9.4

IBM MQ 技术概述

注

在使用本资料及其支持的产品之前，请阅读第 271 页的『声明』中的信息。

内容

IBM MQ 技术概述

使用 IBM MQ 可连接应用程序并管理整个组织中的信息分发。

IBM MQ 使程序能够使用一致的应用程序编程接口在由不同组件 (处理器，操作系统，子系统和通信协议) 组成的网络中相互通信。 使用此接口设计和编写的应用程序称为消息排队应用程序。

使用以下子主题来了解消息排队以及 IBM MQ 提供的其他功能。

相关概念

IBM MQ 简介

在何处查找产品需求和支持信息

相关任务

规划 IBM MQ 体系结构

相关参考

第 6 页的『消息排队的主要功能和优点』

此信息突出显示了消息排队的一些功能和优点。 它描述了诸如消息排队的安全性和数据完整性之类的功能。

消息排队简介

IBM MQ 产品支持程序使用一致的应用程序编程接口在不同组件 (处理器，操作系统，子系统和通信协议) 的网络中相互通信。

使用此接口设计和编写的应用程序称为 消息排队 应用程序，因为它们使用 消息传递 和 排队 样式:

- 消息传递意味着程序通过在消息中相互发送数据而不是直接相互调用来进行通信。
- 排队意味着将消息放在存储器中的队列上，允许程序以不同的速度和时间，在不同的位置独立运行，并且没有它们之间的逻辑连接。

消息排队已在数据处理中使用多年。 现在最常用的是电子邮件。 无需排队，长途发送电子消息需要路由上的每个节点都可用于转发消息，并且收件人要登录并意识到您正在尝试向他们发送消息。 在排队系统中，消息存储在中间节点，直到系统准备好转发这些消息为止。 在他们的最终目的地，他们被存储在一个电子邮箱中，直到收件人准备好阅读它们。

即便如此，今天很多复杂的业务交易都是在不排队的情况下处理的。 在大型网络中，系统可能正在维护数千个处于即用即用状态的连接。 如果系统的一个部分迂到问题，那么系统的许多部分将变为不可用。

您可以将消息排队视为营销计划的电子邮件。 在消息排队环境中，构成应用程序套件的一部分的每个程序都会执行明确定义的自包含函数以响应特定请求。 要与另一个程序通信，程序必须将消息放在预定义的队列上。 另一个程序从队列中检索消息，并处理消息中包含的请求和信息。 所以消息排队是一种程序间通信的样式。

排队是在应用程序准备好处理消息之前保留消息的机制。 排队允许您:

- 在程序之间进行通信 (可能每个程序都在不同的环境中运行) ，而不必编写通信代码。
- 选择程序处理消息的顺序。
- 当消息数超过阈值时，通过安排多个程序为队列提供服务来平衡系统上的负载。
- 通过安排备用系统在主系统不可用时为队列提供服务，提高应用程序的可用性。

什么是消息队列?

消息队列 (简称为队列) 是可将消息发送至的指定目标。 消息会在队列上累积，直到服务这些队列的程序检索到这些消息为止。

队列驻留在队列管理器中并由其管理 (请参阅 第 8 页的『消息排队术语』)。 队列的物理性质取决于运行队列管理器的操作系统。 队列可以是计算机内存中的易失性缓冲区，也可以是永久存储设备 (例如磁盘) 上的数据集。 队列的物理管理由队列管理器负责，不会对参与的应用程序显示。

程序仅通过队列管理器的外部服务访问队列。 他们可以打开队列，将消息放在队列上，从中获取消息，然后关闭队列。 它们还可以设置和查询队列的属性。

消息排队的不同样式

点到点 (point-to-point)

一条消息放置在队列上，一个应用程序接收该消息。

在点到点消息传递中，发送应用程序必须先知道有关接收应用程序的信息，然后才能向该应用程序发送消息。 例如，发送应用程序可能需要知道要将信息发送到的队列的名称，还可能指定队列管理器名称。

发布/预订

由发布应用程序发布的每条消息的副本将传递到每个感兴趣的应用程序。 可能有许多，一个或没有感兴趣的应用程序。 在发布/预订中，感兴趣的应用程序称为订户，并且消息在由预订标识的队列上排队。

发布/预订消息传递允许您将信息提供者与该信息的使用者分离。 对于要发送和接收的信息，发送申请和接收申请不需要相互了解那么多。 有关更多信息，请参阅 第 51 页的『发布/预订消息传递』。

消息排队对应用程序设计者和开发者的好处

IBM MQ 允许应用程序使用 消息排队 来参与消息驱动的处理。 应用程序可以使用相应的消息排队软件产品在不同平台之间进行通信。 例如， z/OS 应用程序可以通过 IBM MQ for z/OS 进行通信。 应用程序与底层通信的机制相屏蔽。 消息排队的其他优点包括:

- 您可以使用可在许多应用程序之间共享的小程序来设计应用程序。
- 您可以通过复用这些构建块来快速构建新应用程序。
- 为使用消息排队技术而编写的应用程序不受队列管理器工作方式更改的影响。
- 您不需要使用任何通信协议。 队列管理器为您处理通信的所有方面。
- 在向其发送消息时，接收消息的程序无需运行。 这些消息保留在队列上。

设计人员可以降低其应用程序的成本，因为开发速度更快，需要的开发者更少，对编程技能的要求也比不使用消息排队的应用程序低。

IBM MQ 在应用程序运行的任何位置实现称为 消息队列接口 (或 MQI) 的公共应用程序编程接口。 这使您可以更轻松地将应用程序从一个平台移植到另一个平台。

有关 MQI 的详细信息，请参阅 消息队列接口概述。

消息排队的主要功能和优点

此信息突出显示了消息排队的一些功能和优点。 它描述了诸如消息排队的安全性和数据完整性之类的功能。

使用消息排队技术的应用程序的主要功能包括:

- 第 7 页的『程序之间没有直接连接』
- 第 7 页的『独立于时间的通信』
- 第 7 页的『小程序』
- 第 7 页的『消息驱动的处理』
- 第 7 页的『事件驱动的处理』
- 第 8 页的『消息优先级』
- 第 8 页的『安全性』
- 第 8 页的『数据完整性』
- 第 8 页的『恢复支持』

注: 在考虑 IBM MQ 客户机和服务器时，您不必更改服务器应用程序以支持新平台上的其他 IBM MQ MQI clients 。 同样， IBM MQ MQI client 可以在不进行更改的情况下使用其他类型的服务器。

程序之间没有直接连接

消息排队是一种用于间接程序间通信的技术。 它可以在程序相互通信的任何应用程序中使用。 通信由一个将消息放入队列 (由队列管理器拥有) 的程序和另一个从队列中获取消息的程序进行。

程序可以获取由其他程序放入队列中的消息。 其他程序可以连接到与接收程序相同的队列管理器，也可以连接到其他队列管理器。 此另一个队列管理器可能位于另一个系统上，另一个计算机系统上，甚至位于另一个业务或企业内。

使用消息队列进行通信的程序之间没有物理连接。 程序将消息发送到队列管理器拥有的队列，而另一个程序从队列中检索消息 (请参阅 第 7 页的图 1)。

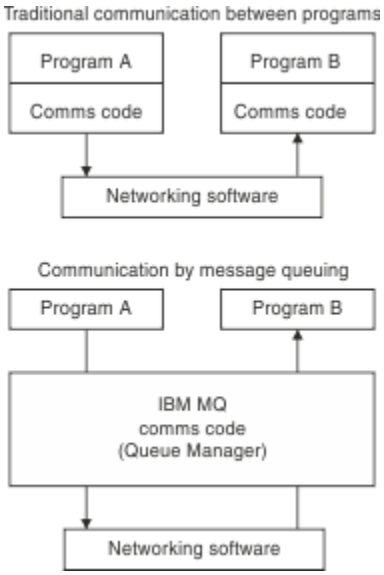


图 *1:* 与传统通信相比的消息排队

与电子邮件一样，作为事务一部分的个别消息通过商店上的网络传输。基础。 如果节点之间的链接失败，那么将保留该消息，直到恢复该链接，或者操作员或程序重定向该消息。

在程序中隐藏消息从队列移动到队列的机制。 因此，程序更简单。

独立于时间的通信

请求他人执行工作的程序不必等待对请求的应答。 他们可以执行其他工作，并在应答到达时或稍后处理该应答。 在编写消息传递应用程序时，当程序发送消息时，或者当目标能够接收消息时，您无需知道 (或关心)。 此消息不会丢失; 队列管理器将保留此消息，直到目标准备好对其进行处理为止。 消息将保留在队列中，直到程序将其除去。 这意味着发送和接收应用程序已解耦; 发送方可以继续处理，而无需等待接收方确认接收消息。 发送消息时，目标应用程序甚至不必正在运行。 它可以在消息启动后检索该消息。

小程序

消息排队允许您使用小型自包含程序的优点。 您可以将作业分布在几个较小的独立程序上，而不是单个大型程序按顺序执行作业的所有部分。 请求程序将消息发送到每个单独的程序，要求它们执行其功能; 当每个程序完成时，结果将作为一条或多条消息发送回。

消息驱动的处理

当消息到达队列时，它们可以使用 触发自动启动应用程序。 如果需要，可以在处理消息 (或消息) 时停止应用程序。

事件驱动的处理

可以根据队列的状态来控制程序。 例如，您可以安排程序在消息到达队列时立即启动，或者您可以指定程序在队列上具有高于特定优先级的 10 条消息或队列上具有任何优先级的 10 条消息之前不启动。

消息优先级

当程序将消息放入队列中时，它可以将优先级分配给消息。 这将确定添加新消息的队列中的位置。

程序可以按消息在队列中的顺序或通过获取特定消息从队列中获取消息。 (如果程序正在查找对其先前发送的请求的应答，那么该程序可能希望获取特定消息。)

安全性

提供了安全设施，包括当应用程序使用队列管理器时对其进行认证，当它们使用资源 (例如队列管理器上的队列) 时对其进行授权检查，以及当消息数据在网络上传输时对其进行加密，以及当消息数据驻留在队列上时对其进行加密。 有关安全性的更多信息，请参阅 安全性概述。

数据完整性

数据完整性由工作单元提供。 在每个 MQGET 或 MQPUT 上，完全支持将工作单元的开始和结束同步作为选项，从而允许落实或回滚工作单元的结果。 同步点支持在内部或外部针对 IBM MQ 运行，具体取决于为应用程序选择的同步点协调形式。

恢复支持

为了实现恢复，将记录所有持久 IBM MQ 更新。 如果需要恢复，那么将复原所有持久消息，回滚所有正在进行的事务，并以控制中的同步点管理器的正常方式处理任何同步点落实和回退。 有关持久消息的更多信息，请参阅 消息持久性。

消息排队术语

此信息提供了对消息排队中使用的一些术语的洞察。

它们包括:

- 通道
- 集群
- IBM MQ MQI client
- z/OS 组内排队
- 消息
- 消息通道代理程序
- 消息描述符
- 点到点
- 发布/预订
- 队列
- 队列管理器
- z/OS 队列共享组
- z/OS 共享队列
- 预订
- 主题

通道

通道用于将消息从一个队列管理器移动到另一个队列管理器，它们保护应用程序免受底层通信协议的保护。 队列管理器可能存在于同一系统上，也可能存在于同一平台上或不同平台上的不同系统上。 发送的消息可能源自许多位置:

- 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 MQ Explorer。
- 将检测事件消息发送到另一个队列管理器的队列管理器。
- 将远程管理命令发送到另一个队列管理器的队列管理器。 例如，使用 MQSC 命令或 administrative REST API。

有关通道的更多信息，请参阅 第 26 页的『通道定义』。

集群

集群 是以某种方式逻辑关联的队列管理器网络。

在不使用集群的分布式排队的 IBM 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，那么它必须已定义传输队列和到远程队列管理器的通道。

使用集群有两种不同的原因: 减少系统管理和提高可用性和工作负载均衡。

一旦建立了即使是最小的集群，您也会从简化的系统管理中获益。 属于集群的队列管理器需要较少的定义，因此减少了在定义中发生错误的风险。

有关集群的更多信息，请参阅 集群。

IBM MQ MQI client

IBM MQ MQI 客户机 是 IBM MQ 的独立可安装组件。 MQI 客户机允许您使用通信协议运行 IBM MQ 应用程序，与其他平台上的一个或多个消息队列接口 (MQI) 服务器进行交互并连接到其队列管理器。

有关如何安装和使用 IBM MQ MQI client 组件的完整详细信息，请参阅以下主题:

- AIX 在 AIX 上安装 IBM MQ 客户机
- Linux 在 Linux® 上安装 IBM MQ 客户机
- Windows 在 Windows 上安装 IBM MQ 客户机
- IBM i 在 IBM i 上安装 IBM MQ 客户机

和 配置服务器与客户机之间的连接。

组内排队

z/OS

队列共享组中的队列管理器可以使用正常通道进行通信，也可以使用称为 组内排队 (IGQ) 的技术，这使您能够在不定义通道的情况下执行快速消息传输。 这仅适用于 IBM MQ for z/OS。

有关组内排队的更多信息，请参阅 第 181 页的『Intra-group queuing』。

消息

在消息排队中，消息是由一个程序发送并用于另一个程序的数据集合。 请参阅 IBM MQ 消息。

有关消息类型的信息，请参阅 消息类型。

消息通道代理程序

消息通道代理程序是通道的一端。 一对消息通道代理 (一个发送和一个接收) 组成一个通道并将消息从一个队列管理器移动到另一个队列管理器。

有关如何使用消息通道代理程序的信息，请参阅 分布式队列管理简介。

消息描述符

IBM MQ 消息由控制信息和应用程序数据组成。

控制信息在消息描述符结构 (MQMD) 中定义，并包含如下内容:

- 消息类型
- 消息的标识
- 消息传递的优先级

应用程序数据的结构和内容由参与程序确定，而不是由 IBM MQ 确定。

有关更多信息，请参阅 MQMD。

点到点消息传递

在点到点消息传递中，每条消息从一个生产应用程序传输到一个消费应用程序。 通过将消息放入队列中的生产应用程序来传输消息，并且使用应用程序从该队列中获取消息。

发布/预订消息传递

在发布/预订消息传递中，由发布应用程序发布的每条消息的副本将传递到每个感兴趣的应用程序。 可能有许多，一个或不感兴趣的应用程序。 在发布/预订中，感兴趣的应用程序称为订户，并且消息在由预订标识的队列上排队。

有关更多信息，请参阅第 51 页的『发布/预订消息传递』。

队列

可将消息发送到的指定目标。 消息会在队列上累积，直到服务这些队列的程序检索到这些消息为止。

有关更多信息，请参阅第 16 页的『队列』。

队列管理器

队列管理器 是向应用程序提供排队服务的系统程序。

它提供了一个应用程序编程接口，以便程序可以将消息放入队列并从队列中获取消息。 队列管理器提供了其他功能，以便管理员可以创建新队列，更改现有队列的属性以及控制队列管理器的操作。

要使 IBM MQ 消息排队服务在系统上可用，必须有一个正在运行的队列管理器。 您可以在单个系统上运行多个队列管理器 (例如，将测试系统与 实时 系统分开)。 对于应用程序，每个队列管理器由 连接句柄 (*Hconn*) 标识。

许多不同的应用程序可以同时使用队列管理器的服务，而这些应用程序可能完全不相关。 要使程序使用队列管理器的服务，它必须建立与该队列管理器的连接。

要使应用程序向连接到其他队列管理器的应用程序发送消息，队列管理器必须能够在它们之间进行通信。 IBM MQ 实现 *store-and-forward* 协议，以确保在此类应用程序之间安全地传递消息。

有关更多信息，请参阅第 23 页的『队列管理器』。

队列共享组

z/OS

可以访问同一组共享队列的队列管理器构成一个称为 队列共享组 (QSG) 的组。 它们通过用于存储共享队列的耦合设施 (CF) 相互通信。 这仅适用于 IBM MQ for z/OS。

有关更多信息，请参阅第 137 页的『Shared queues and queue sharing groups』。

共享队列

z/OS

共享队列 是一种本地队列，其消息可由综合系统中的一个或多个队列管理器访问。 这与多个应用程序使用同一队列管理器共享的队列不同。 这仅适用于 IBM MQ for z/OS。

预订

发布/预订应用程序可以注册对有关特定主题的消息的兴趣。 当应用程序执行此操作时，它称为订户，术语预订定义匹配消息排队等待处理的方式。

预订包含有关订户的身份以及要将发布放置到的目标队列的身份的信息。 它还包含有关如何将发布放在目标队列上的信息。

有关更多信息，请参阅第 53 页的『订户和预订』。

Topic

主题是描述在发布/预订消息中所发布信息的主题的字符串。

主题是发布/预订系统中成功发送消息的关键。 发布者为每条消息分配一个主题，而不是在每条消息中包含一个特定目标地址。 队列管理器使该主题与一组已预订该主题的订户匹配，并将消息传递至其中每个订户。

有关更多信息，请参阅第 55 页的『主题』。

消息和队列

消息和队列是消息排队系统的基本组件。

什么是消息?

消息 是对使用它的应用程序有意义的字节字符串。 消息用于将信息从一个应用程序传输到另一个应用程序(或者在同一应用程序的不同部分之间)。 应用程序可以在同一平台上运行，也可以在不同平台上运行。

IBM MQ 消息由以下内容组成:

- 应用程序数据。应用程序数据的内容和结构由使用它的应用程序定义。
- 消息描述符。消息描述符标识消息并包含其他控制信息，例如消息类型以及发送应用程序分配给消息的优先级。

  消息描述符的格式由 IBM MQ 定义。 有关消息描述符的完整描述，请参阅 MQMD-消息描述符。
- 消息属性。 有关消息的元数据。 消息属性的内容由使用它们的应用程序定义。 有关更多信息，请参阅 消息属性。

消息长度

缺省最大消息长度为 4 MB ，尽管您可以将其增大到最大长度 100 MB (其中 1 MB 等于 1 048 576 字节)。 在实践中，消息长度可能受以下限制:

- 为接收队列定义的最大消息长度
- 为队列管理器定义的最大消息长度
- 队列定义的最大消息长度
- 发送或接收应用程序定义的最大消息长度
- 可用于消息的存储量

可能需要多条消息才能发送应用程序所需的所有信息。

应用程序如何发送和接收消息?

应用程序使用 **MQI 调用**发送和接收消息。

例如，要将消息放入队列，应用程序:

1. 通过发出 MQI MQOPEN 调用打开所需队列

2. 发出 MQI MQPUT 调用以将消息放入队列

另一个应用程序可以通过发出 MQI MQGET 调用从同一队列检索消息

有关 MQI 调用的更多信息，请参阅 MQI 调用。

什么是队列?

队列 是用于存储消息的数据结构。

每个队列都由 队列管理器拥有。 队列管理器负责维护它所拥有的队列，并负责将它接收到的所有消息存储到适当的队列中。 这些消息可由应用程序或队列管理器作为其正常操作的一部分放在队列上。

预定义队列和动态队列

可以通过创建队列的方式来描述这些队列:

- **预定义队列** 由管理员使用相应的 MQSC 或 PCF 命令创建。预定义队列是永久队列; 它们独立于使用它们的应用程序而存在，并且在 IBM MQ 重新启动后仍存在。
- **动态队列** 是在应用程序创建时创建的 发出 MQOPEN 请求，指定 模型队列的名称。 创建的队列基于称为模型队列的 模板队列定义。您可以使用 MQSC 命令 DEFINE QMODEL 来创建模型队列。 模型队列的属性 (例如，可存储在其上的最大消息数) 由从中创建的任何动态队列继承。

  模型队列具有一个属性，该属性指定动态队列是永久队列还是临时队列。 永久队列在应用程序和队列管理器重新启动后仍然存在; 临时队列在重新启动时丢失。

从队列中检索消息

适当授权的应用程序可以根据以下检索算法从队列中检索消息:

- 先进先出 (FIFO)。
- 消息优先级，如消息描述符中所定义。 将根据 FIFO 检索具有相同优先级的消息。
- 针对特定消息的程序请求。

来自应用程序的 MQGET 请求确定所使用的方法。

IBM MQ 对象

队列管理器定义 IBM MQ 对象的属性。 这些属性的值会影响 IBM MQ 处理这些对象的方式。 您可以使用 IBM MQ 命令和界面来创建和管理对象。 在应用程序中，使用消息队列接口 (MQI) 来控制对象。 从程序寻址时，对象由 IBM MQ 对象描述符 (MQOD) 标识。

对象管理

对象管理包括以下任务:

- 启动和停止队列管理器。
- 为应用程序创建对象 (尤其是队列)。
- 显示或改变对象的属性。
- 正在删除对象。
- 使用通道来创建到其他 (远程) 系统上的队列管理器的通信路径。
- 创建队列管理器的 集群 以简化整体管理过程并平衡工作负载。

除了动态队列之外，必须先向队列管理器定义对象，然后才能使用这些对象。

使用 IBM MQ 命令执行对象管理操作时，队列管理器将检查您是否具有执行该操作所需的权限级别。 同样，当应用程序使用 MQOPEN 调用打开对象时，队列管理器将检查应用程序是否具有所需的权限级别，然后才能访问此对象。 针对要打开的对象的名称执行检查。

您可以使用以下方法来定义和管理对象:

- 可编程命令格式参考 和 自动化管理任务 中描述的 PCF 命令
- MQSC 命令 中描述的 MQSC 命令
- z/OS IBM MQ for z/OS 操作和控制面板，如 管理 IBM MQ for z/OS 中所述
- Windows Linux IBM MQ Explorer (仅适用于 Intel 系统的 Windows 和 Linux)。 有关更多信息，请参阅 MQ Explorer 简介。

您还可以使用以下方法来管理对象:

- 从键盘输入的控制命令。 请参阅 使用控制命令管理 IBM MQ for Multiplatforms。
- IBM MQ 程序中的管理接口 (MQAI) 调用。 请参阅 IBM MQ 管理接口 (MQAI)。

ALW 对于 AIX, Linux, and Windows 上的 IBM MQ 命令序列，可以使用 MQSC 工具来运行文件中保存的一系列命令。 有关更多信息，请参阅 使用 MQSC 命令管理 IBM MQ。

IBM i 对于定期使用的 IBM MQ for IBM i 命令序列，可以编写 CL 程序。 有关更多信息，请参阅 使用 CL 命令管理 IBM MQ for IBM i。

z/OS 对于您定期使用的 IBM MQ for z/OS 命令序列，您可以编写用于创建包含命令的消息并将这些消息放入系统命令输入队列的管理程序。 队列管理器处理此队列上的消息的方式与处理从命令行或操作和控制面板中输入的命令的方式相同。 此技术在 编写程序以管理 IBM MQ 中进行了描述，并在随 IBM MQ for z/OS 提供的 Mail Manager 样本应用程序中进行了演示。 有关此样本的描述，请参阅 IBM MQ for z/OS 的样本程序 。

对象属性

对象的属性由其属性定义。 某些可以指定，其他只能查看。

例如，队列可容纳的最大消息长度由其 **MaxMsgLength** 属性定义; 您可以在创建队列时指定此属性。 **DefinitionType** 属性指定如何创建队列; 您只能显示此属性。

在 IBM MQ 中，有两种引用属性的方法:

- 使用其 PCF 名称，例如 **MaxMsgLength**。
- 使用其 MQSC 命令名，例如 MAXMSGL。

队列共享组

z/OS

可以访问同一组共享队列的队列管理器构成一个称为 队列共享组 (QSG) 的组，它们使用存储共享队列的耦合设施 (CF) 相互通信。 请注意， QSG 并非严格意义上的对象。

共享队列是具有可由队列共享组中的一个或多个队列管理器访问的消息的本地队列类型。 这与多个应用程序使用同一队列管理器共享的队列不同。

队列共享组具有最多为四个字符的名称。 该名称在网络中必须是唯一的，并且必须与所有队列管理器名称不同。

要点: 共享队列和队列共享组仅在 IBM MQ for z/OS 上受支持。

请参阅 第 137 页的『Shared queues and queue sharing groups』 以获取更多信息。

系统缺省对象

系统缺省对象 是一组对象定义，每当创建队列管理器时都会自动创建这些对象定义。 您可以复制和修改这些对象定义中的任何对象定义，以便在安装时在应用程序中使用。

缺省对象名具有主干 SYSTEM; 例如，缺省本地队列为 SYSTEM.DEFAULT.LOCAL.QUEUE，缺省接收方通道为 SYSTEM.DEF.RECEIVER。 无法重命名这些对象; 需要这些名称的缺省对象。

定义对象时，将从相应的缺省对象复制未显式指定的任何属性。 例如，如果定义本地队列，那么将从缺省队列 SYSTEM.DEFAULT.LOCAL.QUEUE。

请参阅 系统和缺省对象 以获取更多信息。

对象类型

许多管理任务涉及处理各种类型的 IBM MQ 对象。

有关命名 IBM MQ 对象的信息，请参阅 第 29 页的『对 IBM MQ 对象进行命名』。

有关在队列管理器上创建的缺省对象的信息，请参阅 第 13 页的『系统缺省对象』。

有关不同类型的 IBM MQ 对象的信息，请参阅以下内容:

认证信息对象

认证信息对象提供执行证书撤销检查所需的定义。

队列管理器认证信息对象构成对传输层安全性 (TLS) 的 IBM MQ 支持的一部分。 它提供了检查已撤销证书所需的定义。 认证中心撤销不再可信的证书。

您可以使用 MQSC 命令 **DEFINE AUTHINFO** 来定义认证信息对象。 有关认证信息对象的属性的更多信息，请参阅 **DEFINE AUTHINFO**。

您可以将以下 IBM MQ 控制命令与认证信息对象配合使用:

- **setmqaut** (授予或撤销权限)
- **dspmqaut** (显示对象授权)
- **dmpmqaut** (转储授权)
- **rcrmqobj** (重新创建对象)
- **rcdmqimg** (记录介质图像)
- **dspmqfls** (显示文件名)

有关 TLS 以及使用认证信息对象的概述，请参阅 传输层安全性 (TLS) 概念 和 IBM MQ 中的 TLS 安全协议。

通道

通道是提供从一个队列管理器到另一个队列管理器的通信路径的对象。

请参阅 第 24 页的『通道』 以获取更多信息。

通信信息对象

IBM MQ 多点广播提供低等待时间、高扇出和可靠的多点广播消息传递。 需要通信信息 (COMMINFO) 对象才能使用多点广播传输。

请参阅 第 86 页的『IBM MQ 多点广播』 以获取更多信息。

COMMINFO 对象是包含与多点广播传输关联的属性的 IBM MQ 对象。 有关这些属性的更多信息，请参阅 DEFINE COMMINFO。 有关创建 COMMINFO 对象的更多信息，请参阅 多点广播入门。

侦听器

侦听器 是接受来自其他队列管理器或客户机应用程序的网络请求并启动关联通道的进程。

可以使用 **runmqlsr** 控制命令来启动 侦听器进程 。

侦听器对象 是 IBM MQ 对象，允许您在队列管理器的作用域内管理侦听器进程的启动和停止。 通过定义侦听器对象的属性，可以执行以下操作:

- 配置侦听器进程。
- 指定当队列管理器启动和停止时，侦听器进程是否自动启动和停止。

要点: z/OS IBM MQ for z/OS 上不支持侦听器对象。 有关 IBM MQ for z/OS 如何通过使用通道启动程序实现侦听的更多信息，请参阅 第 133 页的『The channel initiator on z/OS』。

名称列表

名称列表 是一个 IBM MQ 对象，其中包含集群名称，队列名称或认证信息对象名称的列表。 在集群中，它可用于标识队列管理器为其保存存储库的集群的列表。

名称列表是包含其他 IBM MQ 对象列表的 IBM MQ 对象。 通常，名称列表由诸如触发器监视器之类的应用程序使用，在那些应用程序中，名称列表用来标识一组队列。 使用名称列表的优点是它独立于应用程序进行维护; 可以在不停止使用它的任何应用程序的情况下进行更新。 另外，如果一个应用程序失败，那么名称列表不受影响，其他应用程序可以继续使用该名称列表。

名称列表还与队列管理器集群配合使用，以维护由多个 IBM MQ 对象引用的集群列表。

您可以使用 MQSC 命令 DEFINE NAMELIST 和 ALTER NAMELIST 来定义和修改名称列表。

注: z/OS 或者，在 z/OS 上，可以使用 IBM MQ for z/OS 操作和控制面板

程序可以使用 MQI 来查找这些名称列表中包含的队列。 名称列表的组织由应用程序设计者和系统管理员负责。

要获取可供使用的名称列表属性的列表，请参阅 名称列表的属性。

进程定义

进程定义对象允许通过定义应用程序的属性以供队列管理器使用来启动应用程序，而无需操作员干预。

进程定义对象定义用于响应 IBM MQ 队列管理器上的触发器事件而启动的应用程序。 进程定义属性包括应用程序标识，应用程序类型以及特定于应用程序的数据。 有关更多信息，请参阅 第 22 页的『IBM MQ 用于特定用途的队列』中的 启动队列 。

要允许在不需要操作员干预的情况下启动应用程序，如 使用触发器启动 IBM MQ 应用程序中所述，队列管理器必须知道应用程序的属性。 这些属性是在 进程定义对象中定义的。

创建对象时， **ProcessName** 属性是固定的。 但是，您可以使用 IBM MQ 命令来更改其他属性。

注: z/OS 或者，在 z/OS 上，可以使用 IBM MQ for z/OS 操作和控制面板。

您可以使用 MQINQ-Inquire object attributes 来查询所有属性的值。

有关可供使用的进程定义属性的列表，请参阅 进程定义的属性。

队列

IBM MQ 队列 是一个指定对象，应用程序可以在该对象上放置消息，并且应用程序可以从中获取消息。

请参阅 第 16 页的『队列』 以获取更多信息。

队列管理器

IBM MQ 队列管理器向应用程序提供排队服务，并管理属于它们的队列。

请参阅 第 23 页的『队列管理器』 以获取更多信息。

服务

服务 对象是定义在队列管理器启动或停止时要运行的程序的一种方法。

程序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。 服务器服务对象是将在启动指定队列管理器时执行的程序的定义。 只能同时执行服务器进程的一个实例。 运行时，可以使用 MQSC 命令 DISPLAY SVSTATUS 来监视服务器进程的状态。 通常，服务器服务对象是程序 (例如，死信处理程序或触发器监

视器) 的定义，但是可以运行的程序不限于 IBM MQ 随附的程序。 此外，可以定义服务器服务对象以包含将在指定队列管理器关闭以结束程序时运行的命令。

命令

命令 是将参数 SERVTYPE 指定为 COMMAND 的服务对象。 命令服务对象是将在启动或停止指定队列管理器时执行的程序的定义。 可以同时执行命令进程的多个实例。 命令服务对象与服务器服务对象不同，因为一旦执行了程序，队列管理器就不会监视该程序。 通常，命令服务对象是生命周期较短的程序的定义，并且将执行特定任务 (例如，启动一个或多个其他任务)。

要点: z/OS IBM MQ for z/OS 上不支持服务对象。

有关更多信息，请参阅 使用服务。

存储类

z/OS

存储类将一个或多个队列映射至页集。

这意味着该队列的消息将存储在该页集上 (可进行缓冲)。

仅在 IBM MQ for z/OS 上支持存储类。

有关存储类的更多信息，请参阅 在 z/OS 上规划 IBM MQ 环境。

Topic 对象

主题对象 是一个 IBM MQ 对象，允许您将特定非缺省属性分配给主题。

主题 由发布或预订特定 主题字符串的应用程序定义。 主题字符串可以通过使用正斜杠字符 (/) 分隔主题来指定主题层次结构。 这可以通过 主题树进行可视化。 例如，如果应用程序发布到主题字符串 `/Sport/American Football` 和 `/Sport/Soccer`，那么将创建一个主题树，其中包含具有两个子代 `American Football` 和 `Soccer` 的父节点 `Sport` 。

主题从其主题树中找到的第一个父管理节点继承其属性。 如果特定主题树中没有管理主题节点，那么所有主题都将从基本主题对象 SYSTEM.BASE.TOPIC。

您可以在主题树中的任何节点上创建主题对象，方法是在主题对象的 TOPICSTR 属性中指定该节点的主题字符串。 您还可以为管理主题节点定义其他属性。 有关这些属性的更多信息，请参阅 MQSC 命令或 使用 PCF 命令自动管理。 缺省情况下，每个主题对象都从其最接近的父管理主题节点继承其属性。

主题对象还可用于向应用程序开发者隐藏完整主题树。 如果为主题 `/Sport/American Football` 创建了名为 `FOOTBALL.US` 的主题对象，那么应用程序可以发布或预订名为 `FOOTBALL.US` 的对象，而不是具有相同结果的字符串 `/Sport/American Football` 。

如果在主题对象的主题字符串中输入 `#` ， `+` ， `/` 或 `*` 字符，那么该字符将被视为字符串中的正常字符，并被视为与主题对象关联的主题字符串的一部分。

有关主题对象的更多信息，请参阅 第 51 页的『发布/预订消息传递』。

相关概念

第 5 页的『消息排队简介』
IBM MQ 产品支持程序使用一致的应用程序编程接口在不同组件 (处理器，操作系统，子系统和通信协议) 的网络中相互通信。

相关参考

MQSC 命令

队列

IBM MQ 队列和队列属性简介。

消息存储在队列上，因此，如果放置应用程序期望对其消息进行应答，那么在等待该应答时可以执行其他工作。 应用程序使用消息队列接口 (MQI) 访问队列，如 消息队列接口概述中所述。

必须先创建队列，然后才能将消息放入队列中。 队列由队列管理器拥有，并且该队列管理器可以拥有许多队列。 但是，每个队列必须具有在该队列管理器中唯一的名称。

通过队列管理器维护队列。 在大多数情况下，每个队列都由其队列管理器进行物理管理，但这对于应用程序并不明显。 IBM MQ for z/OS 共享队列可以由队列共享组中的任何队列管理器管理。

要创建队列，可以使用 IBM MQ 命令 (MQSC)， PCF 命令或特定于平台的接口。 例如， IBM MQ for z/OS 操作和控制面板特定于平台。

您可以从应用程序 动态 为临时作业创建本地队列。 例如，您可以创建 *reply-to* 队列 (在应用程序结束后不需要这些队列)。 有关更多信息，请参阅 第 21 页的『动态队列和模型队列』。

在使用队列之前，必须打开队列，并指定要对其执行的操作。 例如，您可以打开以下对象的队列:

- 仅浏览消息 (不检索消息)
- 检索消息 (并与其他程序共享访问权或独占访问权)
- 将消息放入队列
- 查询队列的属性
- 设置队列的属性

要获取打开队列时可以指定的选项的完整列表，请参阅 MQOPEN-Open object。

队列的属性

队列的某些属性是在定义队列时指定的，之后无法更改 (例如，队列的类型)。 可以将队列的其他属性分组到可以更改的属性中:

- 在队列处理期间由队列管理器执行 (例如，队列的当前深度)
- 仅通过命令 (例如，队列的文本描述)
- 由应用程序使用 MQSET 调用 (例如，是否允许在队列上执行 put 操作)

您可以使用 MQINQ 调用来查找所有属性的值。

对于多种类型的队列通用的属性包括:

QName
队列的名称。

QTYPE
队列的类型。

QDesc
队列的文本描述。

InhibitGet
是否允许程序从队列获取消息。 但是，您永远无法从远程队列获取消息。

InhibitPut
是否允许程序将消息放入队列。

DefPriority
放入队列的消息的缺省优先级。

DefPersistence
放入队列中的消息的缺省持久性

作用域
控制此队列的条目是否也存在于名称服务中。

z/OS **Scope** 属性在 z/OS 上不受支持

有关这些属性的完整描述，请参阅 队列的属性。

定义队列的方法

您可以使用 MQSC DEFINE 命令或 PCF 创建队列 命令将队列定义到 IBM MQ 。 这些命令指定队列的类型及其属性。 例如，本地队列对象具有一些属性，这些属性指定应用程序在 MQI 调用中引用该队列时发生的情况。 属性的示例包括:

- 应用程序是否可以从队列中检索消息 (已启用 GET)
- 应用程序是否可以将消息放入队列 (已启用 PUT)
- 对队列的访问是由一个应用程序独占还是在应用程序之间共享
- 可同时存储在队列上的最大消息数 (最大队列深度)
- 可以放到队列中的消息的最大长度

您还可以使用各种特定于平台的接口来定义队列。

相关概念

第 47 页的『集群队列』
集群队列是由集群队列管理器托管并可供集群中其他队列管理器使用的队列。

第 40 页的『死信队列』
死信队列 (或未传递的消息队列) 是无法将消息路由到其正确目标时将消息发送到的队列。 每个队列管理器通常都有一个死信队列。

使用 PCF 命令自动管理

IBM MQ Console: 使用队列

相关任务

使用 MQSC 命令管理 IBM MQ

使用 MQ Explorer 创建和配置队列管理器和对象

IBM i 使用 CL 命令管理 IBM MQ for IBM i

z/OS 可在 IBM MQ for z/OS 上发出 MQSC 和 PCF 命令的源

相关参考

第 48 页的『Comparison between shared queues and cluster queues』
This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

相关信息

第 137 页的『What is a shared queue?』

本地队列

传输，启动，死信，命令，缺省，通道和事件队列是本地队列的类型。

如果队列归程序连接到的队列管理器所有，那么该队列对程序而言是本地队列。 您可以从本地队列中获取消息及将消息放在本地队列上。

队列定义对象保存队列的定义信息以及放在该队列上的物理消息。

每个队列管理器都可以有一些用于特殊目的的本地队列:

传输队列

当应用程序将消息发送到远程队列时，本地队列管理器将消息存储在称为 传输队列的特殊本地队列中。 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。

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. 与远程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。

消息通道代理程序 是与传输队列相关联的通道程序，它将消息传递到其下一个目标。 下一个目标是消息通道所连接到的队列管理器。 它不一定是与消息的最终目标相同的队列管理器。 将消息传递到其下一个

目标时，将从传输队列中删除该消息。 该消息可能必须在到达其最终目标的过程中通过许多队列管理器。 您必须在路由的每个队列管理器上定义一个传输队列，每个队列管理器都保存等待传输到下一个目标的消息。 正常传输队列保存下一个目标的消息，尽管这些消息可能具有不同的最终目标。 集群传输队列保存多个目标的消息。 每条消息的 `correlID` 标识放置该消息以将其传输到其下一个目标的通道。

您可以在队列管理器上定义多个传输队列。 您可以为同一目标定义多个传输队列，每个传输队列用于不同的服务类。 例如，您可能想要为发送到同一目标的小消息和大消息创建不同的传输队列。 然后，您可以使用不同的消息通道来传输消息，以便大型消息不会容纳较小的消息。 缺省情况下，集群队列或集群主题的所有消息都放置在单集群传输队列 `SYSTEM.CLUSTER.TRANSMIT.QUEUE` 上。 作为一个选项，您可以更改缺省值，并将流向不同集群队列管理器的消息流量分隔到不同的集群传输队列上。 如果将队列管理器属性 `DEFCLXQ` 设置为 `CHANNEL`，那么每个集群发送方通道都会创建一个单独的集群传输队列。 作为替代方法，您可以手动定义集群传输队列以供集群发送方通道使用。

传输队列可以触发消息通道代理程序以继续发送消息; 请参阅 使用触发器启动 IBM MQ 应用程序。

z/OS 在 IBM MQ for z/OS 上，如果使用组内排队，那么传输队列由 组内排队代理提供服务。 在 IBM MQ for z/OS 上使用组内排队时，将使用共享传输队列。

启动队列

启动队列 是一个本地队列，当应用程序队列上发生触发器事件时，队列管理器在该队列上放置触发器消息。

触发器事件是旨在使程序开始处理队列的事件。 例如，事件可能超过 10 条消息到达。 有关触发如何工作的更多信息，请参阅 使用触发器启动 IBM MQ 应用程序。

死信（未传递的消息）队列

死信 (未传递的消息) 队列 是队列管理器将无法传递的消息放入其中的本地队列。

当队列管理器将消息放入死信队列时，它会向消息添加头。 头信息包括队列管理器将消息放入死信队列的原因。 它还包含原始消息的目标，日期以及队列管理器将消息放入死信队列的时间。

应用程序还可以将队列用于它们无法传递的消息。 有关更多信息，请参阅 使用死信 (未传递的消息) 队列。

系统命令队列

系统命令队列 是适当授权的应用程序可以向其发送 IBM MQ 命令的队列。 这些队列接收平台上支持的 PCF， MQSC 和 CL 命令，以便队列管理器能够对它们进行操作。

z/OS 在 IBM MQ for z/OS 上，队列称为 `SYSTEM.COMMAND.INPUT` ; 在其他平台上称为 `SYSTEM.ADMIN.COMMAND.QUEUE`。 接受的命令因平台而异。 有关详细信息，请参阅可编程命令格式参考。

系统缺省队列

系统缺省队列 包含系统的队列的初始定义。 创建队列定义时，队列管理器将从相应的系统缺省队列复制该定义。 创建队列定义与创建动态队列不同。 动态队列的定义基于您选择作为动态队列模板的模型队列。

事件队列

事件队列 用于保存事件消息。 这些消息由队列管理器或通道报告。

远程队列

对于程序，如果队列由与程序所连接的队列管理器不同的队列管理器拥有，那么该队列是 远程 队列。

在已建立通信链路的情况下，程序可以向远程队列发送消息。 程序永远无法从远程队列获取消息。

定义远程队列时创建的队列定义对象仅保存本地队列管理器查找您希望消息进入的队列所需的信息。 此对象称为 远程队列的本地定义。 远程队列的所有属性都由拥有它的队列管理器挂起，因为它是该队列管理器的本地队列。

打开远程队列时，要标识该队列，必须指定以下任一项:

- 定义远程队列的本地定义的名称。 从应用程序的角度来看，这与打开本地队列相同。 应用程序不需要知道队列是本地队列还是远程队列。

  要在除 IBM i 以外的所有平台上创建远程队列的本地定义，请使用 DEFINE QREMOTE 命令。

  IBM i 在 IBM i 上，使用 CRTMQMQ 命令。
- 远程队列管理器的名称以及该远程队列管理器已知的队列的名称。

除了 第 17 页的『队列的属性』中描述的公共属性外，远程队列的本地定义还具有三个属性。 这三个属性是:

RemoteQName
队列拥有的队列管理器识别该队列的名称。

RemoteQmgrName
拥有队列管理器的名称。

XmitQName
将消息转发到其他队列管理器时使用的本地传输队列的名称。

有关这些属性的更多信息，请参阅 队列的属性。

如果对远程队列的本地定义使用 MQINQ 调用，那么队列管理器仅返回本地定义的属性，即远程队列名称，远程队列管理器名称和传输队列名称，而不是远程系统中匹配的本地队列的属性。

另请参阅 传输队列 (Transmission queue)。

别名队列

别名队列 是可用于访问另一个队列或主题的 IBM MQ 对象。 这意味着多个程序可以使用同一个队列，使用不同的名称访问该队列。

通过解析别名 (称为基本队列) 而产生的队列。 可以是平台支持的以下任何类型的队列:

- 本地队列
- 远程队列的本地定义。
- z/OS 共享队列，这是仅在 IBM MQ for z/OS 上可用的本地队列类型。
- 预定义队列
- 动态队列

别名也可以解析为主题。 如果应用程序当前将消息放入队列，那么可以通过使队列名称成为主题的别名来将其发布到主题。 无需更改应用程序代码。

注: 别名不能直接解析为同一队列管理器上的另一个别名。

使用别名队列的一个示例是系统管理员对基本队列名称 (即，别名解析到的队列) 和别名队列名称授予不同的访问权限。 这意味着可以授权程序或用户使用别名队列，但不能使用基本队列。

或者，可以设置授权以禁止对别名执行放置操作，但允许对基本队列执行这些操作。

在某些应用程序中，使用别名队列意味着系统管理员可以轻松更改别名队列对象的定义，而不必更改应用程序。

当程序尝试使用别名时， IBM MQ 会针对别名进行授权检查。 它不会检查程序是否有权访问别名解析为的名称。 因此，可以授权程序访问别名队列名称，但不能访问已解析的队列名称。

除了 第 16 页的『队列』中描述的常规队列属性外，别名队列还具有 **BaseQName** 属性。 这是别名解析到的基本队列的名称。 有关此属性的更完整描述，请参阅 BaseQName (MQCHAR48)。

InhibitGet 和 **InhibitPut** 属性 (请参阅 第 16 页的『队列』) 别名队列属于别名。 例如，如果别名队列名称 ALIAS1 解析为基本队列名称 BASE ，那么对 ALIAS1 的禁止仅影响 ALIAS1 ，并且不会禁止 BASE。 但是，禁止 BASE 也会影响 ALIAS1。

DefPriority 和 **DefPersistence** 属性也属于别名。 例如，您可以将不同的缺省优先级分配给同一基本队列的不同别名。 此外，您可以更改这些优先级，而不必更改使用别名的应用程序。

动态队列和模型队列

此信息可深入了解动态队列，临时和永久动态队列的属性，动态队列的使用，使用动态队列时的一些注意事项以及模型队列。

当应用程序发出 MQOPEN 调用以打开模型队列时，队列管理器会动态创建与模型队列具有相同属性的本地队列实例。 根据模型队列的 *DefinitionType* 字段的值，队列管理器将创建临时或永久动态队列 (请参阅创建动态队列)。

临时动态队列的属性

临时动态队列 具有以下属性:

- z/OS 它们不能是共享队列，可从队列共享组中的队列管理器访问。

  请注意，队列共享组仅在 IBM MQ for z/OS 上可用。
- 它们仅保存非持久消息。
- 它们是不可恢复的。
- 当队列管理器启动时，将删除它们。
- 当发出创建队列的 MQOPEN 调用的应用程序关闭队列或终止时，将删除这些消息。
  - 如果队列上存在任何已落实的消息，那么将删除这些消息。
  - 如果此时有任何未落实的 MQGET， MQPUT 或 MQPUT1 调用未针对队列执行，那么该队列将标记为正在逻辑上删除，并且仅在落实这些调用后作为关闭处理的一部分进行物理删除，或者在应用程序终止时进行物理删除。
  - 如果此时该队列正在使用 (由创建或另一个应用程序)，那么该队列将标记为正在逻辑上被删除，并且仅当最后一个使用该队列的应用程序关闭时才会物理上被删除。
  - 尝试访问逻辑删除的队列 (而不是关闭该队列) 失败，原因码为 MQRC_Q_DELETED。
  - 在针对创建队列的相应 MQOPEN 调用的 MQCLOSE 调用上指定 MQCO_NONE 时， MQCO_DELETE 和 MQCO_DELETE_PURGE 都将被视为 MQCO_NONE。

永久动态队列的属性

永久动态队列 具有以下属性:

- 它们保存持久或非持久消息。
- 在发生系统故障时，它们是可恢复的。
- 当应用程序 (不一定是发出了创建队列的 MQOPEN 调用的应用程序) 使用 MQCO_DELETE 或 MQCO_DELETE_PURGE 选项成功关闭队列时，将删除这些消息。
  - 如果队列上仍有任何消息 (已落实或未落实)，那么带有 MQCO_DELETE 选项的关闭请求将失败。 带有 MQCO_DELETE_PURGE 选项的关闭请求成功，即使队列上存在已落实的消息 (要在关闭过程中删除的消息)，但如果针对队列有任何未落实的 MQGET， MQPUT 或 MQPUT1 调用未完成，那么将失败。
  - 如果删除请求成功，但该队列恰好正在使用 (由创建应用程序或其他应用程序)，那么该队列将标记为逻辑上已删除，并且仅当最后一个使用该队列的应用程序关闭时才会实际删除该队列。
- 如果未被授权删除队列的应用程序关闭，那么不会将其删除，除非关闭应用程序发出了创建队列的 MQOPEN 调用。 将对用于验证相应 MQOPEN 调用的用户标识 (或备用用户标识 (如果指定了 MQOO_ALTERNATE_USER_AUTHORITY)) 执行授权检查。
- 可以使用与正常队列相同的方式删除这些队列。

动态队列的使用

您可以将动态队列用于:

- 在应用程序终止后不需要保留队列的应用程序。
- 需要由另一个应用程序处理对消息的回复的应用程序。 此类应用程序可以通过打开模型队列来动态创建应答队列。 例如，客户机应用程序可以:

  1. 创建动态队列。

2. 在请求消息的消息描述符结构的 **ReplyToQ** 字段中提供其名称。
3. 将请求放在服务器正在处理的队列上。

然后，服务器可以将应答消息放在应答队列上。 最后，客户机可以处理应答，并使用删除选项关闭应答队列。

使用动态队列时的注意事项

使用动态队列时，请考虑以下几点:

- 在客户机/服务器模型中，每个客户机都必须创建并使用其自己的动态应答队列。 如果在多个客户机之间共享动态应答队列，那么可能会延迟删除应答队列，因为针对该队列存在未落实的活动，或者该队列正由另一个客户机使用。 此外，该队列可能被标记为正在逻辑上被删除，并且对于后续 API 请求 (MQCLOSE 除外) 不可访问。
- 如果应用程序环境要求必须在应用程序之间共享动态队列，请确保仅当已落实针对队列的所有活动时才关闭该队列 (使用删除选项)。 这应该是最后一个用户。 这将确保不会延迟删除该队列，并将由于该队列已标记为已被逻辑删除而无法访问的时间段最小化。

模型队列

模型队列 是您在创建动态队列时使用的队列定义的模板。

您可以从 IBM MQ 程序动态创建本地队列，命名要用作队列属性模板的模型队列。 此时，您可以更改新队列的某些属性。 但是，您无法更改 **DefinitionType**。 例如，如果需要永久队列，请选择定义类型设置为永久的模型队列。 某些会话式应用程序可以使用动态队列来保存对其查询的应答，因为在处理这些应答之后，它们可能不需要维护这些队列。

在 MQOPEN 调用的 对象描述符 (MQOD) 中指定模型队列的名称。 通过使用模型队列的属性，队列管理器会动态地为您创建本地队列。

您可以指定动态队列的名称 (完整) 或名称的主干 (例如， ABC) ，并让队列管理器为此添加唯一部分，也可以让队列管理器为您分配完整的唯一名称。 如果队列管理器指定名称，那么它会将其放入 MQOD 结构中。

不能直接向模型队列发出 MQPUT1 调用，但可以向已通过打开模型队列创建的动态队列发出 MQPUT1 。

无法对模型队列发出 MQSET 和 MQINQ。 使用 MQOO_INQUIRE 或 MQOO_SET 打开模型队列会导致对动态创建的队列进行后续 MQINQ 和 MQSET 调用。

模型队列的属性是本地队列的属性的子集。 有关更完整的描述，请参阅 队列的属性。

IBM MQ 用于特定用途的队列

IBM MQ 将某些本地队列用于与其操作相关的特定用途。

您必须先定义这些队列，然后 IBM MQ 才能使用这些队列。

启动队列

启动队列是在触发中使用的队列。 当发生触发器事件时，队列管理器将触发器消息放在启动队列上。 触发器事件是队列管理器检测到的条件的逻辑组合。 例如，当队列上的消息数达到预定义的深度时，可能会生成触发器事件。 此事件会导致队列管理器将触发器消息放在指定的启动队列上。 此触发器消息由触发器监视器(用于监视启动队列的特殊应用程序) 检索。 然后，触发器监视器启动触发器消息中指定的应用程序。

如果队列管理器要使用触发，那么必须至少为该队列管理器定义一个启动队列。 请参阅 管理用于触发的对象， runmqtrm 和 使用触发器启动 IBM MQ 应用程序

传输队列

传输队列是临时存储发往远程队列管理器的消息的队列。 必须为本地队列管理器要直接向其发送消息的每个远程队列管理器定义至少一个传输队列。 这些队列也用于远程管理; 请参阅 来自本地队列管理器的远程管理。 有关在分布式排队中使用传输队列的信息，请参阅 IBM MQ 分布式排队技术。

每个队列管理器都可以具有缺省传输队列。 如果不属于集群的队列管理器将消息放入远程队列，那么缺省操作是使用缺省传输队列。 如果存在与目标队列管理器同名的传输队列，那么消息将放置在该传输队列上。 如果存在队列管理器别名定义，其中 **RQMNAME** 参数与目标队列管理器匹配，并且指定了 **XMITQ**

参数，那么会将消息放在由 **XMITQ** 指定的传输队列上。 如果没有 **XMITQ** 参数，那么会将消息放在消息中指定的本地队列上。

集群传输队列
集群中的每个队列管理器都有一个名为 SYSTEM.CLUSTER.TRANSMIT.QUEUE 的集群传输队列和一个模型集群传输队列 SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE。 缺省情况下，在定义队列管理器时将创建这些队列的定义。 如果队列管理器属性 **DEFCLXQ** 设置为 CHANNEL，那么将自动为创建的每个集群发送方通道创建永久动态集群传输队列。 这些队列称为 SYSTEM.CLUSTER.TRANSMIT.*ChannelName*。 您还可以手动定义集群传输队列。

属于集群的队列管理器将其中一个队列上的消息发送到同一集群中的其他队列管理器。

在名称解析期间，集群传输队列优先于缺省传输队列，而特定集群传输队列优先于 SYSTEM.CLUSTER.TRANSMIT.QUEUE。

死信队列
死信 (undelivered-message) 队列是存储无法路由到其正确目标的消息的队列。 例如，当目标队列已满时，无法路由消息。 提供的死信队列称为 SYSTEM.DEAD.LETTER.QUEUE。

对于分布式排队，请在涉及的每个队列管理器上定义死信队列。

命令队列
命令队列 SYSTEM.ADMIN.COMMAND.QUEUE 是一个本地队列，适当授权的应用程序可以将 MQSC 命令发送到该队列以进行处理。 然后，这些命令将由称为命令服务器的 IBM MQ 组件检索。 命令服务器验证命令，传递有效的命令以供队列管理器处理，并将任何响应返回到相应的应答队列。

创建队列管理器时，将自动为每个队列管理器创建命令队列。

应答队列
当应用程序发送请求消息时，接收消息的应用程序可以将应答消息发送回发送应用程序。 此消息放在队列上，称为应答队列，通常是发送应用程序的本地队列。 应答队列的名称由发送应用程序指定为消息描述符的一部分。

事件队列
检测事件可用于独立于 MQI 应用程序监视队列管理器。

发生检测事件时，队列管理器会将事件消息放入事件队列中。 然后，监视应用程序可以读取此消息，这可能会通知管理员或在事件指示问题时启动一些补救操作。

注: 触发器事件与检测事件不同。 触发器事件不是由相同的条件引起的，并且不会生成事件消息。

有关检测事件的更多信息，请参阅 检测事件。

队列管理器

队列管理器 及其提供给应用程序的排队服务简介。

程序必须具有与队列管理器的连接，然后才能使用该队列管理器的服务。 程序可以显式建立此连接 (使用 MQCONN 或 MQCONNX 调用) ，也可以隐式建立此连接 (这取决于运行程序的平台和环境)。

IBM MQ 队列管理器确保执行以下操作:

- 根据接收到的命令更改对象属性。
- 满足相应条件时，将生成特殊事件 (例如，触发器事件或检测事件)。
- 根据发出 MQPUT 调用的应用程序的请求，将消息放入正确的队列中。 如果无法执行此操作，那么将通知应用程序，并提供相应的原因码。

每个队列都属于单个队列管理器，并且被认为是该队列管理器的 本地队列 。 应用程序所连接的队列管理器被认为是该应用程序的 本地队列管理器 。 对于应用程序，属于其本地队列管理器的队列是本地队列。

远程队列 是属于另一个队列管理器的队列。 远程队列管理器 是除本地队列管理器以外的任何队列管理器。 远程队列管理器可以存在于网络中的远程机器上，也可以存在于本地队列管理器所在的机器上。 IBM MQ 支持同一机器上的多个队列管理器。

可以在某些 MQI 调用中使用队列管理器对象。 例如，您可以使用 MQI 调用 MQINQ 来查询队列管理器对象的属性。

队列管理器的属性

与每个队列管理器关联的是一组用于定义其特征的属性 (或属性)。队列管理器的某些属性在创建时是固定的; 您可以使用 IBM MQ 命令来更改其他属性。 您可以使用 MQINQ 调用来查询所有属性 (用于传输层安全性 (TLS) 加密的属性除外) 的值。

固定属性包括:

- 队列管理器的名称
- 运行队列管理器的平台 (例如， Windows)
- 队列管理器支持的系统控制命令的级别
- 可以分配给队列管理器处理的消息的最大优先级
- 程序可以向其发送 IBM MQ 命令的队列的名称
- 队列管理器可处理的最大消息长度 z/OS (仅在 IBM MQ for z/OS 中修正)
- 当程序放入和获取消息时，队列管理器是否支持同步

可更改 属性包括:

- 队列管理器的文本描述
- 队列管理器在处理 MQI 调用时用于字符串的字符集的标识
- 队列管理器用于限制触发器消息数的时间间隔
- z/OS 队列管理器用于确定队列扫描到期消息的频率的时间间隔 (仅限 IBM MQ for z/OS)
- 队列管理器的死信 (未传递的消息) 队列的名称
- 队列管理器的缺省传输队列的名称
- 任何一个连接的最大打开句柄数
- 启用和禁用各种类别的事件报告
- 工作单元中未落实的最大消息数

队列管理器和工作负载管理

您可以设置具有相同队列的多个定义的队列管理器集群 (例如，集群中的队列管理器可以相互克隆)。 特定队列的消息可由主管队列实例的任何队列管理器处理。 工作负载管理算法决定哪个队列管理器处理消息，从而在队列管理器之间传播工作负载; 请参阅 集群工作负载管理算法 以获取更多信息。

通道

通道 是分布式队列管理器在 IBM MQ MQI client 与 IBM MQ 服务器之间或两个 IBM MQ 服务器之间使用的逻辑通信链路。

通道用于将消息从一个队列管理器移动到另一个队列管理器，它们保护应用程序免受底层通信协议的保护。 队列管理器可能存在于同一系统上，也可能存在于同一平台上或不同平台上的不同系统上。 发送的消息可能源自许多位置:

- 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 MQ Explorer。
- 将检测事件消息发送到另一个队列管理器的队列管理器。
- 将远程管理命令发送到另一个队列管理器的队列管理器。 例如，使用 MQSC 命令或 administrative REST API。

通道有两个定义: 连接的每个端都有一个定义。 要使队列管理器相互通信，必须在要发送消息的队列管理器上定义一个通道对象，在要接收消息的队列管理器上定义另一个补充对象。 在连接的每一端都必须使用相同的 通道名称 ，并且所使用的 通道类型 必须兼容。

IBM MQ 中有三个类别的通道，这些类别中有不同的通道类型:

- 消息通道 (单向) ，将消息从一个队列管理器传输到另一个队列管理器。
- MQI 通道 (双向) ，用于将 MQI 调用从 IBM MQ MQI client 传输到队列管理器，并将响应从队列管理器传输到 IBM MQ 客户机。
- AMQP 通道，这是双向通道，用于将 AMQP 客户机连接到服务器上的队列管理器。 IBM MQ 使用 AMQP 通道在 AMQP 应用程序和队列管理器之间传输 AMQP 调用和响应

消息通道

消息通道的用途是将消息从一个队列管理器传输到另一个队列管理器。 客户机服务器环境不需要消息通道。

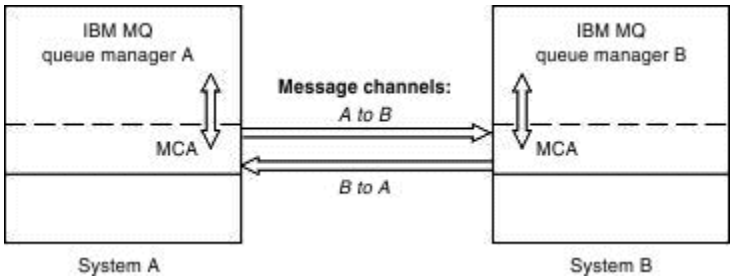


图 *2:* 两个队列管理器之间的消息通道

消息通道是单向链接。 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，那么必须设置第二个通道以将响应发送回本地队列管理器。

消息通道使用 消息通道代理程序 (MCA) 连接两个队列管理器。 在通道的每一端都有一个消息通道代理程序。 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。 有关管道传输的更多信息，请参阅 通道属性。

有关通道的更多信息，请参阅 通道出口调用和数据结构和 第 38 页的『分布式排队组件』。

MQI 通道

消息队列接口 (MQI) 通道将 IBM MQ MQI client 连接到服务器上的队列管理器，并在您从 IBM MQ MQI client 应用程序发出 MQCONN 或 MQCONNX 调用时建立。

它是双向链接，仅用于传输 MQI 调用和响应，包括包含消息数据的 MQPUT 调用和导致返回消息数据的 MQGET 调用。 有不同的方法来创建和使用通道定义 (请参阅 定义 MQI 通道)。

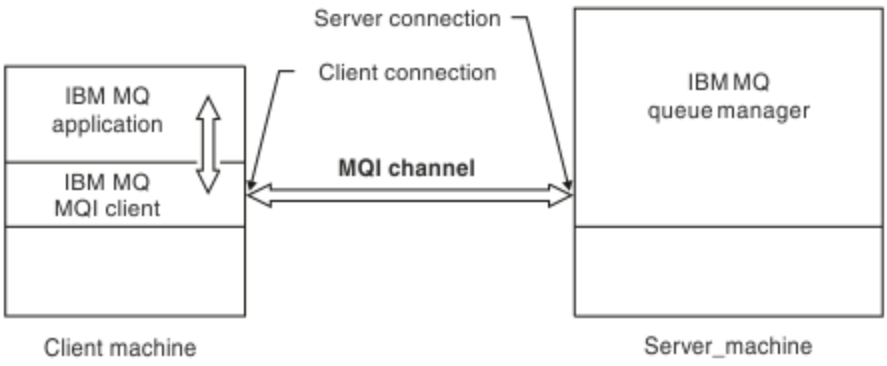


图 *3: MQI* 通道上的客户机连接和服务器连接

z/OS MQI 通道可用于将客户机连接到单个队列管理器，或连接到属于队列共享组的队列管理器 (请参阅 将客户机连接到队列共享组)。

MQI 通道定义有两种通道类型。 它们定义双向 MQI 通道。

客户机连接通道
此类型适用于 IBM MQ MQI client。

服务器连接通道
此类型适用于运行队列管理器的服务器，在 IBM MQ MQI client 环境中运行的 IBM MQ 应用程序将与该服务器进行通信。

AMQP 通道

只有一种类型的 AMQP 通道。

使用此通道连接 AMQP 消息传递应用程序和队列管理器，从而使该应用程序能够与 IBM MQ 应用程序交换消息。 AMQP 允许您使用 MQ Light 开发应用程序，然后将其部署为企业应用程序，从而利用 IBM MQ 提供的企业级功能。

客户机连接通道

客户机连接通道 是提供从 IBM MQ MQI client 到队列管理器的通信路径的对象。

客户机连接通道用于分布式排队，以在队列管理器和客户机之间移动消息。 它们保护应用程序免受底层通信协议的保护。 客户机可能存在于与队列管理器相同的或不同的平台上。

通道定义

请参阅 第 26 页的『通道定义』 以获取每种类型的通道的描述。

相关概念
第 35 页的『分布式排队和集群』
分布式排队意味着将消息从一个队列管理器发送到另一个队列管理器。 接收队列管理器可以在同一台机器上，也可以在另一台机器上; 可以在附近，也可以在世界的另一侧。 它可以在与本地队列管理器相同的平台上运行，也可以在 IBM MQ 支持的任何平台上运行。 您可以手动定义分布式排队环境中的所有连接，也可以创建集群并让 IBM MQ 为您定义许多连接详细信息。

消息队列接口概述
相关任务
管理远程 IBM MQ 对象
停止 MQI 通道
配置服务器与客户机之间的连接
相关参考
通道出口调用和数据结构
第 29 页的『通信』
IBM MQ MQI clients 使用 MQI 通道与服务器通信。

通道定义

描述 IBM MQ 使用的不同类型的消息通道和 MQI 通道的表。

当提到消息通道时，通道一词常用作通道定义的同义词。 通常从上下文中能清楚地分辨出我们正在谈论的是一个完整的通道（带有两端）还是一个通道定义（只有一端）。

消息通道

消息通道定义可以是下列其中一个类型：

消息通道定义类型	描述
发送方	发送方通道是队列管理器用来将消息发送至其他队列管理器的消息通道。 要使用发送方通道来发送消息，还必须在另一队列管理器上创建与此发送方通道同名的接收方通道。 如果要实现“回叫”机制，那么还可以将发送方通道与请求方通道配合使用。

消息通道定义类型	描述
服务器	服务器通道是队列管理器用来将消息发送至其他队列管理器的消息通道。 要使用服务器通道来发送消息，还必须在另一队列管理器上创建与此服务器通道同名的接收方通道。 您还可以将服务器通道与请求方通道配合使用。 在此情况下，在通道另一端的请求方通道定义将请求要启动的服务器通道定义。 服务器会发送消息给请求方。 只要服务器知道对方通道的连接名称，它还可启动通信。
接收方	接收方通道是队列管理器用来从其他队列管理器接收消息的消息通道。 要使用接收方通道来接收消息，还必须在另一队列管理器上创建与此接收方通道同名的发送方通道或服务器通道。
请求者	请求方通道是队列管理器用来从其他队列管理器接收消息的消息通道。 请求方通道可以请求启动在远端定义的伙伴通道。 如果伙伴通道是服务器通道，那么服务器通道会接受启动请求，并开始从服务器通道定义中标识的传输队列向请求方通道发送消息。 如果伙伴通道是发送方通道，那么发送方通道会接受启动请求，然后关闭与请求方的连接。 然后，发送方通道开始与伙伴请求方通道协商会话，并开始从发送方通道定义中标识的传输队列发送消息。 后一种情况实质上提供了一种回调机制，其中请求方通道可以请求发送方通道进行回调。
集群发送方	集群发送方 (CLUSSDR) 通道定义用来定义通道发送端，集群队列管理器可通过它将集群信息发送至其中一个完整存储库。 集群发送方通道用于告知存储库有关队列管理器状态的任何更改，例如，添加或除去队列。 它还用于传输消息。 完整存储库队列管理器自身拥有指向彼此的集群发送方通道。 它们使用这些通道以就集群状态的更改进行相互通信。 队列管理器的 CLUSSDR 通道定义指向哪个完整存储库不是很重要。 在进行了最初的接触之后，会根据需要自动定义更多的集群队列管理器对象，以便队列管理器可将集群信息发送至每个完整存储库并将消息发送至每个队列管理器。
集群接收方	集群接收方 (CLUSRCVR) 通道定义用来定义通道接收端，集群队列管理器可通过它从集群中的其他队列管理器接收消息。 集群接收方通道还可传送有关集群的信息（发往存储库）。 通过定义集群接收方通道，该队列管理器对其他集群队列管理器表示它可用于接收消息。 每个集群队列管理器至少需要有一个集群接收方通道。

对于每个通道，您必须定义两端以便获取通道每一端的通道定义。 通道的两端必须是兼容类型。

您可使用下列通道定义的组合：

- 发送方-接收方
- 服务器-接收方
- 请求方-服务器
- 请求方-发送方（回调）
- 集群发送方-集群接收方

消息通道代理程序

您创建的每个通道定义都属于特定队列管理器。 队列管理器可具有同一类型或不同类型的几个通道。 通道的每一端是一个程序，即消息通道代理程序 (MCA)。 在通道的一端，调用方 MCA 从传输队列获取消息并通过通道发送它们。 在通道的另一端，响应方 MCA 接收这些消息并将它们传递至远程队列管理器。

调用方 MCA 可与发送方通道、服务器通道或请求方通道关联。 响应方 MCA 可与任何类型的消息通道关联。

IBM MQ 在连接的两端支持通道类型的以下组合：

调用方		**消息流方向**	**响应方**	
通道类型	**需要侦听器吗?**		**需要侦听器吗?**	**通道类型**
发送方	否	从调用方至响应方	Yes	接收方
服务器	否	从调用方至响应方	Yes	接收方
服务器	否	从调用方至响应方	Yes	请求者
请求者	否	从响应方至调用方	Yes	服务器
请求者	Yes	从响应方至调用方	Yes	发送方

MQI 通道

MQI 通道可以是下列其中一个类型：

MQI 通道类型	**描述**
服务器连接	服务器连接通道是双向 MQI 通道，用于将 IBM MQ 客户机连接至 IBM MQ 服务器。 服务器连接通道是通道的服务器端。
客户机连接	客户机连接通道是双向 MQI 通道，用于将 IBM MQ 客户机连接至 IBM MQ 服务器。 IBM MQ Explorer 还使用客户机连接来连接至远程队列管理器。 客户机连接通道是通道的客户机端。 当您创建客户机连接通道时，在主管队列管理器的计算机上创建一个文件。 然后，您必须将该客户机连接文件复制到 IBM MQ 客户端计算机。

Multi 多线程支持-流水线

您可以选择允许消息通道代理 (MCA) 使用多个线程来传输消息。 此过程称为 流水线，使 MCA 能够更高效地传输消息，减少等待状态，从而提高通道性能。 每个 MCA 限制为最多两个线程。

您可以使用 qm.ini 文件中的 *PipeLineLength* 参数来控制管道。 此参数将添加到 Channels 节。

注: 流水线仅对 TCP/IP 通道有效。

使用管道时，必须将通道两端的队列管理器配置为具有大于 1 的 *PipeLineLength* 。

通道出口注意事项

管道可能导致某些出口程序失败，因为：

- 可能未按顺序调用出口。
- 可以从不同的线程交替调用出口。

在使用管道之前，请检查出口程序的设计：

- 出口必须在其执行的所有阶段都可重入。
- 使用 MQI 调用时，请记住，从不同线程调用出口时，不能使用相同的 MQI 句柄。

请考虑打开队列的消息出口，并将其句柄用于对该出口的所有后续调用的 MQPUT 调用。 这在管道方式下失败，因为从不同的线程调用了出口。 要避免此故障，请为每个线程保留一个队列句柄，并在每次调用出口时检查线程标识。

通信

IBM MQ MQI clients 使用 MQI 通道与服务器通信。

必须在连接的 IBM MQ MQI client 和服务器端创建通道定义。 定义 MQI 通道中说明了如何创建通道定义。

下表显示了可能的传输协议:

表 *1: MQI* 通道的传输协议				
客户机平台	**LU 6.2**	**TCP/IP**	**NetBIOS**	**SPX**
IBM i IBM i		Yes		
Linux AIX AIX and Linux 系统	是 [1]	Yes		
Windows Windows	Yes	Yes	Yes	Yes

注:

1. Linux LU6.2 在以下平台上不受支持:
   - Linux (POWER 平台)
   - Linux (x86-64 平台)
   - Linux (zSeries s390x 平台)

传输协议- IBM MQ MQI client 和服务器平台的组合 显示了使用这些传输协议的 IBM MQ MQI client 和服务器平台的可能组合。

IBM MQ MQI client 上的 IBM MQ 应用程序可以使用与队列管理器为本地时相同的所有 MQI 调用。 **MQCONN** 或 **MQCONNX** 将 IBM MQ 应用程序与所选队列管理器相关联，从而创建 连接句柄。 然后，连接的队列管理器将处理使用该连接句柄的其他调用。 IBM MQ MQI client 通信需要客户机与服务器之间的活动连接，而队列管理器之间的通信是独立于连接和独立于时间的。

传输协议是使用通道定义指定的，不会影响应用程序。 例如， Windows 应用程序可以通过 TCP/IP 连接到一个队列管理器，并通过 NetBIOS 连接到另一个队列管理器。

性能注意事项

您使用的传输协议可能会影响 IBM MQ 客户机和服务器系统的性能。 在传输速度较慢的某些情况下，可以使用 IBM MQ 通道压缩。

对 IBM MQ 对象进行命名

针对 IBM MQ 对象采用的命名约定取决于对象。 与 IBM MQ 配合使用的机器的名称和用户标识也受到一些命名限制。

队列管理器的每个实例都由其名称识别。 此名称在相互连接的队列管理器的网络中必须唯一，以便一个队列管理器可以毫不含糊地标识将任何给定消息发送到的目标队列管理器。

对于其他类型的对象，每个对象都有一个与其关联的名称，可以通过该名称引用该名称。 这些名称在一个队列管理器和对象类型中必须唯一。 例如，可以有一个队列和一个同名的进程，但不能有两个同名的队列。

在 IBM MQ 中，名称最多可以包含 48 个字符，但最多包含 20 个字符的 通道 除外。 有关命名 IBM MQ 对象的更多信息，请参阅 第 30 页的『用于命名 IBM MQ 对象的规则』。

与 IBM MQ 配合使用的机器的名称和用户标识也受到一些命名限制:

- 确保机器名不包含任何空格。 IBM MQ 不支持包含空格的机器名。 如果在此类机器上安装 IBM MQ，那么无法创建任何队列管理器。
- 对于 IBM MQ 权限，用户标识和组的名称不得超过 20 个字符 (不允许使用空格)。
- Windows 如果客户机以包含 @ 字符的用户标识 (例如， abc@d.) 运行，那么 IBM MQ for Windows 服务器不支持连接 IBM MQ MQI client 。

相关概念

第 32 页的『IBM MQ 文件名』
每个 IBM MQ 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，服务和认证信息对象都由一个文件表示。 由于对象名不一定是有效的文件名，因此队列管理器会在必要时将对象名转换为有效的文件名。

相关参考

第 30 页的『用于命名 IBM MQ 对象的规则』
IBM MQ 对象名具有最大长度并且区分大小写。 并非每个对象类型都支持所有字符，并且许多对象都有关于名称唯一性的规则。

用于命名 IBM MQ 对象的规则

IBM MQ 对象名具有最大长度并且区分大小写。 并非每个对象类型都支持所有字符，并且许多对象都有关于名称唯一性的规则。

有许多不同类型的 IBM MQ 对象，并且每种类型的对象都可以具有相同的名称，因为它们存在于不同的对象名称空间中: 例如，本地队列和发送方通道都可以具有相同的名称。 但是，一个对象不能与同一名称空间中的另一个对象具有相同的名称：例如，本地队列不能与模型队列具有相同的名称，发送方通道不能与接收方通道具有相同的名称。

以下 IBM MQ 对象存在于单独的对象名称空间中:

- 认证信息
- 通道
- 客户机通道
- 侦听器
- 名称列表
- 进程
- 队列
- 服务
- 存储类
- 预订
- Topic

对象名称的字符长度

通常， IBM MQ 对象名的长度最多可以为 48 个字符。 此规则适用于以下对象:

- 认证信息
- 集群
- 侦听器
- 名称列表
- 进程定义
- 队列
- 队列管理器
- 服务

- 预订
- Topic

存在以下限制:

1. z/OS 在 z/OS 系统上，队列管理器必须最多为 4 个字符，并且必须仅为大写字符和数字字符。
2. 通道对象名和客户机连接通道名的最大长度为 20 个字符。 请参阅 定义通道 以获取有关通道的更多信息。
3. 主题字符串最多可以为 10240 个字节。 所有 IBM MQ 对象名都区分大小写。
4. 预订名称最多可以为 10240 字节，并且可以包含空格。
5. 存储类名的最大长度为 8 个字符。
6. CF 结构名称的最大长度为 12 个字符。

对象名称中的字符

IBM MQ 对象名的有效字符为:

个字符之后	限制
大写 A-Z	• 无
小写 a-z	• 在 MQSC 脚本中，必须将带有小写字符的名称括在单引号中。 这将防止将小写字符转换为大写。 • 使用 EBCDIC 片假名的系统不能在对象名中使用小写 a-z 字符。 • z/OS 在 z/OS 系统上使用小写字符时可能存在限制，例如，队列管理器名称不能包含小写字符。 • IBM i 在 IBM i 系统上使用 CL 命令时，必须将带有小写字符的名称括在单引号中。 这将防止将小写字符转换为大写。
数字 0-9	• 无
句点 (.)	• 无
下划线 (_)	• Multi 无 • z/OS 避免使用带有前导或尾部下划线的名称，因为 IBM MQ for z/OS 操作和控制面板无法处理这些名称。
正斜杠 (/)	• Windows 在 Windows 系统上，队列管理器名称的第一个字符不能是正斜杠。 • IBM i 在 IBM i 系统上使用 CL 命令时，必须将包含正斜杠的名称括在单引号中。 • z/OS 无

个字符之后	限制
百分号 (%)	• ALW 无 • z/OS 如果要将 RACF 用作 IBM MQ for z/OS 的外部安全性管理器，请不要在对象名中使用%，因为在使用 RACF 通用概要文件时，这些名称不会包含在安全性检查中。 • IBM i 在 IBM i 系统上使用 CL 命令时，必须将包含百分号的名称括在单引号中。

还有一些关于对象名上的字符的一般规则:

1. 不允许前导空白或嵌入空白。
2. 不允许使用本地语言字符。
3. 任何小于完整字段长度的名称都可以用空格填充到右边。 队列管理器返回的所有短名称总是用空格填充到右边。

队列名称

队列的名称有两个部分:

- 队列管理器的名称
- 队列管理器已知的队列的局部名

队列名称的每个部分都有 48 个字符长。

要引用本地队列，可以省略队列管理器的名称 (通过将其替换为空白字符或使用前导空字符)。 但是，IBM MQ 返回到程序的所有队列名称都包含队列管理器的名称。

z/OS 共享队列 (可供其队列共享组中的任何队列管理器访问) 不能与同一队列共享组中的任何非共享本地队列同名。 此限制避免了应用程序在打算打开本地队列时误打开共享队列的可能性，反之亦然。 共享队列和队列共享组仅在 IBM MQ for z/OS 上可用。

要引用远程队列，程序必须在完整队列名称中包含队列管理器的名称，或者必须存在远程队列的本地定义。

当应用程序使用队列名称时，该名称可以是本地队列的名称 (或一个队列的别名)，也可以是远程队列的本地定义的名称，但应用程序不需要知道哪个，除非它需要从队列中获取消息 (当队列必须是本地队列时)。 当应用程序打开队列对象时， MQOPEN 调用将执行名称解析函数以确定要在哪个队列上执行后续操作。 这一点的重要意义在于，应用程序不依赖于在队列管理器网络中的特定位置定义的特定队列。 因此，如果系统管理员在网络中重新定位队列并更改其定义，那么不需要更改使用这些队列的应用程序。

保留对象名

将为队列管理器定义的对象保留以 SYSTEM. 开头的对象名。 您可以使用 **Alter**， **Define** 和 **Replace** 命令来更改这些对象定义以适合您的安装。 为 IBM MQ 定义的名称在 队列名称中完整列出。

z/OS 在 IBM MQ for z/OS 上，保留耦合设施应用程序结构名称 CSQSYSAPPL。

相关概念

AIX, Linux, and Windows 上的安装名称

IBM MQ 文件名

每个 IBM MQ 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，服务和认证信息对象都由一个文件表示。 由于对象名不一定是有效的文件名，因此队列管理器会在必要时将对象名转换为有效的文件名。

队列管理器目录的缺省路径如下所示:

- 在 IBM MQ 配置信息中定义的前缀:
  - Linux AIX 在 AIX and Linux 上，缺省前缀为 `/var/mqm`。 这是在 mqs.ini 配置文件的 `DefaultPrefix` 节中配置的。
  - Windows 在 Windows 32 位系统上，缺省前缀为 `C:\Program Files (x86)\IBM\WebSphere MQ`。 在 Windows 64 位系统上，缺省前缀为 `C:\Program Files\IBM\MQ`。 对于 32 位和 64 位安装，数据目录将安装到 `C:\ProgramData\IBM\MQ` 中。 这是在 mqs.ini 配置文件的 `DefaultPrefix` 节中配置的。

  如果可用，可以使用 IBM MQ Explorer 中的 IBM MQ 属性页面来更改前缀，否则请手动编辑 `mqs.ini` 配置文件。
- 队列管理器名称将变换为有效的目录名称。 例如，队列管理器:

```
queue.manager
```

  将表示为:

```
queue!manager
```

  此过程称为 名称变换。

在 IBM MQ 中，可以为队列管理器提供最多包含 48 个字符的名称。

例如，可以命名队列管理器:

```
QUEUE.MANAGER.ACCOUNTING.SERVICES
```

但是，每个队列管理器都由一个文件表示，并且对文件名的最大长度以及名称中可以使用的字符都有限制。因此，将自动变换表示对象的文件的名称以满足文件系统的要求。

管理队列管理器名称变换的规则如下所示:

1. 变换单个字符:
   - 从。 到!
   - From / to &
2. 如果名称仍然无效:
   a. 将其截断为 8 个字符
   b. 附加三字符数字后缀

例如，假定缺省前缀和名为 `queue.manager` 的队列管理器:

- Windows 在具有 NTFS 或 FAT32 的 Windows 上，队列管理器名称变为:

```
C:\Program Files\IBM\MQ\qmgrs\queue!manager
```

- Windows 在具有 FAT 的 Windows 上，队列管理器名称变为:

```
C:\Program Files\IBM\MQ\qmgrs\queue!ma
```

- Linux AIX 在 AIX and Linux 上，队列管理器名称变为:

```
/var/mqm/qmgrs/queue!manager
```

变换算法还区分仅在不区分大小写的文件系统上不同的名称。

对象名变换

对象名不一定是有效的文件系统名。 您可能需要变换对象名。 所使用的方法与队列管理器名称的方法不同，因为虽然每台机器上只有几个队列管理器名称，但每个队列管理器可能有大量其他对象。 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在文件系统中表示。

当变换过程生成新名称时，与原始对象名没有简单关系。 您可以使用 **dspmqfls** 命令在实对象名和已变换对象名之间进行转换。

相关参考

dspmqfls (显示文件名)

相关信息

mqs.ini 文件的 AllQueueManagers 节

IBM i IBM i 上的对象名

队列管理器具有具有唯一名称的关联队列管理器库。 可能需要变换队列管理器名称和对象名以满足 IBM i Integrated File System (IFS) 的需求。

创建队列管理器时， IBM MQ 会将队列管理器库与其相关联。 此队列管理器库具有唯一名称，长度不超过 10 个字符，这在很大程度上取决于用户定义的队列管理器名称。 队列管理器和队列管理器库都放在一个目录中，该目录也基于具有前缀 /QIBM/UserData/mqm 的队列管理器名称。 以下是队列管理器，队列管理器库和目录的示例:

队列管理器名称	橙色
队列管理器库名	Qmorange
目录	/QIBM/UserData/mqm/ORANGE

所有队列管理器名称和队列管理器库名都将写入文件 /QIBM/UserData/mqm/mqs.ini 中的节。

IBM MQ IFS 目录和文件

IBM i Integrated File System (IFS) 由 IBM MQ 广泛用于存储数据。 有关 IFS 的更多信息，请参阅 *Integrated File System* 简介。

每个 IBM MQ 对象 (例如，通道或队列管理器) 都由一个文件表示。 由于对象名不一定是有效的文件名，因此队列管理器会在必要时将对象名转换为有效的文件名。

队列管理器目录的路径由以下内容构成:

- 在队列管理器配置文件 qm.ini 中定义的前缀。 缺省前缀为 /QIBM/UserData/mqm。
- 文字 qmgrs。
- 编码的队列管理器名称，即转换为有效目录名的队列管理器名称。 例如，队列管理器 queue/manager 由 queue&manager 表示。

此过程称为名称变换。

IFS 队列管理器名称变换

在 IBM MQ 中，可以为队列管理器提供最多包含 48 个字符的名称。

例如，可以将队列管理器命名为 QUEUE/MANAGER/ACCOUNTING/SERVICES。 与为每个队列管理器创建库的方式相同，每个队列管理器也由一个文件表示。 由于 EBCDIC 中的变体代码点，因此可以在名称中使用的字符存在限制。 因此，表示对象的 IFS 文件的名称会自动变换以满足文件系统的要求。

使用名称为 queue/manager 的队列管理器示例，将字符 / 转换为 &，并采用缺省前缀， IBM MQ for IBM i 中的队列管理器名称将变为 /QIBM/UserData/mqm/qmgrs/queue&manager。

对象名变换

对象名不一定是有效的文件系统名，因此可能需要变换对象名。 所使用的方法与队列管理器名称的方法不同，因为虽然每台机器只有几个队列管理器名称，但每个队列管理器可能有大量其他对象。 只有进程定义，队列和名称列表在文件系统中表示; 通道不受这些注意事项影响。

当变换过程生成新名称时，与原始对象名没有简单关系。 可以使用 DSPMQMOBJN 命令来查看 IBM MQ 对象的已变换名称。

分布式排队和集群

分布式排队意味着将消息从一个队列管理器发送到另一个队列管理器。 接收队列管理器可以在同一台机器上，也可以在另一台机器上; 可以在附近，也可以在世界的另一侧。 它可以在与本地队列管理器相同的平台上运行，也可以在 IBM MQ 支持的任何平台上运行。 您可以手动定义分布式排队环境中的所有连接，也可以创建集群并让 IBM MQ 为您定义许多连接详细信息。

分布式 排队

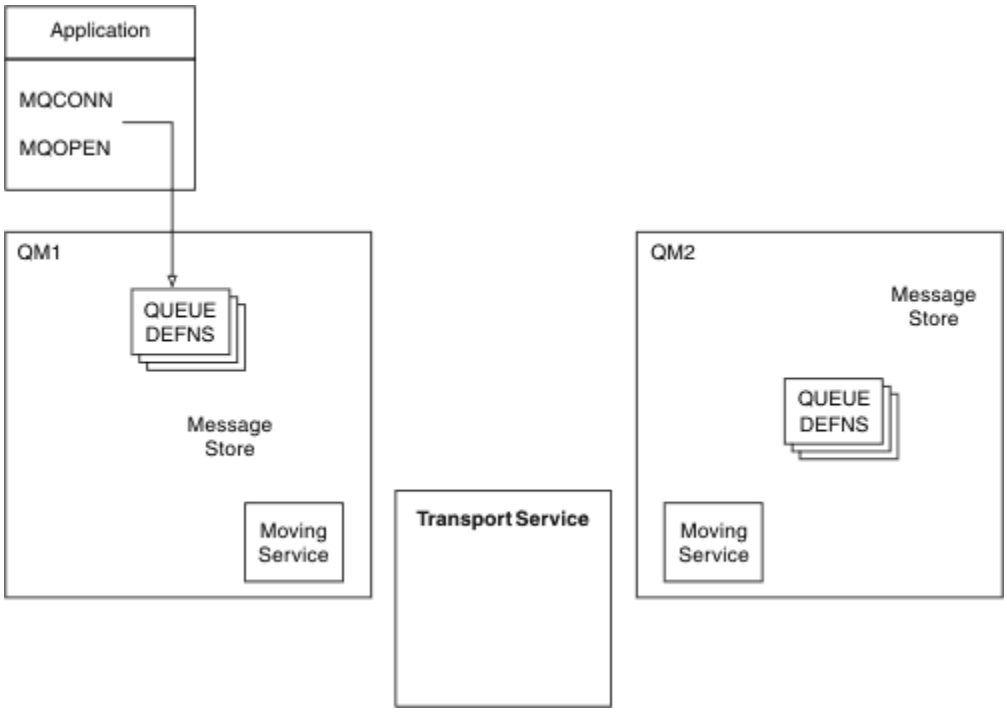


图 *4:* 分布式排队组件概述

在上图中:

- 应用程序使用 MQCONN 调用来连接到队列管理器。 然后，应用程序使用 MQOPEN 调用来打开队列，以便它可以将消息放入队列中。
- 每个队列管理器都有其每个队列的定义。 它可以具有 本地队列 (即，由此队列管理器托管) 的定义以及 远程队列 (即，由其他队列管理器托管) 的定义。
- 如果消息以远程队列为目标，那么本地队列管理器会将它们保存在 传输队列上，该队列会将它们持久存储在消息存储器中，直到可以将它们转发到远程队列管理器为止。
- 每个队列管理器都包含通信软件 (称为 移动服务) ，队列管理器使用这些软件与其他队列管理器进行通信。
- 传输服务 独立于队列管理器，可以是下列任何一项 (取决于平台):
  - 系统网络体系结构高级程序间通信 (SNA APPC)
  - 传输控制协议/因特网协议 (Transmission Control Protocol/Internet Protocol，TCP/IP)
  - 网络基本输入/输出系统 (NetBIOS)
  - 顺序包交换 (SPX)

发送消息所需的组件

如果要将消息发送到远程队列管理器，那么本地队列管理器需要 传输队列 和 通道的定义。 通道是两个队列管理器之间的单向通信链路。 它可以携带发往远程队列管理器中任意数量的队列的消息。

通道的每个端都具有单独的定义，例如，将其定义为发送端或接收端。 简单通道由本地队列管理器上的 发送方 通道定义和远程队列管理器上的 接收方 通道定义组成。 这两个定义必须具有相同的名称，并且它们共同构成一个通道。

用于处理消息发送和接收的软件称为 消息通道代理程序 (MCA)。 在通道的每一端都有一个 消息通道代理程序 (MCA)。

每个队列管理器都应该具有 死信队列 (也称为 未传递的消息队列)。 如果无法将消息传递到其目标，那么会将这些消息放入此队列。

下图显示了队列管理器，传输队列，通道和 MCA 之间的关系:

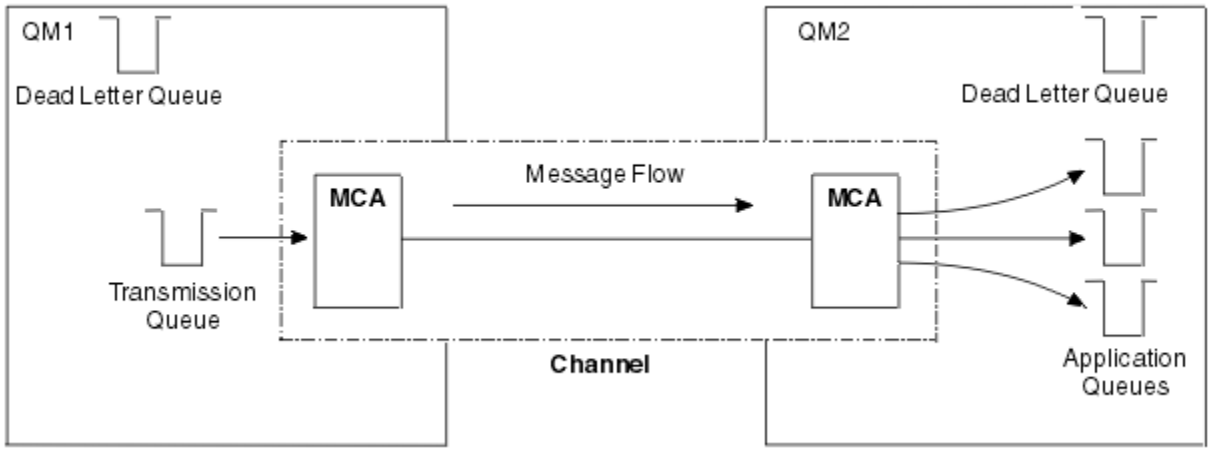


图 *5:* 发送消息

返回消息所需的组件

如果应用程序要求从远程队列管理器返回消息，那么需要定义另一个通道，以便在队列管理器之间以相反的方向运行，如下图所示:

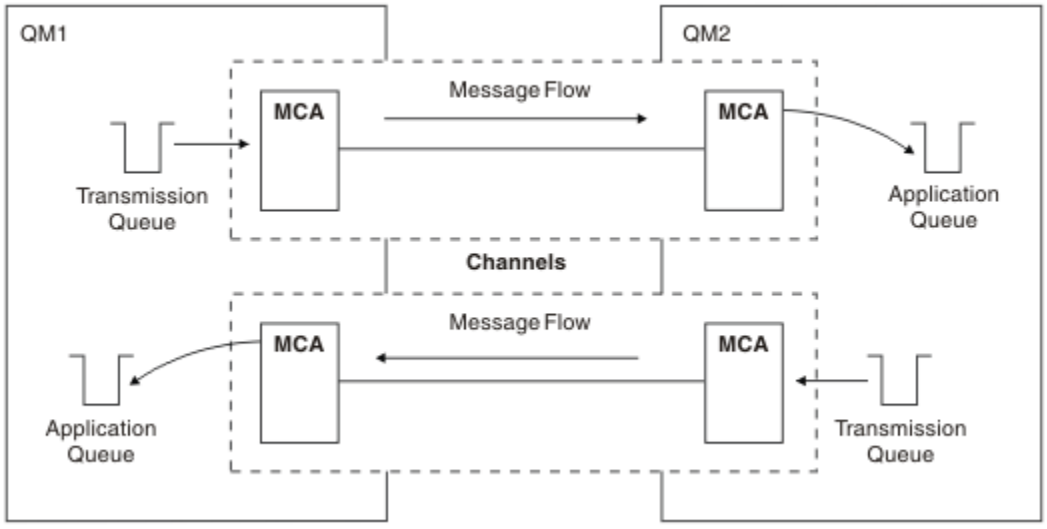


图 *6:* 双向发送消息

集群

您可以在集群中对一组队列管理器进行分组，而不是手动定义分布式排队环境中的所有连接。 执行此操作时，队列管理器可以使其托管的队列可供集群中的其他队列管理器使用，而无需针对每个目标的显式通道定义，远程队列定义或传输队列。 集群中的每个队列管理器都有一个传输队列，用于将消息传输到集群中的任何其他队列管理器。 对于每个队列管理器，您只需要定义一个集群接收方通道和一个集群发送方通道; 任何其他通道都由集群自动管理。

IBM MQ 客户机可以连接到属于集群的队列管理器，就像它可以连接到任何其他队列管理器一样。 与手动配置的分布式排队一样，您使用 MQPUT 调用将消息放入任何队列管理器中的队列。 您可以使用 MQGET 调用从本地队列中检索消息。

支持集群的平台上的队列管理器不必是集群的一部分。 您可以继续手动配置分布式排队，也可以使用集群来代替分布式排队。

使用集群的优点

集群提供了两个关键优势:

- 集群简化了 IBM MQ 网络的管理，通常需要为要配置的通道，传输队列和远程队列提供许多对象定义。 在许多队列管理器需要互连的大型网络中，尤其存在这种情况。 这种架构特别难以配置和积极维护。
- 集群可用于在集群中的队列和队列管理器之间分配消息流量的工作负载。 此类分发允许将单个队列的消息工作负载分布到位于多个队列管理器上的该队列的等效实例中。 工作负载的这种分布可用于实现对系统故障的更大弹性，以及提高系统中特别活跃的消息流的缩放性能。 在这种环境中，分布式队列的每个实例都具有处理消息的使用应用程序。 有关更多信息，请参阅 使用集群进行工作负载管理。

如何在集群中路由消息

您可以将集群视为由良心系统管理员维护的队列管理器网络。 无论何时定义集群队列，系统管理员都会根据需要在其他队列管理器上自动创建相应的远程队列定义。

您不需要进行传输队列定义，因为 IBM MQ 在集群中的每个队列管理器上提供了传输队列。 此单个传输队列可用于将消息传递到集群中的任何其他队列管理器。 您不限于使用单个传输队列。 队列管理器可以使用多个传输队列来分隔发送到集群中每个队列管理器的消息。 通常，队列管理器使用单个集群传输队列。 您可以更改队列管理器属性 DEFCLXQ，以便队列管理器对集群中的每个队列管理器使用不同的集群传输队列。 您还可以手动定义集群传输队列。

所有加入集群的队列管理器都同意以这种方式工作。 他们发送有关自己和他们托管的队列的信息，并接收有关集群的其他成员的信息。

要确保在队列管理器变为不可用时不会丢失任何信息，请在集群中指定两个队列管理器以充当 完整存储库。 这些队列管理器存储有关集群中所有队列管理器和队列的完整信息集。 集群中的所有其他队列管理器仅存储有关这些队列管理器及其交换消息的队列的信息。 这些队列管理器称为 部分存储库。 有关更多信息，请参阅 第 46 页的『集群存储库』。

要成为集群的一部分，队列管理器必须具有两个通道: 集群发送方通道和集群接收方通道:

- 集群发送方通道是类似于发送方通道的通信通道。 必须在队列管理器上手动创建一个集群发送方通道，以将其连接到已是集群成员的完整存储库。
- 集群接收方通道是类似于接收方通道的通信通道。 必须手动创建一个集群接收方通道。 通道充当队列管理器接收集群通信的机制。

然后，将自动创建此队列管理器与集群的其他成员之间的通信所需的所有其他通道。

下图显示了名为 CLUSTER 的集群的组件:

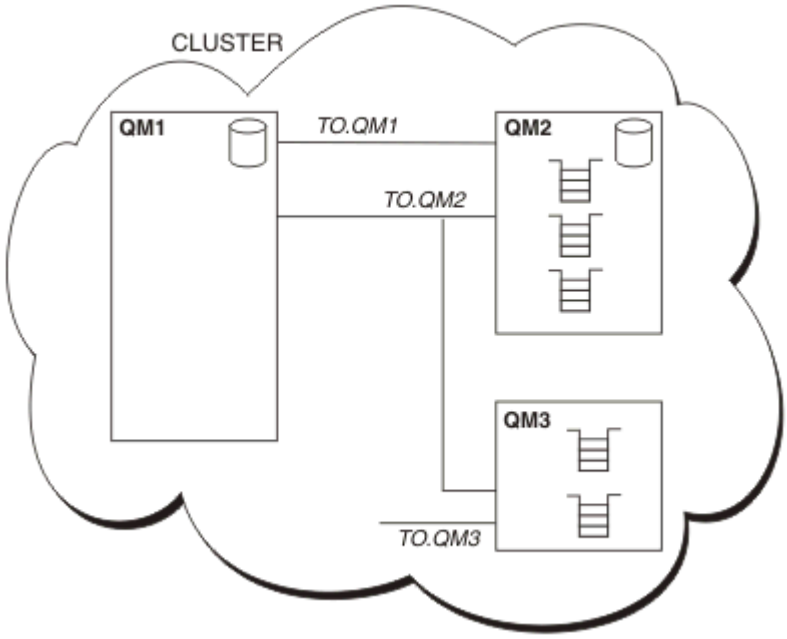


图 7: 队列管理器集群

- CLUSTER 包含三个队列管理器: QM1， QM2 和 QM3。
- QM1 和 QM2 托管有关集群中的队列管理器和队列的信息的完整存储库。
- QM2 和 QM3 托管一些集群队列，即可供集群中任何其他队列管理器访问的队列。
- 每个队列管理器都有一个名为 TO.qmgr 的集群接收方通道，可以在该通道上接收消息。
- 每个队列管理器还具有一个集群发送方通道，在该通道上，它可以将信息发送到其中一个存储库队列管理器。
- QM1 和 QM3 发送到 QM2 上的存储库， QM2 发送到 QM1 上的存储库。

分布式排队组件

分布式排队的组件包括消息通道，消息通道代理程序，传输队列，通道启动器和侦听器以及通道出口程序。消息通道的每个端的定义可以是多种类型之一。

消息通道是将消息从一个队列管理器传递到另一个队列管理器的通道。 请勿将消息通道与 MQI 通道混淆。有两种类型的 MQI 通道: 服务器连接 (SVRCONN) 和客户机连接 (CLNTCONN)。 有关更多信息，请参阅 通道。

消息通道的每个端的定义可以是下列其中一种类型:

- 发件人 (SDR)
- 接收器 (RCVR)
- 服务器 (SVR)
- 请求者 (RQSTR)
- 集群发送方 (CLUSSDR)
- 集群接收方 (CLUSRCVR)

使用在一端定义的其中一种类型和在另一端定义的兼容类型来定义消息通道。 可能的组合包括:

- 发送方-接收方
- 请求方-服务器
- 请求方-发送方（回调）

- 服务器-接收方
- 集群发送方-集群接收方

有关创建发送方/接收方通道的详细指示信息包含在 定义通道中。 有关设置发送方/接收方通道所需的参数的示例，请参阅适用于您的平台的 示例配置信息 。 有关定义任何类型的通道所需的参数，请参阅 DEFINE CHANNEL。

发送方-接收方通道

一个系统中的发送方启动通道，以便可以将消息发送到另一个系统。 发送方请求通道另一端的接收方启动。 发送方将消息从其传输队列发送到接收方。 接收方将消息放在目标队列上。 第 39 页的图 8 对此进行了说明。

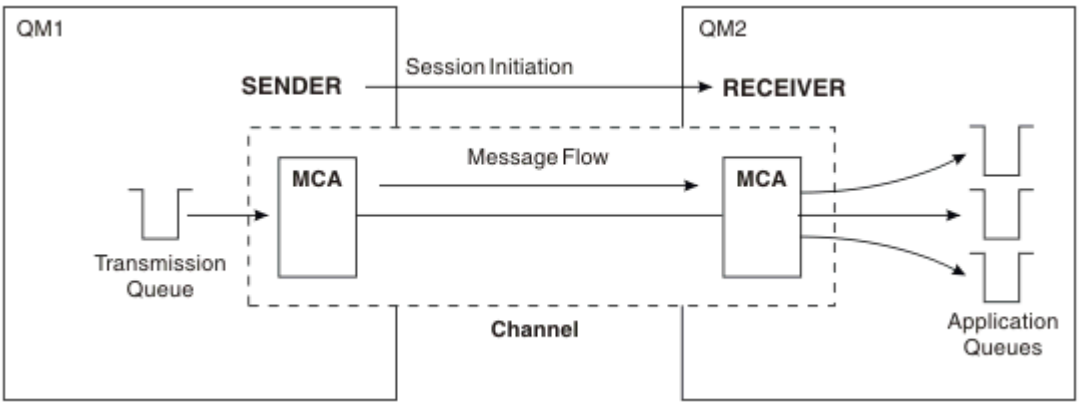


图 *8:* 一种发送方-接收方通道

请求方-服务器通道

一个系统中的请求者启动通道，以便它可以从另一个系统接收消息。 请求者请求通道另一端的服务器启动。 服务器从其通道定义中定义的传输队列向请求者发送消息。

服务器通道还可以启动通信并向请求者发送消息。 这仅适用于 标准 服务器，即在通道定义中指定了伙伴的连接名称的服务器通道。 标准服务器可以由请求者启动，也可以启动与请求者的通信。

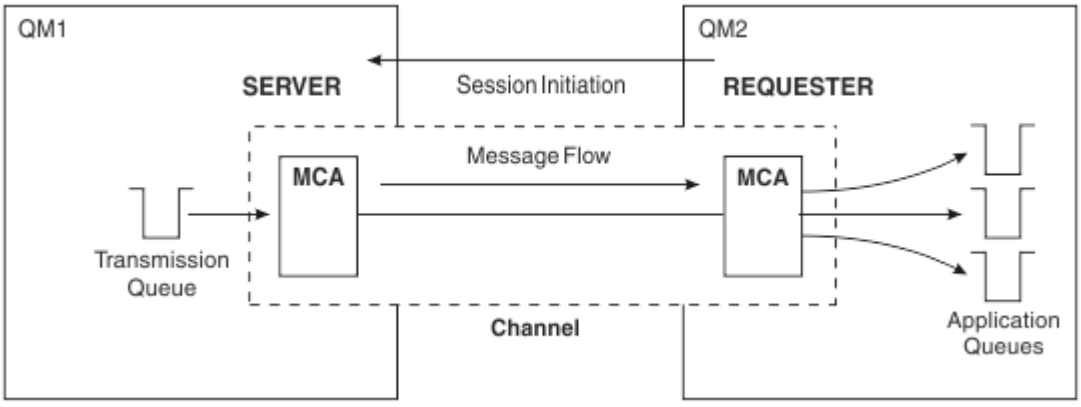


图 *9:* 请求者-服务器通道

请求方-发送方通道

请求者启动通道，发送方终止呼叫。 然后，发送方根据其通道定义 (称为 回调) 中的信息重新启动通信。 它将消息从传输队列发送到请求者。

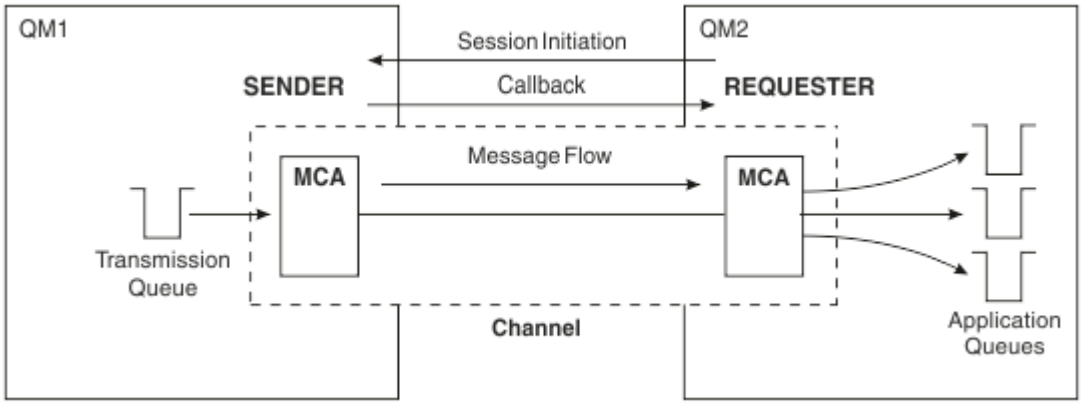


图 *10:* 请求者-发送方通道

服务器-接收方通道

这类似于发送方/接收方，但仅适用于 标准 服务器，即具有在通道定义中指定的伙伴的连接名称的服务器通道。 必须在链路的服务器端启动通道启动。 此图的插图与 第 39 页的图 8 中的插图类似。

集群发送方通道

在集群中，每个队列管理器都有一个集群发送方通道，在该通道上可以将集群信息发送到其中一个完整的存储库队列管理器。 队列管理器还可以将消息发送到集群发送方通道上的其他队列管理器。

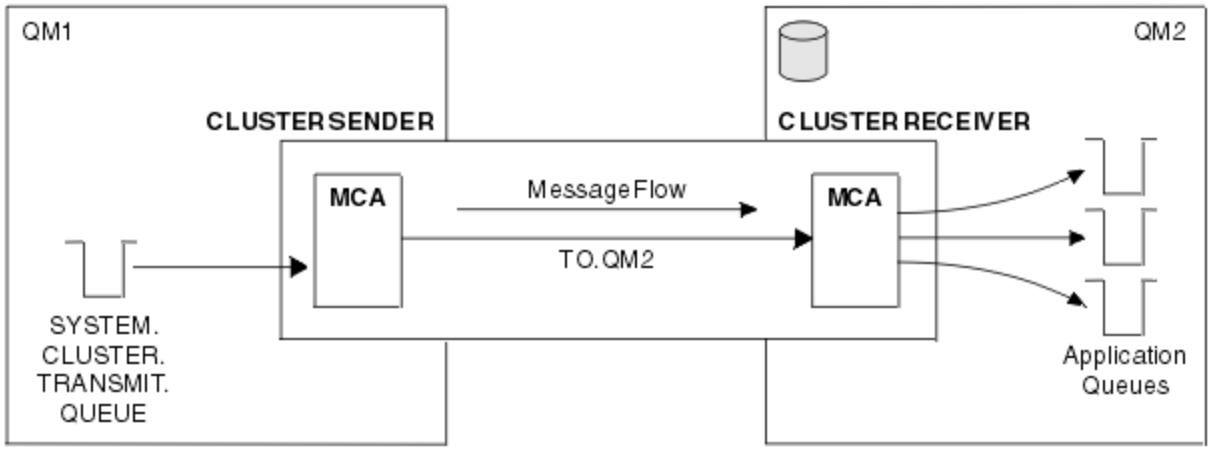


图 *11:* 集群发送方通道

集群接收方通道

在集群中，每个队列管理器都有一个集群接收方通道，在该通道上可以接收有关集群的消息和信息。 此图的插图与 第 40 页的图 11 中的插图类似。

死信队列

死信队列 (或未传递的消息队列) 是无法将消息路由到其正确目标时将消息发送到的队列。 每个队列管理器通常都有一个死信队列。

死信队列 (DLQ) (有时称为 未传递的消息队列) 是无法传递到其目标队列的消息的保留队列，例如，由于该队列不存在或已满。 对于数据转换错误，在通道的发送端也会使用死信队列。 网络中的每个队列管理器通常都有一个要用作死信队列的本地队列，以便无法传递到其正确目标的消息可以存储以供以后检索。

消息可由队列管理器，消息通道代理程序 (MCA) 和应用程序放在 DLQ 上。 死信队列上的所有消息都必须以 死信头 结构 MQDLH 作为前缀。 MQDLH 结构的 原因 字段包含标识消息在 DLQ 上的原因的原因码。

通常应该为每个队列管理器定义死信队列。 如果没有，并且 MCA 无法放入消息，那么会将其保留在传输队列上并停止通道。 此外，如果是快速的非持久消息 (请参阅 快速的非持久消息) 无法传递，并且目标系统上不存在死信队列，将废弃这些消息。

但是，使用死信队列可能会影响传递消息的顺序，因此您可能选择不使用这些消息。

相关任务

使用死信队列

未送达消息故障诊断

相关参考

runmqdlq (运行死信队列处理程序)

远程队列定义

远程队列定义是另一个队列管理器所拥有的队列的定义。

而应用程序只能从本地队列检索消息，它们可以将消息放在本地队列或远程队列上。 因此，除了其每个本地队列的定义外，队列管理器还可以具有 远程队列定义。 远程队列定义的优点是使应用程序能够将消息放入远程队列，而不必指定远程队列或远程队列管理器的名称或传输队列的名称。 远程队列定义使您具有位置独立性。

远程队列定义还有其他用途，稍后将进行描述。

如何访问远程队列管理器

您可能并非总是在每个源队列管理器与目标队列管理器之间具有一个通道。 两者之间还有很多其他的链接方式，包括多跳，共享通道，使用不同的通道和集群。

多跳跃

如果源队列管理器与目标队列管理器之间没有直接通信链路，那么可以在到达目标队列管理器的路上通过一个或多个 中间队列管理器 。 这称为 多跳。

您需要定义所有队列管理器之间的通道，以及中间队列管理器上的传输队列。 第 41 页的图 12 对此作了说明。

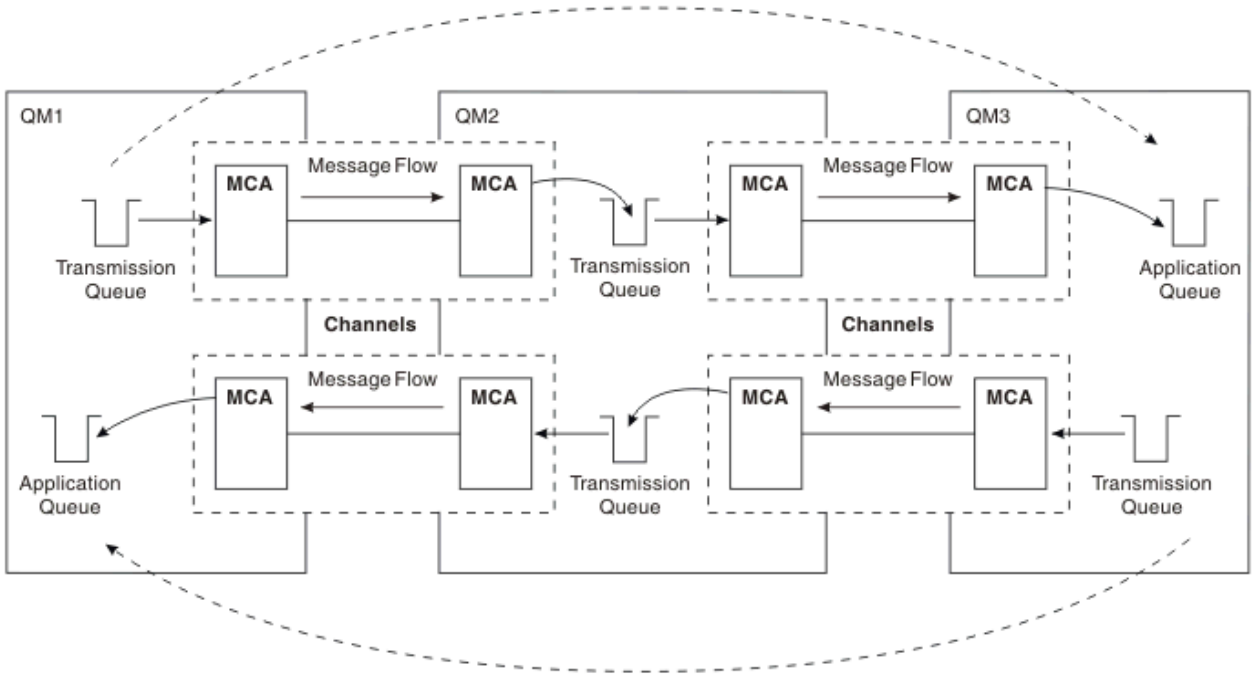


图 *12:* 通过中间队列管理器

共享通道

作为应用程序设计者，您可以选择强制应用程序指定远程队列管理器名称以及队列名称，或者为每个远程队列创建 远程队列定义 。 此定义保存远程队列管理器名称，队列名称和传输队列的名称。 无论哪种方式，来自同一远程位置的所有寻址队列的应用程序的所有消息都通过同一传输队列发送其消息。 第 42 页的图 13 对此作了说明。

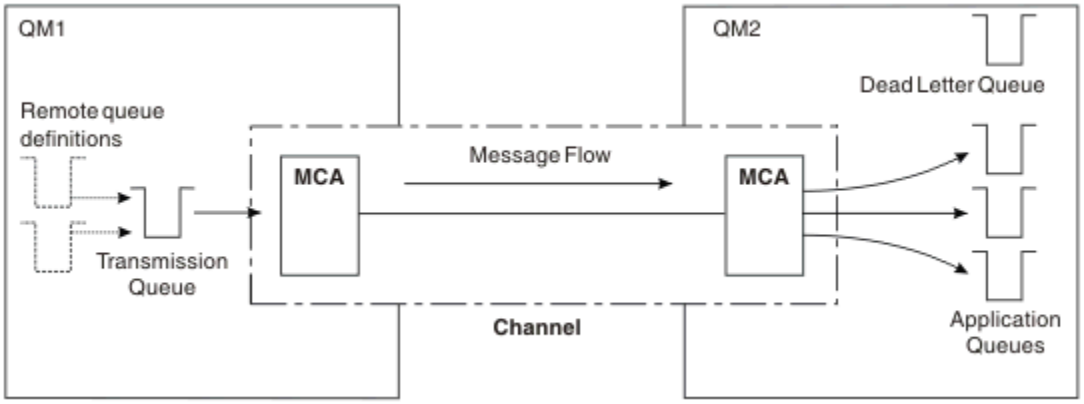


图 *13:* 共享传输队列

第 42 页的图 13 说明从多个应用程序到多个远程队列的消息可以使用同一通道。

使用不同的通道

如果要在两个队列管理器之间发送不同类型的消息，那么可以在这两个队列管理器之间定义多个通道。 有时，您需要备用通道 (可能出于安全目的) ，或者需要将交付速度与大量消息流量进行比较。

要设置第二个通道，您需要定义另一个通道和另一个传输队列，并创建指定位置和传输队列名称的远程队列定义。 然后，应用程序可以使用任一通道，但消息仍会传递到相同的目标队列。 第 42 页的图 14 对此作了说明。

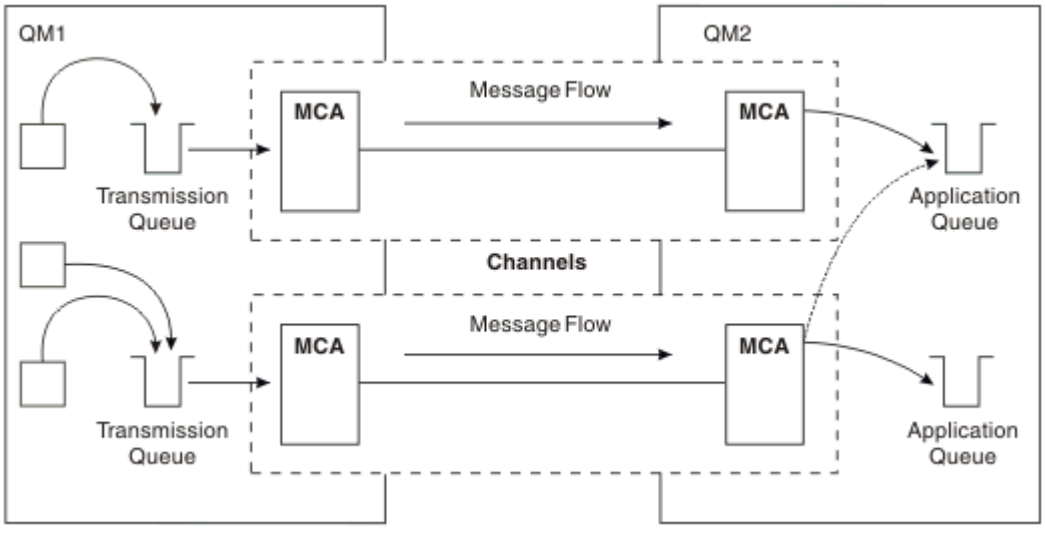


图 *14:* 使用多个通道

使用远程队列定义来指定传输队列时，应用程序 **不得** 指定位置 (即目标队列管理器) 本身。 如果这样做，那么队列管理器不会使用远程队列定义。 远程队列定义使您具有位置独立性。 应用程序可以在不知道队列所在位置的情况下将消息放入 逻辑 队列，并且您可以更改 物理 队列，而不必更改应用程序。

使用集群

集群中的每个队列管理器都定义一个集群接收方通道。 当另一个队列管理器想要向该队列管理器发送消息时，它会自动定义相应的集群发送方通道。 例如，如果集群中有多个队列实例，那么可以向托管该队列的任何队列管理器定义集群发送方通道。 IBM MQ 使用工作负载管理算法，该算法使用循环法例程来选择要将消息路由到的可用队列管理器。 有关更多信息，请参阅 集群。

寻址信息

当应用程序将发往远程队列管理器的消息放入时，本地队列管理器会在将这些消息放入传输队列之前向它们添加传输头。 此头包含目标队列和队列管理器的名称，即 寻址信息。

在单个队列管理器环境中，当应用程序打开要将消息放入的队列时，将建立目标队列的地址。 由于目标队列位于同一队列管理器上，因此不需要任何寻址信息。

在分布式排队环境中，队列管理器不仅需要知道目标队列名称，还需要知道该队列的位置 (即，队列管理器名称) 以及到该远程位置的路径 (即，传输队列)。 此寻址信息包含在传输头中。 接收通道除去传输头并使用其中的信息来查找目标队列。

如果使用远程队列定义，那么可以避免应用程序需要指定目标队列管理器的名称。 此定义指定远程队列的名称，将消息发送至的远程队列管理器的名称以及用于传输消息的传输队列的名称。

什么是别名?

别名用于为消息提供服务质量。 队列管理器别名使系统管理员能够更改目标队列管理器的名称，而不必更改应用程序。 它还使系统管理员能够更改到目标队列管理器的路由，或者设置涉及通过多个其他队列管理器 (多跳跃) 的路由。 应答队列别名为应答提供服务质量。

队列管理器别名和应答队列别名是使用具有空白 RNAME 的远程队列定义创建的。 这些定义不定义实际队列; 队列管理器使用这些定义来解析物理队列名称，队列管理器名称和传输队列。

别名定义的特征是具有空白 RNAME。

队列名称解析

每次打开队列时，都会在每个队列管理器上解析队列名称。 其目的是标识目标队列，目标队列管理器 (可能是本地队列) 以及到该队列管理器的路径 (可能为空)。 解析的名称有三个部分: 队列管理器名称，队列名称以及传输队列 (如果队列管理器是远程的)。

当存在远程队列定义时，不会引用任何别名定义。 应用程序提供的队列名称解析为目标队列，远程队列管理器和远程队列定义中指定的传输队列的名称。 有关队列名称解析的更多详细信息，请参阅 队列名称解析。

如果没有远程队列定义，并且指定了队列管理器名称或由名称服务解析，那么队列管理器将查看是否存在与提供的队列管理器名称匹配的队列管理器别名定义。 如果存在，那么其中的信息将用于将队列管理器名称解析为目标队列管理器的名称。 队列管理器别名定义还可用于确定到目标队列管理器的传输队列。

如果解析的队列名称不是本地队列，那么队列管理器名称和队列名称都包含在应用程序放入传输队列的每条消息的传输头中。

除非由远程队列定义或队列管理器别名定义更改，否则所使用的传输队列通常具有与已解析的队列管理器相同的名称。 如果您尚未定义此类传输队列，但已定义缺省传输队列，那么将使用此传输队列。

z/OS 在 z/OS 上运行的队列管理器的名称限制为四个字符。

队列管理器别名定义

当打开队列以放入消息的应用程序指定队列名称 **和** 队列管理器名称时，队列管理器别名定义适用。

队列管理器别名定义有三种用途:

- 发送消息时，重新映射队列管理器名称
- 发送消息时，更改或指定传输队列
- 接收消息时，确定本地队列管理器是否是这些消息的预期目标

出站消息-重新映射队列管理器名称

队列管理器别名定义可用于重新映射 MQOPEN 调用中指定的队列管理器名称。 例如， MQOPEN 调用指定队列名称 THISQ 和队列管理器名称 YOURQM。 在本地队列管理器上，存在类似于以下示例的队列管理器别名定义:

```
DEFINE QREMOTE (YOURQM) RQMNAME(REALQM)
```

这表明当应用程序将消息放入队列管理器 YOURQM 时要使用的实际队列管理器是 REALQM。 如果本地队列管理器为 REALQM，那么它会将消息放入队列 THISQ(这是本地队列)。 如果本地队列管理器未被称为 REALQM，那么它会将消息路由到名为 REALQM 的传输队列。 队列管理器将传输头更改为 REALQM 而不是 YOURQM。

出站消息-更改或指定传输队列

第 44 页的图 15 显示了消息到达队列管理器 QM1 的场景，其中传输头显示队列管理器 QM3 上的队列名称。 在此场景中，可通过 QM2 进行多跳操作来访问 QM3 。

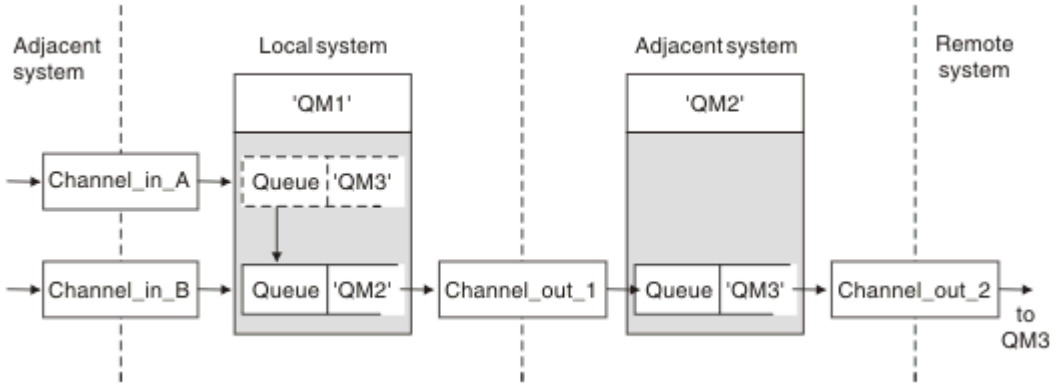


图 *15:* 队列管理器别名

QM3 的所有消息都是在 QM1 上使用队列管理器别名捕获的。 队列管理器别名名为 QM3 ，包含通过传输队列 QM2 的定义 QM3 。 该定义类似于以下示例:

```
DEFINE QREMOTE (QM3) RNAME(' ') RQMNAME(QM3) XMITQ(QM2)
```

队列管理器将消息放在传输队列 QM2 上，但不会改变传输队列头，因为目标队列管理器 QM3 的名称不会改变。

所有到达 QM1 并在 QM2 上显示包含队列名称的传输头的消息也将放在 QM2 传输队列上。 通过这种方式，将具有不同目标的消息收集到公共传输队列中到相应的相邻系统，以便继续传输到它们的目标。

入站消息-确定目标

接收 MCA 打开传输头中引用的队列。 如果存在与所引用的队列管理器同名的队列管理器别名定义，那么在传输头中接收到的队列管理器名称将替换为该定义中的 RQMNAME。

此过程有两种用途:

- 将消息定向到另一个队列管理器
- 将队列管理器名称更改为与本地队列管理器相同

应答队列别名定义

应答队列别名定义指定消息描述符中应答信息的备用名称。 这样做的优点是可以更改队列或队列管理器的名称，而不必更改应用程序。

队列名称解析

当应用程序应答消息时，它使用它接收到的消息的 消息描述符 中的数据来找出要应答的队列的名称。 发送应用程序指示将应答发送到的位置，并将此信息附加到其消息。 此概念必须作为应用程序设计的一部分进行协调。

在将消息放入队列之前，将在应用程序的发送端进行队列名称解析。 因此，在与要向其发送消息的远程应用程序进行交互之前，将进行队列名称解析。 这是在未打开队列的情况下进行名称解析的唯一情况。

使用队列管理器别名解析队列名称

通常，应用程序指定应答队列，并将应答队列管理器名称留空。 队列管理器在放入时完成其自己的名称。 除非您希望将备用通道用于应答 (例如，使用传输队列 `QM1_relief` 而不是使用传输队列 `QM1` 的缺省返回通道) ，否则此方法正常工作。 在此情况下，在传输队列头中指定的队列管理器名称与 "实际" 队列管理器名称不匹配，而是使用队列管理器别名定义重新指定。 为了在备用路由上返回应答，还需要使用应答队列别名定义来映射应答队列数据。

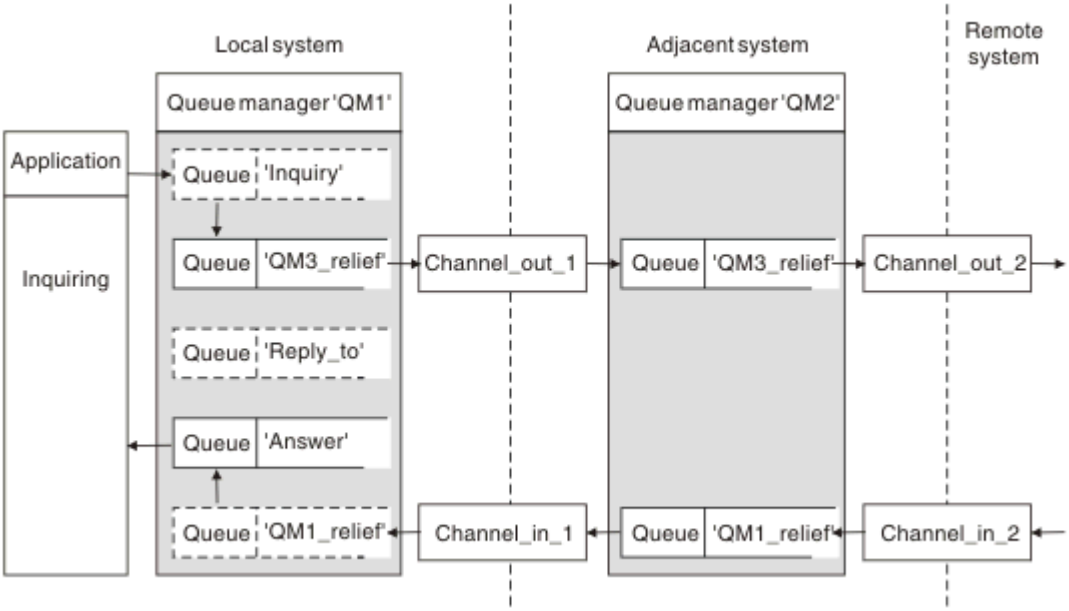


图 *16:* 用于更改应答位置的应答队列别名

在 第 45 页的图 16 中的示例中:

1. 应用程序使用 MQPUT 调用并在消息描述符中指定以下信息来放置消息:

   ```
   ReplyToQ='Reply_to'
   ReplyToQMgr=' '
   ```

   ReplyTo 队列管理器必须为空才能使用应答队列别名。
2. 创建名为 `Reply_to` 的应答队列别名定义，其中包含名称 `Answer` 和队列管理器名称 `QM1_relief`。

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```
3. 将使用显示 `ReplyToQ='Answer'` 和 `ReplyToQMgr='QM1_relief'`的消息描述符来发送消息。
4. 应用程序规范必须包含将在队列 `Answer` 而不是 `Reply_to` 中找到应答的信息。

要准备应答，必须创建并行返回通道，请定义:

- 在 QM2 上，名为 QM1_relief 的传输队列

```
DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
```

- 在 QM1 上，队列管理器别名为 QM1_relief

```
DEFINE QREMOTE ('QM1_relief') RNAME() RQMNAME(QM1)
```

此队列管理器别名将终止并行返回通道链，并捕获 QM1 的消息。

如果您认为您可能希望在将来某个时间执行此操作，请确保应用程序从一开始就使用别名。 现在，这是应答队列的正常队列别名，但稍后可以将其更改为队列管理器别名。

应答队列名称

命名应答队列时需要小心。 应用程序在消息中放入应答队列名称的原因是它可以指定将其应答发送到的队列。 使用此名称创建应答队列别名定义时，不能使用同名的实际应答队列 (即本地队列定义)。 因此，应答队列别名定义必须包含新的队列名称以及队列管理器名称，并且应用程序规范必须包含在此其他队列中找到其应答的信息。

应用程序现在必须从另一个队列中检索消息，而该队列在它们放置原始消息时被指定为应答队列。

集群组件

集群由队列管理器，集群存储库，集群通道和集群队列组成。

请参阅以下子主题以获取有关每个集群组件的信息:

相关概念
集群与分布式排队的比较
相关任务
配置队列管理器集群
设置新集群

集群存储库

存储库是有关作为集群成员的队列管理器的信息集合。

存储库信息包括队列管理器名称，它们的位置，它们的通道，它们托管的队列以及其他信息。 该信息以消息形式存储在名为 SYSTEM.CLUSTER.REPOSITORY.QUEUE 的队列上。 此队列是其中一个缺省对象。 Multi 在 多平台上，它是在创建 IBM MQ 队列管理器时定义的。 z/OS 在 IBM MQ for z/OS 上，它定义为队列管理器定制的一部分。

完整存储库和部分存储库

通常，集群中的两个队列管理器保存完整存储库。 其余队列管理器都持有部分存储库。

托管集群中每个队列管理器的完整信息集的队列管理器具有完整的存储库。 集群中的其他队列管理器具有部分存储库，其中包含完整存储库中的一部分信息。

部分存储库仅包含有关队列管理器需要与之交换消息的那些队列管理器的信息。 队列管理器请求更新它们需要的信息，因此如果它发生更改，那么完整存储库队列管理器将向它们发送新信息。 在大部分时间内，部分存储库包含队列管理器在集群中需要执行的所有信息。 当队列管理器需要某些其他信息时，它可查询完整存储库并更新其部分存储库。 队列管理器使用 SYSTEM.CLUSTER.COMMAND.QUEUE 队列来请求和接收对存储库的更新。

当 迁移 作为集群成员的队列管理器时，请先迁移完整存储库，然后再迁移部分存储库。 这是因为较旧的存储库无法存储较新发行版中引入的较新属性。 它容忍它们，但不存储它们。

集群队列管理器

集群队列管理器是属于集群的队列管理器。

一个队列管理器可以隶属于多个集群。 每个集群队列管理器都必须具有在其所属的所有集群中唯一的名称。

集群队列管理器可以托管队列，它会将这些队列通告给集群中的其他队列管理器。 但是，它不必这样做。 它可以改为将消息馈送到集群中其他位置托管的队列中，并且仅接收显式定向到该队列的响应。

z/OS 在 IBM MQ for z/OS 中，集群队列管理器可以是队列共享组的成员。 在这种情况下，它与同一队列共享组中的其他队列管理器共享其队列定义。

集群队列管理器是自主的。 他们可以完全控制他们定义的队列和通道。 其他队列管理器 (同一队列共享组中的队列管理器除外) 无法修改它们的定义。 存储库队列管理器不控制集群中其他队列管理器中的定义。 它们包含所有定义的完整集合，以便在需要时使用。 集群是队列管理器的联合。

在集群队列管理器上创建或更改定义后，会将信息发送到完整存储库队列管理器。 稍后将更新集群中的其他存储库。

完整存储库队列管理器

完整存储库队列管理器是一个集群队列管理器，用于保存集群资源的完整表示。 要确保可用性，请在每个集群中设置两个或更多完整存储库队列管理器。 完整存储库队列管理器接收集群中其他队列管理器发送的信息并更新其存储库。 它们相互发送消息以确保它们都与有关集群的新信息保持一致。

队列管理器和存储库

每个集群都有至少一个 (最好是两个) 队列管理器，这些队列管理器保存有关集群中的队列管理器，队列和通道的完整信息存储库。 这些存储库还包含来自集群中其他队列管理器的请求，用于更新信息。

其他队列管理器各自保存部分存储库，其中包含有关需要与其通信的队列和队列管理器的子集的信息。 队列管理器通过在首次需要访问另一个队列或队列管理器时进行查询来构建其部分存储库。 他们请求将有关该队列或队列管理器的任何新信息通知他们。

每个队列管理器都将其存储库信息存储在名为 SYSTEM.CLUSTER.REPOSITORY.QUEUE 的队列上的消息中。 队列管理器在名为 SYSTEM.CLUSTER.COMMAND.QUEUE 的队列上的消息中交换存储库信息。

连接集群的每个队列管理器都定义了一个集群发送方 CLUSSDR 通道，用于连接到其中一个存储库。 它会立即了解集群中的哪些其他队列管理器保存完整存储库。 从此，队列管理器可以从任何存储库请求信息。 当队列管理器将信息发送到所选存储库时，它还会将信息发送到另一个存储库 (如果有)。

当托管完整存储库的队列管理器从与其链接的其中一个队列管理器接收新信息时，将更新完整存储库。 还会将新信息发送到另一个存储库，以降低在存储库队列管理器无法使用时将其延迟的风险。 由于将两次发送所有信息，因此存储库必须废弃重复项。 每个信息项都带有一个序号，存储库使用该序号来标识重复项。 通过交换消息使所有存储库保持一致。

集群队列

集群队列是由集群队列管理器托管并可供集群中其他队列管理器使用的队列。

集群队列定义将播发给集群中的其他队列管理器。 集群中的其他队列管理器无需相应的远程队列定义即可将消息放入集群队列。 可以使用集群名称列表在多个集群中播发集群队列。

在播发队列时，集群中的任何队列管理器都可以将消息放入该队列中。 要放入消息，队列管理器必须从完整存储库中查明托管该队列的位置。 然后，它会将一些路由信息添加到消息，并将消息放到集群传输队列上。

集群队列可以是 IBM MQ for z/OS 中队列共享组的成员所共享的队列。

相关任务

定义集群队列

Comparison between shared queues and cluster queues

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Channel Initiator costs

In cluster queues, messages are sent by channels, so allow for channel initiator costs in addition to application costs. There are costs in the network because channels get and put messages. These costs are not present with shared queues, which therefore use less processing power than cluster queues when moving messages between queue managers in a queue sharing group.

Availability of messages

When putting to a queue, cluster queues send the message to one of the queue managers with active channels connected to your queue manager. On the remote queue manager, if applications used to process the messages are not working, the messages are not processed and wait until the applications start. Similarly, if a queue manager is shut down, any messages on the queue manager are not made available until the queue manager restarts. These instances show lower message availability than when using shared queues.

When using shared queues, any application in the queue sharing group can get messages that are sent. If you shut down one queue manager in the queue sharing group, messages are available to the other queue managers, providing higher message availability than when using cluster queues.

Capacity

A coupling facility is more expensive than a disk; therefore the cost of storing 1,000,000 messages in a local queue is lower than having a coupling facility with enough capacity to store the same number of messages.

Sending to other queue managers

Shared-queue messages are only available within a queue sharing group. If you want to use a queue manager outside of the queue sharing group, you must use channels. You can use clustering to workload balance between multiple remote distributed queue managers.

Workload balancing

You can use clustering to give weight to which channels and queue managers get a proportion of the messages sent. For example, you can send 60% of messages to one queue manager, and 40% of messages to another queue manager. This instance does not depend on the ability of the remote queue manager to process work. The system with the first queue manager might be overloaded, and the system with the second queue manager might be idle, but most of the messages still go to the first queue manager.

With shared queues, two CICS® systems can get messages. If one system is overloaded, the other system takes over most the workload.

集群通道

在每个完整存储库上，手动定义一个集群接收方通道和一组集群发送方通道，以连接到集群中的每个其他完整存储库。 添加部分存储库时，请手动定义集群接收方通道以及连接到其中一个完整存储库的单个集群发送方通道。 需要时，集群会自动定义其他集群发送方通道。 自动定义的集群发送方通道从接收队列管理器上的相应集群接收方通道定义中获取其属性。

集群接收方通道: CLUSRCVR

CLUSRCVR 通道定义定义了一个通道的末尾，集群队列管理器可以在该通道上接收来自集群中其他队列管理器的消息。

必须为每个集群队列管理器至少定义一个 CLUSRCVR 通道。 通过定义 CLUSRCVR 通道，队列管理器将显示它可用于接收消息的其他集群队列管理器。

CLUSRCVR 通道定义还使其他队列管理器能够自动定义相应的集群发送方通道定义。 请参阅本文的 第 49 页的『自动定义的集群发送方通道』 部分。

集群发送方通道: CLUSSDR

您可以手动定义从每个完整存储库队列管理器到集群中每个其他完整存储库队列管理器的 CLUSSDR 通道。 由完整存储库交换的所有更新仅在这些通道上流动。 通过手动定义这些通道，可以显式地控制完整存储库的网络。

将部分存储库队列管理器添加到集群时，请手动定义单个 CLUSSDR 通道以连接到其中一个完整存储库。 这与您选择的完整存储库差别不大，因为在进行初始联系之后，将根据需要自动定义队列管理器的更多集群队列管理器对象 (包括 CLUSSDR 通道)。 这使队列管理器能够将集群信息发送到任何完整存储库，并将消息发送到集群中的任何队列管理器。

如本文部分所述，自动定义的发送方通道基于集群接收方通道的配置。 因此，您在集群通道上设置的任何通道属性都应该在匹配的 CLUSSDR 和集群接收方通道上进行相同的设置，或者仅在集群接收方通道上设置。

由于先前描述的原因，您应该仅手动定义 CLUSSDR 通道。 即，最初将部分存储库连接到完整存储库，或者将两个完整存储库连接到一起。 手动配置连接到部分存储库或不在集群中的队列管理器的 CLUSSDR 通道会导致发出错误消息，例如 AMQ9427 和 AMQ9428 。 虽然这有时可能作为临时情况不可避免，例如在修改完整存储库的位置时，应该尽快删除手动定义。

自动定义的集群发送方通道

通常，将部分存储库队列管理器添加到集群时，仅手动定义队列管理器上的两个集群通道:

- 集群发送方 (CLUSSDR) 到集群的完整存储库队列管理器的通道。
- 集群接收方 (CLUSRCVR) 通道。

您定义的 CLUSSDR 通道允许队列管理器与集群进行初始联系。 进行初始联系后，集群会在需要时自动定义更多 CLUSSDR 通道。

自动定义的 CLUSSDR 通道从接收队列管理器上相应的 CLUSRCVR 通道定义中获取其属性。 即使存在手动定义的 CLUSSDR 通道，也会使用自动定义的 CLUSSDR 通道中的属性。 例如，假设您定义 CLUSRCVR 通道而不在 **CONNAME** 参数中指定端口号，并手动定义指定端口号的 CLUSSDR 通道。 当自动定义的 CLUSSDR 通道替换手动定义的通道时，端口号 (取自 CLUSRCVR 通道) 将变为空白。 将使用缺省端口号，并且通道将发生故障。

如果手动定义的 CLUSSDR 通道与相应的 CLUSRCVR 通道定义之间存在配置差异，那么某些差异将立即生效 (例如，工作负载均衡参数) ，而某些差异仅在通道重新启动 (例如， TLS 配置) 时生效。

为避免混淆，请尽可能遵守以下准则:

- 仅手动定义 CLUSSDR 通道以指向完整存储库。
- 如果您已手动定义 CLUSSDR 通道，请将其配置为与接收队列管理器上相应的 CLUSRCVR 通道定义完全匹配。

另请参阅 使用自动定义的通道。

相关概念

使用自动定义的通道

使用集群传输队列和集群发送方通道

相关任务

设置新集群

将队列管理器添加至集群

集群主题

集群主题是定义了 **cluster** 属性的管理主题。 有关集群主题的信息会推送给集群的所有成员，并与本地主题相结合以创建横跨多个队列管理器的主题空间的一些部分。 这支持将一个队列管理器中的某个主题上发布的消息传递给集群中其他队列管理器的预订。

在队列管理器上定义集群主题时，会将集群主题定义发送到完整存储库队列管理器。 然后，完整存储库会将集群主题定义传播到集群中的所有队列管理器，从而使相同的集群主题可用于集群中任何队列管理器处的发布者和订户。 您在其中创建集群主题的队列管理器称为集群主题主机。 集群主题可供集群中的任何队列管理器使用，但是对集群主题的任何修改必须在定义该主题的队列管理器（主机）上执行，此时会通过完整存储库将修改传播到集群的所有成员。

有关配置集群主题以使用 直接路由 或 主题主机路由的信息，以及有关集群主题继承和通配符预订的信息，请参阅 定义集群主题。

有关用于显示集群主题的命令的信息，请参阅相关信息。

相关概念

处理管理主题

使用预订

相关参考

显示 TOPIC

显示主题状态

显示

缺省集群对象

Multi 在多平台上，缺省集群对象包含在定义队列管理器时自动创建的缺省对象集中。

z/OS 在 z/OS 上，可以在定制样本中找到缺省集群对象定义。

注: 您可以通过运行 MQSC 或 PCF 命令，以与任何其他通道定义相同的方式更改缺省通道定义。 请勿更改缺省队列定义， SYSTEM.CLUSTER.HISTORY.QUEUE 除外。

SYSTEM.CLUSTER.COMMAND.QUEUE
集群中的每个队列管理器都有一个名为 SYSTEM.CLUSTER.COMMAND.QUEUE 的本地队列，用于将消息传输到完整存储库。 此消息包含有关队列管理器的任何新信息或已更改的信息，或者有关其他队列管理器的任何信息请求。 SYSTEM.CLUSTER.COMMAND.QUEUE 通常为空。

SYSTEM.CLUSTER.HISTORY.QUEUE
集群中的每个队列管理器都有一个名为 SYSTEM.CLUSTER.HISTORY.QUEUE 的本地队列。 SYSTEM.CLUSTER.HISTORY.QUEUE 用于存储集群状态信息的历史记录以用于服务目的。

在缺省对象设置中， SYSTEM.CLUSTER.HISTORY.QUEUE 设置为 PUT (ENABLED)。 要禁止历史记录收集，请将设置更改为 PUT (DISABLED)。

SYSTEM.CLUSTER.REPOSITORY.QUEUE
集群中的每个队列管理器都有一个名为 SYSTEM.CLUSTER.REPOSITORY.QUEUE 的本地队列。 此队列用于存储所有完整存储库信息。 此队列通常不为空。

SYSTEM.CLUSTER.TRANSMIT.QUEUE
每个队列管理器都有一个名为 SYSTEM.CLUSTER.TRANSMIT.QUEUE 的本地队列的定义。 SYSTEM.CLUSTER.TRANSMIT.QUEUE 是所有消息到集群中所有队列和队列管理器的缺省传输队列。 您可以通过更改队列管理器属性 DEFCLXQ 将每个集群发送方通道的缺省传输队列更改为 SYSTEM.CLUSTER.TRANSMIT. *ChannelName*。 无法删除 SYSTEM.CLUSTER.TRANSMIT.QUEUE。 它还用于定义授权检查所使用的缺省传输队列是 SYSTEM.CLUSTER.TRANSMIT.QUEUE 还是 SYSTEM.CLUSTER.TRANSMIT. *ChannelName*。

SYSTEM.DEF.CLUSRCVR
每个集群都有一个名为 SYSTEM.DEF.CLUSRCVR 的缺省 CLUSRCVR 通道定义。 SYSTEM.DEF.CLUSRCVR 用于为您在集群中的队列管理器上创建集群接收方通道时未指定的任何属性提供缺省值。

SYSTEM.DEF.CLUSSDR

每个集群都有一个名为 SYSTEM.DEF.CLUSSDR 的缺省 CLUSSDR 通道定义。 SYSTEM.DEF.CLUSSDR 用于为您在集群中的队列管理器上创建集群发送方通道时未指定的任何属性提供缺省值。

相关概念

使用缺省集群对象

发布/预订消息传递

发布/预订消息传递允许您使信息的提供者与该信息的使用者分离开来。 发送应用程序和接收应用程序不需要相互了解即可发送和接收信息。

在点到点 IBM MQ 应用程序可以向另一个应用程序发送消息之前，它需要了解有关该应用程序的一些信息。例如，它需要知道要向其发送信息的队列的名称，并且还可以指定队列管理器名称。

IBM MQ 发布/预订使您的应用程序无需了解有关目标应用程序的任何信息。 发送应用程序必须执行以下操作:

- 放置 包含应用程序需要的信息的 IBM MQ 消息。
- 将消息分配给表示信息主题的主题。
- 让 IBM MQ 处理该信息的分发。

类似地，目标应用程序不必知道有关其接收的信息源的任何信息。

下图显示了最简单的发布/预订系统。 有一个发布程序，一个队列管理器和一个订户。 预订由队列管理器上的订户进行，发布从发布者发送到队列管理器，然后由队列管理器将发布转发给订户。

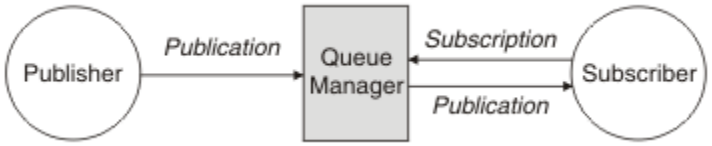


图 *17:* 简单发布/预订配置

典型的发布/预订系统在许多不同的主题上具有多个发布者和多个订户，并且通常具有多个队列管理器。 应用程序可以是发布者和订户。

发布/预订消息传递与点到点之间的另一个显着差异是，发送到点到点队列的消息仅由单个使用应用程序处理。 发布到发布/预订主题的消息 (其中多个订户已注册兴趣) 由每个感兴趣的订户处理。

发布/预订组件

发布/订阅是订阅者以消息形式从发布者接收信息的机制。 发布者与订户之间的交互由队列管理器使用标准 IBM MQ 工具进行控制。

典型的发布/预订系统在许多不同的主题上具有多个发布者和多个订户，并且通常具有多个队列管理器。 应用程序可以是发布者和订户。

信息的提供者称为发布者。 发布者提供关于主题的信息，而不必了解任何关于对该信息有兴趣的应用程序的情况。 发布者以消息形式生成此信息，称为 发布 ，他们要发布这些消息并定义这些消息的主题。

信息的使用者称为订户。 订户创建描述订户感兴趣的主题的 预订 。 因此，订阅确定了转发给订阅者的发布内容。 订户可以创建多个预订，并可以接收来自许多不同发布者的信息。

已发布的信息将在 IBM MQ 消息中发送，并且该信息的主题由其 主题标识。 发布者在发布信息时指定主题，订户指定要接收发布的主题。 仅向订户发送有关其预订的主题的信息。

它是存在的主题，允许信息的提供者和使用者在发布/预订消息传递中解耦，方法是根据点到点消息传递中的要求，消除在每条消息中包含特定目标的需求。

发布者和订户之间的交互都由队列管理器控制。 队列管理器从发布者接收消息，并从订户接收预订 (针对一系列主题)。 队列管理器的作业是将已发布的消息路由到已在消息主题中注册兴趣的订户。

标准 IBM MQ 工具用于分发消息，因此应用程序可以使用可供现有 IBM MQ 应用程序使用的所有功能。 这意味着您可以使用持久消息来获取仅一次有保证的传递，并且您的消息可以是事务性工作单元的一部分，以确保仅当消息由发布者落实时才将其传递给订户。

出版商和出版物

在 IBM MQ 发布/预订中，发布者是一个应用程序，它以称为发布的标准 IBM MQ 消息的形式向队列管理器提供有关指定主题的信息。 发布者可以发布有关多个主题的信息。

发布者使用 MQPUT 动词将消息放入先前打开的主题，此消息是发布。 然后，本地队列管理器会将发布路由到预订该发布主题的任何订户。 已发布的消息可以由多个订户使用。

除了将发布分发给具有相应预订的所有本地订户外，队列管理器还可以直接或通过具有主题订户的队列管理器网络将发布分发给与其连接的任何其他队列管理器。

在 IBM MQ 发布/预订网络中，发布应用程序也可以是订户。

同步点下的出版物

发布者可以在同步点中发出 MQPUT 或 MQPUT1 调用，以在工作单元中包含传递给订户的所有消息。 如果指定了值为 ALL 或 ALLDUR 的 MQPMO_RETAIN 选项或主题交付选项 NPMSGDLV 或 PMSGDLV ，那么队列管理器将在发布程序 MQPUT 或 MQPUT1 调用的作用域内同步点中使用内部 MQPUT 或 MQPUT1 调用。

状态和事件信息

可以将出版物分类为状态出版物 (例如，库存的当前价格) 或事件出版物 (例如，该库存的贸易)。

状态发布

状态发布 包含有关某事物的当前状态的信息，例如，库存价格或足球比赛中的当前分数。 当发生某件事（如股票价格更改或足球比分更改）时，将不再需要先前的状态信息，因为它已被新的信息取代。

订户将希望在启动时接收当前版本的状态信息，并在状态更改时被发送新信息。

如果发布内容包含了状态信息，其通常作为保留发布内容被发布。 新订户通常希望立即获取当前状态信息; 订户不希望等待导致重新发布信息的事件。 除非预订程序使用 MQSO_PUBLIC ICATIONS_ON_REQUEST 或 MQSO_NEW_PUBLIC ICATIONS_ONLY 选项，否则预订程序在预订时将自动接收主题的保留发布。

事件发布

事件发布 包含有关发生的个别事件的信息，例如某个库存的交易或特定目标的评分。 每个事件独立于其他事件。

订户将希望在事件发生时接收有关这些事件的信息。

保留的发布内容

缺省情况下，在将发布发送到所有感兴趣的订户之后，将废弃该发布。 但是，发布者可以指定保留发布的副本，以便可以将其发送给在主题中注册兴趣的未来订户。

在将发布发送给所有感兴趣的订户后删除这些发布适用于事件信息，但并不总是适用于状态信息。 通过保留消息，新订阅者不必等待再次发布信息即可接收到初始状态信息。 例如，预订股票价格的订户将直接收到当前价格，而无需等待股票价格更改 (并因此重新发布)。

队列管理器只能为每个主题保留一个发布，因此当新的保留发布到达队列管理器时，将删除主题的现有保留发布。 但是，删除现有发布可能不会在新保留发布到达时同步发生。 因此，只要有可能，就有不超过一个发布者发送有关任何主题的保留发布。

订户可以使用 MQSO_NEW_publicATIONS_ONLY 预订选项指定他们不希望接收保留发布。 现有订户可以要求向其发送保留发布的重复副本。

有时，您可能不希望保留发布，即使是针对状态信息:

- 如果在对某个主题进行任何发布之前对该主题进行了所有预订，并且您不期望或不允许新预订，那么不需要保留发布，因为这些发布在第一次发布时交付到完整的订户集。
- 如果发布频繁发生 (例如每秒) ，那么新订户 (或从故障中恢复的订户) 几乎在其初始预订之后立即接收到当前状态，因此不需要保留这些发布。
- 如果发布内容较大，那么最终可能需要大量存储空间来存储每个主题的保留发布内容。 在多队列管理器环境中，保留发布由具有匹配预订的网络中的所有队列管理器存储。

在决定是否使用保留发布时，请考虑预订应用程序如何从故障中恢复。 如果发布者不使用保留发布，那么订户应用程序可能需要在本地存储其当前状态。

要确保保留发布，请使用 MQPMO_RETAIN put-message 选项。 如果使用此选项并且无法保留发布，那么将不会发布消息，并且调用将失败并返回 MQRC_PUT_NOT_保留。

如果消息是保留发布，那么此消息由 MQIsRetained 消息属性指示。 消息的持久性与最初发布消息时一样。

相关概念

发布/预订集群中保留的发布的设计注意事项

同步点下的出版物

在 IBM MQ 发布/预订中，同步点可以由发布者使用，也可以由队列管理器在内部使用。

发布程序在使用 MQPMO_SYNCPOINT 选项发出 MQPUT/MQPUT1 调用时使用同步点。 传递到订户的所有消息将计入 work.The MAXUMSGS 队列管理器属性指定此限制。 如果达到限制，那么发布程序将接收到 2024 (07E8) (RC2024) :MQRC_SYNCPOINT_LIMIT_已达到 原因码。

当发布者使用带有 MQPMO_RETAIN 选项的 MQPMO_NO_SYNCPOINT 或带有值 ALL 或 ALLDUR 的主题传递选项 NPMSGDV/PMSGDV 发出 MQPUT/MQPUT1 调用时，队列管理器将使用内部同步点来保证按请求传递消息。 如果在发布程序 MQPUT/MQPUT1 调用范围内达到限制，那么发布程序可以接收 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_已达到 原因码。

订户和预订

在 IBM MQ 发布/预订中，订户是从发布/预订网络中的队列管理器请求有关特定主题的信息的应用程序。 订户可以从多个发布程序接收关于相同或不同主题的消息。

可以使用 MQSC 命令手动创建预订，也可以由应用程序手动创建预订。 这些预订将向本地队列管理器发出，并包含有关订户要接收的发布的信息:

- 订户感兴趣的主题; 如果使用通配符，那么这可以解析为多个主题。
- 要应用于已发布消息的可选的选择字符串。
- 队列 (称为 订户队列) 的句柄，应该将所选发布放在该队列上，以及可选的 CorrelId。

本地队列管理器存储预订信息，当它接收到发布时，扫描该信息以确定是否存在与发布的主题和选择字符串相匹配的预订。 对于每个匹配的预订，队列管理器会将发布定向到订户的订户队列。 可以使用 DIS SUB 和 DIS SBSTATUS 命令来查看队列管理器存储的有关预订的信息。

仅当发生下列其中一个事件时，才会删除预订:

- 订户使用 MQCLOSE 调用取消预订 (如果以非持久方式进行预订)。
- 预订将到期。
- 系统管理员使用 DELETE SUB 命令删除预订。
- 订户应用程序结束 (如果预订以非持久方式进行)。
- 队列管理器将停止或重新启动 (如果预订以非持久方式进行)。

获取消息时，请在 MQGET 调用上使用相应的选项。 如果应用程序仅处理一个预订的消息，那么至少应使用 `get-by-correlid`，如 C 样本程序 `amqssbxa.c` 和 非受管 MQ 订户中所示。 要使用的 **CorrelId** 从 MQSD 中的 MQSUB 返回。 **SubCorrelId** 字段。

相关概念

克隆和共享的预订

相关参考
如何定义 sharedSubscription 属性的示例

受管队列和发布/预订

创建预订时，可以选择使用受管排队。 如果使用受管排队，那么在创建预订时将自动创建预订队列。 将根据预订的耐久性自动对受管队列进行分层。 使用受管队列意味着您不必担心创建队列以接收发布，如果非持久预订连接已关闭，那么将自动从订户队列中除去任何未使用的发布。

如果应用程序不需要将特定队列用作其订户队列 (它接收的发布的目标) ，那么可以使用 MQSO_MANAGED 预订选项来使用 受管预订 。 如果您创建受管预订，那么队列管理器会将对象句柄返回给订户队列的订户，队列管理器将在该订户队列中创建将接收发布的对象句柄。 这是因为 受管预订 是 IBM MQ 处理预订的受管预订。 将返回队列的对象句柄，允许您浏览，获取或查询队列 (除非已显式授予您对临时动态队列的访问权，否则无法放入或设置受管队列的属性)。

预订的耐久性确定在预订应用程序与队列管理器的连接中断时受管队列是否保留。

与非持久预订配合使用时，受管预订特别有用，因为当应用程序的连接结束时，未使用的消息将无限期地保留在订户队列上占用队列管理器中的空间。 如果您正在使用受管预订，那么该受管队列将是临时动态队列，因此，当由于以下任何原因导致连接中断时，将删除该受管队列以及任何未使用的消息:

- 使用带有 MQCO_REMOVE_SUB 的 MQCLOSE ，并关闭受管 Hobj。
- 与使用非持久预订 (MQSO_NON_耐久性) 的应用程序的连接丢失。
- 由于预订已到期并且受管 Hobj 已关闭，因此将除去该预订。

受管预订也可以与持久预订配合使用，但您可能希望在订户队列中保留未使用的消息，以便在重新打开连接时可以检索这些消息。 因此，持久预订的受管队列采用永久动态队列形式，并将在预订应用程序与队列管理器的连接中断时保留。

如果要使用永久动态受管队列，那么可以在预订上设置到期时间，以便尽管该队列在连接中断后仍将存在，但它不会无限期地继续存在。

如果删除受管队列，那么将接收到错误消息。

创建的受管队列在结束时使用数字 (时间戳记) 进行命名，因此每个队列都是唯一的。

预订耐久性

可以将预订配置为持久或非持久。 预订耐久性确定在预订应用程序与队列管理器断开连接时对预订执行的操作。

持久预订

持久预订在预订应用程序与队列管理器之间的连接关闭后继续存在。 如果预订是持久的，那么当预订应用程序断开连接时，预订将保留在原位置，并且可以由预订应用程序在使用创建预订时返回的 **SubName** 重新连接请求预订时使用。

持久预订时，预订名称 (**SubName**) 是必需的。 预订名称在队列管理器中必须唯一，以便可用于标识预订。 如果您有意关闭与预订的连接 (使用 MQCO_KEEP_SUB 选项) 或者已与队列管理器断开连接，那么在指定要恢复的预订时，必须使用此标识方法。 您可以使用带有 MQSO_RESUME 选项的 MQSUB 调用来恢复现有预订。 如果将 DISPLAY SBSTATUS 命令与 SUBTYPE ALL 或 ADMIN 配合使用，那么还会显示预订名称。

当应用程序不再需要持久预订时，可以使用带有 MQCO_REMOVE_SUB 选项的 MQCLOSE 函数调用将其除去，也可以使用 MQSC 命令 DELETE SUB 手动将其删除。

您可以使用 **DURSUB** 主题属性来指定是否可以对主题进行持久预订。

使用 MQSO_RESUME 选项从 MQSUB 调用返回时，预订到期将设置为预订的原始到期时间，而不是剩余到期时间。

队列管理器继续发送发布以满足持久预订，即使该订户应用程序未连接也是如此。 这将导致在订户队列上构建消息。 避免此问题的最简单方法是在适当情况下使用非持久预订。 但是，在需要使用持久预订的情况下，如果订户使用 保留发布 选项进行预订，那么可以避免构建消息。 然后，订户可以通过使用 MQSUBRQ 调用来控制何时接收发布。

非持久预订

仅当预订应用程序与队列管理器的连接保持打开状态时，非持久预订才存在。 当预订应用程序有意地断开或由于连接中断而断开与队列管理器的连接时，将除去此预订。 关闭连接时，将从队列管理器中除去有关预订的信息，如果使用 DISPLAY SBSTATUS 命令显示预订，那么将不再显示这些信息。 不会将更多消息放入订户队列。

对于非持久预订的订户队列上的任何未使用的发布，将按如下所示进行确定。

- 如果预订应用程序正在使用 受管目标，那么将自动除去尚未使用的任何发布。
- 如果预订应用程序在预订时向其自己的订户队列提供句柄，那么不会自动除去未使用的消息。 如果需要，应用程序负责清除队列。 如果队列由多个订户或其他点到点应用程序共享，那么可能不适合完全清除该队列。

虽然非持久预订不需要预订名称，但队列管理器会使用预订名称 (如果提供)。 预订名称在队列管理器中必须唯一，以便可用于标识预订。

相关概念

克隆和共享的预订

相关任务

使用 JMS 2.0 共享预订

相关参考

如何定义 sharedSubscription 属性的示例

选择字符串

选择字符串 是应用于发布以确定其是否与预订匹配的表达式。 选择字符串可以包含通配符。

预订时，除了指定主题外，还可以指定选择字符串以根据其消息属性选择发布。

在修改消息以传递给每个订户之前，将根据发布程序放置的消息对选择字符串进行求值。 在选择字符串中使用可能在发布操作过程中修改的字段时，请小心。 例如， MQMD 字段 `UserIdentifier`， `MsgId` 和 `CorrelId`。

选择字符串不应引用队列管理器在发布操作中添加的任何消息属性字段 (请参阅 发布/预订消息属性)，但包含发布主题字符串的消息属性 `MQTopicString` 除外。

相关概念

选择字符串规则和限制

主题

主题是发布/预订消息中发布的信息的论题。

将点到点系统中的消息发送到特定目标地址。 基于主题的发布/预订系统中的消息将根据描述消息内容的主题发送给订户。 在基于内容的系统中，根据消息本身的内容向订户发送消息。

IBM MQ 发布/预订系统是基于主题的发布/预订系统。 发布者创建一条消息，并使用最适合发布主题的主题字符串来发布该消息。 要接收发布内容，订户创建带有模式匹配主题字符串的预订，该字符串用于选择发布内容主题。 队列管理器将发布内容传递给其预订与发布内容主题匹配并有权接收发布内容的订户。 文章 第 56 页的『主题字符串』描述了用于标识出版物主题的主题字符串的语法。 订户还创建主题字符串以选择要接收的主题。 订户创建的主题字符串可以包含两个备用通配符方案中的任何一个，以便与发布内容中的主题字符串进行模式匹配。 第 57 页的『通配方案』中描述了模式匹配。

在基于主题的发布/预订中，发布者或管理员负责将主题分类为主题。 通常，主题以分层方式组织到主题树中，使用 '/' 字符在主题字符串中创建子主题。 请参阅 第 62 页的『主题树』 以获取主题树的示例。 主题是主题树中的节点。 主题可以是没有进一步子主题的叶节点，也可以是具有子主题的中间节点。

在将主题组织到分层主题树中的同时，可以将主题与管理主题对象相关联。 通过将主题与管理主题对象相关联，可以将属性分配给该主题，例如，该主题是否分布在集群中。 通过使用管理主题对象的 `TOPICSTR` 属性对主题进行命名来进行关联。 如果未显式将管理主题对象与主题关联，那么该主题将继承其在 具有 与管理主题对象关联的主题树中最接近的祖代的属性。 如果您根本未定义任何父主题，那么它将从 `SYSTEM.BASE.TOPIC`。 第 62 页的『管理主题对象』中描述了管理主题对象。

注: 即使您从 SYSTEM.BASE.TOPIC，为直接继承自 SYSTEM.BASE.TOPIC。 例如，在美国各州的主题空间中， USA/Alabama USA/Alaska 等， USA 是根主题。 根主题的主要用途是创建离散的非重叠主题空间，以避免发布与错误的预订匹配。 这也意味着您可以更改根主题的属性以影响整个主题空间。 例如，可以设置 **CLUSTER** 属性的名称。

以发布者或订户身份引用主题时，可以选择提供主题字符串或引用主题对象。 或者可以同时执行这两种操作，在这种情况下，您提供的主题字符串将定义主题对象的子主题。 队列管理器通过将主题字符串附加到主题对象中指定的主题字符串前缀，在两个主题字符串之间插入额外的 '/' (例如， 主题字符串/对象字符串)来标识主题。 第 60 页的『组合主题字符串』 进一步对此进行了描述。 生成的主题字符串用于标识主题并将其与管理主题对象相关联。 管理主题对象不一定是与主主题对应的主题对象相同的主题对象。

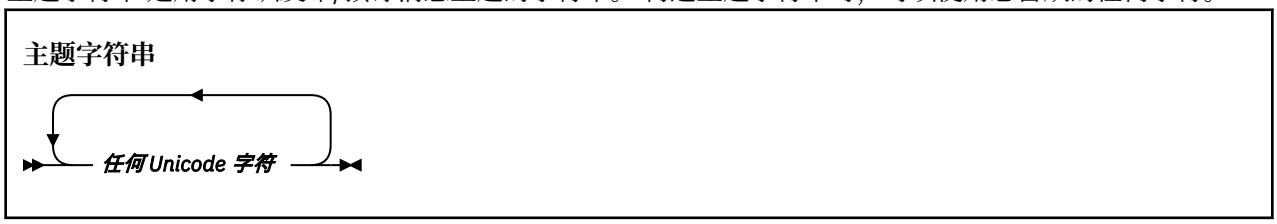
在基于内容的发布/预订中，您可以通过提供用于搜索每条消息内容的选择字符串来定义要接收的消息。 IBM MQ 提供了基于内容的发布/预订的中间形式，使用消息选择器来扫描消息属性而不是消息的完整内容，请参阅 选择器。 消息选择器的原型用途是预订主题，然后使用数字属性限定选择。 选择器使您能够指定您只对特定范围内的值感兴趣; 不能使用字符或基于主题的通配符来执行任何操作。 如果确实需要根据消息的完整内容进行过滤，那么需要使用 IBM Integration Bus。

主题字符串

您使用主题字符串将标签信息发布为主题。 然后，使用基于字符或主题的通配主题字符串来预订一组主题。

主题

主题字符串 是用于标识发布/预订消息主题的字符串。 构造主题字符串时，可以使用您喜欢的任何字符。



三个字符在 IBM WebSphere MQ 7 发布/预订中具有特殊含义。 在主题字符串中的任何位置都允许这些值，但请谨慎使用这些值。 在 第 57 页的『基于主题的通配方案』 中说明了特殊字符的用法。

正斜杠 (/)

主题级别分隔符。 使用 '/' 字符将主题构造为主题树。

如果可以，请避免使用空的主题级别 '//'。 这些节点对应于主题层次结构中没有主题字符串的节点。 主题字符串中的前导或尾部 '/' 对应于前导或尾部空节点，也应避免。

散列符号 (#)

与 '/' 结合使用以在预订中构造多级通配符。 请注意在用于命名已发布主题的主题字符串中使用与 '/' 相邻的 '#' 。 第 56 页的『主题字符串示例』 显示了对 '#'的合理使用。

字符串 '.../#/...'， '#/...' 和 '.../#' 在预订主题字符串中具有特殊含义。 这些字符串与主题层次结构中的一个或多个级别的所有主题匹配。 因此，如果您创建了具有其中一个序列的主题，那么无法预订该主题，同时也无法预订主题层次结构中多个级别的所有主题。

加号 (+)

与 '/' 结合使用以在预订中构造单层通配符。 请注意在用于命名已发布主题的主题字符串中使用与 '/' 相邻的 '+' 。

字符串 '.../+/...'， '+/...' 和 '.../+' 在预订主题字符串中具有特殊含义。 这些字符串与主题层次结构中的一个级别的所有主题匹配。 因此，如果您创建了具有其中一个序列的主题，那么无法预订该主题，也无法在主题层次结构中的一个级别预订所有主题。

主题字符串示例

```
IBM/Business Area#/Results
IBM/Diversity/%African American
```

相关参考
TOPIC

通配方案
有两种用于预订多个主题的通配方案。 对方案的选择是预订选项。

MQSO_WILDCARD_TOPIC
选择要使用基于主题的通配符方案预订的主题。

如果未显式选择通配符模式，那么这是缺省值。

MQSO_WILDCARD_CHAR
选择要使用基于字符的通配符方案预订的主题。

通过在 DEFINE SUB 命令上指定 **wschema** 参数来设置任一方案。 有关更多信息，请参阅 DEFINE SUB。

注: 在 IBM WebSphere MQ 7.0 之前创建的预订始终使用基于字符的通配符方案。

示例

```
IBM/+/Results
#/Results
IBM/Software/Results
IBM/*ware/Results
```

基于主题的通配方案
基于主题的通配符允许订户每次预订多个主题。

基于主题的通配符是 IBM MQ 发布/预订中主题系统的强大功能。 多点传送通配符和单一级别通配符可用于预订，但不能由消息的发布者在主题中使用。

基于主题的通配场景允许您选择按主题级别进行分组的发布内容。 对于主题层次结构中的每个级别，您可以选择该主题级别的预订中的字符串是否必须与发布内容中的字符串完全匹配。 例如，预订 `IBM/+/Results` 选择所有主题，

```
IBM/Software/Results
IBM/Services/Results
IBM/Hardware/Results
```

有两种类型的通配符。

多级别通配符

- 多级别通配符用于订阅中。 在发布中使用时，会将其视为文字。
- 多级通配符 '#' 用于匹配主题中的任意数量的级别。 例如，使用示例主题树，如果预订 'USA/Alaska/#'，那么将接收有关主题 'USA/Alaska' 和 'USA/Alaska/Juneau'的消息。
- 多级别通配符可以不表示任何级别，也可以表示多个级别。 因此， 'USA/#' 还可以与奇异 'USA'匹配，其中 '#' 表示零级别。 因为没有要分隔的级别，所以主题级别分隔符在此上下文中没有含义。
- 多级别通配符仅当单独指定或紧随主题级别分隔符指定时才有效。 因此， '#' 和 'USA/#' 是将 '#' 字符视为通配符的有效主题。 但是，虽然 'USA#' 也是有效的主题字符串，但 '#' 字符不会被视为通配符，并且没有任何特殊含义。 请参阅 第 59 页的『当基于主题的通配符无效时』，以了解更多信息。

单一级别通配符

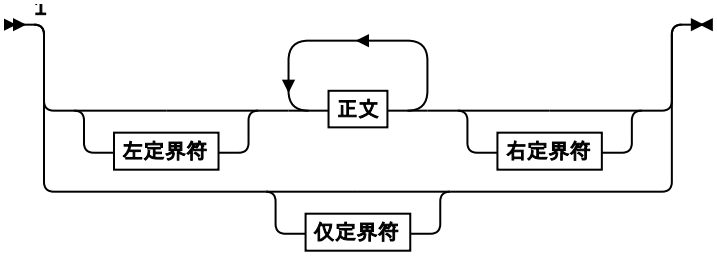
- 单一级别通配符用于订阅中。 在发布中使用时，会将其视为文字。
- 单级别通配符 '+' 与一个且仅与一个主题级别匹配。 例如， 'USA/+' 与 'USA/Alabama'匹配，但与 'USA/Alabama/Auburn'不匹配。 因为单级别通配符仅与单个级别匹配，所以 'USA/+' 与 'USA'不匹配。
- 可以在主题树中的任何级别使用单层通配符，并将其与多层通配符结合使用。 除了单独指定单一级别通配符时以外，必须紧随主题级别分隔符指定单一级别通配符。 因此， '+' 和 'USA/+' 是将 '+' 字

符视为通配符的有效主题。 但是，虽然 'USA+' 也是有效的主题字符串，但 '+' 字符不会被视为通配符，并且没有任何特殊含义。 请参阅 第 59 页的『当基于主题的通配符无效时』，以了解更多信息。

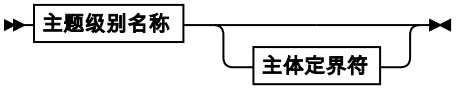
基于主题的通配符方案的语法没有转义字符。 是否将 '#' 和 '+' 视为通配符取决于它们的上下文。 有关更多信息，请参阅 第 59 页的『当基于主题的通配符无效时』。

注: 以特殊方式处理主题字符串的开头和结尾。 使用 '$' 来表示字符串的结尾，那么 '$#/...' 是多级通配符， '$/#/..'。 是根处的空节点，后跟多级通配符。

基于主题的通配符字符串

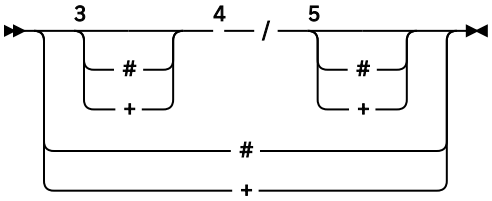


正文



主题级别名称

仅定界符



注:

[1] 空或零长度主题字符串无效

[2] 建议您不要在级别名称字符串中使用任何 *, ?, %，以实现基于字符的通配符方案与基于主题的通配符方案之间的兼容性。

[3] 这些个案等同于 左定界符 模式。

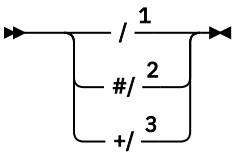
[4] 没有通配符的 / 与单个空主题匹配。

[5] 这些个案等同于 右定界符 模式。

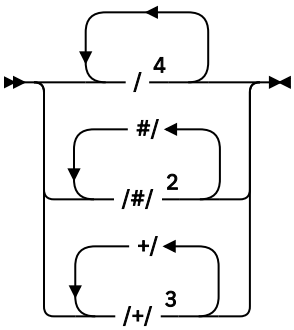
[6] 匹配每个主题。

[7] 匹配每个只有一个级别的主题。

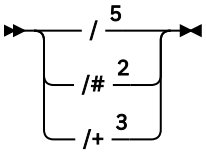
左定界符



主体定界符



右定界符



注:

[1] 主题字符串以空主题开头
[2] 与零个或多个级别匹配。 多个多级匹配字符串与一个多级匹配字符串具有相同的效果。
[3] 正好匹配一个级别。
[4] // 是空主题-没有主题字符串的主题对象。
[5] 主题字符串以空主题结尾

当基于主题的通配符无效时

当通配符 '+' 和 '#' 与主题级别中的其他字符 (包括它们本身) 混合时，它们没有特殊含义。

这意味着可以发布主题级别中包含 '+' 或 '#' 以及其他字符的主题。

例如以下两个主题:

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

在第一个示例中，字符 '+' 和 '#' 被视为通配符，因此在要发布到的主题字符串中无效，但在预订中有效。

在第二个示例中，不会将字符 '+' 和 '#' 视为通配符，因此可以发布和预订主题字符串。

示例

```
IBM/+/Results
#/Results
IBM/Software/Results
```

基于字符的通配符方案
基于字符的通配符方案允许您根据传统字符匹配来选择主题。

您可以使用字符串 '*'在主题层次结构中选择多个级别的所有主题。 在基于字符的通配符方案中使用 '*' 等同于使用基于主题的通配符字符串 '#'

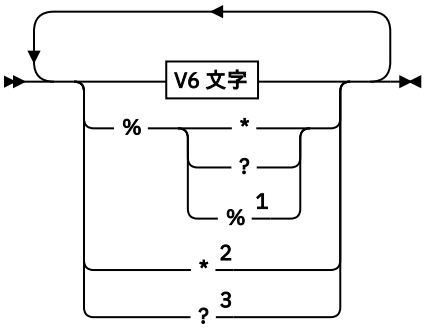
' *x*/*/*y* ' 等同于基于主题的方案中的 ' *x*/#/*y* '，并在级别 ' *x* 和 *y* '之间选择主题层次结构中的所有主题，其中 ' *x* ' 和 ' *y* ' 是不在通配符返回的级别集中的主题名称。

基于主题的方案中的 '/+/' 在基于字符的方案中没有完全相同的内容。 'IBM/*/Results' 还将选择 'IBM/Patents/Software/Results'。 仅当层次结构的每个级别的主题名称集都是唯一的时，才能始终使用生成相同匹配项的两个方案来构造查询。

以一般方式使用，基于字符的方案中的 '*' 和 '?' 在基于主题的方案中没有等效项。 基于主题的方案不会使用通配符执行部分匹配。 基于字符的通配符预订 'IBM/*ware/Results' 没有基于主题的等效项。

注: 使用字符通配符预订的匹配比使用基于主题的预订的匹配慢。

基于字符的通配符字符串



V6 文字

除* *和以外的任何 Unicode 字符?* ***和*** *%*

注:

[1] 表示对以下字符进行转义，以便将其视为文字。 '%' 必须后跟 '*'， '?' 或 '%'。 请参阅第 56 页的『主题字符串示例』。
[2] 表示与预订中的零个或多个字符匹配。
[3] 表示与预订中的一个字符完全匹配。

示例

```
IBM/*/Results
IBM/*ware/Results
```

组合主题字符串

创建预订或打开主题以使您可以向其发布消息时，可以通过组合两个单独的子主题字符串 (即 “子主题”) 来构成主题字符串。 一个子主题由应用程序或管理命令作为主题字符串提供，另一个子主题是与主题对象相关联的主题字符串。 您可以单独使用子主题作为主题字符串，也可以将其组合以形成新的主题名称。

例如，使用 MQSC 命令 **DEFINE SUB** 定义预订时，该命令可以将 **TOPICSTR** (主题字符串) 和/或 **TOPICOBJ** (主题对象) 作为属性。 如果仅提供了 **TOPICOBJ**，那么与该主题对象关联的主题字符串将用作主题字符串。 如果仅提供了 **TOPICSTR**，那么将用作主题字符串。 如果同时提供了这两个字符串，那么将它们并置以形成格式为 **TOPICOBJ** / **TOPICSTR** 的单个主题字符串，其中 **TOPICOBJ** 配置的主题字符串始终是第一个，并且该字符串的两个部分始终以 “/” 字符分隔。

同样，在 MQI 程序中，完整主题名称由 MQOPEN 创建。 它由发布/预订 MQI 调用中使用的两个字段组成，按列出的顺序:

1. 主题对象的 **TOPICSTR** 属性，在 **ObjectName** 字段中指定。
2. 定义应用程序提供的子主题的 **ObjectString** 参数。

生成的主题字符串将在 **ResObjectString** 参数中返回。

如果每个字段的第一个字符不是空白或空字符，并且字段长度大于零，那么这些字段被视为存在。 如果仅存在其中一个字段，那么将其用作主题名称。 如果两个字段都没有值，那么调用将失败，原因码为 MQRC_UNKNOWN_OBJECT_NAME; 如果完整主题名称无效，那么调用将失败 MQRC_TOPIC_STRING_ERROR 。

如果两个字段都存在，那么将在生成的组合主题名称的两个元素之间插入 “/” 字符。

下表显示了主题字符串并置的示例:

表 2: 主题字符串并置示例			
主题对象的 TOPICSTR	**由应用程序或 DEFINE SUB 命令提供的主题字符串**	**完整主题名称**	**注释**
足球/苏格兰人	' '	足球/苏格兰人	单独使用主题对象的 TOPICSTR。
' '	足球/苏格兰人	足球/苏格兰人	单独使用 ObjectString/ TOPICSTR。
美式足球	评分	足球/苏格兰人	在并置点添加 “/” 字符。
美式足球	/得分	足球//苏格兰人	在两个字符串之间生成 "空节点"。 这与 “Football/Scores”不同。
/足球	评分	/足球/苏格兰人	主题以 "空节点" 开头。 这与 “Football/Scores” 不同。

“/” 字符被视为特殊字符，为第 62 页的『主题树』中的完整主题名称提供结构。 不得将 “/” 字符用于任何其他原因，因为主题树的结构受到影响。 主题 “/Football” 与主题 “Football”不同。

注: 如果在创建预订时使用主题对象，那么主题对象主题字符串的值在定义时在预订中固定。 对主题对象的任何后续更改都不会影响预订所定义到的主题字符串。

主题字符串中的通配符

以下通配符是特殊字符:

- 加号 (+)
- 编号符号 (#)
- 星号 (*)
- 问号 (?)

仅当预订使用通配符时，通配符才具有特殊含义。 在其他位置使用时，这些字符不会被视为无效，但是您必须确保了解如何使用这些字符，并且在发布或定义主题对象时，您可能不希望在主题字符串中使用这些字符。

如果在主题级别中使用 # 或 + 与其他字符 (包括其本身) 混合的主题字符串进行发布，那么可以使用通配符方案来预订该主题字符串。

如果发布主题字符串时使用 # 或 + 作为两个 / 字符之间的唯一字符，那么应用程序无法使用通配符方案 MQSO_WILDCARD_TOPIC 显式预订主题字符串。 这种情况会导致应用程序获得比预期更多的发布。

不应在定义的主题对象的主题字符串中使用通配符。 如果执行此操作，那么当发布程序使用对象时，该字符将被视为字面值字符，当预订使用对象时，该字符将被视为通配符。 这会导致混乱。

示例代码片段

从示例程序 Example 2: Publisher to a variable topic 中抽取的此代码片段将主题对象与变量主题字符串组合在一起:

```
MQOD   td = {MQOD_DEFAULT}; /* Object Descriptor      */
td.ObjectType = MQOT_TOPIC;  /* Object is a topic      */
td.Version = MQOD_VERSION_4;  /* Descriptor needs to be V4  */
strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
```

```
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

主题树

您定义的每个主题都是主题树中的一个元素或节点。 主题树可以为空以开始，也可以包含先前使用 MQSC 或 PCF 命令定义的主题。 通过使用“创建主题”命令或通过指定主题，可以在发布或预订中首次定义新的主题。

虽然可以使用任何字符串来定义主题的主题字符串，但建议选择适合分层树结构的主题字符串。 经过深思熟虑的主题刺和主题树设计可以帮助您执行以下操作:

- 预订多个主题。
- 建立安全策略。

虽然您可以将主题树构造为平面的线性结构，但最好在具有一个或多个根主题的分层结构中构建主题树。 有关安全规划和主题的更多信息，请参阅 发布/预订安全性。

第 62 页的图 18 显示了具有一个根主题的主题树的示例。

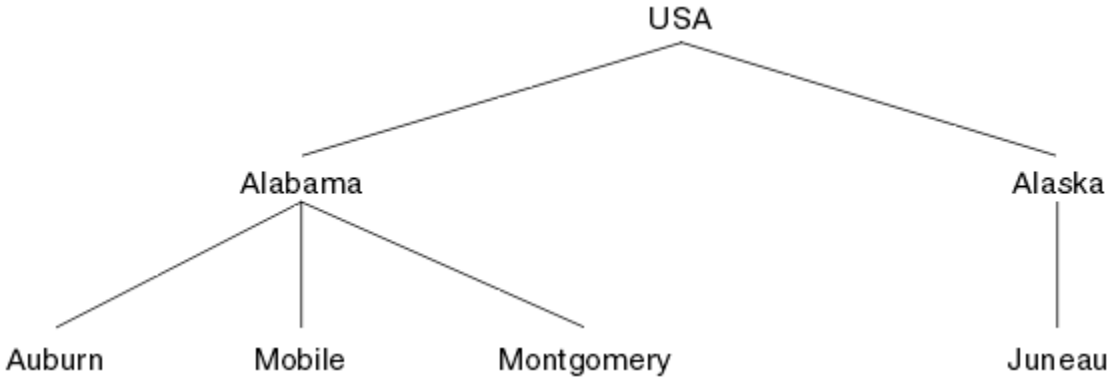


图 *18:* 主题树的示例

图中的每个字符串表示主题树中的一个节点。 通过聚集主题树中一个或多个级别的节点来创建完整的主题字符串。 级别由 "/" 字符分隔。 完全指定的主题字符串的格式为: "root/level2/level3"。

第 62 页的图 18 中显示的主题树中的有效主题包括:

"美国"
"美国/阿拉巴马州"
"美国/阿拉斯加"
"USA/Alabama/Auburn"
"USA/Alabama/Mobile"
"USA/Alabama/Montgomery"
"USA/Alaska/Juneau"

在设计主题字符串和主题树时，请记住队列管理器不会解释或尝试派生主题字符串本身的含义。 它只是使用主题字符串将所选消息发送到该主题的订户。

下列原则应用到主题树的构造和内容:

- 对主题树中级别的数目没有限制。
- 对主题树中级别的名称长度没有限制。
- 对“根”节点的数量无限制，即对主题树的数量无限制。

相关任务

减少主题树中不需要的主题数

管理主题对象

通过使用管理主题对象，可以将特定非缺省属性分配给主题。

第 63 页的图 19 显示了如何将划分为不同体育项目的不同主题的 Sport 高级别主题可视化为主题树:

图 *19:* 主题树的可视化

第 63 页的图 20 显示了如何进一步划分主题树，以分隔有关每项运动的不同类型的信息:

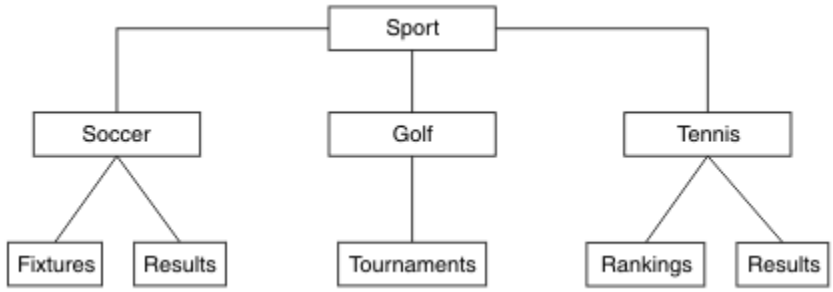


图 *20:* 扩展主题树

要创建说明的主题树，不需要定义任何管理主题对象。 此树中的每个节点都由在发布或预订操作中创建的主题字符串定义。 树中的每个主题都从其父代继承其属性。 属性继承自父主题对象，因为缺省情况下，所有属性都设置为 ASPARENT。 在此示例中，每个主题都具有与 Sport 主题相同的属性。 Sport 主题没有管理主题对象，并且从 SYSTEM.BASE.TOPIC。

请注意，在主题树的根节点 (即 SYSTEM.BASE.TOPIC，因为权限是继承的，但不能限制。 因此，通过授予此级别的权限，您将授予整个树的权限。 您应该在层次结构中的较低主题级别授予权限。

管理主题对象可用于定义主题树中特定节点的特定属性。 在以下示例中，定义了管理主题对象以将足球主题的持久预订属性 DURSUB 设置为值 NO:

```
DEFINE TOPIC(FOOTBALL.EUROPEAN)
TOPICSTR('Sport/Soccer')
DURSUB(NO)
DESCR('Administrative topic object to disallow durable subscriptions')
```

现在可以将主题树可视化为:

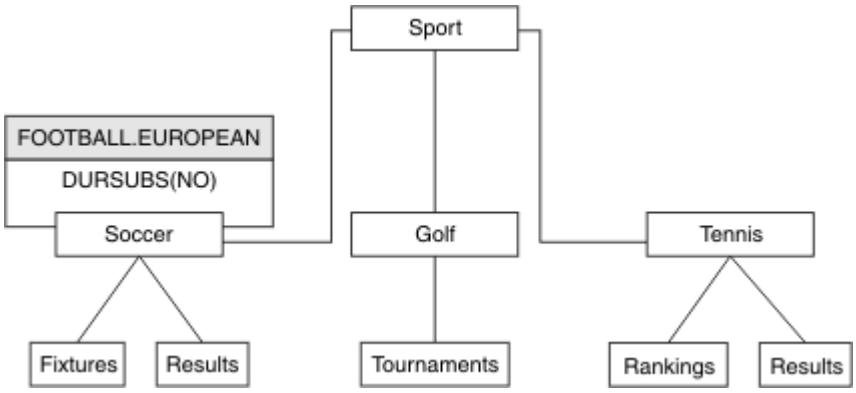


图 *21:* 与 *Sport/Soccer* 主题关联的管理主题对象的可视化

预订树中 Soccer 下的主题的任何应用程序仍可以使用它们在添加管理主题对象之前使用的主题字符串。 但是，现在可以使用对象名 FOOTBALL.EUROPEAN(而不是字符串 /Sport/Soccer) 来编写应用程序以进行预订。 例如，要预订 /Sport/Soccer/Results，应用程序可以将 MQSD.ObjectName 指定为 FOOTBALL.EUROPEAN，将 MQSD.ObjectString 指定为 Results。

通过此功能，您可以向应用程序开发者隐藏主题树的部分内容。 在主题树中的特定节点上定义管理主题对象，然后应用程序开发者可以将自己的主题定义为该节点的子代。 开发者必须了解父主题，但不能了解父树中的任何其他节点。

继承属性

如果主题树具有许多管理主题对象，那么缺省情况下，每个管理主题对象都将从其最接近的父管理主题继承其属性。 先前的示例已在 第 64 页的图 22 中扩展:

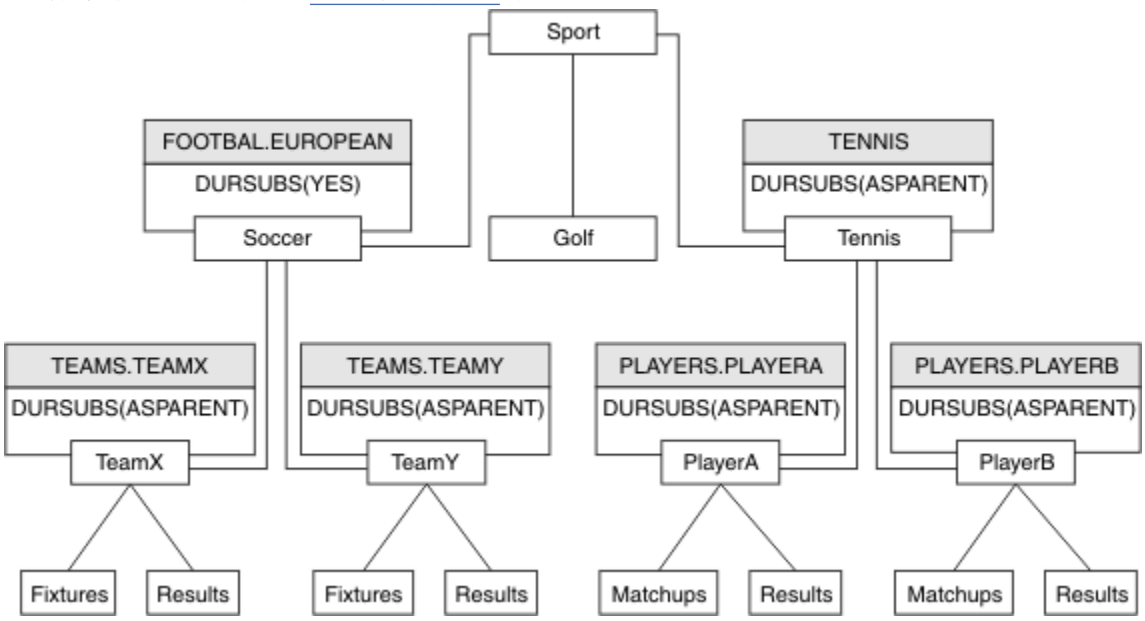


图 *22:* 具有多个管理主题对象的主题树

例如，使用继承为 /Sport/Soccer 的所有子主题提供预订为非持久预订的属性。 将 FOOTBALL.EUROPEAN 的 DURSUB 属性更改为 NO。

可使用以下命令设置此属性:

```
ALTER TOPIC(FOOTBALL.EUROPEAN) DURSUB(NO)
```

Sport/Soccer 的子主题的所有管理主题对象都将属性 DURSUB 设置为缺省值 ASPARENT。 将 FOOTBALL.EUROPEAN 的 DURSUB 属性值更改为 NO 后， Sport/Soccer 的子主题将继承 DURSUB 属性值 NO。 Sport/Tennis 的所有子主题从 SYSTEM.BASE.TOPIC 对象继承 DURSUB 的值。 SYSTEM.BASE.TOPIC 的值为 YES。

尝试对主题 Sport/Soccer/TeamX/Results 进行持久预订现在将失败; 但是，尝试对 Sport/Tennis/PlayerB/Results 进行持久预订将成功。

使用 通配符 属性控制通配符使用

使用 MQSC **Topic** 通配符 属性或等效 PCF 主题 WildcardOperation 属性来控制向使用通配符主题字符串名称的订户应用程序传递发布内容。 通配符 属性可以具有以下两个可能的值之一:

WILDCARD
与此主题有关的通配符预订的行为。

PASSTHRU
对没有此主题对象中主题字符串具体的通配主题进行的预订将接收对此主题以及比此主题更具体的主题字符串进行的发布。

BLOCK
对没有此主题对象中主题字符串具体的通配主题进行的预订不会接收对此主题或比此主题更具体的主题字符串进行的发布。

在定义预订时将使用此属性的值。 如果改变此属性，那么现有预订涵盖的主题集不会因为此修改而受到影响。 如果在创建或删除主题对象时拓扑发生更改，此场景也适用；将使用修改后的拓扑来创建与修改 WILDCARD 属性后创建的预订匹配的主题集。 如果要针对现有预订强制重新评估匹配的主题集，那么必须重新启动队列管理器。

在示例 第 67 页的『示例: 创建 Sport 发布/预订集群』中，您可以遵循步骤来创建 第 65 页的图 23 中显示的主题树结构。

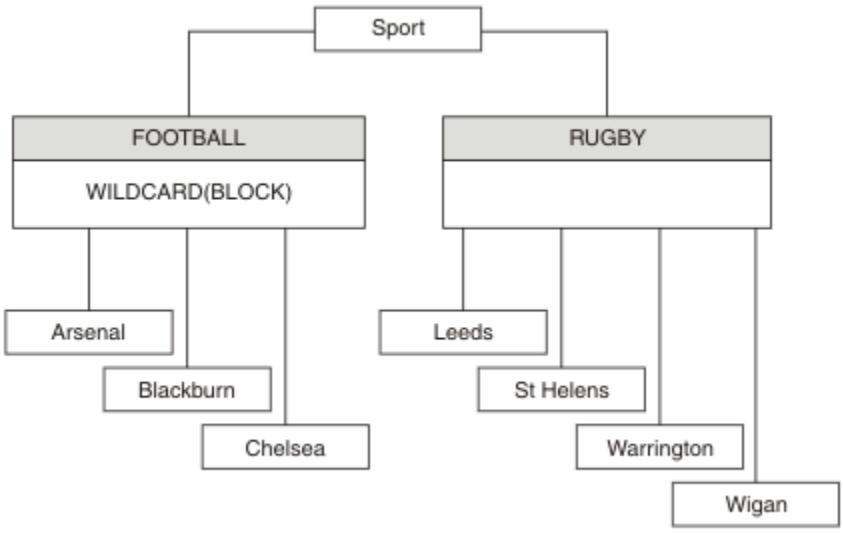


图 *23:* 使用 通配符 属性 *BLOCK* 的主题树

使用通配符主题字符串 *#* 的订户将接收到 Sport 主题和 Sport/Rugby 子树的所有发布。 订户不会接收到 Sport/Football 子树的发布，因为 Sport/Football 主题的 **通配符** 属性值为 BLOCK。

PASSTHRU 是缺省设置。 您可以将 **通配符** 属性值 PASSTHRU 设置为 Sport 树中的节点。 如果节点没有 **通配符** 属性值 BLOCK，那么设置 PASSTHRU 不会改变 Sports 树中节点的订户观察到的行为。

在此示例中，创建预订以查看通配符设置如何影响交付的发布; 请参阅 第 69 页的图 27。 在 第 70 页的图 30 中运行发布命令以创建一些发布。

```
pub QMA
```

图 *24:* 发布到 *QMA*

第 65 页的表 3 显示了结果。 请注意，如何设置 **通配符** 属性值 BLOCK，以防止具有通配符的预订将发布内容接收到通配符作用域内的主题。

表 *3: QMA* 上收到的出版物

预订	**主题字符串**	**收到的出版物**	**注意**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	WILDCARD(BLOCK) 在 Sports/Football 上阻止了对 Football 子树的所有发布
SARSENAL	Sports/#/Arsenal	-	Sports/Football 上的 WILDCARD(BLOCK) 会阻止 Arsenal 上的通配符预订
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby 上的缺省 WILDCARD 不会阻止 Leeds 上的通配符预订。

注:

假设预订具有通配符，该通配符与具有 **通配符** 属性值 BLOCK 的主题对象相匹配。 如果预订还具有匹配通配符右侧的主题字符串，那么预订将从不接收发布内容。 未被阻止的发布集合是作为被阻止通配符的父代的主题的发布。 通配符将阻止发布到作为具有 BLOCK 属性值的主题的子代的主题。 因此，包含通配符右侧主题的预订主题字符串永远不会收到任何要匹配的发布内容。

将 WILDCARD 属性值设置为 BLOCK 并不意味着您无法使用包含通配符的主题字符串进行预订。 这样的订阅是正常的。 预订具有一个显式主题，该主题与具有 **通配符** 属性值 BLOCK 的主题对象匹配。 它将通配符用于作为具有 **通配符** 属性值 BLOCK 的主题的父代或子代的主题。 在 第 65 页的图 23 中的示例中，预订 (例如 Sports/Football/#) 可以接收发布。

通配符和集群主题

集群主题定义将传播到集群中的每个队列管理器。 在集群中的一个队列管理器上预订集群主题会导致队列管理器创建代理预订。 将在集群中的每个其他队列管理器上创建代理预订。 使用包含通配符的主题字符串 (与集群主题结合使用) 的预订可能难以预测行为。 此行为在以下示例中进行了说明。

在为示例 第 67 页的『示例: 创建 Sport 发布/预订集群』 设置的集群中， QMB 具有与 QMA 相同的预订集，但 QMB 在发布者发布到 QMA 之后未收到任何发布，请参阅 第 65 页的图 24。 虽然 Sports/Football 和 Sports/Rugby 主题是集群主题，但 fullsubs.tst 中定义的预订不会引用集群主题。 不会将代理预订从 QMB 传播到 QMA。 如果没有代理预订，那么不会将 QMA 的任何发布转发到 QMB。

某些预订 (例如 Sports/#/Leeds) 可能似乎引用了集群主题，在本例中为 Sports/Rugby。 Sports/#/Leeds 预订实际解析为主题对象 SYSTEM.BASE.TOPIC。

用于解析预订 (例如， Sports/#/Leeds) 所引用的主题对象的规则如下所示。 将主题字符串截断为第一个通配符。 通过主题字符串向左扫描以查找具有关联管理主题对象的第一个主题。 主题对象可以指定集群名称，也可以定义本地主题对象。 在示例 Sports/#/Leeds 中，截断后的主题字符串是 Sports，它没有主题对象，因此 Sports/#/Leeds 继承自 SYSTEM.BASE.TOPIC(这是本地主题对象)。

要查看预订集群主题如何更改通配符传播的工作方式，请运行批处理脚本 upsubs.bat。 该脚本将清除预订队列，并在 fullsubs.tst 中添加集群主题预订。 再次运行 puba.bat 以创建一批出版物; 请参阅 第 65 页的图 24。

第 66 页的表 4 显示了将两个新预订添加到发布发布的同一队列管理器的结果。 结果与预期相同，新预订各接收一个发布，其他预订接收的发布数量保持不变。 意外结果发生在其他集群队列管理器上; 请参阅 第 67 页的表 5。

表 *4: QMA* 上收到的出版物

预订	主题字符串	收到的出版物	注意
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	WILDCARD(BLOCK) 在 Sports/Football 上阻止了对 Football 子树的所有发布
SARSENAL	Sports/#/Arsenal	-	Sports/Football 上的 WILDCARD(BLOCK) 会阻止 Arsenal 上的通配符预订
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby 上的缺省 WILDCARD 不会阻止 Leeds 上的通配符预订。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal 接收发布，因为预订没有通配符。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds 将在任何情况下接收发布。

第 67 页的表 5 显示了在 QMB 上添加两个新预订以及在 QMA 上发布的结果。 请注意，在没有这两个新预订的情况下， QMB 未收到任何发布。 正如预期的那样，这两个新预订将接收发布内容，因为 Sports/

FootBall 和 Sports/Rugby 都是集群主题。 QMB 将 Sports/Football/Arsenal 和 Sports/Rugby/Leeds 的代理预订转发到 QMA，然后将发布发送到 QMB。

意外的结果是，先前未收到任何发布的两个预订 Sports/# 和 Sports/#/Leeds 现在接收发布。 原因是针对其他预订转发到 QMB 的 Sports/Football/Arsenal 和 Sports/Rugby/Leeds 发布现在可用于连接到 QMB 的任何订户。 因此，本地主题 Sports/# 和 Sports/#/Leeds 的预订将接收 Sports/Rugby/Leeds 发布。 Sports/#/Arsenal 继续不接收发布内容，因为 Sports/Football 已将其 **通配符** 属性值设置为 BLOCK。

表 *5: QMB* 上收到的出版物

预订	**主题字符串**	**收到的出版物**	**注意**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*WILDCARD(BLOCK)* 在 *Sports/Football* 上阻止了对 *Football* 子树的所有发布
SARSENAL	Sports/#/Arsenal	-	Sports/Football 上的 WILDCARD(BLOCK) 会阻止 Arsenal 上的通配符预订
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Sports/Rugby* 上的缺省 *WILDCARD* 不会阻止 *Leeds* 上的通配符预订。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal 接收发布，因为预订没有通配符。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds 将在任何情况下接收发布。

在大多数应用程序中，不希望一个预订影响另一个预订的行为。 **通配符** 属性与值 BLOCK 的一个重要用途是使包含通配符的同一主题字符串的预订行为一致。 无论预订是在与发布程序相同的队列管理器上，还是在不同的队列管理器上，预订的结果都是相同的。

通配符和流

对于写入发布/预订 API 的新应用程序，结果是对 * 的预订不会收到任何发布。 要接收所有体育出版物，必须预订 Sports/*或 Sports/#以及类似的 Business 出版物。

将发布/预订代理迁移到更高版本的 IBM MQ 时，现有已排队的发布/预订应用程序的行为不会更改。 **Publish**， **Register Publisher** 或 **Subscriber** 命令中的 **StreamName** 属性将映射到流已迁移到的主题的名称。

通配符和预订点

对于写入发布/预订 API 的新应用程序，迁移的效果是，对 * 的预订不会收到任何发布。 要接收所有体育出版物，必须预订 Sports/*或 Sports/#以及类似的 Business 出版物。

将发布/预订代理迁移到更高版本的 IBM MQ 时，现有已排队的发布/预订应用程序的行为不会更改。 **Publish**， **Register Publisher** 或 **Subscriber** 命令中的 **SubPoint** 属性将映射到预订已迁移到的主题的名称。

示例: 创建 Sport 发布/预订集群

后续步骤将创建一个具有四个队列管理器的集群 CL1: 两个完整存储库 CL1A 和 CL1B 以及两个部分存储库 QMA 和 QMB。 完整存储库仅用于保存集群定义。 QMA 被指定为集群主题主机。 持久预订同时在 QMA 和 QMB 上定义。

注: 此示例针对 Windows 进行编码。 您必须重新编码 Create qmgrs.bat 和 create pub.bat 以在其他平台上配置和测试示例。

1. 创建脚本文件。
   a. 创建 topics.tst
   b. 创建 wildsubs.tst
   c. 创建 fullsubs.tst
   d. 创建 qmgrs.bat
   e. 创建 pub.bat
2. 运行 Create qmgrs.bat 以创建配置。

```
qmgrs
```

在 第 65 页的图 23 中创建主题。 图 5 中的脚本将创建集群主题 Sports/Football 和 Sports/Rugby。

注: REPLACE 选项不会替换主题的 TOPICSTR 属性。 TOPICSTR 是在示例中进行有效更改以测试不同主题树的属性。 要更改主题，请先删除该主题。

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

图 *25:* 删除并创建主题*: topics.tst*

注: 删除主题，因为 REPLACE 不会替换主题字符串。

使用通配符创建预订。 通配符将主题与 第 65 页的图 23 中的主题对象相对应。 为每个预订创建一个队列。 运行或重新运行脚本时，将清除队列并删除预订。

注: REPLACE 选项不会替换预订的 TOPICOBJ 或 TOPICSTR 属性。 TOPICOBJ 或 TOPICSTR 是在用于测试不同预订的示例中进行有效更改的属性。 要对其进行更改，请先删除预订。

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

图 *26:* 创建通配符预订*: wildsubs.tst*

创建引用集群主题对象的预订。

注:

将在 TOPICOBJ 引用的主题字符串与 TOPICSTR 定义的主题字符串之间自动插入定界符 /。

定义 DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) 将创建相同的预订。 TOPICOBJ 用作引用您已定义的主题字符串的快速方法。 创建预订时，不再引用主题对象。

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

图 *27:* 删除并创建预订*: fullsubs.tst*

创建具有两个存储库的集群。 创建两个用于发布和预订的部分存储库。 重新运行脚本以删除所有内容，然后再次启动。 该脚本还会创建主题层次结构和初始通配符预订。

注:

在其他平台上，编写类似的脚本，或者输入所有命令。 通过使用脚本，可以快速删除所有内容，并使用相同的配置重新开始。

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

图 *28:* 创建队列管理器*: qmgrs.bat*

通过将预订添加到集群主题来更新配置。

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

图 *29:* 更新预订*: upsubs.bat*

运行以队列管理器作为参数的 pub.bat，以发布包含发布主题字符串的消息。 Pub.bat 使用样本程序 **amqspub**。

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

图 *30: 发布: pub.bat*

流和主题

已排队的发布/预订具有集成发布/预订模型中不存在的发布流的概念。 在已排队的发布/预订中，流提供了一种分隔不同主题的信息流的方法。 流作为顶级主题实现，可通过管理方式映射到其他主题标识。

将为网络上的所有代理和队列管理器自动设置缺省流 SYSTEM.BROKER.DEFAULT.STREAM，并且无需其他配置即可使用缺省流。 将缺省流视为未命名的缺省主题空间。 发布到缺省流的主题立即可供所有已连接的队列管理器使用，并且已启用排队的发布/预订。 指定的流类似于单独的，指定的主题空间。 必须在使用指定流的每个代理上定义该流。

如果发布者和订户位于不同的队列管理器上，那么在同一代理层次结构中连接代理之后，不需要进一步配置这些发布以及预订之间的流。 相同的互操作性也会逆向工作。

指定的流

解决方案设计人员在使用排队的发布/预订编程模型时，可能会决定将所有体育出版物放入名为 Sport 的指定流中。 要使流可用于在启用了排队发布/预订的 IBM MQ 上运行的队列管理器，必须手动添加该流。

在流 Sport 上预订 Soccer/Results 的已排队发布/预订应用程序工作而不进行更改。 使用 MQSUB 预订主题 Sport 并提供主题字符串 Soccer/Results 的集成发布/预订应用程序也会接收相同的发布。

添加流主题中描述了添加流的任务。 由于两个原因，您可能需要手动添加流。

1. 继续开发在更高版本队列管理器上运行的已排队发布/预订应用程序，而不是将这些应用程序迁移到集成发布/预订 MQI 接口。
2. 流到主题的缺省映射会导致主题空间中的 "冲突"，并且流上的发布与来自其他位置的发布具有相同的主题字符串。

权限

缺省情况下，在主题树的根目录中有多个主题对象: SYSTEM.BASE.TOPIC，SYSTEM.BROKER.DEFAULT.STREAM 和 SYSTEM.BROKER.DEFAULT.SUBPOINT。 权限 (例如，用于发布或预订) 由 SYSTEM.BASE.TOPIC 上的权限确定; 将忽略 SYSTEM.BROKER.DEFAULT.STREAM 或 SYSTEM.BROKER.DEFAULT.SUBPOINT 上的任何权限。 如果删除 SYSTEM.BROKER.DEFAULT.STREAM 或 SYSTEM.BROKER.DEFAULT.SUBPOINT 中的任何一个并使用非空主题字符串重新创建，那么将以与普通主题对象相同的方式使用对这些对象定义的权限。

流与主题之间的映射

通过创建队列并为其提供与流相同的名称，将在 IBM MQ 中模拟已排队的发布/预订流。 有时该队列被称为流队列，因为这就是它在排队的发布/预订应用程序中的显示方式。 通过将队列添加到名为 SYSTEM.QPUBSUB.QUEUE.NAMELIST 的特殊名称列表，可将该队列标识到发布/预订引擎。 通过向名称列表添加其他特殊队列，可以根据需要添加任意数量的流。 最后需要添加主题，名称与流相同，主题字符串与流名称相同，因此可以发布和预订主题。

但是，在特殊情况下，您可以为与流对应的主题提供您在定义主题时选择的任何主题字符串。 主题字符串的目的是在主题空间中为主题提供唯一的名称。 通常，流名称完全满足此目的。 有时，流名称与现有主题名称冲突。 要解决此问题，请为与流关联的主题选择另一主题字符串。 选择任何主题字符串，确保其唯一。

主题定义中定义的主题字符串以常规方式作为发布程序和订户使用 MQOPEN 或 MQSUB MQI 调用提供的主题字符串的前缀。 使用主题对象引用主题的应用程序不受前缀主题字符串选项的影响-因此，您可以选择任何在主题空间中保持出版物唯一的主题字符串。

将不同流重新映射到不同主题依赖于用于唯一主题字符串的前缀，以将一组主题与另一组主题完全分开。 您必须定义严格遵循的通用主题命名约定，才能使映射生效。

在 IBM MQ 中，使用前缀机制将主题字符串重新映射到主题空间中的其他位置。

注: 删除流时，请先删除该流上的所有预订。 如果任何预订源自代理层次结构中的其他代理，那么此操作最为重要。

预订点和主题

指定的预订点由主题和主题对象进行仿真。

要手动添加预订点，请参阅 添加预订点。

IBM MQ 中的预订点

IBM MQ 将预订点映射到 IBM MQ 主题树中的不同主题空间。 没有预订点的命令消息中的主题将以未更改的方式映射到 IBM MQ 主题树的根，并从 SYSTEM.BASE.TOPIC 继承属性。

具有预订点的命令消息正在使用 SYSTEM.QPUBSUB.SUBPOINT.NAMELIST 中的主题对象列表进行处理。 命令消息中的预订点名称与列表中每个主题对象的主题字符串相匹配。 如果找到匹配项，那么会将预订点名称作为主题节点附加到主题字符串。 主题从 SYSTEM.QPUBSUB.SUBPOINT.NAMELIST 中找到的关联主题对象继承其属性。

使用预订点的效果是为每个预订点创建单独的主题空间。 主题空间植根于与预订点同名的主题中。 每个主题空间中的主题从与预订点同名的主题对象继承其属性。

将以正常方式从 SYSTEM.BASE.TOPIC 继承匹配主题对象中未设置的任何属性。

现有已排队的发布/预订应用程序使用 MQRFH2 消息头，通过在 Publish 或 Register subscriber 命令消息中设置 **SubPoint** 属性来继续工作。 预订点与命令消息中的主题字符串组合在一起，并与任何其他主题一样处理生成的主题。

IBM MQ 应用程序不受预订点影响。 如果应用程序使用从其中一个匹配主题对象继承信息的主题，那么该应用程序将使用匹配的预订点与排队的应用程序进行互操作。

示例

现有 WebSphere Message Broker (现在称为 IBM Integration Bus) 迁移到 IBM MQ 的发布/预订应用程序创建了两个主题对象 GBP 和 USD 以及相应的主题字符串 'GBP' 和 'USD'。

已迁移到 IBM MQ 上运行的主题 NYSE/IBM/SPOT 的现有发布程序，这些发布程序使用预订点 USD 在主题 USD/NYSE/IBM/SPOT 上创建发布。 与 NYSE/IBM/SPOT 的现有订户类似，使用预订点 USD 创建对 USD/NYSE/IBM/SPOT 的预订。

通过调用 MQSUB 在 IBM MQ 发布/预订程序中预订美元现货价格。 使用 USD 主题对象和主题字符串 'NYSE/IBM/SPOT'创建预订，如 "C" 代码片段中所示。

```
strncpy(sd.ObjectName, "USD", MQ_TOPIC_NAME_LENGTH);
sd.ObjectString.VSPtr = "NYSE/IBM/SPOT";
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

1. 在集群主题主机上设置 USD 和 GBP 主题对象的 CLUSTER 属性。
2. 删除集群中其他队列管理器上 USD 和 GBP 主题对象的所有副本。
3. 确保在集群中的每个队列管理器上的 SYSTEM.QPUBSUB.SUBPOINT.NAMELIST 中定义了 USD 和 GBP 。

单个队列管理器发布/预订配置的示例

第 72 页的图 31 说明了基本的单个队列管理器发布/预订配置。 此示例显示了新闻服务的配置，其中发布者提供了有关多个主题的信息:

- 发布程序 1 正在使用 "运动" 主题发布有关运动结果的信息
- 发布程序 2 正在使用 "股票" 主题发布有关股票价格的信息
- 发布者 3 正在使用 "电影" 主题发布有关电影评论的信息，并使用 "电视" 主题发布有关电视列表的信息

三个订户已注册了对不同主题的兴趣，因此队列管理器会向他们发送他们感兴趣的信息:

- 订户 1 接收体育结果和股价
- 订户 2 接收影片评论
- 订户 3 接收体育结果

没有一个用户登记了对电视列表的兴趣，所以这些都没有分发。

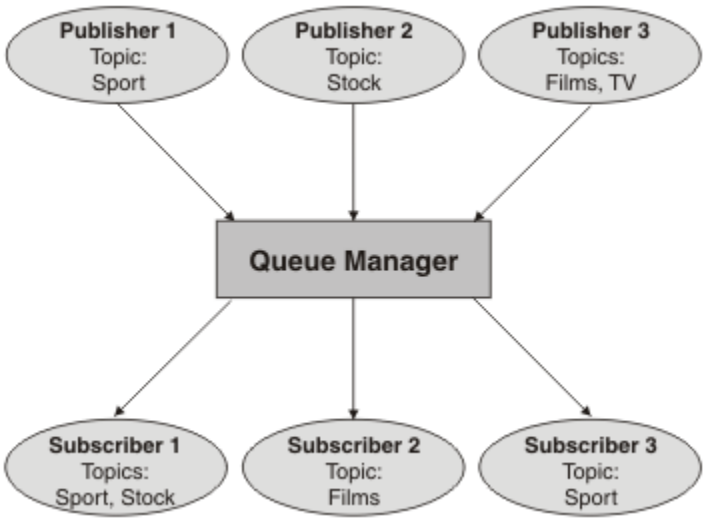


图 *31:* 单个队列管理器发布/预订示例

分布式发布/预订网络

每个队列管理器将发布到主题的消息与本地创建的预订 (已预订该主题) 相匹配。 您可以配置队列管理器网络，以便将连接到一个队列管理器的应用程序发布的消息传递到网络中其他队列管理器上创建的匹配预订。 这需要通过队列管理器之间的简单通道进行其他配置。

分布式发布/预订配置是一组连接在一起的队列管理器。 队列管理器可以全部位于同一物理系统上，也可以分布在多个物理系统上。 将队列管理器连接在一起时，订户可以预订一个队列管理器，并接收最初发布到另一个队列管理器的消息。 为了说明这一点，下图向 第 71 页的『单个队列管理器发布/预订配置的示例』中描述的配置添加了第二个队列管理器。

- 队列管理器 2 由发布者 4 使用 "天气" 主题发布天气预报信息，并使用 "交通" 主题发布有关主要道路交通状况的信息。
- 订户 4 还使用此队列管理器，并使用主题 "流量" 预订有关流量条件的信息。
- 订户 3 还预订有关天气状况的信息，即使它使用与发布程序不同的队列管理器也是如此。 这是可能的，因为队列管理器相互链接。

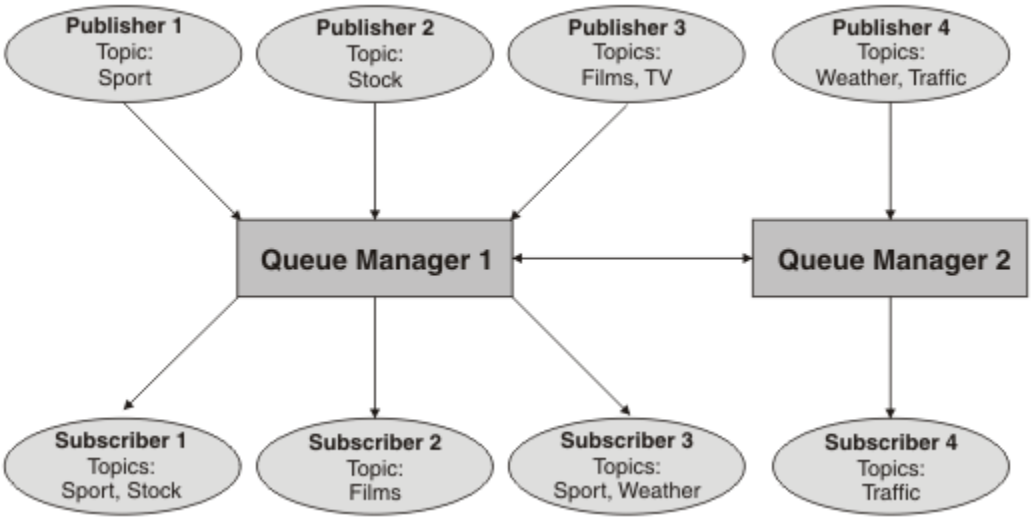


图 *32:* 具有两个队列管理器的发布/预订示例

您可以手动连接父层次结构和子层次结构中的队列管理器，也可以创建发布/预订集群，并让 IBM MQ 为您定义许多连接详细信息。 您还可以组合使用这两种拓扑，例如，通过在层次结构中将多个集群连接在一起。

发布/预订集群概述

发布/预订集群是向集群添加了一个或多个主题对象的标准集群。 当您在集群中的任何队列管理器上定义 管理主题对象，并通过指定集群名称使该主题对象成为集群对象时，该主题的发布者和订户可以连接到集群中的任何队列管理器，并且发布的消息将通过队列管理器之间的集群通道路由到订户。

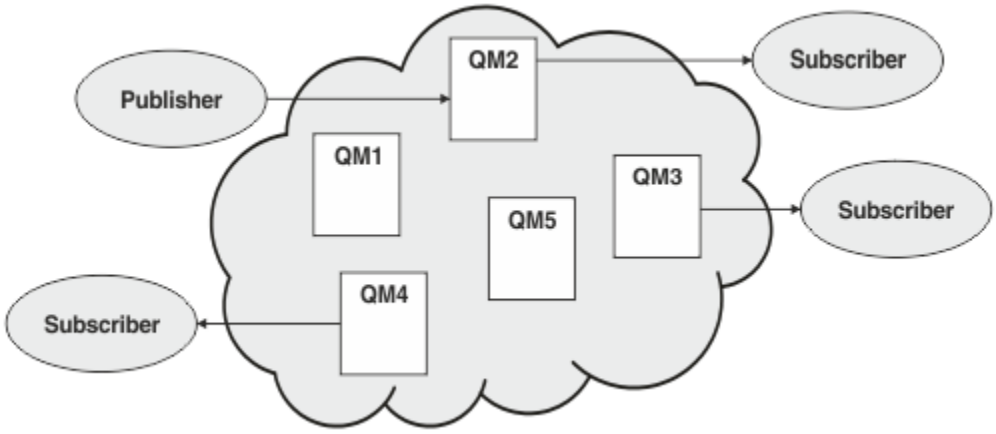


图 *33:* 发布/预订集群 *(publish/subscribe cluster)*

有两种方法可配置如何在集群中路由发布/预订消息:

- 直接路由 (direct routing)
- 主题主机路由 (topic host routing)

当您配置直接路由的集群主题时，在一个队列管理器上发布的消息将直接从该队列管理器发送到集群中任何其他队列管理器上的每个预订。 这可以为发布提供最直接的路径，但确实会导致集群中的所有队列管理器了解所有其他队列管理器，每个队列管理器都可能在它们之间建立集群通道。

使用主题主机路由时，在一个队列管理器上发布的消息将从该队列管理器发送到主管受管主题对象定义的队列管理器。 主题主机队列管理器会将消息路由至集群中其他任何队列管理器上的每个预订。 如果发布者或订户不在主题主机队列管理器上，那么这将导致发布的路径较长。 但是，好处是只有主题主机队列管理器才知道集群中的所有其他队列管理器，并且可能与它们一起建立了集群通道。

有关更多信息，请参阅 第 75 页的『发布/预订集群』。

发布/预订层次结构概述

发布/预订层次结构是由通道连接到分层结构中的一组队列管理器。 每个队列管理器标识其 父 队列管理器，如 将队列管理器连接到发布/预订层次结构中所述。

主题的发布者和订户可以连接到层次结构中的任何队列管理器，并且消息使用分层队列管理器连接在它们之间流动。

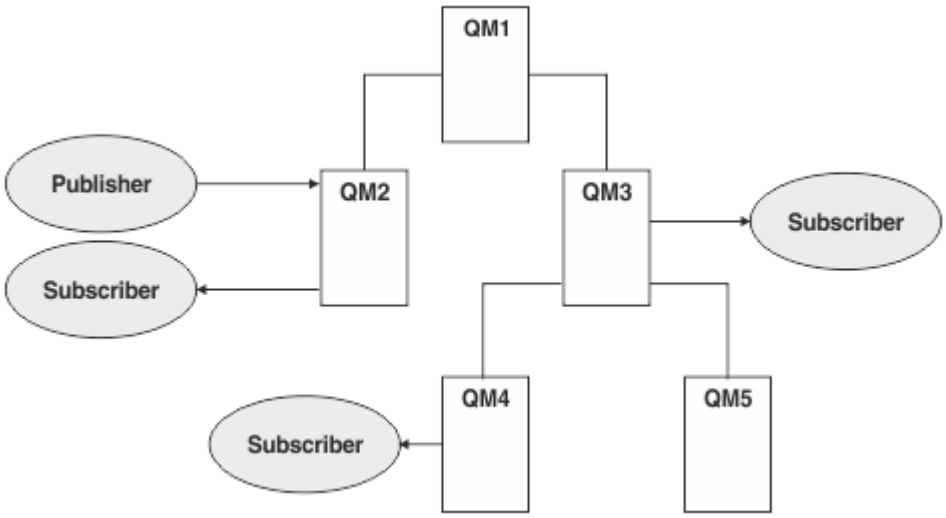


图 *34:* 发布/预订层次结构

在上图中，传递到 QM3 和 QM4 上的订户的出版物已从 QM2 路由到 QM1 ， 然后传递到 QM3 ， 最后传递到 QM4。

层次结构使您能够直接控制层次结构中每个队列管理器之间的关系。 这允许对从发布者到订户的消息路由进行细颗粒度控制，并且在具有受限连接的队列管理器网络之间进行路由时特别有用。 您应该仔细考虑每个队列管理器的可用性和功能，通过这些队列管理器将消息从发布者路由到订户。

有关更多信息，请参阅 第 76 页的『发布/预订层次结构』。

队列管理器之间的发布分布

除了路由选项外，还有两种方法可以在队列管理器网络中分发发布内容:

- 仅将发布从一个队列管理器发送到当前主管该发布的预订的队列管理器。
- 将每个发布发送到所有队列管理器，并使其与预订相匹配。

前者导致仅在必要时发送发布消息，但需要在队列管理器之间共享一定级别的预订知识。 后者不需要共享预订知识，但会导致在队列管理器之间发送不必要的发布消息。

缺省情况下， IBM MQ 使用前一种方法，在此方法中，发布仅发送到具有预订的队列管理器。 预订知识以代理预订形式在队列管理器之间传播。 这取决于预订的分布和生存期，以及发布的频率，因为在分布式发布/预订拓扑中使用的效率最高。 请参阅 发布/预订网络中的预订性能。

相关概念

第 62 页的『主题树』
您定义的每个主题都是主题树中的一个元素或节点。 主题树可以为空以开始，也可以包含先前使用 MQSC 或 PCF 命令定义的主题。 通过使用“创建主题”命令或通过指定主题，可以在发布或预订中首次定义新的主题。

发布/预订层次结构方案

相关任务

设计发布/预订集群

发布/预订集群

发布/预订集群是互连队列管理器的标准集群，在该集群上，发布将自动从发布应用程序移至集群中任何队列管理器上存在的预订。 提供两个选项用于在发布/预订集群中路由发布：直接路由和主题主机路由。 您选择的路由取决于集群的大小和期望的活动模式。

用于发布/预订消息传递的集群与标准 IBM MQ 集群没有任何不同。 因此，发布/预订集群中的队列管理器可以存在于物理上独立的计算机上，必要时，集群通道会自动将每对队列管理器连接在一起。 有关更多信息，请参阅 集群。

要配置标准的队列管理器集群以进行发布/预订消息传递，可在集群中的一个队列管理器上定义一个或多个受管主题对象。 要使主题成为集群主题，请使用集群的名称配置 **CLUSTER** 属性。 执行此操作时， 主题树 中的发布者或订户在该点或以下位置使用的任何主题将在集群中的所有队列管理器之间共享，并且发布到主题树的集群分支的消息将自动路由到集群中其他队列管理器上的预订。

在发布者队列管理器和其他队列管理器之间仅发送每条消息的一个副本，而不考虑目标队列管理器上该消息的订户数。 到达具有一个或多个预订的队列管理器时，将在所有预订之间复制消息。

任何加入集群的队列管理器都会自动了解集群主题，并且该队列管理器上的发布者和订户会自动参与集群。

通过使用不属于集群主题对象的主题字符串，还可以在发布/预订集群中执行非集群发布/预订活动。

提供两个选项用于在发布/预订集群中路由发布：直接路由和主题主机路由。 要选择在集群中使用的消息路由，请将受管主题对象上的 **CLROUTE** 属性设置为以下值之一：

- **DIRECT**
- **TOPICHOST**

缺省情况下，主题路由为 **DIRECT**。 当在队列管理器上配置直接路由集群主题时，集群中的所有队列管理器都可识别集群中的所有其他队列管理器。 在执行发布和预订操作时，每个队列管理器都可以直接连接到集群中的任何其他队列管理器。

从 IBM MQ 8.0 开始，您可以改为将主题路由配置为 **TOPICHOST**。 在您使用主题主机路由时，集群中的所有队列管理器都会知晓托管了路由主题定义的集群队列管理器（即，已定义主题对象的队列管理器）。 在执行发布和预订操作时，集群中的队列管理器只会连接到这些主题主机队列管理器，而不会彼此直接连接。 主题主机队列管理器负责将发布从执行发布的队列管理器路由至具有匹配预订的队列管理器。

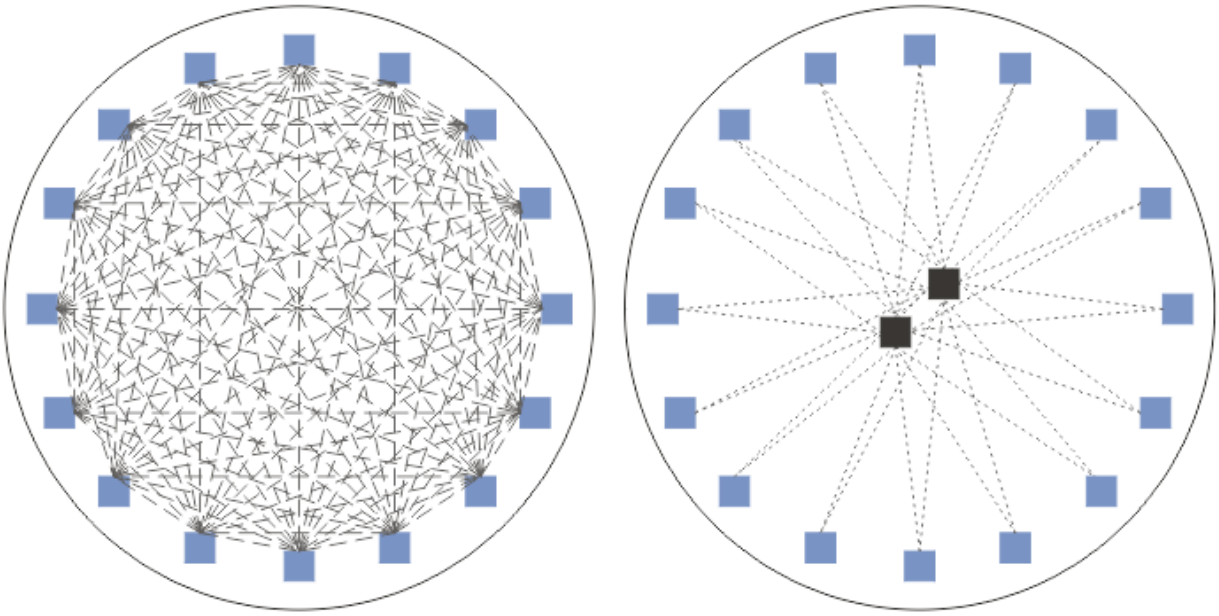


图 *35:* 直接路由和主题主机路由

直接路由概述

配置受管主题对象以进行直接路由时，只需要在集群中的其中一个队列管理器上定义主题对象，所有队列管理器才能了解该对象。 定义主题的队列管理器选项不会影响主题的发布/预订消息传递行为。

每条消息都直接从发布者队列管理器流向集群中其他队列管理器上的每个预订，而不是通过任何中间队列管理器传递。

缺省情况下，消息仅发送到集群中托管一个或多个预订的其他队列管理器。

- 这依赖于每个队列管理器直接向集群中的所有其他队列管理器通知当前具有一个或多个预订的所有主题。 这将导致集群中的所有队列管理器都知道正在预订的所有主题，以及托管预订的任何队列管理器都将建立到每个其他队列管理器的通道。 这与每个队列管理器是否具有发布程序无关。
- 可以通过更改为向集群中的所有队列管理器发送所有发布的模型 (无论它们是否具有预订) 来除去所有队列管理器上的每个单独预订主题的知识。 这将减少预订知识流量，但可能会增加发布流量以及每个队列管理器建立的通道数。 请参阅 发布/预订网络中的预订性能。

使用直接路由的集群主题的发布/预订消息流可以跨多个发布/预订集群，方法是将每个集群中的一个队列管理器添加到发布/预订层次结构中。 请参阅 组合多个集群的主题空间。

有关直接路由的更详细探索，请参阅 发布/预订集群中的直接路由。

主题主机路由概述

当为主题主机路由配置了受管主题对象时，来自集群中队列管理器的发布将通过配置了主题对象的队列管理器 ("主题主机") 进行路由，并从该队列管理器路由到存在预订的队列管理器。

- 这依赖于每个队列管理器向所有主题主机通知当前具有一个或多个预订的每个主题。 托管预订的任何队列管理器都会为与预订相关的主题建立到每个主题主机的通道。
- 出于发布/预订目的，不会使非主题托管队列管理器知道集群中的其他非主题托管队列管理器，并且不会为此目的在它们之间建立通道。
- 如果发布应用程序连接到托管主题的队列管理器，那么已发布的消息将直接路由到已创建匹配预订的队列管理器，而不需要额外的 "中继段"。 同样，如果在托管主题的唯一队列管理器上创建匹配的预订，那么发布到该主题的消息将直接路由到该队列管理器，而不需要额外的中继段。
- 将满足与发布程序相同的队列管理器上的预订，而不首先将发布路由到主题对象的主机。

对于集群队列，多个队列管理器可以配置相同的管理主题对象。 这将提供更高的消息路由可用性，并通过工作负载均衡进行水平缩放。 对于主题主机路由的主题对象，当多个队列管理器为主题树的同一分支配置同一指定主题时，托管预订的每个队列管理器将使每个主题主机了解预订的主题。

- 发布消息时，会将其发送到其中一个主题主机队列管理器，以转发到预订托管队列管理器。 主题主机队列管理器的选择遵循与集群点到点队列相同的缺省工作负载均衡规则。
- 如果发布队列管理器无法联系一个或多个主题主机队列管理器，那么会将消息路由到剩余的可用主题主管队列管理器。

主题树的路由分支中的主题的每个发布都将转发到其中一个主题主机，即使集群中的任何位置都没有该主题的预订也是如此。 缺省情况下，消息仅从此处发送到集群中托管一个或多个预订的其他队列管理器。

- 这依赖于向每个主题主机队列管理器通知集群中每个队列管理器上的所有预订主题字符串。
- 可以通过更改为将路由到主题主机的所有发布发送到集群中所有队列管理器的模型来除去每个单独预订的主题的知识，而不考虑它们是否具有预订。 这将减少预订知识流量，但可能会增加发布流量，并可能增加使用每个主题托管队列管理器建立的通道数。 请参阅 发布/预订网络中的预订性能。

使用主题主机路由的集群主题的发布/预订消息流 **不能** 通过使用发布/预订层次结构跨多个发布/预订集群。

有关主题主机路由的更详细探索，请参阅 发布/预订集群中的主题主机路由。

发布/预订层次结构

通过使用通道将队列管理器链接在一起，然后定义队列管理器对之间的子/父关系，构建发布/预订层次结构。 消息通过层次结构中的直接关系从发布者流向预订。 请注意，这可能意味着要到达多个 “中继段” 。

在任何一对队列管理器之间仅发送消息的一个副本，而不考虑目标队列管理器上消息的订户数。 到达具有一个或多个预订的队列管理器时，将在所有预订之间复制消息。

缺省情况下，仅将消息发送到层次结构中的其他队列管理器，这些队列管理器位于另一个队列管理器上的预订的路由上:

- 这依赖于每个队列管理器通知当前具有一个或多个预订的所有主题的每个直接关系 (在此队列管理器上或在其另一个关系上)。 这将导致层次结构中的所有队列管理器都知道要预订的所有主题。
- 可以将此行为更改为始终向层次结构中的所有队列管理器发送发布，而不考虑是否存在任何预订。 这样就不需要在层次结构中传播预订信息，但可以增加发布流量。

创建集群时，需要注意不要创建导致消息在网络中永久循环的循环。 不能在层次结构中创建此类循环。

每个队列管理器都必须具有唯一的队列管理器名称。

发布/预订消息流可以跨多个发布/预订集群。 为此，请将每个集群中的一个队列管理器添加到发布/预订层次结构中。

有关更详细的探索，请参阅 在发布/预订层次结构中路由。

发布/预订网络中的代理预订

代理预订是一个队列管理器为另一个队列管理器上发布的主题所作的预订。 代理预订会针对预订所预订的每一个主题字符串在队列管理器间流动。 您无需明确创建代理预订，队列管理器会替您执行此操作。

您可以将队列管理器一起连接到发布/预订集群或发布/预订层次结构中。 代理预订在已连接的队列管理器之间流动。 代理预订导致连接到一个队列管理器的发布程序创建的主题的发布由连接到其他队列管理器的该主题的订户接收。 请参阅第 72 页的『分布式发布/预订网络』。

在具有针对各个主题字符串的数千个预订的发布/预订拓扑中，或者在这些预订的存在可能快速变化的情况下，必须考虑代理预订传播的开销。 除了本主题的其余部分中描述的自动聚集外，您还可以进行手动配置更改，以进一步限制已连接队列管理器之间的代理预订和发布的流，并减少等待代理预订传播到所有已连接队列管理器的等待时间。 请参阅 发布/预订网络中的预订性能。

代理预订不包含本地预订所使用的任何选择器，可以简化包含通配符的预订主题字符串。 这可能会导致发布与实际预订不匹配的代理预订相匹配，从而在队列管理器之间产生额外的发布流。 托管预订的队列管理器会过滤掉此类差异，以便不会将其他发布返回给预订。

代理预订聚集

代理预订是使用重复消除系统聚集的。 对于特定的已解析主题字符串，将在第一个本地预订或接收到的代理预订上发送代理预订。 对同一主题字符串的后续预订将使用此现有代理预订。

在取消最后一个本地预订或接收到的代理预订之后，将取消代理预订。

发布聚集

当队列管理器上存在对同一主题字符串的多个预订时，仅从发布/预订拓扑中的其他队列管理器发送与该主题字符串匹配的每个发布的单个副本。 当消息到达时，本地队列管理器会将消息副本传递到每个匹配的预订。

当代理预订包含通配符时，可能有多个代理预订与单个发布的主题字符串匹配。 如果在与单个连接的队列管理器创建的两个或多个代理预订匹配的队列管理器上发布消息，那么仅会将该发布的一个副本转发到远程队列管理器以满足多个代理预订。

相关概念

分布式发布/预订网络中进行回路检测

代理预订中的通配符

预订可以使用主题字符串中的通配符来匹配发布中的多个主题字符串。

预订可以使用两个通配符模式: 基于主题的 和 基于字符的。 请参阅第 57 页的『通配方案』。

在 IBM MQ 中，通配符预订的所有代理预订都将转换为使用基于主题的通配符。 如果找到基于字符的通配符，那么会将其替换为 # 字符，返回到最近的 /。 例如， /aaa/bbb/c*d 将转换为 /aaa/bbb/#。 此转换

导致远程队列管理器发送的发布数略多于显式预订的发布数。 当本地队列管理器将发布内容传递给其本地订户时，会将其他发布内容过滤掉。

使用 通配符 属性控制通配符使用

使用 MQSC **Topic** 通配符 属性或等效 PCF 主题 `WildcardOperation` 属性来控制向使用通配符主题字符串名称的订户应用程序传递发布内容。 通配符 属性可以具有以下两个可能的值之一:

WILDCARD
与此主题有关的通配符预订的行为。

PASSTHRU
对没有此主题对象中主题字符串具体的通配主题进行的预订将接收对此主题以及比此主题更具体的主题字符串进行的发布。

BLOCK
对没有此主题对象中主题字符串具体的通配主题进行的预订不会接收对此主题或比此主题更具体的主题字符串进行的发布。

在定义预订时将使用此属性的值。 如果改变此属性，那么现有预订涵盖的主题集不会因为此修改而受到影响。 如果在创建或删除主题对象时拓扑发生更改，此场景也适用；将使用修改后的拓扑来创建与修改 WILDCARD 属性后创建的预订匹配的主题集。 如果要针对现有预订强制重新评估匹配的主题集，那么必须重新启动队列管理器。

在示例 第 67 页的『示例: 创建 Sport 发布/预订集群』中，您可以遵循步骤来创建 第 65 页的图 23 中显示的主题树结构。

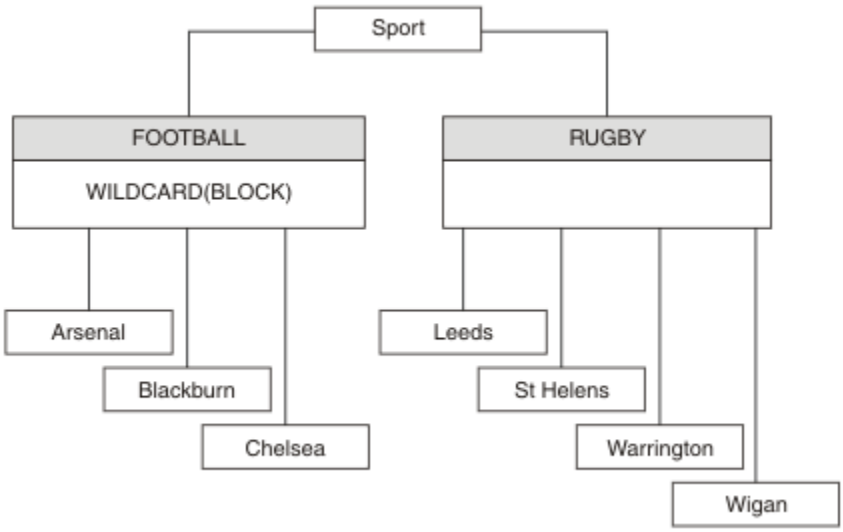


图 *36:* 使用 通配符 属性 *BLOCK* 的主题树

使用通配符主题字符串 *#* 的订户将接收到 `Sport` 主题和 `Sport/Rugby` 子树的所有发布。 订户不会接收到 `Sport/Football` 子树的发布，因为 `Sport/Football` 主题的 **通配符** 属性值为 `BLOCK`。

`PASSTHRU` 是缺省设置。 您可以将 **通配符** 属性值 `PASSTHRU` 设置为 `Sport` 树中的节点。 如果节点没有 **通配符** 属性值 `BLOCK`，那么设置 `PASSTHRU` 不会改变 `Sports` 树中节点的订户观察到的行为。

在此示例中，创建预订以查看通配符设置如何影响交付的发布; 请参阅 第 69 页的图 27。 在 第 70 页的图 30 中运行发布命令以创建一些发布。

```
pub QMA
```

图 *37:* 发布到 *QMA*

第 65 页的表 3 显示了结果。 请注意，如何设置 **通配符** 属性值 `BLOCK`，以防止具有通配符的预订将发布内容接收到通配符作用域内的主题。

表 6: QMA 上收到的出版物			
预订	**主题字符串**	**收到的出版物**	**注意**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	WILDCARD(BLOCK) 在 Sports/Football 上阻止了对 Football 子树的所有发布
SARSENAL	Sports/#/Arsenal	-	Sports/Football 上的 WILDCARD(BLOCK) 会阻止 Arsenal 上的通配符预订
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby 上的缺省 WILDCARD 不会阻止 Leeds 上的通配符预订。

注:

假设预订具有通配符，该通配符与具有 **通配符** 属性值 BLOCK 的主题对象相匹配。 如果预订还具有匹配通配符右侧的主题字符串，那么预订将从不接收发布内容。 未被阻止的发布集合是作为被阻止通配符的父代的主题的发布。 通配符将阻止发布到作为具有 BLOCK 属性值的主题的子代的主题。 因此，包含通配符右侧主题的预订主题字符串永远不会收到任何要匹配的发布内容。

将 WILDCARD 属性值设置为 BLOCK 并不意味着您无法使用包含通配符的主题字符串进行预订。 这样的订阅是正常的。 预订具有一个显式主题，该主题与具有 **通配符** 属性值 BLOCK 的主题对象匹配。 它将通配符用于作为具有 **通配符** 属性值 BLOCK 的主题的父代或子代的主题。 在 第 65 页的图 23 中的示例中，预订 (例如 Sports/Football/#) 可以接收发布。

通配符和集群主题

集群主题定义将传播到集群中的每个队列管理器。 在集群中的一个队列管理器上预订集群主题会导致队列管理器创建代理预订。 将在集群中的每个其他队列管理器上创建代理预订。 使用包含通配符的主题字符串 (与集群主题结合使用) 的预订可能难以预测行为。 此行为在以下示例中进行了说明。

在为示例 第 67 页的『示例: 创建 Sport 发布/预订集群』设置的集群中， QMB 具有与 QMA 相同的预订集，但 QMB 在发布者发布到 QMA 之后未收到任何发布，请参阅 第 65 页的图 24。 虽然 Sports/Football 和 Sports/Rugby 主题是集群主题，但 fullsubs.tst 中定义的预订不会引用集群主题。 不会将代理预订从 QMB 传播到 QMA。 如果没有代理预订，那么不会将 QMA 的任何发布转发到 QMB。

某些预订 (例如 Sports/#/Leeds) 可能似乎引用了集群主题，在本例中为 Sports/Rugby。 Sports/#/Leeds 预订实际解析为主题对象 SYSTEM.BASE.TOPIC。

用于解析预订 (例如， Sports/#/Leeds) 所引用的主题对象的规则如下所示。 将主题字符串截断为第一个通配符。 通过主题字符串向左扫描以查找具有关联管理主题对象的第一个主题。 主题对象可以指定集群名称，也可以定义本地主题对象。 在示例 Sports/#/Leeds 中，截断后的主题字符串是 Sports，它没有主题对象，因此 Sports/#/Leeds 继承自 SYSTEM.BASE.TOPIC(这是本地主题对象)。

要查看预订集群主题如何更改通配符传播的工作方式，请运行批处理脚本 upsubs.bat。 该脚本将清除预订队列，并在 fullsubs.tst 中添加集群主题预订。 再次运行 puba.bat 以创建一批出版物; 请参阅 第 65 页的图 24。

第 66 页的表 4 显示了将两个新预订添加到发布发布的同一队列管理器的结果。 结果与预期相同，新预订各接收一个发布， 其他预订接收的发布数量保持不变。 意外结果发生在其他集群队列管理器上; 请参阅 第 67 页的表 5。

表 *7: QMA* 上收到的出版物			
预订	**主题字符串**	**收到的出版物**	**注意**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	WILDCARD(BLOCK) 在 Sports/Football 上阻止了对 Football 子树的所有发布
SARSENAL	Sports/#/Arsenal	-	Sports/Football 上的 WILDCARD(BLOCK) 会阻止 Arsenal 上的通配符预订
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby 上的缺省 WILDCARD 不会阻止 Leeds 上的通配符预订。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal 接收发布，因为预订没有通配符。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds 将在任何情况下接收发布。

第 67 页的表 5 显示了在 QMB 上添加两个新预订以及在 QMA 上发布的结果。 请注意，在没有这两个新预订的情况下， QMB 未收到任何发布。 正如预期的那样，这两个新预订将接收发布内容，因为 Sports/FootBall 和 Sports/Rugby 都是集群主题。 QMB 将 Sports/Football/Arsenal 和 Sports/Rugby/Leeds 的代理预订转发到 QMA，然后将发布发送到 QMB。

意外的结果是，先前未收到任何发布的两个预订 Sports/# 和 Sports/#/Leeds 现在接收发布。 原因是针对其他预订转发到 QMB 的 Sports/Football/Arsenal 和 Sports/Rugby/Leeds 发布现在可用于连接到 QMB 的任何订户。 因此，本地主题 Sports/# 和 Sports/#/Leeds 的预订将接收 Sports/Rugby/Leeds 发布。 Sports/#/Arsenal 继续不接收发布内容，因为 Sports/Football 已将其 **通配符** 属性值设置为 BLOCK。

表 *8: QMB* 上收到的出版物			
预订	**主题字符串**	**收到的出版物**	**注意**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*WILDCARD(BLOCK)* 在 *Sports/Football* 上阻止了对 *Football* 子树的所有发布
SARSENAL	Sports/#/Arsenal	-	Sports/Football 上的 WILDCARD(BLOCK) 会阻止 Arsenal 上的通配符预订
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Sports/Rugby* 上的缺省 *WILDCARD* 不会阻止 *Leeds* 上的通配符预订。
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal 接收发布，因为预订没有通配符。
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds 将在任何情况下接收发布。

在大多数应用程序中，不希望一个预订影响另一个预订的行为。 **通配符** 属性与值 BLOCK 的一个重要用途是使包含通配符的同一主题字符串的预订行为一致。 无论预订是在与发布程序相同的队列管理器上，还是在不同的队列管理器上，预订的结果都是相同的。

通配符和流

对于写入发布/预订 API 的新应用程序，结果是对 * 的预订不会收到任何发布。 要接收所有体育出版物，必须预订 Sports/*或 Sports/#以及类似的 Business 出版物。

将发布/预订代理迁移到更高版本的 IBM MQ 时，现有已排队的发布/预订应用程序的行为不会更改。 **Publish**， **Register Publisher** 或 **Subscriber** 命令中的 **StreamName** 属性将映射到流已迁移到的主题的名称。

通配符和预订点

对于写入发布/预订 API 的新应用程序，迁移的效果是，对 * 的预订不会收到任何发布。 要接收所有体育出版物，必须预订 Sports/*或 Sports/#以及类似的 Business 出版物。

将发布/预订代理迁移到更高版本的 IBM MQ 时，现有已排队的发布/预订应用程序的行为不会更改。 **Publish**， **Register Publisher** 或 **Subscriber** 命令中的 **SubPoint** 属性将映射到预订已迁移到的主题的名称。

示例: 创建 Sport 发布/预订集群

后续步骤将创建一个具有四个队列管理器的集群 CL1: 两个完整存储库 CL1A 和 CL1B 以及两个部分存储库 QMA 和 QMB。 完整存储库仅用于保存集群定义。 QMA 被指定为集群主题主机。 持久预订同时在 QMA 和 QMB 上定义。

注: 此示例针对 Windows 进行编码。 您必须重新编码 Create qmgrs.bat 和 create pub.bat 以在其他平台上配置和测试示例。

1. 创建脚本文件。
   a. 创建 topics.tst
   b. 创建 wildsubs.tst
   c. 创建 fullsubs.tst
   d. 创建 qmgrs.bat
   e. 创建 pub.bat
2. 运行 Create qmgrs.bat 以创建配置。

```
qmgrs
```

在 第 65 页的图 23 中创建主题。 图 5 中的脚本将创建集群主题 Sports/Football 和 Sports/Rugby。

注: REPLACE 选项不会替换主题的 TOPICSTR 属性。 TOPICSTR 是在示例中进行有效更改以测试不同主题树的属性。 要更改主题，请先删除该主题。

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

图 *38:* 删除并创建主题*: topics.tst*

注: 删除主题，因为 REPLACE 不会替换主题字符串。

使用通配符创建预订。 通配符将主题与 第 65 页的图 23 中的主题对象相对应。 为每个预订创建一个队列。 运行或重新运行脚本时，将清除队列并删除预订。

注: REPLACE 选项不会替换预订的 TOPICOBJ 或 TOPICSTR 属性。 TOPICOBJ 或 TOPICSTR 是在用于测试不同预订的示例中进行有效更改的属性。 要对其进行更改，请先删除预订。

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

图 *39:* 创建通配符预订*: wildsubs.tst*

创建引用集群主题对象的预订。

注:

将在 TOPICOBJ 引用的主题字符串与 TOPICSTR 定义的主题字符串之间自动插入定界符 /。

定义 DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) 将创建相同的预订。 TOPICOBJ 用作引用您已定义的主题字符串的快速方法。 创建预订时，不再引用主题对象。

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

图 *40:* 删除并创建预订*: fullsubs.tst*

创建具有两个存储库的集群。 创建两个用于发布和预订的部分存储库。 重新运行脚本以删除所有内容，然后再次启动。 该脚本还会创建主题层次结构和初始通配符预订。

注:

在其他平台上，编写类似的脚本，或者输入所有命令。 通过使用脚本，可以快速删除所有内容，并使用相同的配置重新开始。

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

图 *41:* 创建队列管理器*: qmgrs.bat*

通过将预订添加到集群主题来更新配置。

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

图 *42:* 更新预订*: upsubs.bat*

运行以队列管理器作为参数的 pub.bat，以发布包含发布主题字符串的消息。 Pub.bat 使用样本程序 **amqspub**。

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

图 *43:* 发布*: pub.bat*

相关概念

通配符预订和保留发布

发布作用域

配置发布/预订集群或层次结构时，发布的作用域将进一步控制队列管理器是否将发布转发到远程队列管理器。 使用 **PUBSCOPE** 主题属性来管理发布的作用域。

如果未将发布转发至远程队列管理器，那么只有本地订户才会接收该发布。

使用发布/预订集群时，发布范围主要由主题树中某些点的集群主题对象定义控制。 必须设置发布作用域以允许发布流至集群中的其他队列管理器。 仅当需要对特定队列管理器上的特定主题进行细粒度控制时，才应限制集群主题的发布范围。

使用发布/预订层次结构时，发布的作用域主要由此属性与 预订作用域 属性组合控制。

PUBSCOPE 属性用于确定对特定主题进行的发布的作用域。 可以将属性设置为下列其中一个值:

QMGR
该出版物仅提供给本地订户。 这些出版物称为 本地出版物。 本地发布不会转发至远程队列管理器，因此连接至远程队列管理器的订户不会接收本地发布。

所有
发布将传递到本地订户以及连接到发布/预订集群或层次结构中的远程队列管理器的订户。 这些出版物称为 全局出版物。

ASPARENT
使用主题树中父主题的 **PUBSCOPE** 设置。

发布者还可以使用 MQPMO_SCOPE_QMGR put 消息选项来指定发布是本地发布还是全局发布。 如果使用此选项，那么它将覆盖已使用 **PUBSCOPE** 主题属性设置的任何行为。

相关概念
第 62 页的『管理主题对象』
通过使用管理主题对象，可以将特定非缺省属性分配给主题。

相关任务
配置分布式发布/预订网络

预订作用域

预订的作用域控制一个队列管理器上的预订是接收在发布/预订集群或层次结构中的另一个队列管理器上发布的发布，还是仅接收来自本地发布程序的发布。

将预订作用域限制为队列管理器将阻止代理预订转发到发布/预订拓扑中的其他队列管理器。 这将减少队列管理器之间的发布/预订消息传递流量。

使用发布/预订集群时，预订的作用域主要由主题树中某些点的集群主题对象的定义控制。 必须设置预订作用域以允许代理预订流向集群中的其他队列管理器。 仅当需要对特定队列管理器上的特定主题进行细粒度控制时，才应限制集群主题的预订范围。

使用发布/预订层次结构时，预订的作用域主要由此属性与 发布作用域 属性组合控制。

SUBSCOPE 主题属性用于确定对特定主题进行的预订的作用域。 可以将属性设置为下列其中一个值:

QMGR
预订仅接收本地发布，并且不会将代理预订传播到远程队列管理器。

所有
代理预订传播到发布/预订集群或层次结构中的远程队列管理器，并且订户接收本地和远程发布。

ASPARENT
使用主题树中父主题的 **SUBSCOPE** 设置。

当主题的预订作用域设置为 ALL(直接设置或通过 ASPARENT 解析) 时，针对该主题的各个预订可以通过在创建预订时指定 MQSO_SCOPE_QMGR 来将其作用域限制为 QMGR 。 对作用域为 QMGR 的主题的预订无法将作用域扩大到 ALL。

相关概念
第 62 页的『管理主题对象』
通过使用管理主题对象，可以将特定非缺省属性分配给主题。

相关任务
配置分布式发布/预订网络

主题空间

主题空间是您可以预订和发布的主题集。 分布式发布/预订拓扑中的队列管理器具有一个主题空间，该主题空间可能包含已在该拓扑中的已连接队列管理器上预订和发布的主题。

注: 有关队列管理器中的主题 (例如管理主题对象，主题字符串和主题树) 的概述，请参阅第 55 页的『主题』。 除非另有指定，否则当前文章中对 主题 的进一步引用将引用 主题字符串 。

主题最初是通过以下任一方式创建的:

- 以管理方式定义主题对象或持久预订时。
- 当应用程序动态地创建新主题的发布或预订时。

主题通过代理预订和通过创建管理集群主题对象传播到其他队列管理器。 代理预订导致发布从发布程序所连接的队列管理器转发到订户的队列管理器。

代理预订将在通过队列管理器层次结构中的父子关系连接在一起的所有队列管理器之间传播。 结果是，您可以在一个队列管理器上预订层次结构中任何其他队列管理器上定义的主题。 只要队列管理器之间存在已连接的路径，那么队列管理器的连接方式就无关紧要。

还会为发布/预订集群中的集群主题的预订传播代理预订。 集群主题是附加到具有 **CLUSTER** 属性或从其父代继承该属性的主题对象的主题。 不是集群主题的主题称为本地主题，不会复制到集群。 不会将任何代理预订从预订传播到集群到本地主题。

总之，在两种情况下会为订户创建代理预订。

1. 队列管理器是层次结构的成员，代理预订将转发给队列管理器的父代和子代。
2. 队列管理器是集群的成员，预订主题字符串解析为与集群主题对象关联的主题。 当主题是 直接路由 集群主题时，会将代理预订转发到集群的所有成员。 当主题是 主题主机路由 集群主题时，仅会将代理预订转发到已定义集群主题对象的集群中的队列管理器。 有关更多信息，请参阅第 75 页的『发布/预订集群』。

如果队列管理器是集群和层次结构的成员，那么代理预订由两种机制传播，而不会将重复的发布传递给订户。

以下列表中描述了三个发布/预订拓扑的主题空间:

- 第 85 页的『案例 1。 发布/预订集群』.
- 第 86 页的『案例 2。 发布/预订层次结构』.

在单独的主题中，以下配置任务描述了如何组合主题空间。

- 在发布/预订集群中创建单个主题空间。
- 组合多个集群的主题空间。
- 组合和隔离多个集群中的主题空间。
- 发布和预订多个集群中的主题空间。

案例 1。 发布/预订集群

在此示例中，假定队列管理器未连接到发布/预订层次结构。

如果队列管理器是发布/预订集群的成员，那么其主题空间由本地主题和集群主题组成。 本地主题与没有 **CLUSTER** 属性的主题对象相关联。 如果队列管理器具有本地主题对象定义，那么其主题空间与集群中另一个也具有其自己的本地定义主题对象的队列管理器不同。

在发布/预订集群中，无法预订另一个队列管理器上定义的主题，除非您预订的主题解析为集群主题对象。

在多个队列管理器上需要集群主题对象的相同指定定义时 (例如，使用 主题主机路由时) ，所有定义在必要时匹配很重要。 有关更多信息，请参阅 在发布/预订集群中创建单个主题空间。

主题对象的本地定义 (无论该定义是针对集群主题还是本地主题) 优先于集群中其他位置定义的相同主题对象。 使用本地定义的主题，即使在其他位置定义的对象是最新的。

集群主题对象与集群中所有位置的相同主题字符串相关联很重要。 不能修改与主题对象关联的主题字符串。 要将同一主题对象与另一主题字符串相关联，必须删除该主题对象，并使用新的主题字符串重新创建该主题

对象。 如果主题是集群的，那么效果是删除存储在集群的其他成员上的主题对象的副本，然后在集群中的所有位置创建新主题对象的副本。 主题对象的副本都引用同一个主题字符串。

可能会意外地在集群中的不同队列管理器上创建两个具有不同主题字符串的相同命名主题对象的定义。 这可能会导致混淆行为，因为具有不同主题字符串的同一主题对象的多个定义可能会产生不同的结果，具体取决于引用主题的方式和位置。 请参阅 同名的多个集群主题定义，以获取有关此重要点的更多信息。

案例 2。 发布/预订层次结构

在此示例中，假定队列管理器不是发布/预订集群的成员。

在 IBM MQ 中，如果队列管理器是发布/预订层次结构的成员，那么其主题空间由本地定义的所有主题以及在已连接的队列管理器上定义的所有主题组成。 层次结构中所有队列管理器的主题空间都相同。 没有将主题划分为局部主题和全局主题。

将 **PUBSCOPE** 和 **SUBSCOPE** 选项中的任一选项设置为 QMGR，以防止主题上的发布从发布者流向连接到层次结构中不同队列管理器的订户。

假设您在队列管理器 QMA 上使用主题字符串 USA/Alabama 定义主题对象 Alabama 。 结果如下:

1. QMA 上的主题空间现在包含主题对象 Alabama 和主题字符串 USA/Alabama。
2. 应用程序或管理员可以使用主题对象名称 Alabama 在 QMA 上创建预订。
3. 应用程序可以在层次结构中的任何队列管理器上创建对任何主题 (包括 USA/Alabama) 的预订。 如果未在本地定义 QMA，那么主题 USA/Alabama 将解析为主题对象 SYSTEM.BASE.TOPIC。

相关概念

第 83 页的『发布作用域』
配置发布/预订集群或层次结构时，发布的作用域将进一步控制队列管理器是否将发布转发到远程队列管理器。 使用 **PUBSCOPE** 主题属性来管理发布的作用域。

第 84 页的『预订作用域』
预订的作用域控制一个队列管理器上的预订是接收在发布/预订集群或层次结构中的另一个队列管理器上发布的发布，还是仅接收来自本地发布程序的发布。

相关任务

配置分布式发布/预订网络

IBM MQ 多点广播

IBM MQ 多点广播提供具有低等待时间和高扇出的可靠多点广播消息传递。

多点广播是一种高效的发布/预订消息传递形式，因为它可以扩展为大量订户，而不会对性能产生不利影响。 IBM MQ 使用应答、否定应答和序号来支持可靠的多点广播消息传递，以实现具有高扇出的低等待时间消息传递。

IBM MQ 多点广播的公平传递支持几乎同时的传递，从而确保不偏向任何接收方。 IBM MQ 多点广播使用网络来传递消息，因此无需发布/预订引擎对数据进行扇出。 将主题映射到组地址后，不需要队列管理器，因为发布程序和订户可以在对等方式下运行。 这就允许减少队列管理器服务器上的负载，而队列管理器服务器将不再是潜在的故障点。

初始多点广播概念

IBM MQ 通过使用 "通信信息" (COMMINFO) 对象，可以轻松地将多点广播集成到现有系统和应用程序中。 两个 TOPIC 对象字段支持快速配置现有 TOPIC 对象以支持或忽略多点广播流量。

多点广播所需的对象

以下信息是 IBM MQ 多点广播所需的两个对象的简要概述:

COMMINFO 对象
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。

必须设置的唯一 COMMINFO 字段是 COMMINFO 对象的名称。 然后，此名称用于向主题标识 COMMINFO 对象。 必须检查 COMMINFO 对象的 **GRPADDR** 字段，以确保该值是有效的多点广播组地址。

主题对象
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。 有关 TOPIC 对象参数的更多信息，请参阅 DEFINE TOPIC。

通过更改以下 TOPIC 对象参数的值，可以将现有主题用于多点广播: **COMMINFO** 和 **MCAST**。

- **COMMINFO** 此参数指定多点广播通信信息对象的名称。
- **MCAST** 此参数指定在主题树中的此位置是否允许多点广播。 缺省情况下， **MCAST** 设置为 ASPARENT 表示从父代继承主题的多点广播属性。 将 **MCAST** 设置为 ENABLED 允许此节点上的多点广播流量。

多点广播网络和主题

以下信息概述了对具有不同类型的预订和主题定义的预订执行的操作。 这些示例都假定 TOPIC 对象 **COMMINFO** 参数设置为有效 COMMINFO 对象的名称:

主题集已启用多点广播
如果主题字符串 **MCAST** 参数设置为 ENABLED，那么允许来自支持多点广播的客户机的预订并进行多点广播预订，除非:

- 它是来自支持多点广播的客户机的持久预订。
- 它是来自支持多点广播的客户机的非受管预订。
- 它是来自不支持多点广播的客户机的预订。

在这些情况下，将进行非多点广播预订，并将 预订降级为正常发布/预订。

已禁用设置为多点广播的主题
如果主题字符串 **MCAST** 参数设置为 DISABLED，那么将始终进行非多点广播预订，并且 预订降级为正常发布/预订。

主题设置为仅多点广播
如果主题字符串 **MCAST** 参数设置为 ONLY，那么将允许来自支持多点广播的客户机的预订，并且将进行多点广播预订，除非:

- 这是持久预订: 已拒绝持久预订，原因码为 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- 它是非受管预订: 非受管预订被拒绝，原因码为 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- 它是来自不支持多点广播的客户机的预订: 这些预订被拒绝，原因码为 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- 这是来自本地绑定应用程序的预订: 将拒绝这些预订，原因码为 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

Windows Linux AIX MQ Telemetry 概述

MQ Telemetry 包含作为队列管理器一部分的遥测 (MQXR) 服务，可自行编写或免费下载的遥测客户机以及命令行和资源管理器管理界面。 遥测指收集远程设备数据并管理各种远程设备。 通过 MQ Telemetry ， 您可以将数据收集和设备控制与 Web 应用程序集成。

MQ Telemetry 是 IBM MQ 的组件。 这些版本的升级基本上是安装更高版本的 IBM MQ。

可从 Eclipse Paho 和 MQTT.org 免费获取样本应用程序。 请参阅 IBM MQ Telemetry Transport 样本程序。

由于 MQ Telemetry 是 IBM MQ 的组件，因此可以随主产品一起安装 MQ Telemetry ， 也可以在安装主产品之后安装。 有关迁移信息，请参阅 在 Windows 上迁移 MQ Telemetry 和 在 Linux 上迁移 MQ Telemetry。

MQ Telemetry 中包含以下组件:

遥测通道
使用遥测通道来管理 MQTT 客户机与 IBM MQ 的连接。 遥测通道使用新的 IBM MQ 对象 (例如 SYSTEM.MQTT.TRANSMIT.QUEUE) 与 IBM MQ 进行交互。

遥测 (MQXR) 服务
MQTT 客户机使用 SYSTEM.MQXR.SERVICE 遥测服务来连接到遥测通道。

IBM MQ Explorer 支持 MQ Telemetry
可以使用 IBM MQ Explorer 来管理 MQ Telemetry 。

文档
MQ Telemetry 文档包含在标准 IBM MQ 产品文档中。 Java 和 C 客户机的 SDK 文档在产品文档中以 Javadoc 和 HTML 形式提供。

遥测概念

您从周围的环境收集信息以决定要执行的操作。 作为消费者，你先检查你在店里有什么，然后再决定买什么食物。 您想知道如果现在离开，在预订连接之前，一段旅程将需要多长时间。 你检查一下你的症状，然后再决定是否去看医生。 在决定是否等待之前，请检查公交车何时到达。 这些决策的信息直接来自计量表和设备，来自纸上的文字或屏幕，以及来自您的信息。 在任何时候，您都需要收集信息，汇集信息，对其进行分析，并对其采取行动。

如果信息来源广泛分散或无法获取，收集最准确的信息就会变得困难和昂贵。 如果您要进行的更改很多，或者很难进行更改，那么这些更改不会进行，或者在效果较差时进行。

如果通过将具有数字技术的设备连接到互联网来大大降低从广泛分散的设备收集信息和控制这些设备的成本，将会怎样? 可以利用互联网和企业的资源对信息进行分析。 您有更多机会做出明智的决策并采取行动。

技术趋势以及环境和经济压力正在推动这些变化发生:

1. 由于标准化和与低成本数字处理器的连接，连接和控制传感器和执行器的成本正在降低。
2. 互联网和互联网技术越来越多地用于连接设备。 在一些国家，在与互联网应用程序的连接数量上，移动电话超过了个人计算机。 其他设备肯定在关注。
3. 互联网和互联网技术使应用程序更容易获取数据。 轻松访问数据正在推动使用数据分析，将来自传感器的数据转化为更多解决方案中有用的信息。
4. 智能使用资源往往是减少碳排放和成本的更快和更便宜的方法。 替代办法: 寻找新的资源，或开发新的技术来利用现有资源，可能是长期的解决办法。 从短期来看，开发新技术或寻找新资源，通常比改进现有解决方案风险更高，速度更慢，成本更高。

示例

一个示例显示了这些趋势如何创造与环境智能交互的新机会。

The International Convention for the Safety of Life at Sea (SOLAS) requires Automatic Identification System (AIS) to be deployed on many ships. 300 吨以上的商船和客船都需要。 AIS 主要是沿海航运的避撞系统。 它被海洋当局用来监测和控制沿海水域。

世界各地的爱好者正在部署低成本的 AIS 跟踪站，并将沿海航运信息放到互联网上。 其他爱好者正在编写将来自 AIS 的信息与来自互联网的其他信息相结合的应用程序。 结果将放在 Web 站点上，并使用 Twitter 和 SMS 进行发布。

在一个应用程序中，来自南安普敦附近 AIS 站的信息与船舶所有权和地理信息相结合。 该应用程序将有关轮渡到达和离开的实时信息提供给 Twitter。 使用南安普敦和怀特岛之间的渡轮的普通通勤者使用推特或短信订阅新闻订阅。 如果饲料显示他们的轮渡运行较晚，通勤者可能会延迟他们的出发，并在其停靠时间晚于其预定到达时间时捕获轮渡。

要获取更多示例，请参阅第 90 页的『遥测用例』。

相关任务
安装 MQ Telemetry
管理 MQ Telemetry
在 Windows 上迁移 MQ Telemetry
在 Linux 上迁移 MQ Telemetry
为 MQ Telemetry 开发应用程序
对 MQ Telemetry 问题进行故障诊断

相关参考

MQ Telemetry 参考

Windows Linux AIX

MQ Telemetry 简介

人们，企业和政府越来越希望使用 MQ Telemetry 与我们生活和工作的环境进行更智能的交互。 MQ Telemetry 将各种设备连接到因特网和企业，并降低为智能设备构建应用程序的成本。

什么是 MQ Telemetry?

- 它是 IBM MQ 的一个功能部件，用于将 IBM MQ 提供的通用消息传递主干扩展至各种远程传感器，执行器和遥测设备。 MQ Telemetry 扩展 IBM MQ ，使其能够将智能企业应用程序，服务和决策制定者与受检测设备网络互连。
- MQ Telemetry 的核心部件为:

  MQ Telemetry (MQXR) 服务。
  此服务在 IBM MQ 服务器中运行，并使用 IBM MQ Telemetry Transport (MQTT) 协议与遥测设备进行通信。

  您编写的 MQTT 应用程序。
  这些应用程序控制遥测设备与 IBM MQ 队列管理器之间传递的信息以及为响应该信息而执行的任何操作。 要帮助创建这些应用程序，请使用 MQTT 客户机库。

它具有什么功能?

- MQTT 是一种开放式消息传递传输，允许为各种设备创建 MQTT 实现。
- MQTT 客户机可以在资源有限的小型占用量设备上运行。
- MQTT 在带宽较低，发送数据的成本较高或可能很脆弱的网络上高效工作。
- 保证消息传递并使其与应用程序分离。
- 应用程序员不需要具备通信编程知识。
- 可以与其他消息传递应用程序交换消息。 这些应用程序可以是另一个遥测应用程序，也可以是 MQI，JMS 或企业消息传递应用程序。

如何使用?

- 提供了用于处理样本 IBM MQ Telemetry Transport v3 客户机应用程序 (`mqttv3app.jar`) 的样本脚本。请参阅 IBM MQ Telemetry Transport 样本程序。
- 使用 IBM MQ Explorer 及其关联工具来管理 IBM MQ 的遥测功能。
- 使用客户机库可帮助您创建连接到队列管理器并使用发布/预订消息传递的 MQTT 应用程序。
- 将应用程序和客户机库分发到要运行应用程序的设备。

它是如何工作的?

- MQTT 是发布预订协议。 MQTT 客户机应用程序可以将消息发布到 MQTT 服务器，或者预订由连接到 MQTT 服务器的应用程序发送的消息。
- MQTT 客户机应用程序使用实现 MQTT 消息传输的客户机库。
- 基本 MQTT 客户机应用程序类似于标准 MQ 客户机，但可以在更广泛的各种平台和网络上运行。
- MQ Telemetry (MQXR) 服务将 IBM MQ 队列管理器转换为 MQTT 服务器。
- 当 IBM MQ 队列管理器充当 MQTT 服务器时，连接到队列管理器的其他应用程序可以从 MQTT 客户机预订和接收消息。
- 队列管理器充当路由器将消息从发布应用程序分发到预订应用程序。
- 可以在不同类型的客户机应用程序之间分发消息。 例如，在 Telemetry 客户机与 JMS 客户机之间。

注: MQ Telemetry 替换在 WebSphere Message Broker V 7 中撤销的 SCADA 节点 (现在称为 IBM Integration Bus) 并在 Windows， Linux 和 AIX 上运行。

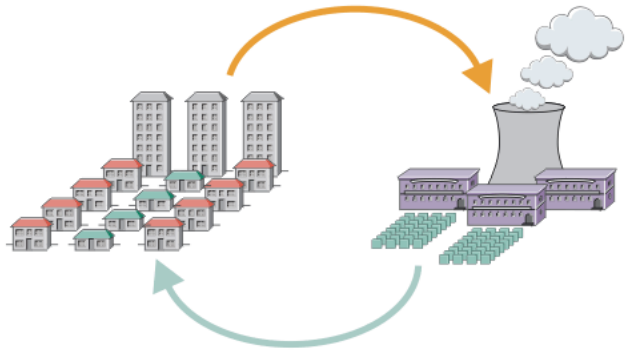
Windows Linux AIX 遥测用例

遥测是对数据的自动感应和度量以及对远程设备的自动控制。 重点在于数据从设备到中央控制点的传输。 遥测还包含向设备发送配置和控制信息。

MQ Telemetry 使用 MQTT protocol 连接小型设备，并使用 IBM MQ 将这些设备连接到其他应用程序。 MQ Telemetry 弥合了设备与互联网之间的差距，使其更容易构建 "智能解决方案"。 智能解决方案为监控和控制设备的应用程序解锁了互联网上和企业应用程序中可用的丰富信息。

下图演示了 MQ Telemetry 的一些典型用法:

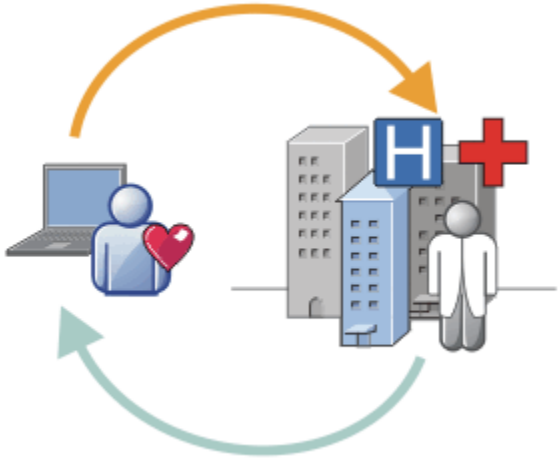
遥测: 智能电力



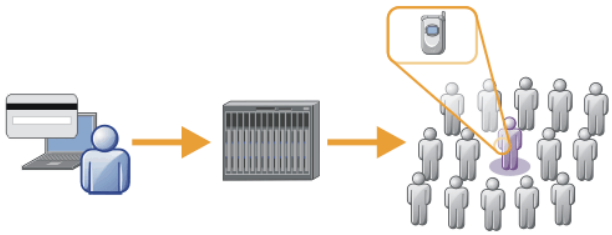
- 包含发送到服务提供者的能源使用数据的 MQTT 消息。
- MQ Telemetry 根据对能源使用情况数据的分析发送 CONTROL COMMANDS。
- 有关更多信息，请参阅以下用例: 第 92 页的『遥测用例: 家庭能源监测与控制』

遥测: 智慧健康服务

- MQ Telemetry 将健康数据发送到您的医院和医生。
- 可以根据运行状况数据分析来发送 MQTT 消息警报或反馈。
- 有关更多信息，请参阅以下用例: 第 91 页的『遥测用例: 家庭患者监控』



遥测: Crowd 中的一个



- 将简单的卡交易发送到银行的服务器。
- MQ Telemetry 识别千人中的一人，提醒客户已使用他们的卡。
- MQ Telemetry 可以使用最简单的信息输入，并找到该个人。

子主题中描述的用例来自实际示例。 它们说明了一些使用遥测的方法，以及遥测技术必须解决的一些常见问题。

Windows Linux AIX 遥测用例: 家庭患者监控

在 IBM 与心脏患者护理系统上的医疗保健提供者之间的协作中，植入的心内膜除颤器与医院进行通信。 使用 RF 遥测将有关患者和植入设备的数据传输到患者家中的 MQTT 设备。

通常，传输会在夜间进行到位于床头的发射器。 发射器通过电话系统将数据安全地传输到医院，在医院中分析数据。

该系统减少了患者必须看医生的次数。 它检测患者或设备何时需要关注，在发生紧急情况时，它会向待命医生发出警报。

IBM 与医疗保健提供者之间的协作具有许多遥测用例所共有的特征:

Invisibility

该设备无需用户干预，只需提供电源，电话线，并在一天中的部分时间与该设备接近。 其操作可靠，使用简单。

为了消除患者设置设备的需求，设备供应商对设备进行预配置。 患者必须将其插入。 患者消除了配置，简化了设备的操作，减少了设备配置错误的机会。

MQTT 客户机作为设备的一部分嵌入。 设备开发者将 MQTT 客户机实现嵌入到设备中，开发者或供应商将 MQTT 客户机配置为预配置的一部分。

MQTT 客户机作为 Java SE JAR 文件交付，开发者在其 Java 应用程序中包含该 JAR 文件。 对于非 Java 环境 (例如，此环境) ，设备开发者可以使用已发布的 MQTT 格式和协议以不同语言实现客户机。 或者，开发者可以使用其中一个作为 Windows， Linux 和 ARM 平台的共享库交付的 C 客户机。

不均衡的连接

除颤器与医院之间的通信具有不均匀的网络特性。 用两个不同的网络来解决从病人那里收集数据，把数据送到医院的不同问题。 在专利与 MQTT 设备之间，使用短程低功耗 RF 网络。 发送器使用 VPN TCP/IP 连接通过低带宽电话线连接到医院。

通常无法找到将每个设备直接连接到 Internet Protocol 网络的方法。 使用两个网络 (通过中心连接) 是常见的解决方案。 MQTT 设备是一个简单的集线器，用于存储来自患者的信息，并将其转发到医院。

安全性

医生必须能够信任患者数据的真实性，患者希望其数据的隐私得到尊重。

在某些情况下，它足以使用 VPN 或 TLS 对连接进行加密。 在其他情况下，即使已存储数据，也应确保数据安全。

有时遥测设备不安全。 例如，它可能位于共享住宅中。 必须对设备的用户进行认证，以确保数据来自正确的患者。 可以使用 TLS 向服务器认证设备本身，并向设备认证服务器。

设备与队列管理器之间的遥测通道支持 JAAS 进行用户认证，支持 TLS 进行通信加密和设备认证。 对发布的访问由 IBM MQ 中的对象权限管理器控制。

用于认证用户的标识可以映射到其他标识，例如公共患者标识。 公共标识可简化对 IBM MQ 中发布主题的授权配置。

连接

MQTT 设备与医院之间的连接使用拨号，并使用低至 300 波特的带宽。

为了在 300 波特处有效运行，除了 TCP/IP 头外， MQTT protocol 仅向消息添加几个额外的字节。

MQTT protocol 提供单次传输 触发并忘记 消息传递，这使等待时间保持较低。 它还可以使用多个传输来保证 至少一次 和 正好一次 传递 (如果保证传递比响应时间更重要)。 为了保证传递，消息将存储在设备上，直到成功传递为止。 如果设备以无线方式连接，那么保证交付特别有用。

扩充性

遥测设备通常大量部署，从数万到数百万。

将许多设备连接到系统会对解决方案产生很大的需求。 存在业务需求，例如设备及其软件的成本，以及管理许可证，设备和用户的管理需求。 技术需求包括网络和服务器上的负载。

与维护打开的连接相比，打开连接使用的服务器资源更多。 但是，在使用电话线之类的用例中，连接费用意味着连接处于打开状态的时间不会超过所需时间。 数据传输在很大程度上具有批处理性质。 可以安排通宵达旦的连接，避免在睡前出现突然的连接高峰。

在客户机上，客户机的可伸缩性由所需的最低客户机配置提供帮助。 MQTT 客户机嵌入在设备中。 不需要在向患者部署设备时内置配置或 MQTT 客户机许可证接受步骤。

在服务器上， MQ Telemetry 的初始目标是每个队列管理器打开 50,000 个连接。

使用 IBM MQ Explorer 来管理连接。 IBM MQ Explorer 会将要显示的连接过滤为可管理的数字。 通过适当选择的将标识分配给客户机的方案，您可以根据地理位置或按患者名称的字母顺序来过滤连接。

Windows Linux AIX 遥测用例: 家庭能源监测与控制

与传统计量表相比，智能计量表收集更多有关能耗的详细信息。

智能电表通常与本地遥测网络耦合在一起，以监控家庭中的各个设备。 还可以远程连接某些智能电表，以在远处进行监控。

远程连接可由个人，电源实用程序或中央控制点设置。 远程控制点可以读取电源使用情况并提供使用情况数据。 它可以提供数据来影响使用情况，例如持续定价和天气信息。 可限制负荷以提高整体发电效率。

智能计量表已开始广泛部署。 例如，英国政府正在就到 2020 年将智能电表部署到每个英国家庭进行协商。

家庭计量用例有以下几个共同特点:

Invisibility

除非用户希望通过使用计量表来参与节能，否则计量表不得要求用户干预。 它不得降低个别电器的能源供应的可靠性。

MQTT 客户机可以嵌入到随计量表一起部署的软件中，并且不需要单独的安装或配置。

不均衡的连接

设备与智能计量表之间的通信要求的连接标准与计量表与远程连接点之间的连接标准不同。

从智能计量表到设备的连接必须高度可用，并且符合家庭区域网络的网络标准。

远程网络可能使用各种物理连接。 其中一些，如细胞，其传播成本很高，可以是间歇性的。 MQTT v3 规范针对远程连接以及本地适配器与智能计量表之间的连接。

电源插座和应用程序与计量表之间的连接使用家庭区域网络，例如 Zigbee。 MQTT 用于传感器网络 (MQTT-S) ，旨在使用 Zigbee 和其他低带宽网络协议。 MQ Telemetry 不直接支持 MQTT-S。 它需要网关将 MQTT-S 连接到 MQTT v3。

与家庭病人监测一样，家庭能源监测和控制的解决方案需要多个网络，使用智能仪表作为枢纽进行连接。

安全性

存在许多与智能计量表相关联的安全问题。 这些问题包括交易的不可抵赖性，启动的任何控制操作的授权以及功耗数据的隐私。

为确保隐私，可以使用 TLS 对 MQTT 在计量表和远程控制点之间传输的数据进行加密。 为确保控制操作的授权，可以使用 TLS 对计量表与远程控制点之间的 MQTT 连接进行相互认证。

连接

远程网络的物理性质可能有很大差异。 它可能使用现有宽带连接，或者使用具有高呼叫成本和间歇性可用性的移动网络。 对于高成本，间歇性的连接， MQTT 是一种高效且可靠的协议; 请参阅 第 91 页的『遥测用例: 家庭患者监控』。

扩充性

最终电力公司或中央控制点，计划部署数千万台智能电表。 最初，每个部署的计量表数在数万到数十万。 此数目与每个队列管理器 50,000 个开放式客户机连接的初始 MQTT 目标相当。

家庭能源监测和控制的架构的一个关键方面是使用智能仪表作为网络集中器。 每个设备适配器都是一个单独的传感器。 通过使用 MQTT 将它们连接到本地集线器，集线器可以将数据流集中到具有中央控制点的单个 TCP/IP 会话上，还可以在短时间内存储消息以克服会话中断。

在家庭能源用例中，远程连接必须保持打开状态，原因有二。 首先，由于打开连接相对于发送请求需要很长时间。 在较短的时间间隔内打开多个连接以发送 “装入限制” 请求的时间太长。 其次，要接收来自电力公司的负载限制请求，必须首先由客户打开连接。 使用 MQTT 时，连接始终由客户机启动，要接收来自电力公司的负载限制请求，连接必须保持打开状态。

如果打开连接的速率是临界值，或者服务器启动时间临界请求，那么解决方案通常是维护许多打开的连接。

Windows Linux AIX 遥测用例: 射频识别 (RFID)

RFID 是使用嵌入式 RFID 标签来无线识别和跟踪物体。 RFID 标签可以被读取到几米的范围内，并且脱离 RFID 阅读器的视线。 无源标签由 RFID 阅读器激活。 在没有外部激活的情况下传输活动标记。 活动标记必须具有电源。 无源标签可以包含电源以增加其范围。

RFID 在很多应用中都有使用，用例的类型也差别很大。 RFID 用例，以及家庭患者监测和家庭能源监测与控制用例，都有一些相似之处和不同之处。

Invisibility

在许多用例中， RFID 阅读器是大量部署的，必须在没有用户干预的情况下工作。 阅读器包含用于与中央控制点通信的嵌入式 MQTT 客户机。

例如，在配送仓库中，阅读器使用运动传感器来检测托盘。 它激活托盘上项目的 RFID 标签，并将数据和请求发送到中央应用程序。 数据用于更新库存位置。 这些请求可控制下一个托盘发生的情况，例如将其移动到特定托架。 航空公司和机场行李系统都在以这种方式使用 RFID。

在某些 RFID 用例中，阅读器具有标准计算环境，例如 Java Platform, Micro Edition (Java ME)。 在这些情况下， MQTT 客户机可能在制造后部署在不同的配置步骤中。

不均衡的连接

RFID 阅读器可能与包含 MQTT 客户机的本地控制设备分离，或者每个阅读器都可能嵌入 MQTT 客户机。 通常，地理或通信因素指示拓扑的选择。

安全性

隐私和真实性是 RFID 标签附件中的安全问题。 RFID 标签具有不可侵扰性，可被秘密监控，欺骗或篡改。

RFID 安全问题的解决方案增加了部署新的 RFID 解决方案的机会。 尽管安全漏洞在 RFID 标签和本地阅读器中，但使用中央信息处理建议了应对不同威胁的方法。 例如，可以通过将库存级别与交付和分派动态关联来检测标记篡改。

连接

RFID 应用通常涉及批量存储和从 RFID 阅读器收集的信息以及即时查询。 在配送仓库用例中， RFID 阅读器一直连接。 读取标记时，会将其与有关读者的信息一起发布。 仓储应用程序将响应发布回阅读器。

在仓储应用程序中，网络通常是可靠的，即时请求可能会使用 触发和忘记 消息以实现低延迟性能。 批处理存储和转发数据可能使用 精确一次 消息传递，以最大限度降低与松散数据相关联的管理成本。

扩充性

如果 RFID 应用程序需要立即响应 (按秒或秒的顺序) ，那么 RFID 阅读器必须保持连接。

Windows Linux AIX 遥测用例: 环境传感

环境传感利用遥测收集有关河流水位和质量，大气污染物以及其他环境数据的信息。

传感器经常位于偏远的地方，无法进行有线通信。 无线带宽昂贵，可靠性低。 通常，在一个小的地理区域内的一些环境传感器被连接到一个安全位置的本地监控设备。 本地连接可以是有线连接或无线连接。

Invisibility

与中央监控设备相比，这些传感器设备的可访问性可能较低，供电能力较低，并且部署的数量更多。 传感器有时是 "哑" 的，本地监控设备包括用于转换和存储传感器数据的适配器。 监视设备可能包含支持 Java Platform, Standard Edition (Java SE) 或 Java Platform, Micro Edition (Java ME) 的通用计算机。 配置 MQTT 客户机时，不太可能主要需要隐蔽性。

不均衡的连接

传感器的功能以及远程连接的成本和带宽通常会导致本地监控中心连接到中央服务器。

安全性

除非在军事或防御用例中使用该解决方案，否则安全不是主要要求。

连接

许多用途不需要持续监控或即时提供数据。 需要立即转发异常数据 (例如，洪水级别警报)。 在本地监视器上聚集传感器数据以降低连接和通信成本，然后使用预定连接进行传输。 一旦在监视器上检测到异常数据，就会将其转发。

扩充性

传感器集中在本地集线器周围，传感器数据聚集到根据调度传输的包中。 这两个因素都降低了中央服务器上通过使用直接连接的传感器施加的负载。

Windows Linux AIX 遥测用例: 移动应用程序

移动应用程序是在无线设备上运行的应用程序。 这些设备是通用应用程序平台或定制设备。

一般平台包括手机和个人数据助手等手持设备，笔记本电脑等便携式设备。 定制设备使用针对特定应用程序定制的特殊用途硬件。 用于记录 "signed-for" 包裹递送的设备是定制移动设备的示例。 定制移动设备上的应用程序通常是在通用软件平台上构建的。

Invisibility

定制移动应用程序的部署是受管的，并且可以包含 MQTT 客户机应用程序的配置。 配置 MQTT 客户机时，不太可能主要需要隐蔽性。

不均衡的连接

与先前用例的本地中心拓扑不同，移动式客户机远程连接。 客户机应用程序层直接连接到中央集线器上的应用程序。

安全性

在物理安全性很小的情况下，必须对移动设备和移动用户进行认证。 TLS 用于确认设备的身份， JAAS 用于认证用户。

连接

如果移动应用程序依赖于无线覆盖，那么它必须能够脱机操作，并且能够高效地处理中断的连接。 在此环境中，目标是保持连接，但应用程序必须能够存储和转发消息。 通常，消息是订单或交货确认，并且具有重要的业务价值。 需要对它们进行可靠的存储和转发。

扩充性

可扩展性不是一个主要问题。 在定制移动应用程序用例中，应用程序客户机的数量可能不会超过数千或数万个。

Windows Linux AIX 将遥测设备连接至队列管理器

遥测设备使用 MQTT v3 客户机连接到队列管理器。 MQTT v3 客户机使用 TCP/IP 连接到称为遥测 (MQXR) 服务的 TCP/IP 侦听器。

将遥测设备连接到队列管理器时， MQTT 客户机将使用 `MqttClient.connect` 方法启动 TCP/IP 连接。 与 IBM MQ 客户机一样， MQTT 客户机必须连接到队列管理器以发送和接收消息。 使用随 MQ Telemetry 一起

安装的 TCP/IP 侦听器 (称为遥测 (MQXR) 服务) 在服务器上建立连接。 每个队列管理器最多运行一个遥测 (MQXR) 服务。

遥测 (MQXR) 服务使用每个客户机在 `MqttClient.connect` 方法中设置的远程套接字地址来分配与遥测通道的连接。 套接字地址是 TCP/IP 主机名和端口号的组合。 使用同一远程套接字地址的多个客户机通过遥测 (MQXR) 服务连接到同一遥测通道。

如果服务器上有多个队列管理器，请在队列管理器之间拆分遥测通道。 在队列管理器之间分配远程套接字地址。 使用唯一的远程套接字地址定义每个遥测通道。 两个遥测通道不得使用相同的套接字地址。

如果为多个队列管理器上的遥测通道配置了相同的远程套接字地址，那么要连接的第一个遥测通道将获胜。 在同一地址上连接的后续通道失败。

如果服务器上有多个网络适配器，请在遥测通道之间拆分远程套接字地址。 只要仅在一个遥测通道上配置任何特定套接字地址，那么套接字地址的分配完全是任意的。

使用 IBM MQ Explorer 的 MQ Telemetry 补充说明中提供的向导配置 IBM MQ 以连接 MQTT 客户机。 或者，遵循 在 Linux 和 AIX 上配置队列管理器以进行遥测 和 在 Windows 上配置队列管理器以进行遥测手动配置 中的指示信息。

相关参考

MQXR 属性

Windows Linux AIX 遥测连接协议

MQ Telemetry 支持 TCP/IP IPv4 和 IPv6 以及 TLS。

Windows Linux AIX 遥测 (MQXR) 服务

遥测 (MQXR) 服务是作为 IBM MQ 服务管理的 TCP/IP 侦听器。 使用 IBM MQ Explorer 向导或 **runmqsc** 命令创建服务。

MQ Telemetry (MQXR) 服务称为 SYSTEM.MQXR.SERVICE 。

IBM MQ Explorer 的 MQ Telemetry 函数中提供的 Telemetry 样本配置向导将创建遥测服务和样本遥测通道; 请参阅 使用 IBM MQ Explorer 验证 MQ Telemetry 的安装 。

从命令行创建样本配置; 请参阅 使用命令行验证 MQ Telemetry 的安装。

遥测 (MQXR) 服务会随队列管理器自动启动和停止。 使用 IBM MQ Explorer 中的 services 文件夹来控制服务。 要查看服务，必须单击该图标以停止 IBM MQ Explorer 从显示中过滤掉 SYSTEM 对象。

有关如何手动创建服务的示例，请参阅

- Linux AIX 在 Linux 上创建 SYSTEM.MQXR.SERVICE.
- Windows 在 Windows 上创建 SYSTEM.MQXR.SERVICE.

这些主题还指定了要求对 MQTT TLS 通道的口令进行加密的缺省密钥。 有关更多信息，请参阅 加密 MQTT TLS 通道的口令。

Windows Linux AIX 遥测通道

创建遥测通道以创建具有不同属性 (例如 Java 认证和授权服务 (JAAS) 或 TLS 认证) 的连接，或者管理客户机组。

使用 IBM MQ Explorer 的 MQ Telemetry 函数中提供的 **New Telemetry Channel** 向导创建遥测通道。 使用向导配置通道以接受来自特定 TCP/IP 端口上的 MQTT 客户机的连接。 从 IBM WebSphere MQ 7.1 开始，您可以使用命令行程序 **runmqsc** 来配置 MQ Telemetry 。

在不同端口上创建多个遥测通道，通过将客户机拆分为多个组，使大量客户机连接更易于管理。 每个遥测通道都有不同的名称。

您可以配置具有不同安全性属性的遥测通道，以创建不同类型的连接。 创建多个通道以接受不同 TCP/IP 地址上的客户机连接。 使用 TLS 对消息进行加密并认证遥测通道和客户机; 请参阅 MQTT 客户机和遥测通道的

TLS 配置。 指定用户标识以简化对 IBM MQ 对象的授权访问。 指定 JAAS 配置以使用 JAAS 认证 MQTT 用户; 请参阅 MQTT 客户机标识，授权和认证。

Windows Linux AIX IBM MQ Telemetry Transport 协议

IBM MQ Telemetry Transport (MQTT) v3 协议旨在以低带宽或昂贵的连接在小型设备之间交换消息，并可靠地发送消息。 它使用 TCP/IP。

MQTT protocol 已发布; 请参阅 IBM MQ Telemetry Transport 格式和协议。 协议版本 3 使用发布/预订，并支持三种服务质量: 触发和忘记， 至少一次和 正好一次。

协议头和字节数组消息有效内容的较小大小使消息保持较小。 这些头包含一个 2 字节的固定头，以及最多 12 个字节的其他变量头。 该协议使用 12 字节变量头来预订和连接，并且仅 2 字节变量头用于大多数发布。

通过三种服务质量，您可以在低延迟和可靠性之间进行权宜之法; 请参阅 MQTT 客户机提供的服务的程度。 触发和忘记 不使用持久设备存储器，仅使用一个传输来发送或接收发布。 至少一次，并且 正好一次 需要设备上的持久存储器以保持协议状态并保存消息，直到确认为止。

Windows Linux AIX MQTT Client

MQTT 客户机应用程序负责从遥测设备收集信息，连接到服务器以及将信息发布到服务器。 它还可以预订主题，接收出版物以及控制遥测设备。

与 IBM MQ 客户机应用程序不同， MQTT 客户机应用程序不是 IBM MQ 应用程序。 它们不指定要连接到的队列管理器。 它们不限于使用特定的 IBM MQ 编程接口。 相反， MQTT 客户机实现 MQTT 3 协议。 您可以编写自己的客户机库，以便在您选择的编程语言和平台上与 MQTT protocol 进行交互。 请参阅 IBM MQ Telemetry Transport 格式和协议。

要简化 MQTT 客户机应用程序的编写，请使用封装多个平台的 MQTT protocol 的 C， Java 和 JavaScript 客户机库。 如果将这些库合并到 MQTT 应用程序中，那么功能齐全的 MQTT 客户机可以短于 15 行代码。 MQTT 客户机库可从 Eclipse Paho 和 MQTT.org 免费获取。 请参阅 IBM MQ Telemetry Transport 样本程序。

MQTT 客户机应用程序始终负责启动与遥测通道的连接。 连接后， MQTT 客户机应用程序或 IBM MQ 应用程序可以启动消息交换。

MQTT 客户机应用程序和 IBM MQ 应用程序发布并预订同一组主题。 IBM MQ 应用程序还可以直接向 MQTT 客户机应用程序发送消息，而无需客户机应用程序首先创建预订。 请参阅 配置分布式排队以将消息发送到 MQTT 客户机。

MQTT 客户机应用程序使用遥测通道连接到 IBM MQ 。 遥测通道充当 MQTT 和 IBM MQ 所使用的不同类型消息之间的网桥。 它代表 MQTT 客户机应用程序在队列管理器中创建发布和预订。 遥测通道将与 MQTT 客户机应用程序预订匹配的发布从队列管理器发送到 MQTT 客户机应用程序。

Windows Linux AIX 向 MQTT 客户机发送消息

IBM 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。 MQTT 客户机可以通过发布到其他客户机预订的主题来相互发送消息。

MQTT 客户机预订从 IBM MQ 接收的发布内容

执行任务 第 98 页的『从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序』 以将发布从 IBM MQ 发送到 MQTT 客户机。

MQTT v3 客户机接收消息的标准方法是它创建对主题或一组主题的预订。 在示例代码片段 第 97 页的图 44 中， MQTT 客户机使用主题字符串 “MQTT Examples”进行预订。 IBM MQ C 应用程序 第 97 页的图 45 使用主题字符串 “MQTT Examples”发布到主题。 在代码片段 第 98 页的图 46 中， MQTT 客户机接收回调方法 `messageArrived` 中的发布内容。

有关如何配置 IBM MQ 以发送发布以响应来自 MQTT 客户机的预订的更多信息，请参阅 发布消息以响应 MQTT 客户机预订。

IBM MQ 应用程序将消息直接发送到 MQTT 客户机

执行任务 第 102 页的『使用 IBM MQ Explorer 向 MQTT 客户机发送消息』 以将消息直接从 IBM MQ 发送到 MQTT 客户机。

以这种方式发送到 MQTT 客户机的消息称为非请求消息。 MQTT v3 客户机作为主题名称集的发布接收自发消息。 遥测 (MQXR) 服务将主题名称设置为远程队列名称。

有关如何配置 IBM MQ 以将消息直接发送到 MQTT 客户机的更多信息，请参阅 直接将消息发送到客户机。

MQTT 客户机发布消息

MQTT v3 客户机可以发布另一个 MQTT v3 客户机接收到的消息，但它无法发送未经请求的消息。 代码片段 第 98 页的图 47 显示了以 Java 编写的 MQTT v3 客户机如何发布消息。

用于将消息发送到一个特定 MQTT v3 客户机的典型模式是每个客户机创建对其自己的 `ClientIdentifier` 的预订。 执行任务 第 103 页的『将消息发布到特定 MQTT v3 客户机』 以使用 `ClientIdentifier` 作为主题字符串将消息从一个 MQTT 客户机发布到另一个 MQTT 客户机。

示例代码片段

第 97 页的图 44 中的代码片段显示了在 Java 中编写的 MQTT 客户机如何创建预订。 它还需要回调方法 `messageArrived` 来接收预订的发布。

```
String    clientId = String.format("%-23.23s",
                     System.getProperty("user.name") + "_" +
                     (UUID.randomUUID().toString())).trim()).replace('-', '_');
MqttClient  client = new MqttClient("localhost", clientId);
String topicString = "MQTT Examples";
int            QoS = 1;
client.subscribe(topicString, QoS);
```

图 *44: MQTT v3* 客户机订户

第 97 页的图 45 中的代码片段显示了以 C 编写的 IBM MQ 应用程序如何发送发布内容。 将从以下任务中抽取代码片段: Create a publisher to a variable topic

```
/* Define and set variables to.defaults */
/* Omitted lines declaring variables    */
char * topicName   = ""
char * topicString = "MQTT Examples"
char * publication = "Hello world!";
do {
  MQCONN(qMgrName, &Hconn, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
  td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
  strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
  td.ObjectString.VSPtr = topicString;
  td.ObjectString.VSLength = (MQLONG)strlen(topicString);
  MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
  MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
```

图 *45: IBM MQ* 发布者

当发布到达时， MQTT 客户机将调用 MQTT application client `MqttCallback` 类的 `messageArrived` 方法。

```
public class CallBack implements MqttCallback {
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
// ... Other callback methods
}
```

图 *46: messageArrived* 方法

第 98 页的图 47 显示了 MQTT v3 将消息发布到 第 97 页的图 44 中创建的预订。

```
      String          address = "localhost";
      String         clientId = String.format("%-23.23s",
                                System.getProperty("user.name") + "_" +
                                (UUID.randomUUID().toString())).trim()).replace('-', '_');
      MqttClient       client = new MqttClient(address, clientId);
      String      topicString = "MQTT Examples";
      MqttTopic         topic = client.getTopic(Example.topicString);
      String     publication  = "Hello world";
      MqttMessage     message = new MqttMessage(publication.getBytes());
      MqttDeliveryToken token = topic.publish(message);
```

图 *47: MQTT v3* 客户机发布程序

Windows Linux AIX 从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序

执行本任务中的步骤以使用 IBM MQ Explorer 发布消息，并使用 MQTT 客户机实用程序预订该消息。 其他任务显示如何配置队列管理器别名，而不是将缺省传输队列设置为 `SYSTEM.MQTT.TRANSMIT.QUEUE`。

开始之前

该任务假定您熟悉 IBM MQ 和 IBM MQ Explorer，并且已安装 IBM MQ 和 MQ Telemetry 功能部件。

为此任务创建队列管理器资源的用户必须具有足够的权限才能执行此操作。 出于演示目的，假定 IBM MQ Explorer 用户标识是 `mqm` 组的成员。

关于此任务

在该任务中，您将在 IBM MQ 中创建主题，并使用 MQTT 客户机实用程序预订该主题。 使用 IBM MQ Explorer 发布到主题时， MQTT 客户机将接收到该发布。

过程

请执行下列其中一项任务：

- 您已安装 MQ Telemetry，但尚未将其启动。 执行任务: 第 99 页的『在尚未定义遥测 (MQXR) 服务的情况下启动任务』。
- 您之前已运行 IBM MQ 遥测，但希望使用新的队列管理器来执行演示。 执行任务: 第 99 页的『在尚未定义遥测 (MQXR) 服务的情况下启动任务』。
- 您希望使用未定义遥测资源的现有队列管理器来执行该任务。 您不希望运行 " **定义样本配置** " 向导。

  a. 执行下列其中一项任务以设置遥测：

- 在 Linux 和 AIX 上配置队列管理器以进行遥测
- 在 Windows 上配置队列管理器以进行遥测

b. 执行任务: 第 100 页的『使用正在运行的遥测 (MQXR) 服务启动任务』

- 如果要使用已定义遥测资源的现有队列管理器来执行任务，请执行以下任务: 第 100 页的『使用正在运行的遥测 (MQXR) 服务启动任务』。

下一步做什么

执行 第 102 页的『使用 IBM MQ Explorer 向 MQTT 客户机发送消息』 以将消息直接发送到客户机实用程序。

在尚未定义遥测 *(MQXR)* 服务的情况下启动任务

创建队列管理器并运行 **定义样本配置** 以定义队列管理器的样本遥测资源。 使用 IBM MQ Explorer 发布消息，并使用 MQTT 客户机实用程序预订该消息。

关于此任务

使用 **定义样本配置**设置样本遥测资源时，向导会设置访客用户标识许可权。 请仔细考虑是否希望以这种方式对访客用户标识进行授权。 `guest` on Windows 和 `nobody` on Linux 被授予发布和预订主题树的根以及将消息放入 `SYSTEM.MQTT.TRANSMIT.QUEUE` 的许可权。

向导还将缺省传输队列设置为 `SYSTEM.MQTT.TRANSMIT.QUEUE`，这可能会干扰在现有队列管理器上运行的应用程序。 可以但费力地配置遥测，而不使用缺省传输队列; 请执行以下任务: 第 101 页的『使用队列管理器别名』。 在此任务中，创建队列管理器以避免干扰任何现有缺省传输队列的可能性。

过程

1. 使用 IBM MQ Explorer 创建并启动新的队列管理器。
   a) 右键单击 `Queue Managers` 文件夹> **新建 > 队列管理器 ...**。 输入队列管理器名称> **完成**。

   组成队列管理器名称; 例如， `MQTTQMGR`。
2. 创建并启动遥测 (MQXR) 服务，并创建样本遥测通道。
   a) 打开 `Queue Managers\`*`QmgrName`*`\Telemetry` 文件夹。
   b) 单击 **定义样本配置 ... > 完成**

   保持选中 **启动 MQTT 客户机实用程序** 复选框。
3. 使用 MQTT 客户机实用程序为 `MQTT Example` 创建预订。
   a) 单击**连接**。

   客户机历史记录 记录 `Connected` 事件。
   b) 在 **预订 \ 主题** 字段> **预订**中输入 `MQTT Example` 。

   客户机历史记录 记录 `Subscribed` 事件。
4. 在 IBM MQ 中创建 `MQTTExampleTopic` 。
   a) 右键单击 **MQ Explorer**> **新建** > **主题**中的 `Queue Managers\`*`QmgrName`*`\Topics` 文件夹。
   b) 输入 `MQTTExampleTopic` 作为 **名称** > **下一步**。
   c) 输入 `MQTT Example` 作为 **主题字符串** > **完成**。
   d) 单击 **确定** 以关闭确认窗口。
5. 使用 IBM MQ Explorer 将 `Hello World!` 发布到主题 `MQTT Example` 。
   a) 单击 IBM MQ Explorer 中的 `Queue Managers\`*`QmgrName`*`\Topics` 文件夹。
   b) 右键单击 `MQTTExampleTopic` > **测试发布 ...**
   c) 在 **消息数据** 字段> **发布消息** > 切换到 " MQTT 客户机实用程序" 窗口中输入 `Hello World!` 。

   客户机历史记录 记录 `Received` 事件。

使用正在运行的遥测 (MQXR) 服务启动任务

创建遥测通道和主题。 授权用户使用主题和遥测传输队列。 使用 IBM MQ Explorer 发布消息，并使用 MQTT 客户机实用程序预订该消息。

开始之前

在此版本的任务中，定义并运行了队列管理器 *QmgrName*。 定义并运行遥测 (MQXR) 服务。 遥测 (MQXR) 服务可能是手动创建的，也可能是通过运行 " **定义样本配置** " 向导创建的。

关于此任务

在此任务中，您将配置现有队列管理器以将发布发送到 MQTT 客户机实用程序。

任务的步骤 第 100 页的『1』 将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE，这可能会干扰在现有队列管理器上运行的应用程序。 可以但费力地配置遥测，而不使用缺省传输队列; 请执行以下任务: 第 101 页的『使用队列管理器别名』。

过程

1. 将 SYSTEM.MQTT.TRANSMIT.QUEUE 设置为缺省传输队列。
   a) 右键单击 Queue Managers*QmgrName* folder > **属性 ...**
   b) 在导航器中单击 **通信** 。
   c) 单击 **选择 ...** > 选择 SYSTEM.MQTT.TRANSMIT.QUEUE > **确定** > **确定**。
2. 创建遥测通道 MQTTExampleChannel 以将 MQTT 客户机实用程序连接到 IBM MQ，并启动 MQTT 客户机实用程序。
   a) 右键单击 **MQ Explorer**> **新建** > **遥测通道 ...**中的 Queue Managers*QmgrName* \Telemetry\Channels 文件夹。
   b) 在 **通道名称** 字段中输入 MQTTExampleChannel > **Next** > **Next**。
   c) 将客户机授权面板上的 **固定用户标识** 更改为要发布和预订 MQTTExample > **下一步**的用户标识。
   d) 保持选中 **启动客户机实用程序** > **完成**。
3. 使用 MQTT 客户机实用程序为 MQTT Example 创建预订。
   a) 单击**连接**。

      客户机历史记录 记录 Connected 事件。
   b) 在 **预订 \ 主题** 字段> **预订**中输入 MQTT Example 。

   客户机历史记录 记录 Subscribed 事件。
4. 在 IBM MQ 中创建 MQTTExampleTopic 。
   a) 右键单击 **MQ Explorer**> **新建** > **主题**中的 Queue Managers*QmgrName*\Topics 文件夹。
   b) 输入 MQTTExampleTopic 作为 **名称** > **下一步**。
   c) 输入 MQTT Example 作为 **主题字符串** > **完成**。
   d) 单击 **确定** 以关闭确认窗口。
5. 如果您希望用户 (不在 mqm 组中) 发布和预订 MQTTExample 主题，请执行以下操作:
   a) 授权用户发布和预订主题 MQTTExampleTopic:

```
setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
```

   b) 授权用户将消息放入 SYSTEM.MQTT.TRANSMIT.QUEUE:

```
setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
```

6. 使用 IBM MQ Explorer 将 Hello World! 发布到主题 MQTT Example 。

a) 单击 IBM MQ Explorer 中的 Queue Managers*QmgrName*\Topics 文件夹。

b) 右键单击 MQTTExampleTopic > **测试发布 ...**

c) 在 **消息数据** 字段> **发布消息** > 切换到 " MQTT 客户机实用程序" 窗口中输入 Hello World! 。

客户机历史记录 记录 Received 事件。

使用队列管理器别名

使用 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序，而不将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE。

该任务是先前任务的延续，并使用队列管理器别名来避免将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE。

开始之前

完成任务 第 99 页的『在尚未定义遥测 (MQXR) 服务的情况下启动任务』 或任务 第 100 页的『使用正在运行的遥测 (MQXR) 服务启动任务』。

关于此任务

当 MQTT 客户机创建预订时， IBM MQ 将使用 ClientIdentifier 作为远程队列管理器名称来发送其响应。 在此任务中，它使用 ClientIdentifier MyClient。

如果没有名为 MyClient 的传输队列或队列管理器别名，那么响应将放在缺省传输队列上。 通过将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE， MQTT 客户机将获取响应。

您可以避免使用队列管理器别名将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE 。 必须为每个 ClientIdentifier 设置队列管理器别名。 通常，客户机过多，以致无法实际使用队列管理器别名。 通常， ClientIdentifier 不可预测，因此无法以此方式配置遥测。

但是，在某些情况下，您可能必须将缺省传输队列配置为 SYSTEM.MQTT.TRANSMIT.QUEUE 以外的传输队列。 过程 中的步骤配置队列管理器别名，而不是将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE。

过程

1. 除去 SYSTEM.MQTT.TRANSMIT.QUEUE 作为缺省传输队列。
   a) 右键单击 Queue Managers*QmgrName* folder > **属性 ...**
   b) 在导航器中单击 **通信** 。
   c) 从 **缺省传输队列** 字段中除去 SYSTEM.MQTT.TRANSMIT.QUEUE > **确定**。
2. 检查您是否无法再使用 MQTT 客户机实用程序创建预订:
   a) 单击**连接**。

      客户机历史记录 记录 Connected 事件。
   b) 在 **预订 \ 主题** 字段> **预订**中输入 MQTT Example 。

   客户机历史记录 记录 Subscribe failed 和 Connection lost 事件。
3. 为 ClientIdentifier MyClient 创建队列管理器别名。
   a) 右键单击 Queue Managers*QmgrName*\Queues 文件夹> **新建** > **远程队列定义**。
   b) 将定义命名为 MyClient > **Next**。
   c) 在 **远程队列管理器** 字段中输入 MyClient 。
   d) 在 **传输队列** 字段> **完成**中输入 SYSTEM.MQTT.TRANSMIT.QUEUE 。
4. 再次连接 MQTT 客户机实用程序。
   a) 检查 **客户机标识** 是否设置为 MyClient。
   b) **CONNECT**

客户机历史记录 记录 Connected 事件。

5. 使用 MQTT 客户机实用程序为 MQTT Example 创建预订。
   a) 单击**连接**。

      客户机历史记录 记录 Connected 事件。

   b) 在 **预订 \ 主题** 字段> **预订**中输入 MQTT Example 。

   客户机历史记录 记录 Subscribed 事件。
6. 使用 IBM MQ Explorer 将 Hello World! 发布到主题 MQTT Example 。
   a) 单击 IBM MQ Explorer 中的 Queue Managers*QmgrName*\Topics 文件夹。
   b) 右键单击 MQTTExampleTopic > **测试发布 ...**
   c) 在 **消息数据** 字段> **发布消息** > 切换到 " MQTT 客户机实用程序" 窗口中输入 Hello World! 。

   客户机历史记录 记录 Received 事件。

Windows Linux AIX 使用 IBM MQ Explorer 向 MQTT 客户机发送消息

通过使用 IBM MQ Explorer 将消息放入 IBM MQ 队列，将消息发送到 MQTT 客户机实用程序。 此任务说明如何配置远程队列定义以将消息直接发送到 MQTT 客户机。

开始之前

执行任务 第 98 页的『从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序』。 使 MQTT 客户机实用程序保持连接。

关于此任务

此任务演示如何使用队列将消息发送到 MQTT 客户机，而不是发布到主题。 您不会在客户机中创建预订。 该任务的步骤 第 102 页的『2』 演示已删除先前的预订。

过程

1. 通过断开连接并重新连接 MQTT 客户机实用程序来废弃任何现有预订。

   将废弃预订，因为除非您更改缺省值，否则 MQTT 客户机实用程序将与清除会话连接; 请参阅 清除会话。

   为了更容易执行该任务，请输入您自己的 ClientIdentifier，而不是使用由 MQTT 客户机实用程序创建的生成的 ClientIdentifier 。

   a) 单击 **断开连接** 以断开 MQTT 客户机实用程序与遥测通道的连接。

      客户机历史记录 记录 Disconnected 事件

   b) 将 **客户机标识** 更改为 MyClient。

   c) 单击**连接**。

   客户机历史记录 记录 Connected 事件
2. 检查 MQTT 客户机实用程序是否不再接收 MQTTExampleTopic 的发布。
   a) 单击 IBM MQ Explorer 中的 Queue Managers*QmgrName*\Topics 文件夹。
   b) 右键单击 MQTTExampleTopic > **测试发布 ...**
   c) 在 **消息数据** 字段> **发布消息** > 切换到 " MQTT 客户机实用程序" 窗口中输入 Hello World! 。

   客户机历史记录中未记录任何事件。
3. 为客户机创建远程队列定义。

   将 ClientIdentifier MyClient 设置为远程队列定义中的远程队列管理器名称。 使用您喜欢的任何名称作为远程队列名称。 远程队列名称将作为主题名称传递到 MQTT 客户机。

   a) 右键单击 Queue Managers*QmgrName*\Queues 文件夹> **新建** > **远程队列定义**。

   b) 将定义命名为 MyClientRemoteQueue > **Next**。

c) 在 **远程队列** 字段中输入 MQTTExampleQueue 。

d) 在 **远程队列管理器** 字段中输入 MyClient 。

e) 在 **传输队列** 字段> **完成**中输入 SYSTEM.MQTT.TRANSMIT.QUEUE 。

4. 将测试消息放在 MyClientRemoteQueue 上。

a) 右键单击 **MyClientRemoteQueue** > **放置测试消息 ...**

b) 在 "消息数据" 字段中输入 Hello queue! > **放入消息** > **关闭**

客户机历史记录 记录 Received 事件。

5. 除去 SYSTEM.MQTT.TRANSMIT.QUEUE 作为缺省传输队列。

a) 右键单击 Queue Managers*QmgrName* folder > **属性 ...**

b) 在导航器中单击 **通信** 。

c) 从 **缺省传输队列** 字段中除去 SYSTEM.MQTT.TRANSMIT.QUEUE > **确定**。

6. 重做步骤 第 103 页的『4』。

MyClientRemoteQueue 是明确指定传输队列的远程队列定义。 您不需要定义缺省传输队列以将消息发送到 MyClient。

下一步做什么

如果缺省传输队列不再设置为 SYSTEM.MQTT.TRANSMIT.QUEUE，那么除非为 ClientIdentifier MyClient 定义了队列管理器别名，否则 MQTT 客户机实用程序无法创建新的预订。 将缺省传输队列复原到 SYSTEM.MQTT.TRANSMIT.QUEUE。

Windows Linux AIX 将消息发布到特定 MQTT v3 客户机

将消息从一个 MQTT v3 客户机发布到另一个客户机，将 ClientIdentifier 用作主题名称，将 IBM MQ 用作发布/预订代理。

开始之前

执行任务 第 98 页的『从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序』。 使 MQTT 客户机实用程序保持连接。

关于此任务

该任务演示了两件事:

1. 在一个 MQTT 客户机中预订主题，并从另一个 MQTT 客户机接收发布内容。
2. 通过使用 ClientIdentifier 作为主题字符串来设置 "点到点" 预订。

过程

1. 通过断开连接并重新连接 MQTT 客户机实用程序来废弃任何现有预订。

将废弃预订，因为除非您更改缺省值，否则 MQTT 客户机实用程序将与清除会话连接; 请参阅 清除会话。

为了更容易执行该任务，请输入您自己的 ClientIdentifier，而不是使用由 MQTT 客户机实用程序创建的生成的 ClientIdentifier 。

a) 单击 **断开连接** 以断开 MQTT 客户机实用程序与遥测通道的连接。

客户机历史记录 记录 Disconnected 事件

b) 将 **客户机标识** 更改为 MyClient。

c) 单击**连接**。

客户机历史记录 记录 Connected 事件

2. 创建主题 MyClient 的预订

MyClient 是此客户机的 ClientIdentifier 。

a) 在 **预订 \ 主题** 字段> **预订**中输入 MyClient 。

客户机历史记录 记录 Subscribed 事件。

3. 启动另一个 MQTT 客户机实用程序。
   a) 打开 Queue Managers*QmgrName*\Telemetry\channels 文件夹。
   b) 右键单击 **PlainText** 通道> **运行 MQTT 客户机实用程序 ...**
   c) 单击**连接**。

   客户机历史记录 记录 Connected 事件
4. 将 Hello MyClient! 发布到主题 MyClient。
   a) 从使用 ClientIdentifier MyClient 运行的 MQTT 客户机实用程序复制预订主题 MyClient。
   b) 将 MyClient 粘贴到每个 MQTT 客户机实用程序实例的 **发布 \ 主题** 字段中。
   c) 在 **Publication \ message** 字段中输入 Hello MyClient! 。
   d) 在这两个实例中单击 **发布** 。

结果

具有 ClientIdentifier MyClient 的 MQTT 客户机实用程序中的 **客户机历史记录** 记录了两个 **已接收** 事件和一个 **已发布** 事件。 另一个 MQTT 客户机实用程序实例记录一个 **已发布** 事件。

如果仅看到一个 **已接收** 事件，请检查以下可能的原因:

1. 队列管理器的缺省传输队列是否设置为 SYSTEM.MQTT.TRANSMIT.QUEUE ?
2. 是否在执行其他练习时创建了引用 MyClient 的队列管理器别名或远程队列定义? 如果存在配置问题，请删除引用 MyClient 的任何资源，例如队列管理器别名或传输队列。 断开客户机实用程序的连接，停止并重新启动遥测 (MQXR) 服务。

Windows Linux AIX 从 MQTT 客户机向 IBM MQ 应用程序发送消息

IBM MQ 应用程序可以通过预订主题从 MQTT v3 客户机接收消息。 MQTT 客户机使用遥测通道连接到 IBM MQ ，并通过发布到同一主题将消息发送到 IBM MQ 应用程序。

执行任务 第 104 页的『从 MQTT 客户机将消息发布到 IBM MQ』，以了解如何将发布从 MQTT 客户机发送到 IBM MQ 中定义的预订。

如果主题是集群主题，或者使用发布/预订层次结构进行分发，那么预订可以位于与 MQTT 客户机所连接的队列管理器不同的队列管理器上。

Windows Linux AIX 从 MQTT 客户机将消息发布到 IBM MQ

使用 IBM MQ Explorer 创建主题预订，并使用 MQTT 客户机实用程序发布到主题。

开始之前

执行任务 第 98 页的『从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序』。 使 MQTT 客户机实用程序保持连接。

关于此任务

此任务演示使用 MQTT 客户机发布消息，并使用使用 IBM MQ Explorer 创建的非受管持久预订接收发布。

过程

1. 创建对主题字符串 MQTT Example 的持久预订。
   执行以下步骤以使用 IBM MQ Explorer 创建队列和预订。
   a) 右键单击 IBM MQ Explorer> **新建** > **本地队列 ...**中的 Queue Managers*QmgrName*\Queues 文件夹。
   b) 输入 MQTTExampleQueue 作为队列名称> **完成**。

c) 右键单击 IBM MQ Explorer> **新建** > **预订 ...**中的 Queue Managers*QmgrName*\Subscriptions 文件夹。

d) 输入 MQTTExampleSubscription 作为队列名称> **Next**。

e) 单击 **选择 ...** > MQTTExampleTopic > **确定**。

您已在 第 98 页的『从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序』的步骤 第 99 页的『4』 中创建主题 MQTTExampleTopic 。

f) 输入 MQTTExampleQueue 作为目标名称> **完成**。

2. 作为可选步骤，在没有 mqm 权限的情况下设置队列以供其他用户使用。

如果要为权限低于 mqm 的用户设置配置，那么必须将 put 和 get 权限授予 MQTTExampleQueue。 在 第 98 页的『从 IBM MQ Explorer 将消息发布到 MQTT 客户机实用程序』中配置了对主题和传输队列的访问权。

a) 授权用户放入并进入队列 MQTTExampleQueue:

```
setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
```

3. 使用 MQTT 客户机实用程序将 Hello IBM MQ! 发布到主题 MQTT Example 。

如果尚未使 MQTT 客户机实用程序保持连接，请右键单击 **PlainText** 通道> **运行 MQTT 客户机实用程序 ...** > **连接**。

a) 在 **发布 \ 主题** 字段中输入 MQTT Example 。

b) 在 **发布 \ 消息** 字段> **发布**中输入 Hello IBM MQ! 。

4. 打开 Queue Managers*QmgrName*\Queues 文件夹并查找 MQTTExampleQueue。

当前队列深度 字段为 1

5. 右键单击 MQTTExampleQueue > **浏览消息 ...** 并检查该出版物。

Windows Linux AIX MQTT 发布/预订应用程序

使用基于主题的发布/预订来编写 MQTT 应用程序。

当 MQTT 客户机连接时，发布在客户机与服务器之间的任一方向上流动。 在客户机上发布信息时，将从客户机发送这些发布。 当将消息发布到与客户机创建的预订匹配的主题时，将在客户机上接收到发布内容。

IBM MQ 发布/预订代理程序管理 MQTT 客户机创建的主题和预订。 MQTT 客户机创建的主题与 IBM MQ 应用程序创建的主题共享相同的主题空间。

将与 MQTT 客户机预订中的主题字符串匹配的发布放在 SYSTEM.MQTT.TRANSMIT.QUEUE 上，并将远程队列管理器名称设置为客户机的 ClientIdentifier 。 遥测 (MQXR) 服务将发布转发到创建预订的客户机。 它使用已设置为远程队列管理器名称的 ClientIdentifier 来标识客户机。

通常，必须将 SYSTEM.MQTT.TRANSMIT.QUEUE 定义为缺省传输队列。 可以将 MQTT 配置为不使用缺省传输队列，但这很繁重; 请参阅 配置分布式排队以将消息发送到 MQTT 客户机。

MQTT 客户机可以创建持久会话; 请参阅 第 108 页的『MQTT 无状态和有状态会话』 。 在持久会话中创建的预订是持久的。 针对具有持久会话的客户机的发布将存储在 SYSTEM.MQTT.TRANSMIT.QUEUE 中，并在客户机重新连接时转发到该客户机。

MQTT 客户机还可以发布和预订保留的发布; 请参阅 保留的发布和 MQTT 客户机。 保留发布主题的订户接收到该主题的最新发布。 订户在创建预订时或在重新连接到其先前会话时接收保留的发布内容。

Windows Linux AIX 遥测应用程序

使用 IBM MQ 或 IBM Integration Bus 消息流编写遥测应用程序。

使用 JMS， MQI 或其他 IBM MQ 编程接口来编程 IBM MQ 中的遥测应用程序。

遥测 (MQXR) 服务在 MQTT v3 消息和 IBM MQ 消息之间进行转换。 它代表 MQTT 客户机创建预订和发布，并将发布转发到 MQTT 客户机。 发布是 MQTT v3 消息的有效内容。 有效内容包含消息头和 jms-bytes 格

式的字节数组。 遥测服务器在 MQTT v3 消息与 IBM MQ 消息之间映射头; 请参阅 第 106 页的『MQ Telemetry 与队列管理器的集成』。

使用 Publication， MQInput 和 JMSInput 节点在 IBM Integration Bus 和 MQTT 客户机之间发送和接收发布。

通过使用消息流，您可以将遥测与使用 HTTP 的 Web 站点以及使用 IBM MQ 和 WebSphere Adapters 的其他应用程序集成。

Windows Linux AIX MQ Telemetry 与队列管理器的集成

MQTT 客户机作为发布/预订应用程序与 IBM MQ 集成。 它可以发布或预订 IBM MQ 中的主题，创建新主题或使用现有主题。 由于 MQTT 客户机 (包括其自身) 或其他发布到其预订主题的 IBM MQ 应用程序，它从 IBM MQ 接收发布。 将应用规则来决定发布的属性。

不支持与 IBM MQ 提供的主题，发布，预订和消息相关联的许多属性。 第 106 页的『MQTT 客户机到 IBM MQ 发布/预订代理程序』 和 第 107 页的『IBM MQ 到 MQTT 客户机』 描述如何设置发布属性。 这些设置取决于发布是要进入还是从 IBM MQ 发布/预订代理程序进行。

在 IBM MQ 中，发布/预订主题与管理主题对象相关联。 MQTT 客户机创建的主题没有任何不同。 当 MQTT 客户机为发布创建主题字符串时， IBM MQ 发布/预订代理程序会将其与管理主题对象相关联。 代理程序将出版物中的主题字符串映射到最近的管理主题对象父代。 此映射与 IBM MQ 应用程序的映射相同。 如果没有用户创建的主题，那么发布主题将映射到 SYSTEM.BASE.TOPIC。 应用于发布的属性派生自主题对象。

当 IBM MQ 应用程序或管理员创建预订时，将命名该预订。 使用 IBM MQ Explorer 或使用 **runmqsc** 或 PCF 命令列出预订。 已命名全部 MQTT 客户机预订。 将为其提供以下格式的名称: *ClientIdentifier*:*Topic name*

MQTT 客户机到 IBM MQ 发布/预订代理程序

MQTT 客户机已将发布发送到 IBM MQ。 遥测 (MQXR) 服务会将发布内容转换为 IBM MQ 消息。 IBM MQ 消息包含三个部分:

1. MQMD
2. RFH2
3. 消息

MQMD 属性设置为其缺省值，但 第 106 页的表 9 中注明的属性除外。

表 *9: MQMD* 属性的设置

MQMD 字段	**类型**	**值**
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	设置为下列其中一项: MqttClient.ClientIdentifier MqttConnectOptions.UserName IBM MQ 管理员为遥测通道设置的用户标识。
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (IBM MQ 的缺省值，与缺省值为 4 的 JMS 不同。)
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

RFH2 头不包含用于定义 JMS 消息类型的 <msd> 文件夹。 遥测 (MQXR) 服务会将 IBM MQ 消息创建为缺省 JMS 消息。 缺省 JMS 消息类型为 jms-bytes 消息。 应用程序可以将其他头信息作为消息属性进行访问; 请参阅 消息属性。

RFH2 值设置为如 第 107 页的表 10 中所示。 "格式" 属性在 RFH2 固定头中设置，其他值在 RFH2 文件夹中设置。

表 *10: RFH2* 属性的设置		
RFH2 属性	**类型/文件夹**	**头**
FORMAT	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	复制长度为 1...23 字节的 MqttClient.ClientIdentifier 。
QoS	mqtt/qos	从入局 MQTT 消息复制 QoS 。
消息标识	mqtt/msgid	如果 QoS 为 1 或 2，请从入局 MQTT 消息复制 **消息标识** 。
MQIsRetained	mqps/Ret	如果原始 MQTT 发布是使用 RETAIN 属性集发送的，并且消息是作为保留发布接收的，请进行设置。
MQTopicString	mqps/Top	将 MQTT 消息发布到的主题。

MQTT 发布中的有效内容将映射到 IBM MQ 消息的内容:

表 *11: MQTT* 出版物中的有效内容如何映射到 *IBM MQ* 消息内容		
消息内容	**类型**	**IBM MQ 消息的内容**
缓冲区	MQBYTE *n*	从入局 MQTT 消息复制字节。 长度可以为零。

IBM MQ 到 MQTT 客户机

客户机已预订发布主题。 IBM MQ 应用程序已发布到主题，从而导致发布由 IBM MQ 发布/预订代理发送到 MQTT 订户。 或者， IBM MQ 应用程序已直接向 MQTT 客户机发送非请求消息。 第 107 页的表 12 描述了如何在发送到 MQTT 客户机的消息中设置固定消息头。 将废弃 IBM MQ 消息头或任何其他头中的任何其他数据。 IBM MQ 消息中的消息数据将作为 MQTT 消息中的消息有效内容发送，而不进行任何更改。 MQTT 消息由遥测 (MQXR) 服务发送到 MQTT 客户机。

表 *12:* 如何在发送到 *MQTT* 客户机的 *IBM MQ* 消息中设置固定消息头		
MQTT 字段	**类型**	**值**
DUP	BOOLEAN	如果设置为 QoS = 1 或 2，并且在先前的传输中已将消息发送到此客户机，并且在一段时间后未确认该消息。
QoS	int	从 IBM MQ 中的发布/预订代理设置出局发布中 QoS 的值的方式取决于传入发布。 这取决于是从 MQTT 客户机发送传入发布，还是从 IBM MQ 应用程序发送传入发布。 **MQTT** 入局发布以及订户所请求的 QoS 中 QoS 的较小值。 **IBM MQ** 从传入发布派生的 QoS 的较小值: MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 以及订户请求的 QoS 。 如果在没有预订的情况下将消息发送到客户机，那么缺省情况下会将 QoS 设置为 2。 客户机可以通过预订具有不同 QoS 的 DEFAULT.QoS 来更改此值。

表 12: 如何在发送到 *MQTT* 客户机的 *IBM MQ* 消息中设置固定消息头 (继续)		
MQTT 字段	**类型**	**值**
RETAIN	BOOLEAN	如果传入发布具有保留属性集，请进行设置。

第 108 页的表 13 描述了如何在发送到 MQTT 客户机的 MQTT 消息中设置变量消息头。

表 13: 如何在发送到 *MQTT* 客户机的 *MQTT* 消息中设置 *MQTT* 变量头属性		
MQTT 字段	**类型**	**值**
Topic name	**字符串**	发布消息所使用的主题字符串。
Message ID	**字符串**	MQMD.MsgId 属性 (当它放置在 SYSTEM.MQTT.TRANSMIT.QUEUE 中时)。
Payload	byte[]	将字节从传入发布直接复制到发布/预订代理。 长度可以为零。

Windows Linux AIX MQTT 无状态和有状态会话

MQTT 客户机可以创建与队列管理器的有状态会话。 当有状态 MQTT 客户机断开连接时，队列管理器会维护客户机创建的预订以及未完成的消息。 当客户机重新连接时，它将解析正在使用的消息。 它会发送排队等候传递的所有消息，并接收断开连接时针对预订发布的所有消息。

当 MQTT 客户机连接到遥测通道时，它将启动新会话或恢复旧会话。 新会话没有未确认的未完成消息，没有预订，也没有等待传递的发布。 当客户机连接时，它指定是从干净会话开始，还是恢复现有会话; 请参阅 干净会话。

如果客户机恢复现有会话，那么它将继续执行，就像连接未中断一样。 等待传递的发布将发送到客户机，并且未落实的任何消息传输都将完成。 当持久会话中的客户机与遥测 (MQXR) 服务断开连接时，客户机创建的任何预订都将保留。 预订的发布将在客户机重新连接时发送到客户机。 如果在不恢复旧会话的情况下重新连接，那么遥测 (MQXR) 服务将废弃这些发布。

会话状态信息由队列管理器保存在 SYSTEM.MQTT.PERSISTENT.STATE 队列中。

IBM MQ 管理员可以断开连接并清除会话。

Windows Linux AIX 未连接 MQTT 客户机时

当客户机未连接时，队列管理器可以继续代表它接收发布。 它们将在客户机重新连接时转发给客户机。 如果客户机意外断开连接，那么客户机可以创建队列管理器代表客户机发布的 “Last will and testament”。

如果您希望在客户机意外断开连接时收到通知，那么可以注册最后一个遗嘱和遗嘱发布; 请参阅 最后一个遗嘱和遗嘱发布。 它由遥测 (MQXR) 服务发送，如果它检测到与客户机的连接已断开，而客户机未发出请求。

客户机可以随时发布保留的出版物; 请参阅 保留的出版物和 MQTT 客户机。 对主题的新预订可以请求发送与主题关联的任何保留发布。 如果创建最后一个遗嘱和遗嘱作为保留发布，那么可以使用它来监视客户机的状态。

例如，客户机在连接时发布保留发布，以公布其可用性。 同时，它创建了一个保留的最后遗嘱和遗嘱出版物，宣布其不可用。 此外，就在它进行计划中的断开连接之前，它将其不可用性作为保留出版物发布。 要了解客户机是否可用，您将预订保留发布的主题。 您将始终收到三个出版物中的一个。

如果客户机要在断开连接时接收发布的消息，请将该客户机重新连接到其先前会话; 请参阅 第 108 页的『MQTT 无状态和有状态会话』。 其预订处于活动状态，直到删除这些预订或客户机创建干净的会话为止。

Windows Linux AIX MQTT 客户机与 IBM MQ 应用程序之间的松耦合

MQTT 客户机与 IBM MQ 应用程序之间的发布流是松散耦合的。 出版物可能源自 MQTT 客户机或 IBM MQ 应用程序，并且没有设置顺序。 出版商和订阅者之间存在松散的联系。 它们通过出版物和订阅进行间接互动。 您还可以从 IBM MQ 应用程序直接将消息发送到 MQTT 客户机。

MQTT 客户机和 IBM MQ 应用程序以两种方式松散耦合:

1. 发布者和订阅者通过出版物和预订与主题的关联进行松散耦合。 发布者和订户通常不知道发布或订阅的其他来源的地址或身份。
2. MQTT 客户机在不同的线程上发布，预订，接收发布和处理传递应答。

MQTT 客户机应用程序不会等到发布已交付。 应用程序将消息传递到 MQTT 客户机，然后应用程序在其自己的线程上继续操作。 传递令牌用于将应用程序与发布的传递同步; 请参阅 传递令牌。

将消息传递到 MQTT 客户机后，应用程序可以选择等待 delivery-token。 客户机可以提供在将发布传递到 IBM MQ 时调用的回调方法，而不是等待。 它还可以忽略 delivery-token。

根据与消息关联的服务质量，会将 delivery-token 立即返回到回调方法，或者可能在一段相当长的时间后返回。 在客户机断开连接并重新连接后，甚至可能返回 delivery-token。 如果服务质量为 *fire and* 算了，那么将立即返回 delivery-token。 在其他两种情况下，仅当客户机接收到已向订户发送发布的确认时，才会返回传递令牌。

由于客户机预订而发送到 MQTT 客户机的发布将传递到 `messageArrived` 回调方法。 `messageArrived` 在与主应用程序不同的线程上运行。

将消息直接发送到 MQTT 客户机

您可以通过两种方法之一向特定 MQTT 客户机发送消息。

1. IBM MQ 应用程序可以在没有预订的情况下直接将消息发送到 MQTT 客户机; 请参阅 直接将消息发送到客户机。
2. 另一种方法是使用 `ClientIdentifier` 命名约定。 使所有 MQTT 订户使用其唯一 `ClientIdentifier` 作为主题来创建预订。 发布到 *ClientIdentifier*。 该出版物将发送到预订了主题 *ClientIdentifier* 的客户机。 通过使用此技术，您可以将发布发送到特定 MQTT 订户。

Windows Linux AIX MQ Telemetry 安全性

保护遥测设备的安全很重要，因为这些设备可能可移动，并且用于无法严格控制的场所。 您可以使用 VPN 来保护从 MQTT 设备到遥测 (MQXR) 服务的连接。 MQ Telemetry 提供了另外两种安全性机制: TLS 和 JAAS。

TLS 主要用于加密设备与遥测通道之间的通信，并认证设备正在连接到正确的服务器; 请参阅 使用 TLS 的遥测通道认证。 您还可以使用 TLS 来检查是否允许客户机设备连接到服务器; 请参阅 MQTT 使用 TLS 的客户机认证。

JAAS 主要用于检查是否允许设备的用户使用服务器应用程序; 请参阅 MQTT 使用密码进行客户机认证。 JAAS 可以与 LDAP 配合使用，以使用单点登录目录来检查密码。

TLS 和 JAAS 可以结合使用以提供双因子认证。 您可以将 TLS 使用的密码限制为符合 FIPS 标准的密码。

拥有至少数以万计的用户，提供个人安全概要文件并不总是切实可行的。 使用概要文件来授权个别用户访问 IBM MQ 对象也并非始终可行。 而是将用户分组到用于授权发布和预订主题以及向客户机发送发布的类中。

配置每个遥测通道以将客户机映射到公共客户机用户标识。 对在特定通道上连接的每个客户机使用公共用户标识; 请参阅 MQTT 客户机身份和授权。

授权用户组不会影响每个人的认证。 每个用户都可以在客户机或服务器上使用其 **用户名** 和 **密码**进行认证，然后在服务器上使用公共用户标识进行授权。

Windows Linux AIX MQ Telemetry 全球化

MQTT v3 协议中的消息有效内容将编码为字节数组。 通常，处理文本的应用程序会在 UTF-8 中创建消息有效内容。 遥测通道将消息有效内容描述为 UTF-8，但不执行任何代码页转换。 发布主题字符串必须为 UTF-8。

应用程序负责将字母数据转换为正确的代码页和数字数据转换为正确的数字编码。

MQTT Java 客户机具有方便的 `MqttMessage.toString` 方法。 此方法将消息有效内容视为以本地平台缺省字符集 (通常为 UTF-8) 进行编码。 它将有效内容转换为 Java 字符串。 Java 具有 String 方法

getBytes，用于将字符串转换为使用本地平台缺省字符集编码的字节数组。 两个 MQTT Java 程序在具有相同缺省字符集的平台之间交换消息有效内容中的文本，在 UTF-8 中轻松高效地执行此操作。

如果其中一个平台的缺省字符集不是 UTF-8，那么应用程序必须建立用于交换消息的约定。 例如，发布程序使用 getBytes("UTF8") 方法指定从字符串到 UTF-8 的转换。 要接收消息文本，订户假定消息以 UTF-8 字符集进行编码。

遥测 (MQXR) 服务将来自 MQTT 客户机消息的所有入局发布的编码描述为 UTF-8。 它设置 MQMD.CodedCharSetId 到 UTF-8， RFH2.CodedCharSetId 到 MQCCSI_INHERIT；请参阅 第 106 页的『MQ Telemetry 与队列管理器的集成』。 出版物的格式设置为 MQFMT_NONE，因此通道或 MQGET 无法执行任何转换。

Windows Linux AIX MQ Telemetry 的性能和可伸缩性

在管理大量客户机和提高 MQ Telemetry 的可伸缩性时，请考虑以下因素。

容量规划

有关 MQ Telemetry 的性能报告的信息，请参阅 MQ 性能文档。

连接

连接所涉及的成本包括

- 在处理器使用情况和时间方面设置连接本身的成本。
- 网络成本。
- 保持连接打开但不使用连接时使用的内存。

当客户保持连接时，会产生额外的负载。 如果连接保持打开状态，那么 TCP/IP 流和 MQTT 消息将使用网络来检查该连接是否仍然存在。 此外，对于保持打开的每个客户机连接，将在服务器中使用内存。

如果每分钟发送多条消息，请保持连接处于打开状态，以避免启动新连接的成本。 如果每 10-15 分钟发送不到一条消息，请考虑断开连接以避免使其保持打开状态的成本。 您可能希望将 TLS 连接保持打开状态但处于空闲状态的时间更长，因为设置成本更高。

此外，请考虑客户的能力。 如果客户机上有存储转发工具，那么您可以对消息进行批处理，并断开发送批处理之间的连接。 但是，如果客户机已断开连接，那么客户机无法从服务器接收消息。 因此，应用程序的用途会影响决策。

如果系统有一个客户机发送许多消息 (例如文件传输)，请不要等待每条消息的服务器响应。 而是发送所有消息并在结束时检查是否已接收到所有消息。 或者，使用 服务质量 (QoS)。

您可以根据消息来改变 QoS，使用 QoS 0 来交付不重要的消息，并使用 QoS 2 来交付重要的消息。 如果 QoS 为 0，那么消息吞吐量可能是 QoS 为 2 的两倍左右。

命名约定

如果要为许多客户机设计应用程序，请实施有效的命名约定。 为了将每个客户机映射到正确的 ClientIdentifier，请使 ClientIdentifier 有意义。 良好的命名约定使管理员能够更轻松地确定哪些客户机正在运行。 命名约定可帮助管理员在 IBM MQ Explorer 中过滤一长串客户机，并帮助确定问题; 请参阅 客户机标识。

吞吐量

主题名称的长度会影响流经网络的字节数。 发布或预订时，消息中的字节数可能很重要。 因此，限制主题名称中的字符数。 当 MQTT 客户机预订主题 IBM MQ 时，为其提供格式的名称:

```
ClientIdentifier: TopicName
```

要查看 MQTT 客户机的所有预订，可以使用 IBM MQ MQSC **DISPLAY** 命令:

```
DISPLAY SUB(' ClientID1:*')
```

在 IBM MQ 中定义供 MQTT 客户机使用的资源

MQTT 客户机连接到 IBM MQ 远程队列管理器。 IBM MQ 应用程序有两种基本方法可将消息发送到 MQTT 客户机: 将缺省传输队列设置为 SYSTEM.MQTT.TRANSMIT.QUEUE 或使用队列管理器别名。 如果存在大量 MQTT 客户机，请定义队列管理器的缺省传输队列。 使用缺省传输队列设置可简化管理工作; 请参阅 配置分布式排队以向 MQTT 客户机发送消息。

通过避免预订来提高可伸缩性。

当 MQTT V3 客户机预订主题时，将通过 IBM MQ 中的遥测 (MQXR) 服务创建预订。 预订将客户机的发布路由到 SYSTEM.MQTT.TRANSMIT.QUEUE。 每个发布的传输头中的远程队列管理器名称都设置为进行预订的 MQTT 客户机的 ClientIdentifier。 如果有许多客户机 (每个客户机都有自己的预订)，那么这将导致在整个 IBM MQ 发布/预订集群或层次结构中维护许多代理预订。 有关不使用发布/预订，而是改为使用基于点到点的解决方案的信息，请参阅 直接向客户机发送消息。

管理大量客户机

要支持许多并发连接的客户机，请通过设置 JVM 参数 **-Xms** 和 **-Xmx** 来增加可用于遥测 (MQXR) 服务的内存。 请按照以下步骤操作:

1. 在遥测服务配置目录中找到 java.properties 文件; 请参阅 Windows 上的遥测 (MQXR) 服务配置目录 或 Linux 上的遥测服务配置目录。
2. 遵循文件中的指示; 1 GB 的堆足以用于 50,000 个并发连接的客户机。

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```

3. 添加其他命令行参数以传递到 java.properties 文件中运行遥测 (MQXR) 服务的 JVM; 请参阅 将 JVM 参数传递到遥测 (MQXR) 服务。

要增加 Linux 上的打开文件描述符数，请将以下行添加到 /etc/security/limits.conf/，然后重新登录。

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

每个套接字都需要一个文件描述符。 遥测服务需要一些额外的文件描述符，因此此数目必须大于所需的打开套接字数。

队列管理器对每个非持久预订使用对象句柄。 要支持许多活动的非持久预订，请增加队列管理器中的最大活动句柄数; 例如:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

图 *48:* 更改 *Windows* 上的最大句柄数

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

图 *49:* 更改 *Linux* 上的最大句柄数

其他注意事项

在规划系统需求时，请考虑重新启动系统所花费的时间长度。 计划的停机时间可能会影响排队等待处理的消息数。 配置系统，以便可以在可接受的时间内成功处理消息。 查看磁盘存储，内存和处理能力。 对于某些

客户机应用程序，可能在客户机重新连接时废弃消息。 要废弃消息，请在客户机连接参数中设置 `CleanSession`；请参阅 清除会话。 或者，在 MQTT 客户机中使用尽力而为的服务质量 `0` 进行发布和预订；请参阅 服务质量。 从 IBM MQ 发送消息时使用 **非持久** 消息。 当系统或连接重新启动时，不会恢复具有这些服务质量的消息。

Windows Linux AIX MQ Telemetry 支持的设备

MQTT 客户机可以在从传感器和执行器到手持设备和车辆系统等一系列设备上运行。

MQTT 客户机很小，并且在受小内存和低处理能力限制的设备上运行。 MQTT protocol 可靠且具有较小的头，这适用于受低带宽，高成本和间歇性可用性约束的网络。

MQ Telemetry 通过 MQTT 客户机应用程序与遥测设备通信。 这些应用程序使用以下资源，所有这些资源都实现 MQTT v3 协议：

- 以下客户机库：
  - *MQTT client for Java*，用于为 (例如) Android， OS X， Linux 或 Windows 设备构建本机应用程序。 使用此客户机库的应用程序可以在 Java 的所有变体上运行，从最小 CLDC (已连接的受限设备配置) /MIDP (移动信息设备概要文件) 到 CDC (已连接的设备配置) /Foundation， J2SE (Java Platform， Standard Edition) 和 J2EE (Java Platform， Enterprise Edition)。 还支持 IBM jclRM 定制类库。 Java ME 平台通常用于小型设备，如执行器，传感器，手机和其他嵌入式设备。 Java SE 平台通常安装在更高端的嵌入式设备上，例如台式计算机和服务器。
  - *MQTT client for C*，用于为 (例如) iOS， OS X， Linux 或 Windows 设备构建本机应用程序。 此客户机库提供 C 参考实现以及针对 Windows 和 Linux 系统的预构建本机客户机。 C 参考实现使 MQTT 能够移植到各种设备和平台。 Intel 上的某些 Windows 系统 (包括 Windows 7， RedHat， Ubuntu 以及 ARM 平台 (例如 Eurotech Viper) 上的某些 Linux 系统) 实现运行 C 客户机的 Linux 版本，但 IBM 不提供对这些平台的服务支持。 如果您打算致电 IBM 支持中心，那么必须在受支持的平台上重现客户机问题。
  - *MQTT client for Java*，用于构建基于浏览器的 Web 应用程序。

MQTT 客户机库可从 Eclipse Paho 和 MQTT.org 免费获取。 请参阅 IBM MQ Telemetry Transport 样本程序。

安全性 IBM MQ

在 IBM MQ 中，有几种提供安全性的方法: 授权服务接口; 用户编写的或第三方的通道出口; 使用传输层安全性 (TLS) 的通道安全性，通道认证记录 和消息安全性。

授权服务接口

对象权限管理器 (OAM) 提供了使用 MQI 调用，命令和访问对象的授权，缺省情况下已启用此授权。 通过 IBM MQ 用户组和 OAM 控制对 IBM MQ 实体的访问。 管理员可以根据需要使用命令行界面来授予或撤销权限。

有关创建授权服务组件的更多信息，请参阅 在 AIX, Linux, and Windows 系统上设置安全性。

用户编写的通道出口或第三方通道出口

通道可以使用用户编写的通道出口或第三方通道出口。 有关更多信息，请参阅 消息传递通道的通道出口程序。

使用 TLS 的通道安全性

传输层安全性 (TLS) 协议提供业界标准的通道安全性，防止窃听，篡改和冒充。

TLS 使用公用密钥和对称技术来提供消息机密性和完整性以及相互认证。

有关 IBM MQ 中安全性的全面复审，包括有关 TLS 的详细信息，请参阅 保护。 有关 TLS 的概述 (包括本节中描述的命令的指针)，请参阅 加密安全协议: TLS。

通道认证记录

使用通道认证记录对授予在通道级别连接系统的访问权进行精确控制。 有关更多信息，请参阅 通道认证记录。

消息安全性

使用 Advanced Message Security(这是 IBM MQ 的单独安装和许可组件) 对使用 IBM MQ 发送和接收的消息提供加密保护。 请参阅 Advanced Message Security。

相关任务

保护

规划安全需求

IBM MQ.NET 受管客户机 TLS 支持

IBM MQ.NET 完全受管客户机提供基于 Microsoft.NET SSLStreams 套件的传输层安全性 (TLS) 支持。 这与基于 IBM Global Security Kit (GSKit)的其他 IBM MQ 客户机不同。

您可以开发 IBM MQ.NET 应用程序以在受管方式或非受管方式下运行。

- 在受管方式下， .NET 应用程序在 .NET CLR (公共语言运行时) 中工作，而无需任何跨平台调用 (例如调用 C MQI)。
- 在非受管方式下，将针对底层 MQI 操作调用 C MQI。 基本上，非受管方式接口包含基于 C MQI 的 .NET 包装程序类。

受管 IBM MQ.NET 客户机使用 Microsoft.NET Framework 库来实现 TLS 安全套接字协议。 Microsoft 中的系统.NET.Security.SSLStream 类用于在 IBM MQ.NET 中实现安全性 (TLS)。

非受管 IBM MQ.NET 客户机方式已支持基于 C MQI (和 GSKit) 的 TLS 功能。 即， TLS 操作由 C MQI 处理。 在此情况下， GSKit 将实现 TLS 安全套接字协议。

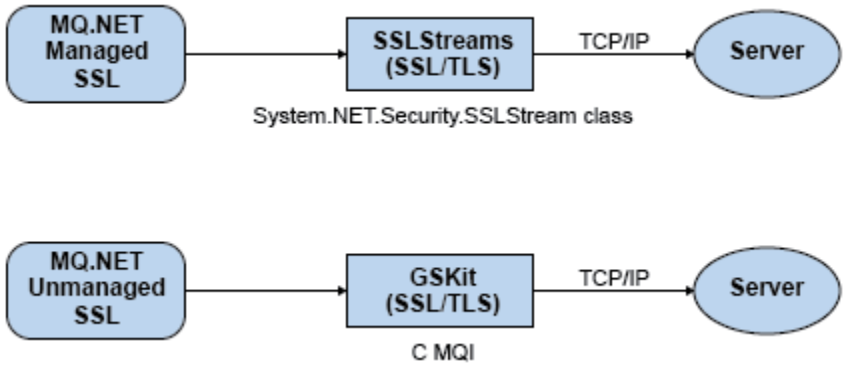


图 *50: IBM MQ.NET* 受管和非受管 *TLS* 比较

下表汇总了受管实现与非受管实现之间的差异:

表 *14:* 受管实施与非受管实施之间的差异

方式	协议	实现	注释
IBM MQ.NET 管理的 SSL	TLS	系统.NET.Security.SSLStream 类 SSLStream 类通过已连接的 TCP 套接字作为流运行	TLS 1.0 TLS 1.2 (仅适用于 Microsoft.NET Framework v4.5)

表 14: 受管实施与非受管实施之间的差异 (继续)			
方式	**协议**	**实现**	**注释**
IBM MQ.NET 非受管 SSL	TLS	GSKit 和 C-MQI	TLS 安全套接字协议

相关概念

.NET 的安全套接字层 (SSL) 和传输层安全性 (TLS) 支持

IBM MQ MQI clients

IBM MQ MQI client 是 IBM MQ 产品的组件，可以安装在没有运行队列管理器的系统上。

IBM MQ MQI 客户机 是一个组件，允许在系统上运行的应用程序向在另一个系统上运行的队列管理器发出 MQI 调用。 来自调用的输出将发送回客户机，客户机会将其传递回应用程序。

通过使用 IBM MQ MQI client，在与客户机相同的系统上运行的应用程序可以连接到在另一个系统上运行的队列管理器。 应用程序可以向该队列管理器发出 MQI 调用。 此类应用程序称为 IBM MQ MQI client 应用程序，队列管理器称为 服务器队列管理器。

IBM MQ 服务器 是向一个或多个客户机提供排队服务的队列管理器。 所有 IBM MQ 对象 (例如，队列) 仅存在于队列管理器机器 (IBM MQ 服务器机器) 上，而不存在于客户机上。 IBM MQ 服务器还可以支持本地 IBM MQ 应用程序。

IBM MQ 服务器与普通队列管理器之间的区别在于，服务器与每个客户机都有专用通信链路。 有关为客户机和服务器创建通道的更多信息，请参阅 配置分布式排队。

IBM MQ MQI client 应用程序和服务器队列管理器使用 *MQI* 通道相互通信。 MQI 通道在客户机应用程序发出 **MQCONN** 或 **MQCONNX** 调用以连接到队列管理器时启动，并在客户机应用程序发出 **MQDISC** 调用以与队列管理器断开连接时结束。 MQI 调用流在 MQI 通道上的一个方向上的输入参数和输出参数在相反方向上的输入参数。

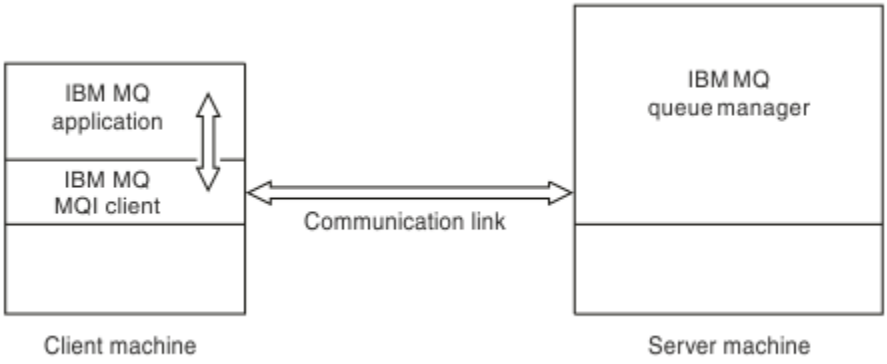


图 51: 客户机与服务器之间的链接

可以使用以下平台。 这些组合取决于您正在使用的 IBM MQ 产品，并在 第 115 页的『针对 IBM MQ 客户机的平台支持』中进行了描述。

IBM MQ MQI client	**IBM MQ 服务器**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

MQI 可用于在客户机平台上运行的应用程序; 队列和其他 IBM MQ 对象保存在已安装在服务器上的队列管理器上。

要在 IBM MQ MQI client 环境中运行的应用程序必须首先与相关客户机库链接。 当应用程序发出 MQI 调用时， IBM MQ MQI client 会将请求定向到队列管理器，在该队列管理器中处理该请求，并从该队列管理器将应答发送回 IBM MQ MQI client。

应用程序与 IBM MQ MQI client 之间的链接是在运行时动态建立的。

您还可以使用 IBM MQ classes for .NET， IBM MQ classes for Java 或 IBM MQ classes for Java Message Service (JMS) 来开发客户机应用程序。 可以在以下平台上使用 Java 和 JMS 客户机:

- 

IBM i
- 

AIX
- 

Linux
- 

Windows

此处未描述 Java 和 JMS 的使用。 有关如何安装，配置和使用 IBM MQ classes for Java 和 IBM MQ classes for JMS 的完整详细信息，请参阅 使用 IBM MQ classes for Java 和 使用 IBM MQ classes for JMS。

客户机/服务器环境中的 IBM MQ 应用程序

当链接到服务器时，客户机 IBM MQ 应用程序可以采用与本地应用程序相同的方式发出大多数 MQI 调用。 客户机应用程序发出 MQCONN 调用以连接到指定的队列管理器。 然后，此队列管理器将处理指定从连接请求返回的连接句柄的任何其他 MQI 调用。

您必须将应用程序链接到相应的客户机库。 请参阅 为 IBM MQ MQI clients 构建应用程序。

相关概念

第 115 页的『为何使用 IBM MQ 客户机?』
使用 IBM MQ 客户机是实现 IBM MQ 消息传递和排队的有效方法。

第 117 页的『什么是扩展事务客户机?』
IBM MQ 扩展事务客户机可以在外部事务管理器的控制下更新由另一个资源管理器管理的资源。

第 118 页的『客户机如何连接到服务器』
客户机使用 MQCONN 或 MQCONNX 连接到服务器，并通过通道进行通信。

第 119 页的『事务管理和支持』
事务管理简介以及 IBM MQ 如何支持事务。

第 120 页的『扩展队列管理器设施』
您可以使用用户出口， API 出口或可安装服务来扩展队列管理器设施。

相关信息

如何设置 IBM MQ MQI client

为何使用 IBM MQ 客户机?

使用 IBM MQ 客户机是实现 IBM MQ 消息传递和排队的有效方法。

您可以使用在一台机器上运行的 MQI 和在另一台机器 (物理或虚拟) 上运行的队列管理器的应用程序。 这样做的好处是:

- 客户机上不需要完整的 IBM MQ 实现。
- 降低了客户机系统上的硬件需求。
- 减少了系统管理需求。
- 在客户机上运行的 IBM MQ 应用程序可以连接到不同系统上的多个队列管理器。
- 可以使用使用不同传输协议的备用通道。

针对 IBM MQ 客户机的平台支持

所有受支持的服务器平台上的 IBM MQ 都接受来自多个平台上的 IBM MQ MQI clients 的客户机连接。

在所有受支持的服务器平台上作为 基本产品和服务器 安装的 IBM MQ 可以接受来自以下平台上的 IBM MQ MQI clients 的连接:

- 

IBM i
- 

AIX
- 

Linux
- 

Windows

客户机连接在编码字符集标识 (CCSID) 和通信协议方面存在差异。

哪些应用程序在 IBM MQ MQI client 上运行?

客户机环境中支持完整 MQI。 这使几乎所有 IBM MQ 应用程序都能够配置为通过将 IBM MQ MQI client 上的应用程序链接到 MQIC 库 (而不是 MQI 库) 来在 IBM MQ MQI client 系统上运行。 例外情况如下所示:

- 带有信号的 MQGET
- 需要与其他资源管理器进行同步点协调的应用程序必须使用扩展事务客户机

如果启用预读，那么为了提高非持久消息传递性能，并非所有 MQGET 选项都可用。 该表显示了允许的选项以及是否可以在 MQGET 调用之间更改这些选项。

表 *15:* 启用预读时允许的 *MQGET* 选项

值	在启用预读并且可以在 MQGET 调用之间进行更改时允许	启用预读时允许，但不能在 MQGET 调用之间更改 [1]	启用预读时不允许使用的 MQGET 选项 [2]
MQGET MD 值	MsgId [3] CorrelId [3]	编码 CodedCharSetId	
MQGET MQGMO 选项	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST [4] MQGMO_BROWSE_NEXT [4] MQGMO_BROWSE_MESSAGE_UNDER_CURSOR [4]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP 备份 (_BACKOUT) MQGMO_MSG_UNDER_CURSOR [4] MQGMO_LOCK MQGMO_UNLOCK
MQGMO 值		MsgHandle	

1. 如果在 MQGET 调用之间更改了这些选项，那么将返回 MQRC_OPTIONS_CHANGED 原因码。
2. 如果在第一个 MQGET 调用上指定这些选项，那么将禁用预读。 如果在后续 MQGET 调用上指定这些选项，那么将返回原因码 MQRC_OPTIONS_ERROR。
3. 客户机应用程序需要意识到，如果 MsgId 和 CorrelId 值在 MQGET 调用之间发生更改，那么具有先前值的消息可能已发送至客户机，并保留在客户机预读缓冲区中，直至被使用（或自动清除）为止。
4. 第一个 MQGET 调用确定在启用了预读时是否要从队列中浏览或获取消息。 如果应用程序尝试同时执行浏览与获取操作，将返回原因码 MQRC_OPTIONS_CHANGED。

5. MQGMO_MSG_UNDER_CURSOR 不能与预读配合使用。 在启用了预读时，可浏览或获取消息，但是不能同时执行这两个操作。

在 IBM MQ MQI client 上运行的应用程序可以同时连接到多个队列管理器，或者在 MQCONN 或 MQCONNX 调用上使用带有星号 (*) 的队列管理器名称 (请参阅 将 IBM MQ MQI client 应用程序连接到队列管理器 中的示例)。

什么是扩展事务客户机?

IBM MQ 扩展事务客户机可以在外部事务管理器的控制下更新由另一个资源管理器管理的资源。

如果您不熟悉事务管理的概念，请参阅 第 119 页的『事务管理和支持』。

请注意， XA 事务客户机现在作为 IBM MQ 的一部分提供。

客户机应用程序可以参与由它所连接的队列管理器管理的工作单元。 在工作单元中，客户机应用程序可以将消息放入该队列管理器所拥有的队列，并从中获取消息。 然后，客户机应用程序可以使用 **MQCMIT** 调用来落实工作单元，或者使用 **MQBACK** 调用来回退工作单元。 但是，在同一工作单元中，客户机应用程序无法更新其他资源管理器的资源，例如 Db2 数据库的表。 使用 IBM MQ 扩展事务客户机将除去此限制。

IBM MQ 扩展事务客户机是具有一些其他功能的 IBM MQ MQI client 。 使用此功能，客户机应用程序可以在同一工作单元中执行以下任务:

- 将消息放入它所连接的队列管理器所拥有的队列，并从中获取消息
- 更新 IBM MQ 队列管理器以外的资源管理器的资源

此工作单元必须由与客户机应用程序在同一系统上运行的外部事务管理器管理。 工作单元不能由客户机应用程序所连接到的队列管理器管理。 这意味着队列管理器只能充当资源管理器，而不能充当事务管理器。 这还意味着客户机应用程序只能使用外部事务管理器提供的应用程序编程接口 (API) 来落实或回退工作单元。 因此，客户机应用程序无法使用 MQI 调用 **MQBEGIN**， **MQCMIT** 和 **MQBACK**。

外部事务管理器使用连接到队列管理器的客户机应用程序所使用的相同 MQI 通道作为资源管理器与队列管理器进行通信。 但是，在发生故障后的恢复情况中，当没有应用程序在运行时，事务管理器可以使用专用 MQI 通道来恢复发生故障时队列管理器正在参与的任何不完整工作单元。

在此部分中，不具有扩展事务功能的 IBM MQ MQI client 称为 IBM MQ 基本客户机。 因此，您可以考虑 IBM MQ 扩展事务客户机由 IBM MQ 基本客户机组成，并添加扩展事务功能。

注: IBM i IBM i 上的 IBM MQ MQI client 不支持 IBM MQ 扩展事务功能。

针对扩展事务客户机的平台支持

Multi

扩展事务客户机可用于支持基本客户机的所有多平台。 客户机不可用于 z/OS。

使用扩展事务客户机的客户机应用程序只能连接到以下 IBM MQ 9.0 或更高版本产品的队列管理器:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ for Linux
- Windows IBM MQ for Windows

z/OS 虽然没有在 z/OS 上运行的扩展事务客户机，但是使用扩展事务客户机的客户机应用程序可以连接到在 z/OS 上运行的队列管理器。

对于每个平台，扩展事务客户机的硬件和软件需求与 IBM MQ 基本客户机的硬件和软件需求相同。 扩展事务客户机支持编程语言 (如果它受 IBM MQ 基本客户机和您正在使用的事务管理器支持)。

有关所有平台的外部事务管理器的信息，请参阅 IBM MQ 的系统需求。

客户机如何连接到服务器

客户机使用 MQCONN 或 MQCONNX 连接到服务器，并通过通道进行通信。

在 IBM MQ 客户机环境中运行的应用程序必须保持客户机与服务器之间的活动连接。

连接由发出 MQCONN 或 MQCONNX 调用的应用程序进行。 客户机和服务器通过 *MQI* 通道进行通信，或者在使用共享对话时，每个对话共享一个 MQI 通道实例。 当调用成功时， MQI 通道实例或对话将保持连接，直到应用程序发出 MQDISC 调用为止。 这是应用程序需要连接到的每个队列管理器的情况。

同一机器上的客户机和队列管理器

当您的机器还安装了队列管理器时，还可以在 IBM MQ MQI client 环境中运行应用程序。

在此情况下，您可以选择链接到队列管理器库或客户机库，但请记住，如果链接到客户机库，那么仍需要定义通道连接。 这在应用程序的开发阶段很有用。 您可以在不依赖他人的情况下在自己的机器上测试程序，并确信将其移至独立 IBM MQ MQI client 环境时仍有效。

不同平台上的客户

在此示例中，服务器在不同平台上与三个 IBM MQ MQI clients 通信。

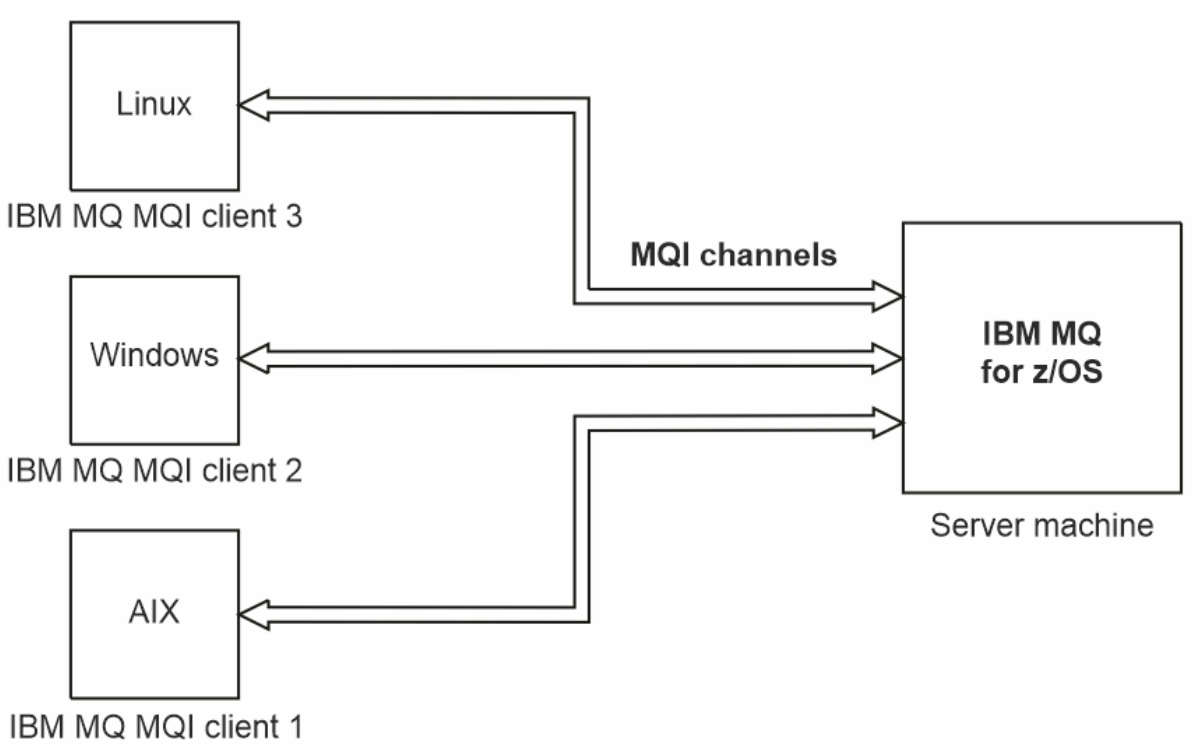


图 *52: IBM MQ* 服务器连接到不同平台上的客户机

其他更复杂的环境是可能的。 例如， IBM MQ 客户机可以连接到多个队列管理器，或者连接到作为队列共享组一部分的任意数量的队列管理器。

使用不同版本的客户机和服务器软件

如果您正在使用先前版本的 IBM MQ 产品，请确保服务器支持从客户机的 CCSID 进行代码转换。

IBM MQ 客户机可以连接到所有受支持的队列管理器版本。 如果要连接到较低版本的队列管理器，那么不能在客户机上的 IBM MQ 应用程序中使用来自较高版本的产品的功能部件和结构。

IBM MQ 队列管理器可以通过向下协商到相互支持的最高协议级别，与不同版本的客户机进行自身通信。 这意味着较旧的客户机可能与较旧的队列管理器级别配合使用。 建议客户机和服务器都是当前支持的 IBM MQ 版本，以促进问题诊断并通过 IBM 启用支持。

有关更多信息，请参阅 开发应用程序中支持的编程语言。

事务管理和支持

事务管理简介以及 IBM MQ 如何支持事务。

资源管理器 是一个计算机子系统，它拥有和管理可由应用程序访问和更新的资源。 以下是资源管理器的示例:

- IBM MQ 队列管理器，其资源是其队列
- Db2 数据库及其表中包含的资源

当应用程序更新一个或多个资源管理器的资源时，可能需要确保某些更新作为一个组成功完成，或者没有任何更新完成。 这种要求的原因是，如果其中一些更新成功完成，但其他更新未成功完成，那么业务数据将处于不一致状态。

以这种方式管理的资源的更新据说发生在 工作单元或 事务中。 应用程序可以将一组更新分组到工作单元中。

在工作单元期间，应用程序向资源管理器发出请求以更新其资源。 当应用程序发出落实所有更新的请求时，工作单元结束。 在落实更新之前，所有更新都不会对访问相同资源的其他应用程序可见。 或者，如果应用程序决定由于任何原因而无法完成工作单元，那么它可以发出请求，以回退其请求的所有更新，直至该点为止。 在这种情况下，任何更新都不会对其他应用程序可见。 这些更新通常在逻辑上相关，并且必须全部成功才能保留数据完整性。 如果一个更新成功，而另一个更新失败，那么数据完整性将丢失。

当工作单元成功完成时，会将其指定为 *commit*。 一旦落实，在该工作单元内进行的所有更新都将永久且不可逆。 但是，如果工作单元失败，那么将改为 回退所有更新。 此过程称为 同步点协调，在此过程中，将落实工作单元或以完整性回退工作单元。

落实或回退工作单元中的所有更新的时间点称为 同步点。 工作单元内的更新据说 在同步点控制内发生。 如果应用程序请求 在同步点控制之外的更新，那么资源管理器会立即落实该更新，即使存在正在进行的工作单元也是如此，并且以后无法回退该更新。

管理工作单元的计算机子系统称为 事务管理器或 点协调程序。

本地工作单元是其中更新的资源仅限于 IBM MQ 队列管理器资源的工作单元。 此处同步点协调由队列管理器本身使用单阶段落实过程提供。

全局 工作单元是其中属于其他资源管理器 (例如，符合 XA 的数据库) 的资源也会更新的工作单元。 在此，必须使用两阶段落实过程，并且工作单元可以由队列管理器本身进行协调，也可以由另一个符合 XA 的事务管理器 (例如 IBM TXSeries 或 BEA Tuxedo) 在外部进行协调。

事务管理器负责确保对工作单元中资源的所有更新都成功完成，或者没有任何更新完成。 应用程序向事务管理器发出落实或回退工作单元的请求。 事务管理器的示例为 CICS 和 WebSphere Application Server，尽管这两个事务管理器都具有其他功能。

某些资源管理器提供自己的事务管理功能。 例如， IBM MQ 队列管理器可以管理涉及对其自己的资源的更新和对 Db2 表的更新的工作单元。 队列管理器不需要单独的事务管理器来执行此功能，尽管如果是用户需求，可以使用此功能。 如果使用单独的事务管理器，那么会将其称为 外部事务管理器。

要让外部事务管理器管理工作单元，事务管理器与参与工作单元的每个资源管理器之间必须有标准接口。 此接口允许事务管理器和资源管理器相互通信。 其中一个接口是 *XA* 接口，它是许多事务管理器和资源管理器支持的标准接口。 XA 接口由开放式组在 分布式事务处理: *XA* 规范中发布。

当多个资源管理器参与工作单元时，事务管理器必须使用 两阶段落实 协议来确保工作单元中的所有更新都成功完成或都未完成，即使发生系统故障也是如此。 当应用程序向事务管理器发出请求以落实工作单元时，事务管理器将执行以下操作:

阶段 1 (准备落实)
事务管理器要求参与工作单元的每个资源管理器确保有关其资源的预期更新的所有信息都处于可恢复状态。 资源管理器通常通过将信息写入日志并确保将信息写入硬盘来执行此操作。 当事务管理器从每个资源管理器接收到有关其资源的预期更新的信息处于可恢复状态的通知时，阶段 1 将完成。

阶段 2 (落实)
当阶段 1 完成时，事务管理器将做出不可撤销的决策来落实工作单元。 它要求参与工作单元的每个资源管理器落实对其资源的更新。 当资源管理器接收到此请求时，它必须落实更新。 在此阶段，它无法选择将它们退出。 当事务管理器从每个资源管理器接收到其已将更新落实到其资源的通知时，阶段 2 完成。

XA 接口使用两阶段落实协议。

有关更多信息，请参阅 事务支持方案。

IBM MQ 还提供对 Microsoft Transaction Server (COM +) 的支持。 使用 Microsoft Transaction Server (COM +) 提供有关如何设置 IBM MQ 以利用 COM + 支持的信息。

扩展队列管理器设施

您可以使用用户出口， API 出口或可安装服务来扩展队列管理器设施。

用户出口

用户出口提供了一种机制，供您将自己的代码插入到队列管理器函数中。 支持的用户出口包括:

通道出口
这些出口会改变通道的运行方式。 通道出口-消息传递通道的出口程序中描述了通道出口。

数据转换出口
这些出口创建可放入应用程序中的源代码片段，以将数据从一种格式转换为另一种格式。 编写数据转换出口中描述了数据转换出口。

集群工作负载出口
此出口执行的函数由该出口的提供程序定义。 调用定义信息在 MQ_CLUSTER_WORKLOAD_EXIT-调用描述中提供。

API 出口

通过 API 出口，可以编写用于更改 IBM MQ API 调用（如 MQPUT 和 MQGET）行为的代码，然后直接在这些调用之前或之后插入该代码。 插入是自动的; 队列管理器在已注册的点处驱动退出代码。 有关 API 出口的更多信息，请参阅 使用和编写 API 出口。

可安装服务

可安装服务具有具有多个入口点的正规化接口 (API)。

可安装服务的实现称为 服务组件。 您可以使用 IBM MQ 随附的组件，也可以编写自己的组件以执行所需的功能。

目前，提供了以下可安装服务:

授权服务
授权服务允许您构建自己的安全设施。

实现服务的缺省服务组件是对象权限管理器 (OAM)。 缺省情况下， OAM 处于活动状态，您不必执行任何操作来对其进行配置。 您可以使用授权服务接口来创建其他组件以替换或扩充 OAM。 有关 OAM 的更多信息，请参阅 在 AIX, Linux, and Windows 系统上设置安全性。

名称服务
名称服务使应用程序能够通过识别远程队列 (就像它们是本地队列一样) 来共享队列。

您可以编写自己的名称服务组件。 例如，如果您打算将名称服务与 IBM MQ 配合使用，那么可能需要执行此操作。 要使用名称服务，您必须具有用户编写的组件或由其他软件供应商提供的组件。 缺省情况下，名称服务处于不活动状态。

相关概念

用户出口、API 出口和 IBM MQ 可安装服务

IBM MQ Java 语言接口

IBM MQ 提供了三个用于 Java 应用程序的应用程序编程接口 (API): IBM MQ classes for Jakarta Messaging， IBM MQ classes for JMS 和 IBM MQ classes for Java。

IBM 支持开放标准，并且是开放标准的积极参与者。

- 从 IBM MQ 8.0 开始，产品实现了 JMS 2.0 标准，该标准引入了新的简化 API 以及诸如共享预订之类的功能。
- 从 IBM MQ 9.3.0 开始，还支持 Jakarta Messaging 3.0 。
- 此外， WebSphere Liberty 支持 JMS 2.0 和 Jakarta Messaging 3.0 与 IBM MQ 配合使用。

在 IBM MQ 中，有三个可在 Java 应用程序中使用的 API:

JM 3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging 是 Jakarta Messaging 提供程序，用于将 IBM MQ 的 Jakarta Messaging 接口实现为消息传递系统。 Jakarta Connectors Architecture 提供了将在 Jakarta EE 环境中运行的应用程序连接到企业信息系统 (EIS) (例如 IBM MQ 或 Db2) 的标准方法。

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS 是 JMS 提供程序，用于将 IBM MQ 的 JMS 接口实现为消息传递系统。 Java Platform, Enterprise Edition Connector Architecture (JCA) 提供一种标准方式将运行在 Java EE 环境中的应用程序连接到企业信息系统 (EIS)，如 IBM MQ 或 Db2。

IBM MQ classes for Java

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

注:

- JMS 2.0 已被 Jakarta Messaging 取代。 IBM MQ classes for JMS 继续支持 JMS 2.0 标准，但 Java 消息传递的未来增强功能将仅在 Jakarta Messaging 中出现，因此在 IBM MQ classes for Jakarta Messaging 中出现。 建议仅使用 IBM MQ classes for JMS 来维护和扩展现有 JMS 2.0 应用程序。 IBM MQ classes for Jakarta Messaging 应该是新开发的首选技术。
- Stabilized IBM MQ classes for Java 功能稳定在 IBM MQ 8.0 随附的级别上。 使用 IBM MQ classes for Java 的现有应用程序仍完全受支持，但此 API 是稳定的，因此将不会添加新的功能部件，并会拒绝针对增强功能的请求。 完全受支持意味着，在完成 IBM MQ 系统需求变化所需的任何更改的同时修复缺陷。

JM 3.0 从 IBM MQ 9.3 开始，使用 Java 8 构建 IBM MQ classes for Java， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 。 必须使用这些级别或更高级别的 Java 运行时环境来运行使用这些接口的应用程序。

相关概念

从 Java 访问 IBM MQ -API 选项

为什么要将 IBM MQ 类用于 Jakarta Messaging?

为什么应该使用 IBM MQ classes for JMS?

为什么应该使用 IBM MQ classes for Java?

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是随 IBM MQ 提供的消息传递提供程序。 其中每个提供程序还提供了两组消息传递 API 的扩展。 Java Platform, Standard Edition (Java SE) 和 Java Platform, Enterprise Edition (Java EE) 应用程序都可以使用这些消息传递提供程序。

JM 3.0 IBM MQ 9.3.0 引入了对 Jakarta Messaging 3.0 的支持。 JMS 2.0 仍完全受支持。

JMS 和 Jakarta Messaging 规范定义了一组接口，应用程序可以使用这些接口来执行消息传递操作。 产品支持 JMS 2.0 版本的 JMS 标准。 此实现提供标准 API 的所有功能，但需要的接口更少，并且更便于使用。 有关更多信息，请参阅 第 124 页的『JMS 和 Jakarta Messaging 模型』 和 JMS 2.0 规范 (Java.net)。

JM 3.0 从 IBM MQ 9.3.0 开始， Jakarta Messaging 也受支持。

jakarta.jms (Jakarta Messaging 3.0) 或 javax.jms (JMS 2.0) 包指定消息传递接口的详细信息，消息传递提供程序为特定消息传递产品实现这些接口。 例如：

- IBM MQ classes for JMS 是 JMS 提供程序，用于实现 IBM MQ 的 JMS 接口，并且还为 JMS API 提供以下两组扩展：
  - IBM MQ JMS 扩展
  - IBM JMS 扩展
- 使用 javax.dims 或 jakarta.jms 创建的连接工厂，队列或主题对象，可以使用这些 API 中的任何一个 API 来寻址接口或一组 JMS 扩展; 即，可以将其强制转换为任何接口。 为保持最大程度的应用程序可移植性，请使用符合您需求的最通用的 API。

由于 JMS 和 Jakarta Messaging 有许多共同之处，因此本主题中对 JMS 的进一步引用可以视为两者的引用。 根据需要突出显示任何差异。

IBM MQ JMS 扩展

IBM MQ classes for JMS 还提供了针对 JMS API 的扩展。 IBM MQ classes for JMS 包含在 MQConnectionFactory， MQQueue 和 MQTopic 对象中实现的扩展。 这些对象包含特定于 IBM MQ 的属性和方法。 这些对象可以是受管对象，也可以由应用程序在运行时动态创建这些对象。 这些扩展称为 IBM MQ JMS 扩展。 请注意，在本文档中，应用程序在运行时动态创建的对象不会被视为受管对象。

IBM JMS 扩展

除 IBM MQ JMS 扩展外， IBM MQ classes for JMS 还提供了一组更通用的 JMS API 或 Java 扩展，作为所使用的编程语言。 这些扩展称为 IBM JMS 扩展，具有以下广泛目标：

- 在 IBM JMS 提供程序之间提供更高级别的一致性。
- 便于在两个 IBM 消息传递系统之间编写网桥应用程序。
- 为了更轻松地将应用程序从一个 IBM JMS 提供程序移植到另一个提供程序。

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

相关任务

使用 IBM MQ classes for JMS/Jakarta Messaging

配置 JMS 和 Jakarta Messaging 资源

JM 3.0 IBM MQ classes for Jakarta Messaging: 概述

IBM MQ 9.3.0 引入了对 Jakarta Messaging 的支持。 对于 Jakarta Messaging 3.0， JMS 规范的控制从 Oracle 移至 Java 社区进程。 但是， Oracle 保留在其他 Java 技术中使用的 "javax" 名称的控制权。 因此，尽管 Jakarta Messaging 3.0 在功能上等同于 JMS 2.0，但在命名方面存在一些差异。 版本 3.0 的正式名称是 Jakarta Messaging 而不是 Java Message Service，并且软件包和常量名称以 `jakarta` 而不是 `javax` 为前缀。

背景

多年来， Java 平台有两种形式: Standard Edition 和 Enterprise Edition。

Java Platform, Standard Edition (有时缩写为 Java SE) 是能够在独立上下文中运行的核心语言和类库。 Java SE 中的大多数 Java 包都具有以 "java." 开头的名称。

Java Platform, Enterprise Edition (Java EE) 对此进行了扩展，添加了诸如 "消息传递" ， "各种 Bean" 和 "事务性" 等功能。 其中一些技术也可以在 Java SE 上下文中使用。 Java EE 中的大多数 Java 包历史上都具有以 "javax" 开头的名称。但是，存在一些交叉，因此某些 Java SE 包具有 "javax"。 作为其名称的前缀。

Java Message Service (JMS) 是 Java Platform, Enterprise Edition 的一部分。 Java EE 7 包含 JMS 2.0。

直到 Java EE 7，这些技术都由 Oracle 管理。

Java EE 技术最近已从 Oracle 的管理流程转变为由 Eclipse Foundation 监督的社区流程。

作为 "javax"。 无法将名称移至新项目，已采用新命名-所有包和属性名称现在都以 "j 惹" 作为前缀。 并且 Java Platform, Enterprise Edition 将在将来被称为 "Jakarta EE"。 版本编号仍在继续: 版本 8 是可在很大程度上被忽略的临时版本，而 Jakarta EE 9 是 "j 惹" 的点。 前缀生效。

在 IBM MQ 上下文中应用的主要 Jakarta EE 技术是 Jakarta Messaging 3.0 - Java Message Service (JMS) 2.0 的后继技术。 因此， Jakarta EE 9 合并 Jakarta Messaging 3.0。

IBM MQ 继续支持 Java EE 7 和 JMS 2.0，同时引入对 Jakarta EE 9 和 Jakarta Messaging 3.0 的支持。

交付的内容: Java SE

对于 Java Platform, Standard Edition，除了 IBM MQ classes for JMS (支持使用 IBM MQ 的 JMS 2.0 操作) IBM MQ 9.3.0 和更高版本之外，还提供 IBM MQ classes for Jakarta Messaging。 这些类提供了与 IBM MQ 集成的 Jakarta Messaging 3.0 提供程序，允许使用 IBM MQ 队列管理器来促进 Jakarta Messaging 操作。

这些文件作为标准 JAR 文件 `com.ibm.mq.jakarta.client.jar` 提供在 IBM MQ 安装的 `java/lib` 子目录中。

为了在 OSGi 容器 (例如 Apache Felix 或 Eclipse Equinox) 中使用， IBM MQ 还提供了一对 OSGi 捆绑软件:

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

其中 *V.R.M.F* 表示 IBM MQ 的版本，例如 9.3.0.0。 可以在 IBM MQ 安装的 `java/lib/OSGi` 子目录中找到这些捆绑软件。

交付的内容: Jakarta EE 9

为了支持 Jakarta EE 9 兼容应用程序服务器中基于 IBM MQ 的消息传递， IBM MQ 提供了 Jakarta EE 9 兼容资源适配器: `wmq.jakarta.jmsra.rar`。 这可以在 IBM MQ 安装的 `java/lib/jca` 子目录中找到。

IBM MQ 继续在 IBM MQ 安装的 `java/lib/jca` 子目录中提供兼容 Java EE 7 的资源适配器 `wmq.jmsra.rar`。

如何交付这些工件

这些 JAR 和资源适配器的 RAR 文件与常规 IBM MQ 安装介质中预先存在的工件一起打包，这些工件包括特定于平台的安装介质 (例如 "`.rpm`" 文件) 和可重新分发的介质 (例如自解压可再分发的客户机 JAR 文件)。

在 JMS 2.0 和 Jakarta Messaging 3.0 之间更改的内容

Jakarta EE 9 和 Jakarta Messaging 3.0 未引入任何新功能。 所有更改都是名称。 例如，在 JMS 2.0 中使用 "javax.jms.Connection" 时，在 Jakarta Messaging 3.0 中使用 "jakarta.jms.Connection"。

随着 Eclipse Foundation 推进 Jakarta EE 平台，它将在此基础上构建，并且此命名约定将用于将来引入的新功能。

在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 之间更改的内容

摘要

IBM MQ classes for JMS(为 JMS 2.0 提供支持) 仍然可用，建议主要用于维护和扩展现有应用程序。 他们得到了充分的支持。

建议将为 Jakarta Messaging 3.0 提供支持的 IBM MQ classes for Jakarta Messaging 用于新开发。

在 IBM MQ 9.3.0 上，这两个产品在功能上等效。 仅命名不同。 但是，与 IBM MQ classes for JMS 相比，新的消息传递功能在 IBM MQ classes for Jakarta Messaging 中出现的可能性更大。

这两个产品可互操作。 IBM MQ classes for JMS 生成的消息可以由 IBM MQ classes for Jakarta Messaging 使用，反之亦然。 但是，这两个产品不得共存于单个应用程序中。

命名更改

表 *16:* 对软件包名称的更改

IBM MQ classes for JMS 包名	**IBM MQ classes for Jakarta Messaging 包名**
com.ibm.mq.jms[.*]	com.ibm.mq.jakarta.jms[.*]
com.ibm.jms	com.ibm.jakarta.jms
com.ibm.msg.client.jms.*	com.ibm.msg.client.jakarta.jms.*
com.ibm.msg.client.wmq.*	com.ibm.msg.client.jakarta.wmq.*

与公共服务 (跟踪，日志记录，本地语言支持等) 和 JMQI 实现 (本地和远程) 相关的包对于 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 都是公共的，因此在这些区域中不需要进行任何更改。

请注意，属性名称也已更改。 例如，用于在 IBM MQ classes for Jakarta Messaging 中启用 IBM MQ 扩展的属性为 **com.ibm.mq.jakarta.jms.SupportMQExtensions**。

独立于 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的属性名称 (例如各种 **com.ibm.msg.client.commonservices.trace.*** 属性) 同样适用于这两个产品。

管理实用程序

crtmqenv 和 **setmqenv** 实用程序现在接受一个选项，以指定应该为 IBM MQ classes for JMS (`-j 2.0`) 还是 IBM MQ classes for Jakarta Messaging (`-j 3.0`) 配置类路径，并且存在称为 **runjms30** 的 **runjms** 实用程序的 IBM MQ classes for Jakarta Messaging 变体和类似名称。

当请求报告 Java 组件时， **dspmqver** 实用程序在其输出中包含 IBM MQ classes for Jakarta Messaging 。

要配置要通过 JNDI 检索的 IBM MQ classes for Jakarta Messaging 对象，新的 **JMS30Admin** 实用程序等同于 IBM MQ classes for JMS 的 **JMSAdmin** 实用程序。

请注意，由于底层对象来自不同的包。 由 **JMSAdmin** 创建的 JNDI 定义不能由 IBM MQ classes for Jakarta Messaging 使用，由 **JMS30Admin** 创建的 JNDI 定义也不能由 IBM MQ classes for JMS 使用。

注: 不支持 IBM MQ Explorer 提供的 IBM MQ classes for Jakarta Messaging 对象; 其 JNDI 集成仅适用于 IBM MQ classes for JMS 。

相关概念

为什么要将 IBM MQ 类用于 Jakarta Messaging?

JMS 和 Jakarta Messaging 模型

JMS 和 Jakarta Messaging 模型定义了一组接口， Java 应用程序可以使用这些接口来执行消息传递操作。 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 是定义 Java 消息传递对象如何与 IBM MQ 概念相关的消息传递提供程序。 JMS 和 Jakarta Messaging 规范期望将某些消息传递对象作为受管对象。

从 IBM MQ 8.0 开始，产品支持 JMS 标准的 JMS 2.0 版本，该版本引入了简化的 API ，同时还保留了 JMS 1.1 中的经典 API。

JM 3.0 IBM MQ 9.3.0 引入了对 Jakarta Messaging 3.0 的支持。 JMS 2.0 仍完全受支持。 由于 JMS 和 Jakarta Messaging 有许多共同之处，因此本主题中对 JMS 的进一步引用可以视为两者的引用。 根据需要突出显示任何差异。

简化的 API

JMS 2.0 引入了简化的 API，同时还保留了 JMS 1.1 中特定于域的接口和独立于域的接口。 简化的 API 减少了发送和接收消息所需的对象数，由以下接口构成：

ConnectionFactory
ConnectionFactory 是供 JMS 客户机用来创建连接的受管对象。 此接口也用于标准 API。

JMS 上下文
此对象结合了标准 API 的 Connection 和 Session 对象。 可以通过复制底层连接来从其他 JMSContext 对象创建 JMSContext 对象。

JMS 生产者
JMSProducer 由 JMSContext 创建，并用于向队列或主题发送消息。 JMSProducer 对象会导致创建发送消息所需的对象。

JMS 使用者
JMSConsumer 由 JMSContext 创建，并用于从主题或队列接收消息。

简化的 API 具有以下影响：

- JMSContext 对象始终自动启动底层连接。
- JMSProducer 和 JMSConsumer 现在可以直接处理消息体，而无需使用消息的 getBody 方法来获取整个消息对象。
- 在发送“消息体”（即消息内容）之前，可以使用方法链在 JMSProducer 对象上设置消息属性。 JMSProducer 将负责创建发送消息所需的所有对象。 通过使用 JMS 2.0，可以按如下所示设置属性和发送消息：

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 还引入了可在多个使用者之间共享消息的共享预订。 所有 JMS 1.1 预订均视为非共享预订。

标准 API

以下列表汇总了标准 API 的主要 JMS 接口：

Destination
目标是应用程序发送消息的位置和/或应用程序从中接收消息的源。

ConnectionFactory
ConnectionFactory 对象封装了连接的一组配置属性。 应用程序使用连接工厂来创建连接。

Connection
Connection 对象将应用程序的活动连接封装到消息传递服务器中。 应用程序使用连接来创建会话。

Session
会话是用于发送和接收消息的单线程上下文。 应用程序使用会话来创建消息、消息生产者和消息使用者。 会话将进行事务处理或不进行事务处理。

消息
Message 对象封装了应用程序发送或接收的消息。

MessageProducer
应用程序使用消息生产者向目标发送消息。

MessageConsumer
应用程序使用消息使用者接收向目标发送的消息。

第 126 页的图 53 显示了这些对象及其关系。

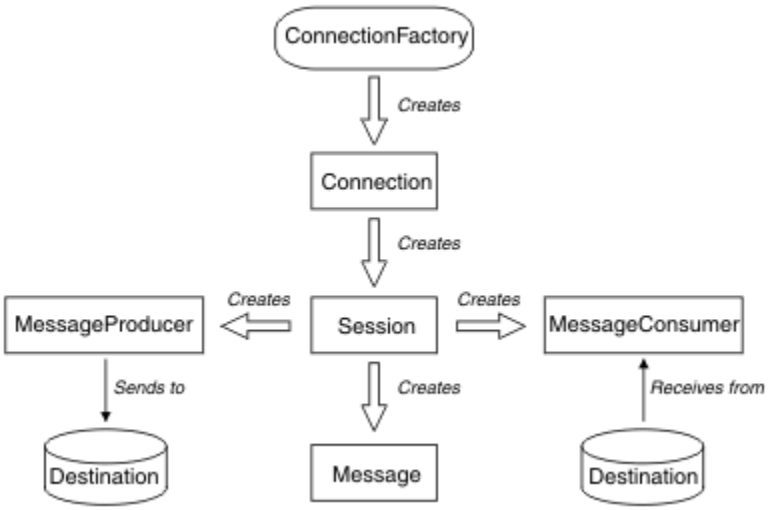


图 *53: JMS* 对象及其关系

此图显示了以下主要接口：ConnectionFactory、Connection、Session、MessageProducer、MessageConsumer、Message 和 Destination。 应用程序使用连接工厂来创建连接，并使用连接来创建会话。 然后，应用程序可以使用会话来创建消息、消息生产者和消息使用者。 应用程序使用消息生产者向目标发送消息，并使用消息使用者接收向目标发送的消息。

Destination、ConnectionFactory 或 Connection 对象可以由多线程应用程序的不同线程并行使用，但 Session、MessageProducer 或 MessageConsumer 对象不能由不同线程并行使用。 确保 Session、MessageProducer 或 MessageConsumer 对象不会被并行使用的最简单方法是为每个线程创建单独的 Session 对象。

JMS 支持以下两种类型的消息传递：

- 点到点消息传递
- 发布/预订消息传递

这两种类型的消息传递也称为消息传递域，您可以在应用程序中结合使用这两种类型的消息传递。 在点到点域中，目标是队列，在发布/预订域中，目标是主题。

对于 JMS 之前的 JMS 1.1 版本，点到点域的编程使用的是一组接口和方法，而发布/预订域的编程则使用的是另一组接口和方法。 二者相似，但相互独立。 从 JMS 1.1 开始，您可以使用一组同时支持这两种消息传递域的通用接口和方法。 通用接口为每个消息传递域提供独立于域的视图。 第 126 页的表 17 列出了独立于 JMS 域的接口及对应的域特定接口。

表 *17:* 独立于 *JMS* 域的接口及其相应的特定于域的接口

独立于域的接口	**点到点域的域特定接口**	**发布/预订域的域特定接口**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	队列	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 支持较早的特定于 JMS 1.1 域的接口和 JMS 2.0 的简化 API。 因此， IBM MQ classes for JMS 2.0 可用于维护现有应用程序，包括在现有应用程序中开发新功能。

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 支持相同接口的 Jakarta Messaging 版本，建议用于新的应用程序开发。

在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中， JMS 对象通过以下方式与 IBM MQ 概念相关:

- Connection 对象具有派生自连接工厂（用于创建连接）属性的属性。 这些属性控制应用程序如何连接到队列管理器。 这些属性的示例包括队列管理器名称以及运行队列管理器的系统的主机名或 IP 地址（针对以 CLIENT 方式连接到队列管理器的应用程序）。
- Session 对象封装了 IBM MQ 连接句柄，因此定义了会话的事务范围。
- MessageProducer 对象和 MessageConsumer 对象各自封装了一个 IBM MQ 对象句柄。

使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 时，将应用 IBM MQ 的所有正常规则。 尤其需要注意，应用程序可以向远程队列发送消息，但只能从应用程序所连接到的队列管理器拥有的队列中接收消息。

JMS 规范认为 ConnectionFactory 和 Destination 对象都是受管对象。 管理员在中央存储库中创建和维护受管对象，而 JMS 应用程序可使用 Java 命名和目录接口 (JNDI) 来检索这些对象。

在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中， Destination 接口的实现是 Queue 和 Topic 的抽象超类，因此 Destination 实例是 Queue 对象或 Topic 对象。 独立于域的接口将队列或主题视为目标。 MessageProducer 或 MessageConsumer 对象的消息传递域由目标是队列还是主题来确定。

因此，在 IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 中，以下类型的对象可以是受管对象:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- 队列
- Topic
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

IBM MQ classes for JMS/Jakarta Messaging 体系结构

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 具有分层体系结构。 最上层代码是任何 IBM Java 消息传递提供程序都可以使用的公共层。

JM 3.0 IBM MQ 9.3.0 引入了对 Jakarta Messaging 3.0 的支持。 JMS 2.0 仍完全受支持。

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 具有分层体系结构，如图 第 128 页的图 54 所示。 最上层代码是可供任何 IBM JMS 或 Jakarta Messaging 提供程序使用的公共层。 当应用程序调用 JMS 或 Jakarta Messaging 方法时，不特定于消息传递系统的调用的任何处理都由公共层执行，这也会提供对调用的一致响应。 对特定于消息传递系统的调用的任何处理将委派到下一层。 在下图中，IBM MQ 消息传递提供者以及另外两个消息传递提供者（消息传递提供者 A 和消息传递提供者 B）都显示在下一层中。

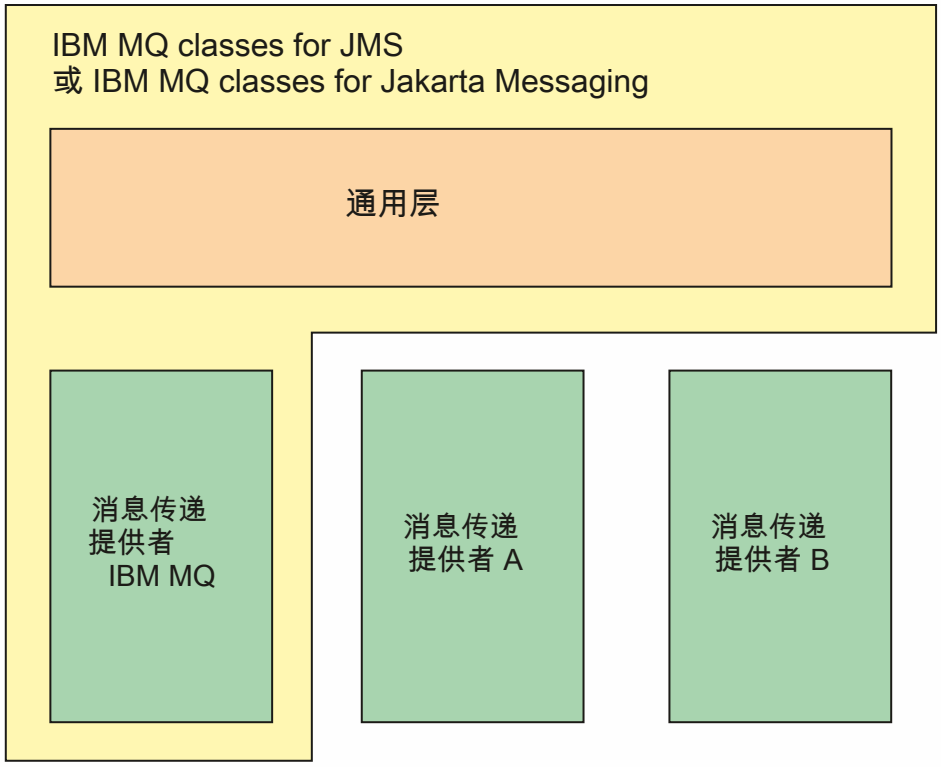


图 *54: IBM JMS* 和 *Jakarta Messaging* 提供程序的分层体系结构

分层体系结构实现了以下目标:

- 提高各种 IBM JMS 和 Jakarta Messaging 提供程序的行为一致性
- 更容易编写两个 IBM 消息传递系统之间的桥接应用程序
- 为了更轻松地将应用程序从一个 IBM JMS 或 Jakarta Messaging 提供程序移植到另一个提供程序

相关任务

使用 IBM MQ classes for JMS/Jakarta Messaging

受管对象支持

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 支持使用受管对象。

JM 3.0 从 IBM MQ 9.3.0 开始， Jakarta Messaging 3.0 支持用于开发新应用程序。 IBM MQ 9.3.0 和更高版本继续支持现有应用程序的 JMS 2.0 。 不支持在同一应用程序中同时使用 Jakarta Messaging 3.0 API 和 JMS 2.0 API。 有关更多信息，请参阅 使用 IBM MQ classes for JMS/Jakarta Messaging。

JMS 或 IBM MQ classes for Jakarta Messaging 应用程序中的逻辑流以 ConnectionFactory 和 Destination 对象开头。 应用程序使用 ConnectionFactory 对象来创建 Connection 对象，该对象表示从应用程序到消息传递服务器的活动连接。 应用程序使用 Connection 对象来创建 Session 对象，它是用于生成和使用消息的单线程上下文。 然后，应用程序可以使用 Session 对象和 Destination 对象来创建 MessageProducer 对象，应用程序使用该对象将消息发送到指定的目标。 目标是消息传递系统中的队列或主题，并由 Destination 对象封装。 应用程序还可以使用 Session 对象和 Destination 对象来创建 MessageConsumer 对象，应用程序使用该对象来接收已发送到指定目标的消息。

JMS 和 Jakarta Messaging 规范期望 ConnectionFactory 和 Destination 对象是受管对象。 管理员在中央存储库中创建和维护受管对象， JMS 或 Jakarta Messaging 应用程序使用 Java Naming Directory Interface (JNDI) 检索这些对象。 受管对象的存储库可以从简单文件到轻量级目录访问协议 (LDAP) 目录。

IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 支持使用受管对象。 应用程序可以使用通过 IBM MQ 公开的 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 的所有功能，而不

必将任何特定于 IBM MQ 的信息硬编码到应用程序本身中。 这种安排使应用程序在一定程度上独立于底层 IBM MQ 配置。

要实现此独立性，应用程序可以使用 JNDI 来检索存储为受管对象的连接工厂和目标，并仅使用 `javax.jms` (JMS 2.0) 或 `jakarta.jms` (Jakarta Messaging 3.0) 包中定义的接口来执行消息传递操作。

JMS 2.0 对于 JMS 2.0，管理员可以使用 IBM MQ JMS 管理工具 **JMSAdmin** 或 IBM MQ Explorer 在中央存储库中创建和维护受管对象。

JM 3.0 对于 Jakarta Messaging 3.0，无法使用 IBM MQ Explorer 来管理 JNDI。 **JMSAdmin** 的 Jakarta Messaging 3.0 变体 (即 **JMS30Admin**) 支持 JNDI 管理。

应用程序服务器通常为受管对象提供自己的存储库，并提供自己的用于创建和维护对象的工具。 因此，Java EE JM 3.0 或 Jakarta EE 应用程序可以使用 JNDI 从应用程序服务器存储库或中央存储库检索受管对象。

相关任务

配置 JMS 和 Jakarta Messaging 资源

Java EE 和 Jakarta EE 平台上受支持的通信类型

在 Java EE 和 Jakarta EE 平台上， IBM MQ classes for JMS 和 IBM MQ classes for Jakarta Messaging 支持应用程序组件与 IBM MQ 队列管理器之间的两种类型的通信。

JM 3.0 IBM MQ 9.3.0 引入了对 Jakarta Messaging 3.0 的支持。 JMS 2.0 仍完全受支持。 由于 JMS 和 Jakarta Messaging 有许多共同之处，因此本主题中对 JMS 的进一步引用可以视为两者的引用。 根据需要突出显示任何差异。

在应用程序组件和 IBM MQ 队列管理器之间支持以下两种类型的通信：

- 出站通信
- 入站通信

出站通信

通过直接使用 JMS 或 Jakarta Messaging API，应用程序组件将创建与队列管理器的连接，然后发送和接收消息。

例如，应用程序组件可以是应用程序客户机、servlet、Java Server Page (JSP)、企业 Java bean (EJB) 或消息驱动的 bean (MDB)。 在此类型的通信中，应用程序服务器容器在消息传递操作支持中仅提供低级功能，例如连接池和线程管理。

入站通信

如果使用的是入站通信，那么到达目标的消息将传递到 MDB，然后由 MDB 负责处理该消息。

Java EE JM 3.0 和 Jakarta EE 应用程序使用 MDB 异步处理消息。 MDB 用作 JMS 消息侦听器，并由用于定义消息处理方式的 onMessage() 方法实现。 已将 MDB 部署到应用程序服务器的 EJB 容器中。 MDB 的精确配置方式取决于所使用的应用程序服务器，但配置信息必须指定要连接到的队列管理器、如何连接到队列管理器、要监视哪个目标中的消息以及 MDB 的事务行为。 然后，MDB 容器将使用这些信息。 当满足 MDB 选择条件的消息到达指定目标时， EJB 容器使用 IBM MQ classes for JMS 或 IBM MQ classes for Jakarta Messaging 从队列管理器检索消息，然后通过调用其 onMessage() 方法将消息传递到 MDB。

与 IBM MQ classes for Java 的关系

IBM MQ classes for Java， IBM MQ classes for Jakarta Messaging 和 IBM MQ classes for JMS 是使用 MQI 的公共 Java 接口的同级。

第 130 页的图 55 显示了 IBM MQ classes for JMS， IBM MQ classes for Jakarta Messaging 和 IBM MQ classes for Java 之间的关系。

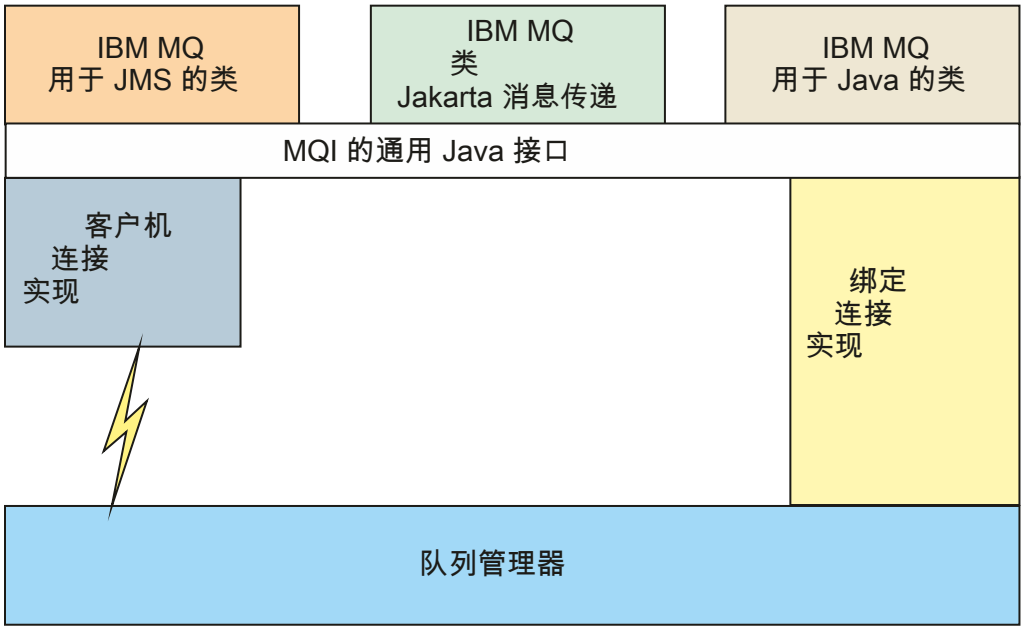


图 55: *IBM MQ classes for JMS*， *IBM MQ classes for Jakarta Messaging* 和 *IBM MQ classes for Java* 之间的关系

通常， Java 程序应仅使用一个接口与 IBM MQ - IBM MQ classes for Java， IBM MQ classes for Jakarta Messaging 或 IBM MQ classes for JMS 进行接口连接。 不支持混合接口，但有一个例外。 为保持与 IBM WebSphere MQ 7.0 之前的发行版的兼容性，用 Java 编写的通道出口类仍可以使用 IBM MQ classes for Java 接口，即使从 IBM MQ classes for JMS 调用这些通道出口类也是如此。 但是，使用 IBM MQ classes for Java 接口意味着应用程序仍依赖于下列其中一项:

- JMS 2.0 IBM MQ classes for Java JAR 文件 `com.ibm.mq.jar`。 如果不想在类路径中包含 `com.ibm.mq.jar`，可以改为使用 `com.ibm.mq.exits` 包中的接口集。
- JM 3.0 与 IBM MQ classes for Jakarta Messaging 进行互操作时使用 `com.ibm.mq.jakarta.client.jar`。

相关概念

为什么要将 IBM MQ 类用于 Jakarta Messaging?

为什么要使用 IBM MQ classes for JMS?

为什么应该将 IBM MQ 类用于 Java?

IBM MQ 消息传递提供程序

IBM MQ 消息传递提供者具有三种运行方式：正常方式、带有限制的正常方式和迁移方式。

IBM MQ 消息传递提供者具有以下三种运行方式:

- IBM MQ 消息传递提供者正常方式
- IBM MQ 消息传递提供者带有限制的正常方式
- IBM MQ 消息传递提供者迁移方式

IBM MQ 消息传递提供者正常方式使用 IBM MQ 队列管理器的所有功能来实现 JMS。 此方式已优化为使用 JMS 2.0 JM 3.0 或 Jakarta Messaging 3.0 API 和功能。

如果:

- 客户机在 **ConnectionFactory** 上指定 6 的提供者版本，客户机的行为方式与随 IBM WebSphere MQ 6.0 提供的客户机兼容。 仅支持 JMS 1.1 和 JMS 2 接口，但某些 JMS 2 功能 (例如共享预订，交付延迟和异步发送) 已禁用。 没有连接共享。
- 客户机在 **ConnectionFactory** 上指定 7 的提供程序版本，完全支持 JMS 1.1 和 JMS 2 接口。

- 未指定提供程序版本，尝试与提供程序版本 7 连接。 如果此操作失败，那么将对提供程序版本 6 进行进一步尝试。

如果要使用 IBM MQ Enterprise Transport 连接到 IBM Integration Bus ，请使用迁移方式。 如果使用 IBM MQ Real-Time Transport，那么会自动选择迁移方式，因为您已在连接工厂对象中显式选择了相应属性。 使用 IBM MQ Enterprise Transport 与 IBM Integration Bus 的连接遵循 配置 JMS **PROVIDERVERSION** 属性中描述的方式选择的一般规则。

相关任务

配置 JMS 资源

z/OS IBM MQ for z/OS concepts

Some of the concepts used by IBM MQ for z/OS are unique to the z/OS platform. For example, the logging mechanism, the storage management techniques, unit of recovery disposition, and queue sharing groups are provided only with IBM MQ for z/OS. Use this topic as an introduction to further information about these concepts.

The concepts include an overview of the objects that IBM MQ for z/OSuses, including:

- The queue manager
- The channel initiator
- Shared queues and queue sharing groups
- Intra-group queuing

The following topics also cover various procedures you need, including:

- System definitions on z/OS
- Storage management
- Recovery and restart
- Security concepts in IBM MQ for z/OS

Related concepts

“The queue manager on z/OS” on page 132
Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

“The channel initiator on z/OS” on page 133
The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

“Terms and tasks for managing IBM MQ for z/OS” on page 135
Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

“Shared queues and queue sharing groups” on page 137
You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

“Intra-group queuing” on page 181
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

“Storage management on z/OS” on page 194
IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

“Logging in IBM MQ for z/OS” on page 198
IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

"Recovery and restart on z/OS" on page 219
Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

"Security concepts in IBM MQ for z/OS" on page 235
Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

"Availability on z/OS" on page 241
IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

"Unit of recovery disposition on z/OS" on page 246
Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Related reference

"System definition on z/OS" on page 209
IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

"Monitoring and statistics on IBM MQ for z/OS" on page 245
IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

z/OS The queue manager on z/OS

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

The queue manager

A *queue manager* is a program that provides messaging services to applications. Applications that use the Message Queue Interface (MQI) can put messages on queues and get messages from queues. The queue manager ensures that messages are sent to the correct queue or are routed to another queue manager. The queue manager processes both the MQI calls that are issued to it, and the commands that are submitted to it (from whatever source). The queue manager generates the appropriate completion codes for each call or command.

The resources managed by the queue manager include:

- Page sets that hold the IBM MQ object definitions and message data
- Logs that are used to recover messages and objects in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the IBM MQ API
- The IBM MQ channel initiator, which allows communication between IBM MQ on your z/OS system and other systems

The queue manager has a name, and applications can connect to it using this name.

Figure 56 on page 133 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

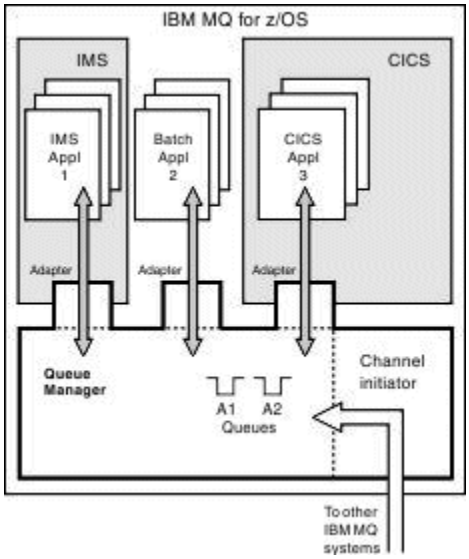


Figure 56. Overview of IBM MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, IBM MQ runs as a z/OS subsystem that is started at IPL time. In the subsystem, the queue manager is started by executing a JCL procedure that specifies the z/OS data sets that contain information about the logs, and that hold object definitions and message data (the page sets). The subsystem and the queue manager have the same name, of up to four characters. All queue managers in your network must have unique names, even if they are on different systems, sysplexes, or platforms.

z/OS The channel initiator on z/OS

The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In IBM MQ for z/OS, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as a z/OS address space under the control of the queue manager. There can be only a single channel initiator connected to a queue manager and it is run inside the same z/OS image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 57 on page 134 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX and Windows are placed on the local queue, from where they are retrieved by an application. Reply messages are returned by a similar route.

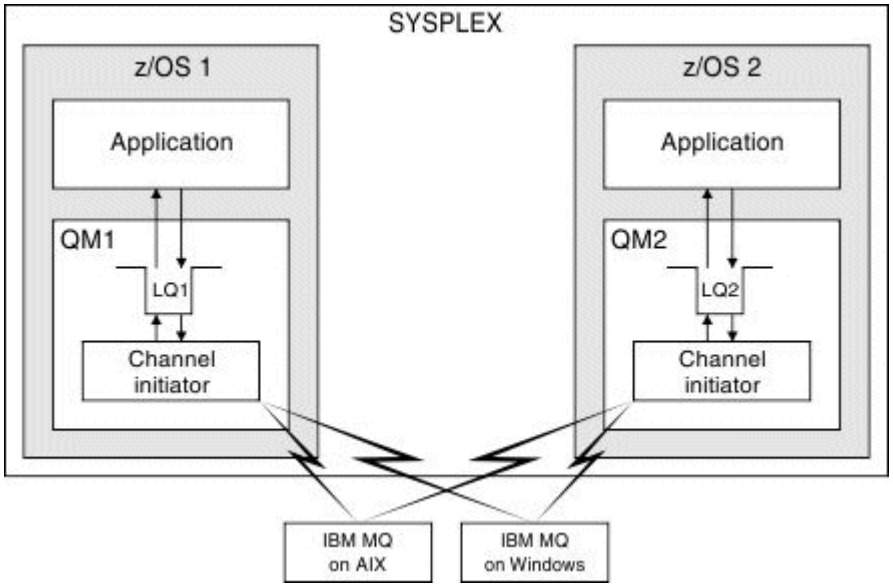


Figure 57. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These processes include:

Listeners
These processes listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor
This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server
This is used to resolve TCP names into addresses.

TLS tasks
These are used to perform encryption and decryption and check certificate revocation lists.

z/OS SMF records for the channel initiator

The channel initiator (CHINIT) can produce SMF statistics records and accounting records with information on tasks and channels.

The CHINIT can produce SMF statistics records and accounting records with the following types of information:

- The tasks: dispatcher, adapter, Domain Name Server (DNS), and SSL. These tasks form what is called CHINIT statistics.
- Channels: provides accounting information similar to that available with the DIS CHSTATUS command. This is called channel accounting.

IBM MQ for Multiplatforms provides similar information by writing PCF messages to the SYSTEM.ADMIN.STATISTICS.QUEUE. See Channel statistics message data for further information on how statistics information is recorded on IBM MQ for Multiplatforms.

Statistics data

You can use this information to find out the following information:

- Whether you need more of the CHINIT tasks, such as number of SSL TCBs and how much CPU is used by these tasks.

- The average time for requests on these tasks.
- The longest duration request in the interval, and the time of day this occurred, for DNS and SSL tasks. You can correlate this time of day with problems you may experience with the channel.

Accounting data

You can use this information to monitor channel usage and find out the following information:

- The channels with the highest throughput.
- The rate at which messages were sent, and the rate of sending data in MB/second.
- The achieved batch size. If the achieved batch size is close to the batch size specified for the channel, the channel might be close to its limit for sending messages.

You use the START TRACE and STOP TRACE commands to control the collection of the accounting trace and the statistics trace. You can use the STATCHL and STATACLS options on the channel and queue manager to control whether channels produce SMF data.

z/OS Terms and tasks for managing IBM MQ for z/OS

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

Some of the terms and tasks required for managing IBM MQ for z/OS are specific to the z/OS platform. The following list contains some of these terms and tasks.

- Shared queues
- Page sets and buffer pools
- Logging
- Tailoring the queue manager environment
- Restart and recovery
- Security
- Availability
- Manipulating objects
- Monitoring and statistics
- Application environments

Shared queues

Queues can be *non-shared*, owned by and accessible to only one queue manager, or *shared*, owned by a *queue sharing group*. A queue sharing group consists of a number of queue managers, running within a single z/OS sysplex, that can access the same IBM MQ object definitions and message data concurrently. Within a queue sharing group, the shareable object definitions are stored in a shared Db2 database. The shared queue messages are held inside one or more coupling facility structures (CF structures). If the message data is too large to store directly in the structure (more than 63 KB in size), or if the message is large enough that installation-defined rules select it for offloading, the message control information is still stored in the coupling facility entry, but the message data is offloaded to a shared message data set (SMDS) or to a shared Db2 database. The shared message data sets, the shared Db2 database, and the coupling facility structures are resources that are jointly managed by all of the queue managers in the group.

Pages sets and buffer pools

When a message is put on to a non-shared queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation gets a message from the same queue. If

the message is removed from the queue, space in the page set that holds the data is later freed for reuse. As the number of messages held on a queue increases, so the amount of space used in the page set increases, and as the number of messages on a queue reduces, the space used in the page set reduces.

To reduce the performance cost of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through IBM MQ objects called *buffer pools*.

For more information about page sets and buffer pools, see Storage management.

Logging

Any changes to objects held on page sets, and operations on persistent messages, are recorded as log records. These log records are written to a log data set called the *active log*. The name and size of the active log data set is held in a data set called the *bootstrap data set* (BSDS).

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an *archive log* data set. Information about these actions, including the name of the archive log data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is offloaded to an archive log, and the active log data set is available for reuse.

For more information about the log and bootstrap data sets, see “Logging in IBM MQ for z/OS” on page 198.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager operates are read. In addition, data sets containing IBM MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for IBM MQ to run, and you can tailor these to define or initialize the IBM MQ objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in Db2.

For more information about initialization parameters and system objects, see “System definition on z/OS” on page 209.

Recovery and restart

At any time during the operation of IBM MQ, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set that is the least recently used by a background task within the queue manager.

If the queue manager terminates abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that IBM MQ can recover persistent message data and object changes right up to the point of failure.

If a queue manager that is a member of a queue sharing group encounters a coupling facility failure, the persistent messages on that queue can be recovered only if you have backed up your coupling facility structure.

For more information about recovery and restart, see “Recovery and restart on z/OS” on page 219.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that IBM MQ owns and manages from access by unauthorized users. You can also use Transport Layer Security (TLS) for channel security. TLS is included as part of the IBM MQ product.

For more information about IBM MQ security, see “Security concepts in IBM MQ for z/OS” on page 235.

Availability

There are several features of IBM MQ that are designed to increase system availability in the event of queue manager or communications subsystem failure. For more information about these features, see “Availability on z/OS” on page 241.

Manipulating objects

When the queue manager is running, you can manipulate IBM MQ objects either through a z/OS console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms enable you to define, alter, or delete IBM MQ objects. You can also control and display the status of various IBM MQ and queue manager functions.

For more information about these facilities, see Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS.

You can also manipulate IBM MQ objects using the IBM MQ Explorer, a graphical user interface that provides a visual way of working with queues, queue managers, and other objects.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

For more information about these facilities, see “Monitoring and statistics on IBM MQ for z/OS” on page 245.

Application environments

When the queue manager has started, applications can connect to it and start using the IBM MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. IBM MQ applications can also access applications on CICS and IMS systems that are not aware of IBM MQ, using the CICS and IMS bridges.

For more information about these facilities, see “IBM MQ and other z/OS products” on page 248.

For information about writing IBM MQ applications, see the following documentation:

- Developing applications
- Using C++
- Using IBM MQ classes for Java

z/OS Shared queues and queue sharing groups

You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

This section describes the attributes and benefits, and offers information about how several queue managers can share the same queues and the messages on those queues.

What is a shared queue?

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

A queue sharing group

The queue managers that can access the same set of shared queues form a group called a *queue sharing group*.

Any queue manager can access messages

Any queue manager in the queue sharing group can access a shared queue. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue sharing group that does not require channels to be active between queue managers.

IBM MQ supports the offloading of messages to Db2 or a shared message data set (SMDS). The offloading of messages of any size is configurable.

Figure 58 on page 138 shows three queue managers and a coupling facility, forming a queue sharing group. All three queue managers can access the shared queue in the coupling facility.

An application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue sharing group can continue processing the queue if one of the queue managers has a problem.

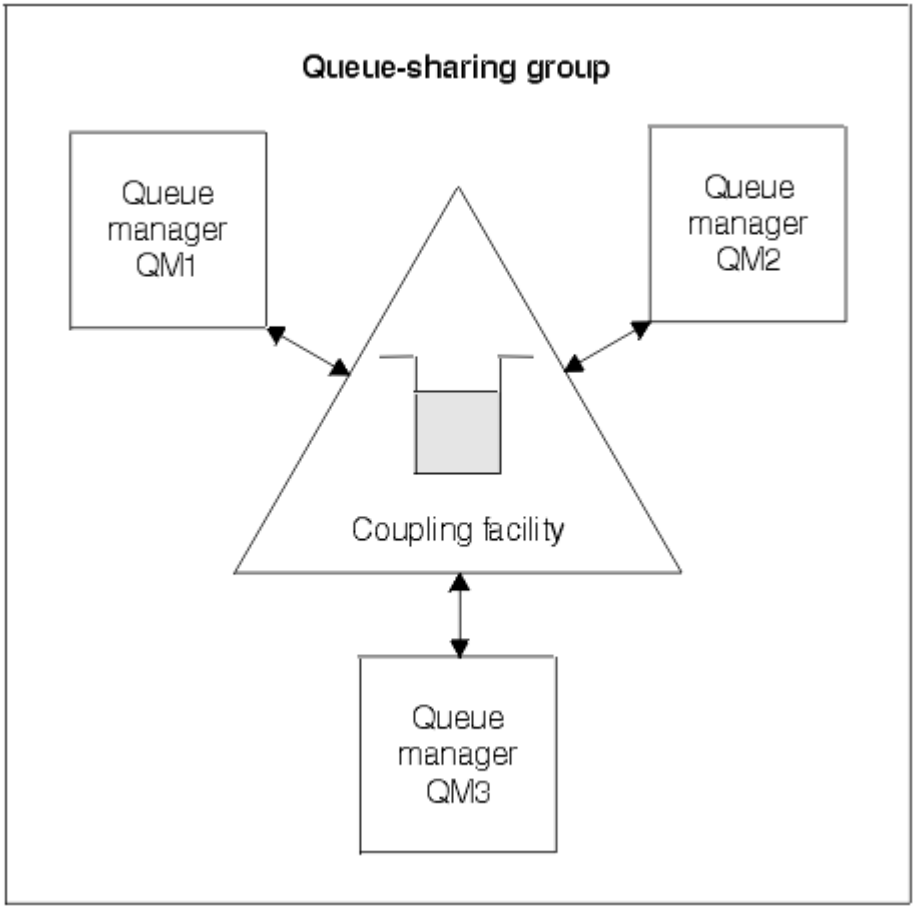


Figure 58. A queue sharing group

Queue definition is shared by all queue managers

Shared queue definitions are stored in the Db2 database table OBJ_B_QUEUE. Because of this, you need to define the queue only once and then it can be accessed by all the queue managers in the queue sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in Page sets).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name exists.

What is a queue sharing group?

A group of queue managers that can access the same shared queues is called a queue sharing group. Each member of the queue sharing group has access to the same set of shared queues.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Figure 59 on page 139 illustrates a queue sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue sharing group must also connect to a Db2 system. The Db2 systems must all be in the same Db2 data-sharing group so that the queue managers can access the Db2 shared repository used to hold shared object definitions. These are definitions of any type of IBM MQ object (for example, queues and channels) that are defined only once and then any queue manager in the group can use them. These are called *global* definitions and are described in Private and global definitions.

More than one queue sharing group can reference a particular data-sharing group. You specify the name of the Db2 subsystem and which data-sharing group a queue manager uses in the IBM MQ system parameters at startup.

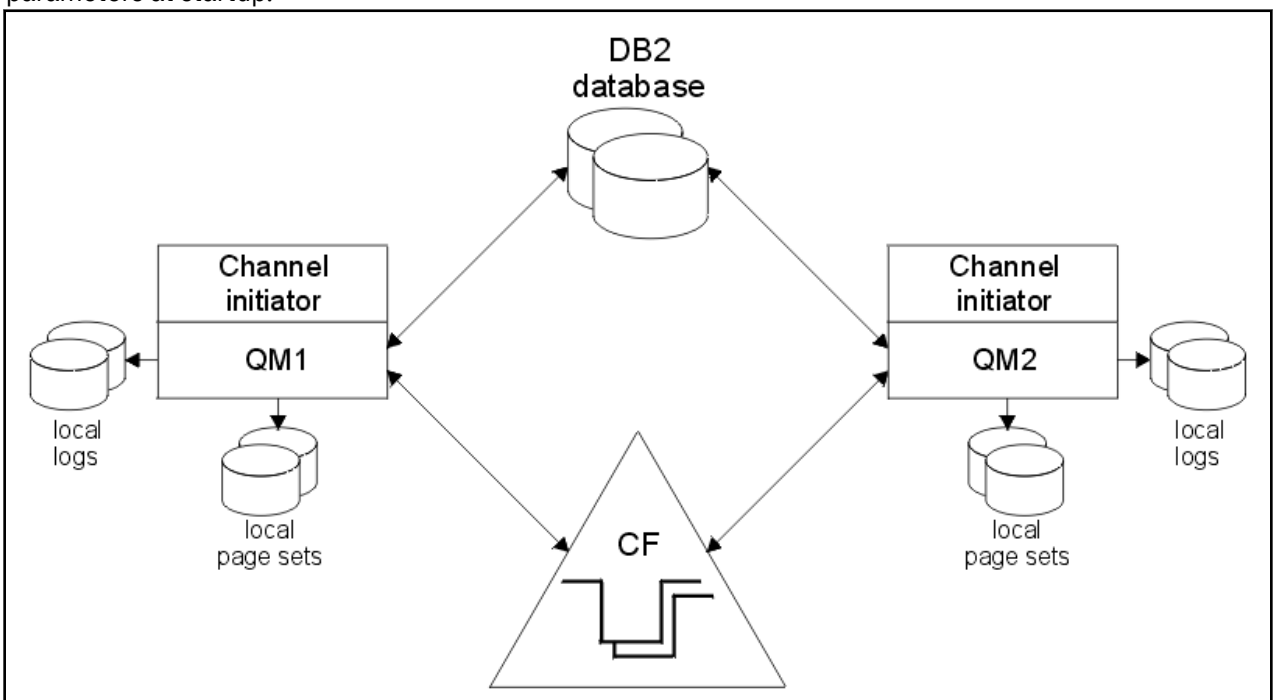


Figure 59. The components of queue managers in a queue sharing group

When a queue manager has joined a queue sharing group, it has access to the shared objects defined for that group, and you can use that queue manager to define new shared objects within the group. If shared queues are defined within the group, you can use this queue manager to put messages to and get messages from those shared queues. Any queue manager in the group can retrieve the messages held on a shared queue.

You can enter an MQSC command once, and have it executed on all queue managers within the queue sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in Directing commands to different queue managers.

When a queue manager runs as a member of a queue sharing group it must be possible to distinguish between IBM MQ objects defined privately to that queue manager and IBM MQ objects defined globally that are available to all queue managers in the queue sharing group. The *queue sharing group disposition* attribute is used for this. This attribute is described in Private and global definitions.

You can define a single set of security profiles that control access to IBM MQ objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to only one queue sharing group, and all queue managers in the group must be in the same sysplex. You specify which queue sharing group the queue manager belongs to in the system parameters at startup.

Related concepts

"Where are shared queue messages held?" on page 140
Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

"Advantages of using shared queues" on page 156
Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

"Distributed queuing and queue sharing groups" on page 175
Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

"Influencing workload distribution with shared queues" on page 179
Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

Related reference

"Where to find more information about shared queues and queue sharing groups" on page 180
Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

z/OS Where are shared queue messages held?

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

If the CF structure has been configured to use System Class Memory (SCM), IBM MQ can use this with no additional configuration.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Shared queue message storage

Messages that are put onto shared queues are not stored on page sets and do not use buffer pools.

The messages in shared queues have entries on list structures in the z/OS coupling facility (CF). Many queue managers in the same sysplex can access those messages using the CF list structure.

The message data for small shared queue messages is normally included in the coupling facility entry. For larger messages, the message data can be stored either in a shared message data set (SMDS), or as one or more binary large objects (BLOBs) in a Db2 table which is shared by a Db2 data sharing group. Message

data exceeding 63 KB is always offloaded to SMDS or Db2. Smaller messages can also optionally be offloaded in the same way to save space in the coupling facility structure. See “Specifying offload options for shared messages” on page 142 for more details.

Messages put on a shared queue are referenced in a coupling facility structure until they are retrieved by an MQGET. Coupling facility operations are used to:

- Search for the next retrievable message
- Lock uncommitted messages on shared queues
- Notify interested queue managers about the arrival of committed messages

MQPUT and MQGET operations on persistent messages are recorded on the log of the queue manager performing that operation. This minimizes the risk of data loss in the event of a coupling facility failure.

The coupling facility

The messages held on shared queues are referenced inside a coupling facility. The coupling facility lies outside any of the z/OS images in the sysplex and is typically configured to run on a different power supply. The coupling facility is therefore resilient to software failures and you can configure it so that it is resilient to hardware failures or power-outages. This means that messages stored in the coupling facility are highly available.

Each coupling facility list structure used by IBM MQ is dedicated to a specific queue sharing group, but a coupling facility can hold structures for more than one queue sharing group. Queue managers in different queue sharing groups cannot share data. Up to 32 queue managers in a queue sharing group can connect to a coupling facility list structure at the same time.

A single coupling facility list structure can contain up to 512 shared queues. The total amount of message data stored in the structure is limited by the structure capacity. However, with **CFLEVEL(5)** you can use the offload parameters to offload data for messages less than 63 KB thereby increasing the number of messages which can be stored in the structure, although each message still requires at least a coupling facility entry plus at least 768 bytes of data, made up of 256 bytes for the entry and 512 bytes for the two elements of header and descriptor.

The size of the list structure is restricted by the following factors:

- It must lie within a single coupling facility.
- It might share the available coupling facility storage with other structures for IBM MQ and other products.

Coupling facility list structures can have storage class memory associated with them. In certain situations this storage class memory can be useful when used with shared queues. See “Use of storage class memory with shared queues” on page 157 for more information.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: IBM MQ for z/OS Capacity planning and tuning supportpac. You can also use the web-based tool CFSizer, which is provided by IBM to assist with CF sizes.

The CF structure object

The queue manager's use of a coupling facility structure is specified in a CF structure (CFSTRUCT) IBM MQ object.

These structure objects are stored in Db2.

When using z/OS commands or definitions relating to a coupling facility structure, the first four characters of the name of the queue sharing group are required. However, an IBM MQ CFSTRUCT object always

exists within a single queue sharing group, and so its name does not include the first four characters of the name of the queue sharing group. For example, CFSTRUCT(MYDATA) defined in queue sharing group starting with SQ03 would use coupling facility list structure SQ03MYDATA.

CF structures have a CFLEVEL attribute that determines their functional capability:

- 1, 2 - can be used for nonpersistent messages less than 63 KB
- 3 - can be used for persistent and nonpersistent messages less than 63 KB
- 4 - can be used for persistent and nonpersistent messages up to 100 MB
- 5 - can be used for persistent and nonpersistent messages up to 100 MB and selectively offloaded to shared message data sets (SMDS) or Db2.

Note: When using IBM MQ you can encrypt a coupling facility structure. See Encrypting coupling facility structure data for more information.

Backup and recovery of the coupling facility

You can back up coupling facility list structures using the IBM MQ command BACKUP CFSTRUCT. This puts a copy of the persistent messages currently within the CF structure onto the active log data set of the queue manager making the backup, and writes a record of the backup to Db2.

If coupling facility fails, you can use the IBM MQ command RECOVER CFSTRUCT. This uses the backup record from Db2 to locate and restore persistent messages from the backup of the CF structure. Any activity since the last backup is replayed using the logs of all the queue managers in the queue sharing group, and the CF structure is then restored up to the point before the failure.

See the BACKUP CFSTRUCT and RECOVER CFSTRUCT commands for more details.

Related concepts

“Specifying offload options for shared messages” on page 142
You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

“Managing your shared message data set (SMDS) environment” on page 144
If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

z/OS *Specifying offload options for shared messages*

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in an IBM MQ managed data set called a *shared message data set* (SMDS).

For messages larger than the coupling facility entry size of 63 KB, offloading message data to a SMDS can have a significant performance improvement compared with offloading to Db2.

Every shared queue message is still managed using a list entry in a coupling facility structure, but when the message data is offloaded to the SMDS, the coupling facility entry only contains some control information and a list of references to the relevant disk blocks where the message is stored. Using this mechanism means the amount of coupling facility element storage required for each message is only a fraction of the actual size of the message.

Selecting where the shared queue messages are stored

The selection of SMDS or Db2 shared message storage is controlled with the **OFFLOAD(SMDS|DB2)** parameter on the **CFSTRUCT** definition. **OFFLOAD(SMDS)** is the default value.

This parameter also requires the **CFSTRUCT** to use **CFLEVEL(5)** or greater.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See DEFINE CFSTRUCT for more details.

OFFLOAD(DB2) is supported primarily for migration purposes.

Selecting which shared queue messages are offloaded

Message data is offloaded to SMDS or Db2 based on the size of the message data, and the current usage of the coupling facility structure. There are three rules, and each rule specifies a matching pair of parameters. These parameters are a corresponding coupling facility structure usage threshold percentage (**OFFLDnTH**) and a message size limit (**OFFLDnSZ**).

The current implementation of the three rules is specified using the following pairs of keywords:

- OFFLD1TH and OFFLD1SZ
- OFFLD2TH and OFFLD2SZ
- OFFLD3TH and OFFLD3SZ

Rule pair	Default value	Description
Rule pair 1	OFFLD1TH(70) and OFFLD1SZ(32K)	If the coupling facility structure is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility structure is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility structure is more than 90% full offload data for messages exceeding 0 KB (all messages)

If an offload rule has the OFFLD *x* SZ value of 64K this indicates that the rule is not in effect. In this case messages will only be offloaded if another offload rule is in effect, or if the message is greater than 63.75 KB and so, too large to store in the structure.

Each message which is offloaded still requires 0.75 KB of storage in the coupling facility.

The three offload rules which can be specified for each structure are intended to be used as follows.

- Performance
  - When there is plenty of space in the application structure, message data should only be offloaded if it is too large to store in the structure, or if it exceeds some lower message size threshold such that the performance value of storing it in the structure is not worth the amount of structure space that it would need.
  - If a specific message size threshold is required, it is conventionally specified using the first offload rule.
- Capacity
  - When there is very little space in the application structure, the maximum amount of message data should be offloaded so as to make the best use of the remaining space.
  - The third offload rule is conventionally used to indicate that when the structure is nearly full, most messages should be offloaded, so the entries in the application structure will be typically of the minimum size (requiring about 0.75K bytes).
  - The usage threshold parameter should be chosen based on the application structure size and the maximum anticipated backlog. For example, if the maximum anticipated backlog is 1M messages, then the amount of structure storage required for this number of messages is about 0.75G bytes. This means for example that if the structure is about 10G bytes, the usage threshold for offloading all messages must be set to 92% or lower.

  - Structure space is divided into elements and entries, and even though there may be enough space overall, one of these may run out before the other. The system provides AUTOALTER capabilities to adjust the ratio when necessary, but this is not very sensitive, so the amount of space actually available may be somewhat less. It may be better therefore to aim to use not more than 90% of the maximum structure space, so in the previous example, the usage threshold for offloading all messages would be better set around 80%.
- Cushioned transition:
  - As the amount of space left in the coupling facility structure decreases, it would be undesirable to have a large sudden change in the performance characteristics. It is also undesirable for coupling facility management to have a sudden threshold change in the typical ratio of entries to elements being used.
  - The second offload rule is conventionally used to provide some intermediate cushion between the performance and capacity biased offload rules. It can be set to cause a significant increase in offload activity when the space used in the coupling facility structure exceeds an intermediate threshold. This means that the remaining space is used up more slowly, and gives the coupling facility automatic alter processing more time to adapt to the higher usage levels.

If the coupling facility structure cannot be expanded, and there is a need to store at least some predetermined number of messages, the third rule can be modified as necessary to ensure that offloading of data for all messages starts at an appropriate threshold to ensure space is reserved for that predetermined number of messages.

For example, if the coupling facility structure size is 4 GB, and the predetermined number of messages is 1 million, then 1,000,000 * 0.75 KB are needed, which is 768 MB, 18.75% of 4 GB. In this case the threshold for offloading all messages needs to be set around 80% rather than 90%. This gives parameters OFFLD3TH(80) and OFFLD3SZ(0K) . The other offload parameters would also need to be adjusted.

If it is found that offloading very small messages has a significant performance impact, but the relative impact is less for larger messages, then the usage thresholds for the other rules can be reduced to offload larger messages earlier, leaving more space in the structure for small messages before they need to be offloaded.

For example, if messages exceeding 32KB occur frequently but the elapsed time performance for offloading them (as determined from RMF statistics or application performance) is very similar to that for keeping them in the coupling facility, then the threshold for the first rule could be set to 0% to offload all such messages. This gives parameters OFFLD1TH(0) and OFFLD1SZ(32K). Again the other offload parameters would need to be adjusted.

If there are many messages around specific intermediate sizes, such as 16 KB and 6 KB, then it might be useful to change the message size option for the second rule so that the larger ones get offloaded at a fairly low usage threshold, saving a significant amount of space, but the smaller ones still get stored only in the coupling facility.

z/OS *Managing your shared message data set (SMDS) environment*

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

SMDS objects

The properties and status of each shared message data set are tracked in a shared SMDS object which can be updated through any queue manager in the queue sharing group.

There is one shared message data set for each queue manager that can access each coupling facility application structure. The shared message data set is identified by the owning queue manager name, specified using the SMDS keyword, and by the application structure name, specified using the CFSTRUCT keyword.

Note: When defining SMDS data sets for a structure, you must have one for each queue manager.

The SMDS object is stored in an array (with one entry per queue manager in the group) which forms an extension of the corresponding CFSTRUCT object stored in Db2.

There is no command to DEFINE or DELETE the SMDS object because it is created or deleted as part of the CFSTRUCT object, but there is a command to ALTER it to change settings for an individual owning queue manager.

For further information on SMDS commands, see "SMDS related commands" on page 155

SMDSCONN information

It is possible for a shared message data set to be in a normal state, but for one or more queue managers to be unable to connect to it, for example because of a problem with a security definition or with direct access device connectivity. It is therefore necessary for each queue manager to keep track of connection status, and availability information for each shared message data set, indicating for example whether it can currently connect to it, and if not why not.

The SMDSCONN information represents a queue manager connection to a shared message data set. As for the shared message data set itself, it is identified by the queue manager which owns the shared message data set (as specified on the SMDS keyword for the shared object itself) combined with the CFSTRUCT name.

There is no parameter to identify the connecting queue manager because commands addressed to a specific queue manager can only refer to SMDSCONN information for that same queue manager.

The SMDSCONN information entries are maintained in main storage in the owning queue manager, and are re-created when the queue manager is restarted. However, if a connection from an individual queue manager has been explicitly stopped, this information is also stored as a flag in a connection array in the corresponding CFSTRUCT or SMDS object, so that it persists across a queue manager restart.

Status and availability information

Status information indicates the state of a resource or connection (for example, whether it is not yet being used, is in normal use or is in need of recovery). It is usually described using the STATUS keyword. The possible values depend on the type of object.

Status information is normally updated automatically, for example when an error is detected while using the resource or connection. However, in some cases a command can also be used to update the status, to allow for cases when it is not possible for a queue manager to determine the correct status automatically.

Availability information indicates whether the resource or connection can be used, and is usually primarily determined by the status information. For the resource or connection types used in shared message data set support, three levels of availability are implemented:

Available
: This means that the resource is available to be used normally. This does not necessarily mean that it is in use at present (which can be determined instead from the STATUS value). For a data set, if it requires restart processing, this allows the owning queue manager to open it, but other queue managers must wait until the data set is back in the ACTIVE state.

Unavailable because of error
: This means that the resource has been made unavailable automatically because of an error and is not expected to be available again until some form of repair or recovery processing has been performed. However, attempts to make it available again are permitted without operator intervention. Such an attempt can also be triggered by a command to mark the resource as enabled, or a command which changes the status in such a way as to indicate that recovery processing has been completed.

  The reason that the resource has been made unavailable is normally obvious from the related STATUS value, but in some cases there may be other reasons to make the resource unavailable, in which case a separate REASON value is provided to indicate the reason.

Unavailable because of operator command
This means that access to the resource has been explicitly disabled by a command. It can only be made available by using a command to enable it again.

SMDS availability

For the shared SMDS object, the availability is described by the ACCESS keyword, with the possible values ENABLED, SUSPENDED and DISABLED.

The availability can be updated using a **RESET SMDS** command for the relevant shared object from any queue manager in the group to set ACCESS(ENABLED) or ACCESS(DISABLED).

If the availability was previously ACCESS(SUSPENDED), changing it to ACCESS(ENABLED) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to ACCESS(SUSPENDED).

SMDSCONN availability

For a local SMDSCONN information entry, the availability is described by the AVAIL keyword, with the possible values NORMAL, ERROR or STOPPED. The availability can be updated using a **START SMDSCONN** or **STOP SMDSCONN** command addressed to a specific queue manager to enable or disable its connection.

If the availability was previously AVAIL(ERROR), changing it to AVAIL(NORMAL) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to AVAIL(ERROR).

Shared message data set shared status and availability

The availability of each shared message data set is managed within the group using shared status information, which can be displayed using the **DISPLAY CFSTATUS** command with TYPE(SMDS). This displays status information for each queue manager that has activated a data set for each structure. Each data set can be in one of the following states:

NOTFOUND
This means that the corresponding data set has not yet been activated. This status only appears when a specific queue manager is specified, as data sets which have not been activated are skipped when all queue managers are selected.

NEW
The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE
This means that the data set is fully available and should be allocated and opened by all active queue managers for the structure.

FAILED
This means the data set is not available at all (except for recovery processing) and must be closed and deallocated by all queue managers.

INRECOVER
This means that media recovery (using RECOVER CFSTRUCT) is in progress for this data set.

RECOVERED
This indicates that a command has been issued to switch a failed data set back to the active state, but further restart processing is required which is not yet complete, so the data set can only be opened by the owning queue manager for restart processing.

EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager, at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be

replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

The command output includes the date and time at which recovery logging was enabled, if any, and the date and time at which the data set failed, if it is not currently active.

A shared message data set can be put into a FAILED state either by a **RESET SMDS** command or automatically when any of the following types of error are detected:

- The data set cannot be allocated or opened by the owning queue manager.
- Validation of the data set header fails after it has been successfully opened by any queue manager.
- A permanent I/O error occurs when the owning queue manager is reading or writing data.
- A permanent I/O error occurs when another queue manager is reading data from a data set which had successfully completed open processing and validation.

When a data set is in the FAILED or INRECOVER state, it not available for normal use, so if the availability state is ACCESS(ENABLED) it is changed to ACCESS(SUSPENDED).

If a data set has been put into the the FAILED state but no media recovery is required, for example because the data was still valid but the storage device was temporarily offline, then the **RESET SMDS** command can be used to request changing the status directly to the RECOVERED state.

When the data set enters the RECOVERED state, either on completion of recovery processing or as a result of the **RESET SMDS** command, then it is ready to be used again once restart processing has been completed. If it was in the ACCESS(SUSPENDED) state, it is automatically switched back to the ACCESS(ENABLED) state, which allows the owning queue manager to perform restart processing. When restart processing completes, the state is changed to ACTIVE and all other queue managers can then connect to the data set again.

Shared message data set connection status and availability

Each queue manager maintains local status and availability information for its connection to each shared message data set owned by itself and by other queue managers in the group. This information can be displayed using the **DISPLAY SMDSCONN** command.

If it is unable to access a shared message data set in the ACTIVE state which belongs to another queue manager it flags the connection as being unavailable from its own point of view.

If the error definitely indicates a problem with the data set itself, the queue manager also automatically changes the shared status to indicate that the data set is now in a FAILED state. However, if the error could be caused by an environmental problem, such as not being authorized to open the data set, the queue manager issues error messages and treats the data set as being unavailable, but it does not modify the shared data set status. If the environmental error turns out to be a problem with the data set anyway (for example it has been allocated on a device which cannot be accessed by some of the queue managers) then an operator can use the RESET SMDS command specifying STATUS(FAILED) to allow the data set to be recovered or repaired as necessary.

If a connection to a shared message data set could not be established but the data set appears to be valid, a new attempt to use it can be triggered by issuing a **START SMDSCONN** command for the owning queue manager.

If there is an operational need to terminate the connection between a specific queue manager and a data set temporarily, but the data set itself is not damaged, then the data set can be closed and deallocated using the **STOP SMDSCONN** command. If the data set is in use, the queue manager will close it normally (although any requests for data in that data set will be rejected with a return code). If it is the owned data set, the queue manager will save the space map during CLOSE processing, avoiding the need for restart processing.

If a data set needs to be taken out of service temporarily from all queue managers (for example to move it) but is not damaged, then it is best to use **STOP SMDSCONN** for the relevant data set with the option CMDSCOPE(*) to stop the queue managers using it first, as this will avoid the need for restart processing when the data set is brought back into service. In contrast, if the data set is marked as FAILED this tells

queue managers that they must stop using it immediately, which means that the space map will not be saved and will need to be rebuilt by restart processing.

Access to any shared message data sets previously in the ACCESS(SUSPENDED) state will be retried if the queue manager is restarted.

Shared message data set recovery logging

Persistent shared messages are logged for media recovery purposes. This means that the messages can be recovered after any failure of coupling facility structures or shared message data sets, provided that the recovery logs are still intact. Persistent messages can also be re-created from the recovery logs at another site for disaster recovery purposes.

When the message data is written to a shared message data set, each block written to the data set is logged separately followed by the message entry (including the data map) as written to the coupling facility. The recovery process always recovers the coupling facility structure, but it does not need to recover individual shared message data sets except when the data set status is FAILED, or when the status is ACTIVE but the data set header record is no longer valid, indicating that the data set has been re-created. A data set is not selected for recovery if its status is ACTIVE and the data set header is still valid, nor if its status is EMPTY, indicating that no messages were stored in it at the time of the failure.

Shared message data set backups

When BACKUP CFSTRUCT is used to make a backup of the shared messages in an application structure, any data for persistent messages stored in shared message data sets is backed up at the same time, as for persistent shared messages previously stored in DB.

Shared message data set recovery

If a shared message data set is corrupted or lost, then it needs to be put into the FAILED state to stop the queue managers from using it until it has been repaired. This normally happens automatically, but can also be done using the **RESET SMDS** command specifying STATUS(FAILED).

If the shared message data set contained any persistent messages, these can be recovered using the RECOVER CFSTRUCT command. This command first restores any persistent message data for that shared message data set from the most recent BACKUP CFSTRUCT command, then applies all logged changes since that time. If no **BACKUP CFSTRUCT** command has been performed since the time that the data set was first activated, it is reset to empty then all changes since activation are applied.

If the CFSTRUCT contents and all of the shared message data sets are unavailable, for example in a disaster recovery situation, they can all be recovered in a single **RECOVER CFSTRUCT** command.

If a shared message data set is damaged but recovery was not active for the CFSTRUCT, or the log containing the latest BACKUP CFSTRUCT is unavailable or unusable, then the messages offloaded to that data set cannot be recovered. In this case, the **RECOVER CFSTRUCT** command with the parameter TYPE(PURGE) can be used to mark the shared message data set as empty and delete any messages from the structure which had data stored in that data set.

When the **RECOVER CFSTRUCT** command is issued, the shared message data set status is changed from FAILED to INRECOVER. If recovery completes successfully, the status is automatically changed to RECOVERED, otherwise it changes back to FAILED.

When the data set is changed to the RECOVERED state, this tells the owning queue manager that it can now try to open the data set and perform restart processing.

Shared message data set recovery and syncpoints

The shared message data set recovery process reapplies the changes for all complete log records up to the end of the log, regardless of syncpoints.

If changes were made within syncpoint, restart or recovery processing for the CFSTRUCT may result in backing out of uncommitted requests, so some of the recovered changes may not actually be used, but there is no harm in recovering them anyway.

It is also possible that an uncommitted MQPUT message may have been written to the structure but the corresponding data may not have been written to the data set or the log (as I/O completion is only forced at the start of syncpoint processing). This is harmless because restart processing will back out the message entry in the structure, so the fact that it refers to unrecovered data does not matter.

Shared message data set restart processing

If a queue manager connection to a CFSTRUCT terminates normally, the queue manager writes out the free block space map for each shared message data set to a checkpoint area within the data set, just before the data set is closed. The space map can then be read in again at connection restart time, provided that neither the CFSTRUCT nor the shared message data set require any recovery processing before the next restart.

However, if a queue manager terminates abnormally, or the structure or data set require any recovery processing, then additional processing is required to rebuild the space map dynamically when the queue manager connection to the structure is restarted.

Provided that the data set itself did not need to be recovered, queue manager restart simply scans the current contents of the structure to locate references to message data owned by the current queue manager, and marks the relevant data blocks as owned in the space map. Other queue managers can continue to use the structure and read the data owned by the restarting queue manager while the space map is being rebuilt.

Shared message data set restart after recovery

If a shared message data set had to be recovered from a backup, then all nonpersistent messages stored in the data set will have been lost, and if the data set was recovered using TYPE(PURGE) then all messages stored in the data set will have been lost. Until recovery has completed, the data set will be marked as FAILED or INRECOVER so any attempt to read one of the affected messages from another queue manager returns an error code indicating that the data set is temporarily unavailable.

When the data set has been recovered, the status is changed to RECOVERED, which allows the owning queue manager to open it for restart processing, but the data set remains unavailable to other queue managers. Queue manager restart scans the structure to rebuild the space map for any remaining messages. The scan also checks for messages for which the data has been lost, and deletes them from the structure (or if necessary flags them as lost, to be deleted later).

The data set status is automatically changed from RECOVERED to ACTIVE when this restart scan completes, at which point other queue managers can start using it again.

Shared message data set usage information

The DISPLAY USAGE command now also shows information about shared message data set space and buffer pool usage for any currently open shared message data sets. This information is displayed if either the new option TYPE(SMDS) or the existing option TYPE(ALL) is specified.

Shared message data performance and capacity considerations

Monitoring data set usage

The current percentage full of each owned shared message data set can be displayed by the **DISPLAY USAGE** command with the option **TYPE(SMDS)**.

The queue manager will normally automatically expand a shared message data set when it reaches 90% full, provided that the option **DSEXPAND(YES)** is in effect for the SMDS definition. This applies when either the SMDS option is set to **DSEXPAND(YES)** or the SMDS option is set to **DSEXPAND(DEFAULT)** and the CFSTRUCT default option is set to **DSEXPAND(YES)**.

If the expansion attempt fails because no secondary allocation size was specified when the data set was created (giving message IEC070I with reason code 203) the queue manager repeats the expansion request using an override secondary allocation of approximately 20% of the current size.

When a data set is expanded, the new data set extents are formatted as part of the expansion processing, which can take tens of seconds, or even minutes for very large extents. The new space becomes available for use after formatting is complete and the catalog has been updated to show the new high used control interval.

If new messages are being created very rapidly, it is possible for the existing data set to become full before expansion processing completes. In this case, any request which could not allocate space is temporarily suspended until the expansion attempt completes and the new space becomes available for use. If the expansion was successful the request is retried automatically.

If an expansion attempt fails, because of a lack of available space or because the maximum extents have already been reached, a message is issued giving the reason for the failure, then the override option for the affected SMDS is automatically altered to **DSEXPAND(NO)** to prevent further expansion attempts. In this case, there is a risk that the data set may become full, in which case further action may be needed as described in Data set becomes full.

Monitoring application structure usage

The usage level of an application structure can be displayed using the MVS **DISPLAY XCF,STRUCTURE** command specifying the full name of the application structure (including the queue sharing group prefix). The IXC360I response message shows current usage of elements and entries.

When the structure usage exceeds the **FULLTHRESHOLD** value specified in the CFRM policy, the system issues message IXC585E and may perform automatic **ALTER** actions if specified, which may either alter the entry to element ratio or increase the structure size.

Optimising buffer pool sizes

Each buffer in a shared buffer pool is used to read or write a contiguous range of pages for one message of up to the logical block size. If the message spills over into further blocks, each range of pages in a separate block requires a separate buffer.

Buffers containing message data after a write or read operation are retained in storage and reused using a least-recently-used (LRU) cache scheme so that a request to read the same data again shortly afterwards will not need to go to disk. This provides a significant optimization when shared messages are written and then read back soon afterwards by applications running on the same system. If messages owned by another queue manager are browsed for selection purposes then retrieved, this also avoids the need to reread the message from disk.

This means that the number of buffers required for each application structure is one for each concurrent API request which reads or writes large messages for that application structure plus some number of additional buffers which will be used to save recently accessed data in order to optimize subsequent read accesses.

For shared buffer pools, if there are insufficient buffers, API requests will simply wait if a buffer is not immediately available. However, this situation should be avoided as it can cause significantly degraded performance.

The statistics from the **DISPLAY USAGE** command for shared buffer pools show whether there have been any buffer waits within the current statistics interval, and also shows the lowest number of free buffers (or a negative value indicating the maximum number of threads which waited for a buffer at any time), the number of buffers which have saved data, and the percentage of the times that a buffer request has successfully found saved data on the LRU chain (“LRU hits”) instead of having to read it (“LRU misses”) [1] .

- If there have been any waits, the number of buffers should be increased.
- If there are many unused buffers, the number of buffers may be reduced to make more storage available in the region for other purposes.

[1] `(Hits / (Hits+Misses))* 100`

- If there are many buffers containing saved data but the proportion of reads which were hits against that saved data is very small, the number of buffers may be reduced if the storage could be better used for other purposes. The number of buffers should not however be reduced by more than the lowest number of free buffers, as that could trigger waits, and it should preferably be high enough that the lowest free buffer count is normally well above zero.

Deleting shared message data sets

The DELETE CFSTRUCT command (which is only allowed when all shared queues in the structure are empty and closed) does not delete the shared message data sets themselves, but they can be deleted in the usual way after this command has completed. If the same data set is to be reused as a shared message data set, it must be reformatted first to reset it to the empty state.

Exception situations for shared message data sets

There are a number of exception situations which can occur during normal use, even when no software or hardware error is present.

Data set becomes full

If a data set becomes full but cannot be expanded, or the expansion attempt fails, applications using the corresponding queue manager to write large messages to the corresponding application structure will receive error 2192, `MQRC_STORAGE_MEDIUM_FULL` (also known as `MQRC_PAGESET_FULL`).

A data set could become full because of a failure in the application which is supposed to process the data, causing a large backlog of messages to accumulate. If so, expanding the data set any further will only be a temporary solution, and it is important to get the processing application going again as soon as possible.

If more space can be made available the **ALTER SMDS** command can then be used to set **DSEXPAND(YES)** or **DSEXPAND(DEFAULT)** (assuming that YES has been set or assumed as the **DSEXPAND** default for the CFSTRUCT definition) to trigger a retry. If the reason for the failure was however that maximum extents had been reached, the new expansion attempt will be rejected with a message and **DSEXPAND(NO)** will be set again. In this case, the only way to expand it any further is to reallocate it, which involves making it temporarily unavailable, as described next.

Data set needs to be moved or reallocated

If a data set needs to be moved or expanded but is otherwise in normal use, it can be taken out of use temporarily to allow it to be moved or reallocated. Any API request which attempts to use the data set while it is unavailable will receive the reason code `MQRC_DATA_SET_NOT_AVAILABLE.`

1. Use the **RESET SMDS** command to mark the data set as **ACCESS(DISABLED)**. This will cause it to be closed normally and deallocated by all currently connected queue managers.
2. Move or reallocate the data set as necessary, copying the old contents to the newly allocated data set, for example using the Access Method Services (AMS) **REPRO** command.

   Do not attempt to preformat the new data set before copying the old data into it, as this would result in the copied data being appended to the end of the formatted data set.
3. Use the **RESET SMDS** command to mark the data set as **ACCESS(ENABLED)** again, to bring it back into use.

   If the old contents are smaller than the size of the new data set, the rest of the space will be preformatted automatically when the new data set is opened.

   If the old contents were larger than the size of the new data set then the queue manager has to scan the messages in the coupling facility structure and rebuild the space map to ensure that none of the active data has been lost. If any reference is found to a data block which is outside the new extents, the data set is marked as **STATUS(FAILED)** and must be repaired by replacing the data

set with one of the correct size and either copying the old data set into it again or using **RECOVER CFSTRUCT** to recover any persistent messages.

Coupling facility structure is low on space

If the coupling facility structure is running out of space, causing message IXC585E, it is worth checking whether the offload rules have been set to ensure that the maximum amount of data is being offloaded in this case. If not, the offload rules can be modified using the **ALTER CFSTRUCT** command.

Error situations for shared message data sets

There are a number of problems to be aware of, which can only be caused by errors and not occur in normal operational situations.

Owned data set cannot be opened

If the queue manager which owns a shared message data set cannot allocate it or open it, or the data set attributes are not supported, the queue manager sets an appropriate **SMDSCONN** status value of **ALLOCFAIL** or **OPENFAIL** and sets the **SMDSCONN** availability to **AVAIL(ERROR)**. It also sets the SMDS availability to **ACCESS(SUSPENDED)**. When the error has been corrected, use the **RESET SMDS** command to set **ACCESS(ENABLED)** to trigger a retry, or issue the **START SMDSCONN** command to the owning queue manager.

Read-only data set cannot be opened

If a queue manager cannot allocate or open a shared message data set owned by another queue manager and marked as **STATUS(ACTIVE)**, it assumes that this is probably due to a specific problem with its connection to the data set (represented by the **SMDSCONN** object) rather than a problem with the data set itself.

It marks the **SMDSCONN** as **STATUS(ALLOCFAIL)** or **STATUS(OPENFAIL)** as appropriate and marks the **SMDSCONN** availability as AVAIL(ERROR) to prevent further attempts to use it.

If the problem can been corrected without affecting the status of the data set itself, use the **START SMDSCONN** command to trigger a retry.

If the problem turns out to be a problem with the data set itself, then the **RESET SMDS** command can be used to mark the data set as **STATUS(FAILED)** until it it has been recovered. When the data set has been recovered, the action of changing the status back to **STATUS(ACTIVE)** will cause other queue managers to be notified. If the **SMDSCONN** is marked as **AVAIL(ERROR)**, it will automatically be changed back to **AVAIL(NORMAL)** to trigger a new attempt to open the data set.

Data set header is corrupt

If the data set was successfully opened but the format of the header information is incorrect, the queue manager closes and deallocates the data set and sets the status set to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**. This allows **RECOVER CFSTRUCT** to be used to recover the contents.

If the error arose because the data set contained residual data from another use and had not been subsequently preformatted, then preformat the data set and use the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set must be recovered.

Data set is unexpectedly empty

If the queue manager opens a data set which is marked as **STATUS(ACTIVE)** but finds that it is uninitialized or newly preformatted but otherwise valid, the queue manager closes and deallocates the shared message data set then sets the status to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**.

Data set has permanent I/O errors

If a data set has permanent I/O errors after successful **OPEN** processing, it probably needs recovery. The queue manager will mark the data set as **STATUS(FAILED)** so that all currently connected queue managers will close and deallocate it.

Data set has recoverable I/O errors

If there are hardware problems with the data set, it is possible that this might result in recoverable I/O errors which are not reflected back to the queue manager but which cause significant performance degradation, and also indicate a risk of permanent I/O errors in the near future.

In this case, the data set may be taken off line for recovery by using the **RESET SMDS** command to mark it as **STATUS(FAILED)**. This will cause it to be closed and deallocated by all queue managers, so for example it could be moved to a new volume before being made available again.

When a data set is made unavailable in this way, the space map is not saved so the queue manager connection restart processing will need to scan the coupling facility structure to locate messages in the data set and rebuild the space map before the data set can be made available again. As an alternative, if the shared message data set is still usable, it set can be made unavailable more gently by using the **RESET SMDS** command to mark the data set **ACCESS(DISABLED)** until it is ready to be made available again.

Data set contents are incorrect

The queue manager cannot detect directly that a data set contains incorrect data or is not up to date, for example because a volume including that data set had to be restored from backups. However, it performs integrity checks which make it very unlikely that any such errors could result in incorrect message data being seen by application programs.

For integrity checking purposes, each message block in the data set is prefixed with a copy of the corresponding coupling facility entry ID, including a unique time stamp, which is checked whenever the message block is read, before the message data is passed to the user program. If the message block prefix does not match the entry ID (and the coupling facility entry was not deleted in the mean time) the message block is assumed to be damaged and unusable.

If the damaged message was persistent, the data set is marked as **STATUS(FAILED)** and the structure contents must be recovered using the **RECOVER CFSTRUCT** command. If the damaged message was non-persistent, there is no way to recover it, so a diagnostic message is issued and the corresponding coupling facility message entry is deleted.

If no saved space map is available when the data set is opened, it is rebuilt by scanning the coupling facility structure for references to data in the data set. During this scan, the queue manager performs a number of actions:

1. The queue manager determines the location of the most recent message (if any) currently remaining in the data set.
2. The queue manager then reads that message from the data set to ensure that the block prefix matches the message entry id

These actions ensure that the queue manager detects any case where the data set is down-level, and marks the data set as FAILED. This check does however tolerate the case where the data set was restored from a previous copy and either no new messages had been added since then or all messages added since that copy had been subsequently read and deleted.

To protect against down-level data in the case where the data set was closed normally, the queue manager performs a number of actions:

1. The queue manager saves a copy of the space map time stamp in the SMDS object within Db2 when the data set is closed normally.
2. The queue manager then checks the space map time stamp is the same, when the data set is opened again

If the time stamp does not match, this suggests that a down-level copy of the data set might have been used, so the queue manager ignores the existing space map and rebuilds it, which will succeed only if no message data was actually lost.

Note: These integrity checks do not guarantee to detect a down-level or damaged data set in all theoretically possible cases. For example, they will not detect a case where the start of a message block is valid but the rest of the data has been partly overwritten.

Recovery scenarios for shared message data sets

This section described shared message data set recovery scenarios.

Data set recovery where no data was lost

In some cases, the correct contents of a failed data set can be restored without needing actual recovery. One example is where a data set contains residual data from a previous use and has not been preformatted again, which can be fixed by preformatting it. Another case is when a data set has been moved, but there was an error in the process of copying the data across, which can be fixed by copying the data again correctly.

In such cases, the corrected data set can be made available again by using the **RESET SMDS** command to set **STATUS(RECOVERED)**. If the availability is currently **ACCESS(SUSPENDED)** this will automatically set it back to **ACCESS(ENABLED)**.

When the owning queue manager is notified that the data set has been recovered, it scans the structure contents to reconstruct the space map, then changes the status to **STATUS(ACTIVE)**. The other queue managers can then start reading the data set again.

Data set recovery with TYPE(NORMAL)

If the contents of a data set have been lost, but the application structure was defined with **RECOVER(YES)** and the appropriate recovery logs are available, the **RECOVER CFSTRUCT** command can be used to recover any persistent messages stored in the structure including persistent message data offloaded to shared message data sets. This command restores the current state using information logged by the **BACKUP CFSTRUCT** command plus all logged changes to persistent messages since the backup time.

The **RECOVER CFSTRUCT** command always recovers all persistent messages in the coupling facility structure together with offloaded message data stored in Db2. For offloaded data stored in shared message data sets, each data set is only selected for recovery processing if it is already marked as **STATUS(FAILED)** or if it is found to be unexpectedly empty or otherwise invalid when opened by recovery processing. Any shared message data set which is marked as active and which passes the validation checks does not need to be recovered, as the existing message data is already correct, but the header is updated to indicate that any saved space map will need to be rebuilt after recovery.

Recovery processing is only possible when the structure has been marked as failed, as the complete contents of the structure need to be reconstructed by recovery processing. However, if at least one shared message data set has been marked as failed the **RECOVER CFSTRUCT** command will automatically mark the structure as failed if necessary to allow recovery processing to proceed.

Recovery may be performed from any queue manager in the queue sharing group, provided that it has been given write access to the relevant data sets.

Only persistent messages are backed up and logged, so normal recovery processing will restore all persistent messages, but will cause any non-persistent messages in the structure to be lost.

When recovery has completed, any data set which was selected for recovery is automatically changed to **STATUS(RECOVERED)**, and if the availability was **ACCESS(SUSPENDED)** it is changed to **ACCESS(ENABLED)**. The queue manager rebuilds the space map for each data set by scanning the messages in the coupling facility, then marks the data set as **STATUS(ACTIVE)** so that it can be used again.

Data set recovery with TYPE(PURGE)

For a recoverable structure, if the data set contents have been lost, but recovery is not possible for some reason, for example because recovery logs are not available or recovery would take too long, the **RECOVER CFSTRUCT** command can be used with **TYPE(PURGE)** to get the structure back to a usable state. This resets the structure to the empty state and marks all of the associated data sets as **STATUS(EMPTY)**.

Deleting the application structure

If a non-recoverable application structure is deleted using the MVS **SETXCF FORCE** command, or as a result of structure failure, then the next time the structure is connected, message CSQE028I is issued to say that the structure has been reset and all existing messages have been discarded, and any existing data sets are automatically reset to **STATUS(EMPTY)** as well. This action makes a non-recoverable structure usable again after loss of data either in the structure or in any of the associated data sets.

If a recoverable application structure is deleted, it will be treated in the same way as if the structure had failed.

Data set recovery fails

If **RECOVER CFSTRUCT** cannot complete for some reason, for example because a log data set is no longer available, or because the queue manager terminated while recovery was in progress, then any data set for which recovery was at least started will be marked in the header to show that partial recovery has been attempted, and the data set will be left in the **STATUS(FAILED)** state.

In this case, the options are to repeat the original recovery request or to recover with **TYPE(PURGE)** instead, discarding the existing data.

If an attempt is made to mark the data set as **STATUS(RECOVERED)** without actually recovering it, then the next time it is opened the queue manager will see that the header indicates incomplete recovery and mark it as **STATUS(FAILED)** again.

Off site disaster recovery

For off site disaster recovery, persistent shared messages can be re-created using only the logs and the Db2 shared objects containing the CFSTRUCT definitions and associated SMDS status information.

After setting up the Db2 tables containing the definitions, the application structure and the shared message data sets can be set up as empty. When a queue manager connects to them and finds that they are unexpectedly empty, it will mark them as failed, after which a single **RECOVER CFSTRUCT** command can be used to recover all persistent messages for all affected structures.

z/OS *SMDS related commands*

This topic describes and provides access to the commands relating to shared message data sets.

Display and alter the **CFSTRUCT** options relating to large message offload (**OFFLOAD** and offload rules) and shared message data sets (**DSGROUP, DSBLOCK, DSBUFS, DSEXPAND)**:

- DISPLAY CFSTRUCT
- DEFINE CFSTRUCT
- ALTER CFSTRUCT
- DELETE CFSTRUCT

Display **CFSTRUCT** status relating to large message offload **(OFFLDUSE)**:

- DISPLAY CFSTATUS

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS
- ALTER SMDS

Display or modify the status and availability of the data sets within the queue sharing group:

- DISPLAY CFSTATUS TYPE(SMDS)
- RESET SMDS

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN
- START SMDSCONN
- STOP SMDSCONN

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT
- RECOVER CFSTRUCT

z/OS ***Advantages of using shared queues***

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

The advantages of shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new z/OS image with a queue manager (in the queue sharing group) and a copy of the application.
- It is highly available.
- It naturally performs *pull* workload balancing, based on the available processing capacity of each queue manager in the queue sharing group.

Using shared queues for high availability

The following examples illustrate how you can use a shared queue to increase application availability.

Consider an IBM MQ scenario where client applications running in the network want to make requests of server applications running on z/OS. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

IBM MQ manages the transportation of the request message from the client machine to the server's input queue on z/OS and of the response from the server back to the client. By defining the server's input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue sharing group. This means that you can configure a queue manager on each z/OS image in the sysplex and, by connecting them all to the same queue sharing group, any one of them can access messages on the server's input queue.

Messages on the input queue of the server are still available, even if one of the queue managers terminates abnormally or you have to stop it for administrative reasons. You can take an entire z/OS image offline and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each z/OS image in the sysplex to provide higher server application capacity and availability, as shown in Figure 60 on page 157.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result that is sent back to the client as an IBM MQ message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that you can use to configure the return path. For more information about these options, see “Distributed queuing and queue sharing groups” on page 175.

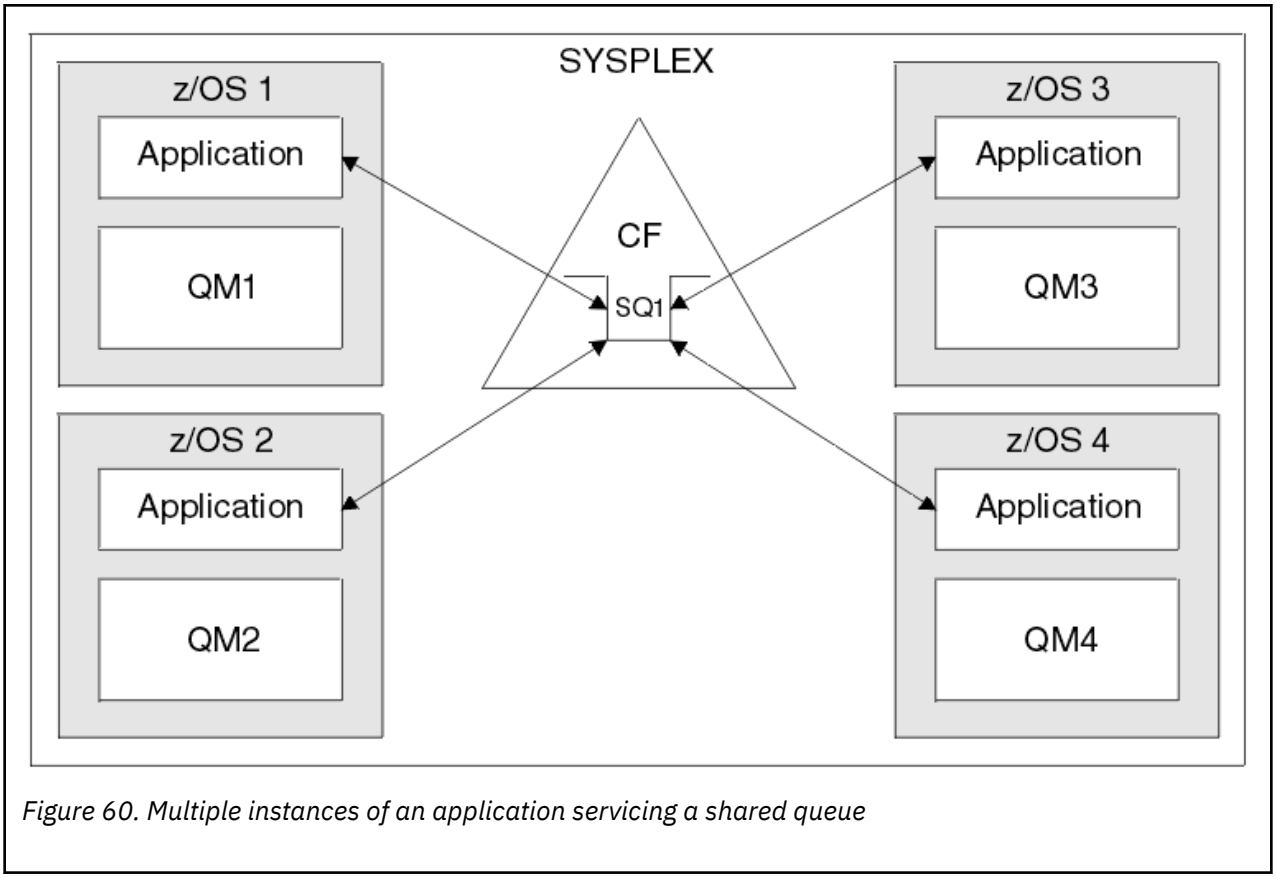


Figure 60. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally and completes units of work for that queue manager that are still pending, where possible. This feature is known as *peer recovery.*

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet put the response message or committed the unit of work. Another queue manager in the queue sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If IBM MQ cannot resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue sharing group to continue processing that work.

z/OS *Use of storage class memory with shared queues*

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

The z13, zEC12, and zBC12 machines allow the installation of Flash Express cards. These cards contain flash solid-state drives (SSD). After installation, flash storage from the cards can be allocated to one or more LPARs where it is typically known as SCM.

SCM sits between real storage and direct access storage device (DASD) in terms of both I/O latency and cost. Because SCM has no moving parts, it exhibits much lower I/O latencies than DASD.

SCM is also much cheaper than real storage. As a result, a large amount of storage can be installed for a relatively low cost; for example, a pair of Flash Express cards contains 1424 GB of usable storage.

These characteristics mean that SCM is useful when a large amount of data must be taken from real storage in a short period of time, because that data can be written to SCM much quicker than it can be written to DASD. This specific point can be very useful when using coupling facility (CF) list structures containing IBM MQ shared queues.

Why list structures fill up

When a CF structure is defined, it is configured with a SIZE attribute that describes the maximum size of the structure. Because CF structures are always permanently resident in real storage, the sum of the SIZE attributes of the structures that are defined on a CF should be less than the amount of the real storage allocated to the CF.

As a result, there is a constant pressure to keep the SIZE value of any given structure to the minimum value possible, so that more structures can fit into the CF. However, ensuring structures are large enough to achieve their purpose can result in a conflicting pressure, because making a structure too small means that it might fill up, disrupting the applications or subsystems making use of it.

There is a strong need to accurately size a structure based on its expected usage. However, this task is difficult to do because workloads can change over time and accounting for their fluctuations is not easy.

IBM MQ shared queues use CF list structures to store messages. IBM MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM MQ CFSTRUCT objects. When a message that is smaller than 63 KB is put on a shared queue, the message is stored entirely in an application structure as a single list entry, and zero or more list elements.

Because IBM MQ shared queues use list structures, the pressures described also affect shared queues. In this case, the maximum number of messages that can be stored on a shared queue is a function of the:

- Size of the messages on the queue
- Maximum size of the structure
- Number of entries and elements available in the structure

Because up to 512 shared queues can use the same structure, and effectively compete for entries and elements, this complicates matters even further

IBM MQ queues are used for the transfer of data between applications, so a common situation is an application putting messages to a queue, when the partner application, which should be getting those messages, is not running.

When this happens the number of messages on the queue increase over time until one or more of the following situations occur:

- The putting application stops putting messages.
- The getting application starts getting messages.
- Existing messages on the queue start expiring, and are removed from the queue.

- The queue reaches its maximum depth in which case an MQRC_Q_FULL reason code is returned to the putting application.
- The structure containing the shared queue reaches its maximum size, or the CF containing the structure runs out of available storage. In either case, an MQRC_STORAGE_MEDIUM_FULL reason code is returned to the putting application.

In the last three cases the queue is full. At this point the putting application has a problem because there is nowhere for its messages to go. The putting application typically solves this problem by using one or more of the following solutions:

- Repeatedly retry putting the message, optionally with a delay between retries.
- Put the messages somewhere else, such as a database or a file. The messages can be accessed later and put to the queue as normal.
- Discard the message if it is nonpersistent.

However, for some classes of applications, for example those with a large volume of incoming messages, or no access to a file system, these solutions are not practical. There is a real need to ensure that queues never, or are extremely unlikely to, fill up in the first place and this is especially pertinent to shared queues.

SMDS and offload rules

The offload rules introduced in IBM WebSphere MQ 7.1 provide a way of reducing the likelihood of an application structure filling up.

Each application structure has three rules associated with it, specified using three pairs of keywords:

- OFFLD1SZ and OFFLD1TH
- OFFLD2SZ and OFFLD2TH
- OFFLD3SZ and OFFLD3TH

Each rule specifies the conditions that must be met for message data to be offloaded to the storage mechanism that is associated with the application structure. Two types of storage mechanisms are currently available:

- Db2
- Group of Virtual Storage Access Method (VSAM) linear data sets, which IBM MQ calls a shared message data set (SMDS).

The following example shows the MQSC command to create an application structure named LIST1, using the DEFINE CFSTRUCT command.

This structure has the default offload rules in place, and uses SMDS as the offload mechanism. This means that when the structure is 70% full (OFFLD1TH), all messages that are 32 KB or larger (OFFLD1SZ) are offloaded to SMDS.

Similarly, when the structure is 80% full (OFFLD2TH) all messages that are 4 KB or larger (OFFLD2SZ) are offloaded. When the structure is 90% full (OFFLD3TH) all messages (OFFLD3SZ) are offloaded.

```
DEFINE CFSTRUCT(LIST1)
CFLEVEL(5)
OFFLOAD(SMDS)
OFFLD1SZ(32K) OFFLD1TH(70)
OFFLD2SZ(4K) OFFLD2TH(80)
OFFLD3SZ(0K) OFFLD3TH(90)
```

An offloaded message is stored in the offload medium, and a pointer to the message is stored in the structure. While the offload rules reduce the chance of the structure filling up, by putting less message data in the structure as it runs out of storage, some data is still written to the structure for each message. That is, the pointer to the offloaded message.

Additionally, the offload rules come with a performance cost. Writing a message to a structure is relatively quick and is largely dominated by the time spent to send the request for the write to the CF. The actual writing to the structure is fast, happening at real storage speeds.

Writing a message to SMDS is much slower because it includes writing to the structure for the message pointer, and writing the message data to SMDS. This second write operation is done at DASD speed and has the potential to add latency. If Db2 is used as the offload mechanism the performance cost is much greater.

z/OS *How storage class memory works with IBM MQ for z/OS*

An overview of the use of storage class memory (SCM) with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

A coupling facility (CF) that is at CFLEVEL 19, or greater, can have SCM allocated to it. The structures defined in that CF can then be configured to make use of SCM to reduce the chances of the structures filling up (known as a structure full condition). When a structure configured to make use of SCM fills up past a system-determined point, the CF starts moving data from the structure into SCM, which frees space in the structure for new data.

Note: Because SCM itself can fill up, allocating SCM to a structure reduces only the likelihood of a structure full condition, but does not entirely remove the chance of one occurring.

A structure is configured to use SCM by specifying both the **SCMALGORITHM** and **SCMMAXSIZE** keywords in the coupling facility resource manager (CFRM) policy, containing the definition of that structure.

Note that after these keywords are specified, and the CFRM policy is applied, the structure must be rebuilt, or deallocated so that they can take effect.

SCMALGORITHM keyword

Because the input/output speed of SCM is slower than that of real storage, the CF uses an algorithm that is tailored to the expected use of the structure in order to reduce the impact of writing to, or reading from, SCM.

The algorithm is configured by the **SCMALGORITHM** keyword in the CFRM policy for the structure, using the *KEYPRIORITY1* value. Note that you should use the *KEYPRIORITY1* value only with list structures used by IBM MQ shared queues.

The *KEYPRIORITY1* algorithm works by assuming that most applications will get messages from a shared queue in priority order; that is, when an application gets a message, it gets the oldest message with the highest priority.

When a structure starts to fill past the system-defined threshold of 90%, the CF starts asynchronously migrating messages that are least likely to be got next. These are messages with lower priorities that were more recently put on the queue.

This asynchronous migration of messages from the structure into SCM is known as "pre-staging".

Pre-staging reduces the performance cost of using SCM because it reduces the chance of an application being blocked during the occurrence of synchronous input/output to SCM.

In addition to pre-staging, the *KEYPRIORITY1* algorithm also asynchronously brings back messages from SCM and into the structure when sufficient free space is available. For the *KEYPRIORITY1* algorithm, this means that when the structure is less than or equal to 70% full.

The act of bringing messages from SCM into the structure is known as "pre-fetching".

Pre-fetching reduces the likelihood of an application trying to get a message that has been pre-staged to SCM and having to wait while the CF synchronously brings back the message into the structure.

SCMMAXSIZE keyword

The **SCMMAXSIZE** keyword defines the maximum amount of SCM that can be used by a structure. Because SCM is allocated to the structure by the CF when it is required, it is possible to specify a **SCMMAXSIZE** that is greater than the total amount of free SCM available. This is known as "over-committing".

Important: Never over-commit SCM. If you do, the applications that are relying on it will not obtain the behavior that they expect. For example, IBM MQ applications using shared queues might get unexpected MQRC_STORAGE_MEDIUM_FULL reason codes.

The CF uses various data structures to track its use of SCM. These data structures reside in the real storage that is allocated to the CF and, as a result, reduce the amount of real storage that can be used by structures. The storage used by these data structures is known as "augmented space".

When a structure is configured with SCM, a small amount of real storage is allocated from the CF to the structure known as fixed augmented space. This is allocated even if the structure never actually uses any SCM. As data from the structure is stored into SCM, extra dynamic augmented space will be allocated from the spare real storage in the CF.

When the data is removed from SCM, the dynamic augmented space is returned to the CF. Augmented space, either fixed or dynamic, is never taken from the real storage that is allocated to a structure.

In addition to augmented storage, when a structure is configured to use SCM, the amount of control storage used by that structure increases. This means that a list structure configured with SCM can contain fewer entries and elements than a structure of the same size without SCM being configured.

To understand the impact of SCM on new or existing structures, use the CFSizer tool.

A final important point to note is that after data is moved from the structure into SCM, and dynamic augmented space has been used, the structure cannot be altered either manually or automatically.

That is, the amount of storage allocated to the structure cannot be increased or decreased, the entry-to-element ratio that is used by the structure cannot be changed, and so on. To make the structure alterable again, the structure must not have any data stored in SCM and must not be making use of dynamic augmented storage.

z/OS *Why use SCM*

Emergency storage and improved performance are two use cases for using SCM with IBM MQ for z/OS.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This section introduces the theory behind the two possible scenarios. For further details on how you set up the scenarios, see:

- “Emergency storage - basic configuration” on page 164
- “Improved performance - basic configuration” on page 170

Important: The use of SCM with CF structures is not dependent on any specific version of IBM MQ. However the emergency storage scenario works only with IBM WebSphere MQ 7.1 and later, because it requires SMDS and the offload rules.

Emergency storage

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

Overview

A single shared queue is configured on an application structure. The putting application puts messages onto the shared queue; the getting application gets messages from the shared queue.

During normal running, the queue depth is expected to be close to zero, but a business requirement indicates that the system must be able to tolerate a two-hour outage of the getting application. This means that the shared queue must be able to contain two hours of messages from the putting application.

Currently, this process is achieved by using the default offload rules, and SMDS, so that the size of the structure is minimized, while reducing the performance cost that is associated with offloading.

The rate of messages being sent to the shared queue is expected to double in the short to medium term. Although the requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure.

Because the CF, which contains the application structure, resides on a zEC12 machine, the possibility exists to associate sufficient SCM with the structure to store enough messages so that a two-hour outage can be tolerated.

Consider what happens over a period of time:

1. Initially, the system is in its steady state. Both the putting and getting application are running normally and the queue depth is close to, or at, zero. The result is that the application structure is largely empty.
2. At a certain time, the getting application suffers an unexpected failure and stops. The putting application continues to put messages to the queue and the application structure starts to fill up.
3. After the structure reaches 70% full, the conditions of the first offload rule are met and all messages with a size greater than or equal to 32 KB are offloaded to SMDS.

   See “SMDS and offload rules” on page 159 for an overview of the offload rules.
4. As messages continue to be put to the shared queue, the structure continues to fill (either because of the message data being stored in the structure, or as a result of the pointers to the offloaded messages being stored in the structure).

   When the structure reaches 80% full, the second offload rule starts to apply and messages that are 4 KB or greater are offloaded to SMDS.
5. When the structure is past 90% full, all messages are offloaded to SMDS and only the message pointers are being placed in the structure.

   About this time, the pre-staging algorithm starts to run, and begins moving data from the structure into SCM. Assuming all messages on the queue are the same priority, the newest messages are pre-staged.

   Because all messages are now being offloaded to SMDS, the data being moved into SCM is not actual message data, but instead the pointers to the messages on SMDS.

   As a result, the number of messages that can be stored on the combination of the structure, and the SCM and SMDS associated with the structure, is very large.

   Performance: During this stage of the outage, the putting application can suffer a degree of performance degradation because of having to write to SMDS. In this case, the use of SCM should not be a limiting factor on the putting application in terms of performance. SCM provides extra space to prevent the structure filling up.
6. Eventually the getting application is available again and the outage is over.

   However SCM is still being used by the structure. The getting application starts reading messages off the queue, getting the oldest, highest priority messages first.

   Because these messages were written before the structure started to fill up, they come out entirely from the real storage portion of the structure.
7. As the structure starts to empty, it goes below the threshold at which pre-staging is active, and so pre-staging stops.
8. The structure usage reduces below the point at which the offload rules take effect, so messages are no longer offloaded to SMDS unless they are more than 63 KB.

At about this time, the pre-fetch algorithm starts moving data from SCM in to the structure. Because the getting application gets messages from the queue in the order expected by the SCM algorithms, messages are brought in before the getting application needs them.

The result is that the getting application never needs to wait for messages to be brought in synchronously from SCM.

9. As the getting application continues to move down the queue, it starts retrieving messages that were offloaded to SMDS.
10. Finally, the system is in a steady state again. No messages are stored in SCM or SMDS, and the queue depth is close to zero.

Improved performance

This scenario describes using SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

Description

For this scenario, a putting and getting application communicate through a shared queue which is stored in application structure.

The putting application tends to run in bursts, when it puts a large number of messages in a short amount of time. Then, in an extended period of time, it produces no messages at all.

The getting application sequentially processes each message, and performs complex processing on each one. As a result, most of the time the queue depth is zero, except for when the putting application starts to run, where the queue depth starts increasing as messages are being put faster than they are being got.

The queue depth increases until the putting application stops, and the getting application has enough time to process all the messages on the queue.

Notes:

1. In this scenario, the key factor is performance. The messages being sent to the queue are always less than 63 KB and so never need to be offloaded to SMDS.
2. The application structure has been sized so that it is large enough to contain all of the messages that will be placed on it by the putting application in a single "burst."
3. The offload rules must all be disabled so that, even when the structure starts to fill, the messages are not offloaded to SMDS. This is because the performance costs that are associated with writing messages to, and reading messages from, SMDS are deemed unacceptable.

Over time, the number of messages that the putting application send in a burst must increase by several orders of magnitude. Because the getting application must process each message sequentially, the number of messages on the queue increases to the point where the structure fills up.

At this point, the putting application receives a reason code (MQRC_STORAGE_MEDIUM_FULL) when putting a message, and the put operation fails. The putting application can only briefly tolerate periods when it is unable to put messages to the queue. If the period is too long, the application ends.

Assuming that you do not have the time, or skills available, to rewrite either the putting application or getting application, this problem has three possible solutions:

1. Increase the size of the application structure.
2. Add offload rules to the application structure so that messages are offloaded to SMDS as the queue starts to fill up.
3. Associate SCM with the structure.

The first solution is quick to implement, but not enough real storage is available on the CF.

The second solution might also be quick to implement, but the performance impact of offloading to SMDS is considered too significant to use this option.

The third solution, associating SCM with the structure, provides an acceptable balance of cost and performance.

Associating SCM with a structure results in a higher use of real storage in the CF because of the augmented storage that get operations used. However, the actual amount of real storage will be less than the amount used in the first option.

Another consideration is the cost of SCM. However this cost is much cheaper than real storage. These factors combine to make the third option cheaper than the first option.

Although the third option, potentially, might not perform as well as the first option, the pre-fetch and pre-staging algorithms used by the CF can combine to make the differences in performance acceptable, or in some cases negligible.

Certainly the performance can be much better than using SMDS to offload messages.

Consider what happens over a period of time:

1. Initially, the getting application is active and waiting for messages to be delivered to the shared queue. The putting application is not active and the shared queue is empty.
2. At a certain time, the putting application becomes active, and starts putting a large number of messages to the shared queue. The getting application starts getting the messages, but the queue depth rapidly starts to increase because the getting application is slower than the putting application.

   As a result, the application structure starts to fill up.
3. As the time increases, the putting application is still active. The application structure fills up to approximately 90%.

   This is when the SCM pre-staging algorithm starts to move messages from the structure into SCM, freeing space in the structure.

   Because the getting application gets the oldest, highest priority messages from the queue first, it is always getting messages from the structure and does not need to wait for messages to be brought synchronously from SCM in to the structure.
4. The putting application is still active and putting messages to the shared queue. However, the application never receives an MQRC_STORAGE_MEDIUM_FULL reason code, because enough space exists in SCM to store all the messages that do not fit in the structure.
5. Eventually, the putting application stops because it has no more messages to put.

   The pre-staging algorithm stops because the structure falls below 90% in use, and the getting application continues processing the messages in the queue.
6. As the getting application starts to free space in the structure, the pre-fetch algorithm starts to bring messages back from SCM in to the structure.

   Because the getting application processes messages in the order expected by the pre-fetch algorithm, the getting application never becomes blocked waiting for message data to be brought synchronously from SCM in to the structure.
7. Finally, the getting application processes all the messages on the shared queue, and waits until the next message is available. The structure and SCM are empty of messages.

z/OS *Emergency storage - basic configuration*

How you set up a basic scenario for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

For example, your enterprise has an application that puts messages onto the queue and an application that gets messages from the queue. During normal running, you expect the queue depth to be close to zero, but a business requirement indicates that the system be able to tolerate a two-hour outage of the application that gets the messages.

This means that the shared queue being used must be able to contain two hours of messages from the putting application. Currently you achieve this by using the default offload rules, and SMDS.

You expect the rate of messages being sent to the shared queue to double in the short to medium term. Although your requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure. Because the CF containing the application structure resides on a zEC12 machine, you have the capability of associating sufficient SCM with the structure to store enough messages, so that a two-hour outage can be tolerated

This initial scenario uses a:

- Queue sharing group, `IBM1`, that contains a single queue manager, `CSQ3`. In addition to the administration structure, the queue sharing group defined a single application structure, `SCEN1`.
- Coupling facility (CF) `CF01`, in which the `SCEN1` application structure is stored as the `IBM1SCEN1` structure. This structure has a maximum size of 1 GB.
- Single shared queue, `SCEN1.Q` that the application structure uses.

This configuration is illustrated in Figure 61 on page 165.

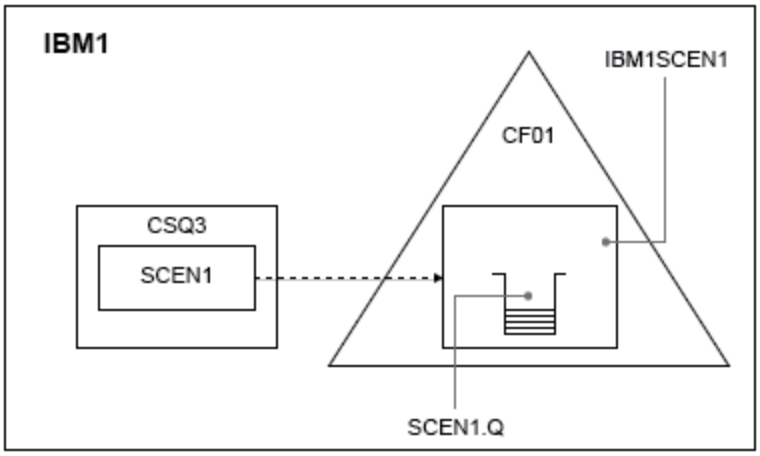


Figure 61. Basic configuration

Furthermore, assume that queue manager `CSQ3` is already the only member of queue sharing group `IBM1`.

You must add the definition for structure `IBM1SCEN1` to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, `CF01`, by specifying `PREFLIST(CF01)`.

Attention: To allow for high availability in your production system, you should include at least two CFs in the `PREFLIST` for any structures that are used by IBM MQ.

Procedure

1. Refresh the CFRM policy by using the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN1
```

Sample CFRM policy for structure IBM1SCEN1:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

2. Verify that the structure has been created correctly, by using the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN1 to create an IBM MQ CFSTRUCT object:

```
DEFINE CFSTRUCT(SCEN1)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 1')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   .
   b. Validate the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN1.Q shared queue, to use the SCEN1 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN1.Q) QSGDISP(SHARED) CFSTRUCT(SCEN1) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN1.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

Check in the output from the command, that the STATUS line shows ALLOCATED.

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

Add SMDS and SCM to the initial structure

Related concepts

“Use of storage class memory with shared queues” on page 157

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SMDS and SCM to the initial structure*

How you add SMDS and SCM for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in "Emergency storage - basic configuration" on page 164. The scenario describes the addition of shared message data sets (SMDS), and then of SCM to the initial structure.

This final configuration is illustrated in Figure 62 on page 167.

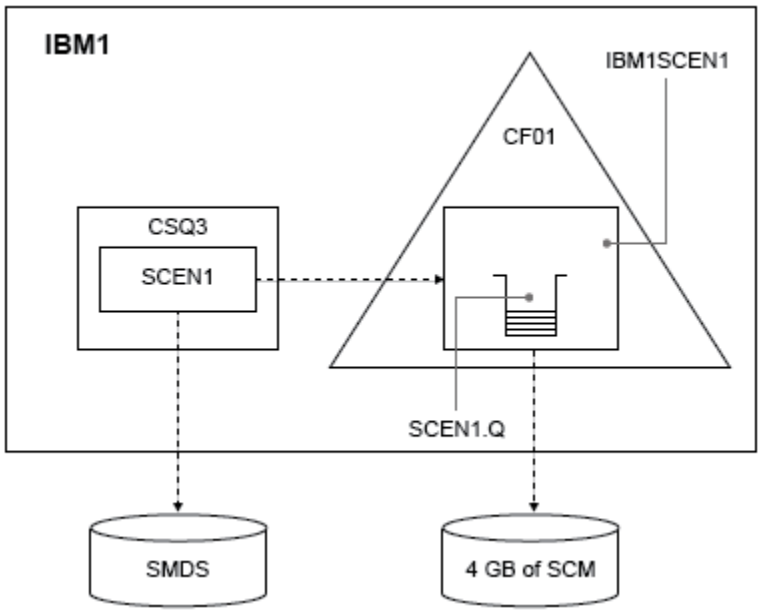


Figure 62. Configuration adding SMDS and SCM for emergency storage

Procedure

1. Create the SMDS data set that the SCEN1 application structure uses, by editing the **CSQ4SMDS** sample JCL, as shown:

```
//CSQ4SMDS JOB NOTIFY=&SYSUID
//*
//* Allocate SMDS
//*
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER           -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS) -
MEGABYTES(5000 3000)     -
LINEAR            -
SHAREOPTIONS(2 3) )      -
DATA                -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS.DATA) )
/*
//*
//* Format the SMDS
//*
//FORM   EXEC PGM=CSQJUFMT,COND=(0,NE),REGION=0M
//STEPLIB DD DSN=MQ800.SCSQANLE,DISP=SHR
//     DD DSN=MQ800.SCSQAUTH,DISP=SHR
```

```
//SYSUT1   DD DISP=OLD,DSN=CSQSMDS.SCEN1.CSQ3.SMDS
//SYSPRINT DD SYSOUT=*
```

2. Issue the ALTER CFSTRUCT command to change the SCEN1 application structure, to use SMDS for offloading, implementing the default offload rules:

```
ALTER CFSTRUCT(SCEN1) OFFLOAD(SMDS) OFFLD1SZ(32K) OFFLD2SZ(4K) OFFLD3SZ(0K)
DSGROUP('CSQSMDS.SCEN1.*.SMDS') DSBLOCK(1M)
```

   Note the following:

   - Because SCEN1.Q is the only shared queue on the SCEN1 application structure, the **DSBLOCK** value has been set to 1M, the largest value possible. This should be the most efficient setting for our scenario.
   - Because the messages sent by the putting application are 30 KB, offloading to SMDS does not start until the second offload rule is met, when the structure is 80% full.

3. Run your test application again.

   Note the increased storage of messages on the queue.

4. Add 4 GB of SCM to structure IBM1SCEN1 by carrying out the following procedure:

   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.

   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

5. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=polname
```

6. Rebuild the IBM1SCEN1 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN1
```

Results

You have successfully added SCM to your configuration.

What to do next

Optimize the performance of your system. See “Optimizing storage class memory usage” on page 169 for more information.

z/OS *Optimizing storage class memory usage*

How you improve your use of storage class memory (SCM).

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Run the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

As the structure was already full with message data, because of the previous tests, part of the rebuild involved pre-staging some of the messages from the structure into SCM. This process was initiated by using the previous command.

The output from this command produces, for example:

```
ACTIVE STRUCTURE
----------------
ALLOCATION TIME: 06/17/2014 09:28:50
CFNAME : CF01
COUPLING FACILITY: 002827.IBM.02.00000000B8D7
PARTITION: 3B CPCID: 00
STORAGE CONFIGURATION ALLOCATED MAXIMUM %
ACTUAL SIZE: 1024 M 1024 M 100
AUGMENTED SPACE: 3 M 142 M 2
STORAGE-CLASS MEMORY: 88 M 4096 M 2
ENTRIES: 120120 1089536 11
ELEMENTS: 240240 15664556 1
SPACE USAGE IN-USE TOTAL %
ENTRIES: 84921 219439 38
ELEMENTS: 2707678 3149050 85
EMCS: 2 282044 0
LOCKS: 1024
SCMHIGHTHRESHOLD : 90
SCMLOWTHRESHOLD : 70
ACTUAL SUBNOTIFYDELAY: 5000
PHYSICAL VERSION: CD5186A0 2BD8B85C
LOGICAL VERSION: CD515C50 CE2ED258
SYSTEM-MANAGED PROCESS LEVEL: 9
XCF GRPNAME : IXCLO053
DISPOSITION : KEEP
ACCESS TIME : NOLIMIT
MAX CONNECTIONS: 32
# CONNECTIONS : 1
CONNECTION NAME ID VERSION  SYSNAME JOBNAME  ASID STATE
--------------- -- -------  ------- -------  ---- ----------------
CSQEIBM1CSQ301  01 00010059 SC61    CSQ3MSTR 0091 ACTIVE
```

Note the following from the output of the command:

- That STORAGE_CLASS MEMORY provides confirmation that a **MAXIMUM** of 4096 MB of SCM has been added to the structure.
- The ALLOCATED figure for the amount of STORAGE-CLASS MEMORY used for pre-staging. There is now free space in the structure where there was none before SCM was added.
- The amount of AUGMENTED SPACE used to track SCM usage.
- The point at which the pre-staging algorithm starts to move data from the structure into SCM is when the structure is 90% full. This is indicated by the non-configurable **SCMHIGHTHRESHOLD** property.
- The point below which the prefetching algorithm starts to move data from SCM into the structure is when the structure is 70% full. This is indicated by the non-configurable **SCMLOWTHRESHOLD** property.

You can now test various ways to optimize the use of SCM. Note the following:

- After SCM is used to store messages, you cannot alter the structure until you have removed all the data from SCM.

  In this case, that means that the entry-to-element ratio is frozen at the value that was in place when SCM was first used. You must carefully ensure that the structure is in the state you want, before the pre-staging algorithm starts moving data into SCM.

- Is the current structure size correct before using SCM?

  For example, have you increased **INITSIZE** from 512 MB to a SIZE of 1 GB?

  If you do not do this, it is possible that although you enabled your structure for auto-alteration, the pre-staging algorithm will start to move data into SCM before the alteration has a chance to start. As a result, the structure is frozen using 512 MB of real storage.
- Is the entry-to-element ratio correct before using SCM?

  The goal of this scenario is to increase the number of offloaded message pointers that can be stored in the structure and SCM as a whole, as well as keeping as many messages entirely in structure storage as possible. Accessing these messages is faster than accessing messages on SMDS.

  Therefore, you need to have a structure that starts with an entry-to-element ratio that is good for storing messages, and then transitions to a ratio that is good for storing message pointers before the prestage algorithm first starts. This transition can be achieved, in part, by making use of the IBM MQ offload rules.

Change the offload rules by issuing the following command:

```
ALTER CFSTRUCT(SCEN1) OFFLD1SZ(0K)
```

You might have to carry out several runs to optimize the entry-to-element ratios.

The following table shows possible improvements in the number of messages put on the queue during the different phases of the emergency storage scenario.

Table 18. Comparison of results for the emergency storage scenario

Test description	Number of messages	Time to fill queue (seconds)
Basic configuration	27,850	3.2
SMDS with default offload rules	205,000	158
SCM with default offload rules	828,610	469
SCM with adjusted offload rules	1,135,775	679

The last row in the table shows that adjusting the offload rules had the required effect.

You need to examine your system to see if you can improve on these figures in any way. For example, you might run out of available SMDS storage. If you can allocate more SMDS storage you should be able to increase the number of messages on the queue quite significantly.

z/OS *Improved performance - basic configuration*

How you set up a basic scenario for improved performance using shared queues on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This scenario describes the use of SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

This initial scenario is very similar to that used for emergency storage and uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN2.

- Coupling facility (CF) CF01, in which the SCEN2 application structure is stored as the IBM1SCEN2 structure. This structure has a maximum size of 2 GB.
- Single shared queue, SCEN2.Q, which is configured to use the application structure.

This configuration is illustrated in Figure 63 on page 171.

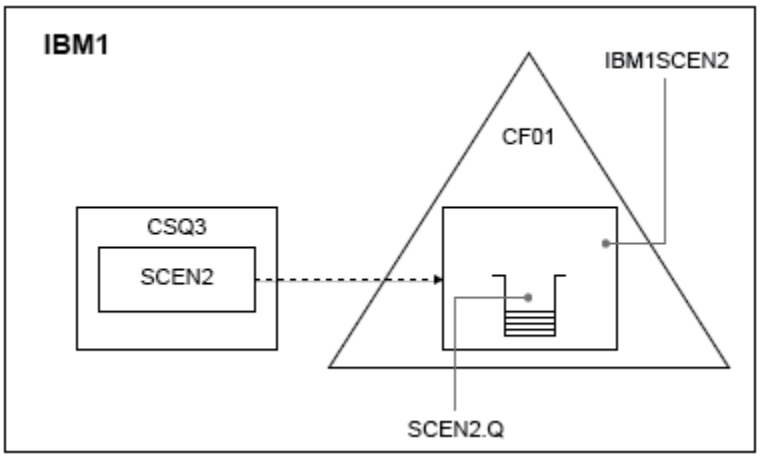


Figure 63. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN2 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Sample CFRM policy for structure IBM1SCEN2:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

Both the **INITSIZE** and **SIZE** keywords have the value 2048M so that the structure cannot resize.

Procedure

1. Refresh the CFRM policy by using the following command:

   ```
   SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
   ```

2. Verify that the structure has been created correctly, by using the following command:

   ```
   D XCF,STR,STRNAME=IBM1SCEN2
   ```

   Issuing the preceding command gives the following output:

   ```
   RESPONSE=SC61
   IXC360I 07.58.51 DISPLAY XCF 581
   STRNAME: IBM1SCEN2
   STATUS: NOT ALLOCATED
   POLICY INFORMATION:
   POLICY SIZE : 2048 M
   POLICY INITSIZE: 2048 M
   ```

```
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY

EVENT MANAGEMENT: MESSAGE-BASED   MANAGER SYSTEM NAME: SC53
MANAGEMENT LEVEL : 01050107
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN2 to create an IBM MQ CFSTRUCT object.

```
DEFINE CFSTRUCT(SCEN2)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 2')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   b. Check the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN2.Q shared queue, to use the SCEN2 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN2.Q) QSGDISP(SHARED) CFSTRUCT(SCEN2) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN2.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output from the command, a portion of which is shown, and ensure that the STATUS line shows ALLOCATED.

```
RESPONSE=SC61
IXC360I 08.31.27 DISPLAY XCF 703
STRNAME: IBM1SCEN2
STATUS: ALLOCATED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: SERIALIZED LIST
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

Additionally, note the values of the fields in the SPACE USAGE section:

- ENTRIES
- ELEMENTS

- EMCS
- LOCKS

An example of the values follows:

```
SPACE USAGE   IN-USE      TOTAL    %
ENTRIES:      344686     345242   99
ELEMENTS:    6548455    6548467   99
EMCS:              2     780318    0
LOCKS:          1024
```

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

You should test the basic scenario. As an example, you can use the following three applications, starting the applications in the order shown and, running them concurrently.

1. Use a PCF application to request the current depth (**CURDEPTH**) value for SCEN2.Q every five seconds. The output can be used to plot the depth of the queue over time.
2. A single threaded getting application repeatedly gets messages from SCEN2.Q, using a get with an infinite wait. To simulate processing of the messages that were removed, the getting application pauses for four milliseconds for every ten messages that it removed.
3. A single threaded putting application puts a total of one million 4 KB non-persistent messages to SCEN2.Q. This application does not pause between putting each message so messages are put on SCEN2.Q faster than the getting application can get them.

   As a result, when the putting application is running, the depth of SCEN2.Q increases.

   When structure IBM1SCEN2 is filled, and the putting application receives a MQRC_STORAGE_MEDIUM_FULL reason code, the putting application sleeps for five seconds before attempting to put the next message to the queue.

You can plot the results of the CURDEPTH application over a period of time. You obtain some form of saw-tooth wave output as the putting application pauses to allow the queue to partially empty.

Go to “Adding SCM to the initial structure” on page 173.

Related concepts

“Use of storage class memory with shared queues” on page 157

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SCM to the initial structure*

How you add SCM for improved performance on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 170. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 174.

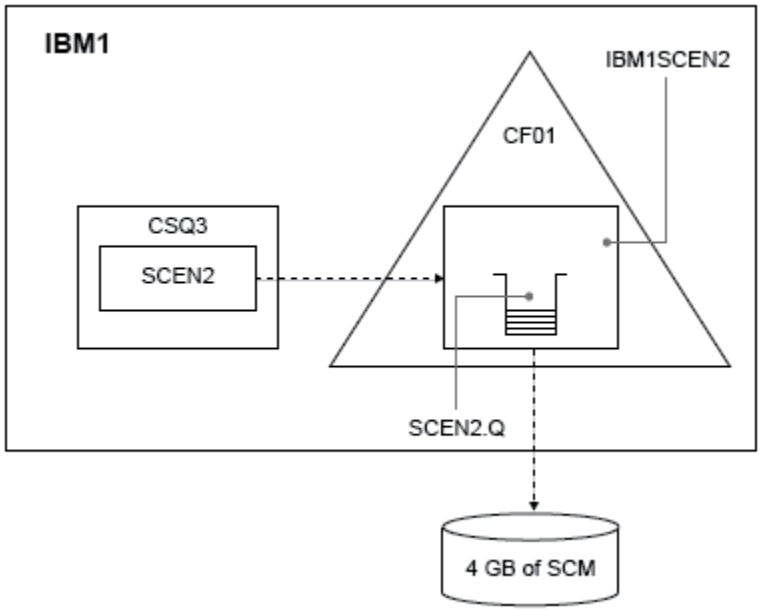


Figure 64. Configuration adding SCM for improved performance

Procedure

1. Add 4 GB of SCM to structure IBM1SCEN2 by carrying out the following procedure:
   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

2. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
```

3. Rebuild the IBM1SCEN2 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN2
```

4. Issue the following command to confirm the new configuration of the structure:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output of the command, a portion of which follows:

```
SPACE USAGE     IN-USE       TOTAL      %
ENTRIES:          33         342684     0
ELEMENTS:         48        6503697     0
EMCS:              2         575600     0
LOCKS:                         1024
```

Results

Calculate the change in the use of real storage by the increase in control storage required to use SCM.

- Before SCM is added to the structure, the structure has these totals as shown in "Improved performance - basic configuration" on page 170:
  - 345,242 entries
  - 6,548,467 elements
  - 780,318 EMCS
- After SCM is added to the structure, the structure has these totals:
  - 342,684 entries
  - 6,503,697 elements
  - 575,600 EMCS

Using these figures, after the SCM was added, the structure is reduced in size by:

- 2558 entries
- 44,770 elements
- 204,718 EMCS

The amount of structure storage that is used to manage SCM, is as follows for a 2 GB structure with 4 GB of SCM allocated:

```
(2558 + 44,770 + 204,718) * 256 = 61.5 MB
```

Note that adding more SCM is likely to achieve only a marginal reduction of the size of the structure, because the amount of control storage used to track SCM increases, both as the structure size, and the amount of allocated SCM increases.

What to do next

Repeat the tests described in the final section of "Improved performance - basic configuration" on page 170.

You can plot the results of the revised application over a period of time. Comparing the plot to the one obtained previously, you now obtain an output without a saw-tooth wave, as the putting application no longer has to wait for the queue to partially empty.

For more information, refer to MP16: WebSphere MQ for z/OS - Capacity planning & tuning.

z/OS Distributed queuing and queue sharing groups

Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

To complement the high availability of messages on shared queues, the distributed queuing component of IBM MQ has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue sharing group.

Figure 65 on page 176 illustrates distributed queuing and queue sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

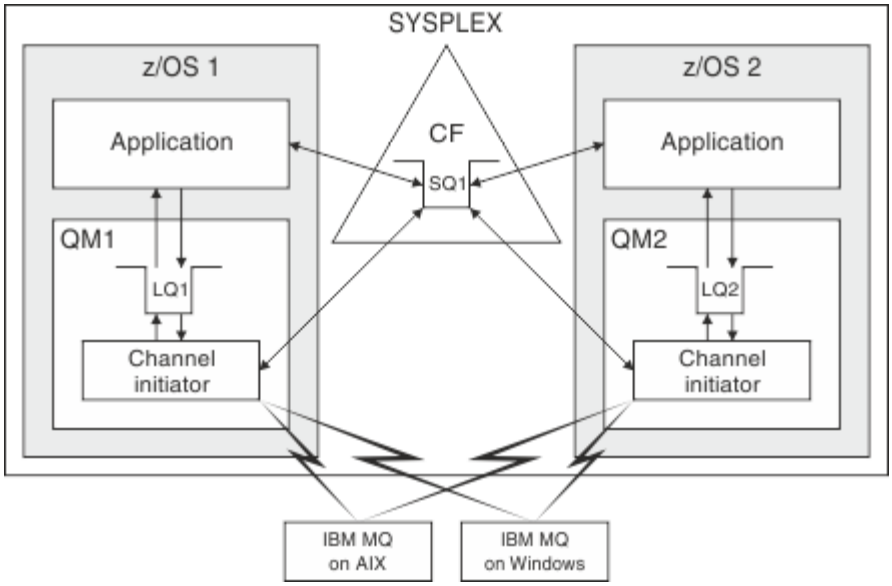


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 176
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 181
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Clusters and queue sharing groups" on page 178
Use this topic to understand how you can use queue sharing groups with clusters.

z/OS *Shared channels*

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. The network products make a *generic port* available for inbound network connection requests, and the inbound request can be satisfied by connecting to one of the eligible servers.

These networking products include:

- VTAM generic resources
- SYSPLEX Distributor

The channel initiator takes advantage of these products to use the capabilities of shared queues

There are two types of shared channels, *shared inbound channel*, and the *shared outbound channel*.

- Shared inbound channels
- Shared outbound channels

For further information about channels see

- Shared channel summary
- Shared channel status

Shared inbound channels

Each channel initiator in the queue sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network by one of the supporting technologies (VTAM, TCP/IP). Inbound network attach requests to the generic port are dispatched by the network technology, to any one of the listeners in the queue sharing group (QSG) that are listening on the generic port.

You can start a channel on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. You can define a channel definition to be private to a queue manager or stored on the shared repository and so available anywhere (a global definition). This means that you can make a channel definition available on any channel initiator in the queue sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue sharing group and not with an individual queue manager. For example, consider a remote queue manager starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed is still correct because the synchronization is with the queue sharing group.

You can use an inbound channel started through the generic port to put messages to any queue. The remote queue manager does not know whether the target queue is shared or not. If the target queue is a shared queue, the remote queue manager connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue.

If the target queue is a private queue, the messages are put to the private queue owned by which ever queue manager the current instance of the channel is connected to. In this environment, known as *replicated local queues*, each queue manager must have the same set of private queues defined.

Configuring SVRCONN channels for a queue sharing group

The optimal configuration for SVRCONN channels in a queue sharing group is to set up private listeners in each CHINIT which use a different port number from the point to point channels. These listener ports are then used as the 'back-end' resources for a new workload distribution mechanism such as Sysplex Distributor using Virtual IP addresses (VIPA). The external VIPA address is then used as the target address for the CLNTCONN definitions in the network. The SVRCONN channel can be defined with QSGDISP(GROUP) so the same definition is available to all queue managers in the QSG. This configuration avoids using a shared listener, and therefore reduces the performance effect of the queue sharing group maintaining shared channel state, which is not needed for client/server channels.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing for shared outbound channels

An outbound shared channel is eligible for starting on any channel initiator within the queue sharing group, if you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by IBM MQ is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a Db2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from private channels in the following ways:

Private channel
Tied to a single channel initiator.

- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel
Workload balanced with high availability.

- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.

You specify whether a channel is private or shared when you start the channel by using CHLDISP options with the START CHANNEL command. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, IBM MQ performs workload balancing and starts the channel on the most appropriate channel initiator within the queue sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue sharing group maintain a shared channel-status table in Db2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue sharing group.

z/OS *Clusters and queue sharing groups*

Use this topic to understand how you can use queue sharing groups with clusters.

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue sharing group (the shared queue is not advertised as being hosted by the queue sharing group). Clients can start sessions with any members of the queue sharing group to put messages to the same shared queue.

Figure 66 on page 179 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

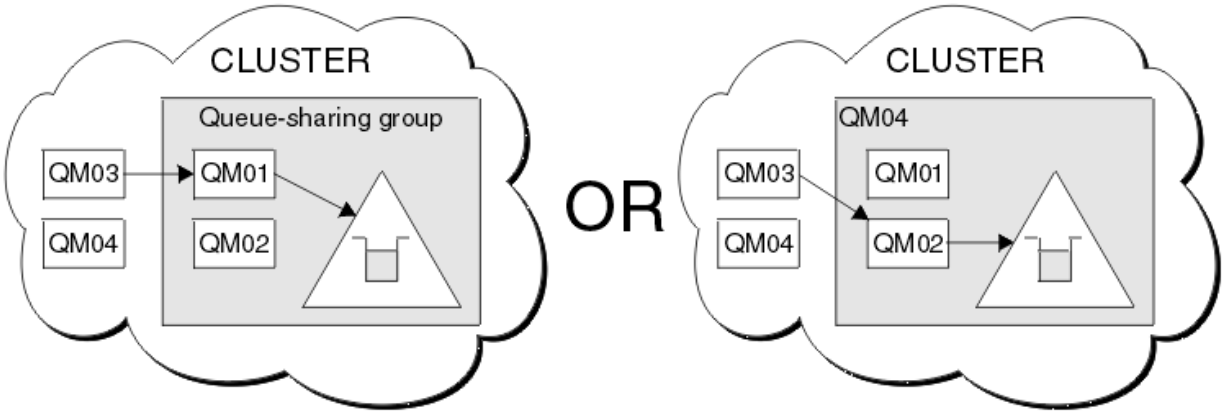


Figure 66. A queue sharing group as part of a cluster

z/OS Influencing workload distribution with shared queues

Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

IBM MQ does not provide workload balancing for shared queues. However, workload distribution in a queue sharing group (QSG) can be influenced in *a pull based fashion*. The choice of which queue manager services a queue (receives a message written to a shared queue) is affected by the available processing capacity of each queue manager in the queue sharing group and the workload management goals defined across the sysplex.

However, it is important to appreciate, the queue manager that performs the MQPUT of a message can also have a large influence in deciding which queue manager gets the message.

A local queue manager is more likely to perform the MQGET

For an application performing an MQPUT, the local queue manager is said to be the queue manager to which the application is connected.

Exactly which queue manager services an MQPUT of a message by performing an MQGET on behalf of a getting application is influenced by the following considerations.

When a message is put to an empty shared queue, the local queue manager is typically posted before any of the other queue manager in the queue sharing group is notified. If the local queue manager is in a position to process the message, it receives a list transition notification from the coupling facility (CF) before any other queue manager in the QSG. (A list transition notification is a notification that the shared queue has changed state from empty to non-empty.)

The possible scenarios, in this case, are as follows:

1. MQPUT of nonpersistent message out of sync point and *fast put to waiting getter*.

   If there is an application with an *MQGET with wait* on the local queue manager for the queue, then the MQPUT of the message is passed directly to the getting application's buffer and not written to the queue. This is true for shared and non-shared queues. This feature is often called *fast put to a waiting getter* mechanism. In the case of shared queues, no other queue manager in the QSG is notified as there is no transition from empty to non-empty of the queue. This means, for example, that provided this queue manager can service all the puts from this application and assuming that no other applications are putting messages to the queue, then no other queue manager in the queue sharing group assists in draining this queue. If however there is no MQGET with wait on the local queue manager, and a message is put to the shared queue then the CF will notify other queue managers in the queue sharing group according to its rules for notifications of list transitions.
2. MQPUT of a persistent or in-syncpoint message.

In this case, if there is an application with an *MQGET with wait* on the local queue manager, then the message is put to the shared queue and the CF notifies other queue managers in the queue sharing group according to its rules for notifications of list transitions. However, the local queue manager does not wait for a transition notification from the CF but honors any local *MQGET with wait* first and usually performs the get of this message on behalf of the application before any other queue manager in the queue sharing group can respond to a CF notification. This is dependent on how busy the local queue manager is. Otherwise, any queue manager notified by the CF due to the arrival of the message on the empty queue will try to service the get first. The first queue manager to respond processes the new message.

3. Finally, if the queue is not drained of messages, where the CF has sent a notification of a state change from empty to non-empty for the queue, all connected queue managers will have an opportunity to assist in the processing of the queue. In this event, the workload is said to be *pull based*.

This design allows for the improved performance over a purely pull based workload distribution. The aim is to take advantage of the high availability offered by queues held in the CF while allowing the queue manager, where possible, to perform the MQGET without needing to reference the CF and so to process the message workload as efficiently as possible.

Alternative approaches can be adopted where emphasis on balance of the workload is more important than the previously described performance enhancements. For example, ensuring that none of the getting applications are connected to the same queue manager that the putting application is connected to. Using this design all messages are put to the queue and all queue managers in the QSG are notified when the queue moves from empty to non-empty, in accordance with the CF algorithm for handling such transitions. In addition, the *fast put to waiting getter* mechanism is not applicable.

z/OS Where to find more information about shared queues and queue sharing groups

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

Table 19. Where to find more information about shared queues and queue sharing groups

Topic	Where to look
Queue sharing group recovery	“Recovery and restart on z/OS” on page 219
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 235
Private and global object definitions Directing commands to different queue managers	Sources from which you can issue commands on z/OS
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	“Shared queues and queue sharing groups” on page 137
Utility programs Migrating queues	IBM MQ utilities on z/OS reference

Table 19. Where to find more information about shared queues and queue sharing groups (continued)	
Topic	**Where to look**
Console messages	Messages for IBM MQ for z/OS
MQSC commands	MQSC commands
IBM MQ clusters	Configuring a queue manager cluster
IBM MQ distributed queuing Channel names	Introduction to distributed queue management
Writing applications	Overview of application design
MQCONNX call	MQCONNX

z/OS Intra-group queuing

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

For information about queue sharing groups, see “Shared queues and queue sharing groups” on page 137.

Intra-group queuing concepts

You can perform fast message transfer between queue managers in a queue sharing group without defining channels. This uses a system queue called the SYSTEM.QSG.TRANSMIT.QUEUE, which is a shared transmission queue. Each queue manager in the queue sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing (IGQ) is enabled and the target queue manager is within the queue sharing group, the SYSTEM.QSG.TRANSMIT.QUEUE is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves nonpersistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, nonpersistent messages are discarded, but persistent messages are backed out and returned to the SYSTEM.QSG.TRANSMIT.QUEUE, and the IGQ agent tries to deliver the messages until it is successful.

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue sharing group can use intra-group queuing to *hop* the message to the correct destination.

There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager. You can use the queue manager attribute SQQMNAME to control this. If you set the value of SQQMNAME to USE, the MQOPEN command is performed on the queue manager specified by the **ObjectQMgrName**.

However, if the target queue is a shared queue and you set the value of SQQMNAME to IGNORE, and the **ObjectQMgrName** is that of another queue manager in the queue sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a

message to the queue, the message is transferred to the specified **ObjectQMgrName** through either IGQ or an IBM MQ channel.

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue sharing group. Intra-group queuing also supports large messages, the largest being 100 MB *minus* the length of the transmission queue header.

Note: If you use this feature, users must have the same access to the queues on each queue manager in the queue sharing group.

The following diagram shows a typical example of intra-group queuing.

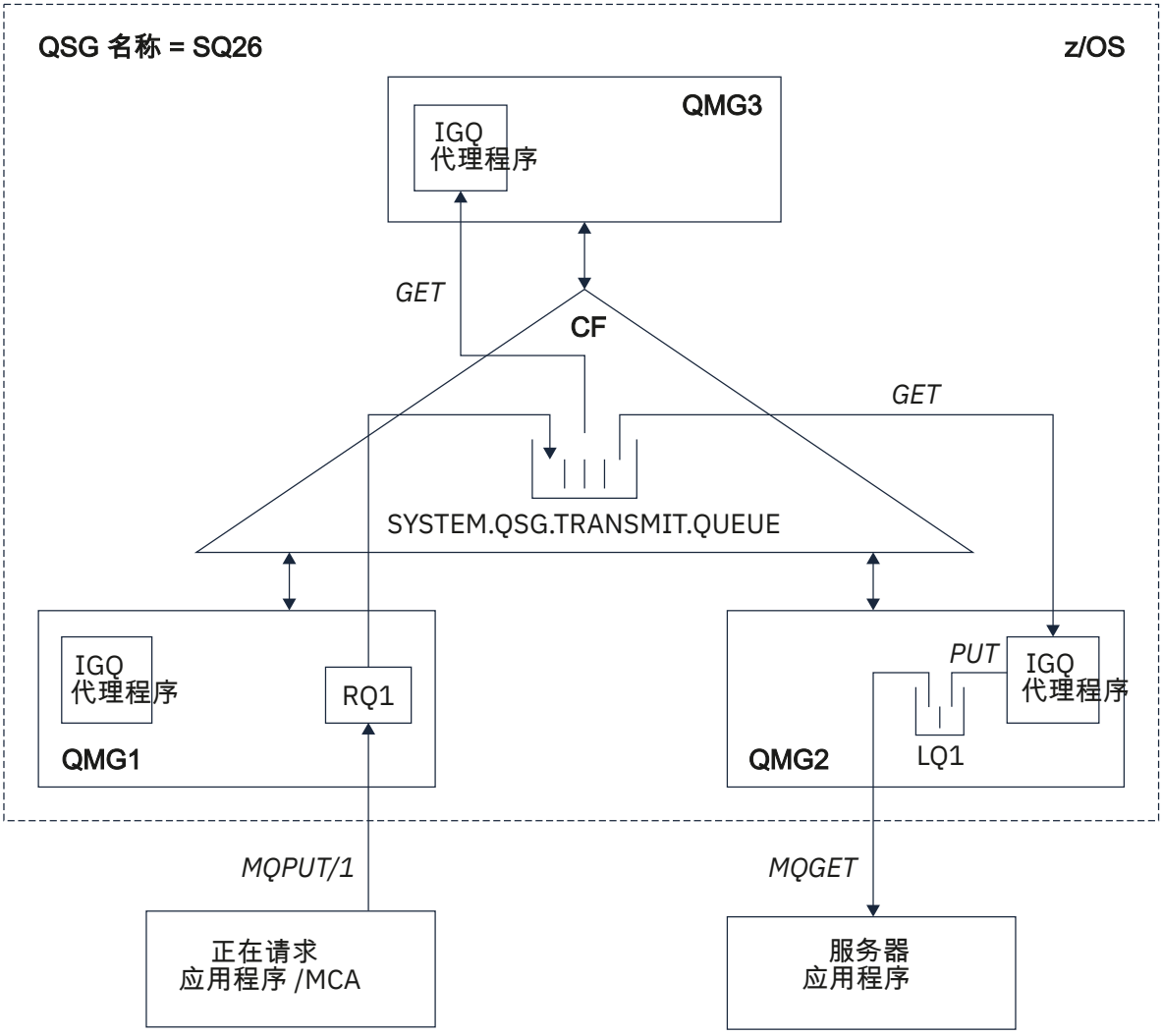


Figure 67. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2, and QMG3) that are defined to a queue sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the coupling facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this application could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.

- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing is used for message transfer. If intra-group queuing is to be used, the local queue manager places the message on to the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Intra-group queuing terminology

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queues.

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

z/OS Benefits of intra-group queuing

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue sharing group.

The benefits of intra-group queuing are:

Reduced system definitions
Intra-group queuing removes the need to define channels between queue managers in a queue sharing group.

Reduced system administration
Because there are no channels defined between queue managers in a queue sharing group, there is no requirement for channel administration.

Improved performance
Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination queue manager for

delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in sync point scope, committed.

Supports migration

Applications external to a queue sharing group can deliver messages to a queue residing on any queue manager in the queue sharing group, while being connected only to a particular queue manager in the queue sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue sharing group without the need to change any systems that are external to the queue sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue sharing group SQ26.

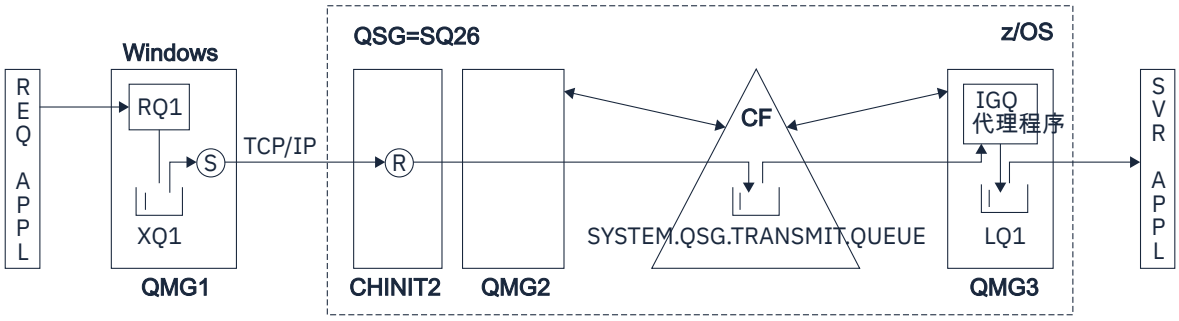


Figure 68. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, using TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Delivery of messages when multi-hopping between queue managers in a queue sharing group

The previous diagram in Supports migration also illustrates the delivery of messages when multi-hopping between queue managers in a queue sharing group. Messages arriving on a queue manager within the queue sharing group, but destined for a queue on another queue manager in the queue sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

z/OS Limitations of intra-group queuing

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

This topic describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing
Because intra-group queuing uses a shared transmission queue that is defined in the coupling facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager
Only one IGQ agent is started per queue manager in a queue sharing group.

Starting and stopping the intra-group queuing agent
The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the SYSTEM.QSG.TRANSMIT.QUEUE (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent encounters an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command. This command avoids the need to recycle the queue manager.

Setting the queue manager attribute IGQ to ENABLED or DISABLED
If the queue manager attribute IGQ is set to ENABLED or DISABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See Getting started with intra-group queueing for more information.

z/OS Getting started with intra-group queuing

You can enable, disable, and use intra-group queuing as described in this topic.

Enabling intra-group queuing
To enable intra-group queuing on your queue managers, you need to do the following:

- Define a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE. The definition of this queue can be found in thlqual.SCSQPROC(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROC(CSQ4INSS), for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing. The IGQ agent processes any messages that are placed on the SYSTEM.QSG.TRANSMIT.QUEUE. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to ENABLED.

  Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 193 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 185 for further information.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. If intra-group queuing is disabled for a particular queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 193 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 185 for further information.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to change user applications, or to application queues, because for eligible messages the queue manager uses the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

z/OS Intra-group queuing configurations

In addition to the typical intra-group queuing configuration, other configurations are possible.

“Intra-group queuing concepts” on page 181 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 186
For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

“Clustering with intra-group queuing (multiple delivery paths)” on page 188
It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

“Clustering, intra-group queuing and distributed queuing” on page 190
It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

z/OS *Distributed queuing with intra-group queuing (multiple delivery paths)*

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

The choice of intra-group queuing over channel communications can be controlled by the CFSTRUCT type level. (3 instead of 4 or 5). The maximum message length as set on the SYSTEM.QSQ.TRANSMIT.QUEUE.

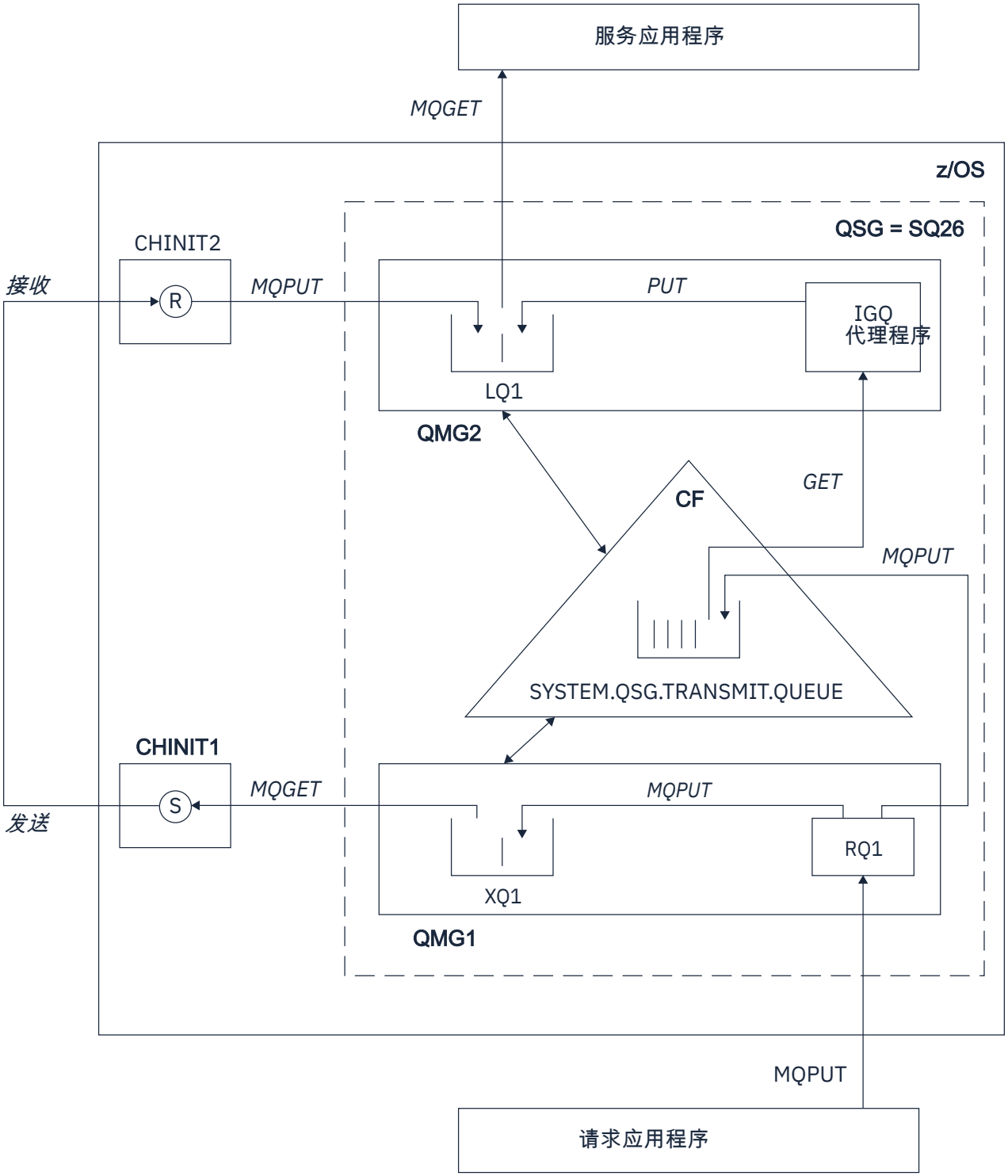


Figure 69. An example configuration

Open/Put processing

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, based on whether intra-group queuing is enabled for outbound transfer on the queue manager and on the message characteristics, the message is put to transmission queue XQ1, or to transmission queue

SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.

3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
4. If transmission queue SYSTEM.QSG.TRANSMIT.QUEUE is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for large messages continue to succeed and are placed on transmission queue XQ1. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages

1. The requesting application puts large messages to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue XQ1.
3. Sender MCA (S) on queue manager QMG1 retrieves the messages from transmission queue XQ1 and sends them to queue manager QMG2.
4. Receiver MCA (R) on queue manager QMG2 receives the messages and places them on to destination queue LQ1.
5. The serving application retrieves and then processes the messages from queue LQ1.

Flow for small messages

1. The requesting application puts small messages on to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. IGQ on queue manager QMG2 retrieves the messages and places them on to the destination queue LQ1.
4. The serving application retrieves the messages from queue LQ1.

Points to note

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence (though this delivery is also possible if messages are delivered using only the normal channel route).
5. When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to transmission queue XQ1.

z/OS *Clustering with intra-group queuing (multiple delivery paths)*

It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue sharing group, intra-group queuing is used for the delivery of small messages (using the SYSTEM.QSG.TRANSMIT.QUEUE), and the delivery of large messages if intra-group queuing supports the size of the message. Also, the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue sharing group. The following diagram illustrates this configuration (the channel initiators are not shown).

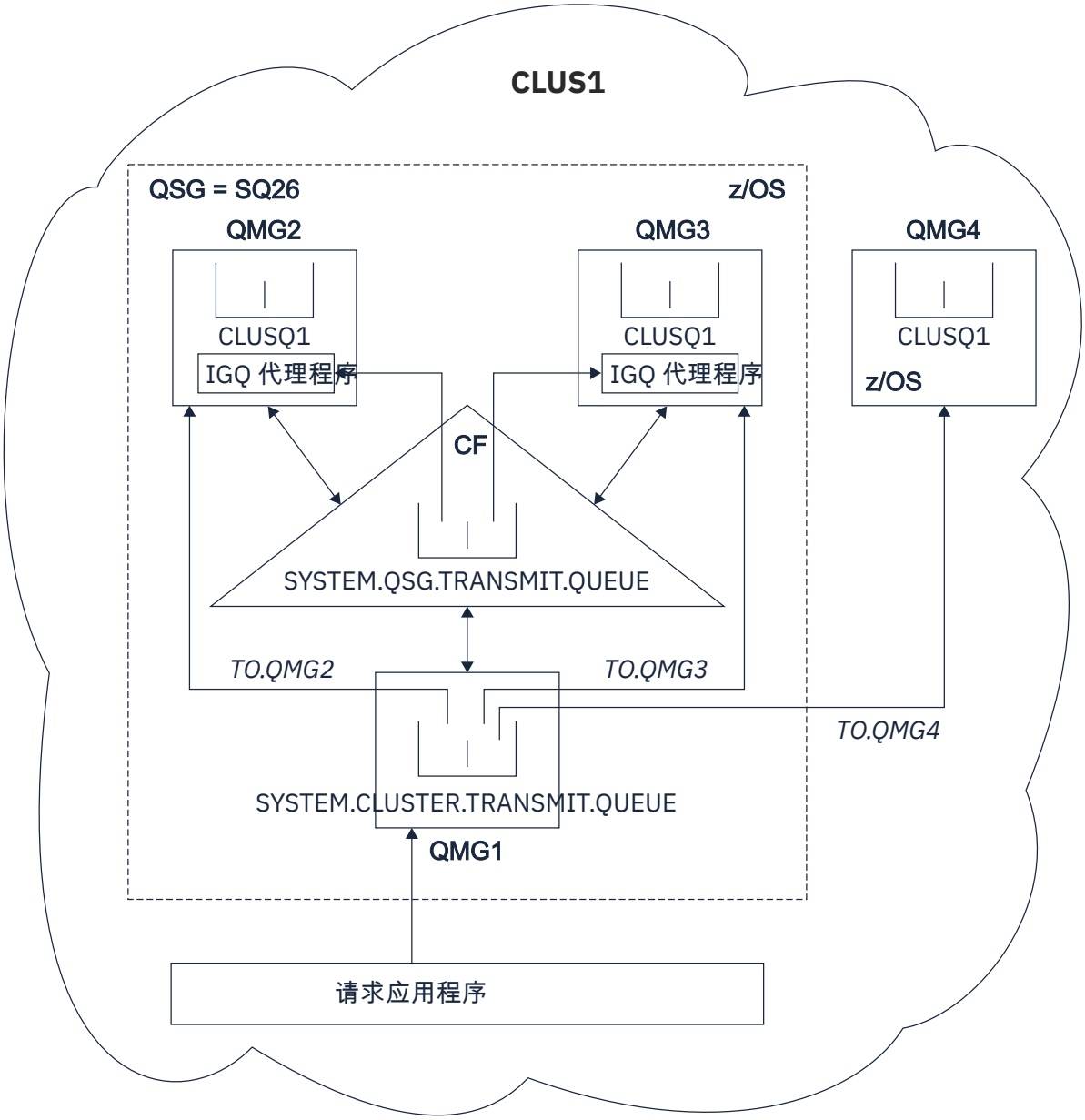


Figure 70. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3, and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2, and QMG3 configured in a queue sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.

  Note: For clarity, the SYSTEM.CLUSTER.TRANSMIT.QUEUE on the other queue managers not shown.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in an IBM MQ structure configured with the CFLEVEL(3) RECOVER(YES) attribute.
- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3, and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are:
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1, because SYSTEM.QSG.TRANSMIT.QUEUE is in a CFLEVEL(3) structure; therefore supports messages only up to 63 KB in size.
  - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
  - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. This queue is in a structure configured with the RECOVER(YES) attribute, so is used for both persistent, and non-persistent, small messages.
  - Retrieved by the IGQ agent on QMG2
  - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue sharing group SQ26, all messages put by the requesting application are
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
  - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence. It is important to note that this delivery is possible without regard to the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, MQOO_BIND_ON_GROUP, or MQOO_BIND_AS_Q_DEF is specified on open.
- When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

z/OS *Clustering, intra-group queuing and distributed queuing*

It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

This configuration is a combination of distributed queuing with intra-group queuing and clustering with intra-group queuing.

Intra-group queuing is described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 186.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 188.

z/OS Intra-group queuing messages

This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In IBM WebSphere MQ 5.3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed might be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of sync point scope, and all persistent messages within sync point scope. In this case, the IGQ agent acts as the sync point coordinator. The IGQ agent therefore processes nonpersistent messages like the way fast, nonpersistent messages are processed on a message channel. See Fast, nonpersistent messages.

Batching of messages

The IGQ agent uses a fixed batch size of 50 messages. Any persistent messages retrieved within a batch are committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the normal rules are applied in the selection of the queue that has default priority and persistence values that are used. (See the IBM MQ messages section for more information about the rules of queue selection).

Related concepts

“Undelivered/unprocessed messages” on page 191
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 192
This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

z/OS *Undelivered/unprocessed messages*

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honors the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it must) and discards the undelivered message.

- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see “Specific properties of intra-group queuing” on page 193.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages are lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent might not be able to process these messages and they remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users then have to use their own methods to deal with these unprocessed messages.

z/OS *Report messages - Intra Group Queuing*

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages
COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages
Expiry report messages are generated by the queue manager.

Exception report messages

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent generates the required exception report and places it on the specified reply-to queue. Intra-group queuing can be used to deliver the exception report to the destination reply-to queue.

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If it is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages, and enters a state of retry. For more information, see “Specific properties of intra-group queuing” on page 193.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

z/OS Security for intra-group queuing

This topic describes the security arrangements for intra-group queuing.

Queue manager attributes IGQAUT (IGQ authority) and IGQUSER (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The IGQAUT attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The IGQAUT attribute is analogous to the PUTAUT attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The IGQUSER attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The IGQUSER attribute is analogous to the MCAUSER attribute that is available on channel definitions.

z/OS Specific properties of intra-group queuing

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following rules for object handle invalidation:

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

Retry capability of the intra-group queuing agent

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Table 20. Short and long retry counts and intervals values

Constant	Value
Short retry count	10
Short retry interval	60 seconds = 1 min
Long retry count	999,999,999

Table 20. Short and long retry counts and intervals values (continued)	
Constant	**Value**
Long retry interval	1200 seconds = 20 min

The intra-group queuing agent and serialization

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

If there is a failure of a queue manager in a queue sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the IGQ agent ends, and shared queue peer recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, it leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. Which in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONNX API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

z/OS Storage management on z/OS

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

Related concepts

“Page sets for IBM MQ for z/OS” on page 194
Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

“Storage classes for IBM MQ for z/OS” on page 195
A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

“Buffers and buffer pools for IBM MQ for z/OS” on page 197
IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

Related reference

“Where to find more information about storage management for IBM MQ for z/OS” on page 198
Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

z/OS Page sets for IBM MQ for z/OS

Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

A *page set* is a VSAM linear data set that has been specially formatted to be used by IBM MQ. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on Db2, and the messages on shared queues. These are not stored on the queue manager page sets. For more information about shared queues, see “Shared queues and queue sharing groups” on page 137, and for more information about global definitions, see Private and global definitions.

IBM MQ page sets can be up to 64 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

IBM MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of IBM MQ it is essential that page set zero does not become full, so do not use it to store messages.

To improve the performance of your system, you should also separate short-lived messages from long-lived messages by placing them on different page sets.

You must format page sets, and IBM MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT). Page sets must also be defined to the IBM MQ subsystem.

IBM MQ for z/OS can be configured to expand a page set dynamically if it becomes full. IBM MQ continues to expand the page set if required until 123 logical extents exist, if there is sufficient disk storage space available. The extents can span volumes if the linear data set is defined in this way, however, IBM MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one IBM MQ queue manager on a different IBM MQ queue manager, or change the queue manager name. If you want to transfer the data from one queue manager to another, you must unload all the objects and messages from the first queue manager and reload them onto another.

It is not possible to use page sets greater than 4 GB in a queue manager running a release earlier than V6. During the migration period, when it is likely that you might need to fall back to a previous release of code:

- Do not change page set 0 to be greater than 4 GB.
- Other page sets greater than 4 GB will be left offline when restarting a queue manager with a previous release.

For further information about migrating existing page sets capable of expanding beyond 4 GB, see Defining a page set to be larger than 4 GB.

It is possible for an administrator to dynamically add page sets to a running queue manager, or remove page sets from a running queue manager (except for page set zero). The DEFINE PSID command can run after the queue manager restart has completed, only if the command contains the DSN keyword.

z/OS Storage classes for IBM MQ for z/OS

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

Introducing storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. You can also use storage classes to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in “IBM MQ and IMS” on page 250).

Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you specify the storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 71 on page 196, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

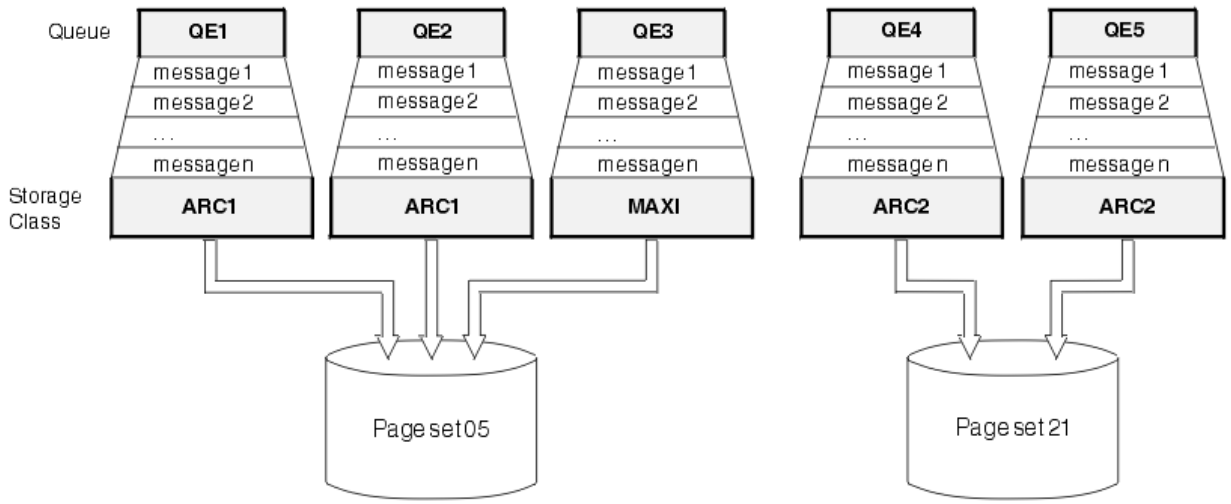


Figure 71. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, IBM MQ uses a default storage class.

If a message is put on a queue that names a nonexistent storage class, the application receives an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

You can change a storage class only when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

z/OS Buffers and buffer pools for IBM MQ for z/OS

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type separation outlined in Figure 72 on page 197, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the **DEFINE BUFFPOOL** command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Figure 72 on page 197 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

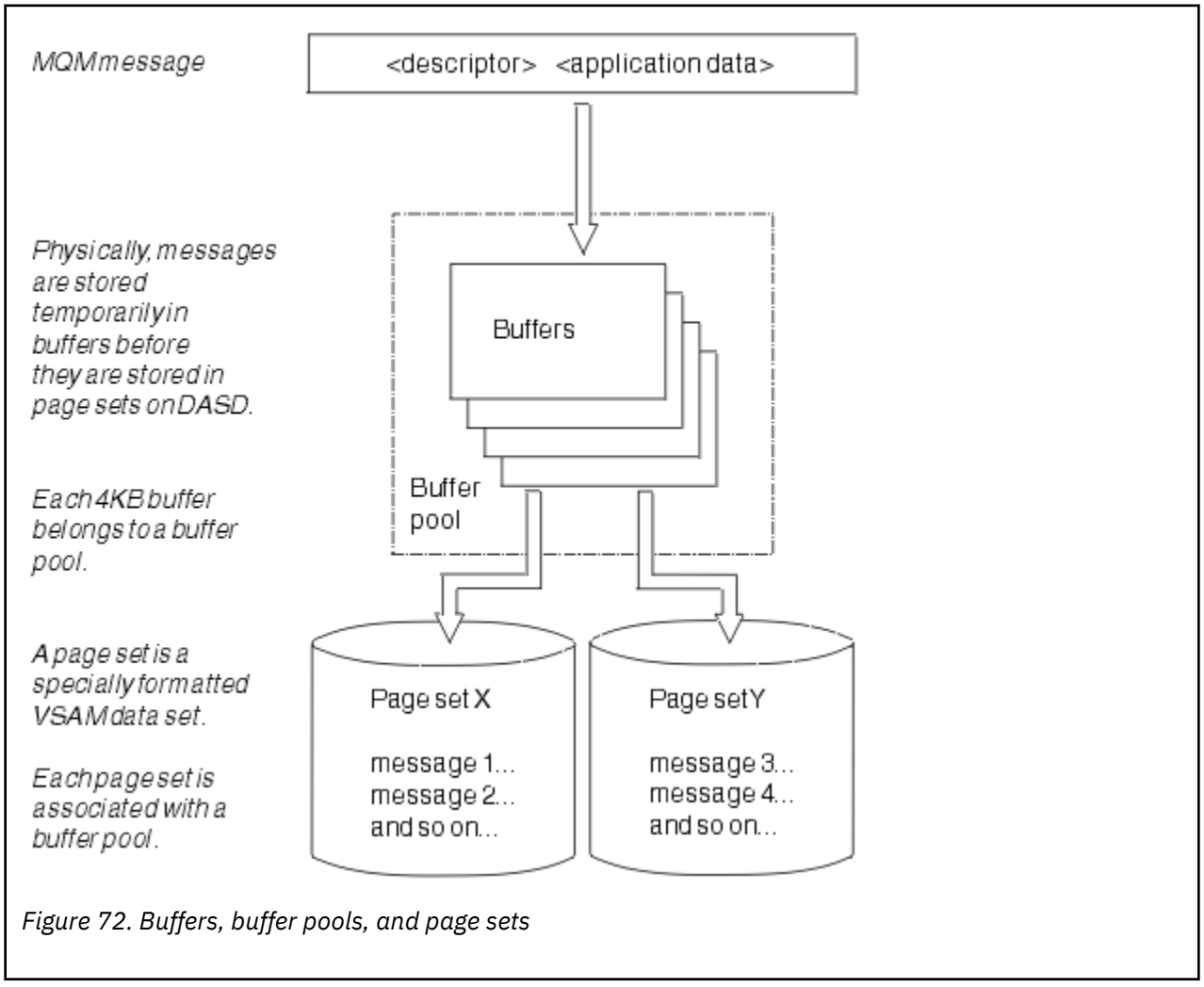


Figure 72. Buffers, buffer pools, and page sets

You can dynamically issue commands to modify buffer pool size, and location, using the **ALTER BUFFPOOL** command. Page sets can be dynamically added by using the **DEFINE PSID** command, or deleted by using the **DELETE PSID** command.

If a buffer pool is too small, IBM MQ issues message CSQP020E. You can then dynamically add more buffers to the affected buffer pool (note that you may have to remove buffers from other buffer pools to do this).

You specify the number of buffers in a pool with the **DEFINE BUFFPOOL** command, and you can dynamically resize buffer pools with the **ALTER BUFFPOOL** command. You determine the current number of buffers in a pool dynamically by displaying a page set that uses the buffer pool, using the **DISPLAY USAGE** command.

For performance reasons, do not put messages and object definitions in the same buffer pool. Use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, keep short-lived messages and long-lived messages in different buffer pools and therefore on different page sets, and in different queues.

The **DEFINE BUFFPOOL** command cannot be used after restart to create a new buffer pool. Instead, if a **DEFINE PSID** command uses the DSN keyword, it can explicitly identify a buffer pool that is not currently defined. That new buffer pool will then be created.

z/OS Where to find more information about storage management for IBM MQ for z/OS

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 21. Where to find more information about storage management

Topic	Where to look
How much storage you need	Planning your storage and performance requirements on z/OS
How large to make your page sets and buffer pools	Plan your page sets and buffer pools
Managing page sets	Managing page sets
MQSC commands	The MQSC commands

z/OS Logging in IBM MQ for z/OS

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

The log does not contain information for statistics, traces, or performance evaluation. For further details about the statistical and monitoring information that IBM MQ collects, see Monitoring and statistics.

For more information about logging, see the following topics:

- “Log files in IBM MQ for z/OS” on page 199
- “How the log is structured” on page 202
- “How the IBM MQ for z/OS logs are written” on page 203
- “Larger log Relative Byte Address” on page 206
- “The bootstrap data set” on page 207

Related tasks

Planning your logging environment

Setting logs using the system parameter module

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS Log files in IBM MQ for z/OS

Log files contain information needed for transaction recovery. Active log files can be archived so that you can keep log data for a long period.

What is a log file

IBM MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- IBM MQ objects, such as queues
- The IBM MQ queue manager

The active log comprises a collection of data sets (up to 310) which are used cyclically.

You can enable log archiving so that when an active log fills a copy is made in an archive data set. Using archiving allows you to keep log data for an extended period. If you do not use archiving, the logs wrap and earlier data is overwritten. To recover a page set, or recover data in a CF structure, you need log data from when the backup of the page set or structure was taken. An archive log can be created on disk or on tape.

Archiving

Because the active log has a fixed size, IBM MQ copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct-access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, IBM MQ uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. You can catalog each data set using the z/OS integrated catalog facility (ICF).

Archiving is an essential component of IBM MQ recovery. If a unit of recovery is a long-running one, log records within that unit of recovery might be found in the archive log. In this case, recovery requires data from the archive log. However, if archiving is disabled, the active log with new log records wraps, overwriting earlier log records. This means that IBM MQ might not be able to back out the unit of recovery and messages might be lost. The queue manager then terminates abnormally.

Therefore, in a production environment, **never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in Using CSQ6LOGP.

To help prevent problems with unplanned long-running units of work, IBM MQ issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload processing.

Dual logging

In dual logging, each log record is written to two different active log data sets to minimize the likelihood of data loss problems during restart.

You can configure IBM MQ to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. Each active log data set is a single-extent VSAM linear data set (LDS). With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart.

Log shunting

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at queue manager restart, or backout, for long running or long term in-doubt units of work.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This technique is known as *shunting*. When the whole of the unit of work has been processed, the unit of work is in a *shunted* state. Any backout or restart activity relating to the shunted unit of work can use the shunted log records instead of using the original unit of work log records.

Detecting a long-running unit of work is a function of the checkpoint process. At checkpoint time, each active unit of work is checked to establish whether it needs to be shunted. If the unit of work has been through two prior checkpoints since it was created, or since it was last shunted, the unit of work is suitable to be shunted. This means that a single unit of work might be shunted more than once. This is known as a *multi-shunted* unit of work.

A unit of work is shunted every three checkpoints. However the checkpoint is performed asynchronously to the log-switch (or the writing of the log record which caused LOGLOAD to be exceeded).

There is only a single checkpoint taking place at a time, so there might be multiple log-switches before a checkpoint completes.

This means that if there are not enough active logs, or if they are too small, then shunting of a large unit of work might not complete before all the logs are filled.

Message CSQR027I results if shunting is unable to complete.

If log archiving is turned off, ABEND 5C6 with reason 00D1032A occurs if there is an attempt to back out the unit of work for which shunting failed. To avoid this problem you should use OFFLOAD=YES.

Log shunting is always active, and runs whether log archiving is enabled or not.

Note: Although all log records for a unit of work are shunted, the entire content of each record is not shunted, only the part that is necessary for backout. This means that the amount of log data written is kept to a minimum, and that shunted records cannot be used if a page set failure occurs. A long running unit of work is one that has been running for more than three queue manager checkpoints.

For more information about log shunting, see Managing the logs.

Log compression

You can configure IBM MQ for z/OS to compress and decompress log records as they are written and read from the log data set.

Log compression can be used to reduce the amount of data written to the log for persistent messages on private queues. The amount of compression that is achieved depends on the type of data contained within messages. For example, Run Length Encoding (RLE) works by compacting repeated instances of bytes which can give good results efficiently for structured or record oriented data.

Attention: Persistent messages that are being put to a shared queue are not subject to log compression.

You can use fields within the Log manager section of the System Management Facility 115 (SMF) records to monitor how much data compression is achieved. For more information about SMF, see Using the System Management Facility and Accounting and statistics messages.

Log compression increases the processor utilization of the system. You should only consider using compression if throughput of your queue manager is constrained by the IO bandwidth writing to the log

data sets or you are constrained by the disk storage needed to hold log data sets. If you are using shared queues then IO bandwidth constraints can be relieved by adding additional queue managers to the queue sharing group and distributing the workload across more queue managers.

The log compression option can be enabled and disabled as required without the need to stop and restart the queue manager. The queue manager can read any compressed log records regardless of the current log compression setting.

The queue manager supports 3 settings for log compression.

NONE
No log data compression is used. This is the default value.

RLE
Log data compression is performed using run-length encoding (RLE).

ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. This option results in RLE compression.

You can control the compression of log records using one of the following:

- The SET and DISPLAY LOG commands in MQSC; see SET LOG and DISPLAY LOG
- The Set Log and Inquire Log functions in the PCF interface; see Set log and Inquire log
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 18 million million million ($1.8*10^{19}$) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The RBA is referenced by a 6-byte or 8-byte field giving a total addressable range of 2^{48} bytes, or 2^{64} bytes, depending on whether 6-byte or 8-byte log RBAs are in use.

However, when IBM MQ detects that the used range is beyond F00000000000 (if 6-byte RBAs are in use) or FFFF800000000000 (if 8-byte log RBAs are in use), messages CSQI045, CSQI046, CSQI047, and CSQJ032 are issued, warning you to reset the log RBA.

If the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) the queue manager terminates with reason code 00D10257.

Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the queue manager can be converted to use 8-byte log RBAs, or the log can be reset. The procedure to reset the log is documented in Resetting the queue manager's log.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the procedure documented in Implementing the larger log Relative Byte Address.

The log consists of *log records*, each of which is a set of log data treated as a single unit. A log record is identified either by the RBA of the first byte of its header, or by its log record sequence number (LRSN). The RBA or LRSN uniquely identifies a record that starts at a particular point in the log.

Whether you use the RBA or LRSN to identify log points depends on whether you are using queue sharing groups. In a queue-sharing environment, you cannot use the relative byte address to uniquely identify a log point, because multiple queue managers can update the same queue at the same time, and each has its own log. To solve this, the log record sequence number is derived from a timestamp value, and does not necessarily represent the physical displacement of the log record within the log.

Each log record has a header that gives its type, the IBM MQ subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are four types of log record, described under the following headings:

- Unit of recovery log records
- Checkpoint records
- Page set control records
- CF structure backup records

Unit of recovery log records

Most of the log records describe changes to IBM MQ queues. All such changes are made within units of recovery.

IBM MQ uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that the queue manager has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be reapplied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, IBM MQ takes periodic checkpoints during normal operation. These occur as follows:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in Using CSQ6SYSP.
- At the end of a successful restart.
- At normal termination.
- Whenever IBM MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, IBM MQ issues the DISPLAY CONN command (described in DISPLAY CONN) internally so that a list of connections currently in doubt is written to the z/OS console log.

Page set control records

These records register the page sets and buffer pools known to the IBM MQ queue manager at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

Certain dynamic changes to page sets and buffer pools are also written as page set control records, so that the changes can be recovered and automatically reinstated at the next queue manager restart.

CF structure backup records

These records hold data read from a coupling facility list structure in response to a BACKUP CFSTRUCT command. In the unlikely event of a coupling facility structure failure, these records are used, together with unit of recovery records, by the RECOVER CFSTRUCT command to perform media recovery of the coupling facility structure to the point of failure.

Related tasks

Implementing the larger log Relative Byte Address

z/OS How the log is structured

Use this topic to understand the terminology used to describe log records.

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information logged at a particular time forms a *logical* record, with a length that varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term *log record* refers to the *logical* record, regardless of how many *physical* records are needed to store it.

z/OS How the IBM MQ for z/OS logs are written

Use this topic to understand how IBM MQ processes log file records.

IBM MQ writes each log record to a DASD data set called the *active log*. When the active log is full, IBM MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 73 on page 204 illustrates the process of logging. Log records typically go through the following cycle:

1. IBM MQ notes changes to data and significant events in recovery log records.
2. IBM MQ processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM Controls Intervals (CI). Each log record is identified by a relative byte address in the range zero through 2^{64} -1.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

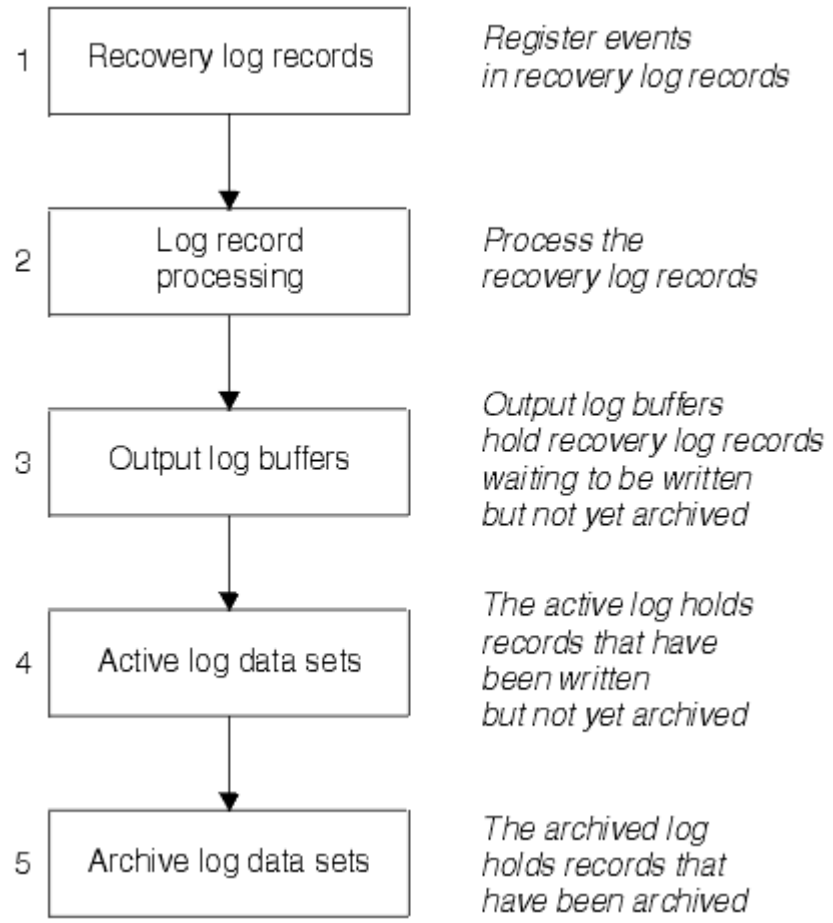


Figure 73. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point, or when an IBM MQ BACKUP CFSTRUCT command is issued.

When the queue manager is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by the queue manager and remain allocated exclusively to IBM MQ until the queue manager terminates.

Dynamically adding log data sets

It is possible to dynamically define new active log data sets while the queue manager is running. This feature alleviates the problem of a queue manager hang when archiving is not able to offload active logs due to a transient problem. See the DEFINE LOG command for more information.

Note: To redefine or remove active logs you must terminate and restart the queue manager.

IBM MQ and Storage Management Subsystem

IBM MQ parameters enable you to specify Storage Management Subsystem (MVS™/DFP SMS) storage classes when allocating IBM MQ archive log data sets dynamically. IBM MQ initiates the archiving of log data sets, but you can use SMS to perform allocation of the archive data set.

Related reference

“When the IBM MQ for z/OS archive log is written” on page 205
Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

z/OS *When the IBM MQ for z/OS archive log is written*

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 74 on page 205.

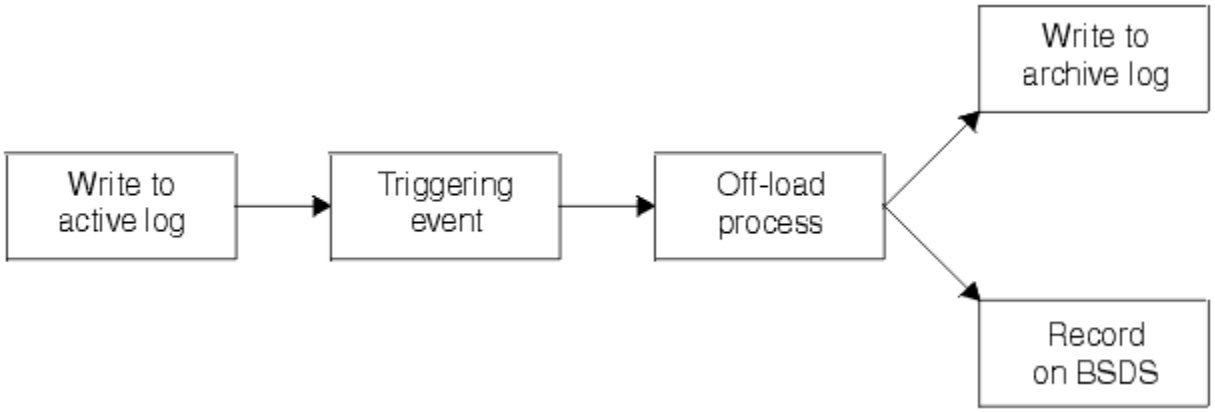


Figure 74. The offloading process

Triggering the offloading process

The offload process of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSC ARCHIVE LOG command.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that was not written becomes the first record of the new data set. Offloading is triggered for the truncated data set as for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 5% full and at 5% increments thereafter, stating the percentage of the log's capacity that is in use. If all the active logs become full, IBM MQ stops processing, until offloading occurs, and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The offload process

When all the active logs become full, IBM MQ runs the offloading process and halts processing until the offloading process has been completed. If the offload processing fails when the active logs are full, IBM MQ abends.

When an active log is ready to be offloaded, a request is sent to the z/OS console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (for further information, see Using CSQ6ARVP) determines whether the request is received. If you are using tape for offloading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the offload process. It does not affect IBM MQ performance unless the operator delays the response for so long that IBM MQ runs out of active logs.

The operator can respond by canceling the offload process. In this case, if the allocation is for the first copy of dual archive data sets, the offload process is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while offloading

A request to stop the queue manager does not take effect until offload processing has finished. If IBM MQ fails while offloading is in progress, offloading begins again when the queue manager is restarted.

Messages during offload processing

Offloaded messages are sent to the z/OS console by IBM MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets.

z/OS Larger log Relative Byte Address

This function improves the availability of the queue manager by increasing the period of time before you have to reset the log.

Recovery data is written to the log so that persistent messages are available when the queue manager is restarted. The term log Relative Byte Address (log RBA) is used to refer to the location of data as an offset from the beginning of the log.

Before IBM MQ 8.0, the 6 byte log RBA could address up to 256 terabytes of data. Before this quantity of log records has been written, you have to reset the queue manager's log by following the procedure documented in Resetting the queue manager's log.

Resetting the logs of queue managers is not a quick process, and can require an extended outage, due to the need to reset the page sets as part of the process. For a high use queue manager this operation might typically be done once a year.

From IBM MQ 8.0, the log RBA can be 8 bytes long and the queue manager can now address over 64,000 times as much data (16 exabytes) before the log RBA needs to be reset. The impact of using the larger log RBA is that the size of the log data written increases by a few bytes.

When is this function enabled?

Queue managers created at IBM MQ 9.3.0 or later already have this function enabled.

If the current log RBA is approaching the end of the log RBA range, consider converting the queue manager to use an 8 byte log RBA rather than resetting the queue manager's log. Converting a queue manager to use 8 byte log RBAs requires a shorter outage than resetting the log, and significantly increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6 byte or 8 byte log RBAs are in use.

How is this function enabled?

8 byte log RBA is enabled by starting the queue manager with a version 2 format BSDS. In summary, this is achieved by:

1. Ensuring that all queue managers in the queue sharing group meet the requirements for enabling 8 byte log RBA
2. Shutting down the queue manager cleanly
3. Running the BSDS conversion utility to create a copy of the BSDS in version 2 format.
4. Restarting the queue manager with the converted BSDS.

Once a queue manager has been converted to use 8 byte log RBAs, it cannot go back to using 6 byte log RBA.

See Implementing the larger log Relative Byte Address for the detailed procedure on how to enable 8 byte log RBAs.

Related tasks

Planning to increase the maximum addressable log range

Related reference

The BSDS conversion utility (CSQJUCNV)

z/OS The bootstrap data set

The bootstrap data set is required by IBM MQ as a mechanism to reference log data sets, and log records. This information is required during normal processing, and restart recovery.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by IBM MQ. It contains the following:

- An inventory of all active and archived log data sets known to IBM MQ. IBM MQ uses this inventory to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  IBM MQ stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each portion of the log held in that data set.
- A *wrap-around* inventory of all recent IBM MQ activity. This is needed if you have to restart the queue manager.

The BSDS is required if the queue manager has an error and you have to restart it. IBM MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure IBM MQ with dual BSDSs, each recording the same information. Using dual BSDSs is known as running in *dual mode*. If possible, place the copies on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Use dual BSDSs rather than dual write to DASD.

The BSDS is set up when IBM MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see 管理 IBM MQ for z/OS. It is referenced by a DD statement in the queue manager startup procedure.

Normally, IBM MQ keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. You can restore dual-mode operation, this is described in the 管理 IBM MQ for z/OS.

The active logs are first registered in the BSDS when IBM MQ is installed. You cannot replace the active logs without terminating and restarting the queue manager.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (IBM MQ does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

You can use the following MQSC command to determine the extent of the log, and the name of the active or archive log data set holding the earliest log RBA, required for various types of media or queue manager recovery:

```
DISPLAY USAGE TYPE(DATASET)
```

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

The BSDS version

The format of the BSDS varies according to its version. Increasing the version of the BSDS allows new features to be used. The following BSDS versions are supported by IBM MQ:

Version 1
Supported by all releases of IBM MQ. A version 1 BSDS supports 6-byte log RBA values.

Version 2
Supported by IBM MQ 8.0 and later. A version 2 BSDS enables 8-byte log RBA values, and up to 310 data sets in each active log copy.

Enabled by default for queue managers created at IBM MQ 9.3.0 or later.

Version 3
Supported by IBM MQ 8.0 and later. The BSDS is automatically converted to version 3, from version 2, when more than 31 data sets are added to either active log copy.

You can determine the version of a BSDS by running the print log map utility (CSQJU004). To convert a BSDS from Version 1 to Version 2, run the BSDS conversion utility (CSQJUCNV).

See “Larger log Relative Byte Address” on page 206 for more information on 6-byte and 8-byte log RBAs.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name
CSQ.ARCHLOG1.E00186.T2336229. *A* 0000001

BSDS copy name
CSQ.ARCHLOG1.E00186.T2336229. *B* 0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the offloading to the new archive log data set continues without the BSDS copy.

z/OS System definition on z/OS

IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

Setting system parameters

In IBM MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that IBM MQ uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP
System parameters, including setting the connection and tracing environment.

CSQ6LOGP
Logging parameters.

CSQ6ARVP
Log archive parameters.

Default parameter modules are supplied with IBM MQ for z/OS. If these do not contain the values that you want to use, you can create your own parameter modules using the sample supplied with IBM MQ. The sample is `thlqual.SCSQPROC(CSQ4ZPRM)`.

You can alter some system parameters while a queue manager is running. See the SET SYSTEM, SET LOG, and SET ARCHIVE commands in The MQSC commands.

For more information about defining , see the following topics:

- “Defining system objects for IBM MQ for z/OS” on page 209
- “Tuning your queue manager on IBM MQ for z/OS” on page 214
- “Sample definitions supplied with IBM MQ for z/OS” on page 215

Related concepts
Customize the sample initialization input data sets
Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS
Related tasks
Administering z/OS
Configuring clusters
Monitoring IBM MQ

z/OS Defining system objects for IBM MQ for z/OS

IBM MQ for z/OS requires additional predefined objects for publish/subscribe applications, cluster, and channel control and other system administration functions.

The system objects required by IBM MQ for z/OS can be divided into the following categories:

- Publish/subscribe objects
- System default objects
- System command objects
- System administration objects
- Channels queues
- Cluster queues

- Queue sharing group queues
- Storage classes
- Defining the system object dead-letter queue
- Default transmission queue
- Internal queues
- “Channel authentication queue” on page 213

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM MQ for z/OS. Sample definitions are supplied with IBM MQ to help you define these objects. These samples are described in CSQ4INSG.

To use publish/subscribe you need to define the following objects:

- A local queue called SYSTEM.RETAINED.PUB.QUEUE, which is used to hold a copy of each retained publication in the queue manager. Each full topic name could have up to one retained publication stored on this queue. If your applications will make use of retained publications on many different topics, or if your retained publication messages are large messages, the requirements for storage for this queue should be carefully planned, including assigning it to its own page set if the storage requirements for it are large. To improve performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.SUBSCRIBER.QUEUE, which is used to hold a persistent copy of the durable subscriptions in the queue manager. To improve performance, you should define this queue with an index type of CORRELID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.MODEL.QUEUE, which is used as a model for managed durable subscriptions.
- A local queue called SYSTEM.NDURABLE.MODEL.QUEUE, which is used as a model for managed non-durable subscriptions.
- A namelist called SYSTEM.QPUBSUB.QUEUE.NAMELIST, which contains a list of queue names monitored by the queued publish/subscribe interface.
- A namelist called SYSTEM.QPUBSUB.SUBPOINT.NAMELIST, which contains a list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.
- A topic called SYSTEM.BASE.TOPIC, which is used as a base topic for resolving attributes.
- A topic called SYSTEM.BROKER.DEFAULT.STREAM, which is the default stream used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.DEFAULT.SUBPOINT, which is the default RFH2 subscription point used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.ADMIN.STREAM, which is the admin stream used by the queued publish/subscribe interface.
- A subscription called SYSTEM.DEFAULT.SUB, which is a default subscription object used to provide default values on DEFINE SUB commands.

System default objects

System default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters "SYSTEM.DEFAULT" or "SYSTEM.DEF." For example, the system default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE.

These objects define the system defaults for the attributes of these IBM MQ objects:

- Local queues

- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes
- Authentication information

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the SYSTEM.DEFAULT.LOCAL.QUEUE. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue sharing group only.

System command objects

The names of the system command objects begin with the characters SYSTEM.COMMAND. You must define these objects before you can use the IBM MQ operations and control panels to issue commands to an IBM MQ subsystem.

There are two system command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the IBM MQ command processor. It must be called SYSTEM.COMMAND.INPUT. An alias named SYSTEM.ADMIN.COMMAND.QUEUE should also be defined, for compatibility with IBM MQ for Multiplatforms, and for use by the IBM MQ Console and administrative REST API.
2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

There are two extra objects for use by the IBM MQ Explorer:

- SYSTEM.MQEXPLORER.REPLY.MODEL queue
- SYSTEM.ADMIN.SVRCONN channel

SYSTEM.REST.REPLY.QUEUE is the reply queue used by the IBM MQ administrative REST API.

Commands are normally sent using nonpersistent messages so both the system command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) get nonpersistent messages by default. If you have an application that uses persistent messages for commands, set the DEFTYPE(PERMDYN) attribute for the reply-to queue, because the reply messages to such commands are persistent.

System administration objects

The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are seven system administration objects:

- The SYSTEM.ADMIN.CHANNEL.EVENT queue
- The SYSTEM.ADMIN.COMMAND.EVENT queue
- The SYSTEM.ADMIN.CONFIG.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.QMGR.EVENT queue

- The SYSTEM.ADMIN.TRACE.ROUTE.QUEUE queue
- The SYSTEM.ADMIN.ACTIVITY.QUEUE queue

Channels queues

To use distributed queuing, you need to define the following objects:

- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue with the name SYSTEM.CHANNEL.INITQ, which is used for channel commands.

You cannot define these queues as shared queues.

Cluster queues

To use IBM MQ clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons, you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

You cannot define these queues as shared queues.

Queue sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:

- A shared queue with the name SYSTEM.QSG.CHANNEL.SYNCQ, which is used to hold synchronization information for shared channels.
- A shared queue with the name SYSTEM.QSG.TRANSMIT.QUEUE, which is used as the transmission queue for intra-group queuing. If you are running in a queue sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by IBM MQ. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)
This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)
This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)
This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV
This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)
This storage class is used for performance critical, system related message queues, for example the SYSTEM.CHANNEL.SYNQ and the SYSTEM.CLUSTER.* queues.

SYSVOLAT
This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Defining the system object dead-letter queue

The dead-letter queue is used if the message destination is not valid. IBM MQ puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, you should regard it as essential, especially if you are using distributed queuing or one of the IBM MQ bridges.

Do **not** define the dead-letter queue as a shared queue. A put to a local queue on one queue manager might get put to the dead letter queue. If the dead letter queue was a shared queue, a dead letter queue handler on a different system could process the message and put it on a queue with the same name, but because this is on a different queue manager, it would be the wrong queue, or have a different security profile. If the queue did not exist, it would fail to reprocess it.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the ALTER QMGR DEADQ(*queue-name*) command. For more information see Displaying and altering queue manager attributes.

Default transmission queue

The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not do this, messages that are put on to the default transmission queue are not transmitted to the remote queue manager and remain on the queue.

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command.

Internal queues

- **Pending data queue**
  - A queue defined for internal use, SYSTEM.PENDING.DATA.QUEUE, supports the use of durable subscriptions in a JMS publish/subscribe environment.
- **JMS 2.0 delivery delay staging queue**
  - If the delivery delay functionality provided by JMS 2.0 is used then an internal staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, must be defined. This queue is used by the queue manager to temporarily store messages sent with a non-zero delivery delay until the delivery delay is completed, and the message is put to its target destination. A sample definition for this queue is provided, commented out, in CSQ4INSG.
  - When you define the SYSTEM.DDELAY.LOCAL.QUEUE queue, you must set the STGCLASS, MAXMSGL and MAXDEPTH attributes for the anticipated number of messages that will be sent with a delivery delay. Additionally when defining the SYSTEM.DDELAY.LOCAL.QUEUE queue ensure that only the queue manager can put messages to this queue. Care should be taken to ensure that no user identifier has the authority to put messages to this queue.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

z/OS Tuning your queue manager on IBM MQ for z/OS

There are few simple steps that you can take to ensure that your queue manager is tuned to avoid basic performance problems.

There are a number of ways in which you can improve the performance of your queue manager, which are controlled by queue manager attributes set by the ALTER QMGR command. This section contains information about how you can do this by setting the maximum number of messages allowed on the queue manager, or by performing 'housekeeping' on the queue manager. IBM MQ SupportPac MP16 - IBM MQ，用于 z/OS 容量规划和调整 gives more information on performance and tuning.

Syncpoints

One of the roles of the queue manager is syncpoint control within an application. An application constructs a unit of work containing any number of MQPUT or MQGET calls terminated with an MQCMIT call.

As the number of MQPUT or MQGET calls within the scope of one MQCMIT increases, the performance cost of the commit increases significantly. Applications, in general, should be designed to not MQPUT/MQGET a large number of messages in a single synchpoint.

You can administratively limit the number of messages within any single syncpoint by using the MAXUMSGS queue manager attribute. If an application exceeds this limit it receives MQRC_SYNCPOINT_LIMIT_REACHED on the MQPUT,MQPUT1, or MQGET call which exceeds the limit. The application should then issue MQCMIT or MQBACK as appropriate.

The default value of MAXUMSGS is 10000. This value can be lowered if you want to enforce a lower limit, which can also help protect against looping applications. Before reducing MAXUMSGS make sure you understand your existing applications to ensure they do not exceed the limit, or can tolerate the MQRC_SYNCPOINT_LIMIT_REACHED return code

Expired messages

Messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired messages are not discarded, and, for some queues, particularly those where message retrieval is done by MessageId, CorrelId, or GroupId and the queue is indexed for performance, many expired messages can accumulate. The queue manager can periodically scan any queue for expired messages, which are then deleted. You can choose how often this scanning takes place, if at all. There are two ways of doing this:

Explicit request
: You can control which queues are scanned and when. Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want to be scanned.

Periodic scan
: You can specify an expiry interval in the queue manager object by using the EXPRYINT attribute. The queue manager maintains information about the expired messages on each queue, and knows at what time a scan for expired messages is worthwhile. Each time that the EXPRYINT interval is reached, the queue manager looks for candidate queues that are worth scanning for expired messages, and scans only those queues that it deems to be worthwhile. It does not scan all queues. This avoids any processor time being wasted on unnecessary scans.

  Shared queues are only scanned by one queue manager in the queue sharing group. Generally, the first queue manager to restart or the first to have the EXPRYINT set performs the scan.

Note: You must set the same EXPRYINT value for all queue managers within a queue sharing group.

z/OS Sample definitions supplied with IBM MQ for z/OS

Use this topic as a reference for the sample JCL, and code supplied with IBM MQ for z/OS.

The following sample definitions are supplied with IBM MQ in the thlqual.SCSQPROC library. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in Initialization commands).

Table 22. IBM MQ sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS [1] CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INSM CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD CSQ4INSC
CSQINPT	CSQ4INST CSQ4INYT
Other	CSQ4DISP CSQ4INPX CSQ4IVPQ CSQ4IVPG CSQ4MSTR CSQ4MSRR CSQ4QMIN

Note:

1. The order of these sample definitions is important: an error occurs if INYS, INSX, and INSG are ordered incorrectly.

CSQINP1 samples

Use the sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) when you are using one page set for each class of message, or thlqual.SCSQPROC(CSQ4INPR) when using multiple page sets for the major classes of message. It contains definitions of buffer pools, page set to buffer pool associations, and an ALTER SECURITY command. Include the sample in the CSQINP1 concatenation of your queue manager started task procedure.

CSQINP2 samples

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:

- System default objects
- System command objects
- System administration objects
- Other objects for system use

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across queue manager shutdown and restart. You must not change the object names, but you can change their attributes if required.

When the following conditions are met, one message is put to the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue (even if publish subscribe is not active):

- The QMGR attribute PSMODE is set to DISABLED
- The sample object CSQ4INST statement `DEFINE SUB('SYSTEM.DEFAULT.SUB')` is present.

To avoid this, delete or comment out the `DEFINE SUB('SYSTEM.DEFAULT.SUB')` statement.

The JMS 2.0 delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE only need be defined if JMS 2.0 delivery delay is used. By default, the queue definition is commented out, which you can uncomment if required.

CSQ4INSA system object and authentication sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSA) contains the channel authentication system queue definition. This queue holds the channel authentication records. It also contains the default channel authentication rules.

You must define the objects in this sample if CHLAUTH is ENABLED on the queue manager and you want to run channels, or you want to SET or DISPLAY CHLAUTH record. You only need to define them once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across a queue manager shutdown and restart, you must not change the queue name.

CSQ4INSS system object sample

You can define additional system objects if you are using queue sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains sample commands for use with CF structures and a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required commands.

When you are defining group or shared objects, you need to include them in the CSQINP2 DD concatenation for only one queue manager in the queue sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or

you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INSJ system JMS object sample

Defines queues used in the JMS publish/subscribe domain.

CSQ4INSM system object sample

If you are using advanced message security you must define additional system objects. Sample data set thlqual.SCSQPROC(CSQ4INSM) contains the queue definitions required.

CSQ4INSR object sample

Defines queues used by WebSphere Application Server and brokers.

CSQ4INYD object sample

If you are using distributed queuing and you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don't have to redefine these objects each time you restart the queue manager).

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed - a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

The sample data set: thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

CSQ4INYG object sample

The sample data set: thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you can make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue
Read the Default transmission queue description before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects
The sample defines an initiation queue named CICS01.INITQ. This queue is used by the IBM MQ -supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4INYS/CSQ4INYR object samples

Storage class definitions for using:

- one page set for each class of message
- multiple page sets for major classes of message

For example, SYSTEM.COMMAND.INPUT uses STGCLASS('SYSVOLAT'), and SYSTEM.CLUSTER.TRANSMIT.QUEUE uses STGCLASS('REMOTE'). In CSQ4INYS, both of those storage classes use the same page set. In CSQ4INYR, those storage classes use different page sets in order to lessen the impact of the transmission queue filling.

CSQINPT samples

CSQ4INST
The sample data set: thlqual.SCSQPROC(CSQ4INST) contains the definition for the system default subscription.

You must define the object in this sample, but you need to do it only once when the publish/subscribe engine is first started. Including the definition in the CSQINPT data set is the best way to do this. It is maintained across queue manager shutdown and restart. You must not change the object name, but you can change their attributes if required.

CSQ4INYT
The sample data set: thlqual.SCSQPROC(CSQ4INYT) contains a set of commands that you might want to run when the publish/subscribe engine is started. This sample displays Topic and Subscription information.

Other

CSQ4DISP display sample

The sample data set: thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your queue manager. This includes the definitions for all IBM MQ objects and definitions such as storage classes and trace. These commands can generate a large amount of output. You can use this sample in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4INPX sample

The sample data set: thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

The sample data sets: thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time the queue manager is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

CSQ4MSTR and CSQ4MSRR samples

These are sample started task procedures for the queue manager: thlqual.SCSQPROC(CSQ4MSTR) and thlqual.SCSQPROC(CSQ4MSRR).

CSQ4MSRR uses CSQ4INYR in the CSQINP2 concatenation so that important queues are spread across different page sets.

You can remove the comments, so that you can use the CSQMINI card for newly created queue managers if required.

CSQ4QMIN sample

A sample QMINI data set, thlqual.SCSQPROC(CSQ4QMIN).

See QMINI data set for details of the QMINI data set and the **TransportSecurity** stanza.

z/OS Recovery and restart on z/OS

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

IBM MQ for z/OS has robust features for restart and recovery. For information about how a queue manager recovers after it has stopped, and what happens when it is restarted, see the following subtopics:

- “How changes are made to data in IBM MQ for z/OS” on page 220
- “How consistency is maintained in IBM MQ for z/OS” on page 221
- “What happens during termination in IBM MQ for z/OS” on page 223
- “What happens during restart and recovery in IBM MQ for z/OS” on page 224
- “How in-doubt units of recovery are resolved” on page 226
- “Shared queue recovery” on page 229

Related concepts

z/OS IBM MQ for z/OS recovery actions

Related tasks

Planning for backup and recovery

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS How changes are made to data in IBM MQ for z/OS

IBM MQ for z/OS must interact with other subsystems to keep all the data consistent. This topic contains information about *units of recovery*, what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single queue manager for an application program, that changes IBM MQ data from one point of consistency to another. A *point of consistency* - also called a *syncpoint* or *commit point* - is a point in time when all the recoverable data that an application program accesses is consistent.

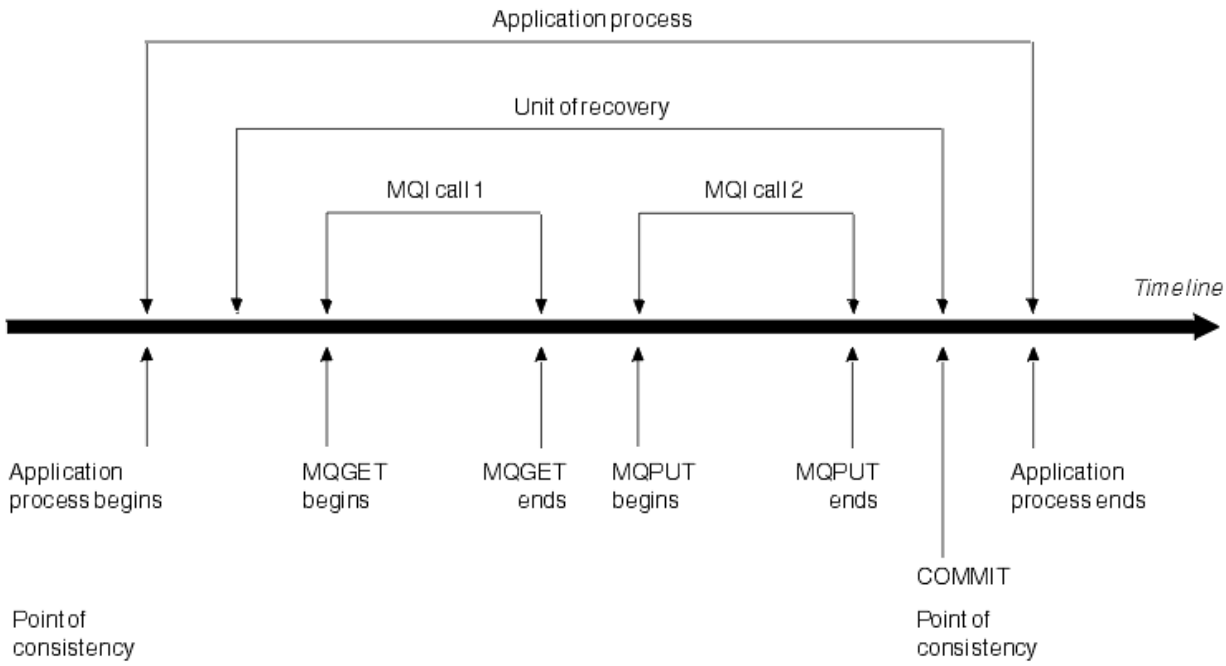


Figure 75. A unit of recovery within an application program

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 75 on page 220 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and IBM MQ cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, IBM MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, IBM MQ backs out the work. The events are shown in Figure 76 on page 221.

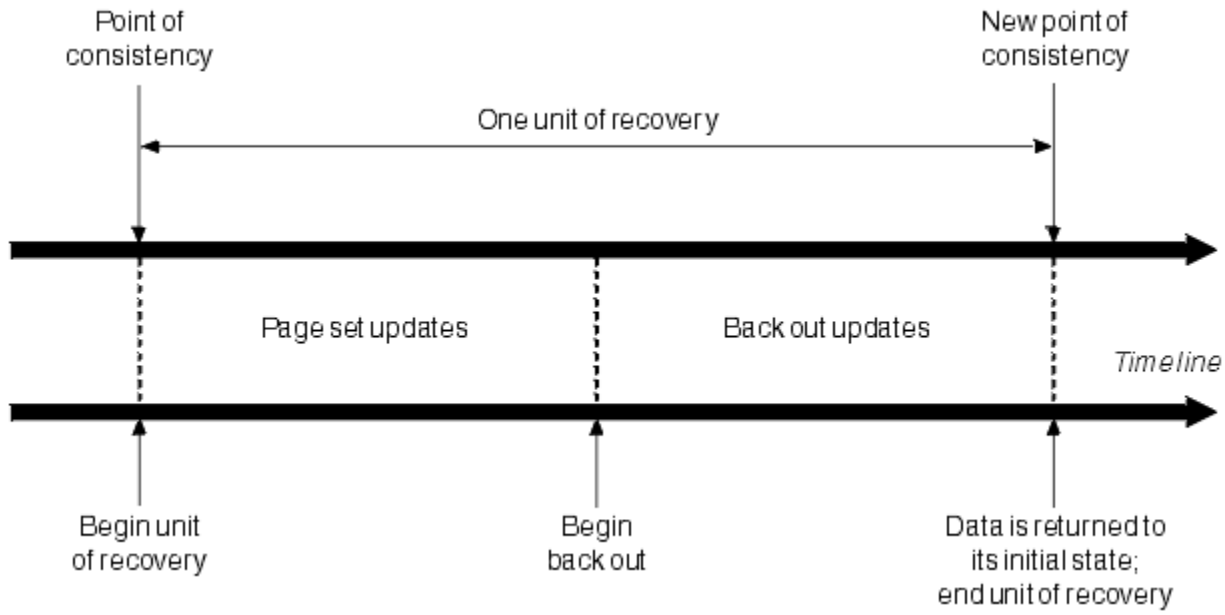


Figure 76. A unit of recovery showing back out

z/OS How consistency is maintained in IBM MQ for z/OS

Data in IBM MQ for z/OS must be consistent with batch, CICS, IMS, or TSO. Any data changed in one must be matched by a change in the other.

Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with IBM MQ, and IBM MQ is always the participant. In the batch or TSO environment, IBM MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), IBM MQ is always the coordinator in the interactions and completely controls the commit process.

In a WebSphere Application Server environment, the semantics of the JMS session object determine whether single-phase or two-phase commit coordination is used.

Consistency with CICS or IMS

The connection between IBM MQ and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit - for transactions that update resources owned by more than one resource manager.

This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.

- Single-phase commit - for transactions that update resources owned by a single resource manager (IBM MQ).

  This protocol is optimized for logging and message flows.
- Bypass of syncpoint - for transactions that involve IBM MQ but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that IBM MQ uses to communicate with CICS or IMS are as follows:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

   At the end of the phase, the systems communicate. If they agree, each begins the next phase.
2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 77 on page 222 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in IBM MQ on the lower line.

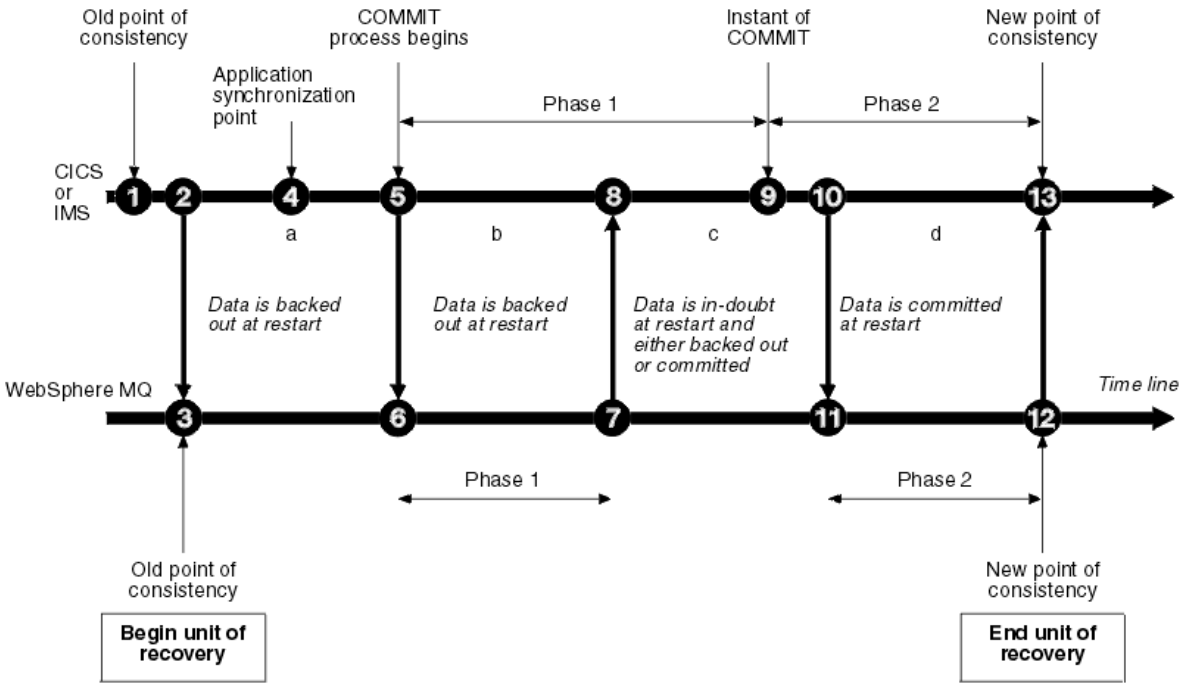


Figure 77. The two-phase commit process

The numbers in the following section are linked to those shown in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls IBM MQ to update a queue by adding a message.
3. This starts a unit of recovery in IBM MQ.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using a

CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.

6. As the coordinator begins phase 1 processing, so does IBM MQ.
7. IBM MQ successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit - the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing - the actual commitment.
10. The coordinator notifies IBM MQ to begin its phase 2.
11. IBM MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for IBM MQ. IBM MQ then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When a queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For some units of recovery, IBM MQ has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 77 on page 222 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be one of the following:

In flight
The queue manager ended before finishing phase 1 (period a or b); during restart, IBM MQ backs out the updates.

In doubt
The queue manager ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, IBM MQ must back out its changes; if it happened after, IBM MQ must make its changes and commit them. At restart, IBM MQ waits for information from the coordinator before processing this unit of recovery.

In commit
The queue manager ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout
The queue manager ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure) during restart, IBM MQ continues to back out the changes.

z/OS What happens during termination in IBM MQ for z/OS

A queue manager terminates normally in response to the STOP QMGR command. If a queue manager stops for any other reason, the termination is abnormal.

Note, that during queue manager termination, IBM MQ internally issues the command

```
DISPLAY CONN(*) TYPE(CONN) ALL WHERE (APPLTYPE NE SYSTEMAL)
```

so that you are aware of what threads might prevent the queue manager from completing shutdown.

SYSTEMAL matches APPLTYPES of either SYSTEM or CHINIT, so the DISPLAY CONN command filtering application types not matching SYSTEMAL, returns to the joblog information about threads that could be preventing normal shutdown.

Normal termination

In a normal termination, IBM MQ stops all activity in an orderly way. You can stop IBM MQ using either quiesce, force, or restart mode. The effects are given in Table 23 on page 224.

Table 23. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping a queue manager in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop a queue manager, in any mode, the steps are:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart RBA is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

IBM MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

z/OS What happens during restart and recovery in IBM MQ for z/OS

IBM MQ uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent IBM MQ checkpoint in the log.

Introduction to restart and recovery

After IBM MQ has been initialized, the queue manager restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until IBM MQ recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in “Rebuilding queue indexes” on page 226).

If dual BSDSs are in use, IBM MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, IBM MQ tests whether the two time stamps are equal. If they are not, IBM MQ issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while the queue manager was stopped. IBM MQ detects the situation at restart.
- If one copy of the BSDS was de-allocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, IBM MQ might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

Understanding the log range required for recovery

During restart, the range of log data which must be read is dependent on many factors:

- At the time of an abnormal termination, there are typically many incomplete units of work in the system. As described earlier, restart processing will bring the system to a state of consistency, which may involve backing out inflight units of work, or recovering locks on indoubt units of work. Unit of work recovery requires that all unit of work log records for inflight, in-backout, and in-doubt units of work are available. IBM MQ will 'shunt' old units of work, so that unit of work recovery can be performed using a much smaller range of log data.
- At the time of an abnormal termination, there are typically many persistent updates which are only held in the buffer pool cache. They have not yet been written to disk. These changes must be read from the log, and reapplied to the data held in page sets. Page set recovery RBAs in the checkpoint describe the lowest log RBA required for updating the page sets to a consistent state.
- If old page sets have been introduced into the system, for example, a page set backup has been introduced to recover from a media failure, all the changes must be read from the log from the time the backup was taken. These changes are reapplied to the data held in the page set being recovered. Page set recovery RBAs held in page 0 of the page set describe the lowest log RBA required for media recovery of a page set.
- If using persistent messages on shared queues, a range of log data is required to recover CFSTRUCTs which are holding persistent messages. The earliest log data that would be required to perform a CFSTRUCT recovery, is from around the time of the old CFSTRUCT BACKUP.

During normal running, the DISPLAY USAGE TYPE(DATASET) command can be used to view the recovery log range associated with these factors (it is unable to provide information due to reintroducing old page sets, of course). To avoid any issues that might prolong a queue manager restart in the event of an abnormal termination, regularly monitor the values output from DISPLAY USAGE TYPE(DATASET).

In addition, the queue manager issues informational messages relating to these factors:

- CSQJ160I and CSQJ161I warn of long running units of work.
- CSQR026I and CSQR027I provide information about whether these long running units of work have been successfully shunted.
- CSQE040I and CSQE041E warn that structure backups are getting old, and consequently a RECOVER CFSTRUCT operation would take a long time.

Determining which application has a long running unit of work

It is possible to determine the application with the long-running unit of work. To do this, you use the DISPLAY CONN command.

The DISPLAY CONN command returns connection information for all the applications connected to the queue manager, together with additional information that helps you determine which application(s) currently have a long-running unit of work. The information returned by the DISPLAY CONN command is similar to the information returned by the DISPLAY QSTATUS command, but the main difference is that DISPLAY CONN displays information about objects, and transactional information for a particular connection, rather than details of which connections are associated with a particular object.

For each connected application, the DISPLAY CONN command returns the following information:

- Basic information including the Connection Id and PID.
- Transactional information for that connection, including the time and date when the transaction was created (that is, when the first MQGET/PUT was made under syncpoint), and when the transaction first wrote to the log.
- Log time information indicating which application still has a long running unit of work.
- A list of all objects that the connection currently has open. Details for each object are returned as a separate message, with the Connection Id used as a key. Because there are different types of object such as queues and queue managers, the information displayed with the object is specific to its object type.

Rebuilding queue indexes

To increase the speed of MQGET operations on a queue where messages are not retrieved sequentially, you can specify that you want IBM MQ to maintain an index of the message or correlation identifiers or groupid for all the messages on that queue.

When a queue manager is restarted, these indexes are rebuilt for each queue. This applies only to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this increases the time taken to restart the queue manager.

You can choose to have indexes rebuilt asynchronously to queue manager startup by using the QINDXBLD parameter of the CSQ6SYSP macro. If you set QINDXBLD=NOWAIT, IBM MQ restarts without waiting for indexes to rebuild.

z/OS How in-doubt units of recovery are resolved

If IBM MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If IBM MQ loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information required to resolve in-doubt units of recovery must come from the coordinating system. The next sections describe the process of resolution for different environments.

- How in-doubt units of recovery are resolved from CICS
- How in-doubt units of recovery are resolved from IMS
- How in-doubt units of recovery are resolved from RRS
- How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

How in-doubt units of recovery are resolved from CICS

Under some circumstances, CICS cannot run the IBM MQ process to resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the IBM MQ for z/OS 消息，完成和原因码 manual.

The resolution of in-doubt units does not effect CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while IBM MQ is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and IBM MQ. If a queue manager abends while connected to CICS, it is possible for CICS to commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*.

IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from IBM MQ, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each in-doubt unit of recovery.

For all resolved units, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in the 管理 IBM MQ for z/OS.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMSdoes not effect DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801'. The existence of in-doubt units of recovery does not imply that DL/I records are locked until IBM MQ connects.

During queue manager restart, IBM MQ makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

During reconnection of an IMS region to IBM MQ, IMS indicates to IBM MQ whether to commit or back out units of work marked in IBM MQ as in doubt.

When in-doubt units are resolved:

1. If IBM MQ recognizes that it has marked an entry for commit and IMS has marked it to be backed out, IBM MQ issues message CSQQ010E. IBM MQ issues this message for all inconsistencies of this type between IBM MQ and IMS.
2. If IBM MQ has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, IBM MQ updates queues as necessary and releases the corresponding locks.

IBM MQ maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in the 管理 IBM MQ for z/OS.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to IBM MQ, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between IBM MQ and other RRS-participating resource managers. If a failure occurs when IBM MQ has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and IBM MQ, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages to the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the IBM MQ for z/OS 消息，完成和原因码 manual.

For all resolved units of recovery, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in the 管理 IBM MQ for z/OS.

How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

In-doubt transactions that have a GROUP unit of recovery disposition can be resolved by the transaction coordinator by any queue manager in the queue sharing group (QSG) where the GROUPUR queue manager attribute is enabled. Whenever a transaction coordinator reconnects it typically requests a list of any outstanding in-doubt transactions and then resolves them using information from its logs.

When a transaction coordinator, that has connected with a GROUP unit of recovery disposition, requests the list of in-doubt transactions, the list returned comprises all in-doubt transactions with a GROUP unit of recovery disposition that exist throughout the queue sharing group. This list is not dependent on which queue manager those in-doubt transactions were started on. A queue manager processing such a request compiles the list by communicating with all other active queue managers in the queue sharing group using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The queue manager then reads the logs of any inactive queue managers, from their last checkpoint, to identify any additional in-doubt transactions that they would have reported had they been active.

When a transaction coordinator requests the resolution of an in-doubt transaction, the queue manager to which it is connected identifies whether the transaction was originated on itself and if so resolves it in the same way as transactions with a QMGR unit of recovery disposition. If the transaction was originated on another active queue manager in the QSG, a request to complete the resolution is routed to that queue manager using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. In the case where the transaction was originated on an inactive queue manager in the QSG, any shared-queue work is resolved immediately, and a request to resolve any remaining private queue work is placed on the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The inactive queue manager processes this request upon start-up before accepting new work. In this scenario, the original queue manager's logs still reflect that the unit of recovery is in doubt until it has restarted and processed the request.

z/OS Shared queue recovery

Use this topic to understand IBM MQ recovery and resilience of various components in the queue sharing group environment.

Transactional recovery

When an application issues a MQBACK call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread-level information stored in the queue manager ensures that the in-flight unit of work is rolled back. MQPUT and MQGET operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the coupling facility structures that it is currently connected to. If the connection between the z/OS instance and the coupling facility fails (for example, physical link failure or power-off of a coupling facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the coupling facility structures involved. Other queue managers in the same queue sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates peer recovery successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery, often referred to as Peer Level Recovery (PLR), is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that IBM MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared Db2 repository used by the queue sharing group. Ensure that adequate procedures are in place for the backup and recovery of the Db2 tables used to hold IBM MQ objects. You can also use the IBM MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine IBM MQ objects, including shared queue and group definitions stored in Db2.

Logging

Queue sharing groups can support persistent messages, because the messages on shared queues can be logged in the queue manager logs.

Coupling facility and structure failures

There are two types of failure that can be reported for a coupling facility (CF) structure: structure failure and loss of connectivity. Sysplex services for data sharing (XES) inform IBM MQ of a CF structure failure or a CF failure with a structure failure event. If XES creates a loss of connectivity event this does not necessarily indicate that there is a problem with the structure, it might be that there is no connection available to communicate with the structure. It is possible that not all of the queue managers receive a loss of connectivity event for the structure; it depends on the configuration of connections to the CF. A loss of connectivity event can also be received because of operator commands, for example VARY PATH OFFLINE or CONFIG CHP OFFLINE.

The CF structures that are used by IBM MQ can be configured to use system-managed duplexing. This means that if there is a single failure, system-managed failover processing hides the failure of a structure or the loss of connectivity, and the queue manager is not informed of the failure. If there is a failure of both instances of a duplexed structure or connection, the queue manager receives the appropriate event and handles it in the same way as a failure event for a simplex structure. Details of how the queue manager handles the events are described in Scenarios.

In the unlikely event of a CF or structure failure, any nonpersistent messages stored in the affected application structures are lost. You can recover persistent messages using the RECOVER CFSTRUCT command. If a recoverable application structure has failed, any further application activity to this structure is prevented until the structure has been recovered.

To ensure that you can recover a CF structure in a reasonable period of time, take frequent backups, using the BACKUP CFSTRUCT command. You can choose to perform the backups on any queue managers in the queue sharing group or dedicate one queue manager to perform all the backups. Automate the process of taking backups to ensure that they are taken on a regular basis.

Each backup is written to the active log data set of the queue manager taking the backup. The shared queue Db2 repository records the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager's log, and the backup time.

The administration structure contains information about incomplete units of work on shared queues at the time of any application structure failure so the administration structure must be available during RECOVER CFSTRUCT processing. If the administration structure has failed, all the queue managers in the queue sharing group must have rebuilt their administration structure entries before you can issue the RECOVER CFSTRUCT command.

Queue managers rebuild their administration structure entries automatically and without terminating. If a queue manager is not running at the time of the failure, its administration structure entries can be rebuilt by another queue manager in the queue sharing group that is running at the same or higher level.

To recover an application structure, issue a RECOVER CFSTRUCT command to the queue manager that you want to perform the recovery. You can recover a single CF structure or you can recover several CF structures simultaneously. You can recover using any queue manager in the queue sharing group, it does not have to be the one that performed the backup, or one that has been connected previously to the failed structure.

The RECOVER CFSTRUCT command uses the backup, located through the Db2 repository information (Db2 must therefore be available on the queue manager where recovery is being carried out), and recovers this to the point of failure.

The RECOVER CFSTRUCT command does this by applying log records from every queue manager in the queue sharing group that has performed an MQPUT or MQGET between the start of the backup and the time of failure, to any shared queue that maps to the CF structure. The resulting merging of the logs might require reading a considerable amount of log data because all the log data written by participating queue managers since the backup is read. You are strongly recommended to make frequent (for example, hourly) backups, especially if there are large messages within the backup.

Structure failure scenarios

Scenarios

If a failure is reported for a CF structure, the action taken by connected queue managers depends on the following:

- The type of failure reported by the XES component of z/OS to IBM MQ.
- The structure type (application or administration)
- The queue manager level
- The CFLEVEL of the IBM MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)
- The RECAUTO attribute of an IBM MQ CFSTRUCT object at CFLEVEL(5)

The following scenarios describe what happens when a failure is reported for the administration structure:

- If a structure failure event is received for the administration structure, the structure is reallocated and rebuilt automatically without the queue manager terminating. A new instance of the structure is allocated by XES when a queue manager attempts to connect to it.

  When the queue manager has connected to the new instance of the structure, the queue manager writes the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

  If a queue manager was not running at the time of the failure, or terminates before recovery of its part of the administration structure has been completed, its administration structure entries can be rebuilt by another queue manager in the queue sharing group.

  Administration structure entries of a queue manager can only be rebuilt by another queue manager running at the same level or higher. If administration structure entries of a queue manager cannot be rebuilt by another queue manager in the queue sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

Certain actions are suspended until the administration structure entries for all queue managers have been rebuilt. The suspended actions include the following:

- Opening and closing of shared queues.
- Committing or backing out units of recovery.
- Serialized applications connecting to or disconnecting from the queue manager.
- Backing up or recovering an application structure.

Any serialized applications that have already connected to the queue manager can continue processing. Any serialized application attempting to connect with the MQCNO_SERIALIZE_CONN_TAG_QSG or MQCNO_RESTRICT_CONN_TAG_QSG parameters receive the MQRC_CONN_TAG_NOT_USABLE return code.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

The following scenarios describe what happens when a failure is reported for an application structure:

- If a structure failure event is received for an application structure, and the CFLEVEL is 1 or 2, the queue manager terminates. Restart the queue manager. The first queue manager to attempt to connect to the structure again causes XES to allocate a new instance of the structure.
- If a structure failure event is received for an application structure, and the CFLEVEL is 3, 4, or 5 the queue managers connected to the structure continue to run. Applications that do not use the queues in the failed structure can continue normal processing.

  However, applications that attempt operations on queues in the failed structure receive an MQRC_CF_STRUC_FAILED error until the structure has been successfully rebuilt, at which point the application can open the queues again.

  Structure rebuild is started automatically for CFLEVEL(5) application structures defined with RECAUTO(YES). Otherwise, the structure will be rebuilt when the RECOVER CFSTRUCT command is issued.

Resilience to coupling facility connectivity failures

What is resilience to coupling facility connectivity failures?

Resilience to coupling facility connectivity failures refers to the ability of queue managers in a queue sharing group to tolerate loss of connectivity to a coupling facility structure without terminating. This function also attempts to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

What is partial loss of connectivity?

IBM MQ defines partial loss of connectivity as a situation where one or more systems in the sysplex lose connectivity to the coupling facility where the structure being accessed by the system is allocated, but at least one system in the sysplex maintains connectivity to the same coupling facility.

What is total loss of connectivity?

IBM MQ defines a total loss of connectivity as a situation where no systems in the sysplex have connectivity to the coupling facility and the structure allocated within it.

Why would you enable this function?

Resilience to coupling facility connectivity failures improves the availability of IBM MQ, allowing non-shared queues to remain available after a queue manager has lost connectivity to one or more coupling facility structures. Additionally, queue managers that lose connectivity to a coupling facility structure automatically attempt to rebuild the structure in another available coupling facility, improving the availability of the shared queues within the queue sharing group.

Considerations when enabling this function

A queue manager that tolerates loss of connectivity to coupling facility structures without terminating may not be able to reconnect to a coupling facility structure for some time if there is no alternative coupling facility available. Shared queues defined on a structure that has suffered loss of connectivity remain unavailable until connectivity to the structure is restored. In this situation, applications that connect into members of the queue sharing group in order to perform shared queue work may find that the shared queues they need to access are not available. To avoid this situation it is recommended that queue managers should be configured to terminate when connectivity to a coupling facility structure is lost. This termination forces applications to connect to another member of the queue sharing group that has connectivity to the coupling facility structures where the shared queues the application requires are defined.

Managing Resilience to coupling facility connectivity failures

How do I enable this functionality?

The following steps must be performed in order to enable resilience to coupling facility connectivity

1. Ensure that the CFRM couple data set has been formatted to support system-managed rebuild. This allows queue managers to initiate a system-managed rebuild to re-create a structure into an available coupling facility. Use the **DISPLAY XCF,COUPLE,TYPE=CFRM** command to determine the format of the CFRM couple data set. To support system-managed rebuild, the CFRM couple data set should be formatted by specifying:

   ```
   "ITEM NAME(SMREBLD) NUMBER(1)"
   ```

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information on formatting a CFRM couple data set.
2. Ensure that an alternative coupling facility is available and is in the CFRM preference list for all IBM MQ coupling facility structures. This enables the queue managers to attempt to rebuild structures into an alternative available coupling facility to restore access to the structures as soon as possible.

   IBM MQ structures must be defined with ENFORCEORDER(NO) in CFRM policy, so that XCF is able to choose the optimum CF in the configuration if IBM MQ needs to reallocate the structure.

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information about structure preference lists.
3. Alter all application coupling facility structures that need to tolerate loss of connectivity to CFLEVEL(5). This is the minimum level that can tolerate a loss of connectivity.
4. Determine the values required for the **QMGR CFCONLOS** and the **CFSTRUCT CFCONLOS** attributes and alter these accordingly. The **QMGR CFCONLOS** attribute controls whether loss of connectivity to the administration structure is tolerated, and the **CFSTRUCT CFCONLOS** attribute controls whether loss of connectivity is tolerated by each application coupling facility structure. If the default values for these attributes are retained, the queue manager terminates following loss of connectivity to any coupling facility structure.
5. Determine the values required for the **CFSTRUCT RECAUTO** attribute for each application coupling facility structure, and alter these accordingly. This attribute controls whether coupling facility structures should be automatically recovered using logged data following total loss of connectivity. If the default value for this attribute is retained, no automatic recovery is performed for application structures following total loss of connectivity.

Scenario 1 - Loss of connectivity to the administration structure

Queue managers can tolerate loss of connectivity to the administration structure without terminating.

When connectivity to the administration structure is lost by any queue manager that has been configured to tolerate loss of connectivity to the administration structure, all members of the queue sharing group disconnect from the administration structure. All active queue managers in the queue

sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in the coupling facility with the best connectivity to all systems in the sysplex, and rebuild the administration structure data.

Note: This may not necessarily be the coupling facility which has the best connectivity to all systems that have active queue managers.

If a queue manager cannot reconnect to the administration structure, for example because none of the coupling facilities in the CFRM preference list for the administration structure are available, some shared queue operations remain unavailable until the queue manager can successfully reconnect to the administration structure and rebuild its administration structure data. Reconnection occurs automatically when a suitable coupling facility becomes available on the system.

Failure to connect to the administration structure during queue manager startup as a result of a lack of connectivity to the coupling facility, or no suitable coupling facility available to allocate the structure, is not tolerated. All active queue managers in the queue sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in another coupling facility if one is available, and rebuild the administration structure data.

Scenario 2- Loss of connectivity to the application structure

Loss of connectivity to application structures at **CFLEVEL(5)** or higher can be tolerated without the queue manager terminating. Queue managers connected to application structures at **CFLEVEL(4)** or lower, or structures at **CFLEVEL(5)** that have not been configured to tolerate loss of connectivity, abend with reason code 00C510AB when connectivity to the structure is lost.

When connectivity is lost to an application structure that has been configured to tolerate loss of connectivity, all queue managers that lost connectivity to the structure disconnect. The subsequent behavior of the queue manager depends on whether the loss of connectivity is partial or total.

Partial loss of connectivity to an application structure

If the loss of connectivity is determined to be partial, queue managers that have lost connectivity to the structure attempt to initiate a system-managed rebuild in order to move the structure to another coupling facility with improved connectivity. If this rebuild is successful, both persistent and non-persistent messages in the structure are copied to the other coupling facility, and access to queues on the structure is restored. Queue managers that did not lose connectivity remain connected to the structure, however, operations that access the structure are delayed during the system-managed rebuild process.

If an application structure cannot be rebuilt to another coupling facility with improved connectivity, or some queue managers still do not have connectivity to the structure after it has been rebuilt in another coupling facility, queues defined on the structure remain unavailable on the queue managers that do not have connectivity to the structure until connectivity is restored to the coupling facility. Queue managers automatically reconnect to the structure when it becomes available and access to the shared queues defined on the structure are restored.

Total loss of connectivity to an application structure

If all MVS systems in the sysplex have lost connectivity to the coupling facility that the application structure is allocated in, z/OS deallocates the structure from the coupling facility whenever an attempt is made to reconnect to the structure. It is possible for the queue manager to attempt to reconnect to the structure for several reasons, such as an attempt by an application to open a shared queue, or a notification from the system that new coupling facility resources may have become available. It is therefore likely that all non-persistent messages in the affected structure are lost following total loss of connectivity to an application structure.

Recoverable application structures are automatically recovered following total loss of connectivity, if they have been defined with **RECAUTO(YES)**. The recovery starts almost immediately if an alternative coupling facility is available to allocate the structure in, or whenever such a coupling facility becomes available. If a structure has not been defined with **RECAUTO(YES)**, recovery can be started by issuing the **RECOVER CFSTRUCT** command. This recovers all persistent messages in the structure, but all non-persistent messages are lost. As this process involves reading the queue manager log it can take

some time to complete, therefore it is recommended that structure backups be taken regularly to reduce the time until access to the shared queues on the structure is restored.

Queue managers attempt to reconnect to non-recoverable application structures as soon as an application attempts to open a shared queue that is defined on the structure or a notification is received from the system that new coupling facility resources have become available. If a suitable coupling facility is available to allocate the structure in, a new structure is allocated and access to the shared queues defined on the structure is restored. As persistent messages cannot be put to queues defined in non-recoverable structures, all messages on the shared queues are lost.

Operational behavior

If an IBM WebSphere MQ 7.1, or later, queue manager, configured to tolerate loss of connectivity to a particular coupling facility structure loses connectivity, the members of the queue sharing group attempt to automatically recover from the failure and reconnect to the structure. This activity may involve reallocating the structure in another coupling facility with better connectivity if one is available. However, operator intervention may still be required to recover from the loss of connectivity.

Typically the required operator action is to:

1. Resolve the cause of the failure that resulting in the loss of connectivity.
2. Ensure that a coupling facility where the IBM MQ structures can be allocated is available on all systems in the sysplex

Any structures that have been automatically reallocated in another coupling facility after the loss of connectivity event, can be moved to the coupling facility with the optimal connectivity to all queue managers in the queue sharing group. If required, this can be done by initiating the system-managed rebuild command **SETXCF START,REBUILD** as documented in z/OS MVS 系统命令参考.

In the case of a partial loss of connectivity to an application structure, the queue managers that lost connectivity to the structure attempt to initiate a system-managed rebuild. This process only allocates the structure in another coupling facility if that coupling facility has connectivity to all active queue managers currently connected to the structure. Therefore, it is possible that where the majority of queue managers in a queue sharing group have lost connectivity to an application structure, they are unable to rebuild the structure into another coupling facility due to the queue managers that are still connected to the original structure. In this situation the queue managers that are still connected to the original structure can be shut down to allow the structure to be rebuilt, or the **RESET CFSTRUCT ACTION(FAIL)** command can be issued to fail the structure. Recovery can be initiated on applicable structures by issuing the **RECOVER CFSTRUCT** command.

Note: When failing and recovering the structure, all non-persistent messages on the structure are lost.

z/OS Security concepts in IBM MQ for z/OS

Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

Why you must protect IBM MQ resources

IBM MQ handles the transfer of information that is potentially valuable. Applying security ensures that the resources IBM MQ owns and manages are protected from unauthorized access. Such access might lead to the loss or disclosure of the information.

You should ensure that none of the following resources are accessed or changed by any unauthorized user or process:

- Connections to IBM MQ
- IBM MQ objects such as queues, processes, and namelists
- IBM MQ transmission links, that is, IBM MQ channels
- IBM MQ system control commands

- IBM MQ messages
- Context information associated with messages

To provide the necessary security, IBM MQ uses the z/OS system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Security Server (previously known as RACF). IBM MQ does no security verification of its own. Where distributed queuing or clients are being used, you might require additional security measures, for which IBM MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and TLS.

The decision to allow access to an object is made by the ESM and IBM MQ follows that decision. If the ESM cannot make a decision, IBM MQ prevents access to the object.

What happens if you do not protect IBM MQ resources

If you do nothing about security, the most likely effect is that *all* users can access and change *every* resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for z/OS.

To enable security checking you must do the following:

- Install and activate an ESM (for example, Security Server).
- Define the MQADMIN class if you are using an ESM other than Security Server.
- Activate the MQADMIN class.

You must consider whether using mixed-case resource names would be beneficial to your enterprise. If you do use mixed-case resource names in your ESM profiles you must define and activate the MXADMIN class.

z/OS Data Set Encryption

Data Set Encryption (DSE) provides the capability to encrypt z/OS data sets, so that the data they contain can only be viewed or modified by user IDs granted the specific permission. This provides encryption of data at rest in the file system, and prevents inadvertent disclosure of sensitive information to users who have a legitimate business need and permissions to manage the data sets themselves.

Prior to IBM MQ for z/OS 9.1.4, IBM MQ for z/OS does not support use of DSE with the active logs, page sets, and shared message data sets (SMDS) that provide the primary persistence mechanisms for IBM MQ messages.

Instead, Advanced Message Security provides an end-to-end encryption solution for IBM MQ messaging, which encompasses the entire IBM MQ network, encryption of data in flight, at rest, and even inside the runtime IBM MQ processes.

Other VSAM and sequential data sets used in an IBM MQ subsystem can be encrypted using DSE. For example:

- Bootstrap data set (BSDS)
- Sequential files holding system configuration (MQSC) commands read at startup using CSQINPx DDNAMEs
- IBM MQ archive logs, often used for long term archival of IBM MQ log data for audit purposes.

You can encrypt using DSE by allocating a dataclass that is defined with a data set key label. For more information, see Planning your log archive storage.

From IBM MQ for z/OS 9.1.4, IBM MQ for z/OS supports use of DSE with the active logs and page sets in addition to the support provided in earlier releases.

IBM MQ for z/OS does not support use of DSE for shared message data sets (SMDS).

See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Related concepts
Security concepts
Channel authentication records
Authority to work with IBM MQ objects on z/OS
Cryptographic security protocols: TLS
Related tasks
Setting up security on z/OS
Comparing link level security and application level security
Related reference
Messages for IBM MQ for z/OS

z/OS Security controls and options in IBM MQ for z/OS

You can specify whether security is turned on for the whole IBM MQ subsystem, and whether you want to perform security checks at queue manager or queue sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done for the whole queue manager. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to IBM MQ (including clients and channels), you can turn security checking off for the queue manager or queue sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue sharing group, no other IBM MQ checking is done; if you leave it turned on, IBM MQ checks your security requirements for other IBM MQ resources.

You can also turn security on or off for particular sets of resources, such as commands.

Queue manager or queue sharing group level checking

You can implement security at queue manager level or at queue sharing group level. If you implement security at queue sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier. It also makes it easy to add a new queue manager to the queue sharing group because it inherits the existing security profiles.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue sharing group that requires different levels of security to the other queue managers in the group.

Queue sharing group level security

Queue sharing group level security checking is performed for the entire queue sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each z/OS image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access only at the queue sharing group level.

You can use queue sharing group level security profiles to protect all types of resource, whether local or shared.

Queue manager level security
You can use queue manager level security profiles to protect all types of resource, whether local or shared.

Combination of both levels
You can use a combination of both queue manager and queue sharing group level security.

You can override queue sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

For more information, see Profiles to control queue sharing group or queue manager level security.

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for IBM MQ resource security. Normally, when a user attempts to access an IBM MQ resource, Security Server checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile for each queue manager. Control is implemented by the access that a user ID has to this profile.

Mixed case or uppercase IBM MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define IBM MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only IBM MQ RACF Classes as in previous releases, or
- Use the new mixed case IBM MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in IBM MQ for z/OS ; however, these resource names can only be protected by generic RACF profiles in the uppercase IBM MQ classes. When using mixed case IBM MQ RACF profile support you can provide a more granular level of protection by defining IBM MQ RACF profiles in the mixed case IBM MQ classes.

z/OS Resources you can protect in IBM MQ for z/OS

When a queue manager starts, or when instructed by an operator command, IBM MQ for z/OS determines which resources you want to protect.

You can control which security checks are performed for each individual queue manager. For example, you can implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager. It is done by issuing an MQCONN or MQCONNX request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager. However, if you do this any user can connect to that queue manager.

For the CICS adapter, only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to IBM MQ, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

You can turn connection security checking on or off at either queue manager or queue sharing group level.

Command security

Command security checking is carried out when a user issues an MQSC command from any of the sources described in Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS. You can make a separate check on the resource specified by the command as described in “Command resource security” on page 239.

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSC commands are entered from a console, the console must have the z/OS SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You must control who is allowed to update these data sets through normal data set protection.

You can turn command security checking on or off at either queue manager or queue sharing group level.

Command resource security

Some MQSC commands, for example defining a local queue, involve the manipulation of IBM MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM MQ checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning "PAYROLL". If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

You can turn command resource security checking on or off at either queue manager or queue sharing group level.

Channel security considerations

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol, although you can use TLS. If you are using APPC, you can flow user ID information from the sending MCA through the network to the destination MCA for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

For more information about the types of channel security available see Channel authentication records and Security exit overview

Related reference

"API-resource security in IBM MQ for z/OS" on page 240
Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

z/OS API-resource security in IBM MQ for z/OS

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into the following checks:

- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (using the MQOO_BROWSE option), but not to remove messages from the queue (using one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

You can turn queue security checking on or off at either queue manager or queue sharing group level.

Process

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

You can turn process security checking on or off at either queue manager or queue sharing group level.

Namelist

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

You can turn namelist security checking on or off at either queue manager or queue sharing group level.

Alternate user

Alternate user security controls whether one user ID can use the authority of another user ID to open an IBM MQ object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternative user ID when opening the reply-to queue.

The alternative user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

You can use alternative user IDs on any IBM MQ object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for z/OS data set security.

If alternate user security is not active, any user can use any other user ID as an alternative user ID.

You can turn alternate user security checking on or off at either queue manager or queue sharing group level.

Context

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section
The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section
The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQPUT or an MQPUT1 call is made. The application might generate the data, the data might be passed on from another message, or the queue manager might generate the data by default. For example, server programs can use context data to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternative user.

You use context security to control whether the user can specify any of the context options on any MQOPEN or MQPUT call. For information about the context options, see the MQOPEN options relating to message context. For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptorMQMD - Message descriptor.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

You can turn context security checking on or off at either queue, queue manager or queue sharing group level.

z/OS Availability on z/OS

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of IBM MQ can increase system availability if the queue manager or channel initiator fails. For more information about these features, see the following sections:

- Sysplex considerations

- Shared queues
- Shared channels
- IBM MQ network availability
- Using the z/OS Automatic Restart Manager (ARM)
- Using the z/OS Extended Recovery Facility (XRF)
- Using the z/OS GROUPUR attribute for recovery in a queue sharing group
- Where to find more information about availability

Sysplex considerations

In a *sysplex*, a number of z/OS operating system images collaborate in a single system image and communicate using a coupling facility. IBM MQ can use the facilities of the sysplex environment for enhanced availability.

Removing the affinities between a queue manager and a particular z/OS image allows a queue manager to be restarted on a different z/OS image in the event of an image failure. The restart mechanism can be manual, use ARM, or use system automation, if you ensure the following:

- All page sets, logs, bootstrap data sets, code libraries, and queue manager configuration data sets are defined on shared volumes.
- The subsystem definition has sysplex scope and a unique name within the sysplex.
- The level of *early code* installed on every z/OS image at IPL time is at the same level.
- TCP virtual IP addresses (VIPA) is available on each TCP stack in the sysplex, and you have configured IBM MQ TCP listeners and inbound connections to use VIPAs rather than default host names.

For more information about using TCP in a sysplex, see *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks® publication.

You can additionally configure multiple queue managers running on different operating system images in a sysplex to operate as a queue sharing group, which can take advantage of shared queues and shared channels for higher availability and workload balancing.

Shared queues

In the queue sharing group environment, an application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue sharing group can continue processing the queue. For information about high availability of shared queues, see “Advantages of using shared queues” on page 156.

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in “Peer recovery” on page 229.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, Db2, and IBM MQ for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in “Using the z/OS Automatic Restart Manager (ARM)” on page 243.

Shared channels

In the queue sharing group environment, IBM MQ provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). IBM MQ uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue sharing group. This is described in “Shared channels” on page 176.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in Shared outbound channels.

IBM MQ network availability

IBM MQ messages are carried from queue manager to queue manager in an IBM MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager, and the ability of an IBM MQ channel to detect a network problem and to reconnect.

TCP *Keepalive* is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The KAINT channel attribute determines the frequency of these packets for a channel.

AdoptMCA allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control AdoptMCA using the ADOPTMCA queue manager property with the MQSC utility or the AdoptNewMCAType property with the Programmable Command Formats interface.

ReceiveTimeout prevents a channel from being permanently blocked in a network receive call. The RCVTIME and RCVTMIN channel initiator parameters, determine the receive timeout characteristics for channels, as a function of their heartbeat interval. See Queue manager parameter for more details.

Using the z/OS Automatic Restart Manager (ARM)

You can use IBM MQ for z/OS in conjunction with the z/OS automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS fails, a whole group of related subsystems and applications also fail. ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue sharing groups.

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure.

To enable automatic restart, you must do the following:

1. Set up an ARM coupling data set.
2. Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
3. Start the ARM policy.

If you want to restart queue managers in different z/OS images automatically, every queue manager in each z/OS image on which that queue manager might be restarted must be defined with a sysplex-wide unique 4-character subsystem name.

Using ARM with IBM MQ is described in Using ARM in an IBM MQ network.

Using the z/OS Extended Recovery Facility (XRF)

You can use IBM MQ in an extended recovery facility (XRF) environment. All IBM MQ-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternative XRF processors.

If you use XRF for recovery, you must stop the queue manager on the active processor and start it on the alternative processor. For CICS, you can do this using the command list table (CLT) provided by CICS, or the system operator can do it manually. For IMS, this is a manual operation and you must do it after the coordinating IMS system has completed the processor switch.

IBM MQ utilities must be completed or terminated before the queue manager can be switched to the alternative processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent the queue manager starting on the alternative processor before the queue manager on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of IBM MQ on the two systems. You must include the BSDS as a protected resource, and you must include the active and alternative XRF processors in the GRS ring.

Using the z/OS GROUPUR attribute for recovery in a queue sharing group

Queue sharing groups (QSG) allow additional transactional facilities which are described in this topic. The GROUPUR attribute allows XA client applications to have any in-doubt transaction recovery that may be required, performed on any member of the QSG.

If an XA client application connects to a queue sharing group (QSG) through a Sysplex it cannot guarantee which specific queue manager it connects to. Use of the GROUPUR attribute by queue managers within the queue sharing group can enable any in-doubt transaction recovery that may be necessary to occur on any member of the QSG. Even if the queue manager to which the application was initially connected is not available, transaction recovery can take place.

This feature frees the XA client application from any dependency on specific members of the QSG and thus extends the availability of the queue manager. The queue sharing group appears to the transactional application as a single entity providing all the IBM MQ features and without a single queue manager point of failure.

This functionality is not apparent to the transactional application.

Where to find more information about availability

You can find more information about these topics from the following sources:

Table 24. Where to find more information about availability

Topic	**Where to look**
Queue sharing groups	“Shared queues and queue sharing groups” on page 137
System parameters	Configuring system parameters
Using the Automatic Restart Manager Utility programs	Using ARM in an IBM MQ network
MQSC commands	MQSC commands

z/OS Monitoring and statistics on IBM MQ for z/OS

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

IBM MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:

- "Online monitoring" on page 245
- "IBM MQ trace" on page 245
- "Events" on page 245

Online monitoring

IBM MQ includes the following commands for monitoring the status of IBM MQ objects:

- DISPLAY CHSTATUS displays the status of a specified channel.
- DISPLAY QSTATUS displays the status of a specified queue.
- DISPLAY CONN displays the status of a specified connection.

For more information about these commands, see The MQSC commands.

IBM MQ trace

IBM MQ supplies a trace facility that you can use to gather the following information while the queue manager is running:

Performance statistics

The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about Db2 usage (Db2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about SMDS usage (shared message data set statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the processor time spent processing MQI calls and about the number of MQPUT and MQGET requests made by a particular user.
- IBM MQ can also gather information about each task using IBM MQ. This data is gathered as a thread-level accounting record. For each thread, IBM MQ also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

IBM MQ events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple IBM MQ applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView®, to monitor reports and create the appropriate alerts.

Related tasks

Using IBM MQ trace

Using IBM MQ events

z/OS Unit of recovery disposition on z/OS

Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Transactions started by applications that have connected using the queue sharing group name also have a GROUP unit of recovery disposition.

When a transactional application connects with a GROUP unit of recovery disposition it is logically connected to the queue sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it has started that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which may be unavailable at that time.

Applications that connect with a QMGR unit of recovery disposition have a direct affinity to the queue manager to which they are connected. In a recovery scenario the transaction coordinator must reconnect to the same queue manager to resolve any in-doubt transactions, irrespective of whether the queue manager belongs to a queue sharing group.

When applications specify a queue sharing group name, and thus connect to a queue manager in a QSG with a GROUP unit of recovery disposition, the queue sharing group is logically a separate resource manager. This means that in-doubt transactions are only visible to an application if it reconnects with the same unit of recovery disposition. In-doubt transactions with a QMGR unit of recovery disposition are not visible to applications that have connected with a GROUP unit of recovery disposition and vice versa.

Related concepts

“Enabling GROUP units of recovery” on page 246
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 247
Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

z/OS Enabling GROUP units of recovery

A queue sharing group can configure and enable support for GROUP units of recovery.

To use GROUP units of recovery on a queue manager within a QSG, enable the GROUPUR queue manager attribute. For more information about this concept, see “Unit of recovery disposition on z/OS” on page 246 before reading the rest of this topic.

When the GROUPUR queue manager attribute is enabled, the queue manager accepts new connections with a GROUP unit of recovery disposition. If you disable this attribute new connections with this disposition are not accepted, although applications already connected is unaffected until they disconnect.

When an application connects with a GROUP unit of recovery disposition and either inquires what transactions are in doubt or attempts to resolve a transaction that was started elsewhere in the queue

sharing group (QSG), the queue manager to which it is now connected must be able to communicate with the other members of the queue sharing group so that it can process the request. To do this it uses a shared queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE. This queue must be on a recoverable application structure called CSQSYSAPPL. The structure must be recoverable because persistent messages are stored on this queue when processing resolution requests.

Before you can enable GROUP units of recovery, you must ensure that the coupling facility structure and the shared queue are defined. You can use the definitions in the CSQ4INSS sample. When the queue is defined, or detected during startup, each queue manager in the queue sharing group opens the queue so that it can receive incoming requests. If you want to delete or move the queue because it has been defined incorrectly you can request that the queue managers close their open handles on it by updating the queue object to inhibit MQGET requests. When you have made the necessary corrections, permitting applications to get messages from the queue once more directs each queue manager to reopen it. Use the DISPLAY QSTATUS command to identify what handles are open on a queue.

When you have completed this setup you can then enable GROUP units of recovery on each queue manager that you want transactional applications to be able to connect to with a GROUP unit of recovery disposition. This need not be all of the queue managers within the queue sharing group but if you choose to only enable this functionality on a subset of the queue sharing group you must ensure that your applications only attempt to connect to queue managers on which you have enabled it. For more information, see “Application support” on page 247.

When you attempt to enable the GROUPUR queue manager attribute, a number of configuration checks are performed. The queue manager checks that:

- It belongs to a queue sharing group.
- The shared-queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE has been defined, according to the the definition in CSQ4INSS.
- The SYSTEM.QSG.UR.RESOLUTION.QUEUE is on a recoverable CF structure called CSQSYSAPPL.

If any of the above checks fail, the GROUPUR attribute remains disabled and a message code is returned.

These configuration checks are also performed at queue manager startup if the queue manager attribute is enabled. If any of the checks fail during startup GROUP units of recovery is disabled and the queue manager issues a message identifying which check failed. When you have performed the necessary corrective action you must then re-enable the queue manager attribute.

z/OS Application support

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Support for the GROUP unit of recovery disposition is limited to certain types of transactional applications for which IBM MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- IBM MQ extended transactional client applications
- IBM MQ classes for JMS applications running in an application server, such as WebSphere Application Server.
- CICS applications running in CICS Transaction Server 4.2 or later, when the CICS MQCONN resource definition is configured with RESYNCMEMBER(GROUPRESYNC).

Related concepts

“ IBM MQ extended transactional client applications” on page 248
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 248
Use this page to determine how CICS can use the GROUP unit of recovery disposition.

z/OS *IBM MQ extended transactional client applications*

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of an IBM MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to IBM MQ over TCP/IP, rather than local bindings. These client applications connect to IBM MQ for z/OS over network connections, such as via TCP/IP. For these applications, it is the value specified for the QMNAME parameter of the xa_info string passed in the xa_open call that specifies whether a QMGR or GROUP unit of recovery disposition is used. For more information about xa_open, see The format of an xa_open string and Additional error processing for xa_open. For JMS applications this is done by specifying the name of the queue sharing group (QSG) in the ConnectionFactory instead of the name of a specific queue manager.

For XA client applications to take advantage of using the GROUP unit of recovery disposition you must configure your TCP/IP setup to allow your client applications to be routed to the queue managers in the queue sharing group that have the GROUPUR attribute enabled, rather than a specific queue manager. One of the dynamic virtual IP address technologies that you can use to do this is the z/OS SysPlex Distributor. See z/OS Communications Server and z/OS 基本技能: 动态虚拟寻址 for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your queue sharing group, ensure that your client applications cannot be routed to those on which it is not enabled.

Your client applications do not have to connect to the queue sharing group using shared channels.

z/OS *CICS applications*

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

CICS 4.2 and later provides the group resynchronization option, RESYNCMEMBER(GROUPRESYNC) in an MQCONN resource definition. A CICS configured with this option can connect to any suitable queue manager in a queue sharing group which is running on the same LPAR as that CICS region. To support the CICS GROUPRESYNC option, a queue manager must be running at MQ V7.1 or later, and be enabled for GROUPUR support.

Transactions running within a CICS region connected to MQ using GROUPRESYNC create units of work with GROUP unit of recovery disposition.

You can use RESYNCMEMBER(GROUPRESYNC) to enable faster recovery after a queue manager failure as it enables the CICS region to immediately connect to an alternative eligible queue manager running on the same LPAR, resolving any indoubt transactions as necessary, without waiting for queue manager restart.

RESYNCMEMBER(GROUPRESYNC) also enables more flexible restart options for CICS. A CICS region with its MQ connection configured to use GROUPRESYNC and MQ shared queues can be restarted on any LPAR where there is a queue manager running as a member of the same queue sharing group.

z/OS IBM MQ and other z/OS products

Use this topic to understand how IBM MQ can work with other z/OS products.

Related concepts

"IBM MQ and CICS" on page 249
All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

"IBM MQ for z/OS and WebSphere Application Server" on page 255
Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Related reference

"IBM MQ and IMS" on page 250
Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

"IBM MQ and the z/OS Batch, TSO, and RRS adapters" on page 253
Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

z/OS IBM MQ and CICS

All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to IBM MQ section of the CICS documentation.

Related tasks

Using IBM MQ with CICS

z/OS CICS group attach

CICS group attach provides the ability for a CICS region to connect to any active member of an IBM MQ queue sharing group on the same LPAR rather than specifying an individual queue manager. CICS still connects to a single queue manager at a time.

You require at least two queue managers on the LPAR to support CICS group attach. Using group attach provides higher availability as you do not need a particular queue manager to be active. CICS connects to any queue manager in the queue sharing group on the LPAR.

For more information, see the CICS documentation on the MQCONN resource.

CICS attempts to connect to MQNAME passed as if it were a queue manager:

- If the queue manager exists and is active, the connection will work.
- If the connection fails, CICS queries the status of queue managers in the group to ascertain which are active on same LPAR.
- If multiple queue managers are active, CICS checks for RESYNCMEMBER(YES) and the UOW status to determine whether CICS needs to connect, or should connect, to a particular member, or wait if not active.
- If there is no need to connect to a particular member, CICS selects a queue manager (using a randomizing algorithm).
- CICS attempts to connect to chosen queue manager.
- If the attempt fails then, depending upon the return code, CICS chooses the next member, then goes through the selection loop again.
- If no queue managers are active, CICS issues multiple connections to the list of queue managers and waits on ECBLIST until the first queue manager becomes available.

Related concepts

“Group units of recovery (GROUPUR) for CICS” on page 249
The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

Related information

Support for IBM MQ queue sharing groups

z/OS Group units of recovery (GROUPUR) for CICS

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

If a CICS region is working with a queue manager, and the queue manager ends abnormally, then any indoubt transactions are recovered. This eliminates the need for the CICS region to wait for the queue

manager that it was working with to restart, and then resolve any in doubt units of work. This means that you need at least two queue managers on the LPAR, so that CICS can connect to another queue manager in the event of an abnormal termination of the first queue manager.

The new RESYNCMEMBER(GROUPRESYNC) setting on the CICS MQCONN definition:

- Uses the IBM MQ group attach function and peer recovery.
- Requires a queue manager with the GROUPUR attribute enabled.
- Still supports the existing CICS MQCONN RESYNCMEMBER settings (YES and NO):
  - Uses the existing CICS group attach function and no peer recovery.
  - Changing RESYNCMEMBER settings takes effect next time CICS connects to IBM MQ.

Related concepts

"Enabling GROUP units of recovery" on page 246
A queue sharing group can configure and enable support for GROUP units of recovery.

z/OS IBM MQ and IMS

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

The optional additional IBM MQ - IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with IBM MQ, without the need to rewrite them.

For more information about these components, see the following subtopics:

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Setting up the IMS adapter

Setting up the IMS bridge

Operating the IMS adapter

Related reference

MQIIH - IMS information header

z/OS The IMS adapter

The IMS adapter is an interface between IMS application programs and an IBM MQ subsystem.

The IBM MQ adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an IBM MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to IBM MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. Usually, IMS connects to IBM MQ automatically without operator intervention.

The IMS adapter provides access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ with IMS acting as the syncpoint coordinator. Conversations where IMS is not the syncpoint coordinator, for example APPC- protected (SYNCLVL=SYNCPT) conversations, are not supported by the IMS adapter.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in "The IMS trigger monitor" on page 251.

You can use IBM MQ with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error.

Note: As of IMS 15.2 Extended Recovery Facility (XRF) is no longer supported. See the IMS documentation for more information.

Using the adapter

The application programs and the IMS adapter run in the same address space. The queue manager is separate, in its own address space.

You must link-edit each program that issues one or more MQI calls to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the IBM MQ-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to IBM MQ. However, the IMS terminal operator has no control over the IBM MQ address space. For example, the operator cannot shut down IBM MQ from an IMS address space.

Restrictions

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

The IMS trigger monitor

The IMS trigger monitor (**CSQQTRMN**) is an IBM MQ-supplied IMS application that starts an IMS transaction when an IBM MQ event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, you can start an instance of CSQQTRMN to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. When it has started, the trigger monitor runs until IBM MQ or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions that are started by the trigger monitor contain the following:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is protected by Security Server (previously known as RACF), you might need to define CSQQTRMN as a user ID to Security Server.

z/OS The IBM MQ - IMS bridge

The IBM MQ - IMS bridge is the component of IBM MQ for z/OS that allows direct access from IBM MQ applications to applications on your IMS system.

The IBM MQ - IMS bridge enables *implicit MQI support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by IBM MQ messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

In bridge applications there are no IBM MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. IBM MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one IBM MQ message.

The IMS bridge is illustrated in Figure 78 on page 252.

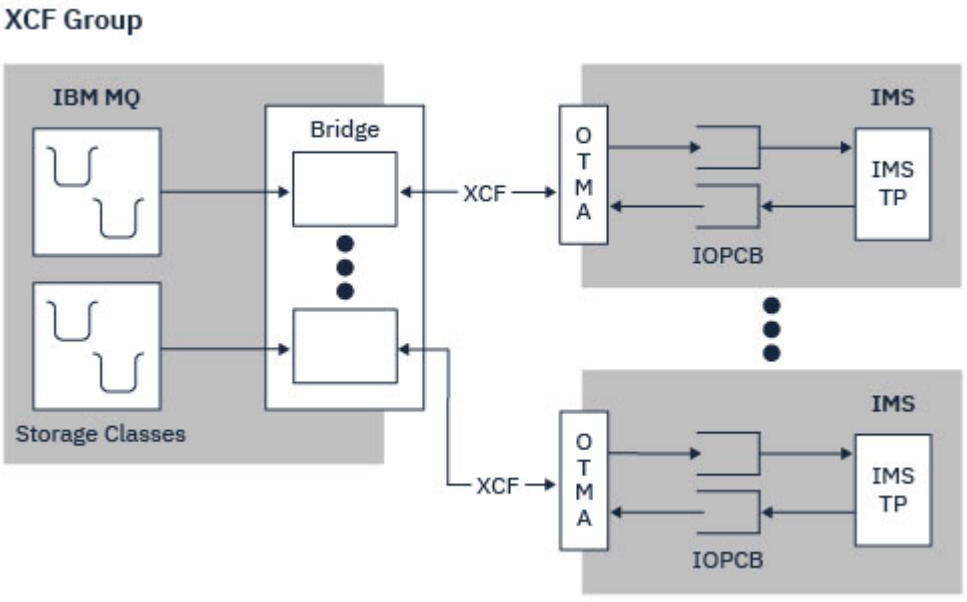


Figure 78. The IBM MQ - IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

See Setting up the IMS bridge for information on setting up an IMS bridge and adding an additional IMS connection to the same queue manager.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS. It functions as an interface for host-based communications servers accessing IMS TM applications through the z/OS Cross Systems coupling facility (XCF).

OTMA enables clients to connect to IMS to provide high performance for interactions between clients and IMS for a large network or large number of sessions. OTMA is implemented in a z/OS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA Resource Monitoring

Support for the x'3C' OTMA protocol messages, available in IMS v10 or higher, is included in the IBM MQ - IMS bridge in IBM MQ for z/OS. These messages are sent to OTMA clients by IMS to report its health status.

If an IMS partner is unable to process the volume of transaction requests being sent then it will notify IBM MQ that a flood warning has occurred. In response IBM MQ will slow down the rate at which requests are sent over the bridge.

If IMS is still unable to process the transaction requests and a full flood condition occurs all TPIPEs to the IMS partner are suspended. Upon notification from the IMS partner that the flood or flood-warning condition has been relieved IBM MQ will resume all suspended TPIPEs, if appropriate, and gradually increase the rate at which transaction requests are sent until the maximum rate is achieved. Console messages are issued by IBM MQ in response to a change in the status of IMS partners.

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from IBM MQ

To submit an IMS transaction that uses the bridge, applications put messages on an IBM MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the IBM MQ - IMS bridge to make assumptions about the data in the message.

IBM MQ then puts the message to an IMS queue (it is queued in IBM MQ first to enable the use of syncpoints to assure data integrity). The storage class of the IBM MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the IBM MQ - IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on IBM MQ for z/OS.

Data returned from the IMS system is written directly to the IBM MQ reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the **ReplyToQMgr** field of the MQMD.)

Related concepts

IMS and IMS bridge applications on IBM MQ for z/OS

Related tasks

Customizing the IMS bridge

Related reference

“IBM MQ and IMS” on page 250

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

z/OS IBM MQ and the z/OS Batch, TSO, and RRS adapters

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between IBM MQ and z/OS application programs running under JES, TSO, or z/OS UNIX System Services. These adaptors enable z/OS application programs to use the MQI.

The adapters provide access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and IBM MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to IBM MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ. It does not support multi-phase commit protocols. The RRS adapter enables IBM MQ applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by z/OS Resource Recovery Services (RRS).

The adapters use the z/OS STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the IBM MQ termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in IBM MQ (for example, a signal or a wait).

The Batch/TSO adapter

The IBM MQ Batch/TSO adapter provides IBM MQ support for z/OS Batch and TSO applications. All application programs that run under z/OS Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of z/OS that provides a system-wide service for coordinating two-phase commit across z/OS products. The IBM MQ Batch/TSO RRS adapter (the RRS adapter) provides IBM MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables IBM MQ to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, Db2).

The RRS adapter provides two stubs; you must link-edit application programs that want to use RRS with one of these stubs.

CSQBRSTB
: This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

  You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.

CSQBRRSI
: This stub allows you to use MQI calls **MQCMIT** and **MQBACK** ; IBM MQ actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see The RRS batch adapter.

Where to find more information about the z/OS Batch, TSO, and RRS adapters

You can find more information about the topics in this section in the following sources:

Table 25. Where to find more information about using z/OS Batch with IBM MQ	
Topic	**Where to look**
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

z/OS IBM MQ for z/OS and WebSphere Application Server

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Applications written in Java that are running under WebSphere Application Server can use the Java Message Service (JMS) specification to perform messaging. Point-to-point messaging in this environment can be provided by an IBM MQ for z/OS queue manager.

A benefit of using an IBM MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of an IBM MQ network. For example, they can use the IMS bridge, or exchange messages with queue managers running on other platforms.

Connection between WebSphere Application Server and a queue manager

See Using IBM MQ and WebSphere Application Server together for more information.

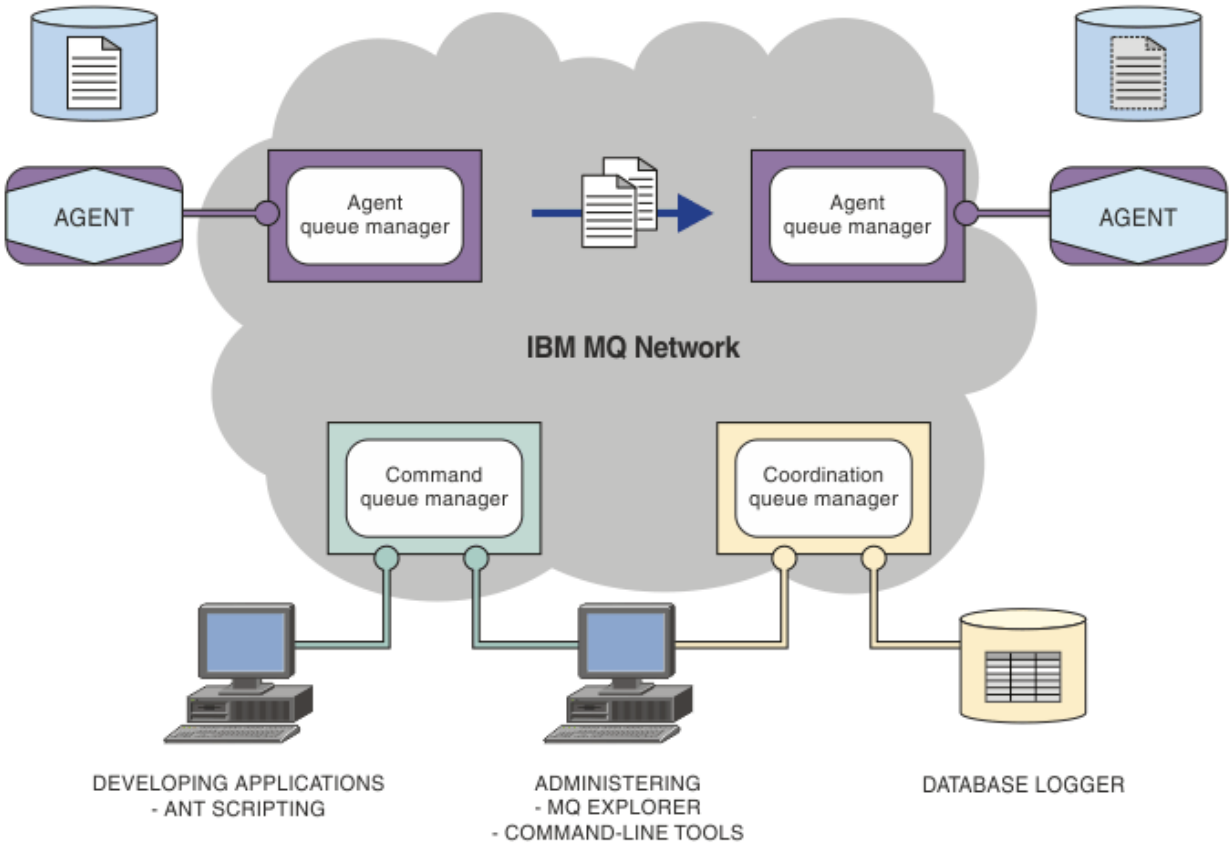
Using IBM MQ functions from JMS applications

By default, JMS messages held on IBM MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy IBM MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for IBM MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto IBM MQ messages.

Managed File Transfer

Managed File Transfer 将以受管和可审计方式在系统之间传输文件，而不用考虑文件大小或使用的操作系统。

您可以使用 Managed File Transfer 构建定制、可扩展且自动化的解决方案，使您能够管理和信任文件传输并保证文件传输的安全。 Managed File Transfer 可消除昂贵的冗余、降低维护成本并最大限度地利用现有的 IT 投资。



该图显示了简单的 Managed File Transfer 拓扑。 其中包含两个代理，分别与它们在 IBM MQ 网络中的代理队列管理器相连。 通过 IBM MQ 网络，将文件从位于图中一侧的代理传输到位于图中另一侧的代理。 在 IBM MQ 网络中还包含一个协调队列管理器和一个命令队列管理器。 应用程序和工具连接到这些队列管理器，以在 IBM MQ 网络中配置，管理，操作和记录 Managed File Transfer 活动。

根据您的操作系统和总体设置，Managed File Transfer 可以作为四个不同的选项进行安装。 这些选项为 Managed File Transfer Agent、Managed File Transfer Logger、Managed File Transfer Service 或 Managed File Transfer Tools。 有关更多信息，请参阅 Managed File Transfer 产品选项。

可以使用 Managed File Transfer 来执行下列任务：

- 创建受管文件传输
  - Windows Linux 在 Linux 或 Windows 平台上从 IBM MQ Explorer 创建新的文件传输。
  - 通过所有受支持平台上的命令行创建新的文件传输。
  - 将文件传输功能集成到 Apache Ant 工具中。
  - 通过将消息放置在代理命令队列上，编写控制 Managed File Transfer 的应用程序。
  - 安排稍后进行文件传输。 还可以根据一系列文件系统事件（例如，正在创建新文件）触发调度的文件传输。
  - 持续监视资源（例如，目录），并在该资源的内容满足某个预定义条件时启动任务。 此任务可以是文件传输， Ant 脚本或 JCL 作业。
  - 与 IBM MQ 队列进行文件传输。
  - 在 FTP、FTPS 或 SFTP 服务器之间进行文件传输。
  - 将文件传输至 Connect:Direct 节点和从这些节点传输文件。
  - 传输文本和二进制文件。 将在源和目标系统的代码页和行结束约定之间自动转换文本文件。
  - 可以使用针对基于安全套接字层 (SSL) 连接的行业标准保证传输安全。

- 查看正在进行的传输并记录有关网络中所有传输的信息。
  - Windows Linux 从 Linux 或 Windows 平台上的 IBM MQ Explorer 查看正在进行的传输的状态。
  - Windows Linux 在 Linux 或 Windows 平台上使用 IBM MQ Explorer 检查已完成传输的状态。
  - 使用 Managed File Transfer 数据库记录器功能将日志消息保存到 Db2 或 Oracle 数据库。

Managed File Transfer 基于 IBM MQ 构建，WebSphere MQ 在应用程序之间提供确定的仅一次性消息传递。 您可以利用 IBM MQ 的各种功能。 例如，您可以使用通道压缩来压缩通过 IBM MQ 通道在代理之间发送的数据，或使用 SSL 通道保证代理之间所发送数据的安全。 文件可靠地传输，并且可以容忍通过其执行文件传输的基础结构的故障。 如果经历网络中断，文件传输将在连接恢复时从断开的位置重新启动。

通过将文件传输与现有的 IBM MQ 网络进行整合，可以避免耗用维护两个独立的基础结构所需的资源。 如果您还不是 IBM MQ 客户，请创建 IBM MQ 网络以支持 Managed File Transfer，从而为未来的 SOA 实施构建主干。 如果您已是 IBM MQ 客户，那么 Managed File Transfer 可以利用现有 IBM MQ 基础结构，包括 IBM MQ Internet Pass-Thru 和 IBM Integration Bus。

您可以利用 IBM MQ 高可用性解决方案来提高 Managed File Transfer 配置的弹性。 如果代理程序使用复制的数据队列管理器 (RDQM)，那么必须将其配置为使用浮动 IP 地址功能。 这意味着代理程序使用相同的 IP 地址与当前正在运行的三个 RDQM 实例中的任何一个进行通信，并在故障转移时自动重新连接 (请参阅 RDQM 高可用性 和 创建和删除浮动 IP 地址)。 如果使用多实例队列管理器解决方案，那么应用程序将使用不同的 IP 地址与每个实例进行通信，由故障转移时的客户机重新连接处理 (请参阅 多实例队列管理器 和 通道和客户机重新连接)。

Managed File Transfer 与许多其他 IBM 产品集成：

IBM Integration Bus

已由 Managed File Transfer 作为 IBM Integration Bus 流的一部分传输的进程文件。 有关更多信息，请参阅 从 IBM Integration Bus 使用 MFT。

IBM Sterling Connect:Direct

使用 Managed File Transfer Connect:Direct 网桥将文件传输到现有 Connect:Direct 网络或从现有网络传输文件。 有关更多信息，请参阅 Connect:Direct 网桥。

IBM Tivoli Composite Application Manager

IBM Tivoli Composite Application Manager 提供可用于监视发布到协调队列管理器的信息的代理。

相关概念

Managed File Transfer 产品选项

第 258 页的『MFT 拓扑概述』
有关 Managed File Transfer 代理如何连接 IBM MQ 网络中的协调队列管理器的概述。

第 257 页的『MFT 如何使用 IBM MQ? 』
Managed File Transfer 可以使用多种方式与 IBM MQ 进行交互。

MFT 如何使用 IBM MQ?

Managed File Transfer 可以使用多种方式与 IBM MQ 进行交互。

- Managed File Transfer 通过将每个文件分割为一条或多条消息并通过 IBM MQ 网络传输这些消息，从而在代理进程之间传输文件。
- 代理进程通过使用非持久消息来移动文件数据，以在最大程度上降低对 IBM MQ 日志的影响。 通过彼此通信，代理进程可控制好传输包含文件数据的消息流。 这可防止包含文件数据的消息在 IBM MQ 传输队列上聚集，并确保当有任何非持久消息未交付时会重新发送文件数据。
- Managed File Transfer 代理会使用多个 IBM MQ 队列。 有关更多信息，请参阅 MFT 系统队列和系统主题。
- 虽然三个队列中的部分队列严格限制为内部使用，但是代理可接受形式为特殊格式的命令消息的请求，这些消息发送至特定队列以供该代理从中读取。 命令行命令和 IBM MQ Explorer 插件均可将 IBM MQ 消息发

送至代理以指示代理执行所需的操作。 您可以使用此方式来编写用于与代理进行交互的 IBM MQ 应用程序。 有关更多信息，请参阅 通过将消息放入代理命令队列来控制 MFT。

- Managed File Transfer 代理会将有关其状态以及传输进度和结果的信息发送至已指定为协调队列管理器的 MQ 队列管理器。 此信息由协调队列管理器发布，可供希望监视传输进度或保留发生的传输记录的应用程序来预订。 命令行命令和 IBM MQ Explorer 插件均可使用已发布的信息。 您可编写使用此信息的 IBM MQ 应用程序。 有关将信息发布到的主题的更多信息，请参阅 SYSTEM.FTE 主题。
- Managed File Transfer 的关键组件将会利用 IBM MQ 队列管理器的功能来存储和转发消息。 这意味着如果您遇到停机，那么基础结构中未受影响的部分仍可继续传输文件。 这可扩展至协调队列管理器，在此情况下，存储转发和持久预订的组合允许协调队列管理器承受变为不可用，而不会丢失有关发生的文件传输的关键信息。

MFT 拓扑概述

有关 Managed File Transfer 代理如何连接 IBM MQ 网络中的协调队列管理器的概述。

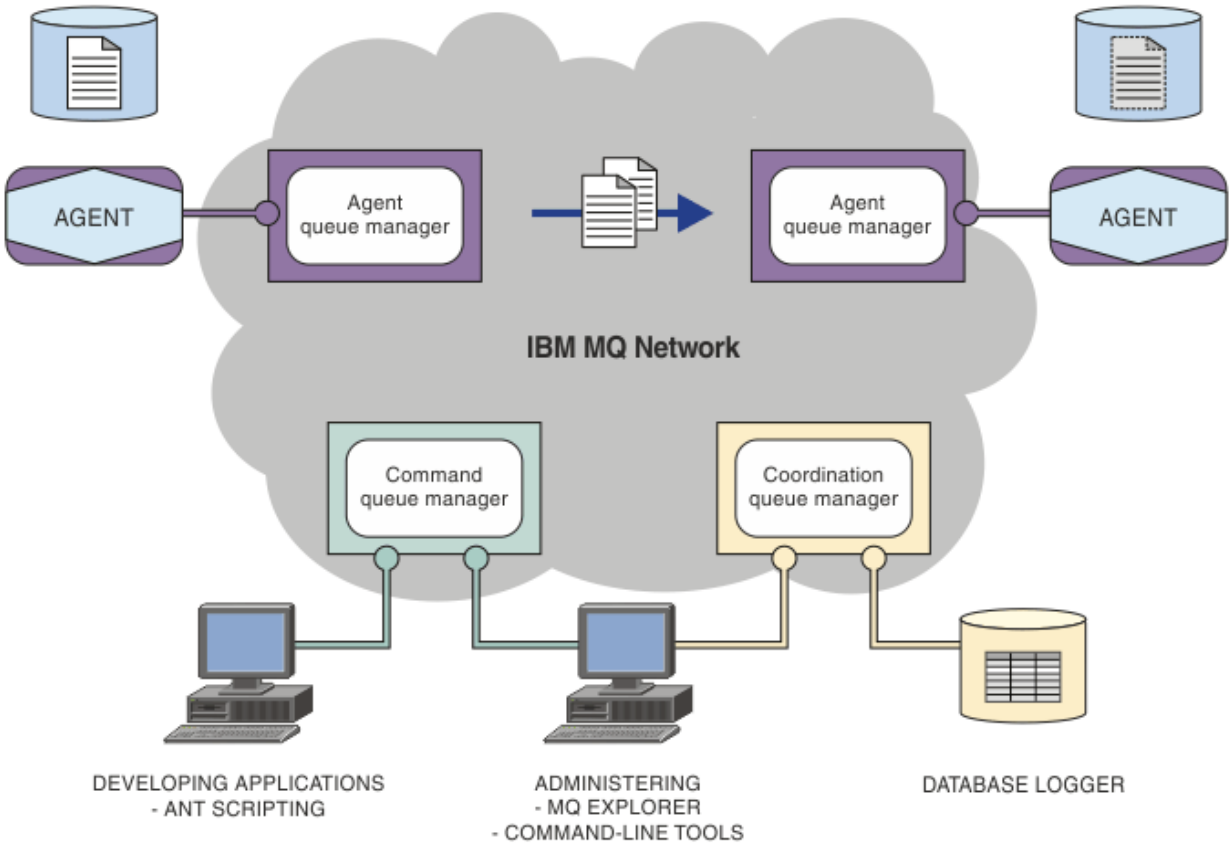
Managed File Transfer 代理发送并接收传输的文件。 每个代理在其关联队列管理器上都有自己的队列集，并且代理将以绑定或客户机方式连接到其队列管理器。 代理还可以使用协调队列管理器作为其队列管理器。

协调队列管理器广播审计和文件传输信息。 协调队列管理器表示代理集合、传输状态和传输审计信息的单个端点。 协调队列管理器并非执行传输所必需的。 如果协调队列管理器临时不可用，传输将照常继续。 审计和状态消息将会存储在代理队列管理器中，直至协调队列管理器可用，然后将照常处理。

代理注册协调队列管理器并将其详细信息发布到队列管理器。 此代理信息供 Managed File Transfer 插件用于支持从 IBM MQ Explorer 启动传输。 协调队列管理器上收集的代理信息还将供命令用来显示代理信息和代理状态。

传输状态和传输审计信息将在协调队列管理器上发布。 传输状态和传输审计信息供 Managed File Transfer 插件用于从 IBM MQ Explorer 监视传输的进度。 可以保留存储在协调队列管理器上的传输审计信息以提供可审计性。

命令队列管理器用于连接到 IBM MQ 网络，也是在您发出 Managed File Transfer 命令时要连接到的队列管理器。



相关概念

第 255 页的『Managed File Transfer』
Managed File Transfer 将以受管和可审计方式在系统之间传输文件，而不用考虑文件大小或使用的操作系统。

第 257 页的『MFT 如何使用 IBM MQ？』
Managed File Transfer 可以使用多种方式与 IBM MQ 进行交互。

Managed File Transfer 方案 (scenario)

MFT REST API 概述

REST API 支持某些 Managed File Transfer 命令，包括列出传输以及有关文件传输代理的详细信息。

REST API 包含用于列出所有当前 Managed File Transfer 传输以及用于查询 Managed File Transfer 代理的状态的选项。 有关更多信息，请参阅 REST API MFT 入门。

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru (MQIPT) 是 IBM MQ 的可选组件，可用于在跨因特网的远程站点之间实现消息传递解决方案。

要获取 IBM MQ 9.4.x 的 MQIPT 安装文件，请转至 IBM Fix Central for IBM MQ。

您可以使用 MQIPT 来连接任何受支持的 IBM MQ 版本。 您不必安装与 MQIPT 版本相同的任何其他 IBM MQ 组件。

如果您已购买 IBM MQ 权利，那么可以安装所需数量的 MQIPT 副本。 MQIPT 安装不计入已购买的 IBM MQ 权利。 有关 IBM MQ 许可的更多信息，请参阅 IBM MQ 许可证信息。

注: 本文档与 IBM MQ 9.4 中的 MQIPT 相关。 有关 IBM Documentation 中的 MQIPT 支持包 (V 2.1) 文档，请参阅 IBM MQ 9.0 文档中的 MQIPT (SupportPac MS81) 。

注: 如果您正在使用 MQIPT 2.1 或更低版本，那么建议您升级到 MQIPT for IBM MQ 9.4，因为 MQIPT 支持包的支持结束日期为 2020 年 9 月 30th 。

IBM MQ Internet Pass-Thru 作为独立的服务运行，可以在两个 IBM MQ 队列管理器之间，或者在 IBM MQ 客户机和 IBM MQ 队列管理器之间接收和转发 IBM MQ 消息流。

当客户机和服务器没有位于同一物理网络时，MQIPT 启用此连接。

可以将 MQIPT 的一个或多个实例放在两个 IBM MQ 队列管理器之间的通信路径中，也可以放在 IBM MQ 客户机与 IBM MQ 队列管理器之间的通信路径中。 MQIPT 实例允许两个 IBM MQ 系统交换信息，两个系统之间不需要直接 TCP/IP 连接。 如果防火墙配置禁止两个系统之间的直接 TCP/IP 连接，那么此体系结构很有用。

MQIPT 在一个或多个 TCP/IP 端口上侦听入局连接。 这些连接可以包含正常的 IBM MQ 消息，在 HTTP 中隧道传送的 IBM MQ 消息，或者使用传输层安全性 (TLS) 或安全套接字层 (SSL) 加密的消息。 MQIPT 可以处理多个并发连接。

发出初始 TCP/IP 连接请求的 IBM MQ 通道被称为调用者，其尝试连接到的通道被称为响应者，最终尝试联系的队列管理器称为目标队列管理器。

MQIPT 将数据从源转发至目标时将数据保留在内存中。 没有任何数据保存在磁盘上 (操作系统将内存页面保存到磁盘的内存除外)。 MQIPT 显式访问磁盘的唯一时间是读取其配置文件并写入连接日志和跟踪记录。

IBM MQ 通道类型的完整范围可以通过 MQIPT 的一个或多个实例进行连接。 通信路径中存在 MQIPT 不会影响已连接的 IBM MQ 组件的功能特征。 但是，消息传输的性能可能会受到影响。

MQIPT 可以与 IBM MQ 配合使用，如 第 263 页的『MQIPT 的可能配置』中所述。

要安装 MQIPT，请参阅 安装 MQIPT。

相关任务

配置 IBM MQ Internet Pass-Thru

管理和配置 IBM MQ Internet Pass-Thru

相关参考

IBM MQ Internet Pass-Thru 配置参考信息

MQIPT 的使用

IBM MQ Internet Pass-Thru (MQIPT) 有许多可能的用途。

MQIPT 可以用作通道集中器

通过这种方式使用 MQIPT，进出多个单独主机的通道似乎是进出防火墙，就好象所有通道都进出 MQIPT 主机。 这样可以简化防火墙过滤规则的定义和管理。

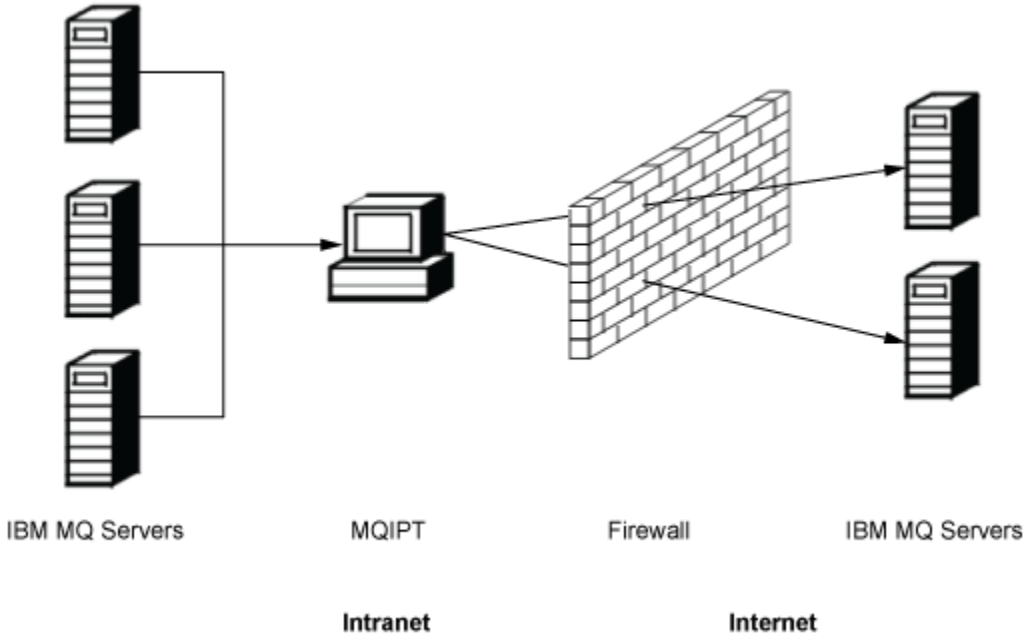


图 *79: MQIPT* 作为通道集中器的示例

可以将 MQIPT 放入 DMZ 以提供单点访问

如果 MQIPT 位于 DMZ 防火墙内（确保局域网安全的防火墙配置）一个具有已知的受信因特网协议 (IP) 地址的计算机上，那么可以使用 MQIPT 侦听入站 IBM MQ 通道连接（该通道连接然后转发到受信内部网）；内部防火墙必须允许此受信计算机进行入站连接。 在此配置中，MQIPT 阻止外部访问请求接收受信内部网内计算机的真实 IP 地址。 MQIPT 以这种方式提供单点访问。 如果需要，可以将 MQIPT 配置为接受 TLS 连接，并使用单独的 TLS 连接将数据转发到目标，从而终止 DMZ 中的 TLS 会话。

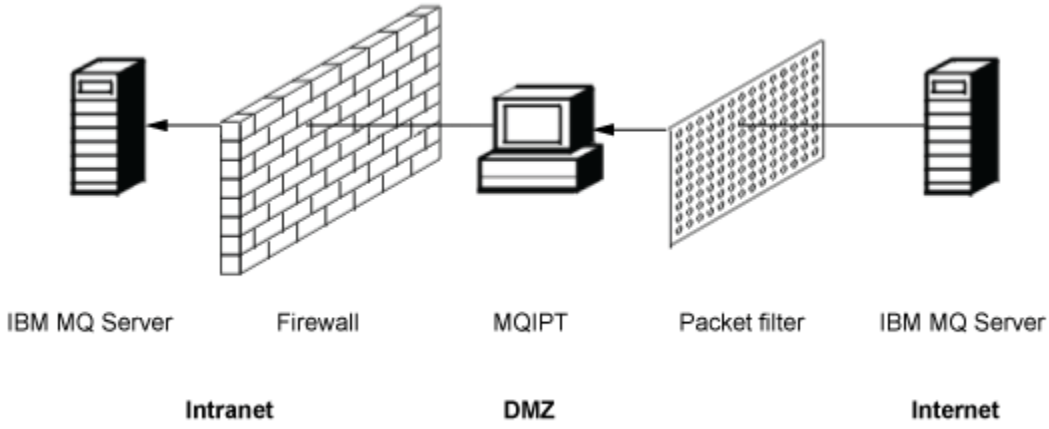


图 *80: DMZ* 防火墙中 *MQIPT* 示例

MQIPT 可以通过 HTTP 隧道传送通信

如果两个 MQIPT 实例部署一致，那么它们可以使用 HTTP 通信。 HTTP 隧道传送功能支持使用现有 HTTP 代理通过防火墙传递请求。 第一个 MQIPT 将 IBM MQ 协议插入 HTTP，第二个从 HTTP 包装器抽取 IBM MQ 协议，并将其转发到目标队列管理器。

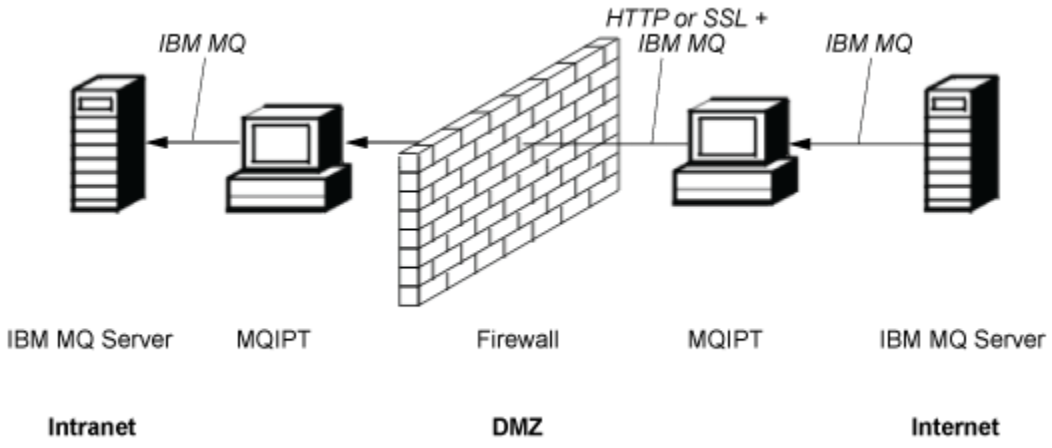


图 *81: MQIPT* 和 *HTTP* 隧道传送示例

MQIPT 可以加密消息

如果 MQIPT 如先前示例配置，通过防火墙传输请求之前可以加密请求。 第一个 MQIPT 对数据进行加密，第二个使用 SSL/TLS 对其进行解密，然后再将其发送到目标队列管理器。

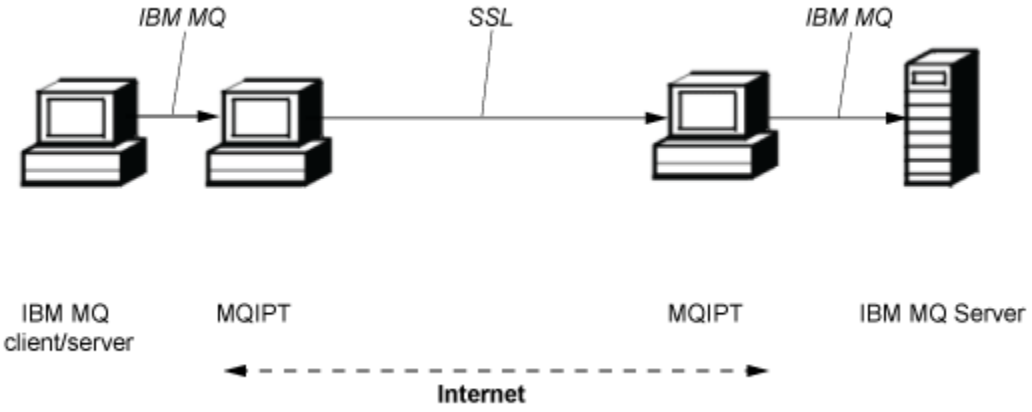


图 *82: MQIPT* 和 *SSL/TLS* 示例

MQIPT 的工作方式

在最简单的配置中，MQIPT 充当 IBM MQ 协议转发器。 它在 TCP/IP 端口上侦听并且接受来自 IBM MQ 通道的连接请求。

如果收到格式正确的请求，MQIPT 将在自身和目标 IBM MQ 队列管理器之间建立进一步的 TCP/IP 连接。 然后，将从入站连接接收到的所有协议包传递给目标队列管理器，并将目标队列管理器的协议包返回至原始的入站连接。

不涉及对 IBM MQ 协议（客户机/服务器或队列管理器到队列管理器）的更改，因为任何一端都不直接觉察到中介的存在。 不需要使用新版本的 IBM MQ 客户机或服务器代码。

要使用 MQIPT，必须将调用者通道配置为使用 MQIPT 主机名和端口，而不是目标队列管理器的主机名和端口。 这是使用 IBM MQ 通道的 **CONNAME** 属性定义的。 MQIPT 读取入站数据，并只将其传递给目标队列管理器。 其他配置字段（例如客户机/服务器通道的用户标识和密码）同样也传递到目标队列管理器。

多个队列管理器

可以使用 MQIPT 以允许访问多个目标队列管理器。 要实现此目的，必须存在一个机制来告知 MQIPT 连接到哪个队列管理器，因此， MQIPT 使用入站 TCP/IP 端口号来确定要连接到哪个队列管理器。

因此，您可以配置 MQIPT 在多个 TCP/IP 端口上侦听。 每一个侦听端口通过 MQIPT 路由映射到目标队列管理器。 您可以定义多达 100 个此类路由，从而将侦听的 TCP/IP 端口与目标队列管理器的主机名和端口关联。 这意味着目标队列管理器的主机名（IP 地址）从未对始发通道可视。 每个路由可以处理其侦听端口和目标之间的多个连接，每个连接独立运作。

MQIPT 配置文件

MQIPT 使用名为 `mqipt.conf` 的配置文件。 此文件包含所有路由及其关联属性的定义。 有关 `mqipt.conf` 的更多信息，请参阅 管理和配置 IBM MQ Internet Pass-Thru 。

启动 MQIPT 时，它将启动配置文件中列出的所有路由。 消息将写入系统控制台，显示每个路由的状态。 当对路由显示 MQCPI078 消息时，该路由准备好接受连接请求。

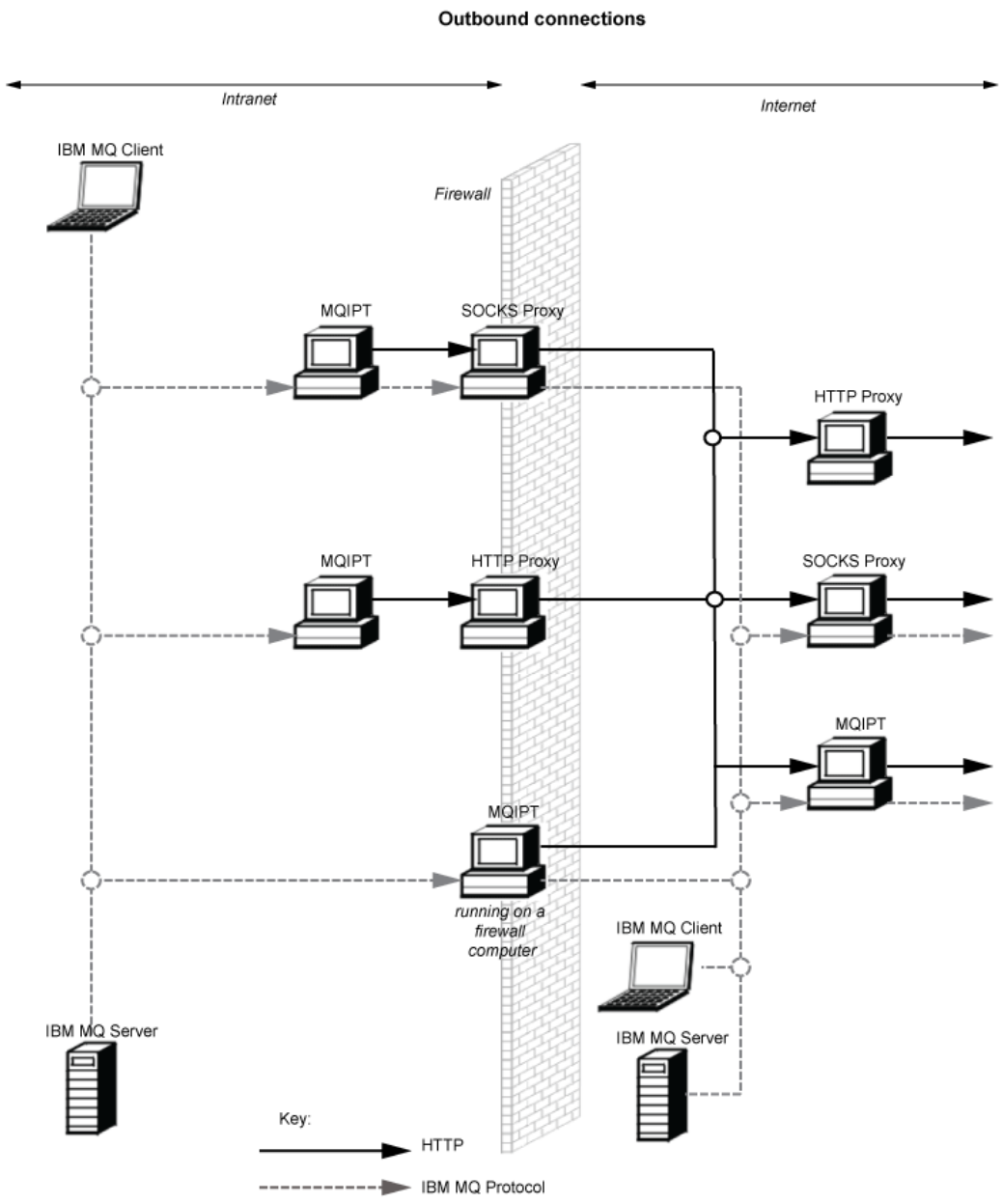
MQIPT 的可能配置

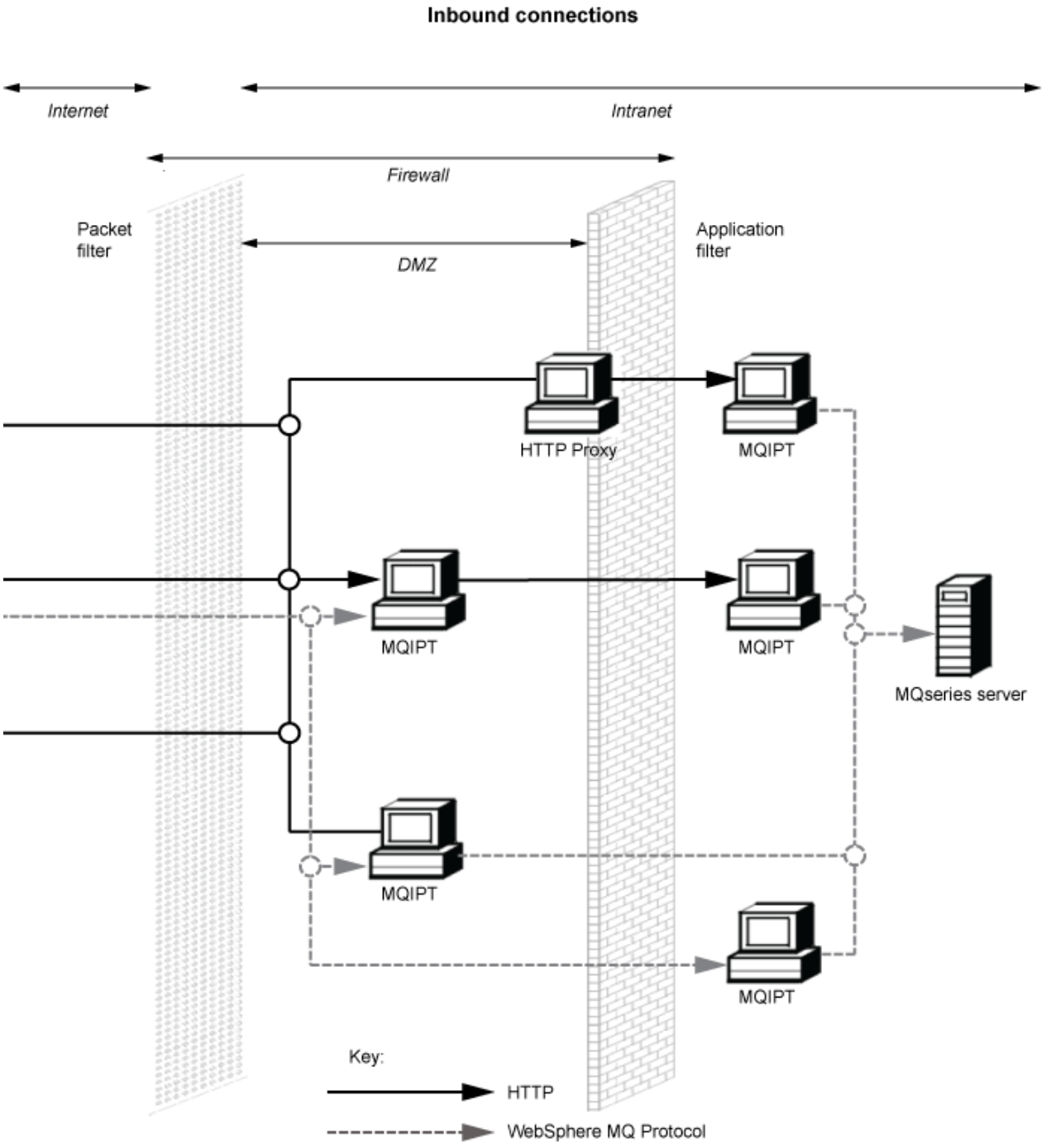
MQIPT 可与 IBM MQ 和 IBM Integration Bus 结合使用。

以下多部分图显示了 IBM MQ 拓扑中 MQIPT 的许多可能的配置。 该图演示了 MQIPT 发送消息的不同方式。 显示了内部网、防火墙内和防火墙外的互联网上的客户机和服务器，显示了向转发这些消息的 MQIPT、HTTP 代理或 SOCKS 代理传递消息。

消息通过入站防火墙传递到服务器之前，先由 DMZ 中的 MQIPT 代理或 HTTP 代理接收。

请注意，防火墙的内部网一侧上的 HTTP 代理、SOCKS 代理和 MQIPT 计算机代表互联网上多个计算机可能链接在一起。 例如， MQIPT 计算机在到达目标前可以通过一个或多个 SOCKS 或 HTTP 代理计算机，或更多 MQIPT 计算机进行通信。


Outbound connections
Intranet
Internet
IBM MQ Client
Firewall
MQIPT
SOCKS Proxy
HTTP Proxy
MQIPT
HTTP Proxy
SOCKS Proxy
MQIPT
MQIPT
running on a firewall computer
IBM MQ Client
IBM MQ Server
IBM MQ Server
Key:
HTTP
IBM MQ Protocol



兼容的配置

IBM MQ 客户机或队列管理器与 MQIPT 通信的兼容连接方案。 使用同一个或第二个 MQIPT 路由与目标队列管理器通信。

使用单个 MQIPT 路由的兼容配置

您可以使用单个 MQIPT 路由与 IBM MQ 通信。

第 266 页的表 26 中的列包含以下信息：

1. IBM MQ 和 MQIPT 路由之间使用的协议。 IBM MQ 客户机或队列管理器可以创建连接，该连接可以使用 IBM MQ 格式和协议 (FAP) 或 SSL/TLS 协议。
2. MQIPT 路由运行的方式。 MQIPT 和 IBM MQ 之间通过因特网通信的格式由 MQIPT 路由的配置确定。 请注意，表上列示 SSL 表示此情况下您还可以使用 TLS。
3. MQIPT 路由和目标队列管理器之间使用的协议。

表 *26:* 具有单个 *MQIPT* 实例的有效配置

1 IBM MQ 源协议	**2.MQIPT 路由的方式**	**3。 IBM MQ 目标协议**
FAP	FAP 代理（缺省）	FAP
	FAP 服务器和 SSL 客户机	SSL/TLS
SSL/TLS	SSL 代理	SSL/TLS
	SSL 服务器和 FAP 客户机	FAP
	SSL 服务器和 SSL 客户机	SSL/TLS

使用多个 MQIPT 路由的兼容配置

您可能在一个或多个 MQIPT 实例上选择使用多个路由，以与 IBM MQ 通信。

第 266 页的表 27 中的列包含以下信息：

1. IBM MQ 和第一个 MQIPT 路由之间使用的协议。 IBM MQ 客户机或队列管理器可以创建连接，该连接可以使用 IBM MQ 格式和协议 (FAP) 或 SSL/TLS 协议。
2. 第一个 MQIPT 路由运行的方式。 MQIPT 和 IBM MQ 之间通过因特网通信的格式由 MQIPT 路由的配置确定。 请注意，表上列示 SSL 表示此情况下您还可以使用 TLS。
3. 第二个 MQIPT 路由运行的方式。
4. 第二个 MQIPT 路由和目标队列管理器之间使用的协议。

表 *27:* 具有多个 *MQIPT* 实例的有效配置

1 IBM MQ 源协议	**2. 第一个 MQIPT 路由的方式**	**3. 第二个 MQIPT 路由的方式**	**4. IBM MQ 目标协议**
FAP（缺省）	FAP 代理（缺省）	FAP 代理（缺省）	FAP
	FAP 服务器和 SSL 客户机	SSL 代理	SSL/TLS
		SSL 服务器和 FAP 客户机	FAP
		SSL 服务器和 SSL 客户机	SSL/TLS
	HTTP 客户机	HTTP 服务器和 SSL 客户机	SSL/TLS
	HTTPS 客户机	HTTPS 服务器和 SSL 客户机	SSL/TLS
	HTTP 客户机	HTTP 服务器	FAP
	HTTPS 客户机	HTTPS 服务器	FAP

表 27: 具有多个 *MQIPT* 实例的有效配置 (继续)			
1 IBM MQ 源协议	**2. 第一个 MQIPT 路由的方式**	**3. 第二个 MQIPT 路由的方式**	**4. IBM MQ 目标协议**
SSL/TLS	SSL 代理	SSL 代理	SSL/TLS
		SSL 服务器和 FAP 客户机	FAP
		SSL 服务器和 SSL 客户机	SSL/TLS
	HTTP 客户机	HTTP 服务器	FAP
	HTTPS 客户机	HTTPS 服务器	SSL/TLS
	HTTP 客户机	HTTP 服务器和 SSL 客户机	FAP
	HTTPS 客户机	HTTPS 服务器和 SSL 客户机	SSL/TLS

支持的通道配置

支持所有 IBM MQ 通道类型，但是配置限制为 TCP/IP 连接。 对于 IBM MQ 客户机或队列管理器，MQIPT 似乎是一个目标队列管理器。 通道配置需要目标主机和端口号时，指定 MQIPT 主机名和侦听器端口号。

客户机/服务器通道
MQIPT 侦听入局客户机连接请求，然后使用 HTTP 隧道， SSL/TLS 或作为标准 IBM MQ 协议包转发这些请求。 如果 MQIPT 正在使用 HTTP 隧道或 SSL/TLS ，那么它会将它们在连接上转发到第二个 MQIPT。 如果不使用 HTTP 隧道传送，那么将在连接上将其转发到它视为目标队列管理器的目标，（尽管这可能还是进一步 MQIPT）。 当目标队列管理器接受客户机连接时，将在客户机与服务器之间传送包。

集群发送方/接收方通道
如果 MQIPT 接收来自集群发送方通道的入站请求，它将假定队列管理器支持 SOCKS，SOCKS 握手过程期间将获取真正的目标地址。 它使用与客户机连接通道完全相同的方式，将请求转发到下一个 MQIPT 或目标队列管理器。 这还包括自动定义的集群发送方通道。

发送方/接收方
如果 MQIPT 接收来自发送方通道的入站请求，它将使用与客户机连接通道完全相同的方式，将请求转发到下一个 MQIPT 或目标队列管理器。 目标队列管理器验证入站请求并且在适当的情况下启动接收方通道。 发送发和接收方通道之间的所有通信（包括安全流）将被传送。

请求方/服务器
本组合的处理方式与前面的配置相同。 服务器通道在目标队列管理器执行连接请求的验证。

请求方/发送方
如果两个队列管理器之间不允许建立直接连接，但是允许连接到 MQIPT 并且接受它的连接，那么可以使用“回调”配置。

服务器/请求方和服务器/接收方
这些由 MQIPT 以处理 `Sender/Receiver` 配置的相同方式进行处理。

通道终止和故障条件

当 MQIPT 检测到 IBM MQ 通道关闭时（正常或异常），会传播通道关闭。 如果使用 MQIPT 关闭路由，那么经过此路由的所有通道都会关闭。

MQIPT 提供了可选空闲超时工具。 如果 MQIPT 检测到通道空闲时段超过超时，会在涉及的两个连接上立即执行关闭。

通道任何一端的 IBM MQ 系统会将这些异常关闭条件视为网络故障或者视为其合作伙伴终止了通道。 然后此通道能够重新启动并恢复（如果故障发生在协议不确定时段），好像未使用 MQIPT 一样。

消息的安全性

IBM MQ 分布式队列管理可确保正确传递消息。 当通道两端之间存在 MQIPT 时，仍然会发生此情况。 MQIPT 不会存储任何消息数据，也不会参与确保正确消息传递的同步点过程。

在使用快速非持久性 IBM MQ 消息时，如果 MQIPT 路由在传输 IBM MQ 消息时发生失败或重新启动，那么消息可能会丢失。 在重新启动路由之前，请确保所有使用 MQIPT 路由的 IBM MQ 通道都处于不活动状态。

有关 IBM MQ 中消息安全的更多信息，请参阅 消息安全。

多实例队列管理器和高可用性

MQIPT 可用于高可用性环境中的多实例队列管理器。

MQIPT 没有持久状态，故障转移 MQIPT 至其他系统没有任何优势。 而是在不同系统上运行多个具有相同 `mqipt.conf` 配置文件的 MQIPT 实例。 监控每一个 MQIPT 实例的可用性，并在必要时重新启动（在同一系统上）。 这样提供一组相同的 MQIPT 实例，可将其用于路由连接。 然后，您必须确保 IBM MQ 可以将连接路由到 MQIPT，并且该 MQIPT 可以将那些连接转发到目标队列管理器。

可以使用各种方式将出站 IBM MQ 通道定向到可用的 MQIPT 实例，例如：

- 使用 WebSphere Edge Components 产品中的负载均衡器或高可用性路由器，例如 IBM Network Dispatcher 。
- 使用逗号分隔列表在 IBM MQ 通道定义中指定多个连接名称。 然后，IBM MQ 尝试依次连接每个 MQIPT 地址，直到找到可用的 MQIPT 实例。

您还必须将连接从 MQIPT 定向到目标队列管理器。 如果高可用性配置确保 IP 地址使用目标队列管理器进行故障转移，那么不需要特别的 MQIPT 配置：在 **Destination** 路由属性中指定目标 IP 地址，允许故障转移操作移动队列管理器的 IP 地址。

但是，如果故障转移后，队列管理器的 IP 地址发生更改，那么您必须安排 MQIPT 将连接转发到正确的目标。 可以通过几种方式完成此任务：

- 写入一个路由出口来检查哪个 IP 地址和端口号是可以访问的，然后覆盖每个连接的路由目标。 MQIPT 随附了一些路由出口样本；可以对其进行改写以用于此目的。
- 使用高可用性负载均衡器重定向连接。
- 定义多个 MQIPT 路由，针对可能在运行队列管理器的每一个 IP 地址和端口定义一个路由。 然后，将 IBM MQ 连接定向到各种 MQIPT 路由，例如，在出站通道的连接名称中，以逗号分隔列表形式列出所有路由 IP 地址和端口号。

网络路径上所有端到端组件的调整也很重要：

1. 不可用系统的连接尝试必须及时断开，这样重新连接尝试将移动到第一个可用目标。

   对于 MQIPT SSL 路由，调整 **SSLClientConnectTimeout** 路由属性以确保不可用目标的及时连接断开。 有关 IBM MQ 调整参数的详细信息，请参阅 IBM MQ 文档。 此外，查看操作系统文档，以获取操作系统 TCP/IP 调整的详细信息。 在任何情况下，失败的连接尝试应快速返回网络故障（例如，TCP 复位分组），或者应该超时而不必过分延迟。
2. 故障系统的活动连接必须及时断开，这样可以建立新连接。

   您还应该考虑连接主动使用 MQIPT 时故障转移的影响。 网络连接很可能在故障转移期间断开。 对于客户机应用程序，您可以使用 IBM MQ 自动客户机重新连接功能重新建立已断开的连接。 对于消息通道，您可以指定较短的重试时间间隔，以便通道及时重新连接。 查看 IBM MQ 文档以获取有关自动客户机重新连接和消息通道重试配置的更多信息。

IBM MQ Console 和 REST API

您可以使用 IBM MQ Console 和 REST API 来管理 IBM MQ，并使用 HTTP 来执行消息传递操作。

- 您可以使用 IBM MQ Console 从 Web 浏览器执行基本管理任务。 有关更多信息，请参阅 使用 IBM MQ Console 进行管理。
- 您可以使用 administrative REST API 来管理 IBM MQ 对象，例如队列管理器和队列以及 Managed File Transfer 代理和传输。 有关更多信息，请参阅 使用 REST API 进行管理。

- 您可以使用 messaging REST API 来执行简单的点到点和发布消息传递。 有关更多信息，请参阅 使用 REST API 进行消息传递。

安装选项

IBM MQ Console 和 REST API 在名为 mqweb 的 WebSphere Liberty 服务器中运行。 从 IBM MQ 9.3.5 开始，您可以将 mqweb 服务器作为 IBM MQ 安装中的可选组件进行安装，也可以作为独立 IBM MQ Web Server 安装进行安装。

Linux V 9.4.0 独立安装的 IBM MQ Web Server

从 IBM MQ 9.4.0 开始， mqweb 服务器可以在 IBM MQ Web Server 的独立安装中运行。 独立 IBM MQ Web Server 安装使您能够在独立于 IBM MQ 安装的系统上安装和运行 mqweb 服务器。 通过安装独立 IBM MQ Web Server ，您可以更灵活地选择在哪些系统上运行 mqweb 服务器，以及选择在哪些系统上运行 mqweb 服务器的系统数量。 如果需要，可以在不同机器上运行 mqweb 服务器的多个实例，以提供所需的可伸缩性和可用性。

如果您已购买 IBM MQ 权利，那么可以安装所需数量的独立 IBM MQ Web Server 副本。 IBM MQ Web Server 安装不会计入已购买的 IBM MQ 权利。 有关 IBM MQ 许可的更多信息，请参阅 IBM MQ 许可证信息。

以下限制适用于独立 IBM MQ Web Server 安装:

- IBM MQ Console 只能用于管理远程队列管理器。
- messaging REST API 只能与远程队列管理器配合使用。
- administrative REST API 不可用。

仅在 Linux 平台上支持独立 IBM MQ Web Server 。

有关安装独立 IBM MQ Web Server 的更多信息，请参阅 安装独立 IBM MQ Web Server。

IBM MQ 安装的可选组件

您可以选择在 IBM MQ 安装过程中安装 IBM MQ Console 和 REST API 组件。

当 mqweb 服务器在 IBM MQ 安装中运行时，所有 IBM MQ Console 和 REST API 功能部件都可用。

- IBM MQ Console 可用于管理本地和远程队列管理器。
- messaging REST API 可与本地和远程队列管理器配合使用。
- administrative REST API 可用于管理本地和远程队列管理器。

要使用 IBM MQ Console 和 REST API 组件，请在 IBM MQ 安装过程中安装以下组件:

- AIX 在 AIX 上，安装 `mqm.web.rte` 文件集。
- IBM i 在 IBM i 上，安装 WEB 组件。
- Linux 在 Linux 上，安装 `MQSeriesWeb` 组件。
- Windows 在 Windows 上，安装 `Web Administration` 功能部件。
- z/OS 在 z/OS 上，安装 `IBM MQ for z/OS UNIX System Services Web Components` 功能部件。

声明

本信息是为在美国国内供应的产品和服务而编写的。

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暗示只能使用 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 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。 您可以用书面方式将许可查询寄往：

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：

Intellectual Property Licensing
Legal and Intellectual Property Law
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 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。 那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：（i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系：

IBM Corporation
软件互操作性协调员，部门 49XA
北纬 3605 号公路
罗切斯特，明尼苏达州 55901
U.S.A.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。 因此，在其他操作环境中获得的数据可能会有明显的不同。 有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。 此外，有些测量是通过推算而估计的， 实际结果可能会有差异。 本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。 IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。 有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息包含日常商业运作所使用的数据和报表的示例。 为了尽可能全面地说明这些数据和报表，这些示例包括个人、公司、品牌和产品的名称。 所有这些名字都是虚构的，若现实生活中实际业务企业使用的名字和地址与此相似，纯属巧合。

版权许可：

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。 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 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 The 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 (https://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: