7.5

管理 *IBM WebSphere MQ*

注

在使用本资料及其支持的产品之前，请阅读第 141 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

管理 IBM WebSphere MQ

管理队列管理器和关联资源包含您为了激活和管理这些资源而经常执行的任务。 选择您首选的方法来管理队列管理器和关联的资源。

您可以在本地或远程管理 IBM WebSphere MQ 对象，请参阅第 7 页的『本地和远程管理』。

可使用多种不同的方法在 IBM WebSphere MQ 中创建和管理队列管理器及其相关资源。 这些方法包括命令行界面，图形用户界面和管理 API。 请参阅本主题中的部分和链接，以获取有关其中每个接口的更多信息。

根据您使用的平台，可使用不同组的命令来管理 IBM WebSphere MQ：

- 第 5 页的『IBM WebSphere MQ 控制命令』
- 第 5 页的『IBM WebSphere MQ 脚本 (MQSC) 命令』
- 第 6 页的『可编程命令格式 (PCF)』

还有以下其他选项，用于创建和管理 IBM WebSphere MQ 对象：

- 第 6 页的『IBM WebSphere MQ Explorer』
- 第 6 页的『Windows 缺省配置应用程序』
- 第 6 页的『Microsoft 集群服务 (MSCS)』

您可以使用 PCF 命令，针对本地和远程队列管理器自动完成某些管理和监视任务。 还可以通过在某些平台上使用 IBM WebSphere MQ 管理接口 (MQAI) 来简化这些命令。 有关自动完成管理任务的更多信息，请参阅第 7 页的『自动执行管理任务』。

IBM WebSphere MQ 控制命令

控制命令允许您对队列管理器本身执行管理任务。

IBM WebSphere MQ for Windows， UNIX and Linux® 系统提供您在系统命令行上发出的 控制命令 。

创建和管理队列管理器中描述了控制命令。 有关控制命令的命令参考，请参阅 IBM WebSphere MQ 控制命令。

IBM WebSphere MQ 脚本 (MQSC) 命令

使用 MQSC 命令来管理队列管理器对象，包括队列管理器本身、队列、进程定义、名称列表、通道、客户机连接通道、侦听器、服务和认证信息对象。

可使用 `runmqsc` 命令发出 MQSC 命令到队列管理器。 可通过从键盘发出命令以交互方式执行此操作，也可以重定向标准输入设备 (stdin)，从 ASCII 文本文件运行一系列命令。 在这两种情况下，命令的格式都是相同的。

根据命令中设置的标志，您可按三种方式运行 `runmqsc` 命令：

- 验证方式，即在本地队列管理器上验证 MQSC 命令，但不运行这些命令
- 直接方式，即在本地队列管理器上运行 MQSC 命令
- 间接方式，即在远程队列管理器上运行 MQSC 命令

MQSC 命令中指定的对象属性在本节中以大写形式显示 (例如， RQMNAME) ，尽管它们不区分大小写。 MQSC 命令属性名称限制为 8 个字符。

MQSC 命令在所有平台) 上都可用。 比较命令集中概述了 MQSC 命令。

在 Windows、UNIX 或 Linux 上，可使用 MQSC 作为在系统命令行上发出的单一命令。 要发出更复杂的命令或多个命令，可以将 MQSC 构建到从 Windows， UNIX 或 Linux 系统命令行运行的文件中。 可将 MQSC 发送到远程队列管理器。 有关完整详细信息，请参阅 MQSC 参考。

第 65 页的『脚本 (MQSC) 命令』 包含每条 MQSC 命令的描述及其语法。

有关在本地管理中使用 MQSC 命令的更多信息，请参阅 第 65 页的『使用 MQSC 命令执行本地管理任务』 。

可编程命令格式 (PCF)

可编程命令格式 (PCF) 定义可在网络中的程序与任何队列管理器（支持 PCF）之间交换的命令和回复消息。 您可以在系统管理应用程序中使用 PCF 命令来管理 IBM WebSphere MQ 对象：认证信息对象、通道、通道侦听器、名称列表、进程定义、队列管理器、队列、服务以及存储类。 可从网络中的单一点运行应用程序，以使用本地队列管理器与任何队列管理器（本地或远程）互通命令和回复信息。

有关 PCF 的更多信息，请参阅第 8 页的『可编程命令格式简介』。

有关命令和响应的 PCF 和结构的定义，请参阅可编程命令格式参考。

IBM WebSphere MQ Explorer

通过使用 IBM WebSphere MQ Explorer，您可以执行以下操作:

- 定义和控制各种资源，包括队列管理器，队列，进程定义，名称列表，通道，客户机连接通道，侦听器，服务和集群。
- 启动或停止本地队列管理器及其关联的进程。
- 在您的工作站上或从其他工作站查看队列管理器及其关联的对象。
- 检查队列管理器、集群和通道的状态。
- 根据队列状态，检查以确定哪些应用程序、用户或通道打开了特定队列。

在 Windows 和 Linux 系统上，可以使用系统菜单， `MQExplorer` 可执行文件或 **strmqcfg** 命令来启动 IBM WebSphere MQ Explorer 。

在 Linux 上，要成功启动 IBM WebSphere MQ Explorer，您必须能够将文件写入主目录，并且主目录必须存在。

您可以使用 IBM WebSphere MQ Explorer 在其他平台（包括 z/OS）上管理远程队列管理器，要获取详细信息以及下载 SupportPac MS0T，请参阅 https://www.ibm.com/support/docview.wss?uid=swg24021041。

请参阅第 51 页的『使用 IBM WebSphere MQ Explorer 进行管理』以获取更多信息。

Windows 缺省配置应用程序

您可以使用 Windows 缺省配置程序来创建 IBM WebSphere MQ 对象的 入门模板 (或缺省) 集。 在 表 1 中列出了创建的缺省对象的摘要: 由 Windows 缺省配置应用程序创建的对象。

Microsoft 集群服务 (MSCS)

Microsoft Cluster Service (MSCS) 使您能够将服务器连接到 集群中，从而提供更高的数据和应用程序可用性，并使系统更易于管理。 MSCS 可自动检测服务器或应用程序故障并从中恢复。

切勿将 MSCS 意义上的集群与 IBM WebSphere MQ 集群相混淆。 区别在于:

IBM WebSphere MQ 集群
是一台或多台计算机上的两个或多个队列管理器的组，提供自动互连，并允许在它们之间共享队列以实现负载均衡和冗余。

MSCS 集群
以如下方式连接和配置的计算机组: 如果一个计算机发生故障， MSCS 将执行 故障转移，将应用程序的状态数据从发生故障的计算机传输到集群中的另一台计算机，然后在那里重新启动其操作。

支持 Microsoft Cluster Service (MSCS) 提供有关如何配置 IBM WebSphere MQ for Windows 系统以使用 MSCS 的详细信息。

相关概念
WebSphere MQ 技术概述
第 61 页的『管理本地 IBM WebSphere MQ 对象』

本节说明如何管理本地 IBM WebSphere MQ 对象以支持使用消息队列接口 (MQI) 的应用程序。 在此上下文中，本地管理意味着创建，显示，更改，复制和删除 IBM WebSphere MQ 对象。

第 92 页的『管理远程 IBM WebSphere MQ 对象』

丢失与 XA 资源管理器的联系时的注意事项

相关任务

规划

配置

相关参考

业务支持方案

本地和远程管理

您可以在本地或远程管理 WebSphere MQ 对象。

本地管理 是指在本地系统上定义的任何队列管理器上执行管理任务。 您可以访问其他系统，例如通过 TCP/IP 终端仿真程序 **telnet**，并在那里执行管理。 在 WebSphere MQ 中，可以将此视为本地管理，因为不涉及任何通道，即，通信由操作系统管理。

WebSphere MQ 支持通过称为 远程管理的单一联系点进行管理。 这允许您从本地系统发出在另一个系统上处理的命令，也适用于 WebSphere MQ Explorer。 例如，可以发出远程命令来更改远程队列管理器上的队列定义。 您不必登录到该系统，尽管您确实需要定义相应的通道。 目标系统上的队列管理器和命令服务器必须正在运行。

某些命令不能以此方式发出，尤其是创建或启动队列管理器以及启动命令服务器。 要执行此类型的任务，必须登录到远程系统并从该系统发出命令，或者创建可以为您发出命令的进程。 此限制也适用于 WebSphere MQ Explorer。

第 92 页的『管理远程 IBM WebSphere MQ 对象』 更详细地描述了远程管理主题。

如何使用 IBM WebSphere MQ 控制命令

本节描述如何使用 IBM WebSphere MQ 控制命令。

如果要发出控制命令，那么您的用户标识必须是 mqm 组的成员。 有关这方面的更多信息，请参阅 在 UNIX， Linux 和 Windows 系统上管理 WebSphere MQ 的权限 。 此外，请注意以下特定于环境的信息:

WebSphere MQ for Windows

可以从命令行发出所有控制命令。 命令名及其标志不区分大小写: 您可以输入大写，小写或大写与小写的组合。 但是，控制命令的自变量 (例如队列名称) 区分大小写。

在语法描述中，连字符（-）用作标志指示符。 可以使用正斜杠 (/) 代替连字符。

针对 UNIX and Linux 系统的 WebSphere MQ

可以从 shell 发出所有 WebSphere MQ 控制命令。 所有命令都区分大小写。

可以使用 IBM WebSphere MQ Explorer 发出部分控制命令。

有关更多信息，请参阅 WebSphere MQ 控制命令

自动执行管理任务

您可能决定自动执行某些管理和监视任务对您的安装是有益的。 您可以使用可编程命令格式 (PCF) 命令自动执行本地和远程队列管理器的管理任务。 本部分假定您具有管理 WebSphere MQ 对象的经验。

PCF 命令

WebSphere MQ 可编程命令格式 (PCF) 命令可用于将管理任务编程到管理程序中。 通过这种方式，您可以从程序中处理队列管理器对象 (队列，进程定义，名称列表，通道，客户机连接通道，侦听器，服务和认证信息对象)，甚至可以处理队列管理器本身。

PCF 命令涵盖 MQSC 命令提供的相同功能范围。 您可以编写程序以从单个节点向网络中的任何队列管理器发出 PCF 命令。 这样，您既可以集中管理任务，也可以自动执行管理任务。

每个 PCF 命令都是嵌入在 WebSphere MQ 消息的应用程序数据部分中的数据结构。 使用 MQI 函数 MQPUT 以与任何其他消息相同的方式将每个命令发送到目标队列管理器。 如果命令服务器正在接收消息的队列管理器上运行，那么命令服务器会将其解释为命令消息并运行该命令。 为了获取回复，应用程序发出 MQGET 调用，并在另一数据结构中返回回复数据。 然后，应用程序可以处理应答并相应地执行操作。

注: 与 MQSC 命令不同， PCF 命令及其回复不是您可以读取的文本格式。

简而言之，以下是创建 PCF 命令消息所需的一些内容:

消息描述符
这是标准 WebSphere MQ 消息描述符，其中:

- 消息类型 (*MsqType*) 为 MQMT_REQUEST。
- 消息格式 (*Format*) 为 MQFMT_ADMIN。

应用程序数据
包含包含 PCF 头的 PCF 消息，其中:

- PCF 消息类型 (*Type*) 指定 MQCFT_COMMAND。
- 命令标识指定命令，例如 *Change Queue* (MQCMD_CHANGE_Q)。

有关 PCF 数据结构及其实现方式的完整描述，请参阅 第 8 页的『可编程命令格式简介』。

PCF 对象属性

PCF 中的对象属性不限于 8 个字符，因为它们适用于 MQSC 命令。 它们以斜体显示在本指南中。 例如，RQMNAME 的 PCF 等效项为 *RemoteQMgrName*。

对 PCF 进行转义

转义 PCF 是在消息文本中包含 MQSC 命令的 PCF 命令。 您可以使用 PCF 将命令发送到远程队列管理器。有关对 PCF 进行转义的更多信息，请参阅 转义。

可编程命令格式简介

可编程命令格式 (PCF) 定义可在网络中的程序与任何队列管理器（支持 PCF）之间交换的命令和回复消息。PCF 简化了队列管理器管理和其他网络管理。 它们可用于解决分布式网络的复杂管理问题，尤其是随着网络规模和复杂性的增长。

本产品文档中描述的可编程命令格式受以下支持:

- IBM WebSphere MQ for AIX
- IBM WebSphere MQ for HP-UX
- IBM WebSphere MQ for Linux
- IBM WebSphere MQ for Solaris
- IBM WebSphere MQ for Windows
- IBM WebSphere MQ for HP Integrity NonStop Server

问题 PCF 命令解决

分布式网络的管理可能会变得复杂。 随着网络规模和复杂性的增加，管理问题继续增加。

特定于消息传递和排队的管理示例包括:

- 资源管理。

  例如，队列创建和删除。
- 性能监视。

例如，最大队列深度或消息速率。

- 控件。

  例如，调整队列参数，例如，最大队列深度，最大消息长度以及启用和禁用队列。

- 消息路由。

  定义通过网络的备用路由。

WebSphere MQ PCF 命令可用于简化队列管理器管理和其他网络管理。 PCF 命令允许您使用单个应用程序从网络中的单个队列管理器执行网络管理。

什么是 PCF?

PCF 定义可在程序与网络中的任何队列管理器 (支持 PCF) 之间交换的命令和应答消息。 您可以在系统管理应用程序中使用 PCF 命令来管理 WebSphere MQ 对象: 认证信息对象，通道，通道侦听器，名称列表，进程定义，队列管理器，队列，服务和存储类。 可从网络中的单一点运行应用程序，以使用本地队列管理器与任何队列管理器（本地或远程）互通命令和回复信息。

每个队列管理器都有一个具有标准队列名称的管理队列，应用程序可以将 PCF 命令消息发送到该队列。 每个队列管理器还具有一个命令服务器，用于处理来自管理队列的命令消息。 因此，网络中的任何队列管理器都可以处理 PCF 命令消息，并且可以使用指定的应答队列将应答数据返回到应用程序。 PCF 命令和应答消息是使用正常消息队列接口 (MQI) 发送和接收的。

有关可用 PCF 命令 (包括其参数) 的列表，请参阅 可编程命令格式的定义。

使用可编程命令格式

您可以在系统管理程序中使用 PCF 进行 WebSphere MQ 远程管理。

本节包括:

- 第 9 页的『PCF 命令消息』
- 第 11 页的『响应』
- 用于命名 IBM WebSphere MQ 对象的规则
- 第 13 页的『PCF 命令的权限检查』

PCF 命令消息

PCF 命令消息由 PCF 头，该头中标识的参数以及用户定义的消息数据组成。 使用 "消息队列" 接口调用来发出消息。

每个命令及其参数都作为单独的命令消息发送，该命令消息中包含后跟多个参数结构的 PCF 头; 有关 PCF 头的详细信息，请参阅 MQCFH-PCF 头，有关参数结构的示例，请参阅 MQCFST-PCF 字符串参数。 PCF 头标识命令以及在同一消息中跟随的参数结构数。 每个参数结构都为命令提供一个参数。

由命令服务器生成的对命令的应答具有类似的结构。 有一个 PCF 头，后跟一些参数结构。 应答可以由多条消息组成，但命令始终仅由一条消息组成。

在 z/OS 以外的平台上，将 PCF 命令发送到的队列始终称为 SYSTEM.ADMIN.COMMAND.QUEUE。

如何发出 PCF 命令消息

使用常规消息队列接口 (MQI) 调用， MQPUT 和 MQGET 等，将 PCF 命令和响应消息放入其队列以及从其队列中检索这些消息。

注:

确保命令服务器正在目标队列管理器上运行，以便 PCF 命令在该队列管理器上进行处理。

有关提供的头文件的列表，请参阅 WebSphere MQ COPY，头，包含和模块文件。

PCF 命令的消息描述符

WebSphere MQ 消息描述符完整记录在 MQMD-消息描述符中。

PCF 命令消息在消息描述符中包含以下字段:

Report
任何有效值 (根据需要)。

MsgType
此字段必须是 MQMT_REQUEST ， 以指示需要响应的消息。

Expiry
任何有效值 (根据需要)。

Feedback
设置为 MQFB_NONE

Encoding
如果要发送到 Windows， UNIX 或 Linux 系统，请将此字段设置为用于消息数据的编码; 必要时将执行转换。

CodedCharSetId
如果要发送到 Windows， UNIX 或 Linux 系统将此字段设置为用于消息数据的编码字符集标识; 必要时执行转换。

Format
设置为 MQFMT_ADMIN。

Priority
任何有效值 (根据需要)。

Persistence
任何有效值 (根据需要)。

MsgId
发送应用程序可以指定任何值，或者可以指定 MQMI_NONE 以请求队列管理器生成唯一消息标识。

CorrelId
发送应用程序可以指定任何值，或者可以指定 MQCI_NONE 以指示无相关标识。

ReplyToQ
用于接收响应的队列的名称。

ReplyToQMgr
响应的队列管理器的名称 (或空白)。

消息上下文字段
可以根据需要将这些字段设置为任何有效值。 通常， Put 消息选项 MQPMO_DEFAULT_CONTEXT 用于将消息上下文字段设置为缺省值。

如果您正在使用 version-2 MQMD 结构，那么必须设置以下其他字段:

GroupId
设置为 MQGI_NONE

MsgSeqNumber
设置为 1

Offset
设置为 0

MsgFlags
设置为 MQMF_NONE

OriginalLength
设置为 MQOL_UNDEFINED

发送用户数据

PCF 结构还可用于发送用户定义的消息数据。 在这种情况下，消息描述符 *Format* 字段必须设置为 MQFMT_PCF。

在指定队列中发送和接收 *PCF* 消息

将 PCF 消息发送到指定队列

要将消息发送到指定队列， mqPutBag 调用会将指定包的内容转换为 PCF 消息，并将消息发送到指定队列。 通话结束后，袋子的内容保持不变。

作为此调用的输入，必须提供:

- MQI 连接句柄。
- 要放置消息的队列的对象句柄。
- 消息描述符。 有关消息描述符的更多信息，请参阅 MQMD-消息描述符。
- 使用 MQPMO 结构放置消息选项。 有关 MQPMO 结构的更多信息，请参阅 MQPMO-Put-message 选项。
- 要转换为消息的包的句柄。

  注: 如果包中包含管理消息，并且 mqAdd 查询调用用于将值插入到包中，那么 MQIASY_COMMAND 数据项的值必须是 MQAI 可识别的 INQUIRE 命令。

有关 mqPutBag 调用的完整描述，请参阅 mqPutBag。

从指定队列接收 PCF 消息

要从指定队列接收消息， mqGetBag 调用从指定队列获取 PCF 消息并将消息数据转换为数据包。

作为此调用的输入，必须提供:

- MQI 连接句柄。
- 要从中读取消息的队列的对象句柄。
- 消息描述符。 在 MQMD 结构中， `Format` 参数必须是 MQFMT_ADMIN ， MQFMT_EVENT 或 MQFMT_PCF。

  注: 如果在工作单元 (即，使用 MQGMO_SYNCPOINT 选项) 中接收到消息，并且该消息具有不受支持的格式，那么可以回退该工作单元。 然后在队列上恢复该消息，并且可以使用 MQGET 调用 (而不是 mqGetBag 调用) 来检索该消息。 有关消息描述符的更多信息，请参阅 MQGMO-Get-message 选项。
- 使用 MQGMO 结构获取消息选项。 有关 MQGMO 结构的更多信息，请参阅 MQMD-消息描述符。
- 用于包含已转换消息的包的句柄。

有关 mqGetBag 调用的完整描述，请参阅 mqGetBag。

响应

作为对每个命令的响应，命令服务器会生成一条或多条响应消息。 响应消息具有与命令消息相似的格式。

PCF 头与它作为响应的命令具有相同的命令标识值 (请参阅 MQCFH-PCF 头 以获取详细信息)。 根据请求的报告选项设置消息标识和相关标识。

如果命令消息的 PCF 头类型为 MQCFT_COMMAND ，那么仅生成标准响应。 此类命令在除 z/OS 以外的所有平台上都受支持。 较旧的应用程序在 z/OS 上不支持 PCF; WebSphere MQ Windows Explorer 是此类应用程序之一 (但是， V 6.0 或更高版本 IBM WebSphere MQ Explorer 在 z/OS 上支持 PCF)。

如果命令消息的 PCF 头类型为 MQCFT_COMMAND_XR ，那么将生成扩展响应或标准响应。 此类命令在 z/OS 和一些其他平台上受支持。 在 z/OS 上发出的命令仅生成扩展响应。 在其他平台上，可能会生成任一类型的响应。

如果单个命令指定通用对象名，那么将在其自己的消息中针对每个匹配对象返回单独的响应。 对于响应生成，具有通用名称的单个命令将被视为多个单独的命令 (控制字段 MQCFC_LAST 或 MQCFC_NOT_LAST 除外)。 否则，一条命令消息将生成一条响应消息。

某些 PCF 响应可能会返回结构，即使未请求该结构也是如此。 此结构在响应定义 (可编程命令格式的定义) 中显示为 始终返回。 对于这些响应，需要对响应中的对象进行命名以标识应用数据的对象的原因。

响应的消息描述符

响应消息在消息描述符中具有以下字段:

MsgType
此字段为 MQMT_REPLY。

MsgId
此字段由队列管理器生成。

CorrelId
根据命令消息的报告选项生成此字段。

Format
此字段为 MQFMT_ADMIN。

Encoding
设置为 MQENC_NATIVE。

CodedCharSetId
设置为 MQCCSI_Q_MGR。

Persistence
与命令消息中的相同。

Priority
与命令消息中的相同。

将使用 MQPMO_PASS_IDENTITY_CONTEXT 生成响应。

标准响应

将生成头类型为 MQCFT_COMMAND 的命令消息，生成标准响应。 此类命令在除 z/OS 以外的所有平台上都受支持。

有三种类型的标准响应:

- 确定响应
- 错误响应
- 数据响应

确定响应

此响应包含以命令格式头开头的消息， *CompCode* 字段为 MQCC_OK 或 MQCC_WARNING。

对于 MQCC_OK， *Reason* 为 MQRC_NONE。

对于 MQCC_WARNING， *Reason* 标识警告的性质。 在这种情况下，命令格式头后面可能跟有一个或多个适合于此原因码的警告参数结构。

在任一情况下，对于查询命令，可遵循以下部分中描述的进一步参数结构。

错误响应

如果该命令有错误，那么将发送一条或多条错误响应消息 (即使对于通常只有一条响应消息的命令，也可能发送多条错误响应消息)。 这些错误响应消息根据需要设置了 MQCFC_LAST 或 MQCFC_NOT_LAST。

每条此类消息都以响应格式头开头， *CompCode* 值为 MQCC_FAILED， *Reason* 字段用于标识特定错误。 通常，每条消息都描述了不同的错误。 此外，每条消息的头后面都有零个或一个 (从不超过一个) 错误参数结构。 此参数结构 (如果有) 是 MQCFIN 结构，其中 *Parameter* 字段包含下列其中一项:

- MQIACF_PARAMETER_ID

  结构中的 *Value* 字段是出错的参数的参数标识 (例如 MQCA_Q_NAME)。

- MQIACF_ERROR_ID

  此值与 *Reason* 值 (在命令格式头中) MQRC_UNEXPECTED_ERROR 配合使用。 MQCFIN 结构中的 *Value* 字段是命令服务器接收到的意外原因码。

- MQIACF_SELECTOR

  如果随命令发送的列表结构 (MQCFIL) 包含重复的选择器或无效的选择器，那么会发生此值。 命令格式头中的 *Reason* 字段标识错误，而 MQCFIN 结构中的 *Value* 字段是发生错误的命令的 MQCFIL 结构中的参数值。

- MQIACF_ERROR_OFFSET

  当 Ping 通道命令中存在数据比较错误时，将发生此值。 结构中的 *Value* 字段是 Ping 通道比较错误的偏移量。

- MQIA_CODED_CHAR_SET_ID

  当入局 PCF 命令消息的消息描述符中的编码字符集标识与目标队列管理器的编码字符集标识不匹配时，会发生此值。 结构中的 *Value* 字段是队列管理器的编码字符集标识。

最后 (或唯一) 错误响应消息是摘要响应，其 *CompCode* 字段为 MQCC_FAILED， *Reason* 字段为 MQRCCF_COMMAND_FAILED。 此消息的头后面没有参数结构。

数据响应

此响应包含对查询命令的 OK 响应 (如前所述)。 "确定" 响应后跟包含请求的数据的其他结构，如 可编程命令格式的定义中所述。

应用程序不得依赖于以任何特定顺序返回的这些附加参数结构。

PCF 命令的权限检查

处理 PCF 命令时，命令消息中的消息描述符中的 *UserIdentifier* 将用于必需的 WebSphere MQ 对象权限检查。 在每个平台上以不同方式实施权限检查，如本主题中所述。

将在正在处理该命令的系统上执行这些检查; 因此，此用户标识必须存在于目标系统上，并且具有处理该命令所需的权限。 如果消息来自远程系统，那么实现目标系统上存在的标识的一种方法是在本地和远程系统上都具有匹配的用户标识。

IBM WebSphere MQ (对于 Windows 和 UNIX and Linux 系统)

Windows UNIX Linux

为了处理任何 PCF 命令，用户标识必须对目标系统上的队列管理器对象具有 *dsp* 权限。 此外，将对某些 PCF 命令执行 WebSphere MQ 对象权限检查，如 第 14 页的表 1 中所示。

要处理以下任何命令，用户标识必须属于组 *mqm*。

注: 仅对于 Windows，用户标识可以属于组 *Administrators* 或组 *mqm*。

- 更改通道
- 复制通道
- 创建通道
- 删除通道
- Ping 通道
- 重置通道
- 启动通道
- 停止通道
- 启动通道启动程序
- 启动通道侦听器
- 解析通道
- Reset Cluster
- 刷新集群
- 暂挂队列管理器

- 恢复队列管理器

WebSphere MQ for HP Integrity NonStop Server

为了处理任何 PCF 命令，用户标识必须对目标系统上的队列管理器对象具有 *dsp* 权限。 此外，将对某些 PCF 命令执行 IBM WebSphere MQ 对象权限检查，如 第 14 页的表 1 中所示。

要处理以下任何命令，用户标识必须属于组 *mqm*:

- 更改通道
- 复制通道
- 创建通道
- 删除通道
- Ping 通道
- 重置通道
- 启动通道
- 停止通道
- 启动通道启动程序
- 启动通道侦听器
- 解析通道
- Reset Cluster
- 刷新集群
- 暂挂队列管理器
- 恢复队列管理器

WebSphere MQ 对象权限

表 *1: Windows， HP Integrity NonStop Server ， UNIX and Linux systems*-对象权限

命令	**WebSphere MQ 对象权限**	**类权限 (针对对象类型)**
更改认证信息	dsp 和 chg	不适用
更改通道	dsp 和 chg	不适用
更改通道侦听器	dsp 和 chg	不适用
更改客户机连接通道	dsp 和 chg	不适用
更改名称列表	dsp 和 chg	不适用
更改进程	dsp 和 chg	不适用
更改队列	dsp 和 chg	不适用
更改队列管理器	chg 请参阅 "注释" *3* 和 "注释" *5*	不适用
更改服务	dsp 和 chg	不适用
清除队列	clr	不适用
复制认证信息	dsp	crt
复制认证信息 (替换) 请参阅注释 *1*	从*:* dsp 到*:* chg	crt
复制通道	dsp	crt
复制通道 (替换) 请参阅注释 *1*	从*:* dsp 到*:* chg	crt
复制通道侦听器	dsp	crt

表 *1: Windows， HP Integrity NonStop Server ， UNIX and Linux systems*-对象权限 *(*继续*)*		
命令	**WebSphere MQ 对象权限**	**类权限 (针对对象类型)**
复制通道侦听器 (替换) 请参阅注释 *1*	从*:* dsp 到*:* chg	crt
复制客户机连接通道	dsp	crt
复制客户机连接通道 (替换) 请参阅注释 *1*	从*:* dsp 到*:* chg	crt
复制名称列表	dsp	crt
复制名称列表 (替换) 请参阅注释 *1*	从 dsp 到 dsp 和 chg	crt
复制进程	dsp	crt
复制过程 (替换) 请参阅注释 *1*	从*:* dsp 到*:* chg	crt
复制队列	dsp	crt
复制队列 (替换) 请参阅注释 *1*	从 dsp 到 dsp 和 chg	crt
创建认证信息	*(*系统缺省认证信息*)* dsp	crt
创建认证信息 (替换) 请参阅注释 *1*	*(*系统缺省认证信息*)* dsp 到*:* chg	crt
创建通道	*(*系统缺省通道*)* dsp	crt
创建通道 (替换) 请参阅注释 *1*	*(*系统缺省通道*)* dsp 到*:* chg	crt
创建通道侦听器	*(*系统缺省侦听器*)* dsp	crt
创建通道侦听器 (替换) 请参阅注释 *1*	*(*系统缺省侦听器*)* dsp 到*:* chg	crt
创建客户机连接通道	*(*系统缺省通道*)* dsp	crt
创建客户机连接通道 (替换) 请参阅注释 *1*	*(*系统缺省通道*)* dsp 到*:* chg	crt
创建名称列表	*(*系统缺省名称列表*)* dsp	crt
创建名称列表 (替换) 请参阅注释 *1*	*(*系统缺省名称列表*)* dsp 到*:* dsp 和 chg	crt
创建进程	*(*系统缺省进程*)* dsp	crt
创建过程 (替换) 请参阅注释 *1*	*(*系统缺省进程*)* dsp 到*:* chg	crt
创建队列	*(*系统缺省队列*)* dsp	crt
创建队列 (替换) 请参阅注释 *1*	*(*系统缺省队列*)* dsp 到*:* dsp 和 chg	crt
创建服务	*(*系统缺省队列*)* dsp	crt
创建服务 (替换) 请参阅注释 *1*	*(*系统缺省队列*)* dsp 到*:* chg	crt
删除认证信息	dsp 和 dlt	不适用
删除权限记录	*(*队列管理器对象*)* chg 请参阅注释 *4*	请参阅注释 *4*
删除通道	dsp 和 dlt	不适用
删除通道侦听器	dsp 和 dlt	不适用
删除客户机连接通道	dsp 和 dlt	不适用
删除名称列表	dsp 和 dlt	不适用
删除进程	dsp 和 dlt	不适用

表 *1: Windows， HP Integrity NonStop Server ， UNIX and Linux systems*-对象权限 *(继续)*		
命令	**WebSphere MQ 对象权限**	**类权限 (针对对象类型)**
删除队列	dsp 和 dlt	不适用
删除服务	dsp 和 dlt	不适用
查询认证信息	dsp	不适用
查询权限记录	请参阅注释 *4*	请参阅注释 *4*
查询通道	dsp	不适用
查询通道侦听器	dsp	不适用
查询通道状态 (针对 **ChannelType** MQCHT_CLSSDR)	inq	不适用
查询客户机连接通道	dsp	不适用
查询名称列表	dsp	不适用
查询进程	dsp	不适用
查询队列	dsp	不适用
查询队列管理器	请参阅注释 *3*	不适用
查询队列状态	dsp	不适用
查询服务	dsp	不适用
Ping 通道	ctrl	不适用
Ping 队列管理器	请参阅注释 *3*	不适用
刷新队列管理器	(队列管理器对象) chg	不适用
刷新安全性 (针对 **SecurityType** MQSECTYPE_SSL)	(队列管理器对象) chg	不适用
重置通道	ctrlx	不适用
重置队列管理器	(队列管理器对象) chg	不适用
重置队列统计信息	dsp 和 chg	不适用
解析通道	ctrlx	不适用
设置权限记录	(队列管理器对象) chg 请参阅注释 *4*	请参阅注释 *4*
启动通道	ctrl	不适用
停止通道	ctrl	不适用
停止连接	(队列管理器对象) chg	不适用
启动侦听器	ctrl	不适用
停止侦听器	ctrl	不适用
启动服务	ctrl	不适用
停止服务	ctrl	不适用
转义	请参阅注释 *2*	请参阅注释 *2*

注意:

1. 如果要替换的对象存在，那么此命令适用，否则权限检查与 "创建" 或 "复制而不替换" 一样。

2. 所需权限由转义文本定义的 MQSC 命令确定，它相当于先前的一个命令。
3. 为了处理任何 PCF 命令，用户标识必须对目标系统上的队列管理器对象具有 dsp 权限。
4. 除非已使用 -a 参数启动命令服务器，否则将授权此 PCF 命令。 缺省情况下，命令服务器在队列管理器启动时启动，并且不带 -a 参数。 See the System Administration Guide for further information.
5. 授予队列管理器的用户标识 *chg* 权限使您能够为所有组和用户设置权限记录。 请勿将此权限授予普通用户或应用程序。

WebSphere MQ 还提供了一些通道安全性出口点，以便您可以提供自己的用户出口程序进行安全性检查。 在 显示通道 手册中提供了详细信息。

使用 MQAI 来简化 PCF 的使用

MQAI 是 WebSphere MQ 的管理接口，可用于 AIX， HP-UX， IBM i， Linux， Solaris 和 Windows 平台。

MQAI 通过使用 数据包在队列管理器上执行管理任务。 数据包允许您以比使用 PCF 更容易的方式处理对象的属性 (或参数)。

通过以下方式使用 MQAI:

简化 PCF 消息的使用
MQAI 是管理 WebSphere MQ 的简单方法; 您不必编写自己的 PCF 消息，从而避免与复杂数据结构相关联的问题。

要传递使用 MQI 调用编写的程序中的参数， PCF 消息必须包含字符串或整数数据的命令和详细信息。 为此，每个结构都需要程序中的多个语句，并且必须分配内存空间。 这个任务可能漫长而费力。

使用 MQAI 编写的程序会将参数传递到相应的数据包中，并且每个结构只需要一个语句。 使用 MQAI 数据包可消除您处理阵列和分配存储器的需求，并提供与 PCF 详细信息的某种程度的隔离。

更轻松地处理错误情况
很难从 PCF 命令获取返回码，但 MQAI 使程序更容易处理错误情况。

创建并填充数据包后，可以使用等待任何响应消息的 `mqExecute` 调用将管理命令消息发送到队列管理器的命令服务器。 `mqExecute` 调用处理与命令服务器的交换，并在 响应包中返回响应。

有关 MQAI 的更多信息，请参阅 第 17 页的『IBM WebSphere MQ 管理界面 (MQAI) 简介』。

IBM WebSphere MQ 管理界面 (MQAI) 简介

IBM WebSphere MQ 管理接口 (MQAI) 是 IBM WebSphere MQ 的编程接口。 它使用数据包在 IBM WebSphere MQ 队列管理器上执行管理任务，以比使用可编程命令格式 (PCF) 更容易的方式处理对象的属性 (或参数)。

MQAI 概念和术语

MQAI 是 WebSphere MQ 的编程接口，使用 C 语言以及 Visual Basic for Windows。 它在 z/OS 以外的平台上可用。

它使用数据包在 WebSphere MQ 队列管理器上执行管理任务。 数据包允许您以比使用其他管理界面 "可编程命令格式" (PCF) 更容易的方式处理对象的属性 (或参数)。 与使用 MQGET 和 MQPUT 调用相比， MQAI 可更轻松地处理 PCF。

有关数据包的更多信息，请参阅 第 43 页的『数据包』。 有关 PCF 的更多信息，请参阅 第 8 页的『可编程命令格式简介』

MQAI 的使用

您可以使用 MQAI 来执行以下操作:

- 简化 PCF 消息的使用。 MQAI 是管理 WebSphere MQ 的简单方法; 您不必编写自己的 PCF 消息，从而避免与复杂数据结构相关联的问题。

- 更轻松地处理错误情况。 很难从 WebSphere MQ 脚本 (MQSC) 命令获取返回码，但 MQAI 使程序更容易处理错误情况。
- 在应用程序之间交换数据。 应用程序数据以 PCF 格式发送，并由 MQAI 打包和解包。 如果消息数据由整数和字符串组成，那么可以使用 MQAI 来利用 PCF 数据的 WebSphere MQ 内置数据转换。 这将避免需要写入数据转换出口。 有关使用 MQAI 来管理 WebSphere MQ 以及在应用程序之间交换数据的更多信息，请参阅 第 17 页的『使用 MQAI 来简化 PCF 的使用』。

使用 MQAI 的示例

显示的列表提供了一些示例程序，用于演示 MQAI 的使用。 样本执行以下任务:

1. 创建本地队列。 第 18 页的『用于创建本地队列的样本 C 程序 (amqsaicq.c)』
2. 使用简单事件监视器在屏幕上显示事件。 第 22 页的『用于使用事件监视器显示事件的样本 C 程序 (amqsaiem.c)』
3. 打印所有本地队列及其当前深度的列表。 第 34 页的『用于查询队列和打印信息的样本 C 程序 (amqsailq.c)』
4. 打印所有通道及其类型的列表。 第 29 页的『用于查询通道对象的样本 C 程序 (amqsaicl.c)』

构建 MQAI 应用程序

要使用 MQAI 构建应用程序，请链接到与 WebSphere MQ 相同的库。 有关如何构建 WebSphere MQ 应用程序的信息，请参阅 构建 WebSphere MQ 应用程序。

有关使用 MQAI 配置 WebSphere MQ 的提示和技巧

MQAI 使用 PCF 消息将管理命令发送到命令服务器，而不是直接处理命令服务器本身。 可在 第 38 页的『有关配置 IBM WebSphere MQ 的提示和技巧』 中找到有关使用 MQAI 配置 WebSphere MQ 的提示

IBM WebSphere MQ 管理接口 (MQAI)

IBM WebSphere MQ for Windows， AIX， Linux， HP-UX 和 Solaris 支持 IBM WebSphere MQ 管理接口 (MQAI)。 MQAI 是 IBM WebSphere MQ 的编程接口，为您提供 MQI 的替代方法，用于发送和接收 PCF。

MQAI 使用 数据包 ，这使您能够比通过 MQAI 直接使用 PCF 更轻松地处理对象的属性 (或参数)。

MQAI 通过将参数传递到数据包中，提供了对 PCF 消息的更容易编程访问，因此每个结构只需要一个语句。 此访问可消除程序员处理阵列和分配存储器的需求，并提供一些与 PCF 详细信息的隔离。

MQAI 通过向命令服务器发送 PCF 消息并等待响应来管理 WebSphere MQ 。

本手册的第二部分描述了 MQAI。 请参阅 使用 Java 文档，以获取对 MQAI 的组件对象模型接口的描述。

用于创建本地队列的样本 C 程序 (amqsaicq.c)

样本 C 程序 `amqsaicq.c` 使用 MQAI 创建本地队列。

```
/********************************************************************************/
/*                                                                              */
/* Program name: AMQSAICQ.C                                                     */
/*                                                                              */
/* Description:  Sample C program to create a local queue using the             */
/*               WebSphere MQ Administration Interface (MQAI).                  */
/*                                                                              */
/* Statement:    Licensed Materials - Property of IBM                           */
/*                                                                              */
/*               84H2000, 5765-B73                                              */
/*               84H2001, 5639-B42                                              */
/*               84H2002, 5765-B74                                              */
/*               84H2003, 5765-B75                                              */
/*               84H2004, 5639-B43                                              */
/*                                                                              */
/*               (C) Copyright IBM Corp. 1999, 2024.                            */
/*                                                                              */
/********************************************************************************/
```

```
/*                                                                          */
/* Function:                                                                */
/*    AMQSAICQ is a sample C program that creates a local queue and is an   */
/*    example of the use of the mqExecute call.                             */
/*                                                                          */
/*     - The name of the queue to be created is a parameter to the program. */
/*                                                                          */
/*     - A PCF command is built by placing items into an MQAI bag.          */
/*       These are:-                                                        */
/*            - The name of the queue                                       */
/*            - The type of queue required, which, in this case, is local.  */
/*                                                                          */
/*     - The mqExecute call is executed with the command MQCMD_CREATE_Q.    */
/*       The call generates the correct PCF structure.                      */
/*       The call receives the reply from the command server and formats into */
/*       the response bag.                                                  */
/*                                                                          */
/*     - The completion code from the mqExecute call is checked and if there */
/*       is a failure from the command server then the code returned by the */
/*       command server is retrieved from the system bag that is            */
/*       embedded in the response bag to the mqExecute call.                */
/*                                                                          */
/* Note: The command server must be running.                                */
/*                                                                          */
/*                                                                          */

/****************************************************************************/
/*                                                                          */
/* AMQSAICQ has 2 parameters - the name of the local queue to be created    */
/*                          - the queue manager name (optional)             */
/*                                                                          */
/****************************************************************************/
/****************************************************************************/
/* Includes                                                                 */
/****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                       /* MQI                              */
#include <cmqcfc.h>                     /* PCF                              */
#include <cmqbc.h>                      /* MQAI                             */

void CheckCallResult(MQCHAR *, MQLONG , MQLONG );
void CreateLocalQueue(MQHCONN, MQCHAR *);

int main(int argc, char *argv[])
{
   MQHCONN hConn;                       /* handle to WebSphere MQ connection   */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name   */
   MQLONG connReason;                   /* MQCONN reason code               */
   MQLONG compCode;                     /* completion code                  */
   MQLONG reason;                       /* reason code                      */

   /*************************************************************************/
   /* First check the required parameters                                   */
   /*************************************************************************/
   printf("Sample Program to Create a Local Queue\n");
   if (argc < 2)
   {
     printf("Required parameter missing - local queue name\n");
     exit(99);
   }

   /*************************************************************************/
   /* Connect to the queue manager                                          */
   /*************************************************************************/
   if (argc > 2)
      strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
      MQCONN(QMName, &hConn, &compCode, &connReason);

/****************************************************************************/
/* Report reason and stop if connection failed                              */
/****************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }
```

```
/*****************************************************************************/
/* Call the routine to create a local queue, passing the handle to the       */
/* queue manager and also passing the name of the queue to be created.       */
/*****************************************************************************/
   CreateLocalQueue(hConn, argv[1]);

   /*****************************************************************************/
   /* Disconnect from the queue manager if not already connected                */
   /*****************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }
   return 0;

}
/*****************************************************************************/
/*                                                                           */
/* Function:    CreateLocalQueue                                             */
/* Description: Create a local queue by sending a PCF command to the command */
/*              server.                                                      */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* Input Parameters:  Handle to the queue manager                            */
/*                    Name of the queue to be created                        */
/*                                                                           */
/* Output Parameters: None                                                   */
/*                                                                           */
/* Logic: The mqExecute call is executed with the command MQCMD_CREATE_Q.    */
/*        The call generates the correct PCF structure.                      */
/*        The default options to the call are used so that the command is sent*/
/*        to the SYSTEM.ADMIN.COMMAND.QUEUE.                                 */
/*        The reply from the command server is placed on a temporary dynamic */
/*        queue.                                                             */
/*        The reply is read from the temporary queue and formatted into the  */
/*        response bag.                                                      */
/*                                                                           */
/*        The completion code from the mqExecute call is checked and if there */
/*        is a failure from the command server then the code returned by the */
/*        command server is retrieved from the system bag that is            */
/*        embedded in the response bag to the mqExecute call.                */
/*                                                                           */
/*****************************************************************************/
void CreateLocalQueue(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG reason;                         /* reason code                     */
   MQLONG compCode;                       /* completion code                 */
   MQHBAG commandBag = MQHB_UNUSABLE_HBAG; /* command bag for mqExecute      */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute     */
   MQHBAG resultBag;                      /* result bag from mqExecute       */
   MQLONG mqExecuteCC;                    /* mqExecute completion code       */
   MQLONG mqExecuteRC;                    /* mqExecute reason code           */

   printf("\nCreating Local Queue %s\n\n", qName);


   /*****************************************************************************/
   /* Create a command Bag for the mqExecute call. Exit the function if the     */
   /* create fails.                                                             */
   /*****************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &commandBag, &compCode, &reason);
   CheckCallResult("Create the command bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;

   /*****************************************************************************/
   /* Create a response Bag for the mqExecute call, exit the function if the    */
   /* create fails.                                                             */
   /*****************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
   CheckCallResult("Create the response bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;

   /*****************************************************************************/
   /* Put the name of the queue to be created into the command bag. This will   */
   /* be used by the mqExecute call.                                            */
   /*****************************************************************************/
   mqAddString(commandBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, qName, &compCode,
               &reason);
```

```
   CheckCallResult("Add q name to command bag", compCode, reason);

   /*************************************************************************/
   /* Put queue type of local into the command bag. This will be used by the  */
   /* mqExecute call.                                                       */
   /*************************************************************************/
   mqAddInteger(commandBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
   CheckCallResult("Add q type to command bag", compCode, reason);

   /*************************************************************************/
   /* Send the command to create the required local queue.                  */
   /* The mqExecute call will create the PCF structure required, send it to   */
   /* the command server and receive the reply from the command server into   */
   /* the response bag.                                                     */
   /*************************************************************************/
   mqExecute(hConn,                  /* WebSphere MQ connection handle          */
             MQCMD_CREATE_Q,         /* Command to be executed               */
             MQHB_NONE,              /* No options bag                       */
             commandBag,             /* Handle to bag containing commands    */
             responseBag,            /* Handle to bag to receive the response*/
             MQHO_NONE,              /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
             MQHO_NONE,              /* Create a dynamic q for the response  */
             &compCode,              /* Completion code from the mqExecute   */
             &reason);               /* Reason code from mqExecute call      */

   if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
   {
      printf("Please start the command server: <strmqcsv QMgrName>\n")
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
      exit(98);
   }


   /*************************************************************************/
   /* Check the result from mqExecute call and find the error if it failed.   */
   /*************************************************************************/
   if ( compCode == MQCC_OK )
      printf("Local queue %s successfully created\n", qName);
   else
   {
      printf("Creation of local queue %s failed: Completion Code = %d
               qName, compCode, reason);
      if (reason == MQRCCF_COMMAND_FAILED)
      {
         /*********************************************************************/
         /* Get the system bag handle out of the mqExecute response bag.      */
         /* This bag contains the reason from the command server why the      */
         /* command failed.                                                   */
         /*********************************************************************/
         mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &resultBag, &compCode,
                      &reason);
         CheckCallResult("Get the result bag handle", compCode, reason);

         /*********************************************************************/
         /* Get the completion code and reason code, returned by the command  */
         /* server, from the embedded error bag.                              */
         /*********************************************************************/
         mqInquireInteger(resultBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                          &compCode, &reason);
         CheckCallResult("Get the completion code from the result bag",
                          compCode, reason);
         mqInquireInteger(resultBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                          &compCode, &reason);
         CheckCallResult("Get the reason code from the result bag", compCode,
                          reason);
         printf("Error returned by the command server: Completion code = %d :
                 Reason = %d\n", mqExecuteCC, mqExecuteRC);
      }
   }
   /*************************************************************************/
   /* Delete the command bag if successfully created.                       */
   /*************************************************************************/
   if (commandBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&commandBag, &compCode, &reason);
      CheckCallResult("Delete the command bag", compCode, reason);
   }

   /*************************************************************************/
   /* Delete the response bag if successfully created.                      */
   /*************************************************************************/
```

```
   if (responseBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&responseBag, &compCode, &reason);
      CheckCallResult("Delete the response bag", compCode, reason);
   }
} /* end of CreateLocalQueue */


/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d :
                 Reason = %d\n", callText, cc, rc);

}
```

用于使用事件监视器显示事件的样本 C 程序 (amqsaiem.c)

样本 C 程序 amqsaiem.c 演示了使用 MQAI 的基本事件监视器。

```
******************************************************************************/
/*                                                                            */
/* Program name: AMQSAIEM.C                                                   */
/*                                                                            */
/* Description:  Sample C program to demonstrate a basic event monitor        */
/*               using the WebSphere MQ Admin Interface (MQAI).               */
/* Licensed Materials - Property of IBM                                       */
/*                                                                            */
/* 63H9336                                                                    */
/* (c) Copyright IBM Corp. 1999, 2024. All Rights Reserved.                   */
/*                                                                            */
/* US Government Users Restricted Rights - Use, duplication or                */
/* disclosure restricted by GSA ADP Schedule Contract with                    */
/* IBM Corp.                                                                  */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAIEM is a sample C program that demonstrates how to write a simple  */
/*    event monitor using the mqGetBag call and other MQAI calls.             */
/*                                                                            */
/*    The name of the event queue to be monitored is passed as a parameter    */
/*    to the program. This would usually be one of the system event queues:-  */
/*            SYSTEM.ADMIN.QMGR.EVENT        Queue Manager events             */
/*            SYSTEM.ADMIN.PERFM.EVENT       Performance events               */
/*            SYSTEM.ADMIN.CHANNEL.EVENT     Channel events                   */
/*            SYSTEM.ADMIN.LOGGER.EVENT      Logger events                    */
/*                                                                            */
/*    To monitor the queue manager event queue or the performance event queue,*/
/*    the attributes of the queue manager needs to be changed to enable       */
/*    these events. For more information about this, see Part 1 of the        */
/*    Programmable System Management book. The queue manager attributes can   */
/*    be changed using either MQSC commands or the MQAI interface.            */
/*    Channel events are enabled by default.                                  */
/*                                                                            */
/* Program logic                                                              */
/*    Connect to the Queue Manager.                                           */
/*    Open the requested event queue with a wait interval of 30 seconds.      */
/*    Wait for a message, and when it arrives get the message from the queue  */
/*    and format it into an MQAI bag using the mqGetBag call.                 */
/*    There are many types of event messages and it is beyond the scope of    */
/*    this sample to program for all event messages. Instead the program      */
/*    prints out the contents of the formatted bag.                           */
```

```
/*    Loop around to wait for another message until either there is an error  */
/*    or the wait interval of 30 seconds is reached.                          */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAIEM has 2 parameters - the name of the event queue to be monitored    */
/*                           - the queue manager name (optional)              */
/*                                                                            */
/******************************************************************************

/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                       /* MQI                                */
#include <cmqcfc.h>                     /* PCF                                */
#include <cmqbc.h>                      /* MQAI                               */

/******************************************************************************/
/* Macros                                                                     */
/******************************************************************************/
#if MQAT_DEFAULT == MQAT_WINDOWS_NT
  #define Int64 "I64"
#elif defined(MQ_64_BIT)
  #define Int64 "l"
#else
  #define Int64 "ll"
#endif

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);
void GetQEvents(MQHCONN, MQCHAR *);
int PrintBag(MQHBAG);
int PrintBagContents(MQHBAG, int);

/******************************************************************************/
/* Function: main                                                             */
/******************************************************************************/
int main(int argc, char *argv[])
{
   MQHCONN hConn;                       /* handle to connection               */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QM name               */
   MQLONG reason;                       /* reason code                        */
   MQLONG connReason;                   /* MQCONN reason code                 */
   MQLONG compCode;                     /* completion code                    */

   /*************************************************************************/
   /* First check the required parameters                                   */
   /*************************************************************************/
   printf("Sample Event Monitor (times out after 30 secs)\n");
   if (argc < 2)
   {
     printf("Required parameter missing - event queue to be monitored\n");
     exit(99);
   }

   /*************************************************************************/
   /* Connect to the queue manager                                         */
   /*************************************************************************/
   if (argc > 2)
     strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(QMName, &hConn, &compCode, &connReason);
   /*************************************************************************/
   /* Report the reason and stop if the connection failed                   */
   /*************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }

   /*************************************************************************/
   /* Call the routine to open the event queue and format any event messages  */
   /* read from the queue.                                                  */
   /*************************************************************************/
   GetQEvents(hConn, argv[1]);
```

```
   /*****************************************************************************/
   /* Disconnect from the queue manager if not already connected                */
   /*****************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }

   return 0;

}

/*****************************************************************************/
/*                                                                           */
/* Function: CheckCallResult                                                 */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* Input Parameters:  Description of call                                    */
/*                    Completion code                                        */
/*                    Reason code                                            */
/*                                                                           */
/* Output Parameters: None                                                   */
/*                                                                           */
/* Logic: Display the description of the call, the completion code and the   */
/*        reason code if the completion code is not successful               */
/*                                                                           */
/*****************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d : Reason = %d\n",
                 callText, cc, rc);

}

/*****************************************************************************/
/*                                                                           */
/* Function: GetQEvents                                                      */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/* Input Parameters:  Handle to the queue manager                            */
/*                    Name of the event queue to be monitored                */
/*                                                                           */
/* Output Parameters: None                                                   */
/*
/* Logic:   Open the event queue.                                            */
/*          Get a message off the event queue and format the message into    */
/*          a bag.                                                           */
/*          A real event monitor would need to be programmed to deal with    */
/*          each type of event that it receives from the queue. This is      */
/*          outside the scope of this sample, so instead, the contents of    */
/*          the bag are printed.                                             */
/*          The program waits for 30 seconds for an event message and then   */
/*          terminates if no more messages are available.                    */
/*                                                                           */
/*****************************************************************************/
void GetQEvents(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG openReason;                      /* MQOPEN reason code            */
   MQLONG reason;                          /* reason code                   */
   MQLONG compCode;                        /* completion code               */
   MQHOBJ eventQueue;                      /* handle to event queue         */

   MQHBAG eventBag = MQHB_UNUSABLE_HBAG;   /* event bag to receive event msg */
   MQOD   od = {MQOD_DEFAULT};             /* Object Descriptor             */
   MQMD   md = {MQMD_DEFAULT};             /* Message Descriptor            */
   MQGMO  gmo = {MQGMO_DEFAULT};           /* get message options           */
   MQLONG bQueueOK = 1;                    /* keep reading msgs while true  */

   /*****************************************************************************/
   /* Create an Event Bag in which to receive the event.                        */
   /* Exit the function if the create fails.                                    */
   /*****************************************************************************/
   mqCreateBag(MQCBO_USER_BAG, &eventBag, &compCode, &reason);
   CheckCallResult("Create event bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;
```

```
/*****************************************************************************/
/* Open the event queue chosen by the user                                   */
/*****************************************************************************/
strncpy(od.ObjectName, qName, (size_t)MQ_Q_NAME_LENGTH);
MQOPEN(hConn, &od, MQOO_INPUT_AS_Q_DEF+MQOO_FAIL_IF_QUIESCING, &eventQueue,
       &compCode, &openReason);
CheckCallResult("Open event queue", compCode, openReason);

/*****************************************************************************/
/* Set the GMO options to control the action of the get message from the     */
/* queue.                                                                    */
/*****************************************************************************/
gmo.WaitInterval = 30000;               /* 30 second wait for message      */
gmo.Options = MQGMO_WAIT + MQGMO_FAIL_IF_QUIESCING + MQGMO_CONVERT;
gmo.Version = MQGMO_VERSION_2;          /* Avoid need to reset Message ID  */
gmo.MatchOptions = MQMO_NONE;           /* and Correlation ID after every  */
                                /* mqGetBag
/*****************************************************************************/
/* If open fails, we cannot access the queue and must stop the monitor.      */
/*****************************************************************************/
if (compCode != MQCC_OK)
   bQueueOK = 0;

/*****************************************************************************/
/* Main loop to get an event message when it arrives                         */
/*****************************************************************************/
while (bQueueOK)
{
  printf("\nWaiting for an event\n");

  /***************************************************************************/
  /* Get the message from the event queue and convert it into the event      */
  /* bag.                                                                    */
  /***************************************************************************/
  mqGetBag(hConn, eventQueue, &md, &gmo, eventBag, &compCode, &reason);

  /***************************************************************************/
  /* If get fails, we cannot access the queue and must stop the monitor.     */
  /***************************************************************************/
  if (compCode != MQCC_OK)
  {
     bQueueOK = 0;

     /************************************************************************/
     /* If get fails because no message available then we have timed out,    */
     /* so report this, otherwise report an error.                           */
     /************************************************************************/
     if (reason == MQRC_NO_MSG_AVAILABLE)
     {
        printf("No more messages\n");
     }
     else
     {
        CheckCallResult("Get bag", compCode, reason);
     }
  }

  /***************************************************************************/
  /* Event message read - Print the contents of the event bag                */
  /***************************************************************************/
  else
  {
    if ( PrintBag(eventBag) )
        printf("\nError found while printing bag contents\n");

  }  /* end of msg found */
} /* end of main loop */
/*****************************************************************************/
/* Close the event queue if successfully opened                              */
/*****************************************************************************/
if (openReason == MQRC_NONE)
{
   MQCLOSE(hConn, &eventQueue, MQCO_NONE, &compCode, &reason);
   CheckCallResult("Close event queue", compCode, reason);
}

/*****************************************************************************/
/* Delete the event bag if successfully created.                             */
/*****************************************************************************/
if (eventBag != MQHB_UNUSABLE_HBAG)
{
   mqDeleteBag(&eventBag, &compCode, &reason);
```

```
      CheckCallResult("Delete the event bag", compCode, reason);
   }

} /* end of GetQEvents */

/******************************************************************************/
/*                                                                            */
/* Function: PrintBag                                                         */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Calls PrintBagContents to display the contents of the bag.          */
/*                                                                            */
/******************************************************************************

int PrintBag(MQHBAG dataBag)
{
    int errors;

    printf("\n");
    errors = PrintBagContents(dataBag, 0);
    printf("\n");

    return errors;
}

/******************************************************************************/
/*                                                                            */
/* Function: PrintBagContents                                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                    Indentation level of bag                                */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Count the number of items in the bag                                */
/*        Obtain selector and item type for each item in the bag.             */
/*        Obtain the value of the item depending on item type and display the */
/*        index of the item, the selector and the value.                      */
/*        If the item is an embedded bag handle then call this function again */
/*        to print the contents of the embedded bag increasing the            */
/*        indentation level.                                                  */
/*                                                                            */
/******************************************************************************/
int PrintBagContents(MQHBAG dataBag, int indent)
{
   /***************************************************************************/
   /* Definitions                                                             */
   /***************************************************************************/
   #define LENGTH 500                     /* Max length of string to be read*/
   #define INDENT 4                       /* Number of spaces to indent     */
                                          /* embedded bag display           */


   /***************************************************************************/
   /* Variables                                                               */
   /***************************************************************************/
   MQLONG  itemCount;                     /* Number of items in the bag     */
   MQLONG  itemType;                      /* Type of the item               */
   int     i;                             /* Index of item in the bag       */
   MQCHAR  stringVal[LENGTH+1];           /* Value if item is a string      */
   MQBYTE  byteStringVal[LENGTH];         /* Value if item is a byte string */
   MQLONG  stringLength;                  /* Length of string value         */
   MQLONG  ccsid;                         /* CCSID of string value          */
   MQINT32 iValue;                        /* Value if item is an integer    */
   MQINT64 i64Value;                      /* Value if item is a 64-bit      */
                                          /* integer                        */
   MQLONG  selector;                      /* Selector of item               */
   MQHBAG  bagHandle;                     /* Value if item is a bag handle  */
   MQLONG  reason;                        /* reason code                    */
   MQLONG  compCode;                      /* completion code                */
   MQLONG  trimLength;                    /* Length of string to be trimmed */
```

```
int     errors = 0;                     /* Count of errors found          */
char    blanks[] = "                      "; /* Blank string used to      */
                                           /* indent display             */

/*****************************************************************************/
/* Count the number of items in the bag                                      */
/*****************************************************************************/
mqCountItems(dataBag, MQSEL_ALL_SELECTORS, &itemCount, &compCode, &reason);

if (compCode != MQCC_OK)
   errors++;
else
{
   printf("
   printf("
   printf("
}

/*****************************************************************************/
/* If no errors found, display each item in the bag                          */
/*****************************************************************************/
if (!errors)
{
   for (i = 0; i < itemCount; i++)
   {

       /*********************************************************************/
       /* First inquire the type of the item for each item in the bag       */
       /*********************************************************************/
       mqInquireItemInfo(dataBag,             /* Bag handle                 */
                         MQSEL_ANY_SELECTOR,  /* Item can have any selector*/
                         i,                   /* Index position in the bag */
                         &selector,           /* Actual value of selector  */
                                              /* returned by call          */
                         &itemType,           /* Actual type of item       */
                                              /* returned by call          */
                         &compCode,           /* Completion code           */
                         &reason);            /* Reason Code               */

       if (compCode != MQCC_OK)
          errors++;

       switch(itemType)
       {
       case MQITEM_INTEGER:
            /*****************************************************************/
            /* Item is an integer. Find its value and display its index,   */
            /* selector and value.                                           */
            /*****************************************************************/
            mqInquireInteger(dataBag,         /* Bag handle                  */
                             MQSEL_ANY_SELECTOR, /* Allow any selector     */
                             i,               /* Index position in the bag */
                             &iValue,         /* Returned integer value
                             &compCode,       /* Completion code           */
                             &reason);        /* Reason Code               */

            if (compCode != MQCC_OK)
               errors++;
            else
               printf("%.*s  %-2d     %-4d     (%d)\n",
                       indent, blanks, i, selector, iValue);
            break

       case MQITEM_INTEGER64:
            /*****************************************************************/
            /* Item is a 64-bit integer. Find its value and display its    */
            /* index, selector and value.                                    */
            /*****************************************************************/
            mqInquireInteger64(dataBag,       /* Bag handle                  */
                               MQSEL_ANY_SELECTOR, /* Allow any selector   */
                               i,             /* Index position in the bag */
                               &i64Value,     /* Returned integer value    */
                               &compCode,     /* Completion code           */
                               &reason);      /* Reason Code               */

            if (compCode != MQCC_OK)
               errors++;
            else
               printf("%.*s  %-2d     %-4d     (%"Int64"d)\n",
                       indent, blanks, i, selector, i64Value);
            break;
```

```
        case MQITEM_STRING:
             /*****************************************************************/
             /* Item is a string. Obtain the string in a buffer, prepare    */
             /* the string for displaying and display the index, selector,  */
             /* string and Character Set ID.                                  */
             /*****************************************************************/
             mqInquireString(dataBag,          /* Bag handle                  */
                             MQSEL_ANY_SELECTOR, /* Allow any selector      */
                             i,                /* Index position in the bag */
                             LENGTH,           /* Maximum length of buffer  */
                             stringVal,        /* Buffer to receive string  */
                             &stringLength,    /* Actual length of string   */
                             &ccsid,           /* Coded character set id    */
                             &compCode,        /* Completion code           */
                             &reason);         /* Reason Code               */

             /*****************************************************************/
             /* The call can return a warning if the string is too long for */
             /* the output buffer and has been truncated, so only check     */
             /* explicitly for call failure.                                  */
             /*****************************************************************/
             if (compCode == MQCC_FAILED)
                 errors++;
             else
             {
                /*************************************************************/
                /* Remove trailing blanks from the string and terminate with*/
                /* a null. First check that the string should not have been */
                /* longer than the maximum buffer size allowed.              */
                /*************************************************************/
                if (stringLength > LENGTH)
                   trimLength = LENGTH;
                else
                   trimLength = stringLength;
                mqTrim(trimLength, stringVal, stringVal, &compCode, &reason);
                printf("%.*s  %-2d     %-4d     '%s' %d\n",
                        indent, blanks, i, selector, stringVal, ccsid);
             }
             break;

         case MQITEM_BYTE_STRING:
             /*****************************************************************/
             /* Item is a byte string. Obtain the byte string in a buffer,  */
             /* prepare the byte string for displaying and display the      */
             /* index, selector and string.                                   */
             /*****************************************************************/
             mqInquireByteString(dataBag,      /* Bag handle                  */
                                 MQSEL_ANY_SELECTOR, /* Allow any selector  */
                                 i,            /* Index position in the bag */
                                 LENGTH,       /* Maximum length of buffer  */
                                 byteStringVal, /* Buffer to receive string */
                                 &stringLength, /* Actual length of string  */
                                 &compCode,    /* Completion code           */
                                 &reason);     /* Reason Code

             /*****************************************************************/
             /* The call can return a warning if the string is too long for */
             /* the output buffer and has been truncated, so only check     */
             /* explicitly for call failure.                                  */
             /*****************************************************************/
             if (compCode == MQCC_FAILED)
                 errors++;
             else
             {
                printf("%.*s  %-2d     %-4d     X'",
                       indent, blanks, i, selector);

                for (i = 0 ; i < stringLength ; i++)
                   printf("

                printf("'\n");
             }
             break;

         case MQITEM_BAG:
             /*****************************************************************/
             /* Item is an embedded bag handle, so call the PrintBagContents*/
             /* function again to display the contents.                       */
             /*****************************************************************/
             mqInquireBag(dataBag,             /* Bag handle                  */
                          MQSEL_ANY_SELECTOR,  /* Allow any selector        */
```

```
                              i,                    /* Index position in the bag */
                              &bagHandle,           /* Returned embedded bag hdle*/
                              &compCode,            /* Completion code           */
                              &reason);             /* Reason Code               */

                  if (compCode != MQCC_OK)
                     errors++;
                  else
                  {
                     printf("%.*s  %-2d     %-4d     (%d)\n", indent, blanks, i,
                             selector, bagHandle);
                     if (selector == MQHA_BAG_HANDLE)
                        printf("
                     else
                        printf("
                     PrintBagContents(bagHandle, indent+INDENT);
                  }
                  break;

               default:
                    printf("
               }
         }
      }
      return errors;
}
```

用于查询通道对象的样本 C 程序 (amqsaicl.c)

样本 C 程序 `amqsaicl.c` 使用 MQAI 查询通道对象。

```
 /********************************************************************************/
/*                                                                              */
/* Program name: AMQSAICL.C                                                     */
/*                                                                              */
/* Description:  Sample C program to inquire channel objects                    */
/*               using the WebSphere MQ Administration Interface (MQAI)         */
/*                                                                              */
/* <N_OCO_COPYRIGHT>                                                            */
/* Licensed Materials - Property of IBM                                         */
/*                                                                              */
/* 63H9336                                                                      */
/* (c) Copyright IBM Corp. 2008, 2024. All Rights Reserved.                     */
/*                                                                              */
/* US Government Users Restricted Rights - Use, duplication or                  */
/* disclosure restricted by GSA ADP Schedule Contract with                      */
/* IBM Corp.                                                                    */
/* <NOC_COPYRIGHT>                                                              */
/********************************************************************************/
/*                                                                              */
/* Function:                                                                    */
/*    AMQSAICL is a sample C program that demonstrates how to inquire           */
/*    attributes of the local queue manager using the MQAI interface. In        */
/*    particular, it inquires all channels and their types.                     */
/*                                                                              */
/*     - A PCF command is built from items placed into an MQAI administration   */
/*       bag.                                                                   */
/*       These are:-                                                            */
/*            - The generic channel name "*"                                    */
/*            - The attributes to be inquired. In this sample we just want      */
/*              name and type attributes                                        */
/*                                                                              */
/*     - The mqExecute MQCMD_INQUIRE_CHANNEL call is executed.                  */
/*       The call generates the correct PCF structure.                          */
/*       The default options to the call are used so that the command is sent   */
/*       to the SYSTEM.ADMIN.COMMAND.QUEUE.                                     */
/*       The reply from the command server is placed on a temporary dynamic     */
/*       queue.                                                                 */
/*       The reply from the MQCMD_INQUIRE_CHANNEL is read from the              */
/*       temporary queue and formatted into the response bag.                   */
/*                                                                              */
/*     - The completion code from the mqExecute call is checked and if there    */
/*       is a failure from the command server, then the code returned by the    */
/*       command server is retrieved from the system bag that has been          */
/*       embedded in the response bag to the mqExecute call.                    */
/*                                                                              */
/* Note: The command server must be running.                                    */
/*                                                                              */
/********************************************************************************/
```

```
/*                                                                          */
/* AMQSAICL has 2 parameter - the queue manager name (optional)             */
/*                          - output file (optional) default varies         */
/****************************************************************************/

/****************************************************************************/
/* Includes                                                                 */
/****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#if (MQAT_DEFAULT == MQAT_OS400)
#include <recio.h>
#endif

#include <cmqc.h>                       /* MQI                              */
#include <cmqcfc.h>                     /* PCF                              */
#include <cmqbc.h>                      /* MQAI                             */
#include <cmqxc.h>                      /* MQCD                             */

/****************************************************************************/
/* Function prototypes                                                      */
/****************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);

/****************************************************************************/
/* DataTypes                                                                */
/****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
typedef _RFILE OUTFILEHDL;
#else
typedef FILE OUTFILEHDL;
#endif

/****************************************************************************/
/* Constants                                                                */
/****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
const struct
{
  char name[9];
} ChlTypeMap[9] =
{
  "*SDR     ",    /* MQCHT_SENDER    */
  "*SVR     ",    /* MQCHT_SERVER    */
  "*RCVR    ",    /* MQCHT_RECEIVER  */
  "*RQSTR   ",    /* MQCHT_REQUESTER */
  "*ALL     ",    /* MQCHT_ALL       */
  "*CLTCN   ",    /* MQCHT_CLNTCONN  */
  "*SVRCONN ",    /* MQCHT_SVRCONN   */
  "*CLUSRCVR",    /* MQCHT_CLUSRCVR  */
  "*CLUSSDR "     /* MQCHT_CLUSSDR   */
};
#else
const struct
{
  char name[9];
} ChlTypeMap[9] =
{
  "sdr      ",    /* MQCHT_SENDER    */
  "svr      ",    /* MQCHT_SERVER    */
  "rcvr     ",    /* MQCHT_RECEIVER  */
  "rqstr    ",    /* MQCHT_REQUESTER */
  "all      ",    /* MQCHT_ALL       */
  "cltconn  ",    /* MQCHT_CLNTCONN  */
  "svrcn    ",    /* MQCHT_SVRCONN   */
  "clusrcvr ",    /* MQCHT_CLUSRCVR  */
  "clussdr  "     /* MQCHT_CLUSSDR   */
};
#endif

/****************************************************************************/
/* Macros                                                                   */
/****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
  #define OUTFILE "QTEMP/AMQSAICL(AMQSAICL)"
  #define OPENOUTFILE(hdl, fname) \
    (hdl) = _Ropen((fname),"wr, rtncode=Y");
  #define CLOSEOUTFILE(hdl) \
    _Rclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
```

```
    _Rwrite((hdl),(buf),(buflen));

#elif (MQAT_DEFAULT == MQAT_UNIX)
  #define OUTFILE "/tmp/amqsaicl.txt"
  #define OPENOUTFILE(hdl, fname) \
    (hdl) = fopen((fname),"w");
  #define CLOSEOUTFILE(hdl) \
    fclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    fwrite((buf),(buflen),1,(hdl)); fflush((hdl));

#else
  #define OUTFILE "amqsaicl.txt"
  #define OPENOUTFILE(fname) \
    fopen((fname),"w");
  #define CLOSEOUTFILE(hdl) \
    fclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    fwrite((buf),(buflen),1,(hdl)); fflush((hdl));

#endif

#define ChlType2String(t) ChlTypeMap[(t)-1].name

/********************************************************************************/
/* Function: main                                                               */
/********************************************************************************/
int main(int argc, char *argv[])
{
   /*****************************************************************************/
   /* MQAI variables                                                            */
   /*****************************************************************************/
   MQHCONN hConn;                          /* handle to MQ connection         */
   MQCHAR qmName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */
   MQHBAG adminBag = MQHB_UNUSABLE_HBAG;   /* admin bag for mqExecute         */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG cAttrsBag;                       /* bag containing chl attributes   */
   MQHBAG errorBag;                        /* bag containing cmd server error */
   MQLONG mqExecuteCC;                     /* mqExecute completion code       */
   MQLONG mqExecuteRC;                     /* mqExecute reason code           */
   MQLONG chlNameLength;                   /* Actual length of chl name       */
   MQLONG chlType;                         /* Channel type                    */
   MQLONG i;                               /* loop counter                    */
   MQLONG numberOfBags;                    /* number of bags in response bag  */
   MQCHAR chlName[MQ_OBJECT_NAME_LENGTH+1];/* name of chl extracted from bag  */
   MQCHAR OutputBuffer[100];               /* output data buffer              */
   OUTFILEHDL *outfp = NULL;               /* output file handle              */

   /*****************************************************************************/
   /* Connect to the queue manager                                              */
   /*****************************************************************************/
   if (argc &gt; 1)
      strncpy(qmName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(qmName, &hConn;, &compCode;, &connReason;);

   /*****************************************************************************/
   /* Report the reason and stop if the connection failed.                      */
   /*****************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("Queue Manager connection", compCode, connReason);
      exit( (int)connReason);
   }

   /*****************************************************************************/
   /* Open the output file                                                      */
   /*****************************************************************************/
   if (argc &gt; 2)
   {
     OPENOUTFILE(outfp, argv[2]);
   }
   else
   {
     OPENOUTFILE(outfp, OUTFILE);
   }

   if(outfp == NULL)
   {
     printf("Could not open output file.\n");
```

```
  goto MOD_EXIT;
}
/*****************************************************************************/
/* Create an admin bag for the mqExecute call                                */
/*****************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &adminBag;, &compCode;, &reason;);
CheckCallResult("Create admin bag", compCode, reason);

/*****************************************************************************/
/* Create a response bag for the mqExecute call                              */
/*****************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag;, &compCode;, &reason;);
CheckCallResult("Create response bag", compCode, reason);

/*****************************************************************************/
/* Put the generic channel name into the admin bag                           */
/*****************************************************************************/
mqAddString(adminBag, MQCACH_CHANNEL_NAME, MQBL_NULL_TERMINATED, "*",
            &compCode;, &reason;);
CheckCallResult("Add channel name", compCode, reason);

/*****************************************************************************/
/* Put the channel type into the admin bag                                   */
/*****************************************************************************/
mqAddInteger(adminBag, MQIACH_CHANNEL_TYPE, MQCHT_ALL, &compCode;, &reason;);
CheckCallResult("Add channel type", compCode, reason);

/*****************************************************************************/
/* Add an inquiry for various attributes                                     */
/*****************************************************************************/
mqAddInquiry(adminBag, MQIACH_CHANNEL_TYPE, &compCode;, &reason;);
CheckCallResult("Add inquiry", compCode, reason);

/*****************************************************************************/
/* Send the command to find all the channel names and channel types.         */
/* The mqExecute call creates the PCF structure required, sends it to        */
/* the command server, and receives the reply from the command server into */
/* the response bag. The attributes are contained in system bags that are  */
/* embedded in the response bag, one set of attributes per bag.              */
/*****************************************************************************/
mqExecute(hConn,                    /* MQ connection handle                  */
          MQCMD_INQUIRE_CHANNEL,    /* Command to be executed                */
          MQHB_NONE,                /* No options bag                        */
          adminBag,                 /* Handle to bag containing commands     */
          responseBag,              /* Handle to bag to receive the response*/
          MQHO_NONE,                /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,                /* Create a dynamic q for the response   */
          &compCode;,               /* Completion code from the mqexecute    */
          &reason;);                /* Reason code from mqexecute call       */

/*****************************************************************************/
/* Check the command server is started. If not exit.                         */
/*****************************************************************************/
if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName="">\n");
   goto MOD_EXIT;
}

/*****************************************************************************/
/* Check the result from mqExecute call. If successful find the channel      */
/* types for all the channels. If failed find the error.                     */
/*****************************************************************************/
if ( compCode == MQCC_OK )                      /* Successful mqExecute      */
{
  /***************************************************************************/
  /* Count the number of system bags embedded in the response bag from the */
  /* mqExecute call. The attributes for each channel are in separate bags. */
  /***************************************************************************/
  mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags;,
               &compCode;, &reason;);
  CheckCallResult("Count number of bag handles", compCode, reason);

  for ( i=0; i<numberOfbags; i++)
  {
    /*************************************************************************/
    /* Get the next system bag handle out of the mqExecute response bag.     */
    /* This bag contains the channel attributes                              */
    /*************************************************************************/
    mqInquireBag(responseBag, MQHA_BAG_HANDLE, i, &cAttrsbag,
                 &compCode, &reason);
    CheckCallResult("Get the result bag handle", compCode, reason);
```

```
      /*********************************************************************/
      /* Get the channel name out of the channel attributes bag            */
      /*********************************************************************/
      mqInquireString(cAttrsBag, MQCACH_CHANNEL_NAME, 0, MQ_OBJECT_NAME_LENGTH,
                      chlName, &chlNameLength, NULL, &compCode, &reason);
      CheckCallResult("Get channel name", compCode, reason);

      /*********************************************************************/
      /* Get the channel type out of the channel attributes bag            */
      /*********************************************************************/

    mqInquireInteger(cAttrsBag, MQIACH_CHANNEL_TYPE, MQIND_NONE, &chlType,
                     &compCode, &reason);
      CheckCallResult("Get type", compCode, reason);

      /*********************************************************************/
      /* Use mqTrim to prepare the channel name for printing.              */
      /* Print the result.                                                 */
      /*********************************************************************/
      mqTrim(MQ_CHANNEL_NAME_LENGTH, chlName, chlName, &compCode, &reason);
      sprintf(OutputBuffer, "%-20s%-9s", chlName, ChlType2String(chlType));
      WRITEOUTFILE(outfp,OutputBuffer,29)
    }
  }

  else                                              /* Failed mqExecute     */
  {
    printf("Call to get channel attributes failed: Cc = %ld : Rc = %ld\n",
                compCode, reason);
    /*********************************************************************/
    /* If the command fails get the system bag handle out of the mqexecute   */
    /* response bag.This bag contains the reason from the command server     */
    /* why the command failed.                                           */
    /*********************************************************************/
    if (reason == MQRCCF_COMMAND_FAILED)
    {
      mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &errorBag,
                   &compCode, &reason);
      CheckCallResult("Get the result bag handle", compCode, reason);

      /*********************************************************************/
      /* Get the completion code and reason code, returned by the command    */
      /* server, from the embedded error bag.                              */
      /*********************************************************************/
      mqInquireInteger(errorBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                       &compCode, &reason );
      CheckCallResult("Get the completion code from the result bag",
                      compCode, reason);
      mqInquireInteger(errorBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                        &compCode, &reason);
      CheckCallResult("Get the reason code from the result bag",
                      compCode, reason);
      printf("Error returned by the command server: Cc = %ld : Rc = %ld\n",
               mqExecuteCC, mqExecuteRC);
    }
  }

MOD_EXIT:
  /*********************************************************************/
  /* Delete the admin bag if successfully created.                     */
  /*********************************************************************/
  if (adminBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&adminBag, &compCode, &reason);
     CheckCallResult("Delete the admin bag", compCode, reason);
  }

  /*********************************************************************/
  /* Delete the response bag if successfully created.                  */
  /*********************************************************************/
  if (responseBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&responseBag, &compCode, &reason);
     CheckCallResult("Delete the response bag", compCode, reason);
  }

  /*********************************************************************/
  /* Disconnect from the queue manager if not already connected        */
  /*********************************************************************/
  if (connReason != MQRC_ALREADY_CONNECTED)
  {
```

```
      MQDISC(&hConn, &compCode, &reason);
       CheckCallResult("Disconnect from Queue Manager", compCode, reason);
   }

   /******************************************************************************/
   /* Close the output file if open                                              */
   /******************************************************************************/
   if(outfp != NULL)
      CLOSEOUTFILE(outfp);

   return 0;
}


/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %ld : Reason = %ld\n", callText,
                cc, rc);
}
```

用于查询队列和打印信息的样本 C 程序 (amqsailq.c)

样本 C 程序 `amqsailq.c` 使用 MQAI 查询本地队列的当前深度。

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAILQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to inquire the current depth of the local   */
/*               queues using the WebSphere MQ Administration Interface (MQAI)*/
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999, 2024.                          */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAILQ is a sample C program that demonstrates how to inquire         */
/*    attributes of the local queue manager using the MQAI interface. In      */
/*    particular, it inquires the current depths of all the local queues.     */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI administration  */
/*       bag.                                                                 */
/*       These are:-                                                          */
/*            - The generic queue name "*"                                    */
/*            - The type of queue required. In this sample we want to         */
/*              inquire local queues.                                         */
/*            - The attribute to be inquired. In this sample we want the      */
/*              current depths.                                               */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_INQUIRE_Q.     */
/*       The call generates the correct PCF structure.                        */
/*       The default options to the call are used so that the command is sent */
/*       to the SYSTEM.ADMIN.COMMAND.QUEUE.                                   */
```

```
/*       The reply from the command server is placed on a temporary dynamic   */
/*       queue.                                                               */
/*       The reply from the MQCMD_INQUIRE_Q command is read from the          */
/*       temporary queue and formatted into the response bag.                 */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server, then the code returned by      */
/*       command server is retrieved from the system bag that has been        */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/*     - If the call is successful, the depth of each local queue is placed   */
/*       in system bags embedded in the response bag of the mqExecute call.   */
/*       The name and depth of each queue is obtained from each of the bags   */
/*       and the result displayed on the screen.                              */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAILQ has 1 parameter - the queue manager name (optional)               */
/*                                                                            */
/******************************************************************************/

/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                       /* MQI                                */
#include <cmqcfc.h>                     /* PCF                                */
#include <cmqbc.h>                      /* MQAI                               */

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);

/******************************************************************************/
/* Function: main                                                             */
/******************************************************************************/
int main(int argc, char *argv[])
{
   /***************************************************************************/
   /* MQAI variables                                                          */
   /***************************************************************************/
   MQHCONN hConn;                       /* handle to WebSphere MQ connection   */
   MQCHAR qmName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG reason;                       /* reason code                        */
   MQLONG connReason;                   /* MQCONN reason code                 */
   MQLONG compCode;                     /* completion code                    */
   MQHBAG adminBag = MQHB_UNUSABLE_HBAG;   /* admin bag for mqExecute         */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG qAttrsBag;                    /* bag containing q attributes        */
   MQHBAG errorBag;                     /* bag containing cmd server error */
   MQLONG mqExecuteCC;                  /* mqExecute completion code          */
   MQLONG mqExecuteRC;                  /* mqExecute reason code              */
   MQLONG qNameLength;                  /* Actual length of q name            */
   MQLONG qDepth;                       /* depth of queue                     */
   MQLONG i;                            /* loop counter                       */
   MQLONG numberOfBags;                 /* number of bags in response bag  */
   MQCHAR qName[MQ_Q_NAME_LENGTH+1];    /* name of queue extracted from bag*/


   printf("Display current depths of local queues\n\n");

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 1)
      strncpy(qmName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(qmName, &hConn, &compCode, &connReason);

   /***************************************************************************/
   /* Report the reason and stop if the connection failed.                    */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("Queue Manager connection", compCode, connReason
);
```

```
   exit( (int)connReason);
}

/******************************************************************************/
/* Create an admin bag for the mqExecute call                                 */
/******************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &adminBag, &compCode, &reason);
CheckCallResult("Create admin bag", compCode, reason);
/******************************************************************************/
/* Create a response bag for the mqExecute call                               */
/******************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
CheckCallResult("Create response bag", compCode, reason);

/******************************************************************************/
/* Put the generic queue name into the admin bag                              */
/******************************************************************************/
mqAddString(adminBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, "*",
            &compCode, &reason);
CheckCallResult("Add q name", compCode, reason);

/******************************************************************************/
/* Put the local queue type into the admin bag                                */
/******************************************************************************/
mqAddInteger(adminBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
CheckCallResult("Add q type", compCode, reason);

/******************************************************************************/
/* Add an inquiry for current queue depths                                    */
/******************************************************************************/
mqAddInquiry(adminBag, MQIA_CURRENT_Q_DEPTH, &compCode, &reason);
CheckCallResult("Add inquiry", compCode, reason);

/******************************************************************************/
/* Send the command to find all the local queue names and queue depths.       */
/* The mqExecute call creates the PCF structure required, sends it to         */
/* the command server, and receives the reply from the command server into    */
/* the response bag. The attributes are contained in system bags that are     */
/* embedded in the response bag, one set of attributes per bag.               */
/******************************************************************************/
mqExecute(hConn,                    /* WebSphere MQ connection handle           */
          MQCMD_INQUIRE_Q,          /* Command to be executed                   */
          MQHB_NONE,                /* No options bag                           */
          adminBag,                 /* Handle to bag containing commands        */
          responseBag,              /* Handle to bag to receive the response*/
          MQHO_NONE,                /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,                /* Create a dynamic q for the response      */
          &compCode,                /* Completion code from the mqExecute       */
          &reason);                 /* Reason code from mqExecute call          */


/******************************************************************************/
/* Check the command server is started. If not exit.                          */
/******************************************************************************/
if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName>\n");
   MQDISC(&hConn, &compCode, &reason);
   CheckCallResult("Disconnect from Queue Manager", compCode, reason);
   exit(98);
}

/******************************************************************************/
/* Check the result from mqExecute call. If successful find the current       */
/* depths of all the local queues. If failed find the error.                  */
/******************************************************************************/
if ( compCode == MQCC_OK )                    /* Successful mqExecute         */
{
  /****************************************************************************/
  /* Count the number of system bags embedded in the response bag from the */
  /* mqExecute call. The attributes for each queue are in a separate bag.  */
  /****************************************************************************/
  mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags, &compCode,
               &reason);
  CheckCallResult("Count number of bag handles", compCode, reason);

  for ( i=0; i<numberOfBags; i++)
  {
    /**************************************************************************/
    /* Get the next system bag handle out of the mqExecute response bag.      */
    /* This bag contains the queue attributes                                 */
    /**************************************************************************/
```

```
      mqInquireBag(responseBag, MQHA_BAG_HANDLE, i, &qAttrsBag, &compCode,
                   &reason);
      CheckCallResult("Get the result bag handle", compCode, reason);

      /*********************************************************************/
      /* Get the queue name out of the queue attributes bag                */
      /*********************************************************************/
      mqInquireString(qAttrsBag, MQCA_Q_NAME, 0, MQ_Q_NAME_LENGTH, qName,
                      &qNameLength, NULL, &compCode, &reason);
      CheckCallResult("Get queue name", compCode, reason);

      /*********************************************************************/
      /* Get the depth out of the queue attributes bag                     */
      /*********************************************************************/
      mqInquireInteger(qAttrsBag, MQIA_CURRENT_Q_DEPTH, MQIND_NONE, &qDepth,
                       &compCode, &reason);
      CheckCallResult("Get depth", compCode, reason);

      /*********************************************************************/
      /* Use mqTrim to prepare the queue name for printing.                */
      /* Print the result.                                                 */
      /*********************************************************************/
      mqTrim(MQ_Q_NAME_LENGTH, qName, qName, &compCode, &reason)
      printf("%4d  %-48s\n", qDepth, qName);
    }
  }

  else                                          /* Failed mqExecute     */
  {
    printf("Call to get queue attributes failed: Completion Code = %d :
            Reason = %d\n", compCode, reason);

    /*********************************************************************/
    /* If the command fails get the system bag handle out of the mqExecute   */
    /* response bag. This bag contains the reason from the command server    */
    /* why the command failed.                                           */
    /*********************************************************************/
    if (reason == MQRCCF_COMMAND_FAILED)
    {
      mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &errorBag, &compCode,
                   &reason);
      CheckCallResult("Get the result bag handle", compCode, reason);

      /*********************************************************************/
      /* Get the completion code and reason code, returned by the command     */
      /* server, from the embedded error bag.                              */
      /*********************************************************************/
      mqInquireInteger(errorBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                       &compCode, &reason );
      CheckCallResult("Get the completion code from the result bag",
                       compCode, reason);
      mqInquireInteger(errorBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                        &compCode, &reason);
      CheckCallResult("Get the reason code from the result bag",
                       compCode, reason);
      printf("Error returned by the command server: Completion Code = %d :
              Reason = %d\n", mqExecuteCC, mqExecuteRC);
    }
  }

  /*********************************************************************/
  /* Delete the admin bag if successfully created.                     */
  /*********************************************************************/
  if (adminBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&adminBag, &compCode, &reason);
     CheckCallResult("Delete the admin bag", compCode, reason);
  }

  /*********************************************************************/
  /* Delete the response bag if successfully created.                  */
  /*********************************************************************/
  if (responseBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&responseBag, &compCode, &reason);
     CheckCallResult("Delete the response bag", compCode, reason);
  }

  /*********************************************************************/
  /* Disconnect from the queue manager if not already connected        */
  /*********************************************************************/
  if (connReason != MQRC_ALREADY_CONNECTED)
```

```
  {
     MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("Disconnect from queue manager", compCode, reason);
  }
  return 0;
}

*****************************************************************************/
*                                                                           */
* Function: CheckCallResult                                                 */
*                                                                           */
*****************************************************************************/
*                                                                           */
* Input Parameters:  Description of call                                    */
*                    Completion code                                        */
*                    Reason code                                            */
*                                                                           */
* Output Parameters: None                                                   */
*                                                                           */
* Logic: Display the description of the call, the completion code and the   */
*        reason code if the completion code is not successful               */
*                                                                           */
*****************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
  if (cc != MQCC_OK)
        printf("%s failed: Completion Code = %d : Reason = %d\n",
                callText, cc, rc);
}
```

有关配置 IBM WebSphere MQ 的提示和技巧

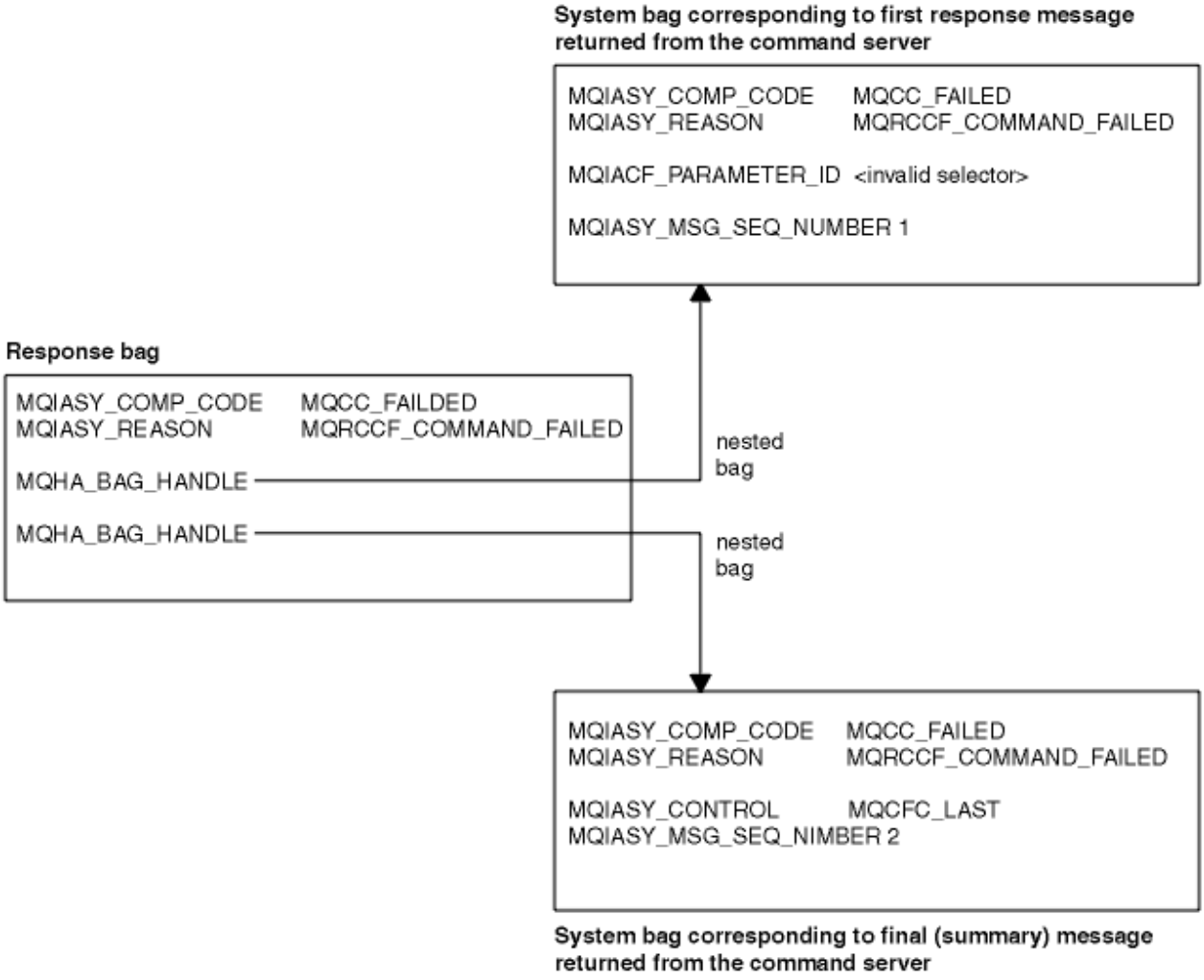
使用 MQAI 时的编程提示和技巧。

MQAI 使用 PCF 消息将管理命令发送到命令服务器，而不是直接处理命令服务器本身。 以下是使用 MQAI 配置 WebSphere MQ 的一些提示:

- WebSphere MQ 中的字符串为空白，填充为固定长度。 通过使用 C ，通常可以将以 null 结束的字符串作为输入参数提供给 WebSphere MQ 编程接口。
- 要清除字符串属性的值，请将其设置为单个空白而不是空字符串。
- 请提前考虑要更改的属性，并仅查询这些属性。
- 无法更改某些属性，例如，队列名称或通道类型。 请确保尝试仅更改可修改的那些属性。 请参阅特定 PCF 更改对象的必需参数和可选参数的列表。 请参阅 可编程命令格式的定义。
- 如果 MQAI 调用失败，那么会将失败的一些详细信息返回到响应包。 然后，可以在可由选择器 MQHA_BAG_HANDLE 访问的嵌套包中找到更多详细信息。 例如，如果 mqExecute 调用失败，并且原因码为 MQRCCF_COMMAND_FAILED ，那么将在响应包中返回此信息。 此原因码的可能原因是指定的选择器对于命令消息类型无效，并且在可由包句柄访问的嵌套包中找到此详细信息。

  有关 MQExecute 的更多信息，请参阅 第 50 页的『使用 mqExecute 调用将管理命令发送到命令服务器』

  下图显示了此场景:



高级 MQAI 主题

关于建立索引，数据转换和使用消息描述符的信息

- 建立索引

  在从包中替换或除去现有数据项以保留插入顺序时，将使用索引。 可在 第 39 页的『在 MQAI 中建立索引』中找到有关建立索引的完整详细信息。

- 数据转换

  MQAI 数据包中包含的字符串可以包含在各种编码字符集中，并且可以使用 mqSet 整数调用来转换这些字符串。 可在 第 40 页的『MQAI 中的数据转换』中找到有关数据转换的完整详细信息。

- 使用消息描述符

  MQAI 生成在创建数据包时设置为初始值的消息描述符。 可以在 第 41 页的『在 MQAI 中使用消息描述符』中找到使用消息描述符的完整详细信息。

在 *MQAI* 中建立索引

当从包中替换或除去现有数据项时，将使用索引。 有三种类型的索引，可以轻松检索数据项。

数据包中数据项内的每个选择器和值都有三个关联的索引号:

- 相对于具有相同选择器的其他项的索引。
- 相对于项所属的选择器 (用户或系统) 类别的索引。
- 相对于包中所有数据项 (用户和系统) 的索引。

这允许按用户选择器和/或系统选择器建立索引，如 第 40 页的图 1 中所示。

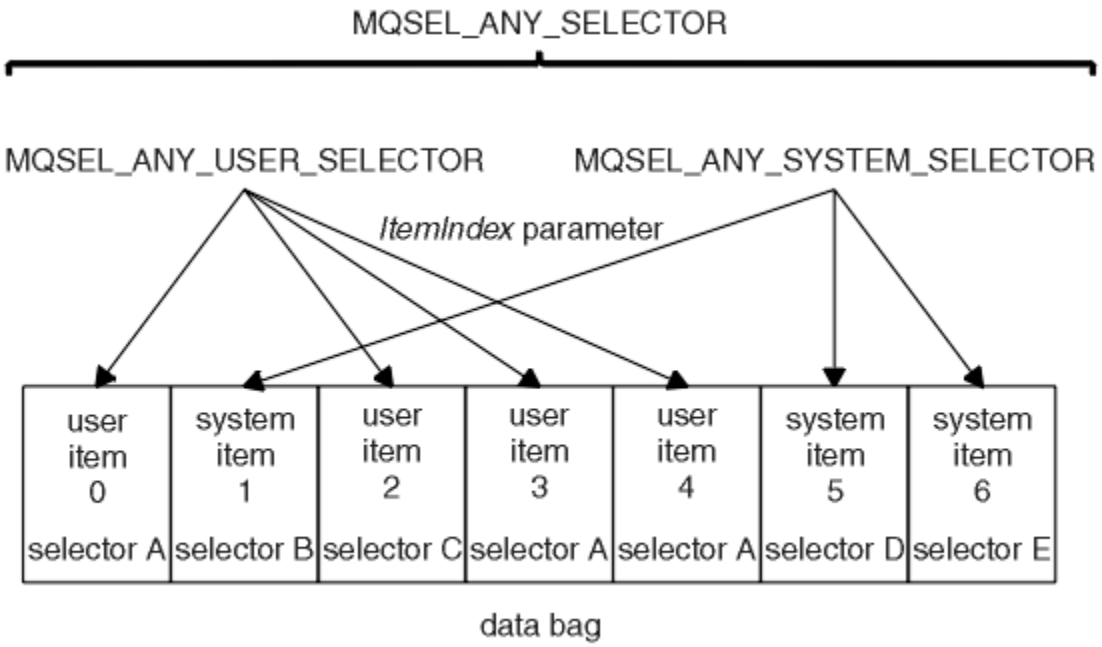


图 *1:* 建立索引

在图 第 40 页的图 1 中，用户项 3 (选择器 A) 可由以下索引对引用:

Selector	***ItemIndex***
选择器 A	1
MQSEL_ANY_USER_SELECTOR	2
MQSEL_ANY_SELECTOR	3

索引以零为基础，类似于 C 中的数组; 如果出现 "n" ，那么索引的范围从 0 到 "n-1" ，没有间隔。

当从包中替换或除去现有数据项时，将使用索引。 以此方式使用时，将保留插入顺序，但可能会影响其他数据项的索引。 有关此操作的示例，请参阅 更改包中的信息 和 删除数据项。

三种类型的索引允许轻松检索数据项。 例如，如果包中有三个特定选择器实例，那么 mqCount 项调用可以计算该选择器的实例数，而 mqInquire* 调用可以同时指定选择器和索引以仅查询这些值。 这对于具有值列表 (例如通道上的某些出口) 的属性很有用。

MQAI 中的数据转换

MQAI 数据包中包含的字符串可以包含在各种编码字符集中。 可以使用 mqSet 整数调用来转换这些字符串。

与 PCF 消息一样， MQAI 数据包中包含的字符串可以包含在各种编码字符集中。 通常， PCF 消息中的所有字符串都使用相同的编码字符集; 即，与队列管理器相同的集合。

数据包中的每个字符串项都包含两个值: 字符串本身和 CCSID。 添加到包中的字符串可从 mqAddString 或 mqSetString 调用的 *Buffer* 参数获取。 从包含 MQIASY_CODED_CHAR_SET_ID 选择器的系统项获取 CCSID。 这称为 包 *CCSID* ，可以使用 mqSet 整数调用进行更改。

当查询数据包中包含的字符串的值时， CCSID 是来自调用的输出参数。

第 40 页的表 2 显示了将数据包转换为消息时应用的规则，反之亦然:

表 *2: CCSID* 处理			
MQAI 调用	**CCSID**	**要调用的输入**	**要调用的输出**
mqBagToBuffer	包 CCSID (1)	已忽略	无变化

表 *2: CCSID* 处理 *(继续)*

MQAI 调用	**CCSID**	**要调用的输入**	**要调用的输出**
mqBagToBuffer	包中的字符串 CCSID	已使用	无变化
mqBagToBuffer	缓冲区中的字符串 CCSID	不适用	从包中的字符串 CCSID 复制
mqBufferToBag	包 CCSID (1)	已忽略	无变化
mqBufferToBag	缓冲区中的字符串 CCSID	已使用	无变化
mqBufferToBag	包中的字符串 CCSID	不适用	从缓冲区中的字符串 CCSID 复制
mqPut 包	MQMD CCSID	已使用	未更改 (2)
mqPut 包	包 CCSID (1)	已忽略	无变化
mqPut 包	包中的字符串 CCSID	已使用	无变化
mqPut 包	发送的消息中的字符串 CCSID	不适用	从包中的字符串 CCSID 复制
mqGet 包	MQMD CCSID	用于消息的数据转换	设置为返回数据的 CCSID (3)
mqGet 包	包 CCSID (1)	已忽略	无变化
mqGet 包	消息中的字符串 CCSID	已使用	无变化
mqGet 包	包中的字符串 CCSID	不适用	从消息中的字符串 CCSID 复制
mqExecute	请求包 CCSID	用于请求消息的 MQMD (4)	无变化
mqExecute	应答包 CCSID	用于应答消息 (4) 的数据转换	设置为返回数据的 CCSID (3)
mqExecute	请求包中的字符串 CCSID	用于请求消息	无变化
mqExecute	应答包中的字符串 CCSID	不适用	从应答消息中的字符串 CCSID 复制

注意:

1. 包 CCSID 是具有选择器 MQIASY_CODED_CHAR_SET_ID 的系统项。
2. MQCCSI_Q_MGR 更改为实际队列管理器 CCSID。
3. 如果请求数据转换，那么返回的数据的 CCSID 与输出值相同。 如果未请求数据转换，那么返回的数据的 CCSID 与消息值相同。 请注意，如果请求数据转换但失败，那么不会返回任何消息。
4. 如果 CCSID 是 MQCCSI_DEFAULT ，那么将使用队列管理器的 CCSID。

在 *MQAI* 中使用消息描述符

创建数据包时， MQAI 生成的消息描述符将设置为初始值。

从具有选择器 MQIASY_TYPE 的系统项获取 PCF 命令类型。 创建数据包时，将根据您创建的数据包类型来设置此项的初始值:

表 *3: PCF* 命令类型

袋子的类型	**MQIASY_TYPE 项的初始值**
MQCBO_ADMIN_BAG	MQCFT_COMMAND

表 3: PCF 命令类型 (继续)	
袋子的类型	**MQIASY_TYPE 项的初始值**
MQCBO_COMMAND_BAG	MQCFT_COMMAND
MQCBO_ *	MQCFT_USER

MQAI 生成消息描述符时， *Format* 和 *MsgType* 参数中使用的值取决于具有选择器 MQIASY_TYPE 的系统项的值，如 第 41 页的表 3 中所示。

表 4: MQMD 的格式和 MsgType 参数		
PCF 命令类型	**格式**	**MsgType**
MQCFT_COMMAND	MQFMT_ADMIN	MQMT_REQUEST
MQCFT_REPORT	MQFMT_ADMIN	MQMT_REPORT
MQCFT_RESPONSE	MQFMT_ADMIN	MQMT_REPLY
MQCFT_TRACE_ROUTE	MQFMT_ADMIN	MQMT_DATAGRAM
MQCFT_EVENT	MQFMT_EVENT	MQMT_DATAGRAM
MQCFT_ *	MQFMT_PCF	MQMT_DATAGRAM

第 42 页的表 4 显示了如果创建管理包或命令包，那么消息描述符的 *Format* 为 MQFMT_ADMIN ， 而 *MsgType* 为 MQMT_REQUEST。 这适用于在期望返回响应时发送到命令服务器的 PCF 请求消息。

消息描述符中的其他参数采用 第 42 页的表 5 中显示的值。

表 5: 消息描述符值	
参数	**值**
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	请参阅 第 42 页的表 4
Expiry	30 秒 (注 第 43 页的『1』)
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	依赖于包 CCSID (注 第 43 页的『2』)
Format	请参阅 第 42 页的表 4
Priority	MQPRI_PRIORITY_AS_Q_DEF
Persistence	MQPER_NOT_PERSISTENT
MsgId	MQMI_NONE
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	请参阅注释 第 43 页的『3』
ReplyToQMgr	空白

表 5: 消息描述符值 (继续)	
参数	**值**
注意: 1. 可以使用 **OptionsBag** 参数在 **mqExecute** 调用上覆盖此值。 有关此信息，请参阅 **mqExecute**。 2. 请参阅 第 40 页的『MQAI 中的数据转换』。 3. 类型为 **MQMT_REQUEST** 的消息的用户指定的应答队列或 MQAI 生成的临时动态队列的名称。 否则为空白。	

数据包

数据包是使用 MQAI 处理对象的属性或参数的一种方法。

数据包

- 数据包包含零个或多个 数据项。 这些数据项在放入包中时在包中进行排序。 这称为 插入顺序。 每个数据项都包含一个 选择器 ，用于标识数据项以及该数据项的 值 ，该数据项可以是整数， 64 位整数，整数过滤器，字符串，字符串过滤器，字节字符串，字节字符串过滤器或另一个包的句柄。 第 45 页的『数据项』 中的详细信息描述了数据项

  有两种类型的选择器: 用户选择器 和 系统选择器。 这些在 MQAI 选择器中进行了描述。 选择器通常是唯一的，但对于同一个选择器可以有多个值。 在这种情况下， 索引 标识所需的特定选择器实例。 第 39 页的『在 MQAI 中建立索引』 中描述了索引。

  图 1 中显示了这些概念的层次结构。

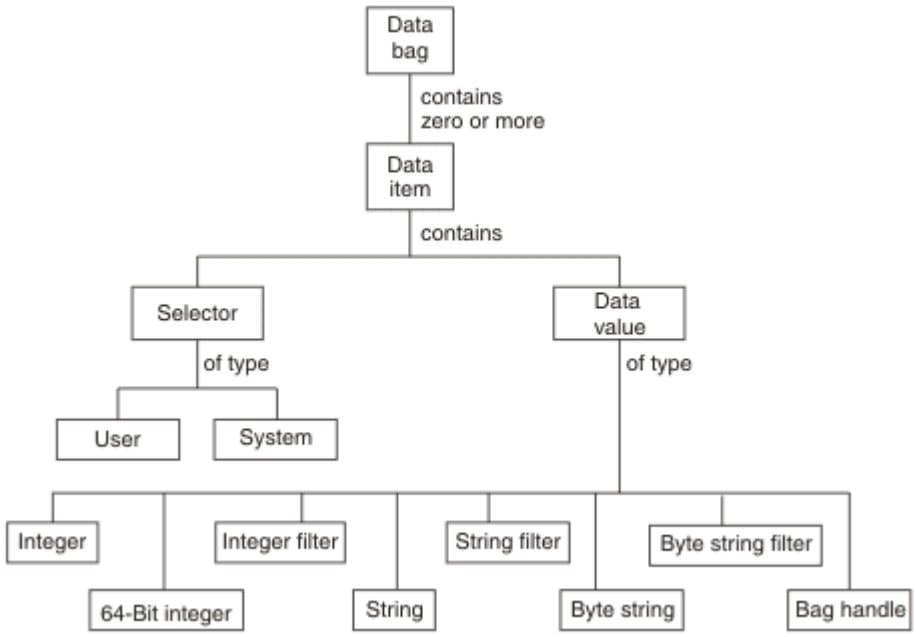


图 2: *MQAI* 概念的层次结构

在上一段中对层次结构进行了说明。

数据包类型

您可以根据要执行的任务来选择要创建的数据包类型:

用户包 (user bag)
用于用户数据的简单包。

管理包 (administration bag)
为用于通过向命令服务器发送管理消息来管理 WebSphere MQ 对象的数据创建的包。 管理包自动暗示某些选项，如 第 44 页的『创建和删除数据包』中所述。

命令包 (command bag)
还为用于管理 WebSphere MQ 对象的命令创建了一个包。 但是，与管理包不同，命令包不会自动暗示某些选项，尽管这些选项可用。 有关选项的更多信息，请参阅 第 44 页的『创建和删除数据包』。

组包
用于存放一组分组数据项的包。 组包不能用于管理 WebSphere MQ 对象。

此外，当从命令服务器返回应答消息并将其放入用户的输出包中时， MQAI 会创建 **系统包** 。 用户无法修改系统包。

使用数据包的不同使用方式在本主题中列出:

使用数据包

以下列表中显示了使用数据包的不同方法:

- 您可以创建和删除数据包 第 44 页的『创建和删除数据包』。
- 您可以使用数据包 第 45 页的『放置和接收数据包』在应用程序之间发送数据。
- 您可以将数据项添加到数据包 第 46 页的『将数据项添加到包』。
- 您可以在数据包 第 46 页的『向包添加查询命令』中添加查询命令。
- 您可以在数据包 第 47 页的『在数据包中查询』中进行查询。
- 您可以对数据包 第 49 页的『对数据项进行计数』中的数据项进行计数。
- 您可以在数据包 第 47 页的『更改包中的信息』中更改信息。
- 您可以清除数据包 第 48 页的『使用 mqClearBag 调用清除包』。
- 您可以截断数据包 第 48 页的『使用 mqTruncateBag 调用截断包』。
- 您可以转换包和缓冲区 第 49 页的『转换包和缓冲区』。

创建和删除数据包

创建数据包

要使用 MQAI ，请首先使用 mqCreateBag 调用创建数据包。 作为此调用的输入，您提供一个或多个选项来控制包的创建。

MQCreateBag 调用的 *Options* 参数允许您选择是创建用户包，命令包，组包还是管理包。

要创建用户包，命令包或组包，您可以选择一个或多个进一步的选项以:

- 当包中出现两个或更多个相邻的相同选择器时，请使用列表表单。
- 将数据项添加到 PCF 消息时对其进行重新排序，以确保参数的顺序正确。 有关数据项的更多信息，请参阅 第 45 页的『数据项』。
- 检查您添加到包中的项的用户选择器值。

管理包会自动暗示这些选项。

数据袋由其手柄标识。 包句柄是从 mqCreate 包返回的，必须在使用该数据包的所有其他调用上提供。

有关 mqCreateBag 调用的完整描述，请参阅 mqCreateBag。

删除数据包

还必须使用 mqDeleteBag 调用删除用户创建的任何数据包。 例如，如果在用户代码中创建了包，那么还必须在用户代码中删除该包。

系统包由 MQAI 自动创建和删除。 有关此操作的更多信息，请参阅 第 50 页的『使用 mqExecute 调用将管理命令发送到命令服务器』。 用户代码无法删除系统包。

有关 mqDeleteBag 调用的完整描述，请参阅 mqDeleteBag。

放置和接收数据包

还可以通过使用 mqPutBag 和 mqGetBag 调用来放置和获取数据包，在应用程序之间发送数据。 这使 MQAI 能够处理缓冲区而不是应用程序。 mqPutBag 调用会将指定包的内容转换为 PCF 消息，并将该消息发送到指定队列，而 mqGetBag 调用会从指定队列中除去该消息并将其转换回数据包。 因此， mqPutBag 调用等效于后跟 MQPUT 的 mqBagToBuffer 调用，而 mqGetBag 等效于后跟 mqBufferToBag 的 MQGET 调用。

有关在特定队列中发送和接收 PCF 消息的更多信息，请参阅 第 11 页的『在指定队列中发送和接收 PCF 消息』

注: 如果选择使用 mqGetBag 调用，那么消息中的 PCF 详细信息必须正确; 如果不正确，那么将不会返回相应的错误结果和 PCF 消息。

数据项

数据项用于在创建数据包时填充这些数据包。 这些数据项可以是用户或系统项。

这些用户项包含用户数据，例如要管理的对象的属性。 系统项应用于对生成的消息进行更多控制: 例如，生成消息头。 有关系统项的更多信息，请参阅 第 45 页的『系统项』。

数据项的类型

创建数据包后，可以使用整数或字符串项填充该数据包。 您可以查询所有三种类型的项。

数据项可以是整数或字符串项。 以下是 MQAI 中可用的数据项类型:

- 整数
- 64 位整数
- 整数过滤器
- 字符串
- 字符串过滤器
- 字节字符串
- 字节字符串过滤器
- 袋柄

使用数据项

以下是使用数据项的方式:

- 第 49 页的『对数据项进行计数』.
- 第 49 页的『删除数据项』.
- 第 46 页的『将数据项添加到包』.
- 第 46 页的『过滤和查询数据项』.

系统项

系统项可用于:

- 生成 PCF 头。 系统项可以控制 PCF 命令标识，控制选项，消息序号和命令类型。
- 数据转换。 系统项处理包中字符串项的字符集标识。

与所有数据项一样，系统项由选择器和值组成。 有关这些选择器及其对象的信息，请参阅 MQAI 选择器。

系统项是唯一的。 一个或多个系统项可由系统选择器标识。 每个系统选择器只有一个实例。

可以修改大多数系统项 (请参阅 第 47 页的『更改包中的信息』) ，但用户无法更改包创建选项。 无法删除系统项。 (请参阅 第 49 页的『删除数据项』。)

将数据项添加到包

创建数据包时，可以使用数据项填充该数据包。 这些数据项可以是用户或系统项。 有关数据项的更多信息，请参阅 第 45 页的『数据项』。

MQAI 允许您将整数项， 64 位整数项，整数过滤器项，字符串项，字符串过滤器，字节字符串项和字节字符串过滤器项添加到包中，如 第 46 页的图 3 中所示。 这些项由选择器标识。 通常，一个选择器仅标识一个项，但并非总是如此。 如果包中已存在具有指定选择器的数据项，那么会将该选择器的其他实例添加到包的末尾。

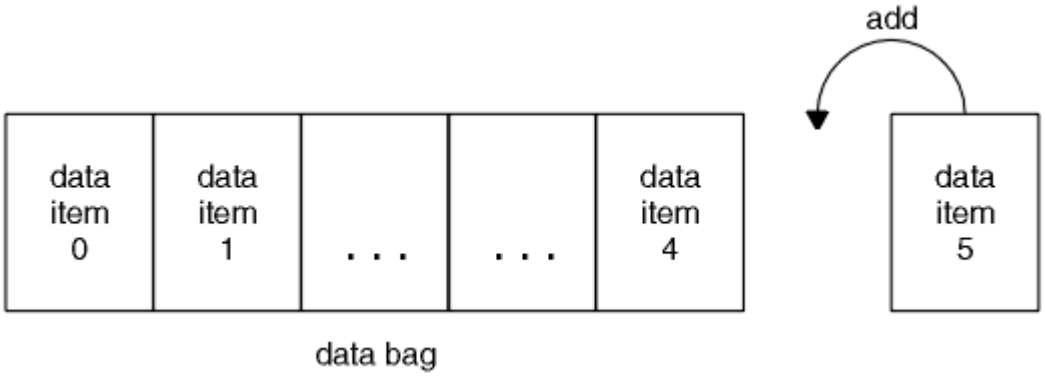


图 *3:* 添加数据项

使用 mqAdd* 调用将数据项添加到包中:

- 要添加整数项，请使用 mqAddInteger 调用，如 mqAddInteger 中所述
- 要添加 64 位整数项，请使用 mqAddInteger64 调用，如 mqAddInteger64 中所述
- 要添加整数过滤器项，请使用 mqAddIntegerFilter 调用，如 mqAddIntegerFilter 中所述
- 要添加字符串项，请使用 mqAddString 调用，如 mqAddString 中所述
- 要添加字符串过滤器项，请使用 mqAddStringFilter 调用，如 mqAddStringFilter 中所述
- 要添加字节字符串项，请使用 mqAddByteString 调用，如 mqAddByteString 中所述
- 要添加字节字符串过滤器项，请使用 mqAddByteString 过滤器调用，如 mqAddByteString 过滤器 中所述

有关将数据项添加到包中的更多信息，请参阅 第 45 页的『系统项』。

向包添加查询命令

mqAdd 查询调用用于将查询命令添加到包中。 该调用专门用于管理目的，因此只能与管理包一起使用。 它允许您指定要从 WebSphere MQ 查询的属性的选择器。

有关 mqAdd 查询调用的完整描述，请参阅 mqAdd 查询。

过滤和查询数据项

使用 MQAI 查询 WebSphere MQ 对象的属性时，可以通过两种方式控制返回到程序的数据。

- 您可以 **过滤** 使用 mqAddInteger 和 mqAddString 调用返回的数据。 此方法允许您指定 *Selector* 和 *ItemValue* 对，例如:

```
mqAddInteger(inputbag, MQIA_Q_TYPE, MQQT_LOCAL)
```

此示例指定队列类型 (*Selector*) 必须是本地 (*ItemValue*) ，并且此规范必须与您要查询的对象 (在本例中是队列) 的属性匹配。

可过滤的其他属性对应于可在 第 8 页的『可编程命令格式简介』中找到的 PCF Inquire * 命令。 例如，要查询通道的属性，请参阅本产品文档中的 "查询通道" 命令。 "查询通道" 命令的 "必需参数" 和 "可选参数" 标识可用于过滤的选择器。

- 您可以使用 mqAdd 查询调用来 **查询** 对象的特定属性。 这将指定您感兴趣的选择器。 如果未指定选择器，那么将返回对象的所有属性。

以下是过滤和查询队列属性的示例:

```
/* Request information about all queues */
mqAddString(adminbag, MQCA_Q_NAME, "*")

/* Filter attributes so that local queues only are returned */
mqAddInteger(adminbag, MQIA_Q_TYPE, MQQT_LOCAL)

/* Query the names and current depths of the local queues */
mqAddInquiry(adminbag, MQCA_Q_NAME)
mqAddInquiry(adminbag, MQIA_CURRENT_Q_DEPTH)

/* Send inquiry to the command server and wait for reply */
mqExecute(MQCMD_INQUIRE_Q, ...)
```

有关过滤和查询数据项的更多示例，请参阅 第 18 页的『使用 MQAI 的示例』。

在数据包中查询

您可以查询:

- 使用 mqInquire 整数调用的整数项的值。 请参阅 mqInquireInteger。
- 使用 mqInquireInteger64 调用的 64 位整数项的值。 请参阅 mqInquireInteger64。
- 使用 mqInquireIntegerFilter 调用的整数过滤器项的值。 请参阅 mqInquireIntegerFilter。
- 使用 mqInquire 字符串调用的字符串项的值。 请参阅 mqInquire 字符串。
- 使用 mqInquireStringFilter 调用的字符串过滤器项的值。 请参阅 mqInquireStringFilter。
- 使用 mqInquireByteString 调用的字节字符串项的值。 请参阅 mqInquireByteString。
- 使用 mqInquireByteString 过滤器调用的字节字符串过滤器项的值。 请参阅 mqInquireByteString 过滤器。
- 使用 mqInquireBag 调用的包句柄的值。 请参阅 mqInquire 包。

您还可以使用 mqInquireItemInfo 调用来查询特定项的类型 (整数， 64 位整数，整数过滤器，字符串，字符串过滤器，字节字符串，字节字符串过滤器或包句柄)。 请参阅 mqInquireItemInfo。

更改包中的信息

MQAI 允许您使用 mqSet* 调用来更改包中的信息。 您可以:

1. 修改包中的数据项。 索引允许通过标识要修改的项的出现来替换参数的单个实例 (请参阅 第 47 页的图 4)。

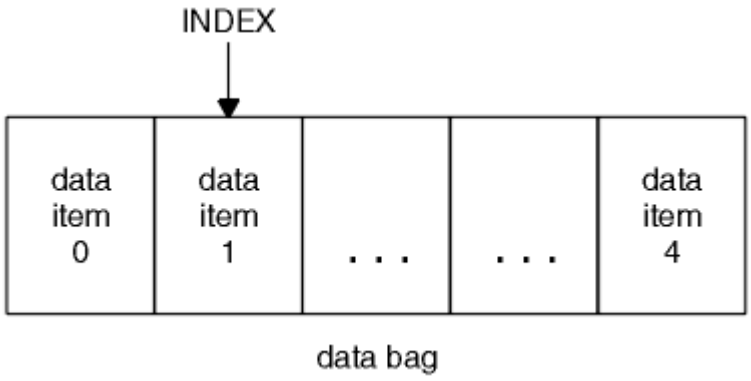


图 *4:* 修改单个数据项

2. 删除指定选择器的所有现有实例，并将新实例添加到包的末尾。 (请参阅 第 48 页的图 5。) 特殊索引值允许替换参数的 **所有** 实例。

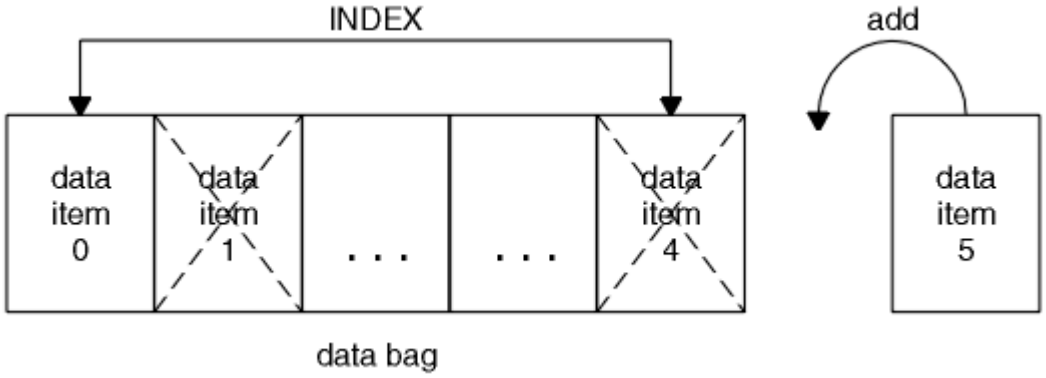


图 *5:* 修改所有数据项

注: 索引会保留包中的插入顺序，但会影响其他数据项的索引。

mqSet 整数调用允许您修改包中的整数项。 mqSetInteger64 调用允许您修改 64 位整数项。 mqSetIntegerFilter 调用允许您修改整数过滤器项。 mqSet 字符串调用允许您修改字符串项。 mqSetStringFilter 调用允许您修改字符串过滤器项。 mqSetByteString 调用允许您修改字节字符串项。 mqSetByteString 过滤器调用允许您修改字节字符串过滤器项。 或者，您可以使用这些调用来删除指定选择器的所有现有实例，并在包的末尾添加新实例。 数据项可以是用户项或系统项。

有关这些调用的完整描述，请参阅:

- mqSet 整数
- mqSetInteger64
- mqSetIntegerFilter
- mqSet 字符串
- mqSetStringFilter
- mqSetByteString
- mqSetByteString 过滤器

使用 *mqClearBag* 调用清除包

mqClearBag 调用从用户包中除去所有用户项，并将系统项重置为其初始值。 还会删除包中包含的系统包。

有关 mqClearBag 调用的完整描述，请参阅 mqClearBag。

使用 *mqTruncateBag* 调用截断包

mqTruncateBag 调用通过从包末尾删除项 (从最近添加的项开始) 来减少用户包中的用户项数。 例如，当使用相同的头信息生成多条消息时，可以使用它。

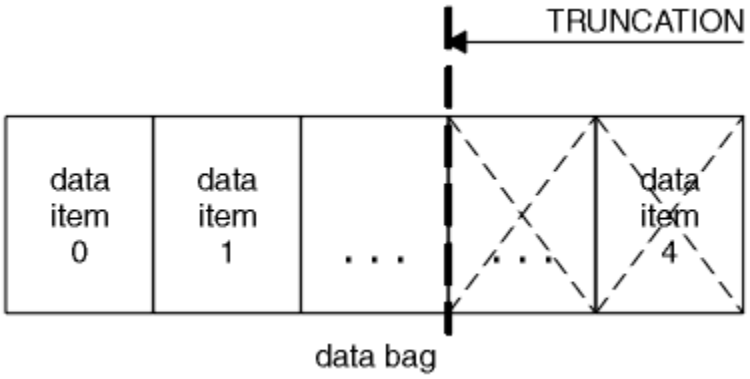


图 *6:* 截断包

有关 mqTruncateBag 调用的完整描述，请参阅 mqTruncateBag。

转换包和缓冲区

要在应用程序之间发送数据，首先将消息数据放在包中。 然后，使用 mqBagToBuffer 调用将包中的数据转换为 PCF 消息。 使用 MQPUT 调用将 PCF 消息发送到所需队列。 如图 第 49 页的图 7 所示。 有关 mqBagToBuffer 调用的完整描述，请参阅 mqBagToBuffer。

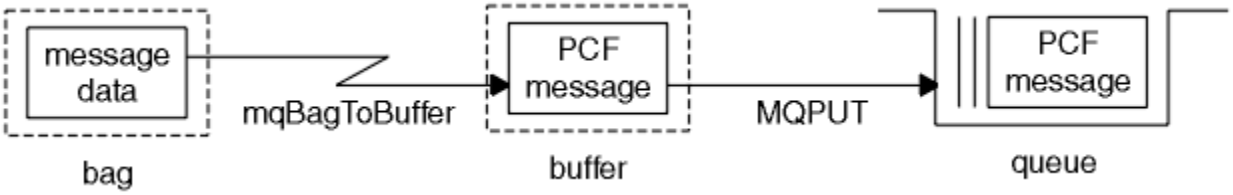


图 *7:* 将包转换为 *PCF* 消息

要接收数据，将使用 MQGET 调用将消息接收到缓冲区中。 然后，使用 mqBufferToBag 调用将缓冲区中的数据转换为包，前提是缓冲区包含有效的 PCF 消息。 如图 第 49 页的图 8 所示。 有关 mqBufferToBag 调用的完整描述，请参阅 mqBufferToBag。

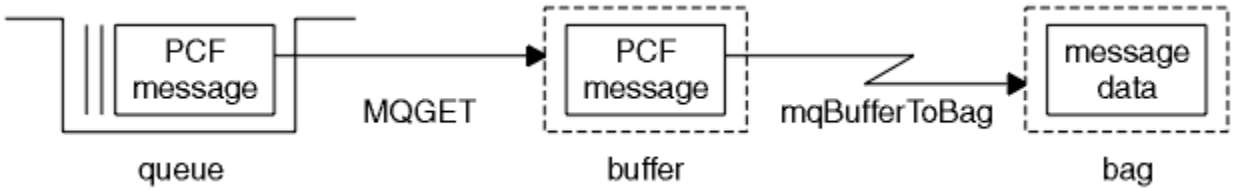


图 *8:* 将 *PCF* 消息转换为包格式

对数据项进行计数

mqCount 项调用对存储在数据包中的用户项数和/或系统项数进行计数，并返回此数字。 例如，`mqCountItems(Bag, 7, ...)`返回包中选择器为 7 的项数。 它可以按单个选择器，按用户选择器，按系统选择器或按所有选择器对项进行计数。

注: 此调用计算数据项的数量，而不是包中唯一选择器的数量。 选择器可以多次出现，因此包中的唯一选择器可能少于数据项。

有关 mqCount 项调用的完整描述，请参阅 mqCount 项。

删除数据项

您可以通过多种方式从包中删除项目。 您可以:

- 从包中除去一个或多个用户项。 有关详细信息，请参阅 第 49 页的『使用 mqDeleteItem 调用从包中删除数据项』。
- 从包中删除 **所有** 用户项，即 清除 一个包。 有关详细信息，请参阅 第 48 页的『使用 mqClearBag 调用清除包』。
- 从包的末尾删除用户项，即 截断 包。 有关详细信息，请参阅 第 48 页的『使用 mqTruncateBag 调用截断包』。

使用 *mqDeleteItem* 调用从包中删除数据项

mqDelete 项调用从包中除去一个或多个用户项。 索引用于删除以下任一项:

1. 一次出现指定的选择器。 (请参阅 第 50 页的图 9。)

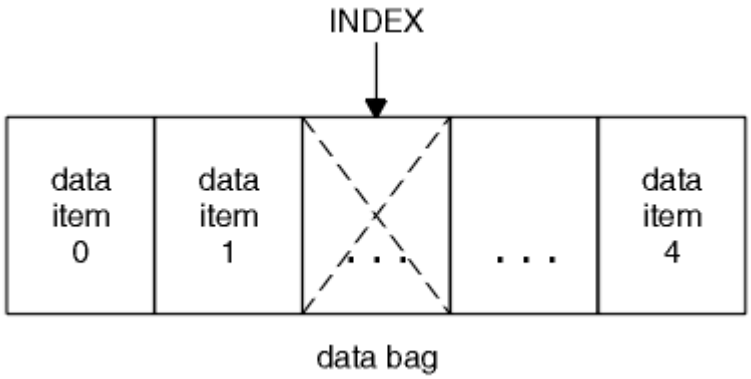


图 *9:* 删除单个数据项

或者

2. 指定选择器的所有实例。 (请参阅 第 50 页的图 10。)

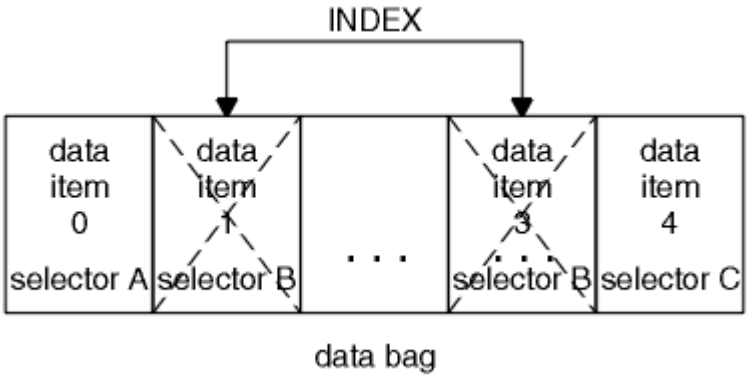


图 *10:* 删除所有数据项

注: 索引会保留包中的插入顺序，但会影响其他数据项的索引。 例如， mqDeleteItem 调用不会保留跟在已删除项后面的数据项的索引值，因为将重组这些索引以填补从已删除项中保留的间隔。

有关 mqDelete 项调用的完整描述，请参阅 mqDelete 项。

使用 mqExecute 调用将管理命令发送到命令服务器

创建并填充数据包后，可以使用 mqExecute 调用将管理命令消息发送到队列管理器的命令服务器。 这将处理与命令服务器的交换，并在包中返回响应。

创建并填充数据包后，可以将管理命令消息发送到队列管理器的命令服务器。 执行此操作的最简单方法是使用 mqExecute 调用。 mqExecute 调用将管理命令消息作为非持久消息发送，并等待任何响应。 响应在响应包中返回。 这些属性可能包含与多个 WebSphere MQ 对象或一系列 PCF 错误响应消息 (例如) 相关的属性的相关信息。 因此，响应包只能包含返回码，或者可以包含 嵌套包。

响应消息将放入系统创建的系统包中。 例如，对于有关对象名称的查询，将创建一个系统包以保存这些对象名称，并将该包插入到用户包中。 然后将这些包的句柄插入到响应包中，选择器 MQHA_BAG_HANDLE 可访问嵌套包。 如果未删除系统包，那么该系统包将保留在存储器中，直到删除响应包为止。

第 51 页的图 11 中显示了 嵌套 的概念。

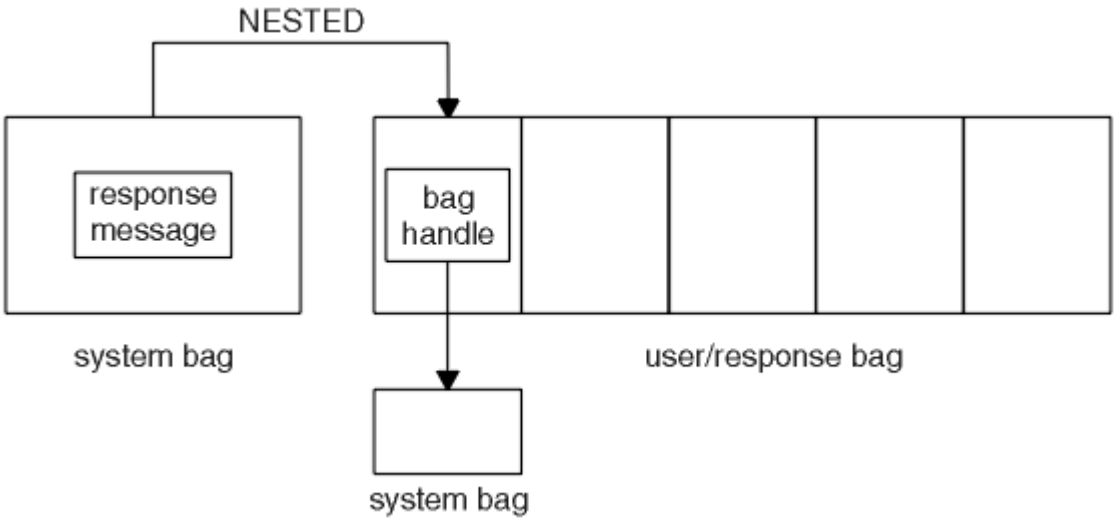


图 *11:* 嵌套

作为 mqExecute 调用的输入，必须提供:

- MQI 连接句柄。
- 要执行的命令。 这应该是 MQCMD_ * 值之一。

  注: 如果 MQAI 无法识别此值，那么仍会接受该值。 但是，如果使用 mqAdd 查询调用将值插入到包中，那么此参数必须是 MQAI 识别的 INQUIRE 命令。 即，参数的格式应为 MQCMD_INQUIRE_ *。
- (可选) 包含控制呼叫处理的选项的包的手柄。 这也是您可以指定 MQAI 应该等待每条应答消息的最大时间 (以毫秒为单位)。
- 管理包的句柄，其中包含要发出的管理命令的详细信息。
- 接收应答消息的响应包的句柄。

以下是可选的:

- 要放置管理命令的队列的对象句柄。

  如果未指定对象句柄，那么会将管理命令放在 SYSTEM.ADMIN.COMMAND.QUEUE 属于当前连接的队列管理器。 这是缺省值。
- 要放置应答消息的队列的对象句柄。

  您可以选择将应答消息放在 MQAI 自动创建的动态队列上。 创建的队列仅在调用期间存在，并且在退出 mqExecute 调用时被 MQAI 删除。

有关使用 mqExecute 调用的示例，请参阅 示例代码

使用 IBM WebSphere MQ Explorer 进行管理

IBM WebSphere MQ Explorer 允许您仅从运行 Windows 或 Linux (x86 和 x86-64 平台) 的计算机对网络执行本地或远程管理。

IBM WebSphere MQ for Windows 和 IBM WebSphere MQ for Linux (x86 和 x86-64 平台) 提供了称为 IBM WebSphere MQ Explorer 的管理接口，以执行管理任务作为使用控制或 MQSC 命令的替代方法。 比较命令集 显示了您可以使用 IBM WebSphere MQ Explorer 执行的操作。

IBM WebSphere MQ Explorer 允许您通过将 IBM WebSphere MQ Explorer 指向您感兴趣的队列管理器和集群，从运行 Windows 或 Linux (x86-64 平台) 的计算机对网络执行本地或远程管理。 第 53 页的『远程队列管理器』中描述了可使用 IBM WebSphere MQ Explorer 管理的 IBM WebSphere MQ 平台和级别。

要配置远程 IBM WebSphere MQ 队列管理器以便 IBM WebSphere MQ Explorer 可以对其进行管理，请参阅第 53 页的『必备软件和定义』。

它允许您执行任务，通常与在 Windows 或 Linux (x86 和 x86-64 平台) 系统域中本地或远程设置和微调 IBM WebSphere MQ 的工作环境相关联。

在 Linux 上，如果有多个 Eclipse 安装，那么 IBM WebSphere MQ Explorer 可能无法启动。 如果发生这种情况，请使用与用于其他 Eclipse 安装的用户标识不同的用户标识来启动 IBM WebSphere MQ Explorer 。

在 Linux 上，要成功启动 IBM WebSphere MQ Explorer，您必须能够将文件写入主目录，并且主目录必须存在。

您可以使用 IBM WebSphere MQ Explorer 执行的操作

这是可以使用 IBM WebSphere MQ Explorer 执行的任务的列表。

通过 IBM WebSphere MQ Explorer ， 您可以:

- 创建和删除队列管理器 (仅在本地机器上)。
- 启动和停止队列管理器 (仅在本地机器上)。
- 定义，显示和变更 WebSphere MQ 对象 (例如队列和通道) 的定义。
- 浏览队列上的消息。
- 启动和停止通道。
- 查看有关通道，侦听器，队列或服务对象的状态信息。
- 查看集群中的队列管理器。
- 检查以查看哪些应用程序，用户或通道打开了特定队列。
- 使用 " 新建集群 " 向导创建新的队列管理器集群。
- 使用 " 将队列管理器添加到集群 " 向导将队列管理器添加到集群。
- 管理与安全套接字层 (SSL) 通道安全性配合使用的认证信息对象。
- 创建和删除通道启动程序，触发器监视器和侦听器。
- 启动或停止命令服务器，通道启动程序，触发器监视器和侦听器。
- 将特定服务设置为在队列管理器启动时自动启动。
- 修改队列管理器的属性。
- 更改本地缺省队列管理器。
- 调用 ikeyman GUI 以管理安全套接字层 (SSL) 证书，使证书与队列管理器相关联，以及配置和设置证书库 (仅在本地计算机上)。
- 从 WebSphere MQ 对象创建 JMS 对象，并从 JMS 对象创建 WebSphere MQ 对象。
- 为当前支持的任何类型创建 JMS 连接工厂。
- 修改任何服务的参数，例如侦听器的 TCP 端口号或通道启动程序队列名称。
- 启动或停止服务跟踪。

您可以使用一系列 内容视图 和 "属性" 对话框来执行管理任务。

内容视图
"内容视图" 是可以显示以下内容的面板:

- 与 WebSphere MQ 本身相关的属性和管理选项。
- 与一个或多个相关对象相关的属性和管理选项。
- 集群的属性和管理选项。

属性对话框
属性对话框是一个面板，用于显示与一系列字段中的对象相关的属性，您可以对其中一些字段进行编辑。

您可以使用 " *Navigator* " 视图浏览 WebSphere MQ Explorer。 Navigator 允许您选择所需的内容视图。

远程队列管理器

您可以连接到的受支持队列管理器有两个例外。

从 Windows 或 Linux (x86 和 x86-64 平台) 系统， WebSphere MQ Explorer 可以连接到所有受支持的队列管理器，但存在以下异常:

- WebSphere MQ for z/OS 队列管理器低于 V 6.0。
- 当前受支持的 MQSeries V2 队列管理器。

IBM WebSphere MQ Explorer 可处理不同命令级别和平台之间的功能差异。 但是，如果它迁到无法识别的属性，那么该属性将不可见。

如果您打算在 Windows 上使用 IBM WebSphere MQ Explorer 在 WebSphere MQ V5.3 计算机上远程管理 V6.0 或更高版本的队列管理器，那么必须在 WebSphere MQ for Windows V5.3 计算机上安装修订包 9 (CSD9) 或更高版本。

如果您打算在 iSeries 上使用 WebSphere MQ V6.0 或更高版本计算机上的 WebSphere MQ Explorer 来远程管理 V5.3 队列管理器， 必须在 WebSphere MQ for iSeries V5.3 计算机上安装 FP11 (CSD11) 或更高版本。此修订包可更正 WebSphere MQ Explorer 与 iSeries 队列管理器之间的连接问题。

决定是否使用 IBM WebSphere MQ Explorer

在决定是否在安装时使用 IBM WebSphere MQ Explorer 时，请考虑本主题中列出的信息。

你需要注意以下几点:

对象名称
如果将队列管理器和其他对象的小写名称与 IBM WebSphere MQ Explorer 配合使用，那么在使用 MQSC 命令处理对象时，必须将对象名称括在单引号中，否则 WebSphere MQ 无法识别这些对象名称。

大型队列管理器
IBM WebSphere MQ Explorer 最适用于小型队列管理器。 如果您在单个队列管理器上具有大量对象，那么当 WebSphere MQ Explorer 抽取要在视图中显示的必需信息时，可能会迁到延迟。

集群
WebSphere MQ 集群可能包含数百或数千个队列管理器。 WebSphere MQ Explorer 使用树结构提供集群中的队列管理器。 集群的物理大小不会显着影响 IBM WebSphere MQ Explorer 的速度，因为 IBM WebSphere MQ Explorer 在您选择它们之前不会连接到集群中的队列管理器。

设置 IBM WebSphere MQ Explorer

本部分概述了设置 IBM WebSphere MQ Explorer 所需的步骤。

- 第 53 页的『必备软件和定义』
- 第 54 页的『安全性』
- 第 57 页的『显示和隐藏队列管理器和集群』
- 第 57 页的『集群成员资格』
- 第 58 页的『数据转换』

必备软件和定义

在尝试使用 IBM WebSphere MQ Explorer 之前，请确保满足以下需求。

IBM WebSphere MQ Explorer 只能使用 TCP/IP 通信协议连接到远程队列管理器。

检查:

1. 命令服务器正在每个远程管理的队列管理器上运行。
2. 必须在每个远程队列管理器上运行合适的 TCP/IP 侦听器对象。 此对象可以是 IBM WebSphere MQ 侦听器，也可以是 UNIX and Linux 系统上的 inetd 守护程序。
3. 缺省情况下名为 SYSTEM.ADMIN.SVRCONN 存在于所有远程队列管理器上。

   您可以使用以下 MQSC 命令创建通道:

```
DEFINE CHANNEL(SYSTEM.ADMIN.SVRCONN) CHLTYPE(SVRCONN)
```

此命令创建基本通道定义。 如果需要更复杂的定义 (例如，用于设置安全性) ，那么需要其他参数。 有关更多信息，请参阅 DEFINE CHANNEL 。

4. 系统队列 SYSTEM.MQEXPLORER.REPLY.MODEL 必须存在。

安全性

如果您在控制用户对特定对象的访问权的环境中使用 WebSphere MQ ，那么可能需要考虑使用 IBM WebSphere MQ Explorer 的安全性方面。

授权使用 *IBM WebSphere MQ Explorer*

任何用户都可以使用 IBM WebSphere MQ Explorer，但需要某些权限才能连接，访问和管理队列管理器。

要使用 WebSphere MQ Explorer 执行本地管理任务，需要用户具有执行管理任务所需的权限。 如果用户是 mqm 组的成员，那么该用户有权执行所有本地管理任务。

要连接到远程队列管理器并使用 WebSphere MQ Explorer 执行远程管理任务，需要执行 WebSphere MQ Explorer 的用户具有以下权限:

- 对目标队列管理器对象的 CONNECT 权限
- 对目标队列管理器对象的 INQUIRE 权限
- 对目标队列管理器对象的 DISPLAY 权限
- 对队列 SYSTEM.MQEXPLORER.REPLY.MODEL
- 对队列 SYSTEM.MQEXPLORER.REPLY.MODEL
- 对队列 SYSTEM.MQEXPLORER.REPLY.MODEL
- 对队列 SYSTEM.ADMIN.COMMAND.QUEUE
- 队列 SYSTEM.ADMIN.COMMAND.QUEUE
- 执行所选操作的权限

注: INPUT 权限与来自队列 (获取操作) 的用户输入相关。 OUTPUT 权限与从用户到队列的输出相关 (放置操作)。

要连接到 WebSphere MQ for z/OS 上的远程队列管理器并使用 IBM WebSphere MQ Explorer 执行远程管理任务，必须提供以下内容:

- 系统队列 SYSTEM.MQEXPLORER.REPLY.MODEL 的 RACF 概要文件
- 队列的 RACF 概要文件 AMQ.MQEXPLORER.*

此外，执行 WebSphere MQ Explorer 的用户需要具有以下权限:

- RACF 对系统队列的 UPDATE 权限， SYSTEM.MQEXPLORER.REPLY.MODEL
- RACF 对队列的 UPDATE 权限， AMQ.MQEXPLORER.*
- 对目标队列管理器对象的 CONNECT 权限
- 执行所选操作的权限
- 对 MQCMDS 类中所有 hlq.DISPLAY.object 概要文件的 READ 权限

有关如何向 WebSphere MQ 对象授予权限的信息，请参阅 在 UNIX 或 Linux 系统和 Windows 上授予对 WebSphere MQ 对象的访问权。

如果用户尝试执行他们无权执行的操作，那么目标队列管理器将调用授权失败过程，并且该操作将失败。

WebSphere MQ Explorer 中的缺省过滤器是显示所有 WebSphere MQ 对象。 如果存在用户没有 DISPLAY 权限的任何 WebSphere MQ 对象，那么将生成授权失败。 如果正在记录权限事件，请将显示的对象范围限制为用户具有 DISPLAY 权限的对象。

用于连接到远程队列管理器的安全性

您必须保护 IBM WebSphere MQ Explorer 与每个远程队列管理器之间的通道。

IBM WebSphere MQ Explorer 作为 MQI 客户机应用程序连接到远程队列管理器。 这意味着每个远程队列管理器都必须具有服务器连接通道的定义和合适的 TCP/IP 侦听器。 如果您不保护服务器连接通道，那么恶意应用程序可能会连接到同一服务器连接通道，并获得对具有无限权限的队列管理器对象的访问权。 为了保护服务器连接通道，请为通道的 MCAUSER 属性指定非空白值，使用通道认证记录或使用安全出口。

MCAUSER 属性的缺省值是本地用户标识。 如果指定非空白用户名作为服务器连接通道的 MCAUSER 属性，那么使用此通道连接到队列管理器的所有程序将以指定用户的身份运行，并且具有相同的权限级别。 如果使用通道认证记录，那么不会发生此情况。

将安全出口与 *WebSphere MQ Explorer* 配合使用

您可以使用 WebSphere MQ Explorer 指定缺省安全出口和特定于队列管理器的安全出口。

您可以定义缺省安全出口，该出口可用于来自 WebSphere MQ Explorer 的所有新客户机连接。 在建立连接时，可以覆盖此缺省出口。 您还可以为单个队列管理器或一组队列管理器定义安全出口，这将在建立连接时生效。 使用 WebSphere MQ Explorer 指定出口。 有关更多信息，请参阅 WebSphere MQ 帮助中心。

使用 *IBM WebSphere MQ Explorer* 通过启用 *SSL* 的 *MQI* 通道连接到远程队列管理器

IBM WebSphere MQ Explorer 使用 MQI 通道连接到远程队列管理器。 如果要使用 SSL 安全性来保护 MQI 通道，那么必须使用客户机通道定义表来建立通道。

有关如何使用客户机通道定义表建立 MQI 通道的信息，请参阅 IBM WebSphere MQ MQI 客户机概述。

使用客户机通道定义表建立通道后，可以使用 IBM WebSphere MQ Explorer 通过启用 SSL 的 MQI 通道连接到远程队列管理器，如 第 55 页的『托管远程队列管理器的系统上的任务』 和 第 55 页的『托管 IBM WebSphere MQ Explorer 的系统上的任务』中所述。

托管远程队列管理器的系统上的任务

在托管远程队列管理器的系统上，执行以下任务:

1. 定义通道的服务器连接和客户机连接对，并为这两个通道上的服务器连接上的 *SSLCIPH* 变量指定相应的值。 有关 *SSLCIPH* 变量的更多信息，请参阅 使用 SSL 保护通道
2. 将在队列管理器的 `@ipcc` 目录中找到的通道定义表 `AMQCLCHL.TAB` 发送到托管 IBM WebSphere MQ Explorer 的系统。
3. 在指定的端口上启动 TCP/IP 侦听器。
4. 将 CA 和个人 SSL 证书放入队列管理器的 SSL 目录中:
   - UNIX and Linux 系统的 `/var/mqm/qmgrs/+QMNAME+/SSL`
   - Windows 系统的 `C:\Program Files\WebSphere MQ\qmgrs\+QMNAME+\SSL`

   其中 `+QMNAME+` 是表示队列管理器名称的令牌。
5. 创建名为 `key.kdb` 的 CMS 类型的密钥数据库文件。 通过检查 iKeyman GUI 中的选项，或者通过将 `-stash` 选项与 **runmqckm** 命令配合使用，将密码隐藏在文件中。
6. 将 CA 证书添加到在上一步中创建的密钥数据库。
7. 将队列管理器的个人证书导入到密钥数据库中。

有关在 Windows 系统上使用安全套接字层的更多详细信息，请参阅 在 UNIX， Linux 和 Windows 系统上使用 SSL 或 TLS 。

托管 IBM WebSphere MQ Explorer 的系统上的任务

在托管 IBM WebSphere MQ Explorer 的系统上，执行以下任务:

1. 创建名为 `key.jks` 的 JKS 类型的密钥数据库文件。 设置此密钥数据库文件的密码。

   IBM WebSphere MQ Explorer 将 Java 密钥库文件 (JKS) 用于 SSL 安全性，因此要为 IBM WebSphere MQ Explorer 配置 SSL 而创建的密钥库文件必须与此匹配。
2. 将 CA 证书添加到在上一步中创建的密钥数据库。
3. 将队列管理器的个人证书导入到密钥数据库中。

4. 在 Windows 和 Linux 系统上，使用系统菜单， `MQExplorer` 可执行文件或 **strmqcfg** 命令来启动 MQ Explorer。
5. 从 IBM WebSphere MQ Explorer 工具栏中，单击 **窗口-> 首选项**，然后展开 **WebSphere MQ Explorer** ，然后单击 **SSL 客户机证书库**。 在 "可信证书库" 和 "个人证书库" 中输入在 第 55 页的『托管 IBM WebSphere MQ Explorer 的系统上的任务』的步骤 1 中创建的 JKS 文件的名称和密码，然后单击 **确定**。
6. 关闭 " **首选项** " 窗口，然后右键单击 **队列管理器**。 单击 **显示/隐藏队列管理器**，然后在 " **显示/隐藏队列管理器** " 屏幕上单击 **添加** 。
7. 输入队列管理器的名称，然后选择 **直接连接** 选项。 单击“下一步”。
8. 选择 **使用客户机通道定义表 (CCDT)** ，并在托管远程队列管理器的系统上的 第 55 页的『托管远程队列管理器的系统上的任务』 中指定从步骤 2 中的远程队列管理器传输的通道表文件的位置。
9. 单击**完成**。 现在可以从 IBM WebSphere MQ Explorer 访问远程队列管理器。

通过另一个队列管理器进行连接

IBM WebSphere MQ Explorer 允许您通过中间队列管理器连接到队列管理器， IBM WebSphere MQ Explorer 已连接到该中间队列管理器。

在这种情况下， IBM WebSphere MQ Explorer 会将 PCF 命令消息放入中间队列管理器，并指定以下内容:

- 对象描述符 (MQOD) 中的 *ObjectQMgrName* 参数作为目标队列管理器的名称。 有关队列名称解析的更多信息，请参阅 名称解析 。
- 消息描述符 (MQMD) 中的 *UserIdentifier* 参数作为本地 userId。

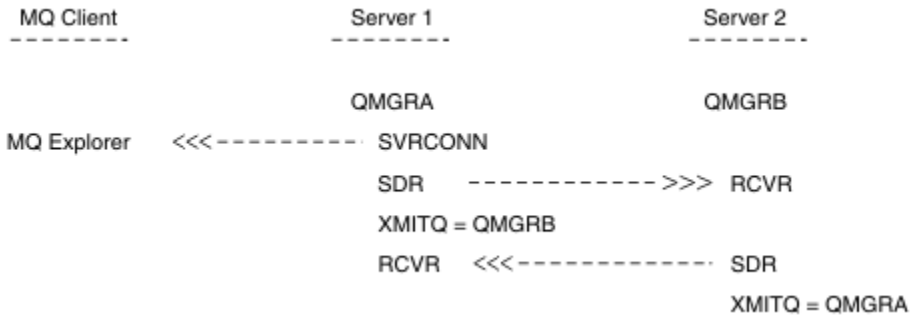
如果连接随后用于通过中间队列管理器连接到目标队列管理器，那么 userId 将再次在消息描述符 (MQMD) 的 *UserIdentifier* 参数中流动。 为了使目标队列管理器上的 MCA 侦听器接受此消息，必须设置 MCAUSER 属性，或者必须已存在具有 put 权限的 userId 。

目标队列管理器上的命令服务器将消息放入传输队列，并在消息描述符 (MQMD) 的 *UserIdentifier* 参数中指定 userId 。 要使此 put 操作成功，具有 put 权限的目标队列管理器上必须已存在 userId 。

以下示例显示如何通过中间队列管理器将队列管理器连接到 WebSphere MQ Explorer。

建立与队列管理器的远程管理连接。 验证:

- 服务器上的队列管理器处于活动状态，并且定义了服务器连接通道 (SVRCONN)。
- 侦听器处于活动状态。
- 命令服务器处于活动状态。
- SYSTEM.MQ EXPLORER.REPLY.MODEL 队列，并且您具有足够的权限。
- 队列管理器侦听器，命令服务器和发送方通道已启动。



在本示例中:

- IBM WebSphere MQ Explorer 使用客户机连接连接到队列管理器 QMGRA (在 Server1 上运行)。
- Server2 上的队列管理器 QMGRB 现在可以通过中间队列管理器 (QMGRA) 连接到 IBM WebSphere MQ Explorer
- 使用 WebSphere MQ Explorer 连接到 QMGRB 时，请选择 QMGRA 作为中间队列管理器

在此情况下，没有从 IBM WebSphere MQ Explorer 到 QMGRB 的直接连接; 到 QMGRB 的连接是通过 QMGRA 进行的。

Server2 上的队列管理器 QMGRB 使用发送方/接收方通道连接到 Server1 上的 QMGRA 。 必须以可进行远程管理的方式设置 QMGRA 与 QMGRB 之间的通道; 请参阅 第 95 页的『为远程管理准备通道和传输队列』。

显示和隐藏队列管理器和集群

IBM WebSphere MQ Explorer 一次可以显示多个队列管理器。 从 "显示/隐藏队列管理器" 面板 (可从 "队列管理器" 树节点的菜单中选择) ，您可以选择是否显示有关另一台 (远程) 机器的信息。 将自动检测本地队列管理器。

要显示远程队列管理器:

1. 右键单击 Queue Managers 树节点，然后选择 显示/隐藏队列管理器 ...。
2. 单击**添加**。 将显示 "显示/隐藏队列管理器" 面板。
3. 在提供的字段中输入远程队列管理器的名称和主机名或 IP 地址。

   主机名或 IP 地址用于使用其缺省服务器连接通道 SYSTEM.ADMIN.SVRCONN 或用户定义的服务器连接通道。
4. 单击**完成**。

"显示/隐藏队列管理器" 面板还显示所有可视队列管理器的列表。 您可以使用此面板在导航视图中隐藏队列管理器。

如果 IBM WebSphere MQ Explorer 显示作为集群成员的队列管理器，那么将检测并自动显示集群。

要从此面板导出远程队列管理器的列表，请执行以下操作:

1. 关闭 "显示/隐藏队列管理器" 面板。
2. 右键单击 WebSphere MQ Explorer 的 "导航" 窗格中的顶部 **IBM WebSphere MQ** 树节点，然后选择 **导出 MQ Explorer 设置**
3. 单击 **MQ Explorer> MQ Explorer 设置**
4. 选择 **连接信息> 远程队列管理器**。
5. 选择要在其中存储导出的设置的文件。
6. 最后，单击 **完成** 以将远程队列管理器连接信息导出到指定文件。

要导入远程队列管理器列表，请执行以下操作:

1. 右键单击 WebSphere MQ Explorer 的 "导航" 窗格中的顶部 **IBM WebSphere MQ** 树节点，然后选择 **导入 MQ Explorer 设置**
2. 单击 **MQ Explorer> MQ Explorer 设置**
3. 单击 **浏览**，然后浏览到包含远程队列管理器连接信息的文件的路径。
4. 单击**打开**。 如果该文件包含远程队列管理器列表，那么将选中 **连接信息> 远程队列管理器** 框。
5. 最后，单击 **完成** 以将远程队列管理器连接信息导入到 WebSphere MQ Explorer 中。

集群成员资格

IBM WebSphere MQ Explorer 需要有关作为集群成员的队列管理器的信息。

如果队列管理器是集群的成员，那么将自动填充集群树节点。

如果队列管理器在 IBM WebSphere MQ Explorer 运行时成为集群的成员，那么必须使用有关集群的最新管理数据来维护 IBM WebSphere MQ Explorer ，以便它可以与这些集群进行有效通信，并在请求时显示正确的集群信息。 为此， WebSphere MQ Explorer 需要以下信息:

- 存储库队列管理器的名称
- 存储库队列管理器的连接名称 (如果它在远程队列管理器上)

通过此信息， WebSphere MQ Explorer 可以:

- 使用存储库队列管理器来获取集群中队列管理器的列表。
- 管理属于集群成员且处于受支持平台和命令级别的队列管理器。

在下列情况下，无法进行管理:

- 所选存储库变为不可用。 WebSphere MQ Explorer 不会自动切换到备用存储库。
- 无法通过 TCP/IP 联系所选存储库。
- 所选存储库正在运行于在 WebSphere MQ Explorer 不支持的平台和命令级别上运行的队列管理器上。

可以管理的集群成员可以是本地成员，也可以是远程成员 (如果可以使用 TCP/IP)。 IBM WebSphere MQ Explorer 直接连接到作为集群成员的本地队列管理器，而不使用客户机连接。

数据转换

IBM WebSphere MQ Explorer 以 CCSID 1208 (UTF-8) 工作。 这使 IBM WebSphere MQ Explorer 能够正确显示来自远程队列管理器的数据。 无论是直接连接到队列管理器，还是使用中间队列管理器， IBM WebSphere MQ Explorer 都要求将所有入局消息转换为 CCSID 1208 (UTF-8)。

如果尝试在 IBM WebSphere MQ Explorer 与具有 IBM WebSphere MQ Explorer 无法识别的 CCSID 的队列管理器之间建立连接，那么将发出错误消息。

代码页转换中描述了受支持的转换。

Windows 上的安全性

"准备 WebSphere MQ " 向导将创建一个特殊用户帐户，以便需要使用该帐户的进程可以共享 Windows 服务。

Windows 服务在 IBM WebSphere MQ 安装的客户机进程之间共享。 将为每个安装创建一个服务。 每个服务都名为 `MQ_InstallationName`，并且显示名称为 `IBM WebSphere MQ(InstallationName)`。 在 Version 7.1 之前，仅在服务器上安装了一个 Windows 服务，该服务的名称为 `MQSeriesServices` ，显示名称为 `IBM MQSeries`。

由于每个服务必须在非交互式和交互式登录会话之间共享，因此必须在特殊用户帐户下启动每个服务。 您可以将一个特殊用户帐户用于所有服务，或者创建不同的特殊用户帐户。 每个特殊用户帐户都必须具有 “作为服务登录”的用户权限，有关更多信息，请参阅 第 59 页的『IBM WebSphere MQ Windows 服务所需的用户权限』。 如果用户标识没有运行该服务的权限，那么该服务不会启动，并且会在 Windows 系统事件日志中返回错误。 通常，您将运行 "准备 IBM WebSphere MQ " 向导，并正确设置用户标识。 但是，如果您手动配置了用户标识，那么是否可能存在需要解决的问题。

首次安装 IBM WebSphere MQ 并运行 "准备 IBM WebSphere MQ " 向导时，它会为名为 MUSR_MQADMIN 的服务创建具有所需设置和许可权 (包括 “作为服务登录”) 的本地用户帐户。

对于后续安装， "准备 IBM WebSphere MQ " 向导将创建名为 MUSR_MQADMINx 的用户帐户，其中 x 是表示不存在的用户标识的下一个可用数字。 MUSR_MQADMINx 的密码是在创建帐户时随机生成的，用于配置服务的登录环境。 生成的密码不会到期。

此 IBM WebSphere MQ 帐户不受系统上设置为要求在特定时间段后更改帐户密码的任何帐户策略的影响。

密码在此一次性处理外部未知，并且由 Windows 操作系统存储在注册表的安全部分中。

使用 Active Directory (仅限 Windows)

在某些网络配置中，如果用户帐户是在使用 Active Directory 的域控制器上定义的，那么正在运行的本地用户帐户 IBM WebSphere MQ 可能没有查询其他域用户帐户的组成员资格所需的权限。 "准备 IBM WebSphere MQ 向导" 通过执行测试并询问用户有关网络配置的问题来确定是否存在这种情况。

如果正在运行的本地用户帐户 IBM WebSphere MQ 没有必需的权限，那么 "准备 IBM WebSphere MQ 向导" 会提示用户输入具有特定用户权限的域用户帐户的帐户详细信息。 有关域用户帐户所需的用户权限，请参阅 第 59 页的『IBM WebSphere MQ Windows 服务所需的用户权限』。 用户在 "准备 IBM WebSphere MQ 向导" 中输入域用户帐户的有效帐户详细信息后，会将 IBM WebSphere MQ Windows 服务配置为在新帐户下运行。 帐户详细信息保存在注册表的安全部分中，用户无法读取。

当服务处于运行状态时，将启动 IBM WebSphere MQ Windows 服务，并且只要该服务处于运行状态，该服务就会保持运行状态。 启动 Windows 服务后登录到服务器的 IBM WebSphere MQ 管理员可以使用 IBM WebSphere MQ Explorer 来管理服务器上的队列管理器。 这会将 IBM WebSphere MQ Explorer 连接到现有 Windows 服务进程。 这两个操作需要不同级别的许可权才能工作:

- 启动过程需要启动许可权。
- IBM WebSphere MQ 管理员需要 "访问" 许可权。

IBM WebSphere MQ Windows 服务所需的用户权限

本主题中的表列出了运行 IBM WebSphere MQ 安装的 Windows 服务所使用的本地和域用户帐户所需的用户权限。

以批处理作业身份登录	使 IBM WebSphere MQ Windows 服务能够在此用户帐户下运行。
作为服务登录	使用户能够设置 IBM WebSphere MQ Windows 服务以使用配置的帐户登录。
关闭系统	如果配置为在服务恢复失败时重新启动服务器，那么允许 IBM WebSphere MQ Windows 服务重新启动服务器。
增加配额	对于操作系统 `CreateProcessAsUser` 调用是必需的。
作为操作系统的组成部分	对于操作系统 `LogonUser` 调用是必需的。
回避遍历检查	对于操作系统 `LogonUser` 调用是必需的。
替换进程层次标记	对于操作系统 `LogonUser` 调用是必需的。

注: 在运行 ASP 和 IIS 应用程序的环境中可能需要调试程序权限。

域用户帐户必须将这些 Windows 用户权限设置为 "本地安全策略" 应用程序中列出的有效用户权限。 如果不是，请在服务器上本地使用 "本地安全策略" 应用程序或使用 "域安全性应用程序" 域来设置它们。

更改与 IBM WebSphere MQ 服务关联的用户名

您可能需要将与 IBM WebSphere MQ 服务关联的用户名从 MUSR_MQADMIN 更改为其他名称。 (例如，如果队列管理器与不接受超过 8 个字符的用户名的 DB2 相关联，那么您可能需要执行此操作。)

过程

1. 创建新的用户帐户 (例如 **NEW_NAME**)
2. 使用 "准备 IBM WebSphere MQ 向导" 来输入新用户帐户的详细信息。

更改 IBM WebSphere MQ Windows 服务用户帐户的密码

关于此任务

要更改 IBM WebSphere MQ Windows 服务本地用户帐户的密码，请执行以下步骤:

过程

1. 标识正在运行服务的用户。
2. 从 "计算机管理" 面板停止 IBMIBM WebSphere MQ 服务。
3. 更改所需密码的方式与更改个人密码的方式相同。
4. 从 "计算机管理" 面板转至 IBM WebSphere MQ 服务的属性。
5. 选择 **登录** 页面。
6. 确认指定的帐户名称与修改了密码的用户匹配。
7. 在 **密码** 和 **确认密码** 字段中输入密码，然后单击 **确定**。

在域用户帐户下运行的安装的 *IBM WebSphere MQ Windows* 服务

关于此任务

如果安装的 IBM WebSphere MQ Windows 服务正在域用户帐户下运行，那么您还可以更改帐户的密码，如下所示:

过程

1. 更改域控制器上的域帐户的密码。 您可能需要请求域管理员为您执行此操作。
2. 执行以下步骤以修改 IBMIBM WebSphere MQ 服务的 " **登录** " 页面。

   IBM WebSphere MQ Windows 服务在其中运行的用户帐户将执行用户界面应用程序发出的任何 MQSC 命令，或在系统启动，关闭或服务恢复时自动执行的任何 MQSC 命令。 因此，此用户帐户必须具有 IBM WebSphere MQ 管理权限。 缺省情况下，会将其添加到服务器上的本地 **mqm** 组。 如果除去此成员资格，那么 IBM WebSphere MQ Windows 服务不起作用。 有关用户权限的更多信息，请参阅 第 59 页的『IBM WebSphere MQ Windows 服务所需的用户权限』

   如果运行 IBM WebSphere MQ Windows 服务的用户帐户出现安全问题，那么系统事件日志中将显示错误消息和描述。

相关概念

第 58 页的『使用 Active Directory (仅限 Windows)』
在某些网络配置中，如果用户帐户是在使用 Active Directory 的域控制器上定义的，那么正在运行的本地用户帐户 IBM WebSphere MQ 可能没有查询其他域用户帐户的组成员资格所需的权限。 "准备 IBM WebSphere MQ 向导" 通过执行测试并询问用户有关网络配置的问题来确定是否存在这种情况。

IBM WebSphere MQ 与 Db2 作为资源管理器进行协调

如果从 IBM WebSphere MQ Explorer 启动队列管理器，或者正在使用 IBM WebSphere MQ V7，并且在协调 Db2 时迁到问题，请检查队列管理器错误日志。

检查队列管理器错误日志以获取类似如下的错误:

```
23/09/2008 15:43:54 - Process(5508.1) User(MUSR_MQADMIN) Program(amqzxma0.exe)
Host(HOST_1) Installation(Installation1)
VMRF(7.1.0.0) QMgr(A.B.C)
AMQ7604: The XA resource manager 'DB2 MQBankDB database' was not available when called
    for xa_open. The queue manager is continuing without this resource manager.
```

说明: 运行 IBM WebSphere MQ 服务进程 `amqsvc.exe` 的用户标识（缺省名为 MUSR_MQADMIN）仍在使用不含 DB2USERS 组的组成员资格信息的访问令牌运行。

求解: 确保 IBM WebSphere MQ 服务用户标识是 DB2USERS 的成员之后，请使用以下命令序列:

- 停止该服务。
- 停止相同用户标识下运行的任何其他进程。
- 重新启动这些进程。

重新引导机器将确保完成先前步骤，但这不是必需的。

扩展 IBM WebSphere MQ Explorer

IBM WebSphere MQ for Windows 和 IBM WebSphere MQ for Linux (x86 和 x86-64 平台) 提供了称为 IBM WebSphere MQ Explorer 的管理接口，以执行管理任务作为使用控制或 MQSC 命令的替代方法。

此信息仅适用于 WebSphere MQ for Windows 和 WebSphere MQ for Linux (x86 和 x86-64 平台)。

IBM WebSphere MQ Explorer 以与 Eclipse 框架以及 Eclipse 支持的其他插件应用程序的样式一致的方式提供信息。

通过扩展 IBM WebSphere MQ Explorer ， 系统管理员能够定制 WebSphere MQ Explorer ， 以改进他们管理 WebSphere MQ 的方式。

有关更多信息，请参阅 IBM WebSphere MQ Explorer 产品文档中的 扩展 *IBM WebSphere MQ Explorer* 。

使用 IBM WebSphere MQ 任务栏应用程序 (仅限 Windows)

IBM WebSphere MQ 任务栏应用程序在服务器上的 Windows 系统托盘中显示图标。 该图标为您提供了 IBM WebSphere MQ 的当前状态以及一个菜单，您可以从该菜单执行一些简单操作。

在 Windows 上， WebSphere MQ 图标位于服务器上的系统托盘中，并以颜色编码的状态符号覆盖，这可能具有以下含义之一:

绿色
工作正常; 目前无警报

蓝色
不确定; WebSphere MQ 正在启动或关闭

黄色
警报; 一个或多个服务发生故障或已发生故障

要显示菜单，请右键单击 WebSphere MQ 图标。 从菜单中，可以执行以下操作:

- 单击 **打开** 以打开 WebSphere MQ 警报监视器
- 单击 **退出** 以退出 WebSphere MQ 任务栏应用程序
- 单击 **WebSphere MQ Explorer** 以启动 IBM WebSphere MQ Explorer
- 单击 **停止 WebSphere MQ** 以停止 WebSphere MQ
- 单击 **关于 WebSphere MQ** 以显示有关 WebSphere MQ 警报监视器的信息

IBM WebSphere MQ 警报监视器应用程序 (仅限 Windows)

IBM WebSphere MQ 警报监视器是一个错误检测工具，用于识别和记录本地机器上的 IBM WebSphere MQ 问题。

警报监视器显示有关 WebSphere MQ 服务器的本地安装的当前状态的信息。 它还监视 Windows 高级配置和电源接口 (ACPI) ，并确保强制实施 ACPI 设置。

从 WebSphere MQ 警报监视器，您可以:

- 直接访问 IBM WebSphere MQ Explorer
- 查看与所有未完成的警报相关的信息
- 在本地机器上关闭 WebSphere MQ 服务
- 通过网络将警报消息路由到可配置的用户帐户，或者路由到 Windows 工作站或服务器

管理本地 IBM WebSphere MQ 对象

本节说明如何管理本地 IBM WebSphere MQ 对象以支持使用消息队列接口 (MQI) 的应用程序。 在此上下文中，本地管理意味着创建，显示，更改，复制和删除 IBM WebSphere MQ 对象。

除了本节中详述的方法外，您还可以使用 IBM WebSphere MQ Explorer 来管理本地 WebSphere MQ 对象; 请参阅 第 51 页的『使用 IBM WebSphere MQ Explorer 进行管理』。

本部分包含以下信息:

- 使用 MQI 的应用程序
- 第 65 页的『使用 MQSC 命令执行本地管理任务』
- 第 72 页的『使用队列管理器』
- 第 73 页的『使用本地队列』
- 第 78 页的『使用别名队列』
- 第 79 页的『使用模型队列』
- 第 85 页的『使用服务』

- 第 91 页的『管理用于触发的对象』

启动和停止队列管理器

使用本主题作为停止和启动队列管理器的简介。

启动队列管理器

要启动队列管理器，请按如下所示使用 strmqm 命令:

```
strmqm saturn.queue.manager
```

在 WebSphere MQ for Windows 和 WebSphere MQ for Linux (x86 和 x86-64 平台) 系统上，可以启动队列管理器，如下所示:

1. 打开 IBM WebSphere MQ Explorer。
2. 从 " Navigator " 视图中选择队列管理器。
3. 单击 Start。 队列管理器将启动。

如果队列管理器启动时间超过几秒，那么 WebSphere MQ 会间歇性地发出参考消息，详细说明启动进度。

在队列管理器启动并准备好接受连接请求之前， strmqm 命令不会返回控制权。

自动启动队列管理器

在 WebSphere MQ for Windows 中，可以在系统使用 WebSphere MQ Explorer 启动时自动启动队列管理器。 有关更多信息，请参阅 第 51 页的『使用 IBM WebSphere MQ Explorer 进行管理』。

停止队列管理器

使用 **endmqm** 命令可停止队列管理器。

注: 必须从与您正在使用的队列管理器相关联的安装中使用 **endmqm** 命令。 您可以使用 dspmq -o installation 命令来查明队列管理器与之关联的安装。

例如，要停止名为 QMB 的队列管理器，请输入以下命令:

```
endmqm QMB
```

在 WebSphere MQ for Windows 和 WebSphere MQ for Linux (x86 和 x86-64 平台) 系统上，可以按如下所示停止队列管理器:

1. 打开 IBM WebSphere MQ Explorer。
2. 从 " Navigator " 视图中选择队列管理器。
3. 单击 Stop...。 将显示 "结束队列管理器" 面板。
4. 选择 "受控" 或 "立即"。
5. 单击 OK。 队列管理器停止。

停顿关闭 (quiesced shutdown)

缺省情况下， **endmqm** 命令执行指定队列管理器的停顿关闭。 这可能需要一段时间才能完成。 停顿的关闭将等待所有已连接的应用程序断开连接。

使用此类型的关闭来通知应用程序停止。 如果您发出:

```
endmqm -c QMB
```

未在所有应用程序停止时告知您。 (endmqm -c QMB 命令等同于 endmqm QMB 命令。)

但是，如果您发出:

```
endmqm -w QMB
```

该命令将一直等到所有应用程序都已停止并且队列管理器已结束。

立即关闭 (immediate shutdown)

对于立即关闭，将允许任何当前 MQI 调用完成，但任何新调用都将失败。 此类型的关闭不会等待应用程序与队列管理器断开连接。

对于立即关闭，请输入:

```
endmqm -i QMB
```

抢先关闭 (preemptive shutdown)

注: 除非使用 **endmqm** 命令停止队列管理器的所有其他尝试都失败，否则请勿使用此方法。 此方法可能会对已连接的应用程序产生不可预测的后果。

如果立即关闭不起作用，那么必须使用 *preemptive* 关闭，并指定 -p 标志。 例如:

```
endmqm -p QMB
```

这将立即停止队列管理器。 如果此方法仍不起作用，请参阅 第 63 页的『手动停止队列管理器』 以获取备用解决方案。

有关 **endmqm** 命令及其选项的详细描述，请参阅 endmqm。

如果您在关闭队列管理器时迁到问题

关闭队列管理器时出现问题通常是由应用程序引起的。 例如，当应用程序:

- 不正确检查 MQI 返回码
- 不请求通知停顿
- 终止而不断开与队列管理器的连接 (通过发出 MQDISC 调用)

如果在停止队列管理器时发生问题，那么可以使用 Ctrl-C 中断 **endmqm** 命令。 然后，您可以发出另一个 **endmqm** 命令，但这次使用指定所需关闭类型的标志。

手动停止队列管理器

如果停止队列管理器的标准方法失败，请尝试此处描述的方法。

停止队列管理器的标准方法是使用 endmqm 命令。 要手动停止队列管理器，请使用本节中描述的其中一个过程。 有关如何使用控制命令对队列管理器执行操作的详细信息，请参阅 创建和管理队列管理器。

在 Windows 上停止队列管理器

如何在 IBM WebSphere MQ for Windows 中结束进程和 IBM WebSphere MQ 服务以停止队列管理器。

要停止在 WebSphere MQ for Windows 下运行的队列管理器:

1. 使用 Windows 任务管理器列出正在运行的进程的名称 (标识)。
2. 使用 Windows 任务管理器或 **taskkill** 命令按以下顺序结束进程 (如果它们正在运行):

AMQZMUC0	关键流程管理器
AMQZXMA0	执行控制器
AMQZFUMA	OAM 进程
AMQZLAA0	LQM 代理程序
AMQZLSA0	LQM 代理程序

AMQZMUF0	实用程序管理器
AMQZMGR0	进程控制器
AMQZMUR0	可重新启动的进程管理器
AMQFQPUB	发布预订流程
AMQFCXBA	代理工作程序进程
AMQRMPPA	进程池进程
AMQCRSTA	非线程响应程序作业进程
AMQCRS6B	LU62 接收方通道和客户机连接
AMQRRMFA	存储库进程 (针对集群)
AMQZDMAA	延迟消息处理器
AMQPCSEA	命令服务器
runmqtrm	调用服务器的触发器监视器
RUNMQDLQ	调用死信队列处理程序
运行 MQCHI	通道启动程序进程
运行 MQLSR	通道侦听器进程
AMQXSSVN	共享内存服务器
AMQZTRCN	跟踪

3. 从 Windows 控制面板上的 **管理工具** > **服务** 停止 WebSphere MQ 服务。
4. 如果尝试了所有方法并且队列管理器尚未停止，请重新引导系统。

Windows 任务管理器和 **tasklist** 命令提供有关任务的有限信息。 有关帮助确定哪些进程与特定队列管理器相关的更多信息，请考虑使用 *Process Explorer* (procexp.exe) 之类的工具，该工具可从 Microsoft Web 站点 https://www.microsoft.com 下载。

在 UNIX and Linux 系统上停止队列管理器

如何结束进程和 IBM WebSphere MQ 服务，以停止 IBM WebSphere MQ for UNIX and Linux 中的队列管理器。 如果停止和除去队列管理器的标准方法失败，您可以尝试此处描述的方法。

要停止在 WebSphere MQ for UNIX and Linux 系统下运行的队列管理器:

1. 使用 ps 命令查找仍在运行的队列管理器程序的进程标识。 例如，如果队列管理器名为 QMNAME，请使用以下命令:

   ```
   ps -ef | grep QMNAME
   ```

2. 结束仍在运行的任何队列管理器进程。 使用 kill 命令，指定使用 ps 命令发现的进程标识。

   按以下顺序结束进程:

amqzmuc0	关键流程管理器
amqzxma0	执行控制器
阿姆格兹富马	OAM 进程
amqzlaa0	LQM 代理程序
amqzlsa0	LQM 代理程序
amqzmuf0	实用程序管理器
amqzmur0	可重新启动的进程管理器
amqzmgr0	进程控制器

阿姆格夫格普卜	发布预订流程
阿姆格夫奇巴	代理工作程序进程
阿姆克姆帕	进程池进程
阿姆克斯塔	非线程响应程序作业进程
amqcrs6b	LU62 接收方通道和客户机连接
阿姆格勒姆法	存储库进程 (针对集群)
阿姆格兹德马	延迟消息处理器
安格普切西	命令服务器
runmqtrm	调用服务器的触发器监视器
runmqdlq	调用死信队列处理程序
鲁姆基	通道启动程序进程
运行 mqlsr	通道侦听器进程

注: 您可以使用 **kill -9** 命令来结束未能停止的进程。

如果手动停止队列管理器，那么可能会采用 FFST，并且放置在 `/var/mqm/errors.` 中的 FDC 文件不会将此视为队列管理器中的缺陷。

队列管理器将正常重新启动，即使在您使用此方法将其停止之后也是如此。

使用 MQSC 命令执行本地管理任务

本部分向您介绍 MQSC 命令，并告诉您如何将这些命令用于某些常见任务。

如果使用 IBM WebSphere MQ for Windows 或 IBM WebSphere MQ for Linux (x86 和 x86-64 平台)，那么还可以使用 IBM WebSphere MQ Explorer 执行本节中描述的操作。 请参阅第 51 页的『使用 IBM WebSphere MQ Explorer 进行管理』以获取更多信息。

您可以使用 MQSC 命令来管理队列管理器对象，包括队列管理器本身，队列，进程定义，通道，客户机连接通道，侦听器，服务，名称列表，集群和认证信息对象。 此部分处理队列管理器，队列和进程定义; 有关管理通道，客户机连接通道和侦听器对象的信息，请参阅 对象。 有关用于管理队列管理器对象的所有 MQSC 命令的信息，请参阅第 65 页的『脚本 (MQSC) 命令』。

使用 `runmqsc` 命令向队列管理器发出 MQSC 命令。 (有关此命令的详细信息，请参阅 runmqsc。) 您可以通过交互方式执行此操作，从键盘发出命令，也可以重定向标准输入设备 (`stdin`) 以从 ASCII 文本文件运行一系列命令。 在这两种情况下，命令的格式都是相同的。 (有关从文本文件运行命令的信息，请参阅第 68 页的『从文本文件运行 MQSC 命令』。)

您可以通过三种方式运行 `runmqsc` 命令，具体取决于该命令上设置的标志:

- 验证命令而不运行该命令，其中 MQSC 命令在本地队列管理器上已验证，但未运行。
- 在本地队列管理器上运行命令，其中 MQSC 命令在本地队列管理器上运行。
- 在远程队列管理器上运行命令，其中 MQSC 命令在远程队列管理器上运行。

您还可以运行后跟问号的命令以显示语法。

MQSC 命令中指定的对象属性在本节中以大写形式显示 (例如， RQMNAME)，尽管它们不区分大小写。 MQSC 命令属性名称限制为 8 个字符。 MQSC 命令在其他平台上可用，包括 IBM i 和 z/OS。

MQSC 命令在 MQSC 参考 部分的主题集合中进行了概述。

脚本 (MQSC) 命令

MQSC 命令提供了在 WebSphere MQ 平台上发出人类可读命令的统一方法。 有关 可编程命令格式 (PCF) 命令的信息，请参阅第 8 页的『可编程命令格式简介』。

命令的一般格式显示在 MQSC 命令中。

使用 MQSC 命令时，应遵守以下规则:

- 每个命令以主参数 (动词) 开头，后跟辅助参数 (名词)。 然后，如果存在对象的名称或通用名称 (在括号中)，那么在大多数命令中都有该名称或通用名称。 在此之后，参数通常可以按任何顺序出现; 如果参数具有相应的值，那么该值必须直接出现在与其相关的参数之后。
- 关键字，括号和值可以用任意数目的空格和逗号分隔。 语法图中显示的逗号可始终替换为一个或多个空格。 每个参数前面必须至少有一个空格 (在主参数之后)。
- 可以在命令的开头或结尾以及参数，标点和值之间出现任意数目的空白。 例如，以下命令有效:

```
  ALTER QLOCAL  ('Account'  )           TRIGDPTH  (  1)
```

  一对引号内的空格很重要。
- 其他逗号可以出现在允许空格的任何位置，并被视为空格 (当然，除非它们位于用引号括起的字符串中)。
- 不允许重复参数。 也不允许重复带有其 "NO" 版本的参数，如 REPLACE NOREPLACE 中的参数。
- 包含空格，小写字符或除以下字符以外的特殊字符的字符串:
  - 句点 (.)
  - 正斜杠 (/)
  - 下划线 (_)
  - 百分号 (%)

  必须用单引号括起，除非它们是:
  - 以星号
  - 单个星号 (例如， TRACE (*))
  - 包含冒号的范围规范 (例如， CLASS (01:03))

  如果字符串本身包含单引号，那么单引号由两个单引号表示。未包含在引号内的小写字符将转换为大写。
- 在 z/OS 以外的平台上，不包含任何字符的字符串 (即，两个中间没有空格的单引号) 将解释为用单引号括起的空格，即，以与 (' ') 相同的方式解释。 此情况的例外情况是所使用的属性是下列其中一项:
  - TOPICSTR
  - SUB
  - USERDATA
  - SELECTOR

  那么两个无空格的单引号将被解释为零长度字符串。
- 在 v7.0 中，基于 MQCHARV 类型 (例如， SELECTOR 和子用户数据) 的字符串属性中的任何尾部空格都将被视为重要值，这意味着 "abc" 不等于 "abc"。
- 左括号后跟右括号，中间没有重要信息，例如

```
NAME ( )
```

  除非特别注明，否则无效。
- 关键字不区分大小写: AltER， alter 和 ALTER 都是可接受的。未包含在引号内的任何内容都将转换为大写。
- 为某些参数定义同义词。例如， DEF 始终是 DEFINE 的同义词，因此 DEF QLOCAL 有效。 但是，同义词不只是最小字符串 ;DEFI 不是 DEFINE 的有效同义词。

  注: DELETE 参数没有同义词。 这是为了避免在使用 DEF (DEFINE 的同义词) 时意外删除对象。

有关使用 MQSC 命令来管理 IBM WebSphere MQ 的概述，请参阅 第 65 页的『使用 MQSC 命令执行本地管理任务』。

MQSC 命令使用某些特殊字符来具有特定含义。 有关这些特殊字符以及如何使用这些特殊字符的更多信息，请参阅 具有特殊含义的字符。

要了解如何使用 MQSC 命令构建脚本，请参阅 构建命令脚本。

有关 MQSC 命令的完整列表，请参阅 MQSC 命令。

相关任务
构建命令脚本

WebSphere MQ 对象名

如何在 MQSC 命令中使用对象名。

在示例中，我们对对象使用一些长名称。 这是为了帮助您确定要处理的对象的类型。

发出 MQSC 命令时，只需要指定队列的局部名。 在我们的示例中，我们使用队列名称，例如:

```
ORANGE.LOCAL.QUEUE
```

名称的 LOCAL.QUEUE 部分用于说明此队列是本地队列。 通常，对于本地队列的名称，它是 **不** 必需的。

我们还使用名称 saturn.queue.manager 作为队列管理器名称。 名称的 queue.manager 部分用于说明此对象是队列管理器。 通常，队列管理器的名称 不 需要此参数。

MQSC 命令中的区分大小写

MQSC 命令 (包括其属性) 可以大写或小写形式编写。 MQSC 命令中的对象名将转换为大写 (即，未区分 QUEUE 和队列) ，除非将这些名称括在单引号内。 如果未使用引号，那么将使用大写名称来处理对象。 请参阅《MQSC 引用》以获取更多信息。

runmqsc 命令调用 (与所有 WebSphere MQ 控制命令一样) 在某些 WebSphere MQ 环境中区分大小写。 请参阅 使用控制命令 以获取更多信息。

标准输入和输出

标准输入设备(也称为 stdin) 是从中获取系统输入的设备。 通常，这是键盘，但您可以指定输入将来自串口或磁盘文件，例如。 标准输出设备(也称为 stdout) 是将系统输出发送到的设备。 通常这是一个显示器，但是您可以将输出重定向到串口或文件。

On operating-system commands and WebSphere MQ control commands, the < operator redirects input. 如果此运算符后跟文件名，那么将从文件中获取输入。 类似地，> 运算符重定向输出; 如果此运算符后跟文件名，那么输出将定向到该文件。

以交互方式使用 MQSC 命令

您可以使用命令窗口或 shell 以交互方式使用 MQSC 命令。

要以交互方式使用 MQSC 命令，请打开命令窗口或 shell 并输入:

```
runmqsc
```

在此命令中，未指定队列管理器名称，因此 MQSC 命令由缺省队列管理器处理。 如果要使用其他队列管理器，请在 **runmqsc** 命令上指定队列管理器名称。 例如，要在队列管理器 jupiter.queue.manager 上运行 MQSC 命令，请使用以下命令:

```
runmqsc jupiter.queue.manager
```

在此之后，您输入的所有 MQSC 命令都将由此队列管理器处理，假定它位于同一节点上并且已在运行。

现在，您可以根据需要输入任何 MQSC 命令。 例如，尝试此操作:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE)
```

对于具有过多参数以适合一行的命令，请使用连续字符来指示命令在以下行上继续:

- 减号 (-) 指示命令将从下一行开始继续。
- 加号 (+) 指示命令将从下一行上的第一个非空白字符继续。

命令输入以非连续字符的非空白行的最终字符终止。 您还可以通过输入分号 (;) 来显式终止命令输入。 (如果在命令输入的最后一行末尾意外地输入了连续字符，那么这特别有用。)

来自 MQSC 命令的反馈

当您发出 MQSC 命令时，队列管理器会返回操作员消息，以确认您的操作或告知您已发生的错误。 例如:

```
AMQ8006: WebSphere MQ queue created.
```

此消息确认已创建队列。

```
AMQ8405: Syntax error detected at or near end of command segment below:-

AMQ8426: Valid MQSC commands are:

    ALTER
    CLEAR
    DEFINE
    DELETE
    DISPLAY
    END
    PING
    REFRESH
    RESET
    RESOLVE
    RESUME
    START
    STOP
    SUSPEND
     4 : end
```

此消息指示您发生了语法错误。

这些消息将发送到标准输出设备。 如果未正确输入命令，请参阅 MQSC 引用 以获取正确的语法。

结束 MQSC 命令的交互式输入

要停止使用 MQSC 命令，请输入 END 命令。

或者，可以将 EOF 字符用于您的操作系统。

从文本文件运行 MQSC 命令

以交互方式运行 MQSC 命令适用于快速测试，但如果您有很长的命令，或者重复使用特定命令序列，请考虑从文本文件重定向 `stdin` 。

第 67 页的『标准输入和输出』 包含有关 `stdin` 和 `stdout` 的信息。 要从文本文件重定向 `stdin` ，请首先使用常规文本编辑器创建包含 MQSC 命令的文本文件。 使用 `runmqsc` 命令时，请使用重定向运算符。 例如，以下命令运行文本文件 `myprog.in` 中包含的一系列命令:

```
runmqsc < myprog.in
```

同样，您也可以将输出重定向到文件。 包含用于输入的 MQSC 命令的文件称为 *MQSC* 命令文件。 包含来自队列管理器的应答的输出文件称为 输出文件。

要在 `runmqsc` 命令上重定向 `stdin` 和 `stdout` ，请使用以下格式的命令:

```
runmqsc < myprog.in > myprog.out
```

此命令调用 MQSC 命令文件中包含的 MQSC 命令 myprog.in. 由于未指定队列管理器名称，因此 MQSC 命令将针对缺省队列管理器运行。 输出将发送到文本文件 myprog.out。 第 69 页的图 12 显示 MQSC 命令文件 myprog.in 的抽取， 第 70 页的图 13 显示 myprog.out 中输出的相应抽取。

要在 runmqsc 命令上重定向 stdin 和 stdout，对于不是缺省值的队列管理器 (saturn.queue.manager)，请使用以下格式的命令:

```
runmqsc saturn.queue.manager < myprog.in > myprog.out
```

MQSC 命令文件

MQSC 命令以人类可读格式 (即 ASCII 文本) 编写。 第 69 页的图 12 是从 MQSC 命令文件中抽取的内容，其中显示 MQSC 命令 (DEFINE QLOCAL) 及其属性。 MQSC 引用 包含每个 MQSC 命令及其语法的描述。

```
  .
  .
  .
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE  +
       DESCR(' ') +
       PUT(ENABLED) +
       DEFPRTY(0) +
       DEFPSIST(NO) +
       GET(ENABLED) +
       MAXDEPTH(5000) +
       MAXMSGL(1024) +
       DEFSOPT(SHARED) +
       NOHARDENBO +
       USAGE(NORMAL) +
       NOTRIGGER;
  .
  .
  .
```

图 *12:* 从 *MQSC* 命令文件中抽取

要在 WebSphere MQ 环境中实现可移植性，请将 MQSC 命令文件中的行长度限制为 72 个字符。 加号指示命令在下一行继续执行。

MQSC 命令报告

runmqsc 命令返回 *report*，该报告将发送到 stdout。 报告包含:

- 用于将 MQSC 命令标识为报告源的头:

  ```
    Starting MQSC for queue manager jupiter.queue.manager.
  ```

  其中， jupiter.queue.manager 是队列管理器的名称。
- 发出的 MQSC 命令的可选编号列表。 缺省情况下，输入的文本将回传到输出。 在此输出中，每个命令都以序号作为前缀，如 第 70 页的图 13 中所示。 但是，可以在 runmqsc 命令上使用 -e 标志来禁止输出。
- 任何发现出错的命令的语法错误消息。
- 操作员消息，指示运行每个命令的结果。 例如，成功完成 DEFINE QLOCAL 命令的操作程序消息为:

  ```
    AMQ8006: WebSphere MQ queue created.
  ```
- 运行脚本文件时由于一般错误而产生的其他消息。
- 报告的简要统计摘要，指示读取的命令数，存在语法错误的命令数以及无法处理的命令数。

  注: 队列管理器尝试仅处理那些没有语法错误的命令。

```
Starting MQSC for queue manager jupiter.queue.manager.
 .
 .
     12:     DEFINE QLOCAL('ORANGE.LOCAL.QUEUE') REPLACE  +
       :             DESCR(' ') +
       :             PUT(ENABLED) +
       :             DEFPRTY(0) +
       :             DEFPSIST(NO) +
       :             GET(ENABLED) +
       :             MAXDEPTH(5000) +
       :             MAXMSGL(1024) +
       :             DEFSOPT(SHARED) +
       :             NOHARDENBO +
       :             USAGE(NORMAL) +
       :             NOTRIGGER;
AMQ8006: WebSphere MQ queue created.
       :
 .
 .
```

图 *13:* 从 *MQSC* 命令报告文件中抽取

运行提供的 MQSC 命令文件

WebSphere MQ 随附了以下 MQSC 命令文件:

amqscos0.tst
样本程序使用的对象的定义。

amqscic0.tst
CICS 事务的队列定义。

在 WebSphere MQ for Windows 中，这些文件位于目录 *MQ_INSTALLATION_PATH*\tools\mqsc\samples 中。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。

在 UNIX and Linux 系统上，这些文件位于目录 *MQ_INSTALLATION_PATH*/samp 中。 *MQ_INSTALLATION_PATH* 表示安装了 WebSphere MQ 的高级目录。

运行这些命令的命令为:

```
runmqsc < amqscos0.tst >test.out
```

使用 runmqsc 验证命令

您可以使用 runmqsc 命令来验证本地队列管理器上的 MQSC 命令，而不必实际运行这些命令。 为此，请在 runmqsc 命令中设置 -v 标志，例如:

```
runmqsc -v < myprog.in > myprog.out
```

对 MQSC 命令文件调用 runmqsc 时，队列管理器会验证每个命令并返回报告，而不会实际运行 MQSC 命令。 这允许您检查命令文件中命令的语法。 如果您执行以下操作，那么这一点尤为重要:

- 从命令文件运行大量命令。
- 多次使用 MQSC 命令文件。

返回的报告与 第 70 页的图 13 中显示的报告类似。

不能使用此方法远程验证 MQSC 命令。 例如，如果尝试此命令:

```
runmqsc -w 30 -v jupiter.queue.manager < myprog.in > myprog.out
```

将忽略用于指示队列管理器是远程的 -w 标志，并以验证方式在本地运行该命令。 30 是 WebSphere MQ 等待来自远程队列管理器的应答的秒数。

从批处理文件运行 MQSC 命令

如果您有很长的命令，或者正在重复使用特定的命令序列，请考虑从批处理文件重定向 `stdin` 。

要从批处理文件重定向 `stdin` ，请首先使用常规文本编辑器创建包含 MQSC 命令的批处理文件。 使用 `runmqsc` 命令时，请使用重定向运算符。 以下示例：

1. 创建测试队列管理器 TESTQM
2. 创建匹配的 CLNTCONN 和侦听器集以使用 TCP/IP 端口 1600
3. 创建测试队列 TESTQ
4. 使用 amqsputc 样本程序将消息放入队列

```
export MYTEMPQM=TESTQM
export MYPORT=1600
export MQCHLLIB=/var/mqm/qmgrs/$MQTEMPQM/@ipcc

crtmqm $MYTEMPQM
strmqm $MYTEMPQM
runmqlsr -m $MYTEMPQM -t TCP -p $MYPORT &

runmqsc $MYTEMPQM << EOF
  DEFINE CHANNEL(NTLM) CHLTYPE(SVRCONN) TRPTYPE(TCP)
  DEFINE CHANNEL(NTLM) CHLTYPE(CLNTCONN) QMNAME('$MYTEMPQM') CONNAME('hostname($MYPORT)')
  ALTER  CHANNEL(NTLM) CHLTYPE(CLNTCONN)
  DEFINE QLOCAL(TESTQ)
EOF

amqsputc TESTQ $MYTEMPQM << EOF
hello world
EOF

endmqm -i $MYTEMPQM
```

图 *14:* 用于从批处理文件运行 *MQSC* 命令的示例脚本

解决 MQSC 命令的问题

如果您无法运行 MQSC 命令，请使用本主题中的信息，了解这些常见问题中的任何一个问题是否适用于您。 当您阅读命令生成的错误时，产生的问题并不总是很明显。

如果无法运行 MQSC 命令，请使用以下信息来查看这些常见问题是否适用于您。 当您读取生成的错误时，问题并不总是很明显。

使用 `runmqsc` 命令时，请记住以下内容：

- 使用 **<** 运算符可从文件重定向输入。 如果省略此运算符，那么队列管理器会将文件名解释为队列管理器名称，并发出以下错误消息：

  ```
  AMQ8118: WebSphere MQ queue manager does not exist.
  ```

- 如果将输出重定向到文件，请使用 **>** 重定向运算符。 缺省情况下，在调用 `runmqsc` 时，会将该文件放入当前工作目录中。 指定标准文件名，以将输出发送至特定文件和目录。
- 通过使用以下命令显示所有队列管理器，检查您是否已创建要运行这些命令的队列管理器：

  ```
  dspmq
  ```

- 该队列管理器必须正在运行。 如果不是，请将其启动; (请参阅 启动队列管理器)。 如果尝试启动已在运行的队列管理器，那么会收到错误消息。

- 如果尚未定义缺省队列管理器，请在 runmqsc 命令上指定队列管理器名称，否则将发生以下错误:

```
AMQ8146: WebSphere MQ queue manager not available.
```

- 不能将 MQSC 命令指定为 runmqsc 命令的参数。 例如，以下内容无效:

```
runmqsc DEFINE QLOCAL(FRED)
```

- 在发出 runmqsc 命令之前，不能输入 MQSC 命令。
- 不能从 runmqsc 运行控制命令。 例如，当您以交互方式运行 MQSC 命令时，无法发出 strmqm 命令来启动队列管理器。 如果执行此操作，那么将收到类似于以下内容的错误消息:

```
runmqsc
 .
 .
Starting MQSC for queue manager jupiter.queue.manager.

    1 : strmqm saturn.queue.manager
AMQ8405: Syntax error detected at or near end of cmd segment below:-s

AMQ8426: Valid MQSC commands are:
    ALTER
    CLEAR
    DEFINE
    DELETE
    DISPLAY
    END
    PING
    REFRESH
    RESET
    RESOLVE
    RESUME
    START
    STOP
    SUSPEND
     2 : end
```

使用队列管理器

可用于显示或更改队列管理器属性的 MQSC 命令的示例。

显示队列管理器属性

要显示在 **runmqsc** 命令上指定的队列管理器的属性，请使用以下 MQSC 命令:

```
DISPLAY QMGR
```

此命令的典型输出显示在 第 73 页的图 15 中

```
DISPLAY QMGR
     1 : DISPLAY QMGR
AMQ8408: Display Queue Manager details.
   QMNAME(QM1)                             ACCTCONO(DISABLED)
   ACCTINT(1800)                           ACCTMQI(OFF)
   ACCTQ(OFF)                              ACTIVREC(MSG)
   ACTVCONO (DISABLED)                     ACTVTRC (OFF)
   ALTDATE(2012-05-27)                     ALTTIME(16.14.01)
   AUTHOREV(DISABLED)                      CCSID(850)
   CHAD(DISABLED)                          CHADEV(DISABLED)
   CHADEXIT( )                             CHLEV(DISABLED)
   CLWLDATA( )                             CLWLEXIT( )
   CLWLLEN(100)                            CLWLMRUC(999999999)
   CLWLUSEQ(LOCAL)                         CMDEV(DISABLED)
   CMDLEVEL(750)                           COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
   CONFIGEV(DISABLED)                      CRDATE(2011-05-27)
   CRTIME(16.14.01)                        DEADQ()
   DEFXMITQ( )                             DESCR( )
   DISTL(YES)                              INHIBTEV(DISABLED)
   IPADDRV(IPV4)                           LOCALEV(DISABLED)
   LOGGEREV(DISABLED)                      MARKINT(5000)
   MAXHANDS(256)                           MAXMSGL(4194304)
   MAXPROPL(NOLIMIT)                       MAXPRTY(9)
   MAXUMSGS(10000)                         MONACLS(QMGR)
   MONCHL(OFF)                             MONQ(OFF)
   PARENT( )                               PERFMEV(DISABLED)
   PLATFORM(WINDOWSNT)                        PSRTYCNT(5)
   PSNPMSG(DISCARD)                        PSNPRES(NORMAL)
   PSSYNCPT(IFPER)                         QMID(QM1_2011-05-27_16.14.01)
   PSMODE(ENABLED)                         REMOTEEV(DISABLED)
   REPOS( )                                REPOSNL( )
   ROUTEREC(MSG)                           SCHINIT(QMGR)
   SCMDSERV(QMGR)                          SSLCRLNL( )
   SSLCRYP( )                              SSLEV(DISABLED)
   SSLFIPS(NO)                             SSLKEYR(C:\Program Files\IBM\WebSphere
MQ\Data\qmgrs\QM1\ssl\key)
   SSLRKEYC(0)                             STATACLS(QMGR)
   STATCHL(OFF)                            STATINT(1800)
   STATMQI(OFF)                            STATQ(OFF)
   STRSTPEV(ENABLED)                       SYNCPT
   TREELIFE(1800)                          TRIGINT(999999999)
```

图 *15: DISPLAY QMGR* 命令的典型输出

ALL 参数是 DISPLAY QMGR 命令的缺省值。 它显示所有队列管理器属性。 特别是，输出会告诉您缺省队列管理器名称，死信队列名称和命令队列名称。

您可以通过输入以下命令来确认这些队列是否存在:

```
DISPLAY QUEUE (SYSTEM.*)
```

这将显示与词干 SYSTEM.*匹配的队列的列表。 括号是必需的。

更改队列管理器属性

要更改在 **runmqsc** 命令上指定的队列管理器的属性，请使用 MQSC 命令 ALTER QMGR，指定要更改的属性和值。 例如，使用以下命令来变更 jupiter.queue.manager 的属性:

```
runmqsc jupiter.queue.manager
ALTER QMGR DEADQ (ANOTHERDLQ) INHIBTEV (ENABLED)
```

ALTER QMGR 命令更改使用的死信队列，并启用禁止事件。

相关参考

队列管理器的属性

使用本地队列

本部分包含可用于管理本地队列，模型队列和别名队列的一些 MQSC 命令的示例。

请参阅 MQSC 引用 以获取有关这些命令的详细信息。

定义本地队列

对于应用程序，本地队列管理器是应用程序所连接的队列管理器。 由本地队列管理器管理的队列据说是该队列管理器的本地队列。

使用 MQSC 命令 DEFINE QLOCAL 来创建本地队列。 您还可以使用缺省本地队列定义中定义的缺省值，也可以修改缺省本地队列的队列特征。

注: 缺省本地队列名为 SYSTEM.DEFAULT.LOCAL.QUEUE ，它是在系统安装时创建的。

例如，下面的 DEFINE QLOCAL 命令定义了具有以下特征的名为 ORANGE.LOCAL.QUEUE 的队列:

- 它针对获取启用，针对放置启用，并按优先级顺序运行。
- 它是 普通 队列; 它不是启动队列或传输队列，并且不会生成触发器消息。
- 最大队列深度为 5000 条消息; 最大消息长度为 4194304 字节。

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) +
       DESCR('Queue for messages from other systems') +
       PUT (ENABLED) +
       GET (ENABLED) +
       NOTRIGGER +
       MSGDLVSQ (PRIORITY) +
       MAXDEPTH (5000) +
       MAXMSGL (4194304) +
       USAGE (NORMAL);
```

注:

1. 除了描述的值以外，显示的所有属性值都是缺省值。 我们在这里展示了这些信息，以示说明。 如果您确定缺省值是您想要的或尚未更改的值，那么可以省略这些值。 另请参阅第 74 页的『显示缺省对象属性』。
2. USAGE (NORMAL) 指示此队列不是传输队列。
3. 如果已在同一队列管理器上具有名为 ORANGE.LOCAL.QUEUE，此命令失败。 如果要覆盖队列的现有定义，请使用 REPLACE 属性，但另请参阅 第 75 页的『更改本地队列属性』。

定义死信队列

每个队列管理器都必须具有要用作死信队列的本地队列，以便可以存储无法传递到其正确目标的消息以供以后检索。 您必须告知队列管理器有关死信队列的信息。

要告知队列管理器有关死信队列的信息，请在 `crtmqm` 命令 (例如 `crtmqm -u DEAD.LETTER.QUEUE`) 上指定死信队列名称，或者通过在 ALTER QMGR 命令上使用 DEADQ 属性来稍后指定一个死信队列名称。 您必须先定义死信队列，然后才能使用该队列。

名为 SYSTEM.DEAD.LETTER.QUEUE 随产品一起提供。 创建队列管理器时，将自动创建此队列。 如果需要，可以修改此定义，并将其重命名。

死信队列没有特殊要求，只是:

- 它必须是本地队列
- 其 MAXMSGL (最大消息长度) 属性必须使队列能够容纳队列管理器必须处理的最大消息 **加上** 死信头 (MQDLH) 的大小

WebSphere MQ 提供了一个死信队列处理程序，允许您指定如何处理或除去在死信队列上找到的消息。 有关更多信息，请参阅 使用 WebSphere MQ 死信队列处理程序处理未传递的消息 。

显示缺省对象属性

您可以使用 DISPLAY QUEUE 命令来显示在定义 WebSphere MQ 对象时从缺省对象获取的属性。

当您定义 WebSphere MQ 对象时，它将采用未从缺省对象指定的任何属性。 例如，当您定义本地队列时，该队列将从缺省本地队列 (称为 SYSTEM.DEFAULT.LOCAL.QUEUE。 要确切查看这些属性的内容，请使用以下命令:

```
DISPLAY QUEUE (SYSTEM.DEFAULT.LOCAL.QUEUE)
```

此命令的语法与相应 DEFINE 命令的语法不同。 在 DISPLAY 命令上，您只能提供队列名称，而在 DEFINE 命令上，您必须指定队列的类型，即 QLOCAL， QALIAS， QMODEL 或 QREMOTE。

您可以通过单独指定属性来选择性地显示这些属性。 例如:

```
DISPLAY QUEUE (ORANGE.LOCAL.QUEUE) +
        MAXDEPTH +
        MAXMSGL +
        CURDEPTH;
```

此命令显示三个指定的属性，如下所示:

```
AMQ8409: Display Queue details.
   QUEUE(ORANGE.LOCAL.QUEUE)               TYPE(QLOCAL)
   CURDEPTH(0)                             MAXDEPTH(5000)
   MAXMSGL(4194304)
```

CURDEPTH 是当前队列深度，即队列上的消息数。 这是要显示的有用属性，因为通过监视队列深度，可以确保队列不会变满。

复制本地队列定义

可以使用 DEFINE 命令中的 LIKE 属性来复制队列定义。

例如:

```
DEFINE QLOCAL (MAGENTA.QUEUE) +
       LIKE (ORANGE.LOCAL.QUEUE)
```

此命令将创建与原始队列 ORANGE.LOCAL.QUEUE，而不是系统缺省本地队列的队列。 输入要复制的队列的名称 **与创建队列时输入的名称** 完全相同。 如果名称包含小写字符，请将名称括在单引号中。

您还可以使用此格式的 DEFINE 命令来复制队列定义，但替换原始属性的一个或多个更改。 例如:

```
DEFINE QLOCAL (THIRD.QUEUE) +
       LIKE (ORANGE.LOCAL.QUEUE) +
       MAXMSGL(1024);
```

此命令将复制队列 ORANGE.LOCAL.QUEUE 到队列 THIRD.QUEUE，但指定新队列上的最大消息长度为 1024 字节，而不是 4194304 字节。

注:

1. 在 DEFINE 命令上使用 LIKE 属性时，仅复制队列属性。 您未在复制队列上的消息。
2. 如果定义本地队列而不指定 LIKE，那么它与 DEFINE LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)。

更改本地队列属性

可以通过两种方式更改队列属性，即使用 ALTER QLOCAL 命令或带有 REPLACE 属性的 DEFINE QLOCAL 命令。

在 第 74 页的『定义本地队列』 中，队列名为 ORANGE.LOCAL.QUEUE 。 例如，假设您希望将此队列上的最大消息长度减少到 10,000 字节。

- 使用 ALTER 命令:

```
ALTER QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000)
```

此命令会更改单个属性，即最大消息长度的属性; 所有其他属性保持不变。

- 使用带有 REPLACE 选项的 DEFINE 命令，例如:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000) REPLACE
```

此命令不仅会更改最大消息长度，还会更改所有其他属性 (给定它们的缺省值)。 现在已启用该队列，而先前已禁止将其放入。 启用放置是缺省值，由队列 SYSTEM.DEFAULT.LOCAL.QUEUE。

如果 **减小** 现有队列上的最大消息长度，那么现有消息不受影响。 但是，任何新消息都必须满足新条件。

清除本地队列

可以使用 CLEAR 命令来清除本地队列。

从名为 MAGENTA.QUEUE，使用以下命令:

```
CLEAR QLOCAL (MAGENTA.QUEUE)
```

注: 没有使您能够改变主意的提示; 当您按 Enter 键时，消息将丢失。

在下列情况下，无法清除队列:

- 有未落实的消息已放在同步点下的队列上。
- 某个应用程序当前打开了该队列。

删除本地队列

可以使用 MQSC 命令 DELETE QLOCAL 来删除本地队列。

如果队列上有未落实的消息，那么无法删除该队列。 但是，如果队列有一条或多条已落实的消息，并且没有未落实的消息，那么仅当指定 PURGE 选项时，才能将其删除。 例如:

```
DELETE QLOCAL (PINK.QUEUE) PURGE
```

指定 NOPURGE 而不是 PURGE 可确保如果队列包含任何已落实的消息，那么不会将其删除。

浏览队列

WebSphere MQ 提供了一个样本队列浏览器，可用于查看队列中消息的内容。 浏览器以源格式和可执行格式提供。

MQ_INSTALLATION_PATH 表示安装 WebSphere MQ 的高级目录。

在 WebSphere MQ for Windows 中，样本队列浏览器的文件名和路径如下所示:

来源
MQ_INSTALLATION_PATH\tools\c\samples\

可执行文件
MQ_INSTALLATION_PATH\tools\c\samples\bin\amqsbcg.exe

在 WebSphere MQ for UNIX and Linux 中，文件名和路径如下所示:

来源
MQ_INSTALLATION_PATH/samp/amqsbcg0.c

可执行文件
MQ_INSTALLATION_PATH/samp/bin/amqsbcg

样本需要两个输入参数，即队列名称和队列管理器名称。 例如：

```
amqsbcg SYSTEM.ADMIN.QMGREVENT.tpp01 saturn.queue.manager
```

此命令的典型结果显示在 第 77 页的图 16 中。

```
AMQSBCG0 - starts here
**********************

 MQOPEN - 'SYSTEM.ADMIN.QMGR.EVENT'


 MQGET of message number 1
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 850
  Format : 'MQEVENT '
  Priority : 0  Persistence : 0
  MsgId : X'414D512073617475726E2E71756575650005D30033563DB8'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'saturn.queue.manager                          '
  ** Identity Context
  UserIdentifier : '            '
  AccountingToken :
   X'0000000000000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                        '
  ** Origin Context
  PutApplType    : '7'
  PutApplName    : 'saturn.queue.manager        '
  PutDate  : '19970417'    PutTime  : '15115208'
  ApplOriginData : '    '

  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '104'

****   Message      ****

 length - 104 bytes

00000000:  0700 0000 2400 0000 0100 0000 2C00 0000 '....→.......,...'
00000010:  0100 0000 0100 0000 0100 0000 AE08 0000 '................'
00000020:  0100 0000 0400 0000 4400 0000 DF07 0000 '........D.......'
00000030:  0000 0000 3000 0000 7361 7475 726E 2E71 '....0...saturn.q'
00000040:  7565 7565 2E6D 616E 6167 6572 2020 2020 'ueue.manager    '
00000050:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060:  2020 2020 2020 2020                     '                '

 No more messages
 MQCLOSE
 MQDISC
```

图 *16:* 来自队列浏览器的典型结果

启用大型队列

IBM WebSphere MQ 支持大于 2 GB 的队列。

在 Windows 系统上，提供了对大型文件的支持，而无需任何其他支持。 在 AIX， HP-UX， Linux 和 Solaris 系统上，您需要显式启用大型文件支持，然后才能创建大于 2 GB 的队列文件。 请参阅操作系统文档，以获取有关如何执行此操作的信息。

某些实用程序 (例如 tar) 无法处理大于 2 GB 的文件。 在启用大型文件支持之前，请查看操作系统文档以获取有关使用的实用程序限制的信息。

有关规划队列所需的存储量的信息，请访问 IBM WebSphere MQ Web 站点以获取特定于平台的性能报告:

```
https://www.ibm.com/software/integration/ts/mqseries/
```

使用别名队列

您可以定义别名队列来间接引用其他队列或主题。

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。

别名队列引用的队列可以是以下任意项:

- 本地队列（请参阅第 74 页的『定义本地队列』）。
- 远程队列的本地定义（请参阅第 99 页的『创建远程队列的本地定义』）。
- 主题。

别名队列不是真实的队列，而是在运行时解析为真实（或目标）队列的定义。 别名队列定义指定目标队列。 当应用程序对别名队列进行 MQOPEN 调用时，此队列管理器将别名解析为目标队列名称。

别名队列无法解析为本地定义的其他别名队列。 然而，别名队列可解析为在本地队列管理器所属集群中的其他位置定义的别名队列。 请参阅名称解析，以获取进一步的信息。

别名队列有助于:

- 为不同的应用程序提供对目标队列的不同访问权限级别。
- 允许不同的应用程序以不同方式使用相同的队列。 （或许您想指定不同的缺省优先级或不同的缺省持久性值。）
- 简化维护、迁移和工作负载均衡。 （或许您想在无须更改应用程序的情况下更改目标队列名称，此时应用程序继续使用别名。）

例如，假设开发了一个应用程序，用于将消息放入队列 MY.ALIAS.QUEUE。 它在发出 MQOPEN 请求时指定此队列的名称，并且在将消息放入此队列时会间接指定名称。 此应用程序不知道此队列是别名队列。 对于使用此别名的每个 MQI 调用，此队列管理器会解析实际队列名称（可以是在此队列管理器上定义的本地队列或远程队列）。

通过更改 TARGET 属性的值，可以将 MQI 调用重定向到另一个队列 (可能在另一个队列管理器上)。 这有助于维护、迁移和负载均衡。

定义别名队列

以下命令创建别名队列:

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGET (YELLOW.QUEUE)
```

此命令会将指定 MY.ALIAS.QUEUE 的 MQI 调用重定向至队列 YELLOW.QUEUE。 此命令不创建目标队列；如果队列 YELLOW.QUEUE 在运行时不存在，那么 MQI 调用会失败。

如果您更改此别名定义，可以将 MQI 调用重定向至另一个队列。 例如:

```
ALTER QALIAS (MY.ALIAS.QUEUE) TARGET (MAGENTA.QUEUE)
```

此命令将 MQI 调用重定向至另一个队列 MAGENTA.QUEUE。

您也可以使用别名队列，使单个队列（目标队列）对于不同的应用程序看起来具有不同的属性。 可通过定义两个别名（对每个应用程序各定义一个）来实现这一点。 假设有两个应用程序:

- 应用程序 ALPHA 可以将消息放入 YELLOW.QUEUE，但不允许它从此队列取出消息。
- 应用程序 BETA 可以从 YELLOW.QUEUE 取出消息，但不允许它将消息放入此队列。

以下命令定义对应用程序 ALPHA 启用放入和禁用取出的别名：

```
DEFINE QALIAS (ALPHAS.ALIAS.QUEUE) +
       TARGET (YELLOW.QUEUE) +
       PUT (ENABLED) +
       GET (DISABLED)
```

以下命令定义对应用程序 BETA 禁用放入和启用取出的别名：

```
DEFINE QALIAS (BETAS.ALIAS.QUEUE) +
       TARGET (YELLOW.QUEUE) +
       PUT (DISABLED) +
       GET (ENABLED)
```

ALPHA 在其 MQI 调用中使用队列名称 ALPHAS.ALIAS.QUEUE；BETA 使用队列名称 BETAS.ALIAS.QUEUE。 它们都访问相同的队列，但以不同的方式访问。

您可以在定义队列别名时使用 LIKE 和 REPLACE 属性，与您将这些属性用于本地队列的方式相同。

对别名队列使用其他命令

您可以使用相应的 MQSC 命令来显示或更改别名队列属性，或删除此别名队列对象。 例如：

使用以下命令来显示别名队列的属性：

```
DISPLAY QALIAS (ALPHAS.ALIAS.QUEUE)
```

使用以下命令来更改将别名解析为的基本队列名称，其中 `force` 选项强制此更改，即使此队列是开放的：

```
ALTER QALIAS (ALPHAS.ALIAS.QUEUE) TARGET(ORANGE.LOCAL.QUEUE) FORCE
```

使用以下命令来删除此队列别名：

```
DELETE QALIAS (ALPHAS.ALIAS.QUEUE)
```

如果应用程序当前打开了该队列，那么您无法删除别名队列。 请参阅 MQSC 引用 ，以获取有关此别名队列命令和其他别名队列命令的更多信息。

使用模型队列

如果队列管理器从指定已定义为模型队列的队列名称的应用程序接收 MQI 调用，那么它将创建 动态队列 。 新动态队列的名称由队列管理器在创建队列时生成。 模型队列 是一个模板，用于指定从中创建的任何动态队列的属性。 模型队列为应用程序提供了一种方便的方法来根据需要创建队列。

定义模型队列

使用一组属性定义模型队列的方式与定义本地队列的方式相同。 模型队列和本地队列具有相同的属性集，但在模型队列上，您可以指定所创建的动态队列是临时的还是永久的。 (在队列管理器重新启动之间保留永久队列，但不保留临时队列。) 例如：

```
DEFINE QMODEL (GREEN.MODEL.QUEUE) +
       DESCR('Queue for messages from application X') +
       PUT (DISABLED) +
       GET (ENABLED) +
       NOTRIGGER +
       MSGDLVSQ (FIFO) +
```

```
       MAXDEPTH (1000) +
       MAXMSGL (2000) +
       USAGE (NORMAL) +
       DEFTYPE (PERMDYN)
```

此命令创建模型队列定义。 从 DEFTYPE 属性中，您可以看到从此模板创建的实际队列是永久动态队列。 将自动从 SYSYTEM.DEFAULT.MODEL.QUEUE 缺省队列。

在定义模型队列时，可以使用 LIKE 和 REPLACE 属性，方式与将它们与本地队列配合使用的方式相同。

将其他命令与模型队列配合使用

您可以使用相应的 MQSC 命令来显示或更改模型队列的属性，或者删除模型队列对象。 例如：

使用以下命令来显示模型队列的属性:

```
DISPLAY QUEUE (GREEN.MODEL.QUEUE)
```

使用以下命令来变更模型，以启用从此模型创建的任何动态队列上的放入:

```
ALTER QMODEL (BLUE.MODEL.QUEUE) PUT(ENABLED)
```

使用以下命令来删除此模型队列:

```
DELETE QMODEL (RED.MODEL.QUEUE)
```

处理管理主题

使用 MQSC 命令来管理管理主题。

请参阅 MQSC 引用 ，以获取有关这些命令的详细信息。

相关概念

管理主题对象

第 80 页的『定义管理主题』
使用 MQSC 命令 **DEFINE TOPIC** 来创建管理主题。 定义管理主题时，可以选择设置每个主题属性。

第 81 页的『显示管理主题对象属性』
使用 MQSC 命令 **DISPLAY TOPIC** 显示管理主题对象。

第 81 页的『更改管理主题属性』
您可以通过两种方式更改主题属性，即使用 **ALTER TOPIC** 命令或带有 **REPLACE** 属性的 **DEFINE TOPIC** 命令。

第 82 页的『复制管理主题定义』
您可以在 **DEFINE** 命令上使用 LIKE 属性来复制主题定义。

第 82 页的『删除管理主题定义』
您可以使用 MQSC 命令 **DELETE TOPIC** 来删除管理主题。

定义管理主题

使用 MQSC 命令 **DEFINE TOPIC** 来创建管理主题。 定义管理主题时，可以选择设置每个主题属性。

未显式设置的主题的任何属性都将从缺省管理主题 SYSTEM.DEFAULT.TOPIC，在安装系统安装时创建。

例如，下面的 **DEFINE TOPIC** 命令定义了具有以下特征的名为 **ORANGE.TOPIC** 的主题:

- 解析为主题字符串 ORANGE。 有关如何使用主题字符串的信息，请参阅 组合主题字符串。

- 设置为 ASPARENT 的任何属性都将使用此主题的父主题所定义的属性。 此操作在主题树上重复到根主题 SYSTEM.BASE.TOPIC 。 有关主题树的更多信息，请参阅 主题树。

```
DEFINE TOPIC (ORANGE.TOPIC) +
       TOPICSTR (ORANGE) +
       DEFPRTY(ASPARENT) +
       NPMSGDLV(ASPARENT)
```

注:

- 除主题字符串的值外，显示的所有属性值都是缺省值。 此处仅显示为示例。 如果您确定缺省值是您想要的或尚未更改的值，那么可以省略这些值。 另请参阅第 81 页的『显示管理主题对象属性』。
- 如果已在同一队列管理器上具有名为 ORANGE.TOPIC，此命令失败。 如果要覆盖主题的现有定义，请使用 REPLACE 属性，但另请参阅 第 81 页的『更改管理主题属性』

显示管理主题对象属性

使用 MQSC 命令 **DISPLAY TOPIC** 显示管理主题对象。

要显示所有主题，请使用:

```
DISPLAY TOPIC(ORANGE.TOPIC)
```

您可以通过单独指定属性来选择性地显示这些属性。例如:

```
DISPLAY TOPIC(ORANGE.TOPIC) +
        TOPICSTR +
        DEFPRTY +
        NPMSGDLV
```

此命令显示三个指定的属性，如下所示:

```
AMQ8633: Display topic details.
   TOPIC(ORANGE.TOPIC)                     TYPE(LOCAL)
   TOPICSTR(ORANGE)                        DEFPRTY(ASPARENT)
   NPMSGDLV(ASPARENT)
```

要在运行时使用主题 ASPARENT 值时显示这些值，请使用 DISPLAY TPSTATUS。 例如，使用:

```
DISPLAY TPSTATUS(ORANGE) DEFPRTY NPMSGDLV
```

该命令显示以下详细信息:

```
AMQ8754: Display topic status details.
   TOPICSTR(ORANGE)                        DEFPRTY(0)
   NPMSGDLV(ALLAVAIL)
```

定义管理主题时，它将采用您未从缺省管理主题 (称为 SYSTEM.DEFAULT.TOPIC。 要查看这些缺省属性是什么，请使用以下命令:

```
DISPLAY TOPIC (SYSTEM.DEFAULT.TOPIC)
```

更改管理主题属性

您可以通过两种方式更改主题属性，即使用 **ALTER TOPIC** 命令或带有 **REPLACE** 属性的 **DEFINE TOPIC** 命令。

例如，如果要更改传递到名为 ORANGE.TOPIC 为 5 ，请使用以下任一命令。

- 使用 **ALTER** 命令:

  ```
  ALTER TOPIC(ORANGE.TOPIC) DEFPRTY(5)
  ```

  此命令将传递到此主题的消息的缺省优先级的单个属性更改为 5; 所有其他属性保持不变。

- 使用 **DEFINE** 命令:

```
DEFINE TOPIC(ORANGE.TOPIC) DEFPRTY(5) REPLACE
```

此命令将更改传递到此主题的消息的缺省优先级。 将为所有其他属性提供其缺省值。

如果更改发送到此主题的消息的优先级，那么现有消息不受影响。 但是，如果发布应用程序未提供任何新消息，那么将使用指定的优先级。

复制管理主题定义

您可以在 **DEFINE** 命令上使用 LIKE 属性来复制主题定义。

例如:

```
DEFINE TOPIC (MAGENTA.TOPIC) +
       LIKE (ORANGE.TOPIC)
```

此命令创建主题 MAGENTA.TOPIC，具有与原始主题 ORANGE.TOPIC，而不是系统缺省管理主题的主题。 输入要复制的主题的名称，与创建主题时输入的名称完全相同。 如果名称包含小写字符，请将名称括在单引号中。

您还可以使用此格式的 **DEFINE** 命令来复制主题定义，但对原始属性进行更改。 例如:

```
DEFINE TOPIC(BLUE.TOPIC) +
       TOPICSTR(BLUE) +
       LIKE(ORANGE.TOPIC)
```

您还可以将主题 BLUE.TOPIC 的属性复制到主题 GREEN.TOPIC ，并指定当发布无法传递到其正确的订户队列时，不会将其放入死信队列中。 例如:

```
DEFINE TOPIC(GREEN.TOPIC) +
       TOPICSTR(GREEN) +
       LIKE(BLUE.TOPIC) +
       USEDLQ(NO)
```

删除管理主题定义

您可以使用 MQSC 命令 **DELETE TOPIC** 来删除管理主题。

```
DELETE TOPIC(ORANGE.TOPIC)
```

应用程序将无法再打开主题以进行发布或使用对象名 ORANGE.TOPIC。 发布打开了主题的应用程序能够继续发布已解析的主题字符串。 已对此主题进行的任何预订都将在删除此主题后继续接收发布。

未引用此主题对象但正在使用此主题对象所表示的已解析主题字符串 (在此示例中为 "ORANGE") 的应用程序将继续工作。 在这种情况下，它们将从主题树中更高的主题对象继承属性。 有关主题树的更多信息，请参阅 主题树。

使用预订

使用 MQSC 命令来管理预订。

预订可以是 **SUBTYPE** 属性中定义的三种类型之一:

管理
由用户以管理方式定义。

PROXY
用于在队列管理器之间路由发布的内部创建的预订。

API
以编程方式创建，例如，使用 MQI MQSUB 调用。

请参阅 MQSC 引用 以获取有关这些命令的详细信息。

相关概念

第 83 页的『定义管理预订』
使用 MQSC 命令 **DEFINE SUB** 来创建管理预订。 您还可以使用缺省本地预订定义中定义的缺省值。 或者，您可以从缺省本地预订 SYSTEM.DEFAULT.SUB 。

第 83 页的『显示预订的属性』
您可以使用 **DISPLAY SUB** 命令来显示队列管理器已知的任何预订的已配置属性。

第 84 页的『更改本地预订属性』
可以通过两种方式更改预订属性，使用 **ALTER SUB** 命令或带有 **REPLACE** 属性的 **DEFINE SUB** 命令。

第 84 页的『复制本地预订定义』
您可以使用 **DEFINE** 命令上的 **LIKE** 属性来复制预订定义。

第 85 页的『删除预订』
可以使用 MQSC 命令 **DELETE SUB** 来删除本地预订。

定义管理预订

使用 MQSC 命令 **DEFINE SUB** 来创建管理预订。 您还可以使用缺省本地预订定义中定义的缺省值。 或者，您可以从缺省本地预订 SYSTEM.DEFAULT.SUB 。

例如，以下 **DEFINE SUB** 命令定义了具有以下特征的名为 ORANGE 的预订:

- 持久预订，意味着它在队列管理器重新启动后持久存在，并且到期时间不限。
- 接收对 ORANGE 主题字符串进行的发布，消息优先级由发布应用程序设置。
- 为此预订交付的发布将发送到本地队列 SUBQ ，必须在定义预订之前定义此队列。

```
DEFINE SUB (ORANGE) +
       TOPICSTR (ORANGE) +
       DESTCLAS (PROVIDED) +
       DEST (SUBQ) +
       EXPIRY (UNLIMITED) +
       PUBPRTY (ASPUB)
```

注:

- 预订和主题字符串名称不必匹配。
- 除了描述和主题字符串的值以外，显示的所有属性值都是缺省值。 此处仅显示为示例。 如果您确定缺省值是您想要的或尚未更改的值，那么可以省略这些值。 另请参阅第 83 页的『显示预订的属性』。
- 如果您已在同一队列管理器上具有名为 TEST 的本地预订，那么此命令将失败。 如果要覆盖队列的现有定义，请使用 **REPLACE** 属性，但另请参阅 第 84 页的『更改本地预订属性』。
- 如果队列 SUBQ 不存在，那么此命令将失败。

显示预订的属性

您可以使用 **DISPLAY SUB** 命令来显示队列管理器已知的任何预订的已配置属性。

例如，使用:

```
DISPLAY SUB (ORANGE)
```

您可以通过单独指定属性来选择性地显示这些属性。 例如:

```
DISPLAY SUB (ORANGE) +
        SUBID +
        TOPICSTR +
        DURABLE
```

此命令显示三个指定的属性，如下所示:

```
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120414141202020202020202020EE921E4E20002A03)
```

```
   SUB(ORANGE)                                TOPICSTR(ORANGE)
   DURABLE(YES)
```

TOPICSTR 是此订户正在运行的已解析主题字符串。 定义预订以使用主题对象时，来自该对象的主题字符串将用作进行预订时提供的主题字符串的前缀。 SUBID 是创建预订时由队列管理器指定的唯一标识。 这是要显示的有用属性，因为某些预订名称可能是长的，或者位于可能变得不切实际的其他字符集中。

显示预订的备用方法是使用 SUBID:

```
DISPLAY SUB +
        SUBID(414D5120414141202020202020202020EE921E4E20002A03) +
        TOPICSTR +
        DURABLE
```

此命令提供与之前相同的输出:

```
AMQ8096: WebSphere MQ subscription inquired.
 SUBID(414D5120414141202020202020202020EE921E4E20002A03)
   SUB(ORANGE)                                TOPICSTR(ORANGE)
   DURABLE(YES)
```

缺省情况下，不会显示队列管理器上的代理预订。 要显示它们，请指定 **SUBTYPE** PROXY 或 ALL。

可以使用 DISPLAY SBSTATUS 命令来显示 "运行时" 属性。 例如，使用以下命令:

```
DISPLAY SBSTATUS(ORANGE) NUMMSGS
```

将显示以下输出:

```
AMQ8099: WebSphere MQ subscription status inquired.
   SUB(ORANGE)
   SUBID(414D5120414141202020202020202020EE921E4E20002A03)
   NUMMSGS(0)
```

定义管理预订时，它将采用您未从缺省预订 (称为 SYSTEM.DEFAULT.SUB。 要查看这些缺省属性是什么，请使用以下命令:

```
DISPLAY SUB (SYSTEM.DEFAULT.SUB)
```

更改本地预订属性

可以通过两种方式更改预订属性，使用 **ALTER SUB** 命令或带有 **REPLACE** 属性的 **DEFINE SUB** 命令。

例如，如果要将传递到名为 ORANGE 的预订的消息的优先级更改为 5 ，请使用以下任一命令:

- 使用 ALTER 命令:

  ```
  ALTER SUB(ORANGE) PUBPRTY(5)
  ```

  此命令将传递到此预订的消息的优先级的单个属性更改为 5; 所有其他属性保持不变。
- 使用 DEFINE 命令:

  ```
  DEFINE SUB (ORANGE) PUBPRTY(5) REPLACE
  ```

  此命令不仅会更改传递到此预订的消息的优先级，还会更改为其缺省值提供的所有其他属性。

如果更改发送到此预订的消息的优先级，那么现有消息不受影响。 但是，任何新消息都具有指定的优先级。

复制本地预订定义

您可以使用 **DEFINE** 命令上的 **LIKE** 属性来复制预订定义。

例如:

```
DEFINE SUB (BLUE) +
       LIKE (ORANGE)
```

您还可以将子 REAL 的属性复制到子 THIRD.SUB，并指定已交付的发布的 correlID 为 THIRD ，而不是发布程序 correlID。 例如：

```
DEFINE SUB(THIRD.SUB) +
       LIKE(BLUE) +
       DESTCORL(ORANGE)
```

删除预订

可以使用 MQSC 命令 **DELETE SUB** 来删除本地预订。

```
DELETE SUB(ORANGE)
```

您还可以使用 SUBID 删除预订：

```
DELETE SUB SUBID(414D5120414141202020202020202020EE921E4E20002A03)
```

检查预订上的消息

关于此任务

定义预订时，它与队列相关联。 与此预订匹配的已发布消息将放入此队列。

请注意，以下 **runmqsc** 命令仅显示接收消息的那些预订。

要检查当前排队等待预订的消息，请执行以下步骤：

过程

1. 要检查排队等待预订类型 **DISPLAY SBSTATUS(<sub_name>) NUMMSGS** 的消息，请参阅 第 83 页的『显示预订的属性』。
2. 如果 **NUMMSGS** 值大于零，请通过输入 **DISPLAY SUB(<sub_name>)DEST** 来标识与预订关联的队列。
3. 通过使用返回的队列名称，您可以遵循 第 76 页的『浏览队列』中描述的方法来查看消息。

使用服务

服务对象是将其他进程作为队列管理器的一部分进行管理的一种方法。 通过服务，您可以定义在队列管理器启动和结束时启动和停止的程序。 IBM WebSphere MQ 服务始终以启动队列管理器的用户的用户标识启动。

服务对象可以是下列其中一种类型：

服务器
服务器是将参数 SERVTYPE 指定为 SERVER 的服务对象。 服务器服务对象是在启动指定队列管理器时执行的程序的定义。 服务器服务对象定义通常运行很长时间的程序。 例如，可以使用服务器服务对象来执行触发器监视器进程，例如 `runmqtrm`。

只能同时运行服务器服务对象的一个实例。 可以使用 MQSC 命令 DISPLAY SVSTATUS 来监视正在运行的服务器服务对象的状态。

命令
命令是将参数 SERVTYPE 指定为 COMMAND 的服务对象。 命令服务对象类似于服务器服务对象，但是命令服务对象的多个实例可以同时运行，并且无法使用 MQSC 命令 DISPLAY SVSTATUS 来监视其状态。

如果执行 MQSC 命令 STOP SERVICE ，那么不会执行检查以确定 MQSC 命令 START SERVICE 启动的程序在执行停止程序之前是否仍处于活动状态。

定义服务对象

定义具有各种属性的服务对象。

这些属性如下所示：

SERVTYPE
定义服务对象的类型。 可能的值如下所示:

服务器
服务器服务对象。

一次只能执行服务器服务对象的一个实例。 可以使用 MQSC 命令 DISPLAY SVSTATUS 来监视服务器服务对象的状态。

COMMAND
命令服务对象。

可以同时执行命令服务对象的多个实例。 无法监视命令服务对象的状态。

STARTCMD
为启动服务而执行的程序。 必须指定程序的标准路径。

STARTARG
传递到启动程序的自变量。

STDERR
指定应该将服务程序的标准错误 (stderr) 重定向到的文件的路径。

STDOUT
指定应将服务程序的标准输出 (stdout) 重定向到的文件的路径。

STOPCMD
为停止服务而执行的程序。 必须指定程序的标准路径。

STOPARG
传递到停止程序的参数。

控制
指定如何启动和停止服务:

手动
服务不会自动启动或自动停止。 它通过使用 START SERVICE 和 STOP SERVICE 命令进行控制。 这是缺省值。

QMGR
要定义的服务将在启动和停止队列管理器的同时启动和停止。

仅启动
该服务将在队列管理器启动的同时启动，但不会在队列管理器停止时被请求停止。

相关概念
第 86 页的『管理服务』
通过使用 CONTROL 参数，服务对象的实例可以由队列管理器自动启动和停止，也可以使用 MQSC 命令 START SERVICE 和 STOP SERVICE 启动和停止。

管理服务

通过使用 CONTROL 参数，服务对象的实例可以由队列管理器自动启动和停止，也可以使用 MQSC 命令 START SERVICE 和 STOP SERVICE 启动和停止。

启动服务对象的实例时，将向队列管理器错误日志写入一条消息，其中包含服务对象的名称和已启动进程的进程标识。 以下是启动的服务器服务对象的示例日志条目:

```
  02/15/2005 11:54:24 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5028: The Server 'S1' has started. ProcessId(13031).

  EXPLANATION:
  The Server process has started.
  ACTION:
  None.
```

以下是命令服务对象启动的示例日志条目:

```
  02/15/2005 11:53:55 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5030: The Command 'C1' has started. ProcessId(13030).

  EXPLANATION:
  The Command has started.
  ACTION:
  None.
```

当实例服务器服务停止时，将向队列管理器错误日志写入一条消息，其中包含服务的名称和结束进程的进程标识。 以下是服务器服务对象停止的示例日志条目:

```
  02/15/2005 11:54:54 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5029: The Server 'S1' has ended. ProcessId(13031).

  EXPLANATION:
  The Server process has ended.
  ACTION:
  None.
```

相关参考

第 87 页的『其他环境变量』

启动某项服务时，服务进程的启动环境将继承自队列管理器的环境。 可以通过将要定义的变量添加到其中一个 service.env 环境覆盖文件，来定义要在服务进程的环境中设置的其他环境变量。

其他环境变量

启动某项服务时，服务进程的启动环境将继承自队列管理器的环境。 可以通过将要定义的变量添加到其中一个 service.env 环境覆盖文件，来定义要在服务进程的环境中设置的其他环境变量。

注:

有两个可能的文件可供您添加环境变量:

- 机器作用域 service.env 文件，位于 UNIX and Linux 系统上的 /var/mqm 中，或者位于 Windows 系统上安装期间选择的数据目录中。
- 队列管理器作用域 service.env 文件，位于队列管理器数据目录中。 例如，名为 QMNAME 的队列管理器的环境覆盖文件的位置为:
  - 在 UNIX and Linux 系统上， /var/mqm/qmgrs/QMNAME/service.env
  - 在 Windows 系统上， C:\Program Files\IBM\WebSphere MQ\qmgrs\QMNAME\service.env

将处理这两个文件 (如果可用)，队列管理器作用域文件中的定义优先于机器作用域文件中的定义。

可以在 service.env 中指定任何环境变量。 例如，如果 IBM WebSphere MQ 服务运行大量命令，那么在 service.env 文件中设置 PATH 用户变量可能很有用。 将变量设置为的值不能是环境变量; 例如，CLASSPATH=*%CLASSPATH%* 不正确。 同样，在 Linux PATH=*$PATH*:/opt/mqm/bin 上会给出意外的结果。

CLASSPATH 必须大写，并且类路径语句只能包含文字。 某些服务 (例如， Telemetry) 设置自己的类路径。 将 service.env 中所定义的 CLASSPATH 添加至此类路径。

文件 service.env 中定义的变量的格式是名称/值变量对的列表。 必须在新行上定义每个变量，并且将每个变量作为显式定义的变量 (包括空格)。 以下是文件 service.env 的示例:

```
  #********************************************************************#
  #*                                                                  *#
  #* <N_OCO_COPYRIGHT>                                                *#
  #* Licensed Materials - Property of IBM                             *#
  #*                                                                  *#
  #* 63H9336                                                          *#
  #* (C) Copyright IBM Corporation 2005, 2024.                        *#
  #*                                                                  *#
  #* <NOC_COPYRIGHT>                                                  *#
```

```
#*                                                                    *#
#**********************************************************************#
#**********************************************************************#
#* Module Name: service.env                                           *#
#* Type       : WebSphere MQ service environment file                 *#
#* Function   : Define additional environment variables to be set     *#
#*              for SERVICE programs.                                 *#
#* Usage      : <VARIABLE>=<VALUE>                                    *#
#*                                                                    *#
#**********************************************************************#
MYLOC=/opt/myloc/bin
MYTMP=/tmp
TRACEDIR=/tmp/trace
MYINITQ=ACCOUNTS.INITIATION.QUEUE
```

相关参考

第 88 页的『服务定义上的可替换插入』

在服务对象的定义中，可以替换令牌。 当执行服务程序时，被替换的标记将自动替换为其展开的文本。 可以从以下公共令牌列表中获取替代令牌，也可以从文件 service.env 中定义的任何变量中获取替代令牌。

服务定义上的可替换插入

在服务对象的定义中，可以替换令牌。 当执行服务程序时，被替换的标记将自动替换为其展开的文本。 可以从以下公共令牌列表中获取替代令牌，也可以从文件 service.env 中定义的任何变量中获取替代令牌。

以下是可用于替换服务对象定义中的令牌的公共令牌:

MQ_INSTALL_PATH
WebSphere MQ 的安装位置。

MQ_DATA_PATH
WebSphere MQ 数据目录的位置:

- 在 UNIX and Linux 系统上， WebSphere MQ 数据目录位置为 `/var/mqm/`
- 在 Windows 系统上， WebSphere MQ 数据目录的位置是在安装 WebSphere MQ 期间选择的数据目录

QMNAME
当前队列管理器名称。

MQ_SERVICE_NAME
服务名称。

MQ_SERVER_PID
此令牌只能由 STOPARG 和 STOPCMD 参数使用。

对于服务器服务对象，此标记将替换为由 STARTCMD 和 STARTARG 参数启动的进程的进程标识。 否则，此令牌将替换为 0。

MQ_Q_MGR_DATA_PATH
队列管理器数据目录的位置。

MQ_Q_MGR_DATA_NAME
队列管理器的变换名称。 有关名称变换的更多信息，请参阅 了解 WebSphere MQ 文件名。

要使用可替换插入，请将 + 字符内的标记插入到 STARTCMD， STARTARG， STOPCMD， STOPARG， STDOUT 或 STDERR 字符串中的任何字符串中。 有关此操作的示例，请参阅 第 88 页的『使用服务对象的示例』。

使用服务对象的示例

此部分中的服务使用 UNIX 样式路径分隔符进行编写，除非另有声明。

使用服务器服务对象

此示例显示如何定义，使用和变更服务器服务对象以启动触发器监视器。

1. 使用以下 MQSC 命令定义了服务器服务对象:

```
DEFINE SERVICE(S1) +
```

```
       CONTROL(QMGR) +
       SERVTYPE(SERVER) +
       STARTCMD('+MQ_INSTALL_PATH+bin/runmqtrm') +
       STARTARG('-m +QMNAME+ -q ACCOUNTS.INITIATION.QUEUE') +
       STOPCMD('+MQ_INSTALL_PATH+bin/amqsstop') +
       STOPARG('-m +QMNAME+ -p +MQ_SERVER_PID+')
```

其中：

+MQ_INSTALL_PATH+ 是表示安装目录的令牌。
+QMNAME+ 是表示队列管理器名称的令牌。
ACCOUNTS.INITIATION.QUEUE 是启动队列。
amqsstop 是随 WebSphere MQ 提供的样本程序，它请求队列管理器中断进程标识的所有连接。
amqsstop 生成 PCF 命令，因此命令服务器必须正在运行。
+MQ_SERVER_PID+ 是表示传递到停止程序的进程标识的标记。

请参阅 第 88 页的『服务定义上的可替换插入』 以获取公共令牌的列表。

2. 下次启动队列管理器时，将执行服务器服务对象的实例。 但是，我们将使用以下 MQSC 命令立即启动服务器服务对象的实例：

```
START SERVICE(S1)
```

3. 将使用以下 MQSC 命令显示服务器服务进程的状态：

```
DISPLAY SVSTATUS(S1)
```

4. 此示例现在显示如何更改服务器服务对象并通过手动重新启动服务器服务进程来获取更新。 将更改服务器服务对象，以便将启动队列指定为 JUPITER.INITIATION.QUEUE 。 使用以下 MQSC 命令：

```
ALTER SERVICE(S1) +
      STARTARG('-m +QMNAME+ -q JUPITER.INITIATION.QUEUE')
```

注: 正在运行的服务在重新启动之前不会获取对其服务定义的任何更新。

5. 将重新启动服务器服务进程，以便使用以下 MQSC 命令进行更改：

```
STOP SERVICE(S1)
```

接下来发出以下命令：

```
START SERVICE(S1)
```

服务器服务进程将重新启动，并选取在 第 89 页的『4』 中进行的更改。

注: 仅当在服务定义中指定了 STOPCMD 参数时，才能使用 MQSC 命令 STOP SERVICE。

使用命令服务对象

此示例显示如何定义命令服务对象以在队列管理器启动或停止时启动将条目写入操作系统的系统日志的程序。

1. 使用以下 MQSC 命令定义了命令服务对象：

```
DEFINE SERVICE(S2) +
       CONTROL(QMGR) +
       SERVTYPE(COMMAND) +
       STARTCMD('/usr/bin/logger') +
       STARTARG('Queue manager +QMNAME+ starting') +
       STOPCMD('/usr/bin/logger') +
       STOPARG('Queue manager +QMNAME+ stopping')
```

其中：

logger 是 UNIX and Linux 系统提供的用于写入系统日志的命令。
+QMNAME+ 是表示队列管理器名称的令牌。

在队列管理器仅结束时使用命令服务对象

此示例显示如何定义命令服务对象以在仅当队列管理器停止时启动将条目写入操作系统的系统日志的程序。

1. 使用以下 MQSC 命令定义了命令服务对象:

```
DEFINE SERVICE(S3) +
       CONTROL(QMGR) +
       SERVTYPE(COMMAND) +
       STOPCMD('/usr/bin/logger') +
       STOPARG('Queue manager +QMNAME+ stopping')
```

其中:

logger 是随 WebSphere MQ 提供的样本程序，可将条目写入操作系统的系统日志。
+QMNAME+ 是表示队列管理器名称的令牌。

有关传递参数的更多信息

此示例显示如何定义服务器服务对象以在启动队列管理器时启动名为 runserv 的程序。

此示例使用 Windows 样式路径分隔符进行编写。

要传递到起始程序的其中一个自变量是包含空格的字符串。 此自变量需要作为单个字符串传递。 要实现此目的，请使用双引号，如以下命令中所示来定义命令服务对象:

1. 使用以下 MQSC 命令定义了服务器服务对象:

```
   DEFINE SERVICE(S1) SERVTYPE(SERVER) CONTROL(QMGR) +
     STARTCMD('C:\Program Files\Tools\runserv.exe') +
     STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\"') +
     STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')


DEFINE SERVICE(S4) +
       CONTROL(QMGR) +
       SERVTYPE(SERVER) +
       STARTCMD('C:\Program Files\Tools\runserv.exe') +
       STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\"') +
       STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')
```

其中:

+QMNAME+ 是表示队列管理器名称的令牌。
"C:\Program Files\Tools\" 是包含空格的字符串，将作为单个字符串传递。

自动启动服务

此示例显示如何定义可用于在队列管理器启动时自动启动触发器监视器的服务器服务对象。

1. 使用以下 MQSC 命令定义了服务器服务对象:

```
DEFINE SERVICE(TRIG_MON_START) +
       CONTROL(QMGR) +
       SERVTYPE(SERVER) +
       STARTCMD('runmqtrm') +
       STARTARG('-m +QMNAME+ -q +IQNAME+')
```

其中:

+QMNAME+ 是表示队列管理器名称的令牌。
+IQNAME+ 是用户在表示启动队列名称的其中一个 service.env 文件中定义的环境变量。

管理用于触发的对象

WebSphere MQ 使您能够在满足队列上的特定条件时自动启动应用程序。 例如，当队列上的消息数达到指定的数目时，您可能想要启动应用程序。 此工具称为 触发。 您必须定义支持触发的对象。

使用触发器启动 WebSphere MQ 应用程序中详细描述了触发。

定义用于触发的应用程序队列

应用程序队列是应用程序通过 MQI 用于消息传递的本地队列。 触发需要在应用程序队列上定义多个队列属性。

触发本身由 *Trigger* 属性 (MQSC 命令中的 TRIGGER) 启用。 在此示例中，当本地队列 MOTOR.INSURANCE.QUEUE，如下所示:

```
DEFINE QLOCAL (MOTOR.INSURANCE.QUEUE) +
       PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
       MAXMSGL (2000) +
       DEFPSIST (YES) +
       INITQ (MOTOR.INS.INIT.QUEUE) +
       TRIGGER +
       TRIGTYPE (DEPTH) +
       TRIGDPTH (100)+
       TRIGMPRI (5)
```

其中:

QLOCAL (MOTOR.INSURANCE.QUEUE)
正在定义的应用程序队列的名称。

PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
定义要由触发器监视器程序启动的应用程序的进程定义的名称。

MAXMSGL (2000)
队列中消息的最大长度。

DEFPSIST (YES)
指定缺省情况下此队列上的消息是持久的。

INITQ (MOTOR.INS.INIT.QUEUE)
队列管理器要在其上放置触发器消息的启动队列的名称。

TRIGGER
是触发器属性值。

TRIGTYPE (DEPTH)
指定当所需优先级 (TRIGMPRI) 的消息数达到 TRIGDPTH 中指定的数目时，将生成触发器事件。

TRIGDPTH (100)
生成触发器事件所需的消息数。

TRIGMPRI (5)
队列管理器在决定是否生成触发器事件时要计算的消息的优先级。 仅计算优先级为 5 或更高的消息。

定义启动队列

发生触发器事件时，队列管理器会将触发器消息放在应用程序队列定义中指定的启动队列上。 启动队列没有特殊设置，但您可以使用本地队列 MOTOR.INS.INIT.QUEUE ，用于指导:

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
       GET (ENABLED) +
       NOSHARE +
       NOTRIGGER +
       MAXMSGL (2000) +
       MAXDEPTH (1000)
```

定义流程

使用 DEFINE PROCESS 命令来创建进程定义。 进程定义定义要用于处理来自应用程序队列的消息的应用程序。 应用程序队列定义对要使用的进程进行命名，从而使应用程序队列与要用于处理其消息的应用程序相关联。 这是通过应用程序队列 MOTOR.INSURANCE.QUEUE。 以下 MQSC 命令定义必需的进程 MOTOR.INSURANCE.QUOTE.PROCESS，在此示例中标识:

```
DEFINE PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
               DESCR ('Insurance request message processing') +
               APPLTYPE (UNIX) +
               APPLICID ('/u/admin/test/IRMP01') +
               USERDATA ('open, close, 235')
```

其中:

MOTOR.INSURANCE.QUOTE.PROCESS
是进程定义的名称。

DESCR ('Insurance request message processing')
描述与此定义相关的应用程序。 此文本在您使用 DISPLAY PROCESS 命令时显示。 这可以帮助您确定流程的作用。 如果在字符串中使用空格，那么必须用单引号将字符串括起来。

APPLTYPE (UNIX)
要启动的应用程序的类型。

APPLICID ('/u/admin/test/IRMP01')
应用程序可执行文件的名称，指定为标准文件名。 在 Windows 系统中，典型的 APPLICID 值将为 `c:\appl\test\irmp01.exe`。

USERDATA ('open, close, 235')
是用户定义的数据，可供应用程序使用。

显示进程定义的属性

使用 DISPLAY PROCESS 命令来检查定义的结果。 例如:

```
DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)


    24 : DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL
AMQ8407: Display Process details.
    DESCR ('Insurance request message processing')
    APPLICID ('/u/admin/test/IRMP01')
    USERDATA (open, close, 235)
    PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
    APPLTYPE (UNIX)
```

您还可以使用 MQSC 命令 ALTER PROCESS 来变更现有进程定义，使用 DELETE PROCESS 命令来删除进程定义。

管理远程 IBM WebSphere MQ 对象

本节说明如何使用 MQSC 命令来管理远程队列管理器上的 IBM WebSphere MQ 对象，以及如何使用远程队列对象来控制消息和应答消息的目标。

本节描述:

- 第 93 页的『通道，集群和远程排队』
- 第 94 页的『从本地队列管理器进行远程管理』
- 第 99 页的『创建远程队列的本地定义』
- 第 101 页的『使用远程队列定义作为别名』
- 第 101 页的『编码字符集之间的数据转换』

通道，集群和远程排队

队列管理器通过发送消息与另一个队列管理器通信，并在需要时接收回响应。 接收队列管理器可以是:

- 在同一机器上
- 在同一位置的另一台机器上 (甚至在世界的另一边)
- 在与本地队列管理器相同的平台上运行
- 在 WebSphere MQ 支持的另一平台上运行

这些消息可能源自:

- 用户编写的应用程序，用于将数据从一个节点传输到另一个节点
- 使用 PCF 命令或 MQAI 的用户编写的管理应用程序
- IBM WebSphere MQ Explorer。
- 队列管理器正在发送:
  - 检测事件消息到另一个队列管理器
  - 从 `runmqsc` 命令以间接方式发出的 MQSC 命令 (其中命令在另一个队列管理器上运行)

在将消息发送到远程队列管理器之前，本地队列管理器需要一种机制来检测消息的到达并传输这些消息，包括:

- 至少一个通道
- 传输队列
- 通道启动程序

要使远程队列管理器接收消息，需要侦听器。

通道是两个队列管理器之间的单向通信链路，可以将消息发送到远程队列管理器上的任意数目的队列。

通道的每一端都有单独的定义。 例如，如果一端是发送方或服务器，那么另一端必须是接收方或请求者。 简单通道由本地队列管理器端的 发送方通道定义 和远程队列管理器端的 接收方通道定义 组成。 这两个定义必须具有相同的名称，并且共同构成单个消息通道。

如果您希望远程队列管理器响应本地队列管理器发送的消息，请设置第二个通道以将响应发送回本地队列管理器。

使用 MQSC 命令 DEFINE CHANNEL 来定义通道。 在此部分中，除非另有指定，否则与通道相关的示例将使用缺省通道属性。

在通道的每一端都有一个消息通道代理 (MCA) ，用于控制消息的发送和接收。 MCA 从传输队列中获取消息，并将它们放在队列管理器之间的通信链路上。

传输队列是专门的本地队列，它在 MCA 选取消息并将其发送到远程队列管理器之前临时保存这些消息。 在远程队列定义上指定传输队列的名称。

您可以允许 MCA 使用多个线程来传输消息。 此过程称为 流水线。 Pipelining 使 MCA 能够更高效地传输消息，从而提高通道性能。 有关如何配置通道以使用管道的详细信息，请参阅 通道属性 。

第 95 页的『为远程管理准备通道和传输队列』 告诉您如何使用这些定义来设置远程管理。

有关通常设置分布式排队的更多信息，请参阅 分布式排队组件。

使用集群的远程管理

在使用分布式排队的 WebSphere MQ 网络中，每个队列管理器都是独立的。 如果一个队列管理器需要将消息发送到另一个队列管理器，那么它必须为要向其发送消息的每个队列定义传输队列，到远程队列管理器的通道以及远程队列定义。

集群 是一组队列管理器，其设置方式使队列管理器可以通过单个网络直接相互通信，而无需复杂的传输队列，通道和队列定义。 可以轻松设置集群，并且通常包含以某种方式逻辑相关且需要共享数据或应用程序的队列管理器。 即使是最小的集群也会降低系统管理成本。

在集群中建立队列管理器网络涉及的定义少于建立传统分布式排队环境。 通过更少的定义，您可以更快速轻松地设置或更改网络，并降低在定义中发生错误的风险。

要设置集群，每个队列管理器需要一个集群发送方 (CLUSSDR) 和一个集群接收方 (CLUSRCVR) 定义。 您不需要任何传输队列定义或远程队列定义。 在集群中使用时，远程管理的原则相同，但定义本身已大大简化。

有关集群及其属性以及如何设置这些集群的更多信息，请参阅 队列管理器集群。

从本地队列管理器进行远程管理

本节说明如何使用 MQSC 和 PCF 命令从本地队列管理器管理远程队列管理器。

对于 MQSC 和 PCF 命令，准备队列和通道基本上是相同的。 在此部分中，示例显示 MQSC 命令，因为它们更易于理解。 有关使用 PCF 命令编写管理程序的更多信息，请参阅 第 9 页的『使用可编程命令格式』。

将 MQSC 命令以交互方式或从包含这些命令的文本文件发送到远程队列管理器。 远程队列管理器可能位于同一台机器上，或者更通常位于另一台机器上。 您可以在其他 WebSphere MQ 环境 (包括 UNIX and Linux 系统， Windows 系统， IBM i 和 z/OS) 中远程管理队列管理器。

要实现远程管理，必须创建特定对象。 除非您有专门的需求，否则缺省值 (例如，最大消息长度) 已足够。

准备队列管理器以进行远程管理

如何使用 MQSC 命令来准备队列管理器以进行远程管理。

第 94 页的图 17 显示了使用 **runmqsc** 命令进行远程管理所需的队列管理器和通道的配置。 对象 `source.queue.manager` 是可以从中发出 MQSC 命令的源队列管理器，这些命令 (操作员消息) 的结果将返回到该源队列管理器。 对象 `target.queue.manager` 是目标队列管理器的名称，用于处理命令并生成任何操作员消息。

注: 如果要将 **runmqsc** 与 -w 选项配合使用，那么 `source.queue.manager` **必须** 是缺省队列管理器。 有关创建队列管理器的更多信息，请参阅 crtmqm。

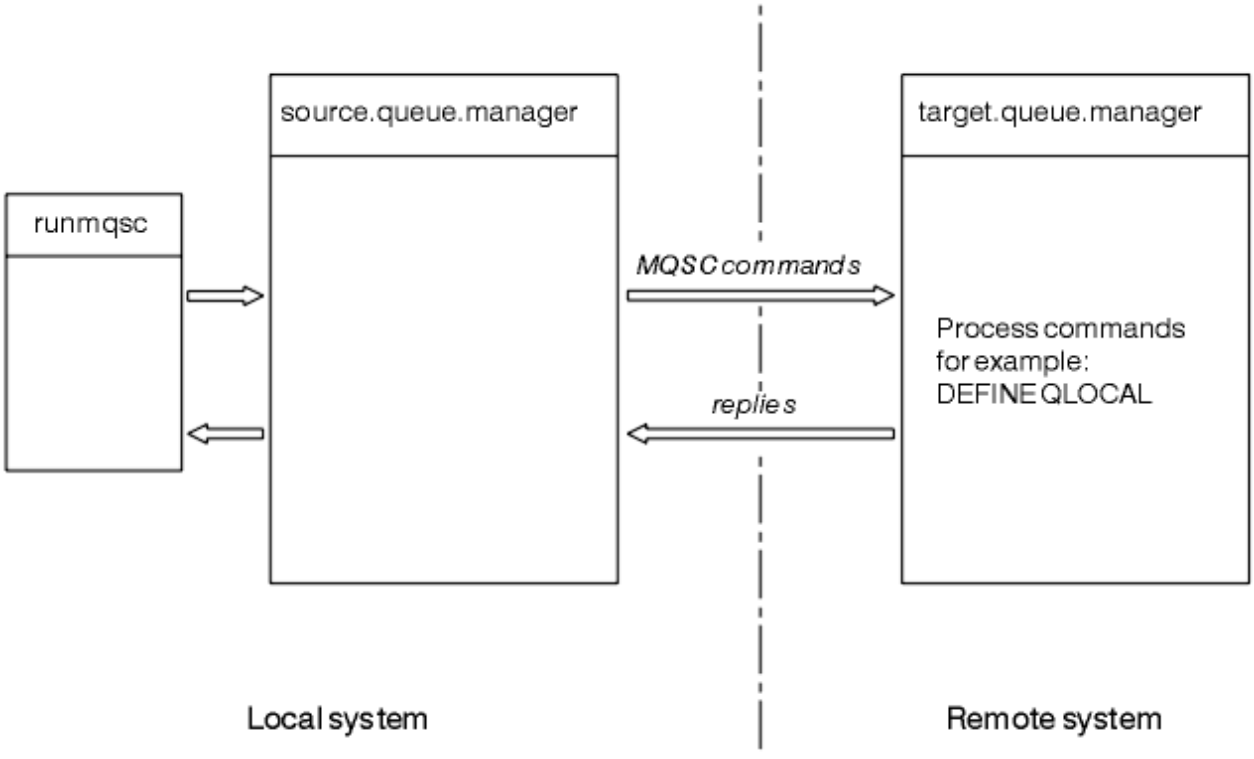


图 *17:* 使用 *MQSC* 命令进行远程管理

在这两个系统上，如果尚未执行此操作:

- 使用 `crtmqm` 命令创建队列管理器和缺省对象。

- 使用 strmqm 命令启动队列管理器。

在目标队列管理器上:

- 命令队列 SYSTEM.ADMIN.COMMAND.QUEUE 必须存在。 缺省情况下，将在创建队列管理器时创建此队列。

您必须在本地或通过 Telnet 之类的网络设施运行这些命令。

为远程管理准备通道和传输队列

如何使用 MQSC 命令来准备用于远程管理的通道和传输队列。

要远程运行 MQSC 命令，请设置两个通道 (每个方向一个通道) 及其关联的传输队列。 此示例假定您使用 TCP/IP 作为传输类型，并且您知道所涉及的 TCP/IP 地址。

通道 source.to.target 用于将 MQSC 命令从源队列管理器发送到目标队列管理器。 其发送方位于 source.queue.manager，其接收方位于 target.queue.manager。 通道 target.to.source 用于返回来自命令的输出以及生成到源队列管理器的任何操作员消息。 您还必须为每个通道定义一个传输队列。 此队列是给定了接收队列管理器的名称的本地队列。 除非使用队列管理器别名，否则 XMITQ 名称必须与远程队列管理器名称匹配才能使远程管理工作。 第 95 页的图 18 汇总此配置。

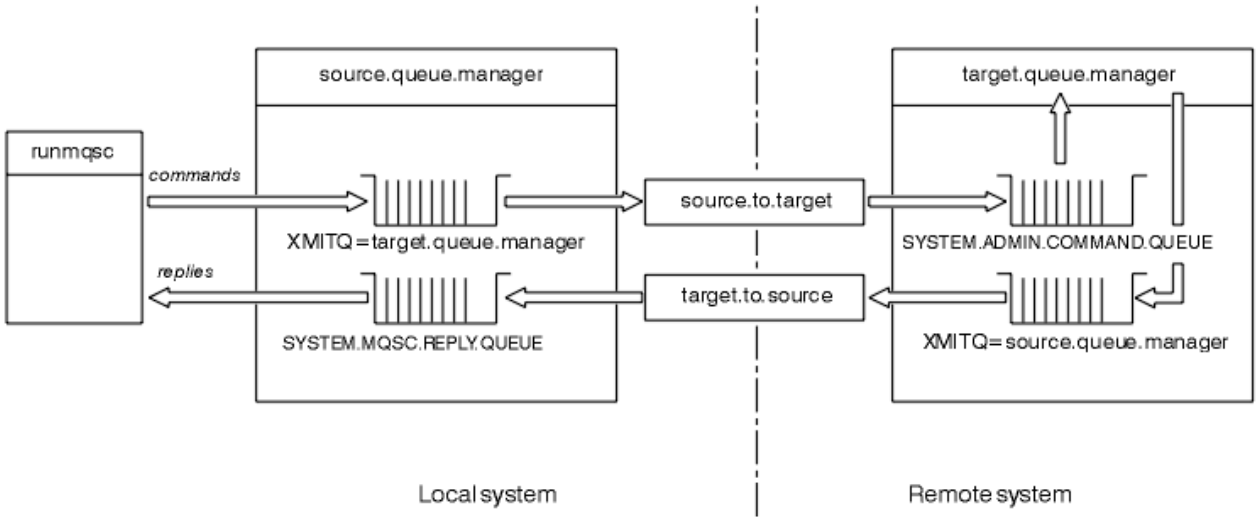


图 *18:* 设置用于远程管理的通道和队列

有关设置通道的更多信息，请参阅 使用分布式排队连接应用程序 。

定义通道，侦听器和传输队列

在源队列管理器 (source.queue.manager) 上，发出以下 MQSC 命令以定义通道，侦听器和传输队列:

1. 在源队列管理器上定义发送方通道:

```
DEFINE CHANNEL ('source.to.target') +
       CHLTYPE(SDR) +
       CONNAME (RHX5498) +
       XMITQ ('target.queue.manager') +
       TRPTYPE(TCP)
```

2. 在源队列管理器上定义接收方通道:

```
DEFINE CHANNEL ('target.to.source') +
       CHLTYPE(RCVR) +
       TRPTYPE(TCP)
```

3. 在源队列管理器上定义侦听器:

```
DEFINE LISTENER ('source.queue.manager') +
       TRPTYPE (TCP)
```

4. 在源队列管理器上定义传输队列:

```
DEFINE QLOCAL ('target.queue.manager') +
       USAGE (XMITQ)
```

在目标队列管理器 (target.queue.manager) 上发出以下命令以创建通道，侦听器和传输队列:

1. 在目标队列管理器上定义发送方通道:

```
DEFINE CHANNEL ('target.to.source') +
       CHLTYPE(SDR) +
       CONNAME (RHX7721) +
       XMITQ ('source.queue.manager') +
       TRPTYPE(TCP)
```

2. 在目标队列管理器上定义接收方通道:

```
DEFINE CHANNEL ('source.to.target') +
       CHLTYPE(RCVR) +
       TRPTYPE(TCP)
```

3. 在目标队列管理器上定义侦听器:

```
DEFINE LISTENER ('target.queue.manager') +
       TRPTYPE (TCP)
```

4. 在目标队列管理器上定义传输队列:

```
DEFINE QLOCAL ('source.queue.manager') +
       USAGE (XMITQ)
```

注: 在发送方通道定义中为 CONNAME 属性指定的 TCP/IP 连接名称仅用于说明。 这是连接的 其他 端的机器的网络名。 使用适合于您的网络的值。

启动侦听器和通道

如何使用 MQSC 命令来启动侦听器和通道。

使用以下 MQSC 命令启动这两个侦听器:

1. 通过发出以下 MQSC 命令在源队列管理器 source.queue.manager 上启动侦听器:

```
START LISTENER ('source.queue.manager')
```

2. 通过发出以下 MQSC 命令在目标队列管理器 target.queue.manager 上启动侦听器:

```
START LISTENER ('target.queue.manager')
```

使用以下 MQSC 命令启动两个发送方通道:

1. 通过发出以下 MQSC 命令在源队列管理器 source.queue.manager 上启动发送方通道:

```
START CHANNEL ('source.to.target')
```

2. 通过发出以下 MQSC 命令在目标队列管理器 target.queue.manager 上启动发送方通道:

```
START CHANNEL ('target.to.source')
```

通道的自动定义
通过使用 MQSC 命令， ALTER QMGR (或 PCF 命令 "更改队列管理器") 更新队列管理器对象来启用接收方和服务器连接定义的自动定义。

如果 WebSphere MQ 接收到入站连接请求，但找不到相应的接收方或服务器连接通道，那么它会自动创建通道。 自动定义基于随 WebSphere MQ 提供的两个缺省定义: SYSTEM.AUTO.RECEIVER 和 SYSTEM.AUTO.SVRCONN。

有关自动创建通道定义的更多信息，请参阅 准备通道。 有关自动定义集群通道的信息，请参阅 自动定义集群通道。

管理命令服务器以进行远程管理

如何启动，停止和显示命令服务器的状态。 对于涉及 PCF 命令， MQAI 以及远程管理的所有管理，都必须使用命令服务器。

每个队列管理器都可以有一个与其关联的命令服务器。 命令服务器处理来自远程队列管理器的任何入局命令或来自应用程序的 PCF 命令。 它向队列管理器提供用于处理的命令，并根据命令的来源返回完成代码或操作员消息。

注: 对于远程管理，请确保目标队列管理器正在运行。 否则，包含命令的消息无法离开从中发出这些命令的队列管理器。 而是在为远程队列管理器提供服务的本地传输队列中对这些消息进行排队。 避免这种情况。

有单独的控制命令用于启动和停止命令服务器。 如果命令服务器正在运行，那么 WebSphere MQ for Windows 或 WebSphere MQ for Linux (x86 和 x86-64 平台) 的用户可以使用 IBM WebSphere MQ Explorer 执行以下部分中描述的操作。 有关更多信息，请参阅 第 51 页的『使用 IBM WebSphere MQ Explorer 进行管理』。

启动命令服务器

根据队列管理器属性 *SCMDSERV* 的值，命令服务器将在队列管理器启动时自动启动，或者必须手动启动。 可以使用 MQSC 命令 ALTER QMGR 指定参数 SCMDSERV 来更改队列管理器属性的值。 缺省情况下，将自动启动命令服务器。

如果 *SCMDSERV* 设置为 MANUAL ，请使用以下命令启动命令服务器:

```
strmqcsv saturn.queue.manager
```

其中， saturn.queue.manager 是要对其启动命令服务器的队列管理器。

显示命令服务器的状态

对于远程管理，请确保目标队列管理器上的命令服务器正在运行。 如果它未在运行，那么无法处理远程命令。 包含命令的任何消息都将在目标队列管理器的命令队列中排队。

要显示队列管理器的命令服务器的状态，请发出以下 MQSC 命令:

```
DISPLAY QMSTATUS CMDSERV
```

停止命令服务器

要结束上一个示例启动的命令服务器，请使用以下命令:

```
endmqcsv saturn.queue.manager
```

可以通过两种方式停止命令服务器:

- 对于受控停止，请使用带有 -c 标志的 endmqcsv 命令，这是缺省值。
- 对于立即停止，请使用带有 -i 标志的 endmqcsv 命令。

注: 停止队列管理器还会结束与其关联的命令服务器。

在远程队列管理器上发出 MQSC 命令

您可以使用特定形式的 **runmqsc** 命令在远程队列管理器上运行 MQSC 命令。

如果要远程处理 MQSC 命令，那么命令服务器 **必须** 正在目标队列管理器上运行。 (这在源队列管理器上不是必需的)。 有关如何在队列管理器上启动命令服务器的信息，请参阅 第 97 页的『管理命令服务器以进行远程管理』。

然后，在源队列管理器上，可以通过输入以下命令以间接方式以交互方式运行 MQSC 命令:

```
runmqsc -w 30 target.queue.manager
```

此格式的 runmqsc 命令 (带有 -w 标志) 以间接方式运行 MQSC 命令，其中命令将 (以修改的形式) 放在命令服务器输入队列上并按顺序执行。

输入 MQSC 命令时，会将其重定向到远程队列管理器，在本例中为 target.queue.manager。 超时设置为 30 秒; 如果在 30 秒内未收到应答，那么将在本地 (源) 队列管理器上生成以下消息:

```
AMQ8416: MQSC timed out waiting for a response from the command server.
```

当您停止发出 MQSC 命令时，本地队列管理器将显示已到达的任何超时响应，并废弃任何进一步的响应。

源队列管理器缺省为缺省本地队列管理器。 如果在 **runmqsc** 命令中指定 -m *LocalQmgrName* 选项，那么可以指示要通过任何本地队列管理器发出的命令。

在间接方式下，您还可以在远程队列管理器上运行 MQSC 命令文件。 例如:

```
runmqsc -w 60 target.queue.manager < mycomds.in > report.out
```

其中 mycomds.in 是包含 MQSC 命令的文件， report.out 是报告文件。

远程发出命令的建议方法

在远程队列管理器上发出命令时，请考虑使用以下方法:

1. 将要在远程系统上运行的 MQSC 命令放在命令文件中。
2. 通过在 runmqsc 命令上指定 -v 标志，在本地验证 MQSC 命令。

   不能使用 runmqsc 在另一个队列管理器上验证 MQSC 命令。
3. 检查命令文件是否在本地运行而没有错误。
4. 在远程系统上运行命令文件。

如果您在远程使用 MQSC 命令时迂到问题

如果您在远程运行 MQSC 命令时迂到困难，请确保您具有:

- 已在目标队列管理器上启动命令服务器。
- 定义了有效的传输队列。
- 定义了两个消息通道的两端:
  - 用于发送命令的通道。
  - 要返回应答的通道。
- 在通道定义中指定了正确的连接名称 (CONNAME)。
- 在启动消息通道之前，已启动侦听器。
- 检查断开连接时间间隔是否未到期，例如，如果通道已启动但在一段时间后关闭。 如果手动启动通道，那么这尤其重要。
- 从源队列管理器发送对目标队列管理器没有意义的请求 (例如，包含在远程队列管理器上不受支持的参数的请求)。

另请参阅第 71 页的『解决 MQSC 命令的问题』。

创建远程队列的本地定义

远程队列的本地定义是本地队列管理器上引用远程队列管理器上的队列的定义。

您不必从本地位置定义远程队列，但这样做的优点是应用程序可以通过其本地定义的名称来引用远程队列，而不必指定由远程队列所在队列管理器的标识限定的名称。

了解远程队列的本地定义如何工作

应用程序连接到本地队列管理器，然后发出 MQOPEN 调用。 在打开的调用中，指定的队列名称是本地队列管理器上的远程队列定义的名称。 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。 要将消息放在远程队列上，应用程序会发出 MQPUT 调用，并指定从 MQOPEN 调用返回的句柄。 队列管理器在消息开头的传输头中使用远程队列名称和远程队列管理器名称。 此信息用于将消息路由到其在网络中的正确目标。

作为管理员，您可以通过改变远程队列定义来控制消息的目标。

以下示例显示应用程序如何将消息放入由远程队列管理器拥有的队列中。 应用程序连接到队列管理器，例如 saturn.queue.manager。 目标队列由另一个队列管理器拥有。

在 MQOPEN 调用上，应用程序指定以下字段:

字段值	描述
ObjectName CYAN.REMOTE.QUEUE	指定远程队列对象的局部名。 这将定义目标队列和目标队列管理器。
ObjectType (队列)	将此对象标识为队列。
ObjectQmgrName 空白或 saturn.queue.manager	此字段是可选字段。 如果为空，那么将采用本地队列管理器的名称。 (这是远程队列定义所在的队列管理器。)

在此之后，应用程序发出 MQPUT 调用以将消息放入此队列。

在本地队列管理器上，可以使用以下 MQSC 命令创建远程队列的本地定义:

```
DEFINE QREMOTE (CYAN.REMOTE.QUEUE) +
       DESCR ('Queue for auto insurance requests from the branches') +
       RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE) +
       RQMNAME (jupiter.queue.manager) +
       XMITQ (INQUOTE.XMIT.QUEUE)
```

其中:

QREMOTE (CYAN.REMOTE.QUEUE)
指定远程队列对象的局部名。 这是连接到此队列管理器的应用程序必须在 MQOPEN 调用中指定的名称，以打开远程队列管理器 jupiter.queue.manager 上的队列 AUTOMOBILE.INSURANCE.QUOTE.QUEUE 。

DESCR ('Queue for auto insurance requests from the branches')
提供用于描述队列使用情况的其他文本。

RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)
指定远程队列管理器上的目标队列的名称。 这是由指定队列名称 CYAN.REMOTE.QUEUE。 队列 AUTOMOBILE.INSURANCE.QUOTE.QUEUE 必须定义为远程队列管理器上的本地队列。

RQMNAME (jupiter.queue.manager)
指定拥有目标队列 AUTOMOBILE.INSURANCE.QUOTE.QUEUE。

XMITQ (INQUOTE.XMIT.QUEUE)
指定传输队列的名称。 这是可选的; 如果未指定传输队列的名称，那么将使用与远程队列管理器同名的队列。

在任一情况下，都必须将相应的传输队列定义为具有 *Usage* 属性的本地队列，该属性指定它是 MQSC 命令中的传输队列 (USAGE (XMITQ))。

将消息放入远程队列的替代方法

使用远程队列的本地定义不是将消息放入远程队列的唯一方法。 应用程序可以在 MQOPEN 调用中指定完整队列名称 (包括远程队列管理器名称)。 在这种情况下，您不需要远程队列的本地定义。 但是，这意味着应用程序必须在运行时知道或有权访问远程队列管理器的名称。

将其他命令与远程队列配合使用

您可以使用 MQSC 命令来显示或更改远程队列对象的属性，也可以删除远程队列对象。 例如:

- 要显示远程队列的属性:

```
DISPLAY QUEUE (CYAN.REMOTE.QUEUE)
```

- 更改远程队列以启用放置。 这不会影响目标队列，仅影响指定此远程队列的应用程序:

```
ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)
```

- 删除此远程队列。 这不会影响目标队列，仅影响其本地定义:

```
  DELETE QREMOTE (CYAN.REMOTE.QUEUE)
```

注: 删除远程队列时，仅删除远程队列的本地表示。 您不会删除远程队列本身或其上的任何消息。

定义传输队列

传输队列是当队列管理器通过消息通道将消息转发到远程队列管理器时使用的本地队列。

通道提供到远程队列管理器的单向链接。 消息将在传输队列中排队，直到通道可以接受这些消息为止。 定义通道时，必须在消息通道的发送端指定传输队列名称。

MQSC 命令属性 USAGE 定义队列是传输队列还是普通队列。

缺省传输队列

当队列管理器向远程队列管理器发送消息时，它将使用以下顺序来标识传输队列:

1. 在远程队列的本地定义的 XMITQ 属性上指定的传输队列。
2. 与目标队列管理器同名的传输队列。 (此值是远程队列的本地定义的 XMITQ 上的缺省值。)
3. 在本地队列管理器的 DEFXMITQ 属性上指定的传输队列。

例如，以下 MQSC 命令在 `source.queue.manager` 上为转至 `target.queue.manager` 的消息创建缺省传输队列:

```
DEFINE QLOCAL ('target.queue.manager') +
       DESCR ('Default transmission queue for target qm') +
       USAGE (XMITQ)
```

应用程序可以将消息直接放置在传输队列上，也可以通过远程队列定义间接放置。 另请参阅第 99 页的『创建远程队列的本地定义』。

使用远程队列定义作为别名

除了在另一个队列管理器上查找队列外，还可以将远程队列的本地定义用于队列管理器别名和应答队列别名。 这两种类型的别名都通过远程队列的本地定义进行解析。 您必须设置相应的通道以使消息到达其目标。

队列管理器别名

别名是一个进程，消息中指定的目标队列管理器的名称由消息路由上的队列管理器修改。队列管理器别名很重要，因为您可以使用它们来控制队列管理器网络中的消息目标。

您可以通过在控制点更改队列管理器上的远程队列定义来执行此操作。 发送应用程序不知道指定的队列管理器名称是别名。

有关队列管理器别名的更多信息，请参阅 什么是别名?。

应答队列别名

(可选) 应用程序可以在将 请求消息 放入队列时指定应答队列的名称。

如果处理消息的应用程序抽取了应答队列的名称，那么它知道在哪里发送 应答消息(如果需要)。

应答队列别名是由消息路由上的队列管理器改变请求消息中指定的应答队列的过程。 发送应用程序不知道指定的应答队列名称是别名。

应答队列别名允许您更改应答队列的名称及其队列管理器 (可选)。 这反过来使您能够控制用于应答消息的路由。

有关请求消息，应答消息和应答队列的更多信息，请参阅 消息类型 和 应答队列和队列管理器。

有关应答队列别名的更多信息，请参阅 应答队列别名和集群 。

编码字符集之间的数据转换

队列管理器可以将 WebSphere MQ 定义的格式 (也称为 内置格式) 中的消息数据从一个编码字符集转换为另一个编码字符集，前提是这两个字符集都与单个语言或一组类似语言相关。

例如，支持在具有标识 (CCSID) 850 和 500 的编码字符集之间进行转换，因为两者都适用于西欧语言。

对于 EBCDIC 换行符 (NL) 字符到 ASCII 的转换，请参阅 所有队列管理器 。

受支持的转换在 数据转换中定义。

当队列管理器无法转换内置格式的消息时

如果消息的 CCSID 表示不同的本地语言组，那么队列管理器无法自动转换内置格式的消息。 例如，不支持 CCSID 850 与 CCSID 1025 (这是使用西里尔文脚本的语言的 EBCDIC 编码字符集) 之间的转换，因为一个编码字符集中的许多字符不能在另一个编码字符集中表示。 如果您具有以不同本地语言工作的队列管理器网络，并且不支持在某些编码字符集之间进行数据转换，那么可以启用缺省转换。 第 102 页的『缺省数据转换』中描述了缺省数据转换。

文件 ccsid.tbl

文件 ccsid.tbl 用于以下目的:

- 在 WebSphere MQ for Windows 中，它会记录所有受支持的代码集。
- 在 AIX 和 HP-UX 平台上，受支持的代码集由操作系统在内部保存。
- 对于所有其他 UNIX and Linux 平台，受支持的代码集保存在 WebSphere MQ 提供的转换表中。
- 它指定任何其他代码集。 要指定其他代码集，您需要编辑 ccsid.tbl (在文件中提供了有关如何执行此操作的指导)。
- 它指定任何缺省数据转换。

您可以更新 ccsid.tbl 中记录的信息; 例如，如果操作系统的未来发行版支持其他编码字符集，那么可能需要执行此操作。

在 WebSphere MQ for Windows 中，缺省情况下 ccsid.tbl 位于目录 `C:\Program Files\IBM\WebSphere MQ\conv\table` 中。

在 WebSphere MQ for UNIX and Linux 系统中， ccsid.tbl 位于目录 `/var/mqm/conv/table` 中。

缺省数据转换

如果在通常不支持数据转换的两台机器之间设置通道，那么必须启用缺省数据转换才能使通道正常工作。

要启用缺省数据转换，请编辑 ccsid.tbl 文件以指定缺省 EBCDIC CCSID 和缺省 ASCII CCSID。 有关如何执行此操作的指示信息包含在文件中。 必须在将使用通道连接的所有机器上执行此操作。 重新启动队列管理器以使更改生效。

缺省数据转换过程如下所示:

- 如果源 CCSID 和目标 CCSID 之间的转换不受支持，但源环境和目标环境的 CCSID 都是 EBCDIC 或 ASCII，那么字符数据将传递到目标应用程序而不进行转换。
- 如果一个 CCSID 表示 ASCII 编码字符集，另一个表示 EBCDIC 编码字符集，那么 WebSphere MQ 将使用 ccsid.tbl 中定义的缺省数据转换 CCSID 来转换数据。

注: 尝试将要转换的字符限制为那些在为消息指定的编码字符集和缺省编码字符集中具有相同代码值的字符。 如果仅使用对 WebSphere MQ 对象名有效的字符集 (如 命名 IBM WebSphere MQ 对象中所定义)，那么通常将满足此需求。 在日本使用 EBCDIC CCSID 290， 930， 1279 和 5026 时发生异常，其中小写字符与其他 EBCDIC CCSID 中使用的代码不同。

以用户定义的格式转换消息

队列管理器无法将用户定义格式的消息从一个编码字符集转换为另一个编码字符集。 如果需要以用户定义的格式转换数据，那么必须为每种此类格式提供数据转换出口。 请勿使用缺省 CCSID 来转换用户定义格式的字符数据。 有关以用户定义格式转换数据和写入数据转换出口的更多信息，请参阅 编写数据转换出口。

更改队列管理器 CCSID

当您使用 ALTER QMGR 命令的 CCSID 属性来更改队列管理器的 CCSID 时，请停止并重新启动队列管理器，以确保停止并重新启动所有正在运行的应用程序 (包括命令服务器和通道程序)。

这是必需的，因为更改队列管理器 CCSID 时正在运行的任何应用程序都将继续使用现有 CCSID。

管理 IBM WebSphere MQ Telemetry

IBM WebSphere MQ Telemetry 使用 IBM WebSphere MQ Explorer 或在命令行上进行管理。 使用资源管理器来配置遥测通道，控制遥测服务以及监视连接到 IBM WebSphere MQ 的 MQTT 客户机。 使用 JAAS, SSL 和 IBM WebSphere MQ 对象权限管理器来配置 IBM WebSphere MQ Telemetry 的安全性。

使用 IBM WebSphere MQ Explorer 进行管理

使用资源管理器来配置遥测通道，控制遥测服务以及监视连接到 IBM WebSphere MQ 的 MQTT 客户机。 使用 JAAS， SSL 和 IBM WebSphere MQ 对象权限管理器来配置 IBM WebSphere MQ Telemetry 的安全性。

使用命令行进行管理

可以使用 IBM WebSphere MQ MQSC 命令在命令行上完全管理 IBM WebSphere MQ Telemetry 。

IBM WebSphere MQ Telemetry 文档还包含样本脚本，用于演示 MQ Telemetry Transport v3 Client 应用程序的基本用法。

在使用 IBM WebSphere MQ Telemetry 样本程序 的 为 IBM WebSphere MQ Telemetry 开发应用程序 部分中的样本之前，请先阅读并了解这些样本。

相关概念

WebSphere MQ Telemetry

第 106 页的『配置分布式队列以将消息发送至 MQTT 客户机』
IBM WebSphere MQ 应用程序可以通过发布到客户机创建的预订或通过直接发送消息来发送 MQTT v3 客户机消息。 无论使用哪种方法，都将消息放在 SYSTEM.MQTT.TRANSMIT.QUEUE 上，并通过遥测 (MQXR) 服务发送到客户机。 可以通过多种方法在 SYSTEM.MQTT.TRANSMIT.QUEUE 上放置消息。

第 108 页的『MQTT 客户机标识、授权和认证』
第 114 页的『使用 SSL 进行遥测通道认证』
第 116 页的『发布在遥测通道上的隐私』
第 116 页的『MQTT 客户机和遥测通道的 SSL 配置』
第 120 页的『遥测通道 JAAS 配置』
配置 JAAS 以认证由客户机发送的**用户名**。

第 122 页的『设备的 IBM WebSphere MQ Telemetry 守护程序概念』
IBM WebSphere MQ 设备的遥测守护程序是高级 MQTT V3 客户机应用程序。 使用它来存储和转发来自其他 MQTT 客户机的消息。 它像 MQTT 客户机一样连接至 IBM WebSphere MQ，但是您还可以将它连接至其他 MQTT 客户机。

相关任务
第 103 页的『在 Linux 和 AIX 上配置队列管理器以进行遥测』
遵循以下手动步骤来配置队列管理器以运行 IBM WebSphere MQ Telemetry。 您可以运行自动化过程，以使用 IBM WebSphere MQ Telemetry 对 IBM WebSphere MQ Explorer 的支持来设置更简单的配置。

第 105 页的『在 Windows 上配置队列管理器以进行遥测』
遵循以下手动步骤来配置队列管理器以运行 IBM WebSphere MQ Telemetry。 您可以运行自动化过程，以使用 IBM WebSphere MQ Telemetry 对 IBM WebSphere MQ Explorer 的支持来设置更简单的配置。

相关参考
MQXR 属性

在 Linux 和 AIX 上配置队列管理器以进行遥测

遵循以下手动步骤来配置队列管理器以运行 IBM WebSphere MQ Telemetry。 您可以运行自动化过程，以使用 IBM WebSphere MQ Telemetry 对 IBM WebSphere MQ Explorer 的支持来设置更简单的配置。

开始之前

1. 有关如何安装 IBM WebSphere MQ 和 IBM WebSphere MQ Telemetry 功能部件的信息，请参阅 安装 IBM WebSphere MQ Telemetry 。
2. 创建并启动队列管理器。 在此任务中，队列管理器称为 *qMgr* 。
3. 在此任务中，您将配置遥测 (MQXR) 服务。 MQXR 属性设置存储在特定于平台的属性文件中: mqxr_unix.properties。 通常不需要直接编辑 MQXR 属性文件，因为几乎所有设置都可以通过 MQSC 管理命令或 MQ Explorer 进行配置。 如果确实决定直接编辑该文件，请先停止队列管理器，然后再进行更改。 请参阅 MQXR 属性。

关于此任务

IBM WebSphere MQ Explorer 的 IBM WebSphere MQ Telemetry 支持包含向导和样本命令过程 sampleMQM。 他们使用访客用户标识设置初始配置; 请参阅 使用 IBM WebSphere MQ Explorer 验证 IBM WebSphere MQ Telemetry 的安装 和 IBM WebSphere MQ Telemetry 样本程序。

遵循本任务中的步骤，使用不同的授权方案手动配置 IBM WebSphere MQ Telemetry 。

过程

1. 打开遥测样本目录中的命令窗口。

   遥测样本目录为 /opt/mqm/mqxr/samples。
2. 创建遥测传输队列。

```
echo "DEFINE QLOCAL('SYSTEM.MQTT.TRANSMIT.QUEUE') USAGE(XMITQ) MAXDEPTH(100000)" | runmqsc
qMgr
```

首次启动遥测 (MQXR) 服务时，它会创建 SYSTEM.MQTT.TRANSMIT.QUEUE。

它是在此任务中手动创建的，因为在启动遥测 (MQXR) 服务之前， SYSTEM.MQTT.TRANSMIT.QUEUE 必须存在，以授权对其进行访问。

3. 设置缺省传输队列

   首次启动遥测 (MQXR) 服务时，它不会更改队列管理器以使 SYSTEM.MQTT.TRANSMIT.QUEUE 成为缺省传输队列。

   要使 SYSTEM.MQTT.TRANSMIT.QUEUE 成为缺省传输队列，请更改缺省传输队列属性。 使用 IBM WebSphere MQ Explorer 或以下示例中的命令来变更属性:

   ```
   echo "ALTER QMGR DEFXMITQ('SYSTEM.MQTT.TRANSMIT.QUEUE')" | runmqsc qMgr
   ```

   更改缺省传输队列可能会干扰现有配置。 将缺省传输队列更改为 SYSTEM.MQTT.TRANSMIT.QUEUE 的原因是为了更轻松地将消息直接发送到 MQTT 客户机。 在不改变缺省传输队列的情况下，必须为接收 IBM WebSphere MQ 消息的每个客户机添加远程队列定义; 请参阅 第 107 页的『直接向客户机发送消息』。
4. 遵循 第 109 页的『授权 MQTT 客户机访问 WebSphere MQ 对象』 中的过程来创建一个或多个用户标识。 用户标识具有向 MQTT 客户机发布，预订和发送发布的权限。
5. 安装遥测 (MQXR) 服务

   ```
   cat /opt/<install_dir>/mqxr/samples/installMQXRService_unix.mqsc | runmqsc qMgr
   ```

   另请参阅 第 104 页的图 19 中的示例代码。
6. 启动服务

   ```
   echo "START SERVICE(SYSTEM.MQXR.SERVICE)" | runmqsc qMgr
   ```

   启动队列管理器时，将自动启动遥测 (MQXR) 服务。

   它在此任务中手动启动，因为队列管理器已在运行。
7. 使用 IBM WebSphere MQ Explorer，配置遥测通道以接受来自 MQTT 客户机的连接。

   必须配置遥测通道，以使其身份是步骤 4 中定义的用户标识之一。

   另请参阅 DEFINE CHANNEL (MQTT)。
8. 通过运行样本客户机来验证配置。

   要让样本客户机使用遥测通道，该通道必须授权客户机发布，预订和接收发布。 缺省情况下，样本客户机连接到端口 1883 上的遥测通道。 另请参阅 IBM WebSphere MQ Telemetry 样本程序。

示例

第 104 页的图 19 显示了用于在 Linux 上手动创建 SYSTEM.MQXR.SERVICE 的 **runmqsc** 命令。

```
DEF    SERVICE(SYSTEM.MQXR.SERVICE) +
    CONTROL(QMGR) +
    DESCR('Manages clients using MQXR protocols such as MQTT') +
    SERVTYPE(SERVER) +
    STARTCMD('+MQ_INSTALL_PATH+/mqxr/bin/runMQXRService.sh') +
    STARTARG('-m +QMNAME+ -d "+MQ_Q_MGR_DATA_PATH+" -g "+MQ_DATA_PATH+"') +
    STOPCMD('+MQ_INSTALL_PATH+/mqxr/bin/endMQXRService.sh') +
    STOPARG('-m +QMNAME+') +
    STDOUT('+MQ_Q_MGR_DATA_PATH+/mqxr.stdout') +
    STDERR('+MQ_Q_MGR_DATA_PATH+/mqxr.stderr')
```

图 *19: installMQXRService_unix.mqsc*

在 Windows 上配置队列管理器以进行遥测

遵循以下手动步骤来配置队列管理器以运行 IBM WebSphere MQ Telemetry。 您可以运行自动化过程，以使用 IBM WebSphere MQ Telemetry 对 IBM WebSphere MQ Explorer 的支持来设置更简单的配置。

开始之前

1. 有关如何安装 IBM WebSphere MQ 和 IBM WebSphere MQ Telemetry 功能部件的信息，请参阅 安装 IBM WebSphere MQ Telemetry 。
2. 创建并启动队列管理器。 在此任务中，队列管理器称为 *qMgr* 。
3. 在此任务中，您将配置遥测 (MQXR) 服务。 MQXR 属性设置存储在特定于平台的属性文件中: `mqxr_win.properties`。 通常不需要直接编辑 MQXR 属性文件，因为几乎所有设置都可以通过 MQSC 管理命令或 MQ Explorer 进行配置。 如果确实决定直接编辑该文件，请先停止队列管理器，然后再进行更改。 请参阅 MQXR 属性。

关于此任务

IBM WebSphere MQ Explorer 的 IBM WebSphere MQ Telemetry 支持包含向导和样本命令过程 `sampleMQM`。 他们使用访客用户标识设置初始配置; 请参阅 使用 IBM WebSphere MQ Explorer 验证 IBM WebSphere MQ Telemetry 的安装 和 IBM WebSphere MQ Telemetry 样本程序。

遵循本任务中的步骤，使用不同的授权方案手动配置 IBM WebSphere MQ Telemetry 。

过程

1. 打开遥测样本目录中的命令窗口。

   遥测样本目录为 *WMQ program installation directory*`\mqxr\samples`。
2. 创建遥测传输队列。

   ```
   echo DEFINE QLOCAL('SYSTEM.MQTT.TRANSMIT.QUEUE') USAGE(XMITQ) MAXDEPTH(100000) | runmqsc qMgr
   ```

   首次启动遥测 (MQXR) 服务时，它会创建 `SYSTEM.MQTT.TRANSMIT.QUEUE`。

   它是在此任务中手动创建的，因为在启动遥测 (MQXR) 服务之前， `SYSTEM.MQTT.TRANSMIT.QUEUE` 必须存在，以授权对其进行访问。
3. 设置缺省传输队列

   ```
   echo ALTER QMGR DEFXMITQ('SYSTEM.MQTT.TRANSMIT.QUEUE') | runmqsc qMgr
   ```

   图 *20:* 设置缺省传输队列

   首次启动遥测 (MQXR) 服务时，它不会更改队列管理器以使 `SYSTEM.MQTT.TRANSMIT.QUEUE` 成为缺省传输队列。

   要使 `SYSTEM.MQTT.TRANSMIT.QUEUE` 成为缺省传输队列，请更改缺省传输队列属性。 使用 IBM WebSphere MQ Explorer 或 第 105 页的图 20 中的命令来变更属性。

   更改缺省传输队列可能会干扰现有配置。 将缺省传输队列更改为 `SYSTEM.MQTT.TRANSMIT.QUEUE` 的原因是为了更轻松地将消息直接发送到 MQTT 客户机。 在不改变缺省传输队列的情况下，必须为接收 IBM WebSphere MQ 消息的每个客户机添加远程队列定义; 请参阅 第 107 页的『直接向客户机发送消息』。
4. 遵循 第 109 页的『授权 MQTT 客户机访问 WebSphere MQ 对象』 中的过程来创建一个或多个用户标识。 用户标识具有向 MQTT 客户机发布，预订和发送发布的权限。
5. 安装遥测 (MQXR) 服务

   ```
   type
   installMQXRService_win.mqsc | runmqsc qMgr
   ```
6. 启动服务

```
echo START SERVICE(SYSTEM.MQXR.SERVICE) | runmqsc qMgr
```

启动队列管理器时，将自动启动遥测 (MQXR) 服务。

它在此任务中手动启动，因为队列管理器已在运行。

7. 使用 IBM WebSphere MQ Explorer，配置遥测通道以接受来自 MQTT 客户机的连接。

   必须配置遥测通道，以使其身份是步骤 4 中定义的用户标识之一。

   另请参阅 DEFINE CHANNEL (MQTT)。

8. 通过运行样本客户机来验证配置。

   要让样本客户机使用遥测通道，该通道必须授权客户机发布，预订和接收发布。 缺省情况下，样本客户机连接到端口 1883 上的遥测通道。 另请参阅 IBM WebSphere MQ Telemetry 样本程序。

手动创建 SYSTEM.MQXR.SERVICE

第 106 页的图 21 显示了用于在 Windows 上手动创建 SYSTEM.MQXR.SERVICE 的 **runmqsc** 命令。

```
DEF    SERVICE(SYSTEM.MQXR.SERVICE) +
    CONTROL(QMGR) +
    DESCR('Manages clients using MQXR protocols such as MQTT') +
    SERVTYPE(SERVER) +
    STARTCMD('+MQ_INSTALL_PATH+\mqxr\bin\runMQXRService.bat') +
    STARTARG('-m +QMNAME+  -d "+MQ_Q_MGR_DATA_PATH+\." -g "+MQ_DATA_PATH+\."') +
    STOPCMD('+MQ_INSTALL_PATH+\mqxr\bin\endMQXRService.bat') +
    STOPARG('-m +QMNAME+') +
    STDOUT('+MQ_Q_MGR_DATA_PATH+\mqxr.stdout') +
    STDERR('+MQ_Q_MGR_DATA_PATH+\mqxr.stderr')
```

图 *21: installMQXRService_win.mqsc*

配置分布式队列以将消息发送至 MQTT 客户机

IBM WebSphere MQ 应用程序可以通过发布到客户机创建的预订或通过直接发送消息来发送 MQTT v3 客户机消息。 无论使用哪种方法，都将消息放在 SYSTEM.MQTT.TRANSMIT.QUEUE 上，并通过遥测 (MQXR) 服务发送到客户机。 可以通过多种方法在 SYSTEM.MQTT.TRANSMIT.QUEUE 上放置消息。

发布消息以响应 MQTT 客户机预订

遥测 (MQXR) 服务代表 MQTT 客户机创建预订。 客户机是与客户机发送的预订匹配的任何发布的目标。 遥测服务会将匹配的发布转发回客户机。

MQTT 客户机作为队列管理器连接到 WebSphere MQ ，其队列管理器名称设置为其 ClientIdentifier。 要发送到客户机的发布的目标是传输队列 SYSTEM.MQTT.TRANSMIT.QUEUE。 遥测服务使用目标队列管理器名称作为特定客户机的密钥，将 SYSTEM.MQTT.TRANSMIT.QUEUE 上的消息转发到 MQTT 客户机。

遥测 (MQXR) 服务使用 ClientIdentifier 作为队列管理器名称来打开传输队列。 遥测 (MQXR) 服务将队列的对象句柄传递到 MQSUB 调用，以转发与客户机预订匹配的发布。 在对象名解析中，将创建 ClientIdentifier 作为远程队列管理器名称，并且传输队列必须解析为 SYSTEM.MQTT.TRANSMIT.QUEUE。 使用标准 WebSphere MQ 对象名解析，将按如下所示解析 *ClientIdentifier*；请参阅 第 107 页的表 6。

1. *ClientIdentifier* 与任何内容都不匹配。

   ClientIdentifier 是远程队列管理器名称。 它与本地队列管理器名称，队列管理器别名或传输队列名称不匹配。

   未定义队列名称。 目前，遥测 (MQXR) 服务将 SYSTEM.MQTT.PUBLICATION.QUEUE 设置为队列的名称。 MQTT v3 客户机不支持队列，因此客户机将忽略已解析的队列名称。

   必须将本地队列管理器属性 **缺省传输队列**的名称设置为 SYSTEM.MQTT.TRANSMIT.QUEUE，以便将发布放在 SYSTEM.MQTT.TRANSMIT.QUEUE 上以发送到客户机。

2. *ClientIdentifier* 与名为 *ClientIdentifier* 的队列管理器别名匹配。

   ClientIdentifier 是远程队列管理器名称。 它与队列管理器别名的名称相匹配。

必须使用 *ClientIdentifier* 作为远程队列管理器名称来定义队列管理器别名。
通过在队列管理器别名定义中设置传输队列名称，不必将缺省传输设置为
SYSTEM.MQTT.TRANSMIT.QUEUE。

表 *6: MQTT* 队列管理器别名的名称解析

	Input		**Output**		
ClientIdentifier	**队列管理器名称**	**队列名称**	**队列管理器名称**	**队列名称**	**传输队列**
不匹配任何内容	*ClientIdentifier*	未定义	*ClientIdentifier*	未定义	缺省传输队列。 SYSTEM.MQTT.TRANSMIT.QUEUE
与名为 *ClientIdentifier* 的队列管理器别名匹配	*ClientIdentifier*	未定义	*ClientIdentifier*	未定义	SYSTEM.MQTT.TRANSMIT.QUEUE

有关名称解析的更多信息，请参阅 名称解析。

任何 WebSphere MQ 程序都可以发布到同一主题。 该出版物将发送到其订户，包括预订该主题的 MQTT v3 客户机。

如果在集群中创建了具有属性 CLUSTER(*clusterName*)的管理主题，那么集群中的任何应用程序都可以发布到客户机; 例如:

```
echo DEFINE TOPIC('MQTTExamples') TOPICSTR('MQTT Examples') CLUSTER(MQTT) REPLACE | runmqsc qMgr
```

图 *22:* 在 *Windows* 上定义集群主题

注: 请勿为 SYSTEM.MQTT.TRANSMIT.QUEUE 提供集群属性。

MQTT 客户机订户和发布程序可以连接到不同的队列管理器。 订户和发布者可以是同一集群的一部分，也可以通过发布/预订层次结构进行连接。 该出版物将使用 WebSphere MQ 从发布者交付到订户。

直接向客户机发送消息

作为创建预订并接收与预订主题匹配的发布的客户机的替代方法，直接将消息发送到 MQTT v3 客户机。 MQTT V3 客户机应用程序无法直接发送消息，但其他应用程序 (例如 WebSphere MQ 应用程序) 可以发送消息。

WebSphere MQ 应用程序必须知道 MQTT v3 客户机的 ClientIdentifier 。 由于 MQTT v3 客户机没有队列，因此会将目标队列名称作为主题名称传递到 MQTT v3 应用程序客户机 messageArrived 方法。 例如，在 MQI 程序中，使用客户机作为 ObjectQmgr 名称创建对象描述符:

```
MQOD.ObjectQmgrName = ClientIdentifier;
MQOD.ObjectName = name;
```

图 *23:* 用于将消息发送到 *MQTT v3* 客户机目标的 *MQI* 对象描述符

如果应用程序是使用 JMS 编写的，请创建点到点目标; 例如:

```
javax.jms.Destination jmsDestination =
                      (javax.jms.Destination)jmsFactory.createQueue
                      ("queue://ClientIdentifier/name");
```

图 *24:* 用于将消息发送到 *MQTT v3* 客户机的 *JMS* 目标

要向 MQTT 客户机发送非请求消息，请使用远程队列定义。 远程队列管理器名称必须解析为客户机的 `ClientIdentifier`。 传输队列必须解析为 SYSTEM.MQTT.TRANSMIT.QUEUE; 请参阅 第 108 页的表 7。 远程队列名称可以是任何内容。 客户机将其作为主题字符串接收。

表 *7: MQTT* 客户机远程队列定义的名称解析

Input		**Output**		
队列名称	**队列管理器名称**	**队列名称**	**队列管理器名称**	**传输队列**
远程队列定义的名称	空白或本地队列管理器名称	用作主题字符串的远程队列名称	*ClientIdentifier*	SYSTEM.MQTT.TRANSMIT.QUEUE

如果客户机已连接，那么会将消息直接发送到 MQTT 客户机，这将调用 `messageArrived` 方法; 请参阅 `messageArrived` 方法。

如果客户机已与持久会话断开连接，那么消息将存储在 SYSTEM.MQTT.TRANSMIT.QUEUE 中; 请参阅 MQTT 无状态和有状态会话。 当客户机再次重新连接到会话时，会将其转发到客户机。

如果发送非持久消息，那么会将其以 "最多一次" 服务质量 QoS=0 发送到客户机。 如果直接向客户机发送持久消息，那么缺省情况下，将以 "精确一次" 服务质量 QoS=2 发送该消息。 由于客户机可能没有持久性机制，因此客户机可以降低它为直接发送的消息接受的服务质量。 要降低直接发送到客户机的消息的服务质量，请预订主题 DEFAULT.QoS。 指定客户机可支持的最大服务质量。

MQTT 客户机标识、授权和认证

遥测 (MQXR) 服务使用 MQTT 通道来代表 MQTT 客户机发布或预订 WebSphere MQ 主题。 WebSphere MQ 管理员配置用于 WebSphere MQ 授权的 MQTT 通道身份。 管理员可以为通道定义一个公共身份，也可以使用与此通道相连的客户机的**用户名**或**客户机标识**。

遥测 (MQXR) 服务可以使用客户机所提供的 `Username` 或者使用客户机证书来认证客户机。 使用客户机所提供的密码来认证**用户名**。

总结：客户机标识就是选择的客户机身份。 根据上下文不同，可通过**客户机标识**、**用户名**、由管理员创建的公共客户机身份或者客户机证书来标识客户机。 用于检查真实性的客户机标识不必是用于授权的同一标识。

MQTT 客户机程序设置通过使用 MQTT 通道发送至服务器的**用户名**和**密码**。 它们还可以设置一些 SSL 属性，对连接进行加密和认证时就需要这些 SSL 属性。 管理员将决定是否对 MQTT 通道进行认证以及如何进行认证。

要授权 MQTT 客户机访问 WebSphere MQ 对象，请对客户机的**客户机标识**或**用户名**授权，或者对公共客户机身份授权。 要允许客户机连接到 WebSphere MQ，请对**用户名**进行认证或使用客户机证书。 配置 JAAS 以对**用户名**进行认证，并配置 SSL 以对客户机证书进行认证。

如果您在客户机中设置**密码**，那么使用 VPN 对连接进行加密或者配置 MQTT 通道以使用 SSL，从而保持密码的私密性。

难以管理客户机证书。 正因为如此，如果可以接受进行密码认证存在的风险，那么通常使用密码认证对客户机进行认证。

如果有一种安全的方法来管理和存储客户机证书，那么可以依赖于证书认证来实现。 但是，在使用 Telemetry 的各种类型的环境中，很少能够安全地管理证书。 然而，通过在服务器中对客户机密码进行认证，可以作为“使用客户机证书对客户机进行认证”的补充。 由于还存在其他复杂性，因此，仅限于对高度敏感的应用程序使用客户机证书。 使用两种形式的认证被称为双因素认证。 您必须知道其中一个因素（例如，密码），并且具有另一个因素（例如，证书）。

在高度敏感的应用程序中（例如，chip-and-pin 设备），在制造期间将锁定此设备，以防止篡改内部硬件和软件。 会将一个可信的、具有时间限制的客户机证书复制到此设备中。 将此设备部署到要使用它的位置。 每当使用此设备时，都会使用密码或者来自智能卡的另一个证书对它执行进一步认证。

MQTT 客户机身份和权限

使用 `ClientIdentifier`、`Username` 或者用于授权的常用客户机身份来访问 WebSphere MQ 对象。

WebSphere MQ 管理员在选择 MQTT 通道的身份时有三个选项可供选择。 当定义或修改客户机所使用的 MQTT 通道时，管理员可以作出选择。 身份用来授予对于 WebSphere MQ 主题的访问权。 这些选项包括：

1. 客户机标识。
2. 管理员为通道提供的身份。
3. 从 MQTT 客户机传递的**用户名**。

用户名是 MqttConnectOptions 类的一种属性。 它必须在客户机连接至服务之前进行设置。 它的缺省值为 NULL。

使用 WebSphere MQ **setmqaut** 命令来选择授权与 MQTT 通道相关联的身份使用哪些对象和哪些操作。 例如，要授予由队列管理器 `QM1` 的管理员提供的通道身份 `MQTTClient`：

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

授权 *MQTT* 客户机访问 *WebSphere MQ* 对象

执行以下步骤以授权 MQTT 客户机发布和预订 WebSphere MQ 对象。 这些步骤遵循四种备用访问控制模式。

开始之前

MQTT 客户机在连接到遥测通道时被分配身份，从而有权访问 WebSphere MQ 中的对象。 WebSphere MQ 管理员使用 WebSphere MQ Explorer 配置遥测通道，以向客户机提供三种类型的身份之一：

1. `ClientIdentifier`
2. **用户名**
3. 管理员分配给通道的名称。

无论使用哪种类型，身份都必须由已安装的授权服务定义到 WebSphere MQ 作为主体。 Windows 或 Linux 上的缺省授权服务称为对象权限管理器 (OAM)。 如果您正在使用 OAM，那么必须将身份定义为用户标识。

使用身份来授予客户机或客户机集合发布或预订 WebSphere MQ 中定义的主题的许可权。 如果 MQTT 客户机已预订主题，请使用该身份向其授予接收生成的发布的许可权。

很难管理具有数万个 MQTT 客户机的系统，每个客户机都需要单独的访问许可权。 一种解决方案是定义公共身份，并将单个 MQTT 客户机与其中一个公共身份相关联。 定义所需数量的公共身份，以定义不同的许可权组合。 另一个解决方案是编写您自己的授权服务，该服务可以比操作系统更轻松地处理数以千计的用户。

您可以使用 OAM 通过两种方式将 MQTT 客户机组合为公共身份：

1. 定义多个遥测通道，每个通道具有管理员使用 WebSphere MQ Explorer 分配的不同用户标识。 使用不同 TCP/IP 端口号进行连接的客户机与不同的遥测通道相关联，并分配不同的身份。
2. 定义单个遥测通道，但让每个客户机从一小组用户标识中选择一个 **用户名** 。 管理员配置遥测通道以选择客户机 **用户名** 作为其身份。

在此任务中，遥测通道的标识称为 *`mqttUser`*，而不考虑其设置方式。 如果客户机集合使用不同的身份，请使用多个 *`mqttUsers`*，每个客户机集合一个。 由于任务使用 OAM，因此每个 *`mqttUser`* 都必须是用户标识。

关于此任务

在此任务中，您可以选择可根据特定需求定制的四个访问控制模式。 这些模式的访问控制粒度不同。

- 第 110 页的『无访问控制』
- 第 110 页的『粗颗粒度访问控制』
- 第 110 页的『中颗粒度访问控制』
- 第 110 页的『细颗粒度访问控制』

模型的结果是分配 *mqttUsers* 组许可权以发布和预订 WebSphere MQ，并从 WebSphere MQ 接收发布。

无访问控制
MQTT 客户机具有 WebSphere MQ 管理权限，并且可以对任何对象执行任何操作。

过程

1. 创建用户标识 *mqttUser* 以充当所有 MQTT 客户机的身份。
2. 将 *mqttUser* 添加到 mqm 组; 请参阅 在 Windows 上向组添加用户或 在 Linux 上向组添加用户

粗颗粒度访问控制
MQTT 客户机具有发布和预订以及向 MQTT 客户机发送消息的权限。 他们无权执行其他操作或访问其他对象。

过程

1. 创建用户标识 *mqttUser* 以充当所有 MQTT 客户机的身份。
2. 授权 *mqttUser* 发布和预订所有主题以及向 MQTT 客户机发送发布内容。

```
setmqaut -m qMgr -t topic -n SYSTEM.BASE.TOPIC -p mqttUser -all +pub +sub
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqttUser -all +put
```

中颗粒度访问控制
MQTT 客户机分为不同的组，以发布和预订不同的主题集，并将消息发送到 MQTT 客户机。

过程

1. 在发布/预订主题树中创建多个用户标识 *mqttUsers* 和多个管理主题。
2. 将不同的 *mqttUsers* 授权给不同的主题。

```
setmqaut -m qMgr -t topic -n topic1 -p mqttUserA -all +pub +sub
setmqaut -m qMgr -t topic -n topic2 -p mqttUserB -all +pub +sub
```

3. 创建组 *mqtt*，并将所有 *mqttUsers* 添加到该组。
4. 授权 *mqtt* 将主题发送到 MQTT 客户机。

```
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqtt -all +put
```

细颗粒度访问控制
MQTT 客户机合并到现有访问控制系统中，该系统授权组对对象执行操作。

关于此任务

根据用户标识所需的权限，将该用户标识分配给一个或多个操作系统组。 如果 WebSphere MQ 应用程序正在发布和预订与 MQTT 客户机相同的主题空间，请使用此模型。 这些组称为 Publish*X*， Subscribe*Y* 和 mqtt

Publish*X*
Publish*X* 组的成员可以发布到 *topicX*。

Subscribe*Y*
Subscribe*Y* 组的成员可以预订 *topicY*。

mqtt
mqtt 组的成员可以将发布发送到 MQTT 客户机。

过程

1. 创建分配给发布/预订主题树中的多个管理主题的多个组 Publish*X* 和 Subscribe*Y* 。
2. 创建组 mqtt。
3. 创建多个用户标识 *mqttUsers*，并将这些用户添加到任何组，具体取决于他们有权执行的操作。
4. 将不同的 Publish*X* 和 Subscribe*X* 组授权给不同的主题，并授权 *mqtt* 组将消息发送给 MQTT 客户机。

```
setmqaut -m qMgr -t topic -n topic1 -p PublishX -all +pub
setmqaut -m qMgr -t topic -n topic1 -p SubscribeX -all +pub +sub
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqtt -all +put
```

使用密码对 MQTT 客户机进行认证

使用客户机密码来认证**用户名**。 使用与用来授权客户机发布和预订主题的身份不同的身份对客户机进行认证。

遥测 (MQXR) 服务使用 JAAS 来认证客户机 Username。 JAAS 使用 MQTT 客户机所提供的**密码**。

WebSphere MQ 管理员决定是通过配置客户机连接至的 MQTT 通道来认证**用户名**，还是根本不认证。 可以将客户机分配给不同的通道，并且可以配置每个通道以采用不同方式对它的客户机进行认证。 通过使用 JAAS，可以配置哪些方法必须对客户机进行认证，哪些方法可以有选择地对客户机进行认证。

为认证选择的身份并不会影响为授权选择的身份。 为了便于管理，您可能想为授权设置一个公共身份，但是对每个用户进行认证以使用该身份。 以下过程概述了对各个用户进行认证以使用公共身份时要执行的步骤：

1. WebSphere MQ 管理员使用 WebSphere MQ Explorer 将 MQTT 通道身份设置为任何名称（例如，MQTTClientUser）。
2. WebSphere MQ 管理员授权 MQTTClient 发布和预订任何主题：

```
setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

3. MQTT 客户机应用程序开发者在连接到服务器之前创建 MqttConnectOptions 对象并设置 **用户名** 和 **密码** 。
4. 安全性开发者创建一个 JAAS LoginModule，以使用**密码**来认证**用户名**，并将此模块包括在 JAAS 配置文件中。
5. WebSphere MQ 管理员配置 MQTT 通道，以使用 JAAS 来认证客户机的**用户名**。

使用 SSL 进行 MQTT 客户机认证

MQTT 客户机与队列管理器之间的连接始终由 MQTT 客户机发起。 MQTT 客户机始终是 SSL 客户机。 服务器的客户机认证和 MQTT 客户机的服务器认证都是可选的。

通过向客户机提供专用签名数字证书，可以向 IBM WebSphere MQ 认证 MQTT 客户机。 IBM WebSphere MQ 管理员可以强制 MQTT 客户机使用 SSL 向队列管理器认证这些客户机自身。 只能在相互认证的过程中请求客户机认证。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

使用 SSL 的客户机认证依赖于具有私钥的客户机。 私钥是客户机的专用密钥（对于自签名证书来说），或者是由认证中心提供的密钥。 密钥用来为客户机的数字证书签名。 拥有相应的公用密钥的任何人都可以验证数字证书。 证书可能是可信的，如果它们是链式证书，那么追溯整个证书链以找到可信根证书。 客户机验证将客户机所提供的证书链中的所有证书发送至服务器。 服务器将检查证书链，直到找到它信任的证书为止。 可信证书是根据自签名证书生成的公用证书，或者是通常由认证中心发放的根证书。 作为最后的可选步骤，可以将可信证书与“实时”的证书撤销列表进行比较。

可信证书可能已由认证中心颁发并已经包含在 JRE 证书库中。 它可以是自签名证书，也可以是已经作为可信证书添加至遥测通道密钥库的任何证书。

注: 遥测通道具有组合密钥库/信任密钥库，该库中包含一个或多个遥测通道的专用密钥以及对客户机进行认证所需的所有公用证书。 由于 SSL 通道必须具有密钥库且它是与通道信任库相同的文件，因此绝不会引用 JRE 证书库。 其含义为，如果客户机认证需要认证中心根证书，那么即使该证书已存在于 JRE 证书库中，也必须将其放入该通道的密钥库中。 绝不会引用 JRE 证书库。

请考虑进行客户机认证是为了对付威胁，以及客户机在服务器对付威胁时所起的作用。 只是对客户机证书进行认证还不足以防止对系统进行未经授权的访问。 如果其他人控制了客户机设备，那么该客户机设备不一定采用证书拥有者的权限来运作。 决不能依赖单一防护措施来对付不希望出现的攻击。 至少应使用双重认证方法，并使用私有信息的知识来补充证书的内容。 例如，使用 JAAS，以及使用由服务器发放的密码来认证客户机。

客户机证书面对的主要威胁是被不适当的人拥有。 证书保存在客户机上的一个受密码保护的密钥库中。 它是如何保存在密钥库中的？ MQTT 客户机如何将密码保存到密钥库中？ 密码保护的安全程度如何？ 遥测设备通常容易被移除，然后可以私下修改。 设备硬件必须防篡改吗？ 分发和保护客户机端证书被认为是一项艰巨的任务；它被称为密钥管理问题。

次要的威胁是，无意识地误用了设备来访问服务器。 例如，如果篡改了 MQTT 应用程序，那么有可能在使用已认证的客户机身份的服务器配置中使用薄弱环节。

要使用 SSL 对 MQTT 客户机进行认证，请配置遥测通道和此客户机。

-
-

使用 *SSL* 进行 *MQTT* 客户机认证的遥测通道配置

IBM WebSphere MQ 管理员在服务器上配置遥测通道。 每个通道被配置为接受不同端口号上的 TCP/IP 连接。 配置了 SSL 通道，以对密钥文件进行受口令保护的访问。 如果没有为 SSL 通道定义口令或密钥文件，那么此通道将不接受 SSL 连接。

将 SSL 遥测通道的属性 `com.ibm.mq.MQTT.ClientAuth` 设置为 `REQUIRED`，以强制在该通道上连接的所有客户机提供已验证数字证书的证明。 使用来自认证中心的证书对客户机证书进行认证，从而生成可信根证书。 如果客户机证书是自签名证书，或者由来自认证中心的证书签署，那么客户机或认证中心的公用签名证书必须安全地存储在服务器上。

将公用签名的客户机证书或来自认证中心的证书放在遥测通道密钥库中。 在服务器上，公用签名证书与专用签名证书存储在同一密钥文件中，而不是存储在单独的信任库中。

服务器使用其拥有的所有公用证书和密码套件来验证其发送的任何客户机证书的签名。 服务器验证密钥链。 可以配置队列管理器以根据证书撤销列表测试证书。 队列管理器撤销名称列表属性为 `SSLCRLNL`。

如果客户机发送的任何证书由服务器密钥库中的证书验证，那么将对客户机进行认证。

WebSphere MQ 管理员可以配置相同的遥测通道，以使用 JAAS 通过客户机 **密码**来检查客户机的 `UserName` 或 `ClientIdentifier`。

您可以将同一密钥库用于多个遥测通道。

验证设备上受密码保护的客户机密钥库中的至少一个数字证书会向服务器认证客户机。 数字证书仅用于 WebSphere MQ 的认证。 它不用于验证客户机的 TCP/IP 地址，也不用于设置客户机的身份以进行授权或记帐。 服务器采用的客户机的身份是客户机的 **用户名** 或 `ClientIdentifier`，或者是由 WebSphere MQ 管理员创建的身份。

您还可以将 SSL 密码套件用于客户机认证。以下是当前受支持的 SSL 密码套件的字母顺序列表：

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`

- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 如果计划使用 SHA-2 密码套件，请参阅将 SHA-2 密码套件与 MQTT 通道配合使用的系统需求。

相关概念

第 114 页的『使用 SSL 进行通道认证的遥测通道配置』

IBM WebSphere MQ 管理员在服务器上配置遥测通道。 每个通道被配置为接受不同端口号上的 TCP/IP 连接。 配置了 SSL 通道，以对密钥文件进行受口令保护的访问。 如果没有为 SSL 通道定义口令或密钥文件，那么此通道将不接受 SSL 连接。

CipherSpecs 和 CipherSuites

相关参考
DEFINE CHANNEL (MQTT)
ALTER CHANNEL (MQTT)

使用 SSL 进行遥测通道认证

MQTT 客户机与队列管理器之间的连接始终由 MQTT 客户机发起。 MQTT 客户机始终是 SSL 客户机。 服务器的客户机认证和 MQTT 客户机的服务器认证都是可选的。

除非客户机被配置为使用支持匿名连接的 CipherSpec，否则客户机始终会尝试对服务器进行认证。 如果认证失败，那么不会建立连接。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

使用 SSL 的服务器认证将对您要将保密信息发送至的服务器进行认证。 客户机根据其信任库或 JRE `cacerts` 库中的证书执行与从服务器发送的证书相匹配的检查。

JRE 证书库是 JKS 文件 `cacerts`。 它位于 `JRE InstallPath\lib\security\` 中。 它是使用缺省密码 `changeit` 进行安装的。 您可以将信任的证书存储在 JRE 证书库或客户机信任库中。 不能同时使用这两个库。 如果要使客户机信任的公用证书独立于其他 Java 应用程序使用的证书，请使用客户机信任库。 如果要针对客户机上正在运行的所有 Java 应用程序使用公共证书库，请使用 JRE 证书库。 如果确定要使用 JRE 证书库，请查看其所含证书以确保您信任这些证书。

可以通过提供另外的信任提供程序来修改 JSSE 配置。 可以定制信任提供程序以对证书执行另外的检查。 在一些使用了 MQTT 客户机的 OGSi 环境中， 该环境提供了不同的信任提供程序。

要使用 SSL 对遥测通道进行认证，请配置服务器和客户机。

-
-

使用 SSL 进行通道认证的遥测通道配置

IBM WebSphere MQ 管理员在服务器上配置遥测通道。 每个通道被配置为接受不同端口号上的 TCP/IP 连接。 配置了 SSL 通道，以对密钥文件进行受口令保护的访问。 如果没有为 SSL 通道定义口令或密钥文件，那么此通道将不接受 SSL 连接。

将使用其专用密钥签名的服务器的数字证书存储在遥测通道将在服务器上使用的密钥库中。 如果要将密钥链传输到客户机，请将其密钥链中的任何证书存储在密钥库中。 使用 WebSphere MQ 资源管理器配置遥测通道以使用 SSL。 为其提供密钥库的路径以及用于访问密钥库的口令。 如果未设置通道的 TCP/IP 端口号，那么 SSL 遥测通道端口号缺省为 8883。

也可使用 SSL 密码套件进行通道认证。以下是当前受支持的 SSL 密码套件的字母顺序列表：

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`

- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 如果计划使用 SHA-2 密码套件，请参阅将 SHA-2 密码套件与 MQTT 通道配合使用的系统需求。

相关概念

第 112 页的『使用 SSL 进行 MQTT 客户机认证的遥测通道配置』

IBM WebSphere MQ 管理员在服务器上配置遥测通道。 每个通道被配置为接受不同端口号上的 TCP/IP 连接。 配置了 SSL 通道，以对密钥文件进行受口令保护的访问。 如果没有为 SSL 通道定义口令或密钥文件，那么此通道将不接受 SSL 连接。

CipherSpecs 和 CipherSuites

相关参考

DEFINE CHANNEL (MQTT)

ALTER CHANNEL (MQTT)

发布在遥测通道上的隐私

通过使用 SSL 对借助连接实现的传输进行加密，即可保护在遥测通道中按任一方向发送的 MQTT 发布的隐私。

连接至遥测通道的 MQTT 客户机使用 SSL 来保护在通道上使用对称密钥密码来传输的发布的隐私。 由于未对端点进行认证，因此您不能只信任通道加密。 应将保护隐私与服务器认证或相互认证结合起来。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

对于典型配置（它会对通道进行加密和对服务器进行认证），请查阅第 114 页的『使用 SSL 进行遥测通道认证』。

只是对 SSL 连接进行加密，而不对服务器进行认证，会使连接受到中间人攻击。 尽管可以保护您交换的信息不被窃听，但是您并不知道是与谁在交换信息。 除非您控制整个网络，否则其他人就有可能拦截您的 IP 传输并且伪装成端点。

通过使用支持匿名 SSL 的 Diffie-Hellman 密钥交换 CipherSpec，即可创建已加密的 SSL 连接，而不对服务器进行认证。 建立了在客户机与服务器之间共享的主密钥，用来对 SSL 传输进行加密，而不交换私下签名的服务器证书。

因为匿名连接不安全，所以大多数 SSL 实现并不缺省设置为使用匿名 CipherSpec。 如果遥测通道接受了客户机请求以建立 SSL 连接，那么此通道必须具有受口令保护的密钥库。 缺省情况下，由于 SSL 实现不使用匿名 CipherSpec，因此密钥库中必须包含私下签名的证书，客户机可以对此证书进行认证。

如果您使用匿名 CipherSpec，那么服务器密钥库必须存在，但是它不需要包含任何私下签名的证书。

另一种建立加密连接的方法是，将客户机中的信任提供程序替换为您自己的实现。 您的信任提供程序将不对服务器证书进行认证，但是连接将加密。

MQTT 客户机和遥测通道的 SSL 配置

MQTT 客户机和 WebSphere MQ Telemetry (MQXR) 服务使用 Java 安全套接字扩展 (JSSE) 通过 SSL 连接遥测通道。 设备的 IBM WebSphere MQ Telemetry 守护程序不支持 SSL。

配置 SSL 以认证遥测通道， MQTT 客户机，并加密客户机与遥测通道之间的消息传输。

除了使用 SSL 以外，某些种类的虚拟专用网 (VPN)（例如，IPSec）将对 TCP/IP 连接的端点进行认证。 VPN 将对流经网络的每个 IP 包进行加密。 一旦建立了这样一个 VPN 连接，您就已经建立了一个可信网络。 您可以使用 VPN 网络上的 TCP/IP 将 MQTT 客户机连接至遥测通道。

您可以配置 Java MQTT 客户机与遥测通道之间的连接，以使用基于 TCP/IP 的 SSL 协议。 所保护的对象取决于您如何配置 SSL 以使用 JSSE。 从最安全的配置开始，您可以配置三种不同级别的安全性：

1. 只允许可信的 MQTT 客户机进行连接。 将 MQTT 客户机仅连接至可信的遥测通道。 对客户机与队列管理器之间的消息进行加密; 请参阅 第 111 页的『使用 SSL 进行 MQTT 客户机认证』
2. 将 MQTT 客户机仅连接至可信的遥测通道。 对客户机与队列管理器之间传递的消息进行加密；请参阅第 114 页的『使用 SSL 进行遥测通道认证』。
3. 对客户机与队列管理器之间传递的消息进行加密；请参阅第 116 页的『发布在遥测通道上的隐私』。

JSSE 配置参数

修改 JSSE 参数，以改变 SSL 连接的配置方式。 JSSE 配置参数分为三个集合：

1. IBM WebSphere MQ 遥测通道
2. MQTT Java 客户机
3. JRE

使用 IBM WebSphere MQ Explorer 配置遥测通道参数。 在 `MqttConnectionOptions.SSLProperties` 属性中设置 MQTT Java 客户机参数。 通过编辑客户机和服务器中的 JRE 安全性目录中的文件来修改 JRE 安全性参数。

IBM WebSphere MQ 遥测通道

使用 WebSphere MQ Explorer 来设置所有遥测通道 SSL 参数。

ChannelName

ChannelName 是所有通道的一个必需参数。

通道名称标识与特定端口号相关联的通道。 为通道命名，以帮助您管理多组 MQTT 客户机。

PortNumber

PortNumber 是所有通道的一个可选参数。 对于 TCP 通道，此参数的缺省值为 1883；对于 SSL 通道，此参数的缺省值为 8883。

TCP/IP 端口号与此通道相关联。 通过指定为通道定义的端口来将 MQTT 客户机连接至该通道。 如果该通道具有 SSL 属性，那么客户机必须使用 SSL 协议进行连接；例如：

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

KeyFileName 是 SSL 通道的一个必需参数。 对于 TCP 通道，必须省略此参数。

KeyFileName 是包含您提供的数字证书的 Java 密钥库的路径。 使用 JKS、JCEKS 或 PKCS12 作为服务器上的密钥库类型。

可使用以下某个文件扩展名来确定密钥库类型：

```
.jks
.jceks
.p12
.pkcs12
```

具有其他任何文件扩展名的密钥库均视为 JKS 密钥库。

您可以将服务器上的一类密钥库与客户机上其他类型的密钥库相合并。

将服务器的私有证书放在密钥库中。 证书被称为服务器证书。 证书可以是自签名证书，也可以是签署机构所签署的证书链的一部分。

如果您要使用证书链，请将关联的证书放在服务器密钥库中。

服务器证书及其证书链中的任何证书被发送至客户机以认证服务器的身份。

如果您已将 ClientAuth 设置为 Required，那么密钥库中必须包含对客户机进行认证时所需要的任何证书。 客户机发送一个自签名证书，或一个证书链，这样会对照密钥库中的证书，通过首次验证此材料，对客户机进行认证。 通过使用证书链，一个证书可以验证许多客户机，即使向它们颁发了不同的客户机证书，也是如此。

PassPhrase

PassPhrase 是 SSL 通道的一个必需参数。 对于 TCP 通道，必须省略此参数。

口令用来保护密钥库。

ClientAuth

ClientAuth 是一个可选的 SSL 参数。 它缺省设置为不进行客户机认证。 对于 TCP 通道，必须省略此参数。

如果您想要遥测 (MQXR) 服务在允许客户机连接至遥测通道之前认证客户机，那么请设置 ClientAuth。

如果您设置 ClientAuth，那么客户机必须使用 SSL 连接至服务器并且对服务器进行认证。 作为对设置 ClientAuth 的响应，客户机将它的数字证书及其密钥库中的任何其他证书发送至服务器。 它的数字证书被称为客户机证书。 将针对保存在通道密钥库以及 JRE cacerts 库中的证书来认证这些证书。

CipherSuite

CipherSuite 是一个可选的 SSL 参数。 它缺省设置为尝试所有已启用的 CipherSpec。 对于 TCP 通道，必须省略此参数。

如果您想使用特定 CipherSpec，那么将 CipherSuite 设置为必须用来建立 SSL 连接的 CipherSpec 的名称。

遥测服务和 MQTT 客户机协商在每一端启用的所有 CipherSpec 中的一个公用 CipherSpec。 如果在连接的其中一端或者两端指定了一个特定 CipherSpec，那么此 CipherSpec 必须与另一端的 CipherSpec 相匹配。

通过将其他提供程序添加至 JSSE 来安装其他密码。

联邦信息处理标准 （FIPS）

FIPS 是一个可选设置。 缺省情况下，不会设置此项。

在队列管理器的“属性”面板中设置 SSLFIPS，或者使用 **runmqsc** 来设置。 SSLFIPS 指定是否仅使用经过 FIPS 证明的算法。

撤销名称列表

“撤销名称列表”是一个可选设置。 缺省情况下，不会设置此项。

在队列管理器的“属性”面板中设置 SSLCRLNL，或者使用 **runmqsc** 来设置。 SSLCRLNL 指定用来提供证书撤销位置的认证信息对象的名称列表。

不会使用其他用于设置 SSL 属性的队列管理器参数。

MQTT Java 客户机

在 MqttConnectionOptions.SSLProperties 中设置 Java 客户机的 SSL 属性; 例如:

```
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
com.ibm.micro.client.mqttv3.MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setSSLProperties(sslClientProperties);
```

MqttConnectOptions 类中描述了特定属性的名称和值。 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。

协议

Protocol 是可选的。

协议是在与遥测服务器协商时选择的。 如果您需要特定协议，那么可以选择一种协议。 如果遥测服务器不支持此协议，那么连接将失败。

ContextProvider

ContextProvider 是可选的。

KeyStore

KeyStore 是可选的。 如果在服务器中设置了 ClientAuth 以强制对客户机进行认证，那么请配置此项。

将使用客户机的专用密钥签名的数字证书放入密钥库中。 指定密钥库的路径和密码。 类型和提供程序是可选的。 JKS 是缺省类型，IBMJCE 是缺省提供程序。

指定另外的密钥库提供程序，以引用用于添加新的密钥库提供程序的类。 传递密钥库提供程序所使用的算法的名称，以通过设置密钥管理器名称来实例化 KeyManagerFactory。

TrustStore

TrustStore 是可选的。 您可以将您信任的所有证书都放在 JRE cacerts 存储库中。

如果要为客户机设置不同的信任库，请配置信任库。 如果服务器正在使用已在 cacerts 中存储其根证书的众所周知的认证中心发放的证书，那么可能无法配置信任库。

将服务器的公开签名的证书或者根证书添加至信任库，并指定信任库的路径和密码。 JKS 是缺省类型，IBMJCE 是缺省提供程序。

指定另外的信任库提供程序，以引用用于添加新的信任库提供程序的类。 传递信任库提供程序所使用的算法的名称，以通过设置信任管理器名称来实例化 TrustManagerFactory。

JRE

在 JRE 中配置了 Java 安全性的其他方面（这些方面影响客户机和服务器上的 SSL 行为）。 Windows 上的配置文件位于 *Java Installation Directory*\jre\lib\security 中。 如果使用 IBM WebSphere MQ 随附的 JRE，那么该路径如下表中所示：

表 *8: JRE SSL* 配置文件所在的不同平台的文件路径	
平台	**菲尔帕特**
Windows	*WMQ Installation Directory*\java\jre\lib\security
其他 UNIX and Linux 平台	*WMQ Installation Directory*/java/jre64/jre/lib/security

众所周知的认证中心

cacerts 文件中包含众所周知的认证中心的根证书。 除非您指定信任库，否则缺省情况下将使用 cacerts。 如果使用 cacerts 存储库或未提供信任库，那么必须查看和编辑 cacerts 中的签署者列表以满足安全性需求。

您可以使用运行 IBM Key Management 实用程序的 WebSphere MQ 命令 strmqikm 来打开 cacerts 。 使用密码 changeit 将 cacerts 作为 JKS 文件打开。 修改密码以保护此文件的安全。

配置安全性类

使用 java.security 文件来注册其他安全性提供程序和其他缺省安全性属性。

许可权

使用 java.policy 文件来修改为资源授予的许可权。 javaws.policy 为 javaws.jar 授予许可权

加密强度

某些 JRE 提供了强度降低的加密。 如果您无法将密钥导入密钥库中，可能是由于加密强度降低引起的。 尝试使用 **strmqikm** 命令来启动 **ikeyman**，或者从 IBM Developer Kit 安全信息中下载强度较高、但是管辖区域受到限制的文件。

要点: 您的产地国对加密软件的进口、拥有、使用或再次出口到其他国家可能有一些限制。 在下载或使用不受限制的策略文件之前，必须针对您所在国家或地区的法律进行检查。 检查相应的规章以及对加密软件进行进口、拥有、使用和再次出口的相关政策，从而确定是否允许下载或使用这些文件。

修改信任提供程序以允许客户机连接至任何服务器

该示例说明了如何添加信任提供程序以及从 MQTT 客户机代码中引用此信任提供程序。 该示例对客户机或服务器不执行认证。 最终获得的 SSL 连接已加密，但是未进行认证。

第 119 页的图 25 中的代码片段将为 MQTT 客户机设置 AcceptAllProviders 信任提供程序和信任管理器。

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

图 *25: MQTT* 客户机代码段

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
    private static final long serialVersionUID = 1L;
    public AcceptAllProvider() {
        super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
        put("TrustManagerFactory.TrustAllCertificates",
                AcceptAllTrustManagerFactory.class.getName());
    }
```

图 *26: AcceptAllProvider.java*

```
    protected static class AcceptAllTrustManagerFactory extends
            javax.net.ssl.TrustManagerFactorySpi {
        public AcceptAllTrustManagerFactory() {}
        protected void engineInit(java.security.KeyStore keystore) {}
        protected void engineInit(
                javax.net.ssl.ManagerFactoryParameters parameters) {}
        protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
            return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
        }
```

图 *27: AcceptAllTrustManagerFactory.java*

```
    protected static class AcceptAllX509TrustManager implements
            javax.net.ssl.X509TrustManager {
        public void checkClientTrusted(
                java.security.cert.X509Certificate[] certificateChain,
                String authType) throws java.security.cert.CertificateException {
            report("Client authtype=" + authType);
            for (java.security.cert.X509Certificate certificate : certificateChain) {
                report("Accepting:" + certificate);
            }
        }
        public void checkServerTrusted(
                java.security.cert.X509Certificate[] certificateChain,
                String authType) throws java.security.cert.CertificateException {
            report("Server authtype=" + authType);
            for (java.security.cert.X509Certificate certificate : certificateChain) {
                report("Accepting:" + certificate);
            }
        }
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return new java.security.cert.X509Certificate[0];
        }
    private static void report(String string) {
           System.out.println(string);
      }
    }
```

图 *28: AcceptAllX509TrustManager.java*

遥测通道 JAAS 配置

配置 JAAS 以认证由客户机发送的**用户名**。

WebSphere MQ 管理员负责配置哪些 MQTT 通道使用 JAAS 对客户机进行认证。 指定每个要执行 JAAS 认证的通道的 JAAS 配置名称。 这些通道可以全部使用同一个 JAAS 配置，也可以使用不同的 JAAS 配置。 配置在 *WMQData directory*\qmgrs*qMgrName*\mqxr\jaas.config 中定义。

jaas.config 文件按 JAAS 配置名称进行组织。 在每个配置名称下是一个登录配置列表；请参阅 第 121 页的图 29。

JAAS 提供了四个标准登录模块。 标准 NT 和 UNIX 登录模块都是有限值。

JndiLoginModule
针对在 JNDI（Java 命名和目录接口）下配置的目录服务进行认证。

Krb5LoginModule
使用 Kerberos 协议进行认证。

NTLoginModule
使用当前用户的 NT 安全性信息来进行认证。

UnixLoginModule
使用当前用户的 UNIX 安全性信息来进行认证。

使用 NTLoginModule 或 UnixLoginModule 时存在的问题是：遥测 (MQXR) 服务使用 mqm 身份而不是 MQTT 通道的身份来运行。 mqm 是传递给 NTLoginModule 或 UnixLoginModule 以进行认证的身份，而不是客户机的身份。

要解决此问题，可编写您自己的登录模块，或者使用其他标准登录模块。 随 WebSphere MQ Telemetry 一起提供了一个样本 JAASLoginModule.java。 它是 javax.security.auth.spi.LoginModule 接口的实现。 使用它来开发您自己的认证方法。

您提供的任何新 LoginModule 类都必须在遥测 (MQXR) 服务的类路径上。 请不要将您的类放在类路径中的 WebSphere MQ 目录下。 创建您自己的目录，并定义遥测 (MQXR) 服务的整个类路径。

您可以通过在 service.env 文件中设置类路径来扩充遥测 (MQXR) 服务所使用的类路径。 CLASSPATH 必须使用大写字母，class path 语句只能包含文字。 不能在 CLASSPATH 中使用变量；例如，CLASSPATH=%CLASSPATH% 就不正确。 遥测 (MQXR) 服务设置其自己的类路径。 将 service.env 中所定义的 CLASSPATH 添加至此类路径。

遥测 (MQXR) 服务提供两个回调，这两个回调将返回与 MQTT 通道连接的客户机的 Username 和 Password。 **用户名** 和 **密码** 在 MqttConnectOptions 对象中设置。 请参阅第 122 页的图 30 以了解如何访问**用户名**和**密码**的示例。

示例

具有一个已命名的配置 MQXRConfig 的 JAAS 配置文件示例。

```
MQXRConfig {
  samples.JAASLoginModule required debug=true;
  //com.ibm.security.auth.module.NTLoginModule required;
  //com.ibm.security.auth.module.Krb5LoginModule required
  //                 principal=principal@your_realm
  //                 useDefaultCcache=TRUE
  //                 renewTGT=true;
  //com.sun.security.auth.module.NTLoginModule required;
  //com.sun.security.auth.module.UnixLoginModule required;
  //com.sun.security.auth.module.Krb5LoginModule required
  //                 useTicketCache="true"
  //                 ticketCache="${user.home}${/}tickets";
};
```

图 *29:* 样本 *jaas.config* 文件

已编写的 JAAS 登录模块示例，用于接收 MQTT 客户机所提供的**用户名**和**密码**。

```
public boolean login()
    throws javax.security.auth.login.LoginException {
  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);
  try {
    callbackHandler.handle(callbacks);
    String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
        .getName();
    char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
        .getPassword();
    // Accept everything.
    if (true) {
      loggedIn = true;
    } else
      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;
}
```

图 *30:* 样本 *JAASLoginModule.Login()* 方法

设备的 IBM WebSphere MQ Telemetry 守护程序概念

IBM WebSphere MQ 设备的遥测守护程序是高级 MQTT V3 客户机应用程序。 使用它来存储和转发来自其他 MQTT 客户机的消息。 它像 MQTT 客户机一样连接至 IBM WebSphere MQ，但是您还可以将它连接至其他 MQTT 客户机。

守护程序是发布/预订代理程序。 MQTT V3 客户机与其连接，以使用要发布的主题字符串和要预订的主题过滤器来发布和预订主题。 主题字符串采用分层形式，通过由 / 隔开的主题级别实现。 主题过滤器是主题字符串，可包含单一级别 + 通配符和多级别 # 通配符作为主题字符串的最后部分。

注: 守护程序中的通配符遵循 WebSphere Message Broker v6 限制性更强的规则。 IBM WebSphere MQ 与之不同。 它支持多个多级别通配符，通配符可代替主题字符串中任意位置、任意数量的层次结构级别。

多个 MQTT v3 客户机使用侦听器端口连接到守护程序。 缺省侦听器端口可修改。 您可以定义多个侦听器端口并向其分配不同的名称空间，请参阅第 128 页的『设备侦听器端口的 WebSphere MQ Telemetry 守护程序』。 守护程序本身就是 MQTT v3 客户机。 配置守护程序网桥连接，将该守护程序连接到另一守护程序的侦听器端口或 WebSphere MQ Telemetry (MQXR) 服务。

您可以为设备的 WebSphere MQ Telemetry 守护程序配置多个网桥。 使用网桥将可以交换发布的守护程序网连接起来。

每个网桥都能发布和预订其本地守护程序的主题。 此外，也能发布和预订所连接的其他守护程序、WebSphere MQ 发布/预订代理程序或其他任何 MQTT v3 代理程序的主题。 利用主题过滤器，您可以选择要从一个代理程序传播到另一个代理程序的发布。 您可以向任一方向传播发布。 您可以将发布从本地守护程序传播到其连接的每个远程代理程序，或从所连接的任一代理程序传播到本地守护程序；请参阅第 122 页的『设备网桥的 IBM WebSphere MQ Telemetry 守护程序』。

设备网桥的 IBM WebSphere MQ Telemetry 守护程序

设备网桥的 IBM WebSphere MQ Telemetry 守护程序使用 MQTT v3 协议连接两个发布/预订代理程序。 该网桥可沿任一方向将发布从一个代理程序传播到另一代理程序。 设备网桥连接的一端是 WebSphere MQ Telemetry 守护程序，另一端可能是队列管理器或另一个守护程序。 队列管理器使用遥测通道连接到网桥连接。 守护程序使用守护程序侦听器连接到网桥连接。

设备的 IBM WebSphere MQ Telemetry 守护程序支持同时连接到一个或多个其他代理程序。 来自守护程序的连接称为网桥，通过守护程序配置文件中的连接条目进行定义。 使用 IBM WebSphere MQ 遥测通道建立到 IBM WebSphere MQ 的连接，如下图所示：

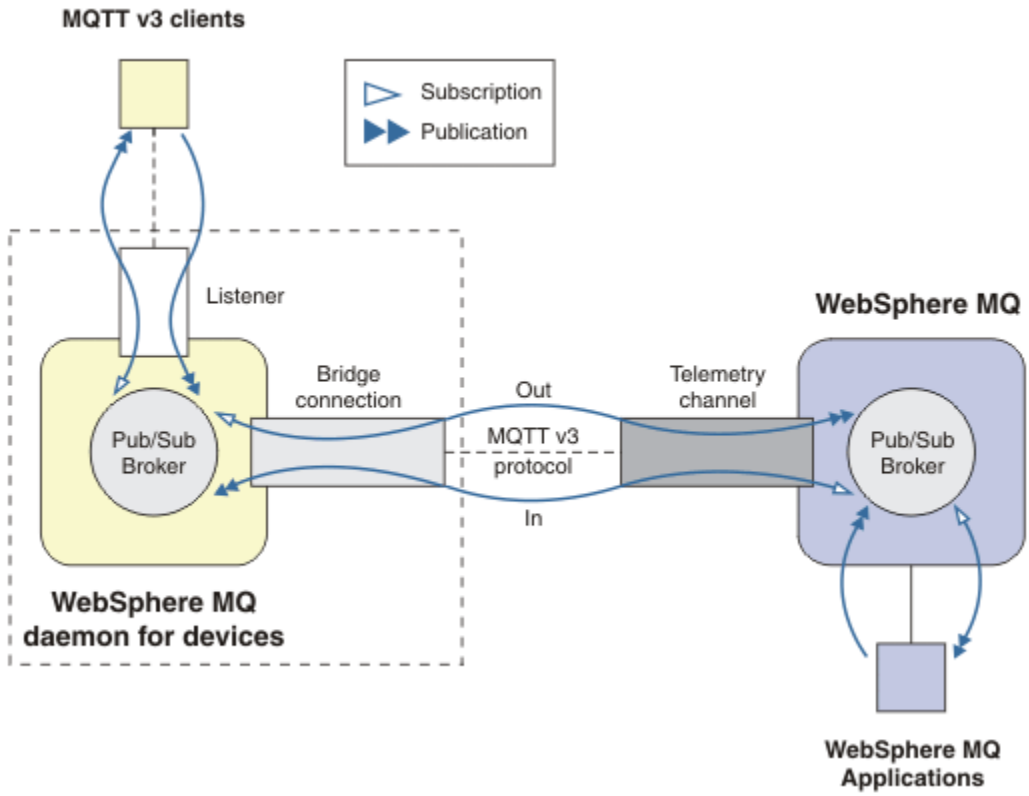


图 *31:* 将 *IBM WebSphere MQ Telemetry daemon for devices* 连接至 *IBM WebSphere MQ*

网桥将守护程序连接到另一个充当 MQTT v3 客户机的代理程序。 网桥参数将镜像 MQTT v3 客户机的属性。

网桥不仅仅是一个连接。 它还充当两个发布/预订代理程序之间的发布和预订代理。 本地代理程序是设备的 IBM WebSphere MQ Telemetry 守护程序，远程代理程序是支持 MQTT V3 协议的任何发布/预订代理程序。 通常，远程代理程序是另一个守护程序或 IBM WebSphere MQ。

网桥的工作是在两个代理程序之间传播发布内容。 网桥是双向的。 它采用任一方向传播发布内容。 第 123 页的图 31 说明了网桥将设备的 IBM WebSphere MQ Telemetry 守护程序连接至 IBM WebSphere MQ 的方式。 第 124 页的『网桥的主题设置示例』使用示例阐明如何使用主题参数配置网桥。

第 123 页的图 31 中的 In 和 Out 箭头指示网桥的双向性。 在箭头的一端创建预订。 在箭头的另一端，将与预订匹配的发布内容发布到代理程序。 根据发布内容的流向对箭头进行标记。 发布内容将流入 (In) 守护程序，然后从守护程序流出 (Out)。 标签的重要性是在命令语法中使用。 请记住，In 和 Out 指发布内容的流向，而非预订发送方向。

可以将其他客户机、应用程序或代理程序连接到 IBM WebSphere MQ 或设备的 WebSphere MQ Telemetry 守护程序。 它们在所连接到的代理程序发布和预订主题。 如果代理程序是 IBM WebSphere MQ，那么可以集群和分发主题，而不在本地队列管理器显式定义。

使用网桥

使用网桥连接和侦听器将守护程序连接在一起。 使用网桥连接和遥测通道将守护程序和队列管理器连接在一起。 将多个代理程序连接在一起时，可以创建环路。 请注意：发布内容可能会围绕未检测到的代理程序环路无止境地循环。

使用桥接到 IBM WebSphere MQ 的守护程序的某些原因如下所示：

减少到 WebSphere MQ 的 MQTT 客户机连接数
通过使用守护程序层次结构，您可以将许多客户机连接到 WebSphere MQ；可以一次连接超过单个队列管理器可以连接数量的客户机。

在 MQTT 客户机与 WebSphere MQ 之间存储转发消息
您可以使用存储和转发操作，避免在客户机与 IBM WebSphere MQ 之间保持持续连接（如果客户机没有自己的存储器）。 您可以在 MQTT 客户机与 WebSphere MQ 之间使用多种类型的连接；请参阅用于监视与控制的遥测概念和方案。

对 MQTT 客户机与 WebSphere MQ 之间交换的发布内容进行过滤
通常，会将发布内容分为在本地处理的消息和涉及其他应用程序的消息。 本地发布内容可能包含传感器和传动结构之间的控制流，远程发布内容包含读数、状态和配置命令的请求。

更改发布内容的主题空间
避免来自与不同侦听器端口连接的客户机的主题字符串彼此冲突。 本示例使用守护程序对来自不同建筑物的计量表读数进行标记；请参阅分隔不同客户机组的主题空间。

网桥的主题设置示例

将所有内容发布至远程代理程序 - 使用缺省值

缺省方向称为 out，网桥将主题发布至远程代理程序。 topic 参数控制使用主题过滤器传播哪些主题。

网桥使用 第 124 页的图 32 中的 topic 参数来预订 MQTT 客户机或其他代理向本地守护程序发布的所有内容。 网桥将主题发布至通过网桥连接的远程代理程序。

```
connection Daemon1
topic #
```

图 *32:* 将所有内容发布至远程代理程序

将所有内容发布至远程代理程序 - 显式

以下代码段中的 topic 设置提供了与使用缺省值相同的结果。 唯一的差别是，**direction** 参数是显式的。 使用 `out` 方向预订本地代理程序、守护程序并发布到远程代理程序。 在远程代理程序发布网桥预订的本地守护程序上创建的发布内容。

```
connection Daemon1
topic # out
```

图 *33:* 将所有内容发布至远程代理程序 - 显式

将所有内容发布至本地代理程序

代替使用方向 `out`，您可以设置相反方向 `in`。 以下代码段配置网桥以预订该网桥连接的远程代理程序发布的所有内容。 网桥将主题发布至本地代理程序、守护程序。

```
connection Daemon1
topic # in
```

图 *34:* 将所有内容发布至本地代理程序

将所有内容从本地代理程序的导出主题发布至远程代理程序的导入主题

使用两个额外的主题参数 **local_prefix** 和 **remote_prefix** 来修改主题过滤器，前面几个示例中的 `#`。 一个参数用于修改要在预订中使用的主题过滤器，另一个参数用于修改要将发布内容发布至的主题。 效果是将一个代理程序中使用的主题字符串的开头替换为另一个代理程序上的其他主题字符串。

根据主题命令的方向，**local_prefix** 和 **remote_prefix** 的含义会相反。 如果方向是 `out`，缺省情况下，**local_prefix** 将用作主题预订的一部分，**remote_prefix** 将替换远程发布内容中主题字符串

的 **local_prefix** 部分。 如果方向是 in，**remote_prefix** 将变成远程预订的一部分，**local_prefix** 将替换主题字符串的 **remote_prefix** 部分。

主题字符串的第一部分通常被认为是定义主题空间。 使用额外的参数来更改主题将发布至的主题空间。 您可以执行此操作以避免传播的主题与目标代理程序上的另一个主题产生冲突，或者用于除去安装点主题字符串。

例如，在以下代码段中，到守护程序的主题字符串 export/# 的所有发布内容将重新发布至远程代理程序的 import/#。

```
topic # out export/ import/
```

图 *35:* 将所有内容从本地代理程序的导出主题发布至远程代理程序的导入主题

将所有内容从远程代理程序的导出主题发布至本地代理程序的导入主题

以下代码段显示了反向配置；网桥预订通过远程代理程序的 export/# 主题字符串发布的所有内容，并将其发布至本地代理程序的 import/#。

```
connection Daemon1
topic # in import/ export/
```

图 *36:* 将所有内容从远程代理程序的导出主题发布至本地代理程序的导入主题

将所有内容从 1884/ 安装点发布至具有原始主题字符串的远程代理程序

在以下代码段中，网桥预订通过与本地守护程序的安装点 1884/ 连接的客户机发布的所有内容。 网桥将发布到该安装点的所有内容发布至远程代理程序。 将从发布至远程代理程序的主题中除去安装点字符串 1884/。 *local_prefix* 与安装点字符串 1884/ 相同，*remote_prefix* 是空字符串。

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

图 *37:* 将所有内容从 *1884/* 安装点发布至具有原始主题字符串的远程代理程序。

分隔与不同守护程序连接的不同客户机的主题空间

将为电力计量表编写应用程序，以发布建筑物的计量表读数。 使用 MQTT 客户机将读数发布至托管在同一个建筑物中的守护程序。 为发布内容选择的主题是 power。 将同一个应用程序部署到复合体中的许多建筑物。 对于站点监视和数据存储，将使用网桥连接聚集来自所有建筑物的读数。 这些连接会将建筑物守护程序链接到中央位置的 WebSphere MQ。

每个建筑物中的客户机应用程序都相同，但必须通过建筑物来区分数据。 每个读数都有一个 power 主题，并且必须以构建号作为前缀以进行区分。 复合体中第一个建筑物中的网桥使用前缀 meters/building01/，第二个建筑物中的网桥使用前缀 meters/building02/。 其他建筑物中的读数遵循相同的模式。 WebSphere MQ 接收具有 meters/building01/power 之类主题的读数。

此示例为 contrived; 在实践中，应用程序发布到的主题空间可能是可配置的。

每个守护程序的配置文件具有遵循以下代码段中模式的主题语句：

```
connection Daemon1
topic power out "" meters/building01/
```

图 *38:* 分隔与不同守护程序连接的客户机的主题空间

指定空字符串作为未使用的 local_prefix 参数的占位符。

分隔与相同守护程序连接的客户机的主题空间

假设单个守护程序用于连接所有电表。 假设在应用程序中可以配置为连接到不同的端口，您可以将不同建筑物中的电表连接到不同的侦听器端口来区分建筑物，如以下代码段所示。 本示例同样是人为的；它阐明了可以如何使用安装点。

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

图 *39:* 分隔与相同守护程序连接的客户机的主题空间

重新映射双向流动的发布内容的不同主题

在以下代码片段中的配置中，网桥在远程代理上预订单个主题 b ，并将有关 b 的发布转发到本地守护程序，将主题更改为 a。 网桥还会在本地代理上预订单个主题 x ，并将有关 x 的出版物转发到远程代理，从而将主题更改为 y。

```
connection Daemon1
topic "" in a b
topic "" out x y
```

图 *40:* 重新映射双向流动的发布内容的不同主题

关于本示例的重要一点是，在两个代理程序处预订和发布不同的主题。 两个代理程序的主题空间互不相交。

重新映射双向流动的发布内容的相同主题（循环）

与上一个示例不同，第 126 页的图 41 中的配置通常会产生一个环路。 在主题语句 `topic "" in a b` 中，网桥远程预订 b 并在本地发布至 a。 在另一个主题语句中，网桥在本地预订 a 并远程发布至 b。 可以如第 126 页的图 42 中所示编写同一配置。

一般结果是，如果客户机远程发布到 b ，那么该出版物将作为主题 a 上的出版物传输到本地守护程序。 但是，当网桥发布到主题 a 上的本地守护程序时，该发布与网桥对本地主题 a 进行的预订相匹配。 预订为 `topic "" out a b`。 因此，该出版物将作为主题 b 上的出版物传输回远程代理。 现在，网桥已预订远程主题 b，并且循环将再次开始。

一些代理程序实施环路检测以防止发生环路。 但当不同类型的代理程序桥接在一起时，环路检测机制必须工作。 如果将 WebSphere MQ 桥接到设备的 WebSphere MQ Telemetry 守护程序，环路检测将不工作。 如果将设备的两个 IBM WebSphere MQ Telemetry 守护程序桥接在一起，环路检测将起作用。 缺省情况下将开启环路检测；请参阅 try_private。

```
connection Daemon1
topic "" in a b
topic "" out a b
```

图 *41: !*重新映射双向流动的出版物的相同主题

```
connection Daemon1
topic "" both a b
```

图 *42: !*使用 *both* 重新映射双向流动的出版物的相同主题。

第 126 页的图 40 中的配置与第 126 页的图 41 的相同。

IBM WebSphere MQ Telemetry daemon for devices 网桥连接的可用性

配置多个 IBM WebSphere MQ Telemetry daemon for devices 网桥连接地址以连接至第一个可用的远程代理程序。 如果此代理程序是一个多实例队列管理器，请提供该队列管理器的两个 TCP/IP 地址。 配置主连接以连接或重新连接到主服务器（可用时）。

连接网桥参数 addresses 是 TCP/IP 套接字地址列表。 网桥尝试轮流连接到每个地址，直到成功连接为止。 round_robin 和 start_type 连接参数可控制成功连接后如何使用地址。

如果 start_type 是 auto、manual 或 lazy，那么当连接失败时，网桥会尝试重新进行连接。 它会轮流使用每个地址，每次尝试连接间约有 20 秒钟延时。 如果 `start_type` 是 once，那么当连接失败时，网桥不会尝试自动进行重新连接。

如果 round_robin 是 true，那么网桥连接会尝试在列表的第一个地址处启动，并轮流尝试列表中的每个地址。 当遍历列表后，它会重新在第一个地址处启动。 如果列表中仅有一个地址，它将每 20 秒重新尝试一次。

如果 round_robin 是 false，那么将优先考虑列表中的第一个地址（称为“主服务器”）。 如果第一次尝试连接到主服务器失败，那么网桥会继续尝试在后台重新连接主服务器。 同时，网桥也会使用列表中的其他地址尝试进行连接。 如果后台尝试连接主服务器获得成功，网桥将从当前连接断开并切换到主服务器连接。

如果连接自动断开（如通过发出 **`connection_stop`** 命令），那么当重新启动连接时，它将重新尝试使用同一地址。 如果由于连接故障或远程代理程序断开连接导致连接断开，网桥将等待 20 秒钟的时间。 然后，它会尝试连接到列表中的下一个地址或相同的地址（如列表中仅有一个地址）。

连接到多实例队列管理器

在多实例队列管理器配置中，队列管理器会在具有不同的 IP 地址的两个不同的服务器上运行。 通常，不使用特定的 IP 地址配置遥测通道。 仅使用端口号进行配置。 启动遥测通道时，缺省情况下它会选择本地服务器上第一个可用的网络地址。

利用队列管理器所使用的两个 IP 地址来配置网桥连接的 addresses 参数。 将 round_robin 设置为 true。

如果活动队列管理器实例发生故障，那么队列管理器会切换到备用实例。 守护程序会检测与活动实例的连接是否中断，并尝试重新连接到备用实例。 它会使用为网桥连接配置的地址列表中的其他 IP 地址。

网桥连接到的队列管理器仍是同一队列管理器。 该队列管理器可恢复自身状态。 如果将 cleansession 设置为 false，那么网桥连接会话将恢复到故障转移前的同一状态。 连接会在一段延迟后恢复。 具有 “至少一次” 或 “最多一次” 服务质量的消息不会丢失，并且预订将继续工作。

重新连接时间取决于备用实例启动时重新启动的通道和客户机数以及使用的消息数。 重新建立连接前，网桥连接可能会多次尝试重新连接两个 IP 地址。

请勿使用特定的 IP 地址配置多实例队列管理器遥测通道。 IP 地址仅在一台服务器上有效。

如果您正在使用可管理 IP 地址的其他高可用性解决方案，那么可使用特定的 IP 地址配置遥测通道。

cleansession

网桥连接是 MQTT v3 客户机会话。 您可以控制连接是否启动新会话或恢复现有会话。 如果恢复现有会话，那么网桥连接会保留上一次会话的预订和保留的发布。

如果 地址 列出了多个 IP 地址，并且这些 IP 地址连接到由不同队列管理器托管的遥测通道或连接到不同的遥测守护程序，请不要将 cleansession 设置为 false。 会话状态不会在队列管理器或守护程序间进行传输。 尝试在不同的队列管理器或正在启动的新的会话中的守护程序结果中重新启动现有会话。 存在疑问的消息将丢失，且预订过程可能不会按预期进行。

notifications

应用程序可跟踪是否使用通知来运行网桥连接。 通知是值为 `1`（已连接）或 `0`（断开连接）的发布。 它将发布到 `notification_topic` 参数定义的 *topicString* 。 *topicString* 的缺省值为 `$SYS/broker/connection/` *clientIdentifier*`/state`。 缺省 *topicString* 包含前缀 $SYS。 通过定义以 $SYS 开头的主题过

滤器来预订以 $SYS 开头的主题。 主题过滤器 # 将预订全部主题，但不会预订守护程序上以 $SYS 开头的主题。 将 $SYS 视为定义与应用程序主题空间不同的特殊系统主题空间。

通知 允许 IBM WebSphere MQ Telemetry daemon for devices 在网桥连接或断开连接时通知 MQTT 客户机。

keepalive_interval
keepalive_interval 网桥连接参数可设置网桥发送 TCP/IP ping 到远程服务器的时间间隔。 缺省时间间隔为 60 秒。 ping 可防止检测连接静止期的远程服务器或防火墙关闭 TCP/IP 会话。

CLIENTID
网桥连接是 MQTT v3 客户机会话，并具有由网桥连接参数 clientid 设置的 `clientIdentifier`。 如果要将 cleansession 参数设置为 false，重新进行连接以恢复先前的会话，那么每个会话中使用的 `clientIdentifier` 必须相同。 clientid 的缺省值 *hostname.connectionName*，与之前相同。

安装、验证、配置和控制设备的 WebSphere MQ Telemetry 守护程序

安装、配置和控制该守护程序均基于文件完成。

通过将 Software Development Kit 复制到要运行守护程序 的设备来安装守护程序。

例如，运行 MQTT 客户机实用程序并作为发布/预订代理连接到设备的 WebSphere MQ Telemetry 守护程序；请参阅 将消息发布到特定 MQTT v3 客户机 。

通过创建配置文件来配置守护程序；请参阅设备 WebSphere MQ Telemetry 守护程序的配置文件。

通过在文件 `amqtdd.upd` 中创建命令来控制正在运行的守护程序。 守护程序会每 5 秒钟阅读一次文件、运行一次命令并删除一次文件；请参阅设备 WebSphere MQ Telemetry 守护程序的命令文件。

设备侦听器端口的 WebSphere MQ Telemetry 守护程序

使用侦听器端口将 MQTT V3 客户机连接到设备的 WebSphere MQ Telemetry 守护程序。 您可以通过安装点和最大连接数来限制侦听器端口。

侦听器端口必须与连接到该端口的客户机的 MQTT 客户机 `connect(serverURI)` 方法上指定的端口号相对应。 缺省情况下，在客户机和守护程序上为 1883。

您可以通过在守护程序配置文件中设置全局定义**端口**来更改守护程序缺省端口。 您可以通过将**侦听器**定义添加到守护程序配置文件来设置特定的端口。

对于缺省端口以外的每个侦听器端口，您都可以指定一个安装点来隔离客户机。 通过安装点连接到端口的客户机与其他客户机相互隔离；请参阅第 128 页的『设备安装点的 WebSphere MQ Telemetry 守护程序』。

您可以限制可连接到任一端口的客户机数。 设置全局定义 `max_connections` 以与缺省端口的连接，或通过 `max_connections` 限制每个侦听器端口。

示例

配置文件示例，此配置文件将缺省端口从 1883 更改为 1880，并将端口 1880 的连接数限制为 10000。 端口 1884 的连接数限制为 1000。 连接到端口 1884 的客户机与连接到其他端口的客户机相互隔离。

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

设备安装点的 WebSphere MQ Telemetry 守护程序

您可以将安装点与 MQTT 客户机所使用的侦听器端口相关联，以连接到设备的 WebSphere MQ Telemetry 守护程序。 安装点使用连接到另一侦听器端口的 MQTT 客户机的一个侦听器端口来隔离由 MQTT 客户机交换的发布和预订。

通过安装点连接到侦听器端口的客户机永远不能与连接到其他任何侦听器端口的客户机直接交换主题。 不通过安装点连接到侦听器端口的客户机可发布或预订任一客户机的主题。 客户机不了解是否通过安装点进行连接，它不会识别客户机创建的主题字符串。

安装点是发布和预订的主题字符串前面的文本字符串。 它位于通过安装点连接到侦听器端口的客户机创建的所有主题字符串之前。 文本字符串会从发送到连接侦听器端口的客户机的所有主题字符串中除去。

如果侦听器端口没有安装点，那么连接到该端口的客户机所创建和接收的发布和预订的主题字符串不会改变。

创建带 / 结尾的安装点字符串。 这样，该安装点将成为安装点主题树的父主题。

示例

配置文件包含以下侦听器端口：

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

连接到端口 1883 的客户机会创建对 MyTopic 的预订。 守护程序会将该预订注册为 1883/MyTopic。 连接到端口 1883 的其他客户机会发布有关 MyTopic 主题的消息。 守护程序会将主题字符串更改为 1883/MyTopic，并搜索匹配的预订。 端口 1883 上的订户通过原始主题字符串 MyTopic 接收发布。 守护程序从主题字符串中除去安装点前缀。

连接到端口 1884 的其他客户机同样发布有关 MyTopic 主题的消息。 这一次，守护程序会将主题注册为 1884/MyTopic。 端口 1883 上的订户不会接收发布，因为不同的安装点会生成具有不同的主题字符串的预订。

连接到端口 1885 的客户机将发布有关 1883/MyTopic 主题的消息。 守护程序不会更改主题字符串。 端口 1883 上的订户将接收对 MyTopic 的发布。

设备服务质量、持久预订和保留发布的 WebSphere MQ Telemetry 守护程序

服务质量设置仅适用于正在运行的守护程序。 如果某个守护程序停止运行，无论是以受控方式还是由于发生故障，所使用的消息的状态都将丢失。 如果该守护程序停止运行，那么将无法保证以“至少一次”或“至多一次”的方式提供消息。 设备的 WebSphere MQ Telemetry 守护程序支持有限的持久性。 设置 **retained_persistence** 配置参数，以便在关闭守护程序时保存保留的发布和预订。

与 WebSphere MQ 不同， 设备的 WebSphere MQ Telemetry 守护程序不会记录持久数据。 会话状态、消息状态及保留的发布不会得到事务性保存。 缺省情况下，守护程序在停止运行时会废弃所有数据。 您可以设置选项以定期检查预订和保留的发布。 守护程序停止运行时，总是丢失消息状态。 所有非保留的发布都将丢失。

将守护程序配置选项 Retained_persistence 设置为 true，以定期将保留的发布保存至文件。 守护程序重新启动时，将恢复上次自动保存的保留发布。 缺省情况下，当守护程序重新启动时，不会恢复客户机所创建的保留消息。

将守护程序配置选项 Retained_persistence 设置为 true，以定期将持久会话中创建的预订保存至文件。 如果 Retained_persistence 设置为 true，那么将复原客户机在将 CleanSession 设置为 false(“持久会话”) 的会话中创建的预订。 守护程序会在重新启动时恢复预订，从而开始接收发布。 如果将 CleanSession 设置为 false，客户机会在重新启动时接收发布。 缺省情况下，当守护程序停止运行时，不会保存客户机会话状态，因此，即使客户机将 CleanSession 设置为 false，也不会恢复预订。

Retained_persistence 是自动保存机制。 它可能不会保存最新保留的发布或预订。 您可以更改保存保留发布和预订的频率。 使用配置选项 autosave_on_changes 和 autosave_interval 设置每次保存的时间间隔或更改次数。

设置持久性示例配置

```
# Sample configuration
# Daemon listens on port 1882 with persistence in /tmp
# Autosave every minute
```

```
port 1882
persistence_location /tmp/
retained_persistence true
autosave_on_changes false
autosave_interval 60
```

设备安全性的 WebSphere MQ Telemetry 守护程序

设备的 WebSphere MQ Telemetry 守护程序可对其连接的客户机进行认证，并使用凭证连接到其他代理程序以及控制对主题的访问。 通过使用不提供 SSL 支持的 WebSphere MQ Telemetry C 客户机来限制该守护程序提供的安全性。 因此，与该守护程序建立的连接不会加密，且无法使用证书进行认证。

缺省情况下，不开启任何安全性。

客户机的认证

MQTT 客户机可使用方法 `MqttConnectOptions.setUserName` 和 `MqttConnectOptions.setPassword` 设置用户名和密码。

根据密码文件中的条目检查客户机提供的用户名和密码，对连接到守护程序的客户机进行认证。 要启用认证，请创建密码文件并在守护程序配置文件中设置 `password_file` 参数；请参阅 password_file。

在守护程序配置文件中设置 `allow_anonymous` 参数，使未通过用户名或密码连接的客户机连接到检查认证情况的守护程序；请参阅 allow_anonymous。 设置 `password_file` 参数之后，如果某客户机未提供用户名或密码，那么将始终比照密码文件检查用户名和密码。

在守护程序配置文件中设置 `clientid_prefixes` 参数以限制与特定客户机的连接。 客户机必须具有以 `clientid_prefixes` 参数中列出的某个前缀开头的 `clientIdentifiers`；请参阅 clientid_prefixes。

网桥连接安全性

用于设备网桥连接的每个 WebSphere MQ Telemetry 守护程序都是一个 MQTT V3 客户机。 您可以为每个网桥连接设置用户名和密码，以作为守护程序配置文件中的网桥连接参数；请参阅用户名和密码。 网桥可针对代理程序进行自认证。

主题访问控制

如果客户机正在进行认证，那么守护程序还可为每个用户提供针对主题的控制访问功能。 根据客户机正在发布或预订的主题与访问控制文件中的访问主题字符串的匹配情况，守护程序可授予访问控制功能；请参阅 acl_file。

访问控制表包括两部分。 第一部分可控制对所有客户机的访问，包括匿名客户机。 第二部分含密码文件中针对所有用户的内容部分。 它列出了每个用户的特定访问控制功能。

示例

以下示例显示了安全性参数。

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

图 *43:* 守护程序配置文件

```
Fred:Fredpassword
Barney:Barneypassword
```

图 *44:* 密码文件，*passwords.txt*

```
topic home/public/#
topic read meters/#
user Fred
topic write meters/fred
topic home/fred/#
user Barney
topic write meters/barney
topic home/barney/#
```

图 *45:* 访问控制文件，*acl.txt*

管理多点广播

使用此信息来了解 WebSphere MQ 多点广播管理任务，例如减少多点广播消息的大小和启用数据转换。

多点广播入门

使用此信息可开始使用 WebSphere MQ 多点广播主题和通信信息对象。

关于此任务

WebSphere MQ 多点广播消息传递使用网络通过将主题映射到组地址来传递消息。 以下任务是测试是否为多点广播消息传递正确配置了必需的 IP 地址和端口的快速方法。

为多点广播创建 COMMINFO 对象

通信信息 (COMMINFO) 对象包含与多点广播传输关联的属性。 有关 COMMINFO 对象参数的更多信息，请参阅 DEFINE COMMINFO。

使用以下命令行示例来定义用于多点广播的 COMMINFO 对象:

```
DEFINE COMMINFO(MC1) GRPADDR(group address) PORT(port number)
```

其中 *MC1* 是 COMMINFO 对象的名称， 组地址 是组多点广播 IP 地址或 DNS 名称， 端口号 是要传输的端口 (缺省值为 1414)。

将创建名为 *MC1* 的新 COMMINFO 对象; 此名称是在下一个示例中定义 TOPIC 对象时必须指定的名称。

为多点广播创建 TOPIC 对象

主题是发布/预订消息中发布的信息的主题，通过创建 TOPIC 对象来定义主题。 TOPIC 对象有两个参数，用于定义它们是否可以与多点广播配合使用。 这些参数为: **COMMINFO** 和 **MCAST**。

- **COMMINFO** 此参数指定多点广播通信信息对象的名称。 有关 COMMINFO 对象参数的更多信息，请参阅 DEFINE COMMINFO。
- **MCAST** 此参数指定在主题树中的此位置是否允许多点广播。

使用以下命令行示例来定义用于多点广播的 TOPIC 对象:

```
DEFINE TOPIC(ALLSPORTS) TOPICSTR('Sports') COMMINFO(MC1) MCAST(ENABLED)
```

将创建名为 *ALLSPORTS* 的新 TOPIC 对象。 它具有主题字符串 *Sports*，其相关通信信息对象称为 *MC1* (这是您在先前示例中定义 COMMINFO 对象时指定的名称) ，并且已启用多点广播。

测试多点广播发布/预订

创建 TOPIC 和 COMMINFO 对象后，可以使用 amqspubc 样本和 amqssubc 样本对其进行测试。 有关这些样本的更多信息，请参阅 发布/预订样本程序。

1. 打开两个命令行窗口; 第一个命令行用于 amqspubc 发布样本，第二个命令行用于 amqssubc 预订样本。
2. 在命令行 1 上输入以下命令:

   ```
   amqspubc Sports QM1
   ```

   其中 *Sports* 是先前示例中定义的 TOPIC 对象的主题字符串， *QM1* 是队列管理器的名称。
3. 在命令行 2 上输入以下命令:

   ```
   amqssubc Sports QM1
   ```

   其中 *Sports* 和 *QM1* 与步骤 第 132 页的『2』 中使用的相同。
4. 在命令行 1 输入 Hello world。 如果正确配置了 COMMINFO 对象中指定的端口和 IP 地址; 那么 amqssubc 样本 (在端口上侦听来自指定地址的发布) 在命令行 2 上输出 Hello world。

IBM WebSphere MQ 多点广播主题拓扑

使用此示例来了解 IBM WebSphere MQ 多点广播主题拓扑。

IBM WebSphere MQ 多点广播支持要求每个子树在总层次结构中都有自己的多点广播组和数据流。

有类网络 IP 寻址方案针对多点广播地址指定了地址空间。 完整的多点广播 IP 地址范围是 224.0.0.0 到 239.255.255.255，但其中一些地址是保留的地址。 有关保留地址的列表，请与系统管理员联系，或者参阅 IPv4 多点广播地址空间注册表 以获取更多信息。 建议您使用 239.0.0.0 到 239.255.255.255 之间的本地作用域多点广播地址。

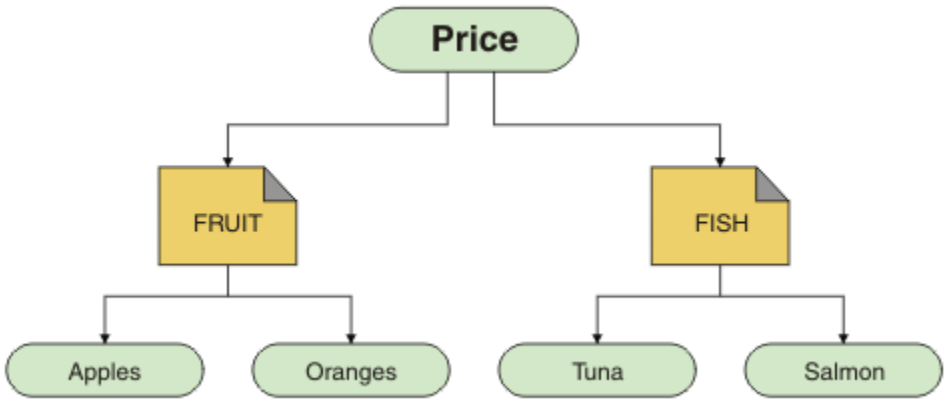
在下图中，有两个可能的多点广播数据流:

```
DEF COMMINFO(MC1) GRPADDR(239.XXX.XXX.XXX
)

DEF COMMINFO(MC2) GRPADDR(239.YYY.YYY.YYY)
```

其中 *239.XXX.XXX.XXX* 和 *239.YYY.YYY.YYY* 是有效的多点广播地址。

这些主题定义用于创建主题树，如下图所示:

```
DEFINE TOPIC(FRUIT) TOPICSTRING('Price/FRUIT') MCAST(ENABLED) COMMINFO(MC1)

DEFINE TOPIC(FISH) TOPICSTRING('Price/FISH') MCAST(ENABLED) COMMINFO(MC2)
```



每个多点广播通信信息 (COMMINFO) 对象都表示一条不同的数据流，因为其组地址不同。 在此示例中，FRUIT 主题定义为使用 COMMINFO 对象 MC1， FISH 主题定义为使用 COMMINFO 对象 MC2，并且 Price 节点没有多点广播定义。

WebSphere MQ 多点广播对主题字符串具有 255 个字符的限制。 此限制意味着必须注意树中节点和叶节点的名称；如果节点和叶节点的名称太长，主题字符串可能会超过 255 个字符，并返回 2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR 原因码。 建议尽可能保持主题字符串简短，因为较长的主题字符串可能对性能造成不利影响。

控制多点广播消息的大小

使用此信息来了解 WebSphere MQ 消息格式，并减小 WebSphere MQ 消息的大小。

WebSphere MQ 消息具有许多与它们相关联的属性，这些属性包含在消息描述符中。 对于小型消息，这些属性可能表示大部分数据流量，并可能对传输速率产生重大不利影响。 WebSphere MQ 多点广播使用户能够配置随消息一起传输的这些属性 (如果有)。

消息属性 (主题字符串除外) 的存在取决于 COMMINFO 对象是否声明必须发送这些属性。 如果未传输属性，那么接收应用程序将应用缺省值。 缺省 MQMD 值不一定与 MQMD_DEFAULT 值相同，在 第 133 页的表 9 中进行了描述。

COMMINFO 对象包含 MCPROP 属性，用于控制随消息一起流动的 MQMD 字段和用户属性的数量。 通过将此属性的值设置为适当的级别，可以控制 WebSphere MQ 多点广播消息的大小:

MCPROP
多点广播属性控制随消息一起流动的 MQMD 属性和用户属性数量。

ALL
传输 MQMD 的所有用户属性和所有字段。

REPLY
将仅传输用户属性和处理消息应答的 MQMD 字段。 这些属性包括:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

USER
将仅传输用户属性。

无
将不会传输任何用户属性或 MQMD 字段。

COMPAT
此值导致以兼容方式将消息传输到 RMM，这允许与当前 XMS 应用程序和 WebSphere Message Broker RMM 应用程序进行某些互操作。

多点广播消息属性

消息属性可以来自各种位置，例如 MQMD ， MQRFH2 中的字段以及消息属性。

下表显示了根据 MCPROP 值和未发送属性时使用的缺省值发送消息时发生的情况。

表 *9:* 消息传递属性及其与多点广播的相关方式

属性	使用多点广播时的操作	缺省值 (如果未传输)
TopicString	始终包含	不适用
MQMQ StrucId	未传输	不适用
MQMD 版本	未传输	不适用
报告	包含 (如果不是缺省值)	0
MsgType	包含 (如果不是缺省值)	MQMT_DATAGRAM

表 9: 消息传递属性及其与多点广播的相关方式 (继续)		
属性	**使用多点广播时的操作**	**缺省值 (如果未传输)**
到期	包含 (如果不是缺省值)	0
Feedback	包含 (如果不是缺省值)	0
编码	包含 (如果不是缺省值)	MQENC_NORMAL (单元)
CodedCharSetId	包含 (如果不是缺省值)	1208
格式	包含 (如果不是缺省值)	MQRFH2
优先级	包含 (如果不是缺省值)	4
持久	包含 (如果不是缺省值)	MQPER_NOT_PERSISTENT
MsgId	包含 (如果不是缺省值)	Null
CorrelId	包含 (如果不是缺省值)	Null
BackoutCount	包含 (如果不是缺省值)	0
ReplyToQ	包含 (如果不是缺省值)	Blank
ReplyToQMgr	包含 (如果不是缺省值)	Blank
UserIdentifier	包含 (如果不是缺省值)	Blank
AccountingToken	包含 (如果不是缺省值)	Null
PutAppIType	包含 (如果不是缺省值)	MQAT_JAVA
PutAppIName	包含 (如果不是缺省值)	Blank
PutDate	包含 (如果不是缺省值)	Blank
PutTime	包含 (如果不是缺省值)	Blank
ApplOriginData	包含 (如果不是缺省值)	Blank
GroupID	已排除	不适用
MsgSeqNumber	已排除	不适用
偏移量	已排除	不适用
MsgFlags	已排除	不适用
OriginalLength	已排除	不适用
UserProperties	已包含	不适用

相关参考

变更命令信息

定义命令信息

为多点广播消息传递启用数据转换

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。

在混合平台安装中，可能是大多数客户机需要的数据格式不是传输应用程序的本机格式。 在此情况下，可以使用多点广播 COMMINFO 对象的 **CCSID** 和 **ENCODING** 值来定义消息传输的编码以提高效率。

WebSphere MQ 多点广播支持以下内置格式的消息有效内容的数据转换:

- MQADMIN
- MQEVENT
- MQPCF
- MQRFH
- MQRFH2
- MQSTR

除了这些格式外，您还可以定义自己的格式并使用 MQDXP-数据转换出口参数 数据转换出口。

有关对数据转换进行编程的信息，请参阅 用于多点广播消息传递的 MQI 中的数据转换。

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。

多点广播应用程序监视

使用此信息可了解有关管理和监视 WebSphere MQ 多点广播的信息。

多点广播流量的当前发布者和订户的状态 (例如，发送和接收的消息数或丢失的消息数) 会定期从客户机传输到服务器。 接收到状态时， COMMINFO 对象的 COMMEV 属性指定队列管理器是否将事件消息放在 SYSTEM.ADMIN.PUBSUB.EVENT。 事件消息包含接收到的状态信息。 此信息是查找问题根源的宝贵诊断帮助。

使用 MQSC 命令 **DISPLAY CONN** 可显示有关连接到队列管理器的应用程序的连接信息。 有关 **DISPLAY CONN** 命令的更多信息，请参阅 DISPLAY CONN。

使用 MQSC 命令 **DISPLAY TPSTATUS** 可显示发布者和订户的 状态。 有关 **DISPLAY TPSTATUS** 命令的更多信息，请参阅 DISPLAY TPSTATUS。

COMMEV 和多点广播消息可靠性指示符

可靠性指示符与 COMMINFO 对象的 **COMMEV** 属性结合使用，是监视 WebSphere MQ 多点广播发布程序和订户的关键元素。 可靠性指示符 (在发布或预订状态命令上返回的 **MSGREL** 字段) 是一个 WebSphere MQ 指示符，用于说明没有错误的传输百分比。有时，由于传输错误 (反映在 **MSGREL** 的值中) ，必须重新传输消息。传输错误的潜在原因包括订户缓慢，网络繁忙和网络中断。 **COMMEV** 控制是否为使用 COMMINFO 对象创建的多点广播句柄生成事件消息，并设置为以下三个可能值之一:

DISABLED
不写入事件消息。

ENABLED
事件消息始终以 COMMINFO **MONINT** 参数中定义的频率进行写入。

异常
如果消息可靠性低于可靠性阈值，将写入事件消息。 90% 或更低的消息可靠性级别指示网络配置可能存在问题，或者一个或多个发布/预订应用程序运行速度太慢:

- 值 **MSGREL(100,100)** 指示在短期或长期时间范围内都没有问题。
- 值 **MSGREL(80,60)** 指示 20% 的消息当前存在问题，但这也是对长期值 60 的改进。

即使到队列管理器的单点广播连接中断，客户机也可能继续传输和接收多点广播流量，因此数据可能已过时。

多点广播消息可靠性

使用此信息可了解如何设置 WebSphere MQ 多点广播预订和消息历史记录。

克服多点广播传输故障的关键要素是 WebSphere MQ 对传输数据的缓冲 (要在链路的传输端保留的消息历史记录)。 此过程意味着在放入应用程序过程中不需要缓冲消息，因为 WebSphere MQ 提供了可靠性。 此历史记录的大小是通过通信信息 (COMMINFO) 对象配置的，如以下信息中所述。 更大的传输缓冲区意味着有更多的传输历史记录需要重新传输 (如果需要) ，但由于组播的性质，不能支持 100% 有保证的传送。

WebSphere MQ 多点广播消息历史记录在通信信息 (COMMINFO) 对象中由 **MSGHIST** 属性控制:

MSGHIST
此值是系统为处理 NACKs (否定应答) 情况下的重新传输而保留的消息历史记录量 (以千字节为单位)。

值 0 给出了最低的可靠性级别。 缺省值为 100 KB。

WebSphere MQ 多点广播新预订历史记录通过 **NSUBHIST** 属性在通信信息 (COMMINFO) 对象中进行控制:

NSUBHIST
新订户历史记录控制加入发布流的订户是接收当前可用的所有数据，还是仅接收预订以来进行的发布。

无
值 NONE 将导致发送设备仅传输从预订时间开始发布的内容。 NONE 是缺省值。

ALL
值 ALL 将导致发送设备重新发送已知的主题历史记录。 在某些情况下，这种情况会给保留的发布提供类似的行为。

注: 如果由于重新传输了所有主题历史记录，因此存在较大的主题历史记录，那么使用 ALL 值可能会对性能产生不利影响。

相关参考
定义命令信息
变更命令信息

高级多点广播任务

使用此信息来了解高级 WebSphere MQ 多点广播管理任务，例如，配置 .ini 文件以及与 WebSphere MQ LLM 的互操作性。

有关多点广播安装中的安全性注意事项，请参阅 多点广播安全性 。

多点广播和非多点广播发布/预订域之间的桥接

使用此信息来了解当非多点广播发布程序发布到已启用多点广播的 WebSphere MQ 主题时发生的情况。

如果非多点广播发布程序发布到定义为 **MCAST** enabled 和 **BRIDGE** enabled 的主题，那么队列管理器会将消息通过多点广播直接传输到可能正在侦听的任何订户。 多点广播发布程序无法发布到未启用多点广播的主题。

可以通过设置主题对象的 **MCAST** 和 **COMMINFO** 参数来启用现有主题多点广播。 有关这些参数的更多信息，请参阅 初始多点广播概念 。

COMMINFO 对象 **BRIDGE** 属性控制来自未使用多点广播的应用程序的发布。 如果 **BRIDGE** 设置为 ENABLED ，并且主题的 **MCAST** 参数也设置为 ENABLED，那么来自未使用多点广播的应用程序的出版物将桥接到执行此操作的应用程序。 有关 **BRIDGE** 参数的更多信息，请参阅 DEFINE COMMINFO。

为多点广播配置 .ini 文件

使用此信息可了解 .ini 文件中的 WebSphere MQ 多点广播字段。

可以在 ini 文件中进行其他 WebSphere MQ 多点广播配置。 必须使用的特定 ini 文件取决于应用程序类型:

- 客户机: 配置 *MQ_DATA_PATH*/mqclient.ini 文件。
- 队列管理器: 配置 *MQ_DATA_PATH*/qmgrs/*QMNAME*/qm.ini 文件。

其中， *MQ_DATA_PATH* 是 WebSphere MQ 数据目录 (/var/mqm/mqclient.ini) 的位置， *QMNAME* 是 .ini 文件所应用于的队列管理器的名称。

.ini 文件包含用于微调 WebSphere MQ 多点广播行为的字段:

```
Multicast:
  Protocol              = IP | UDP
  IPVersion             = IPV4 | IPV6 | ANY | BOTH
  LimitTransRate        = DISABLED | STATIC | DYNAMIC
```

```
TransRateLimit        = 100000
SocketTTL             = 1
Batch                 = NO
Loop                  = 1
Interface             = <IPaddress>
FeedbackMode          = ACK | NACK | WAIT1
HeartbeatTimeout      = 20000
HeartbeatInterval     = 2000
```

协议

UDP
在此方式下，将使用 UDP 协议发送包。 但是，网络元素无法像在 IP 方式下一样在多点广播分发中提供帮助。 包格式与 PGM 保持兼容。 这是缺省值。

IP
在此方式下，发送方发送原始 IP 包。 具有 PGM 支持的网络元素有助于可靠的多点广播包分发。 此方式与 PGM 标准完全兼容。

IPVersion

IPV4
仅使用 IPv4 协议进行通信。 这是缺省值。

IPV6
仅使用 IPv6 协议进行通信。

ANY
使用 IPv4 和/或 IPv6 进行通信，具体取决于可用的协议。

BOTH
支持使用 IPv4 和 IPv6 进行通信。

LimitTrans 速率

DISABLED
没有传输速率控制。 这是缺省值。

静态
实现静态传输速率控制。 发送方的传输速率不会超过 TransRateLimit 参数指定的速率。

动态
发射机根据从接收机获得的反馈来调整其传输速率。 在这种情况下，传输速率限制不能超过 TransRateLimit 参数指定的值。 发射机试图达到最佳传输速率。

TransRate 限制
传输速率限制 (以 Kbps 为单位)。

SocketTTL
SocketTTL 的值确定多点广播流量是可以通过路由器，还是可以通过路由器的数目。

批处理
控制是立即对消息进行批处理还是发送。有 2 个可能的值:

- *NO* 未对消息进行批处理，将立即发送这些消息。
- *YES* 将对消息进行批处理。

循环
将值设置为 1 以启用多点广播循环。 多点广播循环定义发送的数据是否回送到主机。

接口
多点广播流量在其上流动的接口的 IP 地址。 有关更多信息和故障诊断，请参阅: 在非多点广播网络上测试多点广播应用程序 和 为多点广播流量设置相应的网络

FeedbackMode

NACK
通过否定应答进行反馈。 这是缺省值。

ACK
肯定的反馈。

WAIT1
由肯定应答进行的反馈，其中发送方仅等待来自任何接收方的 1 ACK。

HeartbeatTimeout
脉动信号超时 (以毫秒计)。 值 0 指示主题的一个或多个接收方未引发脉动信号超时事件。 缺省值为 20000。

HeartbeatInterval
脉动信号间隔 (以毫秒为单位)。 值 0 指示未发送任何脉动信号。 脉动信号间隔必须远小于 **HeartbeatTimeout** 值，以避免发生错误的脉动信号超时事件。 缺省值为 2000。

与 WebSphere MQ 低等待时间消息传递的多点广播互操作性

使用此信息可了解 WebSphere MQ 多点广播与 WebSphere MQ Low Latency Messaging (LLM) 之间的互操作性。

对于使用 LLM 的应用程序，可以进行基本有效内容传输，而另一个应用程序使用多点广播来双向交换消息。虽然多点广播使用 LLM 技术，但 LLM 产品本身并不是嵌入式的。 因此，可以安装 LLM 和 WebSphere MQ 多点广播，并分别操作和维护这两个产品。

与多点广播通信的 LLM 应用程序可能需要发送和接收消息属性。 WebSphere MQ 消息属性和 MQMD 字段作为 LLM 消息属性与特定 LLM 消息属性代码进行传输，如下表中所示:

表 *10: WebSphere MQ* 消息属性到 *WebSphere MQ LLM* 属性映射

WebSphere MQ 属性	WebSphere MQ LLM 属性类型	LLM 属性类型	LLM 属性代码
MQMD.Report	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1001
MQMD.MsgType	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1002
MQMD.Expiry	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1003
MQMD.Feedback	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1004
MQMD.Encoding	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1005
MQMD.CodedCharSetId	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1006
MQMD.Format	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1007
MQMD.Priority	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1008
MQMD.Persistence	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1009
MQMD.MsgId	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_ByteArray	-1010
MQMD.BackoutCount	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1012
MQMD.ReplyToQ	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1013
MQMD.ReplyToQMger	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1014
MQMD.PutDate	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1020
MQMD.PutTime	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1021
MQMD.ApplOriginData	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1022
MQPubOptions	RMM_MSG_PROP_INT32	LLM_PROP_KIND_int32	-1053

有关 LLM 的更多信息，请参阅 LLM 产品文档: WebSphere MQ Low Latency Messaging。

管理 HP Integrity NonStop Server

使用此信息可了解 IBM WebSphere MQ Client for HP Integrity NonStop Server 的管理任务。

有两个管理任务可供您使用:

1. 从 Pathway 手动启动 TMF/Gateway。
2. 正在从 Pathway 停止 TMF/Gateway。

从 Pathway 手动启动 TMF/Gateway

您可以允许 Pathway 在第一个入伍请求时自动启动 TMF/Gateway ， 也可以从 Pathway 手动启动 TMF/Gateway。

过程

要从 Pathway 手动启动 TMF/Gateway ， 请输入以下 PATHCOM 命令:

```
START SERVER <server_class_name>
```

如果客户机应用程序在 TMF/网关完成对问题事务的恢复之前提出征调请求，此请求将最多暂挂 1 秒。 如果恢复没有在此时间内完成，征调将被拒绝。 然后，客户机从使用事务性 MQI 接收到 MQRC_UOW_ENLISTMENT_ERROR 错误。

从 Pathway 停止 TMF/Gateway

此任务描述如何从 Pathway 停止 TMF/Gateway ， 以及如何在停止 TMF/Gateway 后将其重新启动。

过程

1. 要防止向 TMF/Gateway 发出任何新的入伍请求，请输入以下命令:

   ```
   FREEZE SERVER <server_class_name>
   ```

2. 要触发 TMF/Gateway 以完成任何正在进行的操作并结束，请输入以下命令:

   ```
   STOP SERVER <server_class_name>
   ```

3. 要允许 TMF/Gateway 在首次登记时自动重新启动或手动重新启动，请执行以下步骤 1 和 2，请输入以下命令:

   ```
   THAW SERVER <server_class_name>
   ```

   将阻止应用程序发出新的入伍请求，并且在发出 **THAW** 命令之前无法发出 **START** 命令。

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，则由用户自行负责。

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：

知识产权许可法律和知识产权法 IBM Japan， Ltd. 19-21， Nihonbashi-Hakozakicho， Chuo-ku Tokyo 103-8510， 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。

本信息中对任何非 IBM Web 站点的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 Software Interoperability 协调人， Department 49XA 3605 Highway 52 N Rochester， MN 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: