CipherSpecs*

CipherSpec	**对应的 CipherSuite**	**如果 SFIPS [1] 设置为 YES ，是否可以连接?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	否
NULL_SHA	SSL_RSA_WITH_NULL_SHA	否
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5 (IBM JRE) 没有等效的 Oracle JRE。	否
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	否
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA (IBM JRE) 没有等效的 Oracle JRE。	否
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (IBM JRE) SSL_RSA_EXPORT_WITH_RC4_40_MD5 (Oracle JRE)	否
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA (IBM JRE) 没有等效的 Oracle JRE。	否
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA (IBM JRE) 没有等效的 Oracle JRE。	否

表 89: WebSphere MQ 及其等效 CipherSuites 支持的 CipherSpecs (继续)		
CipherSpec	**对应的 CipherSuite**	**如果 SFIPS [1] 设置为 YES，是否可以连接?**
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (IBM JRE) 没有等效的 Oracle JRE。	否
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA (IBM JRE) 没有等效的 Oracle JRE。	否
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256 (IBM JRE) TLS_RSA_WITH_NULL_SHA256 (Oracle JRE)	否 [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA (Oracle JRE)	是 [5] [7]
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA256 (Oracle JRE)	是 [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA (Oracle JRE)	是 [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA256 (Oracle JRE)	是 [5] [7]
AES_SHA_US [2]		
TLS_RSA_WITH_DES_CBC_SHA [8]	SSL_RSA_WITH_DES_CBC_SHA	否 [3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA [8] [9]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Yes
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA (IBM JRE) 没有等效的 Oracle JRE。	否 [4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (IBM JRE) 没有等效的 Oracle JRE。	否 [6]

注意:

1. 在 IBM WebSphere MQ classes for Java 应用程序中，通过将 MQEnvironment 类中的 sslFips 必需字段设置为 `true` 来指示仅使用 FIPS 认证的算法，并指示还可以通过将 sslFips 必需字段设置为 `false` 来使用非 FIPS 认证的算法。 或者，设置环境属性 CMQC.SSL_FIPS_REQUIRED_PROPERTY。
2. 此 CipherSpec 没有等效的 CipherSuite。
3. 此 CipherSpec 在 2007 年 5 月 19th 之前经过 FIPS 140-$tag3 认证。
4. 此 CipherSpec 在 2007 年 5 月 19th 之前经过 FIPS 140-$tag3 认证。 名称 FIPS_WITH_DES_CBC_SHA 是历史名称，反映了此 CipherSpec 先前 (但不再) 符合 FIPS 的事实。 不推荐使用此 CipherSpec。

5. 这些 CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA， TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA， TLS_RSA_WITH_AES_256_CBC_SHA256) 无法用于保护从 WebSphere MQ Explorer 到队列管理器的连接，除非将适当的无限制策略文件应用于资源管理器所使用的 JRE。

   有关策略文件的更多信息，请参阅 安全性信息 。
6. 名称 FIPS_WITH_3DES_EDE_CBC_SHA 是历史名称，反映了此 CipherSpec 先前 (但不再) 符合 FIPS 的事实。 不推荐使用此 CipherSpec。
7. 这些 CipherSpecs (TLS_RSA_WITH_NULL_SHA256， TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256， TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) 需要 IBM JRE 6.0 SR13 FP2 ， 7.0
8. 这些 CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA， TLS_RSA_WITH_DES_CBC_SHA , TLS_RSA_WITH_RC4_128_SHA256) 可以使用 SSLv3 或 TLS。 缺省情况下，未启用 FIPS 时，将使用 SSLv3 。 要使用 TLS ，请将 Java 系统属性 **com.ibm.mq.cfg.preferTLS** 设置为 `true`。
9. 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。

相关信息

指定运行时在 MQI 客户机上仅使用经过 FIPS 认证的 CipherSpecs

适用于 UNIX， Linux 和 Windows 的联邦信息处理标准 (FIPS)

MQdev 博客: MQ Java， TLS Ciphers， Non-IBM JRE 和 APAR IT06775， IV66840， IT09423， IT10837

MQdev 博客: MQ CipherSpecs 与 Java 密码套件之间的关系

针对 Java 应用程序运行 WebSphere MQ 类

如果使用客户机或绑定方式编写应用程序 (包含 main () 方法的类) ，请使用 Java 解释器运行程序。

使用以下命令:

```
java -Djava.library.path=library_path MyClass
```

其中 *library_path* 是 WebSphere MQ classes for Java 库的路径 (请参阅 WebSphere MQ classes for Java 库)。

针对依赖于 Java 环境的行为的 WebSphere MQ 类

WebSphere MQ Java 类允许您创建可以针对不同版本的 WebSphere MQ 运行的应用程序。 此主题集合描述依赖于这些不同版本的 Java 类的行为。

WebSphere MQ classes for Java 提供了一个核心类，这些类在所有环境中提供了一致的功能和行为。 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。

除此处指出的情况外，所显示的行为如适用于队列管理器的 "应用程序编程参考" 中所述。

WebSphere MQ Java 类中的核心类

WebSphere MQ classes for Java 包含一组核心类，可在所有环境中使用。

以下类集合被视为核心类，只需要执行 第 599 页的『 WebSphere MQ Java 类核心类的限制和变体』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 599 页的『 WebSphere MQ Java 类核心类的限制和变体』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。 (请参阅 第 553 页的『 WebSphere MQ Java 类的连接选项』。)

WebSphere MQ Java 类核心类的限制和变体

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。 此行为如同使用了 Windows， UNIX 或 Linux WebSphere MQ 队列管理器，但以下次要限制和变体除外。

WebSphere MQ Java 类中 *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 使用时，不支持 MQGMO_SET_SIGNAL。

WebSphere MQ Java 类中 *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。

WebSphere MQ Java 类中 *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

WebSphere MQ Java 类中 *MQCNO_ * 值的限制和变体
某些 MQCNO_* 值不受支持。

- WebSphere MQ Java 类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。

WebSphere MQ Java 类中 *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。

WebSphere MQ Java 类核心类外部的功能

WebSphere MQ Java 类包含特定函数，这些函数专门设计为使用并非所有队列管理器都支持的 API 扩展。 本主题集合描述在使用不支持它们的队列管理器时它们表现出的行为。

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 WebSphere MQ 队列管理器。 否则，它将失败并产生 MQRC_ENVIRONMENT_ERROR。

请参阅第 589 页的『使用 WebSphere MQ Java 类进行 JTA/JDBC 协调』以获取更多详细信息。

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。

WebSphere MQ Java 类中的分发列表中的限制
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。 尤其是， 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。

使用 *WebSphere 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
偏移量
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。 请勿将已分组或分段的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 应用程序尝试从不支持这些字段的队列管理器中获取 () 消息，并且要检索的物理消息是一组分段消息的一部分 (即，它具有 MQMD 字段的非缺省值)，那么将在没有错误的情况下检索该消息。 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 下针对 *Java* 的 *WebSphere MQ* 类的限制

在 CICS Transaction Server for z/OS 环境中，仅允许主 (第一个) 线程发出 CICS 或 WebSphere MQ 调用。

请注意，不支持在 CICS Java 应用程序中使用 WebSphere 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。

在 *Java* 平台 *Enterprise Edition* 中运行 *Java* 应用程序的 *IBM WebSphere MQ* 类

在 Java EE 中使用 Java 的 IBM WebSphere MQ 类之前，必须考虑某些限制和设计注意事项

在 Java EE 环境中使用 Java 的 IBM WebSphere MQ 类时存在限制。 在设计，实现和管理在 Java EE 环境中运行的 Java 应用程序的 IBM WebSphere MQ 类时，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

对于使用 IBM WebSphere MQ classes for Java 的应用程序，唯一受支持的事务管理器是 IBM WebSphere MQ 本身。 虽然 JTA 控制下的应用程序可以将 IBM WebSphere MQ 类用于 Java，但通过这些类执行的任何工作都不受 JTA 工作单元控制。 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的事务中直接使用 IBM WebSphere MQ 来执行消息传递工作，必须改为使用 JMS 的 IBM WebSphere MQ 类。

线程创建

Java 的 IBM WebSphere MQ 类在内部为各种操作创建线程。 例如，当以 BINDINGS 方式运行以直接在本地队列管理器上进行调用时，将在由 IBM WebSphere MQ classes for Java 在内部创建的 "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。 当应用程序将 IBM WebSphere MQ 类用于 Java 时，应用程序服务器可能无法区分应用程序代码与 Java 代码的 IBM WebSphere MQ 类，因此先前描述的线程会导致应用程序不符合容器规范。 IBM WebSphere MQ JMS 类不会破坏这些 Java EE 规范，因此可以改为使用这些规范。

安全限制

应用程序服务器实施的安全策略可能会阻止 IBM WebSphere MQ classes for Java API 执行的某些操作，例如创建和操作新的控制线程 (如前面部分中所述)。

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）。 当启用 Java 安全性时， Java 的 IBM WebSphere MQ 类可能会破坏为应用程序服务器定义的 Java 安全策略线程技术规则，并且 API 可能无法创建其功能所需的所有线程。 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将 IBM WebSphere MQ 类用于 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优点是应用程序隔离。 Java 的 IBM WebSphere MQ 类的设计和实现早于 Java EE 环境。 IBM WebSphere MQ Java 的类可以不支持应用程序隔离概念的方式使用。 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
  - 用于受保护客户机连接的 SSL 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，每个应用程序都必须通过创建具有特定属性集的 MQQueueManager 对象来使用其自己的不同配置，而不是缺省为进程范围的 MQEnvironment 类中配置的属性。
- MQEnvironment 类引入了一些静态方法，这些方法在同一 JVM 进程中使用 IBM WebSphere MQ classes for Java 对所有应用程序全局执行操作，因此无法覆盖特定应用程序的此行为。 示例包括：
  - 配置 SSL 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。
- 通过使用 MQEnvironment 类和 MQQueueManager 对象属性，为使用 Java 的 IBM WebSphere MQ 类的应用程序配置 SSL。 它未与应用程序服务器本身的受管安全配置相集成。 必须确保为 Java 相应地配置 IBM WebSphere MQ 类，以提供所需的安全性级别，而不使用应用程序服务器配置。

绑定方式限制

IBM WebSphere MQ 和 WebSphere Application Server 可以安装在同一机器上，以使 WebSphere Application Server 中提供的队列管理器和 IBM WebSphere MQ 资源适配器 (RA) 的主要版本不同。 例如，可将 WebSphere Application Server Version 7.0(提供 IBM WebSphere MQ RA 级别 7.0.1) 安装在与 Version 6.0 队列管理器相同的机器上。

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 到使用资源适配器的队列管理器的任何连接都必须使用客户机类型的连接。 如果版本相同，那么可以使用绑定连接。

使用 WebSphere MQ classes for JMS

WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) 是随 WebSphere MQ 提供的 JMS 提供程序。 除了实现 javax.jms 包中定义的接口外， WebSphere MQ JMS 类还提供了两组 JMS API 扩展。

JMS 规范定义了一组接口，应用程序可以使用这些接口来执行消息传递操作。 javax.jms 包定义 JMS 接口，JMS 提供程序为特定消息传递产品实现这些接口。 WebSphere MQ V 7.5 当前使用 JMS 1.1 规范。 WebSphere MQ classes for JMS 是实现 WebSphere MQ 的 JMS 接口的 JMS 提供程序。

JMS 规范期望 ConnectionFactory 和 Destination 对象是受管对象。 管理员在中央存储库中创建并维护受管对象， JMS 应用程序使用 Java 命名和目录接口 (JNDI) 检索这些对象。 WebSphere MQ JMS 类支持使用受管对象，管理员可以使用 WebSphere MQ JMS 管理工具或 WebSphere MQ Explorer 来创建和维护受管对象。

WebSphere MQ classes for JMS 还提供了两组 JMS API 扩展。 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。

WebSphere MQ JMS 扩展
先前发行版的 WebSphere MQ JMS 类包含在对象 (例如， MQConnectionFactory， MQQueue 和 MQTopic 对象) 中实现的扩展。 这些对象具有特定于 WebSphere 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。 此发行版的 WebSphere MQ JMS 类维护这些扩展，这些扩展现在称为 WebSphere 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
此发行版的 WebSphere MQ JMS 类提供了一组更通用的 JMS API 扩展，这些扩展并非特定于作为消息传递系统的 WebSphere MQ 。 这些扩展称为 IBM JMS 扩展，具有以下广泛目标:

- 在 IBM JMS 提供程序之间提供更高级别的一致性
- 便于在两个 IBM 消息传递系统之间编写网桥应用程序
- 为了更轻松地将应用程序从一个 IBM JMS 提供程序移植到另一个提供程序

这些扩展提供的功能类似于 Message Service Client for C/C++ 和 Message Service Client for .NET 中提供的功能。

为什么应该将 WebSphere MQ 类用于 JMS?

使用 WebSphere MQ classes for JMS 具有以下优点:

- 您可以复用 JMS 技能。

  WebSphere MQ classes for JMS 是一个 JMS 提供程序，用于实现 WebSphere MQ 作为消息传递系统的 JMS 接口。 如果您的组织是 WebSphere MQ 的新组织，但已具备 JMS 应用程序开发技能，那么您可能会发现使用熟悉的 JMS API 来访问 WebSphere MQ 资源 (而不是随 WebSphere MQ 提供的其他某个 API) 更容易。

- JMS 是 Java Platform, Enterprise Edition (Java EE) 的组成部分。

  JMS 是用于在 Java EE 平台上进行消息传递的自然 API。 符合 Java EE 的每个应用程序服务器都必须包含一个 JMS 提供程序。 可以在应用程序客户机， Servlet ， JavaServer 页面 (JSP) ，企业 Java Bean (EJB) 和消息驱动的 Bean (MDB) 中使用 JMS。 请特别注意， Java EE 应用程序使用 MDB 异步处理消息，并且所有消息都作为 JMS 消息传递到 MDB。

- 管理员可以在中央存储库中创建和维护 JMS 受管对象，针对 JMS 应用程序的 WebSphere MQ 类可以使用 Java 命名和目录接口 (JNDI) 来检索这些对象。

  JMS 连接工厂和目标封装了特定于 WebSphere MQ 的信息，例如队列管理器名称，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存储为受管对象，那么不会将此信息硬编码到应用程序中。 因此，此安排为应用程序提供了与底层 WebSphere MQ 配置的独立程度。

- JMS 是可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 来检索存储为受管对象的连接工厂和目标，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 WebSphere MQ classes for JMS) ，并且可以从一个 JMS 提供程序移植到另一个 JMS 提供程序，而无需对应用程序进行任何更改。

  如果 JNDI 在特定应用程序环境中不可用，那么 WebSphere MQ JMS 应用程序类可以使用 JMS API 的扩展在运行时动态地创建和配置连接工厂和目标。 然后，应用程序完全独立，但与作为 JMS 提供程序的 JMS 的 WebSphere MQ 类绑定。

- 使用 JMS 编写网桥应用程序可能更容易。

  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。 使用特定于产品的 API 和消息格式编写网桥应用程序可能很复杂。 相反，您可以使用两个 JMS 提供程序 (每个消息传递系统一个 JMS 提供程序) 来编写网桥应用程序。 然后，应用程序仅使用一个 API，即 JMS API，并且仅处理 JMS 消息。

WebSphere MQ JMS 类入门

本主题提供了针对 JMS 的 WebSphere MQ 类的概述，并告诉您在使用针对 JMS 的 WebSphere MQ 类之前需要了解的内容。

WebSphere MQ JMS 类的先决条件

要为 JMS 应用程序开发和运行 WebSphere MQ 类，您需要某些软件组件作为先决条件。

有关 WebSphere MQ classes for JMS 的先决条件的最新信息，请参阅 WebSphere MQ 自述文件。

要为 JMS 应用程序开发 WebSphere MQ 类，您需要 Java 2 软件开发包 (SDK)。 可以在 WebSphere MQ System 需求页面上找到操作系统支持的 JDK 的详细信息。 请参阅 WebSphere MQ 需求。

要针对 JMS 应用程序运行 WebSphere MQ 类，您需要以下软件组件:

- WebSphere MQ 队列管理器
- 针对运行应用程序的每个系统的 Java 运行时环境 (JRE)

如果需要 SSL 连接以使用经 FIPS 140-$tag1 认证的加密模块，那么需要 IBM Java JSSE FIPS 提供程序 (IBMJSSEFIPS)。 每个 IBM Java 2 SDK 和 JRE V 5 或更高版本都包含 IBMJSSEFIPS。

您可以将 WebSphere MQ 类中的 Internet Protocol V 6 (IPv6) 地址用于 JMS 应用程序，前提是 Java 虚拟机 (JVM) 和操作系统上的 TCP/IP 实现支持 IPv6 地址。 WebSphere MQ JMS 管理工具 (请参阅 第 781 页的『使用 WebSphere MQ JMS 管理工具』) 也接受 IPv6 地址。

WebSphere MQ JMS 管理工具和 WebSphere MQ Explorer 使用 Java 命名和目录接口 (JNDI) 来访问存储受管对象的目录服务。 WebSphere MQ JMS 应用程序类还可以使用 JNDI 从目录服务检索受管对象。 服务提供者是一段代码，其通过将 JNDI 调用映射到目录服务调用来提供对目录服务的访问权。 WebSphere MQ JMS 类随附以下服务提供者:

- 文件 ldap.jar 和 providerutil.jar 中的轻量级目录访问协议 (LDAP) 服务提供者。 LDAP 服务提供者提供对基于 LDAP 服务器的目录服务的访问权。
- 文件 fscontext.jar 和 providerutil.jar 中的文件系统服务提供者。 文件系统服务提供者提供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。 尤其是，必须配置 LDAP 服务器以存储 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。

为 HP Integrity NonStop Server 的 IBM WebSphere MQ 客户机准备 JMS 程序

本主题说明在为 IBM WebSphere MQ Client for HP Integrity NonStop Server 开发和运行 JMS 程序之前需要了解的内容。

JMS 的 IBM WebSphere MQ 类作为 IBM WebSphere MQ Client for HP Integrity NonStop Server 安装的一部分进行安装。 有关安装内容摘要的详细信息，请参阅文件系统。

一些客户机功能方面特定于主机操作系统。 有关 IBM WebSphere MQ Client for HP Integrity NonStop Server 的受支持功能的更多信息，请参阅 IBM WebSphere MQ Client for HP Integrity NonStop Server 支持的环境和功能。

先决条件

要构建和运行 JMS 应用程序，必须安装并提供 *HP Integrity NonStop Server for Java* 组件。

设置

有关设置环境以运行和构建可在其中使用 IBM WebSphere MQ JMS 类的应用程序的信息，请参阅 第 609 页的『IBM WebSphere MQ classes for JMS 使用的环境变量』。

有关配置队列管理器以接受来自客户机应用程序的连接时所需步骤的信息，请参阅 第 646 页的『针对 JMS 应用程序的 WebSphere MQ 类的安装后设置』。

有关验证 IBM WebSphere MQ JMS 环境类的信息，请参阅 第 648 页的『 WebSphere MQ classes for JMS 的点到点安装验证测试』。

编写应用程序

有关编写 JMS 应用程序的更多信息，请参阅 第 674 页的『为 JMS 应用程序编写 WebSphere MQ 类』。

有关使用 IBM WebSphere MQ JMS 管理工具的更多信息，请参阅 第 781 页的『使用 WebSphere MQ JMS 管理工具』。

样本

在安装的下列子目录中提供了样本应用程序：`opt/mqm/samp/jms`。

有关运行样本时所需配置步骤的更多信息，请参阅第 88 页的『准备并运行样本程序』。

问题解决

有关解决问题的信息，请参阅第 667 页的『解决 IBM WebSphere MQ JMS 类的问题』。

安装和配置 WebSphere MQ JMS 类

本节描述安装 WebSphere MQ classes for JMS 时创建的目录和文件，并说明在安装后如何为 JMS 配置 WebSphere MQ 类。

相关概念

第 607 页的『为 JMS 的 IBM WebSphere MQ 类安装的内容』
安装 IBM WebSphere MQ classes for JMS 时，将创建许多文件和目录。 在 Windows 上，通过自动设置环境变量在安装期间执行某些配置。 在其他平台上以及在某些 Windows 环境中，必须先设置环境变量，然后才能针对 JMS 应用程序运行 IBM WebSphere MQ 类。

第 615 页的『在 Java 安全管理器下针对 JMS 应用程序运行 WebSphere MQ 类』
WebSphere MQ JMS 类可以在启用 Java 安全管理器的情况下运行。 要在启用安全管理器的情况下成功地运行应用程序，您必须使用合适的策略配置文件来配置 Java 虚拟机 (JVM)。

第 618 页的『IBM WebSphere MQ 资源适配器』
资源适配器允许在应用程序服务器中运行的应用程序访问 IBM WebSphere MQ 资源。 它支持入站和出站通信。

第 646 页的『针对 JMS 应用程序的 WebSphere MQ 类的安装后设置』
本主题告诉您 WebSphere MQ classes for JMS 应用程序需要哪些权限才能访问队列管理器的资源。 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。

第 648 页的『 WebSphere MQ classes for JMS 的点到点安装验证测试』
WebSphere MQ JMS 类随附了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。

第 652 页的『 WebSphere MQ JMS 类的发布/预订安装验证测试』
WebSphere MQ JMS 类随附了发布/预订安装验证测试 (IVT) 程序。 该程序在绑定或客户机方式下连接到队列管理器，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。

第 655 页的『 WebSphere MQ 资源适配器的安装验证测试程序』
IVT 程序以 EAR 文件形式提供。 要使用该程序，必须部署它并将一些对象定义为 JCA 资源。

第 631 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 666 页的『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 WebSphere MQ classes for JMS 中提供了 9 个 OSGi 捆绑软件。

第 667 页的『解决 IBM WebSphere MQ JMS 类的问题』
您可以运行安装验证程序和使用跟踪和日志功能来研究问题。

相关任务

第 657 页的『在 WAS CE 中安装和测试 MQ 资源适配器』
在 WebSphere Application Server CE 中安装 IBM WebSphere MQ 资源适配器并运行安装验证测试 (IVT) 应用程序。

第 659 页的『使用定制 MQ 环境在 WAS CE 上部署 IVT 应用程序』
如果要使用其他队列，队列管理器，端口，主机，通道或使用绑定方式而不是客户机方式，那么在部署资源适配器或 IVT 应用程序之前，必须在 WebSphere Application Server CE 中修改 IVT 应用程序和关联脚本。

第 661 页的『使用定制 IBM WebSphere MQ 环境在 JBoss 中部署 IVT 应用程序』
在 JBoss 中安装 IBM WebSphere MQ 资源适配器时，如果要使用其他队列，队列管理器，端口，主机，通道或使用绑定方式而不是客户机方式，那么必须先修改 JBoss 中的 IVT 应用程序和关联脚本，然后再部署资源适配器或 IVT 应用程序。

IBM WebSphere MQ 资源适配器的问题确定

相关参考

第 665 页的『随 WebSphere MQ classes for JMS 提供的脚本』
提供了许多脚本以帮助执行在使用 WebSphere MQ JMS 类时需要执行的常见任务。

为 JMS 的 IBM WebSphere MQ 类安装的内容

安装 IBM WebSphere MQ classes for JMS 时，将创建许多文件和目录。 在 Windows 上，通过自动设置环境变量在安装期间执行某些配置。 在其他平台上以及在某些 Windows 环境中，必须先设置环境变量，然后才能针对 JMS 应用程序运行 IBM WebSphere MQ 类。

对于大多数操作系统，在安装 IBM WebSphere MQ 时，会将 IBM WebSphere MQ classes for JMS 作为可选组件进行安装。 对于 HP Integrity NonStop Server 的 IBM WebSphere MQ 客户机，缺省情况下会安装 IBM WebSphere MQ JMS 类。 有关安装 IBM WebSphere MQ 的更多信息，请参阅:

安装 WebSphere MQ 服务器
安装 IBM WebSphere MQ 客户机

第 607 页的表 90 显示了在每个平台上安装 JMS 文件的 IBM WebSphere MQ 类的位置。

表 *90: IBM WebSphere MQ JMS* 安装目录类

平台	**目录**
AIX	*MQ_INSTALLATION_PATH*/java
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/java
HP-UX， Linux 和 Solaris	*MQ_INSTALLATION_PATH*/java
Windows	*MQ_INSTALLATION_PATH*\java
MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 的高级目录。	

安装目录包括:

- JMS JAR 文件的 IBM WebSphere MQ 类，它们位于 *MQ_INSTALLATION_PATH*\java\lib 目录中。
- IBM WebSphere MQ 本机库，由使用 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，可以在 *MQ_INSTALLATION_PATH*\java\lib64 目录中找到 64 位本机库。

有关 IBM WebSphere MQ 本机库的更多信息，请参阅 第 611 页的『配置 Java 本机接口 (JNI) 库』。

- 第 665 页的『随 WebSphere MQ classes for JMS 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。
- JMS API 的 IBM WebSphere MQ 类的规范。 Javadoc 工具已用于生成包含 API 规范的 HTML 页面。

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

  在 UNIX， Linux 和 Windows 系统上，此子目录包含各个 HTML 页面。
- 对 OGSi 的支持。 OSGi 捆绑软件安装在 java \lib\OSGi 目录中，并在 第 666 页的『OSGi 支持』中进行了描述。
- IBM WebSphere MQ 资源适配器，可部署到符合 JCA 1.5 (或更高版本) 的任何应用程序服务器中。

  IBM WebSphere 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 618 页的『IBM WebSphere MQ 资源适配器』
- 在 Windows 上，可用于调试的符号安装在 *MQ_INSTALLATION_PATH*\java\lib\symbol 目录中。

安装目录还包含属于其他 IBM WebSphere MQ 组件的一些文件。 这些目录如下:

- IBM WebSphere MQ Transport for SOAP (提供用于 SOAP 的 JMS 传输) 安装在 *MQ_INSTALLATION_PATH*\java\lib\soap 目录中。 有关 IBM WebSphere MQ Transport for SOAP 的更多信息，请参阅信息中心中描述 第 791 页的『WebSphere MQ Transport for SOAP』的部分。
- 在分布式平台上， IBM WebSphere MQ Bridge for HTTP 安装在 *MQ_INSTALLATION_PATH*\java\lib\http 目录中。 有关 IBM WebSphere MQ Bridge for HTTP 的更多信息，请参阅信息中心中描述 第 858 页的『用于 HTTP 的 WebSphere MQ 网桥』 的部分

某些样本应用程序随 IBM WebSphere MQ classes for JMS 一起提供。 第 608 页的表 91 显示样本应用程序在各平台上的安装位置。

表 *91:* 样本目录

平台	**目录**
AIX	*MQ_INSTALLATION_PATH*/samp/jms
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/samp/jms
HP-UX， Linux 和 Solaris	*MQ_INSTALLATION_PATH*/samp/jms
Windows	*MQ_INSTALLATION_PATH*\tools\jms
MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 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 609 页的『IBM WebSphere MQ classes for JMS 使用的环境变量』 描述了为 JMS 应用程序运行简单 IBM WebSphere MQ 类所需的类路径。 本主题还描述了在特殊情况下需要引用的其他 JAR 文件以及必须设置以运行 IBM WebSphere MQ classes for JMS 随附的脚本的环境变量。

如果您需要针对 JMS 应用程序的 IBM WebSphere MQ 类以链接到以非 Java 语言编写的代码 (例如，在连接到队列管理器时使用绑定传输)，那么 第 611 页的『配置 Java 本机接口 (JNI) 库』 将说明在何处查找要指定为 Java 命令参数的 Java 本机接口 (JNI) 库的位置。

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。 第 613 页的『IBM WebSphere MQ classes for JMS 配置文件』中描述了 JMS 配置属性文件的 IBM WebSphere MQ 类。

安装和升级 WebSphere MQ classes for JMS JAR 文件

将 JMS JAR 文件的 IBM WebSphere MQ 类获取到系统上的唯一受支持方法是安装 IBM WebSphere MQ 产品或 WebSphere MQ V7.5 客户机 SupportPac- MQC75，或者使用软件管理工具 (例如 Apache Maven) 来获取更多信息，请参阅 第 614 页的『IBM WebSphere MQ classes for JMS 和软件管理工具』。

请勿将 JMS JAR 文件或本机库的 IBM WebSphere MQ 类移动或复制到其他机器，或者移动或复制到已安装 IBM WebSphere MQ JMS 类的机器上的其他位置，除非您正在使用软件管理工具。

- 修订包无法应用于已从另一台机器复制 JAR 文件的 "安装"，因为这会使确保所有 JAR 文件保持一致并处于兼容级别变得更加困难。
- 在机器之间复制 JMS JAR 文件的 IBM WebSphere MQ 类还会导致驻留在同一机器上的文件的多个副本，这可能会导致为代码提供服务和调试问题。
- dspmqver 命令用于显示来自 IBM WebSphere MQ 安装的版本信息，仅显示安装到 `\java\lib` 目录中的 IBM WebSphere MQ classes for JMS 的版本信息。

  如果文件的多个副本位于同一机器上，那么运行 **dspmqver** 可能无法提供有关应用程序正在使用的 JMS IBM WebSphere MQ 类的版本的准确信息。

请勿在应用程序归档 (例如企业应用程序归档或 EAR 文件) 中包含 JMS JAR 文件的 IBM WebSphere MQ 类。

- 无法使用 IBM WebSphere MQ 修订包来应用对 JMS 的 IBM WebSphere MQ 类的更新。
- IBM 支持人员无法轻松确定应用程序正在使用的 IBM WebSphere MQ JMS 类的版本。
- 如果在同一 Java 运行时环境中运行的多个应用程序包含不同版本的 IBM WebSphere MQ classes for JMS，那么可能会发生问题，因为多个版本的 IBM WebSphere MQ classes for JMS 会同时装入到 Java 运行时环境中。

  这些问题的示例包括以下异常:

```
java.lang.ClassCastException :
com.ibm.mq.jmqi.system.JmqiSystemEnvironment incompatible with
com.ibm.mq.jmqi.system.JmqiSystemEnvironment

java.lang.ClassCastException :
com.ibm.mq.jms.MQQueue incompatible with com.ibm.mq.jms.MQQueue
```

- 如果应用程序使用 BINDINGS 传输来连接到队列管理器，那么队列管理器的任何主要升级还需要更新应用程序以包含相应级别的 IBM WebSphere MQ JMS 类。

  例如，如果将队列管理器升级到 IBM WebSphere MQ 版本 7.5 级别，那么还需要更新使用 BINDINGS 传输连接到队列管理器的任何应用程序以包含 JMS 的 IBM WebSphere MQ 版本 7.5 类。

IBM WebSphere MQ classes for JMS 使用的环境变量

在可以编译和运行针对 JMS 应用程序的 IBM WebSphere MQ 类之前， CLASSPATH 环境变量的设置必须包含针对 JMS Java 归档 (JAR) 文件的 IBM WebSphere MQ 类。 根据需求，您可能需要向类路径中添加其他 JAR 文件。 要运行随 IBM WebSphere MQ classes for JMS 提供的脚本，必须设置其他环境变量。

要为 JMS 应用程序编译和运行 IBM WebSphere MQ 类，请对平台使用 CLASSPATH 设置，如 第 609 页的表 92 中所示。 此设置包含样本目录，以便您可以针对 JMS 样本应用程序编译和运行 IBM WebSphere MQ 类。 或者，可以在 **java** 命令中指定类路径，而不是使用环境变量。

表 *92:* 用于为 *JMS* 应用程序 *(*包括样本应用程序*)* 编译和运行 *IBM WebSphere MQ* 类的 *CLASSPATH* 设置

平台	**CLASSPATH 设置**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP Integrity NonStop Server	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP-UX， Linux 和 Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms:

表 92: 用于为 JMS 应用程序 (包括样本应用程序) 编译和运行 IBM WebSphere MQ 类的 CLASSPATH 设置 (继续)	
平台	**CLASSPATH 设置**
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms;
z/OS	CLASSPATH=*MQ_INSTALLATION_PATH*/mqm/V7R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V6R0M0/java/samples/jms:
MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 的高级目录。	

JAR 文件 com.ibm.mqjms.jar 的清单包含对 JMS 应用程序的 IBM WebSphere MQ 类所需的大部分其他 JAR 文件的引用，因此您无需将这些 JAR 文件添加到类路径中。 这些 JAR 文件包括使用 Java 命名和目录接口 (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：

- 如果您使用的是实现 com.ibm.mq 包中定义的通道出口接口的通道出口类，而不是 com.ibm.mq.exits 包中定义的通道出口类，那么必须将 IBM WebSphere MQ classes for Java JAR 文件 com.ibm.mq.jar 添加到类路径中。
- 如果使用 Java 2 Software Development Kit (SDK) V 1.4.2 编译 Java 代码，那么必须将以下 JAR 文件添加到类路径中：
  - jms.jar
  - com.ibm.mq.jmqi.jar

  此外，如果应用程序使用 JNDI 从目录服务检索受管对象，那么还必须将以下 JAR 文件添加到类路径中：
  - fscontext.jar
  - jndi.jar
  - ldap.jar
  - providerutil.jar

  并且如果应用程序使用 JTA，还必须向类路径中添加 jta.jar。

  请注意，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。

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。 仅当已安装 Extended Security Edition 时，此文件才存在。

IBM WebSphere MQ classes for JMS 随附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 WebSphere MQ classes for JMS 的目录。

MQ_JAVA_LIB_PATH
此环境变量指定存储 JMS 库的 WebSphere MQ 类的目录，如 第 611 页的表 93 中所示。

在 Windows 上，所有环境变量都会在安装期间自动设置。 在任何其他平台上，您必须自己设置这些环境变量。

如果在 UNIX、HP Integrity NonStop Server 或 Linux 系统上使用 32 位 JVM，要设置环境变量，可以使用脚本 setjmsenv。 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 IBM WebSphere MQ 安装所在的高级目录。

可以通过各种方式使用 setjmsenv 或 setjmsenv64 脚本：可以将其用作设置所需环境变量的基础，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该脚本。 如果选择此选项，那么需要在启动的每个 shell 窗口中通过输入 `. ./setjmsenv` 或 `. ./setjmsenv64` 在 JMS 验证过程中运行脚本

配置 *Java* 本机接口 *(JNI)* 库

需要在访问 Java 本机接口 (JNI) 库的环境中运行针对 JMS 应用程序的 IBM WebSphere MQ 类，这些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 Java 以外的语言编写的通道出口程序。

关于此任务

要设置此环境，必须配置环境的库路径，以便 Java 虚拟机 (JVM) 可以在启动 JMS 应用程序的 IBM WebSphere MQ 类之前装入 mqjbnd 库。

IBM WebSphere MQ 提供了两个 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。 它提供了 IBM WebSphere MQ JMS 类与队列管理器之间的接口。 随 IBM WebSphere MQ V 7.5 一起安装的 mqjbnd 库可用于连接到任何 IBM WebSphere MQ V 7.5 (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，mqjexitstub02 库由 IBM WebSphere MQ classes for JMS 装入。

在某些平台上，IBM WebSphere MQ 安装这些 JNI 库的 32 位和 64 位版本。 各平台的库的位置显示在表 1 中

表 *93:* 每个平台的 *JMS* 库的 *IBM WebSphere MQ* 类的位置

平台	**包含 JMS 库的 IBM WebSphere MQ 类的目录**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32 位库) *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
HP-UX	*MQ_INSTALLATION_PATH*/java/lib (32 位库) *MQ_INSTALLATION_PATH*/java/lib64（64 位库）
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）
Linux (x86 平台) Linux (zSeries 平台)	*MQ_INSTALLATION_PATH* /java/lib
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）
MQ_INSTALLATION_PATH 表示安装 IBM WebSphere MQ 的高级目录。	

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 的环境以使 JVM 设置其自己的 `java.library.path`。 此路径因平台以及安装 IBM WebSphere MQ 的位置而异。 例如，对于 64 位 JVM 及缺省 IBM WebSphere MQ 安装位置，可使用以下设置：

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

     Solaris HP-UX 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 JMS 应用程序的 IBM WebSphere MQ 类：

   ```
   java application-name
   ```

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

   包含 IBM WebSphere MQ Reason Code 2495 (MQRC_MODULE_NOT_FOUND) 的异常由 IBM WebSphere MQ classes for JMS 在以下情况下抛出：

   - JMS 应用程序的 IBM WebSphere MQ 类在 32 位 Java 运行时环境中运行，并且已为 JMS 的 IBM WebSphere MQ 类设置 64 位环境，因为 32 位 Java 运行时环境无法装入 64 位 Java 本机库。
   - JMS 应用程序的 IBM WebSphere MQ 类在 64 位 Java 运行时环境中运行，并且已为 JMS 的 IBM WebSphere MQ 类设置 32 位环境，因为 64 位 Java 运行时环境无法装入 32 位 Java 本机库。

IBM WebSphere MQ classes for JMS 配置文件

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

WebSphere MQ classes for JMS 配置文件的格式是标准 Java 属性文件的格式。 在 WebSphere 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。

您可以为 JMS 配置文件选择 WebSphere MQ 类的名称和位置。 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
```

在命令中， *config_file_url* 是统一资源定位符 (URL)，用于指定 WebSphere MQ JMS 配置文件类的名称和位置。 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 WebSphere MQ classes for JMS 配置文件标识为本地 Windows 系统上的文件 D:\mydir\mjms.config 。

当应用程序启动时， WebSphere MQ JMS 类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** 命令未标识配置文件，或者如果找不到配置文件，那么 WebSphere MQ classes for JMS 将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。

针对 JMS 配置文件的 WebSphere MQ 类可以与应用程序与队列管理器或代理之间的任何受支持传输配合使用。

请注意，不能通过在 WebSphere MQ classes for JMS 配置文件中设置属性来指定启动跟踪。 只能通过在 **java** 命令上设置系统属性来指定启动跟踪，如以下示例中所示：

```
java -Dcom.ibm.msg.client.commonservices.trace.startup=true
     -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config
      MyAppClass
```

覆盖 WebSphere MQ MQI 客户机配置文件中指定的属性

WebSphere MQ MQI 客户机配置文件还可以指定用于配置 WebSphere MQ JMS 类的属性。 但是，仅当应用程序以客户机方式连接到队列管理器时， WebSphere MQ MQI 客户机配置文件中指定的属性才适用。

如果需要，您可以覆盖 WebSphere MQ MQI 客户机配置文件中的任何属性，方法是将其指定为 WebSphere MQ classes for JMS 配置文件中的属性。 要覆盖 WebSphere MQ MQI 客户机配置文件中的属性，请在 JMS 配置文件的 WebSphere MQ 类中使用以下格式的条目：

```
com.ibm.mq.cfg.stanza.propName=propValue
```

该条目中的变量具有以下含义：

stanza
WebSphere MQ MQI 客户机配置文件中包含属性的节的名称

propName
WebSphere MQ MQI 客户机配置文件中指定的属性的名称

propValue
用于覆盖 WebSphere MQ MQI 客户机配置文件中指定的属性值的属性值

或者，可以通过在 **java** 命令中将属性指定为系统属性来覆盖 WebSphere MQ MQI 客户机配置文件中的属性。 使用先前格式将属性指定为系统属性。

只有 WebSphere MQ MQI 客户机配置文件中的以下属性与 WebSphere MQ JMS 类相关。 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

IBM WebSphere MQ classes for JMS 和软件管理工具

软件管理工具（如 Apache Maven）可以与 IBM WebSphere 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 WebSphere MQ classes for JMS 包含大量 JAR 文件。 使用此 API 开发 Java 语言应用程序时，需要在开发应用程序的机器上安装 IBM WebSphere MQ Server， Client 或 Client SupportPac 。

如果要使用这样的工具，并将构成 IBM WebSphere 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 WebSphere MQ 发行版或修订包的一组完整且一致的 JAR 文件。
- 您负责使用 IBM 支持人员提供的任何维护来更新存储库。

对于 IBM WebSphere MQ Version 7.5，需要将以下 JAR 文件安装到存储库中：

- com.ibm.mqjms.jar.
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- 如果您正在使用 IBM WebSphere MQ classes for JMS，那么需要 CL3Export.jar 。
- 如果使用的是 IBM WebSphere MQ classes for JMS，那么需要 CL3Nonexport.jar 。
- 如果您正在使用 IBM WebSphere MQ classes for JMS，那么需要 jndi.jar 。
- 如果使用的是 IBM WebSphere MQ classes for JMS，那么需要 ldap.jar 。
- 如果使用的是 IBM WebSphere MQ classes for JMS，那么需要 rmm.jar 。
- 如果您正在使用 IBM WebSphere MQ classes for JMS，那么需要 dhbcore.jar 。
- 如果正在使用 IBM WebSphere MQ classes for JMS，那么需要安装 jms.jar。
- 如果要使用 IBM WebSphere MQ classes for JMS 并访问存储在文件系统 JNDI 上下文中的 JMS 受管对象，那么需要 fscontext.jar 。
- providerutil.jar (如果您正在使用 IBM WebSphere MQ classes for JMS 并访问存储在文件系统 JNDI 上下文中的 JMS 受管对象)。

在 Java 安全管理器下针对 JMS 应用程序运行 WebSphere MQ 类

WebSphere MQ JMS 类可以在启用 Java 安全管理器的情况下运行。 要在启用安全管理器的情况下成功地运行应用程序，您必须使用合适的策略配置文件来配置 Java 虚拟机 (JVM)。

执行此操作的最简单方法是更改随 JRE 提供的策略配置文件。 在大部分系统上，此文件存储在 `lib/security/java.policy` 路径（相对于 JRE 目录）中。 可以使用您喜爱的编辑器或与 JRE 一起提供的 `policytool` 程序来编辑策略配置文件。

要点: V 7.5.0.8 在可能的情况下，术语 *allowlist*（白名单）已替换术语 *whitelist*（白名单）。 一个例外是以下 Java 系统属性名称。

如果将 Java 安全管理器机制与应用程序配合使用，那么必须授予以下许可权:

- 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.whitelist、com.ibm.mq.jms.whitelist.discover 和 com.ibm.mq.jms.whitelist.mode 属性上的 PropertyPermission（读）。

APAR IT14385 和 IBM WebSphere MQ Version 7.5.0 修订包 8 支持 ClassName 允许列表。 有关更多信息，请参阅第 616 页的『JMS ObjectMessage 中的 ClassName 允许列表』。

以下是策略配置文件中允许 WebSphere MQ classes for JMS 在缺省安全管理器下成功运行的两个条目的示例:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //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
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mqjms.jar" {
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "console.encoding","read";
  permission java.lang.RuntimePermission "setContextClassLoader";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.*","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
  permission java.util.logging.LoggingPermission "control";
  //Wherever trace output is expected
  permission java.io.FilePermission "/tmp/*","read,write";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
```

注意:

- *MQ_INSTALLATION_PATH* 表示安装 WebSphere MQ 的高级目录。
- 第一个 `grant` 语句包含 WebSphere MQ classes for JMS 所需的许可权，而第二个 `grant` 语句包含 WebSphere MQ classes for JMS 应用程序所需的许可权。

- 要允许 WebSphere MQ classes for JMS 访问应用程序的 Java 归档 (JAR) 文件，请向第一个 `grant` 语句添加以下许可权:

```
permission java.io.FilePermission "/path_to_your_app/-", "read";
```

- 要在策略配置文件中使用这些 `grant` 语句，您可能需要根据已安装 WebSphere MQ JMS 类的位置以及存储应用程序的位置来修改路径名。
- 随 WebSphere MQ classes for JMS 提供的样本应用程序以及用于运行这些应用程序的脚本不会启用安全管理器。

V 7.5.0.8 JMS ObjectMessage 中的 ClassName 允许列表

V 7.5.0.8 在 WebSphere MQ classes for JMS 中，对 JMS ObjectMessage 接口实现中类的允许列表的支持提供了针对可能与 Java 对象序列化和反序列化机制相关的某些安全风险的潜在缓解。

注: 在可能的情况下，术语 *allowlist*（白名单）已替换术语 *whitelist*（白名单）。 一个例外是本主题中提及的 Java 系统属性名称。

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是在使用序列化来封装和传输任意对象的 Java 消息服务 (JMS) ObjectMessages 中。

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。

WebSphere MQ JMS 类中的允许列表

通过 APAR IT14385 和 IBM WebSphere MQ Version 7.5.0 修订包 8， WebSphere MQ classes for JMS 支持 JMS ObjectMessage 接口实现中类的允许列表。 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: WebSphere MQ JMS 类不能与允许列表一起分发。 使用 ObjectMessages 传输哪些类是由应用程序设计进行选择， IBM WebSphere 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.whitelist
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，采取文件 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。

如果您正在使用 Java 安全管理器，那么必须确保用于 JMS JAR 文件的 WebSphere MQ 类具有对此文件的读写访问权。

com.ibm.mq.jms.whitelist.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。
- 如果此属性设为 true，并且未使用 com.ibm.mq.jms.whitelist 属性指定允许列表，那么允许列表机制为非活动状态。
- 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.whitelist.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：

- 允许列表文件使用缺省平台文件编码且具有与平台对应的行尾。

  注: 如果在异构系统之间移动文件，需要进行转换。
- 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。

  所以 `com.ibm.mq.*` 匹配 `com.ibm.mq.MQMessage`，而不匹配 `com.ibm.mq.jmqi.remote.api.RemoteFAP`。

  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“*”。
- 格式不当的允许列表文件，例如，包含 `com.ibm.mq.*.Message` 之类条目的文件，其中通配符不是最后一个元素，将导致抛出 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。 如果将允许列表指定为以逗号分隔的类名列表，那么假定操作系统或 shell 不处理它，因为它是许多 UNIX 或 Linux shell 中的缺省注释字符，那么会将它视为正常字符。

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，会将一个条目写入应用程序的 IBM WebSphere 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 IBM WebSphere MQ 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。

如果将 Java 安全管理器机制与应用程序配合使用，那么必须授予以下许可权:

- 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.whitelist、com.ibm.mq.jms.whitelist.discover 和 com.ibm.mq.jms.whitelist.mode 属性上的 PropertyPermission（读）。

相关概念

第 615 页的『在 Java 安全管理器下针对 JMS 应用程序运行 WebSphere MQ 类』
WebSphere MQ JMS 类可以在启用 Java 安全管理器的情况下运行。 要在启用安全管理器的情况下成功地运行应用程序，您必须使用合适的策略配置文件来配置 Java 虚拟机 (JVM)。

IBM WebSphere MQ 资源适配器

资源适配器允许在应用程序服务器中运行的应用程序访问 IBM WebSphere MQ 资源。 它支持入站和出站通信。

Java Platform, Enterprise Edition (Java EE) 连接器体系结构 (JCA) 提供了将 Java EE 环境中运行的应用程序连接到企业信息系统 (EIS) (例如 IBM WebSphere MQ 或 Db2) 的标准方法。 IBM WebSphere MQ 资源适配器实现 JCA 1.5 接口并包含 IBM WebSphere 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的 Bean (MDB) 访问 IBM WebSphere 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 WebSphere 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，然后将 JMS 消息发送到 JMS 目标，并以同步方式从 JMS 目标接收 JMS 消息。

入站通信
到达 JMS 目标的 JMS 消息将传递到 MDB ，该 MDB 以异步方式处理该消息。

有关 IBM WebSphere MQ classes for JMS 的更多信息，请参阅第 603 页的『使用 WebSphere MQ classes for JMS』。

资源适配器还包含 IBM WebSphere 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 WebSphere MQ 队列管理器的资源时使用 IBM WebSphere MQ classes for Java API。 有关 IBM WebSphere MQ classes for Java 的更多信息，请参阅 第 552 页的『使用 WebSphere MQ classes for Java』。

支持在 Java EE 环境中使用 IBM WebSphere MQ classes for Java ，但存在限制。 有关这些限制的信息，请参阅第 602 页的『在 Java 平台 Enterprise Edition 中运行 Java 应用程序的 IBM WebSphere MQ 类』。

支持 *JCA* 资源适配器所需的其他文档

有关如何配置 JCA 资源适配器的信息，请参阅应用程序服务器的文档。

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，但其他应用程序服务器则要求管理员编写 XML 部署计划。 因此，提供有关如何为每个应用程序服务器配置 WebSphere MQ 资源适配器的信息超出了本文档的范围。 本文档仅关注您需要配置的内容。 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。

要了解此文档，您必须熟悉 JMS 和 WebSphere MQ JMS 类。 用于配置 WebSphere MQ 资源适配器的许多属性等同于 WebSphere MQ JMS 对象类的属性，并且具有相同的功能。

安装 *WebSphere MQ* 资源适配器

WebSphere 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。

WebSphere MQ 资源适配器作为名为 wmq.jmsra.rar 的资源归档 (RAR) 文件提供。 此文件随 WebSphere MQ classes for JMS 一起安装在 第 619 页的表 94 中显示的目录中。

表 *94:* 各平台的包含 *wmq.jmsra.rar* 的目录

平台	目录
AIX， HP-UX， Linux 和 Solaris	*MQ_INSTALLATION_PATH* /java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH* \java\lib\jca
MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。	

RAR 文件包含 WebSphere MQ JMS 类以及 JCA 接口的 WebSphere MQ 实现。

必须在应用程序服务器中安装 WebSphere MQ 资源适配器 RAR 文件，但执行此操作的方式取决于应用程序服务器。 有关如何安装资源适配器 RAR 文件的信息，请参阅应用程序服务器的文档。

对于 UNIX and Linux 系统上的绑定连接，必须确保包含 Java 本机接口 (JNI) 库的目录位于系统路径中。 有关此目录 (其中还包含用于 JMS 库的 WebSphere MQ 类) 的位置，请参阅 第 611 页的表 93。 在 Windows 上，在安装 WebSphere MQ classes for JMS 期间，会自动将此目录添加到系统路径中。

事务在客户机和绑定方式下均受支持。

WebSphere MQ 资源适配器与资源适配器所使用的 WebSphere MQ JMS 类的版本必须处于同一发行版级别。

WebSphere Application Server 和 *WebSphere MQ* 资源适配器

请勿将 WebSphere MQ 资源适配器与 WebSphere Application Server V 6 配合使用。 WebSphere Application Server V7 包含 WebSphere MQ V7 资源适配器的版本。

请勿在 WebSphere Application Server V6 中使用 WebSphere MQ 资源适配器。 要从 JMS 应用程序中的 WebSphere Application Server 内访问 WebSphere MQ 队列管理器的资源，请使用 WebSphere MQ 消息传递提供程序。 WebSphere MQ 消息传递提供程序包含 WebSphere MQ JMS 类的版本。

WebSphere Application Server V7 包含 WebSphere MQ V7 资源适配器的版本。

有关更多信息，请参阅 Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ?。

WebSphere Application Server Liberty 和 *IBM WebSphere MQ* 资源适配器

可以使用 wmqJmsClient-1.1 功能部件将 IBM WebSphere MQ Version 7.5 资源适配器安装到 WebSphere Application Server Liberty V 8.5.5 修订包 2 或更高版本中。 或者，根据某些限制，可以使用通用 Java Platform， Enterprise Edition 连接器体系结构 (Java EE JCA) 支持来安装资源适配器。

将资源适配器安装到 Liberty 时的一般限制

使用 wmqJmsClient-1.1 功能部件时以及使用通用 JCA 支持时，以下限制适用于 Version 7.5 资源适配器:

- Liberty 中不支持 IBM WebSphere MQ classes for Java 。 它们不得与 IBM WebSphere MQ Liberty 消息传递功能部件或通用 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 WebSphere MQ 资源适配器具有传输类型 BINDINGS_THEN_CLIENT。 此传输类型在 IBM WebSphere MQ Liberty 消息传递功能部件中不受支持。

- IBM WebSphere MQ Advanced Message Security (IBM WebSphere MQ AMS) 功能部件未包含在 IBM WebSphere MQ Liberty 消息传递功能部件中。

IBM WebSphere MQ Version 7.5 资源适配器不能与 wmqJmsClient-2.0 功能部件配合使用。

WebSphere MQ 资源适配器的配置

要配置 WebSphere MQ 资源适配器，请定义各种 JCA 资源和系统属性。

定义下列类别的 JCA 资源：

- ResourceAdapter 对象的属性，它们表示资源适配器的全局属性，例如诊断跟踪级别。 第 620 页的『ResourceAdapter 对象的配置』中描述了这些属性。
- ActivationSpec 对象的属性，它们确定如何为入站通信激活 MDB。 第 621 页的『配置入站通信的资源适配器』中描述了这些属性。
- ConnectionFactory 对象的属性，应用程序服务器使用此对象为出站通信创建 JMS ConnectionFactory 对象。 第 631 页的『配置出站通信的资源适配器』中描述了这些属性。
- 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 队列对象或 JMS 主题对象。 第 631 页的『配置出站通信的资源适配器』中也描述了这些属性。

WebSphere MQ 资源适配器 RAR 文件包含名为 META-INF/ra.xml 的文件，该文件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://java.sun.com/xml/ns/j2ee/connector_1_5.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。 例如， WebSphere Application Server Community Edition 需要名为 geronimo-ra.xml 的部署计划。

如果要使用安全套接字层 (SSL)，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。 这些属性适用于整个 JVM，因此在应用程序服务器中运行的其他应用程序使用 SSL 连接时，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 SSL 与 WebSphere MQ classes for JMS 配合使用的更多信息，请参阅 第 759 页的『将安全套接字层 (SSL) 与 WebSphere MQ JMS 类配合使用』。

WebSphere MQ 资源适配器随附了安装验证测试 (IVT) 程序，但必须先配置资源适配器，然后才能运行该程序。 有关为运行 IVT 程序而需要进行的配置的信息，请参阅第 655 页的『 WebSphere MQ 资源适配器的安装验证测试程序』。

资源适配器日志，警告和错误消息使用与 IBM WebSphere MQ JMS 类相同的机制，有关详细信息，请参阅第 667 页的『日志记录和 IBM WebSphere MQ classes for JMS』。 对于 WebSphere Application Server，这些消息将自动重定向到应用程序服务器输出日志。 对于其他应用程序服务器 (例如 WAS CE 和 JBoss)，缺省情况下，这些应用程序服务器将转至名为 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
```

有关如何设置 JVM 系统属性的详细信息，请参阅应用程序服务器文档。

ResourceAdapter 对象的配置

ResourceAdapter 对象封装了 WebSphere MQ 资源适配器的全局属性。 使用资源适配器的功能来定义这些属性。

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。

有关定义与诊断跟踪关联的属性的更多信息，请参阅 跟踪 IBM WebSphere MQ 资源适配器。

资源适配器管理用于将消息传递到 MDB 的 JMS 连接的内部连接池。 第 621 页的表 95 列出与连接池关联的 ResourceAdapter 对象的属性。

表 *95:* 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 WebSphere MQ 队列管理器的最大连接数以及已部署的最大 MDB 数。
connectionConcurrency	字符串	1	要共享 JMS 连接的 MDB 的最大数目。 无法共享连接，并且此属性始终具有值 1。
reconnectionRetryCount	字符串	5	资源适配器在连接失败时尝试重新连接到 WebSphere MQ 队列管理器的最大次数。
reconnectionRetryInterval	字符串	300 000	资源适配器在尝试重新连接到 WebSphere MQ 队列管理器之前等待的时间 (以毫秒计)。
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）。

在应用程序服务器中部署 MDB 时，将创建新的 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。

reconnectionRetry 计数和 reconnectionRetry 时间间隔属性用于管理与 WebSphere 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
```

如果尚未启动资源适配器 (例如，当使用 WebSphere MQ 资源的应用程序仅在客户机容器中运行时)，那么这些属性无效。 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。

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 WebSphere 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 WebSphere MQ 队列管理器的 JMS 连接的属性
- 用于创建 JMS 连接使用者的属性，这些属性在消息到达指定队列时以异步方式传递消息

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。

第 622 页的表 96 列出了用于创建与 WebSphere MQ 队列管理器的 JMS 连接的 ActivationSpec 对象的属性。

表 *96:* 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 (其中该字段称为 **APPLTAG**) 或 IBM WebSphere MQ 资源管理器 **应用程序连接** 屏幕 (其中该字段称为 **App name**) 显示。
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 WebSphere MQ 发布/预订代理。 或者，在兼容性方式下使用 WebSphere MQ Integrator， WebSphere Event Broker，WebSphere Business Integration Event Broker 或 WebSphere Business Integration Message Broker 的代理。 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 WebSphere MQ Integrator， WebSphere Event Broker，WebSphere Business Integration Event Broker 或 WebSphere Business Integration Message Broker 的代理。 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

表 96: 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
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
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多个实例可以同时运行，但每个实例必须在单独的 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。	用于入站通信的 TCP/IP 连接名称列表。 指定时，**connectionNameList** 将取代 **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

表 96: 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**
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。 **hostname** 和 **port** 属性在指定时将被 **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。
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	确定消息选择是由 WebSphere MQ classes for JMS 完成还是由代理完成。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

表 96: 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**
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。 **hostname** 和 **port** 属性在指定时将被 **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。
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** • 由一个或多个以逗号分隔的项组成的字符串，其中每个项都是实现 WebSphere 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

表 96: 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**
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， WebSphere MQ JMS 类将在 WebSphere MQ 队列中按队列的 *MsgDeliverySequence* 属性确定的顺序搜索适合的消息。 当 WebSphere MQ classes for JMS 找到合适的消息并将其传递给使用者时， WebSphere MQ classes for JMS 将从其在队列中的当前位置继续搜索下一条合适的消息。 WebSphere MQ classes for JMS 将以这种方式继续搜索队列，直到它到达队列末尾，或者直到此属性的值所确定的时间间隔 (以毫秒为单位) 已到期为止。 在每种情况下， WebSphere MQ JMS 类将返回到队列的开头以继续其搜索，新的时间间隔将开始。
securityExit [3]	字符串	• **null** • 实现 WebSphere 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** • 由一个或多个以逗号分隔的项组成的字符串，其中每个项都是实现 WebSphere 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
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与从同一进程到同一队列管理器的其他顶级 JMS 连接共享其套接字
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

表 96: 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
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，这些服务器拥有用于 SSL 连接的证书撤销列表（CRL）
sslCipherSuite	字符串	• **null** • CipherSuite 的名称	要用于 SSL 连接的 CipherSuite
sslFipsRequired [2]	布尔	• **false** • true	SSL 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 SSL 连接，它是一个模板，用于检查由队列管理器提供的数字证书中的专有名称
sslResetCount	int	• **0** • 范围 0 - 999 999 999 中的整数	重新协商 SSL 所使用的密钥之前，通过 SSL 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 WebSphere MQ JMS 类存储有关活动预订的持久数据的位置
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进行客户机方式连接。

表 96: 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**
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 WebSphere MQ for JMS 类的 V 7.0 配合使用。 它不会影响连接到 V 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 645 页的『IBM WebSphere 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 764 页的『配置 IBM WebSphere MQ classes for JMS 以使用通道出口』。

第 628 页的表 97 列出了用于创建 JMS 连接使用者的 ActivationSpec 对象的属性。

表 97: 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的值。
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。 仅当应用程序在 WebSphere Application Server V 7 或更高版本上运行时，此参数才有效。
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 是非事务性的，那么消息传递是否在 WebSphere 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

表 97: 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
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）
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。
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 97: 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** -将名为 destination 的属性解释为 WebSphere 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

必须显式定义名为 destination 和 destinationType 的 ActivationSpec 属性。 所有其他属性都是可选的。

ActivationSpec 对象可能具有冲突的属性。 例如，您可以在绑定方式下为连接指定 SSL 属性。 在此情况下，行为由传输类型和消息传递域（如 destinationType 属性所确定的点到点或发布/预订）确定。 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，指示要使用持久预订，那么还必须定义 subscriptionName 属性。

如果名为 ccdtURL 和 channel 的属性均已定义，那么将抛出 InvalidPropertyException 异常。 但是，如果仅定义 ccdtURL 属性，那么会保留名为 channel 的属性的缺省值 SYSTEM.DEF.SVRCONN，不抛出异常，并且使用由 ccdtURL 属性标识的客户机通道定义表来启动 JMS 连接。

ActivationSpec 对象的大部分属性等同于 WebSphere MQ classes for JMS 对象的属性或 WebSphere MQ classes for JMS 方法的参数。 但是，三个调整属性和一个易用性属性在 WebSphere MQ JMS 类中没有等效属性:

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 628 页的表 97。

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 619 页的『安装 WebSphere 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。

资源适配器中的入站有害消息处理

在某些情况下，传递到 MDB 的消息可能会回滚到 WebSphere MQ 队列。 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 WebSphere MQ ， 以便 WebSphere MQ classes for JMS 自动将有害消息传输到另一个队列以进行进一步调查或废弃该消息。

有关如何处理有害消息的详细信息，请参阅第 745 页的『在 IBM WebSphere 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

相关参考

适用于 UNIX， Linux 和 Windows 的联邦信息处理标准 (FIPS)

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。

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();
}
```

当 servlet 接收 HTTP GET 请求时，它从 JNDI 名称空间检索 ConnectionFactory 对象和 Queue 对象，并使用这些对象向 WebSphere MQ 队列发送消息。 然后，servlet 接收其已发送的消息。

要配置出站通信，请定义下列类别的 JCA 资源：

- ConnectionFactory 对象的属性，应用程序服务器使用该对象来创建 JMS ConnectionFactory 对象。
- 应用程序服务器用于创建 JMS 队列对象或 JMS 主题对象的受管目标对象的属性。

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。 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。

第 632 页的表 98 列出 ConnectionFactory 对象的属性。

表 *98: 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 (其中该字段称为 **APPLTAG**) 或 IBM WebSphere MQ 资源管理器 **应用程序连接** 屏幕 (其中该字段称为 **App name**) 显示。
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。

表 *98: 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**
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 WebSphere MQ 发布/预订代理程序。 或者，以兼容性方式使用 IBM WebSphere MQ Integrator , WebSphere Event Broker , WebSphere Business Integration Event Broker 或 WebSphere Business Integration Message Broker 的代理。 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 WebSphere MQ Integrator , WebSphere Event Broker , WebSphere Business Integration Event Broker 或 WebSphere Business Integration Message Broker 的代理。 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。
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。

表 98: *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
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。 **hostname** 和 **port** 属性在指定时将被 **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。

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	确定消息选择是由 IBM WebSphere MQ JMS 类完成，还是由代理完成。 当 brokerVersion 值为 1 时，不支持由代理程序进行消息选择。

表 *98: 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**
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。 **hostname** 和 **port** 属性在指定时将被 **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。	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 WebSphere 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 WebSphere 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。

表 98: *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**
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，WebSphere MQ JMS 类将按照队列的 *MsgDeliverySequence* 属性所确定的顺序在 IBM WebSphere MQ 队列中搜索合适的消息。 当 WebSphere MQ classes for JMS 找到合适的消息并将其传递给使用者时， WebSphere MQ classes for JMS 将从其在队列中的当前位置继续搜索下一条合适的消息。 WebSphere MQ classes for JMS 将以这种方式继续搜索队列，直到它到达队列末尾，或者直到此属性的值所确定的时间间隔 (以毫秒为单位) 已到期为止。 在每种情况下，WebSphere MQ JMS 类将返回到队列的开头以继续其搜索，新的时间间隔将开始。
securityExit [3]	字符串	• **null** • 实现 WebSphere 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。
sendCheckCount	int	• **0** • 任何正整数	在单个非事务性 JMS 会话中，在检查异步放置错误之间允许的发送调用数。
sendExit [3]	字符串	• **null** • 由一个或多个以逗号分隔的项组成的字符串，其中每个项都是实现 WebSphere 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与从同一进程到同一队列管理器的其他顶级 JMS 连接共享其套接字。
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。

表 98: *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
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。	拥有要在 SSL 连接时使用的证书撤销列表 (CRL) 的轻量级目录访问协议 (LDAP) 服务器。
sslCipherSuite	字符串	• **null** • CipherSuite 的名称	要用于 SSL 连接的密码套件。
sslFipsRequired [2]	布尔	• **false** • true	SSL 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 SSL 连接，这是用来检查由队列管理器提供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	重新协商 SSL 所使用的密钥之前，通过 SSL 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	确定用于 JMS 的 WebSphere MQ 类存储有关活动预订的持久数据的位置。
targetClientMatching	布尔	• **true** • false	是否仅当入局消息具有 MQRFH2 头时，发送至由该入局消息的 JMSReplyTo 头字段标识的队列的应答消息才具有 MQRFH2 头。

表 98: *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	从中创建 JMS 临时队列的模型队列的名称。 如果以下两种情况均成立，那么使用 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 WebSphere MQ 动态队列的名称的前缀。 构成前缀的规则与构成 IBM WebSphere 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 WebSphere MQ classes for JMS 将使用值 AMQ.* 创建动态队列时。	用于构成 IBM WebSphere MQ 动态队列名称的前缀。

表 *98: 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 WebSphere 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，用于为此连接下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。
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 V 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 645 页的『IBM WebSphere 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 764 页的『配置 IBM WebSphere 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
```

