Websphere MQ 入门教程——

使用 IBM Websphere MQ

提纲

目录

目录.....2
前言.....9
本书范围.....9
本书读者.....9
进一步参考资料.....10
第一部分 Websphere MQ 原理和体系结构.....11
第一章 Websphere MQ 原理.....11
目标.....11
1.1 中间件.....11
1.1.1 中间件的优点.....11
1.1.2 中间件的分类.....12
1.2 三种通信技术的比较.....13
1.3 WebSphere MQ 的原理.....15
1.4 WebSphere MQ 的重要特点.....16
1.4.1 统一接口.....16
1.4.2 处理不依赖时间的限制.....16
1.4.3 给分布式处理提供的强健的中间件.....16
1.5 本章小节.....17
1.6 本章练习.....17
第二章 Websphere MQ 体系结构.....18
目标.....18
2.1 基本概念.....18
2.1.1 WebSphere MQ 对象(objects).....18
2.1.2 消息.....19
2.1.3 队列.....20
2.1.4 队列管理器.....24
2.1.4 通道.....25
2.1.5 进程.....29
2.1.6 群集.....29
2.1.7 名称列表.....29
2.1.8 认证信息对象.....30
2.1.9 系统缺省对象.....30
2.1.10 MQI（message queue interface）.....30
2.2 体系结构.....30
2.2.1 WebSphere MQ 和消息排队.....31
2.2.2 队列管理器的进程.....32
2.3 客户机和服务器.....33
客户机－服务器环境中的 WebSphere MQ 应用程序.....33
2.4 触发机制.....33
2.4.1 触发的概念.....33

2.4.2 触发类型..... 34
2.4.3 触发的工作原理..... 35
2.5 队列管理器群集..... 36
2.5.1 群集的概念..... 36
2.5.2 群集的优点..... 37
2.5.3 群集的组件..... 38
2.5.4 创建群集..... 38
2.5.5 实现负载均衡..... 39
2.5.6 群集管理..... 40
2.6 本章小结..... 41
2.7 本章练习..... 41
第二部分 Websphere MQ 系统管理..... 43
第三章 WebSphere MQ 系统安装..... 43
目标..... 43
3.1 规划安装..... 43
3.1.1 硬件要求..... 43
3.1.2 软件要求..... 44
3.2 安装 WebSphere MQ..... 46
3.2.1 WebSphere MQ 文档..... 46
3.2.2 WebSphere MQ 安装..... 47
3.3 验证安装..... 49
3.3.1 安装验证..... 49
3.3.2 测试对象..... 49
3.4 本章小结..... 50
3.5 本章练习..... 50
第四章 WebSphere MQ 的管理..... 51
目标..... 51
4.1 本地和远程管理..... 51
4.2 使用命令管理 WebSphere MQ..... 51
4.2.1 控制命令..... 52
4.2.2WebSphere MQ 脚本（MQSC）命令..... 52
4.2.3PCF 命令..... 54
4.3 WebSphere MQ 配置..... 56
4.3.1 在 Windows 系统上更改配置信息..... 56
4.3.2 在 UNIX 系统上更改配置信息..... 57
4.4 WebSphere MQ 安全性..... 60
管理 WebSphere MQ 的权限..... 60
使用 WebSphere MQ 对象的权限..... 61
4.5 WebSphere MQ 事务性支持..... 61
4.6 WebSphere MQ 死信队列处理程序..... 62
4.7 本章小结..... 62
4.8 本章练习..... 63
第五章 WebSphere MQ 控制命令..... 64
目标..... 64

5.1 如何使用控制命令 64
WebSphere MQ 对象的名称 64
5.2 控制命令 65
控制命令集 65
控制命令举例 66
5.3 本章小结 66
5.4 本章练习 66
第六章 WebSphere MQ 互连通信 68
目标 68
6.1 基本概念 68
6.1.1 什么是互连通信 68
6.1.2 分布式队列组件 72
6.1.3 死信队列 75
6.1.4 怎样到达远程队列管理器 75
6.2 实现应用程序通信 77
6.2.1 发送消息到远程队列管理器 77
6.2.2 触发通道 79
6.2.3 消息的安全性 80
6.2.4 WebSphere MQ 对象配置实例 81
6.3 通道的维护 83
6.3.1 通道的状态 83
6.3.2 通道维护命令 84
6.3.3 设置 MaxChannels 和 MaxActiveChannels 属性 88
6.4 配置侦听程序 88
6.4.1 Windows 平台 88
6.4.2 unix 平台 88
6.5 本章小结 89
6.6 本章练习 89
第七章 WebSphere MQ 恢复和重新启动 90
目标 90
7.1 WebSphere MQ 的数据日志 91
7.1.1 日志的概念 91
7.1.2 日志控制文件 91
7.1.3 日志类型 92
7.1.4 计算日志的大小 92
7.2 使用数据日志进行恢复 93
7.2.1 从掉电或通信故障中恢复 94
7.2.2 恢复受损对象 94
7.3 保护 WebSphere MQ 日志文件 96
7.4 备份和恢复 WebSphere MQ 96
7.4.1 备份 WebSphere MQ 96
7.4.2 恢复 WebSphere MQ 96
7.5 恢复方案 97
7.5.1 磁盘故障 97

7.5.2 受损的队列管理器对象..... 98
7.5.3 受损的单个对象..... 98
7.5.4 自动媒体恢复故障..... 98
7.6 使用 dmpmqlog 命令转储日志..... 98
7.7 本章小结.....101
7.8 本章练习.....102
第八章 WebSphere MQ 问题诊断.....102
目标.....102
8.1 错误日志.....102
8.1.1 日志文件.....103
8.1.2 忽略 WebSphere MQ for Windows 的错误代码.....104
8.1.3 操作信息.....104
8.2 死信队列.....104
8.3 配置文件和问题确定.....104
8.4 跟踪.....104
8.4.1WebSphere MQ Windows 的跟踪.....104
8.4.2WebSphere MQ AIX 的跟踪.....106
8.5 首次故障支持技术（FFST）.....109
8.5.1FFST: WebSphere MQ Windows 版.....109
8.5.2FFST: WebSphere MQ UNIX 系统版.....110
8.6 本章小结.....112
8.7 本章练习.....112
第三部分 Websphere MQ 应用开发.....113
第九章 设计 Websphere MQ 应用程序.....113
目标.....113
9.1 介绍应用设计.....113
9.1.1 规划设计.....113
9.1.2 WebSphere MQ 对象.....113
9.1.3 设计消息.....114
9.1.4 WebSphere MQ 技术.....114
9.1.5 应用编程.....115
9.1.6 测试应用程序.....116
9.2 WebSphere MQ 消息.....116
9.2.1 消息描述符.....116
9.2.2 消息种类.....116
9.2.3 消息控制信息和消息数据的格式.....117
9.2.4 消息优先级.....117
9.2.5 消息组.....118
9.2.6 消息持久性.....118
9.2.7 检索消息.....119
9.2.8 交付失败的消息.....119
9.3 本章小结.....119
9.4 本章练习.....119
第十章 用 MQI 编程.....119

目标.....119
10.1 概述.....119
10.2 平台和语言.....120
10.3 库和存根模块程序.....121
10.4 体系结构模型.....122
10.5 用 MQI 编程.....124
10.5.1 基本 API 概念.....125
10.5.2 连接到队列管理器.....126
10.5.3 打开 WebSphere MQ 对象.....127
10.5.4 关闭 WebSphere MQ 对象.....130
10.5.5 断开与队列管理器的连接.....130
10.5.6 将消息放入队列.....131
10.5.7 从队列获取消息.....133
10.5.8 从队列浏览消息.....135
10.5.9 查询对象属性.....136
10.5.10 设置对象属性.....138
10.5.11 MQI 中的事务处理.....139
10.5.12 MQI 中的消息分组.....140
10.6 本章小结.....142
10.7 本章练习.....142
第十一章 用 C++ API 编程.....143
目标.....143
11.1 概述.....143
11.2 平台和语言.....144
11.3 库.....144
11.4 体系结构模型.....145
11.5 用 C++ API 编程.....146
11.5.1 连接到队列管理器.....147
11.5.2 打开 WebSphere MQ 对象.....147
11.5.3 关闭 WebSphere MQ 对象.....148
11.5.4 断开与队列管理器的连接.....148
11.5.5 消息放入队列.....148
11.5.6 从队列获取消息.....150
11.5.7 浏览队列上的消息.....153
11.5.8 查询并设置对象属性.....153
11.5.9 事务处理管理.....155
11.5.10 消息分组.....155
11.6 本章小结.....157
11.7 本章练习.....157
第十二章 用 Java 编程.....158
目标.....158
12.1 概述.....158
12.2 平台.....158
12.2.1 获得软件包.....158

12.2.2 WebSphere MQ for Java 的运行环境.....159
12.3 使用 WebSphere MQ for Java.....161
12.3.1 客户机连接模式.....161
12.3.2 绑定模式.....162
12.3.3 类库.....162
12.4 用 WebSphere MQ Java API 开展工作.....164
12.4.1 设置连接.....164
12.4.2 打开队列.....165
12.4.3 处理 WebSphere MQ 消息.....166
12.5 应用程序开发.....167
12.5.1 简单的消息发送器应用程序.....168
12.5.2 简单的消息接收应用程序.....170
12.5.3 请求/回复.....172
12.5.4 回复应用程序.....175
12.5.5 消息分组.....177
12.5.6 简单的组接收应用程序.....180
12.6 本章小结.....183
12.7 本章练习.....183
第十三章 用 ActiveX 编程.....183
目标.....183
13.1 概述.....183
13.2 平台和语言.....184
13.3 库.....185
13.4 架构模型.....185
13.5 用 WebSphere MQ automatin classes for ActiveX 编程.....186
13.5.1 连接到队列管理器.....186
13.5.2 打开 WebSphere MQ 对象.....187
13.5.3 基本操作.....189
13.5.4 关闭对象.....191
13.5.5 关闭连接.....192
13.6 事务处理管理.....192
13.7 分组.....195
13.8 本章小结.....195
13.9 本章练习.....195
第十四章 用 AMI 编程.....195
目标.....195
14.1 概述.....196
14.2 平台和语言.....198
14.3 库和包.....199
14.4 体系结构模型.....201
14.5 用 AMI 编程.....202
14.5.1 连接到队列管理器.....202
14.5.2 打开 WebSphere MQ 对象.....204
14.5.3 基本操作.....208

14.5.4 删除会话并关闭连接.....212
14.6 AMI 和 MQI 的比较.....213
14.7 事务处理管理.....213
14.8 分组.....215
14.9 本章小结.....215
14.10 本章练习.....215
附录一 WebSphere MQ 的缺省系统对象.....215

前言

今天，大多数企业都希望他们的硬件和软件提供者不只受限于一家厂商，相反，大家普遍认为应当面向多家厂商能够运行多种软件的多种硬件平台，这些硬件平台既可以是大型机，也可以是笔记本计算机。其中包括传统的中央集中式系统，通常指大型企业所采用的大型机，部门级小型计算机和个人用个人计算机或工作站。通常这些平台是在“混乱”中发展起来的，当时它们的成长既有独立性又有偶然性。

混乱造成的结果是由企业“玻璃房子”控制的清静、秩序井然的世界退化成了一个个独立而分散的部门，并要求任务能够满足其独立而分散的需求。有些公司一直在寻求一种成熟的策略，以便在企业范围内扩展应用和数据，使其距最终用户最近。这种需求在设计时，存在许多限制，因为目前的交互式主要是同步形式，它要求对方一直处于通讯状态，这必然会大大增加网络代价。

目前，许多企业都是由一些相对于整体业务问题而孤立的解决方案所组成的自动化孤岛。在信息共享的大环境下，如果能在这些孤岛之间架起桥梁，那么效率和利润都将得到提高。从我们与不同行业客户与服务提供者广泛接触的经验来看，这种沟通非常必要，而且正变得越来越重要。

有些公司已经找到了连接网络若干个部分的解决方案，它们或者是自行开发的或者只有较窄的应用领域。如 IBM 用户事务处理的 CICS 就有这样的连接功能，但数据处理软件的设计、维护和开发通常都非常昂贵。因此需要一种通用软件，它能够集成多个运行于供应商所提供的系统上的应用程序。这种软件不仅成本不高，而且可以可靠地处理很高的吞吐量，消息中间件正是解决这种互连问题的解决方案。

商业消息中间件的出现保证了消息传输的可靠性，高效率和安全性，同时也减少了系统的开发周期。目前应用最多的消息中间件产品为 IBM Websphere MQ。本文就针对 Websphere MQ 的体系结构、管理和开发进行详细的阐述，希望对读者有所帮助。

本书范围

全书共分为 3 部分共 14 章，第一部分 WebSphere MQ 原理和体系结构，分为两章；第二部分 WebSphere MQ 系统管理，分为六章，分别介绍安装、配置、管理、控制命令和问题确定；第三部分 WebSphere MQ 应用开发，由五章组成，介绍程序设计、编写和例子程序。

本书读者

本书是 WebSphere MQ 产品的实用指南，所以至少对两种读者有益，一种是 WebSphere MQ 产品的初学者，本书能成为指导性资料；另一种是 WebSphere MQ 的系统管理员和开发者。

进一步参考资料

《WebSphere MQ System Administration Guide》
《Application Messaging Interface》
《Application Programming Guide》
《Application Programming Reference》
《WebSphere MQ Clients》
《Event Monitoring》
《Intercommunication》
《Programmable Command Formats and Administration Interface》
《Queue Manager Clusters》
《Script (MQSC) Command Reference》
《Using C++》
《Using Java》

上述书籍均可到以下网址下载：
http://www-3.ibm.com/software/ts/WebSphere MQ/library/manuals/

第一部分 Websphere MQ 原理和体系结构

第一章 Websphere MQ 原理

目标

1， 了解什么是中间件，以及中间件的特点。
2， 介绍 WebSphere MQ 的原理。
3， 介绍 WebSphere MQ 的特性和优点。

1.1 中间件

中间件处于应用软件和系统软件之间，是一种以自己的复杂换取企业应用简单化的可复用的基础软件。在中间件产生以前，应用软件直接使用操作系统、网络协议和数据库等开发，开发者不得不面临许多很棘手的问题，如操作系统的多样性，繁杂的网络程序设计和管理，复杂多变的网络环境，数据分散处理带来的不一致性，性能和效率、安全问题等等。这些问题与用户的业务没有直接关系，但又必须解决，耗费了大量有限的时间和精力。于是，有人提出将应用软件所要面临的共性问题进行提炼、抽象，在操作系统之上再形成一个可复用的部分，供成千上万的应用软件重复使用。这一技术思想最终形成为了中间件产品。

从技术上讲，中间件是介于应用系统和系统软件之间的一类软件，它使用系统软件所提供的基础服务（功能），衔接网络上应用系统的各个部分或不同的应用，能够达到资源共享、功能共享的目的。

1.1.1 中间件的优点

1　应用开发：The Standish Group 分析了一百个关键应用系统中的业务逻辑程序、应用逻辑程序及基础程序所占的比例，发现了一个有趣的平均百分比，其中，业务逻辑程序、应用逻辑程序仅占总程序量的 30%，而基础程序却占了 70%!若是以新一代的中间件系列产品来组合应用，同时配合以可复用的商务对象构件，则应用开发费用可节省至 80%。

2　系统运行：没有使用中间件的应用系统，其初期投入的资金及运行费用要比同规模的使用中间件的应用系统多一倍。

3　开发周期：时间限制是所有应用系统开发项目的天敌，而基础软件的开发又是一件极耗时的工作，若使用标准商业中间件则可缩短开发周期 50-75%。

4　减少项目开发风险：The Standish Group 对项目失败的定义是：项目中途夭折、费用远远超过预算、无法准时完成项目和偏离既定的目标。研究表明，没有使用标准商业中间件的关键应用系统开发项目的失败率高于 90%。而且，企业自己开发内置的基础(中间件)软件是得不偿失的，项目总的开支至少要翻一倍，甚至会十几倍。

5　合理运用资金：借助标准的商业中间件，企业可以很容易地在现有或遗留系统之上或之外增加新的功能模块，并将它们与原有系统无缝集合。

6　应用集合：依靠标准的中间件可以将现有的应用、新的应用和购买的商务构件融合在一起。

7　系统维护：每年维护自我开发的基础(中间件)软件的开支是当初开发费用的 15%至 25%，每年应用程序的维护开支也还需要当初项目总费用的 10%至 20%。

8　质量：基于企业自我建造的基础(中间件)软件平台上的应用系统，每增加一个新的模块，就要相应地在基础(中间件)软件之上进行改进。The Standish Group 在调研过程中，曾在某个企业中的一个应用系统里，发现了有多达 1 万 7 千多个模块接口，而标准的中间件在接口方面都是清晰和规范的，可以有效地保证应用系统质量及减少新旧系统维护开支。

9　技术革新：企业对自我建造的基础(中间件)软件平台的频繁革新是不容易实现的，也是不实际的，而购买标准的商业中间件，则对技术的发展与变化可以极大地增强其适应性。

10　增加产品吸引力：不同的商业中间件提供有不同的功能模型，合理地使用，可以让用户的应用更容易增添新的表现形式与新的服务项目，从而使得企业的应用系统更完善、更出众。

1.1.2 中间件的分类

表 1　中间件的产品分类情况

种类	作用	典型产品
消息中间件	适用于任何需要进行网络通信的系统，负责建立网络通信的通道，进行数据或文件发送。消息中间件的一个重要作用是可以实现跨平台操作，为不同操作系统上的应用软件集成提供服务。	IBM webSphere MQ
交易中间件	适用于联机交易处理系统，主要功能是管理分布于不同计算机上的数据的一致性，保障系统处理能力的效率与均衡负载。交易中间件所遵循的主要标准是 x/open DTP 模型。	IBM CICS,Bea tuxedo
对象中间件	基于 corba 标准的构件框架，相当于软总线，能使不同厂家的软件交互访问，为软件用户及开发者提供一种即插即用的互操作性，就像现在使用集成块和扩展板装配计算机一样。	IBM componentbroker, iona orbix,borland visibroker

应用服务器	用来构造 internet/intranet 应用和其它分布式构件应用，是企业实施电子商务的基础设施。应用服务器一般是基于 j2ee 工业标准的。	IBM Websphere, Bea weblogic
安全中间件	以公钥基础设施（pki）为核心的、建立在一系列相关国际安全标准之上的一个开放式应用开发平台，向上为应用系统提供开发接口，向下提供统一的密码算法接口及各种 ic 卡、安全芯片等设备的驱动接口。	entrust entrust
应用集成服务器	把工作流和应用开发技术如消息及分布式构件结合在一起，使处理能方便自动地和构件、script 应用、工作流行为结合在一起，同时集成文档和电子邮件。	lss flowman ibm flowmark vitria businessagiliti

1.2 三种通信技术的比较

(1) CPI-C

CPI-C 是一个同步的对话通信模式。参加通信的某一程序发起一次对话，并控制信息的流动，如图，1.2。发起者可随后向对方发送数据，而对方可接收数据，数据也可反向流动。

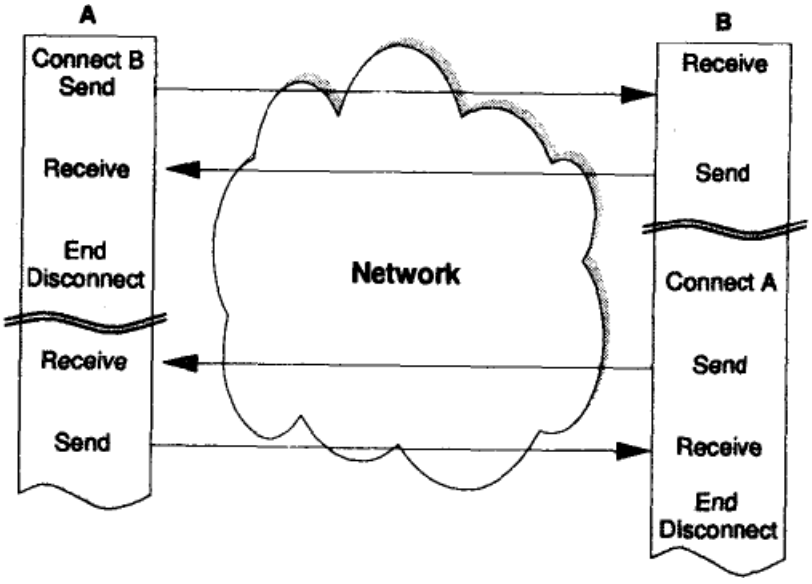


图 1.2 使用 CPI-C 基于连接的同步通信

参加通信的程序必须跟踪对话的状态，以备故障发生时作恢复连接用。在对话过程中两个程序都必须同时参加对话。如果由于某种原因造成连接断开，由连接建立者重建并恢复这次对话。这给应用程序增加了连接的负担。通信双方也可处于对等地位，在程序开始时确定了谁是对话的发起者，并保存下去，也可改变这种关系，但必须在该对话完成之后。这意味着 CPI-C 既支持客户——服务器环境也支持对等通信方式。

前面提到，CPI-C 是一种同步通信模型，但在某些事务处理环境的支持下，CPI-C

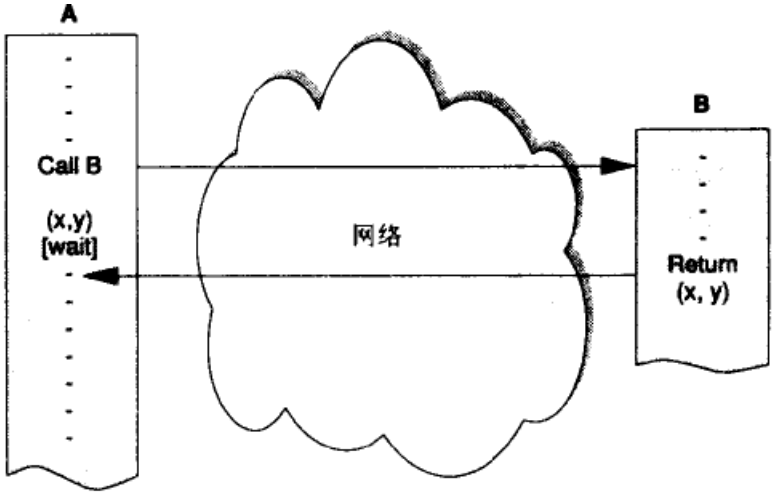
可以实现一定程度的异步，在 CICS 环境中这种支持是通过“临时数据队列”这种技术实现的。

在这以前，只有 SNA 协议支持 CPI-C，现在 TCP/IP 和 SNA 都支持 CPI-C。

由于应用程序必须参与对错误的处理和恢复，所以 CPI-C 的编程接口相当复杂。

（2）RPC

（RPC）远程过程调用也是一种同步的、对话方式的模型，如图 1.3。一个调用程序向服务器提出申请，该调用被一个负责通信的转接器发向远端系统。调用者和被调用者关系是固定的，很难实现对等通信。和 CPI-C 一样，由应用程序处理错误，并且在申请的服务得到响应之前，服务申请者被阻塞。



如图 1.3 使用 RPC，基于连接的同步通信

（3）MQI(Message Queue Interface)

如图 1.4 消息队列接口为程序提供了一种异步通信方式。一个程序以一个队列作为中转与另一个程序相互通信，这个队列相对于该程序而言既可是本地的也可以是远程的。当程序 A 需要和程序 B 通信时，A 只需 PUT 一条消息到一个和 B 相联系的队列上，程序 A 然后可以干别的事。它似乎感觉不到通信的发生，通信以及对通信错误的恢复是由队列管理完成的。

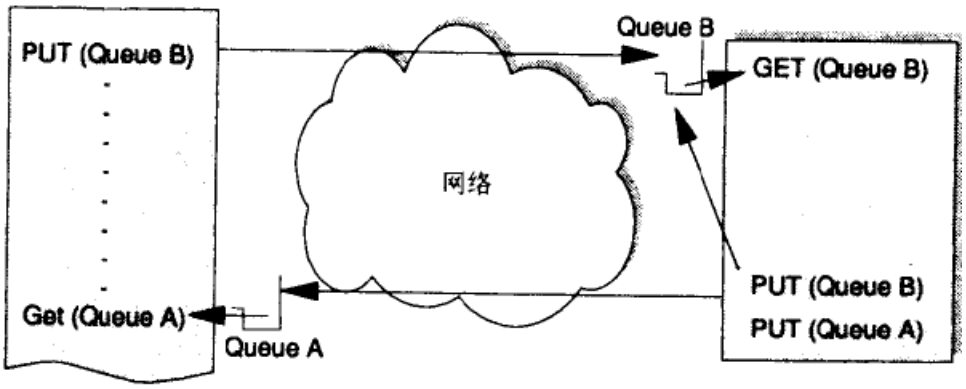


图 1.4 使用 MQI，基于队列的异步通信

通信的方式和使用的传送协议无关。因为应用程序感觉不到通信的发生，因而可以使用各种标准协议，比如 TCP/IP,SNA 或者其他局域网协议。

当程序 A 通过向某一队列 PUT 一条消息来申请程序 B 的服务时，程序 B 不一定必需在运行。而且一个程序可以通过向不同的队列 PUT 消息来实现与多个程序的通信。

最后，应该把 MQI 看作是其它通信方式所缺乏的功能的一个必要补充。每种通信方式都有其优点、缺点和适用范围。

1.3 WebSphere MQ 的原理

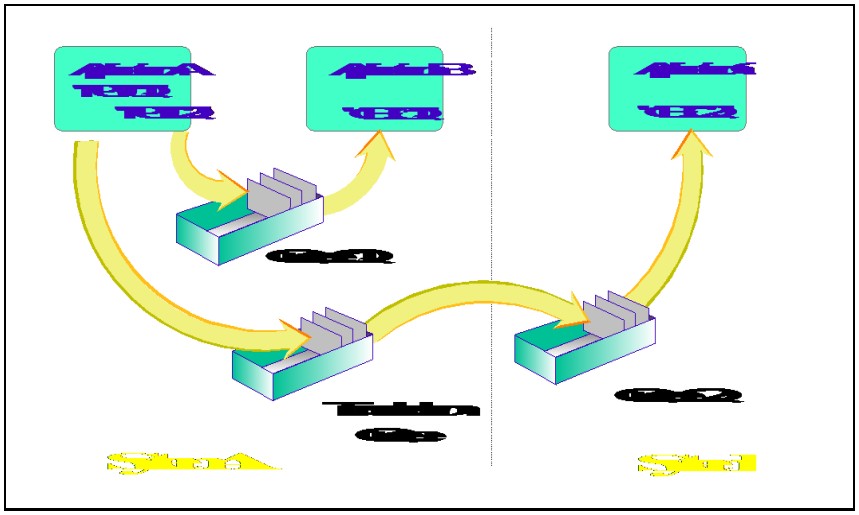
Websphere MQ 是 IBM 的商业通讯中间件(Commercial Messaging Middleware)。Websphere MQ 提供一个具有工业标准、安全、可靠的消息传输系统。它的功能是控制和管理一个集成的商业应用，使得组成这个商业应用的多个分支程序(模块)之间通过传递消息完成整个工作流程。Websphere MQ 基本由一个消息传输系统和一个应用程序接口组成，其资源是消息和队列(Messaging and Queuing)。

消息：消息就是一个信息单元，这个信息单元可以是一个请求（Request message），也可以是一个应答(Reply message)，或者是一个报告(Report message)或一份报文(Datagram messge)。一个消息包含两个因素——消息描述(用于定义诸如消息传输目标等)和数据消息(如应用程序数据或数据库查询等)。程序之间的通信通过传递消息而非直接调用程序。

队列:一个安全的存储消息的地方，消息的存储一般是顺序的，队列是消息分阶段地传送和接收。因为消息存放在队列中，所以应用程序可以相互独立的运行，以不同的速度，在不同的时间，在不同的地点。

消息传输系统: 用于确保队列之间的消息提供，包括网络中不同系统上的远程队列之间的消息提供。并保证网络故障或关闭后的恢复。

应用程序接口:应用程序和消息系统之间通过 Websphere MQ API 实现的接口 Websphere MQ API 在所有 Websphere MQ 平台上是一致的。API 只有 14 个调用，2 个关键动词：发送(PUT)和接收(GET)。



如图所示：虽然应用程序 A 和应用程序 B 运行于同一系统 A，它们不需要直接的通讯。应用程序 A 向队列 1 发送一条消息，而当应用程序 B 需要时就可以得到该消息。

如果消息传输的目标改为在系统 B 上的应用程序 C，这种变化不会对应用程序 A 产生影响，应用程序 A 向队列 Q2 发送一条消息，系统 A 的 Websphere MQ 发现 Q2 实际上在系

统 B，它将消息放到本地的一个特殊队列－传输队列(Transmission Queue)。系统 A 的 Websphere MQ 然后建立一条到系统 B 通讯联接，传递这条消息到系统 B，并等待确认。只有 Websphere MQ 接到系统 B 成功地收到消息的确认后，才从传输队列中移走消息。如果通讯线路不通，或系统 B 不在运行，消息会留在传输队列中，直到被成功地传送到目的地。这是 Websphere MQ 最基本而最重要的技术—可靠消息传输。
事实上，Websphere MQ 具有特殊的技术防止消息重复传送，确保消息一次且仅一次(once-and-only-once)传递。

1.4 WebSphere MQ 的重要特点

WebSphere MQ 提供给用户许多难得的价值。

1.4.1 统一接口

跨越 IBM 和非 IBM 平台。简单的 PUT 和 GET 动词在 WebSphere MQ 支持 35 种 IBM 和非 IBM 平台上完全相同。使得 WebSphere MQ 提供了这样的特性：目标应用程序位置的透明性(targetapplicationlocationtransparency)。对于一个应用程序的开发者，他需要知道的全部只是队列的名字，这个队列与一个特定的服务有关，而与系统的平台或系统在什么地方无关。

使开发人员避开网络的复杂性。因为 WebSphere MQ 负责处理所有的通讯，开发人员不必编写任何通讯方面的程序。并且编程和调试非常简单和直接，不需要具体的系统和通讯方面的知识。尤其在开发客户机/服务器模式的应用时，开发人员可以集中精力在与业务有关的客户端和服务器端的应用，而不必考虑操作系统和通讯，特别是底层的网络通讯，节省大约 50%到 75%of 通讯编程工作。

1.4.2 处理不依赖时间的限制

意思是说在信息创建和发送时，信息的接收方或到接收方的通道不需要激活.不受时间的限制增加了处理的灵活性，允许事务处理在它们想做或有时间做时。彼此通讯程序可以运行在不同的时间。这样程序的运行是独立的，如果逻辑允许，它们不必等待其它程序的应答而继续工作,利用这种异步处理功能，可以更有效的使用资源，更灵活的处理模式，应用处理可以是独立的，并行的，重叠的，从而改进用户服务。

1.4.3 给分布式处理提供的强健的中间件

包括逻辑工作单元支持(logicalunitofwork),备份和恢复机制，大信息传递和高性能等特点。其中最重要的是确保信息传输，意思是一旦 WebSphere MQ 接受一个信息传输的任务,会确保信息被传送到目标平台。信息的传输是一次且仅一次.另外，强健的中间件机制保证业务数据一致性，并可在系统发生故障时，及时恢复，业务不会受到影响。

1.5 本章小节

WebSphere MQ 是基于消息队列（Message Queuing）或消息传送（Message passing）的中间件，主要功能是在应用程序之间传送消息，这些消息可以在不同的网络协议、不同的计算机系统和不同的应用软件之间传递。通过使用 WebSphere MQ 用户可以简单方便的开发出可靠、高效的分布式应用系统。

总之，WebSphere MQ 的技术可实施在广泛的 IBM 和非 IBM 平台上，WebSphere MQ 提供了一个面向业务的信息技术架构:基于 WebSphere MQ 的应用程序可以更接近的模拟商业问题，更容易设计，开发和维护。这种技术使得基于 WebSphere MQ 的应用无结构限制，应用程序之间可以是一对一的关系，也可以是一对多的关系，多对多的关系。应用程序之间的信息传递可以是单向，也可以是双向的。灵活的结构支持平衡工作负荷，并行处理，多路广播以及其它应用程序之间的关系。总之是应用程序可以充分接近业务需求，并且当应用需求改变时，WebSphere MQ 的结构可以很容易的跟着改变。

1.6 本章练习

1.什么叫中间件？
2.请比较三种通信技术。
3.介绍 IBM WebSphere MQ 的原理。
4.下列那些是 IBM WebSphere MQ 的特性？
(1) 不需要 TCP/IP。
(2) 要求发送和接收程序同时运行。
(3) 当一个消息到达队列时，可以启动应用程序。
(4) 支持不同平台之间的异步处理。
(5) 是与时间相关的分布式处理。
答案：（3）（4）
5.当一个消息到达队列时自动启动了处理程序，这个特征是：
（1） 触发(triggering)
（2） 激发(firing)
（3） 信号(signaling)
（4） 自动启动(auto-start)
答案：（1）

第二章 Websphere MQ 体系结构

目标

4， 了解 WebSphere MQ 的对象。
5， 描述 WebSphere MQ 的体系结构。
6， 学习 WebSphere MQ 的客户机和服务器。
7， 理解 WebSphere MQ 的触发机制。
8， 学习使用 WebSphere MQ 的队列管理器群集。

2.1 基本概念

2.1.1 WebSphere MQ 对象(objects)

WebSphere MQ 对象是一种由 WebSphere MQ 管理的具有可恢复能力的资源。在本书中描述的许多任务都和下列对象相关：

- 队列管理器（Queue managers）
- 队列（Queues）
- 名字列表（Namelists）
- 分发列表（Distribution lists）
- 进程定义（Process definitions）
- 通道（Channels）
- 存储类（Storage classes）

这些对象在异种平台上都是统一的。对于系统管理员来说，操纵对象的命令都是可用的。这些命令格式，对于不同平台是有区别的。当你创建队列管理器时，会自动地创建缺省对象。这些缺省对象可以帮助您来定义所需的对象。

每一个对象都有一个名字，以便通过命令和 MQI 调用可以引用它。通常在这些对象类型中的每一种对象的名字必须唯一。例如，一个队列和一个进程的名字可以相同，但是不可以有两个相同名字的队列。这意味着一个本地队列名不能和模板队列、远程队列或别名队列相同。但是也会有些特殊情况。另外在互连的队列管理器网络中，队列管理器名必须唯一。

WebSphere MQ 的对象名是大小写敏感的，因此在定义对象时，需要仔细选择好大小写字母。在 WebSphere MQ 中，除最多有 20 个字符的*通道*之外，名称最多可以有 48 个字符。

2.1.2 消息

*消息*是对使用它的应用程序有意义的以字节为单位的字符串。消息可以用来实现在相同或不同平台上应用程序间的通信。

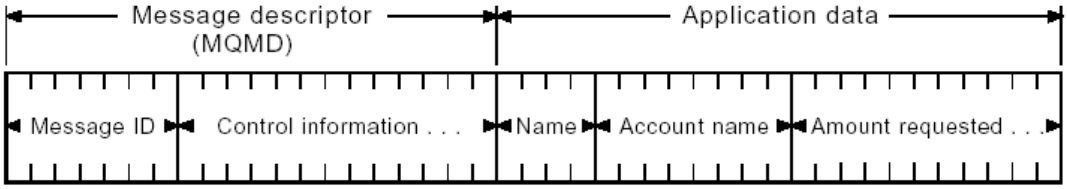
WebSphere MQ 消息由两个部分：

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，并包含其它控制信息，如消息类型和消息的优先级，如图所示：



图，消息结构

消息描述符的格式由 WebSphere MQ 定义。有关消息描述符的完整描述，参看《*WebSphere MQApplication Programming Reference*》。

2.1.2.1 消息的类型

WebSphere MQ 定义了四种基本类型的消息。应用程序可以定义其他类型的消息。四种基本类型是：

- 请求消息 Request message

  请求消息需要应答。从客户端发往服务器的查询和更新信息往往是一条请求消息。请求消息中应该包含回复消息的路由信息，即回复消息发往什么地方。

- 回复消息 Reply message

  回复消息是对请求消息的回应。请求消息中的信息决定了回应消息的目的地。处理请求和回应的应用程序控制着消息间的关联，这种关联和队列管理器没有关系。消息自身带有足够的信息供应用程序实现这种关联。

- 报文消息 Datagram message

  数据报消息是不需要回复的消息，报文消息只是一次单向的信息传送。

- 报告消息 Report message。

  报告消息用于对一些系统故障的响应。有些报告消息是由应用程序创建的，有些报告消息是由队列管理器创建的。后一种情况是由于远程队列已经满或者远程队列不存在引起消息不能正确发送。最初发送者条消息的应用程序不能检测到这种错误，只有等远程队列管理器创建了这样一条报告消息并发往本地队列管理器之后，应用程序才能作相应的处理。

  队列管理器把报告消息也用于其他目的，比如报告一些事件。消息可能有一个失效时间限制。如果一条消息在失效时间过后还没有被某个应用程序处理，该条消息将被队列管理器从系统中清除。当队列管理器清除一条失效消息之后，它将创建一条报告消息，

这条报告消息的目的地址由失效消息提供。

报告消息的另一个用途是确保消息的到达。应用程序可以要求它们所发送的消息到达目的地后，他们收到一条报告消息，这叫做接收确认（Confirmation of arrival）。与此相类似，应用程序也可以要求当另外一个程序取走这条消息时它们收到一条报告消息，这被叫做交付确认（Confirmation of delivery）。这两种情况，都是由队列管理器创建报告消息，并把报告消息发送到适当地目的地。

另外还一类特殊的消息叫触发消息。触发消息是由队列管理器创建的一类特殊消息。WebSphere MQ 的队列管理器提供了一种当满足某一条件时，自动触发应用程序的机制，而触发消息是触发机制的重要组成部分。

应用程序也可以定义新的消息类型。队列管理器不能解释这些类型，应用程序设置的消息类型由一个范围。这些类型值可用来区分不同类型的应用程序在同一个输入队列中放入的消息。

2.1.2.2 消息长度

最大消息长度为 100 MB（其中 1 MB 等于 1 048 576 字节），缺省最大消息长度是 4 MB。实际上，消息长度受以下方面的影响：

- 接收队列定义的最大消息长度
- 队列管理器定义的最大消息长度
- 传输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
- 存储消息的可用空间

所以有时可能需要由多个消息组成的信息才能满足应用程序的要求。

2.1.2.3 应用程序如何发送和接收消息？

应用程序使用 MQI **调用来实现**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** 调用打开所需的队列
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，从同一队列取出消息

2.1.3 队列

*队列*是用于存储消息的数据结构,目前 WebSphere MQ 版本 5.3 支持超过 2 GB 大小的队列。

2.1.3.1 队列的类型

按创建方法分类

- **预定义队列**由管理员使用相应的 MQSC 或 PCF 命令创建。 **预定义**队列是永久的；它们的存在与应用程序是否实用它们无关，并且 WebSphere MQ 重新启动后继续存在。
- **动态队列**在应用程序发出设定*模型队列*名的 **MQOPEN** 调用时创建的。被创建的队列是基于一个*模板队列*。 您可以使用 MQSC 命令 DEFINE QMODEL 创建模板队列。动态队列继承了模板队列的属性。模板队列有一个属性可以说明动态队列是永久的还是临时的。永久队列在应用程序和队列管理器重新启动后继续存在；临时队列在重新启动后消失。

按功能分类

1. 本地队列（local queue）：

一个本地队列是一个物理上位于本地队列管理器中的队列。本地队列实际上存在与本地系统的内存或磁盘存储终。本地队列管理器控制队列的访问。

应用程序可以“PUT”消息到本地队列，也可以从本地队列“GET”消息，另外程序还可以查询或修改这些队列的某些属性。对队列属性的修改需要相应的权限。

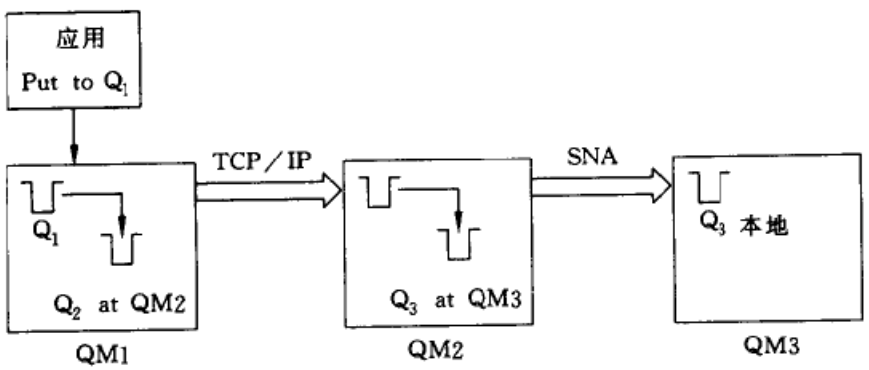
2. 远程队列（remote queue）：

一个远程队列属于一个不与该应用程序直接相连的队列管理器。对这类队列的访问包含有本地队列管理器和远程队列管理器的通信过程。这种通信涉及到通道。

应用程序可对远程队列进行某些操作，比如程序可以向一个远程队列放一条消息，但程序不能从远程队列中去消息。应用程序只能从本地队列读取消息。

应用程序有两种不同的方法可用来访问远程队列。第一种是当程序打开一个远程队列时同时提供队列管理器名和队列名两个参数。这要求程序知道目的队列属于哪个队列管理器。第二种方法是在本地队列管理器上存在一个远程队列的定义，这个定义包含有足够的信息让本地队列管理器确定该远程队列所在的队列管理器。

远程队列定义中的目的队列不一定是远程队列管理器的本地队列，它也可以是一个远程队列定义，如图所示。



应用程序放一条消息到 Q1，Q1 是本地队列管理器 QM1 上的一个远程队列定义：Q2 at QM2。QM2 是远程队列管理器。Q2 是 QM2 上的一个远程队列定义 Q3 at QM3。Q3 是 QM3 的一个本地队列，经过两次传送，消息最终到达 Q3 这个 QM3 的本地队列。

有多种原因使这种多跳（Multihop）传送变得有意义。在一个 TCP/IP 网络内部，所有机器都有 IP 地址，IP 协议本身处理节点间的路由选择。但假设消息需要穿过不同类型的网络，这就需要中间件参加路由选择。在图中，QM2 位于一台连接 TCP/IP 网和 SNA 网的机器上，只需在 QM2 上提供一个远程队列定义 Q2：Q3 at QM3，就可以实现消息的跨网络传输。

因为对远程队列的访问总是涉及到队列管理器之间的通信，因而我们需要定义其他一些资源，比如通道、传输队列（Transmission queue）。

3. 传输队列(Transmission queue)：

传输队列是临时存储目标为远程队列管理器的消息的队列。队列管理器利用传输队列把消息分阶段地发向远程队列。队列管理器和消息移动程序一起负责把数据传送到远程队列。当队列管理器收到把一条消息发往远程队列的要求后，它把消息发送到一个与目的队列管理器相关联的传输队列，传输队列位于本地队列管理器上。目的队列管理器的名称可能由应用程序提供，也可以从远程队列定义中得到。

一个传输队列是两个队列管理器之间的连接的一端。所有直接目的地是同一队列管理器的消息都可放在同一个传输队列上，这些消息的最终目的可能不一样。把消息从一个队列管理器传送到另一个队列管理器只需要一个传输队列，然而也有可能在两个队列管理器之间存在着多个连接以提供不同的传输服务，每个连接都带有一个不同的传输队列。

传输队列是由 MCA 处理的，MCA 负责在队列管理器之间可靠地传送消息。MCA 实际上是处理传输队列上消息的 MQI 应用程序。

4. 动态队列和模板队列：

除了有固定定义的队列之外，WebSphere MQ 还为程序在它们执行时提供了动态地创建队列的能力。例如,一个应用程序作为某种服务的客户，它可能创建一个动态队列，并通知服务器把对服务要求的响应发送到该动态队列。当然，这种情况也可以使用具有永久定义的队列。为了简化在创建动态队列时所必需设置的许多参数，动态队列总是基于模板队列被创建的，模板队列定义了动态队列的所有属性。当应用程序试图打开一个模板队列时，WebSphere MQ 就创建一个动态队列。WebSphere MQ 为应用提供了系统模板队列。

动态队列也可以分成两种，它们的生存周期和故障恢复特性不同。在创建临时动态队列（Temorary dynamic queue）的应用程序关闭时，这些队列将被删除，队列上的消息将丢失。这类动态队列不支持消息的持久性。如果队列管理器发生故障重新启动，临时动态队列也不会被恢复。另一种动态队列是持久动态队列（Permanent dynamic queue）。只有当一个应用程序关闭持久动态队列时定义删除选项，持久动态队列才会被删除。删除持久动态队列的程序不一定是创建持久动态队列的程序，持久动态队列在队列管理器重启后会被恢复，并且支持具有持久性的消息。

5. 启动队列

启动队列是在触发中使用的队列。如果队列管理器将使用触发，则必须至少为此队

列管理器定义一个启动队列。队列管理器在触发器事件发生时将触发器消息放入启动队列。触发器事件是由队列管理器检测的条件的逻辑组合。例如，当队列上的消息数达到预定义深度时，可能会生成触发器事件。此事件使队列管理器将触发器消息放入指定的启动队列。此触发器消息由*触发器监视器*（即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在触发器消息中指定的应用程序。

6. 群集传输队列

每个在群集中的队列管理器有一个称为 SYSTEM.CLUSTER.TRANSMIT.QUEUE 的群集传输队列。当您定义队列管理器时，按缺省情况创建此队列的定义。 作为群集一部分的队列管理器可以将群集传输队列上的消息发送到在同一群集中的任何其它队列管理器。 在路由解析期间，群集传输队列优先于缺省传输队列。 当队列管理器是群集的一部分时，缺省操作是使用 SYSTEM.CLUSTER.TRANSMIT.QUEUE，除非当目标队列管理器不是此群集的一部分。

7. 死信队列 （Dead letter queue）

死信（未传递的消息）队列是存储无法发送到其正确目的地的消息的队列。有时候会出现队列管理器不能把消息发送到目的地的情况，此时消息将被发送到某个死信队列中。死信队列中的消息常常暗示了系统可能出现的问题。例如当一条消息到达目的队列管理器之后却发现目的队列并不存在。或者目的队列出现不能接收信消息的情况，比如目的队列已经满了，或者它被设置成不允许再加入新的消息。并不是所有的放消息操作的失败都导致消息被放入死信队列，例如，由于本地队列出现错误造成应用程序不能“放”消息，此时 MQI 调用直接把错误码返回给应用程序。

有些错误只能由死信队列报告，例如，一条消息穿越网络之后到达目的队列管理器，却发现目的队列已满。发现错误的机器不同于最初“放”消息应用程序所在的机器，甚至可能放消息的应用程序已不在运行状态。此时目的队列管理器把这条消息发往它所拥有的死信队列，而不是简单地扔掉该条消息。这样使得这次错误是可见的，也给应用程序提供了一个改正错误的机会。

死信队列是 WebSphere MQ 面对远端系统错误时的一种解决方案。应用程序可以利用 WebSphere MQ 提供的其他一些工具来监视并确保消息的可靠传送和接收。

在队列管理器创建时，系统会缺省创建一个死信队列，队列名是 SYSTEM.DEAD.LETTER.QUEUE。 建议在生产系统上，需要独立创建一个死信队列，而不使用系统缺省的死信队列。

8. 命令队列

命令队列 SYSTEM.ADMIN.COMMAND.QUEUE 是用来存放由应用管理程序放的具有 PCF（program command format）的消息的队列。该队列主要用于编写管理程序时使用。具体的使用将在后续章节介绍。在创建队列管理器时将为每个队列管理器自动创建命令队列。

9. 回复队列

当应用程序发送请求消息时，接收消息的应用程序可以将回复消息发送给发送应用程序。此回复消息放入一个称为回复队列的队列中，它通常是发送应用程序的本地队列。回复队列的名称由作为消息描述符一部分的发送应用程序指定。

10. 别名队列

别名队列实际上是本地队列、远程队列定义或队列名表的另外一个名字。它是一种简单的名字到名字的映射，它允许应用程序用另外一个名字来访问队列。WebSphere MQ 已经为

应用程序屏蔽了许多底层系统细节，特别是网络通信的细节，而别名队列允许在不修改应用程序的条件下访问其他名字的队列。

2.1.3.2 定义队列

在 WebSphere MQ 中可以使用以下方法定义队列：

- 在 MQSC 命令行中使用 DEFINE
- 在程序中使用 PCF 创建队列命令

在定义队列时需要指定队列的类型和属性。例如，可以设置队列可存放的消息最大长度，以及队列的最大深度等属性。有关定义队列对象的进一步详细信息，请参阅*《WebSphere MQ 脚本（MQSC）命令参考》*或*《WebSphere MQ Programmable Command Formats and Administration Interface》*。

2.1.3.3 从队列检索消息

适当授权的应用程序可以根据以下检索算法从队列检索消息：

- 先进先出（FIFO）。
- 在消息描述符中定义的消息优先级。以 FIFO 为基础检索具有相同优先级的消息。
- 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。

2.1.4 队列管理器

在 WebSphere MQ 中队列管理器是基本的软件系统，队列管理器可看成是队列和其他对象的容器。WebSphere MQ 中的每一个队列都属于一个队列管理器，队列管理器是为应用程序和 WebSphere MQ 部件（一些管理工具）提供对队列管理中对象的访问。

一个队列管理器是 WebSphere MQ 中的一个基本的独立的执行单元。一台机器上可以运行一个或多个队列管理器。

任何需要访问 WebSphere MQ 提供的服务的应用程序都必须先和队列管理器相连，和应用程序相连的队列管理器对该应用程序而言就叫“本地队列管理器”（Local Queue Manager），本地队列管理器为程序提供 MQI 调用的支持。应用程序可以操作、管理本地队列管理器所拥有的各种资源，也可以向其他的队列管理器发送消息。

应用程序通过一种叫做 MQI 的编程接口向队列管理器申请服务。这些服务包括“放”一条消息到队列或从队列“取”一条消息等一些基本操作。队列管理器还使队列成为可靠的存储消息的地方，它也控制安全性管理，并提供一些特殊的队列功能，比如触发队列。

为了减少应用程序对于它所运行环境的依赖性，WebSphere MQ 的应用程序可以和一个它不知道名字的队列管理器相连，这个队列管理器就是一台机器上的缺省队列管理器。如果程序在调用 MQCONN 时，把队列管理器名参数设置为空，MQCONN 将返回与缺省的队列

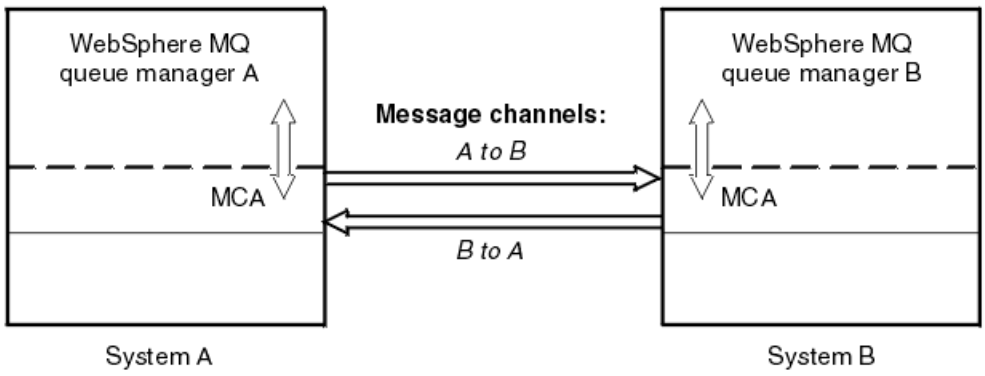
管理器连接的句柄。

*队列管理器*拥有每个队列。队列管理器负责维护它所拥有的队列，以及将它接收到的所有消息存储到相应的队列。可以由应用程序或队列管理器将消息放入队列，这些是它的正常操作的一部分。

2.1.4 通道

2.1.4.1 消息通道（Message channels）

消息通道是一种提供从一个队列管理器到另一个队列管理器的通信路径。消息通道用在分布式的队列把消息从一个队列管理器发送到另一个队列管理器。它们使应用程序屏蔽了底层的通信协议。队列管理器可能存在同种或异种平台之间。为了实现队列管理器之间的通信，您必需在一个队列管理器中定义一个发送消息的通道对象，在另一个队列管理器中定义一个接收消息的通道对象。消息通道是一个单向链接。它通过消息通道代理（message channel agents）把两个队列管理器连接起来。如图所示：



不要和 MQI 通道（MQI channel）通道混淆。MQI 通道有两种类型，分别是服务器连接（server-connection）和客户器连接（client-connection）。

消息通道分类

消息通道的定义可以分为以下 6 种类型：

- 发送通道（Sender）
- 接收通道（Receiver）
- 服务器通道（Server）
- 请求器通道（Requester）
- 群集发送通道（Cluster sender）
- 群集接收通道（Cluster receiver）

消息通道的组合形式

如果要在队列管理之间实现消息传输，必须要在两个队列管理器上都要定义相应的通道。发送方和接收方通道的组合形式如下：

- 发送通道-接收通道（Sender-receiver ）
- 请求器通道-服务器通道（Requester-server）
- 请求器通道-发送通道（Requester-sender (callback) ）
- 服务器通道-接收通道（Server-receiver ）
- 群集发送通道-群集接收通道（Cluster sender –cluster receiver）

注意：通道的组合形式有5 种形式，每种组合方式是固定的，用户不能随意组合。每对通道的名称必需相同例如：在发送通道-接收通道组合中，发送通道名和接收通道名必须一致，否则通道将无法启动。

消息通道用法

发送通道-接收通道（Sender-receiver）

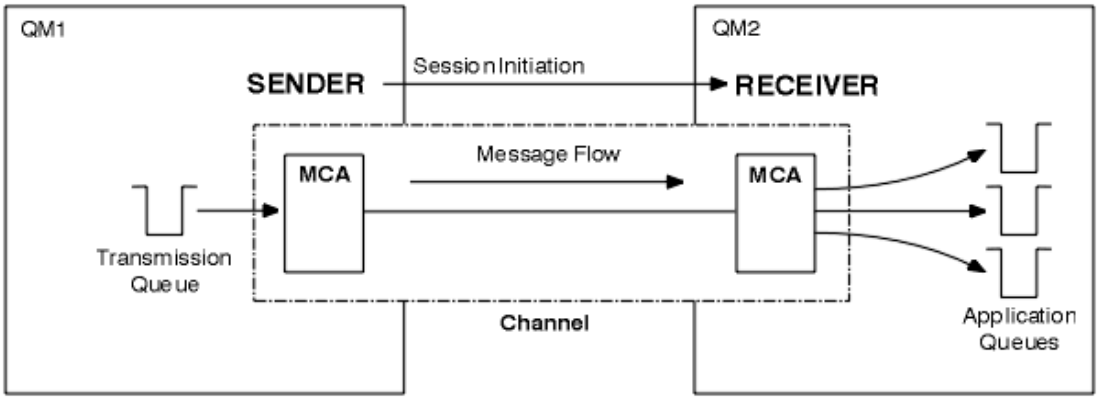
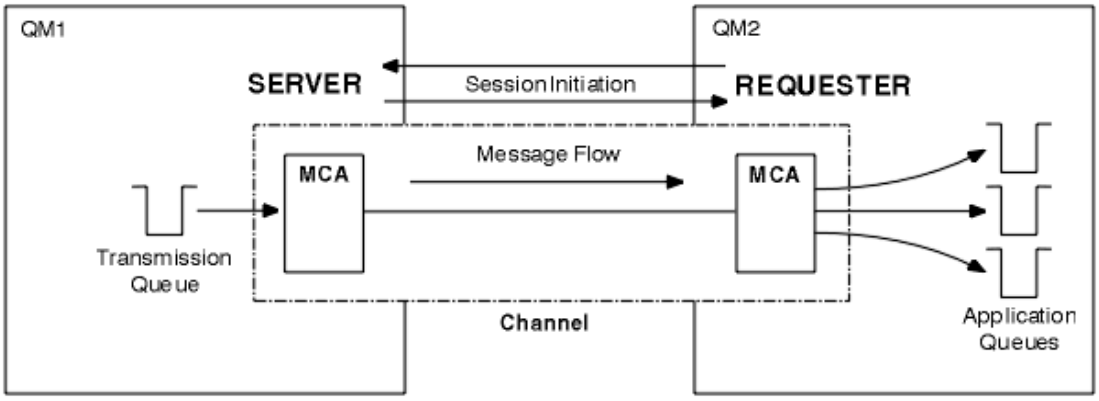


图 5 发送通道-接收通道

用法：由一个系统的发送通道来启动通道，以便它可以发送消息到另一个系统。发送通道向接收通道发送启动请求。发送通道从传输队列发送消息到接收通道。接收通道把消息放到目标队列。

请求器通道-服务器通道（Requester-server）



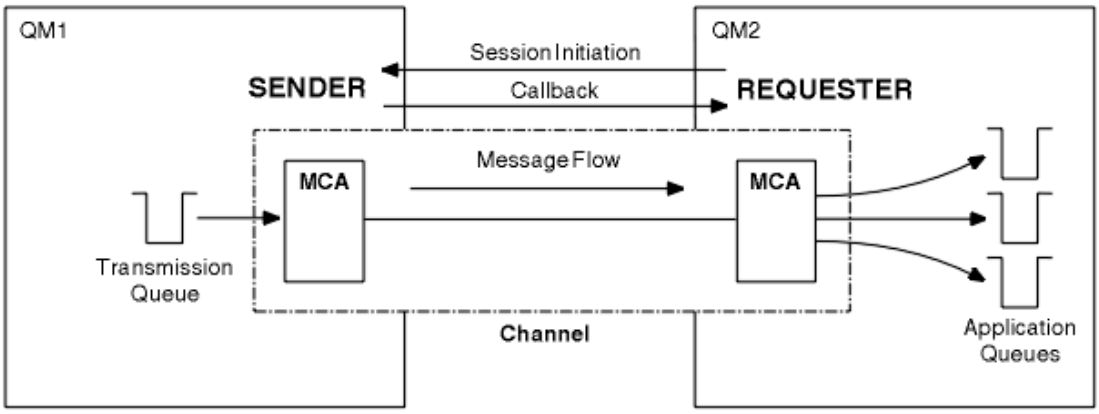
用法：

(1)由一个系统中的请求器通道来启动通道，以便它能从另一个系统接收到消息。 请求器通道向通道另一端的服务器通道发送请求来启动。服务器通道从传输队列把消息发送到请求器通道。

(2) 服务器通道也能启动通道，并发送消息到请求器，但这仅应用于完全意义的服务器通道，也即服务器通道定义中应含有对方的连接名。一个完全意义的服务器通道可以由请求器通道启动，也可以发起和服务器通道的通讯。

(3) 最好不要手工去停止 Server 和 Request 通道，而是依靠 Server 通道的 Discint 的属性来停止通道。

请求器通道-发送通道（Requester-sender (callback)）



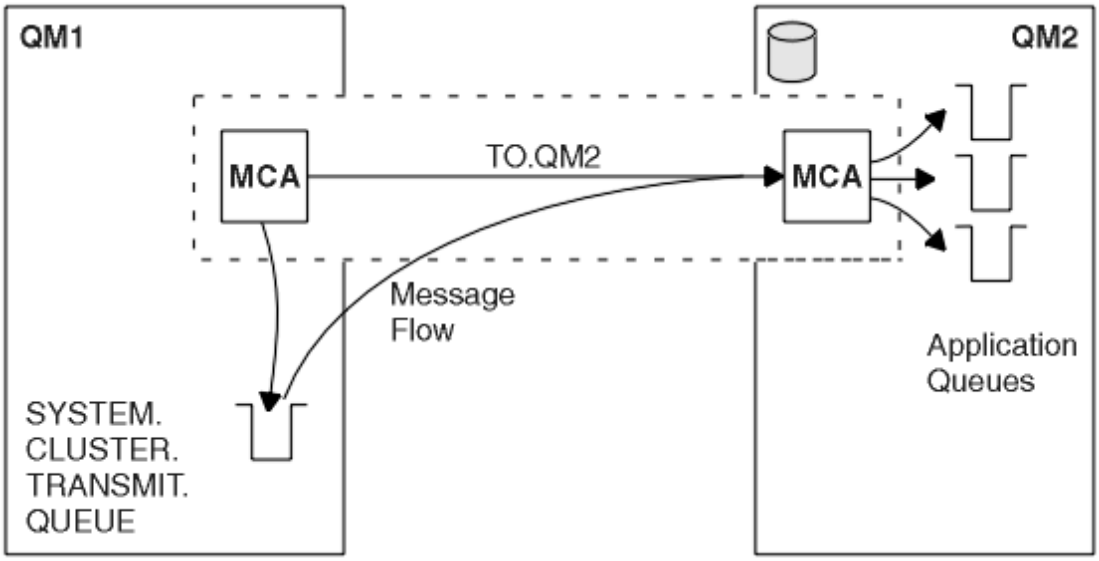
用法：请求器通道启动通道，发送通道终止这个会话。 发送通道然后依据它的通道定义中的信息(称为 callback)来重新启动会话。它把消息从传输队列发送给请求器通道。最好不要手工去停止 Sender 和 Request 通道，而是依靠 Sender 通道的 Discint 属性来停止通道。

服务器通道-接收通道（Server-receiver）

用法：类似于发送通道-接收通道，但仅应用在完全意义的服务器通道。也即服务器通道定义应含有对方的连接名，通道的启动方是服务器通道。

群集发送通道（Cluster sender）

用法：在一个群集中，每个队列管理器都有一个群集发送通道，通过它可以把送群集信息发送到其中一个队列管理器资源管理器。 队列管理器通过这个通道也可以把消息发送到其他的队列管理器。

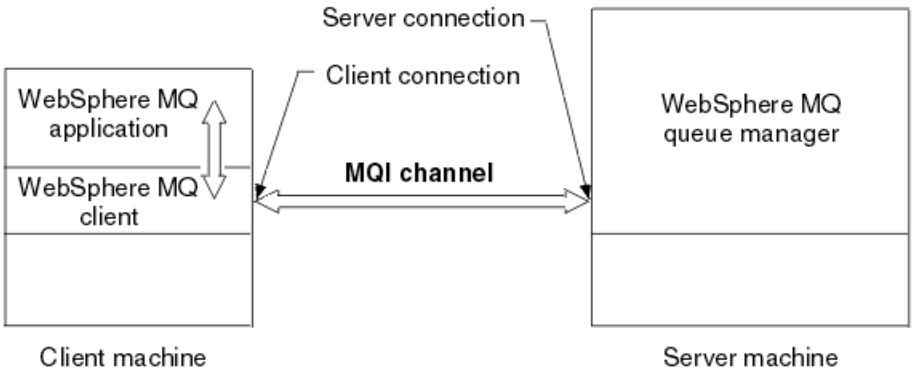


群集接收通道（cluster receiver）

功能：在一个群集中，每一个队列管理器都有一个群集接收通道。通过这个通道可以接收数据消息和关于群集的消息。

2.1.4.2 MQI 通道

MQI 通道是 WebSphere MQ 客户端和服务器上的队列管理器的通信的通道。当客户端应用程序发出 MQCONN 或 MQCONNX 调用时，才开始建立连接。它是一个双向的通道，可以负责发送和接收，被用作 MQI 调用的传送和响应。如图所示：



图，MQI 通道

一个 MQI 通道可以把一个客户端连接到单个队列管理器，MQI 通道有两种类型，它们定义了双向的 MQI 通道。

客户端连接通道（Client-connection channel）

这种类型为 WebSphere MQ 的客户端所使用。

服务器连接通道（Server-connection channel）

这种类型为运行队列管理器的服务器端所使用，运行在 WebSphere MQ 客户端的应用程序将使用这种通道进行通信。

2.1.4.3 比较消息通道和 MQI 通道

消息通道与 MQI 通道之间的区别可以从两方面进行比较：

- MQI 通道是双向的：一个 MQI 通道可以被用来发送请求，也可用来接收响应。而消息通道则只能单向数据通信。如果要在两个队列管理器之间实现双向通信，那么需要定义两个消息通道，一个用来实现数据的发送，另一个用来实现数据的接收。
- MQI 通道的通信是同步的：当一个 MQI 请求从客户端发送服务器端时，WebSphere MQ 的客户端在发送下一个请求之间必须要等待来自服务器端的响应。而消息通道，在通道中传输的消息是与时间无关的。大量的消息可以从一个队列管理器发送到另一个队列管理器，发送队列管理器不必等待来自接收队列管理器的任何响应。

2.1.5 进程

*进程定义*对象定义响应 WebSphere MQ 队列管理器上的触发器事件启动的应用程序。进程定义属性包含应用程序标识、应用程序类型和特定于应用程序的数据。

2.1.6 群集

在使用分布式排队的传统 WebSphere MQ 网络中，每个队列管理器是独立的。如果队列管理器需要将消息发送到另一个队列管理器，则它必须定义一个传输队列、一个到远程队列管理器的通道，以及它要将消息发送到的每个队列的远程队列定义。

*群集*是一组以队列管理器可以在不需要传输队列、通道和远程队列定义的情况下在单个网络上彼此直接通信的方法设置的队列管理器。

2.1.7 名称列表

名称列表是包含其它 WebSphere MQ 对象列表的 WebSphere MQ 对象。通常，应用程

序（如触发器监视器）使用名称列表，它们用于标识一组队列。使用名称列表的优点是它独立于应用程序维护；可以在不停止任何使用它的应用程序的情况下更新它。并且，如果一个应用程序失败，则名称列表不受影响，其它应用程序可以继续使用它。

名称列表还与队列管理器群集一起使用，以维护多个 WebSphere MQ 对象引用的群集列表。

2.1.8 认证信息对象

队列管理器认证信息对象（AUTHINFO）组成安全套接字层（SSL）安全性的 WebSphere MQ 支持的部件。它提供使用 LDAP 服务器检查证书撤销列表（CRL）所需的定义。CRL 允许认证中心取消不再可信的证书。

本书描述对认证信息对象使用 **setmqaut**、**dspmqaut**、**dmpmqaut**、**rcrmqobj**、**rcdmqimg** 和 **dspmqfls** 命令。有关 SSL 概述以及 AUTHINFO 的使用，请参阅 *WebSphere MQ Security*。

2.1.9 系统缺省对象

*系统缺省对象*是一组每次创建队列管理器时自动创建的对象定义。您可以复制和修改这些对象定义中的任何一个，以在安装时用于应用程序。

缺省对象名具有项 SYSTEM.DEFAULT；例如，缺省本地队列是 SYSTEM.DEFAULT.LOCAL.QUEUE，并且缺省接收方通道是 SYSTEM.DEFAULT.RECEIVER。您无法重命名这些对象；这些名称的缺省对象是必需的。当您定义对象时，从相应的缺省对象复制您不明确指定的任何属性。例如，如果您定义本地队列，则从缺省队列 SYSTEM.DEFAULT.LOCAL.QUEUE 获取您未指定的那些属性。

请参阅附录 1, "系统和缺省对象"以获取有关系统缺省的更多信息。

2.1.10 MQI（message queue interface）

MQI(消息队列接口)有下列组成部分：

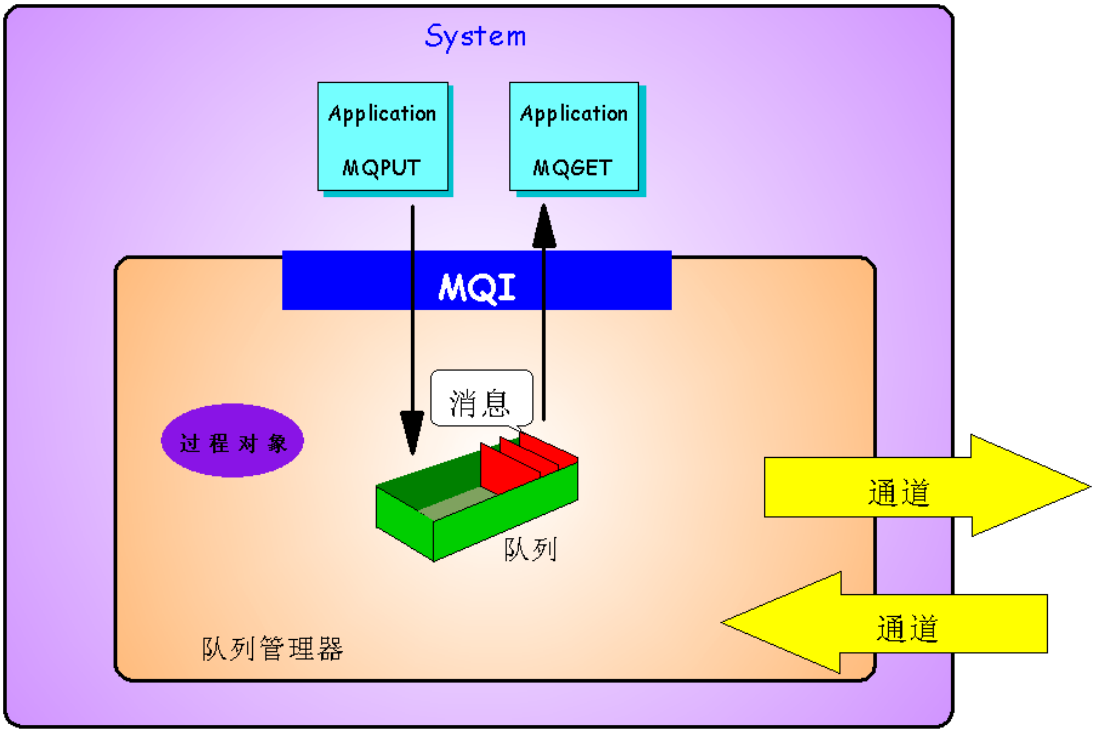
- 函数接口：应用程序通过函数可以访问队列管理器和它的部件。
- 数据结构：应用程序使用提供的数据接口来是实现把数据传递给队列管理器，或从队列管理器中获得数据。
- 基本数据类型：也是用来实现从队列管理器传递数据，或从队列管理器中获得数据。

2.2 体系结构

WebSphere MQ 的体系结构如图所示，它是由许多对象所组成的，主要包括队列管理器、队列、通道、进程定义等对象。队列管理器和 DB2 数据库中的实例相似，它是有一组 MQ 进程和一些内存空间所组成的。它实现了应用程序通过 MQI 调用可以访问 MQ 的对象。*队列管理器*为应用程序提供了排队服务，并管理属于它的队列。它确保：

- 根据接收到的命令更改对象属性。
- 当满足相应条件时，生成特殊事件（如触发器事件或检测事件）。
- 按照发送 **MQPUT** 调用的应用程序的请求，将消息放入正确的队列。如果不能完成，则将通知应用程序并给出适当的原因码。

所以在使用 WebSphere MQ 时，首先需要创建一个队列管理器，然后再队列管理器中在创建队列和通道等其他对象。



图，WebSphere MQ 的结构

2.2.1 WebSphere MQ 和消息排队

2.2.1.1 WebSphere MQ 和消息排队

WebSphere MQ 使应用程序通过使用*消息排队*来实现消息驱动处理，使用对应的消息排队软件产品实现在不同平台之间进行通信。从而使应用程序屏蔽了底层的通讯结构。

WebSphere MQ 产品提供了*消息队列接口*（或 MQI）的公共应用程序编程接口，如果应用程序使用了 MQI，将非常容易将应用程序从一个平台移植至另一个平台。

WebSphere MQ 还提供高级别消息句柄 API，它使程序员更容易在企业内或企业间部署商业流程集成的智能网络。这称为*应用程序消息传递接口*（AMI）。AMI 将许多通常由消息传递应用程序执行的消息处理功能移动到中间件层，在此将代表应用程序应用企业定义的一组策略。

如需更详细地了解 MQI 和 AMI，请参看《*WebSphere MQApplication Programming Reference* 》。

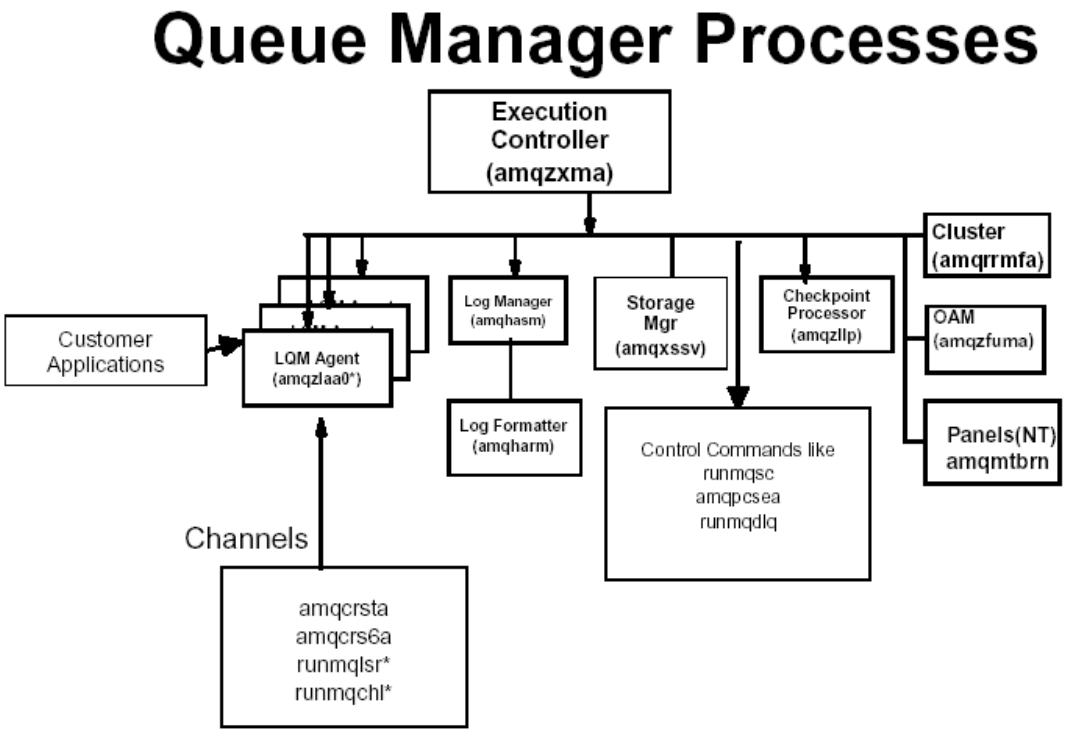
2.1.2 与时间无关的应用程序

程序不在网络上直接相互通信，而是间接地将消息放入消息队列。因为程序没有直接的联系，所以它们不必同时运行。消息放入适当的队列时目标程序可以是忙的。消息的到达并不影响程序的当前处理，也不意味程序需要立即处理该消息。消息放入队列时接收程序甚至根本不需要正在运行。接收程序可以在需要的时候开始执行。

2.1.3 消息驱动处理

当消息到达队列时满足了触发条件，它们可以使用自动*触发*应用程序。

2.2.2 队列管理器的进程



图，队列管理器进程

启动了队列管理器之后，将会启动一组进程（如上图所示），现以unix平台为例，简单介绍一些进程的功能：

- 以amqc开头的进程是和通道相关的进程。
- amqhasmx进程是负责记录日志的进程(Logger)。
- amqharmx进程是负责日志格式化。(仅在线性日志中使用)。
- amqzllp0进程是检查点处理器(Checkpoint processor)。

- amazlaa0进程本地队列管理器的代理（Local queue manager agents）。
- amqzxma0进程是执行控制器（ Execution controller）。

2.3 客户机和服务器

WebSphere MQ 支持其应用程序的客户机－服务器配置。 WebSphere MQ 客户端通过 MQI 通道与 WebSphere MQ 服务器进行通讯。

WebSphere MQ *客户机*是允许在系统上运行的应用程序对在另一个系统上运行的队列管理器发出 MQI 调用的组件。此调用的输出发送回客户机，此客户机将其输出传送回应用程序。

WebSphere MQ *服务器*是为一个或多个客户机提供排队服务的队列管理器。所有 WebSphere MQ 对象，例如，仅存在于队列管理器机器（WebSphere MQ 服务器机器）而不存在于客户机的队列。WebSphere MQ 服务器还可以支持本地 WebSphere MQ 应用程序。WebSphere MQ 服务器和普通队列管理器之间的差异是服务器与每个客户机有专用通信链路。要获取有关创建客户机和服务器的通道的更多信息，请参阅《*WebSphere MQ Intercommunication*》。

要获取有关客户机支持的信息，请参阅 *《WebSphere MQ 客户机》*。

客户机－服务器环境中的 WebSphere MQ 应用程序

当连接到服务器时，客户机 WebSphere MQ 应用程序可以与本地应用程序相同的方法发出大多数 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。然后此队列管理器处理指定从连接请求返回的连接句柄的任何其它 MQI 调用。 您必须将您的应用程序链接到相应的客户机库。请参阅 *《WebSphere MQ 客户机》* 一书，以获取进一步信息。

2.4 触发机制

2.4.1 触发的概念

队列管理器把某种条件叫做触发事件，如果队列被设置为触发类型并且触发事件发生了，队列管理器发送触发消息到一个叫做启动队列的队列中，触发消息被放置到启动队列过程意味着产生了触发事件。

由队列管理器产生的触发消息不是永久性消息，这将减少日志工作量（因此提高性能），并最小化系统重启时间。

处理启动队列中的消息叫做触发监控程序（trigger-monitor application）,他的工作是读取触发消息并根据触发消息的信息作出相应的处理。通常将启动别的应用程序来处理产生触发消息的队列。从队列管理器来看，触发监控程序没有什么特殊的，它只是一个从启动队列读取消息的应用程序。

如果队列被设置成触发形式，你可以选择创建一个进程定义（process-definition）的对象，这个对象包含了一个应用程序名，这个应用程序用来处理引起触发事件的消息。如果创建了进程对象，队列管理器将把应用程序名包含在触发消息中，以便触发监控程序可以使用。进程对象名需要在本地队列的 ProcessName 的属性中设置。每个队列可以配置一个进程对象，或几个队列可以共享同一个进程对象。

对于所有 WebSphere MQ 或 WebSphere MQ V5 的产品，如果你想触发启动通道，可以不必定义进程对象。

有些平台的 WebSphere MQ 客户端也支持触发机制，例如：UNIX，Windows,OS/2。

运行在客户端的应用程序和运行在完全 WebSphere MQ 环境的应用程序是一样的，只是应用程序链接的库有区别。同时触发监控程序和应用程序必需要都在同一系统中运行。

触发所涉及的对象：

- 应用队列：是一个本地队列，并设置为可触发的。当触发条件满足时，将会产生触发消息。
- 进程定义：一个应用队列可能由一个进程定义对象和它关联。进程定义中包含应用程序的信息。该应用程序负责从应用队列中取出消息。
- 传输队列：如果用触发方式来启动通道，则需一个传输队列。传输队列的 TriggerData 属性中设置成将被启动的通道名。这将可省略进程的定义。
- 触发事件：它是一种引起队列管理器产生触发消息的事件。
- 触发消息：当触发事件发生时，队列管理器将产生触发消息。触发信息来自于应用队列和与应用队列关联的进程定义，它包含了将要被启动的程序名。
- 启动队列：也是一个本地队列，它是被用来存放触发消息的队列。一个队列管理器可以拥有多个启动队列。一个启动队列可以为多个应用队列服务。
- 触发监控器：是一个持续运行的程序，当一个触发消息到达启动队列时，触发监控器获取触发消息，并利用触发消息中的信息，启动应用程序来处理应用队列中的消息，并把触发消息头发送传递给应用程序，消息头中包括应用队列名。

在所有平台上，有一个特殊的触发监控器叫做通道启动器（channel initator），它的作用是启动通道。

2.4.2 触发类型

- EVERY:当应用队列中每接收到一个消息时，都将产生触发消息。如果应用程序仅处理一个消息就结束时，可采用这种触发类型。
- FIRST：当应用队列中的消息从 0 变为 1 才会发生触发事件。如果当队列中的一个消息到达时启动应用程序，直到处理完所有消息就结束，则采用这种触发类型。
- DEPTH：当应用队列中的消息数目和 TriggerDepth 的属性值相同时，才会产生触发事件。当一系列请求的恢复都收到时，才启动应用程序，则采用这种触发类型。

当采用 depth 触发时，产生触发消息后，队列将被修改成非触发方式，如果需要再次触

发，将要重新设置成允许触发。

队列的 TriggerDepth 属性表示引起 depth 触发事件发生时，队列中的消息数目。

2.4.3 触发的工作原理

为了更好地能理解触发机制，我们举一个触发类型为 FIRST 的例子。

1. 只有一个本地或远程的应用程序 A，往应用队列(Application Queue)中 PUT 了一条消息。
2. 当队列原来的深度为 0 时，也即队列为空，这时 PUT 一条消息到队列中，将会形成触发事件，同时会产生一条触发消息，触发消息中将包含进程定义（Process）中的信息。
3. 队列管理器创建触发消息，并把它 PUT 到与应用队列相关的启动队列（Initiation Queue）。
4. 触发监控器从启动队列中 GET 出触发消息。
5. 触发监控器处理触发消息，发出启动应用程序 B 的命令。
6. 应用程序 B 打开应用队列，并处理队列中的消息。

注意：

1. 如果队列的触发类型为 FIRST 或 DEPTH，同时有多个应用程序往应用队列发送消息，这种情况下将不会形成触发事件。
2. 如果启动队列设置成不允许 PUT 消息，那么队列管理器将不产生任何触发消息，直到把启动队列的属性修改成允许 PUT 消息。
3. 如果通道设置成触发方式，建议触发类型为 FIRST 或 DEPTH。

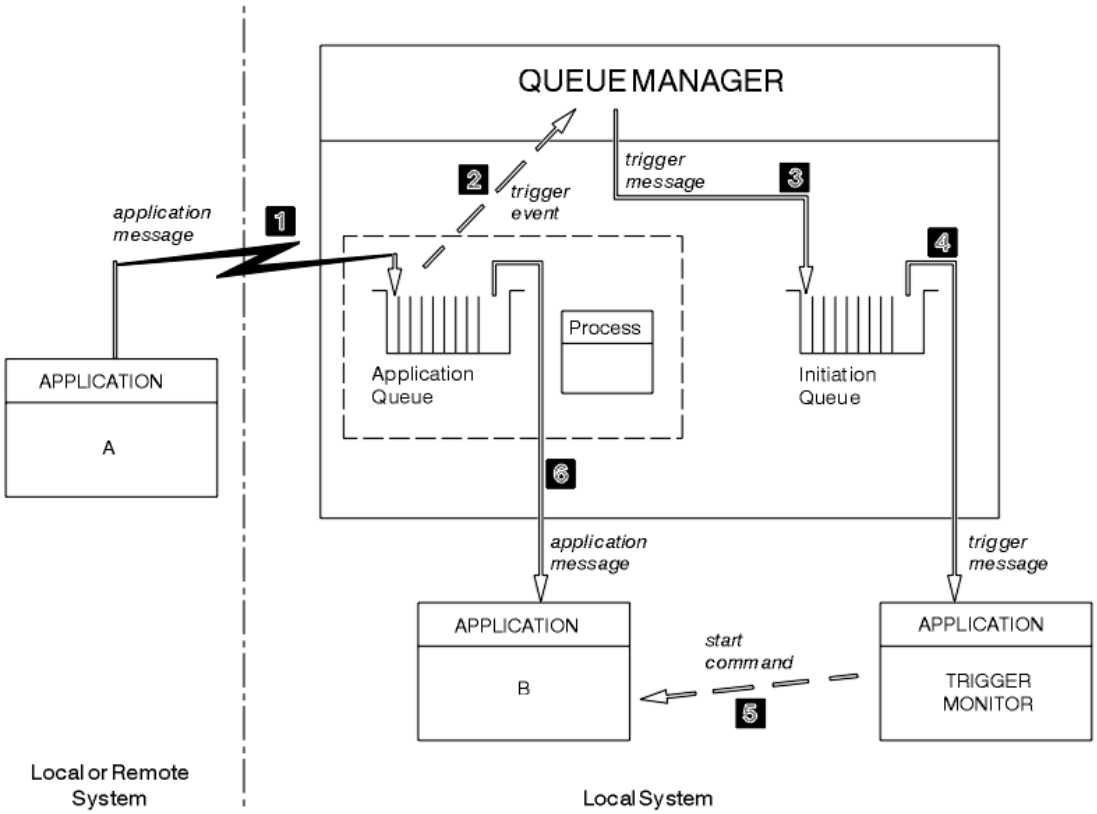


图 2.1 触发消息和应用流程

一般情况下，特别是工作负载不均匀分布时，人们总希望有消息需要处理时，才启动相应的应用程序。

这种自动启动应用程序的机制被称作“触发”（Triggering）。基本原理是：当队列管理器发现有一条消息到达被触发的队列之后，他将产生一条

2.5 队列管理器群集

2.5.1 群集的概念

如果您熟悉 WebSphere MQ 和分布式队列，则可以把群集认为是一个队列管理器的网络，或是一个队列管理器的集合，群集中的队列管理器可以是不同的操作系统平台。每当您在群集中创建一个接收通道或定义一个队列时，则系统将自动在其它队列管理器中创建相应的发送通道和远程队列定义。

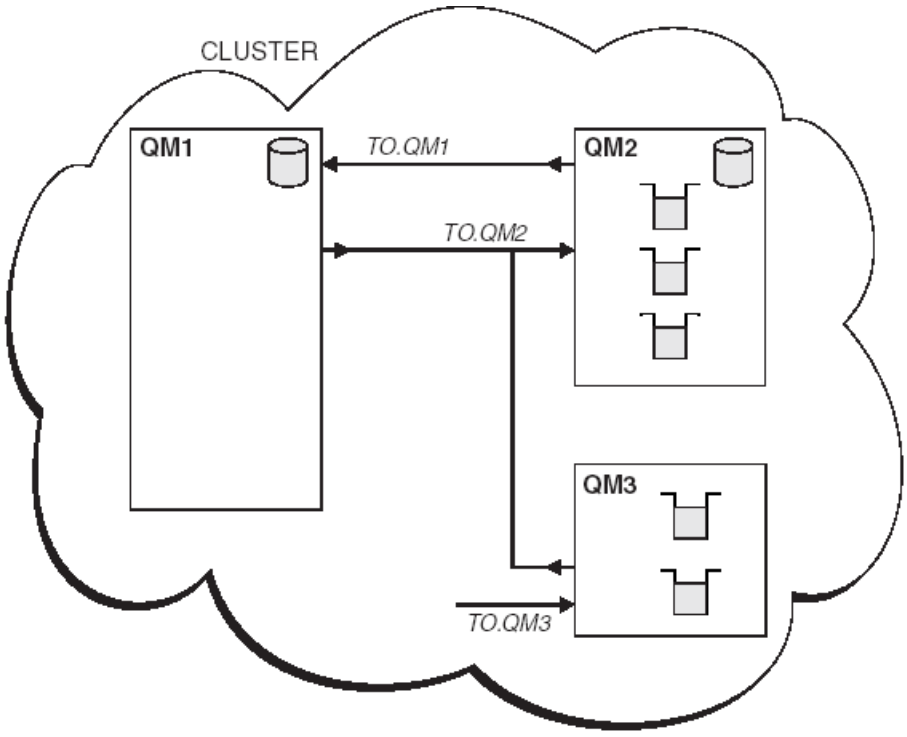
在群集中您不需要创建传输队列定义，因为 WebSphere MQ 在每个队列管理器中已经定义了一个传输队列。这个传输队列可以把消息发送到任何队列管理器中。

群集的信息是存放在资源库中。大多数队列管理器只保留它们自己所需要的信息，这些信息包括与它们通信的队列管理器和队列的信息。一些队列管理器包含了群集中所有队列管理器的所有信息。

下图显示了一个叫 CLUSTER 的群集：

- 在这个群集中有三个队列管理器，QM1,QM2 和 QM3。
- QM1 和 QM2 存放了群集中队列管理器的信息。它们被叫作完全资源队列管理器。
- QM2 和 QM3 中有几个队列，这些队列能被群集中任何其它队列管理器。这些队列被叫作群集队列。
- 每一个队列管理器有一个接收通道的定义，它被叫作群集接收通道。用来接收消息。
- 每一个队列管理器也有一个发送通道的定义，它将和完全资源管理器的群集接收通道相连。这个通道叫做群集发送通道。在下图中,QM1 和 QM3 有群集发送通道连接到 TO.QM2，QM2 有群集发送通道连接到 TO.QM1。

一旦群集接收通道和群集发送通道已经定义好了，通道将自动启动。



图，队列管理器群集

2.5.2 群集的优点

使用群集有两个优点：

1. 减少系统管理

即使您创建了一个很小的群集，都将减少系统管理的工作。在群集中建立队列管理网络比在分布式队列建立网络将使用更少的定义。由于使用更少的定义，您将能够更快和更容易地建立和改变网络。并且降低了定义错误的风险。

2. 增强可用性和实现负载均衡

简单的群集将更容易管理。对于复杂的群集，将提高了扩展性和可用性。因为您可以定义在不同的队列管理器定义相同的队列，因此工作负载可以在群集的队列管理器实现均衡。

2.5.3 群集的组件

- 群集资源库（队列）

资源库中存放了群集中队列管理器的信息，包括队列管理器名，以及它们的通道和队列等。这些资源库信息通过一个叫 SYSTEM.CLUSTER.COMMAND.QUEUE 的队列进行交换，并存放到一个叫 SYSTEM.CLUSTER.REPOSITORY.QUEUE 的固定队列中。

资源库可能是完全或部分的。每个队列管理器至少要连接到一个拥有完全资源库的队列管理器。

每一个群集队列管理器必须有一个叫 SYSTEM.CLUSTER.REPOSITORY.QUEUE 的本地队列，在群集中至少一个群集队列管理器含有完全资源库。对于每个群集队列管理器，必须要预定义一个群集-发送通道连接到资源库队列管理器中。资源库队列管理器之间必须要互连，网络状况要比较好，和具有高可用性。普通队列管理器只包含有部分资源库信息。

- 群集-发送通道

群集-发送通道的类型为 TYPE(CLUSSDR)，群集队列管理器使用群集-发送通道可以把消息发送到完全资源库队列管理器中。这个通道被用来通知队列管理器状态的改变，例如，队列的删除和创建。它仅和第一个完全资源库队列管理器联系。

- 群集-接收通道

群集-接收通道的类型为 TYPE(CLUSRCVR)，群集队列管理器可以使用它接收群集内的消息。每一个群集队列管理器至少需要一个群集-接收通道。

- 群集传输队列

从一个队列管理器发送到其它队列管理器的消息都将被放到 SYSTEM.CLUSTER.TRANSMIT.QUEUE 队列中，在每个队列管理器中必须要存在群集传输队列。

2.5.4 创建群集

下表是一个创建群集的脚本示例，群集名为 NPC。

```
ALTER QMGR REPOS(NPC)

DEFINE CHANNEL(TO.QM0000) +
CHLTYPE(CLUSRCVR) CONNAME('192.168.10.11(1414)') +
CLUSTER(NPC)

DEFINE CHANNEL(TO.QM1000) +
CHLTYPE(CLUSSDR) CONNAME('192.168.10.22(1414)') +
CLUSTER(NPC) +
DESCR('To Other Repository')

DEFINE QLOCAL(0000_1) +
CLUSTER(NPC)
```

其中群集传输队列和命令队列不用显示定义，队列管理器 QM0000 是 NPC 群集中一个完全资源库队列管理器。一个队列管理器可以同时属于多个群集。

在 MQSeries v5.2 以后版本，在群集中能够支持 DHCP：

- 在定义群集-接收通道时，不用说明队列管理器的网络地址。

```
DEFINE CHANNEL(TO.QM0000) +
CHLTYPE(CLUSRCVR) +
TRPTYPE(TCP) +
CLUSTER(NPC)
```

- 在定义群集-发送通道时，不用说明资源库队列管理器名。

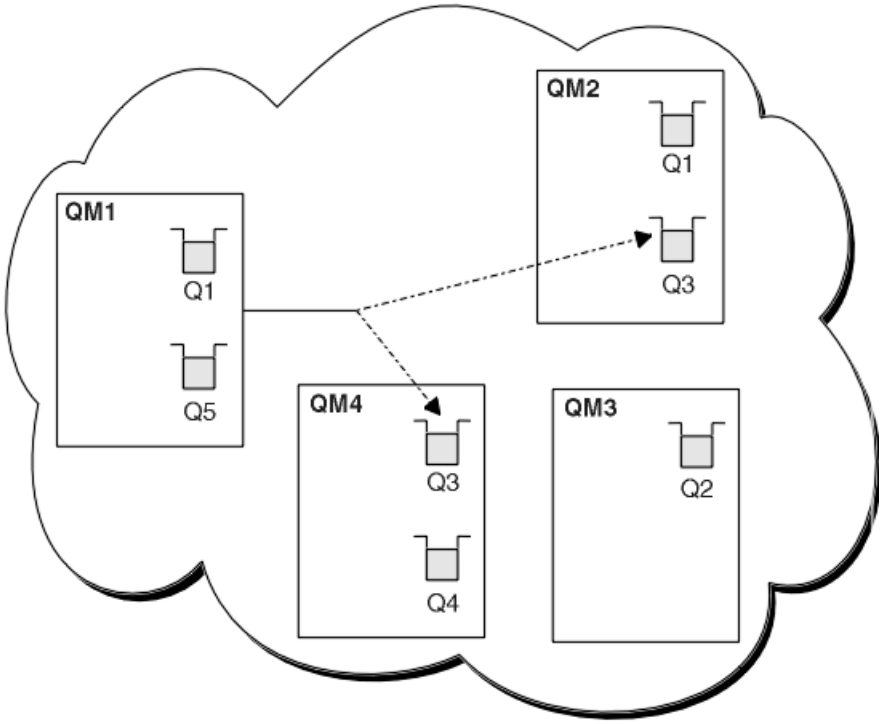
```
DEFINE CHANNEL(TO.+QMNAME+) +
CHLTYPE(CLUSSDR) +
TRPTYPE(TCP) +
CONNAME(...) +
CLUSTER(NPC)
```

2.5.5 实现负载均衡

当在群集中含有同一队列的多个实例时，WebSphere MQ 通过使用负载管理算法把消息发送到最方便的队列管理器中。负载管理算法尽可能地选择本地队列管理器作为目的地。如果在本地队列管理器中没有队列，这个算法将选择最合适的目标。合适的原则是取决于通道状态、队列管理器和队列的可用性。在满足条件的队列管理器之间，这个算法使用了轮循的方法进行选择。从而可以实现负载均衡的功能。

使用群集可以减少系统管理，也可以在群集中的几个队列管理器中创建同名的队列。只有拥有队列的队列管理器才能处理队列中的消息，但应用程序发送消息时不用显示说明队列管理器名。负载管理算法决定了那个队列管理器应该处理消息。

下图中显示了一个群集中定义了多个 Q3 队列的定义。当在 QM1 的应用程序放一个消息到 Q3 时，它不必知道是那个 Q3 队列将处理消息。
当消息正在传输时，群集中的一个本地队列变得不可用，那么消息将被转发到另一个队列中（但需要以 BIND_NOT_FIXED 的选项打开队列）。如果其它队列也不可用，这个消息将被放到死信队列中。
如果目标队列管理器不能工作，那么消息将仍然被放在传输队列中，系统将尝试更换消息的路由。这样做并不会影响消息的完整性，但如果出现队列管理器失败并消息处在可疑（in doubt）状态，这个消息将不能更换路由。



图，在群集中一个队列有多个实例

注意：

2. 在群集中创建同一队列的多个实例时，需要确保您的消息互相之间没有依赖，例如，消息需要按照特定顺序处理或被同一队列管理器处理。
3. 使同一队列的不同实例的定义相同，否则执行 MQINQ 调用将产生不同的结果。

在大多数情况下负载管理算法将满足系统的需求，然而您也可以编写用户出口(user-exit)程序来定制负载管理算法，WebSphere MQ 中包含用户出口和群集负载出口（cluster workload exit），可以使用 ALTER QMGR 命令来激活群集负载出口。例如，ALTER QMGR CLWLEXIT(myexit) 欲了解更详细的有关信息，请参考《Queue Manager Clusters》。

2.5.6 群集管理

您有时需要对群集中的队列管理器进行维护，例如，您可能需要备份队列中的数据，或把补丁应用到系统中。如果队列管理器包含队列，则必须暂停队列管理器。当维护结束后，将继续队列管理器的工作。

- 为暂停队列管理器，可以使用命令 SUSPEND QMGR，例如：

SUSPEND QMGR CLUSTER(SALES)

它将临时从群集中移走一个队列管理器，这样将不会有任何消息发送到这个队列管理

器。暂停的队列管理器和对象的信息都被保留在资源库中，但是队列管理器被标记为暂停。

- 当维护队列管理器的工作完成后，需要让队列管理器继续工作。通过执行 RESUME QMGR 则可以实现，例如：
  RESUME QMGR CLUSTER(SALES)
  RESUME QMGR 命令将通知完全资源库，这个队列管理器将又重新可用。完全资源队列管理器散布这个信息到其它队列管理器。
- 在群集中的队列管理器可以执行刷新启动命令。在正常环境下不需要执行刷新命令，例如：
  REFRESH CLUSTER(SALES)
  执行刷新命令，将丢弃所有本地的关于群集的信息。
- 如果要从群集中强制除去一个队列管理器，则可以通过 RESET CLUSTER 命令实现。或一个队列管理器已经被删除，但是定义到群集的群集-接收通道仍然存在，那么您也可以使用 RESET CLUSTER 命令来快速地清理这些定义信息。

使用 RESET CLUSTER 命令是删除自定义群集-发送通道的唯一方法。在正常的环境中，您不必运行这个命令。这个命令只能在资源管理器中执行，例如：

RESET CLUSTER(SALES) QMNAME(QM0000) ACTION(FORCEREMOVE) QUEUES(NO)

- 查看群集中队列管理器信息，可以使用命令 DISPLAY CLUSQMGR，例如：
  DISPLAY CLUSQMGR(*) CLUSTER(SALES)
- 您可以定义群集队列，例如：
  DEFINE QLOCAL(0000_1) CLUSTER(SALES)

欲了解更详细的有关群集管理的信息，请参考《Queue Manager Clusters》。

2.6 本章小结

本章主要介绍 WebSphere MQ 的基本概念和体系结构，WebSphere MQ 核心是由队列管理器、队列和通道组成。队列和通道分别有多种类型。在实际应用中，根据系统的实际需要选择合适的对象类型进行配置。描述了 WebSphere MQ 队列管理器的触发机制和群集的概念并介绍了它的优点以及实现方法。

2.7 本章练习

1， IBM WebSphere MQ 中包含了那些对象？
2， 练习安装和连接 WebSphere MQ 客户端到服务器端的过程。
3， 在系统中创建的第一个队列管理器是缺省队列管理器？
(1) 是 （2） 否
答案：（2）

4， WebSphere MQ 使用一种什么接口，实现通过程序可以访问队列管理器的资源：

（1） 程序到程序的 API(the program to program API)

（2） 消息队列接口（Message Queue Interface）

（3） 同步模式(the synchronous model)

（4） 触发机制(triggering)

答案：（2）

5， WebSphere MQ 仅异步环境的消息队列。

（1）对 （2）错

答案：（2）

6， 一个消息可以由下列那些部分组成：

（1） 应用数据（application data）

（2） 死信头 (a dead-letter header)

（3） 安全头 (security header)

（4） 消息描述块 (a message descriptor)

（5） 以上所有的 (all the above)

答案：（1）（2）（4）

7， 所有的消息至少有一个头，它是：

（1） MQXQH（transmission header）

（2） MQDLH (dead letter header)

（3） MQMD (message descriptor)

（4） MQTH (trigger header)

答案：（3）

8， 在 WebSphere MQ 触发机制，队列管理器启动被触发的程序：

（1）对 （2）错

答案：（2）

9.临时动态队列中可以包含下列那些消息类型：

（1） 仅回复消息

（2） 仅报告消息

（3） 仅非永久性消息

（4） 永久性和非永久性消息

答案：（3）

第二部分 Websphere MQ 系统管理

第三章 WebSphere MQ 系统安装

目标

1. 由于 WebSphere MQ 支持 35 种以上系统平台，本章仅以 WebSphere MQ Windows 版为例。
2. 介绍 WebSphere MQ 系统安装规划。
3. 学习 WebSphere MQ 系统安装步骤。
4. 描述了 WebSphere MQ 的安装验证过程。

3.1 规划安装

本节概述了运行 WebSphere MQ 所需的硬件和软件环境，所支持的网络协议和编译器、传递媒体以及产品的各种功能部件（以前称为组件）。

内容包括安装、验证和通信设置（假设您使用 TCP/IP 作为通信协议）。可以使用其它协议（例如，SNA、SPX 和 NetBIOS）。本书中没有涉及关于这些协议的特定过程，却有对 WebSphere MQ 库中包含相关信息的其它书籍的引用。但请注意，WebSphere MQ 的以下功能仅在 TCP/IP 下才可使用：

WebSphere MQ MQI 明信片
WebSphere MQ JMS 明信片
WebSphere MQ 资源管理器

注:
尽管在“动态主机配置（DHCP）”的机器上运行的队列管理器可以是群集的成员，但建议群集资源库的队列管理器应该位于具有静态 IP 地址的机器上。

3.1.1 硬件要求

以下是 WebSphere MQ 服务器的硬件要求：

- 任何基于 32 位 Intel 处理器 IBM PC 机（或兼容机）。
- 支持 SNA LU 6.2、TCP/IP、NetBIOS 或 SPX 的通信硬件。

对于典型安装，WebSphere MQ 至少需要大约 85 兆字节（MB）的磁盘空间用于产品代码和数据（如果使用 NTFS）。至少需要 20 MB 作为运行空间。而且，安装进程需要在系统

盘上需要 30M 的临时空间。

3.1.2 软件要求

以下是运行 WebSphere MQ Windows 版的先决条件；列出了最低的支持级别。

操作系统

WebSphere MQ 的安装需要以下操作系统软件之一：

- Microsoft Windows NT 版本 4.0（包括 TCP/IP、NetBIOS 和 SPX）和 Microsoft Windows NT Service Pack 6a。
- Microsoft Windows 2000。可以是 Microsoft Windows 2000 专业版或 Microsoft Windows 2000 服务器版。

连通性

WebSphere MQ 如果需要实现 SNA 连接，则需要安装下列软件之一：

- IBM 通信服务器 Windows NT 版，版本 5.0 和版本 6.1.1。
- Attachmate Extra! Personal Client，版本 6.7。
- Attachmate Extra! Enterprise 2000.
- Microsoft SNA 服务器，版本 4.0。
- Microsoft Host Integrated Server 2000。
- TCP/IP、NetBIOS 和 SPX。它们被包含在操作系统中。

Windows NT 的安装需求

以下是 Windows NT 的安装需求：

- Microsoft Internet Explorer 4.0.1，带有 Service Pack 1 或更高版本。
- Microsoft HTML Help 1.2。

在 WebSphere MQ 服务器 CD-ROM 中提供。

- Microsoft Management Console（MMC）1.1。

在 WebSphere MQ 服务器 CD-ROM 中提供。

- Microsoft Installer（MSI）1.1 或更高版本。

在 WebSphere MQ 服务器 CD-ROM 中提供。

- Microsoft Active Directory Client Extensions（ADCE） Windows NT 版。

在 WebSphere MQ 服务器 CD-ROM 中提供。

- 支持 Java Runtime Environment 版本 1.3 或更高版本。（仅 Java 消息传递需要）。
- Microsoft Windows NT 的 Option Pack 4。（如果需要支持 Microsoft Transaction Server（MTS））。

Windows 2000 的安装需求

- 支持 Java Runtime Environment 版本 1.3 或更高版本。（仅 Java 消息传递需要）。

安装必备软件

要安装 WebSphere MQ 服务器 CD-ROM 中提供的必备软件（不包含服务包或 Web 浏览器），可以执行以下操作：

- 使用 WebSphere MQ 安装程序。

当使用 WebSphere MQ 服务器 CD-ROM 进行安装时，在 WebSphere MQ 安装启动面板中有一个**必备软件**选项。可以使用该选项检查已安装了哪些必备软件和缺少哪些，然后安装缺少的软件。

- 使用 Windows 资源管理器：
  1. 使用 Windows 资源管理器选择 WebSphere MQ 服务器 CD-ROM 上的文件夹 `\MSI`。
  2. 选择要安装的软件项的文件夹。
  3. 如果适合，可以选择所需安装语言的文件夹。它们是：

     de_de
     德语
     en_us
     英语
     es_es
     西班牙语
     fr_fr
     法语
     it_it

意大利语

ja_jp

日语

ko_kr

韩国语

pt_br

巴西葡萄牙语

zh_cn

简体中文

zh_tw

繁体中文

4. 启动安装程序。

WebSphere MQ 功能部件

当安装 WebSphere MQ 时，可选择需要的功能部件。下面显示的功能部件在从服务器 CD 安装 WebSphere MQ 时可用。

服务器

服务器功能部件允许您运行计算机上的队列管理器并与网络上的其它计算机连接。包含 Java 支持。

Windows 客户机

WebSphere MQ 客户机是 MQ 的一个小子集（不包含队列管理器），它使用队列管理器和其它（服务器）计算机上的队列。只有当它所在的计算机和一台正在运行完整 WebSphere MQ 服务器版本的计算机相连接时，它才能使用。根据需要，客户机和服务器可以在同一台机器上。

Java 消息传递

使用 Java（包含 Java Message Service 支持）进行消息传递所需的文件。

开发工具箱

这个功能部件中包含了开发将在 WebSphere MQ 上运行的应用程序时所需的一些源文件样本和各种绑定（.H、.LIB、.DLL 等文件）。绑定和样本是使用以下语言提供的：C、C++、Visual Basic、ActiveX、Cobol、DCE、PL/1。它包括 Java 和 Java Message Service 支持，提供的样本有：MTS、MQSC 和 Lotus Notes。

3.2 安装 WebSphere MQ

3.2.1 WebSphere MQ 文档

WebSphere MQ 文档不再是 WebSphere MQ 产品的部件，现在它作为此产品的单独 CD 软件包而提供。您可以从光盘直接查看文档，或者可以把它们独立安装到计算机上。安装 WebSphere MQ 文档步骤如下：

1. 把 WebSphere MQ 文档 CD 插入 CD-ROM 驱动器。

如果启用自动启动，则会启动安装进程。如果不启用，则双击 CD-ROM 上的根文件夹中的**安装**图标以启动安装程序。

2. 依照安装提示进行操作，即可安装完成。

3.2.2 WebSphere MQ 安装

介绍关于如何安装 WebSphere MQ Windows 版的步骤。安装过程大约要需要 30 分钟。以下步骤演示了如何进行典型安装。

1. 将 WebSphere MQ Windows 版服务器 CD-ROM 插入 CD-ROM 驱动器。
2. 如果安装了自动运行，那么会启动安装进程。如果不启用，则双击 CD-ROM 上的根目录中的 **Setup** 图标以启动安装程序。
3. 请等待，直到出现“WebSphere MQ 安装启动板”窗口为止。
4. 可选地，要更改安装的本地语言，单击**选择语言**图标，然后从列表中选择所需的语言。
5. 选择**必备软件**选项。

如图所示列出典型安装的必备软件，请参阅图，每个安装项的右边，有一个勾号（表示已安装）否则一个叉号（表示没有安装）。
如果出现了叉号：

a. 单击项目左边的 + 符号以显示安装链接。
b. 选择要使用的安装源的选项。从以下各项选择：
   - WebSphere MQ CD
   - 因特网
   - 网络
c. 安装完成时，单击项目左边的 - 符号。

注：对于定制安装，可能不需要所有的必备软件。

6. 当安装了所有的必备软件时，选择**网络先决条件**选项。
7. 选择 **WebSphere MQ 安装**选项。
8. 请选择**启动 WebSphere MQ 安装程序**，然后等待，直到显示了带有欢迎消息的“WebSphere MQ 安装”窗口为止。
9. 单击**下一步**继续。
10. 请阅读面板上的信息和许可证条款。

要更改显示许可证协议的语言，单击**更改语言**，然后从提供的列表中选择您需要的语言。选择此选项接受许可证条款，然后单击**下一步**。

11. 如果机器上未安装过此产品的前一个版本，则显示“安装类型”面板。

选择希望的安装类型，然后单击**下一步**。显示了安装类型以及每种选项所安装的功能部件。 选择“典型”安装。

12. 如果机器上已安装了 WebSphere MQ 的前一版本，则显示“安装进程的类型”面板。选择以下一个选项，然后单击**下一步**：
    - 更新。安装与前面版本相同的功能部件。转至下一步。
    - 定制。可以选择要安装哪些功能部件。
13. “WebSphere MQ 安装”窗口显示以下消息：

安装 WebSphere MQ 就绪，该窗口还显示您选中的安装摘要。要继续，单击**安装**。

14. 会询问您在您的计算机上是否已为处理程序数购买了足够的容量单元。

如果您有足够的容量单元，单击**是**。 如果没有足够的容量单元，单击**否**。会通知您必须获取足够的容量单元，以在计算机上运行此软件。单击**是**继续，或单击**否**取消安装。显示“安装 WebSphere MQ”面板。 等待，直到进展栏结束。

成功安装 WebSphere MQ 后，“WebSphere MQ 安装”窗口显示以下消息： 安装向导成功完成

15. 单击**完成**启动“准备 WebSphere MQ”向导。

3.3 验证安装

3.3.1 安装验证

安装验证的方法有许多种，可以用明信片应用程序，或可以使用一个队列管理器和一个队列的简单配置来验证本地安装。使用样本应用程序将消息放置到队列并从队列读取该消息。 下面介绍手工创建对象验证安装的方法。

使用以下步骤来安装队列管理器和队列：

1. 创建名为 venus.queue.manager 的缺省队列管理器。在窗口的命令提示符下，输入以下命令：

crtmqm -q venus.queue.manager

2. 启动队列管理器。输入以下命令：

strmqm venus.queue.manager

3. 启用 MQSC 命令。输入以下命令：

runmqsc venus.queue.manager

4. 定义名为 ORANGE.QUEUE 的本地队列。输入以下命令：

define qlocal (orange.queue)

MQSC 中的任何小写字母都将自动转换成大写，除非用单引号将它们括起来。这意味着如果用名称 orange.queue 创建了队列，则记住在 MQSC 以外的其它命令中必须使用 ORANGE.QUEUE。

5. 停止 MQSC。输入以下命令：

end

现在已经定义了以下对象：

- 名为 venus.queue.manager 的缺省队列管理器
- 名为 ORANGE.QUEUE 的队列

3.3.2 测试对象

要测试队列和队列管理器，请使用样本程序 **amqsput** （将消息放入队列）和 **amqsget**（从队列获取消息）：

1. 启动 MS-DOS 窗口，进入到 c:\Program Files\IBM\WebSphere MQ\bin 目录下。
2. 将消息放入队列，输入以下命令：

amqsput ORANGE.QUEUE

显示以下消息：

Sample amqsput0 start
target queue is ORANGE.QUEUE

3. 输入一些字符数据，然后按 Enter 键**两次**。显示以下消息：

Sample amqsput0 end

现在消息已经被放在队列。

4. 要从队列获取消息，请输入以下命令：

amqsget ORANGE.QUEUE

在屏幕上将显示您刚才输入的字符数据消息。暂停后，例子程序结束。

如果以上步骤都能完成，则完成了本地安装的验证。

注:
如果在任何阶段中断整个安装过程，则应该从头开始重新运行安装。

3.4 本章小结

本章主要介绍在 Windows 平台 WebSphere MQ 的安装需求和安装过程，并介绍了安装后的安装验证过程。

3.5 本章练习

1. 列出安装 WebSphere MQ 产品的任务列表。
练习安装 WebSphere MQ for Windows 产品。
2. 产品安装完毕后，然后验证安装是否正确？
3. 在 Windows 2000 系统中安装 WebSphere MQ 产品前，必须工作有：
   （1） 安装 MQS.INI 配置文件。
   （2） 用管理组的成员登录系统。
   （3） 创建通道。
   （4） 为每个队列管理器创建通讯链路。

答案：（2）

第四章 WebSphere MQ 的管理

目标

5. 学习和掌握 WebSphere MQ 的管理，管理任务包括创建、启动、修改、查看、停止和删除 WebSphere MQ 对象（队列管理器，队列，群集，进程，名字列表，和通道）。
6. 掌握 WebSphere MQSC 和 PCF 命令 。
7. 学习 WebSphere MQ 配置。
8. 了解 WebSphere MQ 安全性。
9. 了解 WebSphere MQ 事务性。
10. 学习 WebSphere MQ 死信队列处理程序。

4.1 本地和远程管理

WebSphere MQ 可以从本地或者远程管理对象。

*本地管理*是对在本地系统上定义的任何队列管理器执行管理任务。也可以访问其它系统，例如通过 TCP/IP 终端仿真程序 **telnet**，对其他系统进行管理。在 WebSphere MQ 中，认为是本地管理，因为没有通道定义，它们之间的通信是由操作系统提供的。

WebSphere MQ 也支持*远程管理*，可以从本地系统上发出一条命令到远程系统上运行。例如，您可以发出一条命令修改远程队列管理器上的队列定义。如果需要进行远程管理，则需要有相应的通道定义，远程系统上的队列管理器和命令服务器必须正在运行，但您不必登录到远程系统。 在非 Windows 系统的平台上，有些命令不能用这种方式发出，特别是创建或启动队列管理器和启动命令服务器。要执行这种任务，您必须登录到远程系统并从那里发出命令或创建一个进程来处理您的命令。但在 Windows 系统上，您可使用 WebSphere MQ 服务管理单元来实现这个功能。

4.2 使用命令管理 WebSphere MQ

有三种命令集合，可用于管理 WebSphere MQ，分别是控制命令、MQSC 命令和 PCF 命令。

4.2.1 控制命令

控制命令是用来维护队列管理器本身的管理命令。 关于控制命令将在”第六章 WebSphere 控制命令”作详细地介绍。

4.2.2WebSphere MQ 脚本（MQSC）命令

MQSC 命令是用来管理队列管理器对象，包括队列管理器本身、通道、队列和进程定义。 可以使用 **runmqsc** 向队列管理器发出 MQSC 命令。命令的输入有两种方式，一种是交互式命令，另一种是从 ASCII 文本文件中重定向输入命令。在这两种方式中，命令的格式是相同的。

runmqsc 命令有三种运行模式：

- *验证模式*，用这种方式在本地队列管理器上验证 MQSC 命令，但实际上并不运行 。
- *直接模式*，用这种方式在本地队列管理器上运行 MQSC 命令 。
- *间接模式*，用这种方式在远程队列管理器上运行 MQSC 命令 。

使用 MQSC 命令

使用 MQSC 命令的规则

当使用 MQSC 命令时，应该遵循以下规则：

- 每个命令的第一参数是动词，第二个参数是名词。第三个参数可能是对象名或对象统配符。随后的参数没有次序关系。如果参数有一个相应的值，则值必须紧跟在和它相关的参数之后。
- 可以用空格和逗号分隔关键字、括号和值。
- 在命令的开始或结束和参数、标点符号和值之间可以有任意多个空格。例如，以下命令是有效的：

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

- 不允许重复参数。
- 字符串中可以包含空格、小写字母或特殊字母，但必须包含在单引号中，而且不能包含下列特殊字符：
    - 句话（.）
    - 前斜杠（/）
    - 下划线（_）
    - 百分号（%）

如果字符串本身包含引号，则引号用两个单引号替代。如果小写字母没有用单引号包含，则将被自动转换成大写。

- 不包含字符的字符串（即，在两个引号之间没有空格）被认为是('') 或(' ')。
- 左圆括号后跟着右圆括号，它们之间没有重要的信息，例如 NAME ( )，这被认为是无效。
- 关键字不区分大小写 — AltER、alter 或 ALTER 都是同样的含义。
- 一些参数定义同义词。例如，DEF 总是 DEFINE 的同义词，因此 DEF QLOCAL 是有效的。但是，同义词不仅仅是最小字符串；DEFI 不是 DEFINE 有效的同义词。

  注：
  DELETE 参数没有同义词。

有特殊意义的字符

创建 MQSC 命令时，以下字符有特殊意义：

	空格用作分隔符。多个空格相当于一个空格，包含在引号（'）中的字符串除外。
,	逗号用作分隔符。多个逗号相当于一个逗号，包含在引号（'）中的字符串除外。
'	单引号表明字符串的开始或结束。所有被包含在引号中的字符，WebSphere MQ 将不进行转换处理。计算字符串长度时，不包括包含的引号。
''	WebSphere MQ 将一个字符串中的两个引号一起作为一个引号看待，并且字符串没有结束。计算字符串长度时，将双引号作为一个字符。
(	一个左圆括号表明参数列表的开始。
)	一个右圆括号表明参数列表的结束。
:	冒号表示包含的范围。例如（1:5）意思是（1、2、3、4、5）。此方法仅可在 TRACE 命令中使用。
*	星号表示'所有'。例如，DISPLAY TRACE (*) 指显示所有跟踪，DISPLAY QUEUE (PAY*) 指显示所有名称以 PAY 开头的队列。

当您需要在一个字段中使用这些特殊字符时，您必须将整个字符串包含在单引号中。

创建命令脚本

命令脚本的规则如下：

- 每个命令都必须以新行开始。
- 在不同平台上，关于命令行的长度和记录格式的可能会有区别。如果希望脚本能适用于不同平台，则每行的实际长度应该限定为 72 个字符。
- 每一行不必以键盘控制字符结束（例如，tab）。
- 如果行上的最后非空格字符是：
  - 减号（-），表明命令将从下一行的开始继续。
  - 加号（+），表明命令将从下一行的第一个非空格字符继续。如果您使用 + 继续一个命令，则记住要在下个参数前保留至少一个空格（z/OS 除外）。

- 当命令重新组合成单个字符串时，行末使用的 + 和 – 被忽略。
- 忽略第一个位置中以星号（*）开始的行，这代表文件中的注释行。空白行将被忽略。
- MQSC 命令属性名称限制为 8 个字符。

MQSC 命令在其它平台上都可用，包括 OS/400(R) 和 z/OS。*《WebSphere MQ 脚本（MQSC）命令参考》* 包含每个 MQSC 命令和它的语法的描述。

WebSphere MQ 对象的命名规则

WebSphere MQ 队列、进程、名称列表、通道和存储类对象存在于各自独立的对象*名字空间*中，因此不同类型的对象可以有相同的名称。但是，同一类型中的对象不能同名。（例如，本地队列不能和模型队列有相同的名称。）WebSphere MQ 中的名称都区分大小写；住不包含在引号中的小写字符都将转换为大写。

用于命名 WebSphere MQ 对象的字符集如下所示：

- 大写 A-Z
- 小写 a-z
- 数字 0-9
- 句号（.）
- 前斜杠（/）
- 下划线（_）
- 百分号（%）。

注:

1. 不允许首字符为空格或嵌入空格。
2. 首字母或尾字母不能为下划线。
3. 少于完整字段长度的任何名称都可在其右边添加空格以达到规定长度。被队列管理器返回的短名称总是在其右边填满空格。
4. 任何名称结构（例如，使用句号或下划线）对队列管理器而言都是无意义的。
5. 队列名称可长达 48 个字符。
6. 系统缺省的对象名称请参看附录。

4.2.3PCF 命令

WebSphere MQ 可编程命令格式（PCF）命令使得管理任务能编写到应用程序中。在程序中可以创建队列和进程定义和更改队列管理器。PCF 命令和 MQSC 命令具有相同的命令集。 可使用 WebSphere MQ 管理接口（MQAI）可以更容易访问 PCF 消息。

PCF 的原理

PCF 定义了命令和回复消息，应用程序通过这些命令和回复消息实现和队列管理器之间的信息交换，通过 WebSphere MQ 的应用程序可以实现对 MQ 对象的管理包括：队列管理器，进程定义，队列和通道。应用程序可以通过一个本地队列管理器集中管理网络中的任何本地和远程管理器。

每一个队列管理器有一个管理队列，应用程序可以发送 PCF 命令消息到管理队列中，每一个队列管理器也有一个命令服务器，它是为管理队列中的消息而服务。因此在网络中的任何队列管理器可以处理 PCF 消息，通过使用定义的回复队列，回复消息可以被返回给应用程序。PCF 命令和回复消息是使用 MQI 进行发送和接收的。

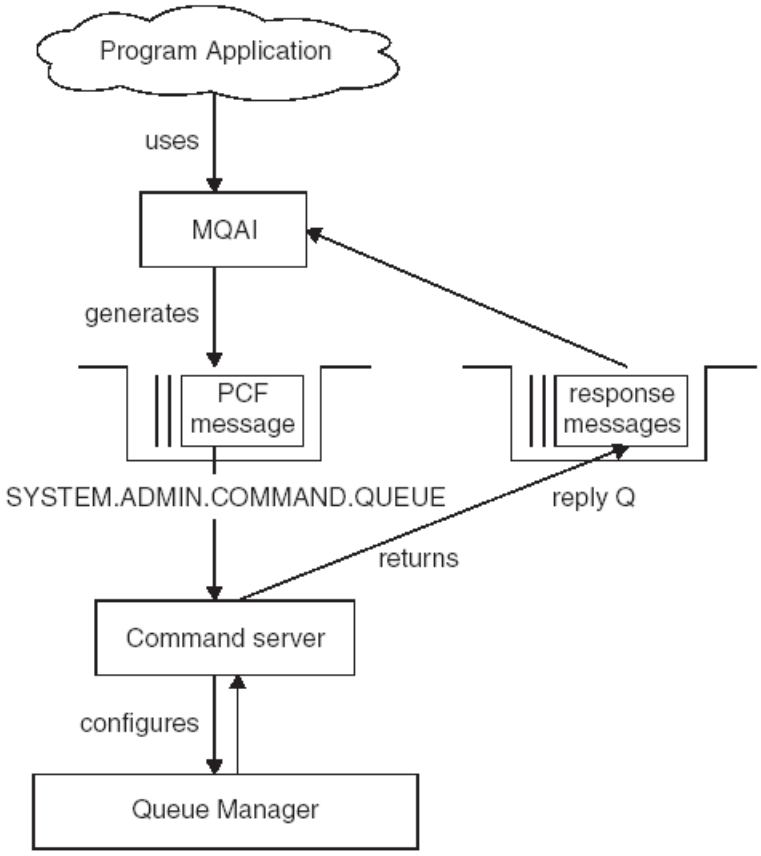
WebSphere MQ 管理接口(MQAI)

在 WebSphere MQ for Windows, AIX, iSeries, Linux, HP-UX, and Solaris and WebSphere MQ for OS/2 Warp 版本，都支持 WebSphere MQ Administration Interface(MQAI)。

MQAI 是 WebSphere MQ 提供的一种实现发送和接收 PCF 的接口。MQAI 通过使用数据报（data bags）来处理对象的属性，这样比直接使用 PCF 更简单。

MQAI 通过传递参数到数据包的方法，从而提供了更简单的访问 PCF 消息的编程接口。这样只要一条语句就可以实现一个结构。编程人员不需要处理数组和分配空间，也不需要了解 PCF 的消息结构。

MQAI 通过发送 PCF 消息到命令服务器并等待回复消息来管理 WebSphere MQ，如图所示。



图：MQAI 的原理图

4.3 WebSphere MQ 配置

本节主要描述在 Windows 系统上和在 UNIX 系统上更改配置信息。

4.3.1 在 Windows 系统上更改配置信息

WebSphere MQ 所有配置信息都存储在 Windows 注册表中。您可使用 WebSphere MQ 服务管理单元或使用 amqmdain 命令编辑配置信息。

建议使用 WebSphere MQ 服务管理单元修改配置信息。**不要**直接编辑注册表系统文件，因为这如果对编辑注册表失败，可能会严重影响 WebSphere MQ 系统或 Windows 操作系统

的运行。

4.3.2 在 UNIX 系统上更改配置信息

在 UNIX 平台上，可以更改以配置文件中的 WebSphere MQ 属性：

- WebSphere MQ 配置文件（**mqs.ini**），它的属性将影响整个 WebSphere MQ 节点系统，每个节点有一个 mqs.ini 文件。
- 队列管理器配置文件（**qm.ini**），它的属性仅影响某个队列管理器，在节点中的每个队列管理器都有一个 qm.ini 文件。

配置文件中包含一个或多个节，每节都是为一个专门功能服务，如日志功能、通道功能和可安装服务。

因为 WebSphere MQ 配置文件用于维护和队列管理器相关的数据，不合法的或不正确的配置文件可能导致一些或全部 MQSC 命令失败。同样，应用程序不能与在 WebSphere MQ 配置文件中没有定义的队列管理器连接。

对配置文件所做的任何修改都不会立即生效，只有重新启动队列管理器才生效。

编辑配置文件前，最好先备份它。 编辑的方式有两种：一种是自动的，在节点上使用更改队列管理器配置的命令 ；另一种是手工的，使用标准文本编辑器 。

如果您在配置文件属性上设置了非法值，则系统忽略此值并会提示错误信息。
当创建新队列管理器时，最好先备份 WebSphere MQ 配置文件和新队列管理器配置文件。

如果出现以下情况时，可能需要编辑配置文件，例如：

- 丢失配置文件。（如果可以，从备份中恢复。）
- 需要移动一个或多个队列管理器到新目录中。
- 需要更改缺省队列管理器。

根据以下优先级设置配置文件属性值：

- 在命令行上输入的参数优先于在配置文件中定义的值。
- qm.ini 文件中定义的值优先于 mqs.ini 文件中定义的值

WebSphere MQ 配置文件（mqs.ini）

WebSphere MQ 配置文件 mqs.ini 包含和节点上所有队列管理器都相关的信息。它在安装期间自动创建。 WebSphere MQ UNIX 系统版的 mqs.ini 文件在 `/var/mqm` 目录中。它包含：

- 队列管理器的名称
- 缺省队列管理器的名称
- 和每个文件关联的文件位置

如图显示 WebSphere MQ 配置文件的示例：

```
#*******************************************************************#
#* Module Name: mqs.ini                                            *#
#* Type         : WebSphere MQ Configuration File                  *#
#* Function     : Define WebSphere MQ resources for the node       *#
#*******************************************************************#
AllQueueManagers:
#*******************************************************************#
#* The path to the qmgrs directory, below which queue manager data *#
#* is stored                                                       *#
#*******************************************************************#
DefaultPrefix=/var/mqm

LogDefaults:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=104
   LogType=CIRCULAR
   LogBufferPages=0
   LogDefaultPath=/var/mqm/log

QueueManager:
   Name=saturn.queue.manager
   Prefix=/var/mqm
   Directory=saturn!queue!manager

QueueManager:
   Name=pluto.queue.manager
   Prefix=/var/mqm
   Directory=pluto!queue!manager

DefaultQueueManager:
   Name=saturn.queue.manager
```

队列管理器配置文件 (qm.ini)

队列管理器配置文件 qm.ini, 包含和特定队列管理器相关的信息。每个队列管理器都有一个队列管理器配置文件。创建队列管理器时，将自动创建此文件。

qm.ini 文件保存在队列管理器的目录树的根中。例如，队列管理器 QMNAME 的配置文件的路径和名称是： /var/mqm/qmgrs/QMNAME/qm.ini

队列管理器名称最大长达 48 个字符。

如图显示了在 WebSphere MQ UNIX 系统版中队列管理器配置文件的配置属性。

```
#*******************************************************************#
#* Module Name: qm.ini                                             *#
#* Type           : WebSphere MQ queue manager configuration file  *#
#   Function     : Define the configuration of a single queue manager *#
#*******************************************************************#
ExitPath:
   ExitsDefaultPath=/var/mqm/exits

Service:
   Name=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQ.UNIX.auth.service
   Module=/opt/mqm/bin/amqzfuno.o [1]
   ComponentDataSize=0

Service:
   Name=NameService
   EntryPoints=5

ServiceComponent:
   Service=NameService
   Name=MQ.DCE.name.service
   Module=/opt/mqm/lib/amqzfa [2]
   ComponentDataSize=0

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=1024
   LogType=CIRCULAR
   LogBufferPages=0
   LogPath=/var/mqm/log/saturn!queue!manager/

XAResourceManager:
  Name=DB2 Resource Manager Bank
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB
  XACloseString=
  ThreadOfControl=THREAD
```

```
CHANNELS:
   MaxChannels = 20              ; Maximum number of Channels allowed.
   MaxActiveChannels  = 100    ; Maximum number of Channels allowed to be
                                  ; active at any time.
TCP:                              ; TCP/IP entries.
   KeepAlive  = Yes              ; Switch KeepAlive  on
```

4.4 WebSphere MQ 安全性

WebSphere MQ 队列管理器传递的信息可能都是有价值的数据，因此您需要使用授权系统来保证未授权的用户无法访问您的队列管理器。所以需要考虑以下安全性控制：

谁可以管理 WebSphere MQ

你可以定义一些用户通过命令来管理 WebSphere MQ。

谁可以使用 WebSphere MQ 对象

您可以定义哪些用户（通常是应用程序）可以使用 MQI 调用和 PCF 命令执行以下操作：

- 连接到队列管理器。
- 访问类似队列、名字列表和进程的对象，访问这些对象的方式。
- 访问 WebSphere MQ 消息。
- 访问与消息相关联的上下文信息。

通道安全性

您需要确保用于将消息发送到远程系统的通道可以访问必需的资源。您还需要确保该通道被已授权的用户操纵。

管理 WebSphere MQ 的权限

WebSphere MQ 管理员有执行以下任务的权限：

- 使用命令（包含为其它用户授权 WebSphere MQ 授权的命令）
- 访问 WebSphere MQ Windows 版上的管理单元

要成为 WebSphere MQ 管理员，您必须是 *mqm* 组的成员（或是 Windows 系统上的管理员组的成员）。mqm 组是在安装 WebSphere MQ 时自动创建的；可以把其他用户加入到 mqm 组中，也可以把用户从 mqm 组中删除。

在 UNIX 平台上，还创建了 mqm 的用户。所有 WebSphere MQ 对象都属于 mqm 用户。

在 Windows 系统上，Administrator 组的成员也可以管理任何队列管理器。您还可以在域控制器上创建 mqm 组，并将其添加到本地 mqm 组。有些命令必需只有属于 mqm 组的

成员才可以操作，例如,创建队列管器 crtmqm。但要执行以下操作，您不需要成为 mqm 组的成员：

- 使用 PCF 命令或 MQSC 命令。
- 从应用程序发出 MQI 调用（除非您要在 **MQCONNX** 调用上使用快速路径绑定）。
- 使用 **crtmqcvx** 命令。
- 使用 **dspmqtrc** 命令。

使用 WebSphere MQ 对象的权限

队列管理器、队列、进程、名称列表和认证信息（authinfo）对象都是从使用 MQI 调用或 PCF 命令的应用程序访问的。这些资源是被 WebSphere MQ 保护，所以应用程序需要有许可权来访问它们。

不同的主体组可以将不同类型的访问权限授权相同的对象。例如，对于特定队列，可能允许一个组执行放入和获取操作；可能仅允许另一个组浏览队列（带有浏览选项的 **MQGET**）。类似地，一些组可能已经放入和获取了队列权限，但未被允许改变队列的属性或删除队列。

一些操作是特别敏感的，应该限制为特权用户使用。例如：

- 访问一些特殊的队列，例如传输队列或命令队列 SYSTEM.ADMIN.COMMAND.QUEUE
- 运行使用全部 MQI 上下文选项的程序
- 创建和删除应用程序队列

所有对象的完全控制许可权是自动给予创建对象的用户标识和 mqm 组的成员的（以及给予 Windows 系统上的本地管理员组的成员的）。

4.5 WebSphere MQ 事务性支持

应用程序可以把一些列更新组合成一个*工作单元*。这些更新通常是逻辑相关的，为了保持数据完整性，所有的更新必须成功，如果一部分更新成功，一部分更新失败，将失去数据完整性。

工作单元成功完成后就**提交**。此时，所有在工作单元内所做的更新都将变成永久的并且是不可逆的。如果工作单元失败了，所有的更新都将被回滚。*同步点协调*是工作单元用来实现提交或回滚保证数据完整的进程。

*本地*工作单元上唯一更新的那些资源是 WebSphere MQ 队列管理器的资源。这里同步点协调是由队列管理器自身使用单阶段提交进程提供的。

*全局*工作单元上属于其它资源管理器的资源，例如符合 XA 的数据库，也同时被更新。必须使用两阶段提交过程，并且工作单元可由队列管理器自身协调，或由其它符合 XA 的事务管理器（例如 IBM TXSeries 或 BEA Tuxedo）。

总之，队列管理器资源可作为本地或全局工作单元的一部分进行更新：

本地工作单元

当要更新的资源仅为 WebSphere MQ 队列管理器的那些资源时，使用本地工作单元。使用 **MQCMIT** 动词提交更新，或使用 **MQBACK** 复原。

全局工作单元

当您还需要将 XA 相应的数据库管理器的更新包含到使用全局工作单元。此处对队列管理器的协调可以是内部或外部的。

队列管理器协调

全局工作单元可以使用 **MQBEGIN** 动词启动，然后使用 **MQCMIT** 提交或使用 **MQBACK** 复原。两阶段提交进程是在 XA 相应的资源管理器（例如，DB2(R)、Oracle 和 Sybase）在首次要求*准备*提交时使用的。仅当所有准备都成功时，然后要求将它们提交。如果任何资源管理器发出其无法准备提交的信号，则要求将它们复原。

外部的协调

此处，协调由 XA 相应的事务管理器（例如，IBM CICS、Transarc Encina(R)、或 BEA Tuxedo）执行。工作单元是在事务管理器的控制下提交的。**MQBEGIN**、**MQCMIT** 和 **MQBACK** 动词都是不可用的。

4.6 WebSphere MQ 死信队列处理程序

死信队列（DLQ），有时指*未送达消息队列*，是一个保存不能发送到它们目标队列的消息的队列。网络中的每个队列管理器应该有一个关联的 DLQ。

消息可用队列管理器、消息通道代理程序（MCA）和应用程序放在 DLQ 上。DLQ 上的所有消息的前缀必须是用*死信头*结构 MQDLH 的结构。

队列管理器或消息通道代理程序放在 DLQ 上的消息总是有 MQDLH；将消息放在 DLQ 上的应用程序必须提供 MQDLH。MQDLH 结构的*原因*字段包含标识为什么消息在 DLQ 上的一个原因码。

所有 WebSphere MQ 环境需要一个在 DLQ 上有规律地处理消息的例程。WebSphere MQ 提供一个缺省例程*死信队列处理程序*（DLQ 处理程序），您可使用 **runmqdlq** 命令来调用它。

用用户写入*规则表*方式将 DLQ 上的处理消息的说明提供给 DLQ 处理程序。即，DLQ 处理程序匹配和规则表中条目相对的 DLQ 上的消息；DLQ 消息匹配规则表中的一个条目时，DLQ 处理程序执行和条目关联的操作。

4.7 本章小结

本章主要介绍了 WebSphere MQ 的管理命令，共有三种命令，第一种是控制命令，主要是用在维护队列管理器本身的管理命令；第二种是 MQSC 命令，是用来管理队列管理器对象和队列管理器本身；第三种是 PCF 命令，它具有和 MQSC 命令相同的命令集，它主要用于在应用程序实现对队列管理器的维护。还介绍了 WebSphere MQ 配置信息和队列管理配置信息的存放位置，并举例介绍了 unix 和 Window 平台的配置文件。对 WebSphere MQ 的安全性、事务性和死信队列处理程序作了简单的介绍。

4.8 本章练习

1. WebSphere MQ 关心下列那个安全服务：
   （1） 认证（authentication）
   （2） 授权（authorization）
   （3） 访问控制（access control）
   答案：（3）

2. WebSphere MQ 管理员需要为 WebSphere MQ 对象设置所有的安全性。
   （1）对 （2）错
   答案：（2）

3. 在 WebSphere MQ 中可以被保护的是：
   （1） 队列
   （2） 通道
   （3） 定义
   答案：（1）（2）

4．在一个工作单元中，changes 是 atomic。
（1）对 (2)错
   答案：（1）

5．在同步点控制的 MQPUT 意味着：
   （1） 直到提交之后，消息才被放到队列中。
   （2） 消息被放到队列，“getting”程序需要检查同步标记。
   （3） 消息被放到队列，但直到提交后“getting”程序才能处理这些消息。
   答案：（3）

6．WebSphere MQ 客户端可以使用全局工作单元处理。
（1）对 （2）错
   答案：（2）

7.下列那些选项可以保证 WebSphere MQ for Windows NT 的应用数据的提交和回滚？
（1） WebSphere Application Server
（2） MQCMIT 和 MQBACK 调用
（3） Windows NT 平台支持的 XA-compliant Syncpoint coordinator
（4） Microsoft Transaction Server
（5） 以上都是
答案：（5）

第五章 WebSphere MQ 控制命令

目标

11. 了解 WebSphere MQ 控制命令
12. 熟悉 WebSphere MQ 控制命令集

5.1 如何使用控制命令

如果需要使用控制命令，则用户必须属于 mqm 组。控制命令在不同平台上的使用会有些注意事项，如下所示：

WebSphere MQ Windows 版

所有控制命令都可以从命令行发出。使用 WebSphere MQ 资源管理器管理单元可以发出子集。命令名和它们的标志是不区分大小写的：您可以用大写、小写或大小写组合进行输入。但是，控制命令的自变量（如队列名）是区分大小写的。

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

WebSphere MQ UNIX 版

所有 WebSphere MQ 控制命令都可以从 shell 发出。所有命令都是区分大小写的。

WebSphere MQ 对象的名称

通常，WebSphere MQ 对象名可以有多达 48 个字符。此规则适用于所有以下对象：

- 队列管理器
- 队列
- 进程定义
- 名称列表
- 群集
- 认证信息（authinfo）对象

通道名的最大长度是 20 个字符。
可用于所有 WebSphere MQ 名称的字符是：

- 大写 A-Z
- 小写 a-z
- 数字 0-9
- 句点（.）
- 下划线（_）
- 正斜杠（/）（请查看注 1）

- 百分号（%）（请查看注 1）

注:

1. 正斜杠和百分号是特殊字符。如果在名称中使用这些字符中的任意一个，则使用此名称时必须加上双引号。
2. 不允许以空格开头或嵌入空格。
3. 不允许使用本地语言字符。
4. 名称可以加双引号，但是仅当名称中包含特殊字符时才需要。

5.2 控制命令

控制命令集

以下是每个 WebSphere MQ 控制命令的参考信息：

命令名	目的
amqmcert	管理 SSL 证书
amqmdain	配置或控制 WebSphere MQ 服务（仅 Windows 系统）
crtmqcvx	转换数据
crtmqm	创建本地队列管理器
dltmqm	删除队列管理器
dmpmqaut	转储打开对象的权限
dmpmqlog	转储日志
dspmq	显示队列管理器
dspmqaut	显示打开对象的权限
dmpmqcap	显示处理程序容量和处理程序数
dspmqcsv	显示命令服务器状态
dspmqfls	显示文件名
dspmqtrc	显示格式化跟踪输出（HP-UX、Linux 和 Solaris）
dspmqrtn	显示事务的详细信息
endmqcsv	停止队列管理器上的命令服务器
endmqlsr	停止队列管理器上的侦听器进程
endmqm	停止本地队列管理器
endmqtrc	停止对实体的跟踪（不用于 AIX）
rcdmqimg	向日志写对象的映象
rcrmqobj	根据它们在日志中的映象重新创建一个对象
rsvmqtrn	提交或逆序恢复事务
runmqchi	启动通道启动器进程

runmqchl	启动发送方或请求者通道
runmqdlq	启动死信队列处理程序
runmqlsr	启动侦听器进程
runmqsc	向队列管理器发出 MQSC 命令
runmqtmc	调用客户机的触发器监控器（仅 AIX 客户机）
runmqtrm	调用服务器的触发器监控器
setmqaut	更改打开对象的权限
setmqcap	设置处理程序容量
setmqcrl	设置证书撤销列表（CRL）服务器定义
setmqscp	设置服务连接点（仅 Windows 系统）
strmqcsv	启动队列管理器的命令服务器
strmqm	启动本地队列管理器
strmqtrc	启用跟踪（不用于 AIX）

控制命令举例

1. 此命令创建一个称为 Paint.queue.manager 的缺省队列管理器，创建系统和缺省对象，并请求两个主日志文件和三个次日志文件：

crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager

2. 下列命令删除队列管理器 travel 并且也抑制任何由该命令发出的消息。

dltmqm -z travel

3. 此命令立即结束名为 saturn.queue.manager 的队列管理器。完成所有当前 MQI 调用，但不允许新的调用。

endmqm -i saturn.queue.manager

5.3 本章小结

本章介绍主要介绍如何使用 WebSphere MQ 控制命令和熟悉 WebSphere MQ 的控制命令集。

5.4 本章练习

1. 使用 CRTMQM 控制命令创建缺省队列管理器的选项是哪一个？
   （5） -d
   （6） -q

（7） -x

（8） -u

答案：（2）

2. 一个 WebSphere MQ 应用使用如下定义创建了一个队列：

DEFINE QLOCAL(TEST)

DEFPRTY(0)

MSGDLVSQ(FIFO)

TRIGMPRI(5)

TRIGTYPE(DEPTH)

TRIGDPTH(10)

TRIGGER

当什么条件发生时，将产生触发消息？

（1） 没有触发消息产生。

（2） 当队列中有 5 个消息时。

（3） 当队列中有 10 个消息时。

（4） 当队列中有 5 个优先级消息时。

（5） 当队列中有 10 个优先级为 5 的消息时。

答案（1）

3. 在 WebSphere MQ for Windows 平台上执行如下控制命令：

crtmqm /t 5000 /u MY.DEAD.LETTER.QUEUE travel

这个命令将能完成如下什么功能？

（1） 它定义了触发间隔。

（2） 它定义了队列 MY.DEAD.LETTER.QUEUE。

（3） 创建了一个名为 travel 的队列管理器。

（4） 设置了队列的最大消息数 5000。

答案：（1）（3）

4. 执行“runmqchl /c CHAN1”命令将产生什么结果？

（1） 通道 CHAN1 将被启动。

（2） 通道 CHAN1 将和队列管理器 CHAN1 相关。

（3） 缺省队列管理器中的 CHAN1 通道被启动。

（4） 由于 sender/requester 参数没有说明，所以将返回错误消息。

答案：（3）

5. 使用下列那个命令，可以实现当前所有 MQI 调用完成之后，停止队列管理器？

（1） endmqm /c

（2） endmqm /i

（3） endmqm /p

（4） endmqm /z

答案：（2）

第六章 WebSphere MQ 互连通信

目标

1. 描述 WebSphere MQ 产品之间的互连。
2. 学习掌握 WebSphere MQ 互连的概念；传输队列，消息通道代理程序和通信链路一起组合形成了消息通道。
3. 学习怎样通过消息通道把地理位置分开的队列管理器构成一个队列管理器网络。
4. 学习判断 WebSphere MQ 出口（exits）的需要，对于指定的需求给出推荐使用的出口（exits）。
5. 学习使用通道的维护和配置侦听程序。

6.1 基本概念

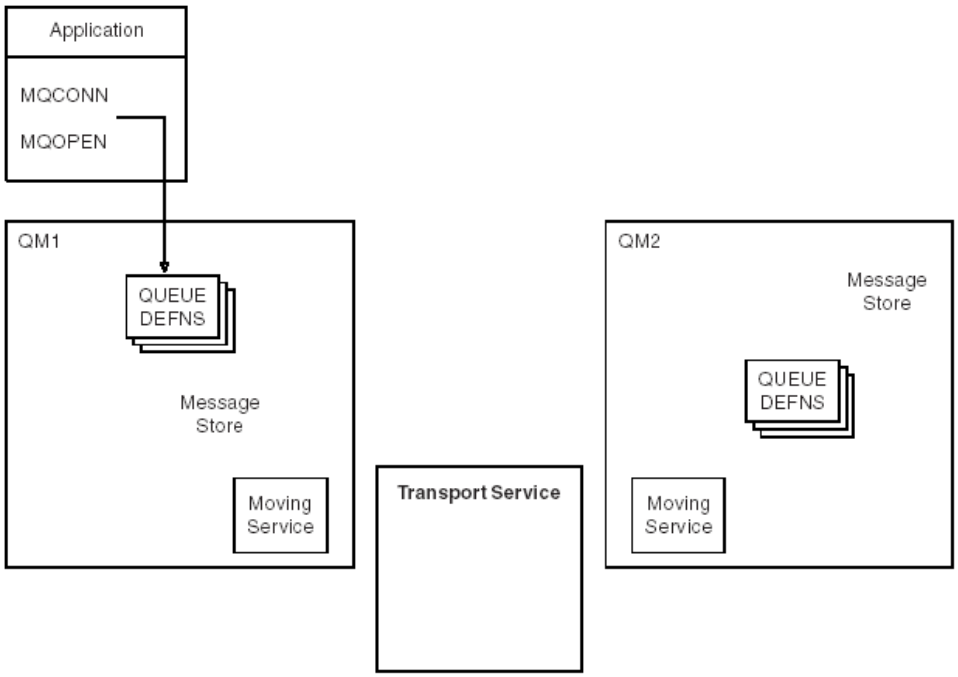
6.1.1 什么是互连通信

对于WebSphere MQ，互连通信意味着把消息从一个队列管理器发送到另一个队列管理器，接收队列管理器可能是在本地机器或远程机器。发送队列管理器和接收队列管理器可以运行在相同的平台，也可以是WebSphere MQ所支持的任何平台，这种环境叫做分布式环境。WebSphere MQ使用DQM(Distributed Queue Management)来实现分布式环境间的通信。

本地队列管理器有时被叫做源队列管理器（source queue manager），而远程队列管理器有时被叫做目标队列管理器（target queue manager），或叫伙伴队列管理器(partner queue manager)。

分布式队列的工作原理

下图概括描述了分布队列的组件。



图，分布式队列组件

1. 应用程序使用 MQCONN 调用连接到队列管理器。
2. 然后应用程序使用 MQOPEN 调用打开一个队列，以便可以把消息放到队列中。
3. 队列管理器拥有每个队列的定义。
4. 如果消息的目的地是远程系统的队列，那么本地队列管理器将保存消息直到消息被发送到远程队列管理器。这些对于应用程序来说是透明的。
5. 每个队列管理器都有叫做 *moving service* 通信软件;通过这个服务,一个队列管理器可以和另一个队列管理器通信。
6. *transport service* 不依赖于队列管理器，它可以是下列其中的一种，这和平台有关。

- SNA APPC （Systems Network Architecture Advanced Program-to Program Communication）
- TCP/IP (Transmission Control Protocol/Internet Protocol)
- NetBIOS (Network Basic Input/Output System)
- SPX (Sequenced Packet Exchange)
- UDP (User-Datagram Protocol)

分布式组件的定义：

1. WebSphere MQ 应用程序可以把消息放到已连接的队列管理器的队列中。
2. 队列管理器可以定义属于自己的队列，也可以定义属于其他队列管理器的队列。这些队列叫做远程队列定义。WebSphere MQ 应用程序也可以把消息放置到远程队列中。
3. 如果消息要发送到远程队列管理器中， 本地队列管理器把消息存放到传输队列（*transmission queue*）中直到它消息发送到远程队列管理器。传输队列是一个特殊的本地队列，在传输队列中的消息一直被保存到被成功地发送到远程队列管理器。
4. 负责消息发送和接收的软件叫消息通道代理（*Message Channel Agent* (MCA)）。
5. 消息是通过通道（*channel*）来实现队列管理器之间传输，通道是一条两个队列管理器之

间的通信链路，它可以把队列中的所有消息发送到远程队列管理器。

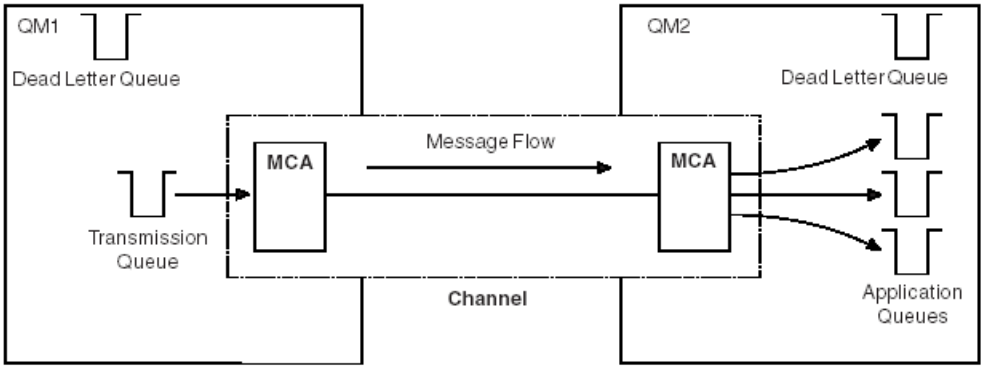
发送消息所需要的组件：

如果要把消息发送到远程队列管理器中，那么本地队列管理器需要定义一个传输队列和一个通道。通道的两端都需要分别定义，例如，发送端或接收端。一个简单通道是由本地队列管理器一个发送通道和远程队列管理器一个接收通道所组成的。这两个定义必须是相同的名字，并共同构成一个通道。

在通道的每一端也被叫做消息通道代理（*Message Channel Agent* (MCA)）。

每一队列管理器都应该有一个死信队列（*dead-letter queue*），也叫做不能交付的消息队列（undelivered message queue）。如果消息不能到达目的队列，则将被放置到死信队列。

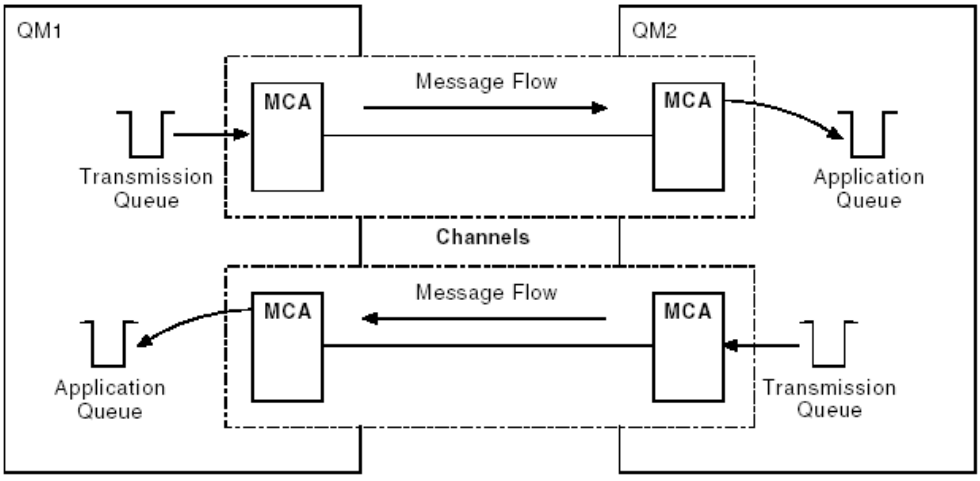
下图显示了队列管理器、传输队列、通道和MCA之间的关系。



图，发送消息

实现接收消息所需的组件：

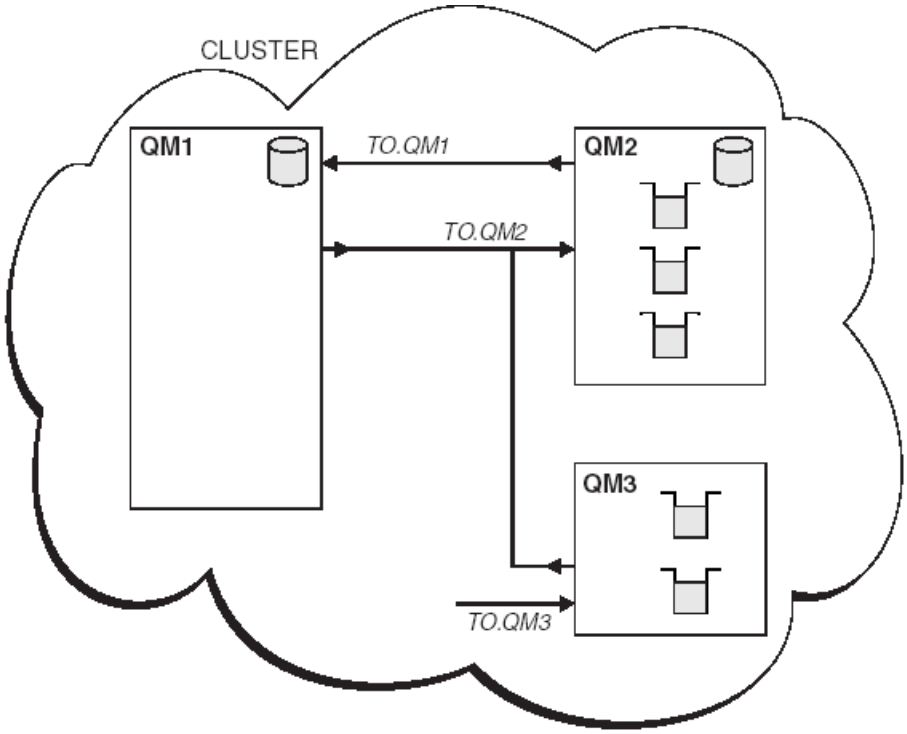
如果您的应用程序要从远程队列管理器返回到本地队列管理器，则需要定义另一个通道，它的传输方向与发送的相反。



图，消息的双向传输

群集组件：

与传统的 WebSphere MQ 网络相比的另一种网络是群集。支持群集的版本有 WebSphere MQ for AIX, iSeries, HP-UX, Solaris, z/OS, and Windows, WebSphere MQ V5.1 for Compaq Tru64 UNIX, 和 OS/2 Warp ，群集是由具有逻辑关系的队列管理器所组成的网络。集群中的任何队列管理器可以发送消息到同一集群其他队列管理器，而不需要显示地定义通道、远程队列和与目标相对应的传输队列。在群集中的每个队列管理器只有一个传输队列来负责把消息发送到群集中的其他队列管理器。每个队列管理器只需要一个 cluster-receiver 通道和一个 cluster-sender 通道。下图显示了名叫“CLUSTER”的集群的组件：



图，队列管理器的群集

- “CLUSTER”集群中有三个队列管理器，分别是 QM1, QM2 和 QM3。
- QM1 和 QM2 存放了关于集群中的队列管理器和队列的完全资源信息。
- QM2 and QM3 有一些群集队列。这些队列可以被群集中的其他队列管理器访问。
- 每个队列管理器有一个名叫“TO.qmgr”的 cluster-receiver 通道，该通道负责接收消息。
- 每个队列管理器也有一个 cluster-sender 通道，该通道可以发送消息到其中一个资源队列管理器中。
- QM1 和 QM3 发送资源信息到 QM2，并且 QM2 发送资源信息到 QM1 中。
- 您可以使用 MQPUT 调用把消息发送到任何其他队列管理器，但是只可以使用 MQGET 调用从本地队列中检索消息。

为了解更详细的关于群集的情况，请参看《*WebSphere MQ Queue Manager Clusters*》。

6.1.2 分布式队列组件

本节描述分布式队列组件，它们分别是：

- 消息通道
- 消息通道代理
- 传输队列
- 通道启动器和侦听程序
- 通道出口程序

6.1.2.1 消息通道

关于消息通道的内容，前面的章节已经详细介绍了，不再赘述。

6.1.2.2 消息通道代理（MCA）

MCA 是一个控制消息发送和接收的程序。在通道的每一端都有一个 MCA。一个 MCA 是把消息从传输队列取出来，然后放到通讯链路上。另一个 MCA 接收消息，并把消息放到远程队列管理器的队列中。

初始化通信链路的 MCA 叫做呼叫 MCA，则另一个 MCA 叫响应 MCA。呼叫 MCA 可以是发送通道，群集发送通道，服务器通道（完全意义的）或请求器通道。响应 MCA 可以是关联除群集发送通道之外的任何通道。

6.1.2.3 传输队列

传输队列是一个特殊的本地队列，它是用来存放将被发送到远程队列管理器的消息的队列。在分布式队列环境中，您需要位每一个发送 MCA 定义一个传输队列，除非使用了 WebSphere MQ 的队列管理器群集。

您在远程队列定义中说明了传输队列名，如果没有说明传输队列名，那么队列管理器将会寻找一个和远程队列管理器相同名字的传输队列。

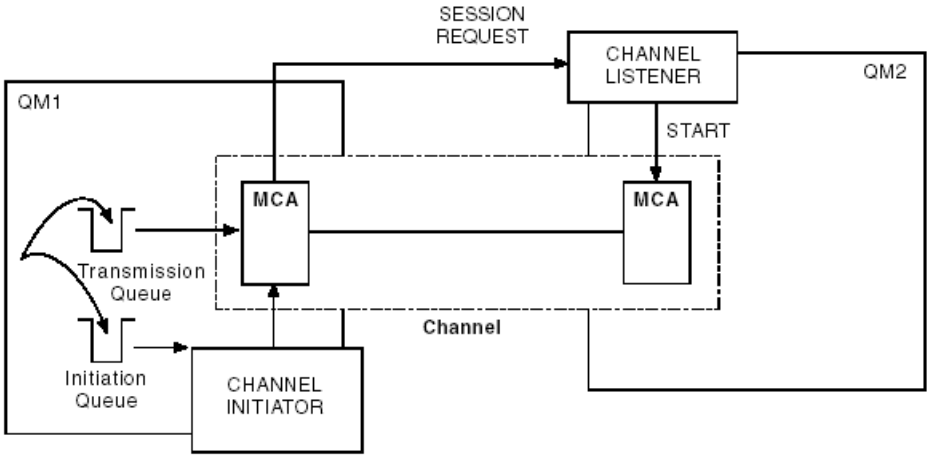
您可以为队列管理器说明一个缺省传输队列。如果您没有说明传输队列名，并且没有定义和远程队列管理器同名的传输队列，那么将会使用这个缺省传输队列。

6.1.2.4 通道启动器和侦听程序

通道启动器为发送通道充当了触发监控器，因为传输队列可以被定义为一个触发队列。当一个消息到达传输队列中，并满足了队列的触发条件，一个触发消息将被放到启动队列中，通道启动器将取出触发消息来启动相应的发送通道。如果在服务器通道中定义了对方的连接名，则也能触发服务器通道。这意味着基于消息到达传输队列中，使得通道能够被自动启动。

您需要一个通道侦听程序来启动接收（响应）MCA。为了响应呼叫 MCA 的启动请求

接收 MCA 被启动；通道侦听程序探测接入网络请求，并启动相应的通道。



图，通道启动器和侦听程序

通道启动器的使用依赖于平台，在 z/OS 平台，每个队列管理器只能有一个通道启动器；在其他平台，您可以启动多个通道启动器，并对每一个通道启动器指定一个启动队列。通常您只需要一个通道启动器。在 WebSphere MQ for AIX, iSeries, HP-UX, Solaris 和 Windows systems, 和 WebSphere MQ for Compaq Tru64 UNIX, 和 OS/2 Warp 平台上可以启动三个通道启动器(缺省值)，但是您可以修改这个值。对于支持群集的平台，当启动队列管理器时，通道启动器也会被自动启动。

通道侦听程序的使用也是和平台相关的。在 OS/2 和 Windows 系统中, 您可以使用 WebSphere MQ 提供的通道侦听程序，也可以使用操作系统提供的程序。MQ 的侦听程序有两种配置和启动方式，一种是通过配置/etc/inetd.conf 文件选择使用 inetd 方式, 也可以使用 MQ 自身提供的 runmqlsr 程序，所不同的是：runmqlsr 是一个线程应用，所以比 inetd 需要更少的内存消耗。因此，采用 runmqlsr 方式可以提高通道相关的性能。与 MQ 应用程序类似，MQ 的通道侦听程序也有 trusted(fastpath)和 non trusted(standard)两种运行方式，采用 trusted 运行方式可以降低 CPU 和内存消耗。将通道和侦听程序设置为 trusted 方式运行的方法是通过修改 qm.ini 配置文件中的 MQIBindType 参数来实现，即创建或修改 qm.ini 文件中与 Channels 相关的小节，例如：

Channels:

MQIBindType=FASTPATH 或者

MQIBindType=STANDARD

其中，FASTPATH 表示 trusted 运行方式，而 STANDARD 表示非 trusted 运行方式。

6.1.2.5 通道出口程序

如有您想做一些额外的处理（例如，加密或数据压缩），则可以编写通道出口程序或使用 SupportPacs 。 使 用 网 址 ： http://www.software.ibm.com/WebSphere MQ/txppacs/txpsumm.html 找到 WebSphere MQ 的事务处理 SupportPacs。

WebSphere MQ 的通道出口程序是在 MCA 执行过程中被调用的。通道出口程序有六种类型：

- **安全出口（Security exit）**

用作安全性检查，例如，接收方认证。

- **消息出口（Message exit）**

对消息的操作，例如，传输前进行加密。

- **发送和接收出口（Send and receive exits）**

对消息切割的操作，例如，数据压缩和解压。

- **消息重试出口（Message-retry exit）**

用于处理消息不能到达目的地的问题。

- **通道自动定义出口（Channel auto-definition exit）**

用于自动定义接收通道和服务器连接通道。

- **传输重试出口（Transport-retry exit）**

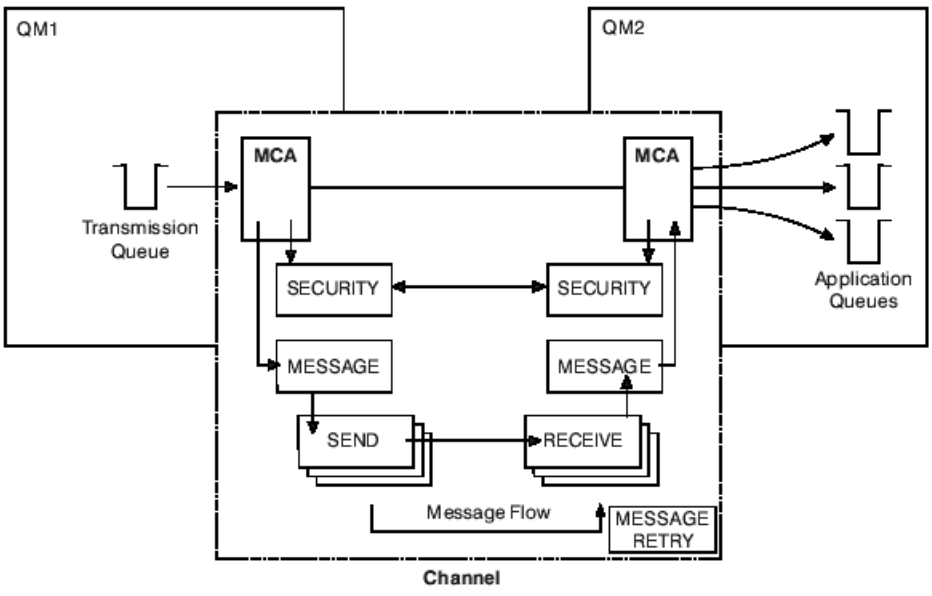
当通讯失败时，用来暂停把数据发送到通道。

通道出口程序的处理顺序如下：

1. 通道双方数据初始化成功后，调用通道安全出口。在通道启动阶段.这些必须要成功完成，才能把消息发送出去。

2. 通道消息出口程序被发送 MCA 调用，被发送到接收 MCA 的每个消息部分都将调用发送出口程序。

3.接收 MCA 当接收到消息的每部分就会调用接收出口程序，然后当整个消息接收完成后调用消息出口程序。



图，通道出口程序被调用的次序

消息重试出口程序被用来决定接收 MCA 在采取其他策略之前试图放消息到目标队列的次数。这个出口程序在 WebSphere MQ for z/OS 上不支持。

6.1.3 死信队列

如果消息不能被发送到正确的目的地，那么消息将被发送到死信队列，例如由于目标队列不存在，或队列已满。死信队列也能在通道的发送方使用，例如，数据转换失败。我们建议针对每个队列管理器定义一个死信队列。如果没有定义死信队列，MCA 将不能放消息，消息就会被滞留在传输队列中，那么通道将被停止。

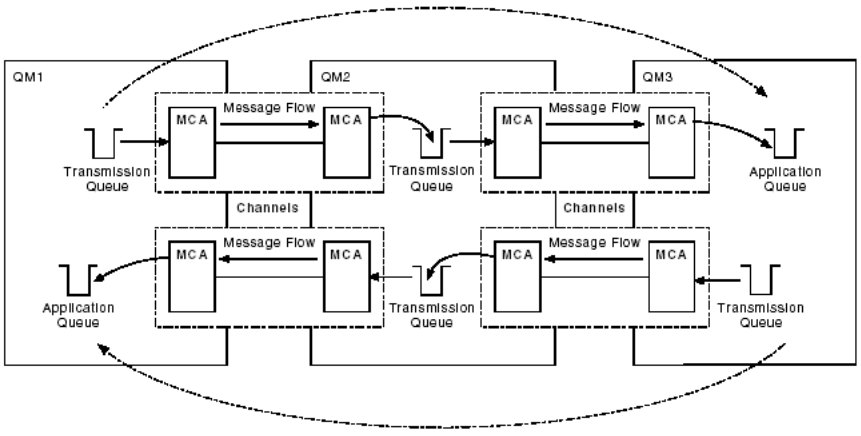
如果快速、非永久的消息不能被交付，并且目标系统没有定义死信队列，那么这些非永久消息就会被丢弃。然而使用死信队列将会影响消息的交付次序。

6.1.4 怎样到达远程队列管理器

在源和目标队列管理器不一定总是有一个通道，可以考虑使用下列方案。

多跳（Multi-hopping）

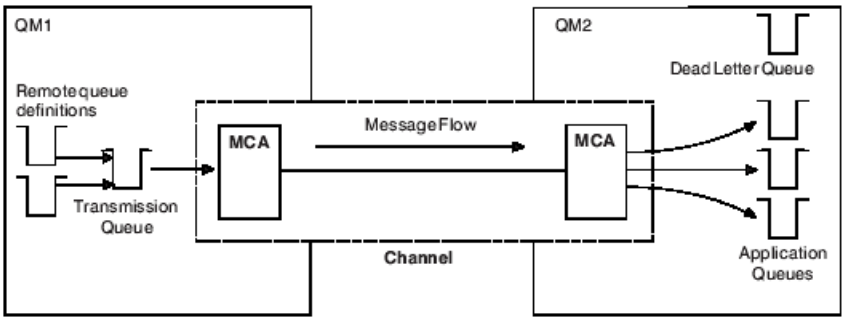
如果源队列管理器和目标队列管理器之间没有直接的通讯链路，从源队列管理器到目标队列管理器之间可能会存在多个中间队列管理器，这种连接方式叫做多跳。在所有的队列管理器之间需要定义通道，在中间的队列管理器需要定义传输队列，如下图所示：



图，通过中间队列管理器到达目标队列管理器

共享通道（Sharing channels）

作为应用程序设计者，为了要把消息发送到目标队列管理器，您可以在应用程序中说明远程队列管理器名和队列名，或为每个远程队列创建一个远程队列定义，这个远程队列定义包含了远程队列管理器名，队列名和传输队列名。这两种方法都是通过相同的传输队列把消息发送到同一目的地，如图所示：

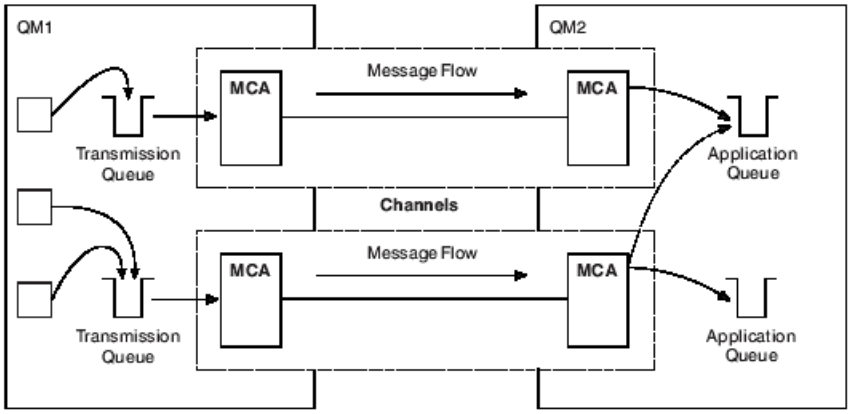


图，共享传输队列

上图描述了多个应用程序的消息可以使用同一个通道到达不同的远程队列。

使用不同的通道

如果您有不同类型的消息需要发送，那么可以在两个队列管理器之间定义多个通道。使用多个通道一般是考虑安全性或传输的负载均衡。
为了又创建另一条通道，则需要定义另一个通道和另一个传输队列，或创建一个远程队列。应用程序可以使用任何一条通道把消息发送到同一目标队列管理器。



图，使用多个通道

当您使用了远程队列定义，那么在应用程序中不用说明目标队列管理器和目标队列。通过使用远程队列定义，应用程序则不用关心目标队列的位置和传输队列。

使用群集

在群集中的每个队列管理器定义了一个群集接收通道。当一个队列管理器要发送消息到另一个队列管理器时，它将自动定义相应的群集发送通道。例如，如果在群集中一个队列有多个实例，那么群集发送通道将被定义到拥有队列的所有队列管理器。WebSphere MQ 使用了一种负载均衡管理算法（轮循机制）来选择一个可用的队列管理器。如果想了解更详细的情况，请参考《*WebSphere MQ Queue Manager Clusters*》。

6.2 实现应用程序通信

本节主要更详细地介绍 WebSphere MQ 产品之间的互连通信，阅读本节之前需要理解通道、队列等概念。

6.2.1 发送消息到远程队列管理器

这部分描述了把消息从一个队列管理器发送到另一个队列管理器的最简单的方法，发送消息前需要做如下的准备工作：

1. 检查通讯链路，确保可用。
2. 启动队列管理器。
3. 启动通道启动器。
4. 启动侦听程序。

您也需要有合适的WebSphere MQ权限来创建对象。
在源队列管理器上需要定义如下对象：
– 发送通道
– 远程队列定义
– 启动队列 (在z/OS平台上是必须的，其它平台是可选的。)
– 传输队列
– 死信队列 (推荐)
在目标队列管理器上需要定义如下对象：
– 接收通道
– 目标队列
– 死信队列(推荐)
针对不同的平台，您可以使用不同的方法来定义对象：
例如，在OS/2, Windows systems, UNIX systems, and Compaq OpenVMS Alpha 平台，您可以使用《*WebSphere MQ Script (MQSC) Command Reference*》书中的WebSphere MQ 命令或《*WebSphere MQ Programmable Command Formats and dministration Interface*》书中PCF 命令。仅在Windows系统中,您可以使用图形界面，WebSphere MQ explorer和WebSphere MQ Web Administration。

定义通道

为了把消息从一个队列管理器发送到另一个队列管理器，您需要定义两个通道；一个是在源队列管理器，另一个是在目标队列管理器。

- **在源队列管理器**

定义一个通道类型为 SENDER 的通道，您需要说明如下属性：
1，使用的传输队列名(XMITQ 属性)。
2，目标系统的连接名(CONNAME 属性)。

3，正使用的通讯协议（TRPTYPE 属性)。在 AIX, iSeries, Compaq Tru64 UNIX, HP-UX, OS/2 Warp, Solaris, 和 Windows, 可以不设置这个属性，而使用通道的缺省值。

- **在目标队列管理器**

定义一个通道类型为 RECEIVER 的通道，并且和发送通道同名。设置正使用的通讯协议(TRPTYPE 属性)。在 AIX, iSeries, Compaq Tru64 UNIX, HP-UX, OS/2 Warp, Solaris, 和 Windows, 可以不设置这个属性，而使用通道的缺省值。

接收通道的定义都普遍相同，如果您有几个队列管理器同时往同一个队列管理器发送消息，那么发送通道和接收通道都是相同的名字，这样在接收方只要定义一个接收通道就可以负责接收所有的消息。

当已经定义了通道，您可以使用“PING CHANNEL”来测试通道。这个命令从发送通道发送了一个特殊的消息到接收通道，并且检测它的返回情况。

定义队列

为把消息从一个队列管理器发送到另一个队列管理器，您需要定义六个队列；在源队列管理器需要定义四个，目标队列管理器要定义两个。

- **在源队列管理器**

1，远程队列定义

在这个定义中需要说明下列属性：

远程队列管理器名

目标队列管理器名

远程队列名

在目标队列管理器上的目标队列名。

传输队列名

传输队列的名称，您可以不说明它。如果没有说明这个属性，将会使用一个和目标队列管理器同名的传输队列，或者这个传输队列不存在，将会使用缺省传输队列。建议传输队列名和目标队列管理器同名，以便这个队列在缺省情况下能找到它。

2，启动队列定义

在z/OS平台上，这是必须定义的；其他平台上使可选的。在z/OS平台上您必须定义队列名为“SYSTEM.CHANNEL.INITQ”的启动队列，在其它平台您也可以使用它。

3，传输队列定义

一个本地队列的“USAGE”属性被设置成“XMITQ”。如果您使用了 WebSphere MQ for iSeries ，那么“USAGE”属性是“*TMQ”。

4，死信队列定义—推荐

定义一个死信队列是为了存放不能交付的消息。

- **在目标队列管理器**

1，本地队列定义

目标队列。这个队列名必须和在源队列管理器中的远程队列定义中远程队列名的属性值相同。

2，死信队列—推荐

定义一个死信队列是为了存放不能交付的消息。

发送消息

当您把消息放在源队列管理器的远程队列定义中时,这消息将被存放在传输队列中，直到通道被启动。当通道被启动，消息被交付到远程队列管理器的目标队列中。

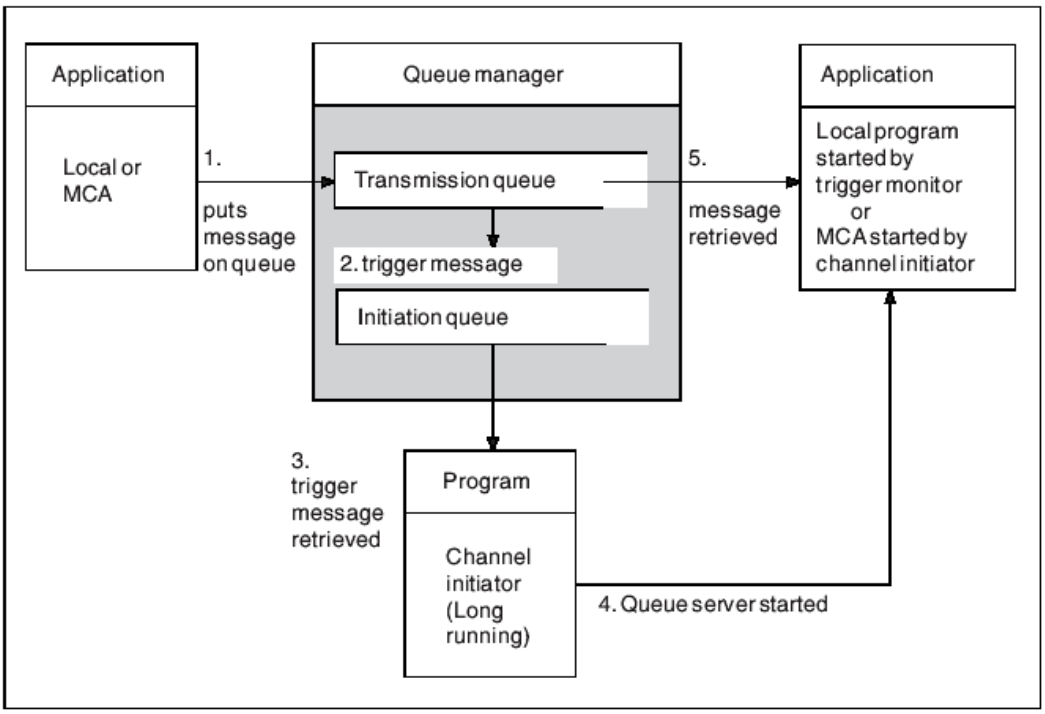
启动通道

在发送队列管理器上使用“START CHANNEL”命令启动通道。当您启动发送通道时，接收通道将被自动地启动（由侦听程序启动），消息然后被发送到目标队列。消息通道的两端都必须处在“running”状态，消息才能被发送。由于通道的两端是处在不同的队列管理器中，它们使用了不同的属性。

发送消息之前，发送 MCA 把大消息分割成小消息，然后通过通道发送出去，在目标队列管理器再把这些小消息组装成大消息，这些步骤对用户来说是透明的。一个 MCA 在传输消息时可以启动多个线程，这种叫“*pipelining*”的过程能更有效地提高消息传输的效率，从而提高通道的性能。

6.2.2 触发通道

下面主要介绍触发的概念，您可以在《WebSphere MQ Application Programming Guide》文档中找到关于触发机制更详细的描述。



图，触发的概念

上图描述了触发所需的对象和事件发生的顺序：
1. 本地队列管理器从应用程序或 MCA 放一个消息到传输队列。
2. 当触发条件满足时，本地队列管理器产生一条触发消息并放到启动队列。
3. 长期运行的通道启动器监控着启动队列，并检索队列中的触发消息。
4. 通道启动器按照消息中的内容处理触发消息。触发消息中可能包括需要被启动的通道名。
5. 被触发的本地应用程序或 MCA 将从传输队列中检索消息。

为了配置触发环境，您需要做如下准备工作:
1，创建传输队列，设置启动队列属性为“SYSTEM.CHANNEL.INITQ”。
2，确保启动队列“SYSTEM.CHANNEL.INITQ”存在。
3，确保通道启动程序是可用的并在运行。启动通道启动器时需要提供启动队列名。
4，创建进程定义，如果没有创建，那么确保进程定义的“*UserData*”属性值包含了通道名，对于 WebSphere MQ for AIX, iSeries, HP-UX, Linux, Solaris 和 Windows systems, 和 WebSphere MQ for Compaq Tru64 UNIX, 和 OS/2 Warp平台, 可以不创建进程定义，如果不创建进程定义，您可以在传输队列的“*TriggerData*”属性中设置为通道名。
5，如果使用了进程定义，请确保传输队列定义中包含了进程定义名，和启动队列名，并设置合适的触发条件。

注意:
1. 通道启动器程序充当了“触发监控器”，监控着启动队列，用来启动通道。
3. 可以定义多个启动队列和触发进程。
4. 推荐的触发类型为“FIRST”。

6.2.3 消息的安全性

分布式队列管理通过通道两端的同步点协调来确保消息被正确地传输。如果协调过程发现了错误，将关闭通道，把消息安全地存放在传输队列中，直到通道被重新启动。当通道启动时，同步过程可以恢复可疑（*in-doubt*）状态。处理通道的可疑状态可疑用下列方法：
1. 用提交或回滚的方法 Resolve 通道。
2. 复位通道的消息序号。

只有意外的情况，才会出现通道可疑的情况，一般情况通道能自动解决可疑状态。

快速, 非永久消息

在 WebSphere MQ for AIX, iSeries, HP-UX, Linux, Solaris, Windows systems, 和 OS/2 Warp 平台，通道的 NPMSPEED (nonpersistent message speed) 属性可以用来说明任何非永久性消息将更快地传输。当快速、非永久消息在传输中，通道出现了中断，消息可能被丢失。如果接收通道不能把消息放到目的队列时，则将被放到死信队列中；如果死信队列没有定义，那么消息将被丢弃。

6.2.4 WebSphere MQ 对象配置实例

```
*echo "   修改队列管理器的死信队列为 DEADQ"
ALTER    QMGR +
         DEADQ('DEADQ')  +
         TRIGINT(10000)

*echo "   定义远程队列"
DEFINE QREMOTE ('0000_1') +
         DEFPSIST(YES) +
         XMITQ('QM0000') +
         RNAME('0000_1')  +
         RQMNAME('QM0000')  +
         REPLACE

DEFINE QREMOTE ('0000_2') +
         DEFPSIST(YES) +
         XMITQ('QM0000') +
         RNAME('0000_2')  +
         RQMNAME('QM0000')  +
         REPLACE

*echo "   定义死信队列"
DEFINE QLOCAL  ('DEADQ') +
         DEFPSIST(YES) +
         MAXDEPTH(20000)  +
         MAXMSGL(4194304)  +
         REPLACE

*echo "   定义错误日志队列"
DEFINE QLOCAL  ('ERRMSG') +
         DEFPSIST(YES) +
         MAXDEPTH(10000)  +
         MAXMSGL(1048576)  +
         REPLACE

*echo "   定义本地接收队列"
DEFINE QLOCAL  ('nnnn_1') +
         DEFPSIST(YES) +
         MAXDEPTH(100000)  +
         MAXMSGL(1048576)  +
         TRIGGER +
         TRIGTYPE(FIRST) +
         PROCESS('CICS.PRO') +
```

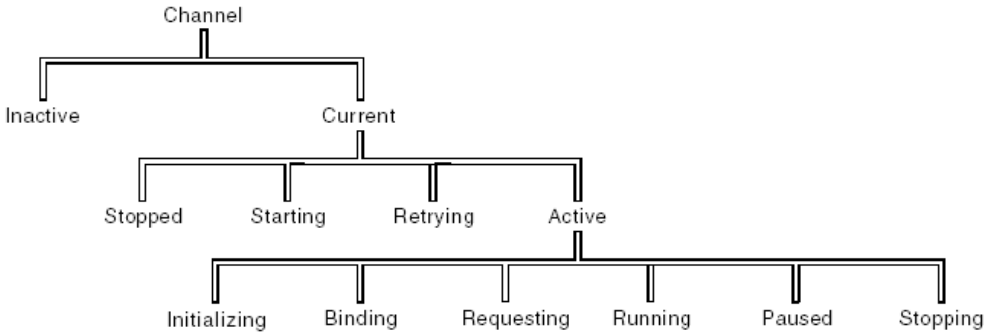
```
        INITQ('SYSTEM.CICS.INITIATION.QUEUE') +
        REPLACE

*echo "  定义传输队列"
DEFINE QLOCAL  ('QM0000') +
        DEFPSIST(YES) +
        MAXDEPTH(100000)  +
        MAXMSGL(1048576)  +
        USAGE(XMITQ)  +
        TRIGGER +
        TRIGTYPE(FIRST) +
        TRIGDATA('nnnn.0000') +
        INITQ('SYSTEM.CHANNEL.INITQ') +
        REPLACE

*echo "  修改 system.cics.initiation.queue 队列的属性"
ALTER QLOCAL  ('SYSTEM.CICS.INITIATION.QUEUE') +
        DESCR('WebSphere MQ Default CICS Initiation  queue') +
        DEFPSIST(YES) +
        MAXDEPTH(100000)  +
        MAXMSGL(1000)


*echo "  定义接收通道"
DEFINE CHANNEL  ('0000.nnnn') CHLTYPE(RCVR)  +
        TRPTYPE(TCP) +
        BATCHSZ(50)  +
        HBINT(300)  +
        MRRTY(10)  +
        MRTMR(1000) +
        REPLACE

*echo "  定义发送通道"
DEFINE CHANNEL  ('nnnn.0000') CHLTYPE(SDR) +
        TRPTYPE(TCP) +
        BATCHSZ(50)  +
        CONNAME('NPC')  +
        DISCINT(1800) +
        HBINT(300)  +
        LONGRTY(999999999)  +
        LONGTMR(300) +
        SHORTRTY(10000)  +
        SHORTTMR(30) +
        XMITQ('QM0000') +
```

```
        REPLACE

DEFINE CHANNEL  ('SYSTEM.ADMIN.SVRCONN')  CHLTYPE(SVRCONN)  +
        TRPTYPE(TCP) +
        HBINT(300)  +
        MAXMSGL(4194304)  +
        MCAUSER('mqm')  +
        REPLACE

*echo "   定义接收进程"
DEFINE PROCESS  ('CICS.PRO') REPLACE +
        APPLTYPE(CICS)  +
        APPLICID('MCTL')
```

6.3 通道的维护

6.3.1 通道的状态

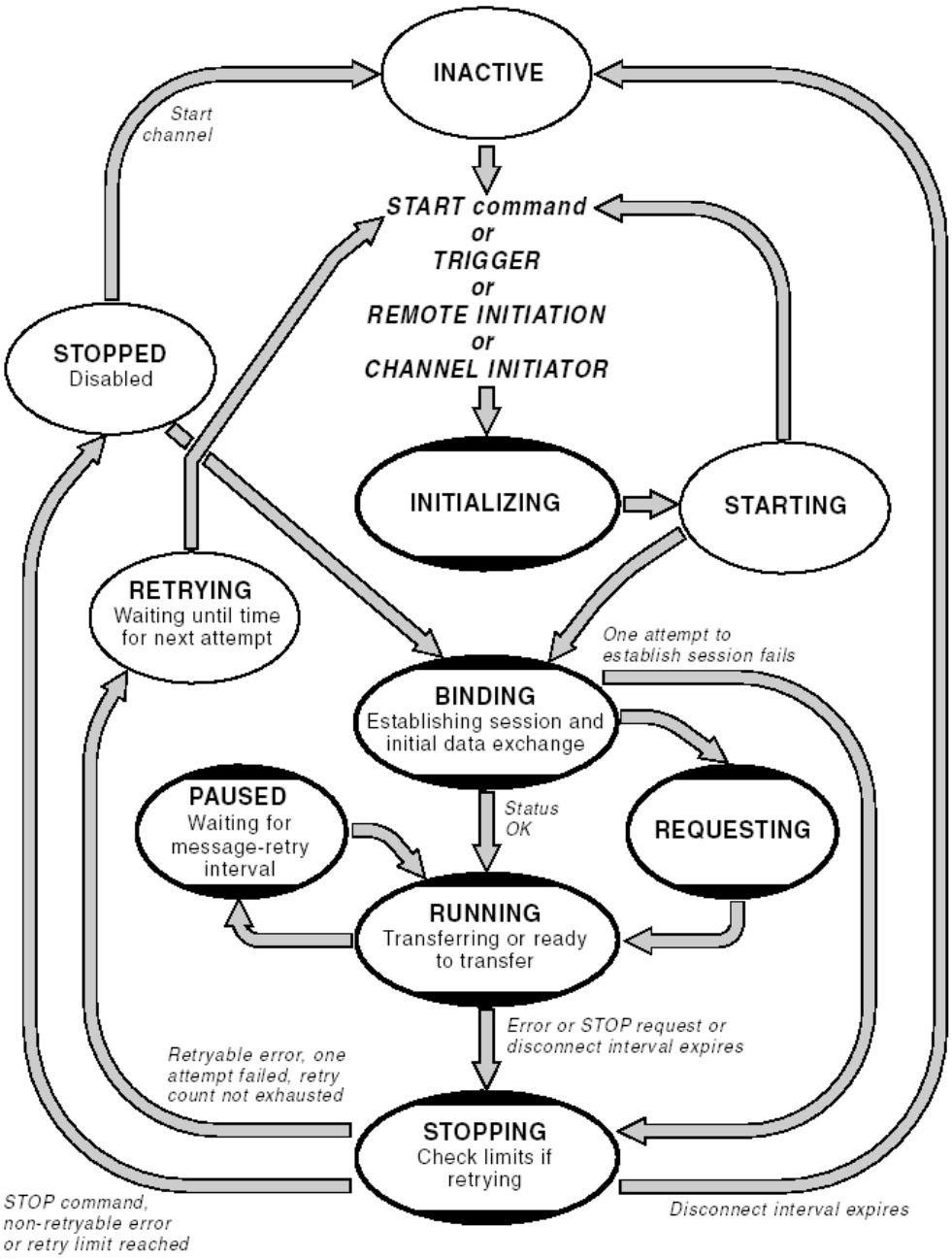
下图显示了所有可能的通道状态层次结构，在 WebSphere MQ for AIX, iSeries, HP-UX, Linux, Solaris, 和 Windows systems, 和 WebSphere MQ V5.1 for OS/2 Warp 平台，这些状态对服务器连接通道也适用。



图，通道状态

如果通道的状态分为 inactive 和 current 两大类；“current”可以是 Stopped,Starting,Retrying 和 Active 的情况。通道 Acitve 的情况又可分为 Initializing,Binding,Requesting,Running,Paused 或 Stopping 的状态。

下图显示通道状态之间的变化关系。



注意：

1. 当通道处于INITIALIZING, BINDING, REQUESTING, RUNNING, PAUSED, or或STOPPING状态时,这将消耗系统资源，并且通道的进程或线程正在运行；因为通道是Active状态。
2. 当通道是STOPPED状态时，会话可能是active，因为下一个状态时未知的。

6.3.2 通道维护命令

下面将详细地介绍通道有关的维护命令。

创建通道

为创建一个新通道，您需要创建两个通道定义，在通信的双方各定义一个。这两个通道的名字必须时相同的，而且两端的通道类型必须匹配，例如：发送和接收。可以使用 MQSC 命令“DEFINE CHANNEL”来创建通道，在命令中需要指定通道名，通道类型，连接名，通道描述（可选），传输队列名（可选）和传输协议，等还有许多可选的属性可以设置。

注意：

建议在 WebSphere MQ 的网络中所有的通道名唯一，并且通道名中最好包含了源队列管理器名和目标队列管理器名。

创建通道的例子

```
DEFINE CHANNEL(QM1.TO.QM2)  CHLTYPE(SDR)  +
DESCR(’Sender channel to QM2’) +
CONNAME(QM2)  TRPTYPE(TCP) XMITQ(QM2) CONVERT(YES)
```

I

修改通道

可以使用 MQSC 命令“ALTER CHANNEL”来修改现有通道定义，但是通道名和通道类型不能修改。

删除通道

可以使用 MQSC命令“DELETE CHANNEL”来删除现有通道定义。

查看通道定义

可以使用 MQSC命令“DISPLAY CHANNEL”来查看现有通道定义。可以说明通道名，通道类型（可选），和其它属性，或查看所有的属性。

查看通道定义的例子

```
DISPLAY CHANNEL(QM1.TO.QM2)  TRPTYPE,CONVERT
DISPLAY CHANNEL(QM1.TO.*)  TRPTYPE,CONVERT
DISPLAY CHANNEL(*)  TRPTYPE,CONVERT
DISPLAY CHANNEL(QM1.TO.QMR34)  ALL
```

查看通道状态

可以使用 MQSC命令“DISPLAY CHSTATUS”来查看现有通道状态。
显示的通道信息包括：

- 通道名
- 通信连接名
- 通道的 In-doubt 状态
- 上一个消息序号
- 传输队列名
- in-doubt 标识
- 上一个提交消息序号
- 逻辑工作单元标识
- 进程 ID
- 线程 ID (仅 OS/2 和 Windows 支持)

查看通道状态的例子

```
DISPLAY CHSTATUS(*) CURRENT
DISPLAY CHSTATUS(QM1.TO.*) SAVED
```

Ping 通道

使用 MQSC 命令“PING CHANNEL”用固定的数据消息来测试和远端的连接.ping 通道并没有使用传输队列和目标队列。它只是使用了通道定义、通讯链路和网络设置。只用通道当前状态不是 active 的情况下才使用它。这个命令只能在发送通道和服务器通道方使用。
命令返回的结果是“Ping complete”或错误消息。

启动通道

使用 MQSC 命令“START CHANNEL”启动发送通道、服务器通道和请求器通道。如果通道是采用触发方式启动，那么不用手工执行启动命令。当接收通道处在 disabled 的状态时，也可以使用“START CHANNEL”命令启动它。在 WebSphere MQ for AIX, iSeries, HP-UX, Linux, Solaris, 和 Windows systems, 和 WebSphere MQ V5.1 for Compaq Tru64 UNIX, 和 OS/2 Warp,如果服务器连接通道处在 disabled 状态，也可以使用“START CHANNEL”命令启动它。启动处在 disabled 状态的接收通道或服务器连接通道，即是复位通道和允许通道被远程启动。当通道被启动时，发送 MCA 读通道定义文件并打开传输队列，并远程启动相应的接收或服务器通道。

通道启动成功的条件：
1，本地和远端的通道定义必须存在。

2，传输队列必须存在，并且没有其它通道使用它。
3，本地和远程的 MCA 必须存在。
4，通讯链路必须可用。
5，本地和远程队列管理器必须是处在运行状态。
6，消息通道一定不在运行。

停止通道

使用 MQSC 命令“STOP CHANNEL”停止通道。停止通道的命令只能对除客户连接之外的通道进行操作。

停止通道的方式：

静态停止（Stop quiesce）

STOP CHANNEL(QM1.TO.QM2) MODE(QUIESCE)
这种方式将顺序地停止通道，必须完成当前的消息处理并确保事务的一致性。

注意：

如果通道是空闲的，将不终止接收通道。

强制停止（Stop force）

STOP CHANNEL(QM1.TO.QM2) MODE(FORCE)

这种方式立即停止通道，但不终止通道的线程或进程。通道并没有完成当前的消息处理，因此可能使通道处在可疑的状态。通常，推荐系统管理员使用静态停止通道。

终止停止（Stop terminate）

STOP CHANNEL(QM1.TO.QM2) MODE(TERMINATE)

这种方式立即停止通道，并终止通道的线程或进程。

复位通道

可以使用 MQSC 命令“RESET CHANNEL”改变消息序号。这个命令可以适用于任何消息通道，但不能用在 MQI 通道(客户连接或服务器连接)。通道被启动后，将使用新的消息序号。如果是对发送通道或服务器通道进行复位，当下次通道重新启动时将通知通道的另一方。

Resolve 通道

从发送通道或服务器通道使用 MQSC 命令“RESOLVE CHANNEL”来处理可疑的消息。“RESOLVE CHANNEL”命令可以接受 BACKOUT 或 COMMIT 参数，Backout 将把可疑消息恢复到传输队列，而 Commit 将丢弃可疑消息。

6.3.3 设置 MaxChannels 和 MaxActiveChannels 属性

MaxChannels 和 MaxActiveChannels 分别代表队列管理器允许配置的通道的最大个数和允许同时运行的通道的个数，MaxChannels 的缺省值是 100，MaxActiveChannels 的缺省值与 MaxChannels 相同。如果您的并发通道连接个数超过了 100，您需要修改这两个参数。这对于大并发的 Client/Server 间通讯尤为重要。

例如，在 unix 平台，修改 qm.ini 文件如下所示：

```
CHANNELS:
   MaxChannels = 200              ; Maximum number of Channels allowed.
   MaxActiveChannels  = 150    ; Maximum number of Channels allowed to be
                                  ; active at any time.
```

6.4 配置侦听程序

对于不同的平台或不同的通讯协议，侦听程序的配置也会存在着差别，下面将举例介绍在 Windows 和 unix 平台上针对 TCP/IP 协议的侦听程序的配置过程。

6.4.1 Windows 平台

为了运行 WebSphere MQ 提供的侦听程序，并把通道作为一个线程运行，则使用 RUNMQLSR 命令，例如：

```
RUNMQLSR  -t tcp [-m QMNAME]  [-p 1822]
```

方括号中的参数是可选的。如果使用缺省队列管理器，则不用说明 QMNAME；如果使用缺省端口号 1414，，则也不用设置端口号的参数。

您可以停止非活动的队列管理器的所有侦听程序，使用如下命令：

```
ENDMQLSR  [-m QMNAME]
```

如果命令中没有队列管理器名，则是指缺省队列管理器。通道能成功启动之前，必须要启动侦听程序。WebSphere MQ for Windows 可以自动地启动队列管理器、通道启动器、通道、侦听程序和命令服务器。可以使用 IBM WebSphere MQ Services snap-in 工具定义队列管理器的服务。

6.4.2 unix 平台

您可以使用 TCP/IP 侦听程序 (INETD) 或 WebSphere MQ 侦听程序。unix 平台的 WebSphere MQ 侦听程序和 Windows 平台的一致。

1，使用 TCP/IP 侦听程序
为了启动 unix 系统上的通道，需要编辑/etc/services 文件和 inetd.conf 文件。

- 编辑/etc/services 文件:

注意:
只有超级用户或 root 用户才能编辑/etc/services 文件。
在文件中增加如下一行：
WebSphere MQ 1414/tcp

其中 1414 表示侦听端口号，可以选择其它未使用的端口号。

- 编辑/etc/inetd.conf 文件:

在文件中增加如下一行：
WebSphere MQ stream tcp nowait mqm /*mqmtop*/bin/amqcrsta amqcrsta [-m Queue_Man_Name]

为了使修改的配置生效，需要用 root 用户执行如下命令：
On AIX:
refresh -s inetd

On other UNIX systems:
kill -1 <*process number*>

6.5 本章小结

本章主要介绍了 WebSphere MQ 互连通信的基本概念，例如，分布式组件，死信队列，和远程队列管理器通信。描述了怎样实现应用程序间的通信，把消息从本地队列管理器发送到远程队列管理器。为了确保 WebSphere MQ 互连通信的可靠性，需要能熟练维护消息通道，并掌握侦听程序的配置步骤。

6.6 本章练习

1.实现两个队列管理之间的数据通讯，下列哪一个可以没有？
(1)通信链路。
(2)MCA 之间的通讯协议。
(3)远程队列定义。
(4)传输队列。
答案：（3）
2.WebSphere MQ 的通道总是成对的。
（1）对 （2）错
答案：（2）
3.传输队列是一种需要定义的特殊队列类型。
（1）对 （2）错

答案：（2）

4.哪一个是合法的 MCA 通道类型，并实现从传输队列中发送消息？

(1)发送器通道（Sender）

(2)接收器通道（Receiver）

(3)请求器通道（Requester）

(4)服务器通道（Server）

答案：（1）（4）

5. 修改 MQSeries for AIX V5.3 的 qm.ini 文件中的 CHANNELS 节的属性，从 PipeLineLength=2 修改成 PipeLineLength=4,将会产生什么效果？

（1） 任何新定义的通道将会启动 4 个执行线程。

（2） MQSeries Version 5.2 系统中的所有通道将有 4 个执行线程。

（3） 非永久性消息的吞吐量将提高 2 倍。

（4） 没有影响。

答案：（4）

6. 以下哪一个通道出口可以在 WebSphere MQ 网络中用来加密消息？

（1） 安全出口（a security exit）

（2） 消息出口（a message exit）

（3） 消息重试出口（a message retry exit）

（4） 消息加密出口（a message encryption exit）

答案：（2）

7. 如果 SENDER 通道的 SEQWRAP 的属性值和 RECEIVER 通道值不相同，那么将会发生什么？

（1） 在通道启动时，以两个通道的 SEQWRAP 最小值启动。

（2） 通道将不启动。

（3） 在通道启动时，以两个通道的 SEQWRAP 最大值启动。

（4） 优先考虑 SENDER 通道的定义值。

（5） 没有任何变化，由 SENDER 通道进行控制，RECEIVER 通道不能说明 SEQWRAP 值。

答案：（2）

8. 练习实现两个队列管理器之间的双向通信。

第七章 WebSphere MQ 恢复和重新启动

目标

1. 了解 WebSphere MQ 的数据日志。
2. 学习在出现失败故障后，怎样恢复消息和 WebSphere MQ 对象。

7.1 WebSphere MQ 的数据日志

消息传递系统确保输入到系统的消息被发送到它们的目的地。这意味着它必须提供跟踪系统中消息和恢复消息的方法（如果系统发生故障）。

WebSphere MQ 通过记录队列管理器（处理消息的接收、传输和传递）的活动日志来确保消息不丢失。并提供了三种恢复方式：

1. *重启恢复*。
2. *崩溃恢复*。
3. *媒体恢复*。

在所有情况下，恢复是把队列管理恢复到它停止时的状态，（除了回滚所有执行中的事务）队列管理器停止时未提交的任何消息将被删除。恢复所有永久消息；非持久性消息将被丢失。

WebSphere MQ 把所有由队列管理器控制的数据的重要更改都记录到日志中。 这包括创建和删除对象（除了通道）、永久消息更新、事务状态、更改对象属性以及通道活动。

7.1.1 日志的概念

WebSphere MQ 日志包含两个组件：

1. 一个或多个日志文件
2. 日志控制文件

可以在队列管理器的配置文件中配置日志文件的大小和个数。在 WebSphere MQ Windows 版中，三个文件都缺省为 1 MB。在 WebSphere MQ UNIX 系统版中，三个文件都缺省为 4 MB。

创建队列管理器时，您可以指定*去*日志文件数和从日志文件数以及日志文件的大小。如果不指定数，则使用缺省值。

在 WebSphere MQ Windows 版中，如果未更改日志路径，则日志文件将存放在 C:\Program Files\IBM\WebSphere MQ\log\<QMgrName>目录下；在 WebSphere MQ UNIX 系统版中，如果未更改日志路径，则日志文件将存放在/var/mqm/log/QmName 目录下。

WebSphere MQ 在创建队列管理器时会自动创建所有的主日志文件，如果主日志用完了，则会动态地创建*次*日志文件。当 WebSphere MQ 不需要从日志空间时，也会动态地删除它们。

7.1.2 日志控制文件

日志控制文件包含监控日志文件使用状况的信息，如它们的大小和位置、下一个可用日志文件的名称等等。

注:

确保队列管理器启动时创建的日志足够大，使它能够满足应用程序处理的需要。

7.1.3 日志类型

在 WebSphere MQ 中，系统所需的日志文件数取决于日志文件大小以及接收到的消息数和消息长度。

日志记录的形式有两种：循环日志和线性日志。

循环日志记录

如果您的 WebSphere MQ 系统恢复使用“重启恢复”就可以满足要求，则可使用循环日志，当系统停止时，正在处理的事务通过日志进行回滚操作。

循环日志是把所有重启数据都记录在日志文件环中。首先使用第一个日志文件，然后下一个，依次类推，直到所有文件都满为止。然后它回到环中的第一个文件又重新开始。只要产品在使用中，就会一直循环下去，并且具有永远不会用完日志文件的优点。

线性日志记录

如果您的 WebSphere MQ 系统恢复需要使用“重启恢复”和“媒体恢复” （通过重播日志内容重新创建已丢失或已损坏的数据）才能满足要求，则使用线性日志。

线性日志记录在连续文件中保留日志数据。它不重用空间。由于磁盘空间是有限的，您可能需要考虑归档方式，来管理日志的磁盘空间，必要时重用或扩展现有的空间。

使用线性日志,则日志文件数可能非常大,这取决于您的消息流和您的队列管理器寿命。但是，有许多文件是*活动的*。活动文件包含重新启动队列管理器所需的日志信息。活动日志文件数通常与配置文件中定义的主日志文件数相同。

WebSphere MQ 检查点是一组日志记录，包含重新启动队列管理器成功的信息。重新启动队列管理器不需要的以前的任何记录信息，称为非活动的。

您必须确定何时不再需要非活动的日志文件。则可以压缩或者删除它们。

7.1.4 计算日志的大小

确定队列管理器应该使用循环还是线性日志后，您需要估计队列管理器需要的日志大小。日志大小是由以下日志配置参数确定的：

LogFilePages

每个主和次日志文件的大小是以 4K 为单位。

LogPrimaryFiles

预分配的主日志文件数

LogSecondaryFiles

主日志用完后，可创建的次日志文件数。

下表显示了队列管理器各种操作所消耗的日志数据量。大多数队列管理器操作只需要最少量的日志空间。 但是，当永久消息放入队列时，**所有**消息数据必须写入日志，以便它能够恢复此消息。通常，日志大小取决于队列管理器需要处理的永久消息的数量和大小。

表 日志开销大小（参考值）

操作	大小
放入永久消息	750 字节 + 消息长度 如果是大消息，则分隔成 15700 字节的段，每个段都会有 300 字节的开销。
取消息	260 字节
同步点，提交	750 字节
同步点，回滚	1000 字节 + 12 字节
创建对象	1500 字节
删除对象	300 字节
修改属性	1024 字节
记录媒体镜象	800 字节 + 镜象 镜象分隔成 260 000 字节的段，每个段有 300 字节开销。
检查点	750 字节 + 200 字节（每个活动的工作单元）

注:

1. 每次启动队列管理器时，您可以修改主和从日志文件数。
2. 日志文件大小只能在创建队列管理器**前**确定，以后不能修改。
3. 主日志文件的数量和日志文件大小决定了队列管理器创建时预分配的日志空间，这些空间应该是由少量的大文件组成，而不是由大量的小文件组成的。
4. 主日志文件和次日志文件的总数不能超过 63 个文件，这是针对循环日志而言， 线性日志则不受限制。
5. 当使用*循环*日志记录时，队列管理器重用主日志空间。主日志文件已满时，队列管理器将分配一个次日志文件（最多到限定值）。

7.2 使用数据日志进行恢复

下列几种情况可能会损坏您的数据：

- 数据对象被损坏
- 系统掉电
- 通信故障

但 WebSphere MQ 可以帮助您恢复被破坏的数据。本节将描述如何使用日志进行数据恢复。

7.2.1 从掉电或通信故障中恢复

WebSphere MQ 可以从通信故障和掉电中进行恢复。另外，它有时还可以从其它类型的问题中进行恢复，例如，文件被意外删除。 在通信故障情况下，消息一直保留在队列中直到被接收应用程序取出。如果消息正在传输中，则消息将一直被保留在传输队列中直到被成功传输。要从通信故障中恢复，通常您可以重新启动使用失败的链接的通道。

如果您掉电了，则重新启动队列管理器时，WebSphere MQ 把队列恢复到失败时的已提交的状态。确保不丢失持久消息。但非持久性消息被丢弃；当 WebSphere MQ 停止时会丢弃它们。

7.2.2 恢复受损对象

有些情况可以使 WebSphere MQ 对象成为不可用的，例如，由于无意中的损坏。您不得不恢复整个系统或部分系统。恢复操作与发现损坏的时间、日志是否支持媒体恢复和被损坏的对象有关。

7.2.2.3 媒体恢复

媒体恢复是从线性日志中的记录信息重新创建对象。例如，如果无意中删除了一个对象文件，则媒体恢复可以重新创建它。用于对象媒体恢复的日志信息被叫做*媒体映象*。*媒体映象*可以通过使用 **rcdmqimg** 命令手工记录或自动记录。 媒体映象中包含对象映象的日志记录序列。

重新创建对象所必需的第一个日志记录叫作*媒体恢复记录*;它是对象的最后一个媒体映象的开始。每个对象的媒体恢复记录是检查点期间记录的信息之一。

从媒体映象重新创建对象时，有必要重播任何描述自采用最后一个映象以来对对象执行更新的日志记录。

例如，考虑在持久消息放入到队列前，采用具有队列对象映象的本地队列。为了重新创建最新的对象映象，有必要重播记录消息放入队列的日志条目，并重播该映象本身。

创建对象时，已编写的日志记录包含完全重新创建此对象所需的足够的信息。这些记录组成对象的第一个媒体映象。接着，每次关机时，队列管理器记录媒体映象自动执行如下操作：

- 所有进程对象和队列的映象不是本地的。
- 空的本地队列的映象

还可以使用 **rcdmqimg** 命令手工记录媒体映象，在 rcdmqimg（记录媒体映象）中对它进行了描述。此命令编写 WebSphere MQ 对象的媒体映象。一旦这样做了，仅保持媒体映象的日志和此时后创建的所有日志需要重新创建受损的对象。这样做的好处取决于某些因素，如可用的空闲存储器量和创建日志文件时的速度。

7.2.2.4 恢复媒体映象

如果发现某些对象被损坏，则 WebSphere MQ 自动从它们的媒体映象中进行恢复。特别地，这适用于正常队列管理器启动期间所找到的受损对象。如果队列管理器最后一次关机时有任何未完成的事务，则也会自动恢复任何受影响的队列，以便完成启动操作。

您必须使用 **rcrmqobj** 命令手工恢复其它对象，它会重播日志中的记录以重新创建 WebSphere MQ 对象。从日志中找到的最新映象中重新创建此对象，并带有保存此映象和发出重新创建命令期间所有可用的日志事件。如果 WebSphere MQ 对象受损，则仅可执行的有效操作是使用此方法删除或重新创建它。 不能用此方法恢复非持久性消息。

请参阅 rcrmqobj（重新创建对象）以获得 **rcrmqobj** 命令的进一步详细信息。

包含媒体恢复记录的日志文件和所有后继的日志文件，在尝试对象的媒体恢复时必须在日志文件中可用。如果找不到必需的文件，发出操作程序消息 AMQ6767，并且媒体恢复操作失败。如果您不采用要重新创建的对象的常规媒体映象，则可能没有足够的磁盘空间来保持重新创建对象所必需的所有日志文件。

7.2.2.5 启动期间恢复受损的对象

如果队列管理器启动期间发现受损的对象，则所使用的操作取决于对象类型和队列管理器是否配置成支持媒体恢复。

如果队列管理器对象受损，则该队列管理器无法启动，除非它可以恢复该对象。如果队列管理器配置成使用线性日志，并支持媒体恢复，则 WebSphere MQ 自动尝试从其媒体映象中重新创建此队列管理器对象。如果所选的日志方法不支持媒体恢复，则可以恢复队列管理器的备份或删除该队列管理器。

如果队列管理器停止时任何事务都是活动的，则包含持久的、未提交的消息（在这些事务中放入或取出）的本地队列还需要成功启动该队列管理器。如果发现这些本地队列中的任何一个受损，并且队列管理器支持媒体支持，它自动尝试从其媒体映象中重新创建它们。如果无法恢复任何队列，则 WebSphere MQ 无法启动。

如果启动不支持媒体恢复的队列管理器的处理期间，发现任何包含未提交的消息的受损本地队列，则这些队列标记为受损对象，并且忽略它们上面的未提交的消息。这是因为不可能执行这样的队列管理器上的受损对象的媒体恢复，并且仅留下的操作是删除它们。发出 AMQ7472 消息报告任何损坏。

7.2.2.6 在其它时间恢复受损的对象

仅在启动期间对象的媒体恢复是自动的。在其它时间，检测到受损对象时，发出操作程序消息 AMQ7472，并且使用此对象的大多数操作失败。如果启动队列管理器后的任何时间该队列管理器对象受损，则该队列管理器执行抢先关机。队列管理器受损后您可以删除它，或者如果该队列管理器使用线性日志，则尝试使用 **rcrmqobj** 命令（请参阅 rcrmqobj（重新创建对象）以获得进一步的详细信息）从其媒体映象中恢复它。

7.3 保护 WebSphere MQ 日志文件

WebSphere MQ 队列管理器运行时不要手工除去这些日志文件。如果用户无意删除了队列管理器需要重新启动的那些日志文件，则 WebSphere MQ 不发出任何出错消息，并且继续处理*包含持久消息*的数据。队列管理器正常关机，但是无法重新启动。则消息的媒体恢复就不可能了。
具有除去活动队列管理器使用的日志权限的用户也具有删除其它重要队列管理器资源（如队列文件、对象目录和 WebSphere MQ 可执行文件）的权限。因此，他们能够以针对 WebSphere MQ 无法保护其本身的途径损坏正在运行的或处于睡眠状态的队列管理器（可能是由于缺乏经验所致）。 所以授予超级用户或 mqm 权限时要谨慎。

7.4 备份和恢复 WebSphere MQ

您可能要定期备份您的队列管理器数据以保护由硬件故障导致的可能的破坏。但是，由于消息数据通常是短期的，您可以选择不进行备份。

7.4.1 备份 WebSphere MQ

要备份队列管理器的数据：

1. 确保队列管理器不在运行。如果您尝试备份正在运行的队列管理器，备份可能会不一致，因为文件复制时正在进行更新。

如果可能，以正常方法停止您的队列管理器。尝试执行 **endmqm -w**（等待关机）；仅当其失败时，使用 **endmqm -i**（立即关机）。

2. 使用配置文件中的信息，查找队列管理器放置其数据和日志文件的目录。
3. 备份所有队列管理器的数据和日志文件目录，包括所有子目录。

确保没有丢失任何文件，特别是日志控制文件和配置文件。某些目录可以为空，但是以后恢复备份时全部需要，因此也要保存它们。

4. 保留文件的权限。对于 WebSphere MQ UNIX 系统版，可以用 **tar** 命令完成。

7.4.2 恢复 WebSphere MQ

要恢复队列管理器的数据备份：

1. 确保队列管理器不在运行。
2. 查找队列管理器放置其数据和日志文件的目录。此信息保持在配置文件中。
3. 清除您要放置备份数据的目录。
4. 把备份的队列管理器数据和日志文件复制到正确的位置。

检查结果目录结构，确保您有所有必需的目录。
确保您有日志控制文件和日志文件。还请检查 WebSphere MQ 和队列管理器配置文件是是否一致，以便 WebSphere MQ 可以在正确的位置查看恢复的数据。
如果正确备份和恢复了数据，则将启动队列管理器。

注:

即使队列管理器数据和日志文件保持在不同的目录，但必须是在相同时间备份的。如果队列管理器数据和日志文件的备份时间不同，则队列管理器无效，并且可能会不启动。如果启动，您的数据很可能被损坏。

7.5 恢复方案

本节针对许多可能出现的问题并讲解如何进行恢复。

7.5.1 磁盘故障

您可能会遇到存放队列管理器数据或/和日志的磁盘出故障、数据被丢失或数据被破坏的情况。
在*所有*情况下，首先检查任何受损的目录结构，若有必要，修复它。如果您丢失队列管理器数据，则队列管理器目录结构可能已受损。如果这样，重新启动该队列管理器前手工重新创建该目录树。
检查受损的结构后，您可以做许多事，这取决于您所使用的日志类型。

- **目录结构的主要受损处或日志的任何受损处在哪里**，全部除去旧文件返回到 QMgrName 级别，包括配置文件、日志和队列管理器目录，恢复最近的备份并重新启动队列管理器。
- **对于具有媒体恢复的线性日志记录**，确保该目录结构是完整的，并且重新启动该队列管理器。如果队列管理器不重新启动，则恢复备份。如果队列管理器重新启动，则使用 MQSC 命令（如 DISPLAY QUEUE）检查是否已损坏了任何其它对象。使用 **rcrmqobj** 命令恢复您找到的损坏。例如：

  rcrmqobj -m QMgrName -t all *

  其中 QMgrName 是要恢复的队列管理器。-t all * 表明要恢复任何类型的所有对象（除了通道外）。如果只有一个或两个对象报告受损了，则您可以在此处按名称和类型指定那些对象。
- **对于具有媒体恢复和未受损的日志的线性日志记录**，您可以恢复队列管理器数据的备份，保留现有日志文件和未更改的日志控制文件。启动队列管理器会应用日志的更改，把队列管理器置回故障发生时的状态。

该方法基于两个因素：

1. 您必须把检查点文件恢复为队列管理器数据部分。此文件包含一些信息，这些信息确定必须应用日志中的多少数据才能给出一致的队列管理器。
2. 备份时，您必须具有启动队列管理器所必需的最旧的日志文件和此日志文件目录中所有的后继日志文件。

如果没有，则恢复队列管理器和日志的备份，它们是在同一时间备份的。

- **对于循环日志记录**，从您具有的最近备份中恢复队列管理器。一旦您恢复了备份，重新启动队列管理器并如上所述检查受损的对象。但是，由于您没有媒体恢复，因此，必须找到重新创建受损对象的其它方法。

7.5.2 受损的队列管理器对象

如果正常操作期间报告了队列管理器对象受损，则该队列管理器执行抢先关机。根据您使用的日志记录类型，这些情况下有两种恢复方法：

- **仅对于线性日志记录**，手工删除包含受损对象的文件，并重新启动该队列管理器。（您可以使用 **dspmqfls** 命令确定受损对象的真实的文件系统名。受损对象的媒体恢复是自动的。
- **对于循环或线性日志记录**，恢复队列管理器数据和日志的最近备份，并重新启动此队列管理器。

7.5.3 受损的单个对象

如果正常操作期间报告单个对象受损：

- **对于线性记录日志**，从其媒体映象中重新创建此对象。
- **对于循环日志记录**，不支持重新创建单个对象。

7.5.4 自动媒体恢复故障

如果带有线性日志的队列管理器启动所必需的本地队列受损了，并且自动媒体恢复失败，则恢复该队列管理器数据和日志的最近备份并重新启动该队列管理器。

7.6 使用 dmpmqlog 命令转储日志

使用 **dmpmqlog** 命令转储队列管理器日志的内容。缺省情况下，转储所有活动的日志记录，即，该命令从日志头开始转储（通常从最近完成的检查点开始）。

通常仅当队列管理器不运行时才能转储日志。由于队列管理器在停止时会采用检查点，因此日志的活动部分通常包含少量的日志记录。然而，您可以设置以下选项之一使用 **dmpmqlog** 命令转储更多日志记录：

- 从日志*基*开始转储。这些日志基是日志文件中包含日志头的第一个日志记录。该情况中的其它转储数据量取决于日志头在日志文件中的位置。如果它靠近日志文件的开头，则仅转储小量的其它数据量。如果头在日志文件的结束部分，则转储更多重要的数据。
- 指定转储的开始位置作为个别的日志记录。每个日志记录是由*日志序列号（LSN）*标识的。在循环日志记录情况中，开始的日志记录不能在日志基之前；此限制不适用于线性日志。您可能需要在运行命令之前恢复非活动的日志文件。必须指定一个有效的 LSN，从前一个 **dmpmqlog** 输出中获取它以作为开始位置。

例如，使用线性日志记录时，您可以从最近的 dmpmqlog 输出指定 nextlsn。nextlsn 在日志文件头中出现，并表明要编写的下一个日志记录的 LSN。使用它作为开始位置，对所有自最近一次转储日志以来所编写的所有日志记录进行格式化。

- **仅对于线性日志**，您可以说明 **dmpmqlog** 从任何给定日志文件范围开始格式化日志记录。在这种情况下，**dmpmqlog** 期望在与活动日志相同的目录中查找此日志文件和每个后继的日志文件。此选项不适用于循环日志，其中 **dmpmqlog** 无法访问日志基之前的日志记录。

dmpmqlog 命令的输出是**日志文件头**和一系列已格式化的日志记录。队列管理器使用几个日志记录记录对其数据的更改。
某些已格式化的信息仅用于内部使用。以下表包括最有用的日志记录：

日志文件头

每个日志有单个日志文件头，它总是由 **dmpmqlog** 命令格式化的第一项。它包含以下字段：

logactive	主日志范围数。
loginactive	次日志范围数。
logsize	每个范围有 4 KB 页面数。
baselsn	包含日志头的日志范围中的第一个 LSN。
nextlsn	要编写的下一个日志记录的 LSN。
headlsn	日志头部分的日志记录的 LSN。
tailsn	LSN 标识日志的末尾位置。
hflag1	日志是 CIRCULAR 还是 LOG RETAIN（线性）。
HeadExtentID	包含日志头的日志范围。

日志记录头

日志中的每个日志记录有一个包含以下信息的固定头：

LSN	日志序列号。
LogRecdType	日志记录的类型。
XTranid	与该日志记录相关联的事务标识（如果有的话）。 MQI 的 TranType 表明仅 WebSphere MQ 事务。XA 的 TranType 涉及其它资源管理器。同一个工作单元中所涉及的更新具有相同的 XTranid。
QueueName	与该日志记录相关联的队列（如果有的话）。
Qid	队列的唯一内部标识。
PrevLSN	同一个事务中的前一个日志记录的 LSN（如果有的话）。

启动队列管理器

队列管理器已启动的日志。

StartDate	队列管理器已启动的日志。
StartTime	队列管理器已启动的时间。

停止队列管理器

队列管理器已停止的日志。

StopDate	队列管理器停止日志。
StopTime	队列管理器停止时间。

ForceFlag	使用的关机类型。

启动检查点

这表示队列管理器检查点的启动。

结束检查点

这表示队列管理器检查点的结束。

ChkPtLSN	启动此检查点的日志记录的 LSN。

放入消息

持久消息放入队列的日志。如果消息放到同步点下，则日志记录头包含非空的 *XTranid*。其余记录包含：

SpcIndex	队列上的消息标识。它可用于与相应的 MQGET（用于从队列中取出此消息）相匹配。在这种情况下，可以发现后继的 Get Message 日志记录包含相同的 QueueName 和 SpcIndex。在此点上，可以为到该队列的后继放入消息重用 SpcIndex 标识。
Data	包含在十六进制转储中的日志记录的数据是各种内部数据，后跟消息描述符（eyecatcher MD）和消息数据本身。

放入部分

对于单个日志记录来说太大的持久消息记录为单个***放入消息***记录，后跟多个***放入部分***日志记录。

Data	在前一个日志记录关闭处继续消息数据。

取出消息

仅记录取出的持久消息。如果在同步点下取出消息，则日志记录头包含非空的 *XTranid*。其余记录包含：

SpcIndex	标识从队列中检索的消息。包含相同的 QueueName 和 SpcIndex 的最近的**放入消息**日志记录标识已检索的消息。
QPriority	已从队列中检索的消息的优先级。

启动事务

表明新事务的启动。MQI 的 TranType 表明仅 WebSphere MQ 事务。XA 的 TranType 表明涉及其它资源管理器的事务类型。由该事务所做的所有更新将具有相同的 *XTranid*。

准备事务

表明队列管理器已准备好提交与指定的 *XTranid* 相关联的更新。此日志记录作为涉及其它资源管理器的两阶段提交部分而编写。

提交事务

表明队列管理器已由事务提交了所有更新。

回滚事务

这表示队列管理器的目的是回滚事务。

结束事务

这表示已回滚的事务结束。

事务表

此记录是在同步期间编写的。它记录已进行持久更新的每个事务的状态。对于每个事务，记录以下信息：

XTranid	事务标识。

FirstLSN	与该事务相关联的第一个日志记录的 LSN。
LastLSN	与该事务相关联的最后一个日志记录的 LSN。

事务参与者

此日志记录由队列管理器的 XA 事务管理器组件编写。它记录参与事务的外部资源管理器。对于每个参与者，记录以下信息：

RMName	资源管理器名。
RMID	资源管理器标识。它还记录到后继的**已准备的事务**日志记录中，该日志记录记录了资源管理器所参与的全局事务。
SwitchFile	此资源管理器的切换装入文件。
XAOpenString	此资源管理器的 XA 打开字符串。
XACloseString	此资源管理器的 XA 关闭字符串。

已准备的事务

此日志记录由队列管理器的 XA 事务管理器组件编写。它表明已成功准备好指定的全局事务。将说明每个参与资源管理器的事务进行提交。日志记录中记录每个已准备的资源管理器的 *RMID*。如果队列管理器本身参与到事务中，则将显示带有 O 的 *RMID* 的*参与者条目*。

事务忘记

此日志记录由队列管理器的 XA 事务管理器组件编写。当提交决定已发送到每个参与者时，它跟在*已准备的事务*日志记录后面。

清除队列

它记录已清除队列上的所有消息这一事实，例如，使用 MQSC 命令 CLEAR QUEUE。

队列属性

它记录队列属性的初始化或更改。

创建对象

它记录 WebSphere MQ 对象的创建。

ObjName	已创建的对象的名称。
UserId	执行此创建的用户标识。

删除对象

它记录 WebSphere MQ 对象的删除。

ObjName	已删除的对象的名称。

7.7 本章小结

记录数据日志是队列管理器的一个重要部分。数据日志是顺序地记录在日志文件中，队列管理器的配置文件中说明了日志参数，缺省的日志参数则是在 WebSphere MQ 配置文件中说明的。数据日志参数包括主日志和从日志数、日志类型和日志目录的路径。确保有足够的日志空间，否则如果磁盘空间不足，将导致对队列管理器的操作失败。

主日志文件是在队列管理器创建时分配的，并且随后它的个数保持不变。从日志仅在主日志用完以后才开始分配。

7.8 本章练习

1.下列那些队列管理器有记录数据日志的功能？

（1） WebSphere MQ for MVS/ESA
（2） WebSphere MQ for OS/2 Warp
（3） WebSphere MQ for HP-UX
（4） WebSphere MQ for AS/400
（5） WebSphere MQ for AIX

答案：（1）（2）（3）（4）（5）

2， 在 AIX 平台上，WebSphere MQ 支持的两种类型是：

（1） journaling
（2） linear
（3） circular
（4） checkpointing

答案：（2）（3）

第八章 WebSphere MQ 问题诊断

目标

1. 描述在那儿能找到能确定问题的消息日志。
2. 学习启动和停止 WebSphere MQ 的跟踪功能。

8.1 错误日志

WebSphere MQ 使用许多错误日志来捕捉 WebSphere MQ 自身的操作、任何队列管理器的启动和正在使用的通道的错误信息。

错误日志的位置取决于队列管理器名，以及错误是否与客户机相关。

在 WebSphere MQ Windows 版中，假设 WebSphere MQ 已经安装在缺省位置中：

- 如果队列管理器名称是已知的，则错误日志位于：

c:\Program Files\IBM\WebSphere MQ\qmgrs*qmname*\errors

- 如果队列管理器不是已知的，则错误日志位于：

c:\Program Files\IBM\WebSphere MQ\qmgrs\@SYSTEM\errors

- 如果错误发生在客户机应用程序，则错误日志位于客户机的根目录中：

c:\Program Files\IBM\WebSphere MQ Client\errors

在 WebSphere MQ Windows 版中，错误信息也被添加到应用程序日志中，所以可以通过查看 Windows 系统的事件查看器来检查 WebSphere MQ 的错误日志。

在 WebSphere MQ UNIX 系统版中：

- 如果队列管理器名称是已知的并且队列管理器是可用的，则错误日志位于：

/var/mqm/qmgrs/*qmname*/errors

- 如果队列管理器不是可用的，则错误日志位于：

/var/mqm/qmgrs/@SYSTEM/errors

- 如果错误发生于客户机应用程序，则错误日志位于客户机的根目录中：

/var/mqm/errors

8.1.1 日志文件

在产品安装时，在 qmgrs 目录下将创建@SYSTEM errors 子目录。errors 子目录最多可以包含 3 个错误日志文件，分别是：

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

在创建队列管理器后，在需要时将创建了 3 个错误日志文件。这些文件名是，AMQERR01、AMQERR02 和 AMQERR03 并且每一个文件的大小都为 256 KB。这些文件被放置在您创建的队列管理器的 errors 子目录中。

当产生错误消息时，它们被放置在 AMQERR01 中。当 AMQERR01 文件比 256 KB 大时，将其复制成 AMQERR02。在复制前，将 AMQERR02 复制到 AMQERR03.LOG。这样将删除了 AMQERR03 的以前内容。

因此最新的错误消息总是放在 AMQERR01 中的，其它文件用来保存错误消息的历史记录。

所有与通道相关的信息也被放在相应的队列管理器的错误文件中，除非队列管理起步可用或队列管理器的名称未知，则通道相关的消息是放在 @SYSTEM 错误子目录。

使用通常的系统编辑器就可以查看错误日志文件的内容。

有些错误是在错误日志还没有创建时发生的，WebSphere MQ 也会尝试记录这样的错误日志。日志的位置取决于创建队列管理器的过程进展情况。

如果配置文件被损坏，WebSphere MQ 不能读取目录信息，则将错误记录到在安装时创建的根（/var/mqm 或 C:\Program Files\IBM\WebSphere MQ）的 errors 目录中。

如果 WebSphere MQ 可以读取配置信息，并且可以访问 Default Prefix 的值，则错误记录在由 Default Prefix 属性标识的目录的 errors 子目录中。例如，如果缺省前缀为 C:\Program Files\IBM\WebSphere MQ，则错误在 C:\Program Files\IBM\WebSphere MQ\errors 中记录。

8.1.2 忽略 WebSphere MQ for Windows 的错误代码

如果您要忽略 WebSphere MQ Windows 的错误代码，则编辑 Windows 注册表。

注册表键是：
HKEY_LOCAL_MACHINE\Software\IBM\MQSeries\CurrentVersion\IgnoredErrorCodes

它的值是由 NULL 字符分隔的字符串数组，例如，如果您要 WebSphere MQ 忽略错误代码 AMQ3045、AMQ6055 和 AMQ8079，则将值设置为：
AMQ3045\0AMQ6055\0AMQ8079\0\0

您对配置文件所做的任何修改，只有在下一次启动队列管理器时才能生效。

8.1.3 操作信息

操作信息是指一般的错误信息，通常由用户在命令上使用无效的参数等类似操作时直接引起。这些信息被写到相关的窗口中。另外，一些操作信息被写到队列管理器目录中的 AMQERR01.LOG 文件的，而其它是写到错误日志的 @SYSTEM 目录副本中。

8.2 死信队列

出于某种原因无法发送的消息都被放置在死信队列。您可以通过 MQSC 命令 DISPLAY QUEUE 来检查队列是否包含消息。如果队列包含消息，则使用所提供的浏览样本应用程序（amqsbcg）来浏览队列上的消息。样本应用程序将显示每个消息的消息描述符和消息上下文字段。您应该通过分析消息的死信头来确定消息被放在死信队列的原因。

8.3 配置文件和问题确定

配置文件错误通常找不到队列管理器，和导致*队列管理器不可用*。确保配置文件存在，并且 WebSphere MQ 配置文件必须和队列管理器和日志目录对应。 在 Windows 注册表中的错误是在启动队列管理器时，通过消息通知的。

8.4 跟踪

本节描述了如何产生 WebSphere MQ 跟踪信息。

8.4.1WebSphere MQ Windows 的跟踪

在 WebSphere MQ Windows 版中，您可以使用 **strmqtrc** 控制命令启用或修改跟踪；使用 **endmqtrc** 控制命令停止跟踪。您还可以使用 WebSphere MQ 服务管理单元启动和停止跟

踪。

8.4.1.1 跟踪的选项

使用 -t 和 -x 选项控制跟踪信息量的详细程度。缺省情况下，启用**所有**跟踪信息。-x 选项指定**不**需要跟踪的信息。

例如，如果您仅跟踪队列管理器 QM1 在通信网络上流动的数据，则使用：

```
strmqtrc -m QM1 -x all -t comms
```

8.4.1.2 跟踪文件

在安装过程期间，您可以选择跟踪文件的存放路径。跟踪文件一般放置在目录 \<mqmwork>\errors 中，其中 <mqmwork> 是 WebSphere MQ 数据文件的安装目录。

跟踪文件名的格式如下：

AMQ*ppppp*.TRC

其中 *ppppp* 是产生跟踪的进程的进程标识（PID）。

注:

1. 进程标识号的数字位数不是固定的。
2. 每个被跟踪的进程都有一个跟踪文件。

8.4.1.3 跟踪数据的示例

下图显示了 WebSphere MQ Windows 版的跟踪数据：

```
Process : C:\Program Files\IBM\WebSphere MQ\bin\amqxssvn.exe
Version : 530 Level : p000-L020213
Date : 02/25/02    Time : 16:35:47

Counter    TimeStamp          Process.Thread      Data
============================================================

0000062F 16:35:47.348386 6278.1          --{   InitProcessInitialisation
00000630 16:35:47.348455 6278.1          ---{   xcsCreateNTSecurityAtts
00000631 16:35:47.348516 6278.1          ----{   xcsRequestThreadMutexSem
00000632 16:35:47.348583 6278.1          ----}   xcsRequestThreadMutexSem (rc=OK)
00000633 16:35:47.348639 6278.1          ----{   xcsInitGlobalSecurityData
00000634 16:35:47.349111 6278.1          ----}   xcsInitGlobalSecurityData (rc=OK)
00000635 16:35:47.349239 6278.1          ----{   xcsReleaseThreadMutexSem
00000636 16:35:47.349261 6278.1          ----}   xcsReleaseThreadMutexSem (rc=OK)
00000637 16:35:47.349275 6278.1          ---}   xcsCreateNTSecurityAtts (rc=OK)
00000638 16:35:47.349303 6278.1          ---{   xcsReleaseThreadMutexSem
```

```
00000639 16:35:47.349319 6278.1          ---}    xcsReleaseThreadMutexSem (rc=OK)
0000063A 16:35:47.349344 6278.1          --}    InitProcessInitialisation (rc=OK)
0000063B 16:35:47.349359 6278.1          --{    xcsCreateThreadMutexSem
0000063C 16:35:47.349395 6278.1          --}    xcsCreateThreadMutexSem (rc=OK)
0000063D 16:35:47.349872 6278.1          --{    xcsProgramInit
0000063E 16:35:47.349900 6278.1          --}    xcsProgramInit (rc=OK)
0000063F 16:35:47.350027 6278.1          --{    xcsInitialize
00000640 16:35:47.350048 6278.1          ---{    xcsRequestThreadMutexSem
00000641 16:35:47.350065 6278.1          ---}    xcsRequestThreadMutexSem (rc=OK)
00000642 16:35:47.350079 6278.1          ---{    xihCheckThreadList
00000643 16:35:47.350101 6278.1          ---}    xihCheckThreadList (rc=OK)
00000644 16:35:47.350115 6278.1          ---{    InitPrivateServices
00000645 16:35:47.350165 6278.1          attributes 32768
00000646 16:35:47.350204 6278.1          ----{    xcsCreateThreadMutexSem
00000647 16:35:47.350233 6278.1          ----}    xcsCreateThreadMutexSem (rc=OK)
00000648 16:35:47.350255 6278.1          pid MQ(6) system(6278)
00000649 16:35:47.350337 6278.1          ---}    InitPrivateServices (rc=OK)
0000064A 16:35:47.350360 6278.1          --{    xxxInitialize
0000064B 16:35:47.350977 6278.1          ---{    xcsGetMem
```

8.4.2WebSphere MQ AIX 的跟踪

WebSphere MQ AIX 使用 AIX 系统标准跟踪。跟踪分为两步：

1. 采集数据 。
2. 格式化结果数据。

WebSphere MQ 使用两个跟踪 hook 标识：X'30D' 和 X'30E' 。

跟踪提供了执行跟踪的详细信息来帮助您分析问题。跟踪产生的文件可能**非常**大，所以合理地设置跟踪。例如，您可以通过时间和组件来限定跟踪。
有两种运行跟踪的方法：

1. 交互地。

以下命令是对程序 `myprog` 运行了交互式跟踪并结束跟踪。

```
trace -j30D,30E -o trace.file
->!myprog
->q
```

2. 异步地。

以下命令对程序 myprog 运行了异步跟踪和结束跟踪。

```
trace -a -j30D,30E -o trace.file
myprog
trcstop
```

您可以用以下命令格式化跟踪文件：

```
trcrpt -t /usr/mqm/lib/amqtrc.fmt trace.file > report.file
```

report.file 是存放格式化的跟踪输出的文件名。

注:

当跟踪是活动的，将跟踪所有的 WebSphere MQ 活动。

8.4.2.1 跟踪选项

可使用环境变量 MQS_TRACE_OPTIONS 来分别激活高级详细信息和参数跟踪的功能。下表定义了 MQS_TRACE_OPTIONS 的各种配置的跟踪行为。

表，MQS_TRACE_OPTIONS 设置

MQS_TRACE_OPTIONS 值	跟踪信息
取消设置（缺省值）	**缺省跟踪（除高级详细信息之外的所有跟踪）**
0	**没有 WebSphere MQ 跟踪**
262148	**入口，出口和参数跟踪**
786436	**入口，出口、参数和高级详细信息跟踪**
4980740	**入口，出口、参数、高级详细信息和 SSL 跟踪**
3407871	**不带参数跟踪的缺省跟踪**
3670015	**缺省跟踪，包含参数跟踪**
7864319	**缺省跟踪，包含参数跟踪和 SSL 跟踪**
4194303	**所有跟踪，包含高级详细信息跟踪**

注:

1. 最好需要在技术支持人员的指导下，设置 MQS_TRACE_OPTIONS 环境变量。
2. 通常在启动队列管理器之前设置 MQS_TRACE_OPTIONS。
3. 在跟踪开始前设置 MQS_TRACE_OPTIONS。

8.4.2.2 SSL 跟踪

如果您请求 SSL 跟踪，请注意以下内容：

- SSL 跟踪是写到目录 /var/mqm/trace 的。
- SSL 跟踪文件是 AMQ.SSL.TRC 和 AMQ.SSL.TRC.1。

- 您无法格式化 SSL 跟踪文件；将它们原封不动地发给 IBM 技术支持中心。

8.4.2.3 跟踪数据的示例

下图显示了 WebSphere MQ AIX 跟踪的数据：

```
ID        ELAPSED_SEC        DELTA_MSEC      APPL          SYSCALL   KERNEL
INTERRUPT

30D       0.000000000        0.000000        MQS FNC Entry. 71540.1 zcpSendOnPipe
30E       0.000000038        0.000038        Msg Unencumbered (T/F)(0)
30D       0.000000176        0.000138        MQS FNC Exit..... 51604.55
                                             aqhCheckMsgUnencumbered rc=00000000
30E       0.000000418        0.000242        aqhCheckMsgChains : internal retcode
                                                  208007d3

30D       0.000000516        0.000098        MQS FNC Entry.. 71540.14 xcsWaitEventSem
30E       0.000000590        0.000074        MessageSent (24 bytes)
30E       0.000000847        0.000257        aqhCheckMsgChains : internal retcode
                                                  208007d3

30E       0.000000936        0.000089        hev=1::0::0-307724 TimeOut(-1)
30E       0.000001173        0.000237        aqhCheckMsgChains : internal retcode
                                                  208007d3

30D       0.000001313        0.000140        MQS FNC Entry..... 51604.55
                                             aqtIdxToSpcFn
30D       0.000001395        0.000082        MQS FNC Exit..... 51604.55
                                             aqtIdxToSpcFn rc=00000000
30D       0.000001439        0.000044        MQS FNC Entry..... 36124.51
                                             xcsCheckProcess
30D       0.000001501        0.000062        MQS FNC Entry..... 51604.55
                                             aqhCheckMsgUnencumbered
30E       0.000001645        0.000144        MQS Data from zcpSendOnPipe Length=0018
                                                  5A525354 000007E5 00000000 00000000
                                                  ZRST 00000000 00000000
30E       0.000001765        0.000120        pBCrsr (0)
30D       0.000001907        0.000142        MQS FNC Entry..... 51604.55
                                             aqhInTrans
30D       0.000001997        0.000090        MQS FNC Exit..... 51604.55
                                             aqhInTrans rc=00000000
30D       0.000002025        0.000028        MQS FNC Entry.. 71540.1 xcsResetEventSem
30E       0.000002243        0.000218        Msg Unencumbered (T/F)(0)
```

```
30D      0.000002363      0.000120    MQS FNC Exit..... 51604.55
                                      aqhCheckMsgUnencumbered rc=00000000
30E      0.000002392      0.000029    hev=1::0::0-305876
30D      0.000002522      0.000130    MQS FNC Entry... 71540.14 xlsLockEvent
30E      0.000002630      0.000108    aqhCheckMsgChains : internal retcode
                                          208007d3
```

8.5 首次故障支持技术（FFST）

本节描述了 WebSphere MQ 的首次故障支持技术（FFST）的角色。

8.5.1FFST: WebSphere MQ Windows 版

在 WebSphere MQ Windows 版中，FFST 信息是存放在 c:\Program Files\IBM\WebSphere MQ\errors 目录下的文件中。这些错误通常都是严重的、不可恢复的错误，要么是系统的配置问题或 WebSphere MQ 内部错误。

FFST 文件命名为 AMQ*nnnnn.mm*.FDC，

其中：*nnnnn* 是报告错误进程的标识

mm 是顺序号，通常为 0

当进程创建了 FFST 记录时，它还将记录发送到事件日志（应用程序级别）中。记录包含 FFST 文件的名称来辅助自动问题跟踪。下图显示了典型的 FFST 日志。

```
+-------------------------------------------------------------------------+
|                                                                         |
| WebSphere MQ First Failure Symptom Report                               |
| =========================================                               |
|                                                                         |
| Date/Time         :- Tue February 19 12:58:42 GMT Standard Time 2002    |
| Host Name         :- NETTLE (NT Version 4.0 Build 1381: Service Pack 6) |
| PIDS              :- 5724B4101                                          |
| LVLS              :- 530                                                |
| Product Long Name :- WebSphere MQ for Windows                           |
| Vendor            :- IBM                                                |
| Probe Id          :- XC371019                                           |
| Application Name  :- MQM                                                |
| Component         :- xstServerRequest                                   |
| Build Date        :- Feb 13 2002                                        |
| CMVC level        :- p000-L020213                                       |
| Build Type        :- IKAP - (Production)                                |
| UserID            :- nigel                                              |
| Process Name      :- C:\Program Files\IBM\WebSphere MQ\bin\amqzdmaa.exe |
| Process           :- 00001678                                           |
```

```
| Thread            :- 00000001                                                |
| QueueManager      :- REGR                                                    |
| Major Errorcode   :- xecF_E_UNEXPECTED_SYSTEM_RC                             |
| Minor Errorcode   :- OK                                                      |
| Probe Type        :- MSGAMQ6119                                              |
| Probe Severity    :- 2                                                       |
| Probe Description :- AMQ6119: An internal WebSphere MQ error has occurred   |
|    (WinNT error 5 from WaitForSingleObject.)                                 |
| FDCSequenceNumber :- 0                                                       |
| Comment1          :- WinNT error 5 from WaitForSingleObject.                 |
|                                                                              |
| Comment2          :- Access is denied.                                       |
|                                                                              |
|                                                                              |
+------------------------------------------------------------------------------+

MQM Function Stack
amqzdmaa.main
xcsTerminate
xcsDisconnectSharedSubpool
xcsDettachSharedSubpool
xcsGetSetConnectCount
xstGetExtentConnectCount
xstStorageRequest
xstServerRequest
xcsFFST

MQM Trace History
------} zcpSendOnPipe rc=OK
------{ zcpReceiveOnPipe
-------{ xcsWaitEventSem

...
```

IBM 使用函数堆栈和跟踪历史来辅助定位问题。在大多数情况下，当生成 FFST 记录时，系统管理员基本上不能自己解决问题，而需要寻求 IBM 支持中心的帮助。

8.5.2FFST: WebSphere MQ UNIX 系统版

对于 WebSphere MQ UNIX 系统版，FFST 信息是存放在 /var/mqm/errors 目录的文件中。这些错误通常都是严重的、不可恢复的错误。要么时系统的配置问题或 WebSphere MQ 内部错误。

文件命名为 AMQ*nnnnn.mm*.FDC，其中：

nnnnn 是报告错误进程的标识
mm 是顺序号，通常为 0

当进程创建了 FFST 记录时，它也将一条记录发送到系统日志（用户级）中。记录包含 FFST 文件名用来辅助自动问题跟踪，下图显示一些典型的 FFST 数据。

```
+------------------------------------------------------------------------+
|                                                                        |
| WebSphere MQ First Failure Symptom Report                              |
| =========================================                              |
|                                                                        |
| Date/Time         :- Friday March 15 17:56:51 SGT 2002                 |
| Host Name         :- sunrts3 (SunOS 5.7)                               |
| PIDS              :- 5724B4102                                         |
| LVLS              :- 530                                               |
| Product Long Name :- WebSphere MQ for Sun Solaris                      |
| Vendor            :- IBM                                               |
| Probe Id          :- RM161000                                          |
| Application Name  :- MQM                                               |
| Component         :- rrmChangeClq                                      |
| Build Date        :- Mar 13 2002                                       |
| CMVC level        :- p000-L020312                                      |
| Build Type        :- IKAP - (Production)                               |
| UserID            :- 00001001 (mqm)                                    |
| Program Name      :- amqrrmfa                                          |
| Process           :- 00019454                                          |
| Thread            :- 00000001                                          |
| QueueManager      :- REGR                                              |
| Major Errorcode   :- rrcE_CLUS_COMMAND_ERROR                           |
| Minor Errorcode   :- OK                                                |
| Probe Type        :- MSGAMQ9413                                        |
| Probe Severity    :- 2                                                 |
| Probe Description :- AMQ9413: Repository command format error, command code |
|   0                                                                    |
| FDCSequenceNumber :- 0                                                 |
|                                                                        |
+------------------------------------------------------------------------+

MQM Function Stack
rrmProcessMsg
rrmChangeClq
```

```
xcsFFST

MQM Trace History
---{ zstVerifyPCD
---} zstVerifyPCD rc=OK
---{ ziiMQCMIT
----{ ziiCreateIPCCMessage
-----{ zcpCreateMessage
-----} zcpCreateMessage rc=OK
----} ziiCreateIPCCMessage rc=OK
----{ ziiSendReceiveAgent
-----{ zcpSendOnPipe

...
```

IBM 使用函数堆栈和跟踪历史来辅助定位问题。在大多数情况下，当生成 FFST 记录时，系统管理员基本上不能自己解决问题，而需要寻求 IBM 支持中心的帮助。

然而，有些问题是系统管理员可以解决的。如果 FFST 显示当调用 IPC 函数（例如，**semop** 或 **shmget**）时*资源用尽*或*空间用尽*，则可能是已经超出了相关的内核参数极限。

如果 FFST 报告显示问题 **setitimer**，则可能需要修改内核定时器参数。 要解决这些问题，增加 IPC 限制、重新建立内核并重新启动机器。

8.6 本章小结

8.7 本章练习

1. 如果队列管理器出现了故障，那么错误信息将记录在下列那个文件中？
   （1） QMGRERR
   （2） ERRORS
   （3） amqerr01.log
   （4） AMQERR.001
   答案：（3）
2. 当队列管理器出现故障重新启动后，非永久性消息能够得到恢复。
   （1） 对
   （2） 错
   答案：（2）

第三部分 Websphere MQ 应用开发

第九章 设计 Websphere MQ 应用程序

目标

1. 介绍消息和队列的概念。
2. 描述使用 WebSphere MQ 提供的服务怎样设计和编写应用程序。

9.1 介绍应用设计

9.1.1 规划设计

首先需要明确操作系统的平台和环境，针对 WebSphere MQ，需要考虑以下因素：

1．队列类型

是使用永久队列、动态队列还是别名队列。

2．消息类型

是使用数据报消息还是请求消息，在一些消息中是否设置不同的优先级。

3．应用程序是否在 WebSphere MQ Client 运行？

在 WebSphere MQ Server 和 WebSphere MQ Client 运行程序所链接的库是不一样的，运行在 WebSphere MQ Client 的程序需要链接 MQIC 库；而运行在 WebSphere MQ Server 的程序需要链接 MQI 库。

注意：运行在 WebSphere MQ Client 的应用程序可以同时连接多个队列管理器。但是运行在 WebSphere MQ Server 的应用程序不能同时连接多个队列管理器

4．数据的安全性和完整性

可以使用上下文信息来验证数据的安全性，使用同步点机制来确保数据的一致性。也可以使用消息的永久性特征来确保重要数据的传递。

5．怎样处理意外和错误

需要考虑怎样处理不能交付的消息，以及怎样处理队列管理器产生的报告消息。

9.1.2 WebSphere MQ 对象

MQI 可以使用以下对象：

- 队列管理器
- 队列
- 名字列表
- 进程定义

- 通道
- 存储类（仅 WebSphere MQ for Z/OS 支持）
- 授权信息对象 （AUTHINFO objects）

除动态队列之外，其他对象都需要在定义之后使用。定义的方法有 PCF，MQSC，这些对象被定义之后可以被查看，修改和删除。

9.1.3 设计消息

当使用 MQI 接口 PUT 消息到队列中时，将产生一个消息。该消息有控制信息（Message Description）和应用数据组成，在应用程序中使用消息时，需要考虑以下因素：

- 消息类型

  在应用程序中的消息是一个简单消息，还是一个请求消息。如果是请求消息，请求-回复的处理方式是异步还是同步。还有就是所有的消息是否在同一个工作单元。

- 消息优先级

  在发送应用程序中可以为每个消息设置优先级再放到队列中。如果队列的消息交付方式也设置为优先级方式，则接收程序将总是先取出优先级最高的消息。如果队列的消息交付方式也设置为先进先出方式，则接收程序将按先进先出方式取出队列中的消息。

- 消息的永久性

  当队列管理器重新启动，是否希望保留队列中的消息，如果需要保留，则把消息设置成永久性的，如果不需要保留，则把消息设置成非永久性的。

9.1.4 WebSphere MQ 技术

如果仅编写一个简单的 WebSphere MQ 应用程序，只需要确定在应用程序中使用的 WebSphere MQ 对象和消息类型，对于编写更高级的应用程序，可能会使用如下技术：

- 等待消息

可以采用轮循机制从队列中取出消息或设置等待消息时间，如果消息到达则取出，否则超时则返回。

- 关联回复

把请求消息中的消息标识（MsgId）复制到回复消息的关联标识（CorrelID）中。

- 上下文信息（Context information）

上下文信息对于安全，审计和问题确定是很有用。

- 自动启动 WebSphere MQ 应用程序

WebSphere MQ 触发机制，当消息到达队列时，自动启动应用程序处理消息。

- 产生消息报告

  在应用程序可以请求产生多种报告消息，例如，意外报告（Exception reports）、失效报告(Expiry reports)、到达确认报告(Confirm-on-arrival reports)、交付确认报告(Confirm-on-delivery reports)、PAN 报告（Positive action notification reports）和 NAN 报告(Negative action notification reports)等。

- 群集和消息的紧密联系

  在 WebSphere MQ 群集里，消息可以被放到群集中任何队列管理器的相应队列，因此消

息间的紧密联系可能被变得松散。

9.1.5 应用编程

WebSphere MQ 支持 IBM MQI(Message Queue Interface)和 AMI（Application Messaging Interface）。MQI 包括一些发送和接收消息以及操作 WebSphere MQ 对象的 API 调用。AMI 是一个比 MQI 更简单的调用接口。

调用接口

MQI 接口可以实现以下功能：

- 连接和断开队列管理器。
- 打开和关闭对象（例如，队列、队列管理器和名字列表和进程）
- 放消息到队列中。
- 从队列中接收消息或浏览消息。
- 查询 WebSphere MQ 对象的属性，并可以设置队列的某些属性。
- 提交和回滚事务。
- 协调队列管理器与别的资源管理器的更新。

MQI 接口提供了放消息和取消息的结构。也提供了许多常量。

应用程序的性能

- 使应用程序的处理尽量并行操作。
- MQCONN/MQDISC 是最耗 CPU 的两个函数，其次是 MQOPEN 和 MQCLOSE 这两个函数，因此要尽量避免必要地重复使用这几个函数。比如，当您需要从队列中读取多条消息时，正确的编程方法应该如下：

```
MQCONN
MQOPEN
MQGET
  .
  .
  .
MQGET
MQCLOSE
MQDISC
```

即：连接/断开队列管理器一次，打开/关闭队列一次，读取消息多次。而不应该反复建立与队列管理器的连接和反复进行队列打开/关闭操作。

- 如果应用程序放一条消息到队列中，则应该使用 MQPUT1 函数。

- 当处理一批消息时，可以采用 MQCMIT 函数，将若干消息作为一个完整的交易来处理，消息将作为一个 batch 统一提交，而不是一个个地分别提交，因此，可以提高性能。尤其对于永久性的消息效果更加明显。
- 尽量减小消息的大小，小消息的读取效率要高。对于 mqget, mqput 这两个函数而言，8k 以下的消息的耗时差别不大，8k 到 128k 的消息的耗时随着消息大小的增加而增加。大于 128k 的消息耗时较大，因为当与队列相关的内存满了的时候，会有硬盘交换。
- 同时要注意，从传输效率而言，如果在广域网上进行消息传输，消息太小会影响传输效率，因为对于每一消息，MQ 都会有一个消息头，它会占有一定的字节数，如果把消息拆分太小，每个消息的传输头都会占据一定的开销。
- 如果消息不必可恢复，则在应用程序中可使用非永久性消息。
- 使用 Distribution List 方式来把相同的消息发往不同的目的地。
- 用 match correlation ID 的方法取消息比不匹配性能要差。
- 通常，我们使用 MQCONN 这个函数建立与队列管理器的连接，除此之外，MQ 支持 trusted application binding，即 fastpath binding，用 MQCONNX 来实现。当从性能方面考虑时，我们可以使用 MQCONNX 来提高性能。

9.1.6 测试应用程序

WebSphere MQ 应用程序的开发环境和其他应用程序的一样。因此您可以使用和 WebSphere MQ trace 工具一样的开发工具。

9.2 WebSphere MQ 消息

9.2.1 消息描述符

通过使用 MQMD 结构，可以访问消息中的控制信息。关于 MQMD 结构的更详细的描述，请参看《WebSphere MQ Application Programming Reference》。

9.2.2 消息种类

WebSphere MQ 定义了四种类型的消息：
- 数据报消息
- 请求消息
- 回复消息
- 报告消息

应用程序可以使用前三种消息来实现它们之间的信息交换。第四种，报告消息是应用程序和队列管理器用来报告关于例如错误发生的事件。
每种消息类型都是使用 MQMT_*来定义的。您也可以自定义消息类型。

9.2.3 消息控制信息和消息数据的格式

队列管理器仅关心消息控制信息格式，而处理消息的程序则既关心消息的控制信息，也关心消息数据格式。

9.2.3.1 消息控制信息格式

在消息描述的character-string字段中的控制信息必须是在队列管理器的CodedCharSetId属性值定义的字符集。因为当应用程序把消息从一个队列管理器发送到另一个队列管理器时，传输消息的消息通道代理需要使用这个属性值来确定是否需要对消息进行数据转换。

9.2.3.2 消息数据格式

在应用程序中可以定义应用数据格式、字符数据的字符集和数字数据的格式。为设置这些格式需要使用下列字段：

- Format
  这个字段向消息的接收者说明了消息中应用数据的格式。

- CodedCharSetId
  这个字段表示了消息中的字符数据的字符集。
- Encoding
  这个字段描述了数字消息数据的格式。

9.2.4 消息优先级

当应用程序放消息到队列中时，您可以设置消息的优先级（在 MQMD 结构的 Priority 字段设置）。
队列的 MsgDeliverySequence 属性决定了队列中的消息是以先进先出方式存放，还是以优先级内先进先出方式存放。如果队列的这个属性设置成 MQMDS_PRIORITY,队列中的消息将以消息描述符的 Priority 字段的优先级进行排队。但如果队列的这个属性设置成 MQMDS_FIFO,队列中的消息将以消队列的缺省优先级进行排队，相同优先级消息的存放次序是取决于到达的次序。

当消息放到队列中时，如果没有设置消息优先级，那么将会自动使用队列的 DefPriority 属性定义的缺省优先级。在 WebSphere MQ 系统中，可以对消息设置 0（最低）至 9（最高）的 10 类优先级。

9.2.5 消息组

一个消息组是由一个或多个逻辑消息组成的。一个逻辑消息是由一个或多个物理消息组成。

- 组

  每个组是通过 GroupId 来标识的。它是由一个或多个包含同样 GroupId 的消息组成。这些消息可以存放在队列中的任何位置。

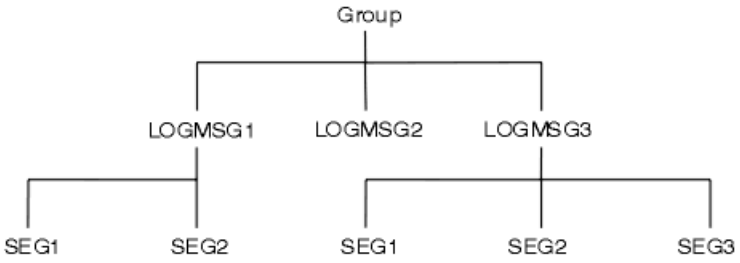
图，一组逻辑消息

- 逻辑消息

在组里的逻辑消息是通过 GroupId 和 MsgSeqNumber 来标识的。在一个组里的第一个消息的 MsgSeqNumber 值是从 1 开始。如果组里的逻辑消息没有被分成段，则组里的逻辑消息是由一个物理消息组成。

- 段（Segment）

  段是用来处理对于应用程序或队列管理器来说太大消息。消息中的段是由 GroupId，MsgSeqNumber 和 Offset 来标识的。每个段是由一个物理消息组成。



图，分段消息

9.2.6 消息持久性

永久消息需要写到日志和队列文件中。当队列管理器失败后重新启动时，它将会从日志数据中恢复所需要的永久消息，如果队列管理器停止，非永久性消息将被丢弃。

当您创建消息时，如果使用缺省值初始化消息描述符（MQMD）。消息的永久性是由 MQOPEN 的队列的 DefPersistence 所决定的。您也可以使用 MQMD 中的 Persistence 字段来设置消息的永久性。

如果使用永久性消息将会影响应用程序的性能。影响的程度由系统 I/O 子系统的特性和

使用的同步点选项所决定的。

9.2.7 检索消息

为从队列中获得一个特殊消息，您需要使用消息描述符中的 MsgId 和 CorrelId 字段。如

果使用了版本 2 的 MQMD 结构，还需要使用 GroupId 字段。
当消息被放到队列中时，队列管理器将产生消息描述符。队列管理器试图确保消息描述符是唯一的，然而 WebSphere MQ 应用程序也能设置消息描述符中的值。

9.2.8 交付失败的消息

当队列管理器不能把消息放到队列时，可以使用下列处理办法：

- 再次把消息放到队列中。
- 把消息返回到发送方。
- 把消息放到死信队列中。

9.3 本章小结

介绍 WebSphere MQ 应用程序设计的要点和怎样编写以及测试应用程序。在设计应用程序过程中的一个重要环节就是设计合适的 WebSphere MQ 消息。

9.4 本章练习

1. 列出 WebSphere MQ 应用程序的设计步骤。
2. 列出 WebSphere MQ 消息的设计种类。

第十章 用 MQI 编程

目标

1.学习使用 C 语言环境下的 WebSphere MQ 编程。
2.掌握 MQI API 的调用和参数的使用。

10.1 概述

消息队列接口或 MQI 是一种编程接口，它向程序员提供 WebSphere MQ 消息发布平台

的所有便利，并使程序员可对消息和消息流动方式进行全面的、细节上的控制。通过 MQI 调用，您可以：

- 把程序连接至队列管理器，或断开程序与队列管理器的连接
- 打开和关闭对象（如队列、队列管理器、名单和线程）
- 将消息放入队列
- 从队列接收消息并浏览消息（仍将消息保存在队列上）
- 查询 WebSphere MQ 对象的属性，并设置队列的某些属性
- 在不具备自然同步点支持的环境中（如 OS/2 和 UNIX 系统），提交和取消在一个工作单元中所做的改变
- 协调队列管理器和其它资源管理器所做的更新

MQI 提供一系列调用或函数来进行这些操作，还可提供各种数据结构和类型用作调用的输入和输出。同时，MQI 提供大量的指定常量，可用来修改这些数据结构中的选择项。数据结构初始化时，可以使用 API 以指定常量形式提供的默认值。MQI 在所有的支持平台上具有一致性，因此应用程序毋需修改代码即可移植到不同的平台上使用，尽管这可能要求程序员添加一些逻辑以保证其可移植性，例如：

```
#ifdef _OS2
OS/2 specific code
#else
generic code
#endif
```

MQI 程序在服务器或客户端环境中都能够运行。由于在客户端环境中到队列管理器的不能以绑定方式连接，因此程序在客户端环境中运行有着一定的限制。另外，在客户端配置中，我们还需要某些环境变量，帮助应用程序找到 WebSphere MQ 服务器。

10.2 平台和语言

以下列出了 MQI 支持的平台和语言：

- WebSphere MQ for MVS/ESA：

- COBOL
- 汇编语言
- C
- PL/I

- WebSphere MQ for AS/400：

- RPG
- COBOL
- C
- C＋＋

- WebSphere MQ for AT&T GIS UNIX，WebSphere MQ for Digital OpenVMS，WebSphere MQ for SINIX 和 DC/OSx 以及WebSphere MQ for SunOS：

- COBOL
- C

- WebSphere MQ for HP－UX 和WebSphere MQ for Sun Solaris：

- COBOL

- C
- C＋＋

- WebSphere MQ for AIX，WebSphere MQ for OS/2 Warp 以及WebSphere MQ for Windows NT：

- COBOL
- C
- C＋＋
- PL/I

- WebSphere MQ for Tandem NonStop Kernel：

- COBOL
- C
- TAL

- WebSphere MQ for Windows：

- C
- Visual Basic

10.3 库和存根模块程序

以下列出了 MQI 支持的库和存根模块程序。

- **WebSphere MQ for Windows**

在 WebSphere MQ for Windows 中，您必须将您的程序链接到应用程序所运行的环境所提供的 MQI 库文件，此外还要链接到那些由操作系统提供的库文件：

MQM16.LIB 使用 16 位 C 的服务器
MQM.LIB 使用 32 位 C 的服务器
MQM16.LIB 使用 16 位 Visual Basic 的服务器
MQMSTD.LIB 使用 32 位 Visual Basic 的服务器

- **WebSphere MQ for Windows NT**

在 WebSphere MQ for Windows NT 中，您必须将您的程序链接到应用程序所运行的环境所提供的 MQI 库文件，此外还要链接到那些由操作系统提供的库文件：

MQM.LIB 使用 32 位 C 的服务器
MQIC.LIB 使用 16 位 C 的客户机
MQIC32.LIB 使用 32 位 C 的客户机
MQMXA.LIB C 的静态 XA 接口
MQMCICS.LIB C 的 CICS for Windows NT 版本 2
MQMCICS4.LIB CICS Transaction Server for Windows NT 版本 4 出口
MQMZF.LIB C 的可安装服务出口
MQMCBB.LIB 使用 32 位 IBM COBOL 的服务器
MQMCB32 使用 32 位 Micro Focus COBOL 的服务器
MQICCBB.LIB 使用 32 位 IBM COBOL 的客户机
MQICCB32 使用 32 位 Micro Focus COBOL 的客户机
MQMENC.LIB 在 C 中用于 Encina 的动态 XA 接口
MQMTUX.LIB 在 C 中用于 Tuxedo 的动态 XA 接口

- **WebSphere MQ for AIX**

在 WebSphere MQ for AIX 中，您必须将您的程序链接到应用程序所运行的环境所提供的

MQI 库文件，此外还要链接到那些由操作系统提供的库文件：
在非线程的应用程序中：
libmqm.a 使用 C 的服务器
libmqic.a 使用 C 的客户机
libmqmzf.a C 的可安装服务出口
libmqmxa.a C 的 XA 接口
libmqmcbrt.o 用于 Micro Focus COBOL 支持的 WebSphere MQ 运行时间库
libmqmcb.a 使用 COBOL 的服务器
libmqicb.a 使用 COBOL 的客户机
在线程的应用程序中：
libmqm_r.a 使用 C 的服务器
libmqmzf_r.a C 的可安装服务出口
libmqmxa_r.a C 的 XA 接口
libmqmxa_r.a 用于 Encina

- **WebSphere MQ for HP－UX**

在 WebSphere MQ for HP－UX 中，您必须将您的程序链接到应用程序所运行的环境所提供的 MQI 库文件，此外还要链接到那些由操作系统提供的库文件：
在非线程的应用程序中：
libmqm.sl 使用 C 的服务器
libmqic.sl 使用 C 的客户机
libmqmzf.sl C 的可安装服务出口
libmqmxa.sl C 的 XA 接口
libmqmcbrt.o 用于 Micro Focus COBOL 支持的 WebSphere MQ 运行时间库
libmqmcb.sl 使用 COBOL 的服务器
libmqicb.sl 使用 COBOL 的客户
在线程的应用程序中：
libmqm_r.sl 使用 C 的服务器
libmqmzf_r.sl C 的可安装服务出口
libmqmxa_r.sl C 的 XA 接口

- **WebSphere MQ for Sun Solaris**

在 WebSphere MQ for Sun Solaris 中，您必须将您的程序链接到应用程序所运行的环境所提供的 MQI 库文件，此外还要链接到那些由操作系统提供的库文件。列举如下：
libmqm.so 使用 C 的服务器
libmqmzse.so 用于 C
libmqic.so 使用 C 的客户机
libmqmcs.so 使用 C 的客户机
libmqmzf.so C 的可安装服务出口
libmqmxa.a C 的 XA 接口

10.4 体系结构模型

MQI 体系结构是 WebSphere MQ 不同特点的简单而直接的实施。不同的调用直接访问某一基本的 WebSphere MQ 操作，例如从队列中获取消息或将消息放入队列。我们可以根据这些调用的不同作用进行分组：

- 连接和断开连接一个队列管理器：

MQCONN，MQCONNX 和 MQDISC

- 打开和关闭 WebSphere MQ 对象（例如队列）：

MQOPEN 和 MQCLOSE

- 将一个或多个消息放入队列：

MQPUT 和 MQPUT1

- 从队列中浏览消息或删除消息：

MQGET

- 查询对象属性：

MQINQ

- 运行时间内设定某些队列属性：

MQSET

- 管理局部或分布式的事务处理：

MQBEGIN，MQCMIT 和 MQBACK

我们利用 API 所提供的数据结构和基本数据类型，可以提供操作所需的不同选择项和基本信息。以下就是 MQI 的数据结构：

- MQBO （开始选项）

为 MQBEGIN 调用确定选择项（仅适用于 WebSphere MQ 版本 5 产品）。

- MQCNO （连接选项）

为 MQCONNX 调用确定选择项（仅适用于 WebSphere MQ 版本 5 产品）。

- MQDH （分配标题）

如果传输队列中的一条消息是分布列表消息，则描述该消息所包含的数据（仅适用于 WebSphere MQ 版本 5 产品和 WebSphere MQ for AS/400）。

- MQGMO（获取消息选项）

为 MQGET 调用确定选择项。

- MQMD（消息描述器）

为放入队列的（使用 MQPUT 或 MQPUT1）或从队列中获取的（使用 MQGET）消息提供控制信息。

- MQMDE （消息描述器扩展）

与 MQMD 版本 1 结合，它包含 MQMD 版本 2 通常采用的分组消息和分段信息（仅适用于 WebSphere MQ 版本 5 产品和 WebSphere MQ for AS/400）。

- MQOD（对象描述器）

确定采用 MQOPEN 时要处理的对象。

- MQOR（对象记录）

确定您在分布列表中要处理的目标(仅适用于WebSphere MQ 版本5 产品和WebSphere MQ for AS/400 V4R2）。

- MQPMO（放置消息选项）

确定 MQPUT 和 MQPUT1 调用的选择项。

- MQPMR（放置消息记录）

包含相关分布列表中个别目标的特定信息（仅适用于 WebSphere MQ 版本 5 产品和 WebSphere MQ for AS/400 V4R2）。

下述结构用于特殊目的：

- MQDLH（死信标题）

定义放入死信（未送达的消息）队列中消息标题的格式（WebSphere MQ for Windows V2.0 不支持）。

- MQRMH （引用消息标题）

定义引用消息的格式(仅适用于 WebSphere MQ 版本 5 产品和 WebSphere MQ for AS/400)。

- MQTM（触发器消息）

定义触发器消息格式。

- MQTMC （触发器消息）

定义作为一组字符字段的触发器消息的格式（仅适用于 WebSphere MQ for AS/400）

- MQTMC2 （触发器消息）

定义包括队列管理器名的触发器消息的格式（仅适用于 WebSphere MQ for MVS/ESA，WebSphere MQ on UNIX systems，WebSphere MQ for OS/2 Warp 和 WebSphere MQ for Windows NT）

- MQXP（出口参数块）结构

用来与 API 交叉出口进行通讯（仅适用于 WebSphere MQ for MVS/ESA）。

- MQXQH（传输队列标题）

定义放入传输队列中的添加至消息的标题格式。

对 C 和 Visual Basic，MQI 提供以下基本数据类型：

- **MQBYTE** 单字节数据
- **MQBYTEn** 16、24、32 或 64 字节的字符串
- **MQCHAR** 单字节字符
- **MQCHARn** 包含 4，8，12，16，20，28，32，48，64，128 或 256 个单字节字符的字符串
- **MQHCONN** 连接句柄（此数据为 32 位）
- **MQHOBJ** 对象句柄（此数据为 32 位）
- **MQLONG** 32 位带符号二进制整数
- **PMQLONG** 指向 MQLONG 类型数据的指针

10.5 用 MQI 编程

MQI 是一组使得程序员可以发送和接收消息的函数和数据类型。一般而言，MQI 程序采用如下简单的操作流程。

1. 程序首先调用 MQCONN 连接到队列管理器。
2. 一旦连接成功建立，就可以调用 MQOPEN 打开一个或多个对象。
3. 可对每个对象进行任何次数的操作(如 GET 或 PUT 操作)，直到不需要该对象为止。
4. 然后调用 MQCLOSE 关闭该对象。
5. 调用 MQDISC 断开与队列管理器的连接。

我们首先来看看所有调用的一些相同元素，然后我们将讨论连接或断开队列管理器所需的调用，并探讨如何打开并关闭 WebSphere MQ 对象。

我们完成上述工作之后，还将讨论一下 MQI 所能完成的四项基本操作：

- 将消息放入队列中
- 从队列中获取消息

- 在队列上浏览消息
- 查询和设定对象属性

10.5.1 基本 API 概念

下面我们将讨论 MQI API 的基本概念。

1，所有调用的公共参数

所有的调用都具有两种类型的参数：

- 句柄

句柄由队列管理器连接返回，打开队列调用，用作后续调用的输入参数。

- 返回码

所有调用都具有两种返回码：完成代码和原因代码。

- 完成码确定调用是成功（MQCC_OK）还是失败（MQCC_FAILED）。它也可以返回一个警告（MQCC_WARNING）。

- 如果完成码是 MQCC_OK，那么原因代码就是 MQCC_NONE。如果完成码不是 MQCC_OK，那么就会返回某个其它的值，说明完成码报告警告或失败。

2，按顺序 GET 消息

我们可以根据物理顺序或逻辑顺序对队列中的消息进行扫描。如果对消息所在的队列发出 GET 或 BROWSE 请求的话，就会使用这种排序。

物理排序与队列接收消息的方式有关。首先到达队列的消息就是获取或浏览操作时的第一个消息。对队列上的消息，物理排序给我们提供了一个 FIFO（先进先出）或优先级序列先进先出的顺序。

如果采用优先级内先进先出的排序方法的话，那么即使优先级较高的消息在优先级较低的消息之后到达，它也会在队列内先出现。

这样，在获取或浏览队列上的消息时，就会产生一些被忽略的消息。这是因为：一旦一个优先级较低的消息已经到达而另一个优先级较高的消息又出现在队列中，那么该消息就会被置于当前消息（指针）之前，因此只有关闭并重新打开队列，才能看到该消息，或者调用一个 MQOGET 并选中 MQOO_BROWSE_FIRST 选项。

让我们以下面发送到优先级顺序队列的消息为例来说明：

1. 消息一，优先级一
2. 消息二，优先级二
3. 消息三，优先级一

我们假设应用程序正在浏览该队列上的消息。程序首先以下面的顺序见到消息：

1. 消息二，优先级二
2. 消息一，优先级一（浏览光标位于此处）
3. 消息三，优先级一

如果程序的浏览光标指向消息一的时候，又出现了优先级为二的消息四，那么队列的顺序将会是：

1. 消息二，优先级二

2. 消息四，优先级二
3. 消息一，优先级一（浏览光标位于此处）
4. 消息三，优先级一
在这种情况下，程序只有重新打开队列，或者利用 MQOO_BROWSE_FIRST 选项将浏览光标移动到队列顶端之后，才能看到消息四。

另一方面，逻辑顺序主要与消息的分组和分段有关。在这种情况下，消息根据 GroupID 的划分，在队列中分组出现，各组的顺序取决于该组第一条消息的物理位置。消息分段的过程与此类似。
我们以按照下列顺序 PUT 消息到队列中为例来作说明：
1. 组 123 的逻辑消息一（非最后）
2. 组 456 的逻辑消息一（非最后）
3. 组 456 的逻辑消息二（最后）
4. 组 123 的逻辑消息二（非最后）
5. 组 123 的逻辑消息三（最后）

那么在应用程序中，这些消息以下列顺序出现：
1. 组 123 的逻辑消息一（非最后）
2. 组 123 的逻辑消息二（非最后）
3. 组 123 的逻辑消息三（最后）
4. 组 456 的逻辑消息一（非最后）
5. 组 456 的逻辑消息二（最后）

10.5.2 连接到队列管理器

利用 MQI 进行 WebSphere MQ 应用编程时，您首先应当使用 MQCONN 或 MQCONNX 函数来连接到队列管理器。函数语法如下：

MQCONN （QMgrName，Hconn，CompCode，Reason）
MQCONNX （QMgrName，ConnectOpts，Hconn，CompCode，Reason）

入口参数：
QMgrName (队列管理器名，如果为空串，则表示连接到缺省队列管理器。);
ConnectOpts (控制 MQCONNX 行为的选项)；
出口参数：
CompCode (完成码)；
Reason (原因码)；
Hconn (队列管理器的连接句柄)；
ConnectOpts

下面这段代码显示了如何利用 C 语言 MQI 连接到队列管理器：

```
MQHCONN Hcon;
```

```
MQLONG CompCode；
MQLONG Reason；
char QMName[50]；
strcpy（QMName，“SampleQM”）；
MQCONN（QMName，  &Hcon，&CompCode，&CReason）；
if （CompCode == MQCC_FAILED） {
  printf（"MQCONN failed with reason code %ld\n"，CReason）；
}
```

10.5.3 打开 WebSphere MQ 对象

在开发平台上可以打开以下三种 WebSphere MQ 对象类型：

- 队列
- 过程定义
- 队列管理器

我们调用 MQOPEN 可以打开任何上述对象：

MQOPEN （ Hconn，ObjDesc，Options，Hobj，CompCode，Reason）

入口参数：

Hconn （MQCONN 调用返回的连接句柄）；
ObjDesc （打开对象的描述，它以对象描述（MQOD）结构的形式出现）；
Options （控制调用行为的一个或多个选项）。

出口参数：

CompCode (完成码)；
Reason (原因码)；
Hobj （访问对象的对象句柄）；
ObjDesc （调用后返回的对象描述结构）。

10.5.3.1 打开队列

从程序员的角度来看，共有三种类型的队列：

- 局部队列
- 远程队列
- 动态队列

局部和远程队列代表着队列管理器从管理角度定义的实际队列。它们之间的主要区别是：在 MQOD 数据结构的 ObjectName 字段中确定队列名称的方法不同。

局部队列名是局部队列管理器所确定的。远程队列名可以通过局部队列管理器所知的远程队列名获得，或者通过远程队列管理器中的名称获得。如果采用远程队列管理器中的名称，那么 ObjectQMgrName 字段必须在下面二者中确定其一：

- 与远程队列管理器名称相同的传输队列名称

- 别名队列对象名称（分解为与远程队列管理器名称相同的传输队列）

下面这段代码显示了如何利用局部队列管理器名称打开局部或远程队列：

```
MQOD od = {MQOD_DEFAULT}；
MQLONG O_options；
strcpy（od.ObjectName，“SampleQueue“）；
O_options = MQOO_INPUT_AS_Q_DEF＋ MQOO_FAIL_IF_QUIESCING；
MQOPEN（Hcon，&od，O_options，&Hobj，&CompCode，&Reason）；
if （Reason != MQRC_NONE） {
        printf（"MQOPEN failed with reason code %ld\n"，Reason）；
}
```

动态队列是根据需要而建立的，一旦不再被使用就会被删除。在 WebSphere MQ 中，动态队列不是在管理员层次上建立的。举例来说，在请求器指定一个“回复到”队列的请求/回复环境中，就可以采用动态队列。

建立动态队列时，我们应采用被称为模型队列的模版，这个模版是在管理员层次上建立的，同时还调用了 MQOPEN。动态队列名可通过 MQOD 结构的 DynamicQName 来确定。我们可以通过三种方法确定动态队列名：

- 赋予一个全名，但不超过 33 个字符
- 赋予一个局部名，在这种情况下，您可以为队列名指定前缀，后跟一个星号（*）。队列管理器随后会用您指定的前缀生成一个唯一的名称。
- 在首个字符处给一个星号（*），允许队列管理器生成一个全名。

如果动态队列成功创建，那么对象描述器结构就会返回 ObjectName 字段中动态队列的实际名称。下面这段代码可以打开动态队列，并输出动态队列名：

```
strcpy（od.ObjectName，“SampleModel“）；
strcpy（od.DynamicQName，“SampleDQ*“）；
O_options = MQOO_INPUT_AS_Q_DEF ＋ MQOO_FAIL_IF_QUIESCING；
MQOPEN（Hcon，&od，O_options，&Hobj，&CompCode， &Reason）；
if （Reason != MQRC_NONE） {
        printf（“MQOPEN failed with reason code %ld\n”，Reason）；
} else {
        printf（“The newly open dynamic queue name is %s”，od.ObjectName）；
}
```

10.5.3.2 打开分布列表

通过使用分布列表可以将一条消息放入多个队列中。为实现这个目的，我们必须调用 MQOPEN 打开分布列表，同时需要下列输入参数：

连接句柄

对象描述器结构（MQOD）中的一般信息

对象描述器结构(MQOD)必须在版本字段中指定 MQOD_VERSION_2，也必须在 RecsPresent 字段中指定对象记录结构的数量(与您希望打开的队列数量相同)。 利用对象记录结构（MQOR），确定您希望打开的每个队列的队列名。

上述调用的输出为：

对分布列表访问的对象句柄;

完成码;

原因码;

响应记录（MQRR 结构），包括每个目的地的结果代码（完成代码和原因代码）必须向每个目的地都提供一个 MQOR 记录，作为队列名和队列管理器名的结合。我们可以通过两种方法确定目的地队列数组的地址：

利用指针字段 ObjectRecPtr，给出 MQOR 记录数组的地址。这是 C 语言中通常采用的方法，请看下面的例子：

```
MQOD MyMqod;
MQOR MyMqor[NUMBER_OF_DESTINATIONS];
MyMqod.ObjectRecPtr = MyMqor;
```

MQRR 结构在分布列表中包含特定目的地的完成和原因代码。如果任何目的地打开失败，MQRR 数组将允许我们发现有问题的队列并采取某种纠正行动。

例如，打开分布列表

```
#define NumQueues
MQLONG Index ;
PMQRR pRR=NULL;
PMQOR pOR=NULL;
char queueNames[MQ_Q_NAME_LENGTH][NumQueues];
char queueMNames[MQ_Q_MGR_NAME_LENGTH][NumQueues];
pRR = （PMQRR）malloc（ NumQueues * sizeof（MQRR））;
pOR = （PMQOR）malloc（ NumQueues * sizeof（MQOR））;
for（ Index = 0 ； Index < NumQueues ； Index ＋＋） {
        strncpy（ （pOR＋Index）－>ObjectName，
                  queueNames[Index]，
                （size_t）MQ_Q_NAME_LENGTH）;
        strncpy（ （pOR＋Index）－>ObjectQMgrName，
                  queueMNames[Index]，
                （size_t）MQ_Q_MGR_NAME_LENGTH）;
}
od.Version =MQOD_VERSION_2 ;
od.RecsPresent = NumQueues ;
od.ObjectRecPtr = pOR;
od.ResponseRecPtr = pRR ;
```

```
O_options = MQOO_OUTPUT ＋ MQOO_FAIL_IF_QUIESCING；
MQOPEN（Hcon， &od，O_options，&Hobj，&CompCode，&Reason）；
```

如欲了解如何利用这些结构的更多细节，请参阅《Application Programming Reference》。

10.5.4 关闭 WebSphere MQ 对象

我们调用 MQCLOSE 函数可以关闭 WebSphere MQ 对象。

MQCLOSE （Hconn，Hobj，Options，CompCode，Reason）

入口参数：
Hconn （连接句柄）；
Hobj（被关闭对象的句柄）；
Options（关闭选项）

出口参数：
结果代码（完成代码和原因代码）
Hobj （对象句柄，重置为 MQHO_UNUSABLE_HOBJ）

如果您关闭的不是永久动态队列的话，那么关闭选项将为 MQCO_NONE。通常说来，一旦建立动态队列的程序对该队列调用 MQCLOSE，该队列就会被删除。但是，就永久动态队列而言，队列管理器可以保存它们，或者也可以根据 MQCLOSE 调用的选项删除它们。

```
MQLONG C_options；
C_options = 0；
MQCLOSE（Hcon， &Hobj，C_options，&CompCode，&Reason）；
```

我们建议您在应用程序结束之前关闭所有 WebSphere MQ 对象。

10.5.5 断开与队列管理器的连接

WebSphere MQ 程序的最后一步就是断开与队列管理器的连接。我们可以调用 MQDISC 来实现。
MQDISC （Hconn，CompCode，Reason）

入口参数：
Hconn （提供到队列管理器的连接句柄）
出口参数：
结果代码（完成代码和原因代码）
Hconn （连接句柄设置为 MQHC_UNUSABLE_HCONN）。

```
MQDISC（&Hcon，&CompCode，&Reason）；
```

10.5.6 将消息放入队列

MQI API 为将消息放入队列向程序员提供了两个选项：

将多个消息放入一个已经打开的队列。

将单一消息放入一个队列，而不显式打开队列。

我们可以调用 MQPUT 将多个消息放入一个队列：

MQPUT （ Hconn ，Hobj ，MsgDesc ，PutMsgOpts ，BufferLength ，Buffer ，CompCode ，Reason ）

入口参数：

Hconn （连接句柄，由 MQCONN 调用返回）

Hobj （队列句柄，由 MQOPEN 调用返回）

MsgDesc （消息的描述）

PutMsgOpts （控制信息，其形式为一个放置消息选项（MQPMO）结构）

BufferLength （消息所包含数据的长度）

Buffer （消息本身）

出口参数：

结果代码（完成代码和原因代码）

MsgDesc （已经更新的消息描述器和选项）

对此函数进行调用之前，队列必须先用 MQOO_OUTPUT 选项打开。

下列代码显示了如何向 SampleQueue 队列中 PUT 一条消息。

```
MQOD od = {MQOD_DEFAULT}；
MQMD md = {MQMD_DEFAULT}；
MQPMO pmo = {MQPMO_DEFAULT}；
MQHCONN Hcon；
MQHOBJ Hobj；
MQLONG O_options；
MQLONG CompCode；
MQLONG Reason；
MQLONG messlen；
char buffer[100]；
MQCONN（...）；
strncpy（od.ObjectName，“SampleQueue”）；
O_options = MQOO_OUTPUT ＋ MQOO_FAIL_IF_QUIESCING；
MQOPEN（Hcon，&od，O_options，&Hobj，&CompCode，&Reason）；
if （Reason != MQRC_NONE） {
        printf（"MQOPEN ended with reason code %ld\n"，Reason）；
```

```
}
if （CompCode == MQCC_FAILED） {
        printf（"unable to open queue for output\n"）；
}
strcpy（buffer，“Message data”）；
messlen = strlen（buffer）；

memcpy（md.MsgId， MQMI_NONE，sizeof（md.MsgId） ）；
memcpy（md.CorrelId，MQCI_NONE，sizeof（md.CorrelId） ）；
MQPUT（Hcon， Hobj，&md，&pmo，messlen，buffer，&CompCode，&Reason）；
if （Reason != MQRC_NONE） {
        printf（"MQPUT ended with reason code %ld\n"，Reason）；
}
```

如果仅将单一消息 PUT 到队列中我们可以调用 MQPUT1，不需要 MQOPEN 打开队列即可将单个消息放入队列。

MQPUT1 （Hconn，ObjDesc，MsgDesc，PutMsgOpts，BufferLength，Buffer，CompCode，Reason）

除了队列句柄之外，MQPUT1 参数接口与 MQPUT 参数接口相同，但 MQPUT1 中必须指定一个对象描述块，就像 MQOPEN 调用所指定的那样。例如，

```
MQOD od = {MQOD_DEFAULT}；
MQMD md = {MQMD_DEFAULT}；
MQPMO pmo = {MQPMO_DEFAULT}；
MQHCONN Hcon；
MQLONG CompCode；
MQLONG Reason；
MQLONG messlen；
char buffer[100]；
MQCONN（...）；
strncpy（od.ObjectName，“SampleQueue”）；
strcpy（buffer，“Message data”）；
messlen = strlen（buffer）；
memcpy（md.MsgId，MQMI_NONE，sizeof（md.MsgId） ）；
memcpy（md.CorrelId，MQCI_NONE，sizeof（md.CorrelId） ）；
MQPUT1（Hcon， &od， &md，&pmo，messlen，buffer， &CompCode，&Reason）；
if （Reason != MQRC_NONE） {
        printf（"MQPUT ended with reason code %ld\n"，Reason）；
}
```

提示：重要的是，如果实际上要利用此函数的操作数量很少，而不必考虑 MQOPEN 和 MQPUT 函数的话，这种方法就是有用的，因为与这个单放置函数相关的总开销要高得多。

同时将消息放入多个队列您也可以将消息放入分布列表。利用单一的 MQPUT 或 MQPUT1 调用，消息被发送到分布列表中的所有队列中。在这种情况下，MQPUT 调用的输入参数如下：

连接句柄
对象句柄，到利用 MQOPEN 调用打开的分布列表，和本章前面所显示的一样
消息描述器结构（MQOD）
总体控制信息
控制信息，以放置消息记录的形式（MQPMR）
消息所包含数据的长度
消息本身

此调用的输出如下：

结果代码（完成代码和原因代码）
响应记录

MQPMR 结构为某些字段（可能和已经存在于 MQMD 结构中的字段不同）给出特定的目的地信息。

10.5.7 从队列获取消息

我们调用 MQGET 从队列中获取消息。

MQGET （ Hconn，Hobj，MsgDesc，GetMsgOpts，BufferLength，Buffer，DataLength，CompCode，Reason）

此调用的入口参数如下：

Hconn （连接句柄，由 MQCONN 调用返回）
Hobj （队列句柄，由 MQOPEN 调用返回）
MsgDesc （消息的描述）
GetMsgOpts （控制信息，以获取信息选项（MQGMO）结构的形式）
BufferLength （消息所包含数据的长度）
Buffer （消息本身）

此调用的出口参数如下：

结果代码（原因代码和完成代码）
Buffer （消息本身）
MsgDesc （已经更新的消息描述器和选项）
DataLength （消息的实际长度）

为了执行 MQGET 调用，必须用 MQOO_INPUT_SHARED 或 MQOO_INPUT_EXCLUSIVE 选项打开队列，才能使用这个调用。从队列获得的消息可能是以物理顺序，也可能是以逻辑顺序排列，这取决于 MQOPEN 调用和 MQGET 调用所采用的选项。
例如， MQGET 调用

```
MQOD od = {MQOD_DEFAULT};
MQMD md = {MQMD_DEFAULT};
```

```
MQGMO gmo = {MQGMO_DEFAULT};
MQHCONN Hcon;
MQHOBJ Hobj;
MQLONG O_options;
MQLONG CompCode;
MQLONG Reason;
MQBYTE buffer[101];
MQLONG buflen;
MQLONG messlen;
MQCONN(...);
strncpy(od.ObjectName, "SampleQueue");
O_options = MQOO_INPUT_AS_Q_DEF + MQOO_FAIL_IF_QUIESCING;
MQOPEN(Hcon, &od, O_options, &Hobj, &CompCode, &Reason);

if (Reason != MQRC_NONE) {
        printf("MQOPEN ended with reason code %ld\n", Reason);
}
if (CompCode == MQCC_FAILED) {
        printf("unable to open queue for input\n");
}

gmo.Version =MQGMO_VERSION_2;

gmo.MatchOptions = MQMO_NONE;
gmo.Options = MQGMO_WAIT + MQGMO_CONVERT;
gmo.WaitInterval = MQWI_UNLIMITED;
buflen = sizeof(buffer) - 1;
md.Encoding = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;

MQGET(Hcon, Hobj, &md, &gmo, buflen, buffer, &messlen, &CompCode, &Reason);

if (CompCode == MQCC_FAILED) {
        printf("MQGET ended with reason code %ld\n", Reason);
} else {
        buffer[messlen] = '\0';
        printf("message <%s>\n", buffer);
}
```

从队列中获取指定消息我们也可以根据消息描述获取消息，正如 MQMD 结构所提供的那样。可利用 MsgId 和 CorrelId 字段来查询某一特定消息，但由于这两个字段是由应用程序设定的，因此它们可能不是唯一的。在这种情况下，我们获取的第一个符合所有标准的消息可以重新调用，以获取其余的消息。如果采用 MQMD 结构第二版的话，那么也可以利用 GroupId、MsgSeqNumber 和 Offset 字段。我们可以为上述任何字段指定一个“不匹配”的

值，这样在查找匹配时就不会考虑该字段。利用 MQMD 结构第二版时，也可以在队列扫描中声明采用哪些字段。下面这段代码显示了如何在调用 MQGET 前设定 correlID 作为消息查找键：

```
gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy（md.CorrelId，myCorrelId，sizeof（md.CorrelId））；
MQGET（Hcon，
Hobj，
&md，
&gmo，
buflen，
buffer，
&messlen，
&CompCode，
&Reason）；
```

队列具备一些索引功能，从而提高了这些操作的性能。索引字段可以是 MsgId 或 CorrelId，具体取决于队列 indexType 属性的值。

10.5.8 从队列浏览消息

下面讨论在队列上怎样浏览消息。
为了在队列上浏览消息，我们应当：

- 调用 MQOPEN 来打开要浏览的队列，选中 MQOO_BROWSE 选项。
- 调用 MQGET，并选中 MQGMO_BROWSE_FIRST 选项，以获得队列上的第一个消息。
- 重复调用 MQGET，并选中 MQGMO_BROWSE_NEXT 选项，以便逐步浏览随后的许多消息，在任何新的 MQGET 调用之前将 MsgId 和 CorrelId 设为空。
- 调用 MQCLOSE 关闭队列。

在浏览消息时，正如我们从队列中获取消息一样，消息排列的顺序可能是物理的，也可能是逻辑的。下例显示了如何浏览物理顺序队列的消息。

```
MQOD od = {MQOD_DEFAULT};
MQMD md = {MQMD_DEFAULT};
MQGMO gmo = {MQGMO_DEFAULT};
MQHCONN Hcon;
MQHOBJ Hobj;
MQLONG O_options;
MQLONG CompCode;
MQLONG Reason;
MQBYTE buffer[101];
MQLONG buflen;
MQLONG messlen;
MQCONN（...）；
```

```
strncpy（od.ObjectName，“SampleQueue”）；
O_options = MQOO_BROWSE ＋ MQOO_FAIL_IF_QUIESCING；
MQOPEN（Hcon，&od，O_options，&Hobj，&CompCode，&Reason）；
if （Reason != MQRC_NONE） {
        printf（"MQOPEN ended with reason code %ld\n"，Reason）；
}
if （CompCode == MQCC_FAILED） {
        printf（"unable to open queue for browse\n"）；
}

gmo.Version =MQGMO_VERSION_2；
gmo.MatchOptions = MQMO_NONE；
gmo.Options = MQGMO_NO_WAIT ＋ MQGMO_BROWSE_NEXT ＋
                MQGMO_ACCEPT_TRUNCATED_MSG；
buflen = sizeof（buffer） － 1；
while （CompCode != MQCC_FAILED） {
        md.Encoding = MQENC_NATIVE；
        md.CodedCharSetId = MQCCSI_Q_MGR；
        MQGET（Hcon，Hobj，&md，&gmo，buflen，buffer，&messlen，&CompCode，&Reason）；
        if （CompCode == MQCC_FAILED） {
                printf（"MQGET ended with reason code %ld\n"，Reason）；
        } else {
                buffer[messlen] = '\0'；
                printf（"message <%s>\n"，buffer）；
        }
}
```

如果您不知道队列中消息的大小，那么您可以在调用 MQGET 时设置下列选项，从而获得消息的大小，再浏览消息或从队列中获取消息：

MQGMO_BROWSE_FIRST 或 MQGMO_BROWSE_NEXT 选项。

MQGMO_ACCEPT_TRUCATED_MSG 选项。

0 缓存区长度

消息大小以 DataLength 参数返回。

10.5.9 查询对象属性

查询对象属性时，我们可以调用 MQINQ。

MQINQ （Hconn，Hobj，SelectorCount，Selectors，IntAttrCount，IntAttrs，CharAttrLength，CharAttrs，CompCode，Reason）

该调用的入口参数如下：

Hconn （连接句柄，由 MQCONN 调用返回）

Hobj （队列句柄，由 MQOPEN 调用返回）

SelectorCount（选择器数量）

Selectors （属性选择器数组）
IntAttrCount （被查询的整型数量）
IntAttrs （整型变量数组，调用向其返回指定的整型）
CharAttrLength （字符属性缓存区的长度）
CharAttrs （字符缓存区，调用把被查询字符属性的值放入其中）

该调用的出口参数如下：
IntAttrs （一系列拷贝到数组中的整型值）
CharAttrs （字符属性返回其中的缓存区）
结果代码（完成代码和原因代码）

就字符属性而言，所得的缓存区由长度固定的属性值一个接一个地填充。如果这些属性中的任何一个实际值小于属性的固定长度，那么其余的空间由空白区填充。如果任何对象（在这种情况下，对象就是队列）请求的属性不适用于该类型的队列，那么空间由星号（*）填充。例如，查询对象属性：

```
MQOD odI = {MQOD_DEFAULT};
MQHCONN Hcon;
MQHOBJ Hinq;
MQLONG O_options;
MQLONG CompCode;
MQLONG Reason;
MQLONG Select[3];
MQLONG IAV[3];
MQCONN（...）;
strcpy（odI.ObjectName，“SampleQueue”）;
O_options = MQOO_INQUIRE ＋ MQOO_FAIL_IF_QUIESCING;
MQOPEN（Hcon，&odI，O_options，
            &Hinq，
            &CompCode，
            &Reason）;

if （Reason != MQRC_NONE） {
        printf（"MQOPEN ended with reason code %ld\n"，Reason）;
}
if （CompCode == MQCC_FAILED） {
        printf（"unable to open queue for inquire\n"）;
} else {
        Select[0] = MQIA_INHIBIT_GET;
        Select[1] = MQIA_CURRENT_Q_DEPTH;
        Select[2] = MQIA_OPEN_INPUT_COUNT;
        MQINQ（Hcon，
                Hinq，
                3L，
```

```
                Select，
                3L，
                IAV，
                0L，
                NULL，
                &CompCode，
                &Reason）；
        if （CompCode == MQCC_OK） {
                sprintf（reply，" has %ld messages，used by %ld jobs"，IAV[1]，IAV[2]）；
                strcat（buffer，reply）；
        if （IAV[0]） {
                strcat（buffer，"； GET inhibited"）；
        }
}
```

10.5.10 设置对象属性

只有队列对象才可以设置属性。我们调用 MQSET 来设定队列属性。

MQSET （Hconn，Hobj，SelectorCount，Selectors，IntAttrCount，IntAttrs，CharAttrLength，CharAttrs，CompCode，Reason）

该调用的参数与 MQINQ 调用的参数相同。除了完成代码和原因代码之外，所有的参数都是输入参数。下面列出了调用 MQSET 时可以设定的属性：

InhibitGet（远程队列不适用）
DistList
InhibitPut
TriggerControl
TriggerType
TriggerDepth
TriggerMsgPriority
TriggerData

下例显示了如何禁止对队列进行放置操作：

```
MQOD odS = {MQOD_DEFAULT}；
MQHCONN Hcon；
MQHOBJ Hset；
MQLONG O_options；
MQLONG CompCode；
MQLONG Reason；
MQLONG Select[1]；
MQLONG IAV[1]；

MQCONN（...）；
```

```
strcpy（odS.ObjectName，“SampleQueue”）；
O_options = MQOO_SET ＋ MQOO_FAIL_IF_QUIESCING；
MQOPEN（Hcon，
            &odS，
            O_options，
            &Hset，
            &CompCode，
            &Reason）；
if （Reason != MQRC_NONE） {
        printf（"MQOPEN ended with reason code %ld\n"，Reason）；
}
if （CompCode == MQCC_FAILED） {
        printf（"unable to open queue for set\n"）；
} else {
        Select[0] = MQIA_INHIBIT_PUT；
        IAV[0] = MQQA_PUT_INHIBITED；
        MQSET（Hcon，
        Hset，
        1L，
        Select，
        1L，
        IAV，
        0L，
        NULL，
        &CompCode，
        &Reason）；
        if （CompCode == MQCC_OK） {
                strcat（buffer，" PUT inhibited"）；
                messlen = strlen（buffer）；
                md.MsgType = MQMT_REPLY；
        } else {
                md.MsgType = MQMT_REPORT；
                md.Feedback = Reason；
        }
        }
```

10.5.11 MQI 中的事务处理

局部或全局工作单元都可以用 MQI API 启动。一旦启动，任何工作单元都可调用 MQCMIT 或 MQBACK 来完成。

MQCMIT （ Hconn，CompCode，Reason）

MQBACK （ Hconn，CompCode，Reason）

我们在 MQPUT 或 MQGET 调用中加入 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT 代码，不调用 MQBEGIN，这样就启动了局部工作单元，请看下面的例子：

```
MQPMO pmo；
pmo.Options = MQPMO_SYNCPOINT；
MQPUT（...）；
```

```
MQGMO gmo；
gmo.Options = MQGMO_SYNCPOINT；
MQGET（...）；
```

工作单元中的每个操作都必须设定 MQPMO 或 MQGMO 选项，正如我们前面代码中所显示的那样。

我们调用 MQBEGIN 来启动全局工作单元。如果局部工作单元已经启动，那么 MQBEGIN 调用会失败，返回 MQRC_UOW_IN_RPOGRESS 原因。
MQBEGIN （ Hconn，BeginOptions，CompCode，Reason）
下面这段伪代码显示的是一个分布式的事务处理，包括基本的获取、放置操作，和一些关系数据库的操作：

```
MQBEGIN（...）；
MQGET（...）；
/*执行一些关系数据库更新。*/
UPDATE tbl1 （f1，f2） VALUES （v1，v2）；
MQPUT（...）；
/*如果任何操作失败，相关处理将停止。*/
if （CompCode != MQCC_OK） {
        MQBACK（...）；
} else {
        MQCOMMIT（...）；
}
```

10.5.12 MQI 中的消息分组

消息分组使得我们可以向消息添加某些分组逻辑，而不必为所有逻辑都编写应用程序代码。分组是由队列管理器来管理的，提供排序和组完成控制等基本特点。调用 MQPUT 或 MQPUT1 时，随消息发出的消息描述器结构（MQMD）引入组标识符。组中的每个消息都必须有 MQMF_MSG_IN_GROUP 标志，但是最后一条消息则应具有 MQMF_LAST_MSG_IN_GROUP 标志。组中消息的顺序储存在 MQMD 结构的 MsgSeqNumber 字段中，它是由队列管理器自动生成的。下面这段代码显示了如何将三条消息作为消息组的一部分发出：

```
md.Version =MQMD_VERSION_2;
pmo.Version =MQPMO_VERSION_2;

pmo.Options = MQPMO_LOGICAL_ORDER |MQPMO_NEW_MSG_ID;
md.MsgFlags = MQMF_MSG_IN_GROUP;
memcpy（md.GroupId，MY_GROUP_ID，sizeof（md.GroupId））；
strcpy（buffer，“First messsage”）；
messlen=strlen（buffer）；
MQPUT（...）；

strcpy（buffer，"Middle Message"）；
messlen=strlen（buffer）；
MQPUT（...）；

md.MsgFlags = MQMF_LAST_MSG_IN_GROUP;
strcpy（buffer，"Final Message"）；
messlen=strlen（buffer）；
MQPUT（...）；



md.Version =MQMD_VERSION_2;
gmo.Version =MQMD_VERSION_2;
gmo.Options = MQGMO_LOGICAL_ORDER ＋ MQGMO_WAIT
              ＋ MQGMO_CONVERT;
gmo.WaitInterval = 1500;

gmo.MatchOptions = MQGMO_NONE;
while （CompCode != MQCC_FAILED） {
      buflen = sizeof（buffer） － 1;
      md.Encoding = MQENC_NATIVE;
      md.CodedCharSetId = MQCCSI_Q_MGR;
      MQGET（Hcon，
                Hobj，
                &md，
                &gmo，
                buflen，
                buffer，
                &messlen，
                &CompCode，
                &Reason）；

      if （Reason != MQRC_NONE） {
            if （Reason == MQRC_NO_MSG_AVAILABLE） {
                  printf（"no more messages\n"）；
```

```
            } else {
                    printf（"MQGET ended with reason code %ld\n"，Reason）；
                    if （Reason == MQRC_TRUNCATED_MSG_FAILED） {
                        CompCode = MQCC_FAILED；
                    }
            }
        }
        if （CompCode != MQCC_FAILED） {
            buffer[messlen] = '\0'；
            printf（"message <%s>\n"，buffer）；
        }
}
```

10.6 本章小结

本章主要介绍了 MQI 的体系结构，并对 MQI API 的使用进行举例说明。

10.7 本章练习

1. MQCONNX 调用中的补充参数是：
   （1） 队列管理器名（queue manager name）
   （2） 连接句柄 (connection handle)
   （3） 连接选项 (connection option)
   （4） 原因码 (reason code)
   答案：（3）
2. MQSET 调用可以改变队列的所有属性。
（1）对　　（2）错
答案：（2）
3. 使用下列那些调用之前，需要执行 MQOPEN 调用：
   （1） MQGET
   （2） MQINQ
   （3） MQPUT1
   （4） MQCLOSE
   （5） ALL of the above
   答案：（1）（2）（4）
4. 下列那种情况，当队列管理器重新启动时，消息可以恢复：
   （1） 被操作员保存的消息
   （2） 永久性消息
   （3） 高优先级消息
   （4） 使用 MsgID 和 CorrelID 检索消息
   答案：（2）
5. 队列管理器决不能设置消息的 CorrelID 属性。
（1）对　　（2）错

答案：（2）

6. 消息组（message group）由什么组成：
   （1） 一个或更多的物理消息
   （2） 一个或更多的逻辑消息
   （3） 仅有一个逻辑消息和多个物理消息
   （4） 以上所有的
   答案：（1）（2）
7. 使用 MQI 编写文件的发送程序。
8. 使用 MQI 编写文件的接收程序。

第十一章 用 C++ API 编程

目标

1. 了解 WebSphere MQ C++ API。该 API 是 MQI API 的面向对象的延伸。
2. 学习这种 API 的基本概念、体系结构模型，以及 API 可用性。
3. 介绍利用这种 API 所能进行的一些基本操作。
4. 最后，我们将探讨如何利用该 API 的编程模式。

11.1 概述

WebSphere MQ C++接口是 MQI API 的延伸。就 WebSphere MQ 消息发布接口而言，为程序员提供了一种面向对象的方法。由于该 API 是以面向对象的模型为基础的，属性和方法都继承到子类中。在下面各节中，我们将确定作为父类的方法。

关键特性：

C++ MQI 可提供 MQI API 的所有特性，如获取、放置及浏览消息等，其还允许用户查询并设置对象选项。此外，还可提供以下特性：

- WebSphere MQ 数据结构的自动初始化；
- 及时的队列管理器连接和队列打开；
- 隐式队列关闭和队列管理器断开；
- 死信标题发送和接收；
- IMS 桥标题发送和接收；
- 参照消息标题发送和接收；
- 触发器消息接收；
- CICS 桥标题发送和接收；
- 工作标题发送和接收；
- 客户机渠道定义。

11.2 平台和语言

WebSphere MQ C++允许您使用 C++语言编写 WebSphere MQ 应用程序，WebSphere MQ C++可以在下列服务器平台使用：

- WebSphere MQ for AIX, Version 5.3
- WebSphere MQ for HP-UX, Version 5.3
- WebSphere MQ for iSeries, Version 5.3
- WebSphere MQ for Linux for Intel, Version 5.3
- WebSphere MQ for Linux for zSeries, Version 5.3
- WebSphere MQ for Solaris, Version 5.3
- WebSphere MQ for Windows, Version 5.3
- WebSphere MQ for z/OS, Version 5.3
- WebSphere MQ for Compaq Tru64 UNIX, Version 5.1
- WebSphere MQ for OS/2 Warp, Version 5.1
- WebSphere MQ for Sun Solaris, Intel Platform Edition, Version 5.1

同时也适用于以下客户机环境：

- AIX
- Compaq Tru64 UNIX
- HP-UX
- Linux for Intel
- Linux for zSeries
- OS/2
- Solaris (SPARC and Intel Platform Editions)
- Windows 3.1
- Windows 95
- Windows NT(R)
- Windows 2000

11.3 库

下表显示了在各个可用平台上使用该 API 开发的 C++程序进行编译时所需的库。

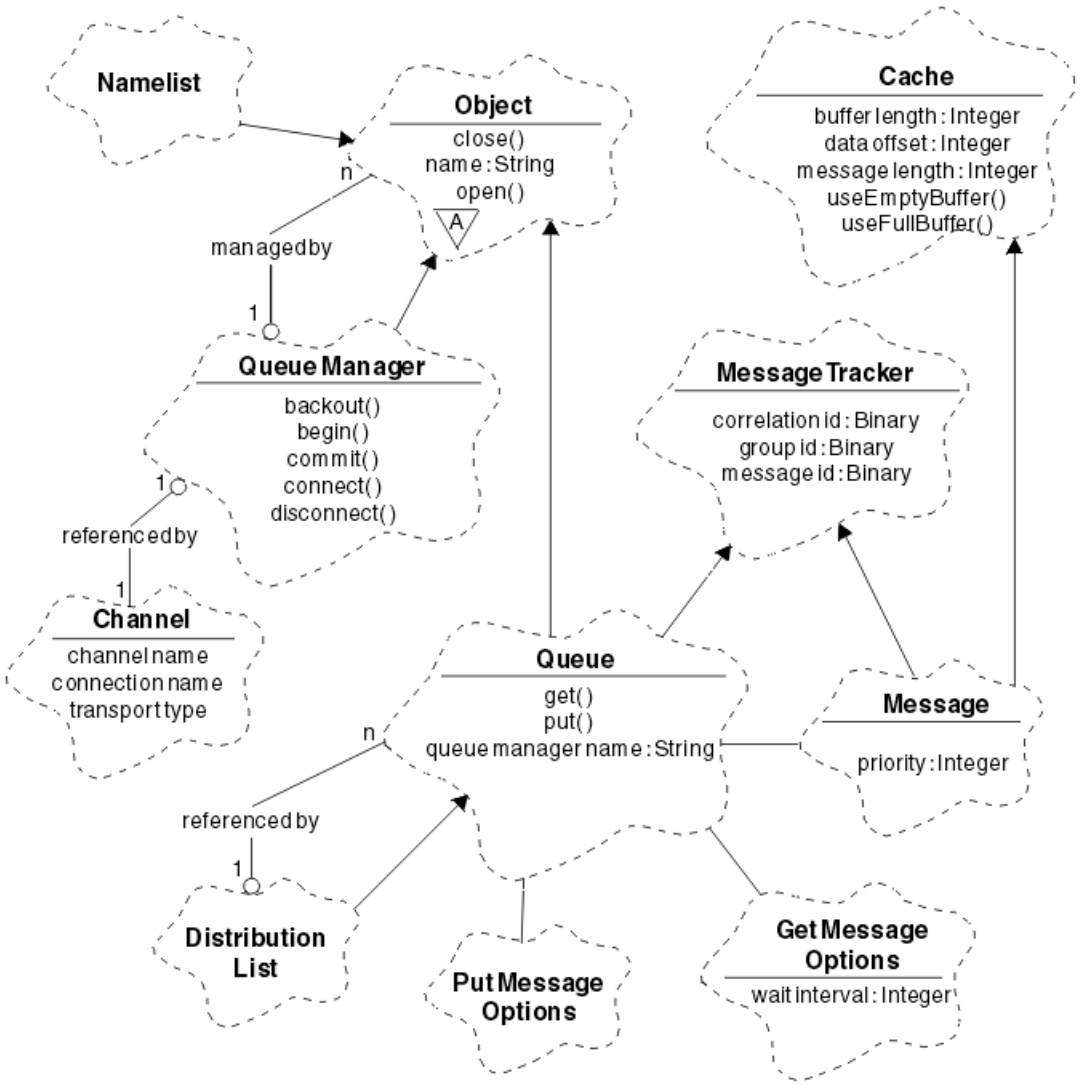
平台	库
WebSphere MQ for Windows NT	IMQ*.LIB
WebSphere MQ for AIX	In a non-threaded application: libimq*.a In a threaded application: libimq*_r.a
WebSphere MQ for Sun Solaris	imq*.so

WebSphere MQ for HP-UX	imq*.so

imqi.hpp 标题包含所有利用此 API 所需的声明。

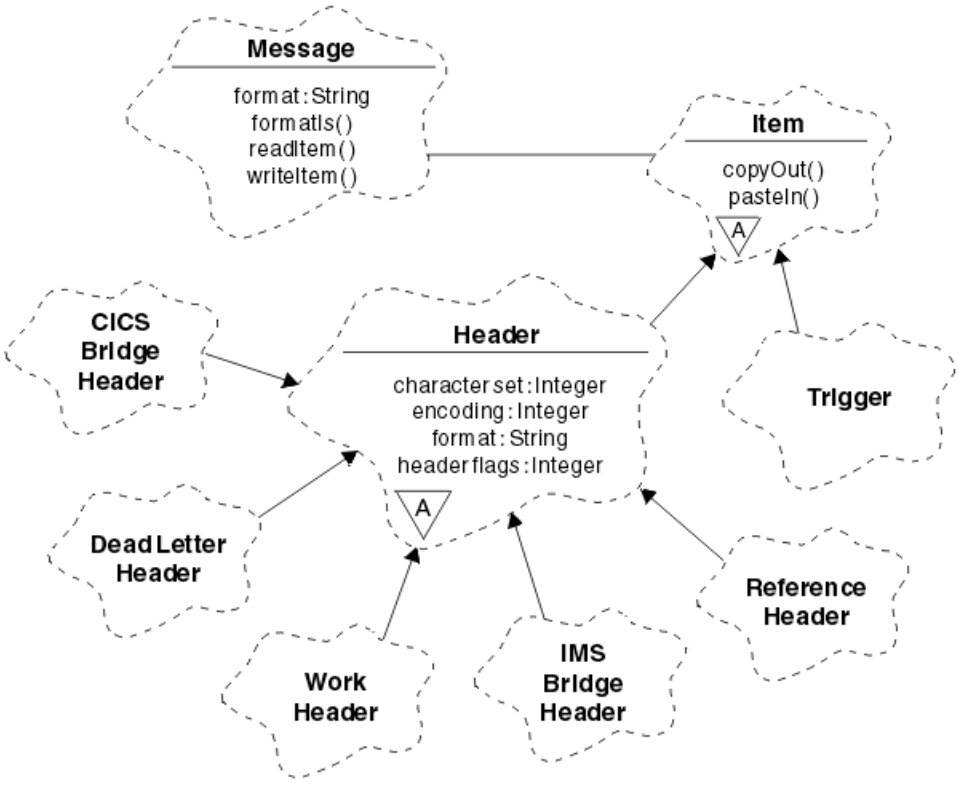
11.4 体系结构模型

此 API 中所有的类均继承于 ImqError 类，其允许错误条件下（error condition）与每个对象相关。如下图（ 队列管理类）。



图，队列管理类

上图显示了与项目有关的类（item-related classes），这些类包含 MQI 提供的消息标题结构（如 IMS 桥标题和死信标题）。



图，项目处理类

队列管理类和项目处理类都利用以下类和数据类型：

ImqBinary 类，包括字节数组（如 MQBYTE24）；

ImqBoolean 数据类型，可定义为 typedef unsigned char ImqBoolean；

ImqString 类，包含字符数组（如 MQCHAR64）。

我们将就其与 MQI API 特性的关系来阐述该 API 体系结构的特性。可将所有的 MQI 数据结构实体归入合适的对象类中，从而使我们可就不同的数据结构字段采用不同的方法。有句柄的实体（继承于 ImqObject 类或其派生物之一）可向我们提供 MQI 的抽象界面。此外，这些对象具有智能行为，与过程 MQI 实施相比，能减少方法调用量。举例来说，到队列管理器的连接根据需要被创建或删除。ImqMessage 类包括 MQMD 数据结构，提供缓冲功能，可以作为用户数据和项目的保存点。缓冲区可以由应用程序提供，也可以由系统自动创建。ImqItem 类代表着消息主题中的项目。项目是指需要连续且分别进行处理的消息的各个部分。除了常规用户信息外，项目还可能是死信标题或触发器消息。每个项目都有对象类，与一种可辨认的 WebSphere MQ 消息格式相对应。用户数据没有对象类，但是可以特别指明 ImqItem，从而将其写入。如果不写入的话，那么用户数据处理则交由应用程序来完成。ImqChannel 类包括渠道定义（MQCD）结构，使得我们可以确定在客户机环境中执行 ImqQueueManager::connect（）时要用到的连接选项。欲了解有关 WebSphere MQ 自动通道定义选项的更多信息，请参见《WebSphere MQ Intercommunication》。

11.5 用 C++ API 编程

现在，我们将阐述如何利用上述 API 来实现基本的 WebSphere MQ 操作，如连接到队列管理器、打开一个队列或发送/接收消息。

11.5.1 连接到队列管理器

为了连接到队列管理器，我们将使用 ImqQueueManager 类（它包括 WebSphere MQ 队列管理器对象）。队列管理器名可以由构造器调用提供，也可以用 ImqQueueManager 类的 setName 方法来提供。

```
ImqQueueManager qmanager;
qmanager.setName（name）;
```

或者

```
ImqQueueManager *pmanager = new ImqQueueManager（name）;
```

提示：我们在本章其余部分中都将用到 qmanager 对象。而后，我们可以利用 ImqQueueManager 的连接方法来建立连接。

```
qmanager.connect（）;
```

队列管理器的信息可以利用 ImqQueueManager 类来访问。

11.5.2 打开 WebSphere MQ 对象

我们可以根据对象是队列还是其他类型的对象，然后利用 ImqObject 或 ImqQueue 类来打开 WebSphere MQ 对象。一般来说，我们都会使用 ImqQueue 类，除非必须要查询或设定某些对象属性。

- 打开队列

ImqQueue 类包括 WebSphere MQ 队列对象，并向队列对象行为添加了某些信息。在可以对队列进行任何放置或获取操作前，必须利用 ImqQueue 类的 setConnectionReference 方法将包含队列的队列管理器分配给 ImqQueue 对象。

```
ImqQueue pqueue;
pqueue.setConnectionReference（pmanager）;
```

可在对象构建过程中提供队列名，也可以利用 ImqObject 类的 setName 方法提供队列名。

```
pqueue.setName（queuename）;
```

当发出放置或获取调用时，将自动采用要求的选项打开队列，也就是说，不需要进行显式打开操作。如果实际的打开选项不符合在队列上进行操作的要求的话，那么 ImqQueue 对象就会关闭并重新打开队列。

在某些情况下，根据被打开队列的类型，将会导致一些额外的开销或某些问题。为了避免自动关闭和重新打开队列，我们必须利用 ImqObject 类的 openFor 方法或 setOpenOptions 直接设置打开选项。我们也可以利用 ImqObject 类的打开方法显式打开队列，但是如果打开选项已经指定的话，那么较之于这种接口提供的隐式打开，它并不能提供什么重大优势。

```
pqueue.setOpenOptions（MQOO_OUTPUT | MQOO_INPUT_SHARED）;
```

或者

```
pqueue.openFor（MQOO_OUTPUT | MQOO_INPUT_SHARED）;
```

openFor 方法不断添加指定的打开选项到实际分配给对象的选项。ImqQueue 对象的默认打开选项是 MQOO_INQUIRE。

- 打开动态队列

动态队列不能通过重新打开方式自动关闭，因为对动态队列进行关闭操作会删除该队列。因此，打开动态队列时，我们必须指定打开选项。
队列模型的名由 ImqObject 类的 setName 方法指定，动态队列名或其前缀可以用 ImqQueue 类的 setDynamicQueueName 方法确定。动态队列的实际名可以在队列打开后用 dynamicQueueName 方法获得。

pqueue.setDynamicQueueName（dynamicqueuename）；

- 打开分布列表

分布列表由 ImqDistributionList 类进行管理，它继承自 ImqQueue 类。可以利用 ImqQueue 类的 setDistributionReference 方法将任意数量的 ImqQueue 对象和一个 ImqDistributionList 对象关联起来。

在打开分布列表之前，相关联的队列必须分配到队列名和包含队列的队列管理器，下面提供了一个打开分布列表的例子：

```
ImqDistributionList dlist;
ImqQueue queueA, queueB;
ImqString queueManagerName（pmanager.name（））；
queueA.setConnectionReference（pmanager）；
queueB.setConnectionReference（pmanager）；
queueA.setName（queuename1）；
queueB.setName（queuename2）；
queueA.setQueueManagerName（ queueManagerName）；
queueB.setQueueManagerName（ queueManagerName）；
queueA.setDistributionListReference（dlist）；
queueB.setDistributionListReference（dlist）；
```

一旦设置好分布列表的队列，就可以向其他任何 ImqQueue 对象一样来打开分布列表并对其进行操作。

11.5.3 关闭 WebSphere MQ 对象

WebSphere MQ 对象在删除与其相应的 ImqObject 后即会自动关闭。

11.5.4 断开与队列管理器的连接

当删除 ImqQueueManager 对象后，将隐式执行断开连接操作。

11.5.5 消息放入队列

我们可以利用 ImqQueue 类的放置方法将消息放入 ImqQueue 或 ImqDistributionList。放置方法提供两种接口：
ImqBoolean put（ImqMessage & msg）；

ImqBoolean put（ImqMessage & msg, ImqPutMessageOptions & pmo）；

消息数据由 ImqMessage 类管理。ImqMessage 类继承自 ImqMessageTracker 类（它包括 MQMD 数据结构）和 ImqCache 类（它处理消息数据缓冲区）。

我们可以利用 ImqMessageTracker 类的 setMessageId 方法来设置消息身份。

```
ImqMessage msg;
msg.setMessageId（msgId）；
```

同样，我们也可以利用某些方法来访问关联性 ID 和组 ID。我们必须用 ImqBinary 类来创建 messageId、correlationId 和 groupId。该类包括用于 MQI 的 BYTExx 数据类型，它提供某些进行基本操作的方法。下例显示了如何利用 ImqBinary 来创建二进制对象。

```
ImqBinary correlationId ；
MQBYTE24 byteId =“BYTEID1234”；
correlationId.set（byteId,sizeof（byteId ））；
```

准备消息数据

该 API 与 MQI API 的不同之处在于准备和处理消息数据的方法不同。在 MQI 中，从分配正确的缓冲区到储存数据，再到读取消息时处理消息中不同的可能标题，消息完全由应用程序管理。

在 C++ API 中，添加了一些缓冲功能，而且缓冲区由 ImqCache 对象管理。缓冲区通过继承与每个消息（ImqMessage 对象）相关联。默认情况下，缓冲区由 ImqCache 自动提供，或者也可以由应用程序利用以下任何一种 ImqCache 对象方法来提供：

useEmptyBuffer：这种方法允许应用程序分配一个固定长度的空白缓冲区给 ImqMesage 对象。如果没有分配实际消息长度的话，那么会将消息长度自动设置为零，而且缓冲区也将是空的。

```
ImqMessage msg;
char pszBuffer[24]=“Hello World”；
msg.useEmptyBuffer（pszBuffer, sizeof（pszBuffer））；
msg.setFormat（MQFMT_STRING）；
msg.setMessageLength（12）；
```

或

```
char pszBuffer[12];
msg.useEmptyBuffer（pszBuffer, sizeof（pszBuffer））；
msg.setFormat（MQFMT_STRING）；
```

useFullBuffer：这种方法允许应用程序分配一个已经准备好的消息缓冲区给 ImqMessage 对象。该缓冲区将不是空的，消息长度也将被设为方法调用所提供的长度。

```
ImqMessage msg;
char pszBuffer[] =“Hello world”；
msg.useFullBuffer（pszBuffer, sizeof（pszBuffer））；
msg.setFormat（MQFMT_STRING）；
```

消息缓冲区可以重复使用，我们也可以利用 ImqCache 类的 setMessegeLength 方法来设置消息长度，因此发送字节的数量也各不相同。由应用程序提供消息缓冲区的优势在于无需

进行数据拷贝，因为数据可以直接在缓冲区中准备。

为了将 ImqCache 再设为自动的缓冲区，应用程序可以用空缓冲区指针和零（0）长度调用 useEmptyBuffer。当自动提供缓冲区时，缓冲区随着消息的增长而增长。这为准备消息前却不知道消息长度提供了更多的灵活性。可以利用 ImqCache 写入方法将消息（数据）拷贝到缓冲区中。

msg.write（12, “Hello world”）；

我们可以利用 ImqMessage 的 writeItem 方法将项目拷贝到缓冲区中。比方说，您可能希望向消息添加一个死信标题，并将其放入死信队列中。

下例显示了如何创建一个 ImqDeadLetterHeader 并将其插入现有消息的开始部分。

```
ImqDeadLetterHeader header;
header.setDestinationQueueManagerName（pmanager.name（））；
header.setDestinationQueueName（pqueue.name（））；
header.setPutApplicationName（/*?*/）；
header.setPutApplicationType（/*?*/）；
header.setPutDate（/* TODAY*/）；
header.setPutTime（/* NOW*/）；
header.setDeadLetterReasonCode（/* REASON*/）；
msg.writeItem（header）；
```

可以在放置方法中使用更多选项，正如 ImqQueue 类的放置方法提供了两种接口。在将消息放到队列上时，必须经常指定更多的选项。我们可以用以 ImqPutMessageOptions 对象为形式的第二个参数调用放置方法来指定这些选项。ImqPutMessageOptions 类包括 MQPMO 数据结构，它允许应用程序指定更多的选项，如同步点控制或消息上下文。

下例显示了如何启动并设置同步点选项。这将启动本地队列管理器事务处理，我们可以利用 ImqQueueManager 类的提交或回滚方法来结束它。

```
ImqQueue pqueue;
ImqMessage msg;
ImqPutMessageOptions pmo;

pmo.setSyncPointParticipation（TRUE）；
pqueue.put（msg, pmo）；
```

如欲了解 ImqQueue 类可用选项的更多信息，请参见《Using C++》手册。

11.5.6 从队列获取消息

我们可以利用该类提供的获取方法从 ImqQueue 对象上获取消息。ImqQueue 获取方法提供四种接口：

```
ImqBoolean get（ImqMessage & msg, ImqGetMessageOptions & options）；
ImqBoolean get（ImqMessage & msg）；
ImqBoolean get（ImqMessage & msg, ImqGetMessageOptions & options,
                                        const size_t buffer-size）；
ImqBoolean get（ImqMessage & msg, const size_t buffer-size）；
```

在方法调用之后，消息信息会包含在 ImqMessage 对象中。缺省情况下，消息缓冲区由系统

提供，可以利用 dataPointer 或 bufferPointer 方法获得。消息数据长度可以利用 ImqCache 类的 dataLength 方法获得。

```
pqueue.get（msg）；
char *pszDataPointer = msg.dataPointer（）；
int iDataLength = msg.dataLength（）；
```

注意：

在每次获取方法调用之后，数据缓冲区的物理位置可能发生改变，因此我们建议不要使用实际的缓冲区指针来访问数据。我们应当利用 dataPointer 或 bufferPointer 方法来重新分配数据指针。

如果应用程序希望提供固定长度的缓冲区来接收消息数据的话，那么我们可以在利用 ImqQueue 的获取方法前使用 ImqCache 的 useEmptyBuffer 方法。给定缓冲区的长度将限制消息长度，因此在应用程序设计中必须考虑到长消息的情况。

```
char pszBuffer[BUFFER_LENGTH]；
pqueue.useEmptyBuffer（pszBuffer, BUFFER_LENGTH）；
pqueue.get（msg）；
```

在这种情况下，我们可以一直使用实际的缓冲区指针 pszBuffer，但是我们还是建议采用 dataPointer 方法以保证可移植性。

读取消息数据

一旦接收到消息，那么根据消息格式，消息数据可能是项目形式的，也可能是原始的用户数据形式。但项目必须是分别连续处理的数据的各个部分。我们可以利用 ImqMessage 的 formatIs 来确保消息格式的有效性。如果消息格式代表着任何已知消息标题数据结构，那么我们可以利用 ImqMessage 的 readItem 方法从消息中获得结构。该 API 具有三个实现定义的消息标题：

死信标题（ImqDeadLetterHeader class）；

IMS 桥标题（ImqIMSBridgeHeader）；

参照标题（ImqReferenceHeader）。

每个标题都对应于一个 WebSphere MQ 定义的消息格式。用户可以指定 ImqItem 类来定义其他类型的格式。

```
if （msg.formatIs（MQFMT_DEAD_LETTER_HEADER））  {
        ImqDeadLetterHeader header；
        /*The readItem method must be called with the right class of object pointer*/
        if （msg.readItem（header））  {
          /*Perform the corresponding operation for this item type*/
        }
}
```

如果消息格式未知的话，那么正如前面所讲解过的那样，我们可以利用 dataPointer 方法来

直接访问消息数据。

更多的获取方法选项

ImqGetMessageOptions 类为消息接收过程提供了更多信息，如：

获取操作的等待间隔；
匹配选项；
消息选项；
同步点参加；
组状态；
细分状态。

我们可以利用 ImqGetMessageOptions 的 setOptions 方法来指定任何 MQI 中可用的消息获取选项。其中 MQGMO_WAIT 选项应用较多，它为要完成的获取操作提供了一个等待间隔。这样，如果期待的消息还未到达队列的话，那么获取方法在返回错误前会等待一定的时间，时间长度是用 ImqGetMessageOptions 类的 setWaitInterval 方法来指定。

下例显示了如何用无限等待选项从队列获取消息。

```
ImqGetMessageOptions gmo；
ImqMessage msg；
gmo.setOptions（MQGMO_WAIT）；
/*Set the wait interval to unlimited meaning that the get operation*/
/*will wait until one message appears in the queue.*/
gmo.setWaitInterval（MQWI_UNLIMITED）；
pqueue.get（msg,gmo）；
```

从队列获取特定消息

通过 ImqMessageTracker 类中的消息属性的任意结合，我们可以确认特定的消息。

MessageId
CorrelationId
GroupId

这些选项必须在获取方法调用中传递的 ImqMessage 对象中指定。ImqGetMessageOptions 类为应用程序提供了一种在指定消息查找过程中将使用什么选项的方法。

```
gmo.setMatchOptions（MQMO_MATCH_MSG_ID）；
msg.setMessageId（msgId）；
If （pqueue.get（msg,gmo））{
    /*Perform any operation with this message*/
}
```

如果一个以上的消息与给定标准相匹配，那么将返回这些消息中的第一个，并且获取方法的后续调用将提供所有消息的访问。如果找到了匹配的消息，消息对象信息在获取方法调用后将发生改变，否则函数会返回假。

11.5.7 浏览队列上的消息

我们可以利用 ImqQueue 获取方法浏览队列上的消息。必须用 MQOO_BROWSE 选项来打开 ImqQueue 对象。我们可以用 setOpenOptions 或 openFor 方法来实现这一目的。

```
pqueue.setOpenOptions（MQOO_BROWSE）；
或者
pqueue.openFor（MQOO_BROWSE）；
```

在队列对象已被打开并用于浏览后，我们必须用以下选项调用 ImqQueue 获取方法：MQGMO_BROWSE_FIRST 消息选项，如果您希望浏览光标位于符合 ImqMessage 对象指定的标准的第一个消息处的话；MQGMO_BROWSE_NEXT 消息选项，如果您希望浏览光标移动到符合 ImqMessage 对象指定的标准的下一个消息的话。

获取方法将返回 ImqMessage 对象的更新版本，浏览光标也将指向当前消息的信息，并且当前消息不会从队列中删除。队列对象一旦打开， 浏览光标就会指向队列中的第一个消息，因此 MQGMO_BROWSE_NEXT 选项与 MQGMO_BROWSE_FIRST 具有相同的行为。

```
gmo.setOptions（MQGMO_BROWSE_NEXT | MQGMO_WAIT）；
/*Browsing all the messages in the queue in sequential order*/
while （pqueue.get（msg,gmo））{
        /*Perform some operation with the message*/
          ...
        /*The MessageId and CorrelationId must be set to null before the next get method call*/
        msg.setMessageId（MQMI_NONE）；
        msg.setCorrelId（MQCI_NONE）；
}
```

可以按照物理顺序或逻辑顺序来浏览消息。

根据队列的消息到达顺序（MsgDeliverySequence），物理排序可以是 FIFO（先进/先出）排序或优先级内 FIFO 排序。逻辑顺序就是说，即便另一个组中的任何消息在本组最后一条消息接收前出现，属于同一个组的消息仍将按照其在队列中的正确位置顺序排列。为了浏览逻辑顺序的消息，我们在调用方法时必须指定 MQGMO_LOGICAL_ORDER 选项。

```
gmo.setOptions（MQGMO_BROWSE_NEXT | MQGMO_WAIT |
                                    MQGMO_LOGICAL_ORDER）；
```

11.5.8 查询并设置对象属性

- 查询属性

利用此种 API，较之于采用 MQI API 时的情况，查询并设定对象属性实在是相当直接的操作。这里，ImqObject 类提供了两种查询方法，可查询任何显示的整数或字符属性。

ImqBoolean inquire（const MQLONG int-attr, MQLONG & value ）；
ImqBoolean inquire（const MQLONG char-attr, char * & buffer, const size_t length）；

int-attr 和 char-attr 参数将 MQIA_*和 MQCA_*指数赋予属性。整数对象属性值在值参数中返回，正如下面这段代码所显示的那样：

```
MQLONG depth；
pqueue.inquire（MQIA_CURRENT_Q_DEPTH, depth）；
printf（“The current queue depth is: %d“,depth）；
```

字符对象属性值在缓冲区参数中返回，正如下面这段代码所显示的那样：

```
char qname[MQCA_Q_MGR_NAME_LENGTH]；
pqueue.inquire（MQCA_Q_MGR_NAME, qname, MQCA_Q_MGR_NAME_LENGTH）；
printf（“The current queue depth is: %s“,qname）；
```

缓冲区必须足够大，以致于可以容纳属性值。缓冲区的长度必须在长度参数中指定。设置对象属性为了设置队列属性，ImqObject 提供了两种查询属性的方法。
ImqBoolean set（const MQLONG int-attr, MQLONG & value）；
ImqBoolean set（const MQLONG char-attr, char * buffer, const size_t length）；

下面这段代码显示了这些函数可能的用法：

```
/*This instruction inhibits any put operation on any type of queue. */
pqueue.set（MQIA_INHIBIT_PUT,MQQA_PUT_INHIBITED）；
```

或者

```
/*This instruction inhibits any get operation on any local queue.*/
pqueue.set（MQIA_INHIBIT_GET,MQQA_GET_INHIBITED）；
```

只有下面这些队列属性值可以利用上述函数进行调整：

- MQIA_INHIBIT_PUT
- MQCA_TRIGGER_DATA
- MQIA_DIST_LISTS
- MQIA_INHIBIT_GET
- MQIA_TRIGGER_CONTROL
- MQIA_TRIGGER_DEPTH
- MQIA_TRIGGER_MSG_PRIORITY
- MQIA_TRIGGER_TYPE
- MQIA_DIST_LISTS
- MQIA_INHIBIT_GET

11.5.9 事务处理管理

本地资源管理器事务处理可以通过在 ImqPutMessage 或 ImqGetMessage 选项类中设置同步点参加来启动。

```
ImqPutMessageOptions pmo；
/*This starts a local resource manager transaction*/
pmo.setSyncPointParticipation（TRUE）；
pqueue.put（msg,pmo）；
```

或者

```
ImqGetMessageOptions gmo；
/*This starts a local resource manager transaction*/
gmo.setSyncPointParticipation（TRUE）；
pqueue.get（msg,gmo）；
```

ImqQueueManager 对象提供了用此 API 开始、提交或取消分布式事务处理所需的事务处理管理接口。分布式事务处理由 ImqQueueManager 开始方法调用来启动。一个事务处理开始和结束调用中的任何操作都是事务处理的一部分。

```
/*This call starts a distributed transaction*/
pmanager.begin（）；
```

只有在不存在其他本地或分布式事务处理管理的情况下，才能启动分布式事务处理管理。如果事务处理成功的话，本地和分布式事务处理管理都可以由 ImqQueueManager 提交方法调用来终止。它们也可以由 ImqQueueManager 取消方法调用来终止。

```
pmanager.commit（）；
或者
pmanager.backout（）；
```

11.5.10 消息分组

我们可以利用 ImqMessageTracker 的 setGroupId 方法将消息分组。我们也必须利用 ImqMessage 的 setMessageFlags 方法给出 MQMF_MSG_IN_GROUP 或 MQMF_LAST_MSG_IN_GROUP 标记，从而确认组中的消息。

MQI API 的例子显示了如何将三条消息作为一个组进行发送。前两条消息将用 MQMF_MSG_IN_GROUP 标记发送，而第三条消息则将利用 MQMF_LAST_MSG_IN_GROUP 发送。下例显示了消息分组。

```
/*Setting put message options and message descriptor versions*/
BYTE24 MY_GROUP_ID = “123456”；
```

```
ImqPutMessageOptions pmo；
ImqMessage message；
ImqBinary grpId；

/*Set the grpId binary object value*/
grpId.set（MY_GROUP_ID,sizeof（MY_GROUP_ID）；

/*Sets the put message options to generate a new message ID for every message put into the queue
and to put the messages in their logical order into the queue*/
pmo.setOptions（MQPMO_LOGICAL_ORDER |MQPMO_NEW_MSG_ID）；
message.setMessageFlags（MQMF_MSG_IN_GROUP）；

/*Assign the GroupId*/
message.setGroupId（grpId）；

/*Puts a first message of the group*/
message.write（“First messsage”）；
pqueue.put（message,pmo）；

/*Puts a second message of the group*/
message.write（“Middle messsage”）；
pqueue.put（message,pmo）；

/*Puts the final message of the group. The final message must be identified by giving the
MQMF_LAST_MSG_IN_GROUP flags in the message descriptor structure*/
message.setMessageFlags（MQMF_LAST_MSG_IN_GROUP）；
message.write（“Last messsage”）；
pqueue.put（message,pmo）；
```

而后，我们可以利用 ImqGetMessageOptions 类 setOptions 方法中的 MQGMO_LOGICAL_ORDER 选项，并以这些消息在队列中的逻辑循序接收它们。

```
ImqPutMessageOptions pmo；
ImqMessage message；
char buffer[101]；
message.setEmptyBuffer（buffer, sizeof（buffer）-1）；

gmo.setOptions（MQGMO_LOGICAL_ORDER + MQGMO_WAIT + MQGMO_CONVERT）；

/* 15 second limit for waiting */
gmo.setWaitInterval（1500）；

gmo.setMatchOptions（MQGMO_NONE）；
while （pqueue.completionCode（） != MQCC_FAILED） {
```

```
            md.setEncoding（MQENC_NATIVE）；
            md.setCharacterSet（MQCCSI_Q_MGR）；
            if （pqueue.get（message））{
                buffer[message.dataLength] = 0；
                printf（"message <%s>\n", buffer）；
                /* report reason, if any*/
                if （queue.reasonCode（） != MQRC_NONE） {
                    printf（"MQGET ended with reason code %ld\n", Reason）；
                } else {
                    if （queue.reasonCode（） == MQRC_NO_MSG_AVAILABLE） {
                        printf（"no more messages\n"）；
                }
            }
}
```

另外，队列管理器可以控制是否完全接收消息组。如果我们仅希望队列中出现完整的消息组的话，那么我们可以在 ImqGetMessageOptions 的 setOptions 方法中设置 MQGMO_ALL_MSGS_AVAILABLE 等选项。

11.6 本章小结

本章已经讨论 WebSphere MQ C++ API。该 API 是 MQI API 的面向对象的延伸。此外我们还讨论这种 API 的基本概念、体系结构模型，以及 API 可用性。而后，我们介绍了利用这种 API 所能进行的一些基本操作，如：

- 连接到队列管理器并断开与队列管理器的连接；
- 打开及关闭 WebSphere MQ 对象（如队列对象）；
- 向队列发送消息并从队列获取消息；
- 事务处理管理；
- 消息分组。

最后，我们探讨了如何利用该 API 的编程模式。

11.7 本章练习

1. 使用 WebSphere MQ C++ API 编写文件的发送程序。
2. 使用 WebSphere MQ C++ API 编写文件的接收程序。

第十二章 用 Java 编程

目标

学习使用 WebSphere MQ for Java 编程。

12.1 概述

WebSphere MQ for Java 允许用 Java 编程语言写成的程序直接访问 WebSphere MQ Server，或作为一个 WebSphere MQ Client 连接到 WebSphere MQ。

12.2 平台

WebSphere MQ for Java 产品可用于以下平台：

AIX
iSeries 和 OS/400
HP-UX
Linux
Sun Solaris
z/OS 和 OS/390 V2R9 或更高版本
Windows 平台

12.2.1 获得软件包

WebSphere MQ base Java 的最新版本的安装可以和 WebSphere MQ 同时安装。关于 WebSphere MQ 的安装可以参考下列资料：

- *AIX* 平台的《WebSphere MQ for AIX, V5.3 Quick Beginnings 》
- *HP-UX* 平台的《WebSphere MQ for HP-UX, V5.3 Quick Beginnings》
- *OS/400 平台*的《WebSphere MQ for iSeries V5.3 Quick Beginnings 》
- *Linux* 平台的《WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings 》
- *Solaris* 平台的《WebSphere MQ for Solaris, V5.3 Quick Beginnings 》
- *Windows* 平台的《WebSphere MQ for Windows, V5.3 Quick Beginnings 》
- *z/OS* 平台的《WebSphere MQ for z/OS System Setup Guide 》

WebSphere MQ base Java 被包含在下列 Java 的.jar 文件中:

- **com.ibm.mq.jar**
  这个 jar 文件支持所有的连接选项。
- **com.ibm.mqbind.jar**

这个 jar 文件仅支持 bindings 连接，并不是在所有的平台都提供或支持，所以我们推荐在新应用程序中不要使用它。

12.2.2 WebSphere MQ for Java 的运行环境

为了运行 WebSphere MQ for Java，需要以下的软件：

- 服务器端平台的 WebSphere MQ ；
- 服务器端平台的 Java Development Kit（JDK）；
- 客户端平台的 Java Development Kit 或 Java Runtime Environment（JRE）或支持 Java 的网络浏览器。

12.2.2.1 安装目录

WebSphere MQ Java V5.3 文件的安装目录如下表所示：

平台	目录
AIX	/usr/mqm/java/
z/OS & OS/390	install_dir/mqm/java/
iSeries & AS/400(R)	/QIBM/ProdData/mqm/java/
HP-UX 和 Sun Solaris	/opt/mqm/java/
Linux	install_dir/mqm/java/
Windows systems	\Program Files\IBM\WebSphere MQ\java
提示：install_dir 是产品安装的目录。在 Linux 系统中，它可能是/opt，而在 z/OS 和 OS/390 系统上，它可能是/usr/lpp。	

还提供了一些例子程序，例如安装验证程序(IVP)。下表中列出了不同平台的例子程序的目录结构。

平台	目录
AIX	/usr/mqm/samp/java/base
z/OS & OS/390	install_dir/mqm/java/samples/base
iSeries & AS/400	/QIBM/ProdData/mqm/java/samples/base
HP-UX and Sun Solaris	/opt/mqm/samp/java/base
Linux	install_dir/mqm/samp/java/base
Windows systems	\Program Files\IBM\WebSphere MQ\tools\Java\base
提示：install_dir 是产品安装的目录。在 Linux 系统中，它可能是/opt，而在 z/OS 和 OS/390 系统上，它可能是/usr/lpp。	

12.2.2.2 环境变量

产品安装完成后，您必须更新 CLASSPATH 环境变量，CLASSPATH 中需要包含 WebSphere MQ base Java 和例子程序的目录，如下表所示：

平台	CLASSPATH 的参考设置
AIX	CLASSPATH=/usr/mqm/java/lib/com.ibm.mq.jar: /usr/mqm/java/lib/connector.jar: /usr/mqm/java/lib: /usr/mqm/samp/java/base:
HP-UX 和 Sun Solaris	CLASSPATH=/opt/mqm/java/lib/com.ibm.mq.jar: /opt/mqm/java/lib/connector.jar: /opt/mqm/java/lib: /opt/mqm/samp/java/base:
Windows systems	CLASSPATH=mq_root_dir(CPDNL1)\java\lib\com.ibm.mq.jar; mq_root_dir\java\lib\connector.jar; mq_root_dir\java\lib\; mq_root_dir\tools\java\base\;
z/OS & OS/390	CLASSPATH=install_dir(CPDNL2)/mqm/java/lib/com.ibm.mq.jar: install_dir/mqm/java/lib/connector.jar: install_dir/mqm/java/lib: install_dir/mqm/java/samples/base:
iSeries & AS/400	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/lib/connector.jar: /QIBM/ProdData/mqm/java/lib: /QIBM/ProdData/mqm/java/samples/base:
Linux	CLASSPATH=install_dir(CPDNL2)/mqm/java/lib/com.ibm.mq.jar: install_dir/mqm/java/lib/connector.jar: install_dir/mqm/java/lib: install_dir/mqm/samp/java/base:
注意： 1. mq_root_dir 表示在 Windows 系统的 WebSphere MQ 安装目录。通常是 C:\Program Files\IBM\WebSphere MQ\。 2. install_dir 是产品的安装目录。	

如果现有的应用程序依赖于 com.ibm.mqbind，您必须要把 com.ibm.mqbind.jar 文件加到 classpath 中。

在某些平台还必须要更新下列附加的环境变量，如下表所示：

平台	环境变量
AIX	LD_LIBRARY_PATH=/usr/mqm/java/lib
HP-UX	SHLIB_PATH=/opt/mqm/java/lib
Sun Solaris	LD_LIBRARY_PATH=/opt/mqm/java/lib
Windows systems	PATH=install_dir\lib
z/OS & OS/390	LIBPATH=install_dir/mqm/java/lib
Linux	LD_LIBRARY_PATH=install_dir/mqm/java/lib
注意: install_dir 是产品的安装目录。	

注意：

确保您追加的 WebSphere MQ 的变量不要覆盖现有的系统环境变量。如果覆盖了系统的环境变量，那么应用程序在编译或运行时将可能会失败。

12.3 使用 WebSphere MQ for Java

应用程序连接到队列管理器后，就可以与访问 WebSphere MQ 对象（例如,队列）。队列管理器为其拥有的WebSphere MQ对象提供消息发送服务。使用WebSphere MQ classes for Java 编程的方法依赖于使用的连接模式。连接的模式有两种，分别是客户连接模式和绑定模式。

12.3.1 客户机连接模式

当 WebSphere MQ classes for Java 作为客户端时，与 WebSphere MQ C 客户端类似，但仍然存在如下区别：

1， 仅支持 TCP/IP。
2， 不支持连接表。
3， 在启动时，不读取任何 WebSphere MQ 环境变量。
4， 通道的定义和环境变量信息都被存放在一个叫做 Environment 的类中，当连接时这些信息也可以被作为入口参数。
5， 错误和意外信息被写到 MQException 类说明的日志中。缺省错误信息被写到 Java 控制台。

WebSphere MQ classes for Java 客户端不支持 MQBEGIN 和快速绑定。

当利用客户机连接时，您必须指定其他一些环境属性，以便建立与队列管理器的连接。这些属性是： 主机名，即作为队列管理器主机的 WebSphere MQ 服务器的名字；以及通道名，即客户 机连接通道的名字。另外，您也可以指定 WebSphere MQ 服务器监听的端口号。如果还没有指定端口号的话，那么将使用默认的端口号 1414。

12.3.2 绑定模式

在绑定模式（也称作服务器连接模式）中，与队列管理器的通讯利用的是进程间通讯。关键因素之一就是，要记住绑定模式只适用于那些运行在作为队列管理器主机的 WebSphere MQ 服务器上的程序。利用绑定模式的程序不会从 WebSphere MQ 客户机机器上运行。换言之，应用程序被绑定在队列管理器所在的同一台机器上。绑定模式是访问 WebSphere MQ 的一种快速而高效的方法。某些功能（如队列管理器的扩展架构事务处理协同）只在绑定模式下才可用。

WebSphere MQ classes for Java 的绑定模式与客户连接模式存在下列区别：

1， 忽略了 MQEnvironmnet 类所提供的大多数参数。

2， 绑定模式支持 MQBEGIN 和快速绑定。

12.3.3 类库

WebSphere MQ classes for Java 提供了一系列可以使 Java applet 和应用程序访问 WebSphere MQ 的类。WebSphere MQ for Java 包括以下类和接口：

12.3.3.1 类

- **MQChannelDefinition**

  该类用来传递有关连接队列管理器的信息至发送、接收和安全退出。当以绑定模式直接连接到 WebSphere MQ 时，此类不适用。

- **MQChannelExit**

  当调用发送、接收和安全退出时，该类定义传递到这些调用的上下文信息。该类的 exitResponse 属性应当通过退出设置，以显示 WebSphere MQ Client for Java 下一步应当采取何
种行动。

- **MQDistributionList**

  该类代表开放式队列集，我们可以利用 put（）方法的单一调用发送消息至这些队列中。我们利用 MQDistributionList 构造器或 MQQueueManager 类的 accessDistributionList（）方法来做出该类的实例。

- **MQDistributionListItem**

  该类代表分配表中的单一项目（单一队列）。该类继承 MQMessageTracker 类。

- **MQEnvironment**

  该类包含控制构建 MQQueueManager 对象（及其相对应的到 WebSphere MQ 的连接）环境的静态元素变量。由于调用 MQQueueManager 构造器使该类值的集生效， 因此 MQEnvironment 类的值应当在 MQQueueManager 实例构建前设置。

- **MQException**

  该类包含 WebSphere MQ 完成代码和错误代码常量的定义。以 MQCC_开始的常量是

WebSphere MQ 完成代码，而以 MQRC_开始的常量则是 WebSphere MQ 原因代码。只要出现 WebSphere MQ
错误，就会给出 MQException。

- **MQGetMessageOptions**
  该类包含控制 MQQueue.get（）方法行为的选项。
- **MQManagedObject**
  该类是 MQQueueManager、MQQueue 和 MQProcess 类的超类。它提供查询并设置这些资源属性的能力。
- **MQMessage**
  该类代表 WebSphere MQ 消息的消息描述器和数据。
- **MQMessageTracker**
  该类用来处理分配表中某个给定目的地的消息参数。MQDistributionListItem 继承它。
- **MQPoolServices**
  用作 WebSphere MQ 连接默认 ConnectionManager 的 ConnectionManager，其实现可以使用该类。
- **MQPoolServicesEvent**
  只要添加或删除 MQPoolToken 到 MQEnvironment 控制的权标集，那么就可用该类来生成一个事件。当默认的 ConnectionManager 改变时， 即会生成 MQPoolServicesEvent。
- **MQPoolToken**
  该类可被用来提供默认的连接集合。
- **MQProcess**
  该类为 WebSphere MQ 进程提供查询操作。
- **MQPutMessageOptions**
  该类包含控制 MQQueue.put（）方法行为的选项。

- **MQQueue**
  该类为 WebSphere MQ 队列提供查询、设置、放置和获取操作。查询和设置能力继承自 MQManagedObject。
- **MQQueueManager**
  该类代表 WebSphere MQ 的队列管理器。
- **MQSimpleConnectionManager**
  该类提供基本的连接集合功能。

12.3.3.2 接口

WebSphere MQ for Jave 具有以下接口：

- **MQReceiveExit**
  该接口使得我们可以用 WebSphere MQ for Java 检查并有可能修改从队列管理器接收的数据。当以绑定模式直接连接到 WebSphere MQ 时，该接口不适用。
- **MQSecurityExit**
  该接口使得我们可以尝试定制连接到队列管理器时出现的安全流。当以绑定模式直接连

接到 WebSphere MQ 时，这一接口不适用。

- **MQSendExit**

该接口使得我们可以检查并有可能修改用 WebSphere MQ Client for Java 发送到队列管理器的数据。当以绑定模式直接连接到 WebSphere MQ 时，这一接口不适用。

12.4 用 WebSphere MQ Java API 开展工作

我们在本节中将探讨利用 WebSphere MQ Java API 进行编程的方法。

12.4.1 设置连接

在本节中，我们将看看绑定模式和客户机连接模式是如何实现的。我们假定从设计的观点出发，您已经决定用绑定模式或客户机连接模式实现，下面我们就来讲解一下应当如何来实现的方法。

我们通过 MQQueueManager 类的构造器调用获得到队列管理器的连接。在这个时候，我们所获得连接的类型是由 MQEnvironment 类的某些静态字段决定的。区别不同连接模式的静态字段设置分别是主机、通道、userId 和口令。在这些用以连接到队列管理器的 MQEnvironment 字段中，最能区别出绑定模式和客户机连接模式的两个字段设置就是主机和通道。在绑定模式中，除了 userId 和口令字段外，您不必为这些字段中的任何一个设置值。您也可以选择在绑定模式中设置它们。

- MQEnvironment.hostName

对客户机连接而言，我们应当将此设为队列管理器所在主机的主机名。由于该主机名用于到队列管理器运行机器的 TCP/IP 连接，因此其值不区分大小写，请看下面的例子：
MQEnvironment.host = “machinename.dmain.com” ；

- MQEnvironment.channel

这是客户机连接通道的名。该字段的值是区分大小写的。一般说来，它就是队列管理器下面服务器连接通道的名。是一个双向链接，它使在客户机和队列管理器之间的 MQI 调用和回复成为可能。对客户机连接而言，我们应当将其设为应用程序尝试连接的队列管理器下面服务器连接通道的名，请看下面的例子：
MQEnvironment.channel = “JAVA.CLIENT.CHNL”；

- MQEnvironment.port

端口号是一个可选字段。在默认情况下，客户机会尝试在主机的 1414 号端口上连接到队列管理器。1414 号端口是 WebSphere MQ 监听器默认使用的端口。如果该端口号与默认的不同，那么您可以用 MQEnvironment.port 字段来指定端口号，请看下面的例子：
MQEnvironment.port = nnnn；

- MQEnvironment.userId 和 MQEnvironment.password

userId 和口令字段在默认情况下是空的。您可以通过设置 userId 和口令字段的值来指定 userId 和口令，请看下面的例子：
MQEnvironment.userId = “userXYZ” ；

MQEnvironment.password = “password”；

- MQEnvironment.properties

这是定义 WebSphere MQ 环境的关键值对的散列表。如果您不是使用 VisiBroker 连接的话，那么就应当将该字段在绑定和客户机连接情况下都做如下设置：
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_WEBSPHERE MQ

MQEnvironment 类中的变量控制着到队列管理器的连接调用。设置连接到队列管理器的第一步就是根据连接模式的类型来设置 MQEnvironment 字段， 我们通过创建 MQQueueManager 类的新的实例、发出 MQQueueManager 类的构造器调用来获得到队列管理器的连接。MQQueueManager 类有过载的构造器。我们要根据创建 MQQueueManager 类新实例时提供的参数来调用合适的构造器建立连接。在最简单的情况中，您可以提供队列管理器名作为字符串，从而创建 QueueManager 类的新实例，请看下面的例子：
MQQueueManager qmgr = new MQQueueManager（“ITSOG.QMGR1”）；
在这里，ITSOG.QMGR1 是队列管理器名。上面的方法在绑定模式和客户机连接模式中都有效。

在第二种方法中，您可以提供队列管理器名以及具有设置环境选项关键值对的散列表，从而创建 MQQueueManager 类的新实例。利用这种方法时，提供的属性会覆盖 MQEnvironment 类中设置的值。如果您希望在队列管理器到队列管理器的情况下设置环境值，那么您就可以使用此方法。请看下面的例子：
MQQueueManager qmgr = new MQQueueManager（queueManagerName ,propertiesHashTable）；

第三种方法就是通过提供队列管理器名和队列管理器打开选项，从而创建 MQQueueManager 类的新实例（它是一个整数字段）。只有在绑定模式中才能使用该方法。选项字段使您可以在快速绑定或正常绑定间作出选择。请看下面的例子：
MQQueueManager qmgr = new MQQueueMager（queueManagerName ，
MQC.MQCNO_FASTPATH_BINDING ）；

12.4.2 打开队列

为了对队列进行操作，我们首先应当通过打开队列以获得队列句柄或队列对象。打开队列有两种方法。我们可以利用 MQQueueManager 对象的 accessQueue 方法，也可以通过调用 MQQueue 类的构造器。

这两种不同调用的形式如下：
MQQueue queue = qmgr.accessQueue（“qName’, openOption, “qMgrName” ，
“dynamicQname”, “alternateUserId”）；

使用 MQQueue 类构造器的第二种方法，需要添加一个队列管理器参数，
MQQueue queue = new MQQueue（qmgr, “qName’, openOption, “qMgrName” ，
“dynamicQname”, “alternateUserId”）；

WebSphere MQ 将在打开队列过程中根据用户认证保证 openOption 的有效性。

MQQueue 类的对象代表着队列。它既拥有有助于消息发送（即放置、获取、设置、查询）的方法，也有对应于队列属性的属性。

12.4.3 处理 WebSphere MQ 消息

MQMessage 类的对象代表着将被放置到队列上或将从队列获取的消息。它既包括应用程序数据，又包括 MQMD。既具有对应于 MQMD 字段的属性，又具有向消息写入或从消息读取不同数据类型的应用程序数据的方法。在应用程序中，MQMessage 代表着一个缓冲区。应用程序不必声明缓冲区的大小，因为它会随着写入的数据而不断改变。但是，如果消息大小超过了队列的 MaximumMessageLength 属性的话，您就不能将消息放到队列中。

为了创建消息，您应当创建 MQMessage 类的新实例。我们利用 writeXXX 方法根据特定应用程序数据类型将应用程序数据写入消息。数字和字符串等数据类型格式可以通过 characterSet 和编码等 MQMD 属性来控制。我们可以在放置消息到队列上之前设置 MQMD 字段，也可以在从队列获取消息时读取 MQMD 字段。应用程序通过设置合适的放置或获取操作选项，从而控制着消息放置到队列或从队列获取消息的方式。我们通过设置合适的放置消息选项值来控制消息放置到队列的方式。同样，我们也可以通过设置合适的获取消息选项来控制从队列接收消息的方式。

- 放置消息选项

消息放置到队列上的方式是由 MQPutMessageOptions 类实例的选项字段的值来决定的。我们可以利用 WebSphere MQ 常量接口 MQC 的 MQPMO 结构来设置选项的值。请看下面的例子：

```
MQPutMessageOptions pmo = new MQPutMessageOption（）；
```

MQPutMessageOptions 类的实例，其选项属性的值设置为默认值。这在大多数简单消息发送情境中都已经足够了。您可以利用 WebSphere MQ 常量接口 MQC 的 MQPMO 结构来设置任意特定的选项，例如：

```
pmo.options = pmo.options + MQC.MQPMO_NEW_MSG_ID
```

上面的例子设置了选项字段的值，指令队列管理器为消息生成新的消息 ID 并将其设为 MQMD 的 MsgId 字段。

- 获取消息选项

从队列接收消息的方式是由 MQGetMessageOptions 类实例的选项字段的值决定的。我们可以利用 WebSphere MQ Constants MQC 的 MQOO 结构来设置选项的值。请看下面的例子：

```
MQGetMessageOptions gmo = new MQGetMessageOption（）；
```

MQGetMessageOptions 类的新实例将选项属性的值设为默认值。您可以利用 MQPOO 结构来设置合适的获取消息选项。请看下面的例子：

```
gmo.options = gmo.options + MQC.MQGMO_NO_WAIT；
```

以上选项指定了如果队列上没有消息的话，那么获取消息调用将立即返回。发送消息我们利

用 MQQueue 类的 put（MQMessage message）或 put（MQMessage message, MQPutMessageOptions pmo）方法来发送消息。放置方法调用控制着消息放置到队列上的方式。

- 获取消息

我们利用 MQQueue 类的 get （ MQMessage message ） 或 get （ MQMessage, MQGetMessageOptions gmo）、get（MQMessage, MQGetMessageOptions gmo, int max MessageSize）方法来从 WebSphere MQ 队列接收消息。所有从给定 MQQueueManager 到 WebSphere MQ 的调用都是同步的。

注意：

如果您进行带有等待的获取调用的话，那么直到获取调用完成之前，所有其他利用相同 MQQueueManager 的线程都将被封锁，不能发出进一步的 WebSphere MQ 调用。如果
您需要多线程同时来访问 WebSphere MQ 的话，那么每个线程都必须创建其自己的 MQQueueManager 对象。

如果没有指定 MaxMessageSize 的话，那么将自动调整消息缓冲区长度为将要到达的消息的大小。如果您以获取方法调用使用 MaxMessageSize 的话，那么该调用将能接收最大的消息。如果队列上的消息比它还要大的话，那么就会出现下面两种情况之一：

1. 如果 MQC.MQGMO_ACCEPT_TRUNCATED_MSG 标记在 MQGetMessageOptions 对象的选项元素变量中得到设置的话，那么将根据指定缓冲区的大小向消息填充最多的数据，并且会返回完成代码为 MQException.MQCC_WARNING 和原因代码为 MQException.MQRC_TRUNCATED_MSG_ACCEPTED 的结果。

2. 如果没有设置 MQC.MQGMO_ACCEPT_TRUNCATED_MSG 标记的话，那么消息将被留在队列上，并且会返回完成代码是 MQException.MQCC_WARNING 和原因代码是 MQeception.MQRC_TRUNCATED_MSG_FAILED 的结果。

12.5 应用程序开发

在本节中，我们将探讨发送-遗忘、请求/回复和消息分组点到点消息发送模式的实现。在点到点模式中，应用程序成对活动。我们称作发送器的发送应用程序将消息放置在发送方的 WebSphere MQ 应用程序队列上。在目的地系统或接收方上，我们称作接收器的应用程序从 WebSphere MQ 应用程序队列接收消息。因此，发送器和接收器应用程序是成对活动的，实现了在来源和目的地系统之间的数据移动或消息发送。

在我们所举的这些例子中，用到了到队列管理器的客户机连接。在这些例子中用到的 WebSphere MQ 对象是在主机 ITSOG 上称作 ITSOG.QMGR1 的队列管理器。用于客户机连接的通道是 JAVA.CLIENT.CHNL，端口就是默认端口 1414。我们所用的应用程序队列是 SAMPLE.QUEUE。

12.5.1 简单的消息发送器应用程序

我们的第一个点到点客户机程序将创建一个简单的消息并发送它到 WebSphere MQ 队列。我们还将讲解处理发送器发送消息的接收器程序。

有关步骤如下：

调入 WebSphere MQ Java API package；
为客户机连接设置环境属性；
连接到队列管理器；
为打开 WebSphere MQ 队列设置选项；
为发送消息打开应用程序队列；
设置选项，放置消息到应用程序队列上；
创建消息缓冲区；
使用用户数据和任何消息描述器字段准备消息；
放置消息到队列上。

以下程序 PtpSender.java 就是将在应用程序队列上发送消息的发送器应用程序：

```
import com.ibm.mq.*;
public class Typesetter {
public static void main（String args[]） {
  try
  {
    String hostName = "ITSOG" ；
    String channel = "JAVA.CLIENT.CHNL" ；
    String qManager = "ITSOG.QMGR1" ；
    String qName = "SAMPLE.QUEUE" ；

    /*设置 MQEnvironment 属性以便客户机连接*/
    MQEnvironment.hostname = hostName ；
    MQEnvironment.channel = channel ；
    MQEnvironment.properties.put（MQC.TRANSPORT_PROPERTY,
                                  MQC.TRANSPORT_WEBSPHERE MQ）；
    /*连接到队列管理器*/
    MQQueueManager qMgr = new MQQueueManager（qManager） ；

    /*设置打开选项以便打开用于输出的队列，如果队列管理器正在停止，我们也已设置了
      选项去应对不成功情况。*/
    int openOptions = MQC.MQOO_OUTPUT | MQC.MQOO_FAIL_IF_QUIESCING ；

    /*打开队列*/
    MQQueue queue = qMgr.accessQueue（qName,
                                    openOptions,
                                    null,
                                    null,
```

```
                                        null）；

    /*设置放置消息选项我们将使用默认设置*/
    MQPutMessageOptions pmo = new MQPutMessageOptions（）；

    /*创建消息，MQMessage 类包含实际消息数据的数据缓冲区，和描述消息的所有
    MQMD 参数*/

    /*创建消息缓冲区*/
    MQMessage outMsg = new MQMessage（）；

    /*设置 MQMD 格式字段*/
    outMsg.format = MQC.MQFMT_STRING ；

    /*准备用户数据消息*/
    String msgString = "Test Message from PtpSender program "；
    outMsg.writeString（msgString）；

    /*在队列上放置消息*/
    queue.put（outMsg, pmo）；

    /*提交事务处理*/
    qMgr.commit（）；

    System.out.println（" The message has been Successfully put！\n"）；

    /*关闭队列和队列管理器对象*/
    queue.close（）；
    qMgr.disconnect（）；
  }
  catch （MQException ex）
  {
    System.out.println（"An MQ Error Occurred: Completion Code is :\t" +
                ex.completionCode + "\n\n The Reason Code is :\t" + ex.reasonCode ）；
    ex.printStackTrace（）；
  }
  catch（Exception e） {
    e.printStackTrace（）；
  }
  }
}
```

12.5.2 简单的消息接收应用程序

我们的下一个点到点客户机程序是消息接收器应用程序，它获取 PtpSender 应用程序所发送的消息并在控制台上将消息打印出来。

有关步骤如下：

调入 WebSphere MQ Java API package；
为客户机连接设置环境属性；
连接到队列管理器；
为打开 WebSphere MQ 队列设置选项；
为获取消息打开应用程序；
设置选项，从应用程序队列获取消息；
创建消息缓冲区；
从队列获取消息到消息缓冲区；
从消息缓冲区读取用户数据并在控制台上显示。

以下程序 PtpReceiver.java 就是将从应用程序队列上获取消息的接收应用程序：

```
import com.ibm.mq.* ；
public class PtpReceiver {
public static void main（String args[]） {
  try
  {
    String hostName = "ITSOG" ；
    String channel = "JAVA.CLIENT.CHNL" ；
    String qManager = "ITSOG.QMGR1" ；
    String qName = "SAMPLE.QUEUE" ；

    /设置 MQEnvironment 属性以便客户机连接*/
    MQEnvironment.hostname = hostName ；
    MQEnvironment.channel = channel ；

    MQEnvironment.properties.put（MQC.TRANSPORT_PROPERTY,
                                  MQC.TRANSPORT_WEBSPHERE MQ）；

    /*连接到队列管理器*/
    MQQueueManager qMgr = new MQQueueManager（qManager） ；

    /*设置打开选项以便打开用于输出的队列，如果队列管理器停止，我们也
    已设置了选项去应对不成功情况*/
    int openOptions = MQC.MQOO_INPUT_SHARED |
                      MQC.MQOO_FAIL_IF_QUIESCING ；

    /*打开队列*/
    MQQueue queue = qMgr.accessQueue（qName,
```

```
                                        openOptions,
                                        null,
                                        null,
                                        null）；

    /*设置放置消息选项*/
    MQGetMessageOptions gmo = new MQGetMessageOptions（）；

    /*在同步点控制下获取消息*/
    gmo.options = gmo.options + MQC.MQGMO_SYNCPOINT ；

    /*如果在队列上没有消息则等待*/
    gmo.options = gmo.options + MQC.MQGMO_WAIT ；

    /*如果队列管理器停顿则失败*/
    gmo.options = gmo.options + MQC.MQGMO_FAIL_IF_QUIESCING ；

    /*设置等待的时间限制*/
    gmo.waitInterval = 3000 ；

    /*创建 MQMessage 类*/
    MQMessage inMsg = new MQMessage（）；

    /*从队列到消息缓冲区获取消息*/
    queue.get（inMsg, gmo） ；

    /*从消息读取用户数据*/
    String msgString = inMsg.readString（inMsg.getMessageLength（））；
    System.out.println（" The Message from the Queue is : " + msgString）；

    /*提交事务*/
    qMgr.commit（）；

    /*关闭队列和队列管理器对象*/
    queue.close（）；
    qMgr.disconnect（）；
  }
  catch （MQException ex）
  {
    System.out.println（"An MQ Error Occurred: Completion Code is :\t" +
            ex.completionCode + "\n\n The Reason Code is :\t" + ex.reasonCode ）；
    ex.printStackTrace（）；
  }
  catch（Exception e） {
```

```
    e.printStackTrace（）；
  }
  }
}
```

12.5.3 请求/回复

在请求/回复消息发送模式中，一个应用程序发送一条消息（请求消息）到另一个回复应用程序（回复生成器）再到请求消息。生成回复的应用程序获取请求消息、处理请求，并向请求应用程序发出回复。回复发送到由请求消息的消息标题属性 replyToQueueManager 指定的队列。请求应用程序在放置消息到队列上之前会在请求消息上设置这些消息标题属性。

请求应用程序让队列管理器生成唯一的 messageId，以及回复应用程序拷贝请求消息的 messageId 到回复消息的 correlationId 上。请求应用程序使用回复消息的 correlationId 值，将回复映射回原始的请求。

我们将利用一对简单的应用程序来讲解请求回复模式。第一个应用程序（我们称作请求器）放置一条简单的消息到队列（请求队列）上。请求器在放置请求消息到队列上之前会在请求消息上设置 replyToQueue 和 replyToQueueManager 消息标题属性。而后，它将打开回复队列并等待 correlationId 匹配已发出请求消息的 messageId 值的消息。服务于请求消息的回复应用程序获取消息，准备回复消息，并将它发送到请求消息指定的队列管理器下的回复队列上。它还将从请求消息拷贝 messageId 到回复消息的 correlationId 消息标题字段。

应用程序 Requester.java 即发送请求消息并且等待从回复应用程序获得回复的应用程序。有关步骤如下：

调入必要的包；
为客户机连接设置 MQEnvironment 属性；
连接到队列管理器；
打开请求队列以获得输出；
设置放置消息选项；
- 准备请求消息；
- 设置到队列名的回复；
设置到队列管理器名的回复；
放置请求消息到请求队列上；
关闭请求队列；
打开回复队列以获得输入；
设置获取消息选项；
- 设置选项，匹配回复消息上的 correlationID；
- 用等待（等待匹配 correlationId 的回复消息）在回复队列上发出获取。

注意：

我们建议您为回复消息在获取调用上使用确定的等待时间。等待间隔可以设为系统所允许的等待回复的最大时间。

```
import com.ibm.mq.*;
public class Requester {
public static void main（String args[]） {
  try
  {
    String hostName = "ITSOG" ；
    String channel = "JAVA.CLIENT.CHNL" ；
    String qManager = "ITSOG.QMGR1" ；
    String requestQueue = "SAMPLE.REQUEST" ；
    String replyToQueue = "SAMPLE.REPLY" ；
    String replyToQueueManager = "ITSOG.QMGR1" ；

    /*设置 MQEnvironment 属性以便客户机连接*/
    MQEnvironment.hostname = hostName ；
    MQEnvironment.channel = channel ；
    MQEnvironment.properties.put（MQC.TRANSPORT_PROPERTY,
                                  MQC.TRANSPORT_WEBSPHERE MQ）；

    /*连接到队列管理器*/
    MQQueueManager qMgr = new MQQueueManager（qManager） ；

    /*设置打开选项以便打开用于输出的队列，如果队列管理器停止，我们也已设置了选项
    去应对不成功情况*/
    int openOptions = MQC.MQOO_OUTPUT | MQC.MQOO_FAIL_IF_QUIESCING ；

    /*打开打开队列*/
    MQQueue queue = qMgr.accessQueue（requestQueue,
                                     openOptions,
                                     null,
                                     null,
                                     null）；

    /*设置放置消息选项，我们将使用默认设置*/
    MQPutMessageOptions pmo = new MQPutMessageOptions（）；
    pmo.options = pmo.options + MQC.MQPMO_NEW_MSG_ID ；
    pmo.options = pmo.options + MQC.MQPMO_SYNCPOINT ；

    /*创建消息缓冲区*/
    MQMessage outMsg = new MQMessage（）；

    /*设置 MQMD 格式字段*/
    outMsg.format = MQC.MQFMT_STRING ；
    outMsg.messageFlags = MQC.MQMT_REQUEST ；
    outMsg.replyToQueueName = replyToQueue；
```

```
    outMsg.replyToQueueManagerName = replyToQueueManager ;

    /*准备用户数据消息*/
    String msgString = "Test Request Message from Requester program ";
    outMsg.writeString（msgString）;

    /*在队列上放置消息*/
    queue.put（outMsg, pmo）;

    /*提交事务*/
    qMgr.commit（）;

    System.out.println（" The message has been Successfully put\n"）;

    /*关闭请求队列*/
    queue.close（）;

    /*设置打开选项以便队列响应*/
    openOptions = MQC.MQOO_INPUT_SHARED | MQC.MQOO_FAIL_IF_QUIESCING ;
    MQQueue respQueue = qMgr.accessQueue（replyToQueue,
                                          openOptions,
                                          null,
                                          null,
                                          null）;
    MQMessage respMessage = new MQMessage（）;
    MQGetMessageOptions gmo = new MQGetMessageOptions（）;

    /*在同步点控制下获取消息*/
    gmo.options = gmo.options + MQC.MQGMO_SYNCPOINT ;
    gmo.options = gmo.options + MQC.MQGMO_WAIT ;
    gmo.matchOptions = MQC.MQMO_MATCH_CORREL_ID;
    gmo.waitInterval = 10000 ;
    respMessage.correlationId = outMsg.messageId ;

    /*获取响应消息*/
    respQueue.get（respMessage, gmo）;
    String response = respMessage.readString（respMessage.getMessageLength（））;
    System.out.println（"The response message is : " + response）;
    qMgr.commit（）;
    respQueue.close（）;
    qMgr.disconnect（）;
  }
  catch （MQException ex）
  {
```

```
      System.out.println（"An MQ Error Occurred: Completion Code is :\t" +
        ex.completionCode + "\n\n The Reason Code is :\t" + ex.reasonCode ）；
        ex.printStackTrace（）；
   }
   catch（Exception e） {
     e.printStackTrace（）；
   }
   }
}
```

12.5.4 回复应用程序

回复器应用程序 Responder.java 处理来自请求队列的请求消息并发送回复到请求应用程序指定的请求队列上。

```
import com.ibm.mq.* ；
public class Responder {
   public static void main（String args[]） {
     try
     {
        String hostName = "ITSOG" ；
        String channel = "JAVA.CLIENT.CHNL" ；
        String qManager = "ITSOG.QMGR1" ；
        String qName = "SAMPLE.REQUEST" ；

        /*设置 MQEnvironment 属性以便客户机连接*/
        MQEnvironment.hostname = hostName ；
        MQEnvironment.channel = channel ；
        MQEnvironment.properties.put（MQC.TRANSPORT_PROPERTY,
                                      MQC.TRANSPORT_WEBSPHERE MQ）；

        /*连接到队列管理器*/
        MQQueueManager qMgr = new MQQueueManager（qManager） ；

        /*设置打开选项以便打开用于输出的队列，如果队列管理器停止，我们也
        已设置了选项去应对不成功情况*/
        int openOptions = MQC.MQOO_INPUT_SHARED |
                          MQC.MQOO_FAIL_IF_QUIESCING ；

        /*打开队列*/
        MQQueue queue = qMgr.accessQueue（qName,
                                          openOptions,
```

```
                                        null,
                                        null,
                                        null）；

    /*设置放置消息选项*/
    MQGetMessageOptions gmo = new MQGetMessageOptions（）；

    /*在同步点控制下取消息*/
    gmo.options = gmo.options + MQC.MQGMO_SYNCPOINT ；

    /*如果队列上没有消息则等待*/
    gmo.options = gmo.options + MQC.MQGMO_WAIT ；

    /*如果队列管理器停止则失败*/
    gmo.options = gmo.options + MQC.MQGMO_FAIL_IF_QUIESCING ；

    /*设置等待的时间限制*/
    gmo.waitInterval = 3000 ；

    /*创建 MQMessage 类*/
    MQMessage inMsg = new MQMessage（）；

    /*从队列到队列缓冲区获取消息*/
    queue.get（inMsg, gmo） ；

    /*从消息读用户数据*/
    String msgString = inMsg.readString（inMsg.getMessageLength（））；
    System.out.println（" The Message from the Queue is : " + msgString）；

    /*检查看消息是否属于类型请求消息并对该请求回复*/
    if （inMsg.messageFlags == MQC.MQMT_REQUEST ） {
        System.out.println（"Preparing To Reply To the Request " ）；
        String replyQueueName = inMsg.replyToQueueName ；
        openOptions = MQC.MQOO_OUTPUT | MQC.MQOO_FAIL_IF_QUIESCING ；
        MQQueue respQueue = qMgr.accessQueue（replyQueueName,
                                              openOptions,
                                            inMsg.replyToQueueManagerName,
                                              null,
                                              null）；
    MQMessage respMessage = new MQMessage（） ；
    respMessage.correlationId = inMsg.MessageId；
    MQPutMessageOptions pmo = new MQPutMessageOptions（）；
    respMessage.format = MQC.MQFMT_STRING ；
    respMessage.messageFlags = MQC.MQMT_REPLY ；
```

```
        String response = "Reply from the Responder Program " ;
        respMessage.writeString（response）;
        respQueue.put（respMessage, pmo）;
        System.out.println（"The response Successfully send "）;
        qMgr.commit（）;
        respQueue.close（）;
      }
      queue.close（）;
      qMgr.disconnect（）;
   }
   catch （MQException ex）
   {
      System.out.println（"An MQ Error Occurred: Completion Code is :\t" +
        ex.completionCode + "\n\n The Reason Code is :\t" + ex.reasonCode ）;
        ex.printStackTrace（）;
   }
   catch（Exception e） {
          e.printStackTrace（）;
   }
   }
}
```

12.5.5 消息分组

应用程序可能需要将一系列更新分组归入一个工作单位中。这样的更新通常都是逻辑上相互关联的，为了保持数据的完整性，更新必须全部成功。如果一个更新成功而另一个更新失败的话，那么就会丢失数据完整性。WebSphere MQ 支持事务处理消息发送。当一个工作单位成功完成时就会被提交。在这时，所有在这个工作单位中所做的更新都将成为永久性的和不可逆的。如果工作单位失败的话，那么所有更新将都被取消。同步点协同就是工作单位在保持其完整性的情况下被提交或取消的过程。逻辑消息组中的逻辑消息是由 GroupId 和 MsgSeqNumber 字段确认的。MsgSeqNumber 从组中第一条消息由 1 开始，如果消息不在组中，那么该字段的值就是 1。

组中的逻辑消息用途如下：

保证排序（如果这在消息传输环境中不能得到保证的话）。

允许应用程序将相似的消息分在一组（例如，那些必须全部由相同服务器实例来处理的消息）。

组中的每条消息如果不是分为段的话，那么就是由一个物理消息构成的。每条消息在逻辑上都是分开的，只有 MQMD 中的 GroupId 和 MsgSeqNumber 字段需要与组中的其他消息关联。MQMD 中的其他字段都是独立的；一些字段可能对组中所有消息都是相同的，而其他字段可能会不同。举例来说，组中的消息可能有着不同的格式名、CCSIDs、编码等。

简单的组发送器应用程序

我们的下一个程序范例 GroupSender.java 将讲解如何发送在组中的消息。该应用程序将把 10 条简单的消息作为工作单位中的一个组放置到队列上。

有关步骤如下：

调入必需的包；

为客户机连接设置 MQEnvironment 属性；

连接到队列管理器；

设置队列打开选项以获得输出；

打开队列以获得输出；

设置放置消息选项；

- 设置选项，维护消息的逻辑顺序；
- 设置选项，请求队列管理器生成 GroupId；

创建消息缓冲区；

设置消息标题属性；

- 设置 messageFlags 属性，从而显示出消息位于组中；
- 给 String 设置格式属性；

创建个体消息并将其放置到队列上；

在组中的最后一条消息上设置 messageFlags 属性，从而显示出消息是组中的最后一条；

提交事务处理。

```
import com.ibm.mq.*;
public class GroupReceiver {
   private MQQueueManager qmgr;
   private MQQueue outQueue;
   private String queueName = "SAMPLE.QUEUE" ;
   private String host = "ITSOG" ;
   private String channel = "JAVA.CLIENT.CHNL" ;
   private String qmgrName = "ITSOG.QMGR1" ;
   private MQMessage outMsg;
   private MQGetMessageOptions pmo;
   public static void main （String args[]） {
      GroupReceiver gs = new GroupSender（）;
      gs.runGoupSender（）;
   }
   public void runGoupSender（） {
   try {
      init（）;
      sendGroupMessages（）;
      qmgr.commit（）;
      System.out.println（"\n Messages successfully Send " ） ;
   }
   catch （MQException mqe） {
      mqe.printStackTrace（）;
      try{
         System.out.println（"\n Backing out Transaction " ） ;
```

```
        qmgr.backout（）；
        System.exit（2）；
      }
   catch（Exception e） {
      e.printStackTrace（）；
      System.exit（2）；
   }
}
catch（Exception e） {
   e.printStackTrace（）；
   System.exit（2）；
}
}
private void init（） throws Exception {
      /设置 MQEnvironment 属性以便客户机连接*/
      MQEnvironment.hostname = host ；
      MQEnvironment.channel = channel ；
      MQEnvironment.properties.put（MQC.TRANSPORT_PROPERTY,
                                    MQC.TRANSPORT_WEBSPHERE MQ）；

      /*连接到队列管理器*/
      qmgr = new MQQueueManager （ qmgrName）；

      /*设置队列打开选项以便输出*/
      int opnOptn =MQC.MQOO_OUTPUT |MQC.MQOO_FAIL_IF_QUIESCING ；

      outQueue = qmgr.accessQueue （ queueName , opnOptn,null,null,null ） ；
}
private void getGroupMessages（） throws Exception {
      /*设置放置消息选项*/
      pmo =new MQPutMessageOptions（）；
      pmo.options =pmo.options +MQC.MQPMO_LOGICAL_ORDER ；
      pmo.options =pmo.options +MQC.MQPMRF_GROUP_ID ；
      outMsg =new MQMessage（）；
      /*设置消息标记，表示该消息属于组*/
      outMsg.messageFlags =MQC.MQMF_MSG_IN_GROUP ；

      /*把消息格式设置成串*/
      outMsg.format =MQC.MQFMT_STRING ；
      String msgData =null；

      /*把 10 个简单消息作为一组发送*/
      int i =10；
      while（i >0）{
```

```
            msgData ="This is the "+i +"th message in the group ";
            outMsg.writeString（msgData）;
            if （i ==1）
               outMsg.messageFlags =MQC.MQMF_LAST_MSG_IN_GROUP ;
            i--;
            /*每次放置一个消息到队列）*/
            outQueue.put（outMsg,pmo）;

            /*清理缓冲区，以便重用*/
            outMsg.clearMessage（）;
        }
    }
}
```

12.5.6 简单的组接收应用程序

我们的下一个程序范例 GroupReceiver.java 将讲解如何在组中获取消息。该应用程序将在组里（通过 GroupSender 应用程序放置）获取消息，该组是以工作单位中作为组的队列。

有关步骤如下：

调入必需的包；
为客户机连接设置 MQEnvironment 属性；
连接到队列管理器；
设置队列打开选项以获得输入；
打开队列以获得输入；
设置放置消息选项；

- 设置选项以便在同步点控制下获得消息；
- 设置选项，当在组里提供所有消息时，以便只处理消息；
- 设置选项以便以逻辑顺序处理信息；

创建消息缓冲区；
设置消息标题属性；
创建个体消息并将其放置到队列上；
从队列上获取消息直到处理完最后消息；
在控制台上显示消息内容；
提交事务处理。

GroupReceiver.java 例子应用程序

```
import com.ibm.mq.*;
public class GroupReceiver {
   private MQQueueManager qmgr;
   private MQQueue inQueue;
   private String queueName ="SAMPLE.QUEUE";
```

```
private String host ="ITSOG"；
private String channel ="JAVA.CLIENT.CHNL"；
private String qmgrName ="ITSOG.QMGR1"；
private MQMessage inMsg；
private MQGetMessageOptions gmo；
public static void main （String args[]）{
  GroupReceiver gs =new GroupReceiver（）；
  gs.runGoupReceiver（）；
}
public void runGoupReceiver（） {
  try {
    init（）；
    getGroupMessages（）；
    qmgr.commit（）；
    System.out.println（"\n Messages successfully Send "）；
  }
  catch （MQException mqe）{
    mqe.printStackTrace（）；
    try{
      System.out.println（"\n Backing out Transaction "）；
      qmgr.backout（）；
      System.exit（2）；
    }
  catch（Exception e）{
    e.printStackTrace（）；
    System.exit（2）；
  }
}
catch（Exception e）{
  e.printStackTrace（）；
  System.exit（2）；
}
}
private void init（）throws Exception {
  /*为客户机连接设置 MQEnvironment 属性*/
  MQEnvironment.hostname =host ；
  MQEnvironment.channel =channel ；
  MQEnvironment.properties.put（MQC.TRANSPORT_PROPERTY,
  MQC.TRANSPORT_WEBSPHERE MQ）；

  /*连接到队列管理器*/
  qmgr =new MQQueueManager （qmgrName）；

  /*设置队列打开选项以输入*/
```

```
    int opnOptn =MQC.MQOO_INPUT_AS_Q_DEF |MQC.MQOO_FAIL_IF_QUIESCING ；

    /*打开队列以输入*/
    inQueue =qmgr.accessQueue （queueName ,opnOptn,null,null,null ）；
  }
  private void getGroupMessages（）throws Exception {
    /*设置获取消息选项*/
    gmo =new MQGetMessageOptions（）；
    gmo.options =MQC.MQGMO_FAIL_IF_QUIESCING；
    gmo.options =gmo.options +MQC.MQGMO_SYNCPOINT ；

    /*等待消息*/
    gmo.options =gmo.options +MQC.MQGMO_WAIT ；

    /*设置等待时间限制*/
    gmo.waitInterval =5000 ；

    /*只获取消息*/
    gmo.options =gmo.options +MQC.MQGMO_ALL_MSGS_AVAILABLE ；

    /*以逻辑顺序获取消息*/
    gmo.options =gmo.options +MQC.MQGMO_LOGICAL_ORDER ；
    gmo.matchOptions =MQC.MQMO_MATCH_GROUP_ID ；

    /*创建消息缓冲区*/
    inMsg =new MQMessage（）；
    String msgData =null；

    /*处理组消息*/
    while（true）{
      inQueue.get（inMsg,gmo）；
      int msgLength =inMsg.getMessageLength（）；
      msgData =inMsg.readString（msgLength）；
      System.out.println（"The message is \n "+msgData）；
      char x =gmo.groupStatus ；

      /*检查看是否是最后消息标记*/
      if（x ==MQC.MQGS_LAST_MSG_IN_GROUP）{
        System.out.println（"B Last Msg in Group"）；
        break；
      }
      inMsg.clearMessage（）；
    }
  }
```

```
}
```

您如欲进行更深入的探讨，可参考《WebSphere MQ 发布/预订应用程序》，您可以从以下网址下载：http://www.ibm.com/redbooks。

12.6 本章小结

本章介绍了如何利用 WebSphere MQ for Java 编程。并且为那些希望编写与 WebSphere MQ 相连接的 Java 应用程序或小程序的程序员提供所需的信息。WebSphere MQ Base Java 可以使用 Java applets、应用程序或 servlets 来调用和查询 WebSphere MQ。在客户端机器上不用安装任何 WebSphere MQ 程序，直接通过 WebSphere MQ for Java 作为一个 Internet 终端用户就可以参与交易的执行。

12.7 本章练习

1.使用 WebSphere MQ Java API 编写文件的发送程序。
2.使用 WebSphere MQ Java API 编写文件的接收程序。

第十三章 用 ActiveX 编程

目标

学习使用 WebSphere MQ automatin classes for ActiveX 编程。

13.1 概述

WebSphere MQ Automation Classes for ActiveX 是又一组允许程序员处理队列管理器对象的 API。WebSphere MQ Automation Classes for ActiveX 的组成部分为希望开发可在 Windows 平台上运行的 WebSphere MQ 应用程序的设计人员和程序员们提供了可用的类。因为我们可以利用实现语言的本身语法来编写 WebSphere MQ 对象的代码，因此这些类可以很容易地被整合到任意应用程序中。应用程序的整体设计与任何 WebSphere MQ 应用程序的设计都是一样的。

我们在这里要重点指出的是那些与 ActiveX 相关的概念。ActiveX 组件是以组件对象模型（Component Object Model，缩写为 COM）为基础的，它是由 Microsoft 定义的基于对象的编程模型。该模型确定了如何使软件组件可以确定彼此的位置并进行相互通讯，而不必在乎使用的是何种计算机语言或其物理位置如何。

COM 是形成更高级软件服务（如 OLE 提供的服务）基础的底层架构。COM 简化了

强大的基于组件的应用程序的开发工作。从其原始的单一机器应用，COM 已经扩展到允许访问其他系统的组件。这一新的模型被称作分布式 COM（Distributed COM），也被简称为 DCOM。

DCOM 使得我们可以利用组件来创建网络式应用程序。它扩展了 COM，可以在局域网、广域网甚至因特网上支持不同电脑间的通讯。利用 DCOM，应用程序的分布位置可以做到对客户和应用程序本身最有意义。COM+是 COM 的另一个延伸。它提供可从任何编程语言或工具使用的运行时间环境和服务，并且使得组件之间的大范围相互操作性成为可能，而不管实现的方式如何。COM+为从互动的对象创建软件系统提供了一种简单、强大的模型。与对象进行的所有通讯必须通过界面发生，并且所有通讯都必须看起来像简单的方法调用，即便目的地对象位于另一进程之中或在另一台机器上。

在利用 WebSphere MQ Automation Classes for ActiveX 设计 ActiveX 应用程序时，最重要的信息项目就是发送的消息或从远程 WebSphere MQ 系统中接收的消息。为了让 WebSphere MQ Automation Classes 脚本工作，发送方和接收应用程序都必须知道消息结构。而且，在考虑如何构建设计中的应用程序的实现时，我们应当记住，WebSphere MQ Automation Classes for ActiveX 脚本运行的机器与 WebSphere MQ 队列管理器或 WebSphere MQ 客户安装的机器相同。

WebSphere MQ Automation Classes for ActiveX 的主要特性如下：

提供对 WebSphere MQ API 所有函数和特性的访问。这就使得可与其他非 Windows 的 WebSphere MQ 平台建立完全的相互关联；

兼容 ActiveX 惯例；

兼容 WebSphere MQ 对象模型；

使得使用它的 ActiveX 应用程序能够在任何通过 WebSphere MQ 可以访问的企业系统上运行事务处理和访问数据。

WebSphere MQ Automation Classes for ActiveX 采用的是自由线程模型，在这种模型中，对象可在线程间使用。类允许使用 MQQueue 和 MQQueueManager 对象，但 WebSphere MQ 一般不允许在不同线程间分享句柄（the sharing of handles）。

尽管使用这些类有一定的好处，但同时也有一些限制。下面就是一些限制的例子：

如果打算用网络浏览器来访问应用程序的话，浏览器必须支持 ActiveX 控件（如 Microsoft Internet Explorer 3 或更高版本）。如果用 WebSphere MQ 向其执行的机器外发送数据的话，那么恶意脚本可能会钻这个空子并制造安全问题；

Automation Classes 常量对 VBScript 和 JavaScript 程序不可用，因此程序员必须对其进行硬编码（这些常量可在随 WebSphere MQ 产品提供的 cmqc.h 标题文件中找到）。

13.2 平台和语言

WebSphere MQ Automation Classes for ActiveX 只能用于 32 位 ActiveX 脚本客户机。因此，这些类只能用于以下平台：

Windows 98/95

Windows NT

Windows 2000

我们可以利用支持 COM 对象创建和使用的语言来编写使用 WebSphere MQ Classes for ActiveX 的应用程序，例如 Visual Basic、Java 和其他 ActiveX 脚本客户机。这些类可以简单地整合到应用程序中，因为我们可以用实现语言的本身语法来编写 WebSphere MQ 对象的代码。为了在 WebSphere MQ 服务器环境中运行 ActiveX 组件，您必须拥有 Windows

NT 4.0 （如果打算使用 MTS 作为事务处理协同器的话，还应具备 Service Pack 6 和 Option Pack4）或 Windows 2000 和 WebSphere MQ 版本 5.1 或更高。

13.3 库

如果应用程序在使用 WebSphere MQ Automation Classes for ActiveX 的话，它将需要 MQAX200.dll 链接库。这个库可以在 WebSphere MQ Base Directory\bin 下找到。当编写应用程序时，其中应包括该库。在 Visual Basic 中，我们可以通过向程序引用添加 mqax200.dll 来实现此目的。欲了解更多信息，请参见 Visual Basic 产品文档。

13.4 架构模型

WebSphere MQ Automation Classes for ActiveX 提供以下对象：
MQSession：这是 WebSphere MQ Automation Classes for ActiveX 的主类。它包含对任何其他类进行的上一次动作的状态。每个 ActiveX 进程只有一个 MQSession 对象。如果尝试创建第二个此类对象的话，将创建原始对象的第二个引用；

MQQueueManager：此类提供到队列管理器的访问。调用该对象的方法和属性将发出通过 MQI 的调用。在此类的对象毁坏后，它将自动从队列管理器断开连接。可以通过此调用访问的某些属性是预备用户 ID、完成代码、连接状态和原因代码。这些属性大多数只有在对象连接到队列时才能进行访问。

MQQueue：此类提供到队列及其属性的访问，如当前深度、深度高度限制等。

MQMessage：此类代表着一条 WebSphere MQ 消息。该类不仅包括访问消息描述器的属性，还提供储存消息数据的缓冲区。它包括写入方法，可以从 ActiveX 应用程序中拷贝数据到 MQMessage 对象，还提供读取方法，可以从 MQMessage 对象中拷贝数据到 ActiveX 应用程序。这个类自动管理向缓冲区分配内存和取消其内存分配。在创建 MQMessage 对象时，应用程序不必声明缓冲区的大小，因为缓冲区会随着写入其中的数据大小而变化；

MQPutMessageOptions：此类包括所有控制放置消息行为的不同的选项；

MQGetMessageOptions：此类包括所有控制获取消息行为的不同的选项；

MQDistributionList：此类包括获得输出的本地、远程或假名队列的集合；

MQDistributionListItem：此类包括 MQOR、MQRR 和 MQMR 结构，并将其与拥有分配表（owning distribution list）相关联。

13.5 用 WebSphere MQ automatin classes for ActiveX 编程

在下面这节中，我们将看看如何利用 WebSphere MQ Automation Classes for ActiveX 连接和操作队列管理器对象。这节中的例子都以 Visual Basic 编写。

13.5.1 连接到队列管理器

WebSphere MQ Automation Classes for ActiveX 遵循与 AMI 相同的结构。您必须先创建会话对象，以便连接到队列管理器。为了创建 MQSession 类的实例，我们如下将采用 New 关键词：

Set SessionObject = New MQSession

一旦创建了会话对象，就必须建立到队列管理器的连接。我们利用 MQSession 类的 AccessQueueManager（）方法来实现这一目的。该方法实际上将连接队列管理器、打开队列管理器并设置初始属性值。

Set QueueManagerObject =SessionObject.AccessQueueManager（QmgrName as string）

该方法所需的唯一参数就是队列管理器名。如果需要访问默认队列管理器名的话，那么就使用空串，而不使用队列管理器名。我们可以利用 WebSphere MQ 客户来建立连接，也可以直接连接到 WebSphere MQ 服务器来建立连接。如果连接对象失败的话，那么将举出错误事件并设置对象的原因代码和完成代码以及 MQSession 对象的原因代码和完成代码。

下例显示了如何连接到默认队列管理器。

例，连接到默认队列管理器

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
```

另一种可以用来连接到队列管理器的方法就是采用 MQQueueManager 类的 Connect（）方法。在调用此方法前，必须设置队列管理器名。我们利用 MQQueueManager 类的 Name 属性来实现这一目的。

例 6-2 显示了如何利用上述调用来连接到名为 SAMPLE.OMGR1 的队列管理器。

例 *6-2* 连接到队列管理器

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Set MQSess = New MQSession
Set QMgr = New MQQueueManager
QMgr.Name=“SAMPLE.QMGR1”
QMgr.Connect
```

13.5.2 打开 WebSphere MQ 对象

一旦我们已经成功建立了到队列管理器的连接，接下来我们就可以打开 WebSphere MQ 对象了。这些对象可以是：

队列（远程队列、本地队列、动态队列）；
分配表；
消息。

为了打开队列，我们要使用 MQQueueManager（）类的 AccessQueue（）方法：

```
Set QueueObject = QueueManagerObject.AccessQueue（QueueName asstring,
                                                  OpenOption as Long,
                                                  QueueManagerName as string,
                                                  DynamicQueueName as string,
                                                  AlternateUserId as string）
```

QueueName 就是队列名。在尝试连接到队列管理器之前，队列必须被创建。OpenOption 参数是用来确定控制打开队列行为的选项的。最常用的打开选项如下：

MQOO_INPUT_SHARED：当应用程序需要从队列获取消息时使用。它在分享访问模式中打开队列，因此多个应用程序可以同时从队列接收消息。该选项只能用于本地队列、别名队列和模型队列。

MQOO_INPUT_EXCLUSIVE：当应用程序需要从队列获取消息时使用。它在独占模式中打开队列。该选项仅对本地队列、别名队列和模型队列有效。

MQOO_OUTPUT：如果应用程序将在队列中放置消息的话，则应使用该选项。它对所有类型的队列有效，包括远程队列和分配表。

QueueManagerName 也用来指定拥有队列的队列管理器名。该参数通常被省略，因为我们已经创建了引用队列管理器的 MQQueueManager 对象。如果 QueueName 是模型队列的话，那么将使用 DynamicQueueName 来向将创建的动态队列分配一个名。

例 6-3 显示了如何打开一个命名为 PTP.QUEUE.LOCAL 的队列，它定义在默认队列管理器中。

例 *6-3* 打开队列

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim ITSOQueue As MQQueue '* input queue object（输入队列对象）
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
Set ITSOQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”,MQOO_OUTPUT）
‘ If PTP.QUEUE.LOCAL needs to be opened to get and receive messages, the
PTP.QUEUE.LOCAL）
‘ call will look like this:（调用看起来将这样）
```

```
‘ Set ITSOQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”，_
‘ MQOO_OUTPUT + MQOO_INPUT_SHARED）
```

提示：如果要打开的队列是本地队列的话，那么请不要设置 QueueManagerName 或将其设为。如果将其设为拥有队列的远程队列管理器的名的话，那么将尝试打开远程队列的本地定义。

当需要发送消息到多个目的地时，我们可以利用 MQDistributionlist 对象。与 MQQueue 对象相似，我们需要创建一个此类型的对象。如下所示，我们可以利用 New 关键词来实现这一目的：

Set DistributionListObject = New MQDistributionList

创建 MQDistributionList 对象的引用后，我们应当指定引用到队列管理器（在此定义分配表）的 MQQueueManager 对象。此目的可以通过利用 MQDistributionList 对象的 ConnectionReference 属性来实现。

DistributionListObject.ConnectionReference = QueueManagerObject

一旦指定到队列管理器的引用后，那么最后要做的一件事就是将队列添加到分配表中。每个队列必须与 MQDistributionListItem 对象相关联。我们可以利用 MQDistributionList 类的 AddDistributionListItem（）方法来实现此目的。

Set DistributionListItemObject =
DistributionListObject.AddDistributionListItem（QueueName as string,
QueueMgrName as string）

QueueName 就是需要成为分配表一部分的队列的名，QueueMgrName 指的是拥有队列的队列管理器。

例 6-4 显示了如何发送一条消息到两个队列 PTP.QUEUE.LOCAL 和 PTP.QUEUE2.LOCAL，它们定义在队列管理器 SAMPLE.QMGR1 中：

例 *6-4* 发送消息

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim DistListItem1 As MQDistributionListItem
Dim DistListItem2 As MQDistributionListItem
Dim SampleMsg As MQMessage
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（"SAMPLE.QMGR1"）
Set DistList = New MQDistributionList
DistList.ConnectionReference = QMgr
Set DistListItem1 = DistList.AddDistributionListItem（“PTP.QUEUE.LOCAL”）
Set DistListItem2 = DistList.AddDistributionListItem（“PTP.QUEUE2.LOCAL”）
Set SampleMsg = MQSess.AccessMessage（）
SampleMsg.MessageData = “Sample Message”
DistList.OpenOptions = MQOO_OUTPUT
DistList.Open
DistList.Put SampleMsg
End Sub
```

另一个需要在发送或接收消息之前被创建的对象就是消息对象。正如我们在前面提到的那

样，消息对象类包含消息数据和访问消息描述器的属性。此类型的对象可以利用 MQSession 类的 AccessMessage（）来创建：
Set MessageObject = SessionObject.AccessMessage（）
消息数据由 MessageData 属性分配。举例来说，如果您需要发送消息“Sample Message”的话，那么该字符串必须被引用到消息对象上，见例 6-5。

例 *6-5* 分配
MessageObject.Message = “Sample Message”

我们可以简单地设置 CharacterSet 属性为队列管理器的编码字符集标识符（MQCCSI_Q_MGR）并将它作为字符串传递，从而把二进制数据传递到 WebSphere MQ 消息中。

13.5.3 基本操作

在第 6.5.2 节《打开 WebSphere MQ 对象》（见本书第 186 页）中，我们看到了如何创建并打开不同的队列管理器对象。现在，对象已经创建了，那么接下来我们将谈谈可以对这些对象进行哪些基本操作。基本操作包括获取消息和发送消息。
发送消息
在发送消息之前，必须用 MQOO_OUTPUT 选项打开队列。为了发送消息，必须使用 MQQueue 对象的 Get（）方法。

QueueObject.Put（MessageObject, PutMsgOptionsObject）

MessageObject 代表将要发送消息的对象。因此，这种类型的对象必须创建在调用之前。另外，我们可以指定 PutMsgOptions 对象，它包含控制放置操作的选项。如果没有指定 PutMsgOptions 对象的话，那么将使用默认的 PMOs。如果要使用该选项的话，那么就应当利用 MQSession 类的 AccessPutMessageOptions （） 对象来创建 MQPutMessageOptions。

下例显示了如何用 PMO 选项 MQPMO_NO_SYNCPOINT（放置消息时无同步点控制）将消息放入一个称作 PTP.QUEUE.LOCAL 的队列。
例， 放置消息

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim ITSOQueue As MQQueue '* input queue object（输入队列对象）
Dim PutOptions As MQPutMessageOptions '* put message options（放置消息选项）
Dim SampleMsg As MQMessage '* message object for put（放置消息对象）
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
Set SampleQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”,MQOO_OUTPUT）
Set SampleMsg = MQSess.AccessMessage（）
Set PutOptions = MQSess.AccessPutMessageOptions（）
```

```
SampleMsg.MessageData = “Sample Message”
PutOptions.Options = MQPMO_NO_SYNCPOINT
SampleQueue.put SampleMsg PutOptions
```

请记住，MQPutMessageOptions 对象包含控制放置消息到 WebSphere MQ 队列上这一行为的各种选项。

为了同时发送多条消息，我们要用到 MQDistributionListObject 类的 put（）方法。正如发送一条消息到单一队列一样，在开始发送多条消息之前，我们需要用 MQDistribution 类的 OpenOptions 属性以 MQOO_OUTPUT 选项打开分配表。然后，我们将用 MQDistributionList 类的 open（）方法打开分配表中定义的对象。

下例显示了如何发送一条消息到两个队列 PTP.QUEUE.LOCAL 和 PTP.QUEUE2.LOCAL，它们定义在队列管理器 SAMPLE.QMGR1 中。

例， 发送消息

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim DistListItem1 As MQDistributionListItem
Dim DistListItem2 As MQDistributionListItem
Dim SampleMsg As MQMessage
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（"SAMPLE.QMGR1"）
Set DistList = New MQDistributionList
DistList.ConnectionReference = QMgr
Set DistListItem1 = DistList.AddDistributionListItem（“PTP.QUEUE.LOCAL”）
Set DistListItem2 = DistList.AddDistributionListItem（“PTP.QUEUE2.LOCAL”）
Set SampleMsg = MQSess.AccessMessage（）
SampleMsg.MessageData = “Sample Message”
DistList.OpenOptions = MQOO_OUTPUT
DistList.Open
DistList.Put SampleMsg
```

接收消息

为了接收消息，我们要利用 MQQueue 类的 get（）方法。请记住，在我们可以接收消息前，必须用 MQOO_INPUT_EXCLUSIVE 或 MQOO_INPUT_SHARE 打开队列。

QueueObject.Get（MessageObject, GetMsgOptionsObject, MsgLength）

MessageObject 即代表将被接收消息的对象。因此，在调用前必须创建这种类型的对象。另外，我们可以指定 GetMsgOptionsObject，它包含控制获取操作的选项。但是，如果没有指定它的话，那么将使用默认的 GMOs。如果将使用该选项的话，那么我们应当用 MQSession 类的 AccessGetMessageOptions（）方法来创建 MQGetMessageOptions 对象。从 WebSphere MQ 接收消息有两种方法：

用 Visual Basic 计时器函数发出 get（）后等待，这是轮询（polling）的方法；

用 Wait 选项发出 get（）。设置 WaitInterval 属性以指定等待时间。如果正在运行的软件

是单线程的，而系统却是多线程的话，那么我们推荐采用此种方法。这将避免使您的系统无限期死锁。
如果 WebSphere MQ 应用程序是消息的发信方并且 WebSphere MQ 生成 AccountToken 、CorrelationId、GroupId 和 MessageId 的话，那么我们建议您利用 AccountTokenHex、CorrelationIdHex、GroupIdHex 和 MessageIdHex 属性，如果您希望查看它们的值或想对其进行任何操作(包括在消息中将其传递回 WebSphere MQ)。这样做的原因在于，WebSphere MQ 生成的值是 0 到 255 之间任意值（包括 0 和 255）的字节串。它们不是可打印的字符串。如果上方法成功的话，那么到来的消息的 MQMD 和消息数据会完全代替 MQMessage 对象的 MQMD 和消息数据。如果不成功的话，MQMessage 对象不会改变。如果消息缓冲区的内容没有定义的话，那么整个消息长度就设为应当接收到的消息的完全长度。如果未指定消息长度参数的话，那么消息缓冲区长度则自动调整为至少是将到达的消息的大小。

下例显示了如何从一个命名为 PTP.QUEUE.LOCAL 的队列获取消息，然后如何在命名为 txtMessageData 的文本框中显示消息。
例，获取消息

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim ITSOQueue As MQQueue '* input queue object（输入队列对象）
Dim GetOptions As MQGetMessageOptions '* get message options（获取消息选项）
Dim SampleMsg As MQMessage '* message object for put（放置消息对象）
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
Set SampleQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”,MQOO_INPUT_SHARED）
Set SampleMsg = MQSess.AccessMessage（）
Set GetOptions = MQSess.AccessGetMessageOptions（）
GetOptions.Options = GetOptions.Options Or MQGMO_NO_SYNCPOINT
SampleQueue.get SampleMsg GetOptions
txtMessageData.text = SampleMsg.MessageData
```

13.5.4 关闭对象

一旦已经发送或接收消息并且已经处理数据，我们就可以关闭对象了。我们可以利用需要关闭对象（队列管理器、队列或分配表）的 close（）方法来实现此目的。

```
QueueManagerObject.Close
QueueObject.Close
DistributionListObject.Close
```

如果要关闭动态队列而且我们就是创建动态队列者的话，那么我们在用 MQQueue 类的 CloseOptions 属性时，应当在关闭选项中指定下面选项中的任意一个：

```
MQCO_DELETE：删除队列；
```

MQCO_DELETE_PURGE：仅在清除所有消息后删除队列。

13.5.5 关闭连接

为了断开与队列管理器的连接，我们可以利用MQQueueManager 类的Disconnect（）方法：

QueueManagerObject.Disconnect

所有与MQQueueManager 对象相关联的队列对象都将不可用，且不能重新打开。任何未提交的改变（消息放置和获取）都被提交。

13.6 事务处理管理

用来控制事务处理的 API 调用取决于使用中的事务处理的类型。有以下三种情境：
当 WebSphere MQ 消息（本地工作单位）是唯一的资源时。在此种情况下，根据 MQPutMessageOptions 或 MQGetMessageOptions 对象的 MQPMO_SYNCPOINT 或 MQGMO_SYNCPOINT 选项的指定，事务处理由同步点控制下的第一个被发送或接收的消息启动。同一个工作单位可以包括多个消息。事务处理可以利用 Commit()方法提交，也可以利用 MQQueueManager 对象的 Backout（）方法取消。
下例显示了如何在同步点控制下放置消息。
例，在同步点下放置消息

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim ITSOQueue As MQQueue '* input queue object（输入队列对象）
Dim PutOptions As MQPutMessageOptions '* put message options（放置消息选项）
Dim SampleMsg As MQMessage '* message object for put（放置消息选项）
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
Set SampleQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”,MQOO_OUTPUT）
Set SampleMsg = MQSess.AccessMessage（）
Set PutOptions = MQSess.AccessPutMessageOptions（）
SampleMsg.MessageData = “Sample Message”
PutOptions.Options = MQPMO_SYNCPOINT
SampleQueue.put SampleMsg PutOptions
‘Perform any actions before the message is put on the queue
QMGr.Commit
```

下例显示了如何在同步点控制下获取消息。
例，在同步点下获取消息

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
```

```
Dim ITSOQueue As MQQueue '* input queue object（输入队列对象）
Dim GetOptions As MQGetMessageOptions '* get message options（获取消息选项）
Dim SampleMsg As MQMessage '* message object for put（放置消息对象）
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
Set SampleQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”,MQOO_INPUT_SHARED）
Set SampleMsg = MQSess.AccessMessage（）
Set GetOptions = MQSess.AccessGetMessageOptions（）
GetOptions.Options = GetOptions.Options Or MQGMO_SYNCPOINT
SampleQueue.get SampleMsg GetOptions
 ‘Perform any validations before the message is physically removed
 ‘from the queue. （执行任意确认）
....
QMgr.Commit
txtMessageData.text = SampleMsg.MessageData
...
 ‘If an error ocurred during the validation, don’t retreive the message and
 ‘display the error message
If MQSess.CompletionCode <> MQCC_OK Then
QMgr.Backout
ErrMsg = Err.Description
StrPos = InStr（ErrMsg, " "） ' * search for first blank（搜索第一个空格）
If StrPos > 0 Then
Print Left（ErrMsg, StrPos）
Else
Print Error（Err） '* print complete error object（打印完整的错误对象）
End If
Print ""
Print "WebSphere MQ Completion Code = " & MQSess.CompletionCode
Print "WebSphere MQ Reason Code = " & MQSess.ReasonCode
Print "（" & MQSess.ReasonName & "）"
End If
```

当 WebSphere MQ 作为扩展架构事务处理协同器（全局工作单位）时。事务处理必须在第一个可恢复资源（如关系数据库）改变前用 MQQueueManager 类的 begin（）方法显式启动。而后，工作单位可以利用 commit（）方法提交，或者利用 MQQueueManager 对象的 backout（）方法取消。

下例显示了在 WebSphere MQ 作为扩展架构事务处理协同器时我们可以如何利用 WebSphere MQ Automation Classes for ActiveX。

```
Dim MQSess As MQSession '* session object（会话对象）
Dim QMgr As MQQueueManager '* queue manager object（队列管理器对象）
Dim ITSOQueue As MQQueue '* input queue object（输入队列管理器对象）
Dim PutOptions As MQPutMessageOptions '* put message options（放置信息选项）
Dim SampleMsg As MQMessage '* message object for put（放置信息选项）
```

```
‘Connect to the database（连接到数据库）
…
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager（""）
Set SampleQueue = QMgr.AccessQueue（“PTP.QUEUE.LOCAL”,MQOO_OUTPUT）
Set SampleMsg = MQSess.AccessMessage（）
Set PutOptions = MQSess.AccessPutMessageOptions（）
QMgr.Begin
SampleMsg.MessageData = “Sample Message”
PutOptions.Options = MQPMO_SYNCPOINT
SampleQueue.put SampleMsg PutOptions
‘Perform any validations and update the table（执行任何确认以及升级表格）
…
QMGr.Commit
‘If an error ocurred during the validation, don’t put the message on the queue
‘and display the error message
If MQSess.CompletionCode <> MQCC_OK Then
QMgr.Backout
ErrMsg = Err.Description
StrPos = InStr（ErrMsg, " "） '* search for first blank（搜索第一个空格）
If StrPos > 0 Then
Print Left（ErrMsg, StrPos）
Else
Print Error（Err） '* print complete error object
End If
Print ""
Print "WebSphere MQ Completion Code = " & MQSess.CompletionCode
Print "WebSphere MQ Reason Code = " & MQSess.ReasonCode
Print "（" & MQSess.ReasonName & "）"
End If
```

当使用外部事务处理协同器（如 Microsoft 事务处理服务器）时。在这种情况下，事务处理由外部事务处理协同器的 API 调用控制。简单说就是，Microsoft 事务处理服务器从本质上是本地和远程计算机上 COM/ActiveX 组件的一个管理工具。

Microsoft 事务处理服务器一般与前端处理机代码（它是 Microsoft 事务处理服务器中对象的 COM 客户）一起使用，也同后端处理机服务（如数据库）一起使用。前端处理机代码可能是巨大的独立程序，也可能是基于因特网信息服务器的活动服务器页。前端处理机代码可能位于 Microsoft 事务处理服务器及其商业对象所处的同一台机器上，并通过 COM 相互连接。

前端处理器代码也可能位于另外一台机器上，通过 DCOM 相连接。在不同的情况下，我们可以利用不同的客户机来访问相同的 Microsoft 事务处理服务器商业对象。使用哪台客户机或者客户机如何连接到 Microsoft 事务处理服务器不会对 Microsoft 事务处理服务器及其商业对象造成任何影响。客户机对象可能由任何语言在任何支持 COM 客户机代码编写的

环境中写成。最常见的是用于巨大客户机的 Visual Basic 和用于小客户机的 VBScript 或 JavaScript。商业对象可以由任何支持 COM 服务器代码编写的语言写成。最常见的是 Visual Basuc、C++和 Java（Microsoft 的）。Microsoft 事务处理服务器被设计用来帮助用户在典型的中间层服务器上运行商业逻辑应用程序。它将工作分为活动，这些活动通常都是比较短的、独立的商业逻辑块。Microsoft 事务处理服务器推出了许多特性，能够简化可升级分布式应用程序的编写。在 Windows2000 中又推出了称作 COM+的新产品。该产品也是 Microsoft 事务处理服务器的一个发展。

13.7 分组

WebSphere MQ Automation Classes for ActiveX 允许在一个消息组中包含一系列相关的消息，并作为一个消息组被发送。组环境信息与每个消息一起发送，使得消息系列可以被保存，对接收应用程序可用。组身份定义在消息描述器结构中，我们可以通过 MQMessage 类来访问它.

除了最后一条消息具有 MQMF_LAST_MSG_IN_GROUP 选项外，组中的每条消息都必须具备 MQMF_MSG_IN_GROUP 选项。组中消息的顺序储存在 MQMD 结构的 MsgSeqNumber 字段中，它由队列管理器自动生成。

另外，队列管理器可以控制是否已完全接收消息组。如果只须显示完成的消息组的话，那么我们可以在获取消息选项结构中设置 MQGMO_ALL_MSGS_AVAILABLE 选项。

13.8 本章小结

本章概括介绍 MQSereis Automation Classes for ActiveX，讲解其定义，介绍我们能够如何利用其来处理 WebSphere MQ 队列管理器对象。

13.9 本章练习

1.使用 WebSphere MQ automatin classes for ActiveX 编写文件的发送程序。
2.使用 WebSphere MQ automatin classes for ActiveX 编写文件的接收程序。

第十四章 用 AMI 编程

目标

学习使用 WebSphere MQ AMI 编程。

14.1 概述

应用程序消息接口（AMI）是对现有 WebSphere MQ API 的最新补充。其可向程序员提供一种可以用于处理队列管理器对象非常简单的接口。利用 AMI，程序员不必深入了解所有 MQI 调用，他们只要专注于应用程序的商业逻辑即可。这就意味着在编程时出现的错误更少，具有更高的处理业务及技术改变的灵活性。AMI 减少了编写新应用程序所需的代码数量。

每种编程模式都有不同的调用。举例说来，如果程序员希望编写发布/预订应用程序代码，那么他将必须采用不同于编写请求/回复应用程序的程序员所采用的调用。单个函数之中的结构更少，但是动词会更多。许多函数现在都是中间件层的一部分，在中间件层，企业确立的一系列策略代表应用程序得以应用。AMI 包括了 Java、C 和 C++等在内的标准编程语言的接口。

AMI 允许中央集中式控制和灵活的改变管理，为点到点模型提供高级接口，可以进行发布/预订工作。AMI 的重要特点是，其是独立于运输和消息发送基础设施的。

可采用以下方式发送和接收消息 AMI：

- 发送-遗忘，不需要回复。
- 分配表，将消息发送到多个目的地。
- 请求/回复，发送消息的应用程序需要请求消息的回复。
- 发布/预订，由代理管理消息的分配。

利用 AMI，程序员通常需要处理三个概念：

- 消息，或者说交换的是“什么”。消息包括：

- 标题信息，确定消息及其属性。
- 消息主体，包含应用程序数据。应用程序数据可以由应用程序生成，也可以由消息服务 API 生成。

在采用 MQI 的 WebSphere MQ 应用程序中，消息属性是用 MQI 显示设定的，因此应用程序设计人员必须理解其目的是什么。利用 AMI 工作时，消息属性包括在消息对象中，或由系统管理员设定的策略所定义，因此程序员就不必再关心这些细节。

- 策略，或者说将要“如何”处理消息。其包含优先级或交付确认等信息。任何数量的应用程序都可以利用相同的策略，而且任何应用程序都可以利用超过一个的策略。IBM 提供了一套通用或默认策略，此外，其也提供了开放式策略处理器框架，该框架允许第三方软件销售商创建更多策略。如果必须修改策略的话，那么通常来说，我们不必修改应用该策略的应用程序。

- 服务，或者说消息将被发送到“何处”或应从“何处”接收消息。就 WebSphere MQ 而言，这是指队列、分配表等。服务可以是指由单个应用程序提供服务的单个目的地，也可以是指目的地列表或消息代理。服务通常定义在储存库（REPOSITORY）中，其可指定到消息发送网络中真实资源的映射。储存库可以定义不同类型的服务。服务由 AMI 暗示打开和关闭。

储存库可为服务及策略提供定义。如果服务名或策略名在储存库中找不到，或者 AMI 应用程序没有储存库，那么就可采用集成在 AMI 中的定义。将储存库定义保存在 XML 格式的储存库文件中。这些定义可以通过管理工具进行更改，但只能在 Windows 平台上才可获得

该管理工具。

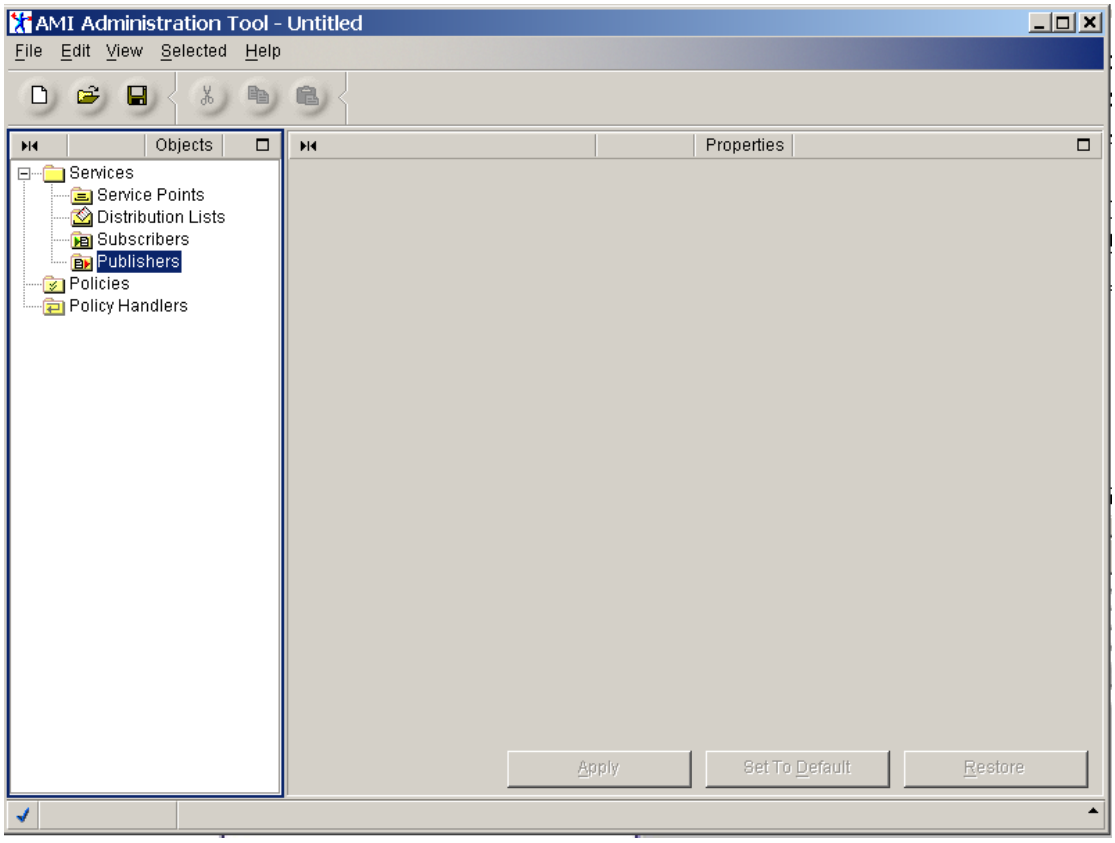
利用标准文件共享工具或者简单的文件发送，我们可以在不同的平台上共享储存库文件。重要的是要明确 AMI 应用程序不管具备或不具备储存库时都可以运行。如果没有储存库，那么就将使用默认值。

在管理工具中，被称为“服务点”（service point）的单个定义可代表储存库中发送者和接收者的定义。利用管理工具，我们可以界定义如下对象：

- 服务点
- 分配表
- 发布者
- 预订者
- 策略

不管有无相对应的储存库定义，我们都可以创建策略以及除分配表之外的服务。只有在创建服务点之后才可创建分配表。利用储存库创建服务或策略时，必须包含专用类型的定义，其命名应与应用程序所指定的名称相对应。

举例来说，创建名为“ORDERS”的发送者对象时，储存库必须有一个名为“ORDERS”的服务点定义。利用储存库创建的策略和服务，其内容由称为命名储存库定义进行初始化。无储存库的策略和服务，其内容由在默认系统定义中定义的值进行初始化。管理工具只是 WebSphere MQ AMI SupportPac （MA0F）for Windows NT 的一部分，并只能安装在运行 Windows NT 4.0 或 Windows 2000 的机器上。启动管理工具时，应选择 IBM WebSphere MQ AMI－>IBM WebSphere MQ AMI Administration Tool，或者从 Windows 资源管理器中双击文件\amt\AMITool\amitool.bat。下图显示了 AMI 管理工具。



图，*AMI* 管理工具

AMI 具有两个不同的角色：管理角色和开发角色。
在开发作用中，开发者专注于利用管理员提供的资源发送信息。这一作用不需要深入了解 WebSphere MQ。管理作用则负责创建和定义储存库中的服务和策略。换言之，管理员定义信息将被发送到“哪里”以及“如何”发送信息。这一作用要求深入了解 WebSphere MQ。利用 AMI，简化连接代码以便在发送或接收消息时确定服务或策略。利用管理工具在储存库中定义策略和服务，因此管理员可以修改它们而不会对应用程序造成任何影响。利用 AMI，我们可以在下面一种或更多种情况中交换消息：
1，与另一个使用 AMI 的应用程序交换。
2，与使用任何不同 WebSphere MQ API （MQI、WebSphere MQ Classes for Java、ActiveX 等）的应用程序交换。
3，与消息代理交换（WebSphere MQ 发布/预订或 WebSphere MQ Integrator）。

AMI 简化了发布/预订应用程序的创建。发布/预订环境中的许多配置都存在储存库中，应用程序从中获取参照。如果程序员希望寻找简单的、毋需深入了解 WebSphere MQ 的 API，那么我们就会推荐 AMI。AMI 可以通过 SupportPac （MA0F）获得，也可以从下面的 IBM 网站地址中下载：http://www.ibm.com/software/ts/WebSphere MQ/txppacs/；
提示：在尝试采用 AMI 的发布/预订功能前，必须先安装 WebSphere MQ Publish/Subscribe SupportPac （MA0C）。

14.2 平台和语言

AMI 适用于 C、C++和 Java 语言。其可应用于以下平台上：

- Windows NT 和 Windows 2000
- AIX 版本 4.3 或更高版本
- Sun Solaris 2.6 或 2.7
- HP－UX 版本 11.0
- AS/400 版本 4R4 或更高版本
- AMI 也适用于 COBOL，但仅限于 OS/390 版本 2R6 或更高，以及 CICS 版本 4.1 和 IMS 版本 5.1。

AMI 的过程应用程序编程有两个级别：
高级：程序 C 和 COBOL（在 OS/390 接口上）。由于操作是隐式的，因此 AMI 函数数量减少。
对象级：Java 和 C++类接口/对象风格 C 接口。

提示：C 高级接口包括适应大多数应用程序要求的功能。但是，如果我们需要更多的功能的话，那么可以采用 C 对象接口和高级接口的组合。

AMI 除了可提供简单的接口来处理队列管理器对象外，其对每种编程语言都有一个自然的风格。下例显示了采用 C API 的发送-遗忘应用程序。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <amtc.h>
#include <time.h>
int main（void）
{
/* 创建会话*/
hSession ＝ amSesCreate（ SAMPLE_SESSION_NAME, &compCode, &reason）；
hPol ＝ amSesCreatePolicy（ hSession, SAMPLE_POLICY_NAME, &compCode, &reason）；
hSender ＝ amSesCreateSender（ hSession, SAMPLE_SENDER_NAME,
                                &compCode, &reason）；
success ＝ amSesOpen（ hSession, hPol, &compCode, &reason）；
success ＝ amSndOpen（ hSender, hPol, &compCode, &reason）；
success ＝ amSndSend（ hSender, hPol, AMH_NULL_HANDLE, AMH_NULL_HANDLE,
                        strlen（sampleMsg）, （unsigned char *）sampleMsg,
                        AMH_NULL_HANDLE, &compCode,&reason ）；
success ＝ amSesDelete（ &hSession, &compCode, &reason ）；
endSample（EXIT_SUCCESS）；
}
```

下例中出现的应用程序与上例中的发送-遗忘应用程序相同，不过这回它是用 Java API 编写的。请注意，这两种编程语言的每一种都具有自然的风格。

例，用 *Java* 编写的简单发送- 遗忘应用程序

```
import java.util.*；
import com.ibm.mq.amt.*；
. . .
public void main（）
{
mySessionFactory ＝ new AmSessionFactory（）；
mySession ＝ mySessionFactory.createSession（SAMPLE_SESSION_NAME）；
myPolicy ＝ mySession.createPolicy（SAMPLE_POLICY_NAME）；

mySender ＝ mySession.createSender（SAMPLE_SENDER_NAME）；
mySendMSG ＝ mySession.createMessage（SAMPLE_MESSAGE_NAME）；
mySession.open（myPolicy）；
mySender.open（myPolicy）；
String sampleMessage ＝ new String（"Sample message"）；
mySendMSG.writeBytes（sampleMessage.getBytes（））；
mySender.send（mySendMSG）；
mySender.close（myPolicy）；
mySession.close（myPolicy）；
}
```

14.3 库和包

现在，我们已经了解了可以用来编写 AMI 应用程序的不同编程语言，接下来，我们就必须

了解用于编写 AMI 应用程序的不同的库。如果利用 C 语言来编写应用程序的话，那么 AMI 会提供头文件，称为 amtc.h。该文件包括 AMI 所用的所有函数、结构和常量。该头文件必须包含在应用程序中，我们可以用下面的语句来做到这一点：
#include <amtc.h>

下图显示了在支持 AMI 的不同平台上 amtc.h 文件所处的位置。
表， *AMI C* 头文件的位置

操作系统平台	位置
AS400	QMQAMI/H
UNIX（包括 AIX, HP－UX 和 Solaris）	{WebSphere MQ 目录}/amt/inc
Windows	{WebSphere MQ 目录}\amt\include
OS/390	hlq.SCSQC370

提示：在编译时间内，程序必须能够访问 amtc.h 文件。

就 C++而言，AMI 还可提供另一个头文件，称为 amtcpp.hpp。该头文件包括 C++的函数、结构和常量，与 C 语言的头文件完全一样。同样，amtcpp.hpp 也必须包含在应用程序中。我们同样也可以用下面的语句来实现这一目的：
#include<amtcpp.hpp>

下表显示了 amtcpp.hpp 文件在支持 AMI 的不同平台上的位置。
表，*AMI C++* 头文件的位置

操作系统平台	位置
AS/400（包括 AIX、HP－UX 和 Solaris）	QMQAMI/H
UNIX	{WebSphere MQ 基目录}/amt/inc
Windows	{WebSphere MQ 基目录}\amt\include

提示：在编译时间内，即便 amtcpp.hpp 是唯一在使用中的头文件，程序也必须可以访问 amtc.h 和 amtcpp.hpp 这两个文件。

如果开发人员希望使用 Java API 的话，那么 AMI 可提供包括构成 AMI package for Java 所有类的 JAR 文件。
Java 包：com.ibm.mq.amt
Java JAR 文件：com.ibm.mq.amt.jar

为了使用 AMI package for Java，我们必须利用导入语句将其导入 Java 应用程序：
import com.ibm.mq.amt.*；

提示：JAR 文件必须是 CLASSPATH 环境变量的一部分（这必须在编译和运行 Java 应用程序的环境中完成）。

下表显示了 AMI JAR 文件在支持 AMI 不同平台上所处的位置。
表， *AMI Java JAR* 文件的位置

操作系统平台	位置
AS/400	/QIBM/ProdData/mqm/amt/Java/lib
UNIX（包括 AIX、HP－UX 和 Solaris）	{WebSphere MQ 基目录}/java/lib
Windows	{WebSphere MQ 基目录}\java\lib

下表显示了由 AMI 支持的 C、COBOL、C++，以及 Java 语言编译程序：

表，语言编译器

操作系统平台	所支持的编译器
AIX	用于 C++ 5.0 版本的 VisualAge JDK 1.1.7 和更高
OS/400	用于 Java（5769JV1）的 AS/400 Developer Kit 用于 AS/400（5769CX2）的 ILE C 用于 AS/400（5799GDW）的 ILE C++ 用于 C++ for 0S/400（5716CX4）的 Visual Age
HP－UX	HP aC++ B3910B A.03.10 HP aC++ B3910B A.03.04（970930）支持库 JDK 1.1.7 和更高
OS/390	OS/390 C/C++ 版本 2（第 6 次发行）或更高 用于 OS/390 & VM 版本 2（第 1 次发行）或更高的 IBM COBOL 用于 MVS 和 VM 版本 1（第 2 次发行）或更高的 IBM COBOL
Sun Solaris	Workshop Compiler 4.2（带 Solaris 2.6） Workshop Compiler 5.0（带 Solaris 7） JDK 1.1.7 或更高
Windows	Microsoft Visual C++ Version 6 JDK 1.1.7 或更高

如欲了解如何在所有支持的语言中准备并运行 AMI 应用程序的更多信息，请参见 WebSphere MQ 应用程序消息发送接口文档。

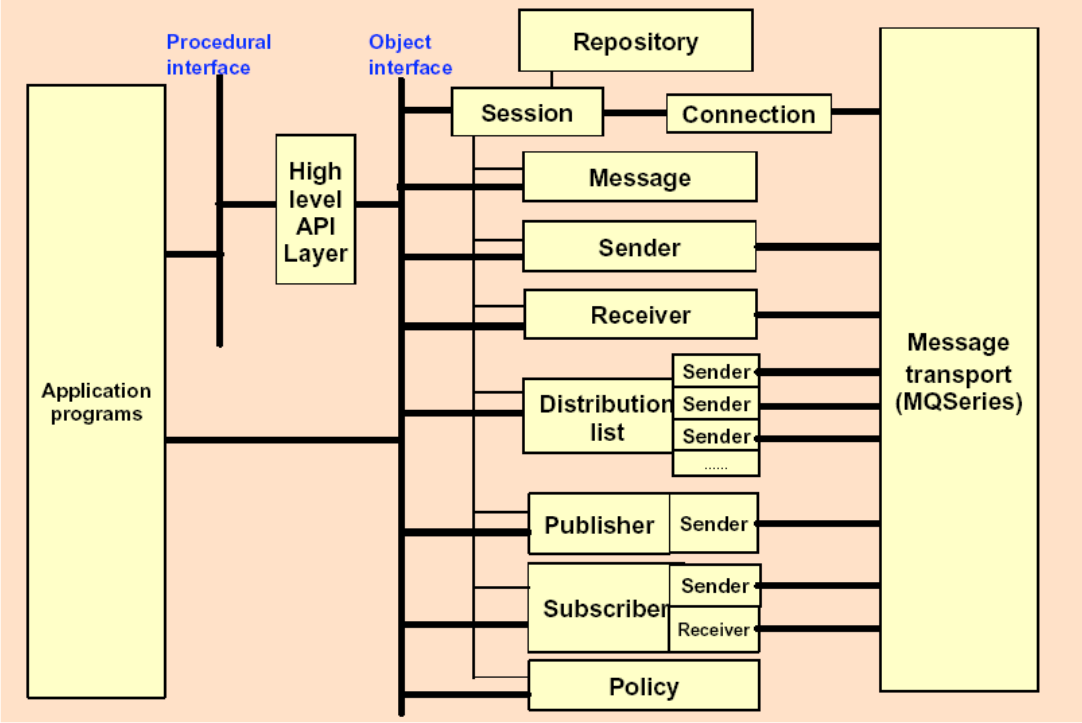
14.4 体系结构模型

AMI 可提供以下对象：

- 会话：用以创建新的 AMI 会话，并用以控制事务处理支持。这是在创建和管理所有其他对象（消息、策略、接收者等）前第一个需要初始化的对象。
- 消息：该对象包含消息数据以及发送或接受消息时用到的消息描述符结构。
- 发送者：代表消息发送目的地的一项服务。MQOD 结构包含在该对象中。
- 接收者：代表消息接收来源的一项服务。MQOD 结构也包含在该类中。
- 分配表：包含发送者服务列表，提供目的地列表。
- 发布者：包含发送者服务，其中目的地是一个发布/预订代理。
- 预订者：包含发送者服务，将预订和取消预订消息发送至发布/预订代理，也包括一个接收者服务，接收来自代理的发布内容。

- 策略：定义应当如何处理消息，包括优先级、持久性，以及其是否应包括在工作单元中等项目。

发送者、接收者、分配表、发布者和预订者都是服务。直接连接到消息传输层的唯一对象是发送者和接收者。因此，分配表和发布者对象包含发送者，而预订者对象包含发送者和接收者。下图显示了 AMI 体系结构模型，并反映了每个对象如何与其他各个对象进行互动。



图， *AMI* 体系结构模型

消息、服务和策略对象是由会话对象创建和管理的。会话对象也提供工作单元的范围。连接、发送者和接收者对象的组合可提供消息的传输。

程序员可以利用系统默认提供的消息属性、服务和策略对象，也可以利用管理员定义的、储存在储存库中的属性。除了以上提到的对象之外，AMI 还为 C++和 Java 提供了更多的对象定义。

这些对像包括：

会话库：该对象用以创建会话对象。没有会话库，就不能创建会话对象。

协助对象和异常对象（Helper and Exception objects）。

14.5 用 AMI 编程

本节所用的实例是采用 Java 接口编写的。如欲了解如何采用其他支持 AMI 的编程语言发出调用，请参考应用程序消息发送接口文档。

14.5.1 连接到队列管理器

正如在其他 WebSphere MQ API 中一样，在尝试访问任何队列服务器对象之前，我们都必须

连接到队列服务器。我们可以利用两个 AMI Java 调用实现这一目的。首先，需要创建新的会话库对象。在此应记住，为了创建会话对象，必须至少有一个会话库对象（这只适用于C++和 Java）。用来创建会话库的命令是：

SessionFactoryObject ＝ new AmSessionFactory（String factoryName）

FactoryName 是一个可选参数。该字符串实际上就是储存库和主文件所在的目录，其也可以是完全指定的目录，包括文件所在的路径，例如：C:\Program Files\WebSphere MQ\amt。如果该参数没有指定，那么我们将使用AMT_DATA_PATH 环境变量所指定的值。该环境变量通常是在WebSphere MQ AMI SupportPac（MA0F）安装时设定的。创建会话库对象之后，我们可以定义新的会话对象。通过利用AmSessionFactory 类的createSession（）方法来进行这一工作。

mySessionFactory.createSession（String sessionName）

sessionName 参数是我们可以用来确定会话对象的名。一旦会话库初始化且创建了会话对象，应用程序就可以处理WebSphere MQ 对象了。在AMI 中，队列管理器名是从位于{WebSphere MQ 基目录}/amt 的主文件（默认情况下是amthost.xml）中读取的。下面显示了用来连接到称为ITSOH 队列管理器的主文件样例。如果defaultConnection 标签没有指定队列管理器的话，那么将采用默认的队列管理器。

例， 主文件样例

```
<?xml version＝”1.0” encoding＝”UTF－8” ?>
<queueManagersNames defaultConnection＝”ITSOH”
connectionName1＝”queueManagerName1” connectionName2＝”queueManagerName2” />
```

下例显示了如何创建AmSessionFactory 对象，以及名为ITSO 的AmSession 对象。
例，创建新对象

```
private AmSessionFactory mySessionFactory ＝ null;
private AmSession mySession ＝ null;
mySessionFactory ＝ new AmSessionFactory（）;
mySession ＝ mySessionFactory.createSession（“ITSO”）;
```

一旦创建了会话，我们在创建其他AMI对象时所采用的顺序就并不重要了。但是，我们建议您以如下顺序对对象进行初始化：

1. 会话
2. 策略
3. 发送者/接收者/发布者/预订者/分配表
4. 消息

14.5.2 打开 WebSphere MQ 对象

创建会话库和会话对象之后，下一步应创建/初始化其他AMI 对象。

14.5.2.1 创建策略

首先，我们需要创建策略对象。我们可以利用AmSession 类的createPolicy（）方法来达到这一目的。

sessionObject.createPolicy（String policyName）

如果我们所指定的policyName 与储存库中已经存在的策略名相匹配，那么将会采用储存库定义来创建策略。如果储存库中不存在这种策略名，则将采用默认值创建策略。在下例中，我们可以看到如何创建AMT.SAMPLE.POLICY。由于储存库已经定义了AMT.SAMPLE.POLICY，因此该策略将用储存库已定义的值进行创建。
例，创建策略对象

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）;
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）;
    myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）;
}
```

创建策略之后，下一步就是根据应用程序要采用的设计模式来创建对象模式。对象类型可以是：

发送者和/或接收者（也称服务点）
发布者
预订者
分配表

14.5.2.2 创建发送者

为了创建发送者对象，我们要用到会话对象的createSender（）方法。

sessionObject.createSender（String senderName）;

如果指定的senderName 与储存器中已存在的相匹配，那么将采用储存器中的定义来创建发

送者对象。如果不存在的话，那么将采用默认值来创建发送者。下例显示了如何创建一个称为AMT.SENDER.NAME 的发送者。

例，创建发送者

```
AmSessionFactory mySessionFactory ＝ null;
AmSession mySession ＝ null;
AmPolicy myPolicy ＝ null;

mySessionFactory ＝ new AmSessionFactory（）；
mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
mySender ＝ mySession.createSender（“AMT.SENDER.NAME”）；
```

14.5.2.3 创建接收者

为了创建接收者对象，我们要用到会话对象的createReceiver（）方法。

sessionObject.createReceiver（String receiverName）；

如果我们指定的receiverName 与储存器中已经存在的相匹配，那么将采用储存器中的定义来创建接收者对象。如果不存在的话，将采用默认值来创建接收者。下例显示了如何创建一个称为MT.RECEIVER.NAME 的接收者。

例， 创建接收者

```
AmSessionFactory mySessionFactory ＝ null;
AmSession mySession ＝ null;
AmPolicy myPolicy ＝ null;
AmReceiver myReceiver ＝ null;
mySessionFactory ＝ new AmSessionFactory（）；
mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
myReceiver ＝ mySession.createReceiver（“AMT.RECEIVER.NAME”）；
```

14.5.2.4 创建发布者

为了创建发布者对象，我们要用到会话对象的createPublisher（）方法。

sessionObject.createPublisher（String publisherName）；

如果我们指定的publisherName 与储存器中已经存在的相匹配，那么将采用储存器中的定义来创建发布者对象。如果不存在的话，那么将采用默认值来创建发送者。发布者和预订者所用的发送者和接收者点的服务类型必须在储存库中定义为MQRFH。这就会产生MQRFH 标题，包含发送/预订名/值等因素，这些将在消息发送时被加到消息中。如欲了解更多信息，请参见应用程序消息发送接口文档。下例显示了如何创建称为AMT.PUBLISHER.NAME 的发布者。

例，创建发布者

```
AmSessionFactory mySessionFactory = null;
AmSession mySession = null;
AmPolicy myPolicy = null;
AmPublisher myPublisher = null;
mySessionFactory = new AmSessionFactory（）;
mySession = mySessionFactory.createSession（“ITSO.SESSION.NAME”）;
myPolicy = mySession.createPolicy（“AMT.SAMPLE.POLICY”）;
myPublisher = mySession.createPublisher（“AMT.PUBLISHER.NAME”）;
```

14.5.2.5 创建预订者

为了创建预订者，我们要用到会话对象的createSubscriber（）方法。

sessionObject.createSubscriber（String subscriberName）;

如果我们指定的subscriberName 与储存器中已经存在的名称相匹配，那么将采用储存器中的定义来创建预订者对象。如果不存在，将采用默认值进行创建。下例显示了如何创建一个称为AMT.SAMPLE.SUBSCRIBER 的预订者。

例， 创建预订者

```
public static void main（）
{...
    AmSessionFactory mySessionFactory = null;
    AmSession mySession = null;
    AmPolicy myPolicy = null;
    mySessionFactory = new AmSessionFactory（）;
    mySession = mySessionFactory.createSession（“ITSO.SESSION.NAME”）;
    myPolicy = mySession.createPolicy（“AMT.SAMPLE.POLICY”）;
    mySubscriber = mySession.createSubscriber（“AMT.SAMPLE.SUBSCRIBER”）;
}
```

14.5.2.6 创建分配表

为了创建分配表对象，我们要用到会话对象的createDistributionList（）方法。

sessionObject.createDistributionList（String distributionlistName）；

如果我们指定的分配表名与储存器中已经存在的相同，那么将采用储存器中的定义创建对象。如果不存在，将采用默认值创建分配表。在我们能使用分配表之前，管理员必须首先定义发送者服务，再将这些服务定义为分配表的一部分。下例显示了应当如何创建分配表。
例，创建分配表

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）；
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
    myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
    mySender ＝ mySession.createDistributionList（“AMT.DISTRIBUTION.LIST”）；
}
```

14.5.2.7 创建消息

为了创建消息对象，我们要用到以下AmSession 类的createMessage（）方法。

AmSession.createMessage（String messageName）

对应用程序而言，MessageName 是具有意义的名称。下例显示了一段代码样例，告诉我们如何创建称为ITSO.SAMPLE.MESSAGE.NAME 的消息。
例， 创建消息对象

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    AmPolicy myPolicy ＝ null;
    AmMessage mySendMSG ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）；
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
    myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
    mySender ＝ mySession.createSender（“AMT.SENDER_NAME”）；
```

```
    mySendMSG ＝ mySession.createMessage（“ITSO.SAMPLE.MESSAGE.NAME”）；
}
```

现在我们已经创建了对象，那么下一步就要打开这些已被定义的对象。利用open（）方法来进行这一工作。举例来说，如果需要打开发送者对象，那么就要用到该发送者对象的open（）方法。所有的打开调用都分享一个共同的参数，这就是策略对象，open()方法的语法如下：

```
AmSession.open（AmPolicy policyObject）；
AmSender.open（AmPolicy policyObject）；
AmReceiver.open （AmPolicy policyObject）；
AmPublisher.open（AmPolicy policyObject）；
AmSubscriber.open（AmPolicy policyObject）；
```

下例显示了如何打开会话、发布者和接收者对象。
例，打开不同类型的对象

```
myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
mySession.open（myPolicy）；
myPublisher.open（myPolicy）；
myRespReceiver.open（myPolicy）；
```

14.5.3 基本操作

下面讨论可对这些WebSphere MQ 对象进行哪些基本操作。这些基本操作包括获取消息、发送消息、发布和预订消息。

可以在MQI 中发送或接收消息之前，我们必须先打开队列，以获得输入或输出。在MQI中，这被称为打开选项（MQOO_INPUT_SHARED, MQOO_OUTPUT）。在AMI 中，由管理员给出这些定义，并将其保存在储存库中，因此在队列管理器对象打开或作为一部分调用的时候，程序员不必对其做明确的定义。

发送消息

为了发送消息，我们要用到发送者类的send（）方法。请记住，在尝试发送消息之前，必须创建至少四个对象（会话、策略、消息和发送者）。如果我们需要获得回复的话，那么还必须创建接收者对象和另外的消息对象。

消息内容总是以字节形式发送的（Java 的本身形式），因此我们建议您在发送消息前用getBytes（）方法（Java 的本身方法）将消息转化成字节数组。一旦转化消息，接下来就应当利用发送者类的writeBytes（）方法将其写入消息对象中。

```
senderObject.send（AmMessage messageObject, AmReceiver receiverObject/AmMessage
                  receivedMessage, AmPolicy policyObject）
```

另外，我们可以指定策略对象和/或接收者或另一消息对象。请记住，策略是指“如何”处理消息。换言之，我们利用策略可以指定优先级，或者说如果送达消息时出现错误的话，那么应当采取哪些行动，等等。

如果我们要求某种类型的回复，比如送达报告确认或远程应用程序的确认，那么我们就必须根据希望如何处理回复而指定接收者对象或消息对象。下例显示了发送消息样例方法。

例，发送消息样例

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    AmSender mySender ＝ null;
    AmMessage mySendMSG ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）；
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
    myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
    mySender ＝ mySession.createSender（“AMT.SENDER_NAME”）；
    mySendMSG ＝ mySession.createMessage（“ITSO.SAMPLE.MESSAGE.NAME”）；
    String sampleMessage ＝ new String（"Sample message"）；
    mySendMSG.writeBytes（sampleMessage.getBytes（））；
    mySender.send（mySendMSG）；
}
```

我们也可以同时将消息发送到多个目的地。我们可以利用distributionList 对象的send（）方法做到这一点。下例显示了将消息样例发送到多个目的地的实例的方法。

例， 将一条消息发送到多个目的地

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    AmDistributionList myDistributionLst ＝ null;
    AmMessage mySendMSG ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）；
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
    myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
    myDistributionLst ＝ mySession.createDistributionList（“ITSO.DISTRIBUTION.LIST”）；
    mySendMSG ＝ mySession.createMessage（“ITSO.SAMPLE.MESSAGE.NAME”）；
    String sampleMessage ＝ new String（"Sample message"）；
    mySendMSG.writeBytes（sampleMessage.getBytes（））；
    myDistributionList.send（mySendMSG）；
}
```

获取消息

为了获取消息，我们要用到接收者类的receive（）方法。

amReceiver.receive（AmMessage messageObject, AmSender senderObject, AmMessage selectionmessageObject, AmPolicy policyObject）；

要该调用中有4 个参数，但其中只有messageObject 是始终需要的。接收的消息将被储存在此参数中。如果发送应用程序请求回复的话，那么就必须指定senderObject。PolicyObject 用来指定等待时间间隔，或是否要求数据转换等。如果我们需要根据关联性ID（correlation ID）获得某个特定消息的话，那么就必须指定selectionmessageObject。下例显示了如何接收消息。例，接收消息

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    AmReceiver myReceiver ＝ null;
    AmMessage myReceiveMSG ＝ null;
    AmMessage mySendMSG ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）；
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
    myPolicy ＝ mySession.createPolicy（“ITSO.SAMPLE.POLICY”）；
    myReceiveMSG ＝ mySession.createMessage（“ITSO.SAMPLE.MESSAGE.NAME”）；
    myReceiver.receive（MyReceiveMSG）；
    String sampleMessage ＝ new
    String（MyReceiveMSG.readbytes（myReceiveMSG.getDataLength（））；
}
```

发布消息

发布消息时，应当遵从以下几个步骤。一旦打开消息和发布者对象，就需要为消息创建新主题。该主题就是描述将要发布数据的词。我们用消息类的addTopic（）方法创建主题。

messageObject.addTopic（String topicName）

唯一所需的参数就是topicName，正如我们前面谈到的那样，其为描述将要发布数据的词。

添加主题之后，我们就可以发布消息了。为了发布消息，要用到发布者类的publish（）方法。

publisherObject.publish（AmMessage messageObject, AmReceiver

receiverObject, AmPolicy policyObject）

messageObject 是包含将发布消息的对象。只有在策略不指定发布者的隐式注册（implicit registration）的情况下，才可以省略receiverObject。总是应当指定MessageObject。如果没有指定policyObject，将采用默认定义。

下例显示了如何发布消息的样例代码。在该例中，没有指定policyObject；因此我们将采用默认定义。默认定义不要求发布者的隐式注册，因此不要求receiverObject。

例， 发布一条消息

```
string pubMessage＝”SUNNY”；
mySendMSG.addTopic（“Weather”）；
mySendMSG.writeBytes（pubMessage.getBytes（））；
myPublisher.publish（mySendMSG）；
```

预订消息

为了预订消息，应用程序应当就感兴趣的信息发送请求。我们通过在请求中包含主题来实现这一目的。一旦请求完成，预订者就可以接收来自许多不同发布者的信息，而且也可以将接收到信息发送给其他的预订者。为了在请求中包括主题，我们可以使用消息类的addTopic（）方法。

messageObject.addTopic（String topicName）

topicName 是确认数据的一个名，该数据对预订者应用程序有用。对应用程序希望预订的每个主题，都必须使用该调用。一旦应用程序定义了所有主题，就用预订者类的subscribe（）方法发送预订请求。

subscriberObject.subscribe（AmMessage messageObject, AmReceiver receiverObject, AmPolicy policyObject）

messageObject 是我们利用addTopic（）方法向其添加主题的对象。该参数是可选的。将匹配预订请求的发布内容发送到receiverObject。但是，该参数并不总是需要的。另一种接收匹配请求的发布信息的方法就是发出预订类的receive（）方法。

可以确定，也可以不指确定PolicyObject。如确定，那么策略就包含匿名注册、仅接收新发布内容等选项。如果未确定的话，那么将采用默认策略。如果receiverObject 在预订请求中未确定的话，那么可以利用receive（）方法接收发布内容。

subscriberObject.receive（AmMessage messageObject, AmMessage selectionmessageObject, AmPolicy policyObject

messageObject 将包含所有已发布的消息，并在消息接收前进行隐式重置。只有在我们希望根据关联性ID 选择某个特定消息时才采用selectionMessageObject。如果未确定的话，那么就接收第一个可用的信息。可以确定PolicyObject，也可以不确定。

一旦我们接收了符合预订请求所确定标准的所有消息，或者说我们不需要更多的消息了，那么应用程序就可以取消预订，换言之，应用程序此时可以发出请求，这样就不会再发送更多的消息了。
应用程序可以发送一个请求，取消预订其最初请求的所有主题，也可仅取消预订某个特定主题。通过发出预订类的unsubscribe（）方法，我们可实现这一目的。

```
subscribeObject.unsubscribe（AmMessage messageObject, AmReceiver
                              receiverObject, AmPolicy policyObject）
```

messageObject 包含取消预订请求所应用的主题。如需要获得取消预订请求的确认，那么必须发出receiverObject。正如前面的调用一样，可以确定策略，也可以不确定。下例显示了如何应用我们在本节所讨论的这些调用。
例，预订消息

```
int iCounter ＝ 0；
String topic ＝ ”Weather”；
mySendMSG.addTopic（“Weather”）；
mySubscriber.subscribe（mySendMSG, myPolicy）；
// Only 5 messages are expected
for （iCounter ＝ 0； iCounter < 5； iCounter++）
    mySubscriber.receive（myReceiveMSG, myPolicy）；
String myRequest ＝ new
String（myReceiveMSG.readBytes（myReceiveMSG.getDataLength（））；
System.out.println（myRequest）；
// The application has received all the messages that it wanted so it proceeds
// to send an unsubscribe request.
mySubscriber.unsubscribe（mySendMSG, myPolicy）；
```

14.5.4 删除会话并关闭连接

我们对队列管理器应用程序的工作完成之后，接下来就可关闭对象了。我们利用每个被用对象的close（）方法来实现这一目的。另一种方法就是仅关闭会话对象。一旦关闭该对象，其他对象也将关闭。但是，我们强烈建议您隐式关闭所有已打开的对象。

```
subscriberObject.close（AmPolicy policy Object）；
sessionObject.close（AmPolicy policy Object）；
receiverObject.close（AmPolicy policy Object）；
publisherObject.close（AmPolicy policy Object）；
distributionlistObject.close（AmPolicy policy Object）；
```

所有这些close（）方法需要的唯一一个参数就是policyObject。请记住，policyObject也包含关闭选项，如关闭时删除动态队列等。如欲了解更多信息，请参见WebSphere MQ 应用程序消息发送接口文档。
提示：最后一个必须关闭的对象是sessionObject。一旦关闭sessionObject，其他参照都会无

效。

14.6 AMI 和 MQI 的比较

我们已经讨论了 WebSphere MQ 所提供的不同 API，现在，我们可以谈谈它们彼此之间的比较。在这一节中，我们将对 AMI 和 MQI 进行比较。利用 MQI，消息目的地和消息的发送/接收选项均由应用程序管理，而利用 AMI，则由策略管理。MQI 提供全部 WebSphere MQ 函数支持，只考虑消息传输。AMI 提供较少的 WebSphere MQ 函数，但提供更多的功能性。MQI 的编程接口属低级的（动词较少，但有很多结构），对不同语言（C、C++、Java 和 COBOL）来说，其 API 都是相似的。AMI 具有高级编程接口，这意味着其动词更多，而结构更少，而且其 API 对每种不同的编程语言都有各自自然的风格。MQI 根据传输而不同，AMI 则独立于传输。MQI 是 IBM 的所有，而 AMI 的子集符合开放应用程序组/开放应用程序中间件 API 标准（C++和 Java）。

14.7 事务处理管理

为了使 AMI 发送和接收的消息成为事务处理工作单元的一部分，管理员必须利用 AMI 管理工具指定策略的同步点属性。在默认情况下，将同步点属性设为关闭。我们可以在策略定义的 General 标签中找到该属性。控制事务处理的 API 调用取决于正被使用的事务处理类型。此处有两种不同的情况：

当唯一的资源是 WebSphere MQ 消息时：

在这种情况中，根据策略中就发送或接收对象所确定的那样，事务处理由同步点控制下第一个发送或接收的消息来启动。多个消息可以包括在同一工作单元中。可以用会话对象的 commit（）方法提交事务处理，或者也可以用会话对象的 rollback（）
方法取消事务处理。

下例显示了 WebSphere MQ 消息作为唯一资源的同步点控制实例。

例，同步点控制

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    AmReceiver myReceiver ＝ null;
    AmMessage myReceiveMSG ＝ null;
    AmMessage mySendMSG ＝ null;
    mySessionFactory ＝ new AmSessionFactory（）;
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）;
    myPolicy ＝ mySession.createPolicy（“ITSO.SAMPLE.POLICY”）;
    myReceiveMSG ＝ mySession.createMessage（“ITSO.SAMPLE.MESSAGE.NAME”）;
    myReceiver.receive（MyReceiveMSG）;
    // If no failures were found commit the action
```

```
    mySession.commit（myPolicy）；
    String sampleMessage ＝ new
    String（MyReceiveMSG.readbytes（myReceiveMSG.getDataLength（））；
    // If some problems were found, don’t retrieve the message
    mySession.rollback（myPolicy）；
}
```

WebSphere MQ 作为 XA 事务处理协调器（XA transaction coordinator）在可恢复资源（如关系数据库）发生改变之前，须用 AmSession 类的 begin（）方法显式启动事务处理。而后，可以用 AmSession 类的 commit（）方法提交工作单元，或者用 AmSession 类的 rollback（）方法予以取消。下例显示了 WebSphere MQ 作为 XA 事务处理协调器的实例。

例，*WebSphere MQ* 作为 *XA* 事务处理协调器

```
public static void main（）
{
    AmSessionFactory mySessionFactory ＝ null;
    AmSession mySession ＝ null;
    AmPolicy myPolicy ＝ null;
    AmSender mySender ＝ null;
    AmMessage mySendMSG ＝ null;

    mySessionFactory ＝ new AmSessionFactory（）；
    mySession ＝ mySessionFactory.createSession（“ITSO.SESSION.NAME”）；
    myPolicy ＝ mySession.createPolicy（“AMT.SAMPLE.POLICY”）；
    mySender ＝ mySession.createSender（“AMT.SENDER_NAME”）；

    mySendMSG ＝ mySession.createMessage（“ITSO.SAMPLE.MESSAGE.NAME”）；
    mySession.Begin（myPolicy）；

    // Update a table.
    // If the update was successful then commit the action and send a message to
    // another application
    String sampleMessage ＝ new String（"Sample message"）；
    mySendMSG.writeBytes（sampleMessage.getBytes（））；
    mySender.send（mySendMSG）；
    mySession.commit（myPolicy）；
    // If problems occurred during the update, backout the changes
    mySession.rollback（myPolicy）；
}
```

提示：如果使用外部事务处理协调器（如 Tuxedo），就可能出现另一种情况。在这种情况下，我们利用外部事务处理协调器的 API 调用来控制事务处理。尽管没有使用 AMI 调用，但同步点属性仍必须在调用使用的策略中确定。

14.8 分组

AMI 允许将一系列相关消息包含在消息组中或作为消息组发送。组上下文信息（group context information）随每个消息发送，以使消息系列得以保存，并可由接收程序获得。为了将消息包含在消息组中，组中第一个和后续消息的组状态信息（group status information）必须进行如下设定：

第一条消息设为 AMGRP_FIRST_MSG_IN_GROUP

除第一和最后一条消息外，所有消息均设为 AMGRP_MIDDLEMSG_IN_GROUP

最后一条消息设为 AMGRP_LAST_MSG_IN_GRP

我们可以利用消息类的 setGroupStatus（）实现上述目的。

messageObject.setGroupStatus（int groupStatus）

groupStatus 可以为下面任何值：

AMGRP_MSG_NOT_IN_GROUP

AMGRP_FIRST_MSG_IN_GRP

AMGRP_MIDDLE_MSG_IN_GRP

AMGRP_LAST_MSG_IN_GROUP

AMGRP_ONLY_MSG_IN_GROUP

如果 AMGRP_FIRST_MSG_IN_GROUP 在序列外， 那么该消息的行为与 AMGRP_MIDDLE_MSG_IN_GRP 相同。

提示：一旦应用程序开始发送在组中的消息，那么在尝试发送不在该组内的任何其他消息之前，必须先发送完该组的消息。

14.9 本章小结

本章概括介绍了应用程序消息接口，讲解什么是 AMI 以及如何应用 AMI。

14.10 本章练习

1.使用 WebSphere MQ AMI 编写文件的发送程序。

2.使用 WebSphere MQ AMI 编写文件的接收程序。

附录一 WebSphere MQ 的缺省系统对象

保留的队列名称

以“SYSTEM.”开始的名称保留为队列管理器定义的队列名。您可使用 ALTER 或 DEFINE REPLACE 命令来更改这些队列定义以适合您的安装。为 WebSphere MQ 定义以下名称：

SYSTEM.ADMIN.CHANNEL.EVENT	通道事件队列
SYSTEM.ADMIN.COMMAND.QUEUE	PCF 命令消息发送到的队列（不用于 z/OS）
SYSTEM.ADMIN.CONFIG.EVENT	配置事件队列
SYSTEM.ADMIN.PERFM.EVENT	性能事件队列
SYSTEM.ADMIN.QMGR.EVENT	队列管理器事件队列
SYSTEM.CHANNEL.COMMAND	在使用 CICS 的 z/OS 上用于分布式排队的队列
SYSTEM.CHANNEL.INITQ	在没有 CICS 的 z/OS 上用于分布式排队的队列
SYSTEM.CHANNEL.SEQNO	在使用 CICS 的 z/OS 上用于分布式排队的队列
SYSTEM.CHANNEL.SYNCQ	在没有 CICS 的 z/OS 上用于分布式排队的队列
SYSTEM.CICS.INITIATION.QUEUE	用于触发的队列（不用于 z/OS）
SYSTEM.CLUSTER.COMMAND.QUEUE	用于队列管理器之间传达资源库更改的队列（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris、Windows 和 z/OS）
SYSTEM.CLUSTER.REPOSITORY.QUEUE	用于保存关于资源库信息的队列（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris、Windows 和 z/OS）
SYSTEM.CLUSTER.TRANSMIT.QUEUE	用于群集支持管理的所有目标的传输队列（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris、Windows 和 z/OS）
SYSTEM.COMMAND.INPUT	在 z/OS 上命令消息发送到的队列
SYSTEM.COMMAND.REPLY.MODEL	用于命令应答的模型队列定义（用于 z/OS）
SYSTEM.DEAD.LETTER.QUEUE	死信队列（不用于 z/OS）
SYSTEM.DEFAULT.ALIAS.QUEUE	缺省别名队列定义
SYSTEM.DEFAULT.INITIATION.QUEUE	用于触发指定进程的队列（不用于 z/OS）
SYSTEM.DEFAULT.LOCAL.QUEUE	缺省本地队列定义
SYSTEM.DEFAULT.MODEL.QUEUE	缺省模型队列定义
SYSTEM.DEFAULT.REMOTE.QUEUE	缺省远程队列定义
SYSTEM.MQSC.REPLY.QUEUE	用于 MQSC 命令应答的模型队列定义（不用于 z/OS）
SYSTEM.QSG.CHANNEL.SYNCQ	用于存储包含共享通道同步信息消息的共享本地队列（仅 z/OS）
SYSTEM.QSG.TRANSMIT.QUEUE	在相同队列共享组中发送队列管理器之间消息时，组内排队代理程序使用的共享本地队列

	（仅 z/OS）

其它对象名

进程、名称列表、群集和认证信息对象对象名称可以长达 48 个字符。通道名称可长达 20 个字符。存储类名称可长达 8 个字符。

保留的对象名

保存以“SYSTEM.”开始的名称为队列管理器定义的对象。您可使用 ALTER 或 DEFINE REPLACE 命令奈更改这些对象定义以适合你的安装。为 WebSphere MQ 定义以下名称：

SYSTEM.ADMIN.SVRCONN	WebSphere MQ Explorer 使用的用于队列管理器远程管理的服务器连接通道（在 z/OS 上远程管理不可用）
SYSTEM.AUTO.RECEIVER	自动定义的缺省接收方通道（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris 和 Windows）
SYSTEM.AUTO.SVRCONN	自动定义的缺省服务器连接通道（AIX、HP-UX、Linux、OS/2 Warp、z/OS、OS/400、Solaris 和 Windows）
SYSTEM.DEF.CLNTCONN	缺省客户机连接通道定义
SYSTEM.DEF.CLUSRCVR	缺省群集接收方通道定义（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris、Windows 和 z/OS）
SYSTEM.DEF.CLUSSDR	缺省群集发送方通道定义（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris、Windows 和 z/OS）
SYSTEM.DEF.RECEIVER	缺省接收方通道定义
SYSTEM.DEF.REQUESTER	缺省请求者通道定义
SYSTEM.DEF.SENDER	缺省发送方通道定义
SYSTEM.DEF.SERVER	缺省服务器通道定义
SYSTEM.DEF.SVRCONN	缺省服务器连接通道定义
SYSTEM.DEFAULT.AUTHINFO.CRLLDAP	缺省认证信息定义
SYSTEM.DEFAULT.NAMELIST	缺省名称列表定义（仅 AIX、HP-UX、Linux、OS/2 Warp、OS/400、Solaris、Windows 和 z/OS）
SYSTEM.DEFAULT.PROCESS	缺省进程定义
SYSTEMST	缺省存储类定义（仅 z/OS）