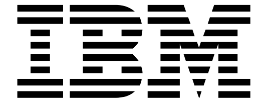
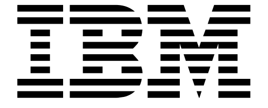
MQSeries®



使用 Java™

SC84-0704-01

MQSeries®



使用 Java™

SC84-0704-01

注意!

在使用本资料和它支持的产品之前，请确保阅读第357页的『附录F. 声明』中的一般信息。

第七版（2001 年 1 月）

本版本适用于 IBM® MQSeries classes for Java 版本 5.2.0 和 MQSeries classes for Java Message Service 版本 5.2，以及所有后续发行版和修订版，除非在新版本中另有声明为止。

© Copyright International Business Machines Corporation 1997, 2001. All rights reserved.

目录

图 vii

表 ix

关于本书 xi
本书中使用的缩写 xi
本书针对的读者 xi
理解本书所需的知识 xi
如何使用本书 xii

更改摘要 xiii
| 对本版的更改（SC84-0704-01） xiii
对第六版的更改（SC84-0704-00） xiv
对第五版的更改（SC84-0704-00） xiv

第1部分 用户指南 1

第1章 入门 3
什么是 MQSeries classes for Java? 3
什么是 MQSeries classes for Java Message Service? . 3
哪些人将使用 MQ Java? 3
连接选项 4
　客户机连接 5
　使用 VisiBroker for Java 6
　绑定连接 6
必备条件 6

| 第2章 安装过程 7
| 安装 MQSeries classes for Java 和 MQSeries classes for Java Message Service 7
| 　在 UNIX 上安装 8
| 　在 AS/400 上安装 9
| 　在 Linux 上安装 9
| 　在 Windows 上安装 10
| 　安装目录 10
| 　环境变量 10
| Web 服务器配置 12

第3章 使用 MQSeries classes for Java（MQ base Java） 13
使用样本小应用程序验证 TCP/IP 客户机 13
　在 AS/400 上使用样本小应用程序 13
　配置队列管理器以接受客户机连接 13
　从小应用程序查看器运行 14
　定制验证小应用程序 15
使用样本应用程序来验证 15
　使用 VisiBroker 连通性 16
　使用 CICS 事务处理服务器 OS/390 版 16
运行自己编写的 MQ base Java 程序 16
解决 MQ base Java 问题 17
　跟踪样本小应用程序 17
　跟踪样本应用程序 17
　错误消息 18

第4章 使用 MQSeries classes for Java Message Service (MQ JMS) 19
安装后设置 19
　“发布／订阅”方式的附加设置 20
| 　非特权用户需要权限的队列 20
运行点到点 IVT 21
　不使用 JNDI 的点到点验证 21
　使用 JNDI 的点到点验证 22
　IVT 错误恢复 24
“发布／订阅”安装验证测试 24
　不使用 JNDI 的发布／订阅验证 25
　使用 JNDI 的发布／订阅验证 26
　PSIVT 错误恢复 26
运行您自己的 MQ JMS 程序 27
解决问题 27
　跟踪程序 27
　日志记录 28

第5章 使用 MQ JMS 管理工具 29
调用管理工具 29
配置 30
　配置 WebSphere 31
　安全性 31
管理命令 32
操纵子上下文 32
管理 JMS 对象 33
　对象类型 33
　与 JMS 对象一起使用的动词 34
　创建对象 35
　特性 36
　特性相关性 39
　ENCODING 特性 39
　样本错误条件 41

第2部分 用 MQ base Java 编程 . . 43

第6章 针对程序员的介绍 45
为何要使用 Java 接口? 45
MQSeries classes for Java 接口 45
Java 开发工具箱 46
MQSeries classes for Java 类库 46

第7章 编写 MQ base Java 程序 49
应编写小应用程序还是应用程序? 49
连接差异 49
　客户机连接 49

© Copyright IBM Corp. 1997, 2001

绑定方式 50
定义要使用的连接 50
示例代码片段 50
示例小应用程序代码 50
示例应用程序代码 54
队列管理器上的操作 56
设置 MQSeries 环境 56
连接到队列管理器 56
访问队列与进程 57
处理消息 57
处理错误 58
获取和设置属性值 59
多线程程序 59
编写用户出口 61
| 连接合用 62
| 控制缺省连接池 62
| 缺省连接池和多个组件 64
| 提供不同的连接池 65
| 提供自己的 ConnectionManager 66
编译和测试 MQ base Java 程序. 67
运行 MQ base Java 小应用程序. 67
运行 MQ base Java 应用程序 68
在 CICS 事务处理服务器 OS/390 版下运行 MQ base Java 应用程序 68
跟踪 MQ base Java 程序 68

第8章 取决于环境的行为. 71
核心类细节 71
核心类的限制与变化 72
运行于其它环境下的版本 5 扩展 74

第9章 MQ base Java 类和接口 77
MQChannelDefinition 78
变量 78
构造器 79
MQChannelExit 80
变量 80
构造器 82
MQDistributionList 83
构造器 83
方法 83
MQDistributionListItem 85
变量 85
构造器 85
MQEnvironment 86
变量 86
构造器 88
方法 89
MQException 91
变量 91
构造器 91
MQGetMessageOptions 93
变量 93
构造器 96
MQManagedObject 97
变量 97
构造器 97
方法 98
MQMessage 100
变量 100
构造器. 108
方法 108
MQMessageTracker 119
变量 119
| MQPoolServices 121
| 构造器. 121
| 方法 121
| MQPoolServicesEvent 122
| 变量 122
| 构造器. 122
| 方法 123
| MQPoolToken 124
| 构造器. 124
MQProcess 125
构造器. 125
方法 125
MQPutMessageOptions 127
变量 127
构造器. 129
MQQueue 130
构造器. 130
方法 130
MQQueueManager 138
变量 138
构造器. 138
方法 140
| MQSimpleConnectionManager 148
| 变量 148
| 构造器. 148
| 方法 148
MQC 150
| MQPoolServicesEventListener 151
| 方法 151
| MQConnectionManager 152
MQReceiveExit 153
方法 153
MQSecurityExit 155
方法 155
MQSendExit 157
方法 157
| ManagedConnection. 159
| 方法 159
| ManagedConnectionFactory 162
| 方法 162
| ManagedConnectionMetaData 164
| 方法 164

第3部分 用 MQ JMS 进行编程 . . . 165

第10章 编写 MQ JMS 程序 167
JMS 模型. 167
构建连接 168

从 JNDI 检索工厂 168
使用工厂创建连接 169
在运行时创建工厂 169
选择客户机或绑定传输 170
获取会话 170
发送消息 171
使用 'set' 方法设置特性 172
消息类型 173
接收消息 173
消息选择器 174
异步传递 175
关闭 175
关闭时“Java 虚拟机”挂起. 175
处理错误 175
异常侦听器 176

第11章 编写“发布／订阅”应用程序 177
编写简单的“发布／订阅”应用程序 177
导入需要的包 177
获取或创建 JMS 对象 177
发布消息 179
接收订阅 179
关闭不想要的资源 179
使用主题 179
主题名称 179
在运行时创建主题 180
订户选项 181
创建非长期订户. 182
创建长期订户 182
使用消息选择器. 182
抑制本地出版物. 182
合并订户选项 183
| 配置基本订户队列 183
解决发布／订阅问题 185
非完整“发布／订阅”关闭 185
处理代理报告 186

第12章 JMS 消息 187
消息选择器 187
将 JMS 消息映射到 MQSeries 消息 191
MQRFH2 头 192
带有相应 MQMD 字段的 JMS 字段和特性. . . 195
将 JMS 字段映射到 MQSeries 字段（外出消息） 196
将 MQSeries 字段映射到 JMS 字段（进入消息） 200
将 JMS 映射到本地 MQSeries 应用程序 . . . 201
消息主体 202

| **第13章 MQ JMS 应用程序服务器设施 205**
| ASF 类和函数 205
| ConnectionConsumer 205
| 规划应用程序 206
| 错误处理 209
| 应用程序服务器样本代码. 211
| MyServerSession.java 213
| MyServerSessionPool.java 213
| MessageListenerFactory.java 214
| ASF 使用示例 214
| Load1.java 215
| CountingMessageListenerFactory.java 216
| ASFClient1.java 216
| Load2.java 218
| LoggingMessageListenerFactory.java 218
| ASFClient2.java 218
| TopicLoad.java 219
| ASFClient3.java 220
| ASFClient4.java 220

第14章 JMS 接口与类 223
Sun Java Message Service 类和接口 223
MQSeries JMS 类 225
BytesMessage 227
方法 227
Connection 235
方法 235
| ConnectionConsumer 238
| 方法 238
ConnectionFactory 239
MQSeries 构造器 239
方法 239
ConnectionMetaData 243
MQSeries 构造器 243
方法 243
DeliveryMode 245
字段 245
Destination 246
MQSeries 构造器 246
方法 246
ExceptionListener 248
方法 248
MapMessage 249
方法 249
Message 257
字段 257
方法 257
MessageConsumer 270
方法 270
MessageListener 272
方法 272
MessageProducer 273
MQSeries 构造器 273
方法 273
MQQueueEnumeration * 277
方法 277
ObjectMessage 278
方法 278
Queue 279
MQSeries 构造器 279
方法 279
QueueBrowser 281
方法 281
QueueConnection 283
方法 283

QueueConnectionFactory 285
MQSeries 构造器 285
方法 285
QueueReceiver 287
方法 287
QueueRequestor 288
构造器. 288
方法 288
QueueSender 290
方法 290
QueueSession. 293
方法 293
Session. 296
字段 296
方法 296
StreamMessage 301
方法 301
TemporaryQueue. 309
方法 309
TemporaryTopic 310
MQSeries 构造器 310
方法 310
TextMessage 311
方法 311
Topic 312
MQSeries 构造器 312
方法 312
TopicConnection. 314
方法 314
TopicConnectionFactory 316
MQSeries 构造器 316
方法 316
TopicPublisher 319
方法 319
TopicRequestor 322
构造器. 322
方法 322
TopicSession 323
MQSeries 构造器 323
方法 323
TopicSubscriber 327
方法 327
XAConnection 328
XAConnectionFactory 329
XAQueueConnection 330
方法 330
XAQueueConnectionFactory 331
方法 331
XAQueueSession 333
方法 333
XASession 334
方法 334
XATopicConnection. 336
方法 336
XATopicConnectionFactory 337
方法 337
XATopicSession 339
方法 339

第4部分 附录 341

附录A. 管理工具特性与可编程特性之间的映射 343

附录B. MQSeries classes for Java Message Service 提供的脚本. 345

附录C. 针对 Java 对象的 LDAP 服务器配置 347
检查 LDAP 服务器配置 347
配置过程 347

附录D. 连接到 MQSeries Integrator V2 349
发布 / 订阅 349
转换和路由 350

附录E. 与 WebSphere 的 JMS JTA/XA 接口. 351
与 WebSphere 的 JMS 接口 351
受管理的对象 351
容器管理的事务与 bean 管理的事务 351
两阶段提交与一阶段优化. 352
定义受管理的对象 352
检索管理对象 352
样本 352
样本 1. 353
样本 2. 353
样本 3. 354

附录F. 声明 357
商标 358

术语与缩写词汇表 359

文献目录. 363
MQSeries 跨平台出版物 363
MQSeries 特定平台出版物 363
软拷贝书籍 364
HTML 格式 364
可移植文档格式 (PDF) 364
BookManager® 格式 365
PostScript 格式 365
Windows 帮助格式. 365
因特网上可用的 MQSeries 信息 365

索引 367

把您的意见发送给 IBM 373

图

1. MQSeries classes for Java 示例小应用程序 51
2. MQSeries classes for Java 样本应用程序 54
3. 主题名称层次结构 180
4. JMS 到 MQSeries 的映射模型 191
5. JMS 到 MQSeries 映射模型 202
6. ServerSessionPool 和 ServerSession 功能 212
7. MQSeries Integrator 消息流 349

© Copyright IBM Corp. 1997, 2001

表

| 1. 平台与连接方式 5
| 2. 产品安装目录 10
| 3. 产品的样本 CLASSPATH 语句 10
| 4. 产品的环境变量 11
| 5. IVT 测试的类 24
6. 管理动词 32
7. 操纵子上下文的命令语法与说明 32
8. 由管理工具处理的 JMS 对象类型 33
9. 操纵管理对象的命令语法与说明 34
10. 特性名与有效值 36
| 11. 特性与对象类型的有效组合 37
12. 核心类限制和变化 72
13. 字符集标识 103
14. MQQueueConnectionFactory 上的设置方法 169
15. 队列 URI 的特性名 172
16. 队列特性的符号值 172
17. JMS 使用的 MQRFH2 文件夹和特性 193
18. 特性数据类型和值 194
19. 映射到 MQMD 字段的 JMS 特性 195
20. 外出消息字段映射 196
21. 进入消息字段映射 200
| 22. Load1 参数和缺省值 215
| 23. ASFClient1 参数和缺省值. 216
| 24. TopicLoad 参数和缺省值 219
| 25. ASFClient3 参数和缺省值. 220
26. 接口摘要 223
27. 类摘要 224
28. 包 'com.ibm.mq.jms' 类摘要 225
29. 包 'com.ibm.jms' 类摘要 226
30. 管理工具内的特性以及可编程等价程序表示法之比较。 343
31. MQSeries classes for Java Message Service 提供的实用程序 345

© Copyright IBM Corp. 1997, 2001

关于本书

本书描述了:

- MQSeries classes for Java，可用来访问 MQSeries 系统。
- MQSeries classes for Java Message Service，可用来访问 Java 消息服务 (JMS) 和 MQSeries 应用程序

注: 本书作为产品一部分，仅以软拷贝格式（PDF 和 HTML）提供，并且可从 MQSeries 系列的 Web 站点上获取，站点位于:

```
http://www.ibm.com/software/mqseries/
```

本书**不能**以印刷书籍订购。

本书中使用的缩写

整本书中使用了以下缩写:

MQ Java
MQSeries classes for Java 和 MQSeries classes for Java Message Service 合在一起

MQ base Java
MQSeries classes for Java

MQ JMS
MQSeries classes for Java Message Service

本书针对的读者

本书是为熟悉 *MQSeries 应用程序设计指南*中描述的过程性 MQSeries 编程接口的程序员所编写的，并演示了如何使用这些知识更有效地使用 MQ Java 编程接口。

理解本书所需的知识

应当具备:

- Java 编程语言知识
- 理解 *MQSeries 应用程序设计指南*中有关消息队列接口的章节以及 *MQSeries Application Programming Reference* 中有关调用描述的章节所描述的消息队列接口 (MQI) 的作用。
- 关于 MQSeries 程序的大体经验，或熟悉其它 MQSeries 出版物内容

用户若要与 CICS® 事务处理服务器 OS/390® 版一起使用 MQ base Java，还应熟悉:

- “客户信息控制系统 (CICS)”的概念
- 使用“CICS Java 应用程序编程接口 (API)”
- 从 CICS 内运行 Java

要使用 VisualAge® for Java 来开发“OS/390 UNIX® 系统服务高性能 Java (HPJ)”应用程序的用户应熟悉 Enterprise Toolkit for OS/390（由 VisualAge for Java 企业版 OS/390 版版本 2 提供）。

如何使用本书

本书的第 1 部分描述了 MQ base Java 和 MQ JMS 的用法，第 2 部分为希望使用 MQ base Java 的程序员提供了帮助，第 3 部分为希望使用 MQ JMS 的程序员提供了帮助。

首先，请阅读第 1 部分中的章节，其中介绍了 MQ base Java 和 MQ JMS。然后，请阅读第 2 或第 3 部分，理解如何在要使用的环境中用类来发送与接收 MQSeries 消息。

本书的后面有一个词汇表和书目。

更改摘要

这一节将描述 *MQSeries 使用 Java* 这一版本中的更改。自本书上一版本以来的更改用这些更改左侧的竖线标记。

对本版的更改（SC84-0704-01)

本版包含了 MQ Java V5.2 引入的新功能的更新。包括:

- 对安装过程的更新。请参阅第7页的『第2章 安装过程』。
- 支持连接合用，能够改进对 MQSeries 队列管理器使用多个连接的应用程序与中间件的性能。请参阅:
  - 第62页的『连接合用』
  - 第86页的『MQEnvironment』
  - 第121页的『MQPoolServices』
  - 第122页的『MQPoolServicesEvent』
  - 第124页的『MQPoolToken』
  - 第138页的『MQQueueManager』
  - 第148页的『MQSimpleConnectionManager』
  - 第152页的『MQConnectionManager』
  - 第151页的『MQPoolServicesEventListener』
  - 第159页的『ManagedConnection』
  - 第162页的『ManagedConnectionFactory』
  - 第164页的『ManagedConnectionMetaData』
- 新的订户队列配置选项，为发布／订阅应用程序提供了多队列和共享队列的方式。请参阅:
  - 第36页的『特性』
  - 第183页的『配置基本订户队列』
  - 第312页的『Topic』
  - 第316页的『TopicConnectionFactory』
- 新的订户清除实用程序，避免了因订户对象非正常结束引起的任何问题。请参阅第185页的『订户清除实用程序』。
- 支持“应用程序服务器设施”，可以并行处理消息。请参阅:
  - 第205页的『第13章 MQ JMS 应用程序服务器设施』
  - 第238页的『ConnectionConsumer』
  - 第283页的『QueueConnection』
  - 第296页的『Session』
  - 第314页的『TopicConnection』
- 对 LDAP 服务器配置信息的更新。请参阅第347页的『附录C. 针对 Java 对象的 LDAP 服务器配置』。

© Copyright IBM Corp. 1997, 2001

- 支持使用 X/Open XA 协议的分布式事务。即，MQ JMS 包含 XA 类以便 MQ JMS 能参与到由适当的事务管理器协调的二阶段提交中。请参阅:
  - 第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』
  - 第328页的『XAConnection』
  - 第329页的『XAConnectionFactory』
  - 第330页的『XAQueueConnection』
  - 第331页的『XAQueueConnectionFactory』
  - 第333页的『XAQueueSession』
  - 第334页的『XASession』
  - 第336页的『XATopicConnection』
  - 第337页的『XATopicConnectionFactory』
  - 第339页的『XATopicSession』

对第六版的更改（SC84-0704-00)

包括 Linux 支持。

对第五版的更改（SC84-0704-00)

WebSphere™ 和 MQSeries Integrator V2 支持

MQ base Java 版本 5.1.2 现已作为产品扩展而可用。它提供以下能力:

- 连接到 MQSeries Integrator Windows NT® 版，版本 2.0 以支持“发布／订阅”。详细信息，请参阅第349页的『附录D. 连接到 MQSeries Integrator V2』。
- 使用 WebSphere 的 CosNaming JNDI 服务供应商。详细信息，请参阅第30页的『配置』。

第1部分 用户指南

第1章 入门 3
什么是 MQSeries classes for Java？ 3
什么是 MQSeries classes for Java Message Service？ . 3
哪些人将使用 MQ Java？ 3
连接选项. 4
  客户机连接 5
  使用 VisiBroker for Java 6
  绑定连接. 6
必备条件. 6

第2章 安装过程 7
安装 MQSeries classes for Java 和 MQSeries classes for Java Message Service 7
  在 UNIX 上安装 8
  在 AS/400 上安装. 9
  在 Linux 上安装 9
  在 Windows 上安装. 10
  安装目录 10
  环境变量 10
Web 服务器配置. 12

第3章 使用 MQSeries classes for Java（MQ base Java） 13
使用样本小应用程序验证 TCP/IP 客户机 13
  在 AS/400 上使用样本小应用程序 13
  配置队列管理器以接受客户机连接 13
    TCP/IP 客户机 13
  从小应用程序查看器运行 14
  定制验证小应用程序 15
使用样本应用程序来验证 15
  使用 VisiBroker 连通性 16
  使用 CICS 事务处理服务器 OS/390 版 16
运行自己编写的 MQ base Java 程序 16
解决 MQ base Java 问题 17
  跟踪样本小应用程序 17
  跟踪样本应用程序 17
    CICS 事务处理服务器 OS/390 版的跟踪 . . . 17
  错误消息 18

第4章 使用 MQSeries classes for Java Message Service (MQ JMS) 19
安装后设置 19
  “发布／订阅”方式的附加设置 20
  非特权用户需要权限的队列 20
运行点到点 IVT 21
  不使用 JNDI 的点到点验证 21
  使用 JNDI 的点到点验证 22
  IVT 错误恢复. 24
“发布／订阅”安装验证测试 24
  不使用 JNDI 的发布／订阅验证. 25
  使用 JNDI 的发布／订阅验证 26
  PSIVT 错误恢复 26
运行您自己的 MQ JMS 程序 27
解决问题 27
  跟踪程序 27
  日志记录 28

第5章 使用 MQ JMS 管理工具 29
调用管理工具. 29
配置. 30
  配置 WebSphere 31
  安全性 31
管理命令 32
操纵子上下文. 32
管理 JMS 对象 33
  对象类型 33
  与 JMS 对象一起使用的动词. 34
  创建对象 35
    LDAP 命名考虑事项 35
  特性. 36
  特性相关性 39
  ENCODING 特性 39
  样本错误条件. 41

© Copyright IBM Corp. 1997, 2001

第1章 入门

本章概述了 MQSeries classes for Java 和 MQSeries classes for Java Message Service，以及它们的用法。

什么是 MQSeries classes for Java?

MQSeries classes for Java (MQ base Java) 允许以 Java 编程语言编写程序:

- 作为 MQSeries 客户机连接到 MQSeries
- 直接连接到 MQSeries 服务器

它支持 Java 小应用程序、应用程序以及小服务程序对 MQSeries 发出调用并查询。这将实现对大型机和旧有应用程序的访问，通常是通过因特网，而不需要在客户机机器上有任何其它 MQSeries 代码。使用 MQ base Java 后，因特网终端用户将能够真正地成为事务的参与者，而不仅仅是信息的提供者和接收者。

什么是 MQSeries classes for Java Message Service?

MQSeries classes for Java Message Service (MQ JMS) 是一组 Java 类，它们实现了 Sun 的 Java 消息服务 (JMS) 接口，以启用 JMS 程序来访问 MQSeries 系统。同时支持 JMS 的点到点和发布与订阅模式。

使用 MQ JMS 作为 API 来编写 MQSeries 应用程序有很多好处。某些优点源于 JMS 是一个有多个实现的开放式标准。其它优点则是因为 MQ JMS 中（而不是在 MQ base Java 中）存在的附加功能。

使用开放式标准带来的好处包括:
- 保护投资，无论是技术还是应用程序代码
- 人员在 JMS 应用程序编程熟练方面的可用性
- 插入不同的 JMS 实现以符合不同需求的能力

有关 JMS API 优点的详细信息，请查看 Sun 公司的 Web 站点: `http://java.sun.com`。

由 MQ base Java 提供的额外功能包括:
- 异步消息传递
- 消息选择器
- 支持发布／订阅消息传递
- 结构化的消息类

哪些人将使用 MQ Java?

若您的企业符合下列方案中的任一个，则使用 MQSeries classes for Java 和 MQSeries classes for Java Message Service 将为您带来极大优势:

© Copyright IBM Corp. 1997, 2001

哪些人将使用 MQ Java

- 引入基于企业内部网的客户机／服务器方案的大中型企业。这里，因特网技术为全球通信提供了低成本的简单访问，而 MQSeries 连通性提供了具有确保传递、时间无关性的高度集成性。
- 需要与伙伴企业进行可靠的商家到商家通信的大中型企业。同样，因特网技术为全球通信提供了低成本的简单访问，MQSeries 连通性提供了传递确定性、时间无关性和高度完整性。
- 希望从公共因特网对其企业的某些应用程序提供访问的大中型企业。这里，因特网提供了低成本的全球连通，而 MQSeries 连通性通过排队范例提供了高度的集成性。除低成本外，通过 24 小时全天开通，快速响应和改进的准确度，企业可提高用户的满意程度。
- 因特网服务供应商或其它增值网络供应商。这些公司能够利用由因特网提供的低成本和方便通信。它们还可以用 MQSeries 连通性所提供的高度集成性的增值。利用 MQSeries 的因特网服务供应商能够立即从 Web 浏览器确认输入数据的接收、确保传递并为 Web 浏览器的用户提供一种监视消息状态的简单方法。

MQSeries 和 MQSeries classes for Java Message Service 提供了访问企业应用和开发复杂 Web 应用的绝佳基础设施。来自 Web 浏览器的服务可以被排队然后有可能则处理，这样，最终用户能及时地获取响应而不必考虑系统的负载如何。通过将队列以网络条件“贴近”用户，网络上的负载将不影响响应的及时性。同样，MQSeries 消息传递的事务性本质表明来自浏览器的简单请求可以以事务性方法安全地扩充到单独的后台进程序列。

MQSeries classes for Java 还能使应用开发人员利用 Java 编程语言的强大能力创建能够在任何支持 Java 运行时环境的平台上运行小应用程序和应用程序。这些因素组合在一起将显著地减少开发多平台 MQSeries 应用程序的时间。而且，如果将来小应用程序功能有所增强，那么在下载小应用程序代码时用户将自动获得这些增强功能。

连接选项

可编程选项允许 MQ Java 以下列方法之一连接到 MQSeries:

- 作为使用传输控制协议／网际协议 (TCP/IP) 的 MQSeries 客户机
- 以绑定方式，直接连接到 MQSeries

Windows NT 上的 MQ base Java 还可以使用 VisiBroker for Java 来连接。第5页的表1显示了每个平台可以使用的连接方式。

表 1. 平台与连接方式

服务器平台	连接方式		
	客户机		绑定
	标准	VisiBroker	
Windows NT	是	是	是
Windows® 2000	是	否	是
AIX®	是	否	是
Sun OS（v4.1.4 和更早版本）	是	否	否
Sun Solaris（v2.6、v2.8、V7 或 SunOS v5.6、v5.7)	是	否	是
OS/2®	是	否	是
OS/400®	是	否	是
HP-UX	是	否	是
AT&T GIS UNIX	是	否	否
SINIX 和 DC/OSx	是	否	否
OS/390	否	否	是
Linux	是	否	否

注:

1. HP-UX Java 绑定支持仅可用于运行 MQSeries 的 POSIX draft10 pthreaded 版本的 HP-UXv11 系统。还需要使用 HP-UX 开发工具箱 Java 版 1.1.7 (JDK™)，发行版 C.01.17.01 或更高版本。
2. 在 HP-UXv10.20、Linux、Windows 95 和 Windows 98 上，仅支持 TCP/IP 客户机连通性。

以下部分更详细地描述了这些选项。

客户机连接

要把 MQ Java 作为 MQSeries 客户机来使用，可以将它安装在可能还包含了 Web 服务器的 MQSeries 服务器机器上，或是安装在单独的机器上。把 MQ Java 和 Web 服务器安装在同一台机器上有一个优点，即可以在本地没有安装 MQ Java 的机器上下载和运行 MQSeries 客户机。

无论选择在哪里安装客户机，都能以三种不同的方式运行它:

从任何支持 Java 的 Web 浏览器内

在这种方式下，可以访问到的 MQSeries 队列管理器的位置受到所使用浏览器安全性限制的限制。

使用小应用程序查看器

要使用这个方式，客户机机器上必须装有 Java 开发工具箱 (JDK) 或 Java 运行时环境 (JRE)。

作为独立的 Java 程序或在 Web 应用程序服务器中

要使用这个方式，客户机机器上必须装有 Java 开发工具箱 (JDK) 或 Java 运行时环境 (JRE)。

使用 VisiBroker for Java

在 Windows 平台上，提供了使用 VisiBroker 的连接，以作为使用标准 MQSeries 客户机协议的一种替代方法。这项支持由 VisiBroker for Java 以及 Netscape Navigator 一同提供，并要求 MQSeries 服务器机器上有 VisiBroker for Java 以及 MQSeries 对象服务器。与 MQ base Java 一起提供了一个适合的对象服务器。

绑定连接

以绑定方式使用时，MQ Java 使用 Java 本地接口 (JNI) 直接调用现有的队列管理器 API，而不是通过网络进行通信。这为 MQSeries 应用程序提供了比使用网络连接更好的性能。与客户机方式不同的是，使用绑定方式编写的应用程序不能作为小应用程序下载。

要使用绑定连接，MQ Java 必须安装在 MQSeries 服务器上。

必备条件

要运行 MQ base Java，需要有以下软件:

- 用于希望使用的服务器平台的 MQSeries。
- 用于服务器平台的 Java 开发工具箱 (JDK)。
- 用于客户机平台的 Java 开发工具箱、Java 运行时环境 (JRE) 或支持 Java 的 Web 浏览器。（请参阅第5页的『客户机连接』。）

  注: 要在 Web 浏览器内运行 MQ base Java 小应用程序（例如安装验证程序），则需要一个能运行 Java 1.1.6 小应用程序的浏览器。Sun 公司的 HotJava™、Netscape Navigator 4 以及 Microsoft® Internet Explorer 4 等浏览器都能满足这一需求。
- VisiBroker for Java（仅当运行在使用 VisiBroker 连接的 Windows 上）。
- 对于 OS/390，带 UNIX 系统服务的 OS/390 版本 2 发行版 5。
- 对于 OS/400，AS/400® Developer Kit for Java，5769-JV1，以及 Qshell Interpreter、OS/400 (5769-SS1) Option 30。

要使用 MQ JMS 管理工具（见第29页的『第5章 使用 MQ JMS 管理工具』），需要以下附加软件:

- 至少有下列服务供应商包之一:
  - 轻量级目录访问协议 (LDAP) - `ldap.jar`、`providerutil.jar`。
  - 文件系统 - `fscontext.jar`、`providerutil.jar`。
- Java 命名与目录服务 (JNDI) 服务供应商。这是一个存储受管理对象的物理表示法的资源。 MQ JMS 的用户可能将 LDAP 服务器用于这一目的，但是该工具也支持文件系统上下文服务供应商。如果使用 LDAP 服务器，则必须配置它以存储 JMS 对象。有关此配置的帮助信息，请参考第347页的『附录C. 针对 Java 对象的 LDAP 服务器配置』。

要使用 MQ JMS XOpen/XA 设施，需要 MQSeries V5.2。

第2章 安装过程

本章描述如何安装 MQSeries classes for Java 和 MQSeries classes for Java Message Service 产品。

安装 MQSeries classes for Java 和 MQSeries classes for Java Message Service

此产品可用在 AIX、AS/400、HP-UX、Linux、Sun Solaris 和 Windows 平台上。它包括:

- MQSeries classes for Java（MQ base Java）版本 5.2.0
- MQSeries classes for Java Message Service（MQ JMS）版本 5.2（非 AS/400）

关于每个特定平台上可用的连通性，请参考第4页的『连接选项』。

此产品是以压缩格式文件提供的，可以从 MQSeries Web 站点 `http://www.ibm.com/software/mqseries/` 上获取。

注: 对于 OS/390，MQ base Java是yi 2 MQSeries SupportPac™ 提供的，可以从 `http://www.ibm.com/software/mqseries/` 上下载。

如果只需要最新版本的 MQ base Java 类，可以单独安装 MQ base Java 版本 5.2.0。要使用 MQ JMS 应用程序，则必须同时安装 MQ base Java 和 MQ JMS（合起来称为 MQ Java）。

MQ base Java 包含在下列 Java `.jar` 文件中:

com.ibm.mq.jar
该代码包括对所有连接选项的支持。

com.ibm.mq.iiop.jar
该代码只支持 VisiBroker 连接。只在 Windows 平台上提供。

com.ibm.mqbind.jar
该代码仅支持绑定连接并且所有平台都不提供或不支持它。建议您不要在任何新的应用程序中使用它。

MQ JMS 包含在以下 Java `.jar` 文件中:

com.ibm.mqjms.jar

在 MQ JMS 产品中重新分布了来自 Sun 公司的下列 Java 库:

connector.jar	版本 1.0 公开草稿
fscontext.jar	早期访问 4 发行版
jms.jar	版本 1.0.2
jndi.jar	版本 1.1.2
ldap.jar	版本 1.0.3
providerutil.jar	版本 1.0

© Copyright IBM Corp. 1997, 2001

注: OS/390 上只提供了 **com.ibm.mq.jar** 文件。该文件同时支持从 UNIX 系统服务以及 CICS 事务处理服务器 OS/390 版至 MQSeries 的绑定连接。

有关安装指示，请参阅与所需平台相关的部分:

AIX、HP-UX 和 Sun Solaris 『在 UNIX 上安装』

AS/400 第9页的『在 AS/400 上安装』

Linux 第9页的『在 Linux 上安装』

Windows 第10页的『在 Windows 上安装』

安装完成时，文件与样本都安装在第10页的『安装目录』中显示的位置上。

安装之后，必须更新环境变量，如第10页的『环境变量』中所示。

注: 如果在安装产品后再安装或重新安装基本 MQSeries，请多加小心。确保没有安装 MQ base Java 版本 5.1，因为 MQSeries Java 支持将回到上一级别。

在 UNIX 上安装

这一部分将描述如何在 AIX、 HP-UX 和 Sun Solaris 上安装 MQ Java。有关在 Linux 上安装 MQ base Java 的信息，请参阅第9页的『在 Linux 上安装』。

注: 如果只安装客户机（也就是说，"不"安装 MQSeries 服务器），则必须设置组和用户标识 mqm。详细信息，请参阅相关平台的 MQSeries 快速入门手册。

1. 作为 Root 登录。
2. 以二进制格式复制文件 `ma88_xxx.tar.Z`，并把它存储在 /tmp 目录下，其中 *xxx* 表示适当的平台标识:
   - `aix` AIX
   - `hp10` HP-UXv10
   - `hp11` HP-UXv11
   - `sol` Sun Solaris
3. 输入下列命令（其中 *xxx* 表示适当的平台标识）:
   ```
   uncompress -fv /tmp/ma88_xxx.tar.Z
   tar -xvf /tmp/ma88_xxx.tar
   rm /tmp/ma88_xxx.tar
   ```
   这些命令将创建必需的文件与目录。
4. 使用对应于每个平台的安装工具:
   - 对于 AIX，使用 `smitty` 并:
     a. 卸载所有以 mqm.java 开头的组件。
     b. 从 /tmp 目录安装组件。
   - 对于 HP-UX，使用 `sam` 并从合适的文件 `ma88_hp10` 或 `ma88_hp11` 安装:

     注: Java 不支持代码页 1051（这是 HP-UX 的缺省值）。要在 HP-UX 上运行"发布／订阅"代理，可能需要把代理的队列管理器的 CCSID 更改为其它值，例如 819:
   - 对于 Sun Solaris，输入以下这个命令并选择所需的选项:
     ```
     pkgadd -d /tmp mqjava
     ```

然后，输入以下命令:

```
rm -R /tmp/mqjava
```

在 AS/400 上安装

这一部分将描述如何在 AS/400 上安装 MQ base Java。

1. 将文件 ma88_400.zip 复制到 PC 上的某一个目录。
2. 使用 InfoZip 的 Unzip 工具解开此文件。

   将创建文件 ma88_400.sav。
3. 在 AS/400 上适当的库中创建一个名为 MA88 的保存文件:
   ```
   CRTSAVF FILE(QGPL/MA88)
   ```
4. 将 ma88_400.sav 作为二进制映象传送到该保存文件中。如果使用 FTP 传送，则 put 命令类似于:
   ```
   PUT C:\TEMP\MA88_400.SAV QGPL/MA88
   ```
5. 安装 MQSeries classes for Java，产品标识 5648C60，通过使用 RSTLICPGM:
   ```
   RSTLICPGM LICPGM(5648C60) DEV(*SAVF) SAVF(QGPL/MA88)
   ```
6. 删除在步骤3中创建的保存文件:
   ```
   DLTF FILE(QGPL/MA88)
   ```

在 Linux 上安装

这一部分将描述如何在 Linux 上安装 MQ Java。

对于 Linux，可以使用两个安装文件 ma88_linux.tgz 和 MQSeriesJava-5.2.0-1.noarch.rpm。每个文件提供的安装是等同的。

如果对目标系统具有 root 访问权，或是使用 Red Hat Package Manager (RPM) 数据库来安装包，请使用 MQSeriesJava-5.2.0-1.noarch.rpm。

如果对目标系统不具有 root 访问权，或目标系统上没有安装 RPM，则使用 ma88_linux.tgz。

要使用 ma88_linux.tgz 进行安装，请:

1. 选择产品的安装目录（例如 /opt）。

   如果该目录不在主目录中，可能需要以 root 登录。
2. 将文件 ma88_linux.tgz 复制到主目录中。
3. 将目录更改为选定的安装目录，例如:
   ```
   cd /opt
   ```
4. 输入以下命令:
   ```
   tar -xpzf ~/ma88_linux.tgz
   ```

   这将创建一个名为 mqm 的目录并且该目录位于当前目录（例如 /opt）下。

要使用 MQSeriesJava-5.2.0-1.noarch.rpm 进行安装，请:

1. 以 root 登录。
2. 将 MQSeriesJava-5.2.0-1.noarch.rpm 复制到工作目录中。
3. 输入以下命令:

```
rpm -i MQSeriesJava-5.2.0-1.noarch.rpm
```

这样，产品将安装到 /opt/mqm/ 中，也可能安装到另一路径下（更详细信息，请参考 RPM 文档）。

在 Windows 上安装

这一部分将描述如何在 Windows 上安装 MQ Java。

1. 创建名为 tmp 的空目录并使它成为当前目录。
2. 将文件 `ma88_win.zip` 复制到该目录。
3. 使用 InfoZip 的 Unzip 工具解开 `ma88_win.zip`。
4. 从该目录运行 `setup.exe` 并遵循结果窗口上的提示。

注： 如果只想安装 MQ base Java，请选择这个阶段上的相关选项。

安装目录

MQ Java V5.2 文件安装在表2显示的目录中。

表 2. 产品安装目录

平台	目录
AIX	usr/mqm/java/
AS/400	/QIBM/ProdData/mqm/java/
HP-UX 和 Sun Solaris	opt/mqm/java/
Linux	*install_dir*/mqm/java/
Windows 95、98、2000 和 NT	*install_dir*\
注： *install_dir* 是安装了产品的目录。在 Linux 上，可能就是 /opt。	

环境变量

安装后，必须更新 CLASSPATH 环境变量，使它包含 MQ base Java 代码和样本目录。表3中显示了各种平台上典型的 CLASSPATH 设置。

表 3. 产品的样本 CLASSPATH 语句

平台	样本 CLASSPATH
AIX	CLASSPATH=*jdk_dir*/lib/classes.zip: /usr/mqm/java/lib/com.ibm.mq.jar: /usr/mqm/java/lib/connector.jar: /usr/mqm/java/lib: /usr/mqm/java/samples/base:
HP-UX 和 Sun Solaris	CLASSPATH=*jdk_dir*/lib/classes.zip: /opt/mqm/java/lib/com.ibm.mq.jar: /opt/mqm/java/lib/connector.jar: /opt/mqm/java/lib: /opt/mqm/java/samples/base:

表 3. 产品的样本 CLASSPATH 语句 (续)

平台	样本 CLASSPATH
Windows 95、98、2000 和 NT	CLASSPATH=C:*jdk_dir*\lib\classes.zip; *install_dir*\lib\com.ibm.mq.jar; *install_dir*\lib\com.ibm.mq.iiop.jar; *install_dir*\lib\connector.jar; *install_dir*\lib\; *install_dir*\samples\base\;
AS/400	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/lib/connector.jar: /QIBM/ProdData/mqm/java/lib: /QIBM/ProdData/mqm/java/samples/base:
Linux	CLASSPATH=*jdk_dir*/lib/classes.zip: *install_dir*/mqm/java/lib/com.ibm.mq.jar: *install_dir*/mqm/java/lib/connector.jar: *install_dir*/mqm/java/lib: *install_dir*/mqm/java/samples/base:
注: 1. *jdk_dir* 是安装 JDK 的目录 2. *install_dir* 是安装产品的目录。	

要使用 MQ JMS，必须在 classpath 中包含附加的 jar 文件。 第19页的『安装后设置』中列出了这些文件。

如果存在与并未建议使用的包 `com.ibm.mqbind` 相关的现有应用程序，则还必须在 classpath 中添加文件 `com.ibm.mqbind.jar`。

必须按表4中所示更新某些平台上的环境变量.

表 4. 产品的环境变量

平台	环境变量
AIX	LD_LIBRARY_PATH=/usr/mqm/java/lib
HP_UX	SHLIB_PATH=/opt/mqm/java/lib
Sun Solaris	LD_LIBRARY_PATH=/opt/mqm/java/lib
Windows 95、98、2000 和 NT	PATH=*install_dir*\lib
注: *install_dir* 是产品的安装目录	

注:

1. 要在 OS/400 上使用 MQSeries Bindings for Java，请确保库 QMQMJAVA 在库列表中。
2. 确保添加了 MQSeries 变量并且没有覆盖任何现有系统环境变量。如果覆盖现有的系统环境变量，则应用程序在编译或运行期间可能会发生故障。

Web 服务器配置

如果要在 Web 服务器上安装 MQSeries Java, 您可以在本地没有安装 MQSeries Java 的机器上下载并运行 MQSeries Java 应用程序。要使您的 Web 服务器能够访问 MQSeries Java 文件, 必须把 Web 服务器配置设置为指向装有客户机的目录。有关如何配置它的详细信息, 请查询 Web 服务器文档。

注: 在 OS/390 上, 已安装的类不支持客户机连接并且无法有效地下载到客户机。但是, 其它平台上的 jar 文件可以被传送到 OS/390 上并提供给客户机使用。

第3章 使用 MQSeries classes for Java（MQ base Java）

本章描述:

- 如何配置系统以运行样本小应用程序和应用程序，以便验证 MQ base Java 安装
- 如何修改运行自己程序的过程

该过程取决于希望使用的连接选项。按照这一部分中符合您要求的指示执行。

使用样本小应用程序验证 TCP/IP 客户机

MQ base Java 中包含了一个安装验证小应用程序 mqjavac.html。可以使用这个小应用程序来验证 MQ base Java 的 TCP/IP 连接客户机方式。（另见第15页的『使用样本应用程序来验证』。）

如果存在任何故障，小应用程序将连接到一个给定的队列管理器，练习所有的 MQSeries 并产生诊断消息。

可以使用 JDK 中提供的小应用程序查看器来运行小应用程序。小应用程序查看器可访问任何主机上的队列管理器。

在所有情况下，如果小应用程序未能成功完成，请按照诊断信息中给出的建议做，并再次尝试运行小应用程序。

在 AS/400 上使用样本小应用程序

OS/400 操作系统没有本地图形用户界面 (GUI)。要运行样本小应用程序，需在支持的图形功能硬件上运行 Remote Abstract Window Toolkit for Java （AWT) 或 Class Broker for Java (CBJ)。也可以从命令行验证客户机（请参阅第15页的『使用样本应用程序来验证』)。

配置队列管理器以接受客户机连接

请使用下列过程配置队列管理器，以便接受来自客户机的进入连接请求。

TCP/IP 客户机

1. 使用以下过程定义服务器连接通道:

   在非 AS/400 平台上:

   a. 使用 strmqm 命令启动队列管理器。
   b. 输入以下命令以启动 runmqsc 程序:

   ```
   runmqsc
   ```

   c. 定义一个称为 JAVA.CHANNEL 的样本通道，输入:

   ```
   DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
   DESCR('Sample channel for MQSeries Client for Java')
   ```

© Copyright IBM Corp. 1997, 2001

对于 AS/400 平台:

a. 通过使用 STRMQM 命令启动队列管理器。
b. 定义一个称为 JAVA.CHANNEL 的样本通道，通过输入:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
MCAUSERID(SOMEUSERID) TEXT('Sample channel for MQSeries Client for Java')
```

其中 QMGRNAME 表示队列管理器的名称，SOMEUSERID 表示对 MQSeries 具有适当权限的 AS/400 用户标识。

2. 用下列命令启动侦听器:

对于 OS/2 与 NT 操作系统:

发出命令:

```
runmqlsr -t tcp [-m QMNAME] -p 1414
```

注: 如果使用的是缺省队列管理器，则可以省略 -m 选项。

在 Windows NT 操作系统上使用 VisiBroker for Java:

使用以下这个命令启动 IIOP（网际 ORB 间协议）:

```
java com.ibm.mq.iiop.Server
```

注: 要停止 IIOP 服务器，请发出下列命令:

```
java com.ibm.mq.iiop.samples.AdministrationApplet shutdown
```

对于 UNIX 操作系统:

配置 inetd 守护程序，以便 inetd 启动 MQSeries 通道。有关如何执行的指示，请参阅 *MQSeries 客户机*。

对于 OS/400 操作系统:

发出命令:

```
STRMQMLSR MQMNAME(QMGRNAME)
```

其中 QMGRNAME 表示队列管理器的名称。

从小应用程序查看器运行

要使用此方式，机器上必须安装了 Java 开发工具箱 (JDK)。

本地安装过程

1. 更改到您所使用语言的样本目录:
2. 输入:

```
appletviewer mqjavac.html
```

Web 服务器安装过程:

输入命令;

```
appletviewer http://Web.server.host/MQJavaclient/mqjavac.html
```

注:

1. 在某些平台上，该命令为 'applet'，而不是 'appletviewer'。
2. 在某些平台上，可能需要在屏幕顶部左边的 '小应用程序' 菜单上选择“特性”，并将“网络访问”设置为“不限制”。

通过使用这种技术，可以连接到可用 TCP/IP 访问的在任何主机上运行的任何队列管理器。

定制验证小应用程序

mqjavac.html 文件中包含了一些可选参数。这些参数允许您修改小应用程序，使之符合您的需求。每个参数都用一行 HTML 定义，如下所示:

```
<!PARAM name="xxx" value="yyy">
```

要指定参数值，请删除原来的感叹号，然后按需编辑其值。可以指定以下参数:

hostname 主机名编辑框中最初显示的值。

port 端口号编辑框中最初显示的值。

channel 通道编辑框中最初显示的值。

queueManager
队列管理器编辑框中最初显示的值。

userID 连接到队列管理器时使用的指定的用户标识。

password 连接到队列管理器时使用的指定的口令。

trace 导致 MQ base Java 写一个跟踪日志。仅在 IBM 服务的指导下才能使用该选项。

使用样本应用程序来验证

MQ base Java 提供了一个安装验证程序 MQIVP。您可以使用这个应用程序来测试 MQ base Java 的所有连接方式。程序会提示一些选项和其它数据，以确定要验证的连接方式。使用下列过程来验证安装:

1. 要测试客户机连接:
   a. 按第13页的『配置队列管理器以接受客户机连接』中描述的内容配置队列管理器。
   b. 在客户机机器上实现此过程的余下部分。

   要测试绑定连接，请在 MQSeries 服务器机器上实现此过程的余下部分。
2. 更改到样本目录。
3. 输入:
   ```
   java MQIVP
   ```
   程序尝试:
   a. 连接到命名的队列管理器，并从命名的队列管理器断开.
   b. 打开、放入、取出并关闭系统缺省本地队列。
   c. 如果操作成功，则返回一条消息。
4. 在提示 (1) 处，保留缺省的 'MQSeries'。
5. 在提示 (2) 处:
   - 若使用 TCP/IP 连接，则输入 MQSeries 服务器主机名。
   - 若使用本机连接（绑定方式），则将字段保留为空白。（请勿输入名称。）

这里有一个可能会看到的提示及响应的例子。实际的提示和响应取决于 MQSeries 网络。

```
Please enter the type of connection (MQSeries)          : (MQSeries)(1)
Please enter the IP address of the MQSeries server      : myhost(2)
Please enter the port to connect to                     : (1414)(3)
Please enter the server connection channel name         : JAVA.CHANNEL(3)
Please enter the queue manager name                     :
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager

Tests complete -
SUCCESS: This transport is functioning correctly.
Press Enter to continue...
```

注:

1. 如果选择服务器连接，则不会看到标记为 (3) 的提示。
2. 在 OS/390 上，不会看到提示 (1)、(2) 或 (3)。
3. 在 OS/400 上，只能从 Qshell 交互式界面（Qshell 是 OS/400, 5769-SS1 的选项 30）运行命令 `java MQIVP`。或者，可以通过使用 CL 命令 `RUNJVA CLASS(MQIVP)` 来运行应用程序。
4. 要在 OS/400 上使用 MQSeries Java 绑定，必须确保库列表中存在库 QMQMJAVA。

使用 VisiBroker 连通性

如果使用 VisiBroker，则第13页的『配置队列管理器以接受客户机连接』中描述的过程不是必需的。

要测试使用 VisiBroker 的安装，请使用第15页的『使用样本应用程序来验证』中描述的信息，但是在提示 (1) 中，请使用正确的大小写输入 `VisiBroker`。

使用 CICS 事务处理服务器 OS/390 版

1. 为 CICS 定义样本应用程序。
2. 定义将运行样本应用程序的事务。
3. 将队列管理器名称放入用于标准输入的文件。
4. 运行该事务。

程序输出被放入用于标准和错误输出的文件。

有关运行 Java 程序和设置输入与输出文件的更详细信息，请参考 CICS 文档。

运行自己编写的 MQ base Java 程序

要运行自己编写的 Java 小应用程序或应用程序，请使用为验证程序描述的过程，将 ’mqjavac.html’ 或 ’MQIVP’ 替换成自己编写的应用程序名。

有关编写 MQ base Java 应用程序和小应用程序的信息，请参阅第43页的『第2部分 用 MQ base Java 编程』。

解决 MQ base Java 问题

若程序未能成功完成，运行安装验证小应用程序或安装验证程序，并遵循诊断消息中给出的建议。第13页的『第3章 使用 MQSeries classes for Java（MQ base Java）』中描述了这两个程序。

若问题仍然存在并需要联系 IBM 服务部门，可能需要打开跟踪设施。要执行的方法取决于是以客户机方式还是绑定方式运行。有关系统的适当过程，请参考以下部分。

跟踪样本小应用程序

要使用样本小应用程序来运行跟踪，编辑 mqjavac.html 文件。找出以下这一行:

```
<!PARAM name="trace" value="1">
```

删除感叹号，并根据要求的具体级别，将值 1 更改为 1 至 5 中的一个数。（数字越大，所收集的信息越多。）然后，这一行为:

```
<PARAM name="trace" value="n">
```

其中，'n' 是 1 到 5 之间的一个数字。

跟踪输出出现在 Java 控制台或 Web 浏览器的 Java 日志文件中。

跟踪样本应用程序

要跟踪 MQIVP 程序，请输入:

```
java MQIVP -trace n
```

其中，'n' 是 1 到 5 之间的一个数字，取决于需要的细节级别。（数字越大，所收集的信息越多。）

| 有关如何使用跟踪的详细信息，请参阅第68页的『跟踪 MQ base Java 程序』。

CICS 事务处理服务器 OS/390 版的跟踪

使用 CICS 事务处理服务器 OS/390 版时，要想直接向程序提供命令行变量是不可能的。应编写一个小的“包装器”程序以适当变量来调用 MQIVP.main()。

错误消息

这里有一些可能经常见到的错误消息:

无法标识本地主机的 IP 地址

服务器未与网络连接。

建议操作: 将服务器连接到网络, 再试一次。

无法装入文件 gatekeeper.ior

如果 `gatekeeper.ior` 文件不在正确的位置上, 那么当在 Web 服务器上部署 VisiBroker 小应用程序时会发生这类故障。

建议操作: 从部署了小应用程序的目录中重新启动 VisiBroker Gatekeeper。`gatekeeper` 文件将被写入此目录。

故障: 软件丢失, 可能是 MQSeries 或 VBROKER_ADM 变量

若 Java 软件环境是不完整的, 则这种失败会在 MQIVP 样本程序中发生。

建议操作: 在客户机上, 确保 VBROKER_ADM 环境变量设置为表示 VisiBroker for Java 管理（`adm`）目录, 然后重试。

| 在服务器上, 确保安装了最新版本的 MQ base Java, 然后重试。

NO_IMPLEMENT

涉及 VisiBroker Smart Agents 的通信问题。

建议操作: 查阅 VisiBroker 文档。

COMM_FAILURE

涉及 VisiBroker Smart Agents 的通信问题。

建议操作: 所有 VisiBroker Smart Agent 都使用同一个端口号, 再试一次。查阅 VisiBroker 文档。

MQRC_ADAPTER_NOT_AVAILABLE

若在试用 VisiBroker 时获知这一错误, 很可能是在 CLASSPATH 中找不到 JAVA 类 org.omg.CORBA.ORB。

建议操作: 确保 CLASSPATH 语句中包含了到 VisiBroker `vbjorb.jar` 和 `vbjapp.jar` 文件的路径。

MQRC_ADAPTER_CONN_LOAD_ERROR

如果是在 OS/390 上运行时发现了这个错误, 请确保 STEPLIB 语句中有 MQSeries SCSQANLE 和 SCSQAUTH 数据集。

第4章 使用 MQSeries classes for Java Message Service (MQ JMS)

本章描述下列任务:

- 如何设置您的系统来使用“测试”和样本程序
- 如何运行点到点“安装验证测试”(IVT) 程序来验证您的 MQSeries classes for Java Message Service 安装
- 如何运行样本“发布/订阅安装验证测试”(PSIVT) 程序来验证您的发布/订阅安装
- 如何运行您自己的程序

安装后设置

要使 MQ JMS 程序能够使用所有必需的资源，需要更新下列系统变量:

Classpath

JMS 程序的成功操作需要 JVM 能够使用几个 Java 包。必须在获取并安装必需的包后在 classpath 上指定它们。

| 将下列 .jar 文件添加到 classpath:

| - com.ibm.mq.jar
| - com.ibm.mqjms.jar
| - connector.jar
| - jms.jar
| - jndi.jar
| - jta.jar
| - ldap.jar
| - providerutil.jar

环境变量

在 MQ JMS 安装的 bin 子目录中有许多脚本。对于许多公共操作，这些用作快捷方式。这些脚本中的大部分假设已定义了指向 MQ JMS 安装目录的环境变量 MQ_JAVA_INSTALL_PATH。不是必须设置该值，但是如果未设置它，则必须相应编辑 bin 目录中的脚本。

在 Windows NT 上，可使用**系统特性**中的**环境**选项卡设置 classpath 和新的环境变量。在 UNIX 上，通常从每个用户的登录脚本设置它们。在任何平台上，都可选择使用脚本来维护不同的 classpath 和不同项目的其它环境变量。

© Copyright IBM Corp. 1997, 2001

“发布／订阅”方式的附加设置

在能够使用 JMS “发布／订阅” 的 MQ JMS 实现之前，需要某些附加设置:

确保正在运行“代理”

要验证是否安装并正在运行 MQSeries 发布／订阅代理，使用命令:

```
dspmqbrk -m MY.QUEUE.MANAGER
```

其中 MY.QUEUE.MANAGER 是正在运行代理的队列管理器名称。如果正在运行代理，则显示类似下列的消息:

```
MQSeries message broker for queue manager MY.QUEUE.MANAGER running.
```

如果操作系统报告无法运行 dspmqbrk 命令，请确保正确安装了 MQSeries 发布／订阅代理。

如果操作系统报告代理是非活动的，则使用下列命令启动它:

```
strmqbrk -m MY.QUEUE.MANAGER
```

创建 MQ JMS 系统队列

要使 MQ JMS 发布／订阅实现正确工作，必须创建一些系统队列。在 MQ JMS 安装的 bin 子目录中提供了帮助完成该任务的脚本。要使用该脚本，请输入下列命令:

```
runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc
```

如果发生错误，请检查是否正确输入了队列管理器名称，并检查是否正在运行队列管理器。

非特权用户需要权限的队列

非特权用户需要授权以访问由 JMS 使用的队列. 关于 MQSeries 中访问控制的详细信息，请参阅 *MQSeries 系统管理*中关于保护 MQSeries 对象的详细信息。

对于 JMS 点到点方式，访问控制问题与 MQSeries classes for Java 的访问控制相似:

- QueueSender 使用的队列需要放入权限
- QueueReceiver 和 QueueBrowser 需要获取、查询和浏览权限。
- QueueSession.createTemporaryQueue 方法需要访问 QueueConnectionFactory temporaryModel 字段（缺省情况下将是 SYSTEM.DEFAULT.MODEL.QUEUE) 中定义的模型队列。

对于 JMS 发布／订阅方式，使用下列系统队列:

SYSTEM.JMS.ADMIN.QUEUE

SYSTEM.JMS.REPORT.QUEUE

SYSTEM.JMS.MODEL.QUEUE

SYSTEM.JMS.PS.STATUS.QUEUE

SYSTEM.JMS.ND.SUBSCRIBER.QUEUE

SYSTEM.JMS.D.SUBSCRIBER.QUEUE

SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE

SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE

SYSTEM.BROKER.CONTROL.QUEUE

同样，发布消息的任何应用程序还需要访问正在使用的主题连接工厂中指定的 STREAM 队列。它的缺省值是:

SYSTEM.BROKER.DEFAULT.STREAM

运行点到点 IVT

本节描述随 MQ JMS 提供的点到点安装验证测试程序 (IVT)。

IVT 试图通过以绑定方式使用 MQ JMS 连接到本地机器上的缺省队列管理器来验证安装。然后它将消息发送到 SYSTEM.DEFAULT.LOCAL.QUEUE 队列并再次读回它。

可用下列两种可能方式中的一种运行程序:

使用受管理对象的 JNDI 查表

JNDI 方式强制程序从 JNDI 名称空间获取它的受管理对象，这是 JMS 客户机应用程序期待的操作。(请参阅第33页的『管理 JMS 对象』以获取受管理对象的描述)。该调用方法需要与管理工具相同的先决条件(请参阅第29页的『第5章 使用 MQ JMS 管理工具』)。

不使用受管理对象的 JNDI 查表

如果不想使用 JNDI，可通过以非 JNDI 方式运行 IVT 在运行时创建受管理的对象。因为基于 JNDI 资源库的设置相对复杂，所以我们建议第一次运行 IVT 时不使用 JNDI。

不使用 JNDI 的点到点验证

提供了一个在 UNIX 上称为 IVTRun 或在 Windows NT 上称为 IVTRun.bat 的脚本来运行 IVT。在安装的 bin 子目录中安装了该文件。

要运行不使用 JNDI 的测试，发出下列命令:

```
IVTRun -nojndi
```

对于客户机方式，要运行不使用 JNDI 的测试，则发出下列命令:

```
IVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
             [-channel <channel>]
```

其中:

qmgr 是您想连接的队列管理器名称

hostname 是正在运行队列管理器的主机

port 是正在运行队列管理器侦听器的 TCP/IP 端口

channel 是客户机连接通道(缺省值是 SYSTEM.DEF.SVRCONN)

点到点 IVT

如果测试成功完成，应该看到类似下列的输出:

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message: Message Class:   jms_text          JMSType:          null
JMSDeliveryMode: 2                              JMSExpiration:    0
JMSPriority:     4                              JMSMessageID:     ID:414d5120716
d312020202020202020203000c43713400000
JMSTimestamp:    935592657000                   JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:      null
JMSRedelivered:  false
JMS_IBM_Format:MQSTR           JMS_IBM_PutApplType:11
JMSXGroupSeq:1                 JMSXDeliveryCount:0
JMS_IBM_MsgType:8              JMSXUserID:kingdon
JMSXApplID:D:\jdk1.1.8\bin\java.exe
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

使用 JNDI 的点到点验证

要运行使用 JNDI 的 IVT，必须运行 LDAP 服务器并将它设置成接受 Java 对象。如果出现下列消息，它表示存在到 LDAP 服务器的连接但未正确配置该服务器:

```
Unable to bind to object
```

该消息表示服务器要么不在存储 Java 对象，要么对象的许可权或后缀不正确。请参阅第347页的『检查 LDAP 服务器配置』。

同样，下列管理对象必须从 JNDI 名称空间可见检索:

- MQQueueConnectionFactory
- MQQueue

提供一个在 UNIX 上称为 IVTSetup 或在 Windows NT 上称为 IVTSetup.bat 的脚本来自动创建这些对象。输入命令;

```
IVTSetup
```

该脚本调用 MQ JMS“管理”工具（请参阅第29页的『第5章 使用 MQ JMS 管理工具』）并在 JNDI 名称空间中创建对象。

MQQueueConnectionFactory 绑定在名称 ivtQCF 下（对于 LDAP cn=ivtQCF）。所有特性均为缺省值:

```
TRANSPORT(BIND)
PORT(1414)
HOSTNAME(localhost)
CHANNEL(SYSTEM.DEF.SVRCONN)
VERSION(1)
CCSID(819)
TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
QMANAGER()
```

MQQueue 绑定在名称 ivtQ 下 (cn=ivtQ)。QUEUE 特性的值成为 QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)。所有其它特性有缺省值。

```
PERSISTENCE(APP)
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
EXPIRY(APP)
TARGCLIENT(JMS)
ENCODING(NATIVE)
VERSION(1)
CCSID(1208)
PRIORITY(APP)
QMANAGER()
```

一旦在 JNDI 名称空间运行中创建了管理对象，请使用下列命令运行 IVTRun 脚本（Windows NT 上的 IVTRun.bat）:

IVTRun [-t] [-url <"**providerURL**"> [-icf <**initCtxFact**>]]

其中:

-t
表示打开跟踪（缺省情况下，跟踪是关闭的）

providerURL
是受管理对象的 JNDI 位置。如果在缺省初始环境工厂中使用，它的形式是 LDAP URL:

```
ldap://hostname.company.com/contextName
```

如果使用文件系统服务供应商（参阅下列 initCtxFact），URL 是格式:

```
file://directorySpec
```

注: 用引号 (") 括起 *providerURL* 字符串。

initCtxFact
是初始环境工厂的类名。缺省值是对于 LDAP 服务供应商，值为:

```
com.sun.jndi.ldap.LdapCtxFactory
```

如果使用文件系统服务供应商，将参数设置为:

```
com.sun.jndi.fscontext.RefFSContextFactory
```

如果测试成功完成，输出与非 JNDI 输出相似，除了 'create' QueueConnectionFactory 和 Queue 行表示对象可从 JNDI 检索。下列代码片段显示了一个示例。

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
```

```
Retrieving a Queue from JNDI
Creating a QueueSender
 ...
 ...
```

虽然没严格必要，从 JNDI 名称空间除去由 IVTSetup 创建的对象却是个好习惯。为实现该目的提供了称为 IVTTidy 的脚本（在Windows NT 上是 IVTTidy.bat）。

IVT 错误恢复

如果测试不成功，下列注解可能有所帮助:

- 对于带有任何参与类路径的错误消息，检查是否正确设置了 classpath，如第19页的『安装后设置』所述。
- 失败的 IVT 可能带有消息 ’failed to create MQQueueManager’ 以及包含号码 2059 的附加消息。它表示 MQSeries 连接运行 IVT 的机器上的缺省本地队列管理器时失败。检查是否正在运行队列管理器，是否将它标记为缺省队列管理器。
- 消息 ’failed to open MQ queue’ 表示 MQSeries 连接了缺省队列管理器，但无法打开 ’SYSTEM.DEFAULT.LOCAL.QUEUE’。它可能表示在缺省队列管理器上不存在该队列或该队列没有启用 PUT 和 GET。为测试的持续性添加或启用队列。

表5 列出了由 IVT 测试的类和它们来自的包:

表 5. IVT 测试的类

类	Jar 文件
MQSeries JMS 类	com.ibm.mqjms.jar
com.ibm.mq.MQMessage	com.ibm.mq.jar
javax.jms.Message	jms.jar
javax.naming.InitialContext	jndi.jar
javax.resource.cci.Connection	connector.jar
javax.transaction.xa.XAException	jta.jar
com/sun/jndi/toolkit/ComponentDirContext	providerutil.jar
com.sun.jndi.ldap.LdapCtxFactory	ldap.jar

“发布／订阅”安装验证测试

仅提供编译形式工具的“发布／订阅安装验证测试”（PSIVT）程序。它在 com.ibm.mq.jms 包中。

PSIVT 试图:

1. 创建发布者，p，在主题 MQJMS/PSIVT/Information 上发布
2. 创建订户，s，在主题 MQJMS/PSIVT/Information 上订阅
3. 使用 p 发布简单文本消息
4. 使用 s 接收等待其输入队列的消息

运行 PSIVT 时发布者发布消息，订户接收并显示消息。发布者发布到代理的缺省流。是非长期订户，不执行消息选择，并且从本地连接接受消息。它执行同步接收，最多花 5 秒钟等待消息到达。

可象 IVT 一样以 JNDI 方式或单机方式运行 PSIVT。JNDI 方式使用 JNDI 从 JNDI 名称空间中检索 TopicConnectionFactory 和 Topic。如果不使用 JNDI，则在运行时创建这些对象。

不使用 JNDI 的发布／订阅验证

提供名为 PSIVTRun 的 'PSIVTRun'（在 Windows NT 上是 PSIVTRun.bat）运行 PSIVT。该文件在安装目录的 bin 子目录中。

要运行不使用 JNDI 的测试，发出下列命令:

```
PSIVTRun -nojndi [-m <qmgr>] [-t]
```

| 对于客户机方式，要运行不使用 JNDI 的测试，则发出下列命令:

```
PSIVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
                [-channel <channel>] [-t]
```

| 其中:

-nojndi 表示受管理对象无 JNDI 查表

qmgr 是要连接到的队列管理器名称

hostname 是运行队列管理器的主机

port 是正在运行队列管理器侦听器的 TCP/IP 端口（缺省 1414）

channel 是客户机连接通道（缺省 SYSTEM.DEF.SVRCONN）

-t 表示打开跟踪（缺省是关闭）

如果测试成功完成，输出类似如下:

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service
发布／订阅 Installation Verification Test
Creating a TopicConnectionFactory
Creating a Topic
Creating a Connection
Creating a Session
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding Text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive...

Got message:

JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d2e504f4c415249532e4254b7dc3753700000
  JMSTimestamp:    937232048000
  JMSCorrelationID:ID:414d5158000000000000000000000000000000000000000
  JMSDestination:  topic
  ://MQJMS/PSIVT/Information
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_Format:MQSTR
  UNIQUE_CONNECTION_ID:937232047753
  JMS_IBM_PutApplType:26
```

```
  JMSXGroupSeq:1
  JMSXDeliveryCount:0
  JMS_IBM_MsgType:8
  JMSXUserID:hollingl
  JMSXApplID:QM.POLARIS.BROKER
A simple text message from the MQJMSPSIVT program

Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT completed OK
PSIVT finished
```

使用 JNDI 的发布／订阅验证

要以 JNDI 方式运行 PSIVT，必须从 JNDI 名称空间可检索以下两个受管理对象:

- 与名称 ivtTCF 绑定的 TopicConnectionFactory
- 与名称 ivtT 绑定的 Topic

可使用MQ JMS“管理”工具（请参阅第29页的『第5章 使用 MQ JMS 管理工具』）和使用下列命令定义这些对象:

```
DEFINE TCF(ivtTCF)
```

该命令定义了 TopicConnectionFactory。

```
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)
```

该命令定义了 Topic。

这些定义假设运行代理的缺省队列管理器是可用的。关于配置这些对象以使用非缺省队列管理器的详细信息，请参阅第33页的『管理 JMS 对象』。这些对象应该驻留在由下述 -url 命令行参数指向的上下文中。

要以 JNDI 方式运行测试，输入下列命令:

```
PSIVTRun -url <purl> [-icf <initcf>] [-t]
```

其中:

-t 表示打开跟踪（缺省情况下，跟踪关闭）

-url <purl> 是驻留管理对象的 JNDI 位置的 URL

-icf <initcf> 是 JNDI 的 initialContextFactory [com.sun.jndi.ldap.LdapCtxFactory]

如果测试成功完成，输出与非 JNDI 的输出相似，除了 'create' QueueConnectionFactory 和 Queue 行表示从 JNDI 的对象恢复。

PSIVT 错误恢复

如果测试不成功，下列注释可能有所帮助:

- 如果看到下列消息:

  ```
  *** The broker is not running! Please start it using 'strmqbrk' ***
  ```

  这表示在目标队列管理器上安装了代理，但是它的控制队列包含一些未完成的消息。它表示不在运行代理。要启动它，可使用 strmqbrk 命令。（请参阅第20页的『“发布／订阅”方式的附加设置』。）

- 如果显示下列消息:

  ```
  Unable to connect to queue manager: <default>
  ```

  请确保您的 MQSeries 系统已经配置了缺省队列管理器。
- 如果显示下列消息:

  ```
  Unable to connect to queue manager: ...
  ```

  请确保用有效的队列管理器名称配置了 PSIVT 使用的受管理的 TopicConnectionFactory。如果使用 -nojndi 选项，确保提供有效的队列管理器（使用 -m 选项）。
- 如果显示下列消息:

  ```
  Unable to access broker control queue on queue manager: ...
  Please ensure the broker is installed on this queue manager
  ```

  请确保用安装代理的队列管理器名称配置了 PSIVT 使用的受管理 TopicConnectionFactory。如果使用 -nojndi 选项，请确保提供队列管理器名称（使用 -m 选项）。

运行您自己的 MQ JMS 程序

有关编写您自己的 MQ JMS 程序的信息，请参阅第167页的『第10章 编写 MQ JMS 程序』。

MQ JMS 包含实用程序文件 runjms（在Windows NT 上是 runjms.bat），它帮助运行提供的程序和您编写的程序。

实用程序提供跟踪和日志文件的缺省位置，并使您能够添加应用程序需要的任何运行时参数。提供的脚本假设将环境变量 MQ_JAVA_INSTALL_PATH 设置成安装 MQ JMS 的目录。脚本还假设该目录中的 trace 和 log 子目录分别用于跟踪和记录输出。这些只是建议的位置，可以编辑脚本以使用任何您选择的目录。

使用下列命令运行您的应用程序:

```
runjms <classname of application> [application-specific arguments]
```

关于编写 MQ JMS 应用程序和小应用程序的信息，请参阅第165页的『第3部分 用 MQ JMS 进行编程』。

解决问题

若程序未能成功完成，请运行安装验证程序，它在第19页的『第4章 使用 MQSeries classes for Java Message Service (MQ JMS)』中有所描述，并请按照诊断消息中给出的建议做。

跟踪程序

提供 MQ JMS 跟踪设施帮助 IBM 员工诊断用户问题。

缺省情况下禁用跟踪，由于输出迅速增大，因而在正常环境下不使用。

如果要提供跟踪输出，可将 Java 特性 MQJMS_TRACE_LEVEL 设置为下列值:

on 仅跟踪 MQ JMS 调用

base 跟踪 MQ JMS 调用和基本 MQ base Java 调用

例如:

```
java -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

要禁用跟踪，将 MQJMS_TRACE_LEVEL 设置为 **off**。

缺省情况下，跟踪输出在当前工作目录中称为 mqjms.trc 的文件。可使用 Java 特性 MQJMS_TRACE_DIR 将它重定向到不同的目录中。

例如:

```
java -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```

runjms 实用程序脚本使用环境变量 MQJMS_TRACE_LEVEL 和 MQ_JAVA_INSTALL_PATH 设置这些特性，如下:

```
java -DMQJMS_LOG_DIR=%MQ_JAVA_INSTALL_PATH%\log
-DMQJMS_TRACE_DIR=%MQ_JAVA_INSTALL_PATH%\trace
-DMQJMS_TRACE_LEVEL=%MQJMS_TRACE_LEVEL% %1 %2 %3 %4 %5 %6 %7 %8 %9
```

这仅是建议，可根据需要修改。

日志记录

提供 MQ JMS 日志设施以报告严重问题，特别是表示配置错误的问题而不是编程错误。缺省状态下日志输出被发送到 System.err 流，它通常在运行 JVM 的控制台 stderr 上出现。

可使用指定新位置的 Java 特性将输出重定向到文件，例如:

```
java -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

MQ JMS 安装的 bin 目录中实用程序脚本 runjms 将该特性设置成:

```
<MQ_JAVA_INSTALL_PATH>/log
```

其中 MQ_JAVA_INSTALL_PATH 是 MQ JMS 的安装路径。这是建议，可按照需要修改。

当将日志重定向到文件时，它以二进制形式输出。提供将文件转换成普通文本格式的实用程序 formatLog（在 Windows NT 上是 formatLog.bat）来查看日志。该实用程序存储在 MQ JMS 安装的二进制目录中。按以下运行转换:

```
formatLog <inputfile> <outputfile>
```

第5章 使用 MQ JMS 管理工具

管理工具使管理员能定义八种类型的 MQ JMS 对象并把它们存储到 JNDI 名称空间中。然后，JMS 客户机可以通过 JNDI 检索名称空间中的这些受管理对象和使用它们。

可以通过使用本工具来管理的 JMS 对象有:
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

有关这些对象的详细信息，请参考第33页的『管理 JMS 对象』。

注: JMSWrapXAQueueConnectionFactory 和 JMSWrapXATopicConnectionFactory 是特定于 WebSphere 的类。它们包含在包 **com.ibm.ejs.jms.mq** 中。

此工具还允许管理员在 JNDI 中操纵目录名称空间子上下文。请参阅第32页的『操纵子上下文』。

调用管理工具

管理工具具有一个命令行界面。可以交互式地使用它，也可以用它来启动一个批处理。交互方式提供了一个命令提示，可以在提示上输入管理命令。在批处理方式中，启动工具的命令中包括了含有管理命令脚本的文件名称。

要以交互方式启动工具，请输入命令:

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

其中:

-t	启用跟踪（缺省是跟踪关闭）
-v	产生详细输出（缺省是 terse 输出）
-cfg config_filename	替代配置文件的名称（请参阅第30页的『配置』）

显示命令提示后，表明此工具可以开始接受管理命令了。此提示最初如以下形式出现:

```
InitCtx>
```

表明当前的上下文（也就是所有命名与目录指向的 JNDI 上下文）是在 PROVIDER_URL 配置参数中定义的初始上下文（请参阅第30页的『配置』）。

当遍历目录名称空间时，提示将更改以反映这个移动，从而使提示总是显示当前的上下文。

© Copyright IBM Corp. 1997, 2001

调用管理工具

要以批处理方式启动工具，请输入命令:

```
JMSAdmin <test.scp
```

其中 *test.scp* 是包含管理命令的脚本文件（请参阅第32页的『管理命令』)。文件中的最后一个命令必须是 END 命令。

配置

必须把管理工具配置成带有下列三个参数的值:

INITIAL_CONTEXT_FACTORY

表示工具使用的服务供应商。这个特性当前有三个受支持的值:

- com.sun.jndi.ldap.LdapCtxFactory（用于 LDAP）
- com.sun.jndi.fscontext.RefFSContextFactory（用于文件系统上下文）
- com.ibm.ejs.ns.jndi.CNInitialContextFactory（和 WebSphere 的 CosNaming 资源库一起使用）

PROVIDER_URL

表示会话初始上下文的 URL，由工具实现的所有 JNDI 操作的根。当前支持该特性的三种形式:

- ldap://hostname/contextname（用于 LDAP）
- file:[drive:]/pathname（用于文件系统上下文）
- iiop://hostname[:port] /[?TargetContext=ctx] （用以访问"基本" WebSphere CosNaming 名称空间）

SECURITY_AUTHENTICATION

表明 JNDI 是否把安全性凭证传递给了您的服务供应商。只有当使用了 LDAP 服务供应商时，才使用此参数。此特性当前可以取以下三个值之一:

- none（匿名认证）
- simple（简单认证）
- CRAM-MD5（CRAM-MD5 认证机制）

如果没有提供有效值，则特性缺省值为 none。有关管理工具具有的安全性的详细信息，请参阅第31页的『安全性』。

在一个配置文件中设置了这些参数。当调用工具时，可以使用 -cfg 命令行参数来指定该配置，如第29页的『调用管理工具』中所示。如果不指定配置文件的名称，则工具将尝试装入缺省配置文件（JMSAdmin.config)。首先在当前目录下查找这个文件，然后是在 <MQ_JAVA_INSTALL_PATH>/bin 目录下。(其中 <MQ_JAVA_INSTALL_PATH> 是至 MQ JMS 安装的路径。)

配置文件是简单文本文件，由一组用 “=” 分隔的关键字-值对组成。以下显示的是一个示例:

```
#设置服务供应商
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#设置初始上下文
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#设置认证类型
    SECURITY_AUTHENTICATION=none
```

（行中第一列 ’#’ 表示一个注解，或表示不使用这行。）

安装带有一个称为 JMSAdmin.config 的样本配置文件，可以在目录 <MQ_JAVA_INSTALL_PATH>/bin 中找到这个文件。编辑此文件以适合您的系统设置。

配置 WebSphere

对于和 WebSphere 的 CosNaming 资源库一起使用的管理工具（或任何需要执行顺序查找的客户机应用程序），需要进行以下配置:

- CLASSPATH 必须包括 WebSphere 的与 JNDI 相关的 jar 文件:
  - 对于 WebSphere V3.5:

    <WSAppserver>\lib\ejs.jar
- WebSphere V.3.5 的 PATH 必须包括:

  <WSAppserver>\jdk\jre\bin

  其中 <WSAppserver> 是指 WebSphere 的安装路径。

安全性

管理员需了解有关第30页的『配置』中描述的 SECURITY_AUTHENTICATION 特性的作用。

- 如果该参数设置为 *none*，则 JNDI 不传递任何安全性凭证给服务供应商，并执行 “匿名认证”。
- 如果把参数设置为 simple 或 CRAM-MD5，安全性凭证将通过 JNDI 传递到基本服务供应商。安全性凭证由用户唯一名称（用户 DN）与口令组成。

如果需要安全性凭证，那么在工具初始化时将提示用户这些。

注: 输入的文本被回显在屏幕上，其中包括口令。因此，请小心别让口令泄露给未授权的用户。

工具本身不认证。LDAP 服务器管理员负责设置与维护对目录不同部分的访问特权。如果认证失败，则工具显示一个适当的错误消息并终止。

Sun 公司的 Java 网站 (http://java.sun.com) 上的文档中有安全性和 JNDI 的更详细信息.

管理命令

当显示命令提示时，表明此工具可以开始接受命令了。管理命令的格式通常是:

```
verb [param]*
```

其中 *verb* 是表6中列出的管理动词中的一个。所有有效命令都至少包含一个（并且只有一个）动词，它以标准或简短形式出现在命令的开头。

动词可使用的参数取决于该动词。例如，动词 END 不能使用任何参数，但是动词 DEFINE 则可以使用任何在 1 和 20 之间的参数。本章以后的部分中将详细讨论至少使用了一个参数的动词。

表 6. 管理动词

动词		描述
标准	简短形式	
ALTER	ALT	更改给定管理对象中的至少一个特性
DEFINE	DEF	创建并存储管理对象，或创建一个新的子上下文。
DISPLAY	DIS	显示存储的一个或多个管理对象的特性或当前上下文的内容
DELETE	DEL	从名称空间中删除一个或多个管理对象或删除一个空的子上下文
CHANGE	CHG	改变当前上下文，允许用户转换初始上下文下任何地方的目录名称空间（暂挂安全性确认）
COPY	CP	复制存储的受管理对象，并用另一个名称存储它
MOVE	MV	改变存储的受管理对象的名称
END		关闭管理工具

动词名不区分大小写。

通常，要终止命令时请按下回车键。但是，也可以在回车之前通过直接输入 '+' 号来覆盖它。这将使您能输入多行的命令，如以下示例所示:

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

以字符 *、# 或 / 开始的行表示一个注解或忽略的行。

操纵子上下文

可以使用动词 CHANGE、DEFINE、DISPLAY 和 DELETE 来操纵目录名称空间子上下文。 表7中描述了它们的用法。

表 7. 操纵子上下文的命令语法与说明

语法	描述
DEFINE CTX(ctxName)	尝试创建当前上下文的新子女子上下文，使用名称 ctxName。如果安全性违例，或是子上下文已经存在，又或者提供的名称无效则失败。

表 7. 操纵子上下文的命令语法与说明（续）

语法	描述
DISPLAY CTX	显示当前上下文的内容。管理对象用 'a' 注释，子上下文用 '[D]' 注释。此外，还显示每个对象的 Java 类型。
DELETE CTX(ctxName)	尝试删除当前上下文的名为 ctxName 的子上下文。如果没有找到上下文，或上下文是非空的，或如果有安全性违例，则失败。
CHANGE CTX(ctxName)	改变当前上下文，以便它现在可以引用名为 ctxName 的子上下文。将提供 ctxName 的两个特殊值中的一个: **=UP** 移动到当前上下文的父代 **=INIT** 直接移动到初始上下文 如果不存在指定的上下文，或是有安全性违例，则失败。

管理 JMS 对象

| 这一部分描述管理工具可以处理的八种对象类型。详细描述它们的每个配置特性以及可操纵它们的每个动词。

对象类型

| 表8显示了八种管理对象类型。“关键字”列显示了可用于替代第34页的表9中所示命令中 *TYPE* 的字符串。

表 8. 由管理工具处理的 JMS 对象类型

对象类型		描述
Java	**关键字**	
MQQueueConnectionFactory	QCF	JMS QueueConnectionFactory 接口的 MQSeries 实现。它表示了用于在 JMS 的点到点域中创建连接的工厂对象。
MQTopicConnectionFactory	TCF	JMS TopicConnectionFactory 接口的 MQSeries 实现。它表示了用于在 JMS 的发布／订阅域中创建连接的工厂对象。
MQQueue	Q	JMS Queue 接口的 MQSeries 实现。它表示了在 JMS 的点到点域中的消息目的地。
MQTopic	T	JMS Topic 接口的 MQSeries 实现。它表示了 JMS 的发布／订阅域中的消息目的地。
MQXAQueueConnectionFactory[1]	XAQCF	JMS XAQueueConnectionFactory 接口的 MQSeries 实现。它表示了用于在使用 XA 版本的 JMS 类的 JMS 的点到点域中创建连接的工厂对象。
MQXATopicConnectionFactory[1]	XATCF	JMS XATopicConnectionFactory 接口的 MQSeries 实现。它表示了用于在使用 XA 版本的 JMS 发布／订阅域中创建连接的工厂对象。

表 8. 由管理工具处理的 JMS 对象类型 (续)

对象类型		描述
Java	**关键字**	
JMSWrapXAQueueConnectionFactory[2]	WSQCF	JMS QueueConnectionFactory 接口的 MQSeries 实现。它表示了用于在使用 XA 版本的 JMS 类的 JMS 域的点到点域中创建与 WebSphere 连接的工厂对象。
JMSWrapXATopicConnectionFactory[2]	WSTCF	JMS TopicConnectionFactory 接口的 MQSeries 实现。它表示了用于在使用 XA 版本的 JMS 类的 JMS 发布／订阅域中创建与 WebSphere 连接的工厂对象。
1. 提供这些类是为了让应用服务器供应商使用它们。而不是让应用程序员直接使用它们。 2. 如果希望您的 JMS 会话参与到由 WebSphere 协调的全局事务中，则使用 ConnectionFactory 的这种样式。		

与 JMS 对象一起使用的动词

可以使用动词 ALTER、DEFINE、DISPLAY、DELETE、COPY 和 MOVE 来操纵目录名称空间中的受管理对象。表9概述了它们的用法。使用表示所需的受管理对象的关键字（如第33页的表8中例举）来替代*TYPE*。

表 9. 操纵管理对象的命令语法与说明

命令语法	描述
ALTER *TYPE*(name) [property]*	试图更新给定受管理对象提供的特性。如果有安全性，或是没有找到指定的对象，或是提供的新特性无效，则失败。
DEFINE *TYPE*(name) [property]*	尝试创建类型为 *TYPE* 并带有所提供特性的管理对象，并尝试把它存储到当前上下文中名称 name 下。如果安全性违例，或是提供的名称无效或已经存在，又或是提供的新特性无效，则失败。
DISPLAY *TYPE*(name)	显示类型为 *TYPE* 的管理对象的特性，它绑定在当前上下文中的名称 name 下。如果对象不存在，或安全性违例，则失败。
DELETE *TYPE*(name)	尝试除去类型为 *TYPE* 的受管理对象的特性，它在当前上下文中，名为 name。如果对象不存在，或有安全性违例，则失败。
COPY *TYPE*(nameA) *TYPE*(nameB)	复制类型为 *TYPE* 的受管理对象 nameA，并把副本命名为 nameB。所有这些都发生在当前上下文的作用域中。如果要复制的对象不存在，或对象名 nameB 已经存在，或者有安全性违例，则失败。
MOVE *TYPE*(nameA) *TYPE*(nameB)	将类型为 *TYPE* 的受管理对象 nameA 移动（重命名）到 nameB。所有这些都发生在当前上下文的作用域中。如果要移动的对象不存在，或对象名 nameB 已经存在，或者有安全性违例，则失败。

创建对象

使用以下命令语法创建对象并存储到 JNDI 名称空间中:

```
DEFINE TYPE(name) [property]*
```

即，动词 DEFINE，后跟 *TYPE*(name) 引用的管理对象，最后是零个或多个*特性*（见第36页的『特性』）。

LDAP 命名考虑事项

要在 LDAP 环境下存储对象，对象名必须遵循一定的约定。其中之一就是对象及子上下文的名称必须包含一个前缀，例如 cn=（公共名称）或 ou=（组织单位）。

管理工具通过允许您引用不带前缀的对象和上下文名称来简化 LDAP 服务供应商的使用。如果不提供前缀，则工具将把缺省前缀（当前是 cn=）自动添加到您提供的名称。

以下示例显示了上述情况。

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a   cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

注意，即使提供的对象名 (testQueue) 没有前缀，工具会自动添加一个以确保符合 LDAP 的命名约定。同样，提交 DISPLAY Q(testQueue) 时也会添加该前缀。

可能需要配置 LDAP 服务器以存储 Java 对象。第347页的『附录C. 针对 Java 对象的 LDAP 服务器配置』中提供了有关此配置的帮助信息。

特性

特性由以下格式的名称-值对组成:

```
PROPERTY_NAME(property_value)
```

特性名不区分大小写，并受限于表10中显示的可识别名称集。该表还显示了每一特性的有效特性值。

表 10. 特性名与有效值

特性		有效值（缺省值用黑体表示）
标准	缩写形式	
DESCRIPTION	DESC	任何字符串
TRANSPORT	TRAN	• **BIND** - 连接使用 MQSeries 绑定。 • CLIENT - 使用客户机连接。
CLIENTID	CID	任何字符串
QMANAGER	QMGR	任何字符串
HOSTNAME	HOST	任何字符串
PORT		任何正整数
CHANNEL	CHAN	任何字符串
CCSID	CCS	任何正整数
RECEXIT	RCX	任何字符串
RECEXITINIT	RCXI	任何字符串
SECEXIT	SCX	任何字符串
SECEXITINIT	SCXI	任何字符串
SENDEXIT	SDX	任何字符串
SENDXITINIT	SDXI	任何字符串
TEMPMODEL	TM	任何字符串
MSGRETENTION	MRET	• **Yes** - 输入队列中保留不需要的消息 • No - 根据配置选项来处理不需要的消息
BROKERVER	BVER	**V1** - 当前只允许这个值。

表 10. 特性名与有效值 (续)

特性		有效值（缺省值用黑体表示）
标准	缩写形式	
BROKERPUBQ	BPUB	任何字符串（缺省值是 **SYSTEM.BROKER.DEFAULT.STREAM**）
BROKERSUBQ	BSUB	任何字符串（缺省值是 **SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE**）
BROKERDURSUBQ	BDSUB	任何字符串（缺省值是 **SYSTEM.JMS.D.SUBSCRIPTION.QUEUE**）
BROKERCCSUBQ	CCSUB	任何字符串（缺省值是 **SYSTEM.JMS.ND.CC.SUBSCRIPTION.QUEUE**）
BROKERCCDSUBQ	CCDSUB	任何字符串（缺省值是 **SYSTEM.JMS.D.CC.SUBSCRIPTION.QUEUE**）
BROKERQMGR	BQM	任何字符串
BROKERCONQ	BCON	任何字符串
EXPIRY	EXP	• **APP** - JMS 应用程序可以定义的失效时间。 • UNLIM - 没有失效时间。 • 任何正整数，表示以毫秒计算的失效时间。
PRIORITY	PRI	• **APP** - JMS 应用程序可以定义的优先级。 • QDEF - 优先级使用队列缺省值。 • 范围在 0-9 的任何整数。
PERSISTENCE	PER	• **APP** - JMS 应用程序可以定义的持续性。 • QDEF - 持续性使用队列缺省值。 • PERS - 消息是持续的。 • NON - 消息是非持续的。
TARGCLIENT	TC	• **JMS** - 消息目标是 JMS 应用程序。 • MQ - 消息目标是一个非 JMS 的传统 MQSeries 应用程序。
ENCODING	ENC	参阅第39页的『ENCODING 特性』
QUEUE	QU	任何字符串
TOPIC	TOP	任何字符串

许多特性仅与特定的对象类型子集相关。表11显示了哪些特性-对象类型的组合是有效的，并对每个特性给出了简要的描述。

表 11. 特性与对象类型的有效组合

特性	有效对象类型						描述
	QCF	**TCF**	**Q**	**T**	**WSQCF XAQCF**	**WSTCF XATCF**	
DESCRIPTION	Y	Y	Y	Y	Y	Y	对存储对象的描述。
TRANSPORT	Y	Y			Y[1]	Y[1]	连接是否使用 MQ Bingdings 或客户机连接
CLIENTID	Y	Y			Y	Y	客户机的字符串标识
QMANAGER	Y	Y	Y		Y	Y	要连接到的队列管理器的名称
PORT	Y	Y					队列管理器正在侦听的端口
HOSTNAME	Y	Y					队列管理器所在的主机名称

表 11. 特性与对象类型的有效组合 (续)

特性	有效对象类型						描述
	QCF	TCF	Q	T	WSQCF XAQCF	WSTCF XATCF	
CHANNEL	Y	Y					要使用的客户机连接通道
CCSID	Y	Y	Y	Y			连接中要使用的 coded-character-set-ID
RECEXIT	Y	Y					使用的接收出口的全限定类名
RECEXITINIT	Y	Y					接收出口初始化字符串
SECEXIT	Y	Y					要使用的安全性出口的全限定类名
SECEXITINIT	Y	Y					安全性出口的初始化字符串
SENDEXIT	Y	Y					要使用的发送出口的全限定类名
SENDEXITINIT	Y	Y					发送出口的初始字符串
TEMPMODEL	Y				Y		从中创建临时队列的模型队列名称
MSGRETENTION	Y				Y		连接使用者是否在输入队列中保留了不需要的消息
BROKERVER		Y				Y	要使用的代理的版本
BROKERPUBQ		Y				Y	代理输入队列（流队列）的名称
BROKERSUBQ		Y				Y	检索非长期订阅消息的队列名称
BROKERDURSUBQ				Y			检索长期订阅消息的队列名称
BROKERCCSUBQ		Y				Y	为 ConnectionConsumer 检索非长期订阅消息的队列名称
BROKERCCDSUBQ				Y			为 ConnectionConsumer 检索长期订阅消息的队列名称
BROKERQMGR		Y				Y	运行着代理的队列管理器
BROKERCONQ		Y				Y	代理的控制队列名称
EXPIRY			Y	Y			一个时间段，在该时间段后目的地上的消息将失效
PRIORITY			Y	Y			发送到目的地的消息的优先级
PERSISTENCE			Y	Y			发送到目的地的消息的持续性
TARGCLIENT			Y	Y			字段表明是否使用 MQSeries RFH2 格式与目标应用程序交换信息
ENCODING			Y	Y			该目的地使用的编码方案
QUEUE			Y				表示目的地的队列的基本名称
TOPIC				Y			表示目的地的主题的基本名称

注:

1. 对于 WSTCF、WSQCF、XATCF 和 XAQCF 对象，只能使用 BIND 传送类型。
2. 第343页的『附录A. 管理工具特性与可编程特性之间的映射』显示了工具设置的特性和可编程特性之间的关系。
3. TARGCLIENT 特性表明了是否使用 MQSeries RFH2 格式与目标应用程序交换信息。

   常量 MQJMS_CLIENT_JMS_COMPLIANT 表明将使用 RFH2 格式来发送信息。使用 MQ JMS 的应用程序了解 RFH2 格式。在与目标 MQ JMS 应用程序交换信息时应设置 MQJMS_CLIENT_JMS_COMPLIANT 常量。

常量 MQJMS_CLIENT_NONJMS_MQ 表明将使用 RFH2 格式来发送信息。通常，该值用于现有的 MQSeries 应用程序（即，它不处理 RFH2）。

特性相关性

某些特性在彼此之间有着相关性。这就是说，仅提供一个特性是毫无意义的，只有把另一个特性设置成某个特定的值时才有意义。会发生这种情况的两个特定特性是“客户机特性”和“出口化初始字符串”。

客户机特性

如果 TRANSPORT(CLIENT) 特性没有显式地设置在连接工厂上，则提供连接上使用的传送的工厂是 MQ Bindings。因此，不能在该连接工厂上配置任何客户机特性。这些特性是:

- HOST
- PORT
- CHANNEL
- CCSID
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT

如果尝试在没有为 CLIENT 设置 TRANSPORT 特性的情况下设置这些特性中的任何一个，将出错。

出口初始化字符串

除非已提供了相应的出口名，否则设置任何出口初始化字符串都是无效的。出口初始化特性有:

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

例如，指定 RECEXITINIT(myString) 时没有指定 RECEXIT(some.exit.classname) 将出错。

ENCODING 特性

ENCODING 可以采用的有效值比其它特性更复杂。编码特性是通过三个子特性来构造的:

整数编码	这是标准编码或反码编码
十进制编码	这是标准编码或反码编码
浮点编码	这是 IEEE 标准编码、IEEE 反码编码或 System/390®

ENCODING 由具有下列语法的三个字符组成的字符串来表示:

```
{N|R}{N|R}{N|R|3}
```

在该字符串中:

- N 表示标准编码
- R 表示反码编码
- 3 表示 System/390
- 第一个字符表示*整数编码*
- 第二个字符表示*十进制编码*
- 第三个字符表示*浮点编码*

它为 ENCODING 特性提供了十二种可能值的集合。

还有一个附加的值，字符串 NATIVE，它用于为 Java 平台设置适当的编码值。

下列示例显示了 ENCODING 的有效组合:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

样本错误条件

这个部分提供了在创建对象期间可能发生的错误条件。

未知特性

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  无法创建有效对象，请检查所提供的参数
  未知特性: PIZZA
```

无效的对象特性

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  无法创建有效对象，请检查所提供的参数
  无效 QCF 特性: PRI
```

无效的特性值类型

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  无法创建有效对象，请检查所提供的参数
  CCS 特性值无效: 英语
```

特性值超出有效范围

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  无法创建有效对象，请检查所提供的参数
  无效的 PRI 特性值: 12
```

特性冲突 - 客户机／绑定

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  无法创建有效对象，请检查所提供的参数
  上下文中无效特性: 客户机-绑定属性冲突
```

特性冲突 - 出口初始化

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  无法创建有效对象，请检查所提供的参数
  上下文中无效特性: 提供了 ExitInit 字符串
  没有 Exit 字符串
```

第2部分 用 MQ base Java 编程

第6章 针对程序员的介绍 45
为何要使用 Java 接口？ 45
MQSeries classes for Java 接口 45
Java 开发工具箱 46
MQSeries classes for Java 类库 46

第7章 编写 MQ base Java 程序 49
应编写小应用程序还是应用程序? 49
连接差异 49
客户机连接 49
绑定方式 50
定义要使用的连接 50
示例代码片段 50
示例小应用程序代码 50
将连接改为使用 VisiBroker for Java 53
示例应用程序代码 54
队列管理器上的操作 56
设置 MQSeries 环境 56
连接到队列管理器 56
访问队列与进程 57
处理消息 57
处理错误 58
获取和设置属性值 59
多线程程序 59
编写用户出口 61
| 连接合用 62
| 控制缺省连接池 62
| 缺省连接池和多个组件 64
| 提供不同的连接池 65
| 提供自己的 ConnectionManager 66
编译和测试 MQ base Java 程序 67
运行 MQ base Java 小应用程序 67
运行 MQ base Java 应用程序 68
在 CICS 事务处理服务器 OS/390 版下运行 MQ base Java 应用程序 68
跟踪 MQ base Java 程序 68

第8章 取决于环境的行为 71
核心类细节 71
核心类的限制与变化 72
运行于其它环境下的版本 5 扩展 74

第9章 MQ base Java 类和接口 77
MQChannelDefinition 78
变量 78
构造器 79
MQChannelExit 80
变量 80
构造器 82
MQDistributionList 83
构造器 83
方法 83
MQDistributionListItem 85
变量 85
构造器 85
MQEnvironment 86
变量 86
构造器 88
方法 89
MQException 91
变量 91
构造器 91
MQGetMessageOptions 93
变量 93
构造器 96
MQManagedObject 97
变量 97
构造器 97
方法 98
MQMessage 100
变量 100
构造器 108
方法 108
MQMessageTracker 119
变量 119
| MQPoolServices 121
| 构造器 121
| 方法 121
| MQPoolServicesEvent 122
| 变量 122
| 构造器 122
| 方法 123
| MQPoolToken 124
| 构造器 124
MQProcess 125
构造器 125
方法 125
MQPutMessageOptions 127
变量 127
构造器 129
MQQueue 130
构造器 130
方法 130
MQQueueManager 138
变量 138
构造器 138
方法 140
| MQSimpleConnectionManager 148
| 变量 148
| 构造器 148
| 方法 148
MQC 150

© Copyright IBM Corp. 1997, 2001

| MQPoolServicesEventListener 151
| 方法 151
| MQConnectionManager 152
MQReceiveExit 153
方法 153
MQSecurityExit 155
方法 155
MQSendExit 157
方法 157
| ManagedConnection 159
| 方法 159
| ManagedConnectionFactory 162
| 方法 162
| ManagedConnectionMetaData 164
| 方法 164

第6章 针对程序员的介绍

本章包含了程序员适用的一般信息。有关编写程序的详细信息，请参阅第49页的『第7章 编写 MQ base Java 程序』。

为何要使用 Java 接口?

作为 MQSeries 应用程序的开发人员，MQSeries classes for Java 编程接口可以使您利用到 Java 的许多优点:

- Java 编程语言是**易于使用的**。

  它不需要头文件、指针、结构、联合以及运算符重载。用 Java 编写的程序比同类的 C 和 C++ 更容易开发与调试。

- Java 是**面向对象的**。

  Java 面向对象的特性能与 C++ 媲美，但它不具有多重继承性。取而代之的是，Java 使用了接口的概念。

- Java先天就是**分布式的**。

  Java 类库包含了使用 TCP/IP 协议（如 HTTP、FTP）的例行程序库。Java 程序可以象访问文件系统一样方便地访问 URL。

- Java 是**健壮的**。

  Java 非常着重于可能发生问题的早期检查、动态（运行时）检查以及消除有出错可能性的情况。Java 使用引用的概念，这将消除覆写内存和破坏数据的可能性。

- Java 是**安全的**。

  Java 要在连网的或分布式环境中运行，并已在安全性问题上下了很大功夫。Java 程序不会超越其运行时堆栈，也不会破坏超过其处理空间的内存。当从因特网下载 Java 程序时，这些程序根本不会读或写本地文件。

- Java 程序是**可移植的**。

  Java 规范中没有“实现相关性”方面的内容。Java 编译器生成体系无关的目标文件格式。只要存在 Java 运行时系统，编译代码就可以在许多处理器上执行。

如果使用 MQSeries classes for Java 编写应用程序，用户可以从因特网上为程序下载 Java 字节代码（称为*小应用程序*）。然后，用户可以在自己的机器上运行这些小应用程序。这意味着访问 Web 服务器的用户不需要先在其机器上安装，就能装入并运行应用程序。

需要更新程序时，可以更新 Web 服务器上的副本。当用户下次访问小应用程序时，他们将接收到最新版本。这能够显著地降低成本，而安装并更新传统客户机应用程序需要涉及大量的台式机。

若将小应用程序放在能从企业防火墙外访问的 Web 服务器上，则无论谁都能在因特网上下载并使用您的应用程序。这意味着 MQSeries 系统可以从网际网上的任何地方获取消息。这将开启建立一组全新的因特网可访问服务以及电子商业应用的大门。

MQSeries classes for Java 接口

过程性 MQSeries 应用编程接口围绕下列动词建立:

© Copyright IBM Corp. 1997, 2001

MQ base Java 接口

```
MQBACK、MQBEGIN、MQCLOSE、MQCMIT、MQCONN、MQCONNX、
MQDISC、MQGET、MQINQ、MQOPEN、MQPUT、MQPUT1、MQSET
```

作为一个参数，这些动词都取得一个指向要操作的 MQSeries 对象的句柄。因为 Java 是面向对象的，所以 Java 编程接口也是面向对象的。程序由一组 MQSeries 对象构成，通过调用那些对象上的方法来执行操作，如下例所示。

在使用过程性接口时，可以通过调用 MQDISC(Hconn, CompCode, Reason) 断开与队列管理器的连接，其中 *Hconn* 是至队列管理器的句柄。

在 Java 接口中，队列管理器由对象类 MQQueueManager 来表示。通过调用该类上的 disconnect() 方法断开与队列管理器的连接。

```
// declare an object of type queue manager

MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

Java 开发工具箱

在可以编译您所编写的任何小应用程序或应用程序之前，必须有所使用开发平台的 Java 开发工具箱 (JDK)。JDK 中包含了所有标准的 Java 类、变量、构造器以及 MQSeries classes for Java 类依赖的接口。它还包含了在每个支持平台上编译与运行小应用程序以及应用程序所需的工具。

JDK 可以从万维网上的“ IBM 软件下载目录”中下载，它的位置是:

```
http://www.ibm.com/software/download
```

也可以使用 IBM Visual Age for Java 集成开发环境所包括的 JDK 来开发应用程序。

若要在 AS/400 平台上编译 Java 应用程序，必须首先安装:
- AS/400 Developer Kit for Java, 5769-JV1
- Qshell Interpreter, OS/400 (5769-SS1) Option 30

MQSeries classes for Java 类库

MQSeries classes for Java 是一组能使 Java 小应用程序以及应用程序与 MQSeries 进行交互的Java 类。

提供了以下类:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage

- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

提供的 Java 接口包括:
- MQC
- MQPoolServicesEventListener
- MQReceiveExit
- MQSecurityExit
- MQSendExit

此外还提供了下列 Java 接口的实现。但是，我们不希望应用程序直接使用这些接口:
- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

在 Java 中，包代表了一种将相关的类组合在一起的机制。MQSeries 类与接口将以 Java 包交付，称为 com.ibm.mq。要将 MQSeries classes for Java 包包含在程序中，请在源文件顶部添加如下行:

```
import com.ibm.mq.*;
```

第7章 编写 MQ base Java 程序

要使用 MQSeries classes for Java 来访问 MQSeries 队列，可以编写 Java 应用程序，在其中包含将消息放入 MQSeries 队列或从队列中取出的调用。这些程序可以采用 Java *小应用程序*、Java *小服务程序*或 Java *应用程序*的形式。

本章提供了编写与 MQSeries 系统交互的 Java 小应用程序、小服务程序和应用程序的信息。有关每个类的详细信息，请参阅第77页的『第9章 MQ base Java 类和接口』。

应编写小应用程序还是应用程序?

是编写小应用程序、小服务程序还是应用程序，应取决于要使用的连接，以及将在哪里运行这些程序。

小应用程序与应用程序的主要区别是:

- 小应用程序运行在小应用程序查看器或 Web 浏览器中，小服务程序运行在 Web 应用程序服务器中，而应用程序则是单独运行的。
- 小应用程序可从 Web 服务器下载到 Web 浏览器机器，但应用程序和小服务程序却不能。

下列通则适用:

- 若要从没有本地安装 MQSeries classes for Java 的机器上运行程序，则应该编写小应用程序。
- MQSeries classes for Java 的本机绑定方式不支持小应用程序。因此，如果要在所有连接方式中都能使用程序，包括本机绑定方式，则必须编写小服务程序或应用程序。

连接差异

为 MQSeries classes for Java 编写的方法取决于要使用的连接方式。

客户机连接

当 MQSeries classes for Java 作为客户机使用时，其用法与 MQSeries C 客户机相似，但有下列不同之处:

- 只支持 TCP/IP。
- 不支持连接表。
- 不在启动时读任何 MQSeries 环境变量。
- 在通道定义和环境变量中存储的信息将存储在一个称为 MQEnvironment 的类中。换句话说，这些信息在连接成功时可作为参数来传递。
- 出错和异常情况将写入 MQException 类指定的日志中。缺省错误目的地是 Java 控制台。

MQSeries classes for Java 客户机不支持 MQBEGIN 动词或快速绑定。

© Copyright IBM Corp. 1997, 2001

有关 MQSeries 客户机上的一般信息，请参阅 *MQSeries 客户机*。

注: 当使用 VisiBroker 连接时，MQEnvironment 中的用户标识与口令设置将不转发到 MQSeries 服务器上。有效的用户标识应可以应用到 IIOP 服务器上。

绑定方式

MQSeries classes for Java 的绑定方式与客户机方式有以下不同:

- MQEnvironment 类提供的大多数参数都被忽略
- 绑定支持 MQBEGIN 动词和至 MQSeries 队列管理器的快速绑定

注: MQSeries AS/400 版不支持使用 MQBEGIN 来启动由队列管理器协调的全局工作单元。

定义要使用的连接

连接由 MQEnvironment 类中的变量设置确定。

MQEnvironment.properties

可包含下列键／值对:

- 对于客户机和绑定连接:

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES
```

- 对于 VisiBroker 连接:

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_VISIBROKER
MQC.ORB_PROPERTY, orb
```

MQEnvironment.hostname

按如下设置该变量的值:

- 对于客户机连接，设置其为要连接到的 MQSeries 服务器主机名
- 对于绑定方式，设置其为空

示例代码片段

这一部分包含了两个示例代码片段；第51页的图1和第54页的图2。每个示例都使用特定连接，并包括描述使用替代连接所必需的更改的说明。

示例小应用程序代码

下列代码片段说明了一个小应用程序，它使用 TCP/IP 连接:

1. 连接到队列管理器
2. 将一条消息放入 SYSTEM.DEFAULT.LOCAL.QUEUE
3. 将该消息返回

```
// ===========================================================================
//
// Licensed Materials - Property of IBM
//
// 5639-C34
//
// (c) Copyright IBM Corp. 1995,1999
//
// ===========================================================================
// MQSeries 客户机 Java 版 样本小应用程序//
// 此样本是通过使用小应用程序查看器和 HTML 文件来运行的小应用程序。
// 使用命令: -
//           appletviewer MQSample.html
// 输出在命令行中, “不”在小应用程序查看器窗口中。
//
// 注意, 如果接收到 MQSeries 错误 2 原因 2059 并确认
// MQSeries 和 TCP/IP 设置是正确的, 则
// 应该单击“小应用程序查看器”窗口中的“小应用程序”选项
// 选择特性, 并将“网络访问”更改为不受限。
import com.ibm.mq.*;                              // 包含 MQSeries classes for Java 包
public class MQSample extends java.applet.Applet
{

  private String hostname = "your_hostname";      // 定义要连接到的
                                                  // 主机名称
  private String channel  = "server_channel";     // 定义客户机将使
                                                  // 用的通道的名称
                                                  // 注, 假设 MQSeries 服务器
                                                  // 正在缺省 TCP/IP 端口
                                                  // 1414 上侦听
  private String qManager = "your_Q_manager";     // 定义队列名称
                                                  // 要连接到的管
                                                  // 理器对象。

  private MQQueueManager qMgr;                    // 定义队列管理器对象
  // 当调用类时, 首先进行初始化。

  public void init()
  {
// 设置 MQSeries 环境
     MQEnvironment.hostname = hostname;           // 可以在这里直接
                                                  // 放入主机 &
     MQEnvironment.channel  = channel;            // 通道字符串!

     MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY,//Set TCP/IP or server
                               MQC.TRANSPORT_MQSERIES);//Connection

  } // 结束初始化
```

图 1. MQSeries classes for Java 示例小应用程序 (1/3)

示例代码

```
public void start()
{

  try {
    // 创建至队列管理器的连接
    qMgr = new MQQueueManager(qManager);

    // 设置希望打开的选项...
   // 注. 在 Java 中, MQSeries 所有选项使用前缀 MQC。
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                      MQC.MQOO_OUTPUT ;
    // 现在指定希望打开的队列, 并打开选项...

    MQQueue system_default_local_queue =
            qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                             openOptions);

// 定义单个 MQSeries 消息, 并写入一些 UTF 格式的文本..
    MQMessage hello_world = new MQMessage();
    hello_world.writeUTF("Hello World!");

    // 指定消息选项...

    MQPutMessageOptions pmo = new MQPutMessageOptions();  // 接受缺省
                                                          // 值, 与
                                                          // MQPMO_DEFAULT
                                                          // 常量相同
    // 将消息放入队列

    system_default_local_queue.put(hello_world,pmo);

    // 再取会消息...
   // 首先定义一个 MQSeries 消息缓冲区用以检索消息到其中..
    MQMessage retrievedMessage = new MQMessage();
    retrievedMessage.messageId = hello_world.messageId;
    // 设置取出消息选项..

    MQGetMessageOptions gmo = new MQGetMessageOptions(); // 接受与
                                                          // MQGMO_DEFAULT
                                                          // 相同的缺省值
    // 从队列中取出消息..

    system_default_local_queue.get(retrievedMessage, gmo);
    // 并通过显示 UTF 消息文本证明我们有消息

    String msgText = retrievedMessage.readUTF();
    System.out.println("The message is: " + msgText);

    // 关闭队列

    system_default_local_queue.close();

    // 从队列管理器断开

    qMgr.disconnect();

  }
  // 如果上述发生错误, 请尝试标识是什么错误。
  // 是否是 MQSeries 错误?
```

图 1. MQSeries classes for Java 示例小应用程序 (2/3)

```
    catch (MQException ex)
    {
System.out.println("An MQSeries error occurred : Completion code " +
                        ex.completionCode +
                        " Reason code " + ex.reasonCode);
    }
    // 是否是 Java 缓冲区空间错误?
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the
      message buffer: " + ex);
    }

  } // 结束开始

} // 结束样本
```

图 1. MQSeries classes for Java 示例小应用程序 (3/3)

将连接改为使用 VisiBroker for Java

将以下行:

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                             MQC.TRANSPORT_MQSERIES);
```

改为:

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                             MQC.TRANSPORT_VISIBROKER);
```

并添加下列行来初始化 ORB(对象请求代理):

```
ORB orb=ORB.init(this,null);
MQEnvironment.properties.put(MQC.ORB_PROPERTY,orb);
```

还需要将下列 import 语句添加到文件的开头:

```
import org.omg.CORBA.ORB;
```

若使用 VisiBroker, 则不需要指定端口号或通道。

示例应用程序代码

下列代码说明一个简单的应用程序，　它使用绑定方式:

1. 连接到队列管理器
2. 将消息放入 SYSTEM.DEFAULT.LOCAL.QUEUE
3. 再次取返回的消息

```
// ======================================================================
// Licensed Materials - Property of IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 1999
// ======================================================================
// MQSeries classes for Java 样本应用程序
//
// 本样本将通过使用命令: - java MQSample 来运行 Java 应用程序

import com.ibm.mq.*;            // 包含 MQSeries classes for Java 包

public class MQSample
{
  private String qManager = "your_Q_manager";  // 定义要连接到的
                                               // 队列管理器的名称。
  private MQQueueManager qMgr;                 // 定义队列管理器
                                               // 对象
  public static void main(String args[]) {
     new MQSample();
  }
  public MQSample() {
   try {

      // 创建至队列管理器的连接
      qMgr = new MQQueueManager(qManager);

      // 设置希望打开的选项...
      // 注. Java 中所有 MQSeries 选项都使用前缀 MQC。

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;
      // 现在指定希望打开的队列,
      // 并打开选项...
      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);

      // 定义单个 MQSeries 消息, 并写入一些 UTF 格式的文本..

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");
      // 指定消息选项...

      MQPutMessageOptions pmo = new MQPutMessageOptions(); // 接受与
                                                           // MQPMO_DEFAULT
                                                           // 相同的缺省值
```

图 2. MQSeries classes for Java 样本应用程序 (1/2)

```
      // 将消息放入队列
      system_default_local_queue.put(hello_world,pmo);

      // 再取会消息...
      // 首先定义一个 MQSeries 消息缓冲区用以检索消息到其中..

      MQMessage retrievedMessage = new MQMessage();
      retrievedMessage.messageId = hello_world.messageId;

      // 设置取出消息选项...

      MQGetMessageOptions gmo = new MQGetMessageOptions(); // 接受与 MQGMO_DEFAULT
                                                           // 相同的缺省值
      // 从队列中取出消息...

      system_default_local_queue.get(retrievedMessage, gmo);

      // 并通过显示 UTF 消息文本证明我们有消息

      String msgText = retrievedMessage.readUTF();
      System.out.println("The message is: " + msgText);
      // 关闭队列...
      system_default_local_queue.close();
      // 从队列管理器断开

      qMgr.disconnect();
    }
    // 如果上述发生错误，请尝试标识是什么错误
      // 是否是n MQSeries 错误？
    catch (MQException ex)
    {
      System.out.println("An MQSeries error occurred : Completion code " +
                         ex.completionCode + " Reason code " + ex.reasonCode);
    }
    // 是否是 Java 缓冲区空间错误？
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the message buffer: " + ex);
    }
  }
} // 结束样本
```

图 2. MQSeries classes for Java 样本应用程序 (2/2)

队列管理器上的操作

这个部分描述了如何使用 MQSeries classes for Java 连接到队列管理器，以及从队列管理器断开。

设置 MQSeries 环境

注: 以绑定方式使用 MQSeries classes for Java 时，本步骤不是必需的。在此情况下，请直接执行『连接到队列管理器』。在使用客户机连接到队列管理器之前，必须设置好 MQEnvironment。

基于 ″C″ 的 MQSeries 客户机根据环境变量来控制 MQCONN 调用的进行。由于 Java 小应用程序不能访问环境变量，所以 Java 编程接口应包含一个类 MQEnvironment。这个类允许您指定在连接尝试期间将用到的下列细节:

- 通道名
- 主机名
- 端口号
- 用户标识
- 口令

要指定通道名和主机名，请使用下列代码:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel  = "java.client.channel";
```

这与 MQSERVER 环境变量的以下设置是等价的:

```
"java.client.channel/TCP/host.domain.com".
```

缺省情况下，Java 客户机尝试在端口 1414 连接到 MQSeries 侦听器。要指定一个不同的端口，请使用代码:

```
MQEnvironment.port = nnnn;
```

用户标识和口令缺省为空。要指定非空的用户标识或口令，请使用代码:

```
MQEnvironment.userID   = "uid";  // 等价于环境变量 MQ_USER_ID
MQEnvironment.password = "pwd";  // 等价于环境变量 MQ_PASSWORD
```

注: 如果正在使用 VisiBroker for Java 来设置连接，请参阅第53页的『将连接改为使用 VisiBroker for Java』。

连接到队列管理器

现在，准备通过创建一个新的 MQQueueManager 类实例，连接到队列管理器:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

要从队列管理器断开，请调用队列管理器上的 disconnect() 方法:

```
queueManager.disconnect();
```

如果调用 disconnect 方法，则通过该队列管理器访问的所有打开的队列和进程都将被关闭。但这对于在结束使用这些资源时显式地关闭它们是一个很好的编程方法。要做到这一点，请使用 close() 方法。

队列管理器上的 commit() 和 backout() 方法替换了与过程性接口一起使用的 MQCMIT 和 MQBACK 调用。

访问队列与进程

要访问队列与进程，请使用 MQQueueManager 类。MQOD（对象描述符结构）已被放入这些方法的参数中。例如，要打开队列管理器“queueManager”上的队列，请使用下列代码:

```
MQQueue queue = queueManager.accessQueue("qName",
                             MQC.MQOO_OUTPUT,
                             "qMgrName",
                             "dynamicQName",
                             "altUserId");
```

options 参数与 MQOPEN 调用中的 Options 参数相同。

accessQueue 方法返回 MQQueue 类的一个新对象。

使用完队列后，请按如下示例使用 close() 方法关闭它:

```
queue.close();
```

使用 MQSeries classes for Java 时，还可以通过使用 MQQueue 构造器来创建队列。参数与带有队列管理器参数的 accessQueue 方法的参数是完全相同的。例如:

```
MQQueue queue = new MQQueue(queueManager,
                             "qName",
                             MQC.MQOO_OUTPUT,
                             "qMgrName",
                             "dynamicQName",
                             "altUserId");
```

用这种方法构造队列对象使您能够编写 MQQueue 的自用子类。

要访问一个进程，请在使用 accessQueue 的地方使用 accessProcess 方法。这个方法中没有*动态队列名*参数，因为它不适用于进程。

accessProcess 方法返回 MQProcess 类的一个新对象。

使用完进程对象后，请按如下示例使用 close() 方法关闭它:

```
process.close();
```

使用 MQSeries classes for Java 时，还可以通过使用 MQProcess 构造器来创建进程。参数与 accessProcess 方法的参数是完全相同的，加上一个附加的队列管理器参数。用这种方法构造进程对象使您能够编写 MQProcess 的自用子类。

处理消息

使用 MQQueue 类的 put() 方法将消息放入队列。使用 MQQueue 类的 get() 方法将消息取出队列。与过程性接口不同的是，过程性接口用 MQPUT 与 MQGET 放入与取出字节数组，Java 编程语言则是放入与取出 MQMessage 类的实例。MQMessage 类封装了包含实际消息数据的数据缓冲区以及描述该消息的所有 MQMD（消息描述符）参数。

要构建一条新的消息，请创建一个新的 MQMessage 类实例，并使用 writeXXX 方法将数据放入消息缓冲区中。

消息处理

在创建新的消息实例时，所有 MQMD 参数都将自动地设置为它们的缺省值，*MQSeries Application Programming Reference* 中定义了这些缺省值。MQQueue 的 put() 方法也取出一个 MQPutMessageOptions 类的实例作为参数。该类表示 MQPMO 结构。以下这个示例将创建一个消息并把它放入队列:

```
// 构建一条包含年龄和名字的新消息
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Wendy Ling";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// 使用缺省的放入消息选项...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// 放入消息！
queue.put(myMessage,pmo);
```

MQQueue 的 get() 方法返回一个 MQMessage 的新实例，它表示刚从队列中取出的消息。它也把 MQGetMessageOptions 类的实例作为参数。该类表示 MQGMO 结构。

不需要指定最大消息大小，因为 get() 方法会自动调整其内部的缓冲区大小以适合进入消息。使用 MQMessage 类的 readXXX 方法访问返回消息中的数据。

列样本显示了如何从队列取出消息:

```
// 从Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // 有缺省值

// 抽取消息数据
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

可通过设置 *encoding* 成员变量串改变读和写方法所使用的数字格式。

可通过设置 *characterSet* 成员变量改变读写字符串使用的字符集。

更详细的内容，请参阅第100页的『MQMessage』。

注: MQMessage 的 writeUTF() 方法将自动编码字符串的长度及其包含的 Unicode 字节。当消息要被另一个 Java 程序（使用 readUTF()）读取时，这是发送字符串信息最简单的方法。

处理错误

Java 接口中的方法不返回完成码与原因码。而是当 MQSeries 调用的完成码与原因码不同为零时，形成一个异常。这简化了程序逻辑，使您无需在每次调用 MQSeries 之后都要去检查返回码。您可以决定程序中要处理可能会发生故障的位置。在这些位置上，可以将代码用“try”和“catch ”块围绕，如下例所示:

```
try {
myQueue.put(messageA,putMessageOptionsA);
myQueue.put(messageB,putMessageOptionsB);
}
```

```
catch (MQException ex) {
// 只有当两个 put 方法中的一个
// 产生一个非零的完成码或原
// 因码，才执行这个代码块。
System.out.println("An error occurred during the put operation:" +
                          "CC = " + ex.completionCode +
                          "RC = " + ex.reasonCode);
}
```

获取和设置属性值

对于许多公共属性，类 MQManagedObject、MQQueue、MQProcess 以及 MQQueueManager 都包含了 getXXX() 与 setXXX() 方法。这些方法允许获取和设置它们的属性值。注意，MQQueue 方法当且仅当在打开队列时设置了正确的“inquire”和“set”标志时才能起作用。

对于不公共的属性，MQQueueManager、MQQueue 和 MQProcess 类都从一个称为 MQManagedObject 的类中继承。该类定义了 inquire() 和 set() 接口。

当使用 *new* 运算符创建了一个新的队列管理器对象时，它自动地为 ’inquiry’ 打开。当使用 accessProcess() 方法访问进程对象时，该对象将自动对于 ’inquiry’ 打开。当使用 accessQueue() 方法访问进程对象时，该对象不会自动对于 ’inquiry’ 或 ’set’ 打开。这是因为自动添加这些选项会导致某些类型的远程队列出现问题。要在队列上使用 inquire、set、getXXX 和 setXXX 方法，必须在 accessQueue() 方法的 openOptions 参数上指定适当的 ’inquire’ 和 ’set’ 标志。

inquire 和 set 方法取三个参数:
- selectors 数组
- intAttrs 数组
- charAttrs 数组

不需要在 MQINQ 找到的 SelectorCount、IntAttrCount 和 CharAttrLength 参数，因为 Java 中的数组长度总是已知的。下列样本显示了如何在队列上查询:

```
// 在队列上查询
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

多线程程序

Java 中很难避免多线程程序。请考虑一个简单的程序，它连接到队列管理器，并在启动时打开队列。该程序在屏幕上显示单个按钮。当用户按下按钮，程序从队列中提取一个消息。

多线程程序

Java 运行时环境是继承性多线程的。因此，应用程序的初始化将在一个线程中进行，作为响应按钮按下而执行的代码在另一个线程中进行（用户界面线程）。

因为不能在多线程上共享句柄，所以基于“C”的 MQSeries 客户机将会产生一个问题。MQSeries classes for Java 打破了这个约束，允许队列管理器对象及其关联的队列与进程对象能够在多线程之间共享。

MQSeries classes for Java 的实现确保了，对于给定的连接（MQQueueManager 对象实例）而言，所有对目标 MQSeries 队列管理器的访问都是同步的。因此，如果一个线程要发出对队列管理器的调用，那么只有当该连接进程中的所有其它调用都完成后才能实现。如果程序内的多个线程需要同时访问同一个队列管理器，应为需要同时访问的每个线程都创建一个新的 newMQQueueManager 对象。（这等同于为每个线程发出一个单独的 MQCONN 调用。）

注: 在 CICS 事务处理服务器 OS/390 版环境中，只有主（第一个）线程才能发出CICS 或 MQSeries 调用。因此，在这种环境下，要在线程之间共享 MQQueueManager 或 MQQueue 对象或在子线程上创建 MQQueueManager 是不可能的。

编写用户出口

MQSeries classes for Java 允许您提供自己的发送、接收以及安全性出口。

要实现一个出口，需定义一个新的 Java 类，实现适当的接口。MQSeries 包中定义了三个出口:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

下列样本定义了一个类，它实现了所有三个接口:

```
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {

  // 这个方法来自 send 出口
public byte[] sendExit(MQChannelExit channelExitParms,                        MQChannelDefini
                          byte agentBuffer[])
  {
     // 在这里填写 send 出口的主体
  }


  // 这个方法来自 receive 出口
public byte[] receiveExit(MQChannelExit channelExitParms,                        MQChannelDef
                             byte agentBuffer[])
  {
     // 在这里填写 receive 出口的主体
  }


  // 这个方法来自 security 出口
public byte[] securityExit(MQChannelExit channelExitParms,                        MQChannelDe
                              byte agentBuffer[])
  {
     // 在这里填写 security 出口的主体
  }

}
```

每个出口都传递了一个 MQChannelExit 和一个 MQChannelDefinition 对象实例。这些对象表示定义在过程性接口中的 MQCXP 和 MQCD 结构。

对于 Send 出口，*agentBuffer* 参数包含了将要发送的数据。对于 Receive 或 Security 出口，*agentBuffer* 参数包含了已经接收到的数据。不需要长度参数，因为表达式 agentBuffer.length 指出了数组的长度。

对于 Send 与 Security 出口，出口码应返回希望发送到服务器的字节数组。对于 Receive 出口，出口码应返回希望 MQSeries classes for Java 解释的修改过的数据。

有可能最简单的出口主体是:

```
{
  return agentBuffer;
}
```

若程序要作为下载的 Java 小应用程序运行，那么所应用的安全性限制是不能读写任何本地文件。若出口需要配置文件，那么可将文件置于 Web 上，然后使用 java.net.URL 类下载并检查其内容。

连接合用

MQSeries classes for Java 版本 5.2 为处理多个至 MQSeries 队列管理器的连接的应用程序提供了附加支持。当不再需要连接时，不是摧毁连接，而是可以将它放入连接池中，等以后再使用。这为要连续或任意连接到队列管理器的应用程序和中间件提供了重要的性能增强。

MQSeries 提供了一个缺省连接池。应用程序可以通过 MQEnvironment 类注册和取消注册，从而激活或释放应用程序。如果该池是活动的，则当 MQ base Java 构造 MQQueueManager 对象时，它将搜索这个缺省池，然后重使用池中合适的连接。当发生 MQQueueManager.disconnect() 调用时，基本连接将返回该池。

或者，应用程序可以构造一个 MQSimpleConnectionManager 连接池用于特定目的。然后，应用程序可以在构造 MQQueueManager 对象期间指定该池，或是将该池传递给 MQEnvironment，把它作为缺省连接池来使用。

MQ base Java 还提供了 Java 2 Platform Enterprise Edition (J2EE) 连接器体系结构的部分实现。运行在带 JAAS（Java 认证和授权服务）1.0 的 Java 2 v1.3 JVM 下的应用程序可以通过实现 **javax.resource.spi.ConnectionManager** 接口提供它们自己的连接池。而且，该接口可以在 MQQueueManager 构造器上指定，或作为缺省值来指定。

控制缺省连接池

请考虑下列示例应用程序，MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : （对 qmgr 执行一些操作）
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 从命令行获取本地队列管理器的列表，依次连接到每个队列管理器，然后执行一些操作。但是，当命令行多次列出同一个队列管理器时，连接一次该队列管理器且多次重使用该连接将更有效。

MQ base Java 为您提供了可用来做到这一点的缺省连接池。要启用该池，使用 MQEnvironment.addConnectionPoolToken() 方法中的一个。要禁用该池，使用 MQEnvironment.removeConnectionPoolToken()。

以下这个示例程序 MQApp2 的功能与 MQApp1 相同，但它对每个队列管理器只连接一次。

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         for (int i=0; i<args.length; i++) {
```

```
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (对 qmgr 执行一些操作)
            :
            qmgr.disconnect();
         }
         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

第一个用黑体字表示的行通过使用注册具有 MQEnvironment 的 MQPoolToken 对象来激活缺省连接池。

MQQueueManager 构造器现在开始搜索该池以获取适当的连接，如果无法找到现有的连接，则仅创建一个至该队列管理器的连接。qmgr.disconnect() 调用将把该连接返回到池以便以后再次使用。这些 API 调用与示例应用程序 MQApp1 相同。

第二个突出显示的行将释放缺省的连接池，这将摧毁存储在池中的所有队列管理器连接。这非常重要，因为，不然的话，应用程序将在池中含有许多活动队列管理器连接的情况下终止。这种情况将导致队列管理器日志中出现出错信息。

缺省连接池中最多可存储十个不使用的连接，并使这些不使用的连接最多保持活动状态五分钟。应用程序可以更改这些内容（详细内容，请参阅第65页的『提供不同的连接池』）。

除了使用 MQEnvironment 来提供 MQPoolToken 外，应用程序构造自己的 MQPoolToken:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

有些应用程序或中间件供应商可能会提供 MQPoolToken 的子类，以便将信息传递到定制连接池中。您可以使用上述方法构造它们并把它们传递给 addConnectionPoolToken()，以便把附加信息传递给连接池。

缺省连接池和多个组件

MQEnvironment 含有一个已注册 MQPoolToken 对象的静态集。要从该集合中添加或除去 MQPoolToken，请使用以下方法:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

应用程序有可能是由多个独立存在的组件构成的，并使用队列管理器来执行任务。在此类应用程序中，每个组件都将把一个 MQPoolToken 添加到 MQEnvironment 以设置其存活时间。

例如，示例应用程序 MQApp3 创建了十个线程并启动了每个线程。每个线程都注册它们自己的 MQPoolToken，等待一段时间后，然后连接到队列管理器。当线程断开后，它将除去其自己的 MQPoolToken。

当 MQPoolToken 集合中至少还有一个令牌时，缺省连接池将保持活动，所以它在应用程序整个运行过程中都将保持活动。应用程序不需要在整个过程中保持一个主对象来控制线程。

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}
class MQApp3_Thread extends Thread
{
      long time;
      public MQApp3_Thread(long time)
      {
         this.time=time;
      }
      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}
         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

提供不同的连接池

这一部分将描述如何使用类 **com.ibm.mq.MQSimpleConnectionManager** 来提供不同的连接池。该类为连接合用提供了基本设施，应用程序可以使用它来定制池的行为。

一旦例示，MQQueueManager 构造器上就可以指定一个 MQSimpleConnectionManager。然后， MQSimpleConnectionManager 管理位于构造的 MQQueueManager 的连接。如果 MQSimpleConnectionManager 包含一个合适的合用连接，那么将重使用该连接，并在 MQQueueManager.disconnect() 调用后将该连接返回到 MQSimpleConnectionManager。

下列代码片段证明了这一行为:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
:（对 qmgr 执行一些操作）
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
:（对 qmgr2 执行一些操作）
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

当 qmgr.disconnect() 调用后，在首个 MQQueueManager 构造器期间打造的连接存储在 myConnMan 中。然后，该连接将在第二次调用 MQQueueManager 构造器期间被重使用。

第二行启用了 MQSimpleConnectionManager。最后一行禁用了 MQSimpleConnectionManager，摧毁了池中保留的所有连接。缺省情况下，MQSimpleConnectionManager 在 MODE_AUTO 中，这将在这一节的以后部分中描述。

MQSimpleConnectionManager 按最近使用原则分配连接，并按最不常用原则摧毁连接。缺省情况下，如果连接不使用的时间超过 5 分钟或是池中不使用的连接超过 10 个时，将摧毁连接。可以使用下面的内容改变这些值:

- MQSimpleConnectionManager.setTimeout()
- MQSimpleConnectionManager.setHighThreshold()

当 MQQueueManager 构造器上没有提供“连接管理器”时，也可以设置一个 MQSimpleConnectionManager 作为缺省连接池来使用。

下列应用程序证明了这一点:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String[] args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setHighThreshold(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

用黑体字表示的行表示设置一个 MQSimpleConnectionManager。它被设置成:

- 当连接有一个小时没有被使用时摧毁它。
- 把池中可保留的不使用的连接数限制为 50
- MODE_AUTO(实际上是缺省值)。这表示,池只有在它就是缺省连接管理器并且 MQEnvironment 含有的 MQPoolTokens 集中至少还有一个令牌时,才保持活动。

然后,新的 MQSimpleConnectionManager 被设置成缺省连接管理器。

最后一行,应用程序调用 MQApp3.main()。这将运行多个线程,而每个线程都独立使用 MQSeries。这些线程将在它们继续连接时使用 myConnMan。

提供自己的 ConnectionManager

在安装了带 JAAS 1.0 的 Java 2 v1.3 后,应用程序和中间件供应商将可以提供连接池的替代实现。MQ base Java 部分地实现了“J2EE 连接器体系结构”。**javax.resource.spi.ConnectionManager** 的实现可为作为缺省连接管理器来使用或在 MQQueueManager 构造器上指定。

MQ base Java 遵循了“J2EE 连接器体系结构”的“连接管理”合同。请在阅读这个部分的同时阅读“J2EE 连接器体系结构”的“连接管理”合同(参考 Sun 的 Web 站点 `http://java.sun.com`)。

ConnectionManager 接口只定义了一个方法:

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

MQQueueManager 构造器调用适当 ConnectionManager 上的 allocateConnection。它把适当的 ManagedConnectionFactory 和 ConnectionRequestInfo 实现作为参数传递以描述所需的连接。

ConnectionManager 搜索池中用等价的 ManagedConnectionFactory 和 ConnectionRequestInfo 对象创建的 javax.resource.spi.ManagedConnection 对象。如果 ConnectionManager 找到任何合适的 ManagedConnection 对象,它将创建包含候选 ManagedConnections 的 java.util.Set。然后,ConnectionManager 调用以下语句:

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

ManagedConnectionFactory 的 MQSeries 实现忽略“subject”参数。这个方法从集中选择并返回一个合适的 ManagedConnection，如果没有找到合适的 ManagedConnection，则返回空。如果池中没有适合的 ManagedConnection，ConnectionManager 只能使用以下语句创建一个:

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

再次忽略“subject”参数。这个方法连接到 MQSeries 队列管理器并返回一个表示最新实现的连接的 javax.resource.spi.ManagedConnection 实现。一旦 ConnectionManager 获得一个 ManagedConnection（无论是从池还是新创建的），它将使用以下方法创建一个连接句柄:

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

这个连接句柄可以从 allocateConnection() 返回。

ConnectionManager 应通过以下方法在 ManagedConnection 中注册一个注意项:

```
mc.addConnectionEventListener()
```

如果在连接上发生错误，或者在调用 MQQueueManager.disconnect() 时，将通知 ConnectionEventListener。当调用 MQQueueManager.disconnect() 时，ConnectionEventListener 可以执行以下步骤之一:

- 使用 mc.cleanup() 调用复位 ManagedConnection，然后把 ManagedConnection 返回到池中
- 使用 mc.destroy() 调用摧毁 ManagedConnection

如果要把该 ConnectionManager 作为缺省 ConnectionManager，它还可以在 MQPoolTokens 的 MQEnvironment-managed 集的状态中注册一个注意项。要做到这点，首先应构造一个 MQPoolServices 对象，然后使用 MQPoolServices 对象注册一个 MQPoolServicesEventListener 对象:

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

当把 MQPoolToken 添加到集合中或从集合中除去，或者，当缺省 ConnectionManager 更改时，将通知侦听器。MQPoolServices 对象也提供了查询 MQPoolTokens 集合当前大小的方法。

编译和测试 MQ base Java 程序

在编译 MQ base Java 程序之前，必须确保 MQSeries classes for Java 安装目录在 CLASSPATH 环境变量中，如第7页的『第2章 安装过程』中所描述。

要编译类 "MyClass.java"，请使用命令:

```
javac MyClass.java
```

运行 MQ base Java 小应用程序

如果编写小应用程序（java.applet.Applet 的子类），则必须在运行该类之前创建一个指向该类的 HTML 文件。样本 HTML 如下所示:

```
<html>
<body>
<applet code="MyClass.class" width=200 height=400>
</applet>
</body>
</html>
```

小应用程序是通过把 HTML 文件装入到一个支持 Java 的 Web 浏览器，或是使用 Java 开发包 (JDK) 提供的小应用程序查看器来运行的。

要使用小应用程序查看器，请输入命令:

```
appletviewer myclass.html
```

运行 MQ base Java 应用程序

如果编写使用客户机或绑定方式的应用程序（包含 main() 方法的类），则请使用 Java 解释器运行程序。使用命令:

```
java MyClass
```

注: 类名中的“.class”扩展名是省略的。

在 CICS 事务处理服务器 OS/390 版下运行 MQ base Java 应用程序

要把 Java 应用程序作为 CICS 下的事务来运行，必须:

1. 使用所提供的 CEDSA 事务将应用程序和事务定义到 CICS。
2. 确保 CICS 系统中安装了 MQSeries CICS 适配器。（详细信息，请参阅 *MQSeries for OS/390 System Management Guide*。）
3. 确保在 CICS 启动 JCL（作业控制语言）的 DHFJVM 参数中指定的 JVM 环境中包含了适当的 CLASSPATH 和 LIBPATH 项。
4. 使用任何标准进程启动事务。

有关运行的 CICS Java 事务上运行的更详细信息，请参考 CICS 系统文档。

跟踪 MQ base Java 程序

MQ base Java 包含了一个跟踪设施，当怀疑代码中有问题时，可用它来生成诊断消息。（通常，只有在 IBM 服务发出请求时才使用此设施。）

跟踪由 MQEnvironment 类的 enableTracing 和 disableTracing 方法控制。例如:

```
MQEnvironment.enableTracing(2);   // 级别 2 的跟踪
 ...                              // 将跟踪这些命令
MQEnvironment.disableTracing();   // 再次关闭跟踪
```

跟踪被写入 Java 控制台 (System.err)。

如果程序是一个应用程序，或是使用 appletviewer 命令从本地磁盘上运行它，那么还可以选择将跟踪输出重定向到所选的文件。下列代码片段显示了如何将跟踪输出重定向到名为 myapp.trc 的文件中的示例:

```
import java.io.*;

try {
  FileOutputStream
```

```
  traceFile = new FileOutputStream("myapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
catch (IOException ex) {
  // 无法打开文件,
  // 改为跟踪 System.err
  MQEnvironment.enableTracing(2);
}
```

有五种不同的跟踪级别:

1. 提供入口、出口和异常跟踪
2. 除 1 以外，还提供参数信息
3. 除 2 以外，还提供发出与收到的 MQSeries 头和数据块
4. 除 3 以外，还提供发出与收到的用户消息数据
5. 除 4 以外，还将跟踪“Java 虚拟机”中的方法

要以跟踪级别 5 来跟踪“Java 虚拟机”中的方法:

- 对于应用程序，通过发出命令 java_g（而不是 java）来运行它
- 对于小应用程序，通过发出命令 appletviewer_g （而不是 appletviewer）来运行它

注:

1. OS/390 上的 High Performance Java (HPJ) 应用程序不支持 java_g。
2. OS/400 上不支持 java_g，但若在 RUNJVA 命令中使用 OPTION(*VERBOSE) 将提供类似的功能。

第8章 取决于环境的行为

本章描述了能在各种环境下使用的 Java 类的行为。MQSeries classes for Java 中的类允许您创建可在以下环境中使用的应用程序:

1. 连接到 UNIX 或 Windows 上 MQSeries V2.x 服务器的 MQSeries 客户机 Java 版
2. 连接到 UNIX 或 Windows 上 MQSeries V5 服务器的 MQSeries 客户机 Java 版
3. 在 UNIX 或 Windows 上 MQSeries V5 服务器中执行的 MQSeries Bindings for Java
4. 在 MQSeries for MVS/ESA™ 服务器上执行的 MQSeries Bindings for Java
5. 在带有 CICS 事务处理服务器 OS/390 版版本 1.3 的 MQSeries for MVS/ESA 服务器上执行 MQSeries Bindings for Java

在所有情况下，MQSeries classes for Java 代码都使用基本 MQSeries 服务器提供的服务。只是在功能级别上有所不同（例如 MQSeries V5 提供的功能是 V2 的一个超集）。有些 API 调用和选项的行为也有一些差异。行为上的差异大多数很小，而且主要发生在 OS/390 (MQSeries for MVS/ESA) 服务器与其它平台上的服务器之间。

MQSeries classes for Java 提供了“核心”类，它在所有环境中都提供了一致的功能和行为。它还提供了“V5 扩展”，这个设计只是为在环境 2 和 3 中使用。下列部分描述了核心与扩展类。

核心类细节

MQSeries classes for Java 中包含了下列核心类，只需稍作变动就能在所有环境中使用它们，第72页的『核心类的限制与变化』中列出较小的变化。

- MQEnvironment
- MQException
- MQGetMessageOptions

  不包括:
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  不包括:
  - inquire()
  - set()
- MQMessage

  不包括:
  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices

- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  不包括:
  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  不包括:
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQC

注:

1. 某些常数不包含在核心中（详细信息，请参阅『核心类的限制与变化』），不应该在可完全移植的程序中使用它们。
2. 有些平台不支持所有的连接方式。在这些平台上，只能使用与支持方式相关的核心类与选项。（请参阅第5页的表1。）

核心类的限制与变化

虽然核心类在所有环境中正常行为都一致，但仍然存在一些较小的限制和变化，表12中记录了这些限制与变化。

除了这些文档化的变化以外，核心类在所有环境中的行为都是一致的，尽管等价的 MQSeries 类正常情况下会有环境差异）。一般而言，其行为与环境 2 和 3 中所表现的一样。

表 12. 核心类限制和变化

类或元素	限制与变化
MQGMO_LOCK MQGMO_UNLOCK MQGMO_BROWSE_MSG_UNDER_CURSOR	在环境 4 或 5 中使用时，会引发 MQRC_OPTIONS_ERROR。
MQPMO_NEW_MSG_ID MQPMO_NEW_CORREL_ID MQPMO_LOGICAL_ORDER	除环境 2 和 3 外给出错误。（请参阅 V5 扩展。）
MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MESSAGE MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE	除环境 2 和 3 外给出错误。（请参阅 V5 扩展。）
MQGMO_SYNCPOINT_IF_PERSISTENT	在环境 1 中给出错误。（请参阅 V5 扩展。）
MQGMO_MARK_SKIP_BACKOUT	除环境 4 和 5 外导致 MQRC_OPTIONS_ERROR。
MQCNO_FASTPATH_BINDING	仅受环境 3 支持。（请参阅 V5 扩展。）

表 12. 核心类限制和变化（续）

类或元素	限制与变化
MQPMRF_* fields	只在环境 2 和 3 中支持。
MQQueue.priority > MaxPriority 时放入消息	在环境 4 和 5 中被拒绝，返回 MQCC_FAILED 和 MQRC_PRIORITY_ERROR。 其它环境则接受它且返回警告 MQCC_WARNING 和 MQRC_PRIORITY_EXCEEDS_MAXIMUM，并如同用 MaxPriority 放入消息一样处理消息。
BackoutCount	环境 4 和 5 返回一个最大复原计数 255，即便消息已复原了 255 次以上。
缺省动态队列名	环境 4 和 5 是 CSQ.*。其它系统是 AMQ.*。
MQMessage.report 选项: MQRO_EXCEPTION_WITH_FULL_DATA MQRO_EXPIRATION_WITH_FULL_DATA MQRO_COA_WITH_FULL_DATA MQRO_COD_WITH_FULL_DATA MQRO_DISCARD_MSG	即使所有环境中可能都设置了报告消息，但如果报告消息是由 OS/390 队列管理器生成的，则仍不支持它们。这个问题会影响所有 Java 环境，因为 OS/390 队列管理器可能远离 Java 应用程序。如果 OS/390 队列管理器有机会参与，请避免与这些选项中的任何一项相关。
MQQueueManager.commit() 和 MQQueueManager.backout()	在环境 5 中，这些方法返回 MQRC_ENVIRONMENT_ERROR。在这种环境下，应用程序应使用 JCICS 任务同步方法: com.ibm.cics.server.Task.commit() 和 com.ibm.cics.server.Task.rollback()。
MQQueueManager 构造器	在环境 4 和 5 中，如果出现在 MQEnvironment 中的选项（以及可选特性变量）暗示一个客户机连接，则构造器将发生故障，返回 MQRC_ENVIRONMENT_ERROR。 在环境 4 和 5 中，构造器还可能返回 MQRC_CHAR_CONVERSION_ERROR。确保安装了 OS/390 Language Environment® 的“国家语言资源”组件。特别是，请确保可在 IBM-1047 和 ISO8859-1 代码页之间进行转换。 在环境 4 和 5 中，构造器还可能返回 MQRC_UCS2_CONVERSION_ERROR。 MQSeries Java 类尝试将 Unicode 转换成队列管理器代码页，如果指定的代码页不可用，则使用缺省值 IBM-500。确保有用于 Unicode 的适当的转换表，这个转换表应该是作为 OS/390 C/C++ 可选功能安装的，此外还应当确保 Language Environment 可以找到这些表。有关启用 UCS-2 转换的更详细信息，请参阅 *OS/390 C/C++ Programming Guide*，SC09-2362。

运行于其它环境下的版本 5 扩展

MQSeries classes for Java 包含下列功能，它们是为使用 MQSeries V5 中引入的 API 扩展而专门设计的。这些功能只能在环境 2 和 3 中运作。本主题描述了它们在其它环境中的行为。

MQQueueManager 构造器选项

MQQueueManager 构造器包含了一个可选整数变量。它映射到 MQI 的 MQCNO.options 字段，用于切换正常连接与快速路径连接。构造器的这个扩展形式在所有环境中都可接受，假设可供使用的仅有选项是 MQCNO_STANDARD_BINDING 或 MQCNO_FASTPATH_BINDING。任何其它选项会导致构造器以 MQRC_OPTIONS_ERROR 失败。快速路径选项 MQC.MQCNO_FASTPATH_BINDING 只有在 MQSeries V5 绑定（环境 3）中使用时才能实现。若把该选项用于其它任何环境，则会被忽略。

MQQueueManager.begin() 方法

只有在环境 3 中才能使用。在其它任何环境中使用，将发生故障，原因 MQRC_ENVIRONMENT_ERROR。MQSeries AS/400 版不支持使用 `begin()` 方法来启动由队列管理器协调的全局工作单元。

MQPutMessageOptions 选项

下列标志可以设置到任何环境中的 MQPutMessageOptions 选项字段中。但是，如果这些标志在除 2 和 3 外的任何环境下与随后的 MQQueue.put() 一起使用，则 put() 将失败，原因 MQRC_OPTIONS_ERROR。

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

MQGetMessageOptions 选项

下列标志可以设置到任何环境中的 MQGetMessageOptions 选项字段中。但是，如果这些标志在除 2 和 3 外的任何环境下与随后的 MQQueue.get() 一起使用，则 get() 将失败，原因 MQRC_OPTIONS_ERROR。

- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

下列标志可以设置到任何环境中的 MQGetMessageOptions 选项字段中。但是如果在环境 1 中使用随后的一个 MQQueue.get() 一起使用，则 get() 将失败，原因 MQRC_OPTIONS_ERROR。

- MQGMO_SYNCPOINT_IF_PERSISTENT

MQGetMessageOptions 字段

值可能被设置到下列字段中，而无需考虑环境因素。但是，如果是在除 2 或 3 外的任何环境下运行时，在使用后续 MQQueue.get() 上使用的 MQGetMessageOptions 包含非缺省值，则 get() 失败，原因 MQRC_GMO_ERROR。这表示在除 2 和 3 之外的环境中，每次 get() 成功后，这些字段总是被设置成它们的初始值。

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

分发列表

下列类用于创建分发列表:

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

您可以在任何环境中创建并驻留 MQDistributionList 和 MQDistributionListItems，但是只有在环境 2 和 3 中，才能成功地创建并打开 MQDistributionList。在其它任何环境下尝试创建和打开都将遭到拒绝，原因 MQRC_OD_ERROR。

MQPutMessageOptions 字段

在 MQPutMessageOptions 类中，MQPMO 中的四个字段反映为下列成员变量:

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

虽然其本意是用于分发列表，但 MQSeries V5 服务器在 MQPUT 到达一个队列后还是填充了 DestCount 字段。例如，如果队列解析成一个本地队列，则 knownDestCount 设置为 1，其它两个字段设置为 0。在环境 2 和 3 中，由 V5 服务器设置的值将在 MQPutMessageOptions 类中返回。在其它环境中，返回的值与以下值相仿:

- 若 put() 成功，则 unknownDestCount 设置为 1，其它设置为 0。
- 若 put() 失败，则 invalidDestCount 设置为 1，其它设置为 0。

recordFields 与分发列表一起使用。无论什么环境中，值都可以在任何时侯写入 recordFields。但是若在随后的 MQQueue.put() 上使用了 MQPutMessage 选项，而不是在 MQDistributionList.put() 上使用它，则被忽略。

MQMD 字段

下列 MQMD 字段与消息分段紧密相关:

- GroupId
- MsgSeqNumber
- Offset MsgFlags
- OriginalLength

若应用程序将这些 MQMD 字段中的任何一个设置为非缺省值，则在 2 或 3 之外的环境中执行 put() 或 get() 将产生一个异常 (MQRC_MD_ERROR)。在 2 或 3 之外的环境中成功执行了 put() 或 get() 后，新的 MQMD 字段总是被设置成其缺省值。成组或分段的消息通常不应发送到一个运行非 MQSeries 版本 5 或更高版本队列管理器的 Java 应用程序。若这样的应用程序确实发出了一个

get，而且要检索的物理消息恰巧是成组或分段消息中的一部分（对于 MQMD 字段有非缺省值），则对它的检索不会出错。但是，不更新 MQMessage 中的 MQMD。 MQMessage 格式特性被设置为 MQFMT_MD_EXTENSION，并且真实消息数据的前面放有一个包含了新字段值的 MQMDE 结构。

第9章 MQ base Java 类和接口

本章描述所有的 MQSeries classes for Java 类和接口。它包含每个类和接口中变量、构造器和方法的细节。

描述了以下类:

- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

描述了以下接口:

- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData

© Copyright IBM Corp. 1997, 2001

MQChannelDefinition

```
java.lang.Object
   │
   └─ com.ibm.mq.MQChannelDefinition
```

public class **MQChannelDefinition**
extends **Object**

MQChannelDefinition 类用于将有关至队列管理器连接的消息传送给发送、接收和安全性出口。

注: 当以绑定方式直接连接到 MQSeries 时，该类不适用。

变量

channelName

```
public String channelName
```

通过其建立连接的通道名称。

queueManagerName

```
public String queueManagerName
```

进行连接的队列管理器的名称。

maxMessageLength

```
public int maxMessageLength
```

能被发送给队列管理器的最大消息长度。

securityUserData

```
public String securityUserData
```

安全性出口使用的存储区。这里放置的消息是在遇到安全性出口调用时保留的，同时适用于发送和接收出口。

sendUserData

```
public String sendUserData
```

发送出口使用的存储区。这里放置的消息是在遇到发送出口调用时保留的，同时适用于安全性和接收出口。

receiveUserData

```
public String receiveUserData
```

接收出口使用的存储区。这里放置的消息是在遇到接收出口调用时保留的，同时适用于发送和安全性出口。

connectionName

```
public String connectionName
```

队列管理器驻留的机器的 TCP/IP 主机名。

remoteUserId

```
public String remoteUserId
```

用于建立连接的用户标识。

remotePassword

```
public String remotePassword
```

用于建立连接的口令。

构造器

MQChannelDefinition

```
public MQChannelDefinition()
```

MQChannelExit

```
java.lang.Object
   │
   └─ com.ibm.mq.MQChannelExit
```

public class **MQChannelExit**
extends **Object**

该类定义了当调用上下文消息时传送到发送、接收和安全性出口的上下文消息。应该由出口设置 exitResponse 成员变量，以表明 MQSeries 客户机 Java 版下一步应采取什么操作。

注: 当以绑定方式直接连接到 MQSeries 时，该类不适用。

变量

MQXT_CHANNEL_SEC_EXIT
```
public final static int MQXT_CHANNEL_SEC_EXIT
```

MQXT_CHANNEL_SEND_EXIT
```
public final static int MQXT_CHANNEL_SEND_EXIT
```

MQXT_CHANNEL_RCV_EXIT
```
public final static int MQXT_CHANNEL_RCV_EXIT
```

MQXR_INIT
```
public final static int MQXR_INIT
```

MQXR_TERM
```
public final static int MQXR_TERM
```

MQXR_XMIT
```
public final static int MQXR_XMIT
```

MQXR_SEC_MSG
```
public final static int MQXR_SEC_MSG
```

MQXR_INIT_SEC
```
public final static int MQXR_INIT_SEC
```

MQXCC_OK
```
public final static int MQXCC_OK
```

MQXCC_SUPPRESS_FUNCTION
```
public final static int MQXCC_SUPPRESS_FUNCTION
```

MQXCC_SEND_AND_REQUEST_SEC_MSG
```
public final static int MQXCC_SEND_AND_REQUEST_SEC_MSG
```

MQXCC_SEND_SEC_MSG
```
public final static int MQXCC_SEND_SEC_MSG
```

MQXCC_SUPPRESS_EXIT
```
public final static int MQXCC_SUPPRESS_EXIT
```

MQXCC_CLOSE_CHANNEL
```
public final static int MQXCC_CLOSE_CHANNEL
```

exitID `public int exitID`

被调用的出口类型。对于 MQSecurityExit 总是 MQXT_CHANNEL_SEC_EXIT。对于 MQSendExit 总是 MQXT_CHANNEL_SEND_EXIT，对于 MQReceiveExit 总是 MQXT_CHANNEL_RCV_EXIT。

exitReason

`public int exitReason`

调用该出口的原因。可能的值有:

MQXR_INIT
出口初始化；在已商议通道连接条件后，但在发送任何安全性流前已被调用。

MQXR_TERM
出口终止；在发送断开流后，套接字连接被摧毁前被调用。

MQXR_XMIT
对于发送出口，表明该数据将被发送到队列管理器。

对于接收出口，表明已从队列管理器接收到该数据。

MQXR_SEC_MSG
对安全性出口表明已从队列管理器接收到安全性消息。

MQXR_INIT_SEC
表明出口是启动队列管理器的安全性对话框。

exitResponse

`public int exitResponse`

由出口设置，表明 MQSeries classes for Java 下一步应该采取的操作。有效值有:

MQXCC_OK
由安全性出口设置，表明安全性交换是完整的。

由发送出口设置，表明被返回的数据要被发送给队列管理器。

由接收出口设置，表明可以由 MQSeries 客户机 Java 版来处理返回的数据。

MQXCC_SUPPRESS_FUNCTION
由安全性出口设置，表明应该关闭与队列管理器的通信。

MQXCC_SEND_AND_REQUEST_SEC_MSG
由安全性出口设置，表明被返回的数据将被发送给队列管理器，且期望来自队列管理器的响应。

MQXCC_SEND_SEC_MSG
由安全性出口设置，表明被返回的数据要被发送给队列管理器，且不期望响应。

MQXCC_SUPPRESS_EXIT
由任何出口设置，表明不再被调用。

MQXCC_CLOSE_CHANNEL
由任何出口设置，表明应该关闭至队列管理器的连接。

maxSegmentLength

```
public int maxSegmentLength
```

至队列管理器的任何一个传输的最大长度。

如果出口返回要被发送到队列管理器的数据，则被返回数据的长度不应该超过该值。

exitUserArea

```
public byte exitUserArea[]
```

出口可使用的存储区。

MQSeries 客户机 Java 版在使用相同 exitID 遇到出口调用时保留 exitUserArea 中放置的任何数据。（即发送、接收和安全性出口都拥有各自独立的用户区。）

capabilityFlags

```
public static final int capabilityFlags
```

表明队列管理器的能力。

仅支持 MQC.MQCF_DIST_LISTS 标志。

fapLevel

```
public static final int fapLevel
```

协商的格式和协议 (FAP) 级别。

构造器

MQChannelExit

```
public MQChannelExit()
```

MQDistributionList

```
java.lang.Object
   │
   └─ com.ibm.mq.MQManagedObject          │
         └─ com.ibm.mq.MQDistributionList
```

public class **MQDistributionList**
extends **MQManagedObject**（请参阅 97 页。）

注: 只有在连接到 MQSeries 版本 5（或更高版）队列管理器时才能使用该类。

MQDistributionList 是使用 MQDistributionList 构造器创建的，或是使用 MQQueueManager 的 accessDistributionList 方法创建的。

分发列表表示一组打开的队列，可以使用对 put() 方法的单一调用将消息发送到这些队列。（见 *MQSeries 应用程序设计指南*中的分发列表。）

构造器

MQDistributionList

```
public MQDistributionList(MQQueueManager qMgr,
              MQDistributionListItem[] litems,
              int openOptions,
              String alternateUserId)
                     throws MQException
```

qMgr 是队列管理器，列表将在其中打开。

litems 是分发列表中将包含的项。

有关保留参数的详细信息，请参阅第146页的『accessDistributionList』。

方法

put

```
public synchronized void put(MQMessage message,
                        MQPutMessageOptions putMessageOptions)
                       throws MQException
```

将消息放入分发列表上的队列。

参数

message
包含消息描述符信息和返回的消息数据的输入／输出参数。

putMessageOptions
控制 MQPUT 操作的选项。（详细信息，请参阅第127页的『MQPutMessageOptions』。）

如果放入操作失败，则掷出 MQException。

getFirstDistributionListItem

```
public MQDistributionListItem getFirstDistributionListItem()
```

返回分发列表中的第一项，或当列表为空时返回空。

getValidDestinationCount

```
public int getValidDestinationCount()
```

返回分发列表中成功打开的项数。

getInvalidDestinationCount

```
public int getInvalidDestinationCount()
```

返回分发列表中打开失败的项数。

MQDistributionListItem

```
java.lang.Object
   │
   └─ com.ibm.mq.MQMessageTracker          │
         └─ com.ibm.mq.MQDistributionListItem
```

public class **MQDistributionListItem**
extends **MQMessageTracker**（参见 119 页。）

注: 只有当连接到 MQSeries 版本 5（或更高版）队列管理器时才可以使用这个类。

MQDistributionListItem 表示分发列表内单一项（队列）。

变量

completionCode

```
public int completionCode
```

从该项上次操作得出的完成码。如果这是 MQDistributionList 的构造，则完成码与队列的打开相关。如果它是放入操作，则完成码与将消息放入该队列的尝试相关。

初始值是 ″0″。

queueName

```
public String queueName
```

想与分发列表一起使用的队列名称。这不能是模型队列的名称。

初始值是 ″″。

queueManagerName

```
public String queueManagerName
```

其上定义队列的队列管理器的名称。

初始值是 ″″。

reasonCode

```
public int reasonCode
```

该项上次操作得出的原因码。如果这是 MQDistributionList 的构造，则原因码与队列的打开相关。如果它是放入操作，则原因码与将消息放入该队列的尝试相关。

初始值是 ″0″。

构造器

MQDistributionListItem

```
public MQDistributionListItem()
```

构造一个新的 MQDistributionListItem 对象。

MQEnvironment

```
java.lang.Object
   │
   └─ com.ibm.mq.MQEnvironment
```

public class **MQEnvironment**
extends **Object**

注: 该类的所有方法和属性都适用于 MQSeries classes for Java 客户机连接，但是适用于绑定连接的只有 enableTracing、disableTracing、properties 和 version_notice。

MQEnvironment 包含控制构建 MQQueueManager 对象（以及至 MQSeries 的相应连接）的环境的静态成员变量。

当调用 MQQueueManager 构造器时，在 MQEnvironment 类中设置的值生效，所以应该在构造 MQQueueManager 实例前在 MQEnvironment 类中设置值。

变量

注: 当以绑定方式直接连接到 MQSeries 时，以 * 标记的变量不适用。

version_notice

```
public final static String version_notice
```

MQSeries classes for Java 的当前版本。

securityExit*

```
public static MQSecurityExit securityExit
```

安全性出口允许您定制试图连接到队列管理器时发生的安全性流。

要提供您自己的安全性出口，请定义实现 MQSecurityExit 接口的类，并将 securityExit 赋给该类的一个实例。否则，可以将 securityExit 设置为空，在这种情况下将不会调用任何安全性出口。

另见第155页的『MQSecurityExit』。

sendExit*

```
public static MQSendExit sendExit
```

发送出口允许您对发送至队列管理器的数据进行检查和必要的改变。它通常与队列管理器上的相应接收出口一起使用。

要提供您自己的发送出口，请定义实现 MQSendExit 接口的类，并将 sendExit 赋给该类的一个实例。否则，可以将 sendExit 设置为空，在这种情况下将不会调用任何发送出口。

另见第157页的『MQSendExit』。

receiveExit*

```
public static MQReceiveExit receiveExit
```

接收出口允许您检查从队列管理器接收到的数据并进行必要的改变。它通常与队列管理器的相应发送出口一起使用。

要提供您自己的接收出口，请定义实现 MQReceiveExit 接口的类，并将 receiveExit 赋值给该类的一个实例。否则，可以将 receiveExit 设置为空，在这种情况下将不会调用任何接收出口。

另见第153页的『MQReceiveExit』。

hostname*

```
public static String hostname
```

MQSeries 服务器所在机器的 TCP/IP 主机名。如果没有设置主机名，且没有设置覆盖属性，则将使用绑定方式连接到本地队列。

port*

```
public static int port
```

要连接至的端口。这是 MQSeries 服务器在其上侦听进入连接请求的端口。缺省值是 1414。

channel*

```
public static String channel
```

目标队列管理器上要连接至的通道名称。在构造以客户机方式使用 MQQueueManager 实例前，*必须*设置这个成员变量或相应的特性。

userID*

```
public static String userID
```

等价于 MQSeries 环境变量 MQ_USER_ID。

如果没有为该客户机定义安全性出口，用户标识值将被发送到服务器，且当调用它时对服务器安全性是可用的。可以使用该值来验证 MQSeries 客户机的身份。

缺省值是 ″″。

password*

```
public static String password
```

等价于 MQSeries 环境变量 MQ_PASSWORD。

如果没有为该客户机定义安全性出口，口令值将被发送到服务器，且当调用它时对服务器安全性是可用的。可以使用该值来验证 MQSeries 客户机的身份。

缺省值是 ″″。

properties

```
public static java.util.Hashtable properties
```

定义 MQSeries 环境的键／值对集。

该散列表允许您将环境属性设置为键／值对，而不是个别变量。

该特性也可作为 MQQueueManager 构造器上参数中的散列表来传送。在构造器上传送的特性比使用该特性变量的值设置的值有更高优先权，否则它们是可交换的。查找特性的优先次序是:

1. MQQueueManager 构造器上的特性参数
2. MQEnvironment.properties
3. 其它 MQEnvironment 变量
4. 常数缺省值

下列表中显示了可能的键／值对:

键	值
MQC.CCSID_PROPERTY	整数（覆盖 MQEnvironment.CCSID）
MQC.CHANNEL_PROPERTY	字符串（覆盖 MQEnvironment.channel）
MQC.CONNECT_OPTIONS_PROPERTY	整数，缺省值是 MQC.MQCNO_NONE
MQC.HOST_NAME_PROPERTY	字符串（覆盖 MQEnvironment.hostname）
MQC.ORB_PROPERTY	org.omg.CORBA.ORB（可选的)
MQC.PASSWORD_PROPERTY	字符串（覆盖 MQEnvironment.password）
MQC.PORT_PROPERTY	整数（覆盖 MQEnvironment.port）
MQC.RECEIVE_EXIT_PROPERTY	MQReceiveExit（覆盖 MQEnvironment.receiveExit）
MQC.SECURITY_EXIT_PROPERTY	MQSecurityExit（覆盖 MQEnvironment.securityExit）
MQC.SEND_EXIT_PROPERTY	MQSendExit（覆盖 MQEnvironment.sendExit）
MQC.TRANSPORT_PROPERTY	MQC.TRANSPORT_MQSERIES_BINDINGS 或 MQC.TRANSPORT_MQSERIES_CLIENT 或 MQC.TRANSPORT_VISIBROKER 或 MQC.TRANSPORT_MQSERIES（缺省值，将根据“hostname”的值选择绑定或客户机。）
MQC.USER_ID_PROPERTY	字符串（覆盖 MQEnvironment.userID。）

CCSID*

```
public static int CCSID
```

客户机使用的 CCSID。

更改该值会影响队列管理器转换 MQSeries 头中消息的方式。除了 MQMessage 类的 applicationIdData 和 putApplicationName 字段中的数据外， MQSeries 头中的所有数据都是从 ASCII 代码集的不变部分取出的。（请参阅第100页的『MQMessage』。）

如果对这两个字段避免使用来自 ASCII 代码集变化部分的字符，就能安全地将 CCSID 从 819 更改到其它任何 ASCII 代码集。

如果将客户机的 CCSID 更改为与正连接至的队列管理器的 CCSID 相同，则将在队列管理器上获得性能好处，因为它不必尝试转换消息头。

缺省值是 819。

构造器

MQEnvironment

```
public MQEnvironment()
```

方法

disableTracing

```
public static void disableTracing()
```

关闭 MQSeries 客户机 Java 版跟踪设施。

enableTracing

```
public static void enableTracing(int level)
```

打开 MQSeries 客户机 Java 版跟踪设施。

参数

level 所需跟踪的级别，从 1 至 5（5 是最详细的）

enableTracing

```
public static void enableTracing(int level,
                                 OutputStream stream)
```

打开 MQSeries 客户机 Java 版跟踪设施。

参数:

level 所需跟踪的级别，从 1 至 5（5 是最详细的）

stream 要写入跟踪的流。

setDefaultConnectionManager

```
public static void setDefaultConnectionManager(MQConnectionManager cxManager)
```

将提供的 MQConnectionManager 设置为缺省的 ConnectionManager。当 MQQueueManager 构造器上没有指定 ConnectionManager 时，将使用缺省的 ConnectionManager。这个方法还将使 MQPoolToken 的集合清空。

参数:

cxManager
将作为缺省 ConnectionManager 的 MQConnectionManager。

setDefaultConnectionManager

```
public static void setDefaultConnectionManager
                        (javax.resource.spi.ConnectionManager cxManager)
```

设置缺省 ConnectionManager，并将 MQPoolToken 的集合清空。当 MQQueueManager 构造器上没有指定 ConnectionManager 时，将使用缺省的 ConnectionManager。

这个方法要求装有 JAAS 1.0 或更新版本的 Java 2 v1.3 或更新版本的 JVM。

参数:

cxManager
: 缺省 ConnectionManager（实现了 javax.resource.spi.ConnectionManager 接口）。

getDefaultConnectionManager

```
public static javax.resource.spi.ConnectionManager
                                    getDefaultConnectionManager()
```

返回缺省 ConnectionManager。如果缺省 ConnectionManager 