

精通 WebSphere MQ

——Mastering WebSphere MQ

陈宇翔 著

安徽科学技术出版社

目 录

前言	1
从系统集成到系统整合	1
消息驱动和消息触发	1
记号约定	2
第 1 章 概念与原理	1
1.1 简介	1
1.1.1 消息中间件	1
1.1.2 WebSphere MQ	1
1.1.3 WebSphere MQ 产品	2
1.2 概念与对象	3
1.2.1 消息 (Message)	3
1.2.2 队列 (Queue)	3
1.2.3 队列管理器 (Queue Manager)	6
1.2.4 通道 (Channel)	6
1.2.5 名称列表 (Name List)	8
1.2.6 分发列表 (Distribution List)	9
1.2.7 进程定义 (Process)	9
1.2.8 认证信息 (Auth Info)	9
1.2.9 客户端和服务端 (Client & Server)	9
1.2.10 操作界面 (MQ Interface)	9
1.2.11 应用程序 (MQ Application)	9
1.3 工作原理	10
1.3.1 PUT 和 GET	10
1.3.2 协同工作	10
1.3.3 互连通信	11
第 2 章 安装	14
2.1 安装环境	14
2.1.1 硬件	14
2.1.2 操作系统	14
2.1.3 通信协议	15
2.2 安装介质	15
2.2.1 正版	15
2.2.2 试用版	15
2.3 安装过程	15
2.4 缺省配置	17
2.4.1 准备 WebSphere MQ 向导	17
2.4.2 远程管理向导	18
2.4.3 缺省配置向导	18
2.5 安装补丁	19
2.6 其他平台	19

- 2.6.1 AIX 19
 - 2.6.2 HP – UX 20
 - 2.6.3 Solaris 22
 - 2.6.4 Linux 24
- 2.7 安装目录 25
 - 2.7.1 Windows 25
 - 2.7.2 AIX 26
- 2.8 安装文档 26
- 第 3 章 控制与管理 27
 - 3.1 MQ 控制命令 27
 - 3.1.1 MQ 队列管理器控制 27
 - 3.1.2 MQ 命令服务器控制 29
 - 3.1.3 MQ 监听器控制 30
 - 3.1.4 MQ 触发监控器控制 31
 - 3.1.5 小结 33
 - 3.2 MQ 对象管理 33
 - 3.2.1 队列管理器管理 34
 - 3.2.2 队列管理 35
 - 3.2.3 通道管理 35
 - 3.2.4 进程定义管理 37
 - 3.2.5 名称列表管理 37
 - 3.2.6 认证信息管理 37
 - 3.2.7 小结 38
 - 3.3 基本队列操作 39
 - 3.4 MQ 配置信息 39
 - 3.4.1 UNIX 配置文件 39
 - 3.4.2 Windows 注册表 40
 - 3.4.3 Windows 中 MQ 运行环境配置 41
 - 3.4.4 Windows 中 MQ 队列管理器配置 41
 - 3.5 MQ 管理方式 43
 - 3.5.1 本地管理 43
 - 3.5.2 远程管理 44
 - 3.6 日志 (Log) 46
 - 3.6.1 队列管理器日志 46
 - 3.6.2 检查点 (Checkpoint) 48
 - 3.6.3 记录和复原 (Record & Recover) 49
 - 3.6.4 备份和恢复 (Backup & Restore) 50
 - 3.6.5 导出日志 (Dump Log) 50
- 第 4 章 通信与配置 51
 - 4.1 消息路由 51
 - 4.1.1 消息路由过程 51
 - 4.1.2 缺省传输队列 51
 - 4.1.3 队列管理器别名 52
 - 4.1.4 多级跳 52
 - 4.1.5 传输中的消息 52

4.2	通道配置	54
4.2.1	Sender (QM1) — Receiver (QM2)	54
4.2.2	Server (QM1) — Receiver (QM2)	55
4.2.3	Server (QM1) — Requester (QM2)	56
4.2.4	Sender (QM1) — Requester (QM2)	56
4.2.5	通道启动命令	57
4.2.6	通道监控程序	58
4.3	通道的属性	58
4.3.1	通道会话	59
4.3.2	通道协议	61
4.4	通道的状态	62
4.4.1	公共状态 (Common Status)	62
4.4.2	当前状态 (Current-Only Status)	63
4.4.3	通道状态分析	64
4.5	互连配置举例	66
4.5.1	单向传送	66
4.5.2	双向传送	67
4.5.3	队列与队列管理器别名	68
4.5.4	三级跳	69
4.5.5	四级跳	70
第5章	应用设计	73
5.1	架构设计	73
5.1.1	两点间通信	73
5.1.2	多点间通信	73
5.1.3	同步和异步	74
5.1.4	Client/Server	75
5.1.5	Internet 通信	75
5.2	通信方式设计	77
5.2.1	进程间会话模式	77
5.2.2	系统间通信方式	77
5.3	并发设计	78
5.3.1	多读多写	78
5.3.2	共享与独占	78
5.3.3	对象绑定	78
5.3.4	队列管理器关闭	79
5.3.5	分发列表 (Distribution List)	79
5.4	消息设计	79
5.4.1	消息大小 (Message Size)	79
5.4.2	消息持久性 (Persistence)	80
5.4.3	消息优先级 (Priority)	81
5.4.4	消息超时 (Expiry)	82
5.5	发送设计	82
5.5.1	消息标识	82
5.5.2	消息类型	83
5.5.3	消息格式	83

- 5.5.4 应答队列 84
 - 5.5.5 动态队列 84
 - 5.5.6 用户替换 86
- 5.6 读取设计 86
 - 5.6.1 等待读取 (Wait & NoWait) 86
 - 5.6.2 信号中断 (Signal) 87
 - 5.6.3 截断消息 (Truncated Message) 87
 - 5.6.4 浏览消息 (Browse)..... 87
 - 5.6.5 格式转换 (Convert) 88
 - 5.6.6 消息匹配 (Match) 88
 - 5.6.7 回滚计数 (Backout Count)..... 88
 - 5.6.8 固化回滚计数 (Harden Backout) 89
- 5.7 容错设计 89
 - 5.7.1 出错处理 89
 - 5.7.2 报告消息 89
 - 5.7.3 死信消息 90
- 5.8 小结 90

第 6 章 消息处理 91

- 6.1 交易 (Transaction) 91
 - 6.1.1 概述 91
 - 6.1.2 本地交易 (Local LUW) 92
 - 6.1.3 全局交易 (Global LUW) 92
- 6.2 触发 (Trigger)..... 97
 - 6.2.1 原理 97
 - 6.2.2 触发方式 98
 - 6.2.3 配置 99
 - 6.2.4 触发过程 99
 - 6.2.5 并发 100
 - 6.2.6 通道触发 101
 - 6.2.7 触发 CICS 交易 102
- 6.3 报告 (Report) 103
 - 6.3.1 原理 103
 - 6.3.2 选项 103
 - 6.3.3 说明 105
- 6.4 分组与分段 (Group & Segment) 106
 - 6.4.1 消息组的发送 106
 - 6.4.2 消息组的接收 107
- 6.5 消息上下文 (Message Context) 108
 - 6.5.1 消息上下文的内容 109
 - 6.5.2 消息上下文的编程 110
- 6.6 死信处理 (DLQ Handler) 111
 - 6.6.1 死信消息 111
 - 6.6.2 死信队列处理器 111
- 6.7 数据转换 (Data Convert)..... 114
 - 6.7.1 转换方式 116

6.7.2	数据转换表 (Convert Table)	119
第 7 章	广播通信	121
7.1	分发列表 (Distribution List)	121
7.1.1	概念	121
7.1.2	配置举例	122
7.1.3	编程	123
7.2	发布和订阅 (Pub & Sub)	128
7.2.1	概念	128
7.2.2	安装	128
7.2.3	Broker 控制命令	128
7.2.4	Broker 网络	130
7.2.5	编程设计	132
7.2.6	发布/订阅命令	133
7.2.7	常见的问题	135
第 8 章	客户端	137
8.1	配置	137
8.1.1	Server 端配置	137
8.1.2	Client 端配置	138
8.2	用户出口	142
8.2.1	用户出口路径	142
8.2.2	排错	143
8.3	安全检查	143
8.4	触发 (Trigger)	144
8.5	跟踪 (Trace)	145
8.5.1	Windows	145
8.5.2	AIX	145
第 9 章	群集	146
9.1	相关概念	146
9.1.1	配置库 (Repository)	146
9.1.2	配置库队列管理器 (Repository Queue Manager)	146
9.1.3	群集通道 (Cluster Channel)	147
9.1.4	群集队列 (Cluster Queue)	147
9.1.5	群集传输队列 (Cluster transmission queue)	147
9.2	群集管理	147
9.2.1	对象属性	147
9.2.2	管理命令	148
9.2.3	管理任务举例	148
9.3	群集配置举例	149
9.3.1	例 1	149
9.3.2	例 2	151
9.3.3	例 3	153
9.3.4	例 4	156
9.3.5	例 5	158
9.3.6	例 6	159
9.4	多群集队列实例与共享队列组	163

- 9.5 群集负载用户出口 (Cluster Workload User Exit) 164
- 第 10 章 监控与性能 165
 - 10.1 事件 (Event) 165
 - 10.1.1 概念 165
 - 10.1.2 队列管理器事件 (Queue Manager Event) 165
 - 10.1.3 通道事件 (Channel Event) 168
 - 10.1.4 性能事件 (Performance Event) 169
 - 10.1.5 配置事件 (Configuration Event) 171
 - 10.1.6 事件消息 172
 - 10.1.7 事件监控 172
 - 10.1.8 实验一 :Queue Depth 173
 - 10.1.9 实验二 :Queue Service Interval 174
 - 10.2 性能设计 (Performance) 174
 - 10.2.1 队列管理器性能比较 175
 - 10.2.2 数据传递的性能比较 181
 - 10.2.3 性能优化 182
 - 10.2.4 小结 184
- 第 11 章 安全协议 185
 - 11.1 安全通信 185
 - 11.1.1 数据加密 185
 - 11.1.2 报文摘要 186
 - 11.1.3 数字签名 186
 - 11.1.4 SSL 187
 - 11.2 数字证书 188
 - 11.2.1 概念 188
 - 11.2.2 格式 188
 - 11.2.3 根签证书与自签证书 190
 - 11.3 WebSphere MQ 配置 SSL 190
 - 11.3.1 Server/Server 消息通道 190
 - 11.3.2 Client/Server MQI 通道 192
 - 11.3.3 SSL 相关的对象属性 193
 - 11.3.4 Client 端程序 194
 - 11.3.5 证书部署 194
 - 11.4 实例 1 根签证书 195
 - 11.4.1 准备证书 195
 - 11.4.2 配置队列管理器 196
 - 11.4.3 配置通道 197
 - 11.5 实例 2 自签证书 198
 - 11.5.1 准备证书 198
 - 11.5.2 配置队列管理器 200
 - 11.5.3 配置通道 200
- 第 12 章 用户出口 201
 - 12.1 概述 201
 - 12.2 Channel Exit 201
 - 12.2.1 Channel Exit 函数 204

12.2.2	Security Exit	205
12.2.3	Message Exit	207
12.2.4	Send Exit	209
12.2.5	Receive Exit	211
12.2.6	Message Retry Exit	212
12.2.7	Channel Auto-Definition Exit	213
12.2.8	Transport-Retry Exit	214
12.3	Data Conversion Exit	215
12.4	Cluster Workload Exit	216
12.5	Pub/Sub Routing Exit	219
12.6	MQ API Exit	219
12.6.1	设置	220
12.6.2	举例	221
12.6.3	编程设计	221
第 13 章	MQI 编程	223
13.1	编程入门	223
13.1.1	数据类型	223
13.1.2	数据结构	223
13.1.3	程序流程	223
13.1.4	例程	224
13.2	头文件	226
13.3	库文件	226
13.4	编程参考	226
13.4.1	MQCONN	226
13.4.2	MQCONNEX	227
13.4.3	MQDISC	227
13.4.4	MQOPEN	227
13.4.5	MQCLOSE	228
13.4.6	MQPUT	228
13.4.7	MQPUT1	228
13.4.8	MQGET	229
13.4.9	MQINQ	229
13.4.10	MQSET	229
13.4.11	MQBEGIN	230
13.4.12	MQCMIT	230
13.4.13	MQBACK	230
第 14 章	Java 编程	231
14.1	安装	231
14.2	编程设计	232
	例程	232
14.3	连接模式	233
14.4	用户出口	234
14.5	多线程	235
14.6	连接池	235
14.6.1	例 1 线程之间串行建立连接	236

- 14.6.2 例 2 线程之间并行建立连接 237
- 14.7 交易保护 238
 - 14.7.1 本地交易 (Local LUW) 238
 - 14.7.2 全局交易 (Global LUW) 238
- 14.8 Trace 239
- 第 15 章 JMS 编程 240
 - 15.1 JMS 对象 240
 - 15.1.1 Context 240
 - 15.1.2 ConnectionFactory 241
 - 15.1.3 Connection 242
 - 15.1.4 Session 242
 - 15.1.5 MessageConsumer 243
 - 15.1.6 MessageProducer 243
 - 15.1.7 MessageListener 244
 - 15.1.8 Message 245
 - 15.2 编程设计 246
 - 15.2.1 Persistence 246
 - 15.2.2 Priority 246
 - 15.2.3 Expiry 246
 - 15.2.4 Transaction 247
 - 15.2.5 Acknowledgment 247
 - 15.2.6 Message Selector 249
 - 15.2.7 Temporary Destination 249
 - 15.2.8 Durable Subscriber 249
 - 15.3 MQ JMS 运行环境 250
 - 15.3.1 JMS Interface 与 MQ JMS Object 250
 - 15.3.2 JNDI 250
 - 15.3.3 Client 253
 - 15.3.4 CCSID & Encoding 254
 - 15.4 ASF 255
- 第 16 章 ActiveX 编程 256
 - 16.1 MQAX 256
 - 16.1.1 程序设计 256
 - 16.1.2 编程参考 259
 - 16.1.3 跟踪信息 (Trace) 263
 - 16.2 MQAI 263
 - 16.3 ADSI 263
- 第 17 章 AMI 编程 265
 - 17.1 安装 265
 - 17.1.1 Windows 265
 - 17.1.2 AIX 265
 - 17.2 概念与配置 265
 - 17.2.1 概念 265
 - 17.2.2 配置 266
 - 17.3 C 编程 266

17.3.1	Object Level	267
17.3.2	High Level	271
17.4	Java 编程	275
第 18 章	PCF & AI 编程	277
18.1	PCF 编程	277
18.1.1	消息流程	277
18.1.2	消息格式	277
18.1.3	格式举例	280
18.2	AI 编程	281
18.2.1	消息流程	281
18.2.2	包的组成	281
18.2.3	编程	282
附录1	WebSphere MQ 进程一览表	289
Windows	平台	289
UNIX	平台	289
进程树		290
附录2	WebSphere MQ 命令一览表	291
队列管理器 (Queue Manager)		291
crtmqm	创建队列管理器 (Create Queue Manager)	291
dltmqm	删除队列管理器 (Delete Queue Manager)	291
strmqm	启动队列管理器 (Start Queue Manager)	292
endmqm	停止队列管理器 (End Queue Manager)	292
dspmq	显示队列管理器 (Display Queue Manager)	292
命令服务器 (Command Server)		292
strmqcvs	启动命令服务器 (Start Command Server)	292
endmqcvs	停止命令服务器 (End Command Server)	293
dspmqcvs	显示命令服务器 (Display Command Server)	293
Listener (监听器)		293
runmqlsr	运行监听器 (Run Listener)	293
endmqlsr	停止监听器 (End Listener)	294
触发监控器 (Trigger Monitor)		294
runmqtrmc	启动 Client 端触发监控器 (Run Trigger Monitor for Client)	294
runmqtrm	启动 Server 端触发监控器 (Run Trigger Monitor for Server)	294
Trace		294
strmqtrc	启动 Trace (Start Trace ,Windows 平台)	294
strmqtrc	启动 Trace (Start Trace ,HP - UX Solaris ,Linux 平台)	294
endmqtrc	停止 Trace (End Trace ,Windows 平台)	295
endmqtrc	停止 Trace (End Trace ,HP - UX Solaris ,Linux 平台)	295
dspmqtrc	显示 Trace (Display Trace ,HP - UX Solaris ,Linux 平台)	295
介质恢复 (Media Recover)		295
redmqimg	记录对象映像 (Record Object Image)	295
rcrmqobj	重建对象 (Recreate Object)	296
日志 (Log)		297
dmpmqlog	输出格式化日志	297
容量单元 (Capacity)		297

- dspmqcap 显示容量单元 (Display Capacity) 297
- setmqcap 设置容量单元 (Set Capacity) 297
- 权限信息 (Authority) 297
 - dmpmqaut 输出权限信息 (Dump Authority) 297
 - dspmqaut 显示权限信息 (Display Authority) 298
 - setmqaut 设置权限信息 (Set Authority) 299
 - amqoamd 输出授权信息 (OAM Dump) 300
- 运行环境 (Environment) 300
 - mqver 显示版本 (WebSphere MQ Version) 300
 - setmqprd 设置生产环境 (Set Production) 301
 - amqicsdn 安装补丁 (Install CSD)..... 301
- 高可用性 (High-Availability ,Windows 平台) 301
 - hadltmqm 删除队列管理器 (HA Delete Queue Manager) 301
 - hamvmqm 移动队列管理器 (HA Move Queue Manager) 301
 - haregtyp 注册队列管理器 (HA Register Type) 301
 - amqmsysn 检查模块版本信息 (System Check) 301
- 高可用性 (High-Availability 其他平台) 302
- 疑问交易 (In-Doubt Transaction)..... 302
 - dspmqtrn 显示疑问交易 302
 - rsvmqtrn 解决疑问交易 302
- 消息 (Message)..... 302
 - amqsput 往队列中放消息 (Server 程序) 302
 - amqsputc 往队列中放消息 (Client 程序) 303
 - amqsget 从队列中取消息 (Server 程序) 303
 - amqsgetc 从队列中取消息 (Client 程序) 303
- 工具 (Utility) 303
 - runmqsc 脚本命令服务器 (Run MQSC) 303
 - mqrc 原因码查询 (MQ Reason Code) 303
 - amqfirst MQ 第一步 ,仅 Windows 平台 304
 - amqapi API 试验程序 ,仅 Windows 平台 304
 - amqpcard MQI 明信片程序 ,仅 Windows 平台 304
 - amqmtbrn MQ Task Bar ,仅 Windows 平台 305
 - amqmjpse 准备 MQ 向导 ,仅 Windows 平台 305
 - amqmgse MQ 缺省配置 305
 - amqinfn MQ 信息中心文档 (MQ Info Center) 305
 - crtmqcvx 创建数据转换程序框架 (Create Conversion) 305
 - runmqdlq 运行死信队列处理器 (Run Dead-Letter Queue Handler) 306
 - runmqchi 运行通道初始化程序 (Run Channel Initiator) 306
 - runmqchl 运行通道 (Run Channel) 306
 - dspmqfls 显示对象对应的文件名 (Display Files) 307
 - setmqscp 设置服务连接点 (Set Service Connection Point , 仅 Windows 平台) 307
 - setmqcrl 设置无效论证列表 [Set Certificate Revocation List (CRL) LDAP Server Definitions , 仅 Windows 平台]..... 307
 - amqmcert Client 证书配置工具 (Utility for Certification) 308
 - ffstsummary FFST 文件摘要 (FFST Summary) 308

mqaxlev	显示 Code Level	309
amqrfdm	查询 MQ Cluster Repository	309
amquiregn	Registry 值列表工具	309
amqmdain	MQ 服务控制命令 ,仅 Windows 平台	310
amqmsrvn	COM 服务器 ,仅 Windows 平台	311
附录3	MQSC 命令一览表	312
RUNMQSC	312
执行脚本	312
抑制回显	312
检验脚本	312
远程管理	312
批处理	312
MQSC 命令	313
结构图	313
DEFINE	315
DELETE	320
ALTER	321
DISPLAY	323
CLEAR	329
START	329
STOP	330
RESOLVE	330
PING	330
RESET	331
REFRESH	331
SUSPEND	331
RESUME	332
参考书目	333

内 容 提 要

IBM WebSphere MQ 是一个优秀的消息中间件,它被广泛地应用于各种企业应用系统之间的互连,已经逐渐成为这方面的标准。本书从原理到实践全面系统地阐述了 IBM WebSphere MQ 产品的安装、配置、管理、设计、编程等各个方面,同时也介绍了产品的扩展功能和一些高级使用技巧。本书从功能上重点介绍了日志管理、死信处理、客户端、群集、交易、触发、报告、事件、分段与分组、分发列表、发布订阅、数据转换、用户出口、安全套接字、性能等。

全书覆盖了 WebSphere MQ 产品的所有相关知识,全文共分 18 章 4 个部分。1~2 章为基础部分,介绍了 WebSphere MQ 产品的原理和简单的安装过程。3~4 章为管理部分,介绍产品的控制、管理及配置。5~12 章为设计部分,介绍了应用设计中可能用到的各种产品高级功能和使用技巧。13~18 章为编程部分,讲解了各种编程模式和方法。

对于 WebSphere MQ 的初学者和使用者,可以从本书的第一部分和第二部分入手,相信通过深入的原理剖析和详细的管理操作,能够帮助这部分读者入门与提高。即使对与 MQ 无关的人员,也能够通读本书后对这类软件的设计思路和工作原理有一定的了解和启发。第三部分和第四部分是本书的精华,介绍了大量的高级功能与技巧,内含作者多年的经验积累和实例模型,对于 WebSphere MQ 设计和编程人员会有相当的吸引力,可以作为有一定经验人士的高级读物,也是相关开发人员必不可少的参考书。

本书注重实践,附有大量例程,以帮助读者在实践中加深理解,也为相关设计和开发人员提供了丰富的参考样例。所有例程都在 WebSphere MQ 5.3 环境下经过测试,供读者参考。全书语言生动并附有很多插图,易于理解。在专业相关的文字叙述上力求简捷,在内容与过程的安排上则力争翔实,使得读者能够容易地自己动手实践。相信本书能使读者的 WebSphere MQ 水平有所提高,从入门到精通。

由于作者水平所限,不足之处在所难免,恳请广大读者不吝指正。

前 言

从系统集成到系统整合

当今的 I/T 世界瞬息万变,各种应用技术层出不穷。企业的 I/T 系统在经过较长时间的建设后,往往会出现系统复杂、数据或功能重复、结构臃肿、难以互联的问题。企业的发展需要不断创新业务,新业务的需求可以通过新建系统来满足,也可以通过重新整合旧系统来满足。很多企业在经过一段时间的发展后会出现不少应用系统,各个系统之间往往互不相连或联系很少。每一次互连都需要单独设立一个项目,将双方的应用系统改造一番。有些应用系统本身又不断地采用新技术进行改造,如果业务本身没有发生变化,那只能是“新瓶装旧酒”。

我们在每次改造的时候,往往会加入一些新技术,与此同时也增加了一些复杂度。有很多的企业新项目并不完全是为了增加新功能,在很大程度上是原有系统功能上的覆盖,这样一来,原来的投资就白白浪费了。众多的企业项目往往是年年要改造,系统在诞生之初就面临着一年一小改,两年一大改,三年一替换的窘境。系统的稳定运行受到干扰,设计效益很难达到,这在很大程度上是因为系统建设的思路存在问题。人们往往倾向于追求一种新技术来创建一套近乎完美的能解决所有问题的复杂系统,这种系统在结构上具有高度的聚合集成能力。

新技术在出现之后往往表现为两种截然不同的方式:一种方式是标新立异,与之前的所有现存技术完全不同,甚至格格不入。技术的拥趸者有时会构想出全方位或大一统的理想境界。但实际上,接受它的同时也意味着对老技术的否定,且往往意味着较大的投资风险。经过一段时间的锤炼后,人们发现所谓“一统天下”的状态只是短暂的,很快又有无法解决的问题不断涌现,为了解决新的问题,系统又很快会出现很多例外。这就好像在宫殿边上进行违章搭建,让人感觉不快,并且随着搭建的增多,宫殿的原貌被破坏殆尽。

另一种方式是改良更新,脱胎于旧技术又不同旧技术,过去的概念和方法仍然有所体现,这种技术往往比较容易被接受,技术的推广者往往勾勒出只要不断升级就能始终领先的画面。不幸的是,频繁地升级、迁移、更新所带来的不便与困扰一点也不比前一种少,而且通常经过了一段时间后它的发展会被原先的技术框架所限制,到了不得不“伤筋动骨”转型或面临淘汰的地步。

无论是哪一种方式都是应用需求更新领先于技术能力更新的表现。其实,在当今世界的各种复杂 I/T 系统中,永远是集中与分散并存、各种新老技术并存的局面。就像是一个有机的生物体,有能力进行自身的新陈代谢,自我更新,各个子系统分工明确,互相之间又具备有机的联系。

我们身处的世界是一个多样化且不断变化更新的世界。要适应这样一个复杂多变的环境,系统整合是一个很好的办法。我们对系统的整合不应该执着于用一种技术将其全盘替换或更新,而是应该试图寻找一种灵活的办法,将它们有机地联系起来。换句话说,不是要改造各个系统,而是要加强各个系统之间的联系,从而更好地使用已有的子系统。

IBM WebSphere MQ 家族就是一个优秀的用于应用系统间联系的软件,它的原理其实很简单也很容易理解,就是消息传递。应用系统就像一群有机生物一样,它们之间可以各自活动,也可以相互交流。通过消息传递,将它们有效地联系起来,每一个应用系统都可以对外提供自身的功能,也可以要求其他应用系统作为它的下一处理环节,消息是应用系统之间请求、应答和中间结果的载体。这样一来,不断流动的消息将松耦合关系的应用系统串起来,使它们之间的关系变成了功能叠加。

消息驱动和消息触发

我们在构建一个应用系统的时候,往往会将其划分成多个模块,各个模块之间需要约定接口规范。对

于消息驱动模块之间需要约定的是消息的报文格式、通信模式和功能定义。报文格式也就是双方模块都能理解的消息语言,如 XML。对于跨网络、跨平台的消息,报文格式应该能够屏蔽双方信息编码上的差异(如 ASCII 编码或 EBCDIC 编码),屏蔽双方信息表达上的差异(如整数的高低字节、浮点数的表示,32 位/64 位的整数长短,等等)。通信模式也就是双方通话的方式,如双向的一问一答方式、单向的汇报方式、点对点方式以及一对多广播方式。对于跨网络、跨平台的消息,通信模式还应该约定双方的网络层通信协议和应用层通信协议。功能定义也就是说模块在收到一条消息后应该做的相应动作。有了这些约定,模块之间就可以通过消息流转将各个模块的功能发挥出来,形成对外的业务功能。如果要增加模块,原有的模块可以不需要改动。如果要改变业务功能或业务流程,可能需要改变的只是消息流转的次序和方式。

消息驱动结构的系统中几乎所有的模块都在等待消息,在消息到来后进行相应的处理,处理结束后又回到这个点等待下一条消息。消息源可以是一个文件,也可以是系统队列、数据库、网络连接,等等,可谓五花八门。监听消息的程序通常被称为监听器(Listener),通常每个模块有各自不同的监听器,在系统空闲的时候,这些模块虽然不在工作,但也一直占用着系统资源。如果模块共用系统提供的监听器,则在监听器上可以设立触发机制(Trigger),在消息到达的时候来启动相应的模块进行处理,这样一来,在系统空闲的时候,只需要开启系统监听器即可,所有的工作模块都可以休息了。

IBM WebSphere MQ 家族就是基于消息驱动和消息触发原理设计的。底层是面向消息的中间件 WebSphere MQ,上层有面向消息整合的 WebSphere MQ Integrator 和面向消息流程的 WebSphere MQ Workflow 两个产品。本书只涉及 WebSphere MQ 产品。

记 号 约 定

A. B
本书中用 A. B 表示对象 A 的属性 B,或者结构 A 中的域 B。如 Queue. CLUSNL 表示队列的 CLUSNL 属性, MQMD. Format 表示消息头 MQMD 结构中的 Format 域。

Config or Code

本书采用阴影来表示代码、数据结构、配置脚本等内容。

```
QM1 :  
-----  
DEFINE    QREMOTE    (QR _ QM2)                +  
          RNAME      (QL _ QM2)                +  
          RQMNAME     (QM2)                    +  
          XMITQ       (QLX _ QM2)              +  
          REPLACE
```

< MQ _ HKEY >

Windows 平台上的 WebSphere MQ 的所有属性设置都存放在注册表中,具体说来,在 \\HKEY _ LOCAL _ MACHINE\\SOFTWARE\\IBM\\MQSeries\\CurrentVersion\\ 路径之下。为了表达简便,我们将其称为 < MQ _ HKEY >。

例如 :< MQ _ HKEY > \\Configuration\\AllQueueManagers\\
\\HKEY _ LOCAL _ MACHINE\\SOFTWARE\\IBM\\MQSeries\\CurrentVersion\\Configuration\\AllQueueManagers\\

有时因为注册表路径太深,用 < MQ _ HKEY > 仍觉得不方便,我们使用进一步的缩写。

< MQ _ HKEY _ Service > 表示 < MQ _ HKEY > \\Configuration\\Services\\ < QMgr >
< MQ _ HKEY _ QM > 表示 < MQ _ HKEY > \\Configuration\\QueueManager\\ < QMgr >
这里 < QMgr > 表示队列管理器名。

第 1 章 概念与原理

1.1 简介

计算机软件发展到今天,很多具有独立功能的应用模块都被逐渐隔离出来形成软件产品,这些软件往往是针对某一种应用需求,在相关的领域中具有很强的通用性。它们通常界于操作系统和应用程序之间,为应用程序提供一些标准的服务,我们称这一类软件为中间件。中间件根据其应用领域也分成多种,比如交易中间件、消息中间件、Web 中间件等。

WebSphere MQ 本质上是一种消息中间件,用于保证异构应用之间的消息传递。应用程序通过 MQ 接口进行互连通信,可以不必关心网络上的通信细节,从而将更多的注意力集中于应用本身。MQ 在所有的平台上有统一的操作界面,这使得应用程序可以很方便地移植到各种操作系统中。

1.1.1 消息中间件

在早先的时候,人们往往使用两个程序之间直接通信的方式。这种办法最大的问题就是通信协议相关性,也就是说与通信协议相关的代码充斥在应用程序之中,而且可能出现在程序的任何地方,甚至影响程序的设计与结构。这种办法的另一个问题就是应用程序不容易写出可靠强壮的代码。应用程序的通信部分会因为要考虑到工作方式的灵活性、网络协议的通用性以及为实现一些实用功能而变得非常庞大,往往超过应用程序本身的逻辑代码,变得本末倒置,且代码很难写好,也很难维护。

人们慢慢意识到应该把通信代码放到外面,变成独立的工作进程或工作模块,不同的工作进程可以适用于不同的通信协议,而应用程序与通信程序之间使用通用的本地通信方式。这样一来,应用程序与通信程序的代码完全分开,各自的逻辑清晰自然,易于管理与维护,在通信方式上前进了一大步。但是,这种方式对通信程序的编程要求比较高,如果考虑到平台的广泛适用性,通信程序可能要写一大堆,要设计一个通用高效的本地通信接口也并非易事,再考虑到通信上的一些附加功能,其实现对普通编程人员是有一定困难的。人们很自然地想到,这种工作最好交由专业的通信软件来完成。

于是,市场上逐渐出现了专门负责消息通信的软件。它通常是一个独立运行的通信环境,有统一的编程调用接口,可以跨平台、跨协议。不同结点之间的软件可以通过配置相互连通,搭起统一的通信平台,从而产生更强大的功能。这种通信软件往往安全可靠、配置灵活,其地位在操作系统之上,在应用程序之下,所以被称为消息中间件(MOM)。WebSphere MQ 就是其中的一种。

1.1.2 WebSphere MQ

WebSphere MQ 是 IBM 公司于 2003 年 2 月推出的面向消息传递的中间件产品,是 IBM MQSeries 产品线的最新力作。目前的版本是 5.3,它的前身是 MQSeries 5.1 和 MQSeries 5.2。到了 5.3 版,产品的功能更丰富、更集成,与 IBM 公司的电子商务应用中间件 WebSphere 产品有很高的集成度和亲和性,从产品更名为 WebSphere MQ 这一点就可见一斑。事实上,WebSphere Application Server 5.0 版就可以内置安装一个嵌入式的 WebSphere MQ。

WebSphere MQ 在面向消息传递的中间件(Message Oriented Middleware, MOM)市场中赫赫有名,是用来连接异构平台之间企业应用的专业产品。通过 WebSphere MQ 可以屏蔽不同的通信协议之间的差别,可以最大限度地简化网络编程的复杂性。通过 MQ 的配置,通信双方的程序可以以松耦合的方式独自运行,并不关心对方所在的位置和状态,通过消息驱动或消息触发的方式来相互联系。它支持多种平台,对消息支持交易式的提交和回滚。

WebSphere MQ 产品主页为 <http://www.ibm.com/software/ts/mqseries> ,相关的 WebSphere MQ 产品家族主页为 <http://www-3.ibm.com/software/integration/mqfamily/>。在相关网站上可以下载有效期为 90 天的 WebSphere MQ 试用版。

WebSphere MQ 的补丁被称为 CSD (Corrective Service Diskette) ,比如 CSD01 就是一号补丁 ,后一号的补丁其内容完全覆盖前一号的补丁。WebSphere MQ 补丁可以在产品首页中找到 ,其下载网址为 <http://www-3.ibm.com/software/integration/mqfamily/support/summary/>。

WebSphere MQ 还有大量的免费资料可以下载 ,其中包括 Client 端软件、例程代码、相关工具、测评文档等 ,内容十分丰富。IBM 公司称之为 MQ SupportPac ,下载网址为 <http://www-3.ibm.com/software/integration/support/supportpacs/>。

1.1.3 WebSphere MQ 产品

IBM 公司将 WebSphere MQ 分成 3 个不同版本的产品 :WebSphere MQ Express ,WebSphere MQ ,WebSphere MQ Extended Security Edition。其中 ,WebSphere MQ 是核心产品 ,本书也只针对这个产品进行介绍。

基本上 ,WebSphere MQ 包含了 WebSphere MQ Express 的全部功能 ,而 WebSphere MQ Extended Security Edition 又包含了 WebSphere MQ 的全部功能。它们的功能关系如图 1 - 1。

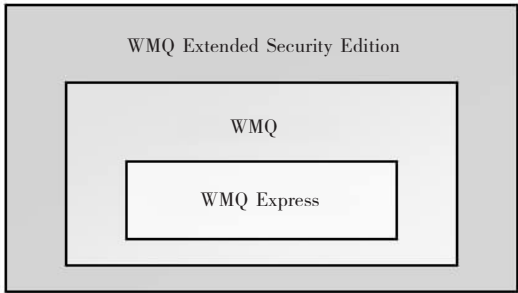


图 1 - 1 WebSphere MQ 产品

1.1.3.1 WebSphere MQ Express

WebSphere MQ Express 可以认为是 WebSphere MQ 的简化版 ,或称为体验版。它由 WebSphere MQ Client 和 WebSphere MQ Server 组成 ,其中 Client 部分可以在 Windows、AIX、HP-UX 和 SUN Solaris 平台上运行 ,每一个 UNIX 平台又可以有 SSL 附加功能。但 Server 部分仅限于 Windows 和 Linux for Intel 平台 ,在使用上也有一定的限制。一个队列管理器的通道连接最多只能有 10 个 ,Client 端通道连接最多也只能有 10 个 ,消息大小不能超过 4MB (当然 ,可以用消息分组的办法突破这一限制) ,等等。WebSphere MQ Express 也不能所有远程资源管理器加入全局交易中。

另外 ,WebSphere MQ Express 产品提供了一些独有的程序。比如 :Express 产品漫游 ,Express 文件传送 ,Express 信息中心等。但是 ,在功能上并没有什么创新 ,仍然被 WebSphere MQ 所包含。

WebSphere MQ Express 适合于那些中小型企业应用 ,在并发度上要求不高 ,传递数据的量也不大 ,投资较少。WebSphere MQ Express 也适用于树状结构中下一层的结点 ,这些结点通常对连接数要求不高。

1.1.3.2 WebSphere MQ

WebSphere MQ 是核心产品 ,它包含了 WebSphere MQ Express 的全部 ,也突破了连接上的限制。完整的版本由 Client 和 Server 及相应的文档组成。Server 部分支持 Windows、AIX、HP-UX、Solaris、Linux for Intel、Linux for zSeries 和 iSeries 平台。Client 部分除了支持所有 Server 平台之外还支持很多其他平台 ,可以在 SupportPac 网站上免费下载。

Client 分为普通 Client 和 Extended Transactional Client。后者可以支持 Server 所有远程资源管理器加入全局交易中。当然 ,这种环境需要有全局资源管理器 ,比如 WebSphere、CICS、Tuxedo 等 ,具体参见“交易”一章。对于 UNIX 平台 ,普通 Client 又可以选用支持 SSL 协议 ,用于配置基于 SSL 的 Client/Server MQI 安全

通道。

WebSphere MQ 是构建完整的异构应用互连平台所必需的 ,由于对连接数量没有限制 ,可以根据需要架构各种复杂的网状或树状结构 ,通常对于大中型企业应用。

1.1.3.3 WebSphere MQ Extended Security Edition

WebSphere MQ Extended Security Edition 由 WebSphere MQ 加上 Tivoli Access Manager for Business Integration (TAMBI) 组成。它除了具有 WebSphere MQ 的全部功能 ,还可以在保证数据传送之外 ,具有 TAMBI 的功能 ,提供一个集成的 MQ 安全环境 ,可以对消息加密传送 ,可以生成数字签名 ,等等。大型企业应用和敏感数据传送应用可以选择这个产品。

1.2 概念与对象

WebSphere MQ 运行环境中较多的概念 ,其中有一部分是可以作为实体进行的操作的 ,称为 MQ 对象。每一个对象都有各自的属性 ,不同的属性决定了对象的特性和工作方式。消息、队列、队列管理器和通道是 MQ 中最重要的概念和对象。

1.2.1 消息 (Message)

消息是 WebSphere MQ 中最小的概念 ,本质上就是一段数据 ,它能被一个或多个应用程序所理解 ,是应用程序之间传递的信息载体。消息可以大致分成两部分 :应用数据体和消息数据头 ,如图 1-2。

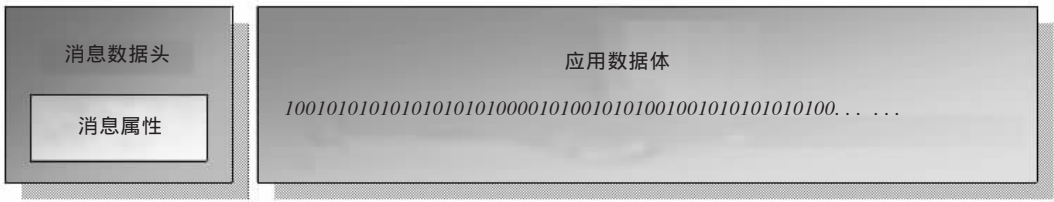


图 1-2 消息的结构

- 消息数据头是对消息属性的描述 ,这段信息往往被队列管理器用来确定对消息的处理。消息数据头可以由应用程序或系统的消息服务程序共同产生 ,它包含了消息在传送中的必要信息 ,如目标队列管理器的名字、目标队列的名字以及消息的其他一些属性。
- 应用数据体是应用间传送的实质的数据消息 ,它可以是字串、数据结构甚至二进制数据。包含的内容可以是文本、文件、声音、图像等任何数据 ,这些数据只对特定的应用具有特定的含义。所以 ,应用数据体的结构和内容 by 应用程序定义 ,通信双方需要事先约定报文格式。

消息可以分成持久 (Persistent) 消息和非持久 (Non-Persistent) 消息。“持久”的意思就是在 WebSphere MQ 队列管理器重新启动后 ,消息仍然能保持。

在 WebSphere MQ 中 ,消息可以是有限长度的任何一段信息。所谓有限长度 ,在开放平台 (Windows NT/2000/XP ,AIX ,HP-UX ,Solaris ,Linux) 上缺省是 4MB ,但这个上限可以调整到 100MB。也就是说 ,每段信息应该控制在这个有限长度内。当然 ,这并不限制 WebSphere MQ 用来传递更大的信息 ,比如文件。相关内容参见“分组与分段”章节。

1.2.2 队列 (Queue)

我们可以简单地把队列看成一个容器 ,用于存放消息。队列按其定义可分成本地队列、远程队列、别名队列和模型队列 ,如图 1-3。其中只有本地队列是真正意义上的队列实体 ,可以存放消息。远程队列和别名队列只是一种队列定义 ,指向另一个队列实体。远程队列指向的是其他队列管理器中的队列 ,而别名队列指向的是本地队列管理器中的队列。模型队列有一点特殊 ,它虽然本身只是一个队列定义 ,描述了模型的属性 ,但是当打开模型队列的时候 ,队列管理器会以这个定义为模型 ,创建一个本地队列 ,被称为动态队列。

一个队列管理器下辖很多个消息队列 ,但每个队列却只能属于一个队列管理器。队列在它所属的管理器中只能有一个惟一的 名字 ,不能与同一个管理器的其他队列重名。当消息被添加到队列中 ,它缺省将被加到最后 ,与之相反 ,删除消息缺省却是从头开始。

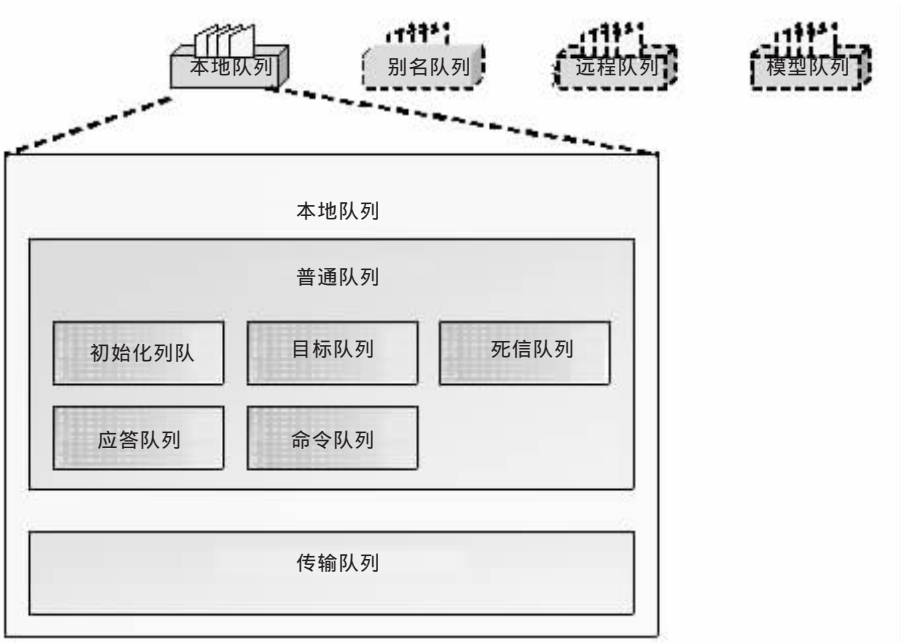


图 1-3 队列的分类

1.2.2.1 本地队列

本地队列按功能又可分成初始化队列、传输队列、目标队列和死信队列。初始化队列用做消息触发。传输队列只是暂存待传的消息 ,在条件许可的情况下 ,通过管道将消息传送其他的队列管理器。目标队列是消息的目的地 ,可以长期存放消息。如果消息不能送达目标队列 ,也不能再路由出去 ,则被自动放入死信队列保存 ,如图 1-4。

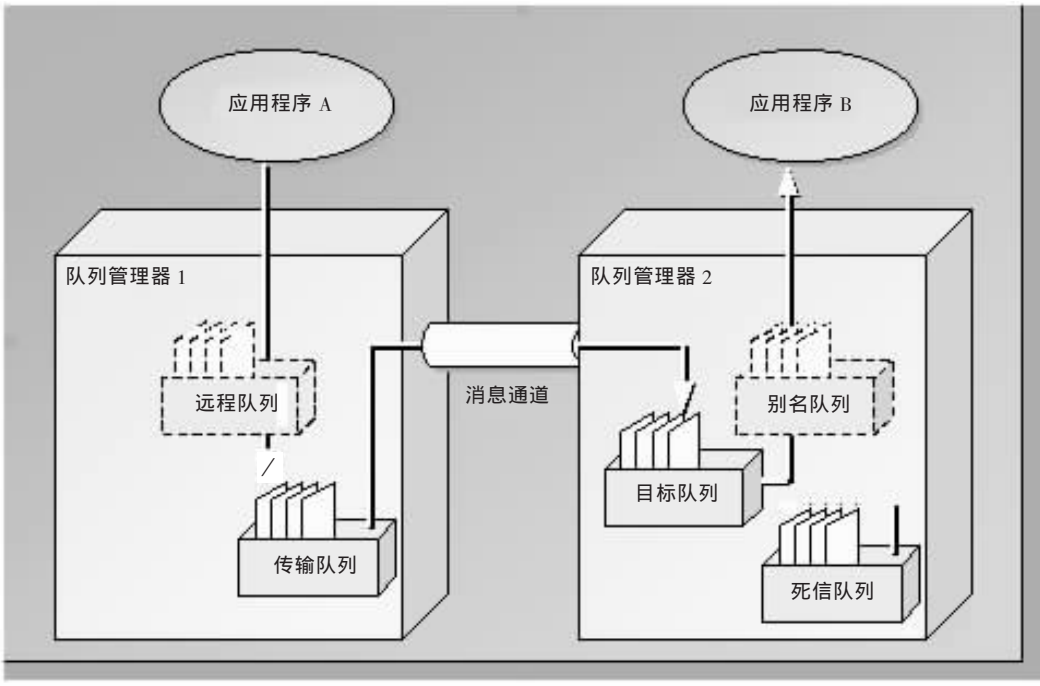


图 1-4 各种队列在消息传送时的作用

1.2.2.1.1 普通队列

能够真正长期存放消息的本地队列,我们称之为普通队列。一般说来,应用程序只对其做简单的 MQGET 和 MQPUT 以收发消息,这也是系统中用得最多的消息队列。通常在不会引起混淆的情况下,我们也将普通本地队列简称为本地队列。

1.2.2.1.2 传输队列

要送往远地的消息将放入传输队列。在适当的时候,消息会被从传输队列中取出并送往远地,最终放入远端的本地队列。所以,从本地系统的角度来看,传输队列是用来暂时存放输出消息的。

传输队列本身是一个本地队列,它与普通队列的差别在于传输队列特有的属性 USAGE (XMITQ)。

1.2.2.1.3 初始化队列

初始化队列是配合消息触发用的,如果队列上配置有消息触发功能,则需要指定另一个相关队列以存放触发消息,这个队列就是初始化队列。初始化队列本质上就是一个普通本地队列。

1.2.2.1.4 目标队列

在消息通信的时候,消息最终的目的地称为目标队列。如果消息是通过传输队列转发的,WebSphere MQ 会自动为消息体添加一个传输消息头,其数据结构为 MQXQH。其中的 RemoteQName 和 RemoteQMgrName 两个域指明了目标队列和目标队列管理器。如果消息被放入死信队列,则 WebSphere MQ 会自动为消息体添加一个死信消息头,其数据结构为 MQDLH,其中 DestQName 和 DestQMgrName 两个域指明了原消息的目标队列和目标队列管理器。

1.2.2.1.5 死信队列

死信队列本质上是普通的本地队列,由于队列管理器的 DEADQ 属性指定该队列为死信队列,所以队列管理器认为无法投递的消息都被自动送去该队列。因无法投递的消息很像信件投递中的死信,故而得名。

队列管理器在将消息放入死信队列的时候,会自动为消息体添加一个死信消息头,其数据结构为 MQDLH。其中 Reason 域指明了消息无法投递的原因。

1.2.2.1.6 应答队列

消息在发送后需要得到对方的回应。这种回应可以是系统自动产生的报告消息,也可以是对方应用生成的应答消息。就应用而言,这些回应消息的目标队列就是应答队列。应答队列通常设置在消息头 (MQMD) 的 ReplyToQ 域中,它也总是与消息头中的另一个域 ReplyToQMgr 一起使用。

1.2.2.1.7 命令队列

命令队列指的是 WebSphere MQ 队列管理器中预定义的 SYSTEM.ADMIN.COMMAND.QUEUE。任何 MQSC 命令都可以送往该队列,并被队列管理器的命令服务器 (Command Server) 接收处理。

1.2.2.2 别名队列

别名队列只是一种队列定义,它自身只是队列的逻辑名字,并不是队列实体本身。别名队列的 TARGQ 属性指明了其代表的目标队列名称,目标队列通常是本地队列。其实别名队列只是提供了队列名称之间的映射关系,可以将别名队列看做指针,指向其目标队列。注意,这里的目标队列自身也可以是一个别名队列,指向下一个目标队列。然而,在对别名队列进行打开 (MQOPEN) 操作时,只允许别名队列指向一个普通本地队列,即一层映射关系。如果存在上述的多层映射关系,则报错 MQRC _ ALIAS _ BASE _ Q _ TYPE _ ERROR。

通过别名定义,WebSphere MQ 也可以动态改变消息流向。比如:某程序在代码中指定消息写入队列 A,但是希望在不变动代码的情况下将消息写入队列 B。这时只要定义别名 B,使之指向 A 即可。再如:某队列 C 是可读可写的本地队列,管理员希望它对于一类程序只可读,对另一类程序只可写,这时可以对 C 定义两个别名 D 和 E,D 只允许读,E 只允许写。也就是说,本地队列 C 的属性为 GET(ENABLED) 且

PUT(ENABLED),将 D 的属性设置为 GET(ENABLED)且 PUT(DISABLED),E 的属性为 GET(DISABLED)且 PUT(ENABLED)。将 D 和 E 分别提供给上述两类程序使用,这样可以避免应用程序的误操作。

1.2.2.3 远程队列

远程队列与别名队列类似,也只是一个队列定义,用来指定远端队列管理器中的队列。使用了远程队列定义,程序就不需要知道目标队列的位置(所在的队列管理器)了。

远程队列定义包括目标队列管理器名和目标队列名,而这种队列定义对于访问地的应用程序是透明的。这种技术不但使应用程序只需要对一个简单的队列名操作,而且可以通过在线修改远程队列定义,动态地改变路由。

1.2.2.4 模型队列

模型队列定义了一套本地队列的属性集合,一旦打开模型队列,队列管理器会按这些属性动态地创建一个本地队列。模型队列的 DEFTYPE 属性可以取值 PERMDYN 和 TEMPDYN,分别代表永久动态队列和临时动态队列。

1.2.2.4.1 永久动态队列

永久动态队列由模型队列动态创建,并可以永久存在。在调用 MQOPEN 时创建,以后就和普通的本地队列一样工作。在调用 MQCLOSE 时,缺省情况下会保留消息和队列,当然也可以通过设置关闭选项(Close Option)来清除消息甚至删除永久动态队列。

1.2.2.4.2 临时动态队列

临时动态队列也是由模型队列动态创建,但只在会话(Session)中临时存在。在调用 MQOPEN 时创建,在同一个线程中 MQCLOSE 时关闭并自动删除。MQCLOSE 时无所谓关闭选项(Close Option)的取值。

1.2.3 队列管理器(Queue Manager)

队列管理器构建了独立的 WebSphere MQ 的运行环境,它是消息队列的管理者,用来维护和管理消息队列。一台机器上可以创建一个或多个队列管理器,每个队列管理器有各自的名字,通常情况下,它不能与网络中的其他队列管理器重名。如果我们把队列管理器比作是数据库,那么队列就是其中的一张表,消息就是表中的一条记录。

队列管理器是负责向应用程序提供消息服务的机构。在 WebSphere MQ 中,队列管理器集对象的定义、配置、管理、调度以及提供各种服务的功能于一身。WebSphere MQ 的系统管理工具提供了对系统部件配置与管理的功能,应用程序必须首先连接到队列管理器,然后在队列管理器的控制下对各种对象进行操作。

WebSphere MQ 中的队列管理器可以含有很多个队列,但一个队列只能属于一个队列管理器。一个操作系统平台可以创建一个队列管理器,也可以创建多个队列管理器。队列管理器、队列、通道等都是 WebSphere MQ 的对象,所有的对象都有各自的属性,有些属性必须在对象创建的时候指定,有些可以在创建以后更改。

1.2.4 通道(Channel)

通道是两个队列管理器之间的一种单向的点对点的通信连接,消息在通道中只能单向流动。如果需要双向交流,可以建立一对通道,一来一去。站在队列管理器的角度,这一对通道可以按消息的流向分成输入通道和输出通道。通过配置,对于放入本地传输队列中的消息,队列管理器会自动将其通过输出通道发出,送入对方的远程目标队列。

两个队列管理器之间可以有多条通道负责传输不同的内容,这样设计往往是为了将不同优先级的消息错开,运行于不同速率的网络连接上。或者即便是所有通道都运行于相同的网络物理连接上,也可以将

不同大小的消息传送分开 ,以免小数据传送被大文件所堵塞。如果多条通道共享一条网络物理连接 ,通道的速率之和受限于网络速度。这样可以增加传送的并发度但并不能增加整体的传送速度。

在通道上可以配置不同的通信协议 ,这样就使得编程接口与通信协议无关。通道两端的配置必须匹配 ,且名字相同 ,否则无法连通。队列管理器之间的通信是通过配置通道来实现的 ,通道两侧的队列管理器对这个通道的相关参数应该能对应起来 ,一个通道只能用一种通信协议 ,但不同的通道可以有不同的通信协议。可见 ,通道是架设在通信协议之上的对象 ,架设在不同通信协议上的通道在应用层看来都大同小异。

1.2.4.1 通道类型 (Channel Type)

WebSphere MQ 用通道类型属性 (CHLTYPE) 约定了通信双方在连接握手协议中的主动方和被动方以及应用消息的流向。可选以下这些类型：

- SDR (Sender)握手协议的主动方 ,消息的发送方
- RCVR (Receiver)握手协议的被动方 ,消息的接收方
- SVR (Server)在握手协议中可以是主动方也可以是被动的方 ,消息的发送方
- RQSTR (Requester)在握手协议中可以是主动方也可以是被动的方 ,消息的接收方
- CLNTCONN (Client Connection)在 Client-Server 连接时 ,定义客户端连接定义表 (Client Channel Definition Table) 时使用。握手协议的主动方 ,消息的发送方
- SVRCONN (Server Connection)在 Client-Server 连接时 ,定义服务器端连接时使用。握手协议的被动方 ,消息的接收方
- CLUSSDR (Cluster Sender)在群集中发送配置信息和应用消息。握手协议的主动方 ,消息的发送方
- CLUSRCVR (Cluster Receiver)在群集中接收配置信息和应用消息。握手协议的被动方 ,消息的接收方

通信双方的通道类型配对并不是可以随意排列组合的 ,共有 6 种 ,如图 1 - 5：

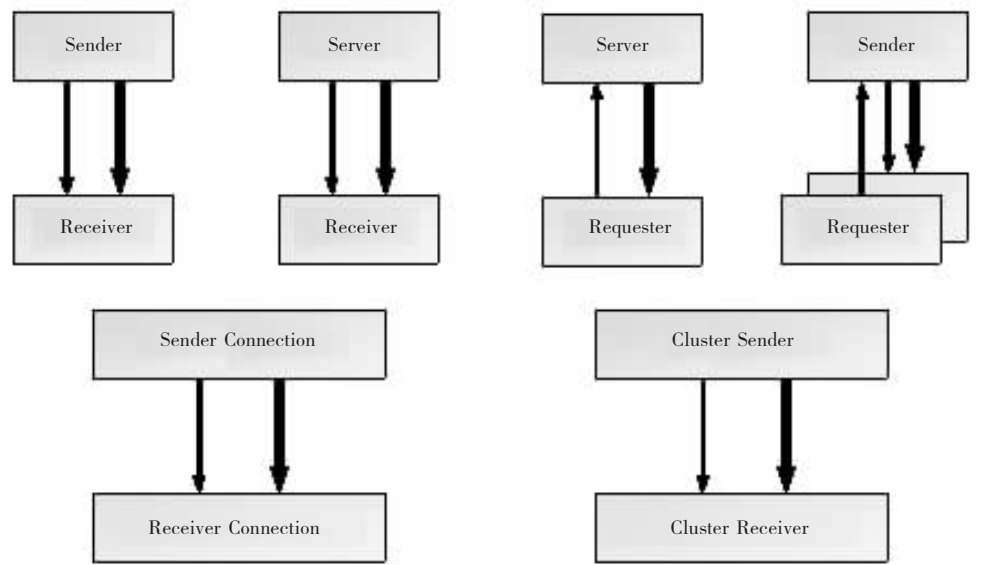


图 1 - 5 通道类型的配对

图中细线箭标表示握手协议中的主动连接 ,粗线箭标表示应用消息流向。消息在所有的通道上都是单向传送的。

- Sender/Receiver 是所有连接中最简单、最常用的一种。Sender 是通道主动方 ,也是消息发送方。
- Requester/Server 也是常用的一种连接方式。Requester 是通道主动方 ,但通道连接后 ,它作为消息接收方 ,Server 是消息发送方。
- Server/Receiver 与 Sender/Receiver 类似 ,Server 是消息的发送方 ,也是连接的主动方。与 Sender 定

义类似, Server 定义中必须指定 CONNAME 参数。

- Sender/Requester 的连接过程稍微复杂一些, Requester 首先与 Sender 连接, 在通知对方连接参数后连接断开。Sender 进行反向连接, 消息也是反向传送的。这种反向连接的方式, 称为 Callback Connection。
- Sender Connection/Receiver Connection 与 Sender/Receiver 方式相同。用于 Client/Server 之间的 MQI 通道。
- Cluster Sender/Cluster Receiver 与 Sender/Receiver 方式相同。用于群集中队列管理器之间的连接。

由于 Sender/Receiver、Server/Receiver 的连接主动方和消息发送方相同, 所以可以在发送端设定通道触发 (Channel Trigger)。

对于 Sender/Receiver、Server/Receiver、Requester/Server 类型的连接, 被动方事先不需要知道主动方的连接参数, 所以可以用于连接主动端是动态地址的应用场合。

Sender/Requester 型连接有反向建立连接的功能, 常用于双向安全认证。

1.2.4.2 消息通道协议 (MCP)

消息通道协议是 WebSphere MQ 用来传递消息时使用的通信协议。MCP (Message Channel Protocol) 可使用多种底层通信协议传递消息 (LU6.2, DECNet ...)。消息通道协议使得消息的传送独立于通信协议, 应用程序通过统一的接口与 MQ 打交道, 而不再需要关心通信层使用的是 TCP/IP 还是 SNA。目前 MCP 支持的通信协议有 LU6.2、DECNet 和 TCP/IP。

不同操作系统的 WebSphere MQ 支持的通信协议的数量和种类可能稍有不同, 详情请参见本操作系统的《WebSphere MQ 用户手册》。

1.2.4.3 消息通道代理 (MCA)

消息通道代理 (MCA, Message Channel Agent) 本质上是一个通信程序, 它用来在队列管理器之间传递消息。通道可以以进程的方式工作, 即独立的 MCA 进程; 也可以以线程的方式嵌入系统 MCA 进程中工作。对于前者, 根据不同的 MCP, 发送端进程和接收端进程的 MCA 名通常是不同的。

1.2.5 名称列表 (Name List)

名称列表是 WebSphere MQ 的一种对象, 它实质上是多个其他 WebSphere MQ 对象的名称集合。其内容由多个字符串组成, 中间用逗号隔开, 每个字符串就是一个对象名称。

名称列表本身无法代表它所含的对象, 例如无法对名称列表进行 MQPUT 或 MQGET 操作, 类似的操作应该由分发列表 (Distribution List) 完成。定义名称列表只是定义了一个集合, 往往是为了方便应用访问多个对象。比如应用程序动态地从名称列表中读出操作对象并依次进行操作, 如果操作的对象有所增减, 只需要修改名称列表即可。名称列表使管理人员可以在不修改应用的前提下, 通过动态地增减名称列表中的内容来进行管理。

名称列表多用于群集 (Cluster) 环境中指定一个队列管理器同时属于多个群集的情况, 这时名称列表的内容就是多个群集的名称集合。名称列表可以用于以下一些对象属性:

- QMgr. REPOSNL
- QMgr. SSLCRLNL
- Queue. CLUSNL
- Channel. CLUSNL

WebSphere MQ 中每个对象都有各自的属性, 它们中的大多数是可以创建后修改的。这里, 我们采用“对象. 属性”的记号方式表示对象的属性, 例 Queue. CLUSNL 表示队列的 CLUSNL 属性。以下同。

1.2.6 分发列表 (Distribution List)

分发列表可以使 WebSphere MQ 应用程序一次将一条消息同时发送到多个队列上。这里的一次发送

指调用一次 MQPUT 或 MQPUT1 ,多个目标队列可以是本地队列也可以是远程队列 ,如果多个远程队列的目标队列管理器相同 ,则在网络上只需要传送一次即可 ,节省了网络开销。当消息到达目标队列管理器后 ,再自动分发到各个目标队列中 ,当然这要求源队列管理器和目标队列管理器都支持分发列表功能。

分发列表的操作是可以在一个交易中完成的 ,也就是说 ,多个队列的发送是可以一起提交或回滚的。

1.2.7 进程定义 (Process)

进程定义对象用于 WebSphere MQ 的触发机制中 ,用来描述触发程序的对象。这个程序可以是一个操作系统程序 ,可以是一个 MQ 应用 ,也可以是一个 CICS 交易。在进程定义的属性中需要设定触发程序的路径、名称、参数等信息。

在消息触发环境中 ,一旦触发条件满足即可引起触发 ,队列管理器在生成触发消息的时候会参考进程定义 ,将定义中某些属性被抄入触发消息头 (MQTM 结构) 中 ,形成触发消息。该触发消息被触发监控器读走并处理 ,监控器可以根据 MQTM 触发消息头中的信息启动相应的进程。

1.2.8 认证信息 (Auth Info)

认证信息 (Authentication Information) 定义了 SSL 认证所需要的证书吊销列表 (CRL ,Certificate Revocation List) 所在的 LDAP 服务器 ,同时定义了连入该 LDAP 服务器所需的用户名和口令。

1.2.9 客户端和服务端 (Client & Server)

WebSphere MQ 分成客户端和服务端 ,只有服务端有对象的概念 ,所以只有服务端的应用程序可以对本地对象进行直接操作。客户端通过 MQI 通道与服务端相连接 ,客户端应用程序发出的所有操作指令都通过该通道传送到服务器 ,在服务器端执行后结果返回客户端。通常情况下 ,客户端的应用程序代码与服务端相同 ,在程序编译时连接的库文件不同。

1.2.10 操作界面 (MQ Interface)

应用程序通过操作界面与 WebSphere MQ 打交道 ,这里的操作界面就是消息队列接口 (Message Queue Interface ,MQI)。MQI 实际上是一套编程接口 ,负责处理应用程序向 WebSphere MQ 提交的各种操作请求 ,应用程序完全不需要关心 WebSphere MQ 的内部结构与具体实现 ,如消息队列、传输队列等。

当应用程序通过 MQI 送出一条消息到远程队列 ,队列管理器会在它的消息数据头中加上路由信息 ,消息被转入传输队列 ,等待送出。MQI 的操作非常简单直观 ,如 MQOPEN、MQCLOSE、MQGET、MQPUT 等。

由于 MQ 的互连通信是通过存储转发机制完成的 ,所以操作与传输是异步的。这意味着应用程序通过操作界面将消息发送出去时 ,消息首先存储在本地 ,当通信畅通时再被转发。应用程序可以继续处理自己的逻辑 ,而不必等待消息传达对方。

1.2.11 应用程序 (MQ Application)

应用程序可以是商业的或用户自行开发的含有对 WebSphere MQ 操作的程序。MQI 提供了支持的有平台的通用编程接口 ,如 VSE/ESA 上的 COBOL ,Tandem Guardian 上的 TAL 和 C ,其他平台上的 C。应用程序只要能够调用相应的库函数 ,它就可以操作 WebSphere MQ。

本节介绍了 WebSphere MQ 中的基本概念和对象 ,其中最核心的部分是消息、队列、队列管理器和通道。对于编程设计人员 ,通常会更关心消息和队列 ,对于维护管理人员 ,通常会更关心队列管理器和通道。下面让我们来看一看这些对象的工作原理。

1.3 工 作 原 理

WebSphere MQ 的工作原理的核心就是存储转发。在单个队列管理器的环境中 ,队列可以用于存储应

用间传递的消息 ,从而使应用程序在各自环节上进行处理 ,并通过队列形成环环相扣的处理流程。在多个队列管理器的环境中 ,消息可以跨平台进行流动 ,从而使整个处理流程在分布式计算环境中完成。

1.3.1 PUT 和 GET

WebSphere MQ 的应用程序可以通过 MQ 界面 (MQI ,MQ Interface) 进行操作。实际上 ,MQI 提供了有限的 API ,其中最本质的两个动作是 PUT 和 GET。PUT 指应用程序将一条消息放入到队列中 ,GET 则相反 ,应用程序将一条消息从队列中取出。WebSphere MQ 通过队列机制来完成消息排队和传递的工作 ,从而使应用程序之间实现松耦合的联系。如图 1 - 6 ,应用程序 A 产生消息 ,通过 PUT 调用放入队列中 ,应用程序 B 将消息取出并进行相应的处理 ,消息的报文格式及内容决定了应用程序 B 处理的具体工作。这样就实现了应用程序 A 到应用程序 B 之间的单向消息传递 ,如果需要双向传递消息则必须再类似地约定反向队列。

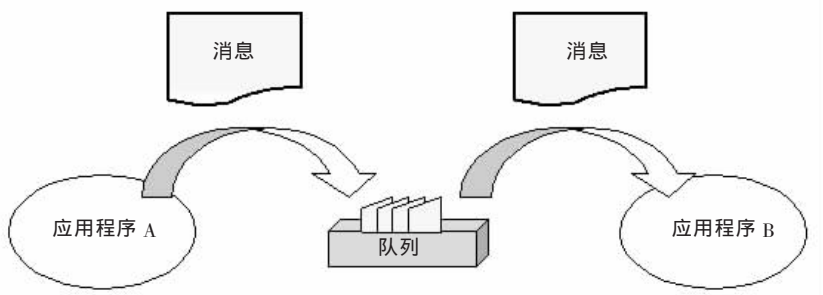


图 1 - 6 应用通过队列传递消息

应用程序设计的时候必须约定双方的报文格式 ,如果用通用格式 (如 XML) 则需考虑由此带来的灵活性和信息冗余 ,在两者之间平衡选择。在运行环境中 ,还需要考虑 PUT 和 GET 的频率与速度 ,以免消息有在队列中堆积起来。

WebSphere MQ 提供的远程队列机制可以将目标队列设定到另外一个队列管理器中 ,这样应用程序 A 和 B 就可以在两台机器上运行而不改动任何代码 ,应用程序 A 仍然做着相同的 PUT 操作 ,将消息放入队列中 ,该消息会自动路由到另一个队列管理器中的队列中 ,应用程序 B 从该队列中 GET 消息 ,与原先一样地处理。也就是说 ,这种配置结构上的改变对应用程序是完全透明的。WebSphere MQ 的这种特性使得其应用的扩展性极佳 ,任何应用在设计之初并不需要考虑太多的性能及扩展性问题 ,在需要时可以很方便地将应用中任何一部分拆到其他的机器上 ,实现多机计算 ,如图 1 - 7。

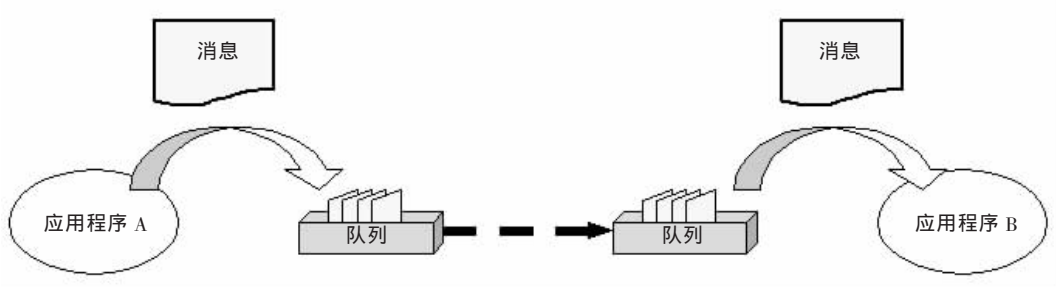


图 1 - 7 应用通过队列跨网络传递消息

1.3.2 协同工作

通常说来 ,一个应用系统会由多个应用模块组成 ,一个处理流程也会由多个处理步骤组成。它们之间可能是串行的关系 ,也可能是并行的关系。在 WebSphere MQ 应用设计中 ,我们可以自然地将多个模块或多个步骤设计成不同的应用程序 ,而它们之间的中间数据则通过消息的方式传递 ,用队列暂存 ,如图 1 - 8。这样一来 ,应用系统会有以下好处 :

- 1. 结构清晰 ,容易并行开发和调试。

2. 扩展性极好 ,能够很容易地部署到跨平台环境中。
3. 灵活性好 ,一旦流程改变了 ,可以较容易地进行修改。

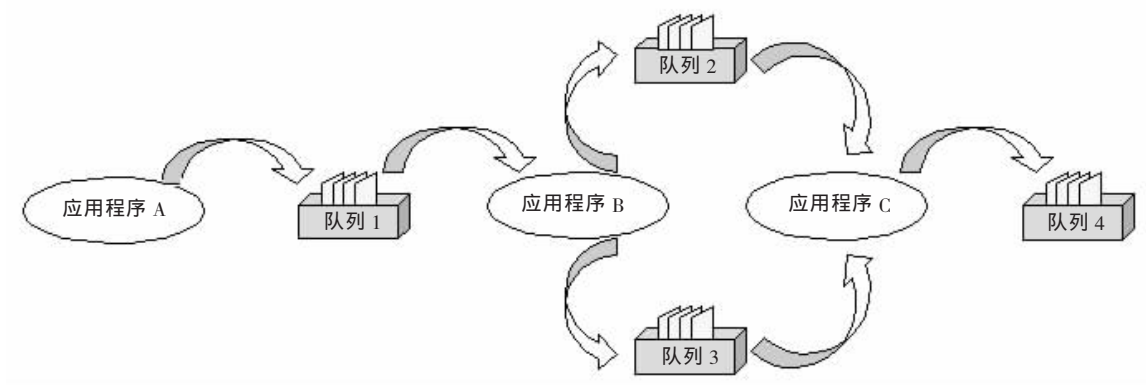


图 1-8 协同工作

这样 ,在应用系统设计的时候只需要考虑它的逻辑架构 ,而无需担心它的物理部署。各个处理环节可以通过 WebSphere MQ 贯穿起来 ,协同工作。

1.3.3 互连通信

1.3.3.1 消息通道 (Message Channel)

WebSphere MQ 跨平台的互连通信是依靠队列管理器之间的消息通道实现的 ,消息通道就是消息传递的管道 ,架设在队列管理器之间 ,消息从一头流入从另一头流出 ,消息的内容和次序完全不变。

配置通道时需要注意通道在两个队列管理器中同名 ,且类型要配对 ,见“概念与对象”章节。WebSphere MQ 中通道中的消息是单向传递的 ,但通道上的协议控制信息可以双向传递。如果需要双向传送消息 ,则需要创建双向的通道定义。

1.3.3.2 消息路由 (Message Routing)

我们首先拿现实生活中寄信做例子来类比 WebSphere MQ 中的一些基本概念 ,从而理解 WebSphere MQ 的工作原理。现实生活中家家户户都可能有一个通信地址 ,对应着一个存放到达信件的信箱。每一封寄出的信件总是先到本地邮局 ,通过邮局之间的信件交换 ,到达对方所在的邮局 ,最后到达对方的信箱里。中间的邮路越复杂 ,时间就越长。这里的邮局相当于 WebSphere MQ 中的队列管理器 ,信箱相当于队列 ,信件相当于消息。我们的每一封信件都有信封和信瓤两部分 ,信封上面往往有收信人通信地址和发信人通信地址 ,信瓤里是真正的内容 (图 1-9)。WebSphere MQ 中的消息也一样 ,它分成消息头和消息体两部分。消息头是消息的属性集合 ,含有目标队列管理器名和目标队列名 ,WebSphere MQ 就是利用这段消息来找到目标队列的。消息体是消息的内容 ,可以是任意的一段内存信息。

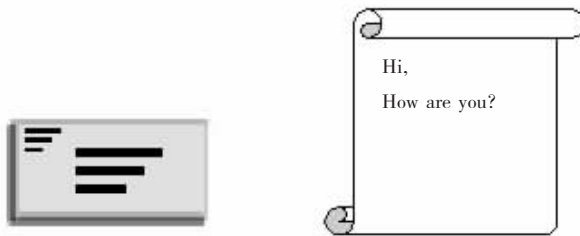


图 1-9 信封和信瓤

WebSphere MQ 依靠每条消息头上所含的路由信息 ,将消息准确地送达目的地。路由信息中的远程队列管理器名指的是远端系统的名字 ,远程队列名指的是远端系统中的目标队列名。一个完整的队列名应

该包含两部分 队列管理器名和队列名 格式为 `queue_name@queue_manager_name`。两部分名字的长度上限都是 48 字节 这两部分名字构成了消息路由的最基本的信息 算法其实很简单：

如果队列管理器名未标明 则缺省加上本地队列管理器的名字。队列名会缺省地匹配本地普通队列，而队列管理器名会缺省地匹配本地队列管理器名或传输队列名。一般说来 建议传输队列名与远程队列管理器同名。消息传送过程如下：

1. 应用程序通过 MQI 发送消息
2. WebSphere MQ 查看 `queue_manager_name` 是否是本地队列管理器
 - a) 如果是 将消息放入本地的名为 `queue_name` 的消息队列中
 - b) 如果不是 将消息放入名为 `queue_manager_name` 的传输队列中
3. MCP 会把传输队列中的消息送达远端
4. 远端的消息队列将消息放入远端系统中名为 `queue_name` 的消息队列中
5. 远端的应用程序从远端的本地队列中取得消息

这种办法提供了简单的路由功能 但有一个明显的缺点 直接使用全名会要求应用程序了解软件的网络结构分布 哪些队列在哪里。这有悖于 WebSphere MQ 对应用程序隐藏网络细节的设计初衷。所以 在跨队列管理器的应用中 通常使用别名队列和远程队列来指定对方队列的名字 从而将队列的分布信息保留在配置中。

1.3.3.3 消息传送

我们在实现消息的跨队列管理器的传送时 通常会在本地队列管理器上配置远程队列和传输队列 在远端的队列管理器上配置本地队列 并通过通道将两者连接起来 如图 1-10。

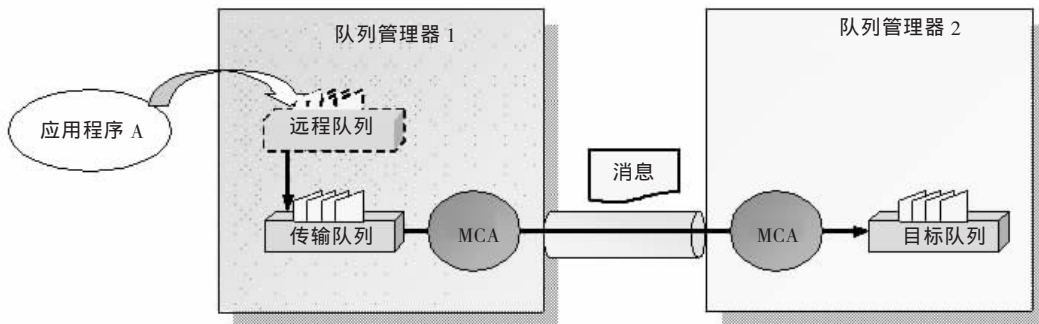


图 1-10 消息的传递过程

这里的远程队列只是一个定义并无队列实体 也就是说远程队列不能存放消息。应用程序一旦将消息通过 MQPUT 送出 则立刻放入传输队列中。传输队列暂时存放待由通道发送的消息 一旦通道连通且条件允许 系统通信程序 MCA 会立即将消息送出。消息到达对方目标队列管理器后 由对方的通信程序 MCA 接收下来并放入相应的目标队列。这里的目标队列就是目标队列管理器上的本地队列。整个过程的效果就好像应用程序直接将消息送入目标队列一样。

远程队列的定义实际上就是指定的目标队列的位置及传输路径。其中 目标队列的位置通过设定目标队列名和目标队列管理器名来确定 消息在路由过程中寻找该目标地址。传输路径就是传输队列名 消息会通过该传输队列送出。不同的远程队列可以共用一个传输队列。

传输队列本质上是一个本地队列 只是由系统通信进程 MCA 监护。应用程序可以人为地通过 MQPUT 放一条消息到传输队列上 但如果该消息没有传输头 (MQXQH) 则不会被发送 消息按以下方式处理：

1. 如果队列管理器设置了缺省死信队列 则消息放入该死信队列 死信消息原因码为 MQFB_XMIT_Q_MSG_ERROR。
2. 如果队列管理器未设置缺省死信队列 则对于非持久性消息 消息被扔掉。对于持久性消息 消息会留在传输队列中无处可去 这时有可能会堵住后继的消息 造成通道无法发送。

消息在经过远程队列放入传输队列时会由队列管理器自动添加一个传输头 (MQXQH) 且传输头中的

内容会根据远程队列定义自动填写。如果说原先的消息是 MQMD + Body ,则放入传输队列的消息为 MQMD + MQXQH + Body。

第 2 章 安 装

我们在前面介绍了 WebSphere MQ 的相关概念及其工作原理,从这一章开始介绍动手操作和管理,让我们从产品安装开始。

WebSphere MQ 的安装过程中有不少选择分支,从而可以安装出不同的效果。事实上,安装中的选择大部分可以在以后的管理和配置中再进行调整。由于初学者对这一过程不熟悉比较容易混淆,从而给以后的操作带来不便。这里以 Windows 平台上的 WebSphere MQ 5.3 为例,详细描述产品安装的全过程。同时也简单介绍了其他 UNIX 平台的安装过程。

2.1 安 装 环 境

在安装前要检查机器的硬件配置及操作系统是否达到相关要求,否则可能引起安装失败。WebSphere MQ 对于机器的环境要求并不高,普通的 PC 机都可以安装。如果有网络通信,则需要有相应的网卡并进行通信配置,比如 IP 地址。对于单机环境,可以用自环方式配置通信,比如配置虚拟 Loopback 网卡。具体环境需要如下:

2.1.1 硬件

- PC 机, Intel 32 位兼容芯片。
- 屏幕至少支持 800 × 600 的分辨率。
- 如果需要通信,应该有网卡等硬件设备,支持 TCP/IP、SNA LU6.2、NETBIOS、SPX 等通信协议。
- 至少 85MB 硬盘空间用于安装, 20MB 空间用于工作, 30MB 系统临时空间用于暂存数据。
- 建议有 128MB 以上内存。

2.1.2 操作系统

- Windows NT
 - Service Pack 6a
 - Microsoft Internet Explorer 4.0.1 +
 - Microsoft HTML Help 1.2 (产品 CD 中含)
 - Microsoft Management Console (MMC) 1.1 (产品 CD 中含)
 - Microsoft Installer (MSI) 2.0 + (产品 CD 中含)
 - Microsoft Active Directory Client Extensions (ADCE) for Windows NT (如果有 ADCE 支持)
 - Java Runtime Environment Version (JRE) 1.3 + (如果有 JAVA 编程支持)
 - Option Pack 4 for Microsoft Windows NT (如果有 Microsoft Transaction Server —MTS 支持)
- Windows 2000 Professional, Server, Advanced Server
 - Service Pack 2 +
 - Microsoft Installer (MSI) 2.0 +
 - Java Runtime Environment Version (JRE) 1.3 + (如果有 JAVA 编程支持)
- Windows XP Professional
 - Java Runtime Environment Version (JRE) 1.3 + (如果有 JAVA 编程支持)

2.1.3 通信协议

对于 TCP/IP、NETBIOS、SPX 操作系统都预置支持。

对于 SNA 协议,则至少需要安装以下软件中的一个来支持。

- IBM Communications Server for Windows NT , Version 5.0 and Version 6.1.1
- Attachmate Extra !Personal Client , Version 6.7
- Attachmate Extra !Enterprise 2000
- Microsoft SNA Server , Version 4.0
- Microsoft Host Integrated Server 2000

2.2 安装介质

2.2.1 正版

联系 IBM 公司,得到 IBM WebSphere MQ 介质,共 2CD。一张 CD 是相关平台的产品,一张 CD 是文档。产品 CD 中含安装所需的全部软件,包括 HTML Help 1.2 ,MMC 1.1 ,MSI 2.0 ,JDK ,ADSI 等。文档 CD 中含各种语言的文档,且有 PDF、HTML、HTMLHelp 三种格式。需要说明的是,简体中文的文档并不完整,如果要深入学习 WebSphere MQ,建议使用英文文档。

从 <http://www-3.ibm.com/software/integration/mqfamily/support/summary/> 下载最新的补丁。

2.2.2 试用版

试用版产品是可以免费下载的,从 <http://www-3.ibm.com/software/integration/mqfamily/> 页面上下载到有效期为 90 天的 WebSphere MQ 5.3 试用版,目前只有 Windows 平台和 Linux 平台上的试用版产品,且皆为英文版。

从 <http://www-3.ibm.com/software/integration/mqfamily/library/manualsa/> 页面下载相关的 pdf 格式的文档。

从 <http://www-3.ibm.com/software/integration/mqfamily/support/summary/> 下载最新的补丁。

2.3 安装过程

在安装介质中的目录中双击 Setup.exe。在一段动画之后(可以用 ESC 键跳过动画),出现安装启动板。按照左边一栏 1、2、3 的步骤顺序来检查系统是否满足安装的先决条件。

- 步骤 1 检查软件安装环境的先决条件。查看右边所需的软件是否全部打钩,打钩表示已经成功安装,打叉表示尚未安装。需要将右边所有项全都打钩,才开始步骤 2。
- 步骤 2 检查网络先决条件。如果不需要特别安装域模式,可以选择“否”。如果需要在安装后自动进行缺省配置,则安装前机器最好安装有网卡,且至少拥有一个 IP 地址。
- 步骤 3 检查安装前状态(图 2-1),确保能通过状态检查、开始安装。

按“启动 WebSphere MQ Installer”→选择“我接受该许可证协议中的条款”→可以选择“定制”安装,并选择程序文件夹安装目录、数据文件夹安装目录、选择日志文件夹安装目录→选择全部部件(图 2-2),开始安装。当问及许可证时,回答“是,已经购买了足够的许可证单元”。最后确认完成。

这里要说明两点:

1. 程序文件夹指的是安装目录,即 WebSphere MQ 产品本身所在的位置。这一部分是不会随着配置的变化而改变的,相对稳定。数据文件夹指的是工作目录,即队列管理器、队列等 WebSphere MQ 对象所在的位置,通常一个对象会对应一个文件,所以,配置的变化会带来目录内容的改变。日志



图 2 - 1 WebSphere MQ 安装启动板

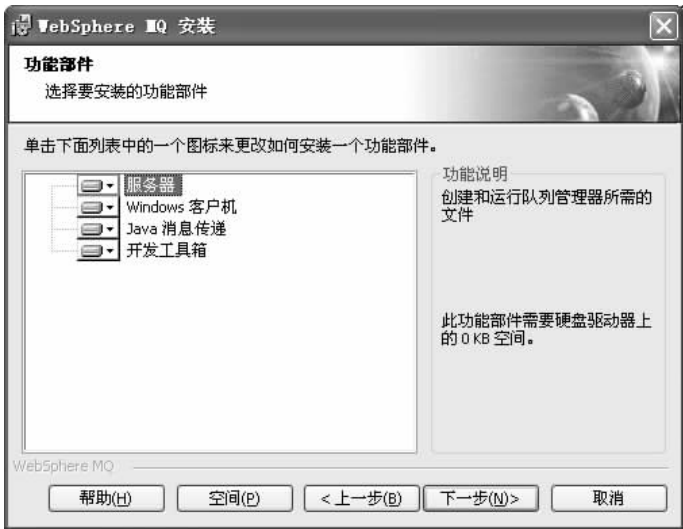



图 2 - 2 选择安装组件

文件夹指的是日志目录 ,WebSphere MQ 的日常动作 ,消息出入队列等会被自动记录在日志中。日志对于保证消息安全起着至关重要的作用 ,在生产环境中 ,为了高可靠性 ,经常将日志安装在另一个硬盘上。具体内容 ,参见日志管理一章。



- 2. WebSphere MQ 5.3 中的许可证单元延续了过去版本中的容量单位 (Capacity Unit) 的概念。基本上容量单位就是机器 CPU 计算能力的一种体现 ,现在的许可证单元就是 CPU 数量。比如一台双 CPU 的 PC Server ,在采购 WebSphere MQ 的时候 ,就应该购买 2 个许可证单元。在安装完 WebSphere MQ 后可以用 setmqcap 命令设置所购买的许可证单元数量。另外 ,dspmqcap 命令可以用来查看 CPU 数量与许可证单元数量是否相符。

```
C :\> dspmqcap
购买的处理器定量为 1
此机器中的处理器数为 1
```

接着安装程序自动进入“准备 WebSphere MQ 向导” ,这时可以先取消 ,等安装完毕后再配置。这时的 WebSphere MQ 实际上已经可以用了 ,如果你是第一次安装 WebSphere MQ 5.3 ,则安装结束。如果以前曾经安装过低版本的 MQ ,或者留有上次创建的队列管理器对象 ,且这些对象需要升级或迁移 ,则需要继续下一步 缺省配置。

一旦安装成功,会在 Windows 2000 或 Windows XP 的屏幕右下角出现图标,这是 WebSphere MQ Task Bar for Windows。同时,在“启动”文件夹中出现“WebSphere MQ 任务栏”。事实上,“WebSphere MQ 任务栏”指向的就是 WebSphere MQ Task Bar 可执行文件,也可以用命令行方式启动:

```
C:\>amqmtbrn.exe -startup。
```

如果将启动组中的“WebSphere MQ 任务栏”删除,则操作系统重启后就不会自动运行 Task Bar 程序,右下角也就不会出现那个图标。这时,如果在命令行中打入 amqmtbrn,则右下角会出现图标,同时用任务管理器可以观察到 amqmtbrn 进程。按此右键点击“启动 WebSphere MQ”,右下角图标由红转绿。同时用任务管理器可以观察到 amqsvc 和 amqmsrvn 进程的出现。按此图标(绿色),右键点击“停止 WebSphere MQ”,经过一段时间,在此期间,右下角图标变成,最后右下角图标由绿转红,同时用任务管理器可以观察到 amqsvc 和 amqmsrvn 进程消失。按此图标(红色),右键点击“隐藏”,发现图标不见了,同时用任务管理器可以观察到 amqmtbrn 进程也消失了。所以,对 Task Bar 程序的操作实际上就是对整个 WebSphere MQ 运行环境的操作,效果上就是启动或停止相关的进程。

至此,产品已经安装完毕。但是,WebSphere MQ 还需要有一个配置的过程,才能真正地使用。缺省的安装过程会紧接着启动缺省配置,而缺省配置又分多个步骤,可以经过相应的配置向导向 MQ 配置成不同的运行模式。我们也可以在这时直接结束安装,将配置工作留待以后实施。

2.4 缺省配置

如果你过去安装过低版本的 MQ 软件,在卸载后留有上次创建的队列管理器对象或者配置文件,在新安装 WebSphere MQ 5.3 以后,你希望将其升级或迁移,从而融入新的软件环境中,而不必重新手工创建且配置一遍。另一种可能是机器迁移,队列管理器数据文件夹是从其他机器上拷贝过来的,通过这一步可以将其放入现在的环境中。

2.4.1 准备 WebSphere MQ 向导

在安装的“IBM WebSphere MQ”程序组中选择“准备 WebSphere MQ 向导”即可出现缺省向导(图 2-3)。或者在命令行窗口中打入 amqmjpse 命令,也有同样的效果。



图 2-3 准备 WebSphere MQ 向导

如果你的 Windows 安装运行于单机模式,而非 Windows 域模式,则在回答“网络中是否有域控制器”时选择“否”。→如果您愿意将现有的队列管理器配置为允许远程管理,则可点击进入“远程管理向导”,并转去 2.4.2。→如果希望简单地设置缺省配置,可点击进入“缺省配置向导”,并转去 2.4.3。→最后确认完成。当然“远程管理向导”和“缺省配置向导”都可以跳过,可以事后再修改设置。

2.4.2 远程管理向导

在“准备 WebSphere MQ 向导”中点击“允许对现有队列管理器进行远程管理”即可进入远程管理向导,如图 2-4。



图 2-4 远程管理向导

在向导中选中允许远程管理的队列管理器→输入侦听器的端口号,缺省为 1414,设置→最后确认完成。

远程管理向导实际上是为队列管理器创建了一个侦听器和命令服务器配置,并且在队列管理器启动的时候将这两个部件一起启动,接受来自远端的网络连接和管理命令。将队列管理器配置成远程管理是为了实现集中式远程控制,便于使用人员的管理。

2.4.3 缺省配置向导

在“准备 WebSphere MQ 向导”中点击“设置缺省配置”或在命令行窗口打入命令 `!amqmgse`,则可进入缺省配置向导,如图 2-5。



图 2-5 缺省配置向导

配置向导会试图创建缺省队列管理器,并且将配置在群集环境中。在向导中选择配置选项,可以保留缺省设置,即允许远程管理且加入缺省群集。→选择队列管理器在群集中的地位。可以选择“是”,将它作为该群集的资源库。→最后确认完成。等待配置完成后,关闭配置向导。

缺省配置向导会试图在机器上创建一个缺省队列管理器,名为 `QM_HostName`,其中 `HostName` 为机器名。然后创建缺省群集,名为 `DEFAULT_CLUSTER`。将缺省队列管理器配置加入该群集且设为该群集的

资源库。

这里有三点需要说明：

1. 缺省配置会创建缺省队列管理器,名为 `QM_HostName`。例如机器名叫 `cxt21`,则创建的缺省队列管理器为 `QM_cxt21`。所谓缺省队列管理器,即在应用时如果不提供队列管理器名称的情况下,系统会自动用默认的队列管理器。可以在 Windows 注册表中找出当前的缺省队列管理器名:`<MQ_HKEY>\Configuration\DefaultQueueManager`。
2. 缺省的群集为 `DEFAULT_CLUSTER`,且缺省配置会将队列管理器配入这个群集环境中。在初次安装时,可能会不明白 WebSphere MQ 中群集的概念,这没关系。对于普通的应用来说,队列管理器是否在群集中并不影响。要深入了解群集,参见“群集”一章。
3. 对于初次安装,您也许根本没有必要进入准备 WebSphere MQ 向导、远程管理向导、缺省配置向导,将环境搞复杂了。在安装结束后直接取消退出即可。这些复杂的配置完全可以留待以后进行。

2.5 安装补丁

复杂的系统软件都难免会有纰漏 (Bug),所以各种商业软件都会有补丁,WebSphere MQ 也不例外,相关补丁可以从 IBM 网站下载。WebSphere MQ 的补丁称为 CSD,后面紧跟的数字是补丁号,比如 CSD06 表示 6 号补丁,后一号的补丁在内容上完全覆盖前一号的补丁。

在安装补丁之前,首先要停止所有的队列管理器及 WebSphere MQ 相关进程,在 Windows 任务管理器中看不见所有 `amq` 打头的进程。然后,将从网站下载的 WebSphere MQ 5.3 补丁展开,选择展开目录,这时安装程序开始自动安装补丁,当然也可以在展开目录中运行 `amqicsdn.exe`,手工开始安装。安装过程十分简单,选择备份文件夹,开始安装→完成。

在安装完补丁后,在 WebSphere MQ 安装目录下会出现补丁目录,例如 CSD06,它实际上是备份文件夹,内容是补丁安装前的相应文件。用 `mqver` 命令,可以显示目前的版本号和补丁号。

```
C:\>mqver
Name :      WebSphere MQ
Version :    530.6 CSD06
CMVC level : p530-06-L040211
BuildType : IKAP - (Production)
```

安装补丁与配置没有必然的先后关系,这里因为 Windows 平台的“准备 WebSphere MQ 向导”在产品安装之后会自动启动,为了显得连贯,在内容上将缺省配置安排在前。事实上,通常建议补丁安装紧接着产品安装,然后再配置,在 UNIX 环境下尤其如此。

2.6 其他平台

WebSphere MQ 不仅在 Windows 平台上有众多的应用,在 UNIX 平台中的应用也十分普遍。由于各种 UNIX 平台都大同小异,软件的安装过程也比较类似。所以在这一节中,读者应该注意安装环境的要求在各个平台上的差异及安装过程中的一些差别。

WebSphere MQ 在 UNIX 或 Linux 上的安装过程中会自动创建 `mqm` 用户和 `mqm` 组,且 `mqm` 用户属于 `mqm` 组。当然,这一点也可以在安装之前用手工完成。以后所有的 WebSphere MQ 管理命令,如 `crtmqm`,`dlmqm`,`strmqm`,`endmqm`,`dspmq` 等,都只有 `mqm` 组中的用户才能使用。

2.6.1 AIX

AIX 是 IBM RS/6000 机器采用的操作系统,目前正在使用的主要有 4.3.3 和 5L 两个版本。该操作系统上的软件大多可以通过系统工具 `smit` 来进行安装和管理。

2.6.1.1 操作系统

WebSphere MQ 可以安装在 AIX 4.3.3 或 5L 操作系统上。具体要求如表 2 – 1。

表 2 – 1 对 AIX 操作系统的要求

操作系统版本	补丁	说明
AIX 4.3.3	U472177 , Y2K fixes	32 位
AIX 5.1	U477366 , U477367 , U477368	32 位或 64 位
AIX 5.2		32 位或 64 位

2.6.1.2 安装过程

1. 插入产品 CD ,作为 root 用户登录。
2. 用 smit 命令安装：

Software Installation and Maintenance

Install and Update Software

Install and Update from LATEST Available Software
3. 选择安装介质 /dev/cd0 (CD-ROM Drive) ,在 SOFTWARE to install 栏中选择需要安装的部件。可以安装的部件有：
 - Runtime
 - Base Kit
 - Server
 - Client for AIX
 - Sample programs
 - DCE support
 - DCE samples
 - Java messaging
 - Message catalogs
 - Man pages
 - IBM Global Security Kit V6
 - IBM Key Management tool (iKeyman)
4. Include corresponding LANGUAGE filesets ?档中选择 YES。
5. 如果是 AIX 4.3.3 ,选择 OK 开始安装。如果是 AIX 5.1 或 5.2 ,对 Preview new LICENSE agreements ?可以选择 No。对 ACCEPT new license agreements ?必须选择 YES。
6. 安装完毕后 ,用 setmqcap 命令指定许可证单元数量 ,如下所示。注意 ,如果 WebSphere MQ 在运行期间发现 CPU 数量与许可证单元数量不符 ,会不断地报错。

```
setmqcap 4
```

2.6.2 HP – UX

HP – UX 是 HP 公司的小型机采用的操作系统 ,目前主要有 11 和 11i 两个版本。WebSphere MQ 可以通过介质中的安装脚本进行安装。注意 ,在安装之前需要调整一些系统参数。

2.6.2.1 操作系统

WebSphere MQ for HP – UX 支持 HP – UX 11 和 11i 两个版本的操作系统 ,如表 2 – 2。

表 2-2 对 HP-UX 操作系统的要求

操作系统版本	补丁	说明
HP-UX 11.0	PHSS_24627, PHSS_22543	32 位
HP-UX 11i (11.11)		32 位

2.6.2.2 安装过程

1. 缺省情况下,HP-UX 操作系统中通信参数对 WebSphere MQ 来说都偏小,需要调整。以 root 身份登录,更改系统参数。表 2-3 为最小推荐值。考虑到不同队列管理器的独立性,shmmni,semmni,semmns,semmnu 的值会与队列管理器数量有关。如果使用环形日志,msgmap 和 msgmax 参数是不必要的。

表 2-3 对 HP-UX 操作系统参数要求

系统参数	最小推荐值
shmmax	536870912
shmseg	1024
shmmni	1024
shmem	1
sema	1
semaem	16384
semvmx	32767
semmns	16384
semmni	1024 (semmni < semmns)
semmap	1026 (semmni + 2)
semmnu	2048
semume	256
msgmni	50
msgtql	256
msgmap	258 (msgtql + 2)
msgmax	4096
msgmnb	4096
msgssz	8
msgseg	1024
maxusers	32
max_thread_proc	66
maxfiles	1024
nfile	10000

2. 以 root 身份登录,插入产品 CD。
3. Mount CDRom

```
cd /usr/sbin
pfs_mountd &
pfsd 4 &
pfs_mount -o xlat=unix /<path to CD-ROM device> /<localdir>
```

4. 接受安装许可

```
cd <mount point> 例 cd /cdrom
./mqlicense.sh
```

- text_only 表示文本方式显示 ,例如 mqlicense.sh -text_only。
- accept 表示缺省接受

5. 用 swinstall 工具安装

```
swinstall -s /cdrom<localdir> /hpux11/<drivername>.v11
```

选择 MQSeries ,在 Action 菜单中选择 Open item ,选择所有需要安装的部件 ,可选择的部件有 :

- Runtime
- Base
- Server
- Client
- Sample programs
- DCE support
- DCE samples
- Java messaging
- Message catalogs
- Man pages
- IBM Global Security Kit V6
- IBM Key Management tool (iKeyman)

6. 在 Action 菜单中选择 Mark for install 到上一层菜单中 ,在 Action 菜单中选择 Install (analysis) ,选 OK 开始安装。
7. 安装完毕后 ,用 setmqcap 命令指定许可证单元数量。注意 ,如果 WebSphere MQ 在运行期间发现 CPU 数量与许可证单元数量不符 ,会不断地报错。Purchased license units not set (use setmqcap)。
- setmqcap 命令用法举例 :

```
setmqcap 4
```

2.6.3 Solaris

Solaris 是 SUN 公司的小型机采用的操作系统 ,WebSphere MQ 可以通过介质中的安装脚本进行安装。在安装之前要注意操作系统的补丁及系统参数。

2.6.3.1 操作系统

WebSphere MQ for Solaris 支持 7 和 8 两个版本的操作系统 ,如表 2-4。

表 2-4 对 Solaris 操作系统的要求

操作系统版本	补丁	说明
Sun Solaris 7	107171-02	32 位
	107544-03	
	106950-16	
	106327-11	

续表

操作系统版本	补丁	说明
	106300 – 10	
	106541 – 18	
	106980 – 17	
Sun Solaris 8	● 基础补丁	32 位
	同上	
	● 附加补丁	
	108827 – 12	
	111177 – 06	
	● SSL 补丁	
	108434 – 02	
	111327 – 02	
	108991	
	108528	

2.6.3.2 安装过程

1. 以 root 身份登录 ,查看系统参数。

```
sysdef -i
```

2. 更改系统参数 ,修改 /etc/system 文件。下表为推荐值：

```
set shmsys :shminfo_shmmax = 4294967295
set shmsys :shminfo_shmseg = 1024
set shmmin :shminfo_shmmin = 1
set shmsys :shminfo_shmmni = 1024
set semsys :seminfo_semmni = 1024
set semsys :seminfo_semaem = 16384
set semsys :seminfo_sevmx = 32767
set semsys :seminfo_semmap = 1026
set semsys :seminfo_semmns = 16384
set semsys :seminfo_semmnl = 100
set semsys :seminfo_semopm = 100
set semsys :seminfo_semmnu = 2048
set semsys :seminfo_semume = 256
set msgsys :msginfo_msgmni = 50
set msgsys :msginfo_msgmap = 1026
set msgsys :msginfo_msgmax = 4096
set msgsys :msginfo_msgmnb = 4096
```

不要修改 shmmin 值。考虑到不同队列管理器的独立性 ,shmmni ,semmni ,semmns ,semmnu 的值会与队列管理器数量有关。

如果不使用环形日志 ,msgmap 和 msgmax 参数是不必要的。

3. 以 root 身份登录 ,插入产品 CD。
4. Mount CDROM ,不妨假设 mount point 为 /cdrom
5. 接受安装许可

```
/cdrom/mqlicense.sh
```

-text _only 表示文本方式显示 ,例如 :mqlicense.sh -text _only。

6. 用 pkgadd 工具安装

```
pkgadd -d /cdrom
```

选择所有需要安装的部件 ,可选择的部件有 :

- Server
- Client
- Sample programs
- DCE support
- DCE samples
- Java messaging
- Message catalogs
- Man pages
- IBM Global Security Kit V6
- IBM Key Management tool (iKeyman)

7. 安装完毕后 ,用 setmqcap 命令指定许可证单元数量。注意 ,如果 WebSphere MQ 在运行期间发现 CPU 数量与许可证单元数量不符 ,会不断地报错。Purchased license units not set (use setmqcap)。

setmqcap 命令用法举例 :

```
setmqcap 4
```

2.6.4 Linux

Linux 是目前流行的操作系统 ,由于它开放源代码和全球性组织开发的特点 ,Linux 成为发展最快的操作系统。WebSphere MQ 在 Linux 上的安装相对简单。

2.6.4.1 操作系统

WebSphere MQ for Linux 可以安装于两类 Linux 之上 :Linux for Intel 和 Linux for zSeries。

- Linux for Intel
 - Red Hat Linux V7.2
 - Caldera OpenLinux V3.1
 - SuSE Linux Enterprise Server V7
 - Turbolinux V7.0
- Linux for zSeries
 - Red Hat Linux for S/390(R)
 - SuSE Linux Enterprise Server V7

2.6.4.2 安装过程

1. 以 root 身份登录 ,查看系统参数。按照需要调整相当的参数值 ,参数文件为 :

```
/proc/sys/kernel/shmmax  
/proc/sys/kernel/shmmni  
/proc/sys/kernel/shmall  
/proc/sys/kernel/sem
```

相应地 ,调整最大打开文件数和最大进程数。

2. 以 root 身份登录 ,插入产品 CD。

3. Mount CDROM ,不妨假设 mount point 为 /cdrom

4. 接受安装许可

```
/cdrom/mqlicense.sh
-text _only 表示文本方式显示 ,例如 :mqlicense.sh -text _only。
```

5. 用 rpm 工具安装

a) 如果是 Linux for Intel ,顺序执行

```
rpm -i MQSeriesRuntime-5.3.0-1.i386.rpm
rpm -i MQSeriesSDK-5.3.0-1.i386.rpm
rpm -i MQSeriesServer-5.3.0-1.i386.rpm
```

b) 如果是 Linux for zSeries ,顺序执行

```
rpm -i MQSeriesRuntime-5.3.0-1.s390.rpm
rpm -i MQSeriesSDK-5.3.0-1.s390.rpm
rpm -i MQSeriesServer-5.3.0-1.s390.rpm
```

6. 安装完毕后 ,用 setmqcap 命令指定许可证单元数量。注意 ,如果 WebSphere MQ 在运行期间发现 CPU 数量与许可证单元数量不符 ,会不断地报错。Purchased license units not set (use setmqcap)。
setmqcap 命令用法举例：

```
setmqcap 4
```

2.7 安 装 目 录

安装目录总共有 4 个 :产品目录 ,数据目录 ,日志目录和出错目录。产品目录是 WebSphere MQ 产品本身 ,一旦安装完毕后 ,不会因为运行而改变。数据目录与运行环境相关 ,通常一个对象对应一个文件或目录 ,对象属性或内容的改变会反映到文件或目录的改变。日志目录中存放的是 WebSphere MQ 运行日志文件 ,这些文件的组织会根据日志类型（环形日志或线性日志）而不同 ,其间记录了 MQ 在运行中的各种操作 ,在 MQ 出现异常时可用于恢复。出错目录用于存放错误日志信息 ,通常随着时间的推移 ,错误文件会越来越多 ,需要定期清理。

对于 Windows 平台 ,可以在安装时将这些目录指定到不同的硬盘上。对于 UNIX 平台 ,安装后生成的目录如下。也可以事先将这些目录指定到不同的文件系统 ,而这些文件系统可以创建在不同的硬盘上。将数据和日志分开可以提高性能 ,如果是线型日志 ,还可以做介质恢复 ,保证数据安全。

- 安装目录为 /opt/mqm
- 数据目录为 /var/mqm
- 日志目录为 /var/mqm/log
- 出错目录为 /var/mqm/errors

WebSphere MQ 的管理命令都以可执行文件的方式存放在 bin 目录下 ,编程的头文件放在 include 或 inc 目录下 ,库文件放在 lib 目录下。对于每个队列管理器 ,会以队列管理器名在 Qmgrs 和 log 目录下各生成一个子目录 ,用以存放所有该队列管理器中的对象和日志 ,其中 qmgrs 下的子目录是队列管理器的工作目录 ,其下还会生成一个 error 子目录 ,用于记录错误日志 ,WebSphere MQ 最基本的问题诊断方法就是查看这些错误日志。

下面以 Windows 和 AIX 平台为例 ,查看缺省情况下的安装目录。注意 ,两者在目录安排和目录取名上有所不同。

2.7.1 Windows

C:\Program Files\IBM\WebSphere MQ	产品目录
└log	日志目录
└bin		
└Errors		

└Qmgrs	数据目录
└Config	
└Java	
└Tools	
└c	
└Samples	
└include	
└Lib	
└Java	

2.7.2 AIX

/usr/lpp/mqm - > /usr/mqm	
/usr/mqm	产品目录
└samp	
└inc	
└java	
└lib	
└bin	
└ssl	
└tivoli	
/var/mqm	数据目录
└log	日志目录
└qmgrs	

2.8 安 装 文 档

WebSphere MQ 各种平台的详细安装文档可以参见 IBM WebSphere MQ 文档。

- Windows Quick Beginnings
- AIX Quick Beginnings
- HP – UX Quick Beginnings
- Solaris Quick Beginnings
- Linux Quick Beginnings
- iSeries Quick Beginnings
- z/OS Concepts and Planning Guide

第 3 章 控制与管理

在成功安装了 WebSphere MQ 产品之后,我们可以进行使用和配置,而这些都是通过控制与管理命令完成的。在这一章中我们会详细介绍各种控制命令的使用方法及管理方式。

WebSphere MQ 中的控制针对的是 MQ 部件,通常使用命令方式完成。管理针对的是 WebSphere MQ 对象,可以用 MQSC 脚本命令或图形界面工具完成。在对 WebSphere MQ 的维护中两者需要结合使用。对于 WebSphere MQ for UNIX,控制与管理都只能通过命令行界面完成。对于 WebSphere MQ for Windows,除了命令行方式之外,系统还提供了“WebSphere MQ 资源管理器”和“WebSphere MQ 服务”两个图形界面管理工具,大多数控制与管理工工作都可以通过点击右键选择对应功能来完成。

对于初次使用的用户可以按以下的步骤来体验一下 WebSphere MQ 的操作。

1. 创建队列管理器,我们不妨假定队列管理器名为 QM

```
C:\> crtmqm QM
```

2. 启动队列管理器

```
C:\> strmqm QM
```

3. 创建队列,假定队列名为 Q。

首先,用命令行交互界面管理工具 RUNMQSC 连接队列管理器:

```
C:\> runmqsc QM
```

```
5724 - B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
```

```
启动队列管理器 QM 的 MQSC。
```

接着,在交互管理工具中使用脚本命令 `define qlocal` 创建出本地队列对象,最后用 `end` 命令退出交互管理工具。

```
define qlocal (Q)
end
```

4. 将消息放入队列

用 `amqspout` 命令将消息逐行放入队列,每行输入代表一条消息,输入空行结束。

```
C:\> amqspout Q QM
```

5. 从队列中取出消息

用 `amqsget` 命令可以将放入的消息全部取出。

```
C:\> amqsget Q QM
```

通过以上的操作,我们可以大致了解 WebSphere MQ 的管理、配置和使用的过程。其中第 1、2 两步是通过控制命令完成的,第 3 步是对队列管理器的配置,通过配置工具完成,第 4、5 两步实际上是运行了 WebSphere MQ 的应用程序。

如果目前对控制命令尚不熟悉也没关系,下面会做详细介绍。在阅读时也可以参照附录“WebSphere MQ 命令一览表”,其中对所有的控制命令语法有详细的描述。

3.1 MQ 控制命令

3.1.1 MQ 队列管理器控制

队列管理器是构建 WebSphere MQ 运行环境的基础,用户需要首先创建并启动队列管理器才能进行以后的操作。队列管理器的控制命令可以创建、删除、启动、停止队列管理器,也可以显示系统中所有的队列管理器及其当前运行状态。

3.1.1.1 创建队列管理器

- 格式

`crtmqm [选项] QMgrName`

- 功能

创建队列管理器

- 说明

`QMgrName` 指的是待建的队列管理器名。`crtmqm ?`显示命令语法,可以列出所有选项,更详细的语法说明可以参见附录“WebSphere MQ 命令一览表”。常用的选项有 `[-q]` `[-d DefXmitQ]` `[-u DeadQ]`等,它们可以组合使用。选项如果取 `-q` ,表示创建缺省队列管理器,一台机器最多只能有一个缺省队列管理器。如果取 `-d` ,表示指明队列管理器的缺省传输队列。如果取 `-u` ,表示指明队列管理器的死信队列。

创建时的选项指定了队列管理器的属性,这些属性有些可以在创建后修改,有些则不可以。未用选项指明的属性就使用缺省值,创建命令可以不带选项(如 `crtmqm QM`),则所有属性皆使用缺省值。

- 举例

```
crtmqm -q QM
```

3.1.1.2 删除队列管理器

- 格式

`dlmqm [-z] QMgrName`

- 功能

删除队列管理器

- 说明

该命令执行的前提是队列管理器的相关进程已经全部停止了。该命令只有一个选项 `-z` ,表示抑制命令执行时发出的信息。

- 举例

```
dlmqm -z QM
```

3.1.1.3 启动队列管理器

- 格式

`strmqm [-z | -c] QMgrName`

- 功能

`strmqm` 用来启动队列管理器,也可以用来用缺省对象重建队列管理器。

- 说明

如果在命令中没有选项,则简单地启动队列管理器。如果用 `-z` ,表示抑制命令执行时发出的信息。`-c` 选项比较特殊,表示重置队列管理器。命令的执行过程为:先启动队列管理器,再覆盖重建所有的缺省系统对象,最后自动停止该队列管理器。

如果上一次队列管理器未能正常停止,则启动后可能会回滚一些未完成的交易,恢复一些消息和对象。如果因为某些异常不能重启队列管理器,可以在出错日志(`errors` 目录)中寻找原因。

- 举例

```
strmqm QM
```

3.1.1.4 停止队列管理器

- 格式

endmqm [-z] [-c | -w | -i | -p] QMgrName

● 功能

endmqm 用来停止队列管理器。

● 说明

通过不同的选项 ,可以设置不同的停止方式。比如 ,选项取 -c ,表示受控 (Controlled) 方式停止 ,即等到连接在该队列管理器上的所有应用全部主动断开连接后 ,才停止队列管理器。不过 ,该命令是立即返回的 ,显示命令已经提交。如果选项取 -w ,则同样是受控方式停止 ,只是命令不是立即返回的 ,在指定的时限 (秒) 内等待 (Wait) 命令执行完毕后或超时返回。若选项取 -i ,即立即 (Immediate) 停止 ,使其他连接在该队列管理器上的应用的所有后继 MQ API 全部失败 ,以促使它们退出。命令在队列管理器停止后返回。若选项取 -p ,即强行 (Preemptive) 停止。不会等待其他应用程序释放资源或断开连接 ,而直接将队列管理器进程退出 ,有可能会造成异常。 -z 选项表示抑制命令执行时发出的信息 ,可以与其他选项组合使用。

● 举例

```
endmqm QM
```

3.1.1.5 显示队列管理器

● 格式

dspmqr [-m QMgrName]

● 功能

dspmqr 用来显示本地的队列管理器的运行状态。

● 说明

如果用 -m 选项 ,表示显示某个具体的队列管理器运行状态 ,否则 ,表示显示所有的队列管理器状态。

● 举例

```
dspmqr
QMNAME(QM)                STATUS( 正在运行 )
QMNAME(QM1)                STATUS( 正常结束 )
QMNAME(QM2)                STATUS( 正常结束 )
```

3.1.2 MQ 命令服务器控制

WebSphere MQ 命令服务器是队列管理器的一个组件 ,用来对外来的命令消息进行解释和执行。在远程管理和编程管理的应用中 ,需要启动命令服务器。一个队列管理器最多只有一个命令服务器 ,缺省情况下在创建队列管理器时由系统一并创建。

3.1.2.1 启动命令服务器

● 格式

strmqcsv [QMgrName]

● 功能

strmqcsv 用来启动命令服务器。

● 说明

QMgrName 指的是命令服务器所在的队列管理器 ,缺省为系统缺省队列管理器。命令服务器是队列管理器的附加部件 ,用来执行管理命令。

● 举例

```
strmqm QM
```

3.1.2.2 停止命令服务器

- 格式

```
endmqcsv [ -c | -i ] QMgrName
```

- 功能

endmqcsv 用来停止命令服务器

- 说明

通过不同的选项 ,可以选择不同的停止方式。 -c 表示受控方式 (Controlled) 停止 ,允许已经开始的命令执行完成 ,然后停止该命令服务器。 -i 表示立即 (Immediate) 停止 ,中止正在执行的命令并立即停止命令服务器。

- 举例

```
endmqm -c QM
```

3.1.2.3 显示命令服务器

- 格式

```
dspmqcsv [ QMgrName ]
```

- 功能

dspmqcsv 用来显示命令服务器的状态

- 说明

这里 QMgrName 指的是命令服务器所在的队列管理器 ,缺省为系统缺省队列管理器。

- 举例

```
dspmqcsv QM
```

3.1.3 MQ 监听器控制

WebSphere MQ 中监听器也是队列管理器的一个组件 ,用来监听外来的连接请求并相应地做出反应。监听器通常需要先配置 ,然后才能运行 ,配置参数与监听器选择的通信协议有关。当然 ,也可以在第一次启动监听器时将配置参数传入 ,隐式地进行配置。一个队列管理器可以有多个监听器 ,分别应用于不同的通信协议或同一协议的不同参数。比如 TCP/IP 的不同端口。

3.1.3.1 启动监听器

- 格式

```
runmqlsr [ -m QMgrName ] -t (TCP | LU62 | NETBIOS | SPX) [ 参数 ]
```

- 功能

runmqlsr 用来启动监听器

- 说明

监听器是通道连接的被动方用来监听网络连接的程序。命令格式中 QMgrName 指的是命令服务器所在的队列管理器 ,缺省为系统缺省队列管理器。 -t 选项指定通信协议 ,参数与监听器选择的通信协议有关 ,具体参见“附录 WebSphere MQ 命令一览表”。

- 举例

```
runmqlsr runmqlsr -t tcp -p 1414 -m QM
```

3.1.3.2 停止监听器

- 格式

```
endmqlsr [ -w ] [ -m QMgrName ]
```

- 功能

endmqlsr 用来停止监听器 ,实质上就是停止网络监听程序。

- 说明

QMGrName 指的是命令服务器所在的队列管理器 ,缺省为系统缺省队列管理器。选项为 -w 时表示等待方式 (Wait) 停止 ,命令执行会等到监听器停止完成后才返回 ,否则立即返回。

- 举例

```
endmqlsr -m QM
```

3.1.3.3 配置 TCP/IP 监听器

TCP/IP 协议是目前使用最广泛的通信协议。WebSphere MQ 队列管理器的 TCP/IP 监听器可以配置成 3 种形式。用带参数的 runmqslsr 进程进行监听 ;UNIX 环境下可以用操作系统 TCP/IP Daemon 进程 inetd 进行监听 ;Windows 环境下可以配置成 WebSphere MQ 服务的形式。一个队列管理器可以有多个监听器 ,分别监听多个端口。

- 配置 runmqslsr

这是配置 MQ 监听器的标准方式 ,格式为 runmqslsr -t tcp [-m <QMGr>] [-p <Port>]。如果缺省 -m 选项 ,表示使用缺省队列管理器 ,如果缺省 -p 选项 ,表示使用缺省端口 1414。

在 Windows 下可以用 `start runmqslsr -t tcp [-m <QMGr>] [-p <Port>]` 将命令放在另一个窗口中执行。

在 UNIX 下则可以用 `runmqslsr -t tcp [-m <QMGr>] [-p <Port>] &` 将命令放在后台执行。

- 配置 inetd

在 UNIX 平台中 ,可以通过配置 inetd 使 MQ 监听器植入系统监听服务中 ,相当于每次开机监听器就自动启动了。具体配置分 3 步 :

1. 在 /etc/services 中添加 :

```
MQSeries1 1414/tcp
MQSeries2 1415/tcp
```

其中 ,MQSeries1 和 MQSeries2 分别是为监听端口起的别名 ,可以是任意字串。

2. 在 inetd.conf 中添加 :

```
MQSeries1 stream tcp nowait mqm /usr/lpp/mqm/bin/amqcrsta amqcrsta [ -m QMGr1 ]
MQSeries2 stream tcp nowait mqm /usr/lpp/mqm/bin/amqcrsta amqcrsta [ -m QMGr2 ]
```

其中 ,MQSeries1 和 MQSeries2 需要与 /etc/services 文件中的定义配合起来。

3. 让修改的配置生效 :

```
refresh -s inetd //AIX
kill -1 <inetd daemon 进程的 pid> //其他 UNIX
```

- 配置 WebSphere MQ 服务

在 WebSphere MQ for Windows 产品中 ,可以通过“WebSphere MQ 服务”工具来进行配置。在配置界面中右键点击启动了的管理队列管理器 ,新建→侦听器 ,设置通信协议和相关参数后 ,会在注册表中配置相应的条目 ,如下。服务可以配置成“自动启动”方式或“手动启动”方式 ,自动启动方式即在队列管理器启动的同时自动启动侦听器程序。在执行启动失败后 ,也可以指定恢复的方式和参数。

<MQ_HKEY>\Configuration\Services\<QMName>\侦听器

在“WebSphere MQ 服务”配置工具中也可以配置其他的服 务 ,如队列管理器、命令服务器、通道启动程序、触发器监控器、通道启动等。其中的启动和恢复的方式和参数都与此相似 ,不再赘述。

3.1.4 MQ 触发监控器控制

触发监控器 (Trigger Monitor) 是 WebSphere MQ 的组件之一 ,用于监控消息触发初始化队列并启动消息处理程序。触发监控器根据其运行的位置分为 MQ Client 端触发监控器和 MQ Server 端触发监控器 ,分

别用于启动各自系统中的消息处理进程。

一个队列管理器可以有多个触发监控器,分别监控不同的初始化队列。程序员也可以根据规范自己编写触发监控器。

3.1.4.1 启动 Client 端触发监控器

- 格式

```
runmqtmc [ -m QMgrName ] [ -q InitiationQName ]
```

- 功能

runmqtmc 用来启动 Client 端触发监控器

- 说明

QMgrName 指的是命令服务器所在的队列管理器,缺省为系统缺省队列管理器。InitiationQName 指的是触发监控器所监控的初始化队列,缺省为 SYSTEM.DEFAULT.INITIATION.QUEUE。

runmqtmc 运行于 Client 端,一旦触发条件满足即会有触发消息自动放入初始化队列中并被触发监控器读走。触发监控器根据触发消息的内容启动相关触发进程,该进程也运行于 Client 端。

- 举例

```
runmqtmc -m QM -q SYSTEM.DEFAULT.INITIATION.QUEUE
```

3.1.4.2 启动 Server 端触发监控器

- 格式

```
runmqtrm [ -m QMgrName ] [ -q InitiationQName ]
```

- 功能

runmqtrm 用来启动 Server 端触发监控器。

- 说明

与 runmqtmc 相同,QMgrName 指的是命令服务器所在的队列管理器,缺省为系统缺省队列管理器。InitiationQName 指的是触发监控器所监控的初始化队列,缺省为 SYSTEM.DEFAULT.INITIATION.QUEUE。

runmqtrm 运行于 Server 端,一旦触发条件满足即会有触发消息自动放入初始化队列中并被触发监控器读走。触发监控器根据触发消息的内容启动相关触发进程,该进程也运行于 Server 端。

Client 端与 Server 端的触发监控器原理是相同的,只是运行的位置不同,触发后在各自的运行平台上启动应用程序。

- 举例

```
runmqtrm -m QM -q SYSTEM.DEFAULT.INITIATION.QUEUE
```

3.1.4.3 停止触发监控器

WebSphere MQ 提供的 Client 端或 Server 端触发监控程序都无法“优雅地停止”(Gracefully Stop)。所以要停止这些缺省的触发监控程序,可以将触发监控器进程找出来并用操作系统命令将其停止,比如 kill。对于程序员自行开发的触发监控器,要注意留有停止接口。

3.1.4.4 配置触发监控器服务

在 WebSphere MQ for Windows 产品中,可以通过“WebSphere MQ 服务”工具来进行配置。在配置界面中右键点击启动了的管理队列管理器,新建→触发器监控器→指定队列名,会在注册表中配置相应的条目:

```
<MQ_HKEY>\Configuration\Services\<QMName>\触发器监控器
```

3.1.5 小结

这一节列举了队列管理器及其部件的控制命令 ,如表 3 - 1。我们通常可以使用这些命令对 MQ 的各个部件实现启动、停止、显示等功能 ,所有的控制命令都可以在操作界面下以命令行方式实现。

表 3 - 1 WebSphere MQ 控制命令

命令	格式	功能说明
队列管理器控制命令		
crtmqm	crtmqm [选项] QMgrName	创建队列管理器
dltmqm	dltmqm [选项] QMgrName	删除队列管理器
strmqm	strmqm [选项] QMgrName	启动队列管理器
endmqm	endmqm [选项] QMgrName	停止队列管理器
dspmq	dspmq [- m QMgrName]	显示队列管理器
命令服务器控制命令		
strmqcsv	strmqcsv [QMgrName]	启动命令服务器
endmqcsv	endmqcsv [选项] QMgrName	停止命令服务器
dspmqcsv	dspmqcsv [QMgrName]	显示命令服务器
监听器控制命令		
runmqlsr	runmqlsr [选项] [- m QMgrName]	启动监听器
endmqlsr	endmqlsr [选项] [- m QMgrName]	停止监听器
触发监控器控制命令		
runmqtmc	runmqtmc [- m QMgrName] [- q InitQName]	启动客户端触发监控器
runmqtrm	runmqtrm [- m QMgrName] [- q InitQName]	启动服务端触发监控器

3.2 MQ 对象管理

WebSphere MQ 为了兼顾各种平台的兼容性 ,提供了一个通用的字符交互界面管理工具 RUNMQSC ,它的用法在所有平台上都一样 ,功能略有差别。RUNMQSC 运行的命令集称为 MQSC (MQ Script Command) ,它是一种交互式的脚本命令 ,在 RUNMQSC 环境下解释执行。从标准输入中读取一条 MQSC 命令 (可能占多行) ,解释执行并通过标准输出打印结果 ,然后再读取下一条 MQSC 命令 ,如此循环。通过将标准输入输出重定向到文件 ,可以将命令存放在脚本文件中 ,将输出存放在结果文件中 ,脚本文件习惯上以 .tst 为后缀。

用户可以在命令行方式下运行 RUNMQSC ,进入交互界面。RUNMQSC 的运行格式为 :runmqsc [QMgrName]。其中 QMgrName 为连接的队列管理器名 ,缺省为系统缺省的队列管理器。一旦连接上队列管理器 ,就可以通过 MQSC 脚本命令创建、删除、显示队列管理器上的 MQ 对象了 ,而大多数对象及其属性都是一旦设置 ,立即生效的。所以 ,RUNMQSC 非常适合于动态配置。

在介绍对象管理的 MQSC 脚本命令之前 ,让我们来了解一下 RUNMQSC 的使用方法 :

- 1. 在命令行中输入 runmqsc < QMgrName > ,其中 QMgrName 是队列管理器名 ,不妨假定为 QM。

```
C :\> runmqsc QM
```

- 2. 在交互界面中输入“ ?” ,即可显示 MQSC 的所有一级命令

```
?
```

- 3. 在交互界面中输入一级命令 ,后面紧跟“ ?” ,则可显示相关的二级命令

DISPLAY ?

4. 在交互界面中输入二级命令 ,后面紧跟“ ?” ,则可显示相关的语法

DISPLAY QLOCAL ?

在一级命令中的 DEFINE 用于创建 WebSphere MQ 对象 ,DELETE 用于删除对象。每一个对象在创建后都会有各自的属性 ,这些属性大多数既可以在创建时指定 ,也可以在创建后修改。如果在创建时不特别指定 ,则属性的设置会参照系统缺省对象的设置。这些缺省对象是在创建队列管理器时自动创建的 ,通常以 SYSTEM 开头命名。一级命令中的 DISPLAY 用于显示对象的属性 ,ALTER 用于修改属性。注意 ,有些对象的属性在创建后是无法修改的 ,比如对象的创建日期和时间或者通道的类型等。

所有的 MQSC 命令开始的语法格式都是由动作 (Action) + 对象 (Object) + 参数 (Parameter) 组成。动作就是一级命令 ,对象就是二级命令 ,每个对象都有惟一的名字 ,参数通常包括多个属性名与值。比如 :

```
DISPLAY QLOCAL (Q)
ALTER QMGR CCSID (1381)
DEFINE CHANNEL (C) CHLTYPE (SDR) CONNAME ('127.0.0.1 (1414)') XMITQ (XQ)
DELETE PROCESS (P)
```

在 RUNMQSC 中大小写无关 ,所有的命令会先转换成全大写再提交执行。所以如果要表示大小相关的字符串 ,比如对象名 ,则用引号将字符串包住。

```
RUNMQSC QM
DEFINE QLOCAL ('LocalQ')
END
amqsput LocalQ QM
```

通常情况下 ,我们会把要执行的命令先放入脚本文件中 ,用重定向的方法将其输入 ,输出也可以定向到另一个结果文件中。执行的时候 ,RUNMQSC 会首先回显命令 ,然后显示执行结果。所以 ,结果文件中既有命令的执行结果 ,也有命令本身。如果用 -e 选项抑制回显命令 ,这样命令输出中就只有执行结果了。

```
runmqsc -e QM < test.tst > test.out
```

MQSC 的每条命令可以有多种选项从而完成不同的功能。为了保持通用性 ,建议每条命令不超过 72 个字符。命令如果太长可以分行 ,将命令行的最后个非空字符置为 ‘ + ’ 或 ‘ - ’ ,表示需要折行。其中 , ‘ + ’ 表示后续内容从下一行的第一个非空字符算起 , ‘ - ’ 表示后续内容从下一行的第一个字符算起。如 :

```
DEFINE CHANNEL (C_QM1.QM2) +
CHLTYPE (SDR) +
TRPTYPE (TCP) +
CONNAME ('127.0.0.1 (1415)') +
XMITQ (QM2) +
REPLACE
```

3.2.1 队列管理器管理

MQSC 中用 ALTER QMGR 命令来修改队列管理器属性 ,用 DISPLAY QMGR 命令来显示队列管理器属性。

例 :

```
//将队列管理器设置为中文字符集
ALTER QMGR CCSID (1381)

//将队列管理器的死信队列设为 Q1 缺省传输队列强行设置为 Q2
ALTER QMGR DEADQ (Q1) DEFXXMITQ (Q2) FORCE

//显示队列管理器的全部属性
DISPLAY QMGR
```

```
// 显示队列管理的字符集
```

```
DISPLAY QMGR CCSID
```

3.2.2 队列管理

MQSC 中用 DEFINE 来创建队列 ,DELETE 来删除队列。这里的队列可以是本地队列、远程队列、别名队列或模型队列。ALTER 命令用来修改队列属性 ,DISPLAY 用来显示队列属性。

例：

```
// 创建本地队列 Q
```

```
DEFINE QLOCAL (Q)
```

```
// 将本地队列 Q 的最大深度属性设置为 5
```

```
ALTER QLOCAL (Q) MAXDEPTH (5)
```

```
// 重新创建本地队列 Q。如果 Q 已经存在 ,则将其属性全部重置为缺省属性
```

```
// 注意 ,这时队列中原有的消息仍然保留
```

```
DEFINE QLOCAL (Q) REPLACE
```

```
// 删除本地队列 Q
```

```
DELETE QLOCAL (Q)
```

3.2.3 通道管理

MQSC 中用 DEFINE 来创建通道 ,DELETE 来删除通道。用 DISPLAY 来显示通道属性 ,用 ALTER 来修改属性。注意 ,通道只有在停止状态下才可以被删除或修改。

例：

```
// 创建接收端通道 C
```

```
DEFINE CHANNEL (C) CHLTYPE (RCVR)
```

```
// 创建发送方通道 C ,连接对方的 IP 地址为 127.0.0.1 ,端口为 1414 ,通道连接传输队列 XQ
```

```
DEFINE CHANNEL (C) CHLTYPE (SDR) CONNAME ('127.0.0.1 (1414)') XMITQ (XQ)
```

```
// 删除通道 C
```

```
DELETE CHANNEL (C)
```

```
// 修改通道 C 的批次消息数量为 50
```

```
ALTER CHANNEL (C) CHLTYPE (SDR) BATCHSZ (50)
```

```
// 显示通道 C 的心跳间隔
```

```
DISPLAY CHANNEL (C) HBINT
```

由于通道是一种特殊的 MQ 对象 ,它的某些状态会随着通信环境的改变而变化 ,比如通道状态、通道流量、通道消息序号等 ,我们称之为通道的动态信息。MQSC 也提供了一些命令用来动态管理通道。

3.2.3.1 启动通道

启动通道有两种方式 :MQSC 命令和控制命令 ,启动通道只有在连接通道的主动方发起才有作用。

1. 用 MQSC 命令 START CHANNEL 可以启动通道。格式为：

```
START CHANNEL (ChannelName)
```

2. 也可以用 WebSphere MQ 控制命令 runmqchl 来启动通道。格式为：

```
runmqchl [-m QMgrName] -c ChannelName
```

3.2.3.2 停止通道

用 MQSC 命令 STOP CHANNEL 可以停止通道,停止通道也只有在连接通道的主动方发起才有作用。格式为:

```
STOP CHANNEL (ChannelName)
```

3.2.3.3 重置消息序号

通道为上面传送的每一条消息分配了一个序列号,它会自动累计增值,每传送一条消息,双方的消息序号都会自动加一。这个消息序号在通道中用 SEQNUM 属性表示,在双方连接通道的时候会约定一个起始值,以后每传递一条消息各自加一。通道的相关属性 SEQWRAP 表示序号的最大值,缺省最大值为 999,999。序列号越界后自动归零,从头开始。通道利用消息序号来标识传送和确认的消息。

通常情况下,通道双方的消息序号计数应该是相同的。然而在某些异常情况下,会出现双序号不一致的情况,这通常是因为通信故障后,双方对前面的某一条(或一批)消息是否发送成功理解不一致。在解决了不确定(In-doubt)的消息后,可以用 MQSC 命令通过重置消息序号将双方调整到一致。一旦连接断开后,通道重连时双方 MCA 会将消息序号同步。如果通信异常造成序号不一致,可以在通道发送端用 MQSC 命令 RESET CHANNEL SEQNUM 手工将两者同步。

在连接通道的主动方重置消息序号会将双方一起调整,在被动方重置则只设置一端。

```
RESET CHANNEL (ChannelName) [ SEQNUM (number) ]
```

3.2.3.4 解决不确定消息

发送方和接收方的通道状态中除了 SEQNUM (通道消息序号) 参数控制消息传递外,还有 LUWID 参数。LUWID 指的消息批次交易号,对于每一批消息发送方都需要收到接收方的确认信息才认为消息完整无误地送达对方,接着产生下一个 LUWID 并开始下一批消息传送。如果没有收到确认而与接收方失去联系,这时发送方认为这批消息为不确定(In-doubt)状态。

大多数时候,WebSphere MQ 会在通道重连时自动解决不确定状态的问题。当然,我们也可以手工解决。事实上,通道的 LUWID 分成 CURLUWID 和 LSTLUWID 两个参数属性,具体工作过程如下:

1. 发送方产生一个批次交易号,设置在 CURLUWID 并通知接收方
2. 接收方将其设置在 CURLUWID
3. 发送方向接收方一条接一条地传送整批消息
4. 接收方在完整地收到消息后,将交易号设置在 LSTLUWID,提交整批消息并回送确认信息
5. 发送方在接收确认信息后,将交易号设置在 LSTLUWID,提交整批消息
6. 重复转到 1

所以,在发送方出现不确定状态时,只需要比较一个发送方的 CURLUWID 和接收方的 LSTLUWID,就可以知道该批消息在接收端是否已经提交,从而在发送方做出相应的动作即可。具体步骤如下:

1) 比较双方的 LUWID

- 对于不确定(In-doubt)状态的发送端:

```
DISPLAY CHSTATUS(name) SAVED CURLUWID
```

- 对于接收端:

```
DISPLAY CHSTATUS(name) SAVED LSTLUWID
```

2) 如果两者相同,说明该批消息在接收端已经完整地收到并提交。在发送端执行:

```
RESOLVE CHANNEL(name) ACTION(COMMIT)
```

3) 如果两者不同,说明该批消息在接收端未能完整地收到并提交。在发送端执行:

```
RESOLVE CHANNEL(name) ACTION(BACKOUT)
```

3.2.3.5 测试通道

类似于 TCP/IP 中的 PING 命令,MQSC 命令中也有对通道的 PING,格式如下。其中,DATALEN 表示

PING 数据包的大小 ,可以用 16 字节到 32,768 字节。

```
PING CHANNEL(channel_name) [DATALEN(16 | integer)]
```

PING 命令可以检查对方的队列管理器或端口监听器是否启动 ,也可以检查对方的通道定义是否正确。但不检查通道的通性状态。换句话说 ,PING CHANNEL 只检查通道能否连连通 ,而不检查目前是否连通。

3.2.4 进程定义管理

与队列管理命令类似 ,MQSC 中用 DEFINE PROCESS 来创建进程定义 ,用 DELETE PROCESS 来删除进程定义 ,用 ALTER PROCESS 来修改进程定义属性 ,用 DISPLAY PROCESS 来显示进程定义属性。如表 3-2 ,具体格式参见附录。

例 :

```
//创建进程定义 P ,对应程序是记事本 (Notepad)
DEFINE PROCESS (P) APPLICID ('C : \WINNT \system32 \notepad.exe') REPLACE

//修改进程定义 P 的属性 ,设置用户数据 abc
ALTER PROCESS (P) USERDATA (abc) REPLACE

//删除进程定义 P
DELETE PROCESS (P)
```

3.2.5 名称列表管理

MQSC 用 DEFINE NAMELIST 来创建名称列表 ,用 DELETE NAMELIST 来删除名称列表 ;用 ALTER NAMELIST来修改名称列表属性 ,用 DISPLAY NAMELIST 来显示名称列表属性。

例 :

```
//创建名称列表 QNL ,含两个名称 Q1 和 Q2
DEFINE NAMELIST (QNL) NAMES (Q1 , Q2) REPLACE

//修改名称列表 QNL ,含三个名称 Q1、Q2 和 Q3
ALTER NAMELIST (QNL) NAMES (Q1 , Q2 , Q3) REPLACE

//删除名称列表 QNL
DELETE NAMELIST (QNL)
```

3.2.6 认证信息管理

MQSC 用 DEFINE AUTHINFO 来创建认证信息 ,用 DELETE AUTHINFO 来删除认证信息 ;用 ALTER AUTHINFO来修改认证信息属性 ,用 DISPLAY AUTHINFO 来显示认证信息属性。

例 :

```
//创建认证信息定义 AI ,指向 LDAP Server 的地址与端口
DEFINE AUTHINFO (AI) AUTHTYPE (CRLLDAP) CONNAME ('127.0.0.1 (4000)')

//显示认证信息定义 AI 的属性
DISPLAY AUTHINFO (AI)

//修改认证信息定义 AI ,指向另一个 LDAP Server 的地址与端口 ,且设定了用户名与密码
ALTER AUTHINFO (AI) CONNAME ('127.0.0.1 (5000)') LDAPUSER (user) LDAPPWD (pwd)

//删除认证信息定义 AI
DELETE AUTHINFO (AI)
```

3.2.7 小结

这一节列举了 WebSphere MQ 对象管理中常用的 MQSC 命令 ,如表 3 -2。我们通常可以使用这些命令对各种 MQ 对象实现创建、删除、修改属性、显示属性等功能 ,所有的 MQSC 命令都可以通过脚本文件的方式提交执行。对于更详细的语法说明可以参见附录“MQSC 命令一览表”。

表 3 -2 MQSC 队列管理命令

命令	说明
队列管理器管理	
ALTER QMGR	修改队列属性
DISPLAY QMGR	显示队列属性
队列管理	
DEFINEQLOCAL/QREMOTE/QALIAS/QMODEL	创建队列
DELETEQLOCAL/QREMOTE/QALIAS/QMODEL	删除队列
ALTER QLOCAL/QREMOTE/QALIAS/QMODEL	修改队列属性
DISPLAY QLOCAL/QREMOTE/QALIAS/QMODEL/QUEUE	显示队列属性
通道管理	
DEFINECHANNEL	创建通道
DELETECHANNEL	删除通道
ALTER CHANNEL	修改通道属性
DISPLAY CHANNEL	显示通道属性
START CHANNEL	启动通道
STOP CHANNEL	停止通道
RESET CHANNEL	重置通道
RESOLVE CHANNEL	解决通道争议
PING CHANNEL	测试通道
进程定义管理	
DEFINEPROCESS	创建进程定义
DELETEPROCESS	删除进程定义
ALTER PROCESS	修改进程定义属性
DISPLAY PROCESS	显示进程定义属性
名称列表管理	
DEFINENAMELIST	创建名称列表
DELETENAMELIST	删除名称列表
ALTER NAMELIST	修改名称列表属性
DISPLAY NAMELIST	显示名称列表属性
认证信息管理	
DEFINEAUTHINFO	创建认证信息
DELETEAUTHINFO	删除认证信息
ALTER AUTHINFO	修改认证信息属性
DISPLAY AUTHINFO	显示认证信息属性

3.3 基本队列操作

我们在了解了 MQ 控制命令和管理命令后 ,就可以自己构建 MQ 运行环境了。先创建并启动队列管理

器 ,再创建 MQ 队列对象 ,然后可以通过队列操作对消息进行存取。如果您要配置 MQ 的多机运行环境 ,可参看“4.1 通道配置”。

WebSphere MQ 提供了一些缺省的基本队列操作命令 ,如表 3 - 3。通过它们可以对队列进行最基本的读写操作 ,这些命令实际上都是 WebSphere MQ 应用程序 ,在安装目录中可以找到它们的源代码。

表 3 - 3 MQ 基本队列操作命令

命令	格式	说明
amqsput	amqsput QueueName [QueueManagerName]	从 Server 端将消息放入队列
amqsputc	amqsputc QueueName [QueueManagerName]	从 Client 端将消息放入队列
amqsget	amqsget QueueName [QueueManagerName]	从 Server 端将消息取出队列
amqsgetc	amqsgetc QueueName [QueueManagerName]	从 Client 端将消息取出队列
amqsbcg	amqsbcg QueueName [QueueManagerName]	从 Server 端查看消息
amqsbcgc	amqsbcg QueueName [QueueManagerName]	从 Client 端查看消息

这些基本队列操作命令分成两类 ,一类在 MQ Server 端运行 ,它们是 amqsput、amqsget、amqsbcg。另一类在 MQ Client 端运行 ,它们是 amqsputc、amqsgetc、amqsbcgc。命令执行的语法都类似 ,以 amqsput 为例 ,格式为 `amqsput QueueName [QueueManagerName]` ,其中 QueueName 为队列名 ,QueueManagerName 为队列管理器 ,如果不指明则为缺省队列管理器。例 :

```
C :\> amqsput Q QM
C :\> amqsget Q QM
C :\> amqsbcg Q QM
```

amqsput 和 amqsputc 可以将消息放入队列中 ,程序把之后的每一行标准输入作为一条独立的消息 ,读到 EOF 或空行时退出。注意 ,UNIX 上的 EOF 为 Ctrl + D ,Windows 上的 EOF 为 Ctrl + Z。可以将标准输入重定向到文件。队列中每放入一条消息 ,队列深度增加一。

amqsget 和 amqsgetc 可以将消息从队列中全部读出并显示。读空后再等待 15 秒 ,在这段时间内如果有新的消息到达会一并读出。如果强行中断该程序 ,比如用 Ctrl + C 强行退出 ,这时等待着的 MQGET 读操作尚未完成 ,用 MQSC 命令 `DISPLAY QSTATUS (Q) TYPE (HANDLE) OPENTYPE (OUTPUT) ALL`也可以观察到。MQGET 读操作会在一段时间后自动撤销 ,在这段时间内如果有新的消息到达 ,则第一条消息会被隐式地读走而丢失。amqsget 和 amqsgetc 执行后队列应该为空 ,即队列深度为零。

amqsbcg 和 amqsbcgc 可以详细查阅队列中现有的消息属性及内容而不将其取出。它与 amqsget 和 amqsgetc 惟一的差别就是查阅后消息仍然保留在队列中 ,队列深度不变。

3.4 MQ 配置信息

WebSphere MQ 的配置信息反映了当前 MQ 运行环境的设置。在 UNIX 中这些信息记录在配置文件中 ,在 Windows 中记录在系统注册表中。通常情况下 ,MQ 提供了管理命令或工具来修改这些设置 ,建议用户不要轻易地进行手工修改。

3.4.1 UNIX 配置文件

UNIX 平台上的 WebSphere MQ 配置文件有两种。一种为 mqs.ini ,它是针对整个 MQ 运行环境的 ,一台机器只有一个这样的文件。另一种为 qm.ini ,它是针对某个队列管理器配置的 ,每个队列管理器有一个这样的文件。管理员可以编辑这些文件以修改配置。

配置文件是分段的 ,每一段 (stanza) 有一个标题 ,以冒号 (:) 结尾。每一段中有多条属性设置 (Entry) ,它们都是“属性名 = 属性值” (Name = Value) 的格式。这里 ,缺省情况下配置文件中的段和设置都是比较精简的 ,不需要的部分可以不出现。在实际运行环境下 ,段和设置都是可以根据需要增减的。配置文件中总共可以有有哪些段 ,各段中可以有有哪些设置 ,在 WebSphere MQ 中都有规定。段不分先后次序 ,段

中的设置也不分先后次序。

下面以 WebSphere MQ for AIX 为例,让我们看一看缺省的 mqs. ini 和 qm. ini (队列管理器名为 QM)。

3.4.1.1 mqs. ini

```
AllQueueManagers :
  DefaultPrefix = /var/mqm
ClientExitPath :
  ExitsDefaultPath = /var/mqm/exits
LogDefaults :
  LogPrimaryFiles = 3
  LogSecondaryFiles = 2
  LogFilePages = 1024
  LogType = CIRCULAR
  LogBufferPages = 0
  LogDefaultPath = /var/mqm/log
QueueManager :
  Name = QM
  Prefix = /var/mqm
  Directory = QM
```

在 mqs. ini 配置文件中规定了整个 WebSphere MQ 的工作目录为 /var/mqm,队列管理器缺省有 3 个主 Log 文件 2 个备用 Log 文件。每个 Log 文件大小为 1024 页 (一页为 4KB),Log 文件缺省采用环形模式组织,且在 /var/mqm/log 目录下。

当前有一个队列管理器,名为 QM。它的工作目录为 /var/mqm 下的 QM,即 /var/mqm/QM 目录。

3.4.1.2 qm. ini

```
ExitPath :
  ExitsDefaultPath = /var/mqm/exits/
Log :
  LogPrimaryFiles = 3
  LogSecondaryFiles = 2
  LogFilePages = 1024
  LogType = CIRCULAR
  LogBufferPages = 0
  LogPath = /var/mqm/log/QM/
  LogWriteIntegrity = TripleWrite
```

在 qm. ini 配置文件中规定了队列管理器 QM 的一些参数。它有 3 个主 Log 文件 2 个备用 Log 文件。每个 Log 文件大小为 1024 页 (一页为 4KB),Log 文件缺省采用环形模式组织,且在 /var/mqm/log/QM 目录下,Log 用 TripleWrite 方式保持日志完整性。注意,qm. ini 中可能有一些参数名与 mqs. ini 相同,但设置值不同。则对于该队列管理器而言,以 qm. ini 为准。

3.4.2 Windows 注册表

在 Windows 注册表中,路径中的一个目录结点称为一个项 (Item),一个项中可以含多个其他项或设置,每个设置都是“名 = 值”的格式。这里最下一级的项就相当于 UNIX 配置文件中的段,项中的设置相当于 UNIX 配置文件中的设置。在配置运行环境时,可以相互参考。

与 UNIX 类似,WebSphere MQ for Windows 的属性设置也分两类,一类是针对整个 MQ 运行环境的,另一类是针对队列管理器的。它们可以通过编辑注册表来修改,也可以通过“WebSphere MQ 资源管理器”来设置或修改。以下介绍后一种方法的操作及相应的注册表项。

无论是 UNIX 的配置文件或是 Windows 的注册表 ,对于日志文件的属性在运行环境和队列管理器配置中各有一份设置。其中运行环境中的设置犹如一份模板 ,在队列管理器创建时如果不特别指明日志参数则将其拷贝到队列管理器中配置中 ,以后就只有队列管理器配置有作用了。

3.4.3 Windows 中 MQ 运行环境配置

在“ WebSphere MQ 服务 ”中右键点击“ WebSphere MQ 服务（本地）” 然后选择“ 属性 ”,即可以配置整个 MQ 运行环境的属性(图 3 - 1)。在 Windows 中对这些属性的配置都会映射到注册表中。



图 3 - 1 WebSphere MQ 服务

所有队列管理器	<MQ_HKEY> \Configuration\AllQueueManagers\
缺省前缀	DefaultPrefix
转换 EBCDIC 新行	ConvEBCDICNewline ,可以取值 NT_TO_LF、TABLE、ISO
客户机出口路径	<MQ_HKEY> \Configuration\ClientExitPath
缺省日志设置	<MQ_HKEY> \Configuration\LogDefaults\
日志类型	LogType ,可以取值 CIRCULAR(环形)或 LINEAR(线性)
日志路径	LogPath
日志文件页面数（每页 4KB）	LogFilePages
主日志文件数	LogPrimaryFiles
次日志文件数	LogSecondaryFiles
日志缓冲区页面数	LogBufferPages
ACPI 设置	<MQ_HKEY> \Configuration\ACPI\
API 出口	<MQ_HKEY> \Configuration\
公共出口	ApiExitCommon
模块出口	ApiExitTemplate

Win2000 支持 Advanced Configuration and Power Interface（ACPI）标准 ,这使得操作系统进入挂起（Suspend）模式或者从挂起模式恢复的时候有机会保护正在运行的通道。

3.4.4 Windows 中 MQ 队列管理器配置

在“ WebSphere MQ 服务 ”中右键点击队列管理器 ,然后选择“ 属性 ”,即可以配置队列管理器的各种属性 ,如图 3 - 2。在 Windows 中对这些属性的配置都会映射到注册表中。

常规	
启动类型	<MQ_HKEY_Services> \队列管理器\Startup 取值为 1 表示将队列管理器配置为自动启动



图 3-2 队列管理器属性配置

缺省队列管理器	<code>< MQ _ HKEY > \Configuration\DefaultQueueManager\Name</code> 一台机器只能有一个缺省队列管理器
恢复	<code>< MQ _ HKEY _ Services > \队列管理器</code> 在失败时可以设置重新启动的方式和重试次数
日志	<code>< MQ _ HKEY _ QM > \Log\</code> 日志类型 <code>LogType</code> ,可以取值 CIRCULAR(环形)或 LINEAR(线性) 日志路径 <code>LogPath</code> 日志文件页面数 (每页 4KB) <code>LogFilePages</code> 主日志文件数 <code>LogPrimaryFiles</code> 次日志文件数 <code>LogSecondaryFiles</code> 日志缓冲区页面数 <code>LogBufferPages</code> 写日志完整性 <code>LogWriteIntegrity</code> ,可以取值 SingleWrite、 DoubleWrite、TripleWrite 来保证消息完整性
资源	<code>< MQ _ HKEY _ QM > \XAResourceManager</code> 交换文件 <code>SwitchFile</code> XA 打开字符串 <code>XAOpenString</code> XA 关闭字符串 <code>XACloseString</code> 线程控制 <code>ThreadOfControl</code> ,可以取值 PROCESS 或 THREAD
服务	<code>< MQ _ HKEY _ QM > \Service\</code>
通道	<code>< MQ _ HKEY _ QM > \Channels\</code> 通道最大数 <code>MaxChannels</code> 活动通道最大数 <code>MaxActiveChannels</code> 启动程序最大数 <code>MaxInitiators</code>
出口	<code>< MQ _ HKEY _ QM > \ < QMgr > \</code>
TCP	<code>< MQ _ HKEY _ QM > \TCP\</code>
Netbios	<code>< MQ _ HKEY _ QM > \Netbios</code>
SPX	<code>< MQ _ HKEY _ QM > \SPX</code>
LU62	<code>< MQ _ HKEY _ QM > \LU62</code>

MQ 队列管理器的参数配置可以设定队列管理器的运行方式和参数。比如 ,通过配置资源 ,可以使队列管理器连接数据库 ,并在 MQ 交易中含数据库操作。通过配置通道 ,可以设定队列管理器的最大通道数、最大活动的通道等限制。通过配置出口 ,可以设定出口程序路径 ,等等。

3.5 MQ 管理方式

WebSphere MQ 的管理方式可以分成本地管理和远程管理器两种。无论哪一种方式,都可以通过命令行或图形界面工具来完成。对于 Windows 平台有“WebSphere MQ 服务”工具进行配置,对于编程方式,可以发送 PCF 命令来实现管理。

本地管理指通过管理工具对本地的队列管理器实施管理,即管理工具与被管理的队列管理器在同一台机器上。远程管理则是通过一定的设置,用相同的管理工具管理另一台机器上的队列管理器。

3.5.1 本地管理

3.5.1.1 用 RUNMQSC 实现本地管理

RUNMQSC 是一个通用的 MQ 对象管理工具,使用 MQSC 命令集可以对 MQ 对象进行全方位的管理,也是各种管理方式最直接、最全面的一种。

在队列管理器所有机器的命令行中执行 `RUNMQSC <QMGrName>` 即可进入 RUNMQSC 的交互界面,其中 `<QMGrName>` 是本地的队列管理器名。

3.5.1.2 用 MQ 资源管理器实现本地管理

Windows 平台的 WebSphere MQ 提供了资源管理器这样一个图形界面管理工具,如图 3-3。它是借助 Microsoft Management Console (MMC) 环境进行管理的,使用十分方便。其中列出了所有被管理的队列管理器,缺省情况下它们都是本地队列管理器,当然也可以通过远程管理将远程队列管理器纳入其中。



图 3-3 WebSphere MQ 资源管理器

在队列管理器下列出了所有的 WebSphere MQ 对象,用户可以点击右键,在菜单中选择各种操作,简单易懂。使用 WebSphere MQ 资源管理器来管理 MQ 对象之前必须启动相应队列管理器的命令服务器。

3.5.1.3 用 MQ 服务实现本地管理

除了对队列管理器中的对象进行管理之外,我们还需要对队列管理器自身以及整个 MQ 运行环境进行管理。为此,WebSphere MQ for Windows 提供了另一个管理工具:WebSphere MQ 服务。它也是借助 Microsoft Management Console (MMC) 环境进行管理的,使用同样方便,如图 3-4。

在界面中可以为队列管理器配置各种服务,缺省有“队列管理器”和“命令服务器”,此外还有“侦听器”、“通道启动程序”、“触发器监控器”和“通道”。其中,前两种每个队列管理器只能各配置一个,后四种每个队列管理器可以配置多个。此外,通过配置定制服务,可以配置在队列管理器启动前或启动后自动执

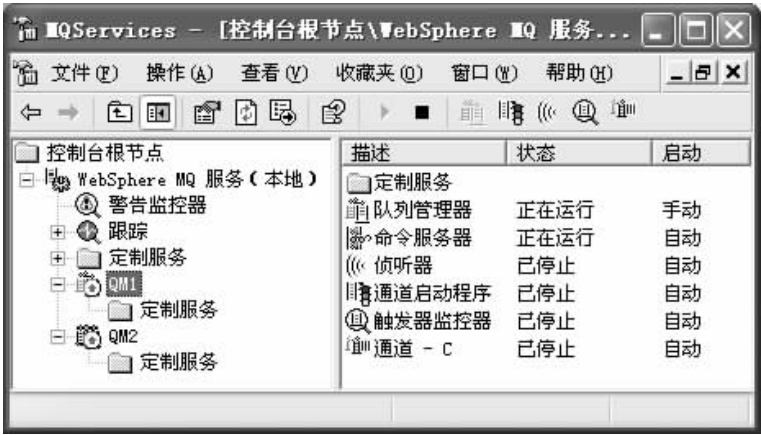


图 3-4 WebSphere MQ 服务

行的命令,它可以是调用一条命令,也可以是以进程方式独立运行的程序。

正如我们前面提到的,WebSphere MQ 中有一些属性是针对队列管理器的,另有一些属性是针对整个 MQ 运行环境的。在 UNIX 中它们分别设置在 qm.ini 和 mqs.ini 文件中,在 Windows 中它们设置在注册表里,这些属性也可以通过 WebSphere MQ 资源管理器设置或修改。

3.5.1.4 用 PCF 编程实现本地管理

WebSphere MQ 资源管理器实际上把管理指令以 PCF 消息的格式放入被管理的队列管理器的 SYSTEM.ADMIN.COMMAND.QUEUE 中,由该队列管理器上的命令服务器执行并返回结果,再在图形界面上表现出来。WebSphere MQ 远程管理实际上也是这个原理。

PCF 消息遵循一定的格式,如果通过编程人为地构造一条 PCF 消息并用 MQPUT 放入 SYSTEM.ADMIN.COMMAND.QUEUE 中,就可以编程向队列管理器发送一条管理指令了。消息中设置 ReplyToQ 指定结果的返回队列,命令服务器在执行完毕后会返回一条结果消息,该消息也遵循同样的格式,也是一条 PCF 消息。基于这个原理,我们可以通过 PCF 编程在程序中随心所欲地控制并管理队列管理器。如果管理指令来自远端,则可以很方便地实现远程管理。

3.5.2 远程管理

WebSphere MQ 的远程管理的原理是将管理命令包装成 PCF 消息并传送到远程队列管理器的命令队列 SYSTEM.ADMIN.COMMAND.QUEUE 中,命令服务器将管理消息读走并执行,再将执行结果也包装成 PCF 格式的消息并返回,如图 3-5。返回队列由管理消息的 ReplyToQ 域指定。另外,要实现远程管理,被管理的队列管理器的命令服务器必须首先启动。

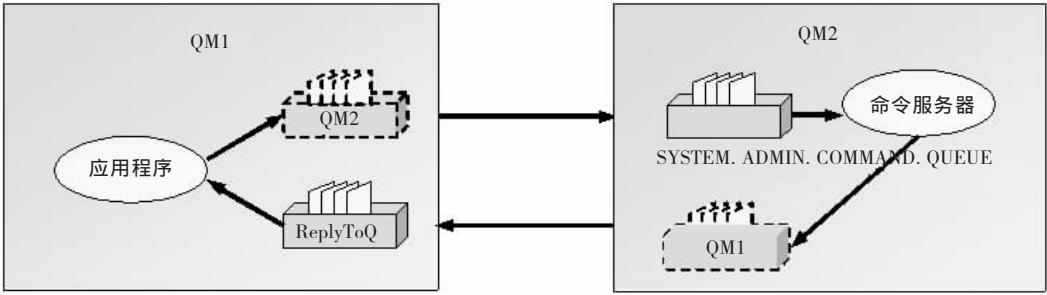


图 3-5 MQ 远程管理的原理

下面让我们详细了解一下用 RUNMQSC 和用 MQ 资源管理器两种方式各自是如何进行远程管理的。

3.5.2.1 用 RUNMQSC 实现远程管理

我们可以用通用的 RUNMQSC 命令行管理工具对远程队列管理器实施管理。这里假定 QM1 为本地缺

省队列管理器 ,QM2 为受管理的远程队列管理器。步骤如下：

- 1. 确认队列管理器上有 SYSTEM. ADMIN. COMMAND. QUEUE 队列 ,并启动远程的队列管理器的命令服务器进程

```
strmqcsv QM2
```

- 2. 在本地建立缺省队列管理器 ,并与远程队列管理器建立双向通道

```
QM1 :
- - - -
DEFINE CHANNEL (C_QM1.QM2) +
        CHLTYPE (SDR) +
        TRPTYPE (TCP) +
        CONNAME ('10.10.10.21 (1416)') +
        XMITQ (QM2) +
        REPLACE
DEFINE CHANNEL (C_QM2.QM1) +
        CHLTYPE (RCVR) +
        TRPTYPE (TCP) +
        REPLACE
DEFINE QLOCAL (QM2) +
        USAGE (XMITQ) +
        REPLACE

QM2 :
- - - -
DEFINE CHANNEL (C_QM1.QM2) +
        CHLTYPE (RCVR) +
        TRPTYPE (TCP) +
        REPLACE
DEFINE CHANNEL (C_QM2.QM1) +
        CHLTYPE (SDR) +
        TRPTYPE (TCP) +
        CONNAME ('10.10.10.11 (1415)') +
        XMITQ (QM1) +
        REPLACE
DEFINE QLOCAL (QM1) +
        USAGE (XMITQ) +
        REPLACE
```

- 3. 本地和远程队列管理器都要有相应的 TCP/IP Listener。

```
runmqclsr -m QM1 -t tcp -p 1415
runmqclsr -m QM2 -t tcp -p 1416
```

- 4. 通过 RUNMQSC 进行管理 ,这时 RUNMQSC 工具首先连接本地缺省队列管理器 ,再通过事先配置并连接好的通道 ,把命令发往指定的队列管理器 ,命令执行结果原路返回。由于涉及队列管理器之间的通信 ,可以用 -w 选项设定超时时间 (秒) ,如果在规定的时间内命令结果没有返回 ,则执行失败。

```
runmqsc -w 100 QM2 < mqsc.tst > result.out
```

3.5.2.2 用 MQ 资源管理器实现远程管理

我们可以用 Windows 下的 WebSphere MQ 资源管理器进行远程监控 ,从而利用该管理工具友好的图形界面进行远程管理。步骤如下：

- 1. 确认队列管理器上有 SYSTEM.ADMIN.COMMAND.QUEUE 队列 ,并启动远程的队列管理器的命令服务器进程。

```
strmqcsv <QMgr>
```

- 2. 远程的队列管理器至少有一个 TCP/IP Listener ,可以是 runmqlsr ,也可以是 inetd。参见“配置”章节。

```
runmqlsr -m <QMgr> -t tcp -p <port>
```

- 3. 在远程的队列管理器上配置名为 SYSTEM.ADMIN.SVRCONN 的 Server – Connection 通道。

```
DEFINECHANNEL (SYSTEM.ADMIN.SVRCONN) +
CHLTYPE (SVRCONN) +
TRPTYPE (TCP) +
MCAUSER ('mqm') +
REPLACE
```

- 4. 在 Windows 平台上的 WebSphere MQ 资源管理器中点击“队列管理器” ,右键选择“显示队列管理器” 。在弹出菜单中选择“显示远程队列管理器” ,给出队列管理器名称和连接名称 ,其中连接名称要与通道的 CONNAME 属性相符 ,需要含有远程队列管理器的地址和监听端口 ,可以是 IP (Port) 格式 ,如图 3 – 6。

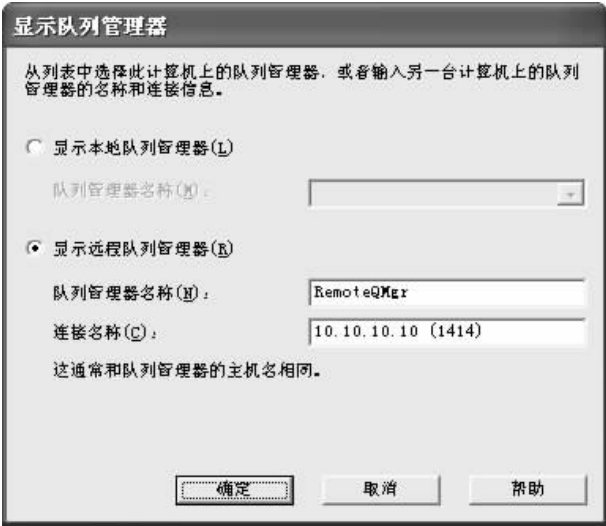


图 3 – 6 用 WebSphere MQ 资源管理器远程管理

3.6 日 志 (Log)

WebSphere MQ 中日志为系统的稳定运行和消息的可靠传递提供了保障 ,在 WebSphere MQ 中扮演着不可缺少的角色。基本上 ,MQ 的对象操作及对持久消息的操作都会记入日志中 ,在系统故障时日志可以帮助 MQ 队列管理器在重新启动后恢复到原来的状态 ,所有的持久消息仍然保留不丢失。日志也可以用来将整个 MQ 对象整体记录 ,以便日后进行对象整体恢复。因为日志极其重要 ,所以对日志文件的定期备份就显得十分必要。

3.6.1 队列管理器日志

队列管理器的日志是随着队列管理器一起创建的 ,在创建队列管理器时可以指定日志属性 ,其中一部分属性也可以在队列管理器创建后再修改调整。对于创建时未指定的参数 ,WebSphere MQ 使用缺省参数值。

日志中记录了队列管理器的各种活动 ,包括创建和删除对象 (除通道外)、持久性消息、交易状态、对象状态的更改、通道的活动 ,等等。所以 ,在 WebSphere MQ 系统出了以下问题的时候 ,日志可以用来做

恢复。

- 重启恢复 (Restart Recovery) ,在重新启动 WebSphere MQ 后保证所有的持久性消息都还在 ,所有的交易回滚到发生前的状态 ,删除未提交的消息。由于非持久消息是不记录日志的 ,所以恢复只针对持久消息。
- 损坏恢复 (Crash Recovery) ,WebSphere MQ 由于内部错误非正常停止 ,重新启动后恢复到检查点 (CheckPoint) ,即队列管理器与日志一致点。
- 介质恢复 (Media Recovery) ,在介质损毁修复后 ,根据日志修复对象。

3.6.1.1 日志文件

WebSphere MQ 日志分成两部分 :多个日志数据文件和一个日志控制文件。

MQ 的日志数据文件在 Windows 中缺省在 `<InstallDir>\log\<QMgrName>\active` 目录下 ,每个文件缺省大小为 1MB。在 UNIX 中 ,缺省在 `/var/mqm/log/QmName/active` 目录下 ,每个文件缺省大小 4MB。

数据文件有主数据文件 (Primary) 和次数据文件 (Secondary) 之分 ,它们在文件结构和大小上都是相同的。队列管理器初始启动的时候只会创建指定数量的主数据文件 ,当需要的时候 (比如含有众多消息操作的长交易) 会一个一个创建出次数据文件 ,加入到日志数据文件中。当不再需要时 ,次数据文件又被会动态删除。如果所有的次数据文件也都用完了 ,仍不能满足系统记录日志的需要 ,则系统对任何日志操作都将返回 MQRC_RESOURCE_PROBLEM。主次数据文件的数量由创建队列管理器时的 LogPrimary-Files 和 LogSecondaryFiles 指定。

MQ 的日志控制文件在 Windows 中为 `<InstallDir>\log\<QMgrName>\amqhlctl.lfh` ,在 UNIX 中 ,为 `/var/mqm/log/<QMgrName>/amqhlctl.lfh`。日志控制文件中记录了目前数据文件的主次分配、活动状态等信息。

日志的空间是有限的 ,理论上最大占用空间为 : (主日志数量 + 次日志数量) × 日志文件大小。如果有长交易占用了太多的日志空间 ,系统会将其强行回滚 ,以释放日志空间。该交易所在的应用无法在同一个交易中再使用 MQPUT 或 MQGET ,MQGET 会返回 MQRC_BACKED_OUT。应用可以用 MQCMIT (返回 MQRC_BACKED_OUT) 或 MQBACK 结束交易 ,从而创建新的交易。

3.6.1.2 环形和线性 (Circular & Linear)

WebSphere MQ 中的日志分为两种 环形日志 (Circular Logging) 和 线性日志 (Linear Logging)。

环形日志可以用来做重启恢复 (Restart Recovery)。它记录了当前所有的交易状态和交易中的持久消息操作 ,维护了所有重启相关需要的数据。

线性日志不仅可以用来做重启恢复 (Restart Recovery) ,还可以用来做介质恢复 (Media Recovery)。它不但记录了环形日志的所有内容 ,还因为日志文件集合是线性增长而不会循环覆盖 ,队列管理器创建以来所有持久消息的活动都记录在案。所以 ,如果队列文件坏了 ,系统可以凭借完整的日志记录重建对象和消息。

3.6.1.3 日志参数

LogPrimaryFiles = 3 2 - 62	主日志数据文件数量。缺省为 3 ,最小为 2 ,最大为 62。主次日志文件之和不大于 63 , 不小于 3
LogSecondaryFiles = 2 1 - 61	次日志数据文件数量。缺省为 2 ,最小为 1 ,最大为 61。主次日志文件之和不大于 63 ,不小于 3
LogFilePages = number	日志文件大小 ,以 4KB 为单位。 Windows 平台 缺省为 256 ,即 1MB。最小为 32 ,最大为 16 384 UNIX 平台 缺省为 1024 ,即 4MB。最小为 64 ,最大为 16 384
LogType = CIRCULAR LINEAR	日志类型。缺省为环形日志 ,CIRCULAR

LogBufferPages = 010 - 512	日志缓冲区大小 ,以 4KB 为单位。最小为 18 ,最大为 512。缺省为 0。如果值为 0 - 17 ,WebSphere MQ 实际使用 18 ,即 72KB。
LogPath = dir	日志文件目录
LogWriteIntegrity = SingleWrite DoubleWrite TripleWrite	日志完整性
● SingleWrite	只写一次日志 ,在高可靠环境下 ,需要硬件保证 ,比如 SSA Cache
● DoubleWrite	可能需要写两次
● TripleWrite	可能需要写三次 缺省值。安全性高 ,效率较低

这些日志参数中 ,有些在队列管理器创建后就不能改动了。比如 LogType、LogPath、LogFilePages 等。有些则可以调整 ,在队列管理器重启后生效。比如 LogPrimaryFiles、LogSecondaryFiles、LogBufferPages 等。

一般来说 ,日志文件较大而文件数量较少与文件较小而数量众多相比 ,更有效率。LogBufferPages 的值较大意味着吞吐量较大 ,适合于大消息和大交易环境。LogWriteIntegrity 的选取在大容量、高压力的环境中在很大程度上会影响运行性能。

在生产系统中 ,为了在必要的时候进行介质恢复 ,在创建队列管理器时通过指定 LogPath 可以把日志文件指定到其他硬盘上去 ,与队列管理器对象工作目录不在同一块硬盘上。这样 ,如果有一块硬盘损坏了 ,系统可以修复。表 3 - 4 统计了日常操作与日志长度的关系 ,可以据此估算系统所需的日志文件大小和数量。

表 3 - 4 WebSphere MQ 操作与日志中记录长度的关系

WebSphere MQ 操作	日志中记录长度
Put (持久消息)	750 字节 + 消息长度
Get	260 字节
Commit 提交	750 字节
Rollback 回滚	1000 字节 + 每个 Get/Put 需要 12 字节
创建对象	1500 字节
删除对象	300 字节
改变属性	1024 字节
Record media image	800 字节 + Image 大小
Checkpoint	750 字节 + 每个活动的交易需要 200 字节

3.6.2 检查点 (Checkpoint)

WebSphere MQ 的检查点是一组记录 ,含有重启队列管理器必需的信息 ,检查点标志着队列 (queue) 与日志 (log) 在这一时刻是一致的 ,检查点之前已经提交的交易被认为在队列中体现了 ,所以检查点之前的日志文件在做重启恢复 (Restart Recovery) 时不需要。WebSphere MQ 中 ,我们把日志文件中做重启恢复所必需的部分称为活动日志 (Active Log) ,通常它们是最近一次检查点以来的所有日志 ,我们把其他部分称为非活动日志 ,即最近一次检查点之前的日志为非活动日志。如果是环形日志 ,非活动日志可以被覆盖重用。如果是线性日志 ,非活动日志可以被存档 (Archive) ,保存在其他介质中。

WebSphere MQ 对于队列上的任何一个持久消息 ,都是先记日志再入队列 ,检查点不会插在这两个动作之间。所以 ,检查点意味着日志与队列在这个时间点上是一致的。WebSphere MQ 在满足以下任何一个条件的时候自动产生检查点 :

- 队列管理器启动
- 队列管理器停止
- 日志空间不足 ,需要主动释放
- 每 10 000 次日志操作 (在 z/OS 平台上 ,这个数字可以由 LOGLOAD 参数控制)

在出现非活动日志文件后,系统会自动生成出相应数量的新日志数据文件,补齐活动日志文件的数量。对于非活动日志文件,管理员可以将其另行存放,也可以在对 WebSphere MQ 做备份后删除它们。因为介质恢复 (Media Recovery) 可以由备份文件及备份点之后的日志文件完成。

3.6.3 记录和复原 (Record & Recover)

记录和复原功能只对线性日志有效,可以用于介质恢复。记录映像 (Record Image) 就好像给 WebSphere MQ 对象拍照,将内容记入日志中,拍照之前的日志内容对于该对象来说就可以不必保留了。复原映像则是一个逆过程,从日志中找到最近的照片,并按其后的日志内容进行相应的修补,将对象复原为最近的样子。

`rcdmqimg` (Record Image) 命令和 `rcrmqobj` (Recreate Object) 命令分别用来记录和复原 WebSphere MQ 对象的映像快照 (Image)。`rcdmqimg` 将快照记入日志文件,`rcrmqobj` 将最近的一次快照从日志文件中取出,并且按以后日志文件中的内容将对象恢复。快照在恢复时可能有两种情况:

- 1) 如果做过快照,则按最近的一次快照所在的日志文件及其以后的所有日志文件来恢复。
- 2) 如果没有做过快照,则按队列管理器创建以来的所有日志文件来恢复。

WebSphere MQ 在启动时如果发现有对象损坏,会试图按照快照自动恢复。如果启动时未发现对象损坏,队列管理器启动成功,接着在运行时发现了,这时只有靠手工 `rcrmqobj` 恢复。注意 `rcdmqimg` 和 `rcrmqobj` 都要在队列管理器运行时操作。

系统会定期自动打印 AMQ7467 和 AMQ7468 信息,记录到错误日志中。使用 `rcdmqimg` 命令也可以主动生成这些信息(如下例所示),这些信息的内容会指明哪些较早的日志文件没有在线保存的必要了。对于启动恢复所需的最早日志文件,其中含有最近的一次检查点。对于介质恢复所需的最早日志文件,其中含有最近的一次 `rcdmqimg`。例:

AMQ7467: 启动队列管理器 QM 所需的最早的日志文件是 S0000000.LOG。

AMQ7468: 执行队列管理器 QM 的介质恢复所需的最早的日志文件是 S0000001.LOG。

3.6.3.1 记录 (Record Image)

- 格式

```
rcdmqimg [ -z ] [ -l ] [ -m QMgrName ] -t ObjType [ ObjName ]
```

- 功能

`rcdmqimg` 命令用于记录对象的快照。

- 说明

选项 `-z` 表示抑制命令执行过程中的输出。`-l` 表示列出重启队列管理器和介质恢复所需的最早的日志文件名。`-m QMgrName` 指明对象所在的队列管理器。`ObjType` 是对象类型,可以是 `qmgr`、`queue`、`process`、`namelist`、`authinfo` 等,也可以用 `all` 或 `*` 表示选择所有的对象。`ObjName` 表示具体需要记录的对象名。

- 例:

```
rcdmqimg -m QM -t queue Q
```

3.6.3.2 复原 (Recreate Object)

- 格式

```
rcrmqobj [ -z ] [ -m QMgrName ] -t ObjType [ ObjName ]
```

- 功能

在日志中查取最近的快照并据此复原对象。

- 说明

选项 `-z` 表示抑制命令执行过程中的输出。`-m QMgrName` 指明对象所在的队列管理器。

ObjType是对象类型,可以是 qmgr、queue、process、namelist、authinfo 等,也可以用 all 或 * 表示选择所有的对象。ObjName 表示具体需要记录的对象名。

- 例:

```
rcrmqobj -m QM -t queue Q
```

3.6.4 备份和恢复 (Backup & Restore)

备份和恢复是通过文件拷贝备份的手段完成的,对环形日志和线性日志都有效。如果是线性日志,在以下情况下,可以只恢复队列管理器的目录,而沿用原来的日志目录:

- 1) 备份时队列管理器处于检查点状态,即备份的对象与日志在备份点上是一致的。
- 2) 日志目录中留有备份点以来的所有日志文件,即有备份点以后的完整变化记录。

对于线性日志,WebSphere MQ 在备份后非活动日志可以认为是不必要的了,所以在备份点之前的日志可以被删除或永久归档。

3.6.4.1 备份 (Backup)

- 1) 首先,队列管理器处于停止状态。
- 2) 拷贝备份相关的目录及文件,包括:
 - <InstallDir>/QMGRS/<QMGRName> (包括所有的目录和文件)
 - <InstallDir>/log (包括数据文件和控制文件)

3.6.4.2 恢复 (Restore)

- 1) 首先,队列管理器处于停止状态。
- 2) 删除原来的相关目录及文件,再拷贝恢复相关的目录及文件,包括:
 - <InstallDir>/QMGRS/<QMGRName> (包括所有的目录和文件)
 - <InstallDir>/log (包括数据文件和控制文件)

3.6.5 导出日志 (Dump Log)

用 dmpmqlog 命令可以将队列管理器的日志内容输出成文本。dmpmqlog 只有在队列管理器停止的时候可以执行,缺省为输出上一个检查点以来的内容。由于在队列管理器停止的时候,会写入检查点,所以 dmpmqlog 通常只会输出较少的日志内容,当然也可以将指定一段日志的所有内容都输出。下面是 dmpmqlog 命令的使用方法。

- 格式

```
dmpmqlog [ -b | -s StartLSN ] [ -e EndLSN ] [ -f LogFilePath ] [ -m QMgrName ]
```

- 功能

将指定的日志文件导出并输出成文件。

- 说明

-b 表示从第一个日志开始,StartLSN 表示开始的日志号,EndLSN 表示结束的日志号,它们都用来指定一段日志文件。LogFilePath 指定日志文件所在的目录。QMGRName 为队列管理器名。

- 例:

```
dmpmqlog -m QM > QMLog.dmp
```

第4章 通信与配置

WebSphere MQ 的互连通信就是将多个队列管理器通过通道配置连接在一起,使消息能够自动地传递到目标队列,形成协同工作的能力。通常情况下,互连配置的复杂度与消息传送路径中经过的队列管理器数量有关。对于最简单的情况,就是只有两个队列管理器,它们之间用通道直接连接。对于多个队列管理器,可以形成链式、树型或网状结构,通过配置技巧形成消息的自动路由。

4.1 消息路由

消息路由就是消息根据其目标地址(目标队列管理器和目标队列)转发传递的过程,如果事先做了恰当的配置,这一过程可以由 WebSphere MQ 自动完成。为了解释清楚,让我们先来了解一下消息的路由过程。

4.1.1 消息路由过程

消息的路由过程大致如下:

1. 消息在源队列管理器中被放入远程队列时,消息的头结构(MQMD)中并未包含消息的目的地,这时消息的结构为 MQMD + Body,且不含目标地址信息。
2. 由于远程队列只是虚的队列定义,指向的队列实体是传输队列。消息在放入传输队列的时候,会自动加上传输消息头,消息结构变为 MQMD + MQXQH + Body。其中, MQXQH 中有 Remote-QName 和 RemoteQMGrName 两个域,分别会自动填入远程队列的目标队列和目标队列管理器属性。这时消息含有目标地址信息。
3. 如果有通道对应于该传输队列,且通道启动,就意味着有 MCA 通信进程在读该传输队列并可以发送,这时消息就被读出并由该通道传送到下一个队列管理器。
4. 消息到达下一个队列管理器后,首先会验证是否到达目标队列管理器。如果队列管理器名与 RemoteQMGrName 一致,则说明到达了目的地,转而寻找目标队列。
5. 如果队列管理器中有名为 RemoteQMGrName 的远程队列定义,指向另一个远程队列管理器为 AnotherRemoteQMGrName,且该定义的远程队列属性为空,这说明目标队列管理器在该队列管理器环境中被重定义了,我们称之为队列管理器别名,真正的目标队列管理器为 AnotherRemoteQMGrName。消息替换目标地址后回到第 4 步重新验证。
6. 如果 4、5 都不满足,说明该队列管理器只是消息路径中的一个节点,消息需要被继续路由。如果在队列管理器上有名为 RemoteQMGrName 的传输队列,则消息被自动放入该传输队列,并经由相应的通道路由去下一站。如果队列管理器没有缺省传输队列,则消息被放入缺省传输队列中,并经由相应的通道路由去下一站。
7. 转到 4。
8. 消息在到达目标队列管理器后,会试图放入目标队列中。如果没有名为 RemoteQName 的目标队列,但有同名的别名队列,则会放入别名队列指向的目标队列中。
9. 如果在放入时发生问题,比如目标队列满、禁放,目标队列非本地队列或根本不存在目标队列,等等,消息会试图就近放入死信队列。当然,这与消息的持久性属性有关。

4.1.2 缺省传输队列

缺省传输队列的作用就是当队列管理器发现消息需要进一步路由,但又找不到预先定义的合适的传

输队列时 ,为队列管理器指明缺省的路由路径。这时消息将被统统放入缺省传输队列 ,并由相应的通道路由去下一站。缺省传输队列由队列管理器的 DEFXMITQ 属性定义。

设置缺省传输队列的想法很简单 :凡是上一站来的消息 ,如果其目的地不是本站 ,就应该去下一站。缺省传输队列适合于单向的链式结构 ,对于无法传递的消息 ,可能会堆积在末端队列管理器的死信队列中。

4.1.3 队列管理器别名

如果远程队列定义中远程队列管理器名 (RQMNAME) 不为空 ,但远程队列名 (RQNAME) 为空 ,我们称之为队列管理器别名。设置队列管理器别名意味着目标队列管理器在新的环境中被重定义了 ,所以消息的目的地也就随之变化了。队列管理器别名用于在中途更改映射消息的目标队列管理器。

4.1.4 多级跳

无论是缺省传输队列还是队列管理器别名 ,都可以在消息传输路径上干预消息的传递和路由。通过两者的结合 ,可以使消息从源目标管理器经过多次路由中转到达目标队列管理器 ,我们称之为多级跳 (Multi-Hopping)。源队列管理器和目标队列管理器之间没有直接连接 ,消息在路由的过程中只经过传输队列和通道的中转而不经中间程序的干预。

4.1.5 传输中的消息

消息在经过远程队列放入传输队列时会由队列管理器自动添加一个传输头 (MQXQH) ,且传输头中的内容会根据远程队列定义自动填写。如果说 ,原先的消息是 MQMD + Body ,则放入传输队列的消息为 MQMD + MQXQH + Body ,如下 :

MQMD

```
StrucId = [ MD      ]
Version = [ MQMD_VERSION_1 ]
Report = [ MQMO_NONE ]
MsgType = [ MQMT_DATAGRAM ]
Expiry = [ -1 ]
Feedback = [ MQFB_NONE ]
Encoding = [ MQENC_NATIVE ]
CodedCharSetId = [ 1381 ]
Format = [ MQXMIT      ]
Priority = [ 0 ]
Persistence = [ MQPER_NOT_PERSISTENT ]
MsgId = [ ... ]
CorrelId = [ ... ]
BackoutCount = [ 0 ]
ReplyToQ = [                                     ]
ReplyToQMgr = [ QM                               ]
UserIdentifier = [ chenyx      ]
AccountingToken :
Length = [ 22 ]
Content = [ ... ]
Type = [ MQACTT_NT_SECURITY_ID ]
ApplIdentityData = [                                     ]
PutApplType = [ MQAT_QMGR ]
PutApplName = [ QM                               ]
PutDate = [ 20031028 ]
```

```

PutTime = [13091072 ]
ApplOriginData = [      ]
GroupId = [ ... ]
MsgSeqNumber = [1 ]
Offset = [0 ]
MsgFlags = [MQMF_NONE ]
OriginalLength = [-1 ]
MQXQH
StrucId = [XQH ]
Version = [MQXQH_VERSION_1 ]
RemoteQName = [Q1 ]
RemoteQMgrName = [QM1 ]
StrucId = [MD ]
Version = [4295468 ]
Report = [4371216 ]
MsgType = [MQMT_DATAGRAM ]
Expiry = [-1 ]
Feedback = [MQFB_NONE ]
Encoding = [MQENC_NATIVE ]
CodedCharSetId = [1381 ]
Format = [MQSTR ]
Priority = [0 ]
Persistence = [MQPER_NOT_PERSISTENT ]
MsgId = [ ... ]
CorrelId = [ ... ]
BackoutCount = [0 ]
ReplyToQ = [ ]
ReplyToQMgr = [QM ]
UserIdentifier = [chenyux ]
AccountingToken :
Length = [22 ]
Content = [ ... ]
Type = [MQACTT_NT_SECURITY_ID ]
ApplIdentityData = [ ]
PutApplType = [MQAT_WINDOWS_NT ]
PutApplName = [D:\WMQ\bin\amqspu.exe ]
PutDate = [20031028 ]
PutTime = [13091072 ]
ApplOriginData = [      ]
Body (Length = 3)
0000:61 62 63
abc

```

这里有几点需要说明：

- 1) 队列管理器同时为消息生成相同的 MQMD. CodedCharSetId 和 MQXQH. CodedCharSetId ,通常会根据 MQPUT 时远程队列的 MQMD. CodedCharSetId 生成。
- 2) 队列管理器会生成 MQMD. Format = MQXMIT ,而 MQXQH. Format 会根据 MQPUT 时远程队列的 MQMD. Format 生成。
- 3) 队列管理器同时为消息生成 MQMD. MsgId、MQMD. CorrelId 和 MQXQH. MsgId ,且为了表示 MQMD 与 MQXQH 之间的关系 ,MQMD. CorrelId = MQXQH. MsgId。

- 4) MQMD. PutApplType = MQAT_QMGR, MQMD. PutApplName = 本地队列管理器名。表示 MQMD 是由本地队列管理器 MCA 生成。MQXQH. PutApplType = MQAT_WINDOWS_NT, MQXQH. PutApplName = 应用程序名。表示 MQXQH 消息是由应用程序产生。
 - 5) 队列管理器同时生成 MQMD. PutDate, MQMD. PutTime, MQXQH. PutDate, MQXQH. PutTime。
- 由于 1)、2) 是可以由程序设定的, 所以可以模仿。3)、4)、5) 是队列管理器设定的, 几乎无法模仿。如果直接将带有传输头的消息写入传输队列, 也可以将消息内容传递出去, 但对于系统设定消息属性往往是一片空白。参见本书例程 MQPutTransmissionQueue.c。
- 传输队列缺省是只写不读的, 即使手工将队列属性改为 GET(ENABLED) 也不行, 队列管理器过一段时间会将传输队列属性改回 GET(DISABLED)。

4.2 通道配置

正如我们前面提到的, WebSphere MQ 中通信双方的通道类型配对共有 6 种, 我们在这里对其中基本的 Sender/Receiver、Server/Receiver、Server/Requester 和 Sender/Requester 四种方式给出配置脚本。至于另外两种类型, Sender - Connection/Receiver - Connection 可以参见“客户端”章节, Cluster - Sender/Cluster - Receiver 可以参见“群集”章节。

4.2.1 Sender (QM1) — Receiver (QM2)

Sender/Receiver 通道是最常见的通道配置方式, Sender 作为通道的发送方也是通道连接的主动发起方, Receiver 作为通道的接收方也是通道连接的被动监听方。在 Receiver 端要配置并运行相应的监听器。

在以下的配置脚本中, 通道连接两个队列管理器 QM1 和 QM2。其中, QM1 为 Sender, QM2 为 Receiver。在 QM1 上配置了远程队列 QR 和传输队列 QX, 其中 QR 指向队列管理器 QM2 上的本地队列 QL, 且 QR 与 QX 对应, 即凡是要放入 QR 队列的消息, 在加上传输消息头后直接放入 QX 中等待发送。QM1 上配置 Sender 通道需要指定对方的通信参数 (IP 地址和端口), 而这些参数必须与 QM2 上的监听器设置对应。Sender 通道与传输队列 QX 对应, 表示凡是在 QX 中等待发送的消息最终都可以由该通道送出。双方通道必须同名。

在连接通道的时候, 我们只需在 QM1 端启动通道 *start channel (C)*。

QM1 :

- - - -

```
DEFINE      QREMOTE      (QR)      +
            RNAME        (QL)      +
            RQMNAME      (QM2)     +
            XMITQ        (QX)      +
            REPLACE
DEFINE      QLOCAL      (QX)      +
            USAGE        (XMITQ)   +
            REPLACE
DEFINE      CHANNEL      (C)      +
            CHLTYPE      (SDR)     +
            TRPTYPE      (TCP)     +
            CONNAME      ('127.0.0.1 (1416)') +
            XMITQ        (QX)      +
            REPLACE
```

```
QM2 :
- - - -
DEFINE      QLOCAL          (QL)                +
            REPLACE
DEFINE      CHANNEL          (C)                +
            CHLTYPE          (RCVR)              +
            TRPTYPE          (TCP)              +
            REPLACE
```

```
start runmqtsr -m QM2 -t tcp -p 1416
runmqsc QM1
    start channel (C)
amqspout QR QM1
```

4.2.2 Server (QM1) — Receiver (QM2)

Server/Receiver 与 Sender/Receiver 类似。Server 端是消息的发送方 ,也是连接的发起方。在配置的时候 ,必须指定对方的通信参数 ,由 CONNAME 设定。下例中 ,QM1 是消息的发送方 ,QM2 是消息的接收方。

```
QM1 :
- - - -
DEFINE      QREMOTE          (QR)                +
            RNAME            (QL)                +
            RQMNAME          (QM2)              +
            XMITQ            (QX)               +
            REPLACE
DEFINE      QLOCAL           (QX)                +
            USAGE            (XMITQ)            +
            REPLACE
DEFINE      CHANNEL          (C)                +
            CHLTYPE          (SVR)              +
            TRPTYPE          (TCP)              +
            CONNAME          ('127.0.0.1 (1416)') +
            XMITQ            (QX)               +
            REPLACE
```

```
QM2 :
- - - -
DEFINE      QLOCAL          (QL)                +
            REPLACE
DEFINE      CHANNEL          (C)                +
            CHLTYPE          (RCVR)              +
            TRPTYPE          (TCP)              +
            REPLACE
```

```
start runmqtsr -m QM2 -t tcp -p 1416
runmqsc QM1
```

```
start channel (C)
amqsput QR QM1
```

4.2.3 Server (QM1) — Requester (QM2)

Server/Requester 通道也是一种较常见的通道配置方式 ,从消息流向来看 ,Server 作为消息的发送方 ,Requester 作为消息的接收方。但是从连接方式来看 ,Requester 却是连接的主动方 ,Server 是被动方。这种模式常用于动态 IP 地址的环境中 ,Server 是静态 IP 地址的服务器 ,Requester 的机器上网后自动分配到一个 IP 地址 ,所以是动态的 ,由 Requester 发起连接后接收数据。在本例中 ,由 QM2 (Requester) 启动通道 *start channel (C)*。

```
QM1 :
- - - -
DEFINE      QREMOTE      (QR)          +
           RNAME         (QL)          +
           RQMNAME       (QM2)         +
           XMITQ          (QX)          +
           REPLACE
DEFINE      QLOCAL       (QX)          +
           USAGE         (XMITQ)       +
           REPLACE
DEFINE      CHANNEL      (C)           +
           CHLTYPE       (SVR)         +
           TRPTYPE       (TCP)         +
           XMITQ          (QX)          +
           REPLACE

QM2 :
- - - -
DEFINE      QLOCAL       (QL)          +
           REPLACE
DEFINE      CHANNEL      (C)           +
           CHLTYPE       (RQSTR)       +
           TRPTYPE       (TCP)         +
           CONNAME       ('127.0.0.1 (1415)') +
           REPLACE
```

```
start runmqlsr -m QM1 -t tcp -p 1415
runmqsc QM2
start channel (C)
amqsput QR QM1
```

4.2.4 Sender (QM1) — Requester (QM2)

Sender/Requester 的连接过程稍微复杂一些 ,Requester 首先与 Sender 连接 ,在通知对方连接参数后连接断开。Sender 进行反向连接 ,消息也是反向传送的 ,即由 Sender 传给 Requester。这种反向连接的方式 ,称为回调连接 (Callback Connection)。这种模式也常用于做双向验证 ,即双方必须都要知道对方的通信参数才能完成回调连接。

```
QM1 :
- - - -
DEFINE      QREMOTE      (QR)          +
            RNAME        (QL)          +
            RQMNAME      (QM2)         +
            XMITQ        (QX)          +
            REPLACE
DEFINE      QLOCAL      (QX)          +
            USAGE        (XMITQ)       +
            REPLACE
DEFINE      CHANNEL      (C)          +
            CHLTYPE      (SDR)         +
            TRPTYPE      (TCP)         +
            CONNAME      ('127.0.0.1 (1416)') +
            XMITQ        (QX)          +
            REPLACE

QM2 :
- - - -
DEFINE      QLOCAL      (QL)          +
            REPLACE
DEFINE      CHANNEL      (C)          +
            CHLTYPE      (RQSTR)       +
            TRPTYPE      (TCP)         +
            CONNAME      ('127.0.0.1 (1415)') +
            REPLACE
```

```
start runmqlsr -m QM1 -t tcp -p 1415
start runmqlsr -m QM2 -t tcp -p 1416
runmqsc QM2
    start channel (C)
amqspout QR QM1
```

4.2.5 通道启动命令

通道的启动可以通过 MQSC 的 start channel 命令完成 ,也可以在命令行通过 runmqchl 控制命令完成 ,两者效果相同 ,在 Windows 中还可以用 MQ 服务配置成自动启动方式。

4.2.5.1 runmqchl

runmqchl 在通道连接的主动方使用 ,使用时需要指定队列管理器名和通道名。

- 格式
runmqchl [-m QMgrName] -c ChannelName
- 功能
启动通道
- 说明
选项 -m QMgrName 表示队列管理器名 缺省为缺省队列管理器。 -c ChannelName 表示通道

名。

- 例：

```
runmqchl -m QM -c Channel
```

4.2.5.2 用 MQ 服务配置通道启动

在 WebSphere MQ for Windows 产品中,可以通过 WebSphere MQ 服务(本地)工具来进行配置。在配置界面中右键点击启动了的管理队列管理器,新建→通道→指定启动通道名,会在注册表中配置相应的条目,如下所示。一个队列管理器可以配置多个通道启动命令,甚至为每一个通道配置一个启动命令,并全部配置成“自动启动”方式,这样队列管理器在启动后,所有的通道自动连通。

<MQ_HEKY> \Configuration\Services\ <QMName> \通道 - <ChannelName>

4.2.6 通道监控程序

无论是用 MQSC 中 start channel 脚本命令还是用 runmqchl 的控制命令来启动通道,都需要输入命令完成。通道的启动也可以用通道监控程序自动触发完成,通道监控程序实际上是触发程序的一种特例,与普通队列触发程序的区别在于通道启动程序是监听传输队列的,一旦传输队列中有消息满足传输条件则自动激活通道并进行消息传送。

4.2.6.1 runmqchi

一个队列管理器可以配置多个通道启动程序,以监控不同的通道。可以用控制命令 runmqchi 启动通道监控,启动时需要指定队列管理器和初始化队列名并在传输通道上配置触发功能(Trigger)。

- 格式

```
runmqchi [ -m QMgrName ] [ -q InitQueueName ]
```

- 功能

启动通道监控程序。

- 说明

选项 -m QMgrName 表示队列管理器名,缺省为缺省队列管理器。-q InitQueueName 表示初始化队列名,缺省为 SYSTEM.DEFAULT.INITIATION.QUEUE。

- 例：

```
runmqchi -m QM -q InitQueue
```

4.2.6.2 用 MQ 服务配置通道监控程序

在 WebSphere MQ for Windows 产品中,可以通过 WebSphere MQ 服务(本地)工具来进行配置。在配置界面中右键点击启动了的管理队列管理器,新建→通道启动程序→指定启动队列名,会在注册表中配置相应的条目：

<MQ_HKEY> \Configuration\Services\ <QMName> \启动通道程序

4.3 通道的属性

通道本质上是通信双方建立起来的通信连接,双方各有一个通信的工作进程(MCA),一对 MCA 之间可以配置通信协议及相应的参数。通道将双方的通信消息打包后传送,同时通过约定的机制来确保消息传递的正确性,这种机制称为 MQ 消息协议(MQ Message Protocol)。通道中消息的传递是单向的,但消息协议却是双向的,通信双方会对消息的发送和接收进行确认。

通道会对每一条传递的消息编号,可以进行批量发送,同时会监测对方是否反应正常。在通信连接断开后会自动恢复现场并试图重连。通道的所有这些功能都是通过通道属性的设置来完成的,通信双方在

连接的时候会交换各自通道的属性并约定双方都能接受的值。通常情况下我们在改动双方通道属性设置后需要重新连接,以使设置生效。

通道的各种功能是可以通道的属性来设置和调节的,下面我们会详细介绍通道的功能与属性之间的关系。

4.3.1 通道会话

4.3.1.1 消息序号 (Message Sequence)

通道为每一条消息的传送分配了一个序列号,它会自动累计增值。通道的相关属性 SEQWRAP 表示序号的最大值,缺省为 999 999 999。序列号越界后自动归零,从头开始。

通道两端的消息序号应该是相同的,一旦连接断开后,通道重连时双方 MCA 会将消息序号同步。如果通信异常造成序号不一致,可以在通道发送端用 MQSC 命令 RESET CHANNEL SEQNUM 手工将两者同步。

4.3.1.2 批量 (Batch)

在两个队列管理器 A 和 B 之间的通道上传送消息时,在消息从 A 送达 B 后,会有应答数据从 B 返回 A 表示收到确认。WebSphere MQ 把这个过程叫做一次同步 (SyncPoint)。如果我们用网络工具侦听通道连接,就可以观察到同一个连接上的这个返回同步数据包。

在 WebSphere MQ 中消息可以配置成批量传送,这时的一个同步点可以对一批消息的到达进行确认,以提高传输效率。为了控制两个同步点之间的时间和传送的消息数量,WebSphere MQ 引入相应的属性进行控制,如表 4-1。其中,批量间隔 (Batch Interval) 表示从本批次第一个消息到下一个同步点之间的最长的时间间隔,批量大小 (Batch Size) 表示一个批次即两个同步点之间最大的消息数量。两个条件只要满足任意一个就会引发一个同步点,即一个批次结束。

表 4-1 批量相关的通道属性

属性描述	属性名	缺省值	单位	有效范围	说明
Batch Interval	BATCHINT	0	毫秒	0 - 999 999 999	0 表示不使用 Batch,即消息送出后立即提交
Batch Size	BATCHSZ	50	条	1 - 9 999	累计到该值后批量传送

批量并不会影响消息传送,消息总是在一旦准备好就送到远端,不会有任何故意的耽搁,批量只会影响同步的时间。一个大的批量设置 (Batch Interval 和 Batch Size 都比较大) 会增加吞吐量,减少需要同步的数据包。但是,这也会增加响应时间,因为消息到达对方后处于未同步提交 (uncommitted) 的状态时间加长了,所以程序等待消息的时间加长了。在网络出错时,回滚的数据量也相对加大。

Batch Size 是通道建立连接的时候取双方通道属性 Batch Size 较小的那一个,也有些平台的 MQ 取双方通道的 Batch Size 以及队列管理器的属性 MAXUMSGS 取四个值中最小的那一个。实际的批量大小因为有了 Batch Interval 的参与,会小于等于这个值。

在稳定的连接上频繁地传送消息,则适当增大 Batch Size 会增加性能。如果消息的传送请求是断断续续的,则 Batch Size = 1 比较合适。通道属性中有 NPMSPEED 可以设置为 NORMAL 或 FAST。当 NPM-SPEED = FAST 时,我们称之为快速通道 (FastPath Channel)。在快速通道上的非持久消息 (Non-Persistent) 的发送是不受批量控制的,它本身也不会引发确认信息包,即没有同步过程,但它们在批量 (Batch) 计数的时候却是计算其中的。

4.3.1.3 心跳 (HeartBeat)

由于在 WebSphere MQ 中通道是单向传递数据的,消息永远从发送端到接收端。发送端能够反向接到

的是一些控制信息。接收端打开通道和相应的队列可能会占用一定的资源 ,如果长时间没有收到任何数据 ,则有效的办法是让接收端收到一个信号 ,表示可以暂关闭通道 ,释放资源。这个信号就是发送端发出的心跳信号。

另外 ,发送端在长时间空闲的情况下 ,也可以选择保持连接或自动断连。WebSphere MQ 通道有相应的属性可以设置 ,如表 4 - 2。

表 4 - 2 心跳相关的通道属性

属性描述	属性名	缺省值	单位	有效范围	说明
Batch HeartBeat Interval	BATCHHB	0	毫秒	0 ~ 999 999	0 表示不使用
HeartBeat Interval	HBINT	300	秒	0 ~ 999 999	0 表示不使用 ,即不发送心跳信号
Disconnect Interval	DISCINT	6000	秒	0 ~ 999 999 0 ~ 9 999	除 z/OS with CICS 外的所有平台 z/OS with CICS
KeepAlive Interval	KAINT	AUTO	秒	0 ~ 99 999	0 表示不使用 ,仅对 z/OS 平台有效

Batch HeartBeat Interval 允许通道发送端对发送的消息批次进行提交之前检测接收端是否还活着。如果接收端不活 ,发送端回滚。如果在本批次内 (Batch Interval) 发送端收到过来自接收端的确认 ,则认为接收端还活着。否则 ,发送端主动发送心跳信号 (HeartBeat) 给接收端并要求心跳回应。此属性只对 Sender、Server、Cluster - Sender 和 Cluster - Receiver 通道有效。

HeartBeat Interval 表示两次心跳之间的间隔时间。在发送端如果一定时间内没有新的消息可以传送 ,则发送端主动发送一个心跳信号 ,接收端在接收这个信号之后 ,可以关闭通道 ,而不必死等新的消息或等到 Disconnect Interval 时间超时。HeartBeat Interval 通常应该比 Disconnect Interval 小得多 ,接收端在收到心跳信号后可以释放大消息准备的内存 ,也可以关闭那些仍打开着的队列。

Disconnect Interval 是 Sender、Cluster - Sender、Server 和 Cluster - Receiver 通道的属性。如果一个批次结束后 (Batch Interval 或 Batch Size 条件满足) 在一定的时间内 (即两个批次之间的最长时间间隔) 没有新的消息可以传送 ,那么通道会自动关闭。这个时间就是 Disconnect Interval。在通道关闭的时候 ,发送端和接收端会交换控制数据指明关闭原因 ,以保证下一次的成功打开。Disconnect Interval 设置得太长 ,则对方的资源 (比如打开的队列) 会长时间挂住。设置得太短 ,则通道会频繁地被打开和关闭。通常可以设置另一个属性 HeartBeat Interval ,这个属性值应该比 Disconnect Interval 小得多 ,通道发送端在一定间隔时间发送心跳数据 ,接收端收到 HeartBeat Interval 后就释放资源。

KeepAlive Interval (KAINT) 表示定时发送的保持连接信号的间隔时间。如果在 MQ 配置文件 (mq.ini) 中的 TCP stanza 段中设置有 KEEPALIVE = YES ,尽管 KeepAlive Interval (KAINT) 的值为 0 ,TCP 链路上的 KeepAlive 信号仍然会自动发送。如果 KAJNT = AUTO ,则 KAJNT 的值与 HBINT 有关 :

- 如果通道连接时约定的 HBINT > 0 ,则 KAJNT = HBINT + 60 秒
- 如果通道连接时约定的 HBINT = 0 ,则 KAJNT = 0

4.3.1.4 连接重试 (Connection Retry)

通道 (Channel) 在暂时无法接通时会试图重试 ,重试方式分为短等待和长等待两种方式 ,每一种方式都会以一定的时间间隔重试多次。属性如表 4 - 3 ,指明了发送方与接收方试图建立连接 (Session) 的重试次数与间隔。

表 4 - 3 连接重试相关的通道属性

属性描述	属性名	缺省值	单位	有效范围	说明
Short Retry Count	SHORTRTY	10	次	0 ~ 999 999 999	短等待次数
Short Retry Interval	SHORTTMR	60	秒	0 ~ 999 999 999	短等待间隔
Long Retry Count	LONGRTY	999 999 999	次	0 ~ 999 999 999	长等待次数
Long Retry Interval	LONGTMR	1200	秒	0 ~ 999 999 999	长等待间隔

通道首先用短等待方式进行连接重试 ,每次等待 SHORTTMR 时长 ,共 SHORTRTY 次。在短等待皆告失败后 ,用长等待方式进行连接重试 ,每次等待 LONGTMR 时长 ,共 LONGRTY 次。所以 ,重试 (Retry) 的总时长为 : (SHORTRTY * SHORTTMR) + (LONGRTY * LONGTMR)。在重试期间 ,通道处于重试 (Retrying) 状态 ,在所有的重试工作皆告失败后 ,通道处于关闭 (Closed) 状态。

4.3.2 通道协议

4.3.2.1 MCA

通道协议决定了消息通道代理 (Message Channel Agent — MCA) 的工作方式。我们可以把 MCA 简单地理解为通信程序 ,它可以以独立的进程方式运行 ,也可以以线程的方式嵌入其他进程中运行。与 MCA 相关的通道属性如表 4 - 4。

表 4 - 4 MCA 相关的通道属性

属性描述	属性名	缺省值	可取值	说明
MCA Name	MCANAME	空	字符串	保留
MCA Type	MCATYPE	PROCESS	PROCESS THREAD	
MCA User	MCAUSER	空	字符串	

MCANAME 表示消息通道代理名 ,是一个保留属性 ,在实际环境中并不起作用。

MCATYPE 表示通道代理类型 ,可以取值 PROCESS 或 THREAD ,分别表示发送端通信程序 MCA 以独立的 RUNMQCHL 进程方式执行还是在 RUNMQCHI 中以线程方式执行。对于接收端的通道代理类型是由启动监听的方式决定的 ,即是用 inetd 还是 RUNMQLSR。

MCAUSER 表示 MCA 访问资源时的用户名 ,如果为空 ,则使用缺省用户。在 Windows 中 ,可以用 user@ domain 的方式指定域用户。

通道上的通信协议可以是 TCP/IP、SNA/LU62、DCENET、NETBIOS、SPX 和 UDP 等。以下选取使用比较普遍的 TCP/IP 和 SNA/LU62 两种通信协议进行说明。

4.3.2.2 TCP/IP

通道定义中与 TCP/IP 相关的属性有三个 TRPTYPE、CONNAME 和 LOCLADDR ,如表 4 - 5。

表 4 - 5 TCP/IP 相关的通道属性

属性描述	属性名	说明
Transport Type	TRPTYPE	设置为 TCP
Local Address	LOCLADDR	本地的 IP + Port
Connection Name	CONNAME	对方的 IP + Port

TRPTYPE 表示通信协议 ,设置为 TCP。LOCLADDR 表示本地通信参数 ,格式为 LOCLADDR ([ip - addr][(low - port [, high - port])]) ,长度为 48 字节。其中 ip - addr 是本地的 IP 地址 ,low - port 和 high - port 划定了一段 port 的有效范围。在建立连接时 ,WebSphere MQ 会根据实际情况选择一个合适的 Port。这种方式对于有防火网的网络特别合适。

CONNAME 表示对方的通信参数 ,格式与 LOCLADDR 相同 ,常见的设置为 CONNAME (IP [(Port)]) ,其中 IP 地址也可以由机器名代替。该属性指定远端的 IP 地址和端口。如果没有指定端口 ,则根据远端的同名通道定义来决定 ,缺省端口值为 1414。

4.3.2.3 SNA/LU62

通道定义中与 SNA 和 LU62 相关的属性有 :USERID、PASSWORD、TRPTYPE、MODENAME、TPNAME、CONNAME 如表 4 – 6。

表 4 – 6 SNA/LU62 相关的通道属性

属性描述	属性名	说明
Transport Type	TRPTYPE	设置为 LU62
User ID	USERID	MCA 使用的启动 SNA Session 时的用户名
Password	PASSWORD	MCA 使用的启动 SNA Session 时的口令
LU62 Mode Name	MODENAME	LU62 连接时的附加信息
LU62 Transaction Program Name	TPNAME	LU62 连接时调用的交易名
Connection Name	CONNAME	LU62 连接对象名

对于 SNA 通信协议属性 TRPTYPE 应该设置为 LU62 ,由 MODENAME 指明 LU62 连接时的附加信息。如果用做提供 SNA 连接侧面信息 ,则 MODENAME 应该在 CPI – C 或 APPC 环境中定义 ,该属性值为空。TPNAME 指定远端的 MCA 交易名。CONNAME 指定远端连接的对象名 ,如果 TPNAME 和 MODENAME 不为空 ,则该属性指定 LU 全名。否则 ,如果 TPNAME 和 MODENAME 为空 ,则该属性为 CPI – C 对象名。

另外 ,SNA 协议在连接时需要提供连接的用户名和口令 ,分别由 USERID 和这 PASSWORD 来设定。

4.4 通道的状态

通道状态分成三类状态属性 SAVED、CURRENT 和 SHORT。其中 ,SHORT 只适用于 z/OS 平台 ,这里不讨论。SAVED 是 CURRENT 的子集 ,这些属性在建立连接时设置 ,此后不变 ,一直到重建连接时被重新设置。CURRENT 中的状态属性则在连接使用的过程中时时刻刻发生着变化 ,反映当前的通道状态。通常来说 ,我们只对 CURRENT 感兴趣。CURRENT 又分两部分 :公共状态 (Common Status) 和当前状态 (Current-Only Status)。

运行中的通道状态可以用 MQSC 命令 `DISPLAY CHSTATUS () ALL` 来显示 ,也可以缩写为 `DIS CHS () ALL`。

下面让我们先来了解一下通道的公共状态和当前状态分别包含哪些属性。

4.4.1 公共状态 (Common Status)

公共状态中包含了通道中变化最快的一些状态属性 ,通常以消息或批次为单位实时更新 ,能反映通道目前的工作状态 ,如表 4 – 7。我们可以通过观察公共状态得知通道当前的批次号、消息的序列号、传递的消息数 ,等等。

表 4 – 7 通道 Common 状态属性

属性	说明
CURLUWID	当前 Batch 的 LUW ID 如果是发送端处于 In-Doubt 状态 ,则它是 In-Doubt Batch 的 LUW ID
CURMSGs	对于发送端 ,它是当前 Batch 中已发送的消息数 ,每发送一条消息自动加一。如果是发送端处于 In-Doubt 状态 ,则它是 In-Doubt Batch 中已发送的消息数 对于接收端 ,它是当前 Batch 中已接收的消息数 ,每接收一条消息自动加一 无论是发送端或是接收端 ,Batch 提交时自动归零

续表

属性	说明
CURSEQNO	对于发送端 ,它是上一条已发送消息的序列号 ,每发送一条消息自动加一。如果是发送端处于 In-Doubt 状态 ,则它是 In-Doubt Batch 中上一条已发送消息的序列号 对于接收端 ,它是上一条已接收消息的序列号
INDOUBT	指定通道当前是否处于 In-Doubt 状态。 对于发送端 ,只有在 MCA 已成功发送消息且等待应答 (Acknowledgment) 的时候 ,为 YES ,其他时候皆为 NO 对于接收端 ,永远是 NO
LSTLUWID	上一个提交 Batch 的 LUW ID
LSTSEQNO	上一个提交 Batch 中最后一条消息的序列号。由于 NPMSPEED (FAST) 通道对于非持久性消息是不需要提交的 ,所以这种情况不会使 LSTSEQNO 自动累计
STATUS	通道的当前状态 ,可能是 : <ul style="list-style-type: none">● STARTING● BINDING● INITIALIZING● RUNNING● STOPPING● RETRYING● PAUSED● STOPPED● REQUESTING

注意 对于 Inactive Channel ,CURMSGs、CURSEQNO、CURLUWID 只有在通道处于不确定 (In-Doubt) 状态时值才有意义。

4.4.2 当前状态 (Current-Only Status)

当前状态包含了通道本次连接时约定的通信参数及连接以来的一些统计信息 ,如表 4 – 8。观察当前状态可以得知通道的数据流量 ,已经完成传递的批次数、消息数、字节数 ,等等。

表 4 – 8 通道 Current-Only 状态属性

属性	说明
BATCHES	在这个 Session 中 (通道启动以来) 完成的 Batch 数量
BATCHSZ	在这个 Session 中 (通道启动时商定的) Batch 大小
BUFSRCVD	在这个 Session 中 (通道启动以来) 收到的消息包数量 ,包括控制信息
BUFSENT	在这个 Session 中 (通道启动以来) 发出的消息包数量 ,包括控制信息
BYTSRCVD	在这个 Session 中 (通道启动以来) 收到的字节总量 ,包括控制信息
BYTSENT	在这个 Session 中 (通道启动以来) 发出的字节总量 ,包括控制信息

属性	说明
CHSTADA	通道启动日期 (yyyy - mm - dd)
CHSTATI	通道启动时间 (hh. mm. ss)
HBINT	在这个 Session 中 (通道启动时商定的) Heartbeat Interval 大小
JOBNAME	当前通道进程的作业名 ,通常是十六进制 <ul style="list-style-type: none">● Compaq OpenVMS PID (HEX)● OS/2 , OS/400 , UNIX , Windows PID + TID (HEX)● NSK CPU ID + PID (HEX)
LOCLADDR	本地地址 ,IP 和 Port
LONGRTS	Long Retry 还剩下的次数
SHORTRTS	Short Retry 还剩下的次数
LSTMSGDA	上一条消息发送 ,或者上一个 MQI call 处理的日期
LSTMSGTI	上一条消息发送 ,或者上一个 MQI call 处理的时间 <ul style="list-style-type: none">● 对于 Sender 或 Server ,这是上一条消息 (如果自动分裂 ,指的是最后一部分) 发送的时间● 对于 Requester 或 Receiver ,这是上一条消息完整放入目标队列的时间● 对于 Server-Connection Channel ,这是上一个 MQI call 处理的时间
MAXMSGL	在这个 Session 中 (通道启动时商定的) 最大消息长度 (z/OS only)
MCASTAT	MCA 当前的状态。可以是“ running ”或“ not running ”。对于一个通道 ,它的状态 (STATUS 域) 是 STOPPED ,然而 MCA 程序却可能依然活着。
MSGS	在这个 Session 中 (通道启动以来) 发送或接收的消息数量。对于 Server-Connection Channel ,这是 MQI call 的数量
NPMSPEED	在这个 Session 中 (通道启动时商定的) NPMSPEED 方式
RQMNAME	远程队列管理器名
SSLPEER	通道另一端的 Peer Queue Manager 或 Client 的 Distinguished Name。最长 256 字节
STOPREQ	是否有突出的用户停止请求。值可以是 YES 或 NO。

4.4.3 通道状态分析

由于通道状态属性实时地反映了通道的工作状态 ,在实际使用中经常被用来检验消息是否完整地送达、通道是否畅通及帮助定位并诊断问题。以下我们用两个队列管理器 QM1 和 QM2 之间的一条单向通道上的通道状态为例 ,分析通道流量与状态的关系。为了方便比较 ,我们把这一过程中通道两端的属性变化记录在表 4 - 9 中。

表 4-9 流量与通道的状态变化

QM1	QM2
1. 开始 用 Start Channel 命令建立连接	
CURSEQNO(0) CURLUWID(2D92B43E10000301) MSGs(0) BATCHES(0) BYTSSSENT(132) BYTSRCVD(132) BUFSSENT(1) BUFSRCVD(1) JOBNAME(00000E9800000428) LSTMSGTI() LSTMSGDA() CHSTATI(15.55.38) CHSTADA(2003-05-04) LSTSEQNO(0) LSTLUWID(0000000000000000) CURMSGs(0)	CURSEQNO(0) CURLUWID(0000000000000000) MSGs(0) BATCHES(0) BYTSSSENT(132) BYTSRCVD(132) BUFSSENT(1) BUFSRCVD(1) JOBNAME(00000BDC00000FE4) LSTMSGTI() LSTMSGDA() CHSTATI(15.55.38) CHSTADA(2003-05-04) LSTSEQNO(0) LSTLUWID(0000000000000000) CURMSGs(0)
2. 接着 用 amqspout 命令发送一条消息 (1 Byte)	
CURSEQNO(1) CURLUWID(2D92B43E10000302) MSGs(1) BATCHES(1) BYTSSSENT(637) BYTSRCVD(160) BUFSSENT(3) BUFSRCVD(2) JOBNAME(00000E9800000428) LSTMSGTI(15.55.56) LSTMSGDA(2003-05-04) CHSTATI(15.55.38) CHSTADA(2003-05-04) LSTSEQNO(0) LSTLUWID(0000000000000000) CURMSGs(0)	CURSEQNO(1) CURLUWID(2D92B43E10000301) MSGs(1) BATCHES(1) BYTSSSENT(160) BYTSRCVD(637) BUFSSENT(2) BUFSRCVD(3) JOBNAME(00000BDC00000FE4) LSTMSGTI(15.55.56) LSTMSGDA(2003-05-04) CHSTATI(15.55.38) CHSTADA(2003-05-04) LSTSEQNO(0) LSTLUWID(0000000000000000) CURMSGs(0)
3. 然后 用 amqspout 命令再发送一条消息 (1 Byte)	
CURSEQNO(2) CURLUWID(2D92B43E10000303) MSGs(2) BATCHES(2) BYTSSSENT(1142) BYTSRCVD(188) BUFSSENT(5) BUFSRCVD(3) JOBNAME(00000E9800000428) LSTMSGTI(15.56.03) LSTMSGDA(2003-05-04) CHSTATI(15.55.38) CHSTADA(2003-05-04) LSTSEQNO(0) LSTLUWID(0000000000000000) CURMSGs(0)	CURSEQNO(2) CURLUWID(2D92B43E10000302) MSGs(2) BATCHES(2) BYTSSSENT(188) BYTSRCVD(1142) BUFSSENT(3) BUFSRCVD(5) JOBNAME(00000BDC00000FE4) LSTMSGTI(15.56.03) LSTMSGDA(2003-05-04) CHSTATI(15.55.38) CHSTADA(2003-05-04) LSTSEQNO(0) LSTLUWID(0000000000000000) CURMSGs(0)

从以上通道的状态变化 ,我们可观察到 ,队列管理器 QM1 在两次消息之间共发送了 505 个字节 (BYTSENT 的差值) ,是分两个数据包发送的 (BUFSENT 的差值)。其中一个消息是 MQ 通道定时自动发送的同步信息包 (28 字节) ,另一个消息包是数据包本身 ,它含有 MQTSH (48 字节) + MQXQH (428 字节) + Message Data (1 字节) 结构如图 4 - 1 ,共 477 字节。两者相加 ,恰好 505 字节。

QM2 是消息接收方 ,但从通道的状态变化中观察到它也有数据包发出 ,每次 28 字节 ,实际上 ,这就是同步信息包 ,用于对批次消息的确认。

另外 ,对于 Windows 和 UNIX ,JOBNAME 是由十六进制的 PID (Process ID) + TID (Thread ID) 合成的。从上面的数据中可以观察到 :(Windows)

- QM1 发送端
JOBNAME(00000E9800000428) PID = 3736 ,TID = 1064 Process = runmqchl
- QM2 接收端
JOBNAME(00000BDC00000FE4) PID = 3036 ,TID = 4068 Process = amqrmppa

MQTSH :(48 Bytes) //MQ 内部结构 StrucId = [TSH] StrucLength = AgentBufferLength ...
MQXQH :(428 Bytes) StrucId = [XQH] ...
Message : ...

图 4 - 1 网络上传送的消息结构

4.5 互连配置举例

在实际工作中 ,我们往往需要构建复杂的互连结构 ,如果将其分解成简单的基本结构后 ,会发现这种基本结构其实是有限的。下面我们举例介绍一些常用的基本结构。

4.5.1 单向传送

QM1 要向 QM2 发送消息 ,双方采用 Sender/Receiver 方式。其中 ,QM1 上要定义远程队列 QR _QM2和传输队列 QLX _QM2 ,QM2 上要定义本地队列 QL _QM2 ,如图 4 - 2。

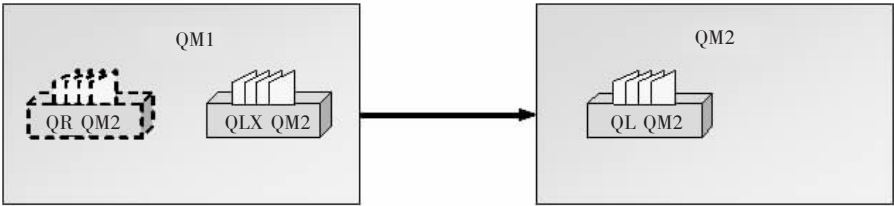


图 4 - 2 单向传递

QM1 上的传输队列可以是对方队列管理器的名字 QM2 ,在远程队列 QR _QM2 的定义中如果不指定 XMITQ 参数 ,则系统会自动用 RQMNAME 参数去寻找同名传输队列 ,即 QM2。如果再找不到 ,会试图根据 QM1 的缺省传输队列参数 DEF XMITQ 再次寻找传输队列。所以 ,在 WebSphere MQ 设置时 ,通常建议设置队列管理器的缺省传输队列 DEF XMITQ ,同时建议传输队列与对方队列管理器同名。

QM1 :
- - - -

```
DEFINE QREMOTE (QR_QM2) +
RNAME (QL_QM2) +
RQMNAME (QM2) +
XMITQ (QLX_QM2) +
REPLACE
DEFINE QLOCAL (QLX_QM2) +
USAGE (XMITQ) +
REPLACE
DEFINE CHANNEL (C_QM1.QM2) +
CHLTYPE (SDR) +
TRPTYPE (TCP) +
CONNAME ('127.0.0.1 (1416)') +
XMITQ (QLX_QM2) +
REPLACE

QM2 :
- - - -
DEFINE QLOCAL (QL_QM2) +
REPLACE
DEFINE CHANNEL (C_QM1.QM2) +
CHLTYPE (RCVR) +
TRPTYPE (TCP) +
REPLACE
```

4.5.2 双向传送

QM1 和 QM2 要互相发送消息 ,双方采用 Sender/Receiver 方式。与上例类似 ,只是需要反向定义一套配置 ,如图 4 - 3。

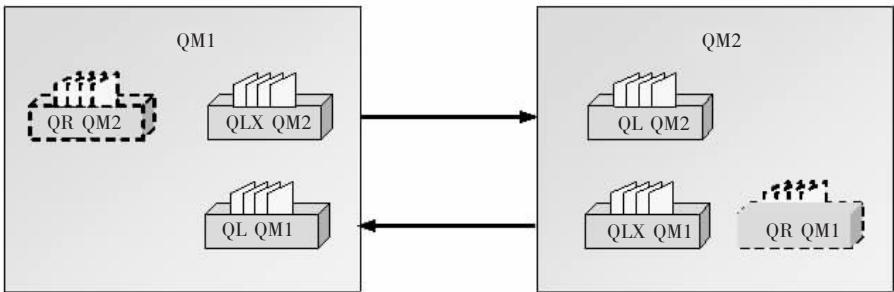


图 4 - 3 双向传递

```
QM1 :
- - - -
DEFINE CHANNEL (C_QM1.QM2) +
CHLTYPE (SDR) +
TRPTYPE (TCP) +
CONNAME ('127.0.0.1 (1415)') +
XMITQ (QM2) +
REPLACE
DEFINE CHANNEL (C_QM2.QM1) +
```

```
CHLTYPE      (RCVR)      +
TRPTYPE      (TCP)       +
REPLACE
DEFINE QLOCAL (QM2)      +
USAGE        (XMITQ)     +
REPLACE

QM2 :
- - - -
DEFINE CHANNEL (C_QM1.QM2) +
CHLTYPE      (RCVR)      +
TRPTYPE      (TCP)       +
REPLACE
DEFINE CHANNEL (C_QM2.QM1) +
CHLTYPE      (SDR)       +
TRPTYPE      (TCP)       +
CONNAME      ('127.0.0.1 (1416)') +
XMITQ        (QM1)       +
REPLACE
DEFINE QLOCAL (QM1)      +
USAGE        (XMITQ)     +
REPLACE
```

4.5.3 队列与队列管理器别名

QM1 要向 QM2 发送消息 ,双方采用 Sender/Receiver 方式。其中 ,QM1 上定义了远程队列 QR_QM2 指向 QM2A 队列管理器上的队列 QL_QM2A ,远程队列指定 QLX_QM2 传输队列 ,如图 4-4。

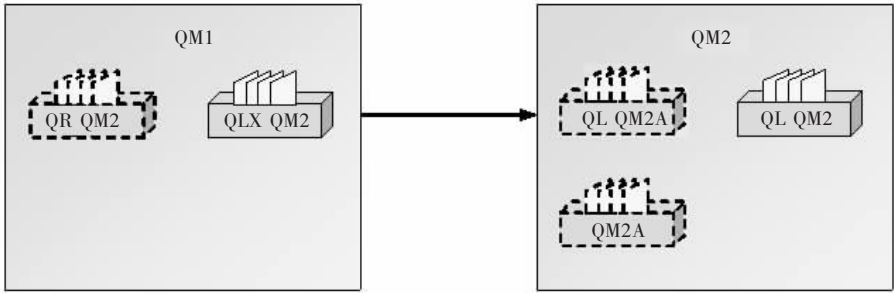


图 4-4 队列与队列管理器别名

放入远程队列 QR_QM2 的消息 ,由于远程队列本身指定了传输路由 (QLX_QM2) ,所以消息自动地转送到 QM2 上。到了 QM2 后 ,消息的目标队列管理器为 QM2A ,目标队列为 QL_QM2A ,由于在 QM2 上定义了队列管理器别名 (QM2A → QM2) ,也定义了别名队列 (QL_QM2A→ QL_QM2)。最终 ,消息放入正确的目标队列。如果在 QM2 上找不到目标队列 ,消息会再试图进一步路由。

队列管理器别名本质上是 RQNAME 为空的远程队列定义。

```
QM1 :
- - - -
DEFINE QREMOTE (QR_QM2)      +
RNAME        (QL_QM2A)     +
RQNAME       (QM2A)        +
```

	XMITQ	(QX_QM2)	+
	REPLACE		
DEFINE	QLOCAL	(QX_QM2)	+
	USAGE	(XMITQ)	+
	REPLACE		
DEFINE	CHANNEL	(C_QM1.QM2)	+
	CHLTYPE	(SDR)	+
	TRPTYPE	(TCP)	+
	CONNNAME	('127.0.0.1 (1415)')	+
	XMITQ	(QX_QM2)	+
	REPLACE		
QM2 :			
- - - -			
DEFINE	QLOCAL	(QL_QM2)	+
	REPLACE		
DEFINE	QALIAS	(QL_QM2A)	+
	TARGQ	(QL_QM2)	+
	REPLACE		
DEFINE	QREMOTE	(QM2A)	+
	RQMNAME	(QM2)	+
	REPLACE		
DEFINE	CHANNEL	(C_QM1.QM2)	+
	CHLTYPE	(RCVR)	+
	TRPTYPE	(TCP)	+
	REPLACE		

4.5.4 三级跳

QM1 要发送消息去 QM3 ,消息在 QM2 被自动转发 ,而不被应用消息处理 ,即通常所说的消息在 QM2 不落地 ,如图 4 - 5。

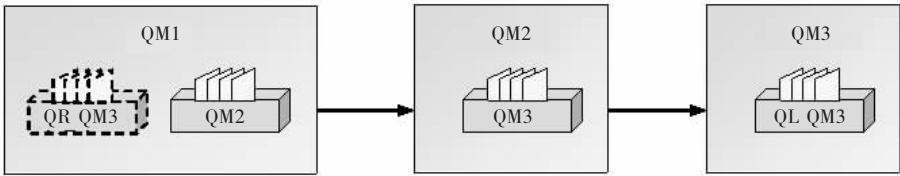


图 4 - 5 三级跳

QM1 上的远程队列 QR_QM3 定义了目标队列管理器为 QM3 ,目标队列为 QL_QM3。消息通过指定的传输队列 QM2 路由到 QM2 上后 ,试图寻找 QM3 ,结果找到名为 QM3 的传输队列 ,消息进一步路由到队列管理器 QM3 上 ,最终放入正确的目标队列。

如果 QM2 上的传输队列名与消息的目标队列管理器名不同 ,即不为 QM3。则可以通过设置 QM2 上的 DEFXMITQ 属性达到同样的效果。

QM1 :			
- - - -			
DEFINE	QREMOTE	(QR_QM3)	+

```

        RNAME          (QL _ QM3 )          +
        RQMNAME        (QM3 )              +
        XMITQ          (QM2 )              +
        REPLACE
DEFINE    QLOCAL      (QM2 )              +
        USAGE        (XMITQ )            +
        REPLACE
DEFINE    CHANNEL     (C _ QM1. QM2 )      +
        CHLTYPE      (SDR )              +
        TRPTYPE      (TCP )              +
        CONNAME      ('127. 0. 0. 1 (1415)') +
        XMITQ        (QM2 )              +
        REPLACE

QM2 :
- - - -
DEFINE    QLOCAL      (QM3 )              +
        USAGE        (XMITQ )            +
        REPLACE
DEFINE    CHANNEL     (C _ QM1. QM2 )      +
        CHLTYPE      (RCVR )             +
        TRPTYPE      (TCP )              +
        REPLACE
DEFINE    CHANNEL     (C _ QM2. QM3 )      +
        CHLTYPE      (SDR )              +
        TRPTYPE      (TCP )              +
        CONNAME      ('127. 0. 0. 1 (1416)') +
        XMITQ        (QM3 )              +
        REPLACE

QM3 :
- - - -
DEFINE    QLOCAL      (QL _ QM3 )          +
        REPLACE
DEFINE    CHANNEL     (C _ QM2. QM3 )      +
        CHLTYPE      (RCVR )             +
        TRPTYPE      (TCP )              +
        REPLACE
```

4.5.5 四级跳

QM1 要发送消息去 QM4 ,消息在 QM2 和 QM3 上被自动转发。本例可以看做上例的扩展 ,如图 4 - 6。消息在传递路径上不落地。

QM1 上的远程队列 QR _ QM4 定义了目标队列管理器为 QM4 ,目标队列为 QL _ QM4。消息放入 QR _ QM4 后 ,经过指定的传输队列 QM2 路由到 QM2。在 QM2 上消息发现这不是目的地 QM4 ,于是就找到名为

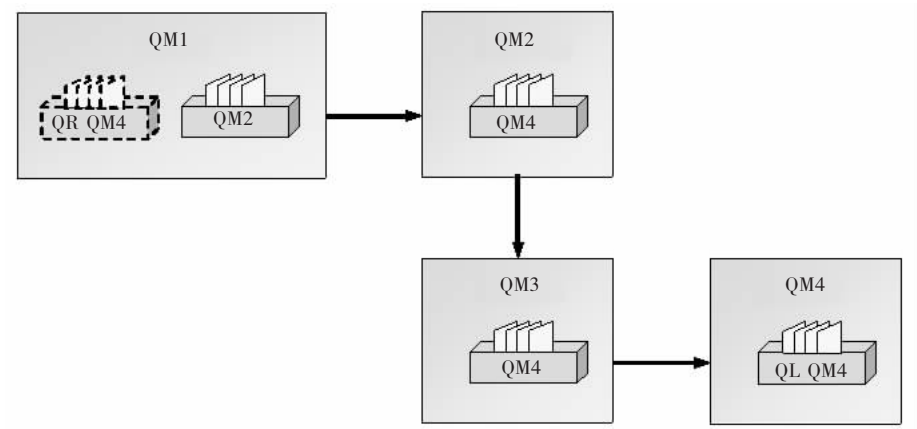


图 4-6 四级跳

QM4 的传输队列继续漫游。类似地 ,到了 QM3 后 ,消息又发现 QM3 仍然不是目的地 ,再次通过名为 QM4 的传输队列漫游。消息最终到达 QM4 ,且成功放入目标队列 QL _ QM4 中。

如果 QM2 和 QM3 上的传输队列名与消息的目标队列管理器名不同 ,即不为 QM4。则可以通过设置 QM2 和 QM3 上的 DEFXMITQ 属性达到同样的效果。

QM1 :

```
-----
DEFINE    QREMOTE      (QR _ QM4)                +
          RNAME        (QL _ QM4)                +
          RQMNAME      (QM4)                     +
          XMITQ        (QM2)                     +
          REPLACE
DEFINE    QLOCAL       (QM2)                     +
          USAGE        (XMITQ)                   +
          REPLACE
DEFINE    CHANNEL      (C _ QM1. QM2)             +
          CHLTYPE      (SDR)                     +
          TRPTYPE      (TCP)                     +
          CONNAME      ('127. 0. 0. 1 (1415)')    +
          XMITQ        (QM2)                     +
          REPLACE
```

QM2 :

```
-----
DEFINE    QLOCAL       (QM4)                     +
          USAGE        (XMITQ)                   +
          REPLACE
DEFINE    CHANNEL      (C _ QM1. QM2)             +
          CHLTYPE      (RCVR)                    +
          TRPTYPE      (TCP)                     +
          REPLACE
DEFINE    CHANNEL      (C _ QM2. QM3)             +
          CHLTYPE      (SDR)                     +
          TRPTYPE      (TCP)                     +
```

```

CONNAME      ('127.0.0.1 (1416)')      +
XMITQ        (QM4)                      +
REPLACE

QM3 :
- - - -
DEFINE QLOCAL (QM4)                      +
      USAGE   (XMITQ)                    +
      REPLACE
DEFINE CHANNEL (C_QM2.QM3)              +
      CHLTYPE (RCVR)                     +
      TRPTYPE (TCP)                       +
      REPLACE
DEFINE CHANNEL (C_QM3.QM4)              +
      CHLTYPE (SDR)                       +
      TRPTYPE (TCP)                       +
      CONNAME ('127.0.0.1 (1417)')       +
      XMITQ    (QM4)                      +
      REPLACE

QM4 :
- - - -
DEFINE QLOCAL (QL_QM4)                  +
      REPLACE
DEFINE CHANNEL (C_QM3.QM4)              +
      CHLTYPE (RCVR)                     +
      TRPTYPE (TCP)                       +
      REPLACE
```

第5章 应用设计

我们在前面的介绍中已经了解了 WebSphere MQ 的概念、原理、安装、管理、配置等内容,对 MQ 已经有了一个初步的了解,可以说在操作上已经没有问题了,但如果要基于 MQ 设计一个应用系统,则可能仍然觉得无所适从。下面我们要进一步介绍 MQ 应用的设计、实现的功能及相关技巧。

5.1 架构设计

5.1.1 两点间通信

WebSphere MQ 的基本功能是用来连接异构平台之间的应用,通过 MQ 的互连,使两个应用系统实现互连通信,如图 5-1。所以,对于已有的应用系统,如果没有通信接口则通常需要进行改造,嵌入 MQI 调用使消息进得来、出得去。

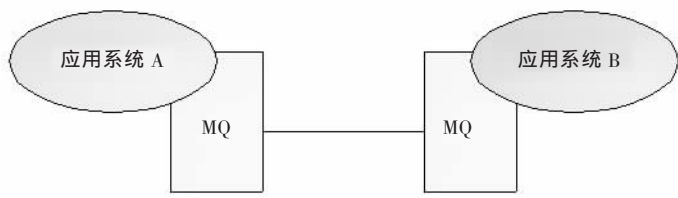


图 5-1 应用系统通过 MQ 实现互连

MQ 的通信方式是异步非实时的,也就是说应用系统 A 调用发送消息的 API,如果返回正常表示发送消息已经被 MQ 接受,但并不表示消息已经传达应用系统 B。首先,在传输的路途中如果发生各种异常情况,应用系统 B 有可能收不到该消息。其次,由于网络或其他的原因,应用系统 B 有可能不能马上收到该消息。在这种情况下,应用系统 A 无法知道消息是否已经成功地到达应用系统 B。所以,在需要有反馈的应用系统中,通常会设计反向应答机制。

应用系统 A 发出请求消息,然后等待对该请求消息的应答反馈,即请求消息。通常情况下,请求消息和应答消息应该放在不同的队列中,以免混淆。每一条 MQ 消息的头结构(MQMD)中包含了 MsgId 和 CorrelId 两个域,分别用于标识消息本身和相关消息。对于应答消息,它有不同于请求消息的 MsgId,但它的 CorrelId 应该等于请求消息的 MsgId,表示针对该请求消息的应答。在应用系统 A 等待应答消息的时候,可以用请求消息的 MsgId 作为匹配条件来明确地等待针对该请求消息的应答。

应用系统 B 在收到请求消息后,根据消息内容做出相应的动作,同时在构建应答消息的时候,应该将请求消息的 MsgId 填入应答消息的 CorrelId 域中。

当然,应用系统 A 也可以利用 MQ 消息的一些属性来设定一些特殊功能。比如有些消息是有时效性的,超过时间后即使送达对方也变得没有意义了,这时可以设置消息超时(MQMD 的 Expiry 域)。有些消息的寄送是需要回执的,在消息送达目标队列或消息被对方取出后由 MQ 自动返回一条报告消息。具体细节参见“报告”章节。

5.1.2 多点间通信

当应用场景中涉及需要互连的系统较多时,如果简单地将所有需要互连的系统之间用 MQ 直接连接起来,就会形成网状结构,如图 5-2 左。网状结构的好处在于从单个应用考虑逻辑简单,直截了当,且为每两个结点之间配置了独立的连接,是通信速度最快最直接的一种结构。由于各条连接的忙闲程度不同,需要传送的数据量不同,数据的重要性也不同,网状结构能最大限度地保证各种通信各行其道、互不干扰。然

而 ,随着结点的增多 ,网状结构会不可避免地出现接口众多而难以管理的局面。太多的连接会给配置、管理、编程都带来不小的麻烦。

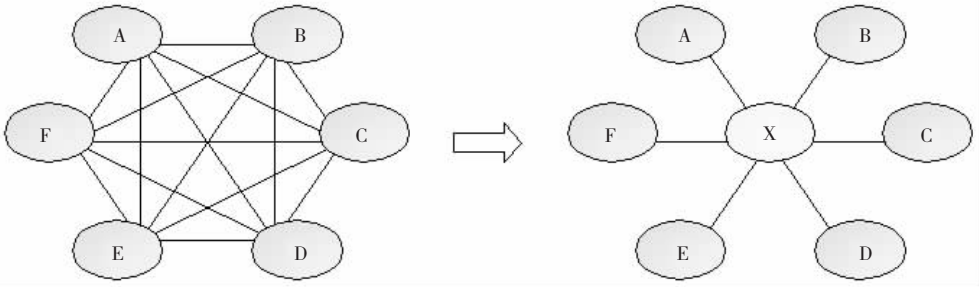


图 5 - 2 从网状结构到星状结构

人们很自然地会想到用星状结构来解决这一问题 ,如图 5 - 2 右。星状结构中有一个结点处于中心位置 ,扮演着数据交换中心的角色。所有的其他结点都与之相连 ,消息通过它转发到目标队列中。星状结构又称中心辐射结构 (Hub and Spoke) ,在众多系统互连时可以有效地降低拓扑结构的复杂度。星状结构有明显的优点 ,在中心简单地配置后 ,消息可以通过多级跳自动路由到目标结点中 ,我们称这种方式为“消息不落地”。如果中心有存储转发的程序 ,即采用“消息落地”的方式 ,则中心可以利用自己的有利位置对所有的消息进行统一管理 ,比如监控干预流量 ,检查消息内容 ,修改消息格式 ,以及由此衍生出来的各种功能。事实上 ,已有一些产品提供类似的功能 ,比如 WebSphere Business Integration Message Broker 等。星状结构也有明显的缺点 ,即中心结点很容易成为消息通信和处理的瓶颈 ,且一旦中心结点出现故障 ,整个结构就无法互连通信了。为了避免中心结点成为单点故障 ,通常需要一些补偿措施 ,比如采用双机热备技术。

现实的应用场景中 ,很难简单地说明哪一种结构更好 ,需要视具体情况而定。

5.1.3 同步和异步

经常有人问 ,既然 MQ 处理消息在本质上是异步的 ,那么基于 MQ 设计的应用如何能做到同步呢 ?这里先让我们来澄清一下应用的同步和异步的概念。传统的应用程序调用后台服务通常采用的是请求/应答模式 ,应用程序在发出请求后希望在合理的时间内收到相应的应答 ,从应答中可以方便地判断出业务是成功还是失败 ,如果超过时间未收到应答则认为服务超时 ,这时应用程序需要有从等待中返回 ,如图 5 - 3 左。应用程序通常只关心请求与应答 ,而不会关心后台是如何将请求变成应答的。对于应用程序而言 ,请求的内容、应答的内容、期望的时间三者是这种调用模式的三个要素 ,也就是调用接口。如果这种调用过程可以一次完成 (最好在一个 API 中完成) ,则我们称之为同步调用。如果需要一次以上调用完成 ,则称为异步调用。



图 5 - 3 请求与应答

在 WebSphere MQ 中 ,应用程序通常将请求消息放入请求队列 ,接着在应答队列中等待读取相应的应答消息 ,如果在期望的时间中没有读到应答消息 ,则认为服务超时 ,应用程序会从读等待中返回 ,如图 5 - 3 右。这一过程是分多个动作完成的 ,所以从原子操作上看是一个异步调用的过程 ,但是如果我们把这一过程封装在一个调用中 ,则又可以看做是同步调用。所以 ,异步和同步都不是绝对的 ,在 MQ 设计中是可以转换的。

同步和异步的另一个区别在于同步调用超时后 ,会有某种机制保证服务无法正常完成。而对于异步

调用超时 ,由于没有这样的机制 ,往往会出现滞后响应 (Later Response)。在 MQ 应用中 ,如果需要达到同步调用同样的效果 ,则需要在设计上动一些脑筋。首先 ,请求消息放入请求队列后有可能迟迟得不到服务而躺在队列中睡觉 ,这时我们可以为消息设计有效寿命 ,超过有效寿命后消息不再具有业务上的时效性 ,可以不处理。通常这时候 ,应用程序已经超时了。其次 ,如果处理中碰到某个环节耽搁了很长时间超过了消息的寿命或应用程序可以忍受的时间 ,在处理中应用可以发现这一问题并放弃进一步处理。最后 ,消息在传递的过程中可以会遇到一些麻烦 ,这时可以通过设定自动报告的方式使应用程序及时得知发生的异常情况。

5.1.4 Client/Server

WebSphere MQ 产品分成客户端和服务端两部分 ,对于客户端而言没有 MQ 对象的概念 ,它通过配置的 MQI 通道将操作命令送到服务器端执行 ,结果再原路返回到客户端 ,所以客户端无法独立于服务器端而自行工作。服务器之间的通道也就是队列管理器之间的通道 ,称为消息通道。客户端和服务端之间的通道 ,称为 MQI 通道。那么 ,在设计时什么时候该采用客户端 ,什么时候又该采用服务器呢 ? 采用何种方式 ,完全取决于应用环境中能否发挥产品的特性 ,不可一概而论。由于客户端的限制 ,我们在设计时需要进行一些特殊的考虑。

首先 ,客户端永远只能作为消息发起的主动方而无法作为被动方。由于自身无法存放消息 ,所以待处理的消息只能暂存于服务器端。由客户端程序读服务器队列或设置客户端触发器来接收消息 ,但无论是哪一种方式客户端都无法独立工作。

其次 ,客户端只是将操作命令封装后送到服务器端执行 ,严格地说 MQI 通道传送的不是消息。客户端没有规范的通信进程 (MCA) ,也没有保证消息通信的 MQ 协议。所以 ,客户端与服务器端之间的 MQI 通道实际上是不能保证消息可靠性的。比如 ,客户端调用 MQGET 将队列中消息取出 ,命令在服务器端被成功执行 ,消息会随 API 的执行参数中返回。正当命令返回过程中 ,网络发生故障 ,这时客户端只取到消息的前半部分 ,返回出错。而在服务端该消息已经从队列中取出并删除了。也就是说 ,客户端再也无法取到消息的后半部分。

正是由于 MQ 客户端自身的这些限制 ,使客户端与服务器端之间的连接不适合应用于广域网 (WAN)。通常情况下 ,客户端与服务器端之间的连接应该在局域网内 ,利用局域网的可靠性和稳定性来弥补客户端的不足。一个典型的客户端与服务器端的应用场景如图 5 - 4 ,两套应用系统在各自的局域网上运行 ,通过 MQ 服务器在广域网上互连。这时 MQ 服务器扮演前置机或通信机的角色 ,应用系统内的机器作为 MQ 客户端连接该通信机 ,通过操作 MQ 服务器与对方系统互连通信。

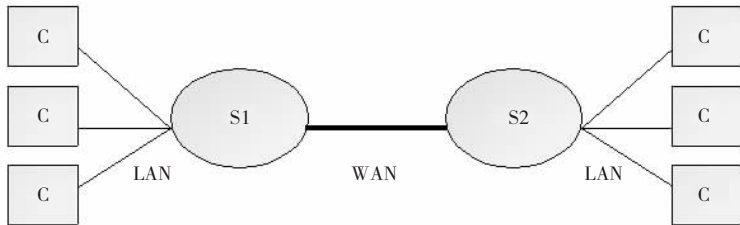


图 5 - 4 客户端与服务端

但是 ,客户端也有它独特的功能 ,比如客户端程序可以配置连接多个队列管理器 ,在一个连接不通的情况下自动连接另一个 ,这使得客户端应用程序有一定程度上有容错功能。如果多个克隆的客户端应用程序读同一个服务器队列 ,则可以形成负载平衡的效果。另外 ,由于 Client/Server 不是对等连接 ,无需在服务端做一对一的配置 ,客户端的连接配置又十分方便 ,所以对于结构多变的通信环境中有相当的用途。

5.1.5 Internet 通信

如果应用系统部署到互联网上 ,则首要考虑的是安全问题。外网接入层通常有防火墙 ,以阻隔来自外网的随意连接 ,有时还会有两道防火墙 ,并采用代理或隧道技术将内网的网络信息彻底对外屏蔽。在这种

情况下 ,WebSphere MQ 队列管理器无法对外直连 ,应用结构的设计可能需要做这方面的考虑。通常 ,我们会用 SupportPac MS81 – WebSphere MQ Internet PassThru (MQIPT) 来解决这方面的问题。

图 5 – 5 展示了使用 MQIPT 作为通道集中器的结构 ,外网的多台 WebSphere MQ 服务器需要与内网的 WebSphere MQ 服务器互连 ,这样需要在防火墙上做相应的设置 ,允许对相应 IP 地址和端口的连接通过 ,俗称在防火墙上钻洞。如果这样的连接需求很多 ,则需要钻很多的洞 ,会引起管理上的不方便。如果采用 MQIPT ,可以将所有的外来连接都集中于一台机器 ,通信数据由 MQIPT 转交 MQ 服务器 ,这样在防火墙上设置和管理上都带来方便。

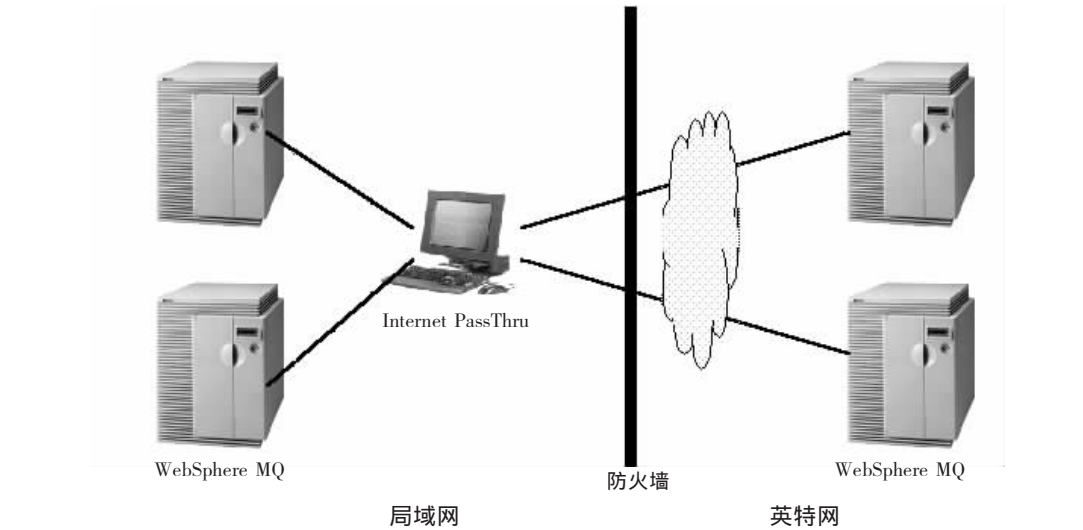


图 5 – 5 MQIPT 作为通道集中器实例

现实中的应用系统往往用两道防火墙将内网和外网相隔 ,两道防火墙的中间部分称为非保护区 (DMZ) ,用于安全缓冲。如图 5 – 6。从外网进入的第一道防火墙往往只留很少的通道 ,允许一些常规的端口对外服务 ,比如 HTTP ,其他连接一概被拒绝。第二道防火墙将安全的内网与不太安全的非保护区隔开 ,将内部的网络结构和信息进一步保护起来。如果我们将应用的连接和数据封装成为 HTTP 协议进入 ,这种技术称为 HTTP 隧道。来自 Internet 的 WebSphere MQ 连接可以通过 MQIPT 将消息包装后进入 HTTP 隧道 ,在通过隧道后再由 MQIPT 将 HTTP 协议外壳脱去 ,还消息的本来面目。另外 ,Internet 上的 SSL 加密认证也是常用的安全手段 ,它可以配置在两端的 MQIPT 上 ,使通信更安全可靠。

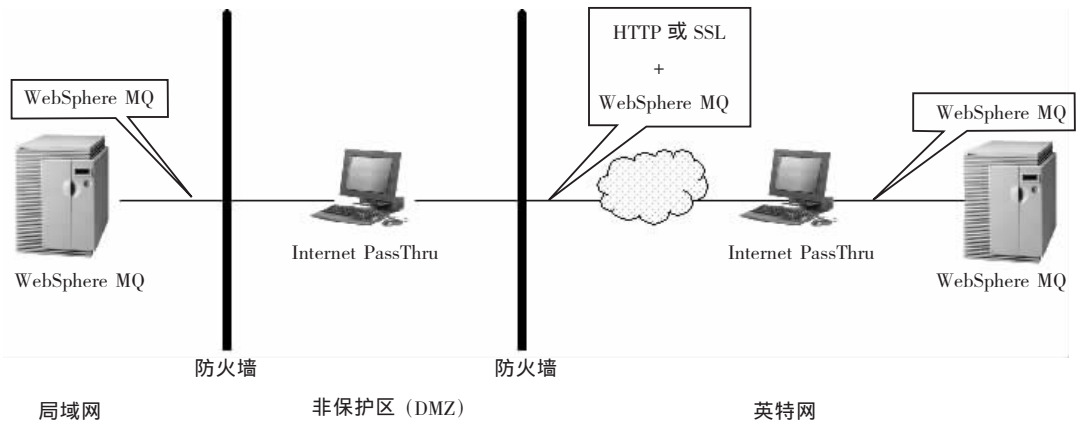


图 5 – 6 采用 HTTP 隧道的技术的两道防火墙实例

5.2 通信方式设计

5.2.1 进程间会话模式

进程间会话模式实际上是程序与程序之间通信时的一种消息通话机制,它是程序间数据沟通的联系模式。WebSphere MQ 提供了下列灵活的基本通信方式:

- 串行通信

程序 A 发出数据给程序 B,等到回应后再发出下一次数据。这有点类似于打电话的方式。

- 并行通信

程序 A 发出数据给程序 B,无需等待 B 的回应,A 继续处理自己的工作。程序 B 可以在任何方便的时候取到消息并做相应的处理。这就需要有消息队列作为中间人将 A 发出的消息保留直被 B 取到。这有点类似于电话留言的方式。并行通信中,A 可以接连不断地发出数据给 B,然后在合适的时候等待 B 统一的回应。

- 无连接通信

程序 A 与程序 B 之间没有私有的或公共的信道,它们仅靠队列机制提供的端对端的服务来通信,而这对于程序 A、B 来说都是完全透明的。

- 触发机制

程序 A 可以将触发消息发送到带有触发机制的队列上,由触发机制触发程序 B 启动执行。

- 并行处理

程序 A 可以将一个复杂任务分成多个子任务,通过发送消息的办法交给多个程序处理,再由程序 A 收集处理结果。由于多个程序间可以没有执行的先后次序,所以多个子任务可以是并行处理的。

- 客户机/服务器

客户程序 A 发送请求给服务程序 B,服务程序 B 处理完相应的工作后将应答自动封装成另一条消息返回给客户程序 A。

- 分布式处理

由以上所有通信方式组合而成的复杂方式。

在 MQ 的应用设计时,程序之间选择何种会话模式是最初要考虑的部分,也是设计的重点之一。程序间的消息流向是单向的还是双向的,请求和应答程序是串行通信还是并行通信,是否采用触发机制,等等,这都是需要在设计最初确定下来的。

5.2.2 系统间通信方式

WebSphere MQ 的基本功能是用来传递消息的,通过配置通道来实现系统之间的互连通信。通信方式分为点对点的方式和订阅/发布的方式,其中点对点的方式可以实现两个单点系统之间的直接互连,也可以实现多点系统链式的间接互连(Multi-hopping)。点对点的另一种扩展方式为分发列表,类似于广播。订阅/发布是一种较高级的互连通信方式,发布者和订阅者首先约定一个消息主题(Topic),之后发布者在 MQ 网络上发布该主题的消息会经过 MQ 网络自动地送达订阅者,如图 5-7。

点对点(也称为一对一)的通信方式是 WebSphere MQ 最拿手的,也是传统设计方法考虑得最多的,它通过定义远程队列、传输队列和通道的方式将队列管理器联系起来。订阅/发布(也称为多对多)的通信方式需要有额外的部件支持,通常会采用 SupportPac MA0C 或者 Message Broker。应用设计采取哪一种通信方式取决于应用中各程序之间的关系,如果消息内容是针对某一个体,则采用点对点的方式比较妥当,如果消息内容是针对一个群体,则采用订阅/发布进行广播更合适一些。

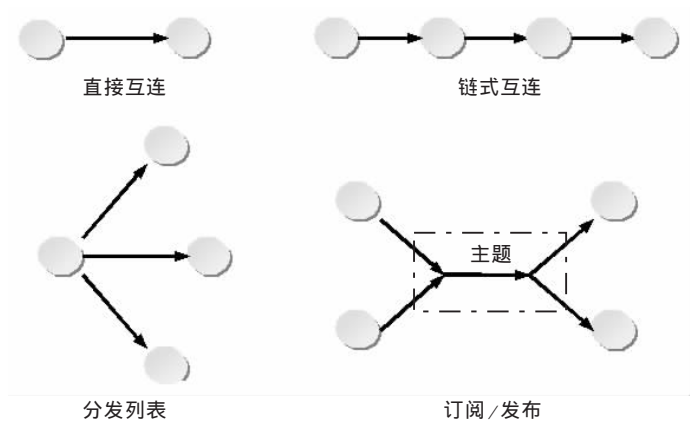


图 5-7 WebSphere MQ 互连通信的方式

5.3 并发设计

5.3.1 多读多写

WebSphere MQ 编程环境是支持多读多写的。无论是读或写 ,都需要应用程序要在打开对象的时候指定功能选项 ,比如 MQOO _ INPUT 和 MQOO _ OUTPUT ,这里的 INPUT 和 OUTPUT 都是站在应用程序的立场而言的。

在应用设计时 ,如果哪一个环节需要增加处理能力 ,可以考虑多实例 (Multi-Instance) 并行工作的方式。由于队列等对象支持多读多写 ,所以多实例的工作效率可能是单实例的数倍。

5.3.2 共享与独占

对于读消息 ,WebSphere MQ 支持共享和独占两种方式。对于写消息 ,WebSphere MQ 只支持共享方式。在 MQOPEN 时的参数选项可取值：

- MQOO _ INPUT _ AS _ Q _ DEF 根据队列的 DEFSOPT 属性来决定
- MQOO _ INPUT _ SHARED 共享式读取消息
- MQOO _ INPUT _ EXCLUSIVE 独占式读取消息
- MQOO _ OUTPUT 共享式发送消息

然而 ,消息所在的队列有自己的属性可以控制打开方式是共享还是独占 ,如表 5-1。

表 5-1 队列与共享相关的属性

队列属性	说明
SHARE 或 NOSHARE	是否允许多个应用程序并发地 MQGET 操作该队列
DEFSOPT () <ul style="list-style-type: none">● EXCL● SHARED	指定 INPUT 方式打开 (即需要 MQGET) 时的缺省共享模式 <ul style="list-style-type: none">● 只能有一个应用程序以 INPUT 方式打开● 可以有多个应用程序以 INPUT 方式打开

队列属性与打开方式 (MQOO _ *) 必须配合起来使用。如果应用程序在 MQOPEN 时希望由队列属性来决定共享或独占打开方式 ,则用 MQOO _ INPUT _ AS _ Q _ DEF ,打开方式会参考 DEFSOPT 属性。反之 ,应用程序 MQOPEN 显式地设置 MQOO _ INPUT _ EXCLUSIVE 或者 MQOO _ INPUT _ SHARED 即可。

5.3.3 对象绑定

WebSphere MQ 应用程序在打开对象的时候可以指定是否绑定对象 ,一旦绑定后 ,操作对象就固定下

来了。反之 ,如果不绑定对象则每次在发送的时候寻找目标对象 ,这在群集多实例队列环境中有用。在 MQOPEN 时的参数选项可取值 :

- MQOO _ BIND _ ON _ OPEN 打开时绑定
- MQOO _ BIND _ NOT _ FIXED 不绑定
- MQOO _ BIND _ AS _ Q _ DEF 根据队列的 DEFBIND 属性来决定

而 DEFBIND 可取值 MQBND _ BIND _ ON _ OPEN 和 MQBND _ BIND _ NOT _ FIXED。它们之间的关系如表 5 - 2。

表 5 - 2 MQOPEN 参数与队列 DEFBIND 属性共同决定队列绑定方式

MQOPEN 参数选项	队列 DEFBIND 属性	
	MQBND _ BIND _ ON _ OPEN	MQBND _ BIND _ NOT _ FIXED
MQOO _ BIND _ ON _ OPEN	BIND 绑定	BIND 绑定
MQOO _ BIND _ NOT _ FIXED	NOT _ FIXED 不绑定	NOT _ FIXED 不绑定
MQOO _ BIND _ AS _ Q _ DEF	BIND 绑定	NOT _ FIXED 不绑定

5.3.4 队列管理器关闭

由于队列管理器是支持并发操作的 ,所以一个应用程序在操作时无法预料其他应用程序的操作。比如 ,一个在队列上做永久等待读操作的应用程序可能会因为队列管理器的突然关闭而僵死 ,所以在操作时使用 FAIL _ IF _ QUIESCING 是个好习惯。队列管理器在关闭的时候 ,可以选择 endmqm - c 或 - w 选项 ,使队列管理器进入待关闭 (Quiescing) 状态。这时 ,队列管理器不再接受新的应用连接 ,而等待原来的应用运行完毕后彻底关闭队列管理器。这时 ,原来的应用应该与之配合尽快退出。有了 FAIL _ IF _ QUIESCING 选项 ,就可以在 API 调用中及时地发现并做出相应的动作 ,比如结束对消息的永久等待并回滚交易。

相关的操作选项有 :

- MQOO _ FAIL _ IF _ QUIESCING
- MQPMO _ FAIL _ IF _ QUIESCING
- MQGMO _ FAIL _ IF _ QUIESCING

5.3.5 分发列表 (Distribution List)

WebSphere MQ 中一次 MQPUT 或 MQPUT1 可以同时多个队列进行操作 ,这需要将多个队列纳入分发列表中 ,以后对该分发列表的操作就如同对列表中所有队列的操作 ,这有点像群发邮件的功能。这些操作可以在交易环境中进行 ,使多个队列的消息分发同时提交或回滚。如果多个队列是远程队列且目标队列管理器相同 ,则在网络上只需要传送一次 ,在到达对方后由目标队列管理器进行二次分发。

并非所有的 WebSphere MQ 平台都支持分发列表 ,所以分发列表正常工作相关平台支持 ,如果需要前文中的二次分发功能 ,则目标队列管理器也要支持分发列表才可以。

5.4 消 息 设 计

5.4.1 消息大小 (Message Size)

WebSphere MQ 中消息可以分成物理消息和逻辑消息 ,其中逻辑消息是由多条物理消息组成的。物理消息缺省可以达到 4MB ,这对大多数应用来说是够用了 ,所以通常情况下逻辑消息与物理消息是统一的。在某种特定的情况下 ,消息需要分组或分段 ,这时逻辑消息可能会含多个物理消息。物理消息是 MQ 对消息计数的单位 ,通常与 MQGET 或 MQPUT 相对应。物理消息的大小是有限的 ,受限于队列管理器、队列或通道对象的相关属性。表 5 - 3 记录了各对象与消息大小相关的属性。

表 5-3 WebSphere MQ 对象中与消息大小相关的属性

对象	属性	缺省值	范围	说明
队列管理器	MAXMSGL	4M	32K ~ 100M	
队列 (Local/Model)	MAXMSGL	4M	0 ~ QM. MAXMSGL	0 表示用 QM. MAXMSGL
通道	MAXMSGL	4M	0 ~ QM. MAXMSGL	0 表示用 QM. MAXMSGL

队列管理器的 MAXMSGL 表示队列管理器中可以容纳的最大消息长度 缺省值为 4 194 403 ,调整范围从 32 768 到 104 857 600。队列管理器的 MAXMSGL 的值应该始终大于等于队列的 MAXMSGL 值 ,如果这个值调小了 ,应该检查队列管理器中所有的本地队列和模型队列的 MAXMSGL 属性。

队列的 MAXMSGL 仅对本地队列和模型队列有效 表示队列中可以容纳的最大消息长度 ,这个属性的调整范围在各个平台上的上限是不一样的。

- 对于 AIX、Compaq OpenVMS、HP – UX、Linux、OS/2 Warp、OS/400、Solaris 和 Windows 平台 ,队列的 MAXMSGL 值可以从 0 到队列管理器的 MAXMSGL (上限为 100 MB)。
- 对于 z/OS 平台 ,大小为 0 ~ 104 857 600 Bytes。
- 对于其他平台 ,大小为 0 ~ 4 194 304 Bytes。

对于传输队列 ,消息的最大长度也包括消息传输头的大小。如果这个值调小了 ,已经在队列中的超长消息不受影响。应用程序可以用这个值来决定需要申请的内存 ,所以调大该值有可能会影响应用程序的正常运行。

通道的 MAXMSGL 表示通道可以一次传送的最大消息长度。

- 对于 AIX、iSeries、HP – UX、Linux、Solaris 和 Windows ,通道的 MAXMSGL 的大小应该大于等于 0 ,小于等于队列管理器的 MAXMSGL。
- 对于 z/OS 平台 ,大小为 0 ~ 104 857 600 Bytes。
- 对于其他平台 ,大小为 0 ~ 4 194 304 Bytes。

通道在建立的时候会有一个握手过程 ,双方会交换各自通道定义上的 MAXMSGL ,最后协商出通道使用的最大消息长度 ,一般会取双方定义中较小的那一个。

如果通道的两端支持消息切分 ,则一旦实际传送的消息长度比通道的可以传送的最大消息长度 (MAXMSGL) 更大也不要紧 ,MQ 会自动将其切分后送出 ,在远端自动拼接起来。只要小于队列管理器的 MAXMSGL 和队列的 MAXMSGL ,使本地队列能够放得下这条消息即可。

如果通道的两端都不支持消息切分 ,且消息实际传送的消息长度比通道的 MAXMSGL 大 ,则出错。

所以 ,鉴于各平台对队列和通道上的最大消息长度定义不一 ,在设计的时候需要将这种差别考虑在内。比如 ,考虑 SCO OpenServer 与 AIX 之间的数据传输 ,消息的大小设计应该小于 4MB。

5.4.2 消息持久性 (Persistence)

消息可以分成持久性消息和非持久性消息 ,前者在被放入队列或从队列中取出时除了操作队列外还会记录日志 ,所以在队列管理器重启后消息会自动恢复。后者由于没有日志操作 ,消息会丢失。另外 ,持久性消息通常表示消息是重要的 ,不可丢失的 ,在传送发生困难时 ,MQ 会尽可能保护消息不丢失。非持久消息在无法送递时往往被丢弃。消息的持久性与存放消息的队列及传递消息的通道有一定的关系 ,表 5-4列举了 MQ 对象中与消息持久性相关的属性。

表 5-4 WebSphere MQ 对象中与消息持久性相关的属性

对象	属性	缺省值	可取值	说明
消息 (MQMD)	DefPersistence	Q_DEF	<ul style="list-style-type: none">● MQPER_PERSISTENCE_AS_Q_DEF● MQPER_PERSISTENT● MQPER_NOT_PERSISTENT	

续表

对象	属性	缺省值	可取值	说明
队列	DEFPSIST	NO	<ul style="list-style-type: none">● YES● NO	z/OS 中用 Y 或 N
通道	NPMSPEED	FAST	<ul style="list-style-type: none">● NORMAL● FAST	

消息头 MQMD 属性 DefPersistence 可以有 3 种取值 ,但实际上只有两种有效值 ,即持久或非持久。对于 MQPER _ PERSISTENCE _ AS _ Q _ DEF 消息在 MQPUT/MQPUT1 时会设置队列的缺省属性。持久的消息可以在队列管理器重启时恢复 ,非持久的消息则不可以。持久的消息写入或读出队列的同时会在 Log 中记录 ,所以性能上比非持久消息差不少。

队列属性 DEFPSIST 表示放入其中的消息缺省是持久的 (Persistence) 还是非持久的 (Non-Persistence)。持久消息不可以放入临时动态队列 (Temporary Dynamic Queue)。因为考虑到临时动态队列可以在 MQOPEN 时创建 ,在 MQCLOSE 时删除 ,在队列管理器重启后也会自动清除。这与持久消息不会丢失的性质不相容 ,所以 WebSphere MQ 禁止持久性消息放入临时动态队列 ,在程序运行时报错 :MQRC _ PERSISTENT _ NOT _ ALLOWED。

通道的属性 NPMSPEED (Nonpersistent Message Speed) 有两种取值 NORMAL 或 FAST。缺省设置为 NORMAL ,但如果设置成 FAST ,则非持久消息的传送可以不参加交易。交易中的非持久消息一旦到达传输队列会立即送出 ,这可以使非持久消息的传送速度大大加快。缺点是消息可能丢失 ,可能泄露到交易之外。丢失指消息传送失败或通道关闭后 ,消息不会被回滚到传输队列中。泄露指虽然消息在发送方交易中进行操作 ,但因为消息是非持久消息 ,一旦放入传输队列就立即送出到达对方 ,无法回滚。

几乎所有的设计人员都会认为系统中出现的每一条消息都是非常重要的 ,所以都应该是持久消息。但是在有些性能优先的系统中 ,适当地采用非持久消息从而保证传递速度也是一种常用的办法 ,通常系统会再设计某种机制在出错时进行纠正或业务上的补偿。

5.4.3 消息优先级 (Priority)

消息可以按其重要程度划分成不同的优先级 ,MQ 中消息的优先级可以取从 0 到 9 的任意一个值 0 表示最低 9 表示最高。高优先级的消息在传递和读取时都具有较高的优先权 ,会被优先处理。消息的优先级与队列管理器和队列的相关属性有一定的内在关系 ,如表 5 - 5。

表 5 - 5 WebSphere MQ 对象中与消息优先级相关的属性

对象	属性	缺省值	可取值	说明
队列管理器	MAXPRTY	9	9	这个值无法修改
队列 (Local/Model)	DEFPRTY	0	0 - 9	
队列 (Local/Model)	MSGDLVSQ	PRIORITY	<ul style="list-style-type: none">● PRIORITY● FIFO	
消息 (MQMD)	Priority	Q. DEFPRTY		

队列管理器的 MAXPRTY 属性表示队列管理器中允许的最大消息优先级 ,缺省为 9。消息的 Priority 表示消息的优先级 ,可以取 0 到 QM. MAXPRTY 中的任何一个值。

本地队列或模型队列的 DEFPRTY 属性表示队列上消息的缺省优先级。在 MQPUT/MQPUT1 时 ,如果 MQMD. Priority = MQPRI _ PRIORITY _ AS _ Q _ DEF (- 1) ,则消息取值为队列的 DEFPRTY 属性。

本地队列或模型队列的 MSGDLVSQ 属性可以设置消息传递次序 (Message Delivery Sequence) 的方式 ,

即 MQGET 时消息选取的次序。它可以有两种取值 :PRIORITY 或 FIFO ,前者表示消息按优先级从高到低排序 ,同优先级按先进先出排序 ,后者表示只按先进先出排序。一旦指定了 FIFO ,则设置 MQMD. Priority 不再生效 ,因为消息只按先后次序排序 ,优先级这时变得没有意义。FIFO 方式排序时 ,新放入队列的消息优先级为队列的缺省优先级属性 DEFPRTY。事实上 ,PRIORITY 和 FIFO 的本质是相同的 ,都是以优先级为第一索引 ,以时间为第二索引。只是 FIFO 方式中优先级都一样 ,等于 DEFPRTY。

如果队列的 MSGDLVSQ 属性从 PRIORITY 变成 FIFO ,那么队列中可能存有一些 Priority 低于 DEFPRTY 的消息 ,新到的消息优先级为 DEFPRTY (不管是否指定 MQMD. Priority) ,所以新到的消息可能比原先的低优先级消息更早地被处理。反过来 ,如果 MSGDLVSQ 从 FIFO 变成 PRIORITY ,队列中原有的消息优先级应该是 DEFPRTY。

5.4.4 消息超时 (Expiry)

在消息头 MQMD 中的 Expiry 域为消息的超时时间 ,即消息的有效生命周期 ,单位是 1/10 秒。这个值表示消息从放入队列开始计算 ,应该在多长的时间内被取出处理。缺省情况下 ,MQMD. Expiry = MQEI _ UNLIMITED (-1) ,表示消息永不超时。

Expiry 的记时与超时原理如下 :

1. 从消息开始放入队列开始计时。不允许 MQMD. Expiry = 0 ,否则返回 MQRC _ EXPIRY _ ERROR。这时 MQMD. PutDate 和 MQMD. PutTime 会记录下 MQPUT/MQPUT1 的时间。
2. 消息在路由的过程中可能会被队列管理器多次放入传输队列并取出传送 ,所有这些在路上耽搁的时间都会计算在内。事实上 ,这些内部的 MQPUT/MQGET 动作会使 Expiry 时间衰减。每次消息进入传输队列 ,队列管理器会在消息头上加上一个 MQXQH ,其中也有 Expiry ,它会标记消息在这段传输路程上的耗时 ,在脱去 MQXQH 时会反映到 MQMD. Expiry。
3. 消息最终到达目标本地队列 ,在那里进入排队。在开放平台的队列管理器中并没有超时监控程序始终监视队列中的每一条消息是否超时。事实上 ,WebSphere MQ 直到消息被 MQGET 扫描到才会被发现超时。
4. 队列可能是 FIFO 方式 ,也可能是 Priority 方式 ,并且 MQGET 还可能有匹配条件。这需要队列管理器对队列中的消息从头开始扫描 ,扫描到的消息都会根据消息创建时间 (MQMD. PutDate ,MQMD. PutTime) 、消息超时 (MQMD. Expiry) 及当时时间计算出消息是否超时。如果超时 ,则被删除。如果消息带有超时报告标志 (MQRO _ EXPIRATION _ *) ,则同时发送超时报告。

由于开放平台上 WebSphere MQ 消息超时的发现依赖于 MQGET 对队列的扫描 ,所以消息的有效生命周期是可以保证的 ,但超时点可能不准确。换句话说 ,如果消息躺在队列中 100 年 ,尽管早已超时 ,只要始终未被 MQGET 扫描到 ,就一直不会收到超时报告。

对 FIFO 队列而言 ,不含匹配条件的 MQGET 会从第一条消息一直扫描到第一个未超时的消息。对 Priority 队列而言 ,同样的 MQGET 会扫描所有更高优先级的消息一直扫描到等高优先级中的第一个未超时的消息。扫描过的部分可能有多条消息被发现超时。

z/OS 上的 WebSphere MQ 有独立的消息自动监控机制。队列管理器有 ExpiryInterval 属性表示每隔多长时间队列管理器自动扫描所有的消息 ,检查是否有消息超时 ,并做出相应的处理。ExpiryInterval 的单位是秒 ,取值 1 ~ 99 999 999。也可以取值 MQEXPI _ OFF ,关闭此功能。

5.5 发送设计

5.5.1 消息标识

WebSphere MQ 中每一条消息都有 3 个标识 :MsgId ,CorrelId 和 GroupId。其中 ,MsgId 是消息自身的标识 ,CorrelId 是消息的相关标识 ,GroupId 是消息所有组的标识。这三个标识都是 MQMD 中的域 ,长度为 24

字节。

在 MQPUT 或 MQPUT1 时,应用程序可以设置 MQMD.MsgId 来设定消息标识。通常来说,应用系统中不应该出现两条标识相同的消息。所以由应用程序自动设定消息标识的办法并不稳妥,这需要程序自己来保证消息标识的惟一性。

如果在发送消息的时候,MQMD.MsgId 为空,或者设置了发送选项 MQPMO.Options = MQPMO_NEW_MSG_ID,则队列管理器会自动生成一个惟一标识来标记这条消息,该标识由消息标识头“AMQ”、队列管理器名简称、时戳等信息构成,理论上队列管理器不会生成两条 MsgId 完全相同的消息。这样就自动地保证了消息标识的惟一性,不至于在匹配消息时引起混乱。然而,自动生成的消息标识可能含有非打印字符,在表达上稍显困难。

在使用分发列表 (Distribution List) 时,一条消息可能在传送的过程中自动复制成多条消息,为了使 WebSphere MQ 自动为这些消息生成不同的 MsgId,必须设置 MQMD.MsgId 为空,或使用 MQPMO_NEW_MSG_ID 选项。

CorrelId 表示相关的消息标识,比如应答消息用 CorrelId 指明相关的请求消息,报告 (Report) 消息指明相关的源消息,等等。在 WebSphere MQ 的应用中,可以有多条消息与某一条源消息相关。也就是说,可以有多条 MsgId 不同的消息,它们的 CorrelId 是相同的。应用程序可以自行设定 MQMD.CorrelId 来设定相关标识。

类似于 MsgId,如果设置发送选项 MQPMO.Options = MQPMO_NEW_CORREL_ID,则消息会自动生成 CorrelId,规则同 MsgId。

GroupId 表示消息所在的组标识,相同的组标识表示这些消息属于同一个组,在匹配时可以整组匹配。详见“分组与分段”章节。

5.5.2 消息类型

MQMD 的 MsgType 域标志着消息类型。它可以表示消息是请求 (Request) 还是相应的应答 (Reply),是应用 (Datagram) 消息还是相应的报告 (Report) 消息。在发送时设定消息类型,主要是使接收程序在收到该消息后便于区别,从而做出正确的反应。

通常编程时,MQMD.MsgType 可以取值:

- MQMT_REQUEST
- MQMT_REPLY
- MQMT_DATAGRAM
- MQMT_REPORT

5.5.3 消息格式

消息内容有一定的组织格式,比如说是一个结构。结构中的域有些是整数类型的,也有些是字串类型的。由于 WebSphere MQ 所处的操作系统平台不同,从而会引起整数型数据在高低字节编码上的不同,也会引起字串型数据所使用的字符集不同。MQMD 中 3 个域 Format、Encoding 和 CodedCharSetId 分别可以指明消息的格式、编码、字符集。

Format 可以取值:

- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_CHANNEL_COMPLETED
- MQFMT_CICS
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER

- MQFMT_DIST_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_MD_EXTENSION
- MQFMT_PCF
- MQFMT_REF_MSG_HEADER
- MQFMT_RF_HEADER
- MQFMT_RF_HEADER_2
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_WORK_INFO_HEADER
- MQFMT_XMIT_Q_HEADER

消息分为消息头 (MQMD) 和消息体两部分。MQMD.Format 表示的是消息体的数据结构,Format 可以认为是消息体的数据结构名。不过也有例外,MQFMT_STRING 表示消息体是字串,并非一个结构。如果不知道消息体的结构,则可以用 MQFMT_NONE 来表示。

编码指的是各种类型数据的表示方式,比如高低字节安排等,字符集指的是消息中字符所属的集合,比如中文字符集。消息的发送和接收平台可以不一样,所以当发送和接收出现编码和字符集不一致的情况时,就会发生消息格式转换,即从一种格式自动转向另一种格式。详见“格式转换”章节。

5.5.4 应答队列

应用程序通过 WebSphere MQ 进行请求/应答方式通信的时候,请求方发送消息后通常希望应答消息自动放入指定的队列,该队列就是应答队列。在请求消息的 MQMD 结构中的 ReplyToQMGr 和 ReplyToQ 域指明了应答队列的位置。应答方收到这样的请求消息后,经过消息处理,产生相应的应答消息,会“心领神会”地将应答消息放入应答队列中(图 5-8)。当然,设置应答队列,对于应答方应用程序而言,只有“指导意义”而无“强制意义”。

应答队列的设置,可以使相同的应答程序与不同的请求程序形成多个请求/应答消息环路。请求方可以指定各自的应答队列,互不干扰。另一种做法是多个请求方共用一个应答队列,发送请求消息的时候保证消息的 MsgID 不重复,取应答消息的时候匹配各自消息的 CorrelID。这要求应答方要请求消息的 MsgID 拷贝到应答消息的 CorrelID,以表示是针对哪一条请求消息的应答。

通常情况下,请求队列与应答队列是两个不同的队列,但在不致引起混淆的情况下请求队列和应答队列也可以共享同一个队列。

5.5.5 动态队列

打开模型队列时,WebSphere MQ 会按照它的属性自动动态地生成一个本地队列,称为动态队列。普通本地队列的 DEFTYPE 属性值为 PREDEFINED,动态队列按模型队列的 DEFTYPE 属性可以取值 PERMDYN 和 TEMPDYN,分别表示永久动态队列和临时动态队列。其中永久动态队列指动态队列一旦生成,如果不是显式地将其删除,它将永久地保留在队列管理器中。临时动态队列指队列在关闭时缺省地自动删除。

- 普通本地队列 (Predefined Queue) 队列属性 DEFTYPE = PREDEFINED
- 动态队列 (Dynamic Queue)
 - 临时动态队列 (Temporary) 队列属性 DEFTYPE = TEMPDYN
 - 永久动态队列 (Permanent) 队列属性 DEFTYPE = PERMDYN

动态队列在打开时自动生成,在关闭时自动删除,这种特性使得它能满足很多临时存储的需求。比如

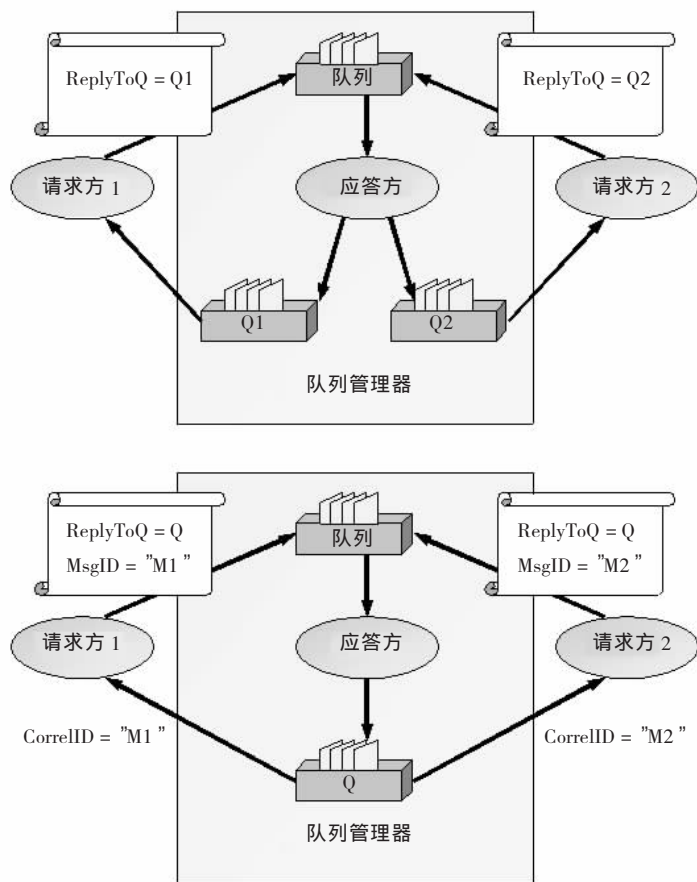


图 5-8 应答队列的设置

偶尔需要生成消息的应答队列来存放应答消息,使用过后就不再需要了;再如一个应用程序在启动时生成一套私用的工作队列,在停止时将现场清理干净。

5.5.5.1 生成动态队列

在 MQOPEN 时,指定对象描述参数 MQOD 的 ObjectType 域为 MQOT_Q,ObjectName 域为模型队列名,DynamicQName 域为动态队列名。这里 DynamicQName 可以取以下值:

- *<Name>* 手工生成,填满队列的名字(48 个字符)。动态队列为 *<Name>*,如果队列管理器中有重名队列则出错。这里 *<Name>* 为符合队列命名规则的任何一个字符串。
- *<Prefix> ** 半自动生成,填队列名字的前缀(少于 33 个字符),最后一个非空字符填“*”。动态队列为 *<Prefix>*,以后的部分自动生成,生成的时候会自动避开已存在的队列名。这里 *<Prefix>* 为符合队列命名规则的任何一个字符串。
- * 全自动生成,填“*”。动态队列为完全由队列管理器自动生成,会自动避开已存在的队列名,不会重复。

MQOPEN 后,生成的动态队列名记录在 MQOD 的 ResolvedQName 域中。动态队列一旦生成了,其 DefinitionType (DEFTYPE) 属性就无法改变了。所以,在一开始选择正确的模型队列(Model Queue)很重要。

5.5.5.2 关闭动态队列

永久动态队列是可以永久存在的。在调用 MQCLOSE 时,可以通过关闭选项(Close Option)的设置来指定永久动态队列不同的关闭行为。

- MQCO_NONE 永久动态队列会保留下来

- MQCO_DELETE 如果永久动态队列中没有消息,也没有未提交的消息,则删除队列
- MQCO_DELETE_PURGE 如果永久动态队列中没有未提交的消息,则清除现有的所有消息后删除队列

临时动态队列是在此会话 (Session) 中临时存在的,即在线程中 MQOPEN 时队列被自动创建,在同一个线程中 MQCLOSE 时队列被自动删除,下一次 MQOPEN 会重新创建一个临时动态队列。MQCLOSE 时无所谓关闭选项 (Close Option) 的取值。

5.5.5.3 动态队列和持久性消息

永久动态队列由于是可以永久存在的,也就是说,程序退出后甚至队列管理器重新启动后,队列仍然存在。这样,队列一旦生成就和普通的本地队列是一样的。所以,动态队列可以存放持久性消息,也可以存放非持久性消息。

临时动态队列由于其本身是不持久的,程序一旦退出队列自动删除,这样的特性与持久性是相悖的,所以临时动态队列只能存放非持久性消息,不可存放持久性消息。

5.5.5.4 动态队列的使用

动态队列大多数都是为了自动生成,临时使用。常见的使用方式可以是为了生成临时的应答队列用来临时存放应答消息或报告消息。在 MQOPEN 时临时生成这个应答队列,在 MQPUT 时消息的 MQMD.ReplyToQ 域来指定该队列,当接收到应答消息后这个队列就没用了,可以用 MQCLOSE 将其删除。

5.5.6 用户替换

消息在从应用程序进入队列的时候,消息的发送者会被填入 MQMD 的 UserIdentifier 域,缺省情况下这就是运行应用程序的用户。

如果在 MQOPEN 时设定 MQOD.AlternateUserId 为指定用户,并选择 MQOD.ALTERNATE_USER_AUTHORITY 选项,则 WebSphere MQ 会用 AlternateUserId 用户身份来打开该队列,实现用户替换。以后的 MQPUT 会将该用户填入 MQMD.UserIdentifier。

类似地,MQPUT1 中含 MQOPEN,可以指定 MQOD.AlternateUserId,并选择 MQPMO.ALTERNATE_USER_AUTHORITY 选项,则效果与 MQOPEN 相同。

在用户替换的过程中,如果 MQOD.AlternateSecurityId 为空,则使用 MQOD.AlternateUserId 来进行权限检查。否则,可以使用 MQOD.AlternateSecurityId。比如,在 Windows 中不同的机器可以有相同的用户名 (UserId),但某个用户名在域中有惟一的安全标识 (SecurityId),可以用这样的 SID 放入 MQOD.AlternateSecurityId 进行权限认证。

5.6 读取设计

5.6.1 等待读取 (Wait & NoWait)

MQGET 可以等待读取消息,即先设定消息的匹配条件,然后在队列上等待消息的到来。当然也可以不设条件,等待第一条消息的到来。这时,设置 MQGMO.Options = MQGMO_WAIT,且设置 MQGMO.WaitInterval 为等待时间,单位为毫秒。若在设定的时间内有匹配的消息到来,则 MQGET 读到消息,返回 MQRC_NONE。否则,MQGET 超时,返回 MQRC_NO_MSG_AVAILABLE。如果 MQGMO.WaitInterval = MQWI_UNLIMITED (-1),即超时设定为无穷大,则 MQGET 会永远等待在队列上,直到匹配的消息到来。

MQGET 也可以非等待方式读取。设置 MQGMO.Options = MQGMO_NO_WAIT,MQGET 时如果队列中没有匹配的消息,则立即返回。

5.6.2 信号中断 (Signal)

对于 WebSphere MQ for z/OS、Compaq NonStop Kernel、Win95/98 环境。可以用信号将消息读取中断。在 MQGET 时设置 MQGMO_SET_SIGNAL 选项,并在 MQGMO 结构的 Signal1 和 Signal2 中设定信号值,则在特定的时候会有信号中断出现。应用程序可以通过截取信号中断的方法,实现并发同时读多个队列。

```
MQLONG Signal1 ;
```

```
MQLONG Signal2 ;
```

对于 z/OS,信号值设在 Signal1 中,队列管理器自动送出的 ECB (Event Control Block) 中含有信号完成码:

- MQEC_MSG_ARRIVED
- MQEC_WAIT_INTERVAL_EXPIRED
- MQEC_WAIT_CANCELED
- MQEC_Q_MGR_QUIESCING
- MQEC_CONNECTION_QUIESCING

对于 Compaq NonStop Kernel,信号值设在 Signal1 中,中断消息会发送到进程的 \$RECEIVE 队列。

对于 Windows,信号值设在 Signal2 中,中断消息以 Windows 系统消息的方式发送给该进程。

5.6.3 截断消息 (Truncated Message)

MQGET 的时候如果队列中的消息长度大于准备接收的 Buffer 长度,则可能发生消息截断,即消息被取出,但只有前一部分放入 Buffer 中,后一部分数据丢失。

有时应用上的确需要这种截断功能,比如只需要读取消息的前 100 个字节。这时,设置 MQGMO.Options = MQGMO_ACCEPT_TRUNCATED_MSG。在 MQGET 时,如果消息长度大于 BufferLength,则 DataLength 表示消息的实际长度,消息被取出,但只读到前一部分,返回 MQRC_TRUNCATED_MSG_ACCEPTED。

有时应用上需要读取完整的消息,避免截断情况的发生。这时,MQGMO.Options 上不要设定 MQGMO_ACCEPT_TRUNCATED_MSG 选项。在 MQGET 时,如果消息长度大于 BufferLength,则 DataLength 表示消息的实际长度,消息仍然保留在队列中,返回 MQRC_TRUNCATED_MSG_FAILED。通常在这种情况下,应用程序应该按 DataLength 分配实际需要的内存,再进行一次 MQGET 将消息读出。

在 MQGET 时如果发生数据转换,则消息的长度可能会有所变化。在 MQGET 返回后,如果 MQGMO.ReturnedLength 不为 MQRL_UNDEFINED (-1),则表示返回的字节数,通常情况下与 DataLength 相同。如果为 MQRL_UNDEFINED,则以 DataLength 为准。

5.6.4 浏览消息 (Browse)

浏览消息本质上是读消息的一种特殊方式,只是消息队列中并没有被自动删除。在打开队列时用 MQOO_BROWSE 指明浏览方式打开并按匹配条件依次读取消息内容。以浏览方式打开队列时会创建一个游标,其工件原理类似于数据库游标,它会记住当前浏览的位置,以后调用 MQGET (MQGMO_BROWSE_FIRST) 可以游标定于匹配消息集合的第一条上,MQGET (MQGMO_BROWSE_NEXT) 可以依次浏览。

MQGET (MQGMO_BROWSE_MSG_UNDER_CURSOR) 可以再次浏览游标指向的消息,游标不动。MQGMO_BROWSE_* 可以和 MQGMO_LOCK 一起使用,将这条消息锁住,此消息对其他并发应用变为不可见。接着可以用非浏览方式的 MQGET (MQGMO_MSG_UNDER_CURSOR) 将该消息读走。

下面是一段伪代码,用于浏览队列中的消息并取走所需的消息:

```
MQOPEN (MQOO _ BROWSE)
MQGET (MQGMO _ BROWSE _ FIRST + MQGMO _ LOCK)
while (not found)
{
    MQGET (MQGMO _ BROWSE _ NEXT + MQGMO _ LOCK)
    if (found)
    {
        MQGET (MQGMO _ MSG _ UNDER _ CURSOR)
        break ;
    }
}
MQCLOSE
```

5.6.5 格式转换 (Convert)

WebSphere MQ 中的数据格式可以根据需要自动地转换成目标字符集和编码方式。所谓字符集 (CodedCharSetId) 就是指消息的字符所属的文字集,如单字节英语码,双字节汉语码等。编码方式 (Encoding) 指的是整数的高低字节安排、浮点数的精度和幂的安排,等等,通常应该符合 IEEE 标准。

WebSphere MQ 中的格式转换可以出现在 3 个地方:

1. 在 MQGET 的时候,用 MQGMO _ CONVERT 选项,这时 MCA 在接收消息的时候,会根据消息中的 CodedCharSetId 和 Encoding 域与应用程序的环境进行比较,如果需要则自动进行转换。
2. 在发送方的通道设置属性 CONVERT = YES,这时 MCA 在发送消息的时候,会根据消息中的 CodedCharSetId 和 Encoding 域与对应队列管理器的环境进行比较,如果需要则自动进行转换。
3. 在发送方或接收方设置数据转换用户出口,在情况 1 或 2 中需要数据转换时,WebSphere MQ 会根据消息类型 (Type 域) 来调用同名的用户出口程序。

5.6.6 消息匹配 (Match)

在读消息的时候,可以从队列中的一大堆消息中匹配出所需的部分。消息匹配有 3 种方式,可以根据消息 MsgId、CorrelId 或 Message Token 来进行匹配。相应地,MQGMO 的匹配选项应该设置为:MQMO _ MATCH _ MSG _ ID、MQMO _ MATCH _ CORREL _ ID 或 MQMO _ MATCH _ MSG _ TOKEN。如果 MQGMO.Options = MQMO _ NONE,则不进行消息匹配,WebSphere MQ 会按优先级 (Priority) 或先进先出的原则选取第一条消息。

例:

队列中有两条消息:

消息 1 :MQMD.MsgId = "M1", MQMD.CorrelId = "C1"

消息 2 :MQMD.MsgId = "M2", MQMD.CorrelId = "C2"

在 MQGET 的时候,如果参数 MQMD 的 MsgId 域设为“M1”,且 MQGMO.Options = MQMO _ MATCH _ MSG _ ID,则取出消息 1。如果参数 MQMD 的 CorrelId 域设为“C2”,且 MQGMO.Options = MQMO _ MATCH _ CORREL _ ID,则取出消息 2。

5.6.7 回滚计数 (Backout Count)

在交易中,MQGET (MQGMO _ SYNCPOINT) 从队列中读取的每一条消息在交易失败时都会回滚放回队列中。回滚后的消息可以参与下一个交易,然而,下一个交易也可能因为相同的原因而不成功,造成消息再次被回滚。为了防止这种反复回滚现象,WebSphere MQ 设计了回滚计数 (Backout Count) 机制。初始时回滚计数器为零,消息回滚后计数器会自动加一,在下一次 MQGET 时 MQMD.BackoutCount 可以取得计数器的值。当计数器的值达到阈值时,应用程序应该将其做特殊处理,而不是简单地再次回滚。

反复回滚的消息,我们称之为“有毒消息”(Poison Message)。因为这种消息往往会使应用陷入反复无效操作,对性能有害。一般来说,发现“有毒消息”应将其转入特殊处理流程,比如放入指定队列中隔离,同时应及时报警,因为这往往意味着某些资源工作不正常,或者应用设计有漏洞。

在本地队列或模型队列上有两个相关属性 BOTHRESH 和 BOQNAME,分别指明 BackoutCount 的阈值和超过阈值后应该放入的队列名。但是,这两个属性除了设定一个建议值以外,WebSphere MQ 本身并不会做相应的隔离动作。换句话说,WebSphere MQ 自身不会发现并处理“有毒消息”,这是应用应该做的。目前,建立在 WebSphere MQ 之上的一些产品,比如 WebSphere Business Integration Message Broker,WebSphere MQ Workflow 等都已经具有这个功能。

5.6.8 固化回滚计数 (Harden Backout)

既然反复回滚可能是由某些单元工作不正常引起的,那么如果队列管理器自身遇到这样的问题,最简单的办法就是重启队列管理器。回滚计数器是可以跨越队列管理器重启后继续使用的,但是如果队列管理器自身有问题,这一点就无法保证。

相应的解决办法是固化计数 (Harden Backout),在每次回滚时,对每条消息及时地将 BackoutCount 记录写到硬盘上去。本地队列或模型队列有一个属性 HARDENBO (也可以将其设置成 NOHARDENBO),这样,这个计数器就可以在内存和硬盘上各有一份拷贝。对于 Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NSK, UNIX, Windows, BackoutCount 只能是固化的,对 HARDENBO 修改无效。

5.7 容错设计

5.7.1 出错处理

WebSphere MQ 的每一个 API 调用都有可能出错,当错误发生时,调用操作返回的完成码 (Completion Code) 可以指明是警告还是错误,原因码 (Reason Code) 可以指明错误原因。一般来说,每一个 API 调用都需要判断这两个返回码,以决定程序是继续执行还是转去相应地出错处理。如果要中止程序的执行,应该注意关闭所有打开的对象句柄,最后 MQDISC 断开与队列管理器的连接。

对于 Java 程序,可以通过处理各种异常 (Exception) 来简化出错处理的逻辑,使代码更加清晰。通常,一个健壮的程序会考虑到各种出错情况并有相应的出错处理代码,越早发现并处理错误对应用系统的代价越小。所以,监控每一个 API 调用是否成功并及时处理是一个好方法。

5.7.2 报告消息

在网络多队列管理器环境中,消息的传递成功与否不由发送方控制。换句话说,发送方发送消息的操作本身是成功的,这只说明 WebSphere MQ 受理了该消息的传递,可是发送方并不知道消息是否真的能够到达对方并被对方处理。在应用设计时如果发送方需要知道这一点,可以要求得到一个“回执”,这就是报告消息。

报告消息常用的有以下几种:

- 异常 (EXCEPTION) 消息在传递过程中出现异常情况,不可送达。
- 超时 (EXPIRATION) 消息在传递过程中超时,未被对方 MQGET 取走。
- 成功到达 (COA) 消息成功到达目标队列。
- 成功阅读 (COD) 消息成功到达目标队列,并被对方 MQGET 取走。
- 处理成功 (PAN) 消息处理成功
- 处理失败 (NAN) 消息处理失败

报告消息中前 4 种只是描述传递是否成功,由队列管理器发出回执消息。后 2 种描述处理是否成功,由处理程序显式发出回执消息。发送方要求的报告消息选项可以叠加,如 COA + COD,表示消息发出后有

可能收到两个相关回执。

5.7.3 死信消息

当消息无法送达目标队列时会成为死信消息,这时往往意味着应用系统中出现了问题使消息无法正常投递,而这个问题是发送方和接收方都无法感知的。所以,及时地发现并处理死信消息对保证系统正常运作大有好处。

通过配置队列管理器的缺省死信队列使死信消息能够就近存放,通过监控死信队列中的死信消息可以得知死信产生的原因。应用设计者可以选择自行设计并编写死信队列管理程序或使用 WebSphere MQ 缺省的死信队列管理器 `runmqdlq`。

对死信消息的处理通常有 3 种方式:重做、隔离和报告。重做就是将死信消息不成功的操作重做一遍,这对于解决系统临时故障是有效的。比如,消息因为目标队列暂时满了而无法放入,隔一段时间重放一遍就可能成功。再如,消息因为网络原因暂无法发送,隔一段时间重发就可能成功。但是,对消息重做一遍很可能会因为相同的原因不成功而再次成为死信消息,形成死循环,我们在设计上一定要避免这种情况的发生。隔离就是将死信消息放入特定的队列,以便做事后的统计和分析,积累到一定程度可以报警以进行人工干预。报告就是在死信消息隔离同时产生报告消息,而报告消息的目标队列与死信消息的 `ReplyToQ` 域或 `DestQName` 域相同,这样消息的发送方或接收方可以感知到死信消息的产生而做出正确的反应。

5.8 小 结

本章我们详细介绍了 WebSphere MQ 应用设计时要考虑的各个要素,这些要素都是设计之初需要确定下来的。通常设计人员应该先从应用结构、通信方式、实现功能等处入手,设计的方案应该尽可能贴近于应用需求而又保持相当的灵活性,这时尚不需要涉及具体的部署和配置、运行平台、采用何种编程语言等实现细节。

以下我们会对 WebSphere MQ 的各种功能做更加详细的介绍,因为大多数功能需要手工配置或编程实现,请读者注意书中内容与书后例程的结合。