第 639 页的表 99 列出 Queue 对象和 Topic 对象的公共属性。

表 *99: 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。

表 99: *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
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。
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 WebSphere MQ 队列的 *DefPersistence* 属性确定。 • PERS - 消息持久。 • NON - 消息不持久。 • HIGH-根据 第 758 页的『JMS 持久消息』 中的说明，消息的持久性由 WebSphere 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 WebSphere 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 *99: 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 WebSphere MQ 应用程序。	发送到目标的消息的目标是否为 JMS 应用程序。 具有作为 JMS 应用程序的目标的消息包含 MQRFH2 头。

第 641 页的表 100 列出特定于 Queue 对象的属性。

表 *100:* 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 WebSphere 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 WebSphere MQ 队列的名称。

第 641 页的表 101 列出特定于 Topic 对象的属性。

表 *101:* 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。

表 *101:* 特定于 *Topic* 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
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。 有关更多信息，请参阅 WebSphere MQ Explorer 文档中的 BROKEDURRSUBQ 属性。
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）。 此属性的值覆盖 ConnectionFactory 对象的 brokerPubQueue 属性的值。但是，如果不设置此属性的值，那么将改用 ConnectionFactory 对象的 brokerPubQueue 属性的值。
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。此属性的值覆盖 ConnectionFactory 对象的 brokerVersion 属性的值。 但是，如果不设置此属性的值，那么将改用 ConnectionFactory 对象的 brokerVersion 属性的值。

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 V 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

相关参考

适用于 UNIX， Linux 和 Windows 的联邦信息处理标准 (FIPS)

V 7.5.0.9 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 WebSphere 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 传统版和 WebSphere Application Server Liberty 中为激活规范配置此属性。

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 WebSphere 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 传统版中，使用管理控制台将 **targetClientMatching** 属性定义为 IBM WebSphere 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 Application Server 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 736 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) 来标识 WebSphere MQ 队列或主题，并 (可选) 指定 Queue 或 Topic 对象的一个或多个属性。

第 631 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。

ASF 和非 *ASF* 方式
应用程序服务器设施 (ASF) 方式是 WebSphere Application Server 中的消息侦听器服务用于处理消息的缺省方法。

消息侦听器服务具有两种操作方式: 应用程序服务器设施 (ASF) 和非应用程序服务器设施 (非 ASF):

- ASF 方式为应用程序提供并行和事务支持。 对于发布/预订消息驱动的 Bean， ASF 方式提供了更好的吞吐量和并行性，因为在非 ASF 方式下，侦听器是单线程的。
- 非 ASF 方式主要用于不支持 JMS ASF 的第三方消息传递提供程序，这是 JMS 规范的可选扩展。 非 ASF 方式也是事务性的，但由于路径长度比 ASF 方式短，因此通常可以提高性能。

要对应用程序服务器上的所有消息驱动的 bean 侦听器启用非 ASF 操作方式，请将此属性设置为非零值。

注:

无法在 z/OS 系统上选择非 ASF 方式，因此在此情况下不得为此属性设置非零值。

ASF 方式下的消息处理

在 ASF 方式下，仅当检测到适合消息驱动的 bean (MDB) 的消息时，才会为工作分配服务器会话和线程。 MDB 可并发处理的线程数由侦听器端口或激活规范的 **Maximum Sessions** 属性值确定。

非 ASF 方式下的消息处理

在非 ASF 方式下，从启动侦听器端口或激活规范时开始，线程处于活动状态。 活动线程数由为 **Maximum Sessions** 属性指定的值指定。 **Maximum Sessions** 属性中指定的线程数处于活动状态，而不考虑可处理的消息数。 每个活动线程都是单独的物理网络连接。

IBM WebSphere MQ V 7.0 或更高版本允许最多 10 个线程共享单个物理网络连接。

相关概念

IBM WebSphere MQ JMS 应用程序服务器设施的类
本主题描述了 WebSphere MQ JMS 类如何在 Session 类中实现 ConnectionConsumer 类和高级功能。 同时还概述了服务器会话池的功能。

相关任务

配置非 ASF 方式的激活规范
激活规范是管理和配置在 WebSphere Application Server 中运行的消息驱动的 bean (MDB) 与 IBM WebSphere MQ 中的目标之间的关系的标准化方法。 此任务说明如何将 WebSphere Application Server 配置为使用非 ASF 方式来处理消息。

相关信息

ASF 方式和非 ASF 方式下的消息处理

配置非 *ASF* 方式的激活规范
激活规范是管理和配置在 WebSphere Application Server 中运行的消息驱动的 bean (MDB) 与 IBM WebSphere MQ 中的目标之间的关系的标准化方法。 此任务说明如何将 WebSphere Application Server 配置为使用非 ASF 方式来处理消息。

开始之前

您定义激活规范属性的方式取决于应用程序服务器提供的管理接口。 此任务假定您正在将 WebSphere Application Server V 7 或更高版本用作应用程序服务器，并将 IBM WebSphere MQ 用作消息传递提供程序。

注:

无法在 z/OS 系统上选择非 ASF 方式。

关于此任务

激活规范的属性确定消息驱动 Bean (MDB) 如何从 IBM WebSphere MQ 队列接收 JMS 消息。 要配置非 ASF 方式，请定义一个或多个激活规范的属性。

您可以在非 ASF 方式下使用多个 IBM WebSphere MQ 配置。 通过以下配置，每个线程使用单独的物理网络连接:

- IBM WebSphere MQ V 7.x 队列管理器，使用将提供程序版本属性设置为 6 的连接工厂。
- IBM WebSphere MQ V 7.x 队列管理器，使用将提供程序版本属性设置为 7 或未指定的连接工厂，通过将 **SHARECNV** (共享对话) 参数设置为 0 的 IBM WebSphere MQ 通道进行连接。

要配置非 ASF ，请将 ActivationSpec 属性 **NON.ASF.RECEIVE.TIMEOUT** 设置为正整数，指示使用非 ASF 交付。 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。 有关更多详细信息，请参阅 **消息侦听器服务定制属性**。

仅当应用程序在 WebSphere Application Server V 7 或更高版本上运行时，此参数才有效。

过程

1. 启动 WebSphere Application Server 管理控制台。
2. 显示侦听器服务设置页面:
   a) 在导航窗格中，选择 **服务器> 服务器类型> WebSphere 应用程序服务器**。
   b) 在内容窗格中，单击应用程序服务器的名称。
   c) 在 " **通信**" 下，单击 **消息传递> 消息侦听器服务**。
3. 将定制属性 **NON.ASF.RECEIVE.TIMEOUT** 设置为消息侦听器服务的定制属性。
   a) 单击**定制属性**。
   b) 单击**新建**。
   c) 在 **名称** 字段中输入属性 **NON.ASF.RECEIVE.TIMEOUT** 的名称。
   d) 在 **值** 字段中输入所需的值。
   e) 单击**确定**。
4. 将更改保存到主配置。
5. 要激活已更改的配置，请停止然后重新启动应用程序服务器。

结果

您已将 WebSphere Application Server 的消息侦听器服务的属性配置为使用非 ASF 方式。

注: 使用非 ASF 方式时，必须确保允许在达到总事务生存期超时之前完成足够的处理时间，以避免不需要的事务超时。 有关更多详细信息，请参阅 WebSphere Application Server 产品文档中的 **NON.ASF.RECEIVE.TIMEOUT** 。

相关概念

第 643 页的『ASF 和非 ASF 方式』
应用程序服务器设施 (ASF) 方式是 WebSphere Application Server 中的消息侦听器服务用于处理消息的缺省方法。

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。

相关信息

消息驱动的 bean
消息侦听器服务
ASF 方式和非 ASF 方式下的消息处理
如何以非 ASF 方式处理消息

IBM WebSphere MQ 资源适配器的限制

在使用 IBM WebSphere MQ 资源适配器时，IBM WebSphere MQ 的一些功能将不可用或受限。

IBM WebSphere MQ 资源适配器具有以下限制:

- IBM WebSphere MQ 资源适配器在所有 IBM WebSphere MQ 平台上受支持，但 z/OS 除外。
- IBM WebSphere MQ 资源适配器不支持与代理程序的实时连接。 它仅支持以客户机或绑定方式连接到 IBM WebSphere MQ 队列管理器。
- IBM WebSphere MQ 资源适配器不支持以 Java 以外的语言编写的通道出口程序。
- 当应用程序服务器在运行时，对于所有 JCA 资源，sslFipsRequired 属性值必须为 true 或 false。 即使未并发使用 JCA 资源，这也是必需的。 如果 sslFipsRequired 属性对不同 JCA 资源具有不同的值，那么 IBM WebSphere MQ 将发出原因码 MQRC_UNSUPPORTED_CIPHER_SUITE，即使未在使用 SSL 连接亦如此。
- 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 JEE 容器 (EJB/Servlet) 中运行时，出站连接不支持 IBM WebSphere MQ Version 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。

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 的联邦信息处理标准 (FIPS)

针对 JMS 应用程序的 WebSphere MQ 类的安装后设置

本主题告诉您 WebSphere MQ classes for JMS 应用程序需要哪些权限才能访问队列管理器的资源。 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 WebSphere MQ 自述文件。 该文件可能包含可取代此主题中信息的信息。

JMS 使用的需要非特权用户授权的对象

非特权用户需要获得授权才能访问 JMS 所使用的队列。 每个 JMS 应用程序都需要对其工作的队列管理器进行授权。

有关 IBM WebSphere MQ 中的访问控制的详细信息，请参阅 在 Windows UNIX and Linux 系统上设置安全性。

WebSphere MQ classes for JMS 应用程序需要对队列管理器具有 `connect` 和 `inq` 权限。 您可以使用 **setmqaut** 控制命令设置相应的权限，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 WebSphere MQ JMS 类以 IBM WebSphere MQ 消息传递提供程序迁移方式连接到 IBM WebSphere 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 WebSphere MQ 消息传递提供程序迁移方式的更多信息，请参阅何时使用 **PROVIDERVERSION**

此外，如果 WebSphere MQ JMS 类以此方式连接到队列管理器，那么任何发布消息的应用程序都需要访问 TopicConnection 工厂或主题对象所指定的流队列。 缺省情况下，此队列为 SYSTEM.BROKER.DEFAULT.STREAM。

如果使用 ConnectionConsumer， IBM WebSphere MQ Resource Adapter 或 WebSphere Application Server IBM WebSphere MQ 消息传递提供程序，那么可能需要其他授权。

要由 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 WebSphere MQ 消息传递提供程序正常方式执行发布/预订消息传递时，应用程序将使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。

WebSphere MQ JMS 类的连接方式

用于 JMS 应用程序的 WebSphere MQ 类可以客户机方式或绑定方式连接到队列管理器。 在客户机方式下，WebSphere MQ JMS 类通过 TCP/IP 连接到队列管理器。 在绑定方式下， WebSphere MQ JMS 类使用 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 WebSphere MQ classes for JMS 的当前或任何较早受支持版本与当前队列管理器配合使用，并且可以将队列管理器的当前或较早受支持版本与 WebSphere MQ classes for JMS 的当前版本配合使用。 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

要以客户机方式连接到队列管理器，针对 JMS 应用程序的 WebSphere MQ 类可以在运行队列管理器的同一系统上运行，也可以在其他系统上运行。 在每种情况下， WebSphere MQ JMS 类都通过 TCP/IP 连接到队列管理器。

绑定方式

要以绑定方式连接到队列管理器，针对 JMS 应用程序的 WebSphere MQ 类必须在运行队列管理器的同一系统上运行。

WebSphere MQ JMS 类使用 Java 本机接口 (JNI) 直接连接到队列管理器。 要使用绑定传输，必须在有权访问 WebSphere MQ Java 本机接口库的环境中运行用于 JMS 的 WebSphere MQ 类; 请参阅 第 611 页的『配置 Java 本机接口 (JNI) 库』 以获取更多信息。

WebSphere MQ classes for JMS 支持 *ConnectOption* 的以下值:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

要更改 WebSphere 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 175 页的『使用 MQCONNX 调用连接到队列管理器』

要使用绑定传输，所使用的 Java 运行时环境必须支持 JMS 的 WebSphere MQ 类所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 使用 WebSphere MQ V7 classes for Java 或 WebSphere MQ V7 classes for JMS 时生成的具有探测器标识 21 的 WebSphere MQ FDC 中找到。

绑定，然后是客户机方式

这是缺省值。 在此方式下连接到队列管理器时， WebSphere MQ classes for JMS 应用程序将尝试以绑定方式进行连接，这要求队列管理器与应用程序位于同一机器上。 如果连接失败，那么应用程序将尝试以客户机方式进行连接，从而允许队列管理器与应用程序位于同一机器上本地或远程。

配置队列管理器，以便用于 *JMS* 应用程序的 *WebSphere MQ* 类可以客户机方式连接

要配置队列管理器以便用于 JMS 应用程序的 WebSphere MQ 类可以客户机方式连接，必须创建服务器连接通道定义并启动侦听器。

在 z/OS 上，必须安装 "客户机连接" 功能部件。

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)
```

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。

您还可以使用 IBM WebSphere MQ Explorer (在 Linux 和 Windows 上运行) 或 z/OS 上的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）。

On all platforms, you can use the MQSC command START LISTENER to start a listener but, except on z/OS, you must first create a listener object by using the MQSC command DEFINE LISTENER. 请参阅以下示例：

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

在 UNIX、Linux 和 Windows 系统上，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。

您还可以使用 WebSphere MQ Explorer (在 Linux 和 Windows 上运行) 或 z/OS 上的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。

WebSphere MQ classes for JMS 的点到点安装验证测试

WebSphere MQ JMS 类随附了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。 有关受管对象的描述，请参阅 第 785 页的『JMS 对象类型』。

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。 该脚本在 UNIX and Linux 系统上称为 IVTRun ，在 Windows 上称为 IVTRun.bat ，并且位于 WebSphere 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 88 页的『准备并运行样本程序』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]
```

在 z/OS 系统上未提供等效脚本，但您可以通过使用以下命令直接调用 Java 类，以绑定方式运行 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。 因此，有关此参数的更多信息 (包括其有效值) ，请参阅 IBM WebSphere MQ classes for JMS 对象的属性 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
启动跟踪。 缺省情况下，将关闭跟踪。

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: WebSphere 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。 您还必须配置 WebSphere MQ JMS 管理工具，以便它可以使用目录服务来存储受管对象。 有关这些先决条件的更多信息，请参阅第 605 页的『 WebSphere MQ JMS 类的先决条件』。 有关如何配置 WebSphere MQ JMS 管理工具的信息，请参阅 第 782 页的『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。 该脚本在 UNIX and Linux 系统上称为 IVTSetup ，在 Windows 上称为 IVTSetup.bat ，位于 WebSphere 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
```

该脚本调用 WebSphere MQ JMS 管理工具以创建受管对象。

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。 如果希望 IVT 程序以客户机方式运行，或者连接到缺省队列管理器以外的队列管理器，那么必须使用 WebSphere MQ JMS 管理工具或 WebSphere MQ Explorer 来更改 MQQueueConnectionFactory 对象的相应属性。 有关如何使用 WebSphere MQ JMS 管理工具的信息，请参阅 第 781 页的『使用 WebSphere MQ JMS 管理工具』。 有关如何使用 WebSphere MQ Explorer 的信息，请参阅 WebSphere 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
启动跟踪。 缺省情况下，将关闭跟踪。

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。 该脚本在 UNIX and Linux 系统上称为 IVTTidy，在 Windows 上称为 IVTTidy.bat，位于 WebSphere 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 609 页的『IBM WebSphere MQ classes for JMS 使用的环境变量』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。 LDAP 服务器未配置为存储 Java 对象，或者对象或后缀上的许可权不正确。 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 88 页的『准备并运行样本程序』。

WebSphere MQ JMS 类的发布/预订安装验证测试

WebSphere MQ JMS 类随附了发布/预订安装验证测试 (IVT) 程序。 该程序在绑定或客户机方式下连接到队列管理器，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。 有关受管对象的描述，请参阅 第 785 页的『JMS 对象类型』。

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。 该脚本在 UNIX and Linux 系统上称为 PSIVTRun ，在 Windows 上称为 PSIVTRun.bat ，位于 WebSphere 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 88 页的『准备并运行样本程序』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。

仅当 -v 参数指定的队列管理器版本号小于 7 ，并且您正在使用 WebSphere Event Broker 或 WebSphere Message Broker 作为发布/预订代理时，此参数才相关。

-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。 因此，有关此参数的更多信息 (包括其有效值) ，请参阅 IBM WebSphere MQ classes for JMS 对象的属性 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
启动跟踪。 缺省情况下，将关闭跟踪。

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
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。 您还必须配置 WebSphere MQ JMS 管理工具，以便它可以使用目录服务来存储受管对象。 有关这些先决条件的更多信息，请参阅第 605 页的『 WebSphere MQ JMS 类的先决条件』。 有关如何配置 WebSphere MQ JMS 管理工具的信息，请参阅 第 782 页的『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。 该脚本在 UNIX and Linux 系统上称为 IVTSetup ，在 Windows 上称为 IVTSetup.bat ，位于 WebSphere 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
```

该脚本调用 WebSphere MQ JMS 管理工具以创建受管对象。

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。 如果希望 IVT 程序以客户机方式运行，请连接到缺省队列管理器以外的队列管理器，或者使用 WebSphere Event Broker 或 WebSphere Message Broker 代替嵌入式发布/预订功能， 您必须使用 WebSphere MQ JMS 管理工具或

WebSphere MQ Explorer 来更改 MQTopicConnectionFactory 对象的相应属性。 有关如何使用 WebSphere MQ JMS 管理工具的信息，请参阅 第 781 页的『使用 WebSphere MQ JMS 管理工具』。 有关如何使用 WebSphere MQ Explorer 的信息，请参阅 WebSphere MQ Explorer 随附的帮助。

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
启动跟踪。 缺省情况下，将关闭跟踪。

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。 该脚本在 UNIX and Linux 系统上称为 IVTTidy，在 Windows 上称为 IVTTidy.bat，位于 WebSphere 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 609 页的『IBM WebSphere MQ classes for JMS 使用的环境变量』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。 LDAP 服务器未配置为存储 Java 对象，或者对象或后缀上的许可权不正确。 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 88 页的『准备并运行样本程序』。

WebSphere MQ 资源适配器的安装验证测试程序

IVT 程序以 EAR 文件形式提供。 要使用该程序，必须部署它并将一些对象定义为 JCA 资源。

安装验证测试（IVT）程序作为企业归档（EAR）文件 wmq.jmsra.ivt.ear 提供。 此文件与 WebSphere MQ classes for JMS 一起安装在与 WebSphere MQ 资源适配器 RAR 文件 wmq.jmsra.rar 相同的目录中。 有关安装这些文件的位置的信息，请参阅第 619 页的『安装 WebSphere 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 和 MDB ，这些 Servlet 和 MDB 测试可将消息发送至 WebSphere MQ 队列以及从该队列接收消息。 (可选) 您可以使用 IVT 程序来验证 WebSphere MQ 资源适配器是否已正确配置为支持分布式事务。

在可以运行 IVT 程序之前，必须将 ConnectionFactory 对象， Queue 对象和可能的激活规范对象定义为 JCA 资源，并确保应用程序服务器根据这些定义创建 JMS 对象并将它们绑定到 JNDI 名称空间。 您可以选择这些对象的属性。下列这组属性是简单的示例：

ConnectionFactory 对象

```
channel:               SYSTEM.DEF.SVRCONN
hostName:              localhost
port:                  1414
queueManager:          ExampleQM
transportType:         CLIENT
```

Queue 对象

```
baseQueueManagerName:  ExampleQM
baseQueueName:         TEST.QUEUE
```

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。

在部署了 IVT 程序，并且应用程序服务器已创建 JMS 对象并将其绑定到 JNDI 名称空间之后，您可以通过在 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 656 页的图 122 显示 IVT 程序的初始页面。

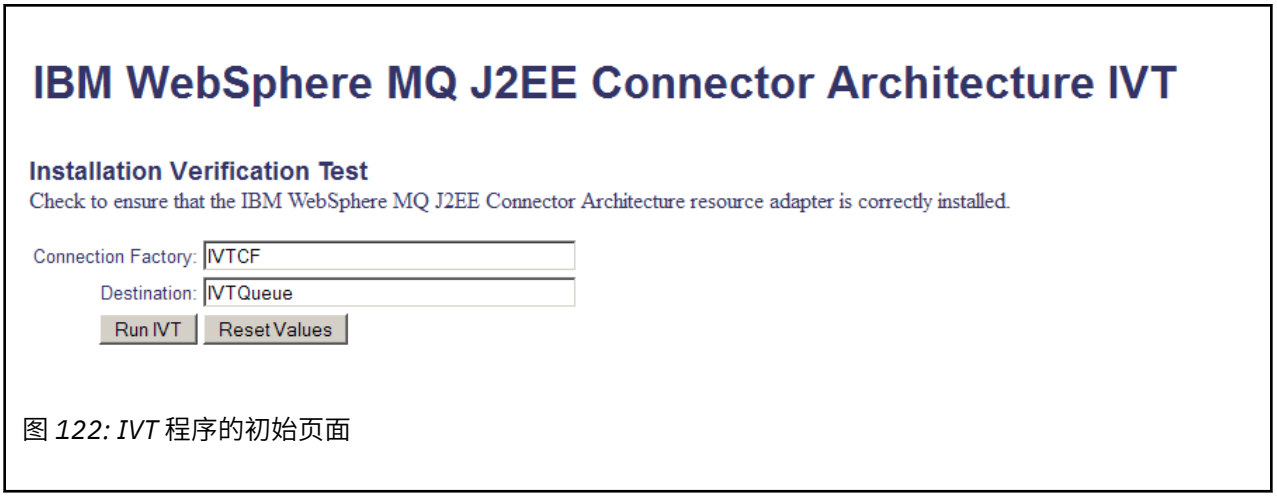


图 *122: IVT* 程序的初始页面

要运行测试，请单击**运行 IVT**。第 656 页的图 123 显示 IVT 成功时显示的页面。

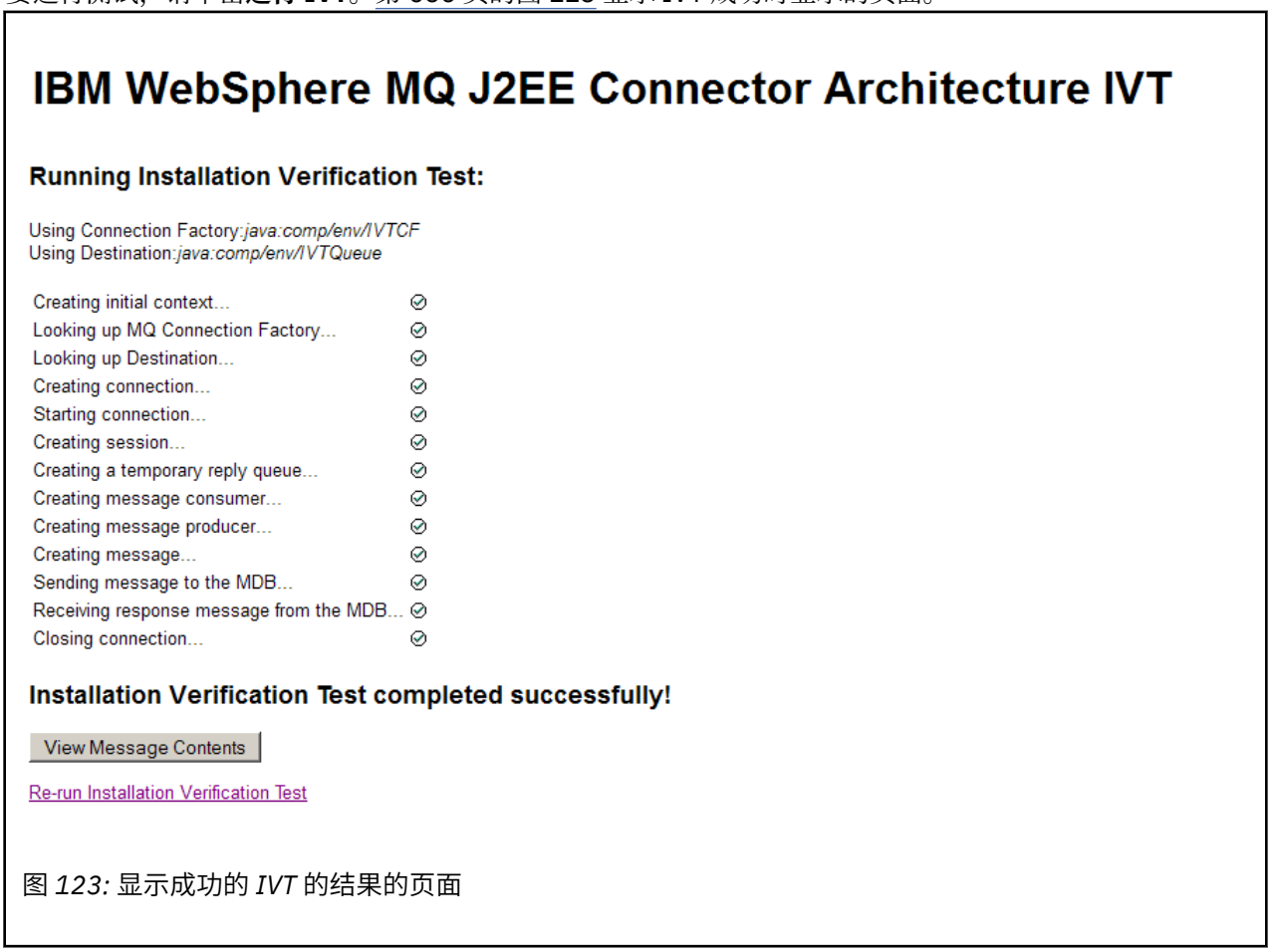


图 *123:* 显示成功的 *IVT* 的结果的页面

如果 IVT 失败，那么将显示类似第 657 页的图 124 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**。

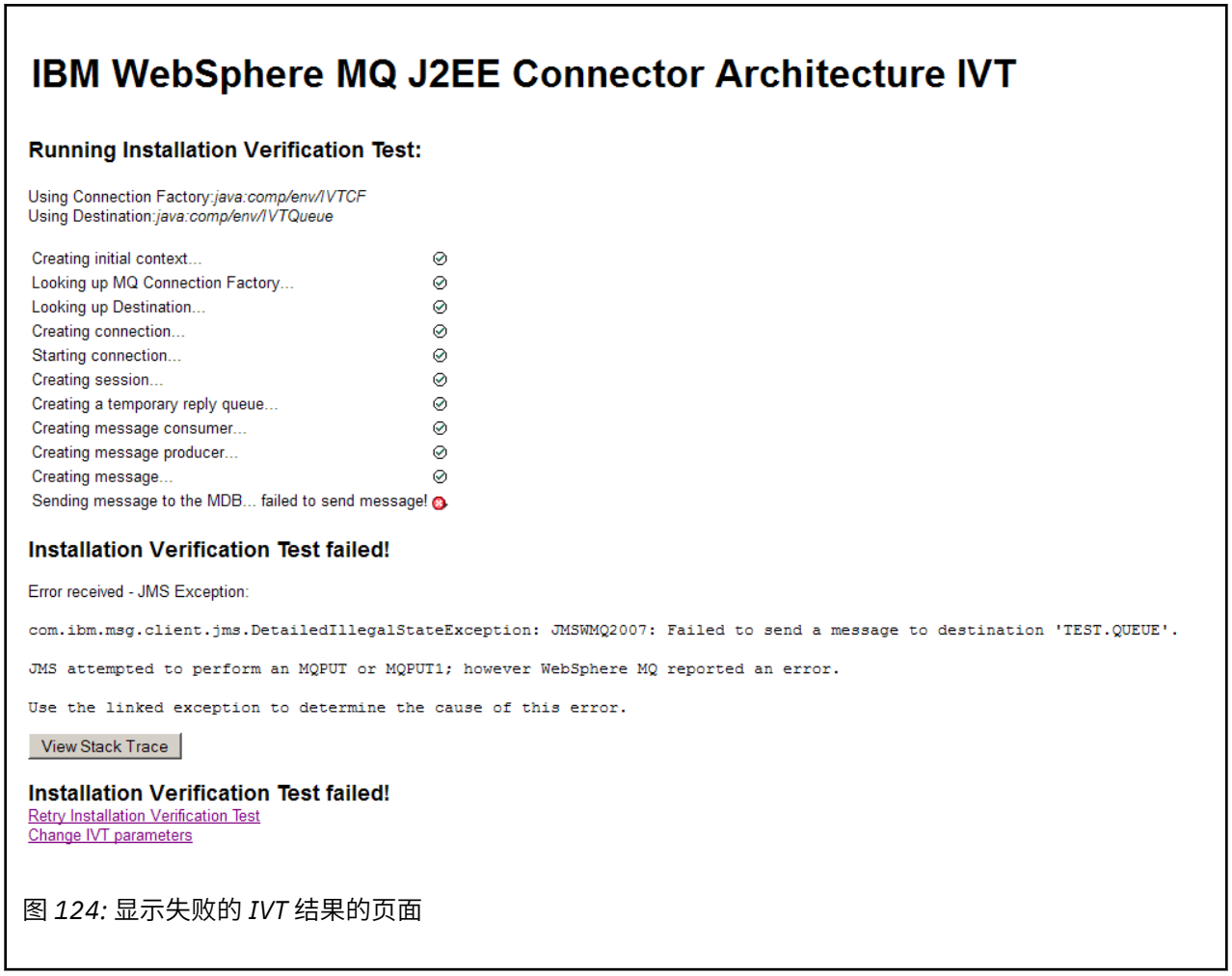


图 *124:* 显示失败的 *IVT* 结果的页面

有关为在 JBoss 和 WAS CE 应用程序服务器上部署 IVT 应用程序而提供的实用程序脚本的详细指示信息和信息，请参阅:

相关任务

第 657 页的『在 WAS CE 中安装和测试 MQ 资源适配器』
在 WebSphere Application Server CE 中安装 IBM WebSphere MQ 资源适配器并运行安装验证测试 (IVT) 应用程序。

第 660 页的『在 JBoss AS 5.1 和 6 中安装和测试资源适配器』
在 JBoss AS 5.1 或 6 上安装 IBM WebSphere MQ 资源适配器之后，可以通过安装并运行安装验证测试 (IVT) 应用程序来测试资源适配器的安装。

在 WAS CE 中安装和测试 MQ 资源适配器

在 WebSphere Application Server CE 中安装 IBM WebSphere MQ 资源适配器并运行安装验证测试 (IVT) 应用程序。

开始之前

此任务假定您具有正在运行的 WebSphere Application Server CE 服务器，并且您熟悉该服务器的标准管理任务。 此任务还假定您在本地系统上安装了 IBM WebSphere MQ ，并且您熟悉标准管理任务。

如果您正在使用资源适配器连接到 IBM WebSphere MQ 客户机，并且需要执行分布式 XA 事务，那么必须遵循标记为 **仅客户机 XA** 的其他步骤。

1. 创建名为 `ExampleQM` 的队列管理器，并按 第 88 页的『准备并运行样本程序』 中所述对其进行设置，注意侦听器应该在端口 1414 上启动，要使用的通道称为 `SYSTEM.DEF.SVRCONN` ， IVT 应用程序使用的队列名为 `TEST.QUEUE`。 还需要向模型队列 `SYSTEM.DEFAULT.MODEL.QUEUE` 授予 DSP 和 PUT 权

限，以便此应用程序可以创建临时应答队列。 如果要使用不同的队列管理器，不同的连接详细信息或不同的队列，请参阅 第 659 页的『使用定制 MQ 环境在 WAS CE 上部署 IVT 应用程序』。

2. 获取资源适配器文件 (`wmq.jmsra.rar`)， IVT 应用程序 (`wmq.jmsra.ivt.ear`) 以及 `WAS_CE_jmsra_deployment_plan.xml` 和 `WAS_CE_jmsra_ivt_deployment_plan.xml deployment plan files`。 有关这些文件的位置的详细信息，请参阅 第 619 页的『安装 WebSphere MQ 资源适配器』。

有关绑定和客户机方式连接的描述，请参阅 第 647 页的『 WebSphere MQ JMS 类的连接方式』。

如果要使用其他队列，队列管理器，端口，主机，通道或使用绑定方式而不是客户机方式，请参阅 第 659 页的『使用定制 MQ 环境在 WAS CE 上部署 IVT 应用程序』。

过程

1. **仅限客户机 XA**: 编辑 `WAS_CE_jmsra_deployment_plan.xml` 文件的副本。
   a) 查找 jms/ivt/IVTCF 连接定义并对其进行修改，以使连接工厂启用 XA 事务。
      i) 注释掉 NonXA 部分:

```
<conn:xa-transaction>
```

      ii) 取消注释 XA 配置部分:

```
<conn:xa-transaction>
    <conn:transaction-caching/>
</conn:xa-transaction>
```

   b) 保存您所做的更改。
2. 可选： **仅限客户机 XA**: 修改 MDB 的组合件描述符以需要事务。 这将强制 IVT 中的 MDB 参与 XA 事务，尽管 IVT 应用程序仍在未进行此修改的情况下工作。
   a) 打开 `wmq.jmsra.ivt.ear` 文件。
   b) 打开其中的 `WMQ_IVT_MDB.jar` 文件。
   c) 编辑 `META-INF/ejb-jar.xml`。
      i) 注释掉或删除组合件描述符中的行:

```
<trans-attribute>NotSupported</trans-attribute>
```

      ii) 取消注释组合件描述符中的行:

```
<trans-attribute>Required</trans-attribute>
```

      iii) 保存更改并在 `WMQ_IVT_MDB.jar` 文件中更新该文件。
      iv) 使用修改后的 `WMQ_IVT_MDB.jar` 文件更新 wmq.jmsra.ivt.ear 文件。
3. 使用修改后的部署计划文件将资源适配器部署到服务器。
   a) 要在命令行上执行此操作，请输入以下 WAS CE 命令:

```
deploy -u system -p manager deploy wmq.jmsra.rar WAS_CE_jmsra_deployment_plan.xml
```

   b) 使用 Web 管理界面，转至 **Applications> Deployer:**
      i) 将归档设置为 wmq.jmsra.rar 文件。
      ii) 将计划设置为 WAS_CE_jmsra_deployment_plan.xml 文件。
      iii) 确保选择 "安装后启动应用程序"。
      iv) 单击**安装**。
4. 使用提供的部署计划将 IVT 应用程序部署到服务器。
   a) 在命令行上，可以使用以下 WAS CE 命令来完成此操作:

```
deploy -u system -p manager deploy wmq.jmsra.ivt.ear WAS_CE_jmsra_ivt_deployment_plan.xml
```

b) 使用 Web 管理界面，转至 **Applications> Deployer:**

i) 将 **归档** 设置为 `wmq.jmsra.ivt.ear` 文件。

ii) 将 **计划** 设置为 `WAS_CE_jmsra_ivt_deployment_plan.xml` 文件。

iii) 确保选择**在安装后启动应用程序**。

iv) 单击**安装**。

5. 运行 IVT 应用程序。 要获取更多详细信息，请参阅第 655 页的『 WebSphere MQ 资源适配器的安装验证测试程序』。 对于 WAS CE，缺省 URL 为 `http://localhost:8080/WMQ_IVT/`。

使用定制 MQ 环境在 WAS CE 上部署 IVT 应用程序

如果要使用其他队列，队列管理器，端口，主机，通道或使用绑定方式而不是客户机方式，那么在部署资源适配器或 IVT 应用程序之前，必须在 WebSphere Application Server CE 中修改 IVT 应用程序和关联脚本。

关于此任务

如果要部署到与 第 657 页的『在 WAS CE 中安装和测试 MQ 资源适配器』中指定的配置不同的配置，即，如果要使用不同的队列，队列管理器，端口，主机，通道或使用绑定方式而不是客户机方式，请在部署资源适配器或 IVT 应用程序之前执行以下步骤。

过程

1. 如果要指定要用于 IVT 应用程序的其他队列管理器和队列，请在 `WAS_CE_jmsra_deployment_plan.xml` 中设置队列管理器和队列的值，有关详细信息，请参阅 第 659 页的『设置队列管理器和队列的值』。
2. 如果要在消息驱动的 bean (MDB) 的配置中指定其他队列管理器和队列，请为您在 `WAS_CE_jmsra_ivt_deployment_plan.xml` 中使用的队列管理器和队列设置值，有关详细信息，请参阅 第 660 页的『设置 MDB 配置的值』。
3. 如果要将资源适配器配置为以绑定方式连接到 IBM WebSphere MQ，请确保 JNI 库位于 WAS CE 的系统路径或路径上。 有关更多信息，请参阅第 657 页的『在 WAS CE 中安装和测试 MQ 资源适配器』。
4. 如果已部署资源适配器，那么可以使用修改后的部署计划重新部署该资源适配器，以使用以下命令更改设置:

```
deploy --user system --password manager redeploy wmq.jmsra.rar
WAS_CE_jmsra_deployment_plan.xml
```

下一步做什么

继续部署资源适配器，如 第 657 页的『在 WAS CE 中安装和测试 MQ 资源适配器』中所述。

设置队列管理器和队列的值

说明如何为您在 `WAS_CE_jmsra_deployment_plan.xml` 中使用的队列管理器和队列设置值。

过程

在 `WAS_CE_jmsra_deployment_plan.xml` 中，设置要用于 IVT 应用程序的队列管理器和队列的值。

对于 jms/ivt/IVTCF 连接定义:

1. 将 queueManager 元素的值设置为队列管理器的名称。
2. 如果您正在使用客户机连接，请将各种客户机连接元素的值设置为适合于与队列管理器的连接。
3. 如果您正在使用绑定连接:

a. 将 transportType 元素的值设置为 BINDINGS。

b. 注释掉或删除各种客户机连接元素。

4. 对于 jms/ivt/IVTQueue 消息目标，将 baseQueueName 元素的值设置为您为 IVT 应用程序创建的队列的名称
5. 保存您所做的更改。

设置 *MDB* 配置的值

说明如何在 `WAS_CE_jmsra_deployment_plan.xml` 中设置 MDB 配置的值。

过程

在 `WAS_CE_jmsra_ivt_deployment_plan.xml` 中，为您在 MDB 配置中使用的队列管理器和队列设置值。

对于 WMQ_IVT_MDB 消息驱动的 Bean:

1. 将 queueManager 元素的值设置为队列管理器的名称。
2. 如果您正在使用客户机连接，请将各种客户机连接元素的值设置为适合于与队列管理器的连接。
3. 如果您正在使用绑定连接:
   a. 将 transportType 元素的值设置为 BINDINGS。
   b. 注释掉或删除各种客户机连接元素。
4. 保存您所做的更改。

在 JBoss AS 5.1 和 6 中安装和测试资源适配器

在 JBoss AS 5.1 或 6 上安装 IBM WebSphere MQ 资源适配器之后，可以通过安装并运行安装验证测试 (IVT) 应用程序来测试资源适配器的安装。

开始之前

要点: 这些指示信息适用于 JBoss AS 5.1 和 6 ，它们对于 JBoss AS 7 无效。

有关在 JBoss EAP 6.3 中安装资源适配器的信息，请参阅 第 662 页的『在 JBoss EAP 6.3 中安装和测试资源适配器』。

此任务假定您具有正在运行的 JBoss 服务器，并且熟悉该服务器的标准管理任务。 此任务还假定您在本地系统上安装了 IBM WebSphere MQ 并且假定您熟悉标准管理任务。

如果您正在使用资源适配器连接到 IBM WebSphere MQ 客户机，并且需要执行分布式 XA 事务，那么必须遵循标记为 **仅客户机 XA** 的其他步骤。 有关绑定和客户机方式连接的描述，请参阅 第 647 页的『 WebSphere MQ JMS 类的连接方式』。

过程

1. 创建名为 ExampleQM 的队列管理器，并按 第 88 页的『准备并运行样本程序』中所述进行设置。
   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。

   如果要使用不同的队列管理器，不同的连接详细信息或不同的队列，请参阅 第 659 页的『使用定制 MQ 环境在 WAS CE 上部署 IVT 应用程序』。
2. 获取资源适配器文件 (`wmq.jmsra.rar`)， IVT 应用程序 (`wmq.jmsra.ivt.ear`) 和 `jboss-jmsra-ds.xml` 文件。
   有关这些文件的位置，请参阅 第 619 页的『安装 WebSphere MQ 资源适配器』。
3. **仅限客户机 XA**: 编辑 `jboss-jmsra-ds.xml` 文件以在连接工厂上启用 XA 事务。

a) 注释掉或删除连接工厂定义 <local-transaction/>中的行。

b) 取消注释连接工厂定义 <xa-transaction/>中的行。

c) 保存您所做的更改。

4. **仅限客户机 XA**: (可选) 修改 MDB 的组合件描述符以需要事务。 这将强制 IVT 中的 MDB 参与 XA 事务，尽管 IVT 应用程序仍在未进行此修改的情况下工作。

a) 打开 wmq.jmsra.ivt.ear 文件。

b) 打开其中的 WMQ_IVT_MDB.jar 文件。

c) 编辑 META-INF/ejb-jar.xml:

i) 注释掉或删除组合件描述符中的行:

```
<trans-attribute>NotSupported</trans-attribute>
```

ii) 取消注释组合件描述符中的行:

```
<trans-attribute>Required</trans-attribute>
```

iii) 保存更改并在 WMQ_IVT_MDB.jar 文件中更新该文件。

iv) 使用已修改的 WMQ_IVT_MDB.jar 更新 wmq.jmsra.ivt.ear 文件。

5. 通过将 wmq.jmsra.rar 文件复制到目录 jboss/server/default/deploy 以将资源适配器部署到服务器。
6. 通过将 jboss-jmsra-ds.xml 文件复制到目录 jboss/server/default/deploy 来创建 IVT 应用程序所需的 JMS 资源。
7. 通过将 wmq.jmsra.ivt.ear 文件复制到目录 jboss/server/default/deploy 来部署 IVT 应用程序。
8. 运行 IVT 应用程序。 要获取更多详细信息，请参阅第 655 页的『 WebSphere MQ 资源适配器的安装验证测试程序』。 对于 JBoss，缺省 URL 为 http://localhost:8080/WMQ_IVT/。

使用定制 IBM WebSphere MQ 环境在 JBoss 中部署 IVT 应用程序

在 JBoss 中安装 IBM WebSphere MQ 资源适配器时，如果要使用其他队列，队列管理器，端口，主机，通道或使用绑定方式而不是客户机方式，那么必须先修改 JBoss 中的 IVT 应用程序和关联脚本，然后再部署资源适配器或 IVT 应用程序。

关于此任务

要点: 这些指示信息仅适用于 Java EE 版本 6 和 5 ， 而不适用于 Java EE 版本 7。 因此，不支持对 JBoss V 8 (WildFly) 使用这些指示信息。

如果要部署到与第 660 页的『在 JBoss AS 5.1 和 6 中安装和测试资源适配器』中指定的配置不同的配置，即，如果要使用不同的队列管理器，队列，端口，主机，通道或使用绑定方式而不是客户机方式，请在部署资源适配器或 IVT 应用程序之前完成以下步骤。

过程

1. 如果要指定要用于 IVT 应用程序的其他队列管理器和队列，请设置队列管理器和队列的值。

a) 对于 jms/ivt/IVTCF 连接定义:

i) 将 queueManager config-property 的值设置为队列管理器的名称。

ii) 如果您正在使用客户机连接，请将各种客户机连接元素的值设置为适合于与队列管理器的连接。

iii) 如果您正在使用绑定连接，请将 transportType 元素的值设置为 BINDINGS ， 然后注释掉或删除各种客户机连接元素。

b) 对于 jms/ivt/IVTQueue MBean ， 将 baseQueueName 元素的值设置为您为 IVT 应用程序创建的队列的名称。

c) 保存您所做的更改。

2. 如果要在消息驱动的 bean (MDB) 的配置中指定其他队列管理器和队列，请修改 MDB 的配置以连接到队列管理器和队列。
   a) 打开 `wmq.jmsra.ivt.ear` 文件。
   b) 打开其中的 `WMQ_IVT_MDB.jar` 。
   c) 编辑 `META-INF/ejb-jar.xml`:
      i) 将 queueManager activation-config-property 的值设置为队列管理器的名称。
      ii) 如果您正在使用客户机连接，请将各种客户机连接激活配置属性的值设置为适合于与队列管理器的连接。
      iii) 如果要使用绑定连接，请将 transportType Activation-config-property 的值设置为 BINDINGS，然后注释掉或删除各种客户机连接元素。
   d) 保存更改并在 `WMQ_IVT_MDB.jar` 文件中更新该文件。
   e) 使用已修改的 `WMQ_IVT_MDB.jar` 更新 `wmq.jmsra.ivt.ear` 文件。
3. 如果要将资源适配器配置为以绑定方式连接到 IBM WebSphere MQ，请确保 JNI 库位于系统路径或 JBoss 的路径上。 有关详细信息，请参阅第 611 页的『配置 Java 本机接口 (JNI) 库』。

下一步做什么

继续部署资源适配器，如 第 660 页的『在 JBoss AS 5.1 和 6 中安装和测试资源适配器』中所述。

.

在 JBoss EAP 6.3 中安装和测试资源适配器

在独立服务器或受管域中运行的服务器上安装 JBoss Enterprise Application Platform (EAP) 6.3 中的 IBM WebSphere MQ 资源适配器后，可以通过安装和运行安装验证测试 (IVT) 应用程序来测试资源适配器的安装。

关于此任务

要点: 这些指示信息仅适用于 JBoss EAP 6.3 。 有关在 JBoss AS 5.1 和 6 中安装资源适配器的信息，请参阅第 660 页的『在 JBoss AS 5.1 和 6 中安装和测试资源适配器』。

此任务假定您具有正在运行的 JBoss 服务器，并且熟悉该服务器的标准管理任务。 此任务还假定您在本地系统上安装了 IBM WebSphere MQ，并且您熟悉标准管理。

过程

1. 创建名为 ExampleQM 的队列管理器，并按 第 88 页的『准备并运行样本程序』中所述进行设置。
   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。

   如果要使用不同的队列管理器，不同的连接详细信息或不同的队列，请参阅 第 659 页的『使用定制 MQ 环境在 WAS CE 上部署 IVT 应用程序』。
2. 获取资源适配器文件 (`wmq.jmsra.rar`) 和 IVT 应用程序 (`wmq.jmsra.ivt.ear`)。
   有关这些文件的位置，请参阅 第 619 页的『安装 WebSphere MQ 资源适配器』。
3. 安装资源适配器，然后通过运行安装验证测试 (IVT) 应用程序来测试安装:
   - 如果要在独立服务器上安装资源适配器，请参阅 第 663 页的『在独立服务器上安装和测试』。

- 如果要在受管域中运行的服务器上安装资源适配器，请参阅 第 664 页的『在受管域中运行的服务器上安装和测试』。

在独立服务器上安装和测试

在独立服务器上的 JBoss EAP 6.3 上安装 IBM WebSphere MQ 资源适配器后，可以通过安装和运行安装验证测试 (IVT) 应用程序来测试资源适配器的安装。

关于此任务

此任务中的信息用于在独立服务器上安装和测试资源适配器。 如果要在受管域中运行的服务器上安装资源适配器，请参阅 第 664 页的『在受管域中运行的服务器上安装和测试』。

要点: 这些指示信息仅适用于 JBoss EAP 6.3 。 有关在 JBoss AS 5.1 和 6 中安装资源适配器的信息，请参阅 第 660 页的『在 JBoss AS 5.1 和 6 中安装和测试资源适配器』。

过程

1. 通过将 `wmq.jmsra.rar` 文件复制到目录 *<EAP_HOME>*`/standalone/deployments` 以将资源适配器部署到服务器。
2. 通过将以下条目添加到 *<EAP_HOME>*`/standalone/configuration/standalone-full.xml` 文件的 `<resource-adapters>` 部分，创建 IVT 应用程序所需的 JMS 资源:

```
<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="port">
           1414
       </config-property>
       <config-property name="hostName">
           localhost
       </config-property>
       <config-property name="channel">
           SYSTEM.DEF.SVRCONN
       </config-property>
       <config-property name="transportType">
           CLIENT
       </config-property>
       <config-property name="queueManager">
           ExampleQM
       </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>
```

3. 将以下信息添加到应用程序服务器的启动参数:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. 通过将 `wmq.jmsra.ivt.ear` 文件复制到目录 *<EAP_HOME>*`/standalone/deployments` 来部署 IVT 应用程序。
5. 启动应用程序服务器。
6. 运行 IVT 应用程序。

有关更多信息，请参阅第 655 页的『 WebSphere MQ 资源适配器的安装验证测试程序』。 对于 JBoss，缺省 URL 为 `http://localhost:8080/WMQ_IVT/`。

注: 用于运行 IVT 应用程序所需的 JMS 资源的 JNDI 名称如下所示:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

使用上面指定的 URL 启动 IVT 应用程序时，请在其各自的字段中输入这些资源的 JNDI 名称，然后通过单击 **运行 IVT** 来运行该应用程序。

在受管域中运行的服务器上安装和测试

在受管域中运行的服务器上的 JBoss EAP 6.3 上安装 IBM WebSphere MQ 资源适配器后，可以通过安装和运行安装验证测试 (IVT) 应用程序来测试资源适配器的安装。

关于此任务

此任务中的信息用于在受管域中运行的服务器上安装和测试资源适配器。 如果要在独立服务器上安装资源适配器，请参阅 第 663 页的『在独立服务器上安装和测试』。

要点: 这些指示信息仅适用于 JBoss EAP 6.3 。 有关在 JBoss AS 5.1 和 6 中安装资源适配器的信息，请参阅第 660 页的『在 JBoss AS 5.1 和 6 中安装和测试资源适配器』。

过程

1. 使用 JBoss 管理控制台或管理 CLI 将资源适配器部署到服务器。
2. 通过将以下条目添加到 `<EAP_HOME>/domain/configuration/domain.xml file` 的 `<resource-adapters>` 部分，创建 IVT 应用程序所需的 JMS 资源:

```
<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="port">
          1414
      </config-property>
      <config-property name="hostName">
          localhost
      </config-property>
      <config-property name="channel">
          SYSTEM.DEF.SVRCONN
      </config-property>
      <config-property name="transportType">
          CLIENT
      </config-property>
      <config-property name="queueManager">
          ExampleQM
      </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>
```

3. 将以下信息添加到应用程序服务器的启动参数:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. 停止和重新启动应用程序服务器。
5. 使用 JBoss 管理控制台或管理 CLI 部署 IVT 应用程序。
6. 运行 IVT 应用程序。

   有关更多信息，请参阅第 655 页的『WebSphere MQ 资源适配器的安装验证测试程序』。 对于 JBoss，缺省 URL 为 `http://localhost:8080/WMQ_IVT/`。

   注: 用于运行 IVT 应用程序所需的 JMS 资源的 JNDI 名称如下所示:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   当您使用上面指定的 URL 启动 IVT 应用程序，然后通过单击 **运行 IVT** 来运行应用程序时，请在其各自的字段中输入这些资源的 JNDI 名称。

随 WebSphere MQ classes for JMS 提供的脚本

提供了许多脚本以帮助执行在使用 WebSphere MQ JMS 类时需要执行的常见任务。

第 665 页的表 102 列出所有脚本及其用途。 这些脚本位于 WebSphere MQ classes for JMS 安装目录的 `bin` 子目录中。

表 *102:* 随 *WebSphere MQ classes for JMS* 提供的脚本

实用程序	**适用平台**
清除 [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 Windows 以外的平台上运行缺省配置应用程序。
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 648 页的『WebSphere MQ classes for JMS 的点到点安装验证测试』中所述。
JMSAdmin[1]	运行 WebSphere MQ JMS 管理工具，如 第 781 页的『调用 IBM WebSphere MQ classes for JMS 管理工具』中所述。
JMSAdmin.config	WebSphere MQ JMS 管理工具的配置文件，如 第 782 页的『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 652 页的『WebSphere MQ JMS 类的发布/预订安装验证测试』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 JMS 应用程序的 WebSphere MQ 类的环境变量，如 第 609 页的『IBM WebSphere MQ classes for JMS 使用的环境变量』中所述。
setjmsenv64	设置用于在 UNIX and Linux 系统上的 64 位 JVM 中运行 JMS 应用程序的 WebSphere MQ 类的环境变量，如 第 609 页的『IBM WebSphere MQ classes for JMS 使用的环境变量』中所述。

表 *102:* 随 *WebSphere MQ classes for JMS* 提供的脚本 *(*继续*)*	
实用程序	**适用平台**
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 WebSphere 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 WebSphere MQ classes for JMS。 包含以下 OSGi 捆绑软件。

com.ibm.msg.client.osgi.jms<version number>.jar
IBM WebSphere MQ classes for JMS 中的代码通用层。 有关 WebSphere MQ JMS 类的分层体系结构的信息，请参阅 第 668 页的『一种分层架构』。

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) 应用程序的公共服务。

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 WebSphere MQ classes for JMS 中的 IBM WebSphere MQ 消息传递提供程序。 有关 IBM WebSphere MQ classes for JMS 的分层体系结构的信息，请参阅 第 668 页的『一种分层架构』。

com.ibm.msg.client.osgi.wmq.prereq_<version number>.jar
IBM WebSphere 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 WebSphere MQ 消息传递提供程序的消息。

com.ibm.mq.osgi.directip_<version number>.jar
用于允许 IBM WebSphere MQ 消息传递提供程序创建与代理程序的实时连接的 JAR 文件。

where <version number> is the version number of WebSphere MQ that has been installed.

这些捆绑软件将安装到 WebSphere MQ 安装的 `java/lib/OSGi` 子目录中，或者安装到 Windows 上的 `java\lib\OSGi` 文件夹中。

The bundle com.ibm.mq.osgi.java <version number>.jar, which is also installed into the `java/lib/OSGi` subdirectory of your WebSphere MQ installation, or the `java\lib\OSGi` folder on Windows, is part of the WebSphere MQ classes for Java. 不得将此捆绑软件装入到已装入 WebSphere MQ JMS 类的 OSGi 运行时环境中。

WebSphere MQ classes for JMS 的 OSGi 捆绑软件已写入 OSGi 发行版 4 规范。 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 WebSphere 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。 用户捆绑软件可以知道 JMS 捆绑软件的 IBM WebSphere MQ 类，但 IBM WebSphere MQ classes for JMS 捆绑软件不知道任何用户捆绑软件。 因此， IBM WebSphere 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。

解决 IBM WebSphere MQ JMS 类的问题

您可以运行安装验证程序和使用跟踪和日志功能来研究问题。

如果未成功完成程序，请如第 648 页的『 WebSphere MQ classes for JMS 的点到点安装验证测试』和第 652 页的『 WebSphere MQ JMS 类的发布/预订安装验证测试』中所述，运行某一安装验证程序，并遵循诊断消息中给出的建议。

日志记录和 *IBM WebSphere MQ classes for JMS*

缺省情况下，将向 `mqjms.log` 文件发送日志输出。 您可以将其重定向到特定的文件或目录。

系统提供 IBM WebSphere MQ classes for JMS 日志功能来报告严重的问题，尤其是可能指出配置错误（而非编程错误）的问题。 缺省情况下，将向位于 JVM 工作目录中的 `mqjms.log` 文件发送日志输出。

您可以通过设置 com.ibm.msg.client.commonservices.log.outputName 属性，将日志输出重定向到另一个文件。 此属性的值可以是：

- 单一路径名。
- 路径名的逗号分隔列表（所有数据均记录到所有文件中）。

每一个路径名都可以是：

- 绝对的或相对的。
- `stderr` 或 `System.err`，表示标准错误流。
- `sttdout` 或 `System.out`，表示标准输出流。

如果属性值可识别目录，那么日志输出将写入该目录下的 `mqjms.log` 中。 如果属性值可识别特定的文件，那么日志输出将写入该文件。

可以在 IBM WebSphere MQ classes for JMS 配置文件中设置此属性，或在 **java** 命令上将此属性设置为系统属性。 在以下示例中，属性将设置为系统属性，并用于识别特定的文件：

```
java -Djava.library.path=library_path
     -Dcom.ibm.msg.client.commonservices.log.outputName=/mydir/mylog.txt
      MyAppClass
```

在此命令中，*library_path* 是包含 IBM WebSphere MQ classes for JMS 库的目录路径（请参阅第 611 页的『配置 Java 本机接口 (JNI) 库』）。

可通过将 com.ibm.msg.client.commonservices.log.status 属性设置为 OFF，来禁用日志输出。 此属性的缺省值为 ON。

可设置 `System.err` 和 `System.out` 的值，以向 `System.err` 和 `System.out` 流发送日志输出。

针对程序员的 WebSphere MQ JMS 类简介

WebSphere MQ JMS 类是随 WebSphere MQ 提供的 JMS 提供程序。 WebSphere MQ classes for JMS 实现 javax.jms 包中定义的接口，并且还提供对 JMS API 的两组扩展。 Java Platform， Standard Edition (Java SE) 和 Java Platform, Enterprise Edition (Java EE) 应用程序都可以将 WebSphere MQ 类用于 JMS。

JMS 规范定义了一组接口，应用程序可以使用这些接口来执行消息传递操作。 规范的最新版本为 V 1.1。 javax.jms 包指定 JMS 接口的详细信息，并且 JMS 提供程序为特定消息传递产品实现这些接口。 WebSphere MQ classes for JMS 是实现 WebSphere MQ 的 JMS 接口的 JMS 提供程序。

JMS 应用程序中的逻辑流以 ConnectionFactory 和 Destination 对象开头。 应用程序使用 ConnectionFactory 对象来创建 Connection 对象，该对象表示从应用程序到消息传递服务器的活动连接。 应用程序使用 Connection 对象来创建 Session 对象，它是用于生成和使用消息的单线程上下文。 然后，应用程序可以使用 Session 对象和 Destination 对象来创建 MessageProducer 对象，应用程序使用该对象将消息发送到指定的目标。 目标是消息传递系统中的队列或主题，并由 Destination 对象封装。 应用程序还可以使用 Session 对象和 Destination 对象来创建 MessageConsumer 对象，应用程序使用该对象来接收已发送到指定目标的消息。

JMS 规范期望 ConnectionFactory 和 Destination 对象是受管对象。 管理员在中央存储库中创建并维护受管对象， JMS 应用程序使用 Java 命名和目录接口 (JNDI) 检索这些对象。 受管对象存储库可以小至单个文件，大至轻量级目录访问协议 (LDAP) 目录。

WebSphere MQ classes for JMS 支持使用受管对象。 应用程序可以使用通过 WebSphere MQ JMS 类公开的 WebSphere MQ 的所有功能部件，而无需将任何特定于 WebSphere MQ 的信息硬编码到应用程序本身中。 此安排为应用程序提供了与底层 WebSphere MQ 配置的独立程度。 为实现这种独立性，应用程序可以使用 JNDI 来检索已存储为受管对象的连接工厂和目标，并且仅使用 javax.jms 包中定义的接口来执行消息传递操作。 管理员可以使用 WebSphere MQ JMS 管理工具或 IBM WebSphere MQ Explorer 在中央存储库中创建和维护受管对象。 但是，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。 因此， Java EE 应用程序可以使用 JNDI 从应用程序服务器存储库或中央存储库检索受管对象。

WebSphere MQ classes for JMS 还提供对 JMS API 的扩展。 先前发行版的 WebSphere MQ JMS 类包含在 MQConnectionFactory， MQQueue 和 MQTopic 对象中实现的扩展。 这些对象具有特定于 WebSphere 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。 此发行版的 WebSphere MQ JMS 类维护这些扩展，并且您可以继续使用任何使用这些扩展的应用程序，而不进行任何更改。 这些扩展称为 *WebSphere MQ JMS* 扩展。 请注意，在此系列文档中，应用程序在运行时动态创建的对象不会被视为受管对象。

除了 WebSphere MQ JMS 扩展之外，此发行版的 WebSphere MQ classes for JMS 提供了一组更通用的 JMS API 扩展。 这些扩展称为 *IBM JMS* 扩展，具有以下广泛目标:

- 在 IBM JMS 提供程序之间提供更高级别的一致性
- 便于在两个 IBM 消息传递系统之间编写网桥应用程序
- 为了更轻松地将应用程序从一个 IBM JMS 提供程序移植到另一个提供程序

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。

使用 javax.jms 接口或一组 JMS 扩展创建的连接工厂，队列或主题对象可以使用这些 API 中的任何一个进行寻址; 即，可以将其强制转换为任何接口。 为保持最大程度的应用程序可移植性，请使用符合您需求的最通用的 API。

Java SE 和 Java EE 应用程序都可以将 WebSphere MQ 类用于 JMS。 在 Java EE 平台上， WebSphere MQ JMS 类支持应用程序组件与 WebSphere MQ 队列管理器之间的两种通信类型:

出站通信
通过直接使用 JMS API ，应用程序组件将创建与队列管理器的连接，然后发送和接收消息。

例如，应用程序组件可以是应用程序客户机， Servlet ， JavaServer 页面 (JSP) ，企业 Java Bean (EJB) 或消息驱动的 Bean (MDB)。 在此类型的通信中，应用程序服务器容器在消息传递操作支持中仅提供低级功能，例如连接池和线程管理。

入站通信
到达目标的消息将传递到 MDB ，然后 MDB 将处理该消息。

Java EE 应用程序使用 MDB 异步处理消息。 MDB 充当 JMS 消息侦听器并由 onMessage() 方法实现，该方法定义如何处理消息。 已将 MDB 部署到应用程序服务器的 EJB 容器中。 MDB 的精确配置方式取决于所使用的应用程序服务器，但配置信息必须指定要连接到的队列管理器、如何连接到队列管理器、要监视哪个目标中的消息以及 MDB 的事务行为。 然后，MDB 容器将使用这些信息。 当满足 MDB 选择条件的消息到达指定目标时， EJB 容器使用 WebSphere MQ classes for JMS 从队列管理器检索消息，然后通过调用其 onMessage() 方法将消息传递到 MDB。

用于 JMS 体系结构的 IBM WebSphere MQ 类

与先前发行版相比， IBM WebSphere MQ V 7.0 和后续发行版中提供的 IBM WebSphere MQ JMS 类包含许多增强功能。 其中一些增强功能是由于对 IBM WebSphere MQ classes for JMS 实现的更改所致，而一些增强功能是由于 IBM WebSphere MQ classes for JMS 利用对底层 IBM WebSphere MQ 函数的更改所致。

以下部分概述了关键增强功能。

一种分层架构

在 WebSphere MQ 的先前发行版中， WebSphere MQ classes for JMS 的实现完全特定于 WebSphere MQ。 提供消息传递系统的其他 IBM 产品也包含 JMS 提供程序，但这些 JMS 提供程序与 WebSphere MQ JMS 类的实现几乎没有任何共同点。

从 WebSphere MQ V7.0 开始， WebSphere MQ classes for JMS 具有分层体系结构。 顶层代码是可供任何 IBM JMS 提供程序使用的公共层。 当应用程序调用 JMS 方法时，不特定于消息传递系统的调用的任何处理都由公共层执行，这也提供对该调用的一致响应。 对特定于消息传递系统的调用的任何处理将委派到下一层。 第 669 页的图 125 显示了分层体系结构。

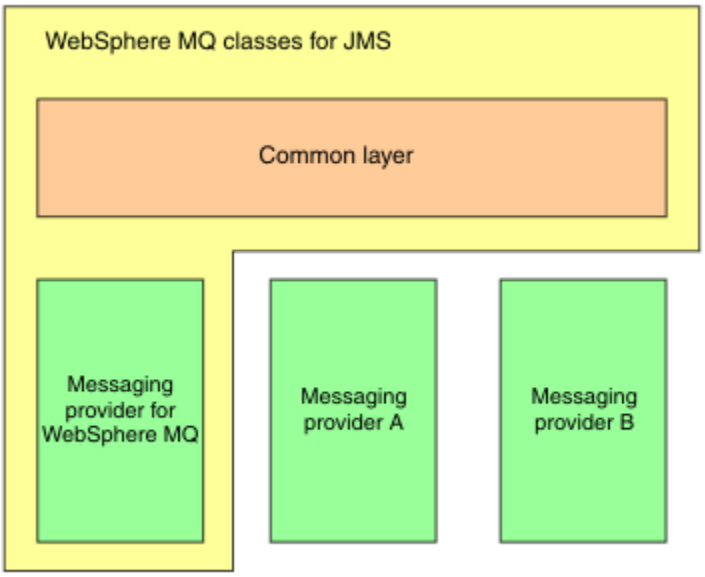


图 *125: IBM JMS* 提供程序的分层体系结构

迁移到分层体系结构具有以下目标:

- 提高各种 IBM JMS 提供程序的行为一致性
- 便于在两个 IBM 消息传递系统之间编写网桥应用程序
- 为了更轻松地将应用程序从一个 IBM JMS 提供程序移植到另一个提供程序

针对 JMS 的 WebSphere MQ 类的此实现还引入了一组新的 JMS API 扩展。 这些扩展称为 *IBM JMS* 扩展。 这些扩展的主要重点是在运行时动态创建和配置连接工厂和目标。

使用 IBM JMS 扩展的应用程序通过创建 JmsFactoryFactory 对象 (将标识所选消息传递系统的常量指定为参数) 来启动。 应用程序使用 JmsFactoryFactory 对象来创建具有所选消息传递系统的正确专用类的连接工厂和目标。

然后，应用程序可以通过设置其属性来配置连接工厂和目标。 IBM JMS 扩展提供了一组用于设置属性的方法。 这些方法独立于任何消息传递系统。 每种数据类型都有自己的 set 方法，每个属性都由一个名称标识，该名称定义为 WMQConstants 类的静态最终成员。 当应用程序调用其中一个方法时，调用上的一个参数是属性的名称，另一个参数是属性的值。

例如，如果 WebSphere MQ 是消息传递系统，那么连接工厂的其中一个属性是要连接到的队列管理器的名称。 通过使用 IBM JMS 扩展，应用程序通过调用以下方法将队列管理器的名称设置为 JUPITER:

```
JmsConnectionFactory myCF;
...
myCF.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "JUPITER");
```

相比之下，应用程序可以通过调用以下方法来执行相同的函数:

```
MQConnectionFactory myCF;
...
myCF.setQueueManager("JUPITER");
```

此方法是 WebSphere MQ JMS 扩展，并且特定于作为消息传递系统的 WebSphere MQ 。 因此，使用此方法会使应用程序不太容易移植到另一个 IBM JMS 提供程序。

WebSphere MQ JMS 类与 WebSphere MQ Java 类之间的关系

在 WebSphere MQ 的发行版 7.0 之前的版本中， WebSphere MQ classes for JMS 几乎完全作为 WebSphere MQ classes for Java 之上的代码层实现。 由于在 MQEnvironment 类中设置字段或调用方法可能会对使用 WebSphere MQ classes for JMS 编写的代码的运行时行为造成意外的影响，因此此安排在应用程序开发者之间造成了一些混乱。 此外， WebSphere MQ classes for JMS 的实现在 JMS API 并非基于 WebSphere MQ classes for Java 的自然拟合的区域中具有一些约束，并且这些约束导致了一些与运行时性能相关的问题。

从 WebSphere MQ V7.0 开始， WebSphere MQ classes for JMS 的实现不再依赖于 WebSphere MQ classes for Java。 WebSphere MQ classes for Java 和 WebSphere MQ classes for JMS 现在是使用 MQI 的公共 Java 接口的同级。 此安排允许更多范围来优化性能，并且意味着在 MQEnvironment 类中设置字段或调用方法不会影响使用 WebSphere MQ classes for JMS 编写的代码的运行时行为。 第 670 页的图 126 显示先前发行版的 WebSphere MQ JMS 类与先前发行版的 WebSphere MQ 以及 WebSphere MQ V7.0 及后续发行版中的 WebSphere MQ Java 类之间的关系。

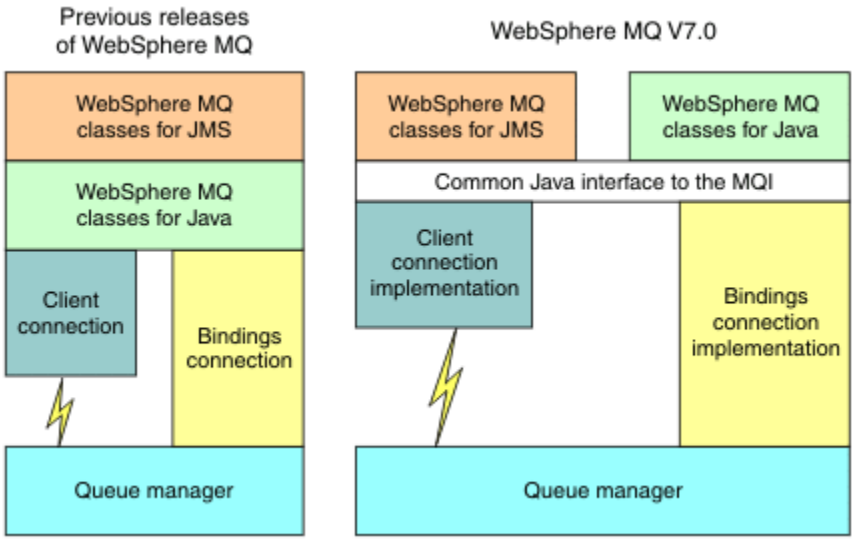


图 *126: WebSphere MQ JMS* 类与 *WebSphere MQ Java* 类之间的关系

为了保持与先前发行版的兼容性，使用 Java 编写的通道出口类仍可以将 WebSphere MQ 类用于 Java 接口，即使从 WebSphere MQ JMS 类调用这些通道出口类也是如此。 但是，使用 WebSphere MQ classes for Java 接口意味着应用程序仍依赖于 WebSphere MQ classes for Java JAR 文件 com.ibm.mq.jar。 如果不希望在类路径中使用 com.ibm.mq.jar ，那么可以改为使用 com.ibm.mq.exits 包中的新接口集。

现在，您可以使用 WebSphere MQ Explorer 来创建和配置 JMS 受管对象。

发布/预订消息传递

WebSphere MQ V7.0 和后续发行版包含嵌入式发布/预订功能。 此功能将替换 WebSphere MQ V6.0 随附的 WebSphere MQ 发布/预订。

针对 JMS 应用程序的 WebSphere MQ 类可以使用嵌入式发布/预订功能，并且可以使用该功能来代替使用 WebSphere Event Broker 或 WebSphere Message Broker 进行发布/预订消息传递，并将 WebSphere MQ 作为传输。 配置 WebSphere MQ JMS 类以使用新功能比配置 WebSphere MQ JMS 类以使用 WebSphere MQ 发布/预订， WebSphere Event Broker 或 WebSphere Message Broker 更简单。 管理员和应用程序开发者不再需要管理发布队列，订户队列，预订存储和订户清除。 此外， ConnectionFactory 和 Topic 对象具有较少数量的属性。

嵌入式发布/预订功能还提供了一些其他功能，例如保留发布和选择两个通配符方案，用于指定应用程序要预订的主题范围。

应用程序仍可以使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接来进行发布/预订消息传递。 此支持保持不变。

使用 WebSphere MQ 发布/预订的应用程序可以在升级它们所连接的队列管理器时使用嵌入式发布/预订功能而不进行更改。 将忽略由应用程序设置但嵌入式发布/预订功能不需要的属性。

WebSphere MQ 消息传递提供程序

WebSphere MQ 消息传递提供程序具有两种操作方式:

- *WebSphere MQ* 消息传递提供程序正常方式
- *WebSphere MQ* 消息传递提供程序迁移方式

WebSphere MQ 消息传递提供程序正常方式使用 WebSphere MQ V 7.0 和后续发行版队列管理器的所有功能来实现 JMS。 此方式仅用于连接到 WebSphere MQ 队列管理器，并且可以在客户机或绑定方式下连接到 WebSphere MQ Version 7.0 以及后续发行版队列管理器。 此方式已优化为使用新的 WebSphere MQ V 7.0 和后续发行版功能。

WebSphere MQ 消息传递提供程序迁移方式基于 WebSphere MQ Version 6.0 函数，并且仅使用 WebSphere MQ Version 6.0 队列管理器中提供的功能来实现 JMS。 您可以使用 WebSphere MQ 消息传递提供程序迁移方式连接到 WebSphere MQ 版本 7.0 和后续发行版队列管理器，但不能使用任何版本 7.0 优化。 此方式允许连接到以下任一队列管理器版本:

1. WebSphere MQ V 7.0 及后续队列管理器处于绑定或客户机方式，但此方式仅使用可用于 WebSphere MQ V 6.0 队列管理器的功能部件
2. 客户机方式下的 WebSphere MQ 版本 6.0 或更低版本的队列管理器

如果要使用 WebSphere MQ Enterprise Transport 连接到 WebSphere Event Broker 或 WebSphere Message Broker，请使用 WebSphere MQ 消息传递提供程序迁移方式。 如果使用 WebSphere MQ 实时传输，那么将自动选择 WebSphere MQ 消息传递提供程序迁移方式，因为您已在连接工厂对象中显式选择属性。 使用 WebSphere MQ Enterprise Transport 与 WebSphere Event Broker 或 WebSphere Message Broker 的连接遵循 用于选择 WebSphere MQ 消息传递提供程序方式的规则 中描述的方式选择的一般规则。

异步消息使用

WebSphere MQ V7.0 和任何后续发行版都支持异步消息使用。 应用程序可以为目标注册回调函数。 将适当的消息发送到目标时， WebSphere MQ 会调用该函数并将该消息作为参数传递。 然后，该函数将异步处理消息。 在 WebSphere MQ 的先前发行版中，仅当使用 WebSphere MQ JMS 类时，此功能才可用。

WebSphere MQ classes for JMS 已更改为在 WebSphere MQ V7.0 和任何后续发行版中使用此新功能。 JMS 消息侦听器的实现现在更适合 WebSphere MQ， WebSphere MQ classes for JMS 不再需要轮询目标以检查是否已将适当的消息发送到目标。 因此，将提高 JMS 消息侦听器的性能，尤其是当应用程序在会话中使用多个消息侦听器来监视多个目标时。 增加了消息吞吐量，并且减少了在消息到达目标之后将消息传递到消息侦听器所花费的时间。

消息驱动的 Bean (MDB) 具有类似的性能改进。 此外，由于 WebSphere MQ 功能的另一个增强功能，使用来自同一目标的消息的多个 MDB 现在迁到的消息争用已减少。

消息选择

除按消息标识或相关标识选择消息外， WebSphere MQ 发行版 7.0 之前的版本中的所有消息选择都由 WebSphere MQ JMS 类完成。 在 WebSphere MQ V7.0 以及任何后续发行版中，所有消息选择都由队列管理器完成。

因此，对于使用消息选择的消息的应用程序，将提高消息吞吐量。 对于以客户机方式连接的应用程序，性能改进更大，因为只有满足选择条件的那些消息才会通过网络传输，而 WebSphere MQ classes for JMS 仅处理它传递给应用程序的那些消息。

共享通信连接

在 WebSphere MQ 的先前发行版中，如果 WebSphere MQ 客户机应用程序使用同一 MQI 通道多次连接到队列管理器，那么 MQI 通道的每个实例都需要单独的 TCP 连接。 在 WebSphere MQ V7.0 和任何后续发行版中，每个使用同一 MQI 通道的队列管理器连接都可以共享单个 TCP 连接。 此安排意味着需要更少的网络资

源，并且减少了创建与队列管理器的多个连接所花费的总时间，尤其是在使用 SSL 时，因为 SSL 握手仅在 TCP 连接启动时发生一次。

WebSphere MQ JMS 类利用此增强功能。 对于以客户机方式连接到队列管理器的应用程序， WebSphere MQ classes for JMS 可能会使用指定为 ConnectionFactory 对象属性的名称的 MQI 通道创建与队列管理器的多个连接。 到队列管理器的每个连接现在都可以共享单个 TCP 连接。

客户机连接上的预读

如果应用程序使用客户机连接来使用来自目标的非持久消息，那么可以配置目标，以便 WebSphere MQ classes for JMS 在将相关消息交付到应用程序之前使用缓冲区来存储这些消息。 此优化称为 预读 ，可由通过调用 receive () 方法同步使用消息的应用程序以及异步使用消息的消息侦听器和 MDB 使用。 对于需要快速使用大量消息的目标，预读特别有效。

预读不适用于持久消息，因为如果持久消息已读入缓冲区，那么队列管理器将无法在失败后恢复消息。 但是，使用来自具有混合持久和非持久消息的目标的消息的应用程序仍可以使用预读。 保留消息的顺序，但预读的运行时优点仅适用于非持久消息。

在决定是否使用预读时，请考虑以下几点:

- 如果应用程序正在使用配置为预读的目标中的消息，并且应用程序因任何原因而结束，那么将废弃当前存储在缓冲区中的任何非持久消息。
- 如果满足以下所有条件，那么发送到会话中队列的消息可能不会按发送顺序接收:
  - 应用程序在同一会话中使用两个消息使用者来使用来自队列的消息。
  - 每个消息使用者对队列使用不同的 Destination 对象。
  - 为预读配置任何或两个 Destination 对象。

发送消息

当应用程序将消息发送到目标时，可以配置目标，以便当应用程序调用 send () 时， WebSphere MQ classes for JMS 将消息转发到队列管理器，并将控制权交还给应用程序，而无需确定队列管理器是否已安全接收消息。 WebSphere MQ classes for JMS 只能以此方式用于非持久消息以及在事务性会话中发送的持久消息。

对于在事务性会话中发送的持久消息，应用程序最终确定队列管理器在调用 commit () 时是否安全地接收了消息。 对于在未处理的会话中发送的任何消息， ConnectionFactory 对象的 SENDCHECKCOUNT 属性指定在 WebSphere MQ classes for JMS 检查队列管理器是否已安全接收消息之前要发送的消息数。

此优化对于以客户机方式连接到队列管理器的应用程序最为有利，需要快速连续发送一系列消息，但不需要队列管理器对发送的每条消息立即进行反馈。

通道出口

从 WebSphere MQ classes for JMS 调用时，使用 C 或 C++ 编写的通道出口程序现在的行为方式与从 Websphere MQ MQI 客户机调用这些程序时的行为方式相同。 使用 Java 编写的通道出口类的性能已提高，您现在可以使用 com.ibm.mq.exits 包中的一组新接口而不是使用 WebSphere MQ classes for Java 中的接口来编写通道出口类。

消息属性

JMS 消息由一组头字段，一组属性以及包含应用程序数据的主体组成。 至少， WebSphere MQ 消息由消息描述符和应用程序数据组成。

当 WebSphere MQ classes for JMS 应用程序发送 JMS 消息时， WebSphere MQ classes for JMS 将 JMS 消息映射到 WebSphere MQ 消息中。 某些 JMS 头字段和属性映射到消息描述符中的字段，而某些 JMS 头映射到称为 MQRFH2 头的其他 WebSphere MQ 头中的字段。 当 WebSphere MQ classes for JMS 应用程序接收 JMS 消息时， WebSphere MQ classes for JMS 将执行反向映射。

因此，使用 MQI 从 WebSphere MQ classes for JMS 应用程序接收消息的应用程序必须能够处理 MQRFH2 头。 如果应用程序无法处理 MQRFH2 头，那么可以设置 Destination 对象的 TARGCLIENT 属性，以指示

WebSphere MQ JMS 类在 WebSphere MQ 消息中不包含 MQRFH2 头。 但是，通过排除 MQRFH2 头，某些 JMS 头字段和属性中包含的信息将丢失。

同样，使用 MQI 将消息发送到 WebSphere MQ classes for JMS 应用程序的应用程序必须在每条消息中包含 MQRFH2 头。 如果未包含 MQRFH2 头，那么 WebSphere MQ classes for JMS 只能设置可从消息描述符中的字段派生的那些 JMS 头字段和属性。

WebSphere MQ V7.0 为使用 MQI 从 WebSphere MQ JMS 应用程序接收消息并向其发送消息的应用程序提供了一些其他支持。

当应用程序调用 MQGET 以从 WebSphere MQ JMS 应用程序类接收消息时，该应用程序可以选择通过下列其中一种方式接收消息:

1. 此消息通过消息描述符，包含从 JMS 头字段和属性派生的数据的 MQRFH2 头以及应用程序数据进行交付。
2. 消息随消息描述符，应用程序数据和一组消息属性一起交付。

在选项 2 中，每个消息属性表示一个 JMS 头字段或最初由 WebSphere MQ JMS 类映射到 MQRFH2 头中的字段的属性。 在 MQGET 调用之后，应用程序可以使用 MQINQMP 调用来获取消息属性的值。 使用选项 2 而不是选项 1 接收消息可通过以下方式简化应用程序逻辑:

- 应用程序不必解析 MQRFH2 头的变量部分，该头包含以类似 XML 的格式编码的 JMS 头字段和属性数据。
- 应用程序不必转换 MQRFH2 头的变量部分中的字符数据。

相应地，在应用程序调用 MQPUT 以将消息发送到 WebSphere MQ classes for JMS 应用程序之前，应用程序可以使用 MQSETMP 调用来设置消息属性的值，而不是构造 MQRFH2 头。

可维护性

WebSphere MQ classes for JMS 包含许多与可维护性相关的改进:

- 跟踪。

  WebSphere MQ classes for JMS 包含应用程序可用于控制跟踪的类。 应用程序可以启动和停止跟踪，在跟踪中指定所需的详细信息级别，并以各种方式定制跟踪输出。

- 日志记录。

  WebSphere MQ classes for JMS 维护日志文件，其中包含有关需要更正的错误的消息。 这些消息以纯文本编写。 WebSphere MQ classes for JMS 包含一个类，应用程序可以使用该类来指定日志文件的位置及其最大大小。

- 首次故障支持技术 (FFST)。

  如果发生严重故障，那么 WebSphere MQ classes for JMS 会在 FDC 文件中生成 FFST 报告。 FFST 报告包含 IBM 服务人员可用于更快地诊断问题的信息。

- 版本信息。

  WebSphere MQ classes for JMS 包含一个类，应用程序可以使用该类来查询 WebSphere MQ classes for JMS 的版本。

- 异常消息。

  已增强异常消息，以提供有关错误原因和更正错误所需的操作的更多信息。

- 应用程序服务器。

  WebSphere MQ JMS 类的可维护性功能与 WebSphere Application Server 的可维护性功能的集成已改进。

MQC 已替换为 MQConstants

IBM WebSphere MQ V 7.0 随附了新软件包 com.ibm.mq.constants。 此包包含类 MQConstants ，它实现了许多接口。 MQConstants 包含 MQC 接口中的所有常量以及许多新常量的定义。 此包中的接口紧跟 IBM WebSphere MQ 中使用的常量头文件的名称。

For example, the interface CMQC contains a constant MQOO_INPUT_SHARED; this interface and constant correspond to the header file `cmqc.h` and the constant MQOO_INPUT_SHARED.

com.ibm.mq.constants 可以与 IBM WebSphere MQ classes for Java 和 IBM WebSphere MQ classes for JMS 一起使用。

MQC 仍然存在，并且具有先前具有的常量。 但是，对于任何新应用程序，必须使用 com.ibm.mq.constants 包。

为 JMS 应用程序编写 WebSphere MQ 类

在简要介绍 JMS 模型之后，本主题提供了有关如何为 JMS 应用程序编写 WebSphere MQ 类的详细指导。

JMS 模型

JMS 模型定义一组 Java 应用程序可用于执行消息传递操作的接口。 WebSphere MQ JMS 类作为 JMS 提供程序，定义 JMS 对象如何与 WebSphere MQ 概念相关。 JMS 规范期望将某些 JMS 对象作为受管对象。

JMS 规范和 javax.jms 包定义了一组接口， Java 应用程序可以使用这些接口来执行消息传递操作。 以下列表概述了主要 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 674 页的图 127 显示了这些对象及其关系。

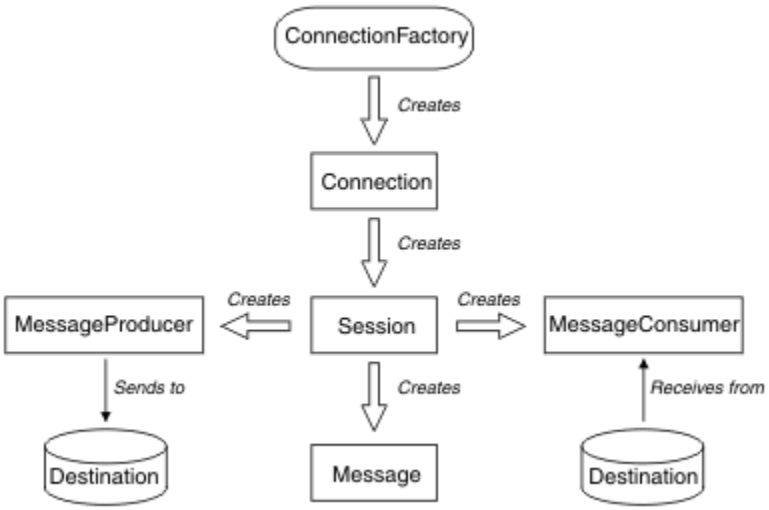


图 *127: JMS* 对象及其关系

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 支持两种样式的消息传递:

- 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 1.1 之前的 JMS 版本，点到点域的编程使用一组接口和方法，而发布/预订域的编程使用另一组接口和方法。 二者相似，但相互独立。 通过 JMS 1.1，您可以使用一组支持这两个消息传递域的公共接口和方法。 通用接口为每个消息传递域提供独立于域的视图。 第 675 页的表 103 列出了独立于 JMS 域的接口及其相应的特定于域的接口。

表 *103:* 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 1.1 保留所有特定于域的接口，因此现有应用程序仍可使用这些接口。 但是，对于新应用程序，请考虑使用独立于域的接口。

在 WebSphere MQ JMS 类中， JMS 对象通过以下方式与 WebSphere 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 WebSphere MQ 连接句柄，因此它定义了会话的事务作用域。
- MessageProducer 对象和 MessageConsumer 对象都封装了 WebSphere MQ 对象句柄。

使用 WebSphere MQ classes for JMS 时，将应用 WebSphere MQ 的所有正常规则。 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 规范期望 ConnectionFactory 和 Destination 对象是受管对象。 管理员在中央存储库中创建并维护受管对象， JMS 应用程序使用 Java 命名和目录接口 (JNDI) 检索这些对象。

在 WebSphere MQ JMS 类中， 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 WebSphere MQ JMS 类中，以下类型的对象可以是受管对象:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- 队列
- Topic
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

JMS 消息

JMS 消息由头，属性和主体组成。 JMS 定义五种类型的消息体。

JMS 消息由以下部分组成:

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 定义了几种类型的消息体，这些消息体涵盖了当前正在使用的大多数消息传递样式。

JMS 定义五种类型的消息体:

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 基本类型。 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 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。 应用程序可以使用消息选择器来获取 JMS 提供程序过滤器消息。

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。 属性允许使用消息选择器的应用程序使用特定于应用程序的条件，让 JMS 提供程序代表其选择或过滤消息。 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 不会为属性中可能包含的内容定义策略。 但是，应用程序开发者必须知道 JMS 提供程序可能比消息属性中的数据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通过使用消息头来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。

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'。 与 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 长整型范围内的数字。
  - 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 双精度值范围内的数字。
  - Boolean 文字 TRUE 和 FALSE。
- 标识符：
  - 标识是 Java 字母和 Java 数字的无限制长度序列，其中第一个必须是 Java 字母。 字母是 Character.isJavaLetter 方法返回 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 开头的任何名称都是 JMS 定义的属性名。
  - 以 JMS_ 开头的任何名称均为特定于提供程序的属性名称。
  - 任何不以 JMS 开头的名称都是特定于应用程序的属性名。 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。
- 空格与为 Java 定义的空格相同: 空格，水平制表符，换页符和行终止符。
- 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。 (所需类型转换由 Java 数字提升规则定义。) 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 is equivalent to 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。

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"
```

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。

虽然 SQL 支持固定十进制比较和算术，但 JMS 消息选择器不支持。 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 *WebSphere MQ* 消息

WebSphere MQ 消息由消息描述符，可选的 MQRFH2 头和主体组成。 JMS 消息的内容将部分映射并部分复制到 WebSphere MQ 消息。

本主题描述如何将本节第一部分中描述的 JMS 消息结构映射到 WebSphere MQ 消息。 希望在 JMS 和传统 WebSphere MQ 应用程序之间传输消息的程序员很感兴趣。 对于希望处理在两个 JMS 应用程序 (例如，在 WebSphere Message Broker 实现中) 之间传输的消息的人员来说，这也很重要。

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。 当应用程序使用实时连接时，将直接通过 TCP/IP 执行所有通信; 不涉及任何 WebSphere MQ 队列或消息。

WebSphere MQ 消息由三个组件组成:

- WebSphere MQ 消息描述符 (MQMD)
- WebSphere MQ MQRFH2 头
- 消息体。

MQRFH2 是可选的，其包含在外发消息中由 JMS 目标类中的标志控制。 可以使用 WebSphere MQ JMS 管理工具来设置此标志。 由于 MQRFH2 包含特定于 JMS 的信息，因此当发送方知道接收目标是 JMS 应用程序时，请始终将其包含在消息中。 通常，将消息直接发送到非 JMS 应用程序时，请省略 MQRFH2 。 这是因为此类应用程序在其 WebSphere MQ 消息中不期望 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 679 页的图 128 显示了如何将 JMS 消息的结构变换为 WebSphere MQ 消息并重新返回:

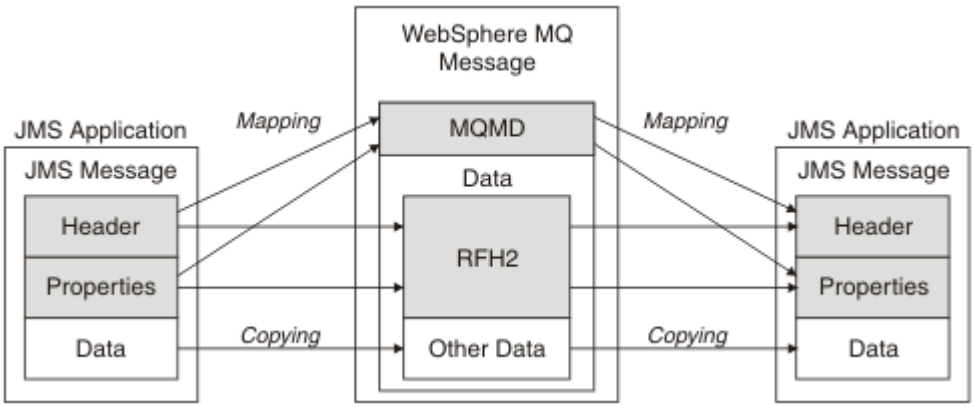


图 *128:* 如何使用 *MQRFH2* 头在 *JMS* 与 *WebSphere MQ* 之间变换消息

结构转换方式有两种:

映射
如果 MQMD 包含与 JMS 字段等效的字段，那么会将 JMS 字段映射到 MQMD 字段。 其他 MQMD 字段显示为 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
如果没有等效的 MQMD ，那么会将 JMS 头字段或属性作为 MQRFH2 中的字段传递 (可能已变换)。

MQRFH2 头和 *JMS*
此主题集合描述 MQRFH 版本 2 头，该头携带与消息内容关联的特定于 JMS 的数据。 MQRFH2 版本 2 是可扩展头，并且还可以包含未与 JMS 直接关联的其他信息。 但是，此部分仅涵盖 JMS 对其的使用。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
固定部分是根据 标准 WebSphere MQ 头模式建模的，由以下字段组成:

StrucId (MQCHAR4)
结构标识。

必须为 MQRFH_STRUC_ID（值：“RFH ”）（初始值）。

MQRFH_STRUC_ID_ARRAY (值: “R”，“F”，“H”，“ ”) 也已定义。

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。）

表 *104: NameValueCCSID* 字段的可能的值

值	含义
1200	UCS2 开放式
1208	UTF8
13488	UCS2 2.0 子集
17584	UCS2 2.1 子集（包含欧元符号 €）

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。 JMS 创建的 MQRFH2 头可以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。

它始终存在于包含从 WebSphere Message Broker 发送的 MQRFH2 的消息中。 它描述消息的域、格式、类型和消息集。

表 *105: 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>

表 *105: 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.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	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 文件夹始终存在于 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 包含仅供 WebSphere Application Server 使用的属性。 仅当应用程序至少设置了一个 IBM 定义的属性时，才会显示该文件夹。

表 *106: 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>

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 WebSphere 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 *107: 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>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>

表 *107: 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**
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 682 页的表 108 显示了属性名称的完整列表。

表 *108: 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	长整型	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	长整型	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 682 页的表 108。

datatype
折叠后，必须匹配第 683 页的表 109 中列出的其中一个数据类型。

value
是要使用第 683 页的表 109 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 *109:* 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	Floating point number, magnitude <= 3.40282347E+38, >= 1.175E-37 expressed using digits 0..9, optional sign, optional fractional digits, optional exponent
r8	Floating point number, magnitude <= 1.7976931348623E+308, >= 2.225E-307 expressed using digits 0..9, optional sign, optional fractional digits, optional exponent

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
这些表显示等同于 JMS 头字段， JMS 属性和特定于 JMS 提供程序的属性的 MQMD 字段。

第 684 页的表 110 列出了 JMS 头字段， 第 684 页的表 111 列出了直接映射到 MQMD 字段的 JMS 属性。 第 684 页的表 112 列出特定于提供程序的属性及其映射到的 MQMD 字段。

表 *110: JMS* 头字段映射到 *MQMD* 字段

JMS 头字段	**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 *111: JMS* 属性映射到 *MQMD* 字段

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 *112:* 特定于 *JMS* 提供程序的属性映射到 *MQMD* 字段

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
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	字符串	Format 第 685 页的『1』	MQCHAR8

表 112: 特定于 JMS 提供程序的属性映射到 MQMD 字段 (继续)			
特定于 JMS 提供程序的属性	**Java 类型**	**MQMD 字段**	**C 类型**
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	编码	MQLONG
JMS_IBM_Character_Set	字符串	CodedCharacterSetId第 685 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 WebSphere 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 属性值是一个字符串值，其中包含与数字 CodedCharacterSetId 值等效的 Java 字符集。 MQMD 字段 CodedCharacterSetId 是一个数字值，其中包含由 JMS_IBM_Character_Set 属性指定的等效 Java 字符集字符串。

将 *JMS* 字段映射到 *WebSphere MQ* 字段 *(*外发消息*)*
这些表显示如何在 send () 或 publish () 时间将 JMS 头和属性字段映射到 MQMD 和 MQRFH2 字段。

第 685 页的表 113 显示了如何在 send () 或 publish () 时间将 JMS 头字段映射到 MQMD/RFH2 字段。 第 686 页的表 114 显示如何在 send () 或 publish () 时间将 JMS 属性映射到 MQMD/RFH2 字段。 第 686 页的表 115 显示如何在 send () 或 publish () 时间将特定于 JMS 提供程序的属性映射到 MQMD 字段。

对于标记为 "由消息对象设置" 的字段，传输的值是紧接在 send () 或 publish () 操作之前的 JMS 消息中保留的值。 操作将保留 JMS 消息中的值不变。

对于 "按发送方法设置" 标记的字段，将在执行 send () 或 publish () 时指定值 (将忽略 JMS 消息中保留的任何值)。 将更新 JMS 消息中的值以显示所使用的值。

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 113: 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
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	消息对象
JMSType		MQRFH2	消息对象
JMSRedelivered			仅接收

表 113: 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

注:

1. MQMD 字段 CodedCharacterSetId 是一个数字值，其中包含由 JMS_IBM_Character_Set 属性指定的等效 Java 字符集字符串。

表 114: 外发消息 *JMS* 属性映射

JMS 属性名	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

表 115: 特定于外发消息 *JMS* 提供程序的属性映射

特定于 JMS 提供程序的属性名	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
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 *send()* 或 *publish()* 中映射 *JMS* 头字段
这些注释与 send () 或 publish () 处 JMS 字段的映射相关。

JMSDestination 到 MQRFH2
这存储为一个字符串，用于序列化目标对象的显着特征，以便接收 JMS 可以重新构成等效的目标对象。MQRFH2 字段编码为 URI（请参阅 第 738 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 738 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 等同于 MQPER_PERSISTENT
- JMS 值 NON_PERSISTENT 等同于 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
从 JMS 发送的所有消息都具有由 WebSphere MQ 分配的唯一消息标识。 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。 WebSphere MQ messageId 是 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 提供了可以设置为禁止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 751 页的『从 WebSphere MQ JMS 应用程序类读取和写入消息描述符』中描述的其中一个 WebSphere 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。 另请参阅第 772 页的『使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接』。

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 WebSphere 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

以下说明指的是特定于 JMS 提供程序的字段到 IBM WebSphere MQ 消息的映射。

JMS_IBM_Report_<name> 到 MQMD 报告
JMS 应用程序可以使用以下 JMS_IBM_Report_XXX 属性来设置 MQMD 报告选项。 单个 MQMD 将映射到多个 JMS_IBM_Report_XXX 属性。 应用程序必须将这些属性值设置为标准 IBM WebSphere MQ MQRO_ 常量（包含在 com.ibm.mq.MQC 中）。 例如，要使用全部数据请求 COD，应用程序必须将 JMS_IBM_Report_COD 设置为 CMQC.MQRO_COD_WITH_FULL_DATA 值。

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

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 *WebSphere MQ* 字段映射到 *JMS* 字段 *(*入局消息*)*
这些表显示如何在 get () 或 receive () 时间将 JMS 头和属性字段映射到 MQMD 和 MQRFH2 字段。

第 689 页的表 116 显示了如何在 get () 或 receive () 时间将 JMS 头字段映射到 MQMD/MQRFH2 字段。 第 690 页的表 117 显示如何在 get () 或 receive () 时间将 JMS 属性字段映射到 MQMD/MQRFH2 字段。 第 690 页的表 118 显示如何映射特定于 JMS 提供程序的属性。

表 *116:* 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 690 页的『1』
JMSDeliveryMode	持久性第 690 页的『2』	jms.Dlv 第 690 页的『2』
JMSExpiration		jms.Exp
JMSPriority	优先级	
JMSMessageID	MsgID	

表 116: 入局消息 *JMS* 头字段映射 (继续)

JMS 头字段名称	MQMD 字段检索自	MQRFH2 字段检索自
JMSTimestamp	PutDate 第 690 页的『2』 PutTime 第 690 页的『2』	jms.Tms 第 690 页的『2』
JMSCorrelationID	CorrelId 第 690 页的『2』	jms.Cid 第 690 页的『2』
JMSReplyTo	应答队列 第 690 页的『2』 应答队列管理器 第 690 页的『2』	jms.Rto 第 690 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 属性值是一个字符串值，其中包含与数字 CodedCharacterSetId 值等效的 Java 字符集。

表 117: 入局消息属性映射

JMS 属性名	MQMD 字段检索自	MQRFH2 字段检索自
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId 第 690 页的『1』	jms.Gid 第 690 页的『1』
JMSXGroupSeq	MsgSeqNumber 第 690 页的『1』	jms.Seq 第 690 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 118: 特定于入局消息提供程序的 *JMS* 属性映射

JMS 属性名	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	

表 118: 特定于入局消息提供程序的 JMS 属性映射 (继续)		
JMS 属性名	**MQMD 字段检索自**	**MQRFH2 字段检索自**
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 691 页的『1』	编码	
JMS_IBM_Character_Set 第 691 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 *WebSphere MQ* 应用程序之间交换消息
本主题描述 JMS 应用程序与无法处理 MQRFH2 头的传统 WebSphere MQ 应用程序交换消息时发生的情况。

. 第 691 页的图 129 显示了映射。

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 WebSphere 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 MQMD 的映射以传统 WebSphere MQ 应用程序为目标，与以 JMS 应用程序为目标的从 JMS 到 MQMD 的映射相同。 如果 WebSphere 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 WebSphere 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，它将作为 JMS 字节消息接收。 由于没有 MQRFH2，因此只能复原 MQMD 中传输的那些 JMS 属性。

如果 WebSphere MQ JMS 类接收到没有 MQRFH2 头的消息，那么缺省情况下，从消息的 JMSReplyTo 头字段派生的 Queue 或 Topic 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。

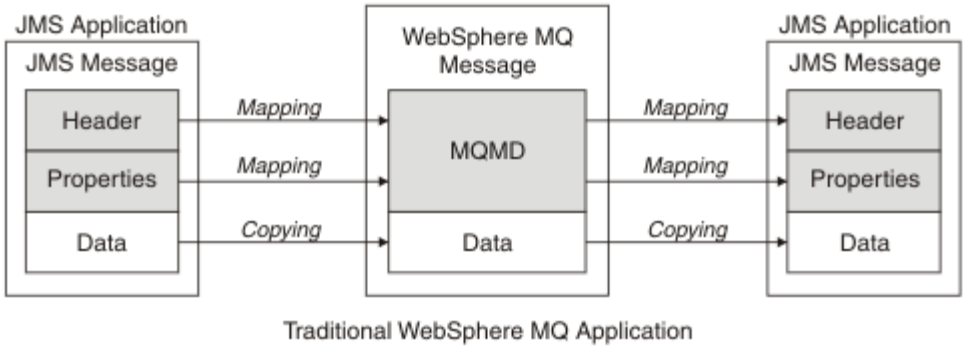


图 *129:* 如何将 *JMS* 消息变换为没有 *MQRFH2* 头的 *WebSphere MQ* 消息

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 运行时以正常方式序列化的对象。

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 WebSphere MQ classes for JMS 支持的任何其他字符集。 此类字符集主要在您向非 JMS 应用程序发送消息时使用。

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。 这些规范位于最后一个 WebSphere MQ 头 (如果没有头，那么为 MQMD) 中。 对于 JMS 消息，最后一个头通常是 MQRFH2。

BytesMessage
缺省情况下， BytesMessage 是由 JMS 1.0.2 规范和关联的 Java 文档定义的字节序列。

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，您可以请求以 S/390 而不是 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。 此规范位于最后一个 WebSphere MQ 头 (如果没有头，那么为 MQMD) 中。 对于 JMS 消息，最后一个头通常是 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。 此字段以 Java UTF8 格式进行编码，并以 2 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 WebSphere MQ 消息中的字符集值不会影响将该消息解释为 JMS BytesMessage。

非 Java 应用程序不大可能识别 Java UTF8 编码。 因此，要使 JMS 应用程序发送包含文本数据的 BytesMessage ，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 683 页的表 109 中列出的其中一个数据类型。 缺省数据类型为 `string`，因此将对字符串元素省略属性 `dt="string"` 。

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。

WebSphere MQ classes for JMS 早于 V 5.3 的版本按以下格式对映射消息的主体进行了编码：

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

WebSphere MQ classes for JMS V 5.3 和更高版本可以解释任一格式，但 WebSphere MQ classes for JMS V 5.3 之前的版本无法解释当前格式。

如果应用程序需要将映射消息发送到使用版本低于 V 5.3 的 WebSphere MQ classes for JMS 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 683 页的表 109 中列出的其中一个数据类型。 缺省数据类型为 string，因此将对字符串元素省略属性 dt="string" 。

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

JMS 中的消息数据转换在发送和接收消息时执行。 WebSphere MQ 自动执行大多数数据转换。 在 JMS 应用程序之间传输消息时，它会转换文本和数字数据。 在 JMS 应用程序与 WebSphere 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 WebSphere MQ 应用程序与 JMS 应用程序之间传输非文本消息。
- 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

数据转换是在应用程序之间 (甚至在两个 JMS 应用程序之间) 交换文本和数字数据所必需的。 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。 WebSphere MQ 管理 JMS 消息 (JMSObjectMessage 除外) 中文本和数字的编码，请参阅 第 699 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 消息的 JMS 头 MQRFH2 字段中。 如果消息类型是 MQ，而不是 JMS 类型的消息，那么属性将存储在消息描述符 MQMD 中。 这些属性用于转换 JMS 消息数据。 JMS 消息数据在 WebSphere 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) 在 JMS 消息的 MQRFH2 头部分中交换，除非已在没有 MQRFH2 的情况下发送该消息。 只有 JMSTextMessage 和 JMSBytesMessage 可以在没有 MQRFH2 的情况下发送。 如果 JMS 属性作为 WebSphere MQ 消息属性存储在消息描述符 MQMD 中，那么它将作为 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 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
许多数据转换方法都向 JMS 应用程序设计者开放。 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。 如果应用程序仅交换文本或仅与其他 JMS 应用程序交换消息，那么通常不考虑数据转换。 数据转换由 WebSphere MQ 自动执行。

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?

在某些情况下 (例如， JMS 到 JMS 消息传输以及与 IBM WebSphere MQ 程序交换文本消息) ， IBM WebSphere MQ 会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 694 页的『JMS 客户机数据转换』
2. 第 694 页的『应用程序数据转换』
3. 第 695 页的『队列管理器数据转换』
4. 第 695 页的『消息通道数据转换』

应在何处执行转换?

第 696 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。 “接收方正常” 也适用于 JMS 数据转换。

JMS 客户机数据转换

JMS 客户机 [4] 数据转换是在将 Java 原语和对象发送到目标时将其转换为 JMS 消息中的字节，并在接收到 Java 原语和对象时再次进行转换。 JMS 客户机数据转换使用 JMSMessage 类的方法。 JMSMessage 类类型在第 696 页的表 119 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 “接收方生成良好” 模式，或者如果要在 JMS 应用程序之间交换消息，那么应用程序不需要设置 JMS_IBM_ENCODING。 在大多数情况下，可将 Encoding 属性保留为 Native。

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。 如果要应用的目标字符集属性，那么 JMS 客户机应用程序在发送消息之前不必设置 JMS_IBM_CHARACTER_SET 。

要获取消息中的数据，应用程序将调用 JMS 消息读取或获取方法。 这些方法引用先前消息头中定义的代码页和编码，以正确创建 Java 原语和对象。

JMS 客户机数据转换满足在一个 JMS 客户机与另一个 JMS 客户机之间交换消息的大多数 JMS 应用程序的需求。 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 705 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 类执行显式字符数据转换; 请参阅 第 698 页的图 132 和 第 698 页的图 133 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。 通常将消息作为 JMSBytesMessage 发送或接收，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 [5] 也可以使用 JMSStreamMessage、JMSMapMessage 或 JMSObjectMessage 发送和接收字节。

没有 Java 方法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。

---

[4] “JMS 客户机” 是指用于 JMS 的 WebSphere MQ 类，这些类实现以客户机或绑定方式运行的 JMS 接口。

[5] 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。

应用程序数据转换的典型用途是 JMS 客户机从非 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 应用程序将消息格式指定为 “MQSTR ”，否则消息将构造为 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。 有两种方法可以将格式化消息转换为易于使用的 Java 对象。

1. 构造对应于记录的 Java 类，以封装读取和写入消息。 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 711 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

在 WebSphere MQ V7.0 中，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 698 页的图 131。 队列管理器对正在接收消息的 JMS 客户机程序执行转换，如果 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 695 页的图 130。

在 V7.0 之前，转换始终由 JMS 客户机执行。 JMS 客户机数据转换仅限于转换 JMS 客户机已知的类型和长度的数字和文本序列。 它无法转换数据结构；请参阅第 711 页的『与非 JMS 应用程序交换格式化记录』。 在 V7.0 中，在修订包 7.0.1.5 之前，如果队列管理器可执行转换，那么将始终由队列管理器执行。 从 7.0.1.5 开始，缺省转换行为将还原为与 V6.0 相同，并且所有转换都由 JMS 客户机执行。 从具有 APAR IC72897 的 7.0.1.5 或 7.0.1.4 开始，可设置新的目标选项 WMQ_RECEIVE_CONVERSION，来控制执行转换的位置；设置 WMQ_RECEIVE_CCSID，来设置目标代码页；请参阅第 695 页的图 130。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

或者，

```
((MQDestination)destination).setReceiveConversion
             (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

图 *130:* 启用队列管理器数据转换

与非 JMS 应用程序交换消息时，队列管理器转换的主要优点。 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某个预定义 WebSphere MQ 消息类型 (例如 CICS 网桥头 (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 客户机不需要执行转换。 非 JMS 接收程序依赖于转换出口，以确保以所需的代码页和编码传递消息。 对于发送 JMS 客户机和非 JMS 接收方，此示例适用于 IBM WebSphere MQ pre- and post-V7.0。 通过 IBM WebSphere MQ V7.0，还可以对接收 JMS 程序调用转换出口。

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 将其作为 Java 类进行访问，并使用自己的转换出口对其进行转换。 在 z/OS 上，该实用程序称为 **CSQUCVX**，在 IBM i 上称为 **CVTMQMDTA**。 请参阅第 711 页的『与非 JMS 应用程序交换格式化记录』，

消息通道数据转换

WebSphere MQ 发送方，服务器，集群接收方和集群发送方通道具有消息转换选项 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”

WebSphere MQ 应用程序设计中用于代码转换的通常方法是 “接收方实现良好”。 “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。

“接收方良好” 也是 JMS 客户机应用程序的良好一般指导。 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，那么转换为接收方所需的字符集可能会使非 JMS 接收方不需要执行转换。 如果接收方是 JMS 客户机，那么无论如何都将再次转换以解码消息数据并创建 Java 原语和对象。

JMS 客户机应用程序与使用语言 (例如 C) 编写的应用程序之间的区别在于 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，可以设置 WebSphere MQ 用于对消息中的数字和文本进行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`。

WebSphere MQ 不会转换字节数组。 要将字符串和字符数组编码到字节数组，请使用 `java.nio.charset` 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。 对字符串和字符数组编码时，依靠 `java.nio.charset.Charset.defaultCodePage` 不是一个好办法。 缺省值 `Charset` 通常为 `windows-1252` (在 Windows 上) 和 `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` 。 如果要与 JMS 应用程序交换包含数字或文本的消息，请选择适合您用途的 `JMSTextMessage`, `JMSStreamMessage`， `JMSMapMessage` 或 `JMSObjectMessage` 消息类型之一。 没有要完成的任何其他任务。

如果要与使用记录格式的非 JMS 应用程序交换消息，那么更复杂。 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` 来避免 IBM WebSphere MQ JMS 类向消息数据添加其他头和标记信息。 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 711 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 *119:* 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`

表 *119:* 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**
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 *131: amqsget0.c* 的代码片段

在 JMSBytesMessage 中发送和接收文本

第 698 页的图 132 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。 要接收包含混合类型的文本字符串 (以字节计) 消息，必须知道该字符串的长度 (以字节计)，在 第 698 页的图 133 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 *132:* 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 *133:* 从 *JMSBytesMessage* 接收 *String*

相关概念

JMS 客户机消息转换和编码
列出了用于执行 JMS 客户机消息转换和编码的方法，以及每种类型的转换的代码示例。

队列管理器数据转换
队列管理器数据转换始终可用于接收来自 JMS 客户机的消息的非 JMS 应用程序。 从 V7.0 开始，接收消息的 JMS 客户机也使用队列管理器数据转换。 从 7.0.1.5 或具有 APAR IC72897 的 7.0.1.4 开始，队列管理器数据转换功能是可选功能。

相关任务

与非 JMS 应用程序交换格式化记录

请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使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 与非 JMS 应用程序交换格式化消息时可以调用数据转换出口，也可以不调用数据转换出口。

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。 针对 JMS 消息类型 JMSObjectMessage， JMSTextMessage，JMSMapMessage， JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型的交互。

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。 针对 JMS 消息类型 JMSObjectMessage， JMSTextMessage，JMSMapMessage， JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型的交互。

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 的缺点是只能与其他 JMS 应用程序进行交换。 通过选择其他某个 JMS 消息类型，可以与非 JMS 应用程序交换 JMS 消息。

第 701 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 WebSphere 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 ，并且将在 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 701 页的『发送和接收 JMSTextmessage』中的示例。

当 JMS 客户机调用时， consumer.receive 方法队列管理器转换是可选的。 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。 在将消息传输到 JMS 客户机之前，队列管理器会从为消息指定的 JMS_IBM_CHARACTER_SET 转换文本消息。 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 701 页的『发送和接收 JMSTextmessage』中的示例。

可以在没有 JMS MQRFH2 头的 MQ 样式消息体中发送 JMSTextMessage 。 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。 如果尝试将消息作为 JMSTextMessage 接收，那么会导致异常 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。 如果启用了队列管理器转换，那么队列管理器将转换文本，但 JMS 客户机仍必须将其作为字节数组在 JMSBytesMessage 中接收。

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

这两种 JMS 消息类型相似。 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 703 页的『消息类型和转换类型表』。 第 705 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 691 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 702 页的图 138 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 Class JMSStreamMessage 以获取详细信息。 第 702 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　　”。 如果目标属性 WMQ_RECEIVE_CONVERSION 设置为 WMQ_RECEIVE_CONVERSION_QMGR，那么消息数据在发送到 JMS 客户机之前由队列管理器进行转换。 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 703 页的『消息类型和转换类型表』。 第 699 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 客户机可以使用 Charset 类将文本编码和解码为字节。 它在 JVM 和消息之间传输字节，而不需要 WebSphere MQ classes for JMS 执行任何转换; 请参阅 第 702 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。 如果将其发送到 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。

如果使用 MQ 样式主体发送 JMSBytesMessage ，那么可以从定义 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。

无论消息是具有 MQ 还是 JMS 主体样式，接收消息的方式都不受设置 WMQ_TARGET_DEST 的影响。

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。 要将 com.ibm.mq.header 包与 JMS 配合使用，需要基于示例 第 702 页的『使用 DataInputStream 和 DataOutputStream 读写消息』的代码。

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 *134:* 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 *135:* 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 *136:* 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 *137:* 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 *138:* 在 *JMSStreamMessage* 和 *JMSMapMessage* 中发送数据

在 JMSBytesMessage 中发送和接收文本

第 702 页的图 139 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。 要接收包含混合类型的文本字符串 (以字节计) 消息，必须知道该字符串的长度 (以字节计)，在 第 702 页的图 140 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 *139:* 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 *140:* 从 *JMSBytesMessage* 接收 *String*

使用 DataInputStream 和 DataOutputStream 读写消息

第 703 页的图 141 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 *141:* 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 时不起作用。 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 703 页的图 142 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 *142:* 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。 在输入 JMS_IBM_CHARACTER_SET 上是 Java 代码页，而不是数字编码字符集标识。

消息类型和转换类型表

表 *120:* 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			

表 120: 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**
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

许多数据转换方法都向 JMS 应用程序设计者开放。 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。 如果应用程序仅交换文本或仅与其他 JMS 应用程序交换消息，那么通常不考虑数据转换。 数据转换由 WebSphere MQ 自动执行。

JMS 客户机消息转换和编码

列出了用于执行 JMS 客户机消息转换和编码的方法，以及每种类型的转换的代码示例。

队列管理器数据转换

队列管理器数据转换始终可用于接收来自 JMS 客户机的消息的非 JMS 应用程序。 从 V7.0 开始，接收消息的 JMS 客户机也使用队列管理器数据转换。 从 7.0.1.5 或具有 APAR IC72897 的 7.0.1.4 开始，队列管理器数据转换功能是可选功能。

相关任务

与非 JMS 应用程序交换格式化记录

请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使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 与非 JMS 应用程序交换格式化消息时可以调用数据转换出口，也可以不调用数据转换出口。

JMS 客户机消息转换和编码

列出了用于执行 JMS 客户机消息转换和编码的方法，以及每种类型的转换的代码示例。

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。 该转换称为 JMS 客户机数据转换，以将其与队列管理器数据转换和应用程序数据转换区分开来。 当从 JMS 消息读取数据或将数据写入 JMS 消息时，将严格执行转换。 文本可与内部 16 位 Unicode 表示相互转换 [6] 为用于消息文本的字符集。 将数字数据转换为 Java 基本数字类型，并将其转换为为为消息定义的编码。 是否执行转换以及执行的转换类型取决于 JMS 消息类型以及读或写操作。

第 705 页的表 121 按执行的转换类型对不同 JMS 消息类型的读写方法进行分类。 下表后的文本描述了转换类型。

表 *121:* 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

[6] 某些 Unicode 表示的长度必须超过 16 位。 请参阅 Java SE 参考。

表 121: 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
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。[7].

第 708 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 708 页的图 147 中的代码片段可接收发送到目标的消息。 消息中的文本将从代码页 IBM037 转换回 Unicode。

注: 检查文本是否转换为编码字符集 37 的简单方法是使用 WebSphere MQ Explorer。 浏览队列并显示消息属性，然后进行检索。

对比第 708 页的图 146 中的代码片段与第 706 页的图 143 中不正确的代码片段。 在不正确的片段中，文本字符串将转换两次，一次由应用程序转换，再次由 WebSphere MQ 转换。

```
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 *143:* 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。 在接收时，消息中的数字将正确转换为数字 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。

[7] 接收消息时， JMS_IBM_CHARACTER_SET 是 Java Charset 代码页名称。

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。 可由设置消息属性 JMS_IBM_ENCODING 的应用程序覆盖消息的目标编码; 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。 它们将数字转换为 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 708 页的图 148 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 708 页的图 148 中的示例与第 709 页的图 149 中的示例进行比较。 差异在于必须在 JMSBytesMessage 中设置 JMS_IBM_ENCODING。

注: 检查数字是否正确编码的简单方法是使用 WebSphere 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 709 页的『在 JMSBytesMessage 中发送和接收文本』中一样。

WebSphere MQ 不会在客户机中执行任何代码页转换，因为消息是 JMSBytesMessage，并且因为使用了 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以下约定: 使用消息中的 JMS_IBM_CHARACTER_SET 属性将表示消息中文本的任何字节数组转换为字符串或字符。

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。 WebSphere MQ Java 类使用 JMS_IBM_CHARACTER_SET 在 MQRFH2 或消息描述符 MQMD 中的 JMS 属性中设置 CodedCharacterSetId 字段:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);[8]
```

如果将字节数组写入 JMSStringMessage 或 JMSMapMessage 中，那么 WebSphere MQ classes for JMS 不会执行数据转换，因为字节类型为十六进制数据，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 708 页的图 144 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

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

平台。 Windows 上的 JVM 通常具有缺省 Charset windows-1252 和 UNIX UTF-8。 Windows 与 UNIX 之间的交换要求您选择显式代码页以交换文本 (以字节为单位)。

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
           .getIntProperty(WMQConstants.WMQ_CCSID))));
```

图 *144:* 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 *145:* 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 *146:* 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 *147:* 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 *148:* 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 *149:* 在 *JMSBytesMessage* 中使用目标编码发送数字

在 JMSBytesMessage 中发送和接收文本

第 709 页的图 150 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。 要接收包含混合类型的文本字符串 (以字节计) 消息，必须知道该字符串的长度 (以字节计)，在 第 709 页的图 151 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 *150:* 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 *151:* 从 *JMSBytesMessage* 接收 *String*

相关概念

JMS 消息转换方法
许多数据转换方法都向 JMS 应用程序设计者开放。 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。 如果应用程序仅交换文本或仅与其他 JMS 应用程序交换消息，那么通常不考虑数据转换。 数据转换由 WebSphere MQ 自动执行。

队列管理器数据转换
队列管理器数据转换始终可用于接收来自 JMS 客户机的消息的非 JMS 应用程序。 从 V7.0 开始，接收消息的 JMS 客户机也使用队列管理器数据转换。 从 7.0.1.5 或具有 APAR IC72897 的 7.0.1.4 开始，队列管理器数据转换功能是可选功能。

相关任务

与非 JMS 应用程序交换格式化记录
请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使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 与非 JMS 应用程序交换格式化消息时可以调用数据转换出口，也可以不调用数据转换出口。

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。 针对 JMS 消息类型 JMSObjectMessage， JMSTextMessage，JMSMapMessage， JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型的交互。

队列管理器数据转换

队列管理器数据转换始终可用于接收来自 JMS 客户机的消息的非 JMS 应用程序。 从 V7.0 开始，接收消息的 JMS 客户机也使用队列管理器数据转换。 从 7.0.1.5 或具有 APAR IC72897 的 7.0.1.4 开始，队列管理器数据转换功能是可选功能。

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，通过设置 GetMessage **选项** GMO_CONVERT 来始终提供转换功能。 在 V7.0 之前，队列管理器转换功能对于接收消息的 JMS 应用程序不可用。

可以将 V7.0 之前的队列管理器转换与发送消息的 JMS 客户机应用程序配合使用。 JMS 客户机构建格式化记录，设置与消息中放置的数据对应的 CodedCharacterSetId， Encoding 和 Format 属性。 非 JMS 接收应用程序使用 GMO_CONVERT 读取消息，并导致调用用户编写的数据转换出口。 数据转换出口是一个共享库，在 Format 字段中设置其名称。

从 V7.0 开始，队列管理器能够转换发送到 JMS 客户机的消息。 从 7.0.0.0 到 7.0.1.4 (含)，始终针对 JMS 客户机调用队列管理器转换。 从 7.0.1.5 或应用 APAR IC72897 的 7.0.1.4 开始，可通过将目标属性 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功能。 WMQ_RECEIVE_CONVERSION_CLIENT_MSG 是缺省设置，与不支持 JMS 客户机的队列管理器数据转换的 WebSphere MQ V6.0 的行为相匹配。 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 *152:* 启用队列管理器数据转换

当客户机调用 consumer.receive 方法时，将进行 JMS 客户机的队列管理器数据转换。 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将对从 UTF-8 接收到的数据中的文本进行解码，并以内部 Unicode 编码创建 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 *153:* 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。 某些 JMS 客户机平台在以 UTF-8 交付消息文本时迁到问题。

JMS 代码等同于 第 711 页的图 154 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 *154: amqsget0.c* 的代码片段

注:

仅对具有已知 WebSphere 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
许多数据转换方法都向 JMS 应用程序设计者开放。 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。 如果应用程序仅交换文本或仅与其他 JMS 应用程序交换消息，那么通常不考虑数据转换。 数据转换由 WebSphere MQ 自动执行。

JMS 客户机消息转换和编码
列出了用于执行 JMS 客户机消息转换和编码的方法，以及每种类型的转换的代码示例。

第 347 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
请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使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 与非 JMS 应用程序交换格式化消息时可以调用数据转换出口，也可以不调用数据转换出口。

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。 针对 JMS 消息类型 JMSObjectMessage， JMSTextMessage，JMSMapMessage， JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型的交互。

与非 *JMS* 应用程序交换格式化记录
请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使用 JMSBytesMessage 与非 JMS 应用程序交换消息的 JMS 客户机应用程序。 与非 JMS 应用程序交换格式化消息时可以调用数据转换出口，也可以不调用数据转换出口。

开始之前

您可能能够设计一个更简单的解决方案，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

如果 JMS 客户机未包含在与其他 JMS 客户机交换的 JMS 消息的格式设置详细信息中，那么 JMS 客户机更易于编写。 只要消息类型为 `JMSTextMessage`， `JMSMapMessage`， `JMSStreamMessage` 或 `JMSObjectMessage`， WebSphere MQ 就会查找有关格式化消息的详细信息。 WebSphere MQ 处理不同平台上代码页和数字编码的差异。

可以使用这些消息类型与非 JMS 应用程序交换消息。 要执行此操作，您必须了解 WebSphere MQ JMS 类如何构造这些消息。 您可能能够修改非 JMS 应用程序以解释消息; 请参阅 第 678 页的『将 JMS 消息映射到 WebSphere MQ 消息』。

使用其中一种消息类型的优点是 JMS 客户机编程不依赖于它与之交换消息的应用程序的类型。 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。

另一种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 715 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』中的示例 JMS 客户机作为构建可与非 JMS 应用程序交换格式化记录的 JMS 客户机的起点。

过程

1. 定义记录布局，或者使用其中一个预定义的 WebSphere MQ 头类。

   要处理预定义的 WebSphere MQ 头，请参阅 处理 WebSphere MQ 消息头。

   第 713 页的图 155 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 715 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 类以封装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 715 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 第 696 页的『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 类型的消息，而不使用 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 715 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` 样本程序对于测试接收 JMS 客户机发送的消息很有用。 请参阅建议的修改以使用第 714 页的图 156 中的示例头 `RECORD.h`。 通过这些修改， `amqsgetc0.c` 将接收由示例 JMS 客户机 `TryMyRecord.java` 发送的消息; 请参阅 第 715 页的『编写用于在 JMSBytesMessage 中封装记录布局的类』。
   - 样本 WebSphere MQ 浏览程序 `amqsbcg0.c` 对于检查消息头， JMS 头 `MQRFH2` 和消息内容的内容很有用。
   - 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 *155: 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 *156:* 修改 *amqsget0.c*

相关概念

JMS 消息转换方法

许多数据转换方法都向 JMS 应用程序设计者开放。 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。 如果应用程序仅交换文本或仅与其他 JMS 应用程序交换消息，那么通常不考虑数据转换。 数据转换由 WebSphere MQ 自动执行。

JMS 客户机消息转换和编码

列出了用于执行 JMS 客户机消息转换和编码的方法，以及每种类型的转换的代码示例。

队列管理器数据转换

队列管理器数据转换始终可用于接收来自 JMS 客户机的消息的非 JMS 应用程序。 从 V7.0 开始，接收消息的 JMS 客户机也使用队列管理器数据转换。 从 7.0.1.5 或具有 APAR IC72897 的 7.0.1.4 开始，队列管理器数据转换功能是可选功能。

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。 针对 JMS 消息类型 JMSObjectMessage， JMSTextMessage，JMSMapMessage， JMSStreamMessage 和 JMSBytesMessage 描述了消息转换和消息类型的交互。

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。 在该任务中，您将创建一些 Java 类以在 JMSBytesMessage 中交换示例记录结构。 您可以修改此示例以编写可交换其他记录结构的类。

JMSBytesMessage 是用于与非 JMS 程序交换混合数据类型记录的最佳 JMS 消息类型选择。 它没有由 JMS 提供程序插入到消息体中的其他数据。 因此，如果 JMS 客户机程序与现有 IBM WebSphere MQ 程序进行互操作，那么这是要使用的消息类型的最佳选择。 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。 对于特定记录类型，封装读取和写入 JMSBytesMessage 的类使在 JMS 程序中发送和接收固定格式记录更容易。 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. 您的技能级别: 您必须熟悉 Java 编程和 JMS。 未提供有关设置 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. 此示例需要 IBM WebSphere MQ V7.0。
3. 此示例是使用 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 WebSphere MQ Explorer 中的 Java 透视图来开发和运行 Java 类。 或者，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 WebSphere 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 WebSphere MQ 头布局上建模。 将忽略很多 IBM WebSphere 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 721 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
一个抽象类，用于将 JMS 连接，目标和会话封装在单个类中。 其接口只满足测试 RECORD 和 MyRecord 类的需要。 它不是用于编写 JMS 应用程序的既定设计模式。

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 WebSphere 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 WebSphere 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， IBM WebSphere 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 WebSphere 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。 “setStructID(new String(structID, getMessageCharset()))” 行设置眼睛捕捉器。

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 封装在单个类中。 稍后使用的 MyConsumer 将 JMS Connection， Destination， Session 和 MessageConsumer 封装在单个类中。

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());” 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，因为编写 JMSBytesMessage 时 IBM WebSphere MQ JMS 类不会使用此值。

“messageCharset = myProducer.getCharset();” 调用

```
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());
    }
```

它从编码字符集标识获取 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，从而将其强制转换为 IBM WebSphere MQ MQI 客户机。 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();
```

如果必须覆盖 JMS 设置，那么此代码具有将 IBM WebSphere MQ 主体样式设置为未指定的副作用。

注:

IBM WebSphere MQ JMS 类将消息的格式，编码和字符集标识写入消息描述符 MQMD 或 JMS 头 MQRFH2。 这取决于消息是否具有 IBM WebSphere 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 WebSphere MQ JMS 类如何构造 JMS 消息。 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; }Â .
```

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 711 页的『与非 JMS 应用程序交换格式化记录』中的第 714 页的图 156。
  - 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-±=¼¶§>
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 WebSphere MQ 并运行主程序 TryMyRecord; 请参阅 第 721 页的图 157。

1. 配置 IBM WebSphere MQ 和 JMS 以运行此示例。 指示信息用于在 Windows 上运行示例。

   1. 创建队列管理器

```
crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

   2. 创建队列

```
echo DEFINE QL('Q1') REPLACE | runmqsc QM1
```

   3. 创建 JNDI 目录

```
cd c:\
md JNDI-Directory
```

   4. 切换到 JMS bin 目录
   必须从此处运行 JMS 管理程序。 路径为 *MQ_INSTALLATION_PATH*\java\bin。

5. 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
```

6. 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 WebSphere 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 721 页的图 157 到 第 725 页的图 162 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 *157: 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 *158: 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; }Â .


    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 *159: 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 *160: 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 *161: 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 *162: MyConsumer*

在 WebSphere MQ JMS 应用程序类中创建和配置连接工厂和目标

针对 JMS 应用程序的 WebSphere MQ 类可以通过从 Java 命名和目录接口 (JNDI) 名称空间中检索作为受管对象的连接工厂和目标，通过使用 IBM JMS 扩展或通过使用 WebSphere 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 WebSphere 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 WebSphere MQ JMS 管理工具或 WebSphere MQ Explorer 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。 检索到受管对象后，如果需要，应用程序可以使用 IBM JMS 扩展或 WebSphere MQ JMS 扩展来设置或更改其一个或多个属性。

使用 IBM JMS 扩展
应用程序可以使用 IBM JMS 扩展在运行时动态地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 WebSphere MQ JMS 扩展
应用程序还可以使用 WebSphere 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。

使用 *JNDI* 来检索 *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 () 方法来检索对象。

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 WebSphere MQ JMS 管理工具或 WebSphere MQ Explorer 在 JNDI 名称空间中创建和维护受管对象。 有关如何使用 WebSphere MQ JMS 管理工具的信息，请参阅 第 781 页的『使用 WebSphere MQ JMS 管理工具』。 有关如何使用 WebSphere MQ Explorer 的信息，请参阅 WebSphere MQ Explorer 随附的帮助。 但是，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

使用 *IBM JMS* 扩展

WebSphere MQ classes for JMS 包含一组称为 IBM JMS 扩展的 JMS API 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，并为 JMS 对象设置 WebSphere MQ 类的属性。 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

此包可以在位于 <MQ_Install_Dir>/java/lib 的 com.ibm.mqjms.jar 中找到。

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，而不是将它们作为受管对象从 Java 命名和目录接口 (JNDI) 名称空间中检索
- 一组用于设置 WebSphere MQ JMS 对象类的属性的方法
- 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
- 一组方法，用于获取有关 WebSphere MQ classes for JMS 的版本信息

关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 扩展提供了一组与 WebSphere MQ JMS 扩展的备用接口。 但是，虽然 WebSphere MQ JMS 扩展特定于 WebSphere MQ 消息传递提供程序，但 IBM JMS 扩展并非特定于 WebSphere MQ ，并且可以与 第 668 页的『一种分层架构』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 JMS 对象的 WebSphere MQ 类的属性时使用这些定义。 该接口包含 WebSphere MQ 消息传递提供程序的常量以及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 WebSphere MQ 消息传递提供程序标识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 728 页的『为 JMS 对象设置 WebSphere MQ 类的属性』。

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 WebSphere MQ 队列，该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 WebSphere MQ 队列的名称以及 (可选) 拥有该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 对象表示队列管理器 QM2 拥有的名为 Q2 的 WebSphere MQ 队列，发送到此目标的所有消息都是持久消息，优先级为 5。 有关队列 URI 的更多信息，请参阅 第 738 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 728 页的『为 JMS 对象设置 WebSphere MQ 类的属性』。

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 738 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 728 页的『为 JMS 对象设置 WebSphere MQ 类的属性』。

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。

为 JMS 对象设置 WebSphere MQ 类的属性

要使用 IBM JMS 扩展为 JMS 对象设置 WebSphere MQ 类的属性，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， JmsProperty 上下文接口包含用于设置具有该数据类型的属性值的方法，以及用于获取具有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。

当应用程序为 JMS 对象创建 WebSphere MQ 类时，将自动设置具有缺省值的任何属性。 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无效的值，那么 WebSphere MQ classes for JMS 将抛出 JMSException 异常。 如果应用程序尝试获取尚未设置的属性，那么行为如 JMS 规范中所述。 WebSphere MQ JMS 类针对基本数据类型抛出 NumberFormat 异常异常，并针对引用的数据类型返回 null。

除了 WebSphere MQ JMS 对象类的预定义属性外，应用程序还可以设置自己的属性。 WebSphere MQ JMS 类将忽略这些应用程序定义的属性。

有关用于 JMS 对象的 WebSphere MQ 类的属性的更多信息，请参阅 IBM WebSphere MQ classes for JMS 对象的属性。

以下代码是如何使用 IBM JMS 扩展来设置属性的示例。 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: 在 z/OS 平台上运行的队列管理器不支持设置应用程序名称，因此将忽略此设置。

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 JmsProperty 上下文接口的方法来设置或获取 WebSphere MQ classes for JMS 对象的属性时，可以将该属性的值从一种数据类型隐式转换为另一种数据类型。

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）。

第 729 页的表 122 中显示了 WebSphere MQ classes for JMS 支持的转换。

表 *122:* 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
字节	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 (`char` 除外) 的值，前提是该字符串具有正确的转换格式。 如果应用程序尝试转换格式不正确的字符串，那么 WebSphere MQ classes for JMS 会抛出 NumberFormat 异常。
- 如果应用程序尝试进行不受支持的转换，那么 WebSphere MQ classes for JMS 将抛出 MessageFormat 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，那么 WebSphere MQ classes for JMS 会抛出 NumberFormat 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:

  [空白] [*sign*]数字[*e_char*[*e_sign*]*e_number*]

  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，那么 WebSphere MQ classes for JMS 会抛出 NumberFormat 异常。
- 将数字值 (包括数据类型为 `byte` 的值) 转换为字符串时，该值将转换为该值作为十进制数字的字符串表示，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 728 页的『为 JMS 对象设置 WebSphere MQ 类的属性』 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 JmsProperty 上下文接口的方法来设置和获取 WebSphere MQ JMS 对象类的属性，那么应用程序可以通过以下任何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 WebSphere 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， WebSphere MQ JMS 类将属性名称和值存储为 com.ibm.msg.client.wmq.WMQConstants 接口中定义的字面值。 这是已定义的用于属性名称和值的规范格式。 作为一般规则，如果应用程序使用另两种指定属性名称和值的方法之一来设置属性，那么 WebSphere MQ classes for JMS 必须将名称和值从指定的输

入格式转换为规范格式。 同样，如果应用程序使用另两种指定属性名称和值的方法之一来获取属性，那么 WebSphere MQ classes for JMS 必须将名称从指定的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。

异常，跟踪文件或 WebSphere MQ classes for JMS 日志中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 接口的方法来访问 WebSphere 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 *WebSphere MQ JMS* 扩展

WebSphere MQ JMS 类包含一组称为 WebSphere MQ JMS 扩展的 JMS API 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。

WebSphere MQ classes for JMS 包含包 com.ibm.jms 和 com.ibm.mq.jms 中的一组类。 这些类实现 JMS 接口并包含 WebSphere 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.*;
```

应用程序可以使用 WebSphere MQ JMS 扩展来执行以下功能:

- 在运行时动态创建连接工厂和目标，而不是将它们作为受管对象从 Java 命名和目录接口 (JNDI) 名称空间中检索
- 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 733 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。

要与代理程序进行实时连接，应用程序可以使用以下代码：

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

此代码假设代理程序在主机名为 HOST2 的系统上运行，在侦听端口号 1507。

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 WebSphere MQ classes for JMS 对象的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 WebSphere 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 WebSphere MQ 队列，该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");
```

此语句创建的 MQQueue 对象表示队列管理器 QM2 拥有的名为 Q2 的 WebSphere MQ 队列。 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 WebSphere MQ ，以便当应用程序将消息发送到此目标时， Websphere MQ 可以将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 WebSphere MQ 队列的名称以及 (可选) 拥有该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 WebSphere MQ 队列，该队列由队列管理器 QM3 拥有，并且发送到此目标的所有消息都是持久消息，优先级为 5。 有关队列 URI 的更多信息，请参阅 第 738 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 734 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 738 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 734 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 WebSphere 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 应用程序使用 ConnectionFactory 对象来创建 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 WebSphere 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 规范声明以 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 WebSphere 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 应用程序可以通过首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 会话的应答方式

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_应答和 DUPS_OK_应答方式下， 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) 来标识 WebSphere MQ 队列或主题，并 (可选) 指定 Queue 或 Topic 对象的一个或多个属性。

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 WebSphere MQ 队列，该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 WebSphere MQ 队列的名称以及 (可选) 拥有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 WebSphere MQ 队列，该队列由名为 QM2 的队列管理器拥有，并且发送到此目标的所有消息都是持久消息，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 WebSphere MQ ，以便当应用程序将消息发送到此目标时， Websphere MQ 可以将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 738 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。

当应用程序创建临时队列时， WebSphere MQ JMS 类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 738 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， WebSphere MQ classes for JMS 将创建一个名称以字符 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 WebSphere MQ 队列的名称以及 (可选) 拥有该队列的队列管理器的名称以及应用程序创建的 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 WebSphere MQ ，以便当应用程序向队列发送消息时， Websphere MQ 可以将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
WebSphere 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 用于命名 IBM WebSphere MQ 对象的规则 。

topicName
主题的名称。

有关创建主题名称的规则，请参阅 IBM WebSphere 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 738 页的表 123 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 738 页的表 123 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 738 页的表 123 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 WebSphere MQ JMS 管理工具对属性值使用符号常量，但 URI 不能包含符号常量。

表 *123:* 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	当 WebSphere 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	• WebSphere MQ 支持的任何编码字符集标识。

表 *123:* 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
编码	当 WebSphere 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	• WebSphere 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。 与每个值相关联的是 WebSphere MQ JMS 管理工具中使用的多点广播属性的相应值。 有关 MULTICAST 属性及其有效值的描述，请参阅 IBM WebSphere 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 WebSphere MQ 队列或主题的 *DefPersistence* 属性指定。 • 1 - 非持久性。 • 2 - 持久性。 • 3-相当于 WebSphere MQ JMS 管理工具中使用的 PERSISTENCE 属性的值 HIGH。 有关此值的说明，请参阅第 758 页的『JMS 持久消息』。
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 WebSphere 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 WebSphere 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 标识名为 Q2 的 WebSphere MQ 队列，该队列由名为 QM2 的队列管理器拥有。 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，然后才能将消息发送到目标。 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 726 页的『使用 JNDI 来检索 JMS 应用程序中的受管对象』
- 第 726 页的『使用 IBM JMS 扩展』
- 第 732 页的『使用 WebSphere MQ JMS 扩展』
- 第 736 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， 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，即 Message 接口的子接口，以及 Session 接口中用于创建具有该类型主体的消息的方法。 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 676 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。

可以配置目标，以便当应用程序向其发送消息时， WebSphere MQ classes for JMS 会转发消息并将控制权交还给应用程序，而无需确定队列管理器是否已安全地接收消息。 这一共能有时称为异步输入。 有关更多信息，请参阅第 770 页的『将消息异步放入 IBM WebSphere MQ JMS 类中』。

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 726 页的『使用 JNDI 来检索 JMS 应用程序中的受管对象』
- 第 726 页的『使用 IBM JMS 扩展』
- 第 732 页的『使用 WebSphere MQ JMS 扩展』
- 第 736 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 标识以下两部分中描述的两种类型的消息使用者，即非持久主题订户和持久主题订户。

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 WebSphere MQ JMS 类中的扩展，非持久主题订户还会接收保留发布，但在使用与代理的实时连接时不接收。

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 WebSphere MQ JMS 类中的扩展，持久主题订户还会接收保留发布。

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 676 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，当应用程序连接到版本 7 队列管理器时，连接工厂的 MSGSELECTION 属性没有效果。 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)
- 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 745 页的『在 IBM WebSphere MQ classes for JMS 中处理有害消息』。

V 7.5.0.8 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 WebSphere 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 Version 7.5.0, Fix Pack 8 开始，当从包含带有 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 WebSphere 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 WebSphere MQ classes for JMS 应用程序可以调用：

```
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 679 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

关闭用于 JMS 应用程序的 WebSphere MQ 类

WebSphere MQ JMS 应用程序类在停止之前显式关闭某些 JMS 对象很重要。 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。

仅靠垃圾回收无法及时释放所有 WebSphere MQ JMS 类和 WebSphere MQ 资源，尤其是在应用程序在会话级别或更低级别创建许多短暂的 JMS 对象时。 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。

请勿在应用程序中使用终结器来关闭 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。

在 IBM WebSphere MQ classes for JMS 中处理有害消息

有害消息是接收 MDB 应用程序无法处理的消息。 如果迁到有害消息，那么 JMS MessageConsumer 和 ConnectionConsumer 对象可以根据两个队列属性 BOQNAME 和 BOTHRESH 对其进行重新排队。

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。 这些消息称为中毒消息。 JMS MessageConsumer 对象检测有害消息并将其重新路由到备用目标。

IBM WebSphere MQ 队列管理器将保留每个消息的回退次数记录。 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。 请参阅第 777 页的『在 ASF 中从队列移除消息』以获取更多详细信息。

MessageConsumer 和 ConnectionConsumer 对中毒消息重新排队的方式有所区别。 ConnectionConsumer 可以在不影响消息发送的情况下对中毒消息进行重新排队。 重新排队过程的发生位于与发往应用程序代码的实际消息关联的任何工作单元外。 因为 ConnectionConsumer 操作的多线程性质，这是有可能的。

但是，MessageConsumer 是会话级别以下的单线程，因此任何中毒消息的重新排列都发生在当前工作单元内。 这不会影响应用程序的操作，但是，当中毒消息在事务性或客户机应答会话下重新排队时，应用程序代码（或者应用程序容器，如果适用）提交当前工作单元之前，不会提交重新排队操作本身。

JMS 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 WebSphere 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 WebSphere MQ 消息传递提供程序正常模式中的发布/预订消息传递，该主题的受管队列从模型队列中创建。
- 对于 IBM WebSphere 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。

IBM WebSphere MQ classes for JMS 将查询队列的 BackoutThreshold 和 BackoutRequeueQName。 因此，您必须为应用程序上运行的用户授予对队列的问询访问权。

V 7.5.0.9 如果目标队列是集群队列，那么所需的权限取决于所使用的 IBM WebSphere MQ classes for JMS 版本：

- 使用 IBM WebSphere MQ classes for JMS for Version 7.5.0, Fix Pack 9 以及 APAR IT26482 的临时修订时，需要查询访问权。
- 对于所有其他版本，请授予查询，浏览和访问权。

要设置 BackoutThreshold 和 BackoutRequeueQName 属性，请发出以下 MQSC 命令：

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

如果 BackoutThreshold 属性设置为零之外的值，为了避免意外行为，请将 BackoutRequeueQName 属性设置为有效的队列名称。

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 WebSphere 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) BOQNAME(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，但消息无法转至回退队列，该消息将发送到死信队列或被废弃。 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。 请参阅第 777 页的『在 ASF 中从队列移除消息』以获取进一步详细信息。

在将消息重新排队到回退重新排队队列时，消息的消息描述符 (MQMD) 中的一些字段值会发生更改。 请参阅 MQMD - 消息描述符，以获取有关 MQMD 格式的详细信息。

在消息进入回退队列时，以下 MQMD 字段的值会发生更改。

- PutDate 将更新为进入回退重新排队队列的日期。
- PutTime 将更新为进入回退重新排队队列的时间。
- 回退计数将重置为零。
- 将更新消息的到期时间以反映 JMS 应用程序接收原始消息时的剩余到期时间。

在消息进入回退队列时，以下字段中的值将保持不变：

- StructId
- 版本
- 报告
- MessageType
- Feedback
- 编码
- CodedCharSetId
- MsgId

- CorrelId
- ReplyToQ
- ReplyToQMgr
- 格式
- 持久
- 优先级

IBM WebSphere MQ classes for JMS 中的异常

IBM WebSphere MQ classes for JMS 应用程序必须能够处理由 JMS API 调用抛出或交付到异常处理程序的异常。

IBM WebSphere MQ classes for JMS 通过抛出异常来报告运行时问题。 JMSException 是 JMS 方法抛出的异常的根类，捕获 JMSException 异常提供了处理所有 JMS 相关异常的通用方法。

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由较低级别的问题导致的，该问题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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ 消息传递提供程序抛出。
- 下一个异常 (com.ibm.mq.jmqi.JmqiException 的实例) 由 MQI 的公共 Java 接口抛出。
- 最终异常 (java.net.ConnectionException 的实例) 由 Java 类库抛出。

有关 IBM WebSphere MQ classes for JMS 的分层体系结构的更多信息，请参阅 第 668 页的『用于 JMS 体系结构的 IBM WebSphere MQ 类』。

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 WebSphere 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 WebSphere 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， JmqiException 异常会将信息封装在 AMQ*nnnn* 或 CSQ*nnnn* 消息中 (如果与异常相关联)。 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 747 页的『链接异常』中的样本代码。

从 IBM WebSphere MQ classes for JMS 的先前版本升级

与先前版本的 IBM WebSphere MQ classes for JMS 相比，大多数错误代码和异常消息在 V 7 中已更改。 这些更改的原因是 IBM WebSphere MQ classes for JMS 现在具有分层体系结构，并且从代码中的不同层抛出异常。

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 WebSphere 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， V 7 of IBM WebSphere MQ classes for JMS 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: WebSphere 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，并且您正在从版本 7 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版本 7 of IBM WebSphere MQ classes for JMS。

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 WebSphere 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。

V 7.5.0.8 IBM WebSphere MQ Version 7.5.0 修订包 8 中包含的 APAR IT14820 修复了以下缺陷: 即使应用程序所使用的 JMS 连接工厂上的 ASYNC_EXCEPTION 属性设置为 ASYNC_EXCEPTIONS_ALL ， 也不会对非连接中断异常 (例如 MQRC_GET_INITED) 调用应用程序的 JMS ExceptionListener 。 这是 Version 7.5.0, Fix Pack 8 之前的缺省值。

V 7.5.0.8 为维护配置 JMS MessageListener 和 JMS ExceptionListener 的当前 JMS 应用程序的行为，并确保 IBM WebSphere MQ classes for JMS 与 JMS 规范一致，已将 IBM WebSphere MQ classes for JMS 的 ASYNC_EXCEPTIONS JMS ConnectionFactory 属性的缺省值更改为 ASYNC_EXCEPTIONS_CONNECTIONBROKEN。 因此，缺省情况下，只会将与中断连接错误代码对应的异常传递至应用程序的 JMS ExceptionListener。

V 7.5.0.8 从 Version 7.5.0, Fix Pack 8 开始，还更新了 IBM WebSphere MQ classes for JMS ，以便当应用程序使用的 JMS ConnectionFactory 将 ASYNC_EXCEPTIONS 属性设置为值 ASYNC_EXCEPTIONS_ALL 时，仍会将与非连接中断错误相关的 JMSExceptions (在消息传递到异步消息使用者期间发生) 传递到已注册的 ExceptionListener 。

V 7.5.0.8 有关 Version 7.5.0, Fix Pack 8 异常侦听器的更改内容以及从较早发行版进行更改的原因的更多信息，请参阅 JMS: 版本 7.5 中的异常侦听器更改。

对于任何其他类型的问题，当前 JMS API 调用将抛出 JMSException 异常。

如果应用程序未向 Connection 对象注册异常侦听器，那么会将已传递到异常侦听器的任何异常写入 IBM WebSphere MQ classes for JMS 日志。

相关参考

ASYNCEXCEPTION

WebSphere MQ JMS 类中的日志记录错误

有关可能需要用户执行更正操作的运行时问题的信息将写入 WebSphere MQ classes for JMS 日志。

例如，如果应用程序尝试设置连接工厂的属性，但无法识别该属性的名称，那么 WebSphere MQ classes for JMS 会将有关该问题的信息写入其日志。

缺省情况下，包含日志的文件名为 mqjms.log，该文件位于当前工作目录中。 但是，您可以通过在 WebSphere MQ classes for JMS 配置文件中设置 com.ibm.msg.client.commonservices.log.outputName 属性来更改日志文件的名称和位置。 有关 WebSphere MQ classes for JMS 配置文件的信息，请参阅 第 613 页的『IBM WebSphere MQ classes for JMS 配置文件』，有关 com.ibm.msg.client.commonservices.log.outputName 属性的有效值的更多详细信息，请参阅 第 667 页的『日志记录和 IBM WebSphere MQ classes for JMS』。

WebSphere MQ JMS 类中的首次故障支持技术 *(FFST)*

如果 WebSphere MQ JMS 类中发生严重内部错误，那么将生成首次故障支持技术 (FFST) 信息。

将 FFST 信息写入名为 JMSC*nnnn* 的文件。FDC ，其中 *nnnn* 是四位数字。 此文件位于名为 FFDC 的目录中，该目录是跟踪输出写入的目录的子目录。 缺省情况下，跟踪输出将写入当前工作目录，但您可以通过在 JMS 配置文件的 WebSphere MQ 类中设置 **com.ibm.msg.client.commonservices.trace.outputName** 属性，将跟踪输出重定向到另一个目录。 有关 WebSphere MQ classes for JMS 配置文件的信息，请参阅 第 613 页的『IBM WebSphere MQ classes for JMS 配置文件』。

如果在生成 FFST 信息时启用了跟踪，那么还会将 FFST 信息写入跟踪文件。 有关跟踪 JMS 程序的更多信息，请参阅 跟踪 IBM WebSphere MQ classes for JMS 应用程序。

要禁止生成 FFDC 文件，请设置属性 **com.ibm.msg.client.commonservices.ffst.suppress**，如下所示:

0
输出所有 FFDC 文件 (缺省值)。

-1
仅输出特定类型的第一个 FFDC 文件。

integer
禁止所有 FFDC 文件，但那些是此数字的倍数的文件除外。

从 WebSphere MQ JMS 应用程序类访问 WebSphere MQ 功能部件

WebSphere MQ classes for JMS 提供了一些工具来利用 WebSphere MQ 的许多功能。

注意: 这些功能部件在 JMS 规范之外，或者在某些情况下违反 JMS 规范。 如果使用它们，那么应用程序不太可能与其他 JMS 提供程序兼容。 那些不符合 JMS 规范的功能部件将使用 "注意" 声明进行标注。

从 *WebSphere MQ JMS* 应用程序类读取和写入消息描述符

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 WebSphere MQ 应用程序要求在发送给它们的消息的 MQMD 中设置特定值。 WebSphere MQ JMS 类提供消息属性，这些属性允许 JMS 应用程序设置 MQMD 字段，从而使 JMS 应用程序能够 "驱动" WebSphere 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 753 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 目标对象属性

"目标对象" 的两个属性控制对来自 JMS 的 MQMD 的访问，第三个属性控制消息上下文。

表 *124:* 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

表 *124:* 属性名称和描述 *(*继续*)*

属性	缩写	描述
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 *125:* 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 已启用	• **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 已启用	• **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 *125:* 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。 应用程序无法修改所收到消息的属性，除非首先根据 JMS 规范清除了所有属性。 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 *126:* 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	反馈
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	格式
JMS_IBM_MQMD_Priority [1]	消息优先级	整数	优先级
JMS_IBM_MQMD_Persistence	消息持久性	整数	持久性
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 126: 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:** 如果为 JMS_IBM_MQMD_Priority 指定不在 0-9 范围内的值，那么这将违反 JMS 规范。
2. **注意:** JMS 规范声明消息标识必须由 JMS 提供程序进行设置且必须唯一或为空值。 如果您为 JMS_IBM_MQMD_MsgId 指定值，此值将复制到 JMSMessageID。 因此，它不是由 JMS 提供程序设置的，并且可能不是唯一的: 这违反了 JMS 规范。
3. **注意:** 如果为 JMS_IBM_MQMD_CorrelId 指定以字符串 "ID:" 开头的值，那么这违反了 JMS 规范。
4. **注意:** 在消息上使用字节数组属性违反了 JMS 规范。

使用 *WebSphere MQ classes for JMS* 从应用程序访问 *IBM WebSphere MQ* 消息数据

您可以使用 IBM WebSphere MQ classes for JMS 在应用程序中访问完整的 WebSphere 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 WebSphere 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，那么将返回不带 JMS MQRFH2 头的消息体，并且 JMSBytesMessage 的属性将反映 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。 如果应用程序已连接到 WebSphere MQ V6 队列管理器，或者如果已将 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 WebSphere MQ JMS 类会根据 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，那么 WebSphere MQ classes for JMS 将发送 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，那么可以在构造 WebSphere MQ 消息时覆盖 JMS 消息体的缺省设置。 请使用以下属性:

- JMS_IBM_Format 或 JMS_IBM_MQMD_Format: 此属性指定用于启动 JMS 消息体的 WebSphere MQ 头或应用程序有效内容 (如果没有前面的 Websphere MQ 头) 的格式。
- JMS_IBM_Character_Set 或 JMS_IBM_MQMD_CodedCharSetId: 此属性指定用于启动 JMS 消息体的 WebSphere MQ 头或应用程序有效内容的 CCSID (如果没有前置 Websphere MQ 头)。

- JMS_IBM_Encoding 或 JMS_IBM_MQMD_Encoding: 此属性指定 WebSphere MQ 头或应用程序有效内容的编码，如果没有前面的 Websphere MQ 头，那么将启动 JMS 消息体。

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 WebSphere 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，将在 JMS RFH2 头中设置这些属性。
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 756 页的图 163 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 "ëÈÊ'>...??>?"
```

   图 *163:* 编码不一致的 *MQMD* 和消息数据

第 756 页的图 164 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 *164:* 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 消息类型和主体由接收到的 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 WMQ_MESSAGE_BODY_UNSPECIFIED，那么缺省值 IBM WebSphere 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: 此属性指定用于启动 JMS 消息体的 WebSphere MQ 头或应用程序有效内容 (如果没有前面的 Websphere MQ 头) 的格式。
- JMS_IBM_Character_Set 或 JMS_IBM_MQMD_CodedCharSetId: 此属性指定用于启动 JMS 消息体的 WebSphere MQ 头或应用程序有效内容的 CCSID (如果没有前置 Websphere MQ 头)。
- JMS_IBM_Encoding 或 JMS_IBM_MQMD_Encoding: 此属性指定 WebSphere MQ 头或应用程序有效内容的编码，如果没有前面的 Websphere MQ 头，那么将启动 JMS 消息体。

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 确定 JMS 应用程序是否将 WebSphere MQ 消息的 MQRFH2 作为消息有效内容的一部分 (即，作为 JMS 消息主体的一部分) 进行处理。

表 *127:* 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 WebSphere MQ 消息的 MQRFH2 作为消息有效内容的一部分 (即，作为 JMS 消息体的一部分) 进行处理。

表 *128:* 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， WebSphere MQ classes for JMS 会生成或不生成并包含 MQRFH2 头，具体取决于 WMQ_TARGET_CLIENT 的值。 接收时，充当值 JMS。 • JMS 发送时， WebSphere MQ classes for JMS 会自动生成 MQRFH2 头，并将其包含在 WebSphere MQ 消息中。 接收时， WebSphere MQ JMS 类会根据 MQRFH2 (如果存在) 中的值来设置 JMS 消息属性; 它不会将 MQRFH2 作为 JMS 消息体的一部分提供。 • MQ 发送时， WebSphere MQ JMS 类不会生成 MQRFH2。 接收时， WebSphere MQ classes for JMS 将 MQRFH2 作为 JMS 消息体的一部分提供。	• **WMQ_MESSAGE_BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS 持久消息

针对 JMS 应用程序的 WebSphere MQ 类可以使用 **NonPersistentMessageClass** 队列属性来为 JMS 持久消息提供更好的性能，从而牺牲某些可靠性。

WebSphere 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。

您可以使用带有以下任一参数的 WebSphere MQ Script (MQSC) 命令 DEFINE QLOCAL 来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。

本主题描述了针对 JMS 应用程序的 WebSphere MQ 类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 WebSphere MQ JMS 管理工具来设置此值，或者应用程序可以调用传递值 WMQConstants.WMQ_PER_NPHIGH 作为参数的 Destination.setPersistence() 方法。

如果应用程序将 JMS 持久消息或 JMS 非持久消息发送到 PERSISTENCE 属性值为 HIGH 的目标，并且底层 WebSphere MQ 队列设置为 NPMCLASS (HIGH) ，那么该消息将作为 WebSphere MQ 非持久消息放在队列上。 如果目标的 PERSISTENCE 属性没有值 HIGH ，或者如果底层队列设置为 NPMCLASS (NORMAL) ，那

么会将 JMS 持久消息作为 WebSphere MQ 持久消息放在队列上，并将 JMS 非持久消息作为 WebSphere MQ 非持久消息放在队列上。

如果将 JMS 持久消息作为 WebSphere MQ 非持久消息放在队列上，并且您希望确保在队列管理器停顿或立即关闭后不会废弃该消息，那么必须将所有可能路由该消息的队列设置为 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, WebSphere 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 WebSphere 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作为 JMS 持久消息接收，作为 JMS 非持久消息发送的消息作为 JMS 非持久消息接收。

当应用程序将第一条消息发送到 PERSISTENCE 属性具有值 HIGH 的目标时，或者当应用程序为 PERSISTENCE 属性具有值 HIGH 的目标创建第一条消息使用者时， WebSphere MQ JMS 类发出 MQINQ 调用以确定是否在底层 WebSphere MQ 队列上设置了 NPMCLASS (HIGH)。 因此，应用程序必须有权问询此队列。 此外，针对 JMS 的 WebSphere MQ 类会保留 MQINQ 调用的结果，直到删除目标为止，并且不会发出更多 MQINQ 调用。 因此，如果在应用程序仍在使用目标时更改底层队列上的 NPMCLASS 设置，那么 WebSphere MQ JMS 类不会注意到新设置。

通过允许将 JMS 持久消息作为 WebSphere MQ 非持久消息放在 WebSphere MQ 队列上，您将以某种可靠性为代价获取性能。 如果您需要 JMS 持久消息的最大可靠性，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。 如果您希望使用临时队列来接受持久消息，那么您必须使用 SYSTEM.JMS.TEMPQ.MODEL，或将模型队列更改为您选择的替代队列。

将安全套接字层 *(SSL)* 与 *WebSphere MQ JMS* 类配合使用

WebSphere MQ JMS 应用程序类可以使用 SSL 加密。 为此，这些应用程序需要 JSSE 提供程序。

使用传输 (CLIENT) 支持安全套接字层 (SSL) 加密的 WebSphere MQ 类用于 JMS 连接。 SSL 提供了通信加密、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。

WebSphere MQ JMS 类使用 Java 安全套接字扩展 (JSSE) 来处理 SSL 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 WebSphere MQ classes for JMS 应用程序使用客户机通道定义表 (CCDT) 来连接到队列管理器，请参阅第 766 页的『将客户机通道定义表与 IBM WebSphere MQ classes for JMS 配合使用』。

SSLCIPHERSUITE 对象属性
设置 SSLCIPHERSUITE 以对 ConnectionFactory 对象启用 SSL 加密。

要对 ConnectionFactory 对象启用 SSL 加密，使用 JMSAdmin 将 SSLCIPHERSUITE 属性设为您的 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 762 页的『JMS 中的 SSL CipherSpecs 和 CipherSuites』包含一个表，该表将 WebSphere MQ 支持的 CipherSpecs 映射到 JSSE 已知的等效 CipherSuites 。 有关具有 WebSphere MQ 的 CipherSpecs 和 CipherSuites 的更多信息，请参阅 安全性。

例如，要设置可用于通过 CipherSpec 为 RC4_MD5_EXPORT 的已启用 SSL 的 MQI 通道创建连接的 ConnectionFactory 对象，请向 JMSAdmin 发出以下命令:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

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 766 页的『将客户机通道定义表与 IBM WebSphere MQ classes for JMS 配合使用』。

SSLFIPSREQUIRED 对象属性
如果需要连接以使用 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 ，这表示连接可以使用 WebSphere 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
适用于 UNIX， Linux 和 Windows 的联邦信息处理标准 (FIPS)

SSLPEERNAME 对象属性
使用 SSLPEERNAME 来指定专有名称模式，以确保 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有名称 (DN) 模式来确保它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 命令将 ConnectionFactory 对象设置为期望队列管理器使用以字符 QMGR. 开头的公共名称以及至少两个组织单元名称 (第一个必须是 IBM 和第二个 WEBSPHERE) 来标识自身:

```
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 或更高版本下指定用于 CRL 检查的 LDAP 服务器。 以下 JMSAdmin 示例指示 JMS 使用在名为 crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

注: 要成功将 CertStore 与在 LDAP 服务器上托管的 CRL 配合使用，请确保 Java 软件开发包 (SDK) 与 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 允许通过提供由空格字符定界的 LDAP 服务器列表来提供多个 LDAP 服务器。 例如:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

当指定了多个 LDAP 服务器时， JMS 会依次尝试每个服务器，直到找到可以成功验证队列管理器证书的服务器为止。 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。 字节数还包括 WebSphere MQ JMS 类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 SSL 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
要为应用程序定制 SSL 连接的其他方面，请创建 SSLSocketFactory 并配置 JMS 以使用该连接。

您可能要定制应用程序的 SSL 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。 获取合适的 SSLSocketFactory 对象后，请使用 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。 攻击者可以模仿 LDAP 服务器，误导 JMS 应用程序连接到错误的服务器而不引起注意。 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，那么此时运行的 JMS 应用程序的 WebSphere MQ 类不会自动获取更改。 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。

JMS 中的 *SSL CipherSpecs* 和 *CipherSuites*
WebSphere MQ 及其等效的 CipherSuites 支持 CipherSpecs 。

第 762 页的表 129 列出了 WebSphere MQ 及其等效 CipherSuites 支持的 CipherSpecs 。 如果 ConnectionFactory 属性 SSLFIPSREQUIRED 设置为 NO ，那么如果在 MQI 通道的服务器端指定了任何受支持的 CipherSpec ，并且在客户机端指定了等效的 CipherSuite ，那么针对 JMS 应用程序的 WebSphere MQ 类可以连接到队列管理器。 如果 SSLFIPSREQUIRED 设置为 YES ，那么 CipherSpec 和 CipherSuite 的组合将确定应用程序是否可以连接到队列管理器。

在 MQI 通道的服务器端，可以在 DEFINE CHANNEL CHLTYPE (SVRCONN) 命令中将 CipherSpec 的名称指定为 SSLCIPH 参数的值。 在 MQI 通道的客户机端，可以通过以下方式指定 CipherSuite 的名称：

- 应用程序可以调用 ConnectionFactory 对象的 setSSLCipherSuite () 方法。
- 通过使用 WebSphere MQ JMS 管理工具，可以设置 ConnectionFactory 对象的 SSLCIPHERSUITE 属性。

表 *129: WebSphere MQ* 及其等效 *CipherSuites* 支持的 *CipherSpecs*

CipherSpec	**对应的 CipherSuite**	**如果 SFIPS [1] 设置为 YES ，是否可以连接?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	否
NULL_SHA	SSL_RSA_WITH_NULL_SHA	否
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5	否
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	否
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA	否
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	否
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA	否
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	否
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	否
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA	否
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	否 [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	是 [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	是 [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	是 [5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[第 763 页的『8』] [9]	SSL_RSA_WITH_DES_CBC_SHA	否 [3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[8]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Yes

表 129: WebSphere MQ 及其等效 CipherSuites 支持的 CipherSpecs (继续)		
CipherSpec	**对应的 CipherSuite**	**如果 SFIPS [1] 设置为 YES，是否可以连接?**
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA	否 [4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	否 [6]

注意:

1. 使用 WebSphere MQ JMS 管理工具时， SFIPS 是 ConnectionFactory 属性 SSLFIPSREQUIRED 的短名称。
2. 此 CipherSpec 没有等效的 CipherSuite。
3. 此 CipherSpec 在 2007 年 5 月 19th 之前经过 FIPS 140-$tag3 认证。
4. 此 CipherSpec 在 2007 年 5 月 19th 之前经过 FIPS 140-$tag3 认证。 名称 FIPS_WITH_DES_CBC_SHA 是历史名称，反映了此 CipherSpec 先前 (但不再) 符合 FIPS 的事实。 不推荐使用此 CipherSpec。
5. 这些 CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA， TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) 无法用于保护从 WebSphere MQ Explorer 到队列管理器的连接，除非将相应的不受限制的策略文件应用于 Explorer 所使用的 JRE。

   有关策略文件的更多信息，请参阅 安全性信息 。
6. 名称 FIPS_WITH_3DES_EDE_CBC_SHA 是历史名称，反映了此 CipherSpec 先前 (但不再) 符合 FIPS 的事实。 不推荐使用此 CipherSpec。
7. 这些 CipherSpecs (TLS_RSA_WITH_NULL_SHA256， TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA， TLS_RSA_WITH_AES_256_CBC_SHA256) 需要 IBM JRE 6.0 SR13 FP2 ， 7.0 SR4 FP2 或更高版本。
8. 这些 CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA， TLS_RSA_WITH_DES_CBC_SHA , TLS_RSA_WITH_RC4_128_SHA256) 可以使用 SSLv3 或 TLS。 缺省情况下，未启用 FIPS 时，将使用 SSLv3 。 要使用 TLS ，请将 Java 系统属性 **com.ibm.mq.cfg.preferTLS** 设置为 `true`。
9. 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。

相关信息

指定运行时在 MQI 客户机上仅使用经过 FIPS 认证的 CipherSpecs

适用于 UNIX， Linux 和 Windows 的联邦信息处理标准 (FIPS)

使用 *Java for WebSphere MQ JMS* 类编写通道出口

您可以通过定义实现指定接口的 Java 类来创建通道出口。

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。 接收出口返回要由 WebSphere 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 SSL 加密。 类似地，从连接的服务器端接收数据时，将在调用任何通道出口之前执行 SSL 解密。

在 WebSphere MQ classes for JMS 低于 V 7.0 的版本中，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 WebSphere MQ classes for JMS* 以使用通道出口

IBM WebSphere 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 WebSphere MQ 资源适配器用于入站通信的激活规范上的 **sendexit** 属性。
- IBM WebSphere 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 WebSphere MQ 资源适配器用于入站通信的激活规范上的 **receiveexit** 属性。
- IBM WebSphere 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 WebSphere MQ 资源适配器用于入站通信的激活规范上的 **securityexit** 属性。
- IBM WebSphere 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 WebSphere MQ JMS 管理工具或 IBM WebSphere 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 WebSphere 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 WebSphere MQ 出口目录，如 第 765 页的表 130 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 *130: IBM WebSphere MQ* 出口目录

平台	**目录**
UNIX and Linux	`/var/mqm/exits`（32 位通道出口） `/var/mqm/exits64`（64 位通道出口）
Windows	*install_data_dir*\exits 其中，*install_data_dir* 是您在安装期间为 IBM WebSphere MQ 数据文件所选的目录。 缺省目录为 `C:\Program Files\IBM\WebSphere MQ`。

注: 如果在多个位置存在通道出口，那么 IBM WebSphere 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 WebSphere 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 WebSphere 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。

如果要在启用 Java Security Manager 的情况下运行应用程序，那么运行应用程序的 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许可权。 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 Version 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 330 页的『消息传递通道的通道出口程序』。 必须在第 765 页的表 130 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 766 页的『将客户机通道定义表与 IBM WebSphere MQ classes for JMS 配合使用』。

指定在使用 *WebSphere MQ classes for JMS* 时要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 WebSphere MQ JMS 管理工具或 WebSphere 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 WebSphere MQ JMS 管理工具或 WebSphere 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 766 页的『将客户机通道定义表与 IBM WebSphere MQ classes for JMS 配合使用』。

将客户机通道定义表与 *IBM WebSphere MQ classes for JMS* 配合使用

JMS 应用程序的 IBM WebSphere MQ 类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。 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 对象的某些属性来创建客户机连接通道定义的替代方法， IBM WebSphere MQ classes for JMS 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。 这些定义由 IBM WebSphere MQ 脚本命令 (MQSC) 或 IBM WebSphere MQ 可编程命令格式 (PCF) 命令创建。 当应用程序创建 Connection 对象时， IBM WebSphere MQ classes for JMS 会在客户机通道定义表中搜索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，必须将 ConnectionFactory 对象的 CCDTURL 属性设置为 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。 您可以使用 IBM WebSphere MQ JMS 管理工具来设置 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() 方法来设置该属性。

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
- 星号 (*)
- 空字符串或包含的全部是空白字符的字符串

这些值可以用于使用消息队列接口 (MQI) 的客户机应用程序发出的 MQCONN 调用上的 *QMgrName* 参数。 有关这些值的含义的更多信息，请参阅 MQCONN。 您可以使用 WebSphere MQ JMS 管理工具或 IBM WebSphere MQ Explorer 来设置 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，那么 IBM WebSphere MQ classes for JMS 将访问由 CCDTURL 属性标识的客户机通道定义表，使用 QMANAGER 属性在该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。

当 IBM WebSphere MQ classes for JMS 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，它将仅使用从表中抽取的信息来启动 MQI 通道。 将忽略 ConnectionFactory 对象的任何通道相关属性。

如果您正在使用安全套接字层（SSL），请特别注意以下几点：

- 仅当从客户机通道定义表中抽取的通道定义指定 IBM WebSphere MQ classes for JMS 支持的 CipherSpec 的名称时， MQI 通道才会使用 SSL。
- 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 WebSphere 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联机证书状态协议 (OCSP) 响应程序的位置。 IBM WebSphere MQ classes for JMS 无法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所述来配置 OCSP。

有关将 SSL 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 SSL 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抽取的通道定义可以指定以 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 763 页的『使用 Java for WebSphere MQ JMS 类编写通道出口』。

  还支持使用以 Java 以外的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 *JMS* 客户机重新连接

配置 JMS 客户机以在发生网络，队列管理器或服务器故障后自动重新连接。

使用 `MQConnectionFactory` 类的 `CONNECTIONNAMELIST` 和 `CLIENTRECONNECTOPTIONS` 属性来配置要在连接失败后自动重新连接的客户机连接，或在停止队列管理器后重新连接客户机应用程序的管理请求。

connectionName 列表中连接名称的完整列表仅可供可处理连接名称列表的 set/getconnectionNameList 方法访问。 不处理名称列表的方法 (例如 get/setHostname) 访问列表中的名字。

仅当建立连接后，自动可重新连接的客户机连接才变为可重新连接。

应用程序在自动重新连接后是否继续正常工作取决于其设计。 阅读相关主题以了解如何设计可重新连接的客户机。 在自动重新连接之后，某些现有客户机可能未进行修改而正常工作。

WebSphere MQ classes for Java 不支持客户机自动重新连接。

为了防止连接到失败队列管理器的所有客户机同时重新连接，重新连接尝试将按部分固定和部分随机的时间间隔延迟。

缺省情况下，将按以下时间间隔尝试重新连接：

1. 第一次尝试是在初始延迟 1 秒之后进行的，加上一个随机元素，最长可达 250 毫秒。
2. 第二次尝试将在第一次尝试失败后进行两秒，外加最多 500 毫秒的随机时间间隔。
3. 第三次尝试是在第二次尝试失败后进行 4 秒，加上最多 1 秒的随机时间间隔。
4. 第四次尝试是在第三次尝试失败后的 8 秒，加上最多 2 秒的随机时间间隔。
5. 第五次尝试是在第四次尝试失败后的 16 秒，加上最多 4 秒的随机时间间隔。
6. 第六次尝试以及所有后续尝试将在上次尝试失败后进行 25 秒，加上最长 6 秒和 250 毫秒的随机时间间隔。

此重新连接过程将继续，直到客户机成功地重新连接到队列管理器，或者直到达到最大重新连接时间间隔为止。

如果需要增加缺省值，为了更准确地反映队列管理器恢复或备用队列管理器激活所需的时间量，请使用 **ReconDelay** 属性更改 MQCLIENT.INI 文件中的延迟值。

相关概念

自动客户机重新连接

相关任务

使用配置文件配置客户机

在 *IBM WebSphere 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。

可以使在同一 Java 运行时环境中运行且使用 IBM WebSphere MQ classes for JMS 或 IBM WebSphere MQ 资源适配器通过使用 CLIENT 传输连接到队列管理器的应用程序共享同一通道实例。

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。

如果定义通道时所使用的 **SHARECNV** 参数设为大于 1 的值，那么该数目的对话可共享一个通道实例。 要使连接工厂或激活规范使用此功能，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， JMS 资源适配器的 IBM WebSphere MQ 类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 WebSphere MQ classes for JMS 配合使用时，不保证新的连接请求将始终共享已建立的连接。

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

Number of channel instances = (<**maxPoolDepth**> + 1) / <**SHARECNV**>

Where <**maxPoolDepth**> is the value of the **maxPoolDepth** property and <**SHARECNV**> is the value of **SHARECNV** property on the channel that is used by the activation specification.

其他 JMS 应用程序

Number of channel instances = (<JMS connections> + <JMS sessions>) / <**SHARECNV**>

Where <JMS connections> is the number of connections that are created by the application, where <JMS sessions> is the number of JMS sessions that are created by the application, and <**SHARECNV**> is the value of **SHARECNV** property on the channel that is used by the activation specification.

示例

以下示例说明如何使用公式，计算应用程序通过 IBM WebSphere MQ classes for JMS 或 IBM WebSphere MQ classes for JMS 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 应用程序连接使用 CLIENT 传输连接到队列管理器，并创建 JMS 连接和三个 JMS 会话。 应用程序将用于连接到队列管理器的通道的 **SHARECNV** 属性已设置为值 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。

激活规范示例

激活规范使用 CLIENT 传输连接到队列管理器。 已配置激活规范且 **maxPoolDepth** 属性设置为 10。 配置激活规范使用的通道将 **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。 配置激活规范以将 **maxPoolDepth** 属性设置为 5。 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，每个对话都使用自己的通道实例。

在 *WebSphere MQ JMS* 类中指定客户机连接的端口范围

使用 LOCALADDRESS 属性指定应用程序可绑定到的端口范围。

当 WebSphere MQ classes for JMS 应用程序尝试以客户机方式连接到 WebSphere MQ 队列管理器时，防火墙可能仅允许源自指定端口或一系列端口的那些连接。 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。

可以通过使用 WebSphere 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，那么将抛出 JMSException ，其中包含 WebSphere 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 WebSphere MQ JMS 类可能创建应用程序所需的连接以外的连接，因此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，而 WebSphere MQ classes for JMS 可能需要另外三个或四个端口。 如果发生连接错误，请增大端口范围。

缺省情况下在 WebSphere MQ classes for JMS 中使用的连接池可能会影响可复用端口的速度。 因此，释放端口时，可能会发生连接错误。

WebSphere MQ JMS 类中的通道压缩

WebSphere MQ classes for JMS 应用程序可以使用 WebSphere MQ 工具来压缩消息头或数据。

压缩 WebSphere MQ 通道上流动的数据可以提高通道的性能并减少网络流量。 通过使用 WebSphere MQ 随附的函数，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。

WebSphere MQ classes for JMS 应用程序指定可用于通过创建 java.util.Collection 对象来压缩连接上的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。

将消息异步放入 *IBM WebSphere MQ JMS* 类中

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 PUTSYNCALLOWED (短名称-PAALD)。

  PUTASYNCALLOWED 控制 JMS 应用程序是否可以异步放置消息 (如果 JMS 目标表示的底层队列或主题允许此选项)。

- IBM WebSphere 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利用异步放置功能。

下表显示 PUTASYNCALLOWED 和 DEFPRESP 属性的可能值以及要启用异步放置功能需要哪些值：

表 *131: PUTASYNCALLOWED* 和 *DEFPRESP* 属性确定是否异步放置消息。

WebSphere 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 WebSphere MQ JMS 类检查队列管理器是否已安全地接收消息之前要发送的消息数。

如果检查发现未安全地接收到一条或多条消息，并且应用程序已向连接注册异常侦听器，那么 IBM WebSphere MQ classes for JMS 会调用异常侦听器的 onException() 方法以将 JMS 异常传递给应用程序。

JMS 异常具有错误代码 JMSWMQ0028 ，此代码显示以下消息:

```
At least one asynchronous put message failed or gave a warning.
```

JMS 异常还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。

将预读与 *WebSphere MQ classes for JMS* 配合使用

WebSphere MQ 提供的预读功能允许在事务外部接收的非持久消息在应用程序请求它们之前发送到 IBM WebSphere MQ classes for JMS 。 IBM WebSphere 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 WebSphere 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:

- 允许 JMS 目标属性 READAHEADALLOWED (短名称-RAALD)。 READAHEADALLOWED 控制 JMS 应用程序在获取或浏览事务外部的非持久消息时是否可以使用预读 (如果 JMS 目标表示的底层队列或主题允许此选项)。
- IBM WebSphere 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 属性的可能值以及要启用预读功能需要哪些值:

表 *132: READAHEADALLOWED* 和 *DEFREADA* 属性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。

WebSphere MQ 目标属性	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST 或 AS_Q_DEF 或 AS_T_DEF**
WebSphere MQ 队列属性			
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 WebSphere 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 WebSphere MQ classes for JMS 发送非持久消息:

- MessageConsumer.receive()

- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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，对内部缓冲区中的消息执行的操作取决于 JMS 目标属性 READAHEADCLOSEPOLICY (short name-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。 如果此属性设置为 DELIVER_CURRENT，那么将在应用程序处理当前消息后关闭 JMS 会话，并且将废弃内部缓冲区中的所有剩余消息。

WebSphere MQ JMS 类中的保留出版物

可以将用于 JMS 客户机的 WebSphere MQ 类配置为使用保留发布。

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。 通过将整数属性 JMS_IBM_RETAIN 设置为值 1，在 WebSphere MQ classes for JMS 中完成此操作。 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。

WebSphere MQ classes for JMS 中的 XA 支持

JMS 通过受支持的事务管理器在绑定和客户机方式中支持 XA 兼容的事务。

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。

使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接

WebSphere MQ JMS 应用程序类可以使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接来进行发布/预订消息传递。 必须配置代理和 WebSphere MQ classes for JMS 以启用实时连接。

当应用程序使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接时，应用程序和代理将使用 WebSphere MQ 实时传输来交换消息。 根据配置，还可以使用 WebSphere MQ 多点广播传输将消息传递到应用程序。

有关应用程序如何连接到 WebSphere MQ 队列管理器并使用 WebSphere MQ Enterprise Transport 与 WebSphere Event Broker 或 WebSphere Message Broker 的代理交换消息的信息， 请参阅先前发行版的 WebSphere MQ classes for JMS 的文档。 请注意，为了使用 WebSphere MQ Enterprise Transport，应用程序必须使用以 WebSphere MQ 消息传递提供程序迁移方式运行的连接工厂连接到队列管理器。

配置 *WebSphere Event Broker* 或 *WebSphere Message Broker* 的代理以进行实时连接

要使 WebSphere MQ JMS 应用程序类使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接，必须通过创建和部署消息流以从代理正在侦听并发布消息的 TCP/IP 端口读取消息来配置代理。 根据您的需求，您可能需要以其他方式配置代理。

要配置代理，必须创建并部署下列其中一个消息流:

- 包含 Real-timeOptimizedFlow 消息处理节点的消息流
- 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 节点以侦听用于实时连接的 TCP/IP 端口。 缺省情况下，实时连接的端口号为 1506。

如果您有以下任何需求，那么还必须配置代理:

- 如果希望应用程序使用安全套接字层 (SSL) 认证连接到代理程序
- 如果您希望应用程序使用 HTTP 隧道连接到代理程序
- 如果您希望使用多点广播将消息传递到消息使用者

有关如何配置代理的信息，请参阅 *WebSphere Event Broker* 产品文档 或 *WebSphere Message Broker* 产品文档。

为 *JMS* 配置 *WebSphere MQ* 类以实时连接到 *WebSphere Event Broker* 或 *WebSphere Message Broker* 的代理

要使 WebSphere MQ JMS 应用程序类使用与 WebSphere Event Broker 或 WebSphere Message Broker 的代理的实时连接，必须通过设置连接工厂的特定属性来配置 WebSphere MQ JMS 类。 根据您的需求，可能需要以其他方式配置 WebSphere MQ JMS 类。

要为 JMS 配置 WebSphere MQ 类，必须设置连接工厂的以下属性:

- TRANSPORT 属性必须设置为 DIRECT。

  但是，对于使用 HTTP 隧道连接的应用程序，必须改为将 TRANSPORT 属性设置为 DIRECTHTTP。 请参阅第 774 页的『使用 HTTP 隧道』。
- HOSTNAME 属性必须设置为运行代理的系统的主机名或 IP 地址。
- PORT 属性必须设置为代理侦听实时连接的端口号。

应用程序可以在运行时使用 IBM JMS 扩展或 WebSphere MQ JMS 扩展来动态设置这些属性。 或者，如果连接工厂是受管对象，那么管理员可以使用 WebSphere MQ JMS 管理工具或 WebSphere MQ Explorer 来设置这些属性。

有关属性以及应用程序用于设置其值的方法的信息，请参阅 IBM WebSphere MQ classes for JMS 对象的属性。 有关如何使用 WebSphere MQ JMS 管理工具的信息，请参阅 第 781 页的『使用 WebSphere MQ JMS 管理工具』。 有关如何使用 WebSphere MQ Explorer 的信息，请参阅 WebSphere MQ Explorer 随附的帮助。

如果您有以下任何需求，那么 WebSphere MQ classes for JMS 需要其他配置:

- 如果希望应用程序使用安全套接字层 (SSL) 认证连接到代理程序
- 如果您希望应用程序使用 HTTP 隧道连接到代理程序
- 如果您希望应用程序通过代理服务器连接到代理
- 如果您希望使用多点广播将消息传递到消息使用者

以下部分描述了如何针对其中每个需求配置 WebSphere MQ JMS 类。

使用安全套接字层 (SSL) 认证

可以在与代理程序的实时连接上使用 SSL 认证。 此类型的连接仅支持认证。 不能使用 SSL 对应用程序与代理程序之间流动的消息数据进行加密和解密，也不能使用 SSL 来检测数据的篡改。

请注意此情况与应用程序以客户机方式连接到队列管理器时的差别。 在后一种情况下，您可以使用 WebSphere MQ SSL 支持来加密和解密应用程序与队列管理器之间流动的消息数据，检测数据的篡改以及提供认证。

如果要保护与代理的实时连接上的消息数据，那么可以改为使用代理提供的功能。 您可以将保护质量 (QoP) 值分配给包含要保护的消息的每个主题。 因此，您可以为每个主题选择不同的消息保护级别。 有关代理提

供的消息保护的更多信息，请参阅 *WebSphere Event Broker* 产品文档 或 *WebSphere Message Broker* 产品文档。

要在与代理程序的实时连接上使用 SSL 认证，必须将连接工厂的 DIRECTAUTH 属性设置为 CERTIFICATE。

如果要使用 SSL 进行相互认证，那么代理的 "认证协议类型" 属性必须为对称 SSL 指定选项 R。 如果要仅将 SSL 用于认证代理，那么代理的 "认证协议类型" 属性必须为非对称 SSL 指定选项 S。 但是，在这种情况下，应用程序必须通过调用以用户标识和密码作为参数的 createConnection() 来连接到代理，如以下示例中所示:

```
factory.createConnection("user1", "user1pw");
```

然后，代理使用用户标识和密码 (而不是 SSL) 来认证应用程序。 有关如何配置代理以进行 SSL 认证的更多信息，请参阅 *WebSphere Event Broker* 产品文档 或 *WebSphere Message Broker* 产品文档。

注意:

1. DIRECTAUTH 属性的值确定是否在与代理的实时连接上使用 SSL 认证，而不是 SSLCIPHERSUITE 属性的值。
2. 在与代理程序的实时连接上使用 SSL 认证时， SSLPEERNAME 和 SSLCRL 属性用于执行与应用程序以客户机方式连接到队列管理器时执行的检查相同的检查。
3. WebSphere MQ classes for JMS 可以使用相同的 Java 安全套接字扩展 (JSSE) 密钥库和信任库配置在以下任一情况下提供 SSL 支持:
   - 当应用程序使用与代理程序的实时连接时
   - 应用程序以客户机方式连接到队列管理器时

使用 HTTP 隧道

用于 JMS 应用程序的 WebSphere MQ 类可以使用 HTTP 隧道连接到代理，这意味着应用程序使用 HTTP 协议连接到代理，就像连接到 Web 站点一样。

要在与代理的实时连接上使用 HTTP 隧道，必须将连接工厂的 TRANSPORT 属性设置为 DIRECTHTTP。

HTTP 隧道不能与 SSL 认证结合使用，不能通过代理服务器进行连接，也不能使用多点广播来传递消息。 受支持的 HTTP 协议版本为 1.0。 不支持 HTTP V 1.1 。

通过代理服务器进行连接

WebSphere MQ JMS 应用程序类可以通过代理服务器连接来使用与代理的实时连接。 WebSphere MQ classes for JMS 直接连接到代理服务器，并使用 RFC 2817 中定义的因特网协议要求代理服务器将连接请求转发到代理。

要通过代理服务器连接到代理，必须设置连接工厂的以下属性:

- PROXYHOSTNAME 属性必须设置为运行代理服务器的系统的主机名或 IP 地址。
- PROXYPORT 属性必须设置为代理服务器正在侦听的端口号。

如果 PROXYHOSTNAME 属性未设置或设置为空字符串，那么 WebSphere MQ classes for JMS 会尝试仅使用 HOSTNAME 和 PORT 属性直接连接到代理，并且不会尝试通过代理服务器进行连接。

使用多点广播传递消息

通过使用与代理的实时连接，可以使用多点广播将消息传递到消息使用者。

要启用多点广播，必须将 Topic 对象的多点广播属性设置为必需的多点广播选项。 或者，如果 Topic 对象的多点广播属性设置为 ASCF ，那么连接工厂的多点广播属性必须设置为必需的多点广播选项。

WebSphere MQ classes for JMS 支持包传输层 (PTL) 和实用常规多点广播 (PGM) 多点广播协议，并支持封装的 PGM 协议， PGM/IP 和 PGM UDP 的两种实现。 但是， PGM/IP 支持仅在以下平台上可用:

- AIX (仅限 32 位)
- Linux（x86 平台）
- Linux (zSeries 平台，仅限 32 位)

- Solaris SPARC (仅限 32 位)
- Windows (仅 32 位)
- z/OS

用于 JMS 应用程序服务器设施的 WebSphere MQ 类

本主题描述了 WebSphere MQ JMS 类如何在 Session 类中实现 ConnectionConsumer 类和高级功能。 同时还概述了服务器会话池的功能。

WebSphere MQ classes for JMS 支持在 *Java* 消息服务规范 *V 1.1* 中指定的应用程序服务器设施 (ASF) (请参阅 Sun 的 Java Web 站点 https://java.sun.com)。 此规范确定了该编程模型中的三个角色:

- **JMS 提供程序** 提供了 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，并且 JMS 实现会使消息可供这些线程使用。 JMS 感知应用程序服务器 (例如 WebSphere Application Server) 可以使用此功能来提供高级消息传递功能，例如消息驱动的 bean。

正常应用程序不使用 ConnectionConsumer，但专家 JMS 客户机可能会使用它。 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。 当消息到达队列或主题时， JMS 从池中选择线程并向其传递一批消息。 为此， JMS 运行关联的 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 775 页的『使用 ASF 执行点到点消息传递的常规准则』
- 第 776 页的『使用 ASF 执行发布/预订消息传递的常规准则』
- 第 777 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 745 页的『在 IBM WebSphere 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。

当应用程序从 QueueConnection 对象创建 ConnectionConsumer 时，它指定 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 WebSphere 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 WebSphere 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 777 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 WebSphere 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，请参阅特定平台的 WebSphere 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 引用。

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 WebSphere MQ classes for JMS 处于 WebSphere MQ 消息传递提供程序迁移方式，那么将对非持久 ConnectionConsumer 预订使用单独的队列。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 WebSphere MQ classes for JMS 处于 WebSphere 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 WebSphere 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，请参阅适用于您的平台的 WebSphere 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 ConnectionConsumers 时， JMS 可能需要在许多情况下从队列中除去消息。

这些情况如下所示：

消息格式错误
消息可能到达 JMS 无法解析。

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，会在消息前面添加 WebSphere MQ 死信头 (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)`
- PutAppl 名称为 `"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 778 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 Application Server 工具中，有害消息处理与 WebSphere MQ classes for JMS 中的其他位置略有不同。

有关 WebSphere MQ classes for JMS 中有害消息处理的信息，请参阅 第 745 页的『在 IBM WebSphere 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。

WebSphere Application Server V 5.0 和 V 5.1 中的嵌入式 JMS 提供程序以不同于刚才针对 JMS 的 WebSphere MQ 类描述的方式处理有害消息。 有关嵌入式 JMS 提供程序如何处理有害消息的信息，请参阅相关 WebSphere Application Server 产品文档。

错误处理

本章节介绍了错误处理的各个方面，包括第 778 页的『在 ASF 中从错误情况恢复』和第 778 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 通常以 BytesMessage 或 TextMessage 形式传递意外格式的消息。 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 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 780 页的图 165 概述了 ServerSessionPool 和 ServerSession 功能的原理。

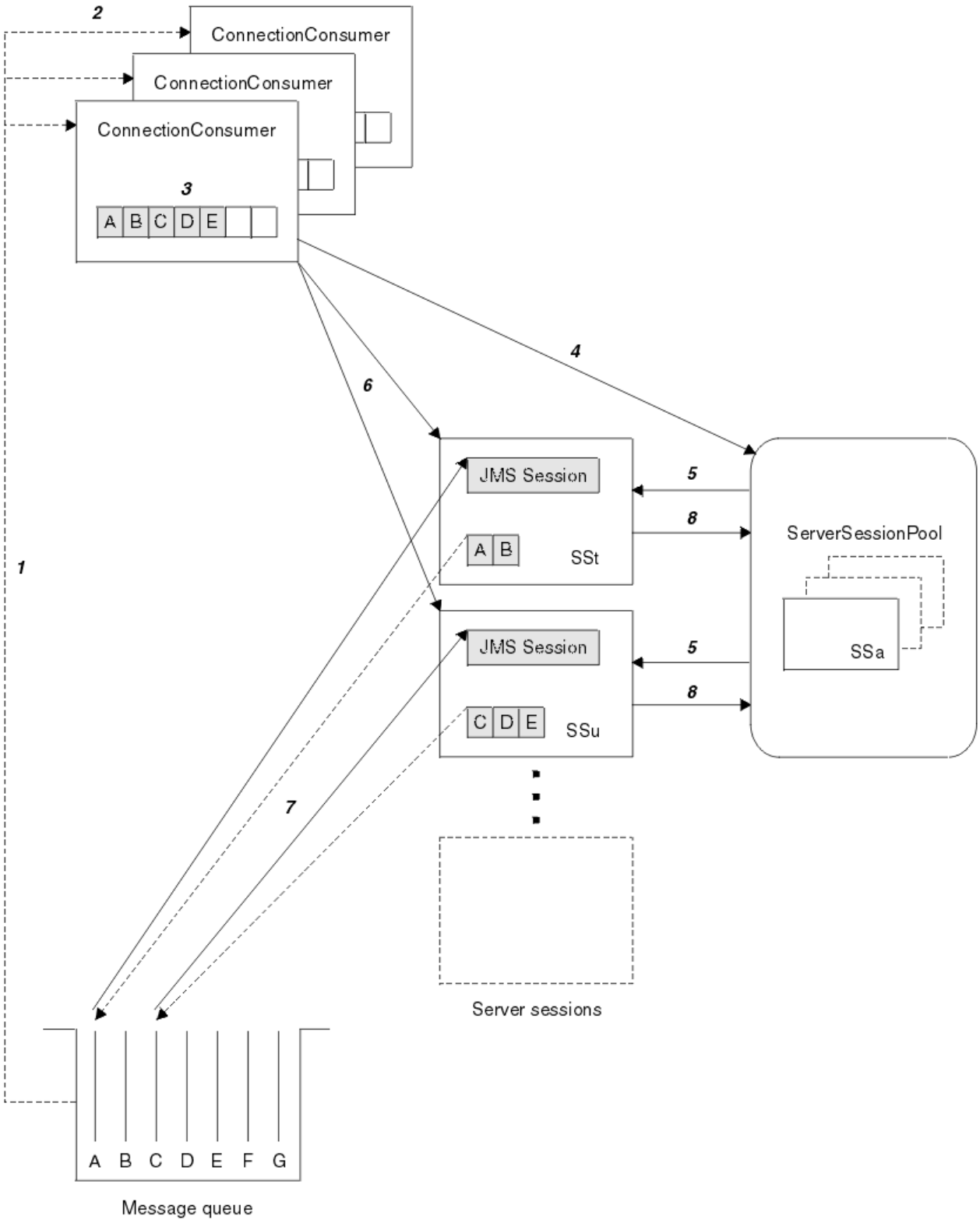


图 *165: ServerSessionPool* 和 *ServerSession* 功能

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。

使用 WebSphere MQ JMS 管理工具

使用管理工具来定义八种类型的 WebSphere MQ JMS 对象类的属性，并将其存储在 JNDI 名称空间中。 然后，应用程序可以使用 JNDI 从名称空间检索这些受管对象。

您可以使用此工具来管理的 WebSphere MQ 类 JMS 对象包括:

- MQConnectionFactory
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

有关这些对象的详细信息，请参阅 第 784 页的『管理 JMS 对象』 。

IBM WebSphere MQ classes for JMS 对象的属性中列出了使用此工具所需的属性类型和值。

该工具还允许管理员处理 JNDI 中的目录名称空间子上下文。 请参阅第 784 页的『使用 WebSphere MQ JMS 管理工具处理子上下文』。

您还可以使用 WebSphere MQ Explorer 创建和配置 JMS 受管对象。

调用 IBM WebSphere MQ classes for JMS 管理工具

管理工具具有命令行界面。 您可以以交互方式使用此过程，也可以使用此过程来启动批处理过程。

交互方式提供命令提示符，您可以在其中输入管理命令。 在批处理方式下，用于启动工具的命令包含包含管理命令脚本的文件的名称。

交互方式

要以交互方式启动工具，请输入以下命令:

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

其中:

-t

启用跟踪 (缺省值为 trace off)

跟踪文件在 "%MQ_JAVA_DATA_PATH%"\errors (Windows) 或 /var/mqm/trace (UNIX) 中生成。 跟踪文件的名称格式如下:

mqjms_*PID*.trc

其中 *PID* 是 JVM 的进程标识。

-v

生成详细输出 (缺省值为 terse 输出)

-cfg 配置文件名
指定备用配置文件。 如果省略此参数，那么将使用缺省配置文件 JMSAdmin.config。 (请参阅 第 782 页的『配置 JMS 管理工具』)

将显示命令提示符，指示工具已准备好接受管理命令。 此提示最初显示为:

```
InitCtx>
```

指示当前上下文 (即，所有命名和目录操作当前引用的 JNDI 上下文) 是 PROVIDER_URL 配置参数中定义的初始上下文 (请参阅 第 782 页的『配置 JMS 管理工具』)。

当您遍历目录名称空间时，提示会更改以反映这一点，以便提示始终显示当前上下文。

批处理方式

要以批处理方式启动工具，请输入以下命令:

```
JMSAdmin <test.scp
```

其中 *test.scp* 是包含管理命令的脚本文件 (请参阅 第 783 页的『WebSphere MQ JMS 管理工具中的管理命令』)。 文件中的最后一个命令必须是 END 命令。

配置 JMS 管理工具

WebSphere MQ JMS 管理工具使用配置文件来设置某些属性的值。 提供了样本文件，您可以根据自己的系统进行定制。

配置文件是由一组由等号 (=) 分隔的键/值对组成的纯文本文件。 如以下示例所示:

```
#Set the service provider
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Set the initial context
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Set the authentication type
    SECURITY_AUTHENTICATION=none
```

(行的第一列中的 # 表示注释或未使用的行。)

WebSphere MQ 随附了样本配置文件。 该文件称为 JMSAdmin.config，可在 <MQ_JAVA_INSTALL_PATH>/bin 目录中找到。 编辑此文件以适合您的系统设置。

使用以下属性的值配置管理工具:

INITIAL_CONTEXT_FACTORY
工具使用的服务提供者。 此属性的受支持值如下所示:

- com.sun.jndi.ldap.LdapCtxFactory (对于 LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (针对文件系统上下文)

您还可以使用不在上述列表中的 InitialContext 工厂。 请参阅第 783 页的『将未列示的 InitialContext 工厂与 WebSphere MQ JMS 管理工具配合使用』以获取更多详细信息。

PROVIDER_URL
会话初始上下文的 URL; 工具执行的所有 JNDI 操作的根。 支持此属性的两种形式:

- ldap://hostname/contextname
- 文件: [drive:] /pathname

LDAP URL 的格式可能有所不同，具体取决于您的 LDAP 提供程序。 请参阅 LDAP 文档以获取更多信息。

SECURITY_AUTHENTICATION
JNDI 是否将安全凭证传递给服务提供者。 仅当使用 LDAP 服务提供者时，才会使用此属性。 此属性可以采用以下三个值之一:

- 无 (匿名认证)

- 简单 (简单认证)
- CRAM-MD5 (CRAM-MD5 认证机制)

如果未提供有效值，那么该属性缺省为 none。 请参阅 第 783 页的『配置 JMS 管理工具的安全性』，以获取有关管理工具的安全性的更多详细信息。

这些属性在配置文件中设置。 调用该工具时，可以使用 `-cfg` 命令行参数指定此配置，如 第 781 页的『调用 IBM WebSphere MQ classes for JMS 管理工具』中所述。 如果未指定配置文件名，那么工具会尝试装入缺省配置文件 (`JMSAdmin.config`)。 它首先在当前目录中搜索此文件，然后在 `<MQ_JAVA_INSTALL_PATH>/bin` 目录中搜索此文件，其中 `<MQ_JAVA_INSTALL_PATH>` 是用于 JMS 安装的 WebSphere MQ 类的路径。

将未列示的 *InitialContext* 工厂与 *WebSphere MQ JMS* 管理工具配合使用

支持两个 InitialContextFactory 值。 您可以通过在 JMS 管理配置文件中设置参数来使用其他 JNDI 上下文。

通过使用 JMSAdmin 配置文件中定义的三个参数，可以使用管理工具来连接到 第 782 页的『配置 JMS 管理工具』中列出的 JNDI 上下文以外的 JNDI 上下文。

要使用其他 InitialContext 工厂:

1. 将 INITIAL_CONTEXT_FACTORY 属性设置为所需的类名。
2. 使用 USE_INITIAL_DIR_CONTEXT， NAME_PREFIX 和 NAME_READABILITY_MARKER 属性定义 InitialContext 工厂的行为。

这些属性的设置在样本配置文件注释中进行了描述。

如果使用其中一个受支持的 INITIAL_CONTEXT_FACTORY 值，那么无需定义此处列出的三个属性。 但是，您可以为它们提供值以覆盖系统缺省值。 如果省略三个 InitialContextFactory 属性中的一个或多个，那么管理工具将根据其他属性的值提供合适的缺省值。

配置 *JMS* 管理工具的安全性

使用 SECURITY_AUTHENTICATION 属性来确定是否将安全凭证传递到服务提供者。

SECURITY_AUTHENTICATION 属性在 第 782 页的『配置 JMS 管理工具』中进行了描述。 其效果如下:

- 如果将此参数设置为 none，那么 JNDI 不会将任何安全凭证传递到服务提供者，并且会执行 匿名认证 。
- 如果将该参数设置为 `simple` 或 `CRAM-MD5`，那么会将安全凭证通过 JNDI 传递到底层服务提供者。 这些安全凭证的格式为用户专有名称 (用户 DN) 和密码。

如果需要安全凭证，那么在工具初始化时会提示您输入这些凭证。 通过在 JMSAdmin 配置文件中设置 PROVIDER_USERDN 和 PROVIDER_PASSWORD 属性来避免此问题。

注: 如果不使用这些属性，那么输入的文本 (包括密码) 将回传到屏幕。 这可能涉及安全问题。

该工具本身不进行认证; 该任务将委派给 LDAP 服务器。 LDAP 服务器管理员必须设置并维护对目录的不同部分的访问特权。 请参阅 LDAP 文档以获取更多信息。 如果认证失败，那么工具将显示相应的错误消息并终止。

有关安全性和 JNDI 的更多详细信息，请参阅 Sun 的 Java Web 站点 (https://java.sun.com) 上的文档。

WebSphere MQ JMS 管理工具中的管理命令

管理工具接受由管理动词及其相应参数组成的命令。

当显示命令提示符时，工具已准备好接受命令。 管理命令通常采用以下格式:

```
verb [param]*
```

其中 **verb** 是 第 784 页的表 133 中列出的其中一个管理动词。 所有有效命令都包含一个动词，该动词以其标准或短格式出现在命令的开头。

动词可以采用的参数取决于动词。 例如， END 动词不能采用任何参数，但 DEFINE 动词可以采用任意数量的参数。 相关主题中讨论了至少包含一个参数的动词的详细信息。

表 133: 管理动词		
动词	**缩写**	**描述**
更改	ALT	更改受管对象的至少一个属性
定义	DEF	创建和存储受管对象，或者创建子上下文
DISPLAY	DIS	显示一个或多个存储的受管对象的属性或当前上下文的内容
DELETE	DEL	从名称空间中除去一个或多个受管对象，或者除去空子上下文
更改	chg	更改当前上下文，允许用户在初始上下文下的任何位置遍历目录名称空间 (暂挂安全隔离)
COPY	CP	创建已存储的受管对象的副本，将其存储在备用名称下
移动	MV	更改存储受管对象的名称
END		关闭管理工具

动词名称不区分大小写。

通常，要终止命令，请按回车键。 但是，您可以通过在回车符之前直接输入加号 (+) 来覆盖此值。 这使您能够输入多行命令，如以下示例中所示:

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

以以下任何字符开头的行将被视为注释并被忽略: * #/。

使用 WebSphere MQ JMS 管理工具处理子上下文

使用动词 **CHANGE**， **DEFINE**， **DISPLAY** 和 **DELETE** 来处理目录名称空间子上下文。

第 784 页的表 134 中描述了这些动词的用法。

表 134: 用于处理子上下文的命令的语法和描述	
命令语法	**描述**
DEFINE CTX (ctxName)	尝试创建名称为 ctxName 的当前上下文的子上下文。 如果存在安全违例，子上下文已存在或提供的名称无效，那么将失败。
显示 CTX	显示当前上下文的内容。 使用 a 对受管对象进行注释，使用 [D]对子上下文进行注释。 还会显示每个对象的 Java 类型。
DELETE CTX (ctxName)	尝试删除名称为 ctxName 的当前上下文的子上下文。 如果上下文未找到，非空或存在安全违例，那么将失败。
CHANGE CTX (ctxName)	更改当前上下文，以便它现在引用名为 ctxName 的子上下文。 可以提供 ctxName 的两个特殊值之一: **= 向上** 移至当前上下文的父代 **= INIT** 直接移至初始上下文 如果指定的上下文不存在或存在安全违例，那么将失败。

管理 JMS 对象

本部分描述了管理工具可以处理的八种类型的对象。 它包含有关每个可配置属性以及可操作这些属性的动词的详细信息。

您还可以使用 WebSphere MQ Explorer 创建和配置 JMS 受管对象。

JMS 对象类型

此表显示八种类型的受管对象。

"关键字" 列显示可在 第 786 页的表 136 中显示的命令中替代 *TYPE* 的字符串。

表 *135:* 管理工具处理的 *JMS* 对象类型

对象类型	关键字	描述
MQConnectionFactory	CF	JMS ConnectionFactory 接口的 WebSphere MQ 实现。 这表示用于在点到点域和发布/预订域中创建连接的工厂对象。
MQQueueConnectionFactory	QCF	JMS QueueConnectionFactory 接口的 WebSphere MQ 实现。 这表示用于在点到点域中创建连接的工厂对象。
MQTopicConnectionFactory	TCF	JMS TopicConnectionFactory 接口的 WebSphere MQ 实现。 这表示用于在发布/预订域中创建连接的工厂对象。
MQQueue	Q	JMS 队列接口的 WebSphere MQ 实现。 这表示点到点域中消息的目标。
MQTopic	T	JMS 主题接口的 WebSphere MQ 实现。 这表示发布/预订域中消息的目标。
MQXAConnectionFactory[1]	XACF	JMS XAConnectionFactory 接口的 WebSphere MQ 实现。 这表示一个工厂对象，用于在点到点域和发布/预订域中创建连接，并且这些连接使用 XA 版本的 JMS 类。
MQXAQueueConnection 工厂[1]	XAQCF	JMS XAQueueConnection 工厂接口的 WebSphere MQ 实现。 这表示用于在使用 XA 版本的 JMS 类的点到点域中创建连接的工厂对象。
MQXATopicConnection 工厂[1]	XATCF	JMS XATopicConnection 工厂接口的 WebSphere MQ 实现。 这表示用于在使用 XA 版本的 JMS 类的发布/预订域中创建连接的工厂对象。

注:

1. 提供这些类供应用程序服务器的供应商使用。 它们不太可能对应用程序员直接有用。

用于 *JMS* 对象的动词

可以使用动词 `ALTER`， `DEFINE`， `DISPLAY`， `DELETE`， `COPY` 和 `MOVE` 来处理目录名称空间中的受管对象。

第 786 页的表 136 总结了这些动词的用法。 将 *TYPE* 替换为表示所需受管对象的关键字，如 第 785 页的表 135 中所示。

表 *136:* 用于处理受管对象的命令的语法和描述

命令语法	描述
ALTER *TYPE*(名称) [属性] *	尝试使用提供的属性来更新受管对象的属性。 如果存在安全违例，如果找不到指定的对象，或者如果提供的新属性无效，那么将失败。
DEFINE *TYPE*(名称) [属性] *	尝试使用提供的属性创建类型为 *TYPE* 的受管对象，并将其存储在当前上下文中的名称 name 下。 如果存在安全违例，如果提供的名称无效或存在该名称的对象，或者如果提供的属性无效，那么将失败。
DISPLAY *TYPE*(名称)	显示在当前上下文中名称 name 下绑定的类型为 *TYPE* 的受管对象的属性。 如果对象不存在，或者存在安全违例，那么将失败。
DELETE *TYPE*(名称)	尝试从当前上下文中除去名称为 name 的 *TYPE* 类型的受管对象。 如果对象不存在，或者存在安全违例，那么将失败。
COPY *TYPE*(nameA) 类型(nameB)	生成类型为 *TYPE* 的受管对象的副本，其名称为 nameA，命名为副本 nameB。 这一切都发生在当前上下文的作用域内。 如果要复制的对象不存在，名称为 nameB 的对象存在或者存在安全违例，那么将失败。
MOVE *TYPE*(nameA) *TYPE*(nameB)	将名为 nameA 的类型为 *TYPE* 的受管对象移动 (重命名) 到 nameB。 这一切都发生在当前上下文的作用域内。 如果要移动的对象不存在，名称为 nameB 的对象存在或者存在安全违例，那么将失败。

使用 *WebSphere MQ JMS* 管理工具创建对象

使用 DEFINE 命令创建对象并将其存储在 JNDI 名称空间中。

使用以下命令语法:

```
DEFINE TYPE(name) [property]*
```

即， DEFINE 动词，后跟 *TYPE*（name）受管对象引用，后跟零个或多个 属性 (请参阅 IBM WebSphere MQ classes for JMS 对象的属性)。

JMS 对象的 *LDAP* 命名注意事项

要在 LDAP 环境中存储对象，必须为其提供符合特定约定的名称。 管理工具可以通过添加缺省前缀来帮助您遵守命名约定。

一种命名约定是对象和子上下文名称必须包含前缀，例如 cn= (公共名称) 或 ou= (组织单元)。

管理工具允许您引用没有前缀的对象和上下文名称，从而简化了 LDAP 服务提供者的使用。 如果未提供前缀，那么工具会自动向您提供的名称添加缺省前缀。 对于 LDAP ，这是 cn=。

您可以通过在 JMSAdmin 配置文件中设置 NAME_PREFIX 属性来更改缺省前缀，如 第 783 页的『将未列示的 InitialContext 工厂与 WebSphere MQ JMS 管理工具配合使用』中所述。

如以下示例所示。

```
InitCtx> DEFINE Q(testQueue)
InitCtx> DISPLAY CTX
    Contents of InitCtx
      a  cn=testQueue              com.ibm.mq.jms.MQQueue
     1 Object(s)
```

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

虽然提供的对象名 (testQueue) 没有前缀，但工具会自动添加一个前缀以确保符合 LDAP 命名约定。 同样，提交命令 DISPLAY Q(testQueue) 也会导致添加此前缀。

您可能需要配置 LDAP 服务器以存储 Java 对象。 有关帮助此配置的信息，请参阅 LDAP 服务器的文档。

创建 *JMS* 对象的样本错误条件

创建对象时，可能会出现许多常见错误情况。

以下是这些错误条件的示例:

CipherSpec 已映射到 CipherSuite

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
  WARNING: Converting CipherSpec RC4_MD5_US to
  CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

对象的属性无效

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Unable to create a valid object, please check the parameters supplied
  Invalid property for a QCF: PRI
```

属性值的类型无效

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for CCS property: English
```

属性冲突-客户机/bin 编码

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: Client-bindings attribute clash
```

属性冲突-退出初始化

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: ExitInit string supplied
  without Exit string
```

属性值超出有效范围

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for PRI property: 12
```

未知的属性

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Unable to create a valid object, please check the parameters supplied
  Unknown property: PIZZA
```

以下是在 Windows 上从 JMS 应用程序中查找 JNDI 受管对象时可能出现的错误情况的示例。

1. 如果您正在使用 WebSphere JNDI 提供程序 com.ibm.websphere.naming.WsnInitialContextFactory，那么必须使用正斜杠 (/) 来访问在子上下文中定义的受管对象; 例如， jms/MyQueueName。 如果使用反斜杠 (\) ，那么将抛出 InvalidName 异常。
2. 如果您正在使用 Sun JNDI 提供程序 com.sun.jndi.fscontext.RefFSContextFactory，那么必须使用反斜杠 (\) 来访问子上下文中定义的受管对象; 例如， ctx1\\fred。 如果使用正斜杠 (/) ，那么将抛出 NameNotFoundException 。

使用 WebSphere MQ Explorer for JMS 配置

使用 IBM WebSphere MQ Explorer 图形用户界面从 WebSphere MQ 对象和来自 JMS 对象的 WebSphere MQ 对象创建 JMS 对象，以及用于管理和监视其他 WebSphere MQ 对象。

开始之前

在使用 WebSphere MQ Explorer 创建和配置 JMS 受管对象之前，请添加初始上下文以定义将 JMS 对象存储在命名和目录服务中的 JNDI 名称空间的根目录。 有关更多信息，请参阅 IBM WebSphere MQ Explorer 用户帮助以获取 JMS 受管对象。

关于此任务

您可以使用 IBM WebSphere MQ Explorer 执行以下任务，可以从 IBM WebSphere MQ Explorer 中的现有对象上下文执行，也可以从 "创建新对象" 向导中执行。 请参阅 WebSphere MQ Explorer 帮助，以获取有关某些典型任务的 WebSphere MQ Explorer 用户帮助的示例。

过程

- 从以下任何 WebSphere MQ 对象创建 JMS 连接工厂：
  a) WebSphere MQ 队列管理器 (无论是在本地计算机上还是在远程系统上)。
  b) WebSphere MQ 通道
  c) WebSphere MQ 侦听器
- 使用 JMS 连接工厂将 WebSphere MQ 队列管理器添加到 WebSphere MQ Explorer
- 从 WebSphere MQ 队列创建 JMS 队列
- 从 JMS 队列创建 WebSphere MQ 队列
- 从 WebSphere MQ 主题创建 JMS 主题，该主题可以是 WebSphere MQ 对象或动态主题
- 从 JMS 主题创建 WebSphere MQ 主题

使用 WebSphere MQ Headers 软件包

WebSphere MQ 头包提供了一组帮助程序接口和类，可用于处理消息的 WebSphere MQ 头。 通常，您使用 WebSphere MQ Headers 软件包，因为您希望通过使用命令服务器 (通过使用可编程命令格式 (PCF) 消息) 来执行管理服务。

关于此任务

WebSphere MQ 头包位于 `com.ibm.mq.headers` 和 `com.ibm.mq.pcf` 包中。 您可以将此工具用于 WebSphere MQ 提供用于 Java 应用程序的两个备用 API:

- WebSphere MQ classes for Java (也称为 WebSphere MQ Headers Base Java)。
- WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS ， 也称为 WebSphere MQ JMS)。

WebSphere MQ Base Java 应用程序通常处理 MQMessage 对象，头支持类可以直接与这些对象交互，因为它们本机了解 WebSphere MQ Base Java 接口。

在 WebSphere MQ JMS 中，消息的有效内容通常是字符串或字节数组对象，可以使用 DataInput 和 DataOutput 流来处理这些对象。 WebSphere MQ 头包可用于与这些数据流交互，并且适用于处理由 WebSphere MQ JMS 应用程序发送和接收的任何 MQ 消息。

因此，虽然 WebSphere MQ 头包包含对 WebSphere MQ Base Java 包的引用，但它也适用于 WebSphere MQ JMS 应用程序，并且适合在 Java Platform, Enterprise Edition (Java EE) 环境中使用。

使用 WebSphere MQ 头包的典型方法是处理可编程命令格式 (PCF) 中的管理消息，例如，出于以下任何原因:

- 用于访问有关 WebSphere 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 WebSphere 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 WebSphere MQ Base Java API。

过程

- 要使用 WebSphere MQ Headers 包来处理 WebSphere MQ classes for Java 的消息头，请参阅 第 789 页的『与 WebSphere MQ classes for Java 配合使用』。
- 要使用 WebSphere MQ 头包来处理 JMS 的消息头，请参阅 第 789 页的『与 WebSphere MQ classes for JMS 配合使用』。

与 WebSphere MQ classes for Java 配合使用

WebSphere MQ classes for Java 应用程序通常处理 MQMessage 对象，而 Headers 支持类可以直接与这些对象进行交互，因为它们本机了解 Java 接口的 WebSphere MQ 类。

关于此任务

WebSphere MQ 提供了一些样本应用程序，用于演示如何将 WebSphere MQ 头包与 WebSphere MQ 基本 Java API (WebSphere MQ Java 类) 配合使用。

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 WebSphere MQ Java 类发送此 PCF 消息。

根据您使用的平台，这些样本安装在 WebSphere MQ 安装的 `samples` 或 `tools` 目录中的 `pcf` 目录下 (请参阅 第 554 页的『 WebSphere MQ Java 类的安装目录』)。

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 WebSphere MQ Java 类发送此 PCF 消息。

相关概念

第 572 页的『使用 WebSphere MQ Java 类处理 WebSphere MQ 消息头』
提供了表示不同类型的消息头的 Java 类。 同时还提供了两个助手类。

第 577 页的『使用 WebSphere MQ Java 类处理 PCF 消息』
提供了 Java 类以创建和解析 PCF 结构化消息，并促进发送 PCF 请求和收集 PCF 响应。

与 WebSphere MQ classes for JMS 配合使用

要将 WebSphere MQ 头与 WebSphere MQ classes for JMS 配合使用，请执行与 WebSphere MQ classes for Java 相同的基本步骤。 可以使用 WebSphere MQ 头包和与 WebSphere MQ classes for Java 相同的样本代码，以完全相同的方式创建 PCF 消息并解析响应。

关于此任务

要使用 WebSphere 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 或 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 消息以及从响应消息中抽取信息的方式与针对 Java 的 WebSphere MQ 类相同 (请参阅 第 789 页的『与 WebSphere MQ classes for Java 配合使用』)。

过程

1. 创建表示 SYSTEM.ADMIN.COMMAND.QUEUE。

WebSphere 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 MQMD 值的 JMS 字节消息。

   需要创建 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 字节消息并将 com.ibm.mq.headers，pcf.PCFMessage 对象写入其中。 PFMessage 对象 (pcfCmd) 先前是使用 WebSphere MQ 头包构建的。 （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 676 页的『JMS 消息』

JMS 消息由头，属性和主体组成。 JMS 定义五种类型的消息体。

在 WebSphere MQ 中使用 Web Service

您可以使用 IBM WebSphere MQ Transport for SOAP 或 IBM WebSphere MQ Bridge for HTTP 为 Web Service 开发 IBM WebSphere MQ 应用程序。

IBM WebSphere MQ Transport for SOAP 提供针对 SOAP 的 JMS 传输。 IBM WebSphere MQ Transport for SOAP 还集成到其他环境中，例如 Microsoft Windows Communication Foundation， WebSphere Application Server 和 CICS Transaction Server。

有关 IBM WebSphere MQ Transport for SOAP 的更多信息，请参阅 第 791 页的『WebSphere MQ Transport for SOAP』。

通过 IBM WebSphere MQ Bridge for HTTP，客户机应用程序可以与 IBM WebSphere MQ 交换消息，而无需安装 WebSphere MQ MQI 客户机。 您可以从具有 HTTP 功能的任何平台或语言调用 WebSphere MQ。

有关 IBM WebSphere MQ Bridge for HTTP 的更多信息，请参阅 第 858 页的『用于 HTTP 的 WebSphere MQ 网桥』。

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 WebSphere MQ 应用程序。 使用本主题中的链接可获取有关对应用程序开发者有用的 IBM WebSphere MQ 概念的信息。

第 64 页的『决定要使用的编程语言』
使用此信息可了解 IBM WebSphere MQ 支持的编程语言和框架以及使用这些语言和框架的一些注意事项。

第 73 页的『设计 IBM WebSphere MQ 应用程序』
当您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 WebSphere MQ 提供的功能。

第 78 页的『样本 WebSphere MQ 程序』
使用此主题集合来了解不同平台上的样本 WebSphere MQ 程序。

第 162 页的『编写排队应用程序』
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 293 页的『编写客户机应用程序』
在 WebSphere MQ 上编写客户机应用程序所需的知识。

第 232 页的『编写发布/预订应用程序』
开始编写发布/预订 WebSphere MQ 应用程序。

第 357 页的『构建 IBM WebSphere MQ 应用程序』
使用此信息可了解如何在不同平台上构建 IBM WebSphere MQ 应用程序。

第 462 页的『处理程序错误』
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。

WebSphere MQ Transport for SOAP

WebSphere MQ Transport for SOAP 提供了用于 SOAP 的 JMS 传输。 WebSphere MQ Transport for SOAP 还集成到其他环境中，例如 Microsoft Windows Communication Foundation， WebSphere Application Server 和 CICS Transaction Server。

IBM WebSphere MQ Transport for SOAP 的介绍

IBM WebSphere MQ Transport for SOAP 提供针对 SOAP 的 JMS 传输。 WebSphere MQ SOAP 发送方和侦听器提供了调用 Web Service 的方法。

WebSphere MQ SOAP 侦听器支持由 .NET Framework 1， .NET Framework 2 和 Axis 1.4 托管的服务。 WebSphere MQ SOAP 发送方支持在 .NET Framework 1， .NET Framework 2， Axis 1.4 和 Axis2 上运行的 Web Service 客户机。 客户机可以是 WebSphere MQ 服务器或客户机应用程序。 IBM WebSphere MQ Transport for SOAP 还集成到其他环境中，例如 Microsoft Windows Communication Foundation，WebSphere Application Server 和 CICS Transaction Server。

集成到 Microsoft Windows Communication Foundation 是 IBM WebSphere MQ .NET Framework 支持 3 的一部分。

IBM WebSphere MQ Transport for SOAP 是一组协议和工具，用于通过 IBM WebSphere MQ 使用 JMS 来传输 SOAP 消息。 针对不同应用程序环境，此传输打包方式不同，如第 792 页的表 137 中所示。

表 *137: IBM WebSphere MQ Transport for SOAP* 的应用程序环境

	与其他 WebSphere MQ 组件集成	**集成到框架中**
作为 WebSphere MQ 安装的一部分提供	.NET Framework 1 。NET Framework 2 Axis 1.4	Windows Communication Foundation (.NET Framework 3) Axis2 (仅限客户机)
在其他软件包中提供		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server for Multiplatforms

将 IBM WebSphere MQ Transport for SOAP 集成到应用程序框架中可简化 Web Service 的开发，还可简化将其部署到 IBM WebSphere MQ 的过程。

通过其他 IBM WebSphere MQ SOAP 组件，您可以直接与 WebSphere MQ SOAP 组件进行交互以开发和部署服务。 使用 IBM WebSphere MQ SOAP 工具来配置 Web Service 和 Web Service 客户机，并将其部署到 IBM WebSphere MQ。

在集成环境中，开发和部署较为简单。 使用开发和部署 SOAP HTTP Web Service 时所用的相同工具来开发和部署。 您仍必须使用 WebSphere MQ 工具配置所需的 IBM WebSphere MQ 队列，通道和队列管理器。

可以通过任一此类环境混合和匹配 IBM WebSphere MQ SOAP 客户机和服务器。

优势

WebSphere MQ Transport for SOAP 为现有 IBM WebSphere MQ 用户提供了以下主要优势:

使用 IBM WebSphere MQ 网络连接现有 Web Service。

服务可为编写的服务，或作为已部署的其他打包软件应用程序的接口提供的服务。

使用现有 WebSphere MQ 网络来连接 Web Service 的好处。 IBM WebSphere MQ 传输从受管且可靠的排队消息传递服务中获得优势。

编写新应用程序或转换现有应用程序，以使用 SOAP 而不是 IBM WebSphere MQ 接口。

通常，应用程序需要开发特定的 WebSphere MQ 适配器以与另一个应用程序集成。 适配器具有两个部分：用于将消息放入传输和从传输获取消息的连接器部分，以及用于将数据转换为特定于应用程序的格式以及从这些格式转换数据的适配器部分。 集成每对应用程序是一个全新的挑战。

SOAP 的优势来自于可基于 SOAP 实现标准化，以定义应用程序接口并可选择传输。 无需编写特定于应用程序的适配器，可选择是将 IBM WebSphere MQ 还是 HTTP 用作连接器。 所选的传输取决于所需的服务质量和连接。

对于现有 SOAP over HTTP 用户， WebSphere MQ Transport for SOAP 的优势来自使用受管且可靠的异步传输。 这提供了双重优势:

真正异步的编程模型，可实现可用性和提高性能。
通过使用异步客户机接口，客户机和服务应用程序不需要同时可用。 将存储客户机发送的请求，直到提供了服务对其进行处理。

旨在实现可靠性和可用性的已构建受管网络。

通过将 IBM WebSphere MQ 选作传输，可获得使用提供可靠消息传递的受管网络的优势。

相反，诸如 HTTP 和 FTP over TCP/IP 之类的传输是不受管理的。 未受管网络非常适用于不可预测的连接：具有的管理任务较少。

摘要

IBM WebSphere MQ Transport for SOAP 提供以下组件：

- SOAP/JMS 传输绑定在 WSDL 文档中用于将 SOAP 服务绑定到 JMS 传输。 SOAP/JMS 绑定的 WebSphere MQ 实现使用采用以下两种格式之一的 URI:

  WebSphere MQ Transport for SOAP

  ```
  jms:/queue?&Name=Value&Name=Value...
  ```

  W3C 候选者建议的 WebSphere MQ 有线格式

  ```
  jms:queue:qName?connectionFactory=connectQueueManager(qMgrName)&Name=Value&Name=Value...
  ```

- SOAP 消息到 WebSphere MQ 消息的映射。
- 两个 IBM WebSphere MQ 用于接收 SOAP 请求的 SOAP 侦听器，一个用于 Java ，另一个用于 .NET Framework 1 或 .NET Framework 2。 侦听器使用 .NET 或 Axis 1.4 来处理 SOAP 请求。
- 用于创建 IBM WebSphere MQ SOAP 请求的两个 IBM WebSphere MQ SOAP 发送方。 Web Service 客户机向发送方注册以处理 `jms:` SOAP 请求。
- 与 Windows Communication Foundation (WCF) (有时称为 .NET 3) 集成，以发送和接收 WebSphere MQ Transport for SOAP 消息。
- 将客户机与 Axis2(有时称为 JAX-WS) 集成，以发送 WebSphere MQ Transport for SOAP 或 W3C SOAP JMS 消息。
- 命令 **amqwdeployWMQService**，用于创建开发和运行时组件和脚本，以使用 IBM WebSphere MQ Transport for SOAP 来部署 Web Service。
- 样本 Java 和 .NET 客户机和服务代码。
- 用于设置类路径的脚本以及其他实用程序脚本。

在集成环境中，发送方和侦听器作为开发和部署工具的扩展集成到每个环境中。

SOAP 与 WebSphere MQ 的集成

WebSphere MQ Transport for SOAP 扩展了 SOAP 以及 Web Service 工具和运行时， WebSphere MQ 作为 HTTP for SOAP 的替代传输。 您无需修改现有 Web Service ，即可将 WebSphere MQ Transport for SOAP 用作传输。 该传输对 SOAP/JMS 使用定制 URI 格式。 Axis2 客户机以有限方式支持 SOAP/JMS 的 W3C URI 格式。

必须向 .NET Framework 1 ， .NET Framework 2 和 Axis 1.4 环境中的客户机添加额外的代码行。 在 Axis 2 和 Windows Communication Foundation (WCF) 客户机中不需要其他代码。 WebSphere MQ SOAP 侦听器在 .NET Framework 1 ， .NET Framework 2 和 Axis 1.4 环境中运行服务。 WebSphere MQ Transport for SOAP 集成到其他一些应用程序服务器环境中，包括 WCF ， CICS 和 WebSphere Application Server。

什么是 SOAP?

SOAP[9] 描述应用程序用于交换请求、回复和数据报的消息与交互协议的标准化格式。 SOAP 独立于用于传输消息的传输，也独立于用于发送和接收消息的应用程序环境。 W3C 简洁地定义了 SOAP V1.2：

SOAP V1.2 提供基于 *XML* 的信息的定义，可使用此定义在分散的分布式环境中的同级之间交换结构化和类型化信息。[10].

要使用 SOAP ，必须将其绑定到传输，例如 HTTP ，电子邮件或 WebSphere MQ。

SOAP 协议绑定框架是用于在其他协议（例如，HTTP）上承载 SOAP 消息的规则集。 SOAP Version 1.2 Part 2: Adjuncts (Second Edition) 描述了 SOAP HTTP 绑定。

W3C 候选建议 (4 2009 年 6 月， SOAP over Java Message Service 1.0) 描述了 SOAP JMS 绑定的建议。 由于 JMS 是 API 规范，而不是传输协议，因此 JMS SOAP 建议不会描述 SOAP JMS 消息的有线格式。 它描述了 SOAP 交互协议和 JMS API 绑定。 因此，使用 JMS SOAP 建议时，仍必须对 SOAP 客户机和 SOAP 服务

[9] 历史上，首字母缩略词代表简单对象访问协议。
[10] W3C: SOAP 版本 1.2 部件 0

器使用相同的 JMS 实现。 它使 SOAP JMS 应用程序能够在 JMS 的任何实现上运行。 如果服务器和 JMS 实现都符合 JCA 规范，那么可以将 JMS 实现插入到 J2EE 应用程序服务器中。 WebSphere MQ JMS 符合 JCA 规范，并且可以插入到兼容的应用程序服务器中。

WebSphere MQ Transport for SOAP 绑定类似于建议的 W3C 标准，但它不相同。 其用法在主题 MQRFH2 SOAP 设置中进行了描述。 与 W3C 候选推荐不同，未正式指定 SOAP 绑定。 实际上，它是 HTTP 绑定，服务地址采用 `jms:/queue?name=value&name=value...` 格式而不是 `http://authority/path?query#fragment`。 `jms:` 不是正式注册的 IANA URI 方案。

什么是 Web Service?

SOAP 可以使采用不同语言编写并在不同平台上运行的程序使用各种传输协议进行通信。 SOAP 是一种协议规范。 Web Service 是通过可使用因特网协议访问的 SOAP 接口提供服务的应用程序。

SOAP 的重要目标是提供客户机可轻松使用的服务。 将客户机设计为使用服务后，可对调用编程以调用服务，而无需参考外部文档。 服务接口在 WSDL 文档中使用 XML 进行了描述。 查询 `http://authority/path?wsdl` 返回 SOAP 服务的 WSDL 描述。

提示: 部署 Web Service 以使用 WebSphere MQ 时，还会将该服务部署到 HTTP ，以便标准 WSDL 查询正常工作。

开发 Web Service

Web Service 具有客户机和服务部分。 首先从 WSDL 中的接口描述开始或通过遵循编写服务类的规则编写了服务。 Web Service 工具箱具有实用程序，用于从类的接口定义生成 WSDL；例如，**java2wsdl** 或 **disco**。 同时还提供了工具，用于从 WSDL 接口描述生成框架或对其进行分类；例如，**wsdl2java**、**wsimport** 或 **wsdl**。 前者称为自底向上开发，后者称为自顶而下开发。

WebSphere MQ Transport for SOAP 中的 **amqwdeployWMQService** 命令使用这些工具来生成 WSDL ，客户机存根和客户机代理。

通常，使用目标为特定应用程序服务器环境的集成开发环境来编写 Web Service：

Eclipse IDE for Java EE 开发者
为 Axis 2 创建 Web Service。 支持 JAX-RPC 和 JAX-WS

Rational Application Developer V7.5
为 WebSphere Application Server V7 和先前版本以及 Axis 创建 Web Service。 支持 JAX-RPC 和 JAX-WS。

WebSphere Integration Developer V6.2
为 WebSphere Process Server 和 WebSphere ESB 创建 Web Service。 支持 JAX-RPC 和 JAX-WS。

Visual Studio 2008 (V9)
为 .NET Framework 3.5 和更低版本创建 Web Service (Windows Communication Foundation)

Visual Studio 2005 (V8)
为 .NET Framework 2 和更低版本创建 Web Service

您可以将这些工具中的任何一个与 WebSphere MQ Transport for SOAP 结合使用。 开发要与 HTTP 配合使用的服务后，请使用 **amqwdeployWMQService** 工具来部署要将 WebSphere MQ 用作传输的服务。 您可以使用工具的输出来编写新客户机，或者修改现有客户机以使用 WebSphere MQ Transport for SOAP。

如果 WebSphere MQ Transport for SOAP 集成到应用程序环境中，那么您不需要使用 **amqwdeployWMQService** 工具或修改客户机代码。 客户机 SOAP 层将具有前缀 `jms:` 的 URI 的客户机请求定向到 WebSphere MQ Transport for SOAP。 服务器 SOAP 层调用 WebSphere MQ Transport for SOAP 以等待 `jms:` SOAP 请求，并将响应返回到 WebSphere MQ Transport for SOAP。

通常， .NET 服务是使用代码中的 Web Service 注释自底向上开发的，而 Java 服务是使用 WSDL 接口定义自顶向下开发的。 方法的差异正在缩小，因为 Java Standard Edition V 6 支持 JAX-WS 2.0，并使用注释来限定服务接口的定义。 现在，自下而上地开发 Java 服务就像自上而下一样容易。 选择哪种方法与所使用的开发方法有关。

Web Service 客户机通过使用 WSDL 服务定义和生成的客户机存根和代理在服务之后编写。 在一些应用程序中，编写客户机时，服务定义未知。 客户机检索服务 WSDL 并自动创建服务请求。 更为常见的是，服务定

义已知，但部署服务的地址未知。 Web Service 工具箱生成接口，以供客户机用于发出服务请求。 在必要时，客户机提供服务地址。 在第三种情况下，WSDL 包含客户机所需的所有信息。 WSDL 包含服务的接口和地址。 Web Service 工具箱生成的代码包含客户机发出服务请求所需的所有信息。

您可以将这三种样式中的任何一种用于 WebSphere MQ Transport for SOAP。

Web Service 应用程序环境

Web Service 工具箱需要将服务的 WSDL 定义映射到在 SOAP 请求和响应中传输的字节流。 字节流由 SOAP 规范定义，且包含在 SOAP 包络中。 SOAP 包络显示在第 795 页的图 166 中。

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->
<!-- headers... -->
</soap:Header>
<soap:Body>
<!-- payload or fault message -->
</soap:Body>
</soap:Envelope>
```

图 *166: SOAP* 包络

从 SOAP 包络到语言绑定的映射以及从语言绑定到 SOAP 的映射部分标准化，部分专有。 映射是 .NET 体系结构的基础，并作为公共语言运行时 (CLR) 的一部分提供。 此映射在 Java 中通过 JAX 规范进行标准化。 由于 Java 映射是标准化的，因此 Java Web Service 客户机和服务可在不同的基于 Java 的应用程序环境之间移植。 JAX-RPC（有时称为 JAX-WS 1.0）是目前最常用的映射。 Axis 1.4 支持此映射。 JAX-WS（有时称为 JAX-WS 2.0）是经过大幅改善的标准，可能很快会替代 JAX-RPC。 Axis 2.0 支持 JAX-WS。 WebSphere MQ 7.0.1 不支持 JAX-WS 和 Axis 2。

WebSphere MQ Transport for SOAP 不会改变 SOAP 包络的内容，并且内容不会影响传输。 语言绑定会影响 WebSphere MQ Transport for SOAP。 WebSphere MQ 7.0.1 使用 WebSphere MQ Transport for SOAP 随附的代码和实用程序支持 .NET Framework 1， .NET Framework 2 和 Axis 1.4 。 对 WebSphere Transport for SOAP in .NET Framework 3 和 3.5 的支持是使用 Windows Communication Foundation 的 WebSphere MQ 定制通道实现的。

其他 SOAP 开发和运行时环境可能会提供对 WebSphere MQ Transport for SOAP 的支持，并支持不同的语言。 例如，在 CICS 上运行的 Web Service 支持 COBOL 和 PL/1 之类的语言。

注: 使用的映射不会影响 Web Service 的互操作性。 您可以混合并匹配使用 .NET， JAX-RPC 和 JAX-WS 映射编写的客户机和服务。

什么是 WebSphere MQ Transport for SOAP?

WebSphere MQ Transport for SOAP 是 SOAP 绑定和 Web Service 工具箱。 它们一起使应用程序能够使用 WebSphere MQ 而不是 HTTP 来交换 SOAP 消息。 第 795 页的图 167 将 WebSphere MQ 显示为作为 SOAP 传输的 HTTP 的替代方法。

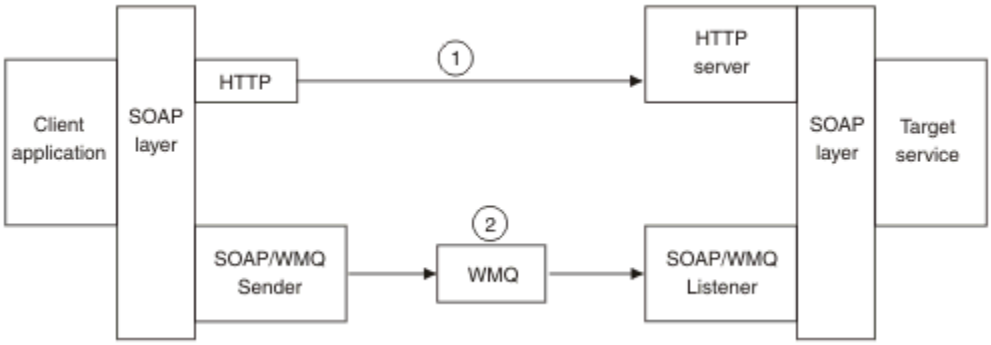


图 *167: WebSphere MQ Transport for SOAP* 概述

SOAP over HTTP 在图中显示为 (1)。 客户机 SOAP 层将请求转换为 SOAP 消息，HTTP 组件则通过 TCP/IP 发送。 HTTP 服务器组件通常在 TCP/IP 端口 80 上侦听 HTTP 请求。 如果请求针对 SOAP 服务，那么 HTTP 服务器组件会调用 SOAP 层以将 SOAP 请求转换为方法调用。 然后，会返回响应。

SOAP over WebSphere MQ 显示为 (2)。 客户机应用程序将 WebSphere MQ SOAP 发送方组件注册为具有 SOAP 层的 `jms:` 协议的处理程序。 SOAP 层将寻址到 `jms:` 的 SOAP 消息传递到 WebSphere MQ SOAP 发送方。 发送方在消息中使用 URI，将消息置于具有所需服务质量的请求队列上。 相应的 WebSphere MQ SOAP 侦听器等待其请求队列上的消息，并调用 SOAP 层以处理请求和返回响应。

SOAP 发送方和侦听器是正常的 WebSphere MQ 程序。 可以将其连接到相同队列管理器（如第 796 页的图 168 中所示），或者连接到不同队列管理器（请参阅第 797 页的图 169）。 可通过客户机连接来连接客户机。

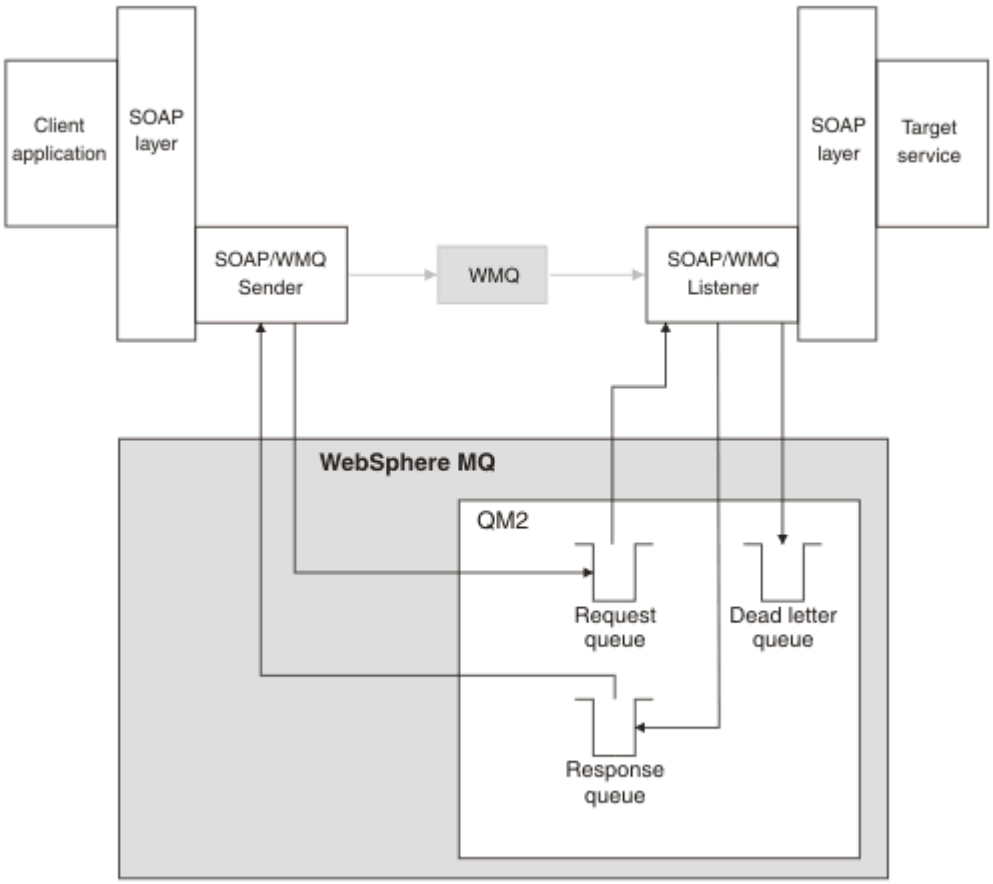


图 *168: SOAP/WebSphere MQ (*单个队列管理器*)* 使用的队列

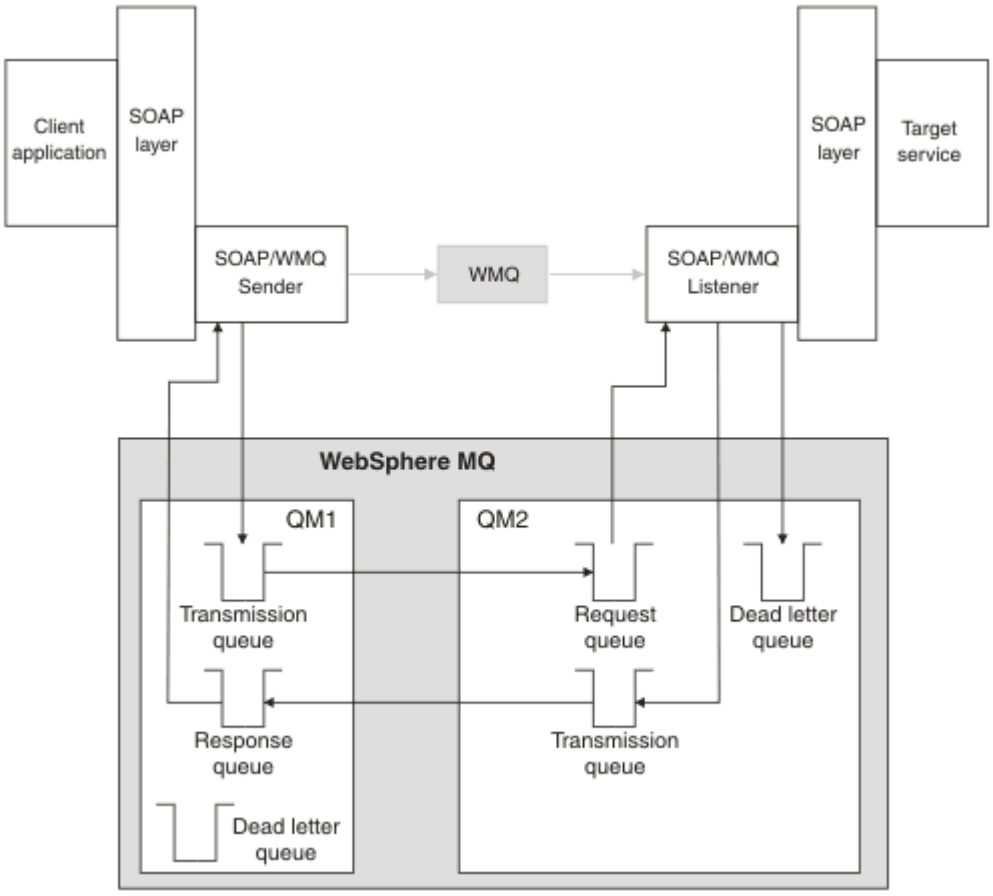


图 *169: SOAP/WebSphere MQ* 使用的队列 *(*单独的队列管理器*)*

用于将 SOAP 绑定到 JMS 的 W3C 候选建议。

W3C 候选值建议定义了 SOAP over JMS 绑定；SOAP over Java Message Service 1.0。 URI Scheme for Java(tm) Message Service 1.0 对其示例也很有用。[11].

某些应用程序框架 (例如 WebSphere Application Server v7) 支持 W3C 候选建议。 使用 Axis2 客户机发送使用与 W3C 候选建议兼容的 URI 格式化的 SOAP 请求; 请参阅 W3C WebSphere MQ Axis 2 客户机的 SOAP over JMS URI 。 Axis2 客户机根据 SOAP 请求中的 URI 发送使用 W3C 或 WebSphere MQ Transport for SOAP 格式化的 SOAP 请求。

7.0.1.3 修订包中引入了针对 W3C 推荐的 Axis2 客户机支持。 未提供对其他客户机的支持以及对 WebSphere MQ 提供的 SOAP 侦听器的支持。

相关概念

WebSphere Transport for SOAP on .NET Framework 1， .NET 2 和 Axis 1.4 的实现
您可能想要编写自己的 WebSphere MQ SOAP 发送方和侦听器。 使用 WebSphere MQ Transport for SOAP on .NET Framework 1， .NET Framework 2 和 Axis 1.4 的实现作为指南。

WebSphere MQ Transport for SOAP 和 Web Service 可靠消息传递
Web Service 可靠消息传递是一种协议，用于通过不稳定的连接以稳定方式交换 Web Service 请求和响应。最适合解决短暂的连接中断问题。

WebSphere Transport for SOAP on .NET Framework 1， .NET 2 和 *Axis 1.4* 的实现

您可能想要编写自己的 WebSphere MQ SOAP 发送方和侦听器。 使用 WebSphere MQ Transport for SOAP on .NET Framework 1， .NET Framework 2 和 Axis 1.4 的实现作为指南。

11 在 W3C 规范参考中查找 *URI Scheme for JMS*，以获取最新草案。

1. 客户机程序通过与将用于 HTTP 传输相同的方式使用相应 Web Service 框架。 它还必须注册 `jms:` 前缀。 前缀是使用 `com.ibm.mq.soap.Register.extension()` Java 方法或 `IBM.WMQSOAP.Register.Extension()` CLR 方法注册的。
2. Axis 1.4 或 .NET Framework 1 或 2 框架将调用编组为 SOAP 请求消息，与 SOAP/HTTP 完全相同。
3. WebSphere MQ 服务由前缀为 `jms:`的 URI 标识。 当框架标识 `jms:` URI 时，它将调用 WebSphere MQ 传输发送方代码 `com.ibm.mq.soap.transport.jms.WMQSender` (对于 Axis 1.4) 或 `IBM.WMQSOAP.MQWebRequest` (对于 .NET1 和 2)。 如果框架遇到前缀为 `http:` 的 URI，那么会调用标准 SOAP over HTTP 发送方。
4. SOAP 消息由 WebSphere MQ SOAP 发送方使用请求队列进行传输。 **SimpleJavaListener** (对于 Java) 或 **amqwSOAPNETListener** (对于 .NET) 接收请求消息。

   WebSphere MQ SOAP 侦听器是独立进程，具有可定制线程数的多线程。
5. WebSphere MQ SOAP 侦听器读取入局 SOAP 请求，并将其传递到相应的 Web Service 基础结构。
6. Web Service 基础结构解析 SOAP 请求消息并调用服务。 该过程与到达 HTTP 传输的消息过程相同。
7. 基础结构将响应格式化为 SOAP 响应消息，并将其返回到 WebSphere MQ SOAP 侦听器。
8. 侦听器将消息放在响应队列上，并将消息传输到 WebSphere MQ SOAP 发送方。 发送方将消息传递到客户机 Web Service 基础结构。
9. 客户机基础结构解析响应 SOAP 消息，并将结果返回给客户机应用程序。

每个应用程序上下文由单独的 WebSphere MQ 请求队列提供服务。

通过确保 WebSphere MQ SOAP 侦听器和服务在相应目录中执行，可以在 Axis 1.4 中控制应用程序上下文。 Axis 1.4 为目录设置正确的 CLASSPATH。

应用程序上下文在 .NET 中由 WebSphere MQ SOAP 侦听器控制，该侦听器在通过调用 `ApplicationHost.CreateApplicationHost` 创建的上下文中执行服务。 此调用指定目标执行目录。 然后，每个服务在部署目录中运行。

amqwdeployWMQService 生成请求队列和响应队列。 同时还会生成用于处理队列并将服务部署到 Axis 1.4 所需的基础结构。

相关概念

SOAP 与 WebSphere MQ 的集成

WebSphere MQ Transport for SOAP 和 Web Service 可靠消息传递

Web Service 可靠消息传递是一种协议，用于通过不稳定的连接以稳定方式交换 Web Service 请求和响应。 最适合解决短暂的连接中断问题。

WebSphere MQ Transport for SOAP 和 *Web Service* 可靠消息传递

Web Service 可靠消息传递是一种协议，用于通过不稳定的连接以稳定方式交换 Web Service 请求和响应。 最适合解决短暂的连接中断问题。

WebSphere MQ for SOAP 利用 WebSphere MQ 管理的可靠网络来传递 SOAP 消息。 HTTP 和 FTP 之类的传输为非受管。 非受管网络非常适用于不可预测的连接，这种网络内管理连接的困难和代价要超过不丢失请求和响应所带有的好处。

为了克服在非受管网络中发生连接中断时松开文件的问题，诸如受管 FTP 之类的服务会在 FTP 的基础上构建一个管理层。 管理层将负责检查文件是否已从用户那里成功传输，必要时会重新传输丢失的文件。 要使用受管 FTP，必须在连接两端安装管理软件。

Web Service 可靠消息传递采用其他方法解决连接不稳定的问题。 它的目标是可靠地传输 Web Service 请求和响应，而连接两端无需使用相同的软件。 通过实现 Web Service 可靠消息传递协议，任何软件都可以与另一软件可靠地交换消息。

连接失败时，发送方和接收方必须使用生成的 URI 作为键以保留 WSRM 消息传输的上下文。 发送方和接收方将继续尝试建立新连接。 如果新连接成功建立，那么传输完成。 WSRM 规范没有指定如何保留上下文，也没有指定何时尝试建立新连接。

您可能会决定只关注短暂的中断。 对于长时间中断，您可能要准备废弃一段时间后仍无法重新启动的传输。 同样，如果客户机或服务故障，您可能也会准备废弃传输。 让用户负责保证传输，减少管理客户机和服务的协调性方面的要求。

如果网络中断长时间存在（超过 30 分钟左右），或者如果客户机或服务器故障，某些连接不会重新建立的可能性会更大。 您不能再依靠 WSRM 在非受管方式下自动恢复消息传输。 必须考虑管理失败的 WSRM 连接，即开发软件来管理客户机和服务的网络。

使用 WSRM 解决短暂中断可以大大减少处理移动网络上的丢失消息。 如果您不必确保消息传送，那么减少消息丢失所带来的好处可以抵消开发 WSRM 实现所需的额外成本。

SOAP over JMS 提供有保证的消息传递，并处理客户机，服务器和网络的持续时间较长的中断。 如果您正在寻求比 HTTP 更可靠的 SOAP 服务质量，那么您选择哪种解决方案: WebSphere MQ Transport for SOAP 或 WSRM? 答案取决于多种因素。 下面列出了一些需要考虑的因素：

1. 不稳定性是否是因为连接失败。
2. 连接失败的持续时间长短。
3. 您是否可以管理连接的客户机端和服务器端。
4. 用户或管理员是否最终负责消息传送。

相关概念

SOAP 与 WebSphere MQ 的集成

WebSphere Transport for SOAP on .NET Framework 1， .NET 2 和 Axis 1.4 的实现
您可能想要编写自己的 WebSphere MQ SOAP 发送方和侦听器。 使用 WebSphere MQ Transport for SOAP on .NET Framework 1， .NET Framework 2 和 Axis 1.4 的实现作为指南。

安装和验证 WebSphere MQ Web Service

使用这些主题中的指示信息来安装和验证 WebSphere MQ Transport for SOAP。

安装 *WebSphere MQ Web Transport for SOAP*

使用以下指示信息来安装 WebSphere MQ Web Transport for SOAP。 安装将创建工具以使用 WebSphere MQ 作为 SOAP 传输来运行 Web Service 客户机或服务。 这些工具在 .NET Framework 1， .NET 2， Axis 1.4 或 Axis2 SOAP 环境中使用。

开始之前

在 IBM WebSphere MQ 的系统需求 中查看必备产品。 安装流程不会检查必备软件是否存在和是否可用。 您必须确认必备软件已安装。

WebSphere MQ 提供了 Axis 1.4 运行时的副本。 将此版本与 WebSphere MQ 配合使用，而不是与您可能已安装的任何其他版本配合使用。 IBM 不会为 Apache Axis 提供技术支持。 如果您有 Apache Axis 方面的技术问题，请联系 Apache 软件基金会。

要在 .NET Framework 3 SOAP 环境中运行 Web Service， WebSphere MQ 使用 Windows Communication Foundation。 用于 Windows Communication Foundation 的 WebSphere MQ 定制通道使用 WebSphere MQ 作为 SOAP 消息的传输来运行 Web Service 客户机和服务。

关于此任务

您可以将 WebSphere MQ Web Transport for SOAP 作为 WebSphere MQ MQI 客户机或服务器应用程序进行安装。 如果已将 WebSphere MQ 作为客户机或服务器安装在计算机上，请检查是否已安装列出的组件。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

执行以下安装步骤。

过程

1. 选择要安装的 “Java 和 .Net Messaging and Web Service” 组件。
2. 在 Solaris 和 HP-UX 上，选择要安装的 “Java 运行时环境” 组件。

3. 为安装选择开发工具包。
4. 安装并验证 WebSphere MQ ， 如您的平台的 "快速入门" 中所述。
5. 从 WebSphere MQ 安装介质上的 `prereqs/axis` 目录复制 Apache Axis 1.4 运行时 `axis.jar` 。 将其复制到 第 800 页的表 138、第 800 页的表 139 或第 800 页的表 140 中所述的安装目录。

   Windows

   ```
   Copy D:\PreReqs\axis\axis.jar MQ_INSTALLATION_PATH\java\lib\soap
   ```

   AIX

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

   HP-UX， Solaris 和 Linux (所有平台) 安装目录

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

6. 在 Windows 2003 上，运行 **Aspnet_regiis.exe** 以更新脚本映射，从而指向您正在使用的公共语言运行时版本。

   在 %SystemRoot%Microsoft.NET/Framework/*version-number* 中查找 **Aspnet_regiis.exe** 实用程序。
7. 设置环境变量 WMQSOAP_HOME 以指向 WebSphere MQ 安装目录。

结果

表 *138: Windows* 安装目录

位置	内容
MQ_INSTALLATION_PATH\programs\bin	二进制、命令、DLL 和可执行文件
MQ_INSTALLATION_PATH\programs\java\lib	.jar files
MQ_INSTALLATION_PATH\programs\java\lib\soap	SOAP .jar files
MQ_INSTALLATION_PATH\programs\soap\samples	样本和 IVT

表 *139: AIX* 安装目录

位置	内容
MQ_INSTALLATION_PATH/bin	shell 脚本
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar 和其他 JAX-RPC .jar 文件
MQ_INSTALLATION_PATH/samp/soap	样本和 IVT

表 *140: HP-UX， Solaris* 和 *Linux (*所有平台*)* 安装目录

位置	内容
MQ_INSTALLATION_PATH/bin	shell 脚本
MQ_INSTALLATION_PATH/java/lib	.jar files

表 140: HP-UX， Solaris 和 Linux (所有平台) 安装目录 (继续)	
位置	**内容**
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar 和其他 JAX-RPC .jar 文件
MQ_INSTALLATION_PATH/samp/soap	样本和 IVT

下一步做什么

1. 仅适用于 .NET，您必须向全局组合件高速缓存注册 WebSphere MQ Transport for SOAP 文件。 如果在安装 WebSphere MQ 时已安装 .NET，那么将在安装时自动执行注册。 如果在 WebSphere MQ 之后安装 .NET，那么首次运行 IVT 时将自动执行注册。

   您可以运行 **amqiregisterdotnet.cmd** 以执行 .NET 组合件的注册。 您还可以在任何阶段运行 **amqiregisterdotnet.cmd** 以强制执行重新注册。 一旦执行此操作，系统重新启动后重新注册仍存在，并且通常不再需要后续重新注册。
2. 按照第 801 页的『验证 IBM WebSphere MQ Transport for SOAP』中所述运行安装验证测试。
3. 如果您打算开发 Axis2 客户机，那么必须从 Apache 下载 Axis2 1.4.1；请参阅第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』。

验证 *IBM WebSphere MQ Transport for SOAP*

使用 **runivt** 命令验证 IBM WebSphere MQ Transport for SOAP。 该命令可运行大量演示应用程序并确保安装后环境设置正确无误。

开始之前

在运行 **runivt** 命令之前，请确保您具有以下运行时环境：

- 要仅在 Axis 上运行: 您必须在系统上具有可用的 Java SDK (在 SOE 中)。 还必须在系统 **PATH** 环境变量中包括 java.exe 和 javac.exe 命令的位置。
- 要仅在 .NET 上运行测试 (仅在 Windows 上受支持): 您必须在系统上同时具有 Java SDK 以及 .NET 编译器和工具。 要执行此操作，请访问 Visual Studio 命令提示符或 Microsoft Windows SDK 命令提示符，然后将 java.exe 和 javac.exe 文件的位置添加到 **PATH** 环境变量。
- 要运行所有可用的测试: 对于 Windows 平台，必须按 .NET 测试运行中所述配置环境。 在 UNIX and Linux 平台上，必须按照“仅在 Axis 上运行测试”中所述配置环境。

关于此任务

您可能希望仅在 Axis 上或仅在 .NET 上运行验证测试，而不是同时在 .NET 和 Axis 上运行验证测试。

如果测试时遇到问题并希望重新开始，请执行下列操作：

1. 使用 immediate 选项停止队列管理器 WMQSOAP.DEMO.QM。
2. 停止其他窗口中已启动的侦听器。
3. 删除队列管理器。
4. 删除您创建的临时 samples 目录，然后重新开始。

在 UNIX and Linux 平台上，必须使用 X Windows 系统会话来运行该命令。

runivt 命令将更改 soap/samples 目录的内容。 为了使安装映像保持不变，请将 samples 目录复制到临时位置，然后从临时位置运行验证测试。

可以任意多次运行安装验证。

执行以下步骤以验证 IBM WebSphere MQ Transport for SOAP on .NET Framework 1， .NET Framework 2 和 Axis 1.4:

过程

1. 将 ./tools/soap/samples 目录树复制到临时位置。

2. 启动命令窗口并将临时目录用作当前目录。
3. 使用 **runivt** 命令开始安装测试。 runivt 脚本在部署和运行测试类、示例客户机和服务之前先要对其进行编译。 对于要运行的测试类，样本客户机和服务，请完成 安装 WebSphere(r) MQ Web Transport for SOAP 中概述的安装步骤，并确保用于运行 runivt 命令的命令提示符具有必需的运行时环境集。 使用以下任一方法运行 **runivt** 命令：
   - 仅在 Axis 上运行测试： runivt Axis。
   - 仅在 .NET 上运行测试 (仅在 Windows 上受支持): runivt DotNet。
   - 运行所有可用的测试:runivt。

   有关 runivt 命令语法和参数的更多信息，请参阅 **runivt**：IBM WebSphere MQ Transport for SOAP 安装验证测试。 可以运行的测试在 Windows 上的文件 ivttests.txt 和 UNIX and Linux 平台上的 ivttests_unix.txt 中列出。

相关参考

runivt: WebSphere MQ Transport for SOAP 安装验证测试

为 WebSphere MQ Transport for SOAP 开发 Web Service

使用常规 Web Service 开发环境来开发用于 WebSphere MQ Transport for SOAP 的服务。

开始之前

1. 如果计划使用 WebSphere MQ Transport for SOAP 随附的命令行工具:
   a. 为服务创建部署目录。
   b. 在此目录中启动命令窗口。
   c. 对于 .NET， csc.exe 和 wsdl.exe 必须位于路径中，并且必须来自同一版本的 .NET Framework。
   d. 对于 Java ,
      i) 运行 **amqwsetcp** 命令以设置类路径。
      ii) 处于相同版本级别的 IBM JRE 和 JDK 必须位于当前路径中。 版本级别必须至少为 5.0。
      iii) 定制类路径以包括任何其他 .jar 库的位置和包含 .java 包的目录（包括您正在开发的服务）。 在类路径中输入当前目录 “.”。
      iv) 创建一个与您正在开发的服务的软件包名称对应的目录（相对于命令窗口的当前目录）。
2. 或者，使用支持 Web Service 开发的工作台工具。 示例开发任务使用 Microsoft Visual Studio 2008 , Eclipse IDE for Java EE Developers 和 WebSphere Application Server Community Edition。

关于此任务

现有 Web Service 无需修改即可使用 WebSphere Transport for SOAP。 随 WebSphere MQ Transport for SOAP 提供的工具将部署 Web Service ，并使用 WebSphere MQ SOAP 侦听器来运行该服务。 这些工具还会生成 WSDL ， .NET 客户机存根和 .java 代理类，以便为 SOAP 客户机开发 WebSphere MQ 传输。

遵循以下步骤创建服务，并为客户机部署和生成做准备。 执行相关任务中的步骤以使用 Eclipse 或 Microsoft Visual Studio 2008 创建服务。

过程

1. 使用标准开发环境开发服务。
2. 使用 HTTP Web service 客户机测试服务
3. 遵循以下步骤准备部署目录:
   - 对于 Java
     a. 将定义了服务接口的 .java 文件复制到部署目录。
     b. 将服务的所有 .class 文件复制到与软件包名称对应的目录中。

c. 检查类路径是否可以找到所需的所有类: 使用 **javac** 编译服务 `.java` 文件。

- 对于 .NET

  a. 将定义了服务的 `.asmx` 文件复制到部署目录。

  b. 如果您使用的是代码隐藏模型，请将所有 `.dll` 文件复制到 *deployment directory*\bin 目录。

使用 *Eclipse* 为 *WebSphere MQ Transport for SOAP* 开发 *JAX-RPC* 服务

开发 Axis 1.4 Web Service 以使用 WebSphere MQ 作为服务提供者来运行。 使用常规 Web Service 开发环境来创建用于部署到 Axis 1.4 的服务。

开始之前

请考虑将 Web 服务器部署到 Axis 1.4 的 WebSphere MQ SOAP 侦听器的需求。

- WebSphere MQ Axis 的 SOAP 侦听器 1.4 需要版本为 5.0 或更高版本的 IBM JRE。 用于开发的 JRE 和 JDK 必须处于同一版本级别。
- WebSphere MQ SOAP 侦听器 for Axis 1.4 需要随 WebSphere MQ 一起安装的 `axis.jar`。 更改开发环境中的构建路径，以引用随 WebSphere MQ 一起安装的 `axis.jar` 文件，而不是随开发环境一起安装的 `axis.jar` 文件。
- 直到并包括 WebSphere MQ V7.0.1，为已部署服务生成的 WSDL 都是 RPC/编码的。 从 V7.1 开始，您还可以请求 RPC/文字样式 WSDL。 生成的 WSDL 仅用于部署。 您可以使用符合 WS-I 的 WSDL 来定义服务。

关于此任务

使用常规 Web Service 开发环境创建服务。

在本任务中，我们使用免费的开放式源代码 Eclipse Java EE IDE for Web Developers (称为 Galileo)。 对于应用程序服务器，我们使用基于 Geronimo 的 WebSphere Application Server Community Edition v2.1 (Community Edition)。 有关如何获取，安装和配置 IDE 和服务器的信息，请参阅相关任务。

使用先前在 IDE 中提供的 Web Service 资源管理器将 HTTP 作为传输来测试服务。 或者，生成 HTTP 客户机代理并使用您自己的客户机代码来测试服务。

您可以遵循以下步骤来开发自底向上的 Web Service。 使用样本程序 `StockQuoteAxis.java` 作为示例。

过程

1. 使用新工作空间启动 Eclipse IDE for Java EE Developers。
2. 配置工作空间以使用 Java50，
   WebSphere Application Server Community Edition 2.1.4 不适用于 Java60。
   a) **窗口** > **首选项** > **Java** > **已安装的 JRE** > **添加 ...** > **标准 VM** > **下一步** > **目录 ...**
   b) 浏览至 **Java50** > **确定** > **完成** 的安装目录
   c) 检查 **Java50** JRE> 确定
3. 添加 Community Edition 运行时环境并启动 Community Edition。
   a) **窗口** > **首选项** > **服务器** > **运行时环境** > **添加 ...**
   b) 从 " **新建服务器运行时环境** " 列表中选择 **IBM WASCE v2.1** > 选中 **创建新的本地服务器** > **下一步**
   如果 **IBM WASCE 2.1** 不在列表中，那么需要完成另外两个任务:
      i) 安装 WebSphere Application Server Community Edition。
      ii) 安装 Community Edition 的 Eclipse 更新。

      在 WebSphere Application Server Community Edition 上查找详细信息
   c) 浏览至 Application Server 安装目录> **确定** > **完成** > **确定**。
   d) 在 "服务器" 视图中右键单击 **IBM WASCE v2.1 服务器** > **启动**

提示: 您可以在 Eclipse 中管理 WASCE: 右键单击 **IBM WASCE v2.1 服务器** > **启动 WASCE 控制台** 。 缺省值 **Username** 和 **Password** 为 system 和 manager 。

4. 设置 Web Service 的服务器和运行时。
   a) **窗口** > **首选项** > **Web Service** > **服务器和运行时**
   b) 选择 **IBM WASCE v2.1 Server** 作为服务器。
   c) 将 **Apache Axis** 保留为 Web Service 运行时。
5. 创建动态 Web 项目。
   a) **文件** > **新建** > **动态 Web 项目**。
   将项目命名为 StockQuoteAxis。
   b) 选中 **将项目添加到 EAR** > **新建 ...**
   c) 在 " **EAR 应用程序项目** " 页面中，输入 **Project name** StockQuoteAxisEAR > **完成**
   回复 **确定** 以响应建议您切换到 Java EE 透视图的对话框，或者保留 Java 透视图以准确遵循这些指示信息。
   d) 选择 **IBM WASCE 2.1 服务器** 作为目标运行时。 接受它，其他缺省值> **完成**。
   回复 **确定** 以响应建议您切换到 Java EE 透视图的对话框，或者保留 Java 透视图以准确遵循这些指示信息。
6. 导入 StockQuoteAxis.java 样本程序
   a) 打开 **StockQuoteAxis** Web 项目> 右键单击 **src** 文件夹> **导入 ...**
   b) 选择 **常规** > **文件系统** > **下一步**
   c) 浏览至 *MQ_INSTALLATION_PATH*\tools\soap\samples\java\server > 检查 **StockQuoteAxis.java** > **完成**
   MQ_INSTALLATION_PATH 表示安装了 WebSphere MQ 的高级目录。 您必须突出显示服务器目录以查看其包含的文件。
7. 通过将 StockQuoteAxis.java 移动到其正确的包中来更正编译错误。
   a) 打开 StockQuoteAxis.java 并右键单击问题> **快速修订**
   b) 双击 **移动 "StockQuoteAxis.java" 以打包 "soap.server"** > **保存**。
8. 从 StockQuoteAxis.java 创建 Web Service。
   a) 右键单击 StockQuoteAxis.java > **Web Service** > **创建 Web Service** > **下一步**。
   接受服务的缺省配置:

   Web Service 类型
   自底向上 Java beanWeb 服务

   服务实现
   soap.server.StockQuoteAxis

   服务器
   IBM WASCE v2.1 server

   Web Service 运行时
   Apache Axis

   服务项目
   StockQuote 轴

   服务 EAR 项目
   StockQuoteAxisEAR

   配置
   无客户机生成
9. 选择要访问的方法和 Web Service 的样式> **下一步**。
   如果出现提示，请启动服务器。
   a) 使所有方法保持选中状态。
   b) 选择文档/文字 (换行) 样式。

10. **完成**

    部署服务后，请查看 StockQuoteAxis Web 项目中的 WebContent\wsdl 文件夹，并查找生成的 StockQuoteAxis.wsdl 文件。

11. 将 HTTP 与 Web Service 资源管理器配合使用来测试服务。

    a) 右键单击 StockQuoteAxis.wsdl > **使用 Web Service 资源管理器进行测试**。

    b) 单击 " **Web Service 资源管理器** " 窗口中的 **StockQuoteAxisSoap 绑定** 操作中的操作 **getQuote** 。

    c) 在 **符号** 输入字段> **Go** 中输入 ibm

12. 使用 WebSphere MQ Transport for SOAP 测试服务。

    要部署服务，请使用 com.ibm.mq.soap.jar 中的 **SimpleJavaListener**。 您必须将 WebSphere MQ Java 和 SOAP 库添加到构建路径。

    a) 右键单击 **StockQuoteAxis** Web 项目> **构建路径** > **配置构建路径 ...**

    b) 单击 **库** 选项卡> **添加外部 JAR ...**。 浏览至 *MQ_INSTALLATION_PATH*\java\lib 。 并选择所有 .jar 文件> **Open** > **Add External Jars ...**。 浏览至 *WMQ Install directory*\java\lib\soap 并选择所有 .jar 文件> **打开** > **确定**。

    MQ_INSTALLATION_PATH 表示安装了 WebSphere MQ 的高级目录。

    c) 在 Project Explorer 中，右键单击 StockQuoteAxis\Java Resources\Libraries\com.ibm.mq.soap.jar\com.ibm.mq.soap.transport.jms\SimpleJavaListener.class\ SimpleJavaListener > **运行方式 ...> 运行配置 ...**

    提示:

    如果没有 SimpleJavaListener 的配置，请单击 " **运行配置** " 向导的 **创建，管理和运行配置** 页面上的 **新建配置** 图标。

    SimpleJavaListener 没有用于将其停止的命令。 要监视或停止 **SimpleJavaListener**，请在 Eclipse 中打开 **调试透视图** 。

    d) 打开 **(x) = Arguments** 选项卡。 在 **程序参数** 输入区域中，将参数输入到 **SimpleJavaListener**。

    对于此示例，请输入

    ```
    -u "jms:/queue?destination=REQUESTAXIS@QM1&connectionFactory=()
    &initialContextFactory=com.ibm.mq.jms.Nojndi
    &targetService=StockQuoteAxis
    &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" -n 10
    ```

    注: 目标服务为 StockQuoteAxis ， 以匹配在服务部署描述符 StockQuoteAxis\WebContent\WEB-INF\server-config.wsdd 中创建的目标服务名称。 amqwdeployWMQService 创建名为 soap.server.StockQuoteAxis 的目标服务。 在此示例中，您将使用与 HTTP 服务器相同的 StockQuoteAxis.class 和 service-config.wsdd 。

    e) 在同一选项卡上，配置 **工作目录** 以引用 server-config.wsdd 文件: ${workspace_loc:StockQuoteAxis/WebContent/WEB-INF}

    a) **运行**

    错误将写入控制台。 如果控制台保留为空，那么 **SimpleJavaListener** 已启动正常。

    b) 要测试部署，请运行在任务 第 814 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-RPC 客户机』中开发的 StockQuoteAxis 客户机。

示例: StockQuoteAxis 样本程序

样本 Java Web Service StockQuoteAxis.java 安装在 *WMQ install directory*\tools\soap\samples\java\server 中。 StockQuoteAxis.java 或 第 806 页的图 170 有四种方法:

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)

4. float getQuoteTran(String symbol)

```
package soap.server;
import java.lang.Thread;
import java.io.FileWriter;
public class StockQuoteAxis {
    public float getQuote(String symbol) throws Exception {
        return ((float) 55.25);
    }
    public void getQuoteOneWay(String symbol) throws Exception {
        try {
            // Write the results for this service to a file
            FileWriter f = new FileWriter("getQuoteOneWay.txt", true);
            f.write("One way service result via proxy is: 44.44\n");
            f.close();
        } catch (Exception ee) {
            System.out.println("Error writing result file in getQuoteOneWay");
            ee.printStackTrace();
        }
    }
    public int asyncQuote(int delay) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            System.out.println("Exception in asyncQuote during sleep");
        }
        return delay;
    }

    public float getQuoteTran(String symbol) throws Exception {
        if (symbol.equalsIgnoreCase("ROLLBACK")) {
            System.out.println("Rollback was requested,
                exiting from service by calling System.exit().");
            System.exit(0);
        }
        return ((float) 55.25);
    }
}
```

图 *170: StockQuote* 轴

下一步做什么

使用 WebSphere MQ Transport for SOAP (而不是使用命令 **amqwdeployWMQService** 的 HTTP) 来部署服务。

该命令有一个选项 axisDeploy，用于通过创建 Apache Axis 1.4 部署描述符来部署服务。 WebSphere MQ SOAP 侦听器运行服务。 SOAP 侦听器称为 SimpleJava 侦听器，随 WebSphere MQ Transport for SOAP 提供。

相关任务

使用 Microsoft Visual Studio 2008 为 WebSphere MQ Transport for SOAP 开发 .NET 1 或 2 服务
使用 Microsoft Visual Studio 2008 开发 SampleStockQuote Web Service for .NET 1 或 .NET 2

为 W3C SOAP over JMS 开发 JAX-WS EJB Web Service
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 JEE 应用程序服务器的 EJB 容器中运行。此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 2。

使用 *Microsoft Visual Studio 2008* 为 *WebSphere MQ Transport for SOAP* 开发 *.NET 1* 或 *2* 服务

使用 Microsoft Visual Studio 2008 开发 SampleStockQuote Web Service for .NET 1 或 .NET 2

关于此任务

使用 Visual Studio 2008 通过代码隐藏实现来创建 StockQuote 服务。

过程

1. 为服务创建模板，然后检查在 HTTP 上的运行情况。
   a) 启动 Visual Studio 2008 > **文件** > **新建** > **项目...**。 选择 **C#** 项目类型， ， **NET Framework 2** 和 **ASP.NET Web Service 应用程序**。 依次输入**名称：**和**解决方案名称：**StockQuoteDotNet > **确定**
   b) 右键单击**“解决方案资源管理器”**中的 **Service1.asmx** > **重命名** >StockQuote.asmx。
   c) 将代码片段 public class Service1 更改为 public class StockQuote。
   d) 右键单击**“解决方案资源管理器”**中的 **StockQuote.asmx** > **打开方式...** > **XML 编辑器**。 将 Class="StockQuoteDotNet.Service1" 更改为 Class="StockQuoteDotNet.StockQuote"
   e) 将代码片段 [WebService(Namespace = "http://tempuri.org/")] 更改为 [WebService(Namespace = "http://stock.samples/")]。
   f) 移除代码行 [ToolboxItem(false)]。
   g) 检查目前为止的所有内容是否正确：**调试** > **启动调试 (F5)**。 验证资源管理器中的输出。
2. 从样本 SQDNNonInline.asmx.cs 添加方法，并在 HTTP 上测试此服务。
   a) 打开 *MQ_INSTALLATION_PATH* \tools\soap\samples\dotnet\SQDNNonInline.asmx.cs 并使用四个 Quote 方法替换 HelloWorld 方法；请参阅第 808 页的图 171。 *MQ_INSTALLATION_PATH* 表示 WebSphere MQ 的安装目录。
   b) **构建** > **重新构建**解决方案 > 右键单击其中一个有错的**线程** > **解决** > 使用 **System.Threading**。
   c) 按 F5 以启动调试。
   该服务不符合 WS-I Basic Profile v1.1。 您可以选择将 WebMethod 注释从 [SoapRpcMethod] 更改为 [SoapDocumentMethod]，或移除注释 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]。
   d) 按 F5 以使用 HTTP 验证实现。
3. 使用 WebSphere MQ Transport for SOAP 生成 WSDL ， 客户机和运行服务。
   a) 在存储 StockQuote.asmx 的项目目录树中打开命令窗口。
   b) （可选）使用 amqwdeployWMQService 生成工件。 必须启动队列管理器：

```
amqswdeployWMQService -f StockQuote.asmx
-u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   所有工件都创建在 ./generated 目录树中。
   c) (可选) 仅生成用于使用 WebSphere MQ Transport for SOAP 调用服务的 WSDL。

```
amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   a) 运行 .NET 侦听器。 使用 .\generated\server\startWMQNListener.cmd 或输入以下命令：

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```

4. 使用从 WSDL 生成的客户机或使用 **amqwdeployWMQService** 生成的客户机测试服务。

样本代码

样本 .NET Web Service StockQuoteDotNet 安装在 *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet 中。 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。 已发布样本的 Web Service 绑定与该任务中使用的绑定稍有不同。 该任务使用 Visual Studio 2008 中使用的缺省值。

有两个 .NET Framework 1 和 .NET Framework 2 Web Service 示例。 StockQuoteDotNet.asmx 是内联服务。 SQDNNoninline.asmx 是由 SQDNNoninline.asmx.cs 实现的代码后盾 Web Service。

StockQuoteDotNet 有四个方法:

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)
4. float getQuoteDOC(String symbol)

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
     [WebMethod] [ SoapRpcMethod(OneWay=true) ]
     public void getQuoteOneWay(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
          System.Console.WriteLine("getQuoteOneWay was invoked.");
     }
     [WebMethod] [SoapRpcMethod]
     public float getQuote(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
          return 88.88F;
     }
     [WebMethod] [SoapRpcMethod]
     public int asyncQuote(int delay) {
          Thread.Sleep(delay);
          return delay;
     }
     [WebMethod]
     public float getQuoteDOC(String symbol) {
          return 77.77F;
     }
}
```

图 *171:* 内嵌服务：*StockQuoteDotNet.asmx*

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

图 *172:* 代码隐藏：设计 *SQDNNonInline.asmx*

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }

    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

图 *173:* 代码隐藏：实现：*SQDNNonInline.asmx.cs*

相关任务

使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务
开发 Axis 1.4 Web Service 以使用 WebSphere MQ 作为服务提供者来运行。 使用常规 Web Service 开发环境来创建用于部署到 Axis 1.4 的服务。

为 W3C SOAP over JMS 开发 JAX-WS EJB Web Service
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 JEE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 2。

为 *W3C SOAP over JMS* 开发 *JAX-WS EJB Web Service*

绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 JEE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 2。

开始之前

使用 Rational Application Developer 来创建 EJB Web Service。 Rational Application Developer 中的 Web Service 向导可以选择使用 SOAP over JMS 绑定的 W3C 候选建议来创建 Web Service。 Rational Application Developer 7.54 是必需的。 此练习使用了 Rational Software Architect for WebSphere Software v7.5.5.1 中包含的 Rational Application Developer 。

作为此任务的一部分， EJB 将从 Rational Application Developer 部署到 WebSphere Application Server。 您必须完成 第 843 页的『配置 WebSphere Application Server 以使用 W3C SOAP over JMS』

要创建任务中实际使用的 WSDL，必须首先完成任务 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』。 然后，可以从 Eclipse Galileo 工作空间中的动态 Web 项目导入 WSDL，也可以从部署到 WASCE 的正在运行的 HTTP Web Service 导入 WSDL。

由于执行 第 845 页的『配置 WebSphere Application Server 资源』， WebSphere Application Server 可能仍在运行。 如果没有运行，您可以在 RAD 的服务器视图中启动此服务器。

关于此任务

在此任务中，您将 StockQuoteAxis 服务从作为 **SimpleJavaListener** 使用 WebSphere MQ Transport for SOAP 运行的 JAX-RPC Axis 服务运行，重新部署为使用 W3C SOAP over JMS 协议在 WebSphere Application Server 中运行的 JAX-WS 服务。

将服务从 **SimpleJavaListener** 迁移到 WebSphere Application Server 有两个部分:

1. 使用 Rational Application Developer 中的自顶向下 EJB Web Service 向导从服务的 WSDL 生成 Web Service 接口。
2. 通过导入 WebSphere MQ SOAP 样本 StockQuoteAxis.java 来实现服务。

替代方法是从 StockQuoteAxis.java 自下往上生成服务。 但是，为了确保迁移后的服务的接口完全相同，最好使用自顶向下方法，因为此方法使用相同的 WSDL。

Web Service 是针对 EJB 容器而不是 Web 容器开发的，因为 JMS 支持是 EJB 容器的一部分。

过程

1. 启动 Rational Application Developer，并验证 WebSphere Application Server 是否正在运行。
   a) 在新工作空间中启动 Rational Application Developer 。
   b) 打开 Java EE 透视图。
   c) 打开 **服务器** 选项卡，然后检查 WebSphere Application Server 是否正在运行。
   - 如果视图中没有 WebSphere Application Server v7.0 ，请在视图> **新建** > **服务器**中右键单击。 遵循向导中的选项来创建 WebSphere Application Server v7.0 实例。
   - 如果服务器存在但未启动，请单击箭头以启动服务器。
   - 要验证属性并快速访问服务器日志，请右键单击 **localhost 上的 WebSphere Application Server v7.0** > **属性** > **WebSphere Application Server**。
   - 要管理服务器，请使用外部浏览器并打开 URL http://localhost:9061/ibm/console/unsecureLogon.jsp 或右键单击 **localhost 上的 WebSphere Application Server v7.0** > **运行管理控制台**。
   - 缺省设置为自动发布。 但很多人选择手动对服务器部署更新。 双击 **localhost 上的 WebSphere Application Server v7.0**，然后在**概述** 窗口中展开 **发布** 折叠标记。 单击**永不自动发布**。
   - 另一个您可能想更改的缺省值是清除**“概述”**窗口中的**在工作台关闭时终止服务器**复选框。
2. 创建 JEE 项目
   必须创建企业应用程序项目 (EAR) 和企业 Java Bean (EJB) 项目。
   a) **文件** > **新建** > **企业应用程序项目**。 将项目命名为 W3CJMSEAR > **完成**。
   缺省值必须将 WebSphere Application Server v7.0 标识为目标运行时和 EAR 版本 5.0。 必须选择缺省配置。
   b) **文件** > **新建** > **EJB 项目**。 将项目命名为 W3CJMSEJB。 选择 W3CEARJMS 作为 **EAR 项目名称** > **下一步**。
   缺省 EJB 模块版本为 3.0，将再次使用缺省配置。
   c) 取消选中 **创建 EJB 客户机 JAR 模块** 复选框> **完成**。
3. 从 StockQuoteAxis WSDL 生成和部署 EJB Web service。
   a) **运行** > **启动 Web Services 浏览器**。
   b) 使用 "**Web Service 资源管理器** " 窗口中的图标选择 WSDL 页面> 单击 Navigator 中的 **WSDL main** 。

c) 在“**操作**”窗口中，输入或浏览到 StockQuoteAxis.wsdl 的 WSDL URL。

如果您已使用部署为 HTTP 服务的 StockQuoteAxis 运行 WASCE，那么 URL 为:

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

如果您的文件系统中有此 WSDL，那么该 URL 可能为:

```
File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl
```

d) 在导航树中单击包含已导入 URL 的行。

即 **WSDL Main** 后紧跟的行，如果这是您导入 Web Services 浏览器的第一个 WSDL。

e) 在“**操作**”窗口中单击**启动 Web Service 向导** > **Web Service 框架** > **前往**

f) 在 Web Service 向导中，选择**自顶向下 EJB Web Service**

使用第 811 页的表 141 中的信息选择或验证配置 选中**在无警告的情况下覆盖文件** > **下一步**。

表 *141:* 自顶向下 *EJB Web Service* 配置

字段	**值**
服务器	WebSphere Application Server v7.0
Web Service 运行时	IBM WebSphere JAX-WS
服务项目	W3CJMSEJB
服务 EAR 项目	W3CJMSEAR
配置:	No client generation

g) 在标题为 **指定用于创建 WebSphere JAX-WS EJB 自顶向下 Web Service 的选项**的页面上，选中 **切换到 JMS 绑定**框。 另请选中 **启用包装器样式**， **将 WSDL 复制到项目** 和 **生成 Web Service 部署描述符** > **下一步**。

h) 在标题为 " **WebSphere JAX-WS JMS 绑定配置**" 的页面上，选中 **使用 SOAP/JMS 互操作性协议** 并提供 第 811 页的表 142 中的值，将其他字段留空> **下一步**。

表 *142: WebSphere JAX-WS JMS* 绑定配置

字段	**值**
JMS 目标	queue
目标 JNDI 名称:	requestaxis
JMS 连接工厂	qm1
应答名称	W3CJMSEAR
配置:	replyaxis

a) 在标题为“**WebSphere JAX-WS 路由器项目配置**”的页面上，在 **ActivationSpec JNDI 名称**字段中输入 qm1as > **下一步**。

RAD 大约需要 30 秒到 1 分钟来生成和部署项目。

b) 忽略 " **Web Service 发布** " 页面中的选项> **完成**。

4. 检查生成的 WSDL。

您要求生成特定于服务的 WSDL 并保存在项目中。

a) 在“企业资源管理器”导航器中，打开文件夹 **W3CJMSEJB** > **ejbmodule** > **META-INF** > **wsdl**。 双击 StockQuoteAxis.wsdl 以在 WSDL 编辑器中打开。

检查绑定; 您将看到 JMS URL:

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qm1&targetService=StockQuoteAxis
```

5. 可选步骤：使用 JAX-WS 将 EJB 绑定到 SOAP over HTTP。

   向 EJB 提供两个绑定可以让客户机选择 SOAP 绑定来调用 Web service。 还为客户机提供了一种使用 HTTP 查询 Web 服务器来获取其 WSDL 的方法。

   将 EJB 绑定到 SOAP over HTTP 的步骤不属于此任务的一部分。
6. 使用样本 StockQuoteAxis.java 实现和重新部署 StockQuoteAxis
   a) 在 "企业资源管理器" 导航器中，打开文件夹 **W3CJMSEJB** > **服务** 双击 StockQuoteAxisService 以在 Java 编辑器中打开实现类。
   b) 打开 *WebSphere MQ Installation directory*\tools\soap\samples\java\server 文件夹中的 StockQuoteAxis.java 样本程序> 选择所有方法，但不选择类名> **复制**。
   c) 在 StockQuoteAxisSoapBindingImpl.java 中，选择所有方法，但不是选择类名，并粘贴到 StockQuoteAxis.java 中的方法内。
   d) 调用服务时，将 print 语句添加到输出到 WebSphere Application Server 控制台。
   更改 getQuote(String symbol) 方法：

```
public float getQuote(String symbol) {
    System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
                       + symbol);
    return ((float) 55.25);
```

   e) 纠正导入：**源** > **组织导入** > **保存**。
   f) 更正由于实施与接口不匹配而产生的三个错误。

      这些错误是因为 StockQuoteAxis.java 中的三个方法抛出异常，该服务的 WSDL 不包含任何故障消息。 该问题被诊断为方法特征符与方法 Web Service 注释不匹配。

      使用 @WebFault 注释方法并重新生成 WSDL，或者使接口保持不变并移除异常。

      要使接口保持相同，请从方法特征符中移除这三个 throws exception > **保存**。

下一步做什么

第 853 页的『使用 W3C SOAP over JMS 部署到 Axis2 客户机』

相关任务

使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务
开发 Axis 1.4 Web Service 以使用 WebSphere MQ 作为服务提供者来运行。 使用常规 Web Service 开发环境来创建用于部署到 Axis 1.4 的服务。

使用 Microsoft Visual Studio 2008 为 WebSphere MQ Transport for SOAP 开发 .NET 1 或 2 服务
使用 Microsoft Visual Studio 2008 开发 SampleStockQuote Web Service for .NET 1 或 .NET 2

为 WebSphere MQ Transport for SOAP 开发 WebSphere MQ Web Service 客户机

使用常规开发环境来开发 Web Service 客户机，以便与 WebSphere MQ Transport for SOAP 配合使用。

开始之前

创建服务。 您可以使用 第 802 页的『为 WebSphere MQ Transport for SOAP 开发 Web Service』中的一个示例。

选择如何开发、部署和使用客户机，以及从哪里获取 WSDL 以生成客户机。

决定您为 WebSphere MQ Transport for SOAP 开发客户机和服务的方法。

有两种方法。

1. 使用标准开发工具，开发 HTTP 服务和客户机，然后使用 WebSphere MQ Transport for SOAP 的 URL。
2. 使用 WebSphere MQ Transport for SOAP 随附的工具和样本。

如果采用 HTTP 路由，那么可以在 HTTP 服务器上运行该服务，也可以使用 WebSphere MQ Transport for SOAP 来运行该服务。 要使用 WebSphere MQ Transport for SOAP 运行该服务，请为 SOAP 配置相应的 WebSphere MQ 侦听器，并设置路径和部署描述符以运行该服务。 WebSphere MQ Transport for SOAP 提供的工具将为您执行配置。 或者，您可以对环境进行配置以运行这些侦听器。

WebSphere MQ Transport for SOAP 随附的工具在入门和学习如何部署传输时很有用。 对于生产工作，使用标准工具并部署相同的服务以供不同的 SOAP 传输访问有很多好处。

决定要开发的客户机类型

您必须决定要开发的 Web Service 客户机类型。 该选择取决于您是否知道服务接口和服务地址。

如果接口已知，请使用 Axis 或 .NET 工具从服务接口生成代理客户机类。 代理客户机类能让编写客户机以调用此服务更加简单。 如果开发客户机时知道服务的位置，请使用静态代理接口。 如果服务的位置发生更改，例如，如果服务已重新部署到生产服务器，请使用动态代理接口。

如果开发客户机时不知道服务接口，在 Axis 上，您可以为 Axis 1.4 创建动态调用接口 (DII) 客户机。 DII 客户机使用类属接口调用所有服务。 要正确地将参数传递到特定服务，您需要以编程方式构建特定的服务接口。 在客户机中以编程方式构建接口，或通过将服务的 WSDL 装入客户机来构建接口。 在 Axis2 上，您可以创建 Dispatch client。 Dispatch client 使用文档模型描述客户机请求，而 DII 客户机使用调用模型。 这两种客户机都是动态构建请求。

获取服务的 WSDL

除了以编程方式构建服务接口的情况之外，您必须先获取服务 WSDL 才能创建 Web Service 客户机。 服务 WSDL 可从以下三个不同的源中获取：

1. 使用 **java2wsdl** (Axis) 或 **disco** (.NET) 之类的工具直接从 Web Service 实现。
2. 通过使用以下 URL 查询 Web Service: *Web service http url*?wsdl。
3. 来自文件系统上的文件，或者来自 UDDI 或 WebSphere Service Registry and Repository 之类的注册表。

注: 如果此服务不能使用 HTTP 访问，那么 WSDL 查询不起作用。 只能使用 WebSphere MQ Transport for SOAP 来提供服务本身。

amqwdeployWMQService 生成的 WSDL 与使用 **java2wsdl** 或 **disco** 生成的 WSDL 不同。 生成的 WSDL 与您已开始创建的“自顶向下”服务的任何 WSDL 也不同。 在 Axis 上，`server-config.wsdd` 部署描述符可将客户机生成的 SOAP 消息映射到操作和服务。 **amqwdeployWMQService** 可从 Eclipse 中生成不同的部署描述符。

用于构建客户机的 WSDL 取决于服务的部署方式：

使用 amqwdeployWMQService 部署
使用 **amqwdeployWMQService** 生成的 WSDL。 指定 -w 标志并选择 `rpcLiteral` WSDL。 出于兼容性考虑，您可以选择 `rpcEncoded` WSDL。 `rpcEncoded` WSDL 仅适用于 .NET 和 Axis 1.4 客户机。

使用 SimpleJavaListener 手动部署
请使用以下 WSDL 文件之一：

1. 用于定义服务的 WSDL，或存储库中存储的 WSDL。
2. **java2wsdl** 从服务生成的 WSDL。
3. 使用 URL *Web service http url* ?wsdl 查询的 WSDL (如果可从 HTTP 服务器获取)。 您可以运行 Web Services 浏览器之类的工具，将服务定义直接导入 Eclipse。

您可能需要更改服务的 URI。 将其从 HTTP 服务的地址更改为 WebSphere MQ Transport for SOAP 的 URI。

使用 amqSOAPNETListener 手动部署。
请使用以下 WSDL 文件之一：

1. 用于定义服务的 WSDL，或存储库中存储的 WSDL。
2. 从 .NET 服务类 (.asmx) 获取的 WSDL。 使用 **disco**。

3. 使用 URL *Web service http url* ?wsdl 查询的 WSDL (如果可用)。 您可以运行 Web Services 浏览器之类的工具，将服务定义直接导入 Eclipse。
4. 通过对 .NET 服务类 (.asmx) 运行 **amqswsdl** 获取的 WSDL。

您可能需要更改服务的 URI。 将其从 HTTP 服务的地址更改为 WebSphere MQ Transport for SOAP 的 URI。

已部署到 Windows Communication Foundation
使用 URL *Web service http url*?wsdl 获取服务 WSDL。 必须在服务定义中使用 serviceMetaData 行为配置定义该服务。

部署到其他服务器平台。
遵循平台随附的有关如何获取正确的服务 WSDL 的指导信息。

关于此任务

使用标准开发工具开发客户机。 以下任务说明如何为 .NET 1 和 2， Axis 1.4 (JAX-RPC) 和 Axis2 (JAX-WS) 构建客户机。 对于 Windows Communication Foundation，请参阅相关任务链接。

使用 *Eclipse* 为 *WebSphere Transport for SOAP* 开发 *JAX-RPC* 客户机

开发 Axis 1.4 Web Service 客户机以使用 WebSphere MQ Transport for SOAP 运行。

开始之前

您必须有可用的服务。 如果要将任务作为实际练习来执行，请使用您在任务 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』中创建的工作空间和服务。 Eclipse 中需要正在运行支持 Axis 1.4 Web Service 的应用程序服务器。 在本任务中，我们使用免费提供的 WebSphere Application Server Community Edition V 2.1.4。 它配置为任务 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』的一部分。 您也可以使用 Tomcat，这是一种较小的开放式源代码应用程序服务器。

关于此任务

该任务显示使用在 Windows 上运行的 Eclipse 为样本 StockQuoteAxis 服务开发三种类型的客户机。 这三种客户机为使用客户机代理开发的静态和动态客户机以及 DII 客户机。

下面展示了两种从 WSDL 生成客户机代理的替代方法:

1. 正在使用 **amqwdeployWMQService** 生成客户机代理。
2. 将 WSDL 导入 Eclipse，然后使用 Web service 向导生成客户机代理。

过程

1. 针对 Java EE 开发者启动 Eclipse IDE 。
2. 创建名为 StockQuoteAxisClient 的 Java 项目:
   a) 切换到 Java 透视图> **文件** > **新建** > **Java 项目**。 在 **"创建 Java 项目" 页面** 类型的 **Project name** 字段中， StockQuoteAxisEclipseClient。 确保执行环境为 **J2SE1-1.4** 或 **J2SE-1.5** > **Next** 。
   b) 在“**Java 设置**”页面上，选择**库**选项卡 > **添加外部 JAR...**
   c) 浏览至 *MQ_INSTALLATION_PATH*/java/lib，然后选择所有 .jar 文件> **打开**。
   MQ_INSTALLATION_PATH 是 WebSphere MQ 的安装目录。
   d) 浏览至 *MQ_INSTALLATION_PATH*/java/lib/soap，然后选择所有 .jar 文件> **打开**。 您必须已将 axis.jar 从 WebSphere MQ 安装介质安装到此目录中。
   MQ_INSTALLATION_PATH 是 WebSphere MQ 的安装目录。
   e) **库** 选项卡现在引用构建客户机所需的所有 .jar 文件> **完成**。
3. 使用以下两种方法中的一种在 Eclipse 中为样本 StockQuoteAxis Web service 创建代理:
   - 使用 **amqwdeployWMQService** 生成客户机代理。
     a. 创建队列管理器。 对于此任务，请创建 QM1 作为缺省队列管理器。

b. 创建工作目录 samples。 将 StockQuoteAxis.java 样本程序复制到 samples/soap/server 中。

c. 修改 *MQ_INSTALLATION_PATH*/bin 中的 amqwsetcp.cmd 以在类路径中包含当前目录。 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。

d. 在 samples 中打开命令窗口并运行修改后的 **amqwsetcp** 命令

e. 通过运行以下命令为 StockQuoteAxis 服务创建 WSDL：

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWsdl
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

切记: 使用 Java 命令时，请使用 "/"，而不是 "." 或 "\" 。

提示: 您可以从 .samples/generated 导入生成的 WSDL，而不是将生成的代理导入 Eclipse。 生成的代理在以下两方面有所不同：

i) 软件包名称不同，您可以重构名称。

ii) Eclipse 生成的代理包含其他帮助程序类 StockQuoteAxisProxy.java

f. 通过运行以下命令为 StockQuoteAxis 服务创建客户机代理：

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

g. 将客户机代理导入 StockQuoteAxisClient：

i) 右键单击 **StockQuoteAxisClient\src** > 选择**文件系统** > **下一步** > **浏览...** > 查找文件夹 .\samples\generated\client\remote\soap\server > **确定**。

ii) 在 "**导入** " 页面> **完成**中检查 **服务器** 。

h. 将软件包名称重构为 soap.server。

i) 右键单击包含客户机代理的软件包 > **重构** > **重命名**。 输入 **New name**: soap.server > 保留其他选项的所选缺省值> **确定** 。 将修复所有错误。

- 使用 Eclipse 生成客户机代理。

  您可以选择获取服务的 WSDL 的方式。 在此示例中，服务已部署到 WebSphere Application Server Community Edition ，您可以从 Web 服务器获取 WSDL。 在任务 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』中描述了部署。

  a. 在 Eclipse 中，切换到 Web 透视图，并检查 WebSphere Application Server Community Edition v2.1 Server 是否正在运行以及 StockQuoteAxis 是否已部署并同步。

  b. 将 WSDL 导入 Web Services 浏览器：

  i) 单击操作栏中的 **Web Services 浏览器**图标，或单击**运行** > **启动 Web Services 浏览器**。

  ii) 单击 Web Services 浏览器中的 WSDL 页面图标以切换到 WSDL 页面。

  iii) 单击 Web Services 浏览器的导航器窗口中的 **WSDL Main**。

  iv) 输入 Web Service 的 URL ，后跟 ?WSDL 。 任务 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』中部署的 StockQuoteAxis 的 URL 为：

  ```
  http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
  ```

  c. 生成客户机代理：

  i) 在 "Web Service 资源管理器" 导航器中，单击 **http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl** 。

  ii) 在**"操作"**窗口中，单击**启动 Web Service 向导** > 保留选择 **Web Service 客户机** > **前往**。

iii) 在向导首页上，单击配置中的**客户机**项目链接 > 选择 **StockQuoteAxisClient** 客户机项目 > **确定**。

提示: 向导窗口可能会丢失焦点。 您需要手动将其带回焦点。

iv) Web service 运行时必须为 Apache Axis 才能生成 JAX-RPC 客户机。

v) 单击**完成**。

vi) 更改服务的静态 URL 以指向 StockQuoteAxis 服务的 WebSphere MQ Transport for SOAP 地址。 在使用 HTTP 服务器完成对客户机的测试之前，您可以选择跳过此步骤。

a) 打开 `StockQuoteAxisServiceLocator.java` 并找到 `StockQuoteAxis_address` 的声明。

b) 将 URL 更改为

```
"jms:/queue?destination=REQUESTAXIS
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
&amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

提示: Eclipse 可自动将 & 转换为 `&`，并在您将字符串复制粘贴到 `.java` 代码中时反向转换。

d. 创建三个 Java 客户机类，每个 Java 客户机类都具有主方法:

i) 创建软件包。 右键单击 **StockQuoteAxisClient/src** > **新建软件包**。 将其命名为 `soap.client` > **完成**。

ii) 选择 **soap.client** > **新建** > **类**。 将类命名为 `SQAStaticClient` > 选中 **public static void main(string [] args)** > 完成

iii) 重复此过程以创建 `SQADynamicClient.java` 和 `SQADIIClient.java`

e. 编写客户机代码。

第 819 页的图 177 到第 821 页的图 181 提供了这三种客户机代码的示例。 这些示例使用 HTTP URL 通过部署到 HTTP 服务器的 StockQuoteAxis 服务测试客户机。 要针对使用 WebSphere MQ Transport for SOAP 部署的 StockQuoteAxis 服务运行客户机，请将 URL 更改为:

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.Nojndi
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

– 第 819 页的图 177 和第 820 页的图 179 使用 Eclipse 生成的代理，该代理具有额外的 StockQuoteAxisproxy 帮助程序类，能够让编码稍微简单一些。

– 第 820 页的图 178 和 第 820 页的图 180 使用 **amqwdeployWMQService** 生成的代理。

– 第 821 页的图 181 不使用代理类。

每个客户机都调用 `com.ibm.mq.soap.Register.extension()` 以链接到 WebSphere MQ Transport for SOAP。 该扩展注册在客户机部署描述符中。第 848 页的『将 Web Service 客户机部署到 Axis 1.4 以使用 IBM WebSphere MQ Transport for SOAP』 中描述了到 Axis 1.4 的客户机部署。

f. 通过向工作空间中配置的 WebSphere Application Server Community Edition 服务器托管的 StockQuoteAxis 发送 SOAP 请求来运行客户机。

i) 检查服务器是否在运行以及是否已部署和同步 StockQuoteAxis。

ii) 选择或打开您要测试的客户机 > 单击操作栏中的**运行**。 或者，单击绿色运行图标或在导航器中右键单击客户机 > **运行方式** > **运行配置...**。 配置运行客户机所需的参数。

g. 使用 WebSphere MQ Transport for SOAP 运行客户机。

此过程使用 **amqwdeployWMQService** 来部署服务，并且仅使用使用由 **amqwdeployWMQService** 构建的 WSDL 或代理的客户机。 要运行使用原始 WSDL 的客户机，或运行使用 Eclipse 构建的代理的客户机，请使用 Eclipse 构建的服务部署描述符部署此服务。 使用服务端口绑定名称作为 `targetService` **名称** 手动启动 **SimpleJavaListener** 。

i) 遵循 第 838 页的『使用 amqwdeployWMQService 将服务部署到 Axis 1.4 以用于 WebSphere Transport for SOAP』 中的指示信息将服务部署到 WebSphere MQ 简单 Java SOAP 侦听器。该服务部署仅对使用 WSDL 的客户机或使用 **amqwdeployWMQService** 构建的客户机代理有效。

ii) 在命令窗口中，运行 **amqwclientconfig** 以创建客户机部署描述符文件 `client-deploy.wsdd` 。

iii) 将 `client-deploy.wsdd` 导入到要使用 WebSphere MQ Transport for SOAP 测试的 Java 项目的根目录中。

a) 右键单击 Java 项目 **StockQuoteAxisEclipseClient** > **导入** > **文件系统** > **下一步** > **浏览 ...**

b) 浏览至包含 `client-deploy.wsdd` > **打开** > 在 " **导入** " 向导页面中选择目录> 在右侧窗格中选中 `client-deploy.wsdd` 的目录。

c) 验证**目标文件夹：** 已输入 `StockQuoteAxisEclipseClient` > **完成**。

iv) 确认用于在此项目中运行 Java 应用程序的工作目录是 `StockQuoteAxisEclipseClient` 目录:

右键单击 Java 项目 **StockQuoteAxisEclipseClient** > **运行方式 ....** > **运行配置 ...** > 选择 **(x) = 参数** 选项卡> 验证是否选中了 "工作目录" 中的 **缺省值** 单选按钮，并且路径为 `StockQuoteAxisEclipseClient` 。 或者选择下面其中一项以选择其他位置或包含客户机配置的文件:

- 选中**其他：** > 输入您选择的目录路径。
- 在 " **VM 参数** " 窗口中，输入 `-Daxis.ClientConfigFile=`*`full path to client deployment descriptor file`*

v) 确保 URL 配置为指向使用 WebSphere MQ Transport for SOAP 部署的服务。 按照步骤 ii 中所述运行客户机。

提示: 通常，您可能会遇到以下错误之一:

i) `Exception: No client transport named 'jms' found!.`

ii) JMS 连接错误。

iii) `Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java`

iv) `Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis`

说明:

i) 找不到 `client-config.wsdd` ，或者在 `client-config.wsdd` 中包含行 `<transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/>` 。

ii) 可能是构建路径问题-不包含 *`MQ_INSTALLATION_PATH`*`/java/lib` 中的 .jar 文件。*`MQ_INSTALLATION_PATH`* 是 WebSphere MQ 的安装目录。

iii) 使用 `server-config-wsdd` 或传递到 **SimpleSoapListener** 的参数时发生服务部署问题。

iv) 部署描述符和服务实现不匹配。

如果您在 Eclipse 中运行客户机时遇到问题，请尝试使用命令窗口:

i) 切换到工作空间目录树中的 `StockQuoteAxisEclipseClient\bin` 目录。

ii) 运行 **amqwsetcp** 和 **amqwclientconfig**

iii) 运行 `java soap/client/SQAStaticClient`。

样本 JAX-RPC Web service 客户机

WebSphere MQ 随附的样本 Java Web Service 客户机安装在 *`MQ_INSTALLATION_PATH`*`\tools\soap\samples\java\clients` 中。 *`MQ_INSTALLATION_PATH`* 是 WebSphere MQ 的安装目录。

SQAxis2Axis.java
第 818 页的图 174 中的 SQAxis2Axis.java 是用于调用 StockQuoteAxis 服务的动态代理客户机。 您可以通过在命令行中提供 URL 来覆盖此服务的 URL（编译成动态代理）。

SQAxis2DotNet.java
第 818 页的图 175 中的 SQAxis2DotNet.java 是用于调用 StockQuoteDotNet 服务的动态代理客户机。 您可以通过在命令行中提供 URL 来覆盖此服务的 URL（编译成动态代理）。

WsdlClient.java
WsdlClient.java 第 819 页的图 176 是动态调用客户机，用于调用 StockQuoteDotNet 或 StockQuoteAxis 服务。 缺省情况下，客户机调用 StockQuoteAxis 服务。 添加命令行选项 -D 调用 StockQuoteDotNet 服务和 -w 以提供与 .\generated\StockQuoteDotNet_Wmq.wsdl 中的端口不同的端口

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
   public static void main(String[] args) {
      com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
         StockQuoteAxis service = null;
         if (args.length == 0)
            service = locator.getSoapServerStockQuoteAxis_Wmq();
         else
            service = locator.getSoapServerStockQuoteAxis_Wmq(
                new java.net.URL(args[0]));
         System.out.println("Response: " + service.getQuote("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
```

图 *174: SQAxis2Axis.java*

```
public class SQAxis2DotNet {
public static void main(String[] args) {
   com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteDotNet locator = new StockQuoteDotNetLocator();
         StockQuoteDotNetSoap_PortType service = null;
         if (args.length == 0)
            service = locator.getStockQuoteDotNetSoap();
         else
            service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                  args[0]));
         System.out.println("Response: " + service.getQuoteDOC("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
}
```

图 *175: SQAxis2DotNet.java*

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdlClient {
public static void main(String[] args) {
      String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
      try {
         Register.extension();
         Options opts = new Options(args);
         if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
         } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
         }
         if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
         System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
               + " resolving uri to ...");
         QName servQN = new QName(namespace, wsdlService);
         QName portQN = new QName(namespace, wsdlPort);
         Service service = ServiceFactory.newInstance().createService(
               new URL(wsdlSource), servQN);
         Call call = (Call) service.createCall(portQN, "getQuote");
         wsdlTargetURI = call.getTargetEndpointAddress().toString();
         System.out.println("  '" + wsdlTargetURI + "'");
         Object ret = call.invoke(new Object[] { "XXX" });
         System.out.println("Response: " + ret);
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
         e.printStackTrace();
         System.exit(2);
      }
   }
}
```

图 *176: WsdlClient.java*

此任务中使用的示例客户机:

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
      System.out.println("Static client synchronous result is:"
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

图 *177:* 使用 *Eclipse* 生成的代理的静态客户机

```
package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
      System.out.println("Static client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

图 *178:* 使用 *amqwdeployWMQService* 生成的代理的静态客户机

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

图 *179:* 使用 *Eclipse* 生成的代理的动态客户机

```
package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL sqaURL = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq(sqaURL);
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

图 *180:* 使用 *amqwdeployWMQService* 生成的代理的动态客户机

```
package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQADIIClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL wsdl = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
      Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
          new QName("http://server.soap", "StockQuoteAxisService"));
      Call SQACall = SQAService.createCall(new QName("http://server.soap",
          "StockQuoteAxis"), "getQuote");
      System.out.println("DII client synchronous result is "
          + SQACall.invoke(new Object[] { "ibm" }));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

图 *181: DII* 客户机（无代理）

相关任务

使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机
开发要使用 WebSphere MQ Transport for SOAP 运行的 Axis2 Web Service 客户机。 列出了随 WebSphere MQ Transport for SOAP 提供的样本 Axis2 客户机，并列出了用于生成代理的 **wsimport** 命令。

使用 Microsoft Visual Studio 2008 为 WebSphere Transport for SOAP 开发 .NET 1 或 2 客户机
开发 .NET 1 或 2 Web Service 客户机以使用 WebSphere MQ Transport for SOAP 运行。

使用 *Eclipse* 为 *WebSphere Transport for SOAP* 开发 *JAX-WS* 客户机

开发要使用 WebSphere MQ Transport for SOAP 运行的 Axis2 Web Service 客户机。 列出了随 WebSphere MQ Transport for SOAP 提供的样本 Axis2 客户机，并列出了用于生成代理的 **wsimport** 命令。

开始之前

获取 Axis2 库，然后配置开发和测试环境以运行客户机。

注: Axis 使用的版本和发行版命名会产生混淆。 通常，Axis 1.4 指的是 JAX-RPC 实现，Axis2 指的是 JAX-WS 实现。

Axis 1.4 是版本级别。 如果在因特网上搜索 Axis 1.4，那么您将转至 http://ws.apache.org/axis/。 页面包含 Axis 先前版本（1.2 和 1.3）以及 2006 年 4 月 22 日最终发行版 Axis 1.4 的列表。 Axis 1.4 存在修订了错误的更新发行版，但它们全都称为 Axis 1.4。 这是 WebSphere MQ 随附的其中一个错误修订发行版。 对于轴 1.4，请使用 WebSphere MQ 随附的 `axis.jar` 版本，而不是可从 http://ws.apache.org/axis/获取的版本。

Axis Web 站点也用 Axis 1.1 来指代通常称为 Axis 1.4 的所有版本。 Axis 1.2 用于指代通常称为 Axis2 的版本。

Axis 1.5 不是 Axis 1.4 的更新发行版，而是 Axis2 发行版。 如果搜索 Axis 1.5，那么您将转至 http://ws.apache.org/axis2/。https://ws.apache.org/axis2/download.cgi 包含 Axis2 发行版的列表，标记为 0.9 到 1.5.1（并包含令人混淆的 V1.4）。 要与 WebSphere MQ Transport for SOAP 配合使用的 Axis2 发行版为 1.4.1。 从 http://ws.apache.org/axis2/download/1_4_1/download.cgi 下载 Axis2 1.4.1。

您可以选择使用 **wsimport** 或 IDE 随附的工具为 WebSphere MQ Transport for SOAP 的 Web Service 客户机生成代理。 Eclipse IDE for Java EE Developer 3.5 SR1 使用 **wsdl2java**。**wsimport** 随 Java 6 一起提供。 您可以使用 Java 5 来运行使用 **wsimport** 或 **wsdl2java** 生成的客户机代理。

随 WebSphere MQ Transport for SOAP 提供的样本 Web Service Axis2 客户机是使用 **wsimport** 开发的; 请参阅 第 826 页的『样本 Axis2 客户机』。

以下任务演示了如何生成和使用随 Eclipse IDE for Java EE Developers 打包的 Web Service 向导生成的代理。 样本客户机展示了如何使用 **wsimport** 生成的代理。

要使用 Web service 向导，您必须在工作台中添加支持 Axis2 的应用程序服务器。 这些步骤说明如何使用工作台配置 WASCE 以支持 Axis2 。

1. 配置 Eclipse IDE for Java EE Developers 中使用的应用程序服务器以支持 Axis2。 在此示例中，配置 WASCE 2.1.4 应用程序服务器，这是在 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』中创建的工作空间的一部分。
   a. 打开工作空间首选项以对服务器进行配置：打开**窗口** > **首选项**。
   b. 检查所安装的 JRE 是否为 `Java50`：单击**已安装的 JRE**。
   c. 添加 WASCE 作为服务器: 单击 **服务器** > **运行时环境** > **添加 ...** > **IBM** > **WASCE v2.1** > 下一步。 JRE 必须是 Java50 > 浏览到 WASCE 安装目录> **确定** > **完成**。 您必须已安装用于 Eclipse Java EE IDE for Web Developers 的 WASCE 插件。
   d. 添加 Axis2：单击 **Web Service** > **Axis2 首选项**。 在 **Axis2 运行时** 选项卡上> **浏览 ...** 打开包含多个 `Axis2` jar 文件> **Apply** 的目录。
   e. 将 WASCE 与 Axis2: 单击 **Web Service** > **服务器和运行时**。 在 **`Server`** select **IBM WASCE v2.1 Server** 下，在 **`Web service runtime`** 下，选择 **Apache Axis2** > **Apply** > **OK**
   f. 启动服务器：打开 Web 透视图，然后打开服务器视图。 在“服务器”视图中右键单击 > **新建** > **服务器**。 选择并配置了 **IBM WASCE v2.1 Server** > **完成**。 启动服务器。
2. 检查是否已将 StockQuoteAxis 服务部署到 WASCE 以运行 Web Service 向导。
3. 要使用 WebSphere MQ Transport for SOAP 服务测试服务，请将该服务部署到用于 Axis 1.4 的 SOAP 侦听器的 WebSphere MQ 传输; 请参阅 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』。

关于此任务

Eclipse IDE for Java EE 开发者使用 Java50 和 Web Service 向导来生成服务的代理类。 代理类与 Java 6 随附的 **`wsimport`** 工具创建的类不同。 另一种方法是使用 **`wsimport`** 生成代理类，并将其创建的包导入到 Eclipse Java EE IDE for Web Developers 中。

Eclipse IDE for Java EE Developers 中的 Web Service 向导在 Web 项目中构建 Web Service 客户机。 您可以将客户机作为简单 Java 应用程序运行; 它不需要应用程序服务器。 您还可以将代码传输到 Java 项目，并配置构建路径以包含 Axis2 JAR 文件。

过程

1. 在新企业项目中创建 Web 项目：
   a) Project Explorer 中不选择任何内容 > 右键单击空白处 > **新建** > **企业应用程序项目** > 命名为 `StockQuoteAxis2EAR` > **完成**。 将 No 回复到为您提供打开 Java EE 透视图的选项的窗口。
   缺省值设置为使用 WASCE。
   b) 右键单击 `StockQuoteAxis2EAR` > **新建** > **动态 Web 项目**。 将项目命名为 `StockQuoteAxis2WebClient` >选中 EAR 成员框以将此项目添加到 **StockQuoteAxis2EAR**。 选择 WASCE 2.1 作为目标运行时。
   c) 在 " **新建动态 Web 项目** " 页面的 "配置" 部分中> **修改 ...** > 检查 Axis2 Web Service 项目构面。 **已检查动态 Web 模块 2.5**， **Java 6.0** 和 **WASCE 部署 1.2** 。 > **确定** > **完成**。 将 No 回复到为您提供打开 Java EE 透视图的选项的窗口。
2. 将该服务的 WSDL 导入工作空间，然后生成客户机代理：
   在此示例中，WSDL 文档包含 HTTP 服务绑定并成为静态 Web 客户机代理的目标。 在生成客户机代理之前，可以修改 Web Service 绑定中的 URL 以指向 WebSphere MQ Transport for SOAP URL。 然后，静态 Web 客户机代理是部署到 WebSphere MQ Transport for SOAP 的服务。
   a) 启动 Web Services 浏览器：使用操作栏中的图标，或**运行** > **启动 Web Services 浏览器**。
   b) 通过单击“**Web Services 浏览器**”窗口中的 WSDL 图标选择 WSDL 资源管理器 > 在导航器窗口中单击 **WSDL Main** > 输入 StockQuoteAxis WSDL 文件的 URL >**前往**。
   在此示例中，直接从 HTTP 服务获取 WSDL：`http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`

c) 在导航器中，单击带有 Web service 的 URL 的行。 在**"操作"**窗口中，单击**将 WSDL 导入工作台** > 选择 **StockQuoteAxis2WebClient** 作为**工作台项目** > 输入 **WSDL 文件名** `StockQuoteAxisHTTP.wsdl` > **前往**。

d) 右键单击 **StockQuoteAxisHTTP.wsdl** > **Web Service** > **生成客户机**。 检查有关向导的 Web Service 页面的配置信息如下: 服务器: IBM WASCE v2.1 Server， Web Service 运行时: Apache Axis2，客户机项目: StockQuoteAxis2WebClient，客户机 EAR 项目: StockQuoteAxisEAR。 要更正配置，请单击错误行。

e) 单击**下一步** > 验证代码生成设置 > **完成**。

注意，将创建新的软件包 `soap.server`，其中包含您需要的代理。

3. 配置项目以将 WebSphere MQ Transport for SOAP 作为 JMS 传输运行。

WebSphere MQ Transport for SOAP 提供了 `transportSender`，但没有 `transportReceiver`。 换言之， WebSphere MQ Transport for SOAP 支持 Axis2 客户机。 目前，它不支持 Axis2 服务。

a) 在 **StockQuoteAxis2WebClient** 项目中，右键单击 `WebContent\WEB-INF\conf\axis2.xml` > **打开方式...** > **XML 编辑器**。

b) 搜索最后一个 `transportSender` (接近文件末尾)，并找到注释掉的 JMS `transportSender` > 右键单击行> **添加之前 ...** > **transportSender**。

c) 右键单击 **transportSender** > **添加属性** > **名称** > 右键单击 **transportSender** > **添加属性** > **类**。

d) 右键单击**名称** > **编辑属性** > 输入**值：** `jms`

e) 右键单击 **类** > **编辑属性** > 输入 **值:** `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender`。> 保存。

f) 将 `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender` 添加到构建路径: 右键单击 **StockQuoteAxis2WebClient** > **构建路径** > **配置构建路径 ...** > 单击 **库** 选项卡> **添加外部 JAR ...**。 选择 *MQ_INSTALLATION_PATH*`\java\lib` > **确定**中的所有 JAR。

MQ_INSTALLATION_PATH 是 WebSphere MQ 的安装目录。

4. 创建同步静态客户机，使用 HTTP 对其进行测试，然后使用 WebSphere MQ Transport for SOAP 转换代理以运行静态客户机。

a) 右键单击 **Java 资源 :src** > **新建** > **包** > 命名包 `soap.client` > 完成

b) 右键单击 **soap.client** > **新建** > **类** > 将类命名为 `SQA2StaticClient` > **完成**。

c) 将类替换为以下代码，然后单击 **保存**。

图 *182: SQA2DynamicClient.java*

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("Response is: "
          + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

5. 使用部署到 WASCE 的 StockQuoteAxis 服务以及 WebSphere MQ Transport for SOAP 测试客户机。

a) 在 Project Explorer 中，右键单击 **SQA2StaticClient** > **运行方式 ...** > **Java 应用程序**。

结果 `Response is 55.25` 将显示在 "控制台" 视图中。 您还可以在 "控制台" 视图中选择 WASCE 控制台窗口，并查看 WASCE 服务器 `StockQuoteAxis called with parameter: ibm` 上的输出。

b) 代理已使用服务地址 http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis 构建，因此静态客户机将调用在 HTTP 上运行的服务。 您可以更改静态客户机以使用 WebSphere MQ Transport for SOAP 来调用服务。 以下指示信息用于在不重建代理的情况下更改 StockQuoteAxisServiceStub.java 中的服务地址，以及配置 SQA2StaticClient 运行时参数来加载 axis2.xml。 配置 axis2.xml 配置 Axis2 以使用 WebSphere MQ Transport for SOAP。

c) 打开 StockQuoteAxisServiceStub.java > 使用以下代码替换两个 http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis：

```
jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

d) 如果现在运行 SQA2StaticClient ，那么它将抛出异常，因为找不到为 JMS 配置的 transportSender
异常为：

```
Exception: null java.lang.NullPointerException at
soap.server.StockQuoteAxisServiceStub.getQuote(StockQuoteAxisServiceStub.java:547)
at soap.client.SQA2StaticClient.main(SQA2StaticClient.java:11)
```

e) 在 Project Explorer 中，右键单击 **SQA2StaticClient** > **运行方式...** > **运行配置...**。 切换到 **(x) = Arguments** 选项卡，并在 **VM 自变量** 输入区域中，输入 axis2.conf 文件> **Apply** > **Run** 的路径。 VM 自变量是： -Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml。 或者，您可以提供 Axis2 配置文件的标准路径。

f) 再次运行 SQA2StaticClient。 在此运行中，您将使用 WebSphere MQ Transport for SOAP。 通过检查 WASCE 控制台中是否没有新输出来确认它。 打开与 SimpleJava 侦听器关联的控制台或命令窗口，其中的输出为 StockQuoteAxis called with parameter: ibm。

6. 为 HTTP 和 WebSphere MQ Transport for SOAP 创建动态客户机，并对其进行测试。

a) 右键单击 **soap.client** > **新建** > **类** > 将类命名为 SQA2DynamicClient > **完成**。

b) 将类替换为以下代码，然后单击 **保存**。

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
       System.out.println("JMS sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
     } catch (Exception e) {
       System.out.println("Exception: " + e.getMessage());
       e.printStackTrace();
     }
   }
 }
```

c) 为 SQA2DynamicClient.java 创建运行配置，并添加 axis2.xml 的路径：
-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml

d) 运行 SQA2DynamicClient。 检查 SQA2DynamicClient， WASCE 和 **SimpleJavaListener** 的控制台输出。

7. 创建一个异步客户机，然后在回调处理程序和主程序线程中访问结果。

Web Service 向导为 Eclipse Java EE IDE for Web Developers 创建的异步客户机代理与 **wsimport** 创建的代理不同。 **wsimport** 代理使用 Future、Response 和 AsyncHandler 通用类型。

Eclipse Java EE IDE for Web Developers 的 Web Service 向导将创建 StockQuoteAxisServiceCallbackHandler 抽象类。 您必须扩展 StockQuoteAxisServiceCallbackHandler 并创建回调处理程序。

a) 右键单击 **soap.client** > **新建** > **类** > 将类命名为 SQA2CallbackHandler > **完成**。

b) 将该类替换为以下代码。

```
package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
  private boolean complete = false;
  SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
  }
  public void receiveResultgetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
  }
  public boolean isComplete() {
    return complete;
  }
}
```

c) 右键单击 **soap.client** > **新建** > **类** > 将类命名为 SQA2AsyncClient > **完成**。

d) 将该类替换为以下代码。

图 *183: SQA2AsyncClient.java*

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      SQA2CallbackHandler callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      do {
        System.out.println("Waiting for HTTP callback");
        Thread.sleep(2000);
      } while (!callback.isComplete());
      System.out.println("HTTP poll: "
          + ((GetQuoteResponse) (callback.getClientData()))
                .getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
      System.out.println("JMS Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      while (!callback.isComplete()) {
        System.out.println("Waiting for JMS callback");
        Thread.sleep(2000);
      }
      System.out.println("JMS poll: "
        + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
    } catch (Exception e) {
```

```
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
      }
    }
  }
```

控制台输出如下所示:

```
HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
```

样本 Axis2 客户机

样本代理是使用随 Java 6 打包的 **wsimport** 工具生成的。 提供了六个样本:

1. DynamicProxyClientSync.java
2. DynamicProxyClientAsyncPolling.java
3. DynamicProxyClientAsyncCallback.java
4. DispatchClientSync.java
5. DispatchClientAsyncPolling.java
6. DispatchClientAsyncCallback.java

这些客户机样本是为样本 StockQuoteAxis 服务器生成的。 使用 **amqwdepoyWMQServer** 命令生成 WSDL, 指定 -w 开关以选择 rpcLiteral 样式。 使用以下命令为这些样本生成代理:

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

图 *184: DynamicProxyClientSync.java*

```
package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientSync");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      service.getQuoteOneWay("48");
      System.out.println(" > getQuoteOneWay has returned");

      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      float result = service.getQuote("48");
      System.out.println(" > getQuote has returned result of " + result);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
```

```
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

图 *185: DynamicProxyClientAsyncPolling.java*

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncPolling");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously by
polling...");
      Response<Float> response = service.getQuoteAsync("49");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** Retrieve the result **/
      try {
        Float result = response.get();
        System.out.println(" > getQuoteAsync call has returned result of " + result);
      }
      catch (CancellationException ce) {
        // processing was cancelled via response.cancel()
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
```

```
  }
}
```

图 *186: DynamicProxyClientAsyncCallback.java*

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncCallback");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = service.getQuoteAsync("50", handler);
      System.out.println(" > Invoke call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<Float> response) {
    try {
      Float result = response.get();
      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());
```

```
        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

图 *187: DispatchClientSync.java*

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientSync");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service **/
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /************************************
       * Create OneWay SOAPMessage request.
       ************************************/
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a OneWay SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");
```

```
      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      dispatch.invokeOneWay(request);
      System.out.println(" > getQuoteOneWay call has returned");

      /*****************************************
       * Create Request Reply SOAPMessage request.
       *****************************************/
      mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a Request Reply SOAP Message");
      request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      part = request.getSOAPPart();
      env = part.getEnvelope();
      header = env.getHeader();
      body = env.getBody();

      /** Construct the message payload **/
      operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
      value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      SOAPMessage ans = dispatch.invoke(request);
      System.out.println(" > getQuote call has returned");

      /** Retrieve the result **/
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      System.out.println("Parsing SOAP response...");
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

图 *188: DispatchClientAsyncPolling.java*

```
package com.ibm.mq.axis2.samples;
```

```
import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncPolling");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation asynchronously by
polling...");
      Response<SOAPMessage> response = dispatch.invokeAsync(request);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** retrieve the result **/
      SOAPMessage ans = response.get();
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");
```

```
      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");

    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

图 *189: DispatchClientAsyncCallback.java*

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncCallback");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");
```

```
      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload. **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint. **/
      DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

      System.out
          .println("Invoking getQuote Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = dispatch.invokeAsync(request, handler);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<SOAPMessage> response) {
    try {
      // retrieve the result
      SOAPMessage ans = response.get();
      SOAPPart part = ans.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPBody body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String result = responseElement.getValue();

      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
```

```
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

相关任务

使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-RPC 客户机
开发 Axis 1.4 Web Service 客户机以使用 WebSphere MQ Transport for SOAP 运行。

使用 Microsoft Visual Studio 2008 为 WebSphere Transport for SOAP 开发 .NET 1 或 2 客户机
开发 .NET 1 或 2 Web Service 客户机以使用 WebSphere MQ Transport for SOAP 运行。

使用 *Microsoft Visual Studio 2008* 为 *WebSphere Transport for SOAP* 开发 *.NET 1* 或 *2* 客户机

开发 .NET 1 或 2 Web Service 客户机以使用 WebSphere MQ Transport for SOAP 运行。

开始之前

可以通过多种不同方式启动 .NET 1 或 2 客户机的开发:

1. 使用 **amqwdeployWMQService** 从 Web Service 生成客户机存根，并将其导入 Visual Studio。
2. 使用 **java2wsdl** 从 Web Service 的 Java 实现生成 WSDL，然后使用 .NET 随附的 `wsdl.exe` 生成客户机存根。
3. 使用 **amqswsdl** 从服务的 .NET `.asmx` 实现生成 WSDL，然后使用 `wsdl.exe`。
4. 如果已开发并部署 HTTP 服务，请使用 **添加 Web 引用 ...** 用于配置客户机以访问 HTTP 服务的向导。 更改 URL 是指部署到 WebSphere MQ Transport for SOAP 的服务。

该任务使用 第 806 页的『使用 Microsoft Visual Studio 2008 为 WebSphere MQ Transport for SOAP 开发 .NET 1 或 2 服务』 中部署的服务。

关于此任务

执行以下步骤以创建 .NET 1 或 2 Client for HTTP 和 WebSphere MQ Transport for SOAP。

过程

1. 创建客户机控制台应用程序并将其修改为调用 StockQuote HTTP Web 服务。
   a) 在 " **解决方案资源管理器** > 添加 ...> 新建项目" 中右键单击 **解决方案 "StockQuoteDotNet"** 。 选择 **C#** 项目类型 。**NET Framework 2.0** 和 **控制台应用程序**。 将项目命名为 `StockQuoteClientDotNet` > **确定**
   b) 在 " **解决方案资源管理器** > 添加 ...> 新建项目" 中右键单击 **解决方案 "StockQuoteDotNet"** 。 选择 **C#** 项目类型 。**NET Framework 2.0** 和 **控制台应用程序**。 将项目命名为 `StockQuoteClientDotNet` > **确定**
   c) 右键单击 **StockQuoteClientDotNet** > **设置为 Startup 项目**。
   d) 右键单击 **StockQuoteClientDotNet** > **添加 Web 参考 ...** > 浏览到此解决方案中的 Web Service> 选择 **StockQuote** > **添加参考**。 请注意，您已将 Web 引用添加到本地主机和新配置文件 `app.config`。

e) 在解决方案资源管理器中，将控制台应用程序的名称从 Program.cs 更改为 StockQuoteClientDotNet.cs > 单击**确定**以将 Program.cs 的所有用法更改为 StockQuoteClientDotNet.cs。

f) 将 StockQuoteClientDotNet.cs 的内容替换为 第 835 页的图 190 中的代码。

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

图 *190: HTTP StockQuoteClientDotNet* 程序

g) 启动 StockQuoteClientDotNet 以针对 StockQuote.asmx 服务进行测试：

i) 按 **F5**，单击操作栏中的绿色箭头，或单击**调试 > 启动调试 (F5)**。

如果 StockQuoteDOtNet 项目位于同一解决方案内，此项目将自动启动。 否则，您需要先启动此服务。

带有结果的命令窗口将在工作空间后打开。 按 **Enter** 键之前，Console.ReadLine(); 语句会阻止此窗口关闭。

提示: 确保 StockQuote.asmx 为 StockQuoteDotNet 项目中的起始页。

2. 修改 StockQuoteClientDotNet 以使用 WebSphere MQ Transport for SOAP 调用 StockQuote.asmx 服务。

a) 将以粗体显示的行添加到客户机。

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        IBM.WMQSOAP.Register.Extension();
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
        stockobj.Url = "jms:/queue?"
          + "initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
          + "&targetService=StockQuote.asmx";
        Console.WriteLine("jms reply is: "
          + stockobj.getNonInlineQuote("jms request"));
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

图 *191:* 修改后的 *StockQuoteClientDotNet* 程序

或者，修改缺省 URL。 打开 **StockQuoteClientDotNet** > **Properties** > **Settings.settings** ， 并将 StockQuoteClientDotNet_localhost_StockQuote 属性的值更改为 WebSphere MQ Transport for SOAP URL。

b) 添加对 amqsoap.dll 的引用

i) 在 " **解决方案资源管理器**" 中的 **StockQuoteClientDotNet** 项目中，右键单击 **引用** > **添加引用 ...** > 单击 **浏览** 选项卡> 浏览至 *MQ_INSTALLATION_PATH*\bin > 选择 **amqsoap.dll** > **确定**。 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。

3. 使用 WebSphere MQ Transport for SOAP 测试客户机与 StockQuote.asmx 服务。

a) 在 StockQuoteDotNet 项目目录中打开命令窗口： .\StockQuoteDotNet\StockQuoteDotNet > 验证 .bin\StockQuoteDotNet.dll 是否存在。 如果不存在，请重建解决方案。

b) 输入命令 **amqwRegisterdotNet**。

每次安装只需要运行一次 **amqwRegisterdotNet**。

c) 如果已使用 genAsmxWMQBits 运行 **amqwdeployWMQServer** ， 请运行 .NET SOAP 侦听器: generated\server\startWMQNListener

d) 或者直接运行侦听器:

```
amqwSOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. 在 Visual Studio 2008 中， 按 **F5** 来运行 StockQuoteClientDotNet。

.NET Framework 1 和 .NET Framework 2 Web Service 客户机

随 WebSphere MQ Transport for SOAP 提供的样本 .NET 客户机使用生成的存根来调用样本 Axis 和 .NET 服务。

对于 .NET Framework 1 和 .NET Framework 2 客户机， WebSphere MQ 提供对使用 .NET 客户机的 Web Service 的访问。 **amqwdeployWMQService** 命令有一个选项 genProxiestoDotNet，用于为 Web Service 生成 .NET Framework 1 或 .NET Framework 2 客户机存根。 您还可以使用 .NET **wsdl** 工具或 Microsoft Visual Studio 2005 或 2008 生成的客户机存根。

样本 .NET Framework 1 和 .NET Web Service 客户机安装在 *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet 中。 *MQ_INSTALLATION_PATH* 是 WebSphere MQ 的安装目录。

SQVB2Axis.vb
SQVB2Axis.vb 或 第 837 页的图 192 是用于调用 **StockQuoteAxisService** 服务的 Visual Basic 客户机。

SQVB2DotNet.vb
QVB2DotNet.vb 或 第 837 页的图 193 是用于调用 **StockQuoteDotNet** 服务的 Visual Basic 客户机。

SQCS2Axis.cs
SQCS2Axis.cs 或 第 837 页的图 194 是用于调用 **StockQuoteAxisService** 服务的 C# 客户机。 您可以通过在命令行中提供 URL 来覆盖此服务的 URL。

SQCS2DotNet.cs
SQCS2DotNet.cs 或 第 838 页的图 195 是用于调用 **StockQuoteDotNet** 服务的 C# 客户机。 您可以通过在命令行中提供 URL 来覆盖此服务的 URL。

```
Module SQVB2Axis
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj As New StockQuoteAxisService()
      Dim res As Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
   End Function
End Module
```

图 *192: SQVB2Axis*

```
Module SQVB2DotNet
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj as new StockQuoteDotNet()
      Dim res as Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
   End Function
End Module
```

图 *193: SQVB2DotNet*

```
using System;
class SQCS2Axis {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteAxisService stockobj = new StockQuoteAxisService();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("SQCS2Axis RPC reply is: " + res);
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

图 *194: SQCS2Axis*

```
using System;
class SQCS2DotNet {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteDotNet stockobj = new StockQuoteDotNet();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("RPC reply is: " + res);
         if (args.GetLength(0) == 0) {
            res = stockobj.getQuoteDOC("XXX");
            Console.WriteLine("DOC reply is: " + res);
         }
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

图 *195: SQCS2DotNet*

相关任务

使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-RPC 客户机
开发 Axis 1.4 Web Service 客户机以使用 WebSphere MQ Transport for SOAP 运行。

使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机
开发要使用 WebSphere MQ Transport for SOAP 运行的 Axis2 Web Service 客户机。 列出了随 WebSphere MQ Transport for SOAP 提供的样本 Axis2 客户机，并列出了用于生成代理的 **wsimport** 命令。

使用 WebSphere MQ Transport for SOAP 部署 Web Service

将 Web Service 部署到多个不同的服务器环境之一，并使用 WebSphere MQ Transport for SOAP 连接到该服务器。

开始之前

在目标环境中开发 Web service 并使用 SOAP over HTTP 对其进行测试。

关于此任务

您可以部署 Web Service 以在多个不同的 SOAP 运行时环境中与 WebSphere MQ Transport for SOAP 一起运行。 只能使用随 WebSphere MQ 一起安装的软件将服务部署到 Axis 1.4 。 对于其他运行时环境，必须安装其他软件。

您不限于将 WebSphere MQ Transport for SOAP 运行到具有部署指示信息的服务器。 使用这些指示信息将服务部署到其中一个所列环境中。

注: 某些集成环境使用 W3C 建议的 JMS SOAP 绑定以及 WebSphere MQ Transport for SOAP 绑定来提供 SOAP over JMS。 WebSphere MQ 发行版 (最高为 7.0.1.2) 仅支持 WebSphere MQ Transport for SOAP 绑定。 从 7.0.1.3 开始，您可以使用符合 SOAP over JMS 的 W3C 候选建议的 URI 来部署 Axis2 客户机。 请参阅教程 使用 WebSphere Application Server V7 和 Rational Application Developer V7.5 开发 SOAP/JMS JAX-WS Web Service 应用程序。

使用 *amqwdeployWMQService* 将服务部署到 *Axis 1.4* 以用于 *WebSphere Transport for SOAP*

通过创建部署目录，运行 **amqwdeployWMQService** 命令并启动 Axis 1.4 侦听器，将 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP。

开始之前

1. 遵循安装 WebSphere MQ Transport for SOAP 的指示信息
2. 使用 **runivt** 命令验证安装和您的环境。
3. 重新部署服务：
   a. 删除 ./generated 子目录及其所有子目录。
   b. 从目标队列中移除请求并删除队列。
   c. 继续从第 第 839 页的『2』 步执行指示信息。

关于此任务

这些指示信息用于首次部署 Axis 1.4 服务。 要重新启动 Axis 1.4 服务，请重新运行 Axis 1.4 SOAP 侦听器：第 第 840 页的『11』 步。

使用以下指示信息将新的 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP:

过程

1. 创建目录 *deployDir* 来保存部署文件。
   部署实用程序要求从单独目录部署每个服务。
2. 在 Windows 上打开命令窗口，或在 UNIX and Linux 系统上使用 X Window System 打开命令 shell，在 *deployDir* 中运行 **amqwdeployWMQService**。
3. 运行 **amqwsetcp** 以设置类路径。
   JRE 和 JDK 必须在 V5.0 或更高版本的类路径中，并属于同一版本级别。
4. 将类源 *className*.java 复制到 *deployDir*
5. 将 *className* 所在包中的所有 Java 源文件复制到 *deployDir*/*packageName* 中，其中 *packageName* 是对应于包名的目录树。
6. 运行 **javac** *packageName.className*.
   您可能需要添加当前目录 “.”的路径，或者添加到 **javac** 的 *packageName* 目录以查找其他类。
7. 为服务创建 Axis WSDL：

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdl
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. 为服务创建 WebSphere MQ 资源：

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

   提示:

   如果您想设置新的队列管理器以及队列管理器所需的资源以执行开发和测试，请运行 **setupWMQSOAP**。

   如果要将新队列管理器设置为缺省值，请从 *WMQ install directory*\tools\soap\samples 目录获取 **setupWMQSOAP** 的副本，并将 -q 参数添加到行中

```
call :try -q crtmqm %QMGR%
```

9. 创建 Axis 侦听器并部署服务：

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. 如果您需要为服务生成 WSDL，生成客户机存根或客户机代理，请运行带有以下参数之一的 **amqwdeployWMQService** 命令：
    - genAsmxWsdl
    - genAxisWsdl
    - genProxiesToDotNet
    - genProxiestoAxis

    注: 生成代理之前必须先生成 WSDL。 如果 CLASSPATH 没有设置为查找导入以编译 *className*.java 的所有类，AllAxis 选项将失败。 如果包含 *className*.java 的包中有多个 Java 文件，那么必须首先使用 **javac** 进行编译。 **amqwdeployWMQService** -f *packageName*.*className*.java -c CompileJava 仅编译 *className*.java。
11. 启动生成的 Axis 侦听器。

```
.\generated\server\startWMQJListener.cmd
```

相关任务

将服务部署到 .NET Framework 1 或 2 服务以使用 WebSphere MQ Transport for SOAP
将 .NET Framework 1 或 2 服务部署到 WebSphere MQ Transport for SOAP。 创建部署目录，运行 **amqwdeployWMQService** 命令，然后启动 .NET 侦听器。

将服务部署到 CICS Transaction Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 CICS Transaction Server 4.1 Web Service 支持中。

将服务部署到 WebSphere Application Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 WebSphere Application Server 上的服务集成总线中。

配置 WebSphere Application Server 以使用 W3C SOAP over JMS
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 1。 配置 WebSphere MQ 和 WebSphere Application Server 资源，以开发和部署绑定到 W3C SOAP over JMS 的 Web Service 作为传输。

将服务部署到 WebSphere ESB 和 Process Server 服务端点以使用 WebSphere Transport for SOAP
WebSphere ESB 和 Process Server 不直接支持 WebSphere MQ Transport for SOAP。 您必须配置定制导出。

将服务部署到 *.NET Framework 1* 或 *2* 服务以使用 *WebSphere MQ Transport for SOAP*

将 .NET Framework 1 或 2 服务部署到 WebSphere MQ Transport for SOAP。 创建部署目录，运行 **amqwdeployWMQService** 命令，然后启动 .NET 侦听器。

开始之前

1. 遵循安装 WebSphere MQ Transport for SOAP 的指示信息
2. 使用 **runivt** 命令验证安装和您的环境。
3. 必须设置 .NET Framework 文件 wsdl.exe 和 csc.exe 的路径。 由 PATH 变量标识的 wsdl.exe 和 csc.exe 的副本必须处于 .NET Framework 的同一级别。 如果已安装多个 .NET Framework，或者正在使用 Visual Studio，请仔细检查 PATH 变量。
4. 重新部署服务：
    a. 删除 ./generated 子目录及其所有子目录
    b. 从目标队列中移除请求并删除队列。
    c. 继续从第 第 841 页的『2』 步执行指示信息。

关于此任务

这些指示信息用于首次部署 .NET 服务。 要重新启动 .NET 服务，请重新运行 .NET SOAP 侦听器，步骤 第 841 页的『9』。

使用以下指示信息将新的 .NET Framework 1 或 .NET Framework 2 服务部署到 WebSphere MQ Transport for SOAP:

过程

1. 创建目录 *deployDir* 来保存部署文件。
   部署实用程序要求从单独目录部署每个服务。
2. 在 *deployDir* 中打开命令窗口以运行 **amqwdeployWMQService**。

```
C:\IBM\ID\QuoteClient>
```

3. 运行 **amqwsetcp** 以设置类路径。
   只有 Axis 客户机需要类路径。
4. 将 .NET 服务 *className*.asmx 复制到 *deployDir*
5. 将服务实现构建到库 (.dll) 中。

   内嵌服务实现位于 *className*.asmx 中。 代码隐藏服务实现可能是 *className*.asmx.cs。

   第 841 页的图 196 显示了用于将 .NET Framework V2 服务构建为库的命令示例。

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

   图 *196: .NET Framework V2* 服务的构建命令

6. 将 *className*.dll 复制到 *deployDir**bin* 中。
7. 设置 WebSphere MQ 资源，并创建服务所需的侦听器:

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```

8. 如果您需要为服务生成 WSDL，生成客户机存根或客户机代理，请运行带有以下参数之一的 **amqwdeployWMQService** 命令:
   - genAsmxWsdl
   - genAxisWsdl
   - genProxiesToDotNet
   - genProxiestoAxis

   注: 生成代理之前必须先生成 WSDL。
9. 启动生成的 .NET 侦听器。

```
.\generated\server\startWMQNListener.cmd
```

相关任务

使用 amqwdeployWMQService 将服务部署到 Axis 1.4 以用于 WebSphere Transport for SOAP
通过创建部署目录，运行 **amqwdeployWMQService** 命令并启动 Axis 1.4 侦听器，将 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP。

将服务部署到 CICS Transaction Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 CICS Transaction Server 4.1 Web Service 支持中。

将服务部署到 WebSphere Application Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 WebSphere Application Server 上的服务集成总线中。

配置 WebSphere Application Server 以使用 W3C SOAP over JMS
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 1。 配置 WebSphere MQ 和 WebSphere Application Server 资源，以开发和部署绑定到 W3C SOAP over JMS 的 Web Service 作为传输。

将服务部署到 WebSphere ESB 和 Process Server 服务端点以使用 WebSphere Transport for SOAP
WebSphere ESB 和 Process Server 不直接支持 WebSphere MQ Transport for SOAP。 您必须配置定制导出。

将服务部署到 *CICS Transaction Server* 以使用 *WebSphere Transport for SOAP*

WebSphere MQ Transport for SOAP 集成到 CICS Transaction Server 4.1 Web Service 支持中。

开始之前

使用相同的工具为 WebSphere MQ 的客户机或服务进行开发，就像为 HTTP 进行开发一样。 CICS 具有对应于 **Java2wsdl** 和 **wsdl2Java** 的工具:

- **DFHWS2LS** 使用 Web Service 描述作为起点。 它使用消息描述以及这些消息中使用的数据类型来构造高级语言数据结构。 可以在以其他语言编写的应用程序的结构中使用。
- **DFHLS2WS** 使用高级语言数据结构作为起点。 它使用结构来构造包含消息描述的 Web Service 描述。 它还可以根据语言数据结构来创建消息模式。

遵循 CICS 产品文档中的指示信息 创建 Web Service 以创建 Web Service。

关于此任务

遵循 CICS 产品文档中的指示信息 配置 CICS 以使用 WebSphere MQ 传输 。 通过使用指示信息，可以将 Web Service 部署到 WebSphere MQ Transport for SOAP。

相关任务

使用 amqwdeployWMQService 将服务部署到 Axis 1.4 以用于 WebSphere Transport for SOAP
通过创建部署目录，运行 **amqwdeployWMQService** 命令并启动 Axis 1.4 侦听器，将 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP。

将服务部署到 .NET Framework 1 或 2 服务以使用 WebSphere MQ Transport for SOAP
将 .NET Framework 1 或 2 服务部署到 WebSphere MQ Transport for SOAP。 创建部署目录，运行 **amqwdeployWMQService** 命令，然后启动 .NET 侦听器。

将服务部署到 WebSphere Application Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 WebSphere Application Server 上的服务集成总线中。

配置 WebSphere Application Server 以使用 W3C SOAP over JMS
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 1。 配置 WebSphere MQ 和 WebSphere Application Server 资源，以开发和部署绑定到 W3C SOAP over JMS 的 Web Service 作为传输。

将服务部署到 WebSphere ESB 和 Process Server 服务端点以使用 WebSphere Transport for SOAP
WebSphere ESB 和 Process Server 不直接支持 WebSphere MQ Transport for SOAP。 您必须配置定制导出。

将服务部署到 *WebSphere Application Server* 以使用 *WebSphere Transport for SOAP*

WebSphere MQ Transport for SOAP 集成到 WebSphere Application Server 上的服务集成总线中。

开始之前

使用 Rational Application Developer， WebSphere Integration Developer 或 Web Service 工具箱来开发 Web Service。

关于此任务

使用以下指示信息在 WebSphere Application Server 上使用 WebSphere MQ Transport for SOAP 作为 SOAP 传输来部署服务。

过程

1. 将 WebSphere MQ 配置为 WebSphere Application Server 上服务集成总线的 JMS 消息传递提供程序。
2. 配置服务所需的 WebSphere MQ 资源。
3. 请遵循 WebSphere Application Server Network Deployment 产品文档中的指示信息 为同步 SOAP over JMS 端点侦听器配置 JMS 资源。

   对于其他 WebSphere Application Server 平台，有相应的指示信息。
4. 修改服务 URI 以符合 WebSphere MQ Transport for SOAP URI。
5. 将服务部署到 WebSphere Application Server。

下一步做什么

使用 HTTP 作为传输协议部署服务，以便客户机可以查询服务并接收 WSDL 作为响应。

相关任务

使用 amqwdeployWMQService 将服务部署到 Axis 1.4 以用于 WebSphere Transport for SOAP
通过创建部署目录，运行 **amqwdeployWMQService** 命令并启动 Axis 1.4 侦听器，将 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP。

将服务部署到 .NET Framework 1 或 2 服务以使用 WebSphere MQ Transport for SOAP
将 .NET Framework 1 或 2 服务部署到 WebSphere MQ Transport for SOAP。 创建部署目录，运行 **amqwdeployWMQService** 命令，然后启动 .NET 侦听器。

将服务部署到 CICS Transaction Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 CICS Transaction Server 4.1 Web Service 支持中。

配置 WebSphere Application Server 以使用 W3C SOAP over JMS
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 1。 配置 WebSphere MQ 和 WebSphere Application Server 资源，以开发和部署绑定到 W3C SOAP over JMS 的 Web Service 作为传输。

将服务部署到 WebSphere ESB 和 Process Server 服务端点以使用 WebSphere Transport for SOAP
WebSphere ESB 和 Process Server 不直接支持 WebSphere MQ Transport for SOAP。 您必须配置定制导出。

配置 *WebSphere Application Server* 以使用 *W3C SOAP over JMS*

绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 1。 配置 WebSphere MQ 和 WebSphere Application Server 资源，以开发和部署绑定到 W3C SOAP over JMS 的 Web Service 作为传输。

开始之前

该任务需要 WebSphere Application Server v7.0.0.9 和 WebSphere MQ v7.0.1.3。

关于此任务

该任务有两个步骤:

过程

1. 第 844 页的『配置 WebSphere MQ 资源』
2. 第 845 页的『配置 WebSphere Application Server 资源』

下一步做什么

第 844 页的『配置 WebSphere MQ 资源』

相关任务

使用 amqwdeployWMQService 将服务部署到 Axis 1.4 以用于 WebSphere Transport for SOAP
通过创建部署目录，运行 **amqwdeployWMQService** 命令并启动 Axis 1.4 侦听器，将 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP。

将服务部署到 .NET Framework 1 或 2 服务以使用 WebSphere MQ Transport for SOAP
将 .NET Framework 1 或 2 服务部署到 WebSphere MQ Transport for SOAP。 创建部署目录，运行 **amqwdeployWMQService** 命令，然后启动 .NET 侦听器。

将服务部署到 CICS Transaction Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 CICS Transaction Server 4.1 Web Service 支持中。

将服务部署到 WebSphere Application Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 WebSphere Application Server 上的服务集成总线中。

将服务部署到 WebSphere ESB 和 Process Server 服务端点以使用 WebSphere Transport for SOAP
WebSphere ESB 和 Process Server 不直接支持 WebSphere MQ Transport for SOAP。 您必须配置定制导出。

配置 *WebSphere MQ* 资源

开始之前

对于 Axis2 支持，您需要 WebSphere MQ 7.0.1.3 或更高版本。

关于此任务

为简单起见，该任务假定 WebSphere MQ 与其他软件安装在同一工作站上，并使用绑定连接。 WebSphere Application Server 和 Axis2 客户机配置使用客户机连接。 要使用客户机连接与任务一起运行，请检查您是否可以在 Axis2 客户机和 WebSphere Application Server 计算机上的请求和应答队列中放入和获取消息。

同样，为了简单起见，不使用任何安全性配置。 用户标识具有完整的 mqm 权限。

过程

1. 创建缺省队列管理器 QM1。

   使用 WebSphere MQ Explorer 将 QM1 创建为缺省队列管理器。 将其配置为自动启动，然后选择用于创建侦听器的选项。 或者，使用以下命令:

   ```
   crtmqm -q -sa QM1
   strmqm
   echo define listener (LISTENER.TCP) trptype(tcp) ipaddr(localhost) port(1414)
            control(qmgr) replace | runmqsc
   echo start listener(LISTENER.TCP) | runmqsc
   ```

2. 定义请求队列 REQUESTAXIS 和应答队列 REPLYAXIS。

   使用资源管理器或以下命令:

   ```
   echo define ql(REQUESTAXIS) replace | runmqsc
   echo define ql(REPLYAXIS) replace | runmqsc
   ```

下一步做什么

第 845 页的『配置 WebSphere Application Server 资源』

开始之前

对于 W3C SOAP over JMS 支持，您需要 WebSphere Application Server v7。 此配置是在 WebSphere Application Server V 7.0 Test Environment v7.0.0.9 更新 1 上执行的。 WebSphere Application Server 是随 Rational Software Architect for WebSphere Software 7.5.4 提供的。 通过应用可用的最新更新， Rational Software Architect 已更新为 v7.5.5.1。

在安装过程中，为 WebSphere Application Server 创建概要文件。 在该任务中，未启用管理安全性。 缺省概要文件名称为 `was70profile1`，服务器为 `server1`。

关于此任务

配置 WebSphere Application Server。 您可以从 Rational Application Developer 启动服务器，从 "服务器" 视图启动管理控制台，也可以使用命令文件启动服务器并使用浏览器管理服务器。 该任务使用第二个方法。

服务器命令文件位于文件夹 *Rational Installation Root*\SDP\runtimes\base_v7\profiles\was70profile1\bin 中。 您可能要检查的日志文件位于 *Rational Installation Root*\SDP\runtimes\base_v7\profiles\was70profile1\logs\server1 中。

作为约定，所有 WebSphere MQ 对象名都是大写，引用 WebSphere MQ 对象的所有 JNDI 名称都是小写。

过程

1. 启动服务器。

   ```
   startServer server1
   ```

2. 启动浏览器，打开管理控制台，然后登录。

   ```
   http://localhost:9061/ibm/console/unsecureLogon.jsp
   ```

   在用户标识字段中输入任何字符串。

3. 创建连接工厂 qm1
   a) 在导航器中，打开 **资源** > **JMS** > **连接工厂**。
   b) 在 "连接工厂" 窗口中，选择作用域 **Node =*nodename***，然后单击 **新建**。
   c) 选择 **WebSphere MQ 消息传递提供程序** > **确定**。
   d) 从 第 845 页的表 143 > **下一步**提供队列管理器连接信息。

表 *143:* 队列管理器连接信息	
字段名称	**值**
名称	qm1
JNDI 名称	qm1

   e) 选择 **在此向导中输入所有必需信息** 作为连接方法> **下一步**。
   f) 输入 QM1 作为队列连接详细信息> **下一步**。
   g) 从 第 845 页的表 144 > **下一步**输入连接详细信息。

表 *144:* 连接详细信息	
字段名称	**值**
传输	Bindings, then client
Hostname	localhost
端口	1414

表 *144:* 连接详细信息 *(继续)*	
字段名称	**值**
服务器连接通道	SYSTEM.DEF.SVRCONN

h) **测试连接** > **下一步** > **完成** > **保存**。

4. 创建 JMS 请求队列 requestaxis。
   a) 在 Navigator 中，打开 **资源** > **JMS** > **队列**。
   b) 在 "连接工厂" 窗口中，选择作用域 **Node =*nodename***，然后单击 **新建**。
   c) 选择 **WebSphere MQ 消息传递提供程序** > **确定**。
   d) 从 第 846 页的表 145 > **确定** > **保存**输入队列详细信息。

表 *145:* 队列详细信息	
字段名称	**值**
名称	requestaxis
JNDI 名称	requestaxis
队列名称	REQUESTAXIS
队列管理器名称	QM1

5. 重复步骤 第 846 页的『4』 以创建 JMS 应答队列 replyaxis。
6. 创建激活规范 qm1as。

   当消息到达请求队列时，激活规范将触发 Web Service 路由器消息驱动的 Bean (MDB)。 MDB 是在由 Rational Application Developer Web Service 向导创建的 Web Service 的部署描述符中定义的。

   a) 在 " Navigator" 中，打开 **资源** > **JMS** > **激活规范**。
   b) 在 "连接工厂" 窗口中，选择作用域 **Node =*nodename***，然后单击 **新建**。
   c) 选择 **WebSphere MQ 消息传递提供程序** > **确定**。
   d) 从 第 846 页的表 146 > **下一步**输入激活规范的基本属性。

表 *146:* 激活规范名称	
字段名称	**值**
名称	qm1as
JNDI 名称	qm1as

   e) 从 第 846 页的表 147 > **下一步**中指定其 MDB 信息。

表 *147: MDB* 信息	
字段名称	**值**
目标 JNDI 名称	requestaxis
消息选择器	留空
目标类型	Queue

   f) 选择 **在此向导中输入所有必需信息** 作为连接方法> **下一步**。
   g) 输入 QM1 作为队列连接详细信息> **下一步**。
   h) 从 第 845 页的表 144 > **下一步**输入连接详细信息。

表 *148:* 连接详细信息	
字段名称	**值**
传输	`Bindings, then client`
Hostname	`localhost`
端口	`1414`
服务器连接通道	`SYSTEM.DEF.SVRCONN`

i) **测试连接** > **下一步** > **完成** > **保存**。

7. 为应答队列创建队列连接工厂 `jms/WebServicesReplyQCF`。

Web Service 路由器使用队列连接工厂来访问应答队列。 在 Web Service 的部署描述符中，将为队列连接工厂提供缺省 JNDI 名称 `jms/WebServicesReplyQCF`。 您可以在部署描述符中更改名称。 在此任务中，将缺省名称添加到 JMS 资源定义。

a) 在 " Navigator" 中，打开 **资源** > **JMS** > **队列连接工厂**。

b) 在 "连接工厂" 窗口中，选择作用域 **Node =*nodename***，然后单击 **新建**。

c) 选择 **WebSphere MQ 消息传递提供程序** > **确定**。

d) 从 第 847 页的表 149 > **下一步**输入队列连接工厂的基本属性。

表 *149:* 队列连接工厂名称	
字段名称	**值**
名称	`WebServicesReplyQCF`
JNDI 名称	`jms/WebServicesReplyQCF`

e) 选择 **在此向导中输入所有必需信息** 作为连接方法> **下一步**。

f) 输入 `QM1` 作为队列连接详细信息> **下一步**。

g) 从 第 845 页的表 144 > **下一步**输入连接详细信息。

表 *150:* 连接详细信息	
字段名称	**值**
传输	`Bindings, then client`
Hostname	`localhost`
端口	`1414`
服务器连接通道	`SYSTEM.DEF.SVRCONN`

h) **测试连接** > **下一步** > **完成** > **保存**。

下一步做什么

第 809 页的『为 W3C SOAP over JMS 开发 JAX-WS EJB Web Service』

将服务部署到 *WebSphere ESB* 和 *Process Server* 服务端点以使用 *WebSphere Transport for SOAP*

WebSphere ESB 和 Process Server 不直接支持 WebSphere MQ Transport for SOAP。 您必须配置定制导出。

关于此任务

WebSphere Integration Developer 提供可绑定到 WebSphere MQ JMS 导出以创建定制 WebSphere MQ JMS SOAP 导出的 SOAP 数据变换。

遵循指示信息来创建定制的导出，以通过 WebSphere MQ Transport for SOAP 接收 SOAP 请求。

过程

1. 请参阅 WebSphere Process Server for Multiplatforms V6.2 产品文档中的 导入和导出概述 和 How to connect to WebSphere MQ 。
2. 请执行 IBM Business Process ManagerV 8.6 产品文档中的任务 生成 MQ JMS 导出绑定 。
   使用 预打包 JMS 数据格式变换 中描述的 SOAP 数据绑定来格式化 SOAP 消息。

相关任务

使用 amqwdeployWMQService 将服务部署到 Axis 1.4 以用于 WebSphere Transport for SOAP
通过创建部署目录，运行 **amqwdeployWMQService** 命令并启动 Axis 1.4 侦听器，将 Axis 1.4 服务部署到 WebSphere MQ Transport for SOAP。

将服务部署到 .NET Framework 1 或 2 服务以使用 WebSphere MQ Transport for SOAP
将 .NET Framework 1 或 2 服务部署到 WebSphere MQ Transport for SOAP。 创建部署目录，运行 **amqwdeployWMQService** 命令，然后启动 .NET 侦听器。

将服务部署到 CICS Transaction Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 CICS Transaction Server 4.1 Web Service 支持中。

将服务部署到 WebSphere Application Server 以使用 WebSphere Transport for SOAP
WebSphere MQ Transport for SOAP 集成到 WebSphere Application Server 上的服务集成总线中。

配置 WebSphere Application Server 以使用 W3C SOAP over JMS
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 1。 配置 WebSphere MQ 和 WebSphere Application Server 资源，以开发和部署绑定到 W3C SOAP over JMS 的 Web Service 作为传输。

部署 Web Service 客户机以使用 WebSphere MQ Transport for SOAP

将 Web Service 客户机部署到多个不同的客户机环境之一，并使用 WebSphere MQ Transport for SOAP 连接到服务。

开始之前

开发 Web Service 并将其部署为使用 WebSphere MQ Transport for SOAP。

关于此任务

您可以部署 Web Service 客户机以在多个不同的客户机环境中与 WebSphere MQ Transport for SOAP 一起运行。 只能使用随 WebSphere MQ 一起安装的软件将 Java 客户机部署到 Axis 1.4 。 对于其他客户机环境，必须安装其他软件。

您不限于在有部署指示信息的客户机环境中运行 WebSphere Transport for SOAP。 使用这些指示信息将客户机部署到其中一个受支持的环境中。

注: 某些集成环境使用 W3C 建议的 JMS SOAP 绑定以及 WebSphere MQ Transport for SOAP 绑定来提供 SOAP over JMS。 WebSphere MQ 发行版 (最高为 7.0.1.2) 仅支持 WebSphere MQ Transport for SOAP 绑定。 从 7.0.1.3 开始，您可以使用符合 SOAP over JMS 的 W3C 候选建议的 URI 来部署 Axis2 客户机。 请参阅教程 使用 WebSphere Application Server V7 和 Rational Application Developer V7.5 开发 SOAP/JMS JAX-WS Web Service 应用程序。

将 *Web Service* 客户机部署到 *Axis 1.4* 以使用 *IBM WebSphere MQ Transport for SOAP*

为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 IBM WebSphere MQ 队列和通道，启动服务并测试客户机。

开始之前

提示: 将服务部署到 HTTP，针对 HTTP 开发和测试客户机，然后针对 IBM WebSphere MQ Transport for SOAP 修改客户机：

1. 将 Register.extension() 调用添加到客户机。
2. 更改客户机代理定位器类中的静态 Web Service 地址以使用 IBM WebSphere MQ Transport for SOAP 的 URI。

关于此任务

与 HTTP 客户机相比，部署 Axis 1.4 客户机以使用 IBM WebSphere MQ Transport for SOAP 需要用到另外一个部署步骤。 您必须创建客户机部署描述符 `client-config.wsdd` 以将 `jms:` 传输映射到发送方类 `com.ibm.mq.soap.transport.jms.WMQSender`。

如果您使用命令 **amqwdeployWMQService** 生成客户机代理，您可以使用命令生成的目录部署客户机。

过程

1. 创建目录 *deployDir* 来保存客户机部署文件。
2. 在 Windows 系统上打开命令窗口，或在 UNIX and Linux 系统上使用 X Window System 打开命令 shell，转至 *deployDir*。
3. 运行 **amqwsetcp.cmd** 命令以设置 CLASSPATH
4. 运行 **amqwclientconfig.cmd** 命令以在 *deployDir* 目录中创建 Axis 1.4 客户机部署描述符 `client-config.wsdd`。
5. 确保客户机软件包中的类、客户机代理类和客户机使用的库均位于 CLASSPATH 中。

   amqwdeployWMQService 将 .NET 客户机代理放入 `./generated/server/soap/client/remote/dotnetService` 中，将 Axis 1.4 代理放入 `./generated/server/soap/client/remote/`*`client package`* 中。

示例

此示例显示了来自 Axis 1.4 Java 客户机的配置和输出 第 850 页的图 199。 客户机（第 850 页的图 198）调用用于回传其输入参数的 Web Service。 服务定义（第 849 页的图 197）显示了从服务 WSDL 获取的 URI。

```
<wsdl:service name="QuoteSOAPImplService">
    wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
                        name="org.example.www.QuoteSOAPImpl_Wmq">
       <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
           &connectionFactory=(connectQueueManager(QM1)binding(server))
           &initialContextFactory=com.ibm.mq.jms.Nojndi
           &targetService=org.example.www.QuoteSOAPImpl.java
           &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
</wsdl:service>
```

图 *197:* 服务定义

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                  + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

图 *198: Axis 1.4 Java* 客户机

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

图 *199:* 客户机配置和输出

下一步做什么

1. 如果要将此客户机部署为 IBM WebSphere MQ 客户机，请配置客户机和服务器连接通道。
2. 如果要将此客户机部署到不同于服务的队列管理器，那么必须使目标队列可用于此客户机。 将服务队列管理器上的目标队列配置为集群队列，或将客户机队列管理器上的目标队列配置为远程队列定义。

相关任务

将 Web Service 客户机部署到 Axis2 以使用 WebSphere MQ Transport for SOAP
为客户机准备部署目录和 Axis2 配置文件。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

使用 W3C SOAP over JMS 部署到 Axis2 客户机
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 4。 修改为 WebSphere MQ Transport for SOAP 开发的 Axis2 客户机中的 URL ，以使用 SOAP over JMS 的 W3C 候选建议。

将 Web Service 客户机部署到 .NET Framework 1 和 2 以使用 WebSphere MQ Transport for SOAP
为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

将 *Web Service* 客户机部署到 *Axis2* 以使用 *WebSphere MQ Transport for SOAP*

为客户机准备部署目录和 Axis2 配置文件。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

开始之前

提示: 将服务部署到 HTTP。 针对 HTTP 开发和测试客户机，然后使用 WebSphere MQ Transport for SOAP 修改 URL 以引用服务。

此任务显示如何将非受管 Axis2 客户机部署到 Java Standard Edition。 您可能希望将 Axis2 客户机部署到 Web 容器。 在 第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』中，您在 Web 容器中开发了客户机，并将其部署到 WebSphere Application Server Community Edition。 作为服务器配置的一部分，您启用了 Axis2 构面并将此构面包含在 Web 容器的配置中。 要在其他应用程序服务器上配置 Web 容器，请参考位于网址 http://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_container 中的 Axis2 文档，或 Web 服务器随附的文档。

注: Axis2 使用术语 Servlet 容器。 Servlet 容器与 Web 容器一样。

关于此任务

部署 Axis2 客户机以使用 WebSphere MQ Transport for SOAP 类似于部署 Axis2 客户机以使用 HTTP。 需要执行其他步骤才能提供 WebSphere MQ JAR 文件的类路径，以及修改 Axis2 配置文件。 Axis2 配置文件需要 JMS 的附加条目。 此条目引用实现 JMS `transportSender` 的 WebSphere MQ Transport for SOAP JAR 文件。

Axis2 提供了脚本 `axis2.bat` 或 `axis2.sh`，这些脚本可简化客户机部署；请参阅第 853 页的图 203 和 第 853 页的图 204 中的示例。

注:

1. `axis2.bat` 中包含必须更正的错误。 必须将字符串 `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` 更改为 `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`。
2. 在 `axis2.bat` 和 `axis2.sh` 中，不是将 Axis2 JAR 文件单独添加到类路径中，而是使用 `-Djava.ext.dirs` 引用所有 Axis2 JAR 文件这一快捷方式。 遗憾的是，这种方法存在缺陷，只能用于某些 JRE。 它无法与 IBM JRE 配合使用。

   JVM 参数 `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`使 Axis JAR 文件可供 JVM 使用。 JVM 尝试实例化某些 Axis JAR 文件，但导致错误，错误的详细信息取决于 JVM。 通常，您可能会在堆栈跟踪中看到以下其中一行:

   ```
   org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)
   ```

   或者 `org.apache.axis2.deployment.DeploymentException: java.security.NoSuchAlgorithmException: MD5 MessageDigest not available`

   运行非受管 Axis2 客户机的正确方式是将 Axis2 JAR 文件添加到类路径。 类路径只能用于客户机应用程序，不能用于 JVM。

此过程描述了不使用 axis2 脚本的情况下运行非受管 Axis2 客户机的常规步骤。 第 852 页的图 201 和 第 853 页的图 202 中的示例是用于 Windows 和 Linux 的脚本。

过程

1. 从 http://ws.apache.org/axis2/download/1_4_1/download.cgi 下载 Axis2 1.4.1 并将其解包到文件夹 `Axis2-1.4.1`。
2. 更新 `Axis2-1.4.1\conf` 中的 `axis2.xml`。
   a) 更新 `Axis2-1.4.1\conf` 中的 `axis2.xml`。 将 WebSphere MQ Transport for SOAP 添加为 `transportSender`:

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
   ```

   b) 如果需要，请更改连接池大小缺省值 10。

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
   <parameter name="ResourcePoolCapacity">20</parameter>
   </transportSender>
   ```

   `ResourcePoolCapacity` 定义缓存中保留的服务端点条目数量。 该值必须至少为 1。 如果服务端点条目数超过高速缓存大小，那么将删除条目以为新条目腾出空间。 端点条目的大小各有不同。 请设置足够大的数目以避免缓存颠簸。

请参阅第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』中的第 3 步。

3. 创建目录 *deployDir*。 在此目录下，复制包含客户机和客户机代理的文件夹结构。 *deployDir* 等同于 Eclipse Java 项目中的 *project*\bin 文件夹。
4. 在 Windows 上打开命令窗口，或者在 *deployDir* 中使用 UNIX and Linux 系统上的 X Window System 打开命令 shell。
5. 更新类路径以包括当前目录、Axis2 JAR 文件、com.ibm.mqjms.jar 和 com.ibm.mq.axis2.jar。 com.ibm.mqjms.jar 引用所需的所有其他 WebSphere MQ JAR 文件。
6. 使用 **Java** 命令启动客户机程序。

示例

第 853 页的图 202 到第 853 页的图 204 列出了四个运行 Axis2 客户机的示例。 第 852 页的图 200 显示了运行第 825 页的图 183 中列出的异步客户机产生的输出。

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

图 *200:* 运行 *SQA2AsyncClient* 产生的输出

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
".;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

图 *201: runpojo.bat: Windows*，使用类路径

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
  AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME":"$JAVA_HOME/lib/tools.jar":"$AXIS2_CLASSPATH":"$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

图 *202: runpojo.sh*：*Linux*，使用类路径。

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

图 *203: runaxis2.bat: Windows*，使用 *axis2.bat*

注意

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

图 *204: runaxis2.sh*：*Linux*，使用 *axis2.sh*

注意

相关任务

将 Web Service 客户机部署到 Axis 1.4 以使用 IBM WebSphere MQ Transport for SOAP
为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 IBM WebSphere MQ 队列和通道，启动服务并测试客户机。

使用 W3C SOAP over JMS 部署到 Axis2 客户机
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 4。 修改为 WebSphere MQ Transport for SOAP 开发的 Axis2 客户机中的 URL，以使用 SOAP over JMS 的 W3C 候选建议。

将 Web Service 客户机部署到 .NET Framework 1 和 2 以使用 WebSphere MQ Transport for SOAP
为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

使用 *W3C SOAP over JMS* 部署到 *Axis2* 客户机

绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 4。 修改为 WebSphere MQ Transport for SOAP 开发的 Axis2 客户机中的 URL，以使用 SOAP over JMS 的 W3C 候选建议。

开始之前

必须首先完成任务 第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』，以使用 Axis2 客户机和 WebSphere MQ Transport for SOAP 协议来调用 **SimpleJavaListener**。

您还必须在先前任务中创建了 Web Service 并配置了 WebSphere MQ 和 WebSphere Application Server :

1. 第 844 页的『配置 WebSphere MQ 资源』.
2. 第 845 页的『配置 WebSphere Application Server 资源』.
3. 第 809 页的『为 W3C SOAP over JMS 开发 JAX-WS EJB Web Service』.

在此任务中，客户机在 Eclipse Galileo 中运行。 可以通过修改 Axis2 随附的 `Axis2.bat` 文件从命令行中运行客户机。

关于此任务

要调用 WebSphere Application Server 托管的 StockQuoteAxis 服务，您必须对现有 Axis2 `StockQuoteAxis` 静态客户机进行的唯一更改是更改传递给客户机的 URL。 因为 WSDL 尚未更改，您可以使用 `soap.server` 软件包中相同的代理类。

您可以使用两种方法定义要传递到客户机的 URL。 您可以使用与生成的 `StockQuoteAxis.wsdl` 中的 URL 相同的 URL。 您必须添加 `jndiInitialContextFactory` 和 `jndiURL` 参数以访问 WebSphere Application Server JNDI 目录。 另一种方法是更改 URL 并给予客户机直接访问 `QM1` 上的 `REQUESTAXIS` 和 `REPLYAXIS` 队列的权限，无需使用 JNDI 查找。

在传递到 Axis2 客户机的 URL 中定义的连接参数用于连接到发送和接收 SOAP 消息所需的 WebSphere MQ 队列管理器和队列。 服务不需要使用传递到 Axis2 客户机的连接参数。 您可以使用 WebSphere MQ 的分布式排队功能将客户机和服务与使用同一队列管理器或同一名称服务器的客户机和服务分离。

过程

1. 保存生成的 `StockQuoteAxis.wsdl` 中的 URL，并关闭 Rational Application Developer 以节省内存。

   如果未更改服务器配置，那么关闭 Rational Application Developer 将停止应用程序服务器。 这种情况下，请使用以下命令启动服务器:

   ```
   startserver server1
   ```

2. 在包含 Axis2 客户机项目的工作空间中打开 Eclipse Galileo。
3. 打开 `SQA2StaticClient.java`。

   请参阅 SQA2StaticClient.java。
4. 使用 URI 的 `queue` 变体调用服务。

   a) 修改 URL。

   新的 URI 为:

   ```
   jms:queue:REQUESTAXIS
             ?replyToName=REPLYAXIS
             &connectionFactory=connectQueueManager(QM1)Bind(Server)
             &targetService=StockQuoteAxis;
   ```

   比较此 URL 与 `StockQuoteAxis.wsdl` 中的 URL:

   ```
   jms:jndi:requestaxis
            ?jndiConnectionFactoryName=qm1
            &targetService=StockQuoteAxis
   ```

   图 *205: `StockQuoteAxis.wsdl` 中的 URL*

   - 现在，`REQUESTAXIS` 作为队列名称（而非 JNDI 名称）时为大写。
   - 与 `QM1` 的连接很简单。
   - URI 不包含回复目标的名称。 客户机必须定义要回复到的队列。

   b) 使用相同的 **运行方式 ...** 运行 `SQA2StaticClient.java` 配置，如您在任务 第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』中所执行的那样。
5. 使用 URI 的 `jndi` 变体 (使用 WebSphere Application Server 作为命名服务器) 来调用服务。

a) 使用来自 StockQuoteAxis.wsdl 第 854 页的图 205 的 URL，提供缺少的参数以在 WebSphere Application Server 中使用命名服务。

必须提供的缺少参数和值有：

表 *151:* 其他 *JNDI* 参数

参数	该示例中使用的值	描述
&jndiURL	iiop://localhost:2810 或 corbaname:iiop:localhost:2810	命名提供者的 URI。 对于 WebSphere Application Server，该值缺省为 2809。也称为 RMI 接口的端口号和引导程序端口。 该值列在 SystemOut.log 中 `00000000 NameServerImp A NMSV0018I: Name server available on bootstrap port 2810`
&jndiInitialContextFactory	com.ibm.websphere.naming.WsnInitialContextFactory	WebSphere Application Server 使用的初始上下文工厂的名称。
&replyToName	replyaxis	REPLYAXIS 队列的 JNDI 名称。

```
jms:jndi:requestaxis?
         &jndiURL=iiop//localhost:2810
         &jndiConnectionFactoryName=qm1
         &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
         &targetService=StockQuoteAxis
         &replyToName=replyaxis;
```

b) 添加 JNDI 查找所需的 JAR 文件。

在此配置中，以下 JAR 文件已添加到构建路径中，以使用 JMS URL 的 jndi 变体来运行任务：

- 来自 *Rational install directory*\SDP\runtimes\base_v7\runtimes 的 com.ibm.jaxws.thinclient_7.0.0.jar。
- com.ibm.ws.runtime.jar from *Rational install directory*\SDP\runtimes\base_v7\plugins

对于其他 JNDI 提供者，您需要其他 JAR 文件。

构建路径中的其他 JAR 文件为：

i) *WebSphere MQ Install directory*\java\lib 中的所有 JAR 文件。

ii) *Axis2-1.5.1*\lib 中的所有 JAR 文件。

iii) Java 6.0 JRE。

c) 使用相同的 **运行方式 ...** 运行 SQA2StaticClient.java 配置，如您在任务 第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』中所执行的那样。

结果

这两种情况下，来自服务的回复都会显示在客户机控制台视图中。

相关任务

将 Web Service 客户机部署到 Axis 1.4 以使用 IBM WebSphere MQ Transport for SOAP
为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 IBM WebSphere MQ 队列和通道，启动服务并测试客户机。

将 Web Service 客户机部署到 Axis2 以使用 WebSphere MQ Transport for SOAP

为客户机准备部署目录和 Axis2 配置文件。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

将 Web Service 客户机部署到 .NET Framework 1 和 2 以使用 WebSphere MQ Transport for SOAP
为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

将 *Web Service* 客户机部署到 *.NET Framework 1* 和 *2* 以使用 *WebSphere MQ Transport for SOAP*

为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

开始之前

提示: 使用 Visual Studio 开发和测试服务和客户机。 然后修改 WebSphere MQ Transport for SOAP 的客户机。

1. 如果要使用 .NET Framework 1 或 2 部署服务，请将该服务构建为库 (`.dll`)。 使用 WebSphere MQ Transport for SOAP 进行部署。
2. 将 Register.Extension() 调用添加到客户机。
3. 添加对位于 *MQ_Install*`\bin` 中的 amqsoap.dll 的引用。
4. 将客户机代理类构造函数中的静态 `Url` 属性更改为 `jms:/` URI，以用于 WebSphere MQ Transport for SOAP。

关于此任务

部署 Web Service Client for .NET Framework 1 或 2 以使用 WebSphere MQ Transport for SOAP 需要额外的部署步骤。 您需要向 .NET Framework 注册 `amqsoap.dll` 。 `amqsoap.dll` 是在安装 WebSphere MQ Transport for SOAP 的过程中自动注册的，但您可能需要重新注册它。

如果您使用命令 **amqwdeployWMQService** 生成客户机代理，您可以使用命令生成的目录部署客户机。

过程

1. 创建目录 *deployDir* 来保存客户机部署文件。
2. 在 *deployDir* 中打开命令窗口。
3. 如果此服务是在 Axis 1.4 上运行，请运行 **amqwsetcp** 以设置 CLASSPATH。
4. 如有必要，请运行 **amqwRegisterDotNet** 以向 .NET Framework 注册 `amqsoap.dll` 。

示例

此示例显示来自 .NET Framework V2 客户机的配置和输出 第 857 页的图 208。 客户机（第 857 页的图 207）调用用于回传其输入参数的 Web Service。 第 856 页的图 206 中的静态 URL 定义显示了客户机代理的构造函数。

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
               &connectionFactory=(connectQueueManager(QM1)binding(server))
               &initialContextFactory=com.ibm.mq.jms.Nojndi
               &targetService=Quote.asmx
               &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

图 *206:* 静态客户机代理构造函数

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
```

图 *207:* 客户机程序

```
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM
```

图 *208:* 配置和输出

下一步做什么

1. 如果要将客户机部署为 WebSphere MQ MQI 客户机，请配置客户机和服务器连接通道。
2. 如果要将此客户机部署到不同于服务的队列管理器，那么必须使目标队列可用于此客户机。 将服务队列管理器上的目标队列配置为集群队列，或将客户机队列管理器上的目标队列配置为远程队列定义。

相关任务

将 Web Service 客户机部署到 Axis 1.4 以使用 IBM WebSphere MQ Transport for SOAP
为客户机准备部署目录和部署描述符。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 IBM WebSphere MQ 队列和通道，启动服务并测试客户机。

将 Web Service 客户机部署到 Axis2 以使用 WebSphere MQ Transport for SOAP
为客户机准备部署目录和 Axis2 配置文件。 提供客户机代理和客户机类，然后设置 CLASSPATH。 配置 WebSphere MQ 队列和通道，启动服务并测试客户机。

使用 W3C SOAP over JMS 部署到 Axis2 客户机
绑定到 SOAP over JMS 的 W3C 候选建议的 Web Service 必须在 Java EE 应用程序服务器的 EJB 容器中运行。 此任务是使用 W3C SOAP over JMS 协议连接 Axis2 Web Service 客户机和部署到 WebSphere Application Server 的 Web Service 的步骤 4。 修改为 WebSphere MQ Transport for SOAP 开发的 Axis2 客户机中的 URL ，以使用 SOAP over JMS 的 W3C 候选建议。

使用 W3C SOAP over JMS 和 WebSphere Application Server 将 Axis2 客户机连接到 JAX-WS 服务

完成此任务后，您将从 Axis2 客户机调用在 WebSphere Application Server 中运行的 JAX-WS Web Service。 Axis2 客户机和 WebSphere Application Server 将 W3C 候选建议用于在 WebSphere MQ 上运行的 SOAP over JMS 协议。 使用 Eclipse Galileo 和 Rational Application Developer 分别构建 Web Service 客户机和 Web Service。

开始之前

该任务需要 Rational Software Development Environment 和 WebSphere Application Server V 7。 该任务是使用 理性 Software Architect for WebSphere Software v7.5.5.1 和 WebSphere Application Server V 7.0 Test Environment v7.0.0.9 Update 1 打包的 理性 Application Developer 创建的。 您还需要 WebSphere MQ v7.0.1.3。

该任务基于另外两个任务 (第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』和 第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』) 进行构建。 要完成这些任务，您的开发环境已安装 Eclipse Galileo ， WASCE ， WASCE 的 Eclipse 插件和 Axis2 1.4.1 。 此任务不需要 WASCE。

某些步骤非常复杂。 这些步骤假定您已熟悉使用 Rational Application Developer 为 WebSphere Application Server 开发 Web Service 应用程序。 此任务的处理器和内存需求很大。 该任务是在分配了 1.8GB 内存的 VMWare Windows XP SP3 虚拟机中执行的。

开始执行此任务之前，请先安装所有软件。 这些软件的下载大概需要一天时间，安装需要一天时间，具体取决于您的带宽。 此任务至少需要半天时间。

关于此任务

此任务的场景是您已使用开放式源代码工具 Eclipse Galileo 开发库存报价 Web Service `StockQuoteAxis`。 `StockQuoteAxis` 使用在开放式源代码服务器 WASCE 上运行的 SOAP over HTTP 进行部署。

您要将部署的 Web Service 绑定到基于标准的消息传递传输 (例如， SOAP over JMS) ，或者绑定到 Web Service 可靠消息传递以及 SOAP over HTTP。 您希望客户机和服务均使用基于标准的接口。 因此，尽管未来的项目开发团队已使用 WebSphere MQ Transport for SOAP 实现解决方案，但您尚未投入生产。

Axis2 客户机已除去 WebSphere MQ Transport for SOAP 的 SOAP 客户机需要从 HTTP 客户机进行更改的问题。 问题仍然存在: 由 IBM WebSphere MQ Transport for SOAP 连接的服务由 WebSphere MQ 提供的特殊侦听器托管: `SimpleJavaListener`。

由于 W3C SOAP over JMS 标准处于候选推荐状态，因此某些供应商正在提供对 W3C SOAP over JMS 的支持。 该支持能让您将 Web Service 部署到应用程序服务器并使用各种不同的连接协议连接到同一服务。 WebSphere Application Server v7 提供的支持可消除必须单独托管 Web Service 以使用基于消息的 SOAP 传输的问题。 使用基于标准的消息传输接口 JMS 意味着您可以使用来自不同供应商的工具来开发解决方案。 您希望 Eclipse 中的 Web Service 工具将来包含 SOAP over JMS 绑定。

大部分步骤是使用 Eclipse 或 WebSphere 产品随附的管理工具执行的。 针对 Windows 环境描述了这些步骤。 通过对其中某些命令稍作修改，可以在其他平台上执行这些步骤。

列出了创建 HTTP Web service 以及使用 Axis2 连接到此服务的预备步骤。 将使用这些步骤中的客户机和 WSDL 来创建解决方案

过程

1. 使用 Axis2 客户机和 IBM WebSphere MQ Transport for SOAP 连接到 StockQuoteAxis Web service
   a) 第 803 页的『使用 Eclipse 为 WebSphere MQ Transport for SOAP 开发 JAX-RPC 服务』
   b) 第 821 页的『使用 Eclipse 为 WebSphere Transport for SOAP 开发 JAX-WS 客户机』
   c) 第 850 页的『将 Web Service 客户机部署到 Axis2 以使用 WebSphere MQ Transport for SOAP』
2. 使用 Axis2 客户机和 SOAP over JMS 的 W3C 候选建议连接到 StockQuoteAxis Web Service。
   a) 第 844 页的『配置 WebSphere MQ 资源』
   b) 第 845 页的『配置 WebSphere Application Server 资源』
   c) 第 809 页的『为 W3C SOAP over JMS 开发 JAX-WS EJB Web Service』
   d) 第 853 页的『使用 W3C SOAP over JMS 部署到 Axis2 客户机』

用于 HTTP 的 WebSphere MQ 网桥

通过 WebSphere MQ bridge for HTTP ，客户机应用程序可以与 WebSphere MQ 交换消息，而无需安装 WebSphere MQ MQI 客户机。 您可以从具有 HTTP 功能的任何平台或语言调用 WebSphere MQ 。

WebSphere MQ Bridge for HTTP 简介

WebSphere MQ Bridge for HTTP 是一个 Java， Enterprise Environment (JEE) Web 应用程序。 HTTP 客户机可以向其发送 **POST**， **GET** 和 **DELETE** 请求，以从 WebSphere MQ 队列中放入，浏览和删除消息。 WebSphere MQ Bridge for HTTP 不适合与消息一起使用 (如果需要有保证的传递)。

优势

通过 WebSphere MQ Bridge for HTTP，您可以使用 HTTP 从各种环境发送和接收 WebSphere MQ 消息:

- 支持 HTTP 但不支持 WebSphere MQ 的环境。
- 存储空间不足的环境，无法安装 WebSphere MQ MQI 客户机。
- 环境过多，无法在需要访问 WebSphere MQ 的每个系统上安装 WebSphere MQ MQI 客户机。
- 基于 Web 的应用程序，您希望从中发送或接收消息，而无需对自己的网桥进行编码以连接到 WebSphere MQ。
- 基于 Web 的应用程序，您希望使用异步技术（例如，AJAX）来增强该应用程序。 WebSphere MQ Bridge for HTTP 使 WebSphere MQ 队列和主题可以使用基于 HTTP 的表示状态传输 (REST)。

可以将 HTTP 支持与点到点和发布/预订消息传递拓扑一起使用。

HTTP 支持的工作原理

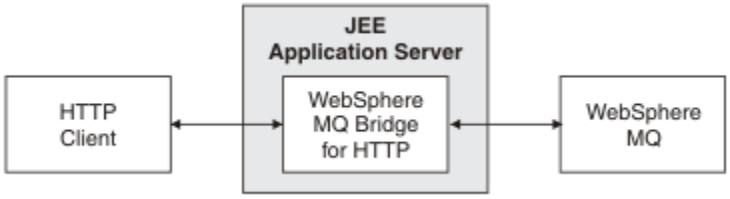


图 *209: 用于 HTTP 的 WebSphere MQ* 网桥

WebSphere MQ Bridge for HTTP Web 应用程序接收来自一个或多个客户机的 HTTP 请求。 它代表它们与 WebSphere MQ 进行交互，并返回对它们的 HTTP 响应。

WebSphere MQ Bridge for HTTP 是使用资源适配器连接到 WebSphere MQ 的 JEE servlet。 HTTP servlet 可以处理三种不同类型的 HTTP 请求：**POST**、**GET** 和 **DELETE**。

表 *152: WebSphere MQ Bridge for HTTP* 动词

HTTP 请求	**结果**
POST	将消息放到队列或主题上。
GET	浏览队列上的第一个消息。 根据 HTTP 协议，**GET** 不从队列中删除消息。 请勿将 **GET** 用于发布/预订消息传递。
DELETE	从队列或主题中获取并删除消息。

HTTP POST 示例

HTTP **POST** 将消息放到队列或将发布放到主题。 **HTTPPOST** Java 样本是将消息放入队列的 HTTP **POST** 请求示例。 不使用 Java，您可以改用浏览器表单或 AJAX 工具包创建 HTTP **POST** 请求。

第 860 页的图 210 显示了用于将消息放入名为 `myQueue` 的队列的 HTTP 请求。 此请求包含 HTTP 头 `x-msg-correlId`，用于设置 WebSphere MQ 消息的相关标识。

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

图 *210:* 队列的 *HTTP* ***POST*** 请求示例

第 860 页的图 211 显示了发送回客户机的响应。 没有任何响应内容。

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

图 *211: HTTP POST* 响应示例

HTTP DELETE 示例

HTTP **DELETE** 从队列中获取并删除消息，或者检索并删除发布。 **HTTPDELETE** Java 样本是从队列中读取消息的 HTTP **DELETE** 请求示例。 不使用 Java，您可以改用浏览器表单或 AJAX 工具包创建 HTTP **DELETE** 请求。

第 860 页的图 212 是用于删除队列中名为 myQueue 的下一条消息的 HTTP 请求。 作为响应，消息体将返回给客户机。 在 WebSphere MQ 术语中， HTTP **DELETE** 是破坏性获取。

该请求包含 HTTP 请求头 `x-msg-wait`，用于指示 WebSphere MQ Bridge for HTTP 等待消息到达队列的时间长度。 此请求还包含 `x-msg-require-headers` 请求头，用于指定客户机将在响应中收到消息相关标识。

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

图 *212: HTTP* ***DELETE*** 请求示例

第 860 页的图 213 是返回给客户机的响应。 相关标识将返回至客户机，如请求的 `x-msg-require-headers` 中所请求。

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

图 *213: HTTP* ***DELETE*** 响应示例

HTTP GET 示例

HTTP **GET** 从队列中获取消息。 该消息保留在队列中。 在 WebSphere MQ 术语中， HTTP **GET** 是浏览请求。 您可以使用 Java 客户机、浏览器表单或 AJAX 工具包创建 HTTP **GET** 请求。

第 861 页的图 214 是用于浏览名为 myQueue 的队列上的下一条消息的 HTTP 请求。

该请求包含 HTTP 请求头 x-msg-wait，用于指示 WebSphere MQ Bridge for HTTP 等待消息到达队列的时间长度。 此请求还包含 x-msg-require-headers 请求头，用于指定客户机将在响应中收到消息相关标识。

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

图 *214: HTTP **GET*** 请求示例

第 861 页的图 215 是返回到客户机的响应。 相关标识将返回至客户机，如请求的 x-msg-require-headers 中所请求。

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

图 *215: HTTP **GET*** 响应示例

安装，配置和验证 WebSphere MQ Bridge for HTTP

通过从 WebSphere MQ MQI 客户机或服务器安装材料安装 “Java 消息传递和 Web Service”，获取 WebSphere MQ Bridge for HTTP。 将 WebSphere MQ Bridge for HTTP 部署到合适的应用程序服务器。

开始之前

在 IBM WebSphere MQ 的系统需求 中查看必备产品。 安装过程不会检查用于运行 WebSphere MQ Bridge for HTTP 的必备软件是否存在和是否可用。 您必须确认必备软件已安装。

WebSphere MQ Bridge for HTTP 通过安装 WebSphere MQ 资源适配器在任何符合 Java EE 1.4 的应用程序服务器上运行。 您还可以在低于 V 6.0.2.1 的 WebSphere Application Server 发行版上运行 WebSphere MQ Bridge for HTTP。 使用 WebSphere Application Server 消息侦听器端口 (MLP) 将 WebSphere MQ 集成为 JMS 提供程序。

仅为以下应用程序服务器提供对 WebSphere MQ Bridge for HTTP 的支持:

- WebSphere Application Server 6.0.2.1 和更高版本。
- WebSphere Application Server Community Edition V 1.1 和更高版本。

关于此任务

WebSphere MQ Bridge for HTTP 作为 .war 文件 WMQHTTP.war 提供。

- 在 UNIX 平台和 Linux 上,
  - WMQHTTP.war 包含在 “Java 消息传递和 Web Service” 安装选项中。 在客户机和服务器安装材料中都有此选项。
  - WMQHTTP.war 安装到 *<mqmtop>*/java/http/WMQHTTP.war。 *<mqmtop>* 是 WebSphere MQ 的安装目录。
  - WMQHTTP.samples 安装到 *<mqmtop>*/java/http/samples。 *<mqmtop>* 是 WebSphere MQ 的安装目录。

执行以下安装步骤以安装 WebSphere MQ Bridge for HTTP，进行部署和配置，并验证配置。 配置步骤的详细信息会因不同的应用程序服务器而异。 将第 862 页的『在 WebSphere Application Server V6.1.0.9 上部署和验证 WebSphere MQ Bridge for HTTP』作为指导您在应用程序服务器上执行步骤的模板。

过程

1. 通过安装 WebSphere MQ MQI 客户机或服务器来获取 `WMQHTTP.war`。
2. 将 `WMQHTTP.war` 复制到某个服务器（可以通过该服务器将该文件部署到应用程序服务器）。
3. 将 `WMQHTTP.war` 部署到应用程序服务器。
4. 如有必要，请将 WebSphere MQ 作为资源适配器安装在应用程序服务器上。

   了解 WebSphere MQ 是否已配置为应用程序服务器上的消息传递提供程序。 使用应用程序服务器随附的管理工具来查找 WebSphere MQ。 可以在以下路径下找到 WebSphere MQ：**资源> JMS> 消息传递提供程序**。
5. 在应用程序服务器上配置连接工厂以连接到使用 WebSphere MQ MQI 客户机传输的队列管理器 [12]。
6. 在应用程序服务器上将 `WMQHTTP.war` Web 应用程序配置为使用连接工厂
7. 验证配置。

   a) 设置在连接工厂中和本地队列中命名的队列管理器。

   b) 在本地队列中放置消息。

   c) 使用对本地队列的读写权限，创建在连接工厂中命名的服务器连接通道。

   d) 启动队列管理器和侦听器。

   e) 启动应用程序服务器和 `WMQHTTP.war`（如果它们尚未运行）。

   f) 打开浏览器并输入 `http://`*`hostname`*`:`*`web port`*`/`*`Context root`*`/msg/queue/`*`local queue`*

结果

浏览器窗口将显示您放到本地队列上的消息。

下一步做什么

1. 尝试示例：第 862 页的『在 WebSphere Application Server V6.1.0.9 上部署和验证 WebSphere MQ Bridge for HTTP』。
2. 运行样本 HTTP Java 应用程序。

在 *WebSphere Application Server V6.1.0.9* 上部署和验证 *WebSphere MQ Bridge for HTTP*

使用以下示例为 HTTP 准备 WebSphere MQ 网桥的部署，以运行样本 HTTP Java 程序。 部署在 WebSphere Application Server V6.1.0.9 上。

开始之前

1. 遵循 第 861 页的『安装，配置和验证 WebSphere MQ Bridge for HTTP』中的指示信息，将 `WMQHTTP.war` 复制到 WebSphere Application Server 安装可访问的服务器上。
2. 配置一个队列管理器和一个队列，用于测试此配置：
   - 在此示例中，队列管理器配置为使用第 862 页的表 153 中的值：

表 *153:* 队列管理器配置	
Object	**值**
主机名	`itso-01`
队列管理器	`QM1`

[12] 最初，至少配置客户机传输。 某些应用程序服务器可以使用直接或绑定方式连接来连接到 WebSphere MQ。

表 153: 队列管理器配置 (继续)	
Object	**值**
本地队列	HTTPTESTQ
服务器连接通道	MYSVRCON。 配置具有足够权限来读写 HTTPTESTQ 的 MCA 用户标识。
侦听器端口	1414

3. 启动队列管理器和侦听器
4. 将测试消息放到 HTTPTESTQ 上。 例如：
   a. 启动 WebSphere MQ 资源管理器。
   b. 在 QM1 的本地队列列表中，右键单击 **HTTPTESTQ > Put test message > type `First Message` > Put message > Close**
5. 启动应用程序服务器，并登录到集成解决方案控制台。

关于此任务

此示例显示了将 WebSphere Application Server V6.1.0.9 作为应用程序服务器运行时要执行的步骤。 如果您正在运行其他版本的 WebSphere Application Server，或者正在运行其他应用程序服务器，那么步骤会有所不同。 WebSphere Application Server V6.1.0.9 预先配置了作为消息提供程序安装的 WebSphere MQ，使用 WebSphere MQ MQI 客户机库。 如果未将 WebSphere MQ 预先配置为消息传递提供程序，或者如果要使用 WebSphere MQ 服务器绑定，那么需要将 JEE 的 WebSphere MQ 资源适配器安装并配置到应用程序服务器中。

按照指示信息将 WebSphere MQ bridge for HTTP 部署到 WebSphere Application Server V6.1.0.9 上，并使用浏览器验证部署:

过程

1. 在导航窗格中，单击 **资源> JMS 提供程序> WebSphere MQ 消息传递提供程序**。

   您可以在 "节点"， "单元" 或 "服务器" 级别进行配置，具体取决于 WebSphere Application Server 部署。 此示例使用服务器级别部署。
2. 在“**其他属性**”下，单击**连接工厂 > 新建**。
3. 在 JMS 提供程序表单中，提供 第 863 页的表 154 中的信息或选择的替代方法，单击 **应用> 保存**。

表 154: 设置或修改以下字段	
字段	**值**
名称	WMQHTTPBridge
JNDI 名称	jms/WMQHTTPJCAConnectionFactory
队列管理器	QM1
主机	itso-01
端口	1414
通道	MYSVRCON
传输类型	CLIENT

4. 在导航窗格中，单击**应用程序 > 安装新应用程序**。
5. 将 WMQHTTP.war 的路径插入到表单中，并提供一个上下文根，然后单击**下一步**。
   a) 上下文根是可选项。 mq 是样本 HTTP 应用程序的缺省上下文根。
   b) 上下文根构成用于 HTTP 的标识 WebSphere MQ 网桥的 URI 的一部分。 您可以忽略上下文根，也可以稍后进行更改。

6. 在安装向导的“**选择安装选项**”页面上，无需更改任何缺省设置，直接单击**下一步**。
7. 在 " **将模块映射到服务器** " 页面上，选择集群或服务器，选中 "选择" 框，然后单击 **应用> 下一步**。
8. 在 " **将资源引用映射到资源** " 页面上的 **javax.jms.ConnectionFactory** 表单中，单击 **浏览 ...** 在 IBM WebSphere MQ Bridge for HTTP 行上。
9. 在“**企业应用程序 > 可用资源**”页面上，选择 **WMQHTTPBridge**，然后单击**应用**。
10. 返回到 **javax.jms.ConnectionFactory** 表单，选择认证方法。
    a) 对于此示例，请选择**无**，然后单击**应用**。 其他选项需要额外配置。
11. 选中 IBM WebSphere MQ Bridge for HTTP 的 **选择** 复选框，单击 **下一步> 下一步> 完成> 保存**
12. 在导航窗格中，单击**应用程序 > 企业应用程序**。
13. 选中 `WMQHTTP.war` 对应的选择框，然后单击**开始**。
14. 打开浏览器窗口。 使用相应的主机名和端口键入 `http://itso-01:9080/mq/msg/queue/HTTPTESTQ`。

结果

如果配置成功，浏览器窗口将显示 `First Message`。

下一步做什么

运行样本 HTTP Java 应用程序。

使用 WebSphere MQ Bridge for HTTP 发布/预订

WebSphere MQ Bridge for HTTP 使用 WebSphere MQ classes for JMS 发布/预订接口。 HTTP **POST** 会创建一个发布。 HTTP **DELETE** 会创建一个非持久性受管预订。 在使用主题 URI 之前，必须配置 JMS 的发布/预订。

发布/预订完全集成到 V 7 中的 WebSphere MQ 中。 在 V 7 之前，单独的发布/预订代理程序处理发布和预订。 它称为 “已排队” 发布/预订，以将其与版本 7 中完全集成的发布/预订区分开。 版本 7 模拟使用集成发布/预订的已排队发布预订。 枚举使排入队列的现有发布/预订应用程序能够与在同一队列管理器上运行的集成应用程序共存。 排入队列的发布/预订应用程序还可以与集成应用程序互操作，从而共享相同的主题。 在 V 6 中，代理随 WebSphere MQ 一起提供; 在 V 6 之前，代理作为 SupportPack 提供。

配置

WebSphere MQ Bridge for HTTP 使用 JMS 接口来发布和预订。 在 V 7 中，您可以使用 `PROVIDERVERSION` JMS 属性来控制 WebSphere MQ JMS 类是使用已排队还是集成发布/预订。

另一个注意事项是，您可以将 WebSphere MQ MQI 客户机库与 WebSphere MQ Bridge for HTTP 或服务器库配合使用。 版本 6 客户机库仅支持排队的发布/预订，而版本 7 库同时支持排队的发布/预订和集成的发布/预订。 将 WebSphere MQ 用作消息传递提供程序的大多数 Web 或应用程序服务器都使用客户机库来执行此操作。 为了使用集成发布/预订， WebSphere MQ MQI 客户机和服务器库都必须至少为 V 7。 如果其中一个运行的 WebSphere 版本低于 7 ，那么必须配置已排队的发布/预订; 请参阅 第 864 页的表 155。 查看您所使用的 Web 服务器或应用程序服务器安装或配置了哪些库。

表 *155:* 发布/预订配置模式

	客户机 V6 或更低版本	**客户机 V7 或更高版本**
服务器 V6 或更低版本	1. 运行 `\java\bin\MQJMS_PSQ.mqsc` 脚本	不支持

表 155: 发布/预订配置模式 (继续)		
	客户机 V6 或更低版本	**客户机 V7 或更高版本**
服务器 V7 或更高版本	1. 运行 \java\bin\MQJMS_PSQ.mqsc 脚本 2. 将队列管理器设置为 PSMODE=ENABLED	1. 如果 PROVIDERVERSION 为 7 a. 将队列管理器设置为 PSMODE=ENABLED 或 PSMODE=COMPAT 2. 如果 PROVIDERVERSION 为 6 a. 将队列管理器设置为 PSMODE=ENABLED

发布

使用以下 URI 发送一个 HTTP **POST** 请求：

```
http://hostname:port/context_root/msg/topic/topicString
```

将使用主题字符串 *topicString* 发布消息内容。

预订

使用以下 URI 发送一个 HTTP **DELETE** 请求：

```
http://hostname:port/context_root/msg/topic/topicString
```

WebSphere MQ Bridge for HTTP 创建对主题字符串 *topicString* 的受管非持久预订。 返回发布后，或者在定制实体头 `x-msg-wait` 设置的等待间隔到期后，会立即删除此订阅。

针对 HTTP 样本运行 WebSphere MQ 网桥

WebSphere MQ Bridge for HTTP 样本仅在 Windows 操作系统上可用。 样本显示如何将 HTTP **POST** 和 HTTP **DELETE** 命令提交到 WebSphere MQ bridge for HTTP from Java 程序。

开始之前

通过在 第 861 页的『安装，配置和验证 WebSphere MQ Bridge for HTTP』中运行步骤 第 862 页的『7』，验证用于 HTTP 安装的 WebSphere MQ 网桥。

HTTP 样本安装到第 865 页的表 156 中所示的目录中。 在每种情况下，源代码都安装到 `/src` 子目录中。

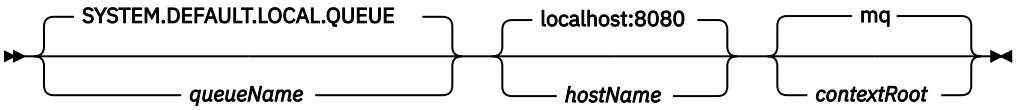
表 156: HTTP 样本的位置	
平台	**位置**
Windows	*MQ_INSTALLATION_PATH*/tools/http/samples
所有其他平台	*MQ_INSTALLATION_PATH*/samp/http
MQ_INSTALLATION_PATH 表示 WebSphere MQ 的安装目录。	

关于此任务

这些样本模拟 WebSphere MQ AMQSPUT 和 AMQSGET 样本应用程序。 它们展示了点到点消息传递环境中的以下功能：

- **HTTPPOST** -在 Java 应用程序中发送 HTTP **POST** 请求，以使用 WebSphere MQ Bridge for HTTP 将消息放入 WebSphere MQ 队列并处理响应。
- **HTTPDELETE** -使用 WebSphere MQ 网桥 for HTTP 在 Java 应用程序中发送 HTTP **DELETE** 请求以从 WebSphere MQ 队列获取消息，并处理包含 WebSphere MQ 消息的响应。

HTTPPOST 和 HTTPDELETE 的参数



要运行 **HTTPPOST** 样本，请完成以下步骤：

过程

1. 在命令窗口中，导航到 HTTP 样本所在的目录。
2. 运行 **HTTPPOST** 样本。

   ```
   java -classpath . HTTPPOST [parameters]
   ```

   当 **HTTPPOST** 样本启动时，将显示以下输出：

   ```
   HTTP POST Sample start
   Target server is 'hostName'
   Target queue is 'queueName'
   Target context-root is 'contextRoot'
   ```

3. 在命令提示符下，键入您希望构成消息正文的文本。
4. 按 Enter 键以将消息发布到 WebSphere MQ 队列。
   a) 如果您希望发送另一个消息，请再输入一些文本。
   这些文本构成第二条 WebSphere MQ 消息的主体。
   b) 按 Enter 键以将消息发布到 WebSphere MQ 队列。
5. 按两次 Enter 以结束 **HTTPPOST**。
   将显示以下输出：

   ```
   HTTP POST Sample end
   ```

下一步做什么

HTTPDELETE 样本对您放置在 WebSphere MQ 队列上的所有消息执行破坏性获取。

通过完成以下步骤来运行 **HTTPDELETE** 样本：

1. 在命令窗口中，导航到 *MQ_INSTALLATION_PATH*/tools/samples。*MQ_INSTALLATION_PATH* 表示 WebSphere MQ 的安装目录。
2. 运行 **HTTPDELETE** 样本。

   ```
   java -classpath . HTTPPOST [parameters]
   ```

   当 **HTTPDELETE** 样本启动时，将显示以下输出：

   ```
   HTTP DELETE Sample start
   Target server is 'host:port'
   Target queue is 'your queue name'
   Target context-root is 'your context-root'
   message
   message
   ...
   ```

WebSphere Bridge for HTTP 的安全注意事项

对 Web 浏览器客户机进行认证时，标准 Web 安全性注意事项适用。 对 WebSphere MQ 资源的授权属于运行 WebSphere Bridge for HTTP servlet 的用户级别，而不是单个 Web 浏览器客户机级别。 标准 WebSphere MQ 安全性注意事项适用于 WebSphere MQ。

使用 WebSphere Bridge for HTTP 从 Web 浏览器流至 WebSphere MQ 应用程序并返回的数据将执行三个步骤:

客户机连接
通过使用 HTTP 的 TCP/IP 连接从浏览器到 WebSphere Bridge for HTTP。

与 WebSphere MQ 的资源适配器连接
连接从 WebSphere Bridge for HTTP 到 WebSphere MQ 队列管理器。 该连接是基于 TCP/IP 的客户机连接或本地 WebSphere MQ 绑定连接。 建立此连接后，系统会将 HTTP 请求置于标准本地队列或传输队列中。

通过一个或多个通道从 WebSphere MQ 本地队列到目标队列。
使用标准技术来保护队列、主题、队列管理器和通道。

回复过程将反向执行这三个步骤。

客户机连接

在 HTTP 客户机与使用 Web 容器的应用程序服务器之间建立安全连接。 使用标准 HTTP 服务器技术，例如，使用 HTTPS。 请参阅您的应用程序服务器对应的文档，以获取相关信息。

与 WebSphere MQ 的资源适配器连接

仅授权使用单一用户标识在资源适配器与队列管理器之间建立连接。 分配单个用户标识以标识来自 WebSphere Bridge for HTTP 的请求。 用户标识必须具有受限的 WebSphere MQ 权限，只有外部用户才能访问这些资源。 您必须单独对实际客户机进行认证，并使用标准 Web 安全性技术来建立信任，以便与该客户机进行持续交互。

使用单一用户标识来保护资源适配器与队列管理器之间的连接。 对该用户标识的权限进行限制，使其只能在队列和主题中读写消息。 WebSphere Bridge for HTTP 是因特网与内部网之间的攻击点。

如何保护资源适配器与 WebSphere MQ 之间的连接取决于特定资源适配器。 请参阅 资源适配器的 文档。

使用组件对象模型接口 (WebSphere MQ Automation Classes for ActiveX)

WebSphere MQ Automation Classes for ActiveX (MQAX) 是 ActiveX 组件，用于提供可在应用程序中用于访问 WebSphere MQ 的类。

MQAX 需要 WebSphere MQ 环境以及要与之进行通信的相应 WebSphere MQ 应用程序。

它使 ActiveX 应用程序能够在可通过 WebSphere MQ 访问的任何企业系统上运行事务和访问数据。

WebSphere MQ Automation Classes for ActiveX:

- 授予您对 WebSphere MQ API 的功能和功能的访问权，从而允许与其他 WebSphere MQ 平台进行完全互连。
- 符合对 ActiveX 组件期待的一般约定。
- 符合 WebSphere MQ 对象模型，也可用于 .NET， C++， Java 和 LotusScript。

提供了 MQAX 启动器样本。 最初可以使用这些样本来检查 MQAX 的安装是否成功，以及您是否具有基本的 WebSphere 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，用于在 Microsoft Windows 平台上集成应用程序开发，可复用组件和因特网技术。 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。

WebSphere MQ 环境支持

WebSphere MQ Automation Classes for ActiveX 只能与 **32 位** ActiveX 脚本编制客户机配合使用。

COM 组件只能用于 **32 位**应用程序。 如果要编写 64 位 COM 应用程序，可以使用 .NET 接口。

要在 WebSphere MQ 服务器环境中运行 MQAX，必须在系统上安装 Windows 2000 或更高版本。

要在 WebSphere MQ MQI 客户机环境中运行 MQAX，您需要在系统上的 Windows 2000 或更高版本上安装 WebSphere MQ MQI 客户机:

WebSphere MQ MQI 客户机需要访问至少一个 WebSphere MQ 服务器。 当系统上安装了 WebSphere MQ MQI 客户机和 WebSphere MQ 服务器时， MQAX 应用程序始终针对该服务器运行。 MQAI 的 ActiveX 接口仅在 WebSphere MQ 服务器环境中可用。

使用 WebSphere 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

WebSphere MQ 自动化类提供对 WebSphere MQ API 的功能的访问。 因此，您可以从使用 WebSphere MQ 可以为 Windows 应用程序带来的所有优势中获益。

应用程序的总体设计与任何 WebSphere MQ 应用程序相同，因此请考虑 第 7 页的『开发应用程序』 部分中描述的所有设计方面。

要使用 WebSphere MQ 自动化类，请使用支持创建和使用 COM 对象的语言对应用程序中的 Windows 程序进行编码。 例如， Visual Basic， Java 和其他 ActiveX 脚本编制客户机。 然后，可以将这些类轻松集成到应用程序中，因为您需要的 WebSphere MQ 对象可以使用实现语言的本机语法进行编码。

使用 WebSphere MQ Automation Classes for ActiveX

设计使用 WebSphere MQ Automation Classes for ActiveX 的 ActiveX 应用程序时，最重要的信息项是从远程 WebSphere 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。 对于工作的 MQAX 脚本，它和选取或发送消息的 WebSphere MQ 应用程序都必须知道消息结构。

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。

在考虑如何构造设计的系统实现时，请记住 MQAX 脚本与安装了 WebSphere MQ 队列管理器或 WebSphere MQ 客户机的脚本在同一机器上运行。

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 WebSphere MQ 应用程序是消息的发起方，并且 WebSphere MQ 生成这些属性，那么最好使用 AccountingTokenHex， CorrelationIdHex， GroupIdHex 和 MessageIdHex 属性 (如果要查看它们的值或

以任何方式处理它们)， 包括将它们传递回消息中的 WebSphere MQ。 原因是 WebSphere MQ 生成的值是具有从 0 到 255 (含) 的任何值的字节字符串，它们不是可打印字符的字符串。

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。

WebSphere MQ 常量

WebSphere MQ 常量作为库 MQAX200 中枚举 WebSphere MQ 的成员提供。

WebSphere MQ 字符串常量

WebSphere MQ 字符串常量及其对应的字符串。

当使用 WebSphere MQ Automation Classes for ActiveX 时， WebSphere 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:

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_STRIsNG	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "
MQFMT_REF_MSG_HEADER	"MQHREF "
MQFMT_RF_HEADER	"MQHRF "
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	"MQXMIT "

空字符串常量

WebSphere MQ Automation Classes for ActiveX 不支持用于初始化四个 MQMessage 属性 (MQMI_NONE (24 个 NULL 字符)， MQCI_NONE (24 个 NULL 字符)， MQGI_NONE (24 个 NULL 字符) 和 MQACT_NONE (32 个 NULL 字符) 的 WebSphere MQ 常量。 将它们设置为空字符串也有相同效果。

例如，要将 MQMessage 的各种标识设置为以下值: *mymessage*。**MessageId** = "" *mymessage*。**CorrelationId** = "" *mymessage*。**AccountingToken** = ""

从 WebSphere MQ 接收消息

有多种方法可从 WebSphere MQ 接收消息:

- 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。 但是，如果其他线程需要访问 WebSphere MQ，那么它们需要使用其他 MQAX 队列管理器和队列对象与 WebSphere MQ 进行第二次连接。

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

WebSphere 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

可以使用提供的 WebSphere MQ 常量来设置和解释 "编码" 属性。 第 871 页的图 216 显示以下示例:

```
/* 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 *216:* 提供了用于编码的 *WebSphere MQ* 常量

例如，要将整数从 Intel 系统发送到 System/390 编码的 System/390 操作系统:

```
Dim msg As New MQMessage   'Define a WebSphere 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）。

WebSphere 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 a WebSphere MQ message
msg.CharacterSet = 437                'Set code page required
msg.WriteString "A character string"'Put character string in message
```

WriteString 方法接收字符串数据 (在示例中为 "A character string") 作为 Unicode 字符串。 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，当您使用 ReadString 方法时，入局消息具有由 WebSphere MQ 消息描述符 (MQMD) 值建立的字符集，并且在将此代码页传递回脚本语言之前，会将其转换为 Unicode。

线程化

WebSphere MQ Automation Classes for ActiveX 实现了一个自由线程模型，可以在线程之间使用对象。

虽然 MQAX 允许使用 MQQueue 和 MQQueueManager 对象，但 WebSphere 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这些内容会导致错误， WebSphere MQ 会返回 MQRC_HCONN_ERROR 返回码。

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 871 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 属性返回 ReasonCode 的当前值的 WebSphere 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 类具有方法 *ReasonCodeName* ，该方法可用于将 WebSphere 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。

WebSphere MQ Automation Classes for ActiveX 参考

本节描述为 ActiveX 开发的 WebSphere MQ Automation Classes for ActiveX (MQAX) 的类。 这些类使您能够编写 ActiveX 应用程序，这些应用程序可以使用 WebSphere MQ 来访问在非 ActiveX 环境中运行的其他应用程序。

WebSphere MQ Automation Classes for ActiveX 接口

WebSphere MQ Automation Classes for ActiveX 提供了使用类所需的预定义数字 ActiveX 常量 (例如 MQMT_REQUEST)。

ActiveX 自动化类由下列类组成：

- 第 874 页的『MQSession 类』
- 第 877 页的『MQQueueManager 类』
- 第 887 页的『MQQueue 类』
- 第 902 页的『MQMessage 类』
- 第 921 页的『MQPutMessageOptions 类』
- 第 924 页的『MQGetMessageOptions 类』
- 第 926 页的『MQDistributionList 类』
- 第 930 页的『MQDistributionListItem 类』

此外， WebSphere MQ Automation Classes for ActiveX 提供了使用这些类所需的预定义数字 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 WebSphere MQ C 头文件 (cmqc * .h) 中定义的那些常量的子集，其中包含一些额外的 WebSphere MQ Automation Classes for ActiveX 原因码。

关于 WebSphere 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。

请参阅 在 Windows 上只能与主安装配合使用的功能部件 以获取重要信息。

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 872 页的『错误处理』以获取更多信息。

MQQueueManager 和 MQQueue 类提供对底层 WebSphere MQ 对象的访问。 这些类的方法或属性访问通常会导致在 WebSphere MQ MQI 上进行调用。

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 WebSphere MQ 调用相关。 以下每种方法都会创建一个对象，然后在该对象中保存参考信息，然后连接或打开 WebSphere MQ 对象:

当连接到队列管理器时，它保存由 WebSphere MQ 生成的 "连接句柄" 属性。

当队列打开时，它保存由 WebSphere MQ 生成的 "对象句柄" 属性。

这些 WebSphere MQ 属性不可用于 MQAX 程序。

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。

WebSphere MQ ActiveX 类都包含两个特殊只读属性- 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发出任何方法调用可能会从 WebSphere 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 WebSphere MQ Automation Classes for ActiveX 的根类。

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 875 页的『CompletionCode 属性』.
- 第 875 页的『ExceptionThreshold 属性』.
- 第 875 页的『ReasonCode 属性』.
- 第 875 页的『ReasonName 属性』.

方法

- 第 876 页的『AccessGetMessageOptions 方法』.
- 第 876 页的『AccessMessage 方法』.
- 第 876 页的『AccessPutMessageOptions 方法』.
- 第 876 页的『AccessQueueManager 方法』.
- 第 876 页的『ClearErrorCodes 方法』.

- 第 876 页的『ReasonCodeName 方法』.

CompletionCode 属性

只读。 返回由针对任何 WebSphere MQ 对象发出的最新方法或属性集设置的 WebSphere MQ 完成代码集。

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 871 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：

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法：

要获取: *completioncode& = MQSession*。CompletionCode

ExceptionThreshold 属性

读 — 写。 定义 MQAX 将抛出异常的 WebSphere 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**

要设置: *MQSession*.**ExceptionThreshold** = *ExceptionThreshold$*

ReasonCode 属性

只读。 返回针对任何 WebSphere MQ 对象发出的最新方法或属性集所设置的原因码。

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 871 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 936 页的『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 871 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 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

创建新的 MQQueueManager 对象，并通过 WebSphere MQ MQI 客户机或 WebSphere MQ 服务器将其连接到实际队列管理器。 除了执行连接外，此方法还为队列管理器对象执行打开。

当系统上安装了 WebSphere MQ MQI 客户机和 WebSphere MQ 服务器时，缺省情况下， MQAX 应用程序将针对该服务器运行。 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。 如果未设置此变量，那么 WebSphere MQ 会尝试装入 amqzst.dll。 如果此 DLL 不存在 (仅适用于客户机安装) ，那么 WebSphere MQ 会尝试装入 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: 调用 *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。 该名称仍有些隐秘 (例如，ReasonCodeName (2059) 为 **MQRC_Q_MGR_NOT_AVAILABLE**)，因此应该捕获可能的错误并将其替换为适合于应用程序的描述性文本。

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。 队列管理器可以在本地 (WebSphere MQ 服务器) 运行，也可以远程运行，由 WebSphere MQ 客户机提供访问权。 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 879 页的『AlternateUserId 属性』.
- 第 879 页的『AuthorityEvent 属性』.
- 第 879 页的『BeginOptions 属性』.
- 第 879 页的『ChannelAutoDefinition 属性』.
- 第 879 页的『ChannelAutoDefinitionEvent 属性』.
- 第 880 页的『ChannelAutoDefinitionExit 属性』.
- 第 880 页的『CharacterSet 属性』.
- 第 880 页的『CloseOptions 属性』.
- 第 880 页的『CommandInputQueueName 属性』.
- 第 880 页的『CommandLevel 属性』.
- 第 881 页的『CompletionCode 属性』.
- 第 881 页的『ConnectionHandle 属性』.
- 第 881 页的『ConnectionStatus 属性』.
- 第 881 页的『ConnectOptions 属性』.
- 第 881 页的『DeadLetterQueueName 属性』.
- 第 882 页的『DefaultTransmissionQueueName 属性』.
- 第 882 页的『Description 属性』.
- 第 882 页的『DistributionLists 属性』.
- 第 882 页的『InhibitEvent 属性』.

- 第 882 页的『IsConnected 属性』.
- 第 883 页的『IsOpen 属性』.
- 第 883 页的『LocalEvent 属性』.
- 第 883 页的『MaximumHandles 属性』.
- 第 883 页的『MaximumMessageLength 属性』.
- 第 883 页的『MaximumPriority 属性』.
- 第 883 页的『MaximumUncommittedMessages 属性』.
- 第 884 页的『Name 属性』.
- 第 884 页的『ObjectHandle 属性』.
- 第 884 页的『PerformanceEvent 属性』.
- 第 884 页的『Platform 属性』.
- 第 884 页的『ReasonCode 属性』.
- 第 885 页的『ReasonName 属性』.
- 第 885 页的『RemoteEvent 属性』.
- 第 885 页的『StartStopEvent 属性』.
- 第 885 页的『SyncPointAvailability 属性』.
- 第 885 页的『TriggerInterval 属性』.

方法

- 第 886 页的『AccessQueue 方法』.
- 第 886 页的『AddDistributionList 方法』.
- 第 886 页的『Backout 方法』.
- 第 886 页的『Begin 方法』.
- 第 887 页的『ClearErrorCodes 方法』.
- 第 887 页的『Commit 方法』.
- 第 887 页的『Connect 方法』.
- 第 887 页的『Disconnect 方法』.

属性访问

以下属性可随时访问。

- 第 879 页的『AlternateUserId 属性』.
- 第 881 页的『CompletionCode 属性』.
- 第 881 页的『ConnectionStatus 属性』.
- 第 884 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，那么 WebSphere MQ Automation Classes for ActiveX 会尝试连接到队列管理器并自动打开该队列管理器以进行查询。 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*。**ChannelAuto 定义**

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 WebSphere MQ 队列管理器实现的版本和级别 (MQI CommandLevel 属性)

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。 WebSphere MQ 队列管理器对象的连接句柄。

定义于:
MQQueueManager 类

数据类型:
长

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*。**描述**

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**

要设置: *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。 WebSphere MQ 队列管理器对象的对象句柄。

定义于：
MQQueueManager 类

数据类型
长

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 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 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 186 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。 WebSphere 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: 调用 *MQQueueManager*。**落实 ()**

Connect 方法

通过 WebSphere MQ MQI 客户机或服务器将 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: 调用 *MQQueueManager*。**连接 ()**

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: 调用 *MQQueueManager*。**断开连接 ()**

MQQueue 类

此类表示对 WebSphere MQ 队列的访问权。 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 890 页的『AlternateUserId 属性』.
- 第 890 页的『BackoutRequeueName 属性』.
- 第 890 页的『BackoutThreshold 属性』.
- 第 890 页的『BaseQueueName 属性』.
- 第 891 页的『CloseOptions 属性』.
- 第 891 页的『CompletionCode 属性』.
- 第 891 页的『ConnectionReference 属性』.
- 第 891 页的『CreationDateTime 属性』.
- 第 892 页的『CurrentDepth 属性』.
- 第 892 页的『DefaultInputOpenOption 属性』.
- 第 892 页的『DefaultPersistence 属性』.
- 第 892 页的『DefaultPriority 属性』.
- 第 892 页的『DefinitionType 属性』.
- 第 892 页的『DepthHighEvent 属性』.
- 第 893 页的『DepthHighLimit 属性』.
- 第 893 页的『DepthLowEvent 属性』.
- 第 893 页的『DepthLowLimit 属性』.
- 第 893 页的『DepthMaximumEvent 属性』.
- 第 892 页的『DepthHighEvent 属性』.
- 第 893 页的『DepthHighLimit 属性』.
- 第 893 页的『DepthLowEvent 属性』.
- 第 893 页的『DepthLowLimit 属性』.
- 第 893 页的『DepthMaximumEvent 属性』.
- 第 893 页的『Description 属性』.
- 第 893 页的『DynamicQueueName 属性』.
- 第 894 页的『HardenGetBackout 属性』.
- 第 894 页的『InhibitGet 属性』.
- 第 894 页的『InhibitPut 属性』.
- 第 894 页的『InitiationQueueName 属性』.
- 第 895 页的『IsOpen 属性』.
- 第 895 页的『MaximumDepth 属性』.
- 第 895 页的『MaximumMessageLength 属性』.
- 第 895 页的『MessageDeliverySequence 属性』.
- 第 896 页的『ObjectHandle 属性』.
- 第 896 页的『OpenInputCount 属性』.
- 第 896 页的『OpenOptions 属性』.
- 第 896 页的『OpenOutputCount 属性』.
- 第 896 页的『OpenStatus 属性』.
- 第 896 页的『ProcessName 属性』.

- 第 897 页的『QueueManagerName 属性』.
- 第 897 页的『QueueType 属性』.
- 第 897 页的『ReasonCode 属性』.
- 第 897 页的『ReasonName 属性』.
- 第 897 页的『RemoteQueueManagerName 属性』.
- 第 897 页的『RemoteQueueName 属性』.
- 第 898 页的『ResolvedQueueManagerName 属性』.
- 第 898 页的『ResolvedQueueName 属性』.
- 第 898 页的『RetentionInterval 属性』.
- 第 898 页的『Scope 属性』.
- 第 898 页的『ServiceInterval 属性』.
- 第 898 页的『ServiceIntervalEvent 属性』.
- 第 899 页的『Shareability 属性』.
- 第 899 页的『TransmissionQueueName 属性』.
- 第 899 页的『TriggerControl 属性』.
- 第 899 页的『TriggerData 属性』.
- 第 899 页的『TriggerDepth 属性』.
- 第 900 页的『TriggerMessagePriority 属性』.
- 第 900 页的『TriggerType 属性』.
- 第 900 页的『Usage 属性』.

方法

- 第 900 页的『ClearErrorCodes 方法』
- 第 900 页的『Close 方法』
- 第 900 页的『Get 方法』
- 第 901 页的『Open 方法』
- 第 901 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 891 页的『CompletionCode 属性』
- 第 896 页的『OpenStatus 属性』
- 第 897 页的『ReasonCode 属性』

并且您可以读和写：

- 第 890 页的『AlternateUserId 属性』
- 第 891 页的『CloseOptions 属性』
- 第 891 页的『ConnectionReference 属性』
- 第 895 页的『Name 属性』
- 第 896 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

未在上一节中列出的属性是底层 WebSphere MQ 队列的所有属性。 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该用户标识，那么将改为检查备用用户标识以进行授权。

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。 WebSphere MQ 在打开队列过程中针对用户授权验证 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**

要设置: *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 WebSphere MQ 队列管理器对象的引用

语法: 要设置: *set MQQueue*。**ConnectionReference** = *ConnectionReference*

要获取: *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.***DepthHigh 事件**

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.***DepthHigh 限制**

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.***DepthLow 事件**

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.***DepthLow 限制**

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.***DepthMaximum 事件**

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*。**描述**

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 WebSphere MQ 队列名称。

语法： 要设置: *MQQueue*。**DynamicQueueName** = *dynamicqueuename $*

要获取: *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**

要设置: *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。 WebSphere 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: 要获取: *openincount& = MQQueue*.**OpenInput 计数**

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: 要获取: *openopt& = MQQueue*.**OpenOptions**

要设置: *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: 要获取: *openoutcount& = MQQueue*.**OpenOutput 计数**

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: 要获取: *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。 WebSphere 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**

要设置: *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 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 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.***ServiceInterval 事件**

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**

要设置: *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: 调用 *MQQueue*。**关闭 ()**

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 WebSphere MQ 消息的长度
- **DataLength** 设置为 WebSphere MQ 消息的长度
- **DataOffset** 设置为零

定义于：
MQQueue 类

语法： 调用 *MQQueue*。**获取**(消息， *GetMsg* 选项， *GetMsg* 长度)

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，用于控制从队列中检索的 WebSphere 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： 调用 *MQQueue*。**打开 ()**

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。 它们在此方法运行后立即具有的值是放在 WebSphere MQ 上的值。

Put 完成后对 MQMessage 对象的修改不会影响 WebSphere MQ 队列上的实际消息。

定义于：
MQQueue 类

语法: 调用 *MQQueue*。**放置**(消息， *PutMsg* 选项)

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

此类表示 WebSphere MQ 消息。 它包含用于封装 WebSphere MQ 消息描述符 (MQMD) 的属性，并提供用于保存应用程序定义的消息数据的缓冲区。

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。

如果缓冲区大小超过 WebSphere MQ 队列的 MaximumMessage 长度属性，那么不能将消息放入该队列。

构造 MQMessage 对象后，可以使用 MQQueue.Put 方法将其放入 WebSphere MQ 队列。 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 之后修改或删除 MQMessage 对象，而不会影响 WebSphere MQ 队列上的消息。 在 WebSphere MQ 队列上复制消息时，队列管理器可以调整 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 or Set msg = New MQMessage
```

属性

控制属性为:

- 第 904 页的『CompletionCode 属性』
- 第 905 页的『DataLength 属性』
- 第 905 页的『DataOffset 属性』
- 第 905 页的『MessageLength 属性』
- 第 905 页的『ReasonCode 属性』
- 第 906 页的『ReasonName 属性』

消息描述符属性为:

- 第 906 页的『AccountingToken 属性』

- 第 906 页的『AccountingTokenHex 属性』
- 第 906 页的『ApplicationIdData 属性』
- 第 907 页的『ApplicationOriginData 属性』
- 第 907 页的『BackoutCount 属性』
- 第 907 页的『CharacterSet 属性』
- 第 907 页的『CorrelationId 属性』
- 第 908 页的『CorrelationIdHex 属性』
- 第 908 页的『Encoding 属性』
- 第 909 页的『Expiry 属性』
- 第 909 页的『Feedback 属性』
- 第 909 页的『Format 属性』
- 第 909 页的『GroupId 属性』
- 第 910 页的『GroupIdHex 属性』
- 第 910 页的『MessageData 属性』
- 第 910 页的『MessageFlags 属性』
- 第 910 页的『MessageId 属性』
- 第 911 页的『MessageIdHex 属性』
- 第 911 页的『MessageSequenceNumber 属性』
- 第 911 页的『MessageType 属性』
- 第 911 页的『Offset 属性』
- 第 912 页的『OriginalLength 属性』
- 第 912 页的『Persistence 属性』
- 第 912 页的『Priority 属性』
- 第 912 页的『PutApplicationName 属性』
- 第 912 页的『PutApplicationType 属性』
- 第 913 页的『PutDateTime 属性』
- 第 913 页的『ReplyToQueueManagerName 属性』
- 第 913 页的『ReplyToQueueName 属性』
- 第 913 页的『Report 属性』
- 第 914 页的『TotalMessageLength 属性』
- 第 914 页的『UserId 属性』

方法

- 第 914 页的『ClearErrorCodes 方法』
- 第 914 页的『ClearMessage 方法』
- 第 914 页的『Read 方法』
- 第 914 页的『ReadBoolean 方法』
- 第 915 页的『ReadByte 方法』
- 第 915 页的『ReadDecimal2 方法』
- 第 915 页的『ReadDecimal4 方法』
- 第 915 页的『ReadDouble 方法』
- 第 915 页的『ReadDouble4 方法』
- 第 916 页的『ReadFloat 方法』

- 第 916 页的『ReadInt2 方法』
- 第 916 页的『ReadInt4 方法』
- 第 916 页的『ReadLong 方法』
- 第 916 页的『ReadNullTerminatedString 方法』
- 第 916 页的『ReadShort 方法』
- 第 917 页的『ReadString 方法』
- 第 917 页的『ReadUInt2 方法』
- 第 917 页的『ReadUnsignedByte 方法』
- 第 917 页的『ReadUTF 方法』
- 第 917 页的『ResizeBuffer 方法』
- 第 918 页的『Write 方法』
- 第 918 页的『WriteBoolean 方法』
- 第 918 页的『WriteByte 方法』
- 第 918 页的『WriteDecimal2 方法』
- 第 919 页的『WriteDecimal4 方法』
- 第 919 页的『WriteDouble 方法』
- 第 919 页的『WriteDouble4 方法』
- 第 919 页的『WriteFloat 方法』
- 第 920 页的『WriteInt2 方法』
- 第 920 页的『WriteInt4 方法』
- 第 920 页的『WriteLong 方法』
- 第 920 页的『WriteNullTerminatedString 方法』
- 第 920 页的『WriteShort 方法』
- 第 920 页的『WriteString 方法』
- 第 921 页的『WriteUInt2 方法』
- 第 921 页的『WriteUnsignedByte 方法』
- 第 921 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，将消息放入 WebSphere MQ 队列时，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

可以将二进制数据传递到 WebSphere MQ 消息，方法是将 CharacterSet 属性设置为队列管理器的编码字符集标识 (MQCCSI_Q_MGR)，并向其传递字符串。 您可以使用 chr $函数将非字符数据设置到字符串中。

Read 和 Write 方法执行数据转换。 它们在 ActiveX 内部格式与 WebSphere MQ 消息格式之间进行转换，如消息描述符中的编码和 CharacterSet 属性所定义。 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。 返回针对此对象发出的最新方法或属性访问设置的 WebSphere 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 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 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**

要设置: *MQMessage.***AccountingToken** = *actoken $*

请参阅第 868 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**

要设置: *MQMessage.***AccountingTokenHex** = *actokenh $*

请参阅第 868 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*。**ApplicationId 数据**

要设置: *MQMessage*。**ApplicationId 数据** = *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*。**ApplicationOrigin 数据**

要设置: *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。

其初始值为特殊值 **MQCCSI_Q_MGR** 。

如果 CharacterSet 设置为 **MQCCSI_Q_MGR**，那么当前语言环境的代码页将用于 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 868 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**

要设置: *MQMessage*。**CorrelationIdHex** = *correlidh $*

请参阅第 868 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*。**格式**

要设置: *MQMessage*。**格式** = 格式 *$*

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**

要设置: *MQMessage*。**GroupId** = *groupid $*

请参阅第 868 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**

要设置: *MQMessage*。**GroupId 十六进制** = *groupidh $*

请参阅第 868 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**

要设置: *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**

要设置: *MQ* 消息。**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**

要设置: *MQMessage*。**MessageId** = *messageid $*

请参阅第 868 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**

要设置: *MQMessage*。**MessageIdHex** = *messageidh $*

请参阅第 868 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**

要设置: *MQ* 消息。**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*。**偏移量**

要设置: *MQ* 消息。**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**

要设置: *MQ* 消息。**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*。**PutApplication 名称**

要设置: *MQMessage*。**PutApplication 名称** = *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 WebSphere MQ MQMD 中使用的日期和时间格式之间进行转换。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**

要设置: *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**

要设置: *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**

要设置: *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**

要设置: *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: 调用 *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

此方法从 "消息数据" 缓冲区中读取 1 字节 (从 DataOffset 所引用的字符开始)，并将其返回为 -128 到 127 范围内的整数 (带符号的 2 字节) 整数值。

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: 调用 *value& = MQMessage*。**读决定 4**

ReadDouble 方法

此方法从消息数据缓冲区读取 8 字节 (从 DataOffset 引用的字节开始)，并将其作为 Double (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 4 字节 System/390 浮点消息值，这些值太大，无法转换为 4 字节 IEEE 浮点格式。

此方法从消息数据缓冲区读取 4 字节 (从 DataOffset 引用的字节开始)，并将其作为 Double (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

此方法从消息数据缓冲区读取 4 字节 (从 DataOffset 引用的字节开始)，并将其作为单个 (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*。**读取国际 2**

ReadInt4 方法

此方法等同于 ReadLong 方法。

语法: *bigint& = MQMessage*.**ReadInt4**

ReadLong 方法

此方法从 "消息数据" 缓冲区读取 4 字节 (从 DataOffset 引用的字节开始)，并将其作为长整型 (带符号的 4 字节) 整数值返回。

如果发出此方法时 MQMessage.DataLength 小于 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: 字符串 *$ = MQ* 消息.**ReadNullTerminatedString(length&)**

参数：
length& 长长的 字符串字段的长度，单位是字节。

ReadShort 方法

此方法从 "消息数据" 缓冲区读取 2 字节 (从 DataOffset 引用的字节开始)，并将其作为整数 (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

此方法从 "消息数据" 缓冲区读取 2 字节 (从 DataOffset 引用的字节开始) ，并将其作为长整型 (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

此方法从 "消息数据" 缓冲区读取 1 字节 (从 DataOffset 引用的字节开始) ，并将其作为整数 (带符号的 2 字节) 整数值返回，范围为 0 到 255。

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

此方法从以 DataOffset 引用的字节开始的消息中读取 UTF 格式字符串，并将其作为 ActiveX 字符串返回。 消息中的字符串由后跟字符数据的 2 字节长度组成。

如果 MQMessage.DataLength 小于发出该方法时的字符串长度，那么该方法将失败。

DataOffset 按字符串长度递增，如果方法成功，那么 DataLength 按字符串长度递减。

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: 调用 *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: 调用 *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： 调用 *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 *MQ* 消息.**已写入 Decimal4**(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 915 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： 调用 *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: 调用 *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: 调用 *MQMessage*。**WriteInt4**(值和)

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: 调用 *MQ* 消息**.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: 调用 *MQ* 消息。**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: 调用 *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: 调用 *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: 调用 *MQ* 消息**.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**(value$)

参数：
value$ 字符串。 要写的值。

MQPutMessageOptions 类

此类封装用于控制将消息放入 WebSphere MQ 队列的操作的各种选项。

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 922 页的『CompletionCode 属性』.
- 第 922 页的『Options 属性』.
- 第 922 页的『ReasonCode 属性』.
- 第 923 页的『ReasonName 属性』.
- 第 923 页的『RecordFields 属性』.
- 第 923 页的『ResolvedQueueManagerName 属性』.
- 第 923 页的『ResolvedQueueName 属性』.

方法

- 第 923 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 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 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 标志。 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。 请参阅 *WebSphere MQ Application Programming Reference* 以获取更多详细信息。

定义于：
MQPutMessageOptions 类

数据类型：
长

语法： 要获取: *recordfields& = PutOpts*.**RecordFields**

要设置: *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

此类封装了用于控制从 WebSphere MQ 队列获取消息的操作的各种选项。

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 924 页的『CompletionCode 属性』
- 第 924 页的『MatchOptions 属性』
- 第 925 页的『Options 属性』
- 第 925 页的『ReasonCode 属性』
- 第 925 页的『ReasonName 属性』
- 第 925 页的『ResolvedQueueName 属性』
- 第 925 页的『WaitInterval 属性』

方法

- 第 925 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**

要设置: *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 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 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 926 页的『AlternateUserId 属性』
- 第 926 页的『CloseOptions 属性』
- 第 927 页的『CompletionCode 属性』
- 第 927 页的『ConnectionReference 属性』
- 第 927 页的『FirstDistributionListItem 属性』
- 第 927 页的『IsOpen 属性』
- 第 928 页的『OpenOptions 属性』
- 第 928 页的『ReasonCode 属性』
- 第 928 页的『ReasonName 属性』

方法

- 第 928 页的『AddDistributionListItem 方法』
- 第 929 页的『ClearErrorCodes 方法』
- 第 929 页的『Close 方法』
- 第 929 页的『Open 方法』
- 第 929 页的『Put 方法』

语法

Dim *distlist*。**As 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**

要设置: *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**

要设置: *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**

To set: *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 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 原因码。

语法： 获取: *reasonname $= MQDistributionList*.**ReasonName**

AddDistributionListItem 方法

创建新的 MQDistributionList 项对象，并将其与分发列表对象相关联。 队列名称参数是必需的。

将分发列表项的 DistributionList 属性设置为拥有的分发列表，并将分发列表的 FirstDistributionListItem 属性设置为引用此新的分发列表项。

对于新的分发列表项， PreviousDistributionListItem 属性设置为无，而 NextDistributionListItem 属性设置为引用先前第一个的任何分发列表项，如果先前没有任何分发列表项 (即，在已存在的分发列表项之前插入新的分发列表项) ，那么将不引用任何分发列表项。

如果分发列表已打开，那么这将返回错误。

定义于：
MQDistributionList 类

语法： set distributionlistitem = *MQDistributionList*.**AddDistributionListItem** (QName$, QMgrName$)

参数：

QName$ 字符串。 WebSphere MQ 队列的名称。

QMgrName$ 字符串。 WebSphere 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

使用当前值 AlternateUserId 打开由 QueueName 和 (适当情况下) 与当前对象关联的分发列表项的 QueueManager 名称属性指定的每个队列。

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

调用 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 931 页的『AccountingToken 属性』.
- 第 931 页的『AccountingTokenHex 属性』.
- 第 931 页的『CompletionCode 属性』.
- 第 932 页的『CorrelationId 属性』.
- 第 932 页的『CorrelationIdHex 属性』.
- 第 932 页的『DistributionList 属性』.
- 第 932 页的『Feedback 属性』.
- 第 933 页的『GroupId 属性』.
- 第 933 页的『GroupIdHex 属性』.
- 第 933 页的『MessageId 属性』.
- 第 933 页的『MessageIdHex 属性』.
- 第 933 页的『NextDistributionListItem 属性』.
- 第 934 页的『PreviousDistributionListItem 属性』.
- 第 934 页的『QueueManagerName 属性』.
- 第 934 页的『QueueName 属性』.
- 第 934 页的『ReasonCode 属性』.
- 第 934 页的『ReasonName 属性』.
- 第 935 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 $= MQDistributionList* 项。**AccountingToken**

要设置: *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**

要设置: *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 $= MQDistributionList* 项。**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**

要设置: *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**

要设置: *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*。**反馈**

要设置: *MQDistributionList* 项。**反馈** = *feedback&*

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 $= MQDistributionList* 项。**GroupId**

要设置: *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**

要设置: *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 $= MQDistributionList* 项。**MessageId**

要设置: *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 $= MQDistributionList* 项。**MessageIdHex**

要设置: *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。 WebSphere 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**

要设置: *MQDistributionListItem*。**QueueManagerName** = *qmname $*

QueueName 属性

读 — 写。 WebSphere 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 $= MQDistributionList* 项。**QueueName**

要设置: *MQDistributionListItem*.**QueueName** = *qname $*

ReasonCode 属性

只读。 最近一次打开或放入到拥有的分发列表对象中的完成代码集。

定义于:
MQDistributionListItem 类

数据类型:
长

值:
请参阅 API 原因码。

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

语法: 要获取: *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 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： 调用 *MQDistributionListItem*。**ClearErrorCodes**

故障诊断

有关所提供的跟踪设备和常见缺陷的信息，并提供有关如何避免这些缺陷的帮助。

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 935 页的『使用跟踪』
- 第 936 页的『当 WebSphere MQ Automation Classes for ActiveX 脚本失败时』
- 第 936 页的『原因码』
- 第 938 页的『代码级工具』

使用跟踪

MQAX 包含跟踪工具，用于帮助服务组织识别发生问题时发生的情况。 它显示运行 MQAX 脚本时采用的路径。 除非您迁到问题，否则请在设置跟踪的情况下运行，以避免不必要使用系统资源。

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

请注意，为 OMQ_TRACE 指定 *any* 值将打开跟踪事实。 即使将 OMQ_TRACE 设置为 OFF ，跟踪仍处于活动状态。

要关闭跟踪，请不要为 OMQ_TRACE 指定值。

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

在确定要将跟踪文件写入的位置时，请确保您有足够的权限来写入磁盘，而不仅仅是从磁盘中读取。

开启跟踪后，它会降低 MQAX 的运行速度，但不会影响 ActiveX 或 WebSphere MQ 环境的性能。 当您不再需要某个跟踪文件时，可将其删除。

必须停止 MQAX 运行才能更改 OMQ_TRACE 变量的状态。

跟踪文件名和目录

跟踪文件名采用格式 OMQnnnnn.trc，其中 nnnnn 是当时正在运行的 ActiveX 进程的标识。

表 *157:* 命令及其效果

命令	结果
SET OMQ_TRACE_PATH=drive:\directory	设置将在其中写入跟踪文件的跟踪目录。
SET OMQ_TRACE_PATH =	除去使用当前工作目录 (启动 ActiveX 时) 的 OMQ_PATH 环境变量。
ECHO %OMQ_TRACE_PATH%	显示 Windows 上跟踪目录的当前设置。
SET OMQ_TRACE = xxxxxxxx	这会将跟踪设置为 ON。 通过在 "=" 符号后面放置一个或多个字符来打开跟踪。 例如: SET OMQ_TRACE=yes SET OMQ_TRACE = no。 在这两个示例中，跟踪都将设置为 ON。 这仅对单个窗口/会话有效
SET OMQ_TRACE=	将跟踪设置为 OFF
ECHO %OMQ_TRACE%	在 Windows 上显示环境变量的内容。
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 WebSphere MQ Automation Classes for ActiveX 脚本失败时

如果 WebSphere MQ Automation Classes for ActiveX 脚本失败，那么有许多信息源。

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 OMQnnnnn.fdc 的文件中找到该报告，其中 nnnnn 是当时运行的 ActiveX 进程的号码。 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

WebSphere MQ 提供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 Windows NT 应用程序事件日志。

原因码

除了针对 WebSphere MQ MQI 记录的原因码外，还可能出现以下原因码。 有关其他代码，请参阅 WebSphere MQ 应用程序事件日志。

表 *158:* 原因码及其含义

原因码	说明
MQRC_LIBRARY_LOAD_ERROR (6000)	无法装入一个或多个 WebSphere MQ 库。 检查所有 WebSphere MQ 库是否位于您正在使用的系统上的正确搜索路径中。 例如，确保包含 WebSphere MQ 库的目录位于 PATH 中。
MQRC_CLASS_LIBRARY_ERROR (6001)	其中一个 WebSphere MQ 类库调用返回了意外的 ReasonCode/CompletionCode。 请检查首次故障症状报告以获取详细信息。 记下正在使用的最后一个方法/属性和类，并通知 IBM 支持人员该问题。

表 *158:* 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)	已尝试从消息缓冲区读取压缩十进制数，但数据指针处的数据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** ，需要一个或多个附加选项。 需要隐式地重新打开，但是不能关闭。 显式设置 **OpenOptions** 以涵盖所有事件，因此不需要隐式重新打开。 已阻止关闭，因为队列已打开用于独占输入，而关闭将为其他用户提供一个机会窗口，以便他们可以访问该队列。
MQRC_REOPEN_INQUIRE_ERROR (6101)	打开的对象没有正确的 OpenOptions ，需要一个或多个附加选项。 需要隐式地重新打开，但是不能关闭。 显式设置 OpenOptions 以包含 MQOO_INQUIRE。 由于需要在关闭之前动态检查对象的一个或多个特征，并且 OpenOptions 尚未包含 MQOO_INQUIRE ，因此已阻止关闭。
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	打开的对象没有正确的 OpenOptions ，需要一个或多个附加选项。 需要隐式地重新打开，但是不能关闭。 显式设置 OpenOptions 以涵盖所有事件，因此不需要隐式重新打开。 因为已使用 MQOO_SAVE_ALL_CONTEXT 打开此队列，且之前执行了破坏性的 get 操作，因此会阻止关闭。 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。
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	打开的对象没有正确的 **OpenOptions** ，需要一个或多个附加选项。 需要隐式重新打开，但已阻止关闭。 显式设置 **OpenOptions** 以涵盖所有事件，因此不需要隐式重新打开。 已阻止关闭，因为该队列是定义类型为 MQQDT_TEMPORARY_DYNAMIC 的本地队列，将被关闭破坏。
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。 显式设置 OpenOptions 以涵盖所有事件，因此不需要隐式重新打开。
MQRC_ENCODING_ERROR (6106)	下一个消息项的编码需要为 MQENC_NATIVE 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。 这可能是由于对从 VBA 用作调用参数的 WebSphere MQ 对象使用显式声明 (例如， dim msg as Object is ok ， dim msg as MqMessage 可能导致问题)。 例如，在 Excel 中，使用 q 定义 dim msg 并将 dim msg 设置为 MqMessageq.put msg 提供 reasonCode MQRC_NULL_POINTER。 它可从 VisualBasic 正确运行。
MQRC_NO_CONNECTION_REFERENCE (6109)	**MQQueue** 对象已失去与 **MQQueueManager** 的连接。 如果 **MQQueueManager** 已断开连接，那么将发生此情况。 删除此 **MQQueue** 对象。
MQRC_NO_BUFFER (6110)	没有可用的缓冲区。 对于 **MQMessage** 对象，无法分配一个对象，这表示不应发生的对象状态内部不一致。
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。 24 是 **CorrelationId** 的正确长度，对于 **MessageId** 32 是 **AccountingToken** 的正确长度
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 *158:* 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) 与引用的 MQQueueManager 对象 (未连接) 之间存在不一致。
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	MQPutMessage 选项上下文引用未引用有效的 MQQueue 对象。 该对象已被销毁。
MQRC_CONTEXT_OPEN_ERROR (6122)	MQPutMessage 选项上下文引用引用无法打开以建立上下文的 MQQueue 对象。 这可能是因为 MQQueue 对象具有不适当的打开选项。 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，此长度不足以包含固定字段和所有偏移量数据。
MQRC_NOT_CONNECTED (6124)	方法失败，因为与队列管理器的必需连接不可用，并且无法隐式建立连接。
MQRC_NOT_OPEN (6125)	方法失败，因为 WebSphere MQ 对象未打开，并且无法隐式完成打开。
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	无法打开 MQDistributionList ，因为分发列表中没有 MQDistributionList 项对象。 更正操作: 将至少一个 MQDistributionList 项对象添加到分发列表。
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: 使用相应的打开选项打开对象，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 服务团队可能会询问您已安装的代码级别。

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 867 页的『使用组件对象模型接口 (WebSphere 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 WebSphere MQ 管理接口 (MQAI) 简介。 MQAI ActiveX MQBag 类以任何支持创建 COM 对象的语言 (例如， Visual Basic ， C + + ， Java 和其他 ActiveX 脚本编制客户机) 中可以使用的方式封装 MQAI 支持的数据包。

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 868 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 WebSphere 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 939 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和应答队列，可以将命令发送到 WebSphere 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 940 页的『Item 属性』.
- 第 941 页的『Count 属性』.
- 第 941 页的『Options 属性』.

MQBag 方法

以下列表中解释了 MQBag 对象的方法：

- 第 942 页的『Add 方法』.
- 第 943 页的『AddInquiry 方法』.

- 第 943 页的『Clear 方法』.
- 第 943 页的『Execute 方法』.
- 第 944 页的『FromMessage 方法』.
- 第 944 页的『ItemType 方法』.
- 第 945 页的『Remove 方法』.
- 第 945 页的『Selector 方法』.
- 第 946 页的『ToMessage 方法』.
- 第 947 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: 由检测到的错误的 WebSphere MQ 原因码和 COM 设施代码组成。 作为 COM 标准的工具字段指明了错误的责任区域。 对于 WebSphere MQ 检测到的错误，它始终是 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: 提供 WebSphere 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，将在 Err 对象中返回信息，并通过从 Err.Number 中减去常量 vbObject 错误来获取 WebSphere MQ 原因码。

ReasonCode = Err.Number - vbObjectError

如果 MQBag Execute 消息发送 PCF 消息并接收到应答，那么虽然发送的命令可能已失败，但该操作被视为成功。 在这种情况下，应答包本身包含完成和错误原因码，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。

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/输出**
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**
要由 INQUIRE 管理命令返回的 WebSphere 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

执行 *(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*（长整型）- 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 WebSphere MQ Automation Classes for ActiveX Starter 样本

本附录描述了 WebSphere MQ Automation Classes for ActiveX 入门模板样本，并说明了如何使用这些样本。

WebSphere 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。

在同一目录中，您还将找到 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

这些样本演示如何使用 WebSphere MQ Automation Classes for ActiveX 来执行以下操作:

- 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 948 页的『准备运行样本』 和

第 948 页的『样本中的错误处理』

运行 ActiveX 启动程序样本

在运行 WebSphere MQ Automation Classes for ActiveX 入门模板样本之前，请检查您是否具有正在运行的缺省队列管理器以及是否已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。 样本使用应在任何正常设置的 WebSphere 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 Basic Version 4 或更高版本的 MQAX 入门模板样本的信息，请参阅 第 948 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 950 页的『启动 MQAXCLSS 样本』

有关 MQAXDLST 样本的信息，请参阅第 950 页的『MQAXDLST 样本』

有关运行 Microsoft Excel 95 或更高版本的 MQAX 入门模板样本的信息，请参阅 MQAXTRIV.XLS，请参阅 第 950 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 950 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 951 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 版本 4 (或更高版本)
- Microsoft Excel 95 (或更高版本)
- Web 浏览器

您还需要：

- 正在运行 WebSphere MQ 队列管理器。
- 已定义 WebSphere MQ 队列。

样本中的错误处理

WebSphere MQ Automation Classes for ActiveX 包中提供的大部分样本几乎没有错误处理。 有关错误处理的更多信息，请参阅第 872 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 File Manager 中，选择样本 MQAXTRIV.VBP (Visual Basic 项目文件) 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 小测试程序" 中的任意位置。

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 a WebSphere MQ message to
'* and from a WebSphere 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   '* get message option

Dim GetOptions As MQGetMessageOptions   '* put 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 "WebSphere MQ Completion Code = " & MQSess.CompletionCode
    Print "WebSphere 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. 打开文件 MQAXCLSS.VBP，方法是在 Windows 资源管理器中双击文档图标，或者从 Visual Basic 中的文件菜单单击 "文件打开"。
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。

Microsoft Excel 95 或更高版本的 MQAX 入门模板样本

本节说明如何针对 Microsoft Excel 95 或更高版本 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 WebSphere MQ MQSC 命令文件 BANK.TST。 这将设置必要的 IBM WebSphere MQ 队列定义。

   要了解如何使用 MQSC 命令文件，请参阅脚本 (MQSC) 命令。
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 WebSphere 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 Internet Explorer (但不是 Netscape Navigator) 是兼容的 Web 浏览器。

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”。

   您可以通过双击 Windows 资源管理器中的文件图标来执行此操作，也可以从兼容 ActiveX 的 Web 浏览器的 "文件" 菜单中选择 "文件-打开"。
3. 按照屏幕上的指示信息进行操作。

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 IBM 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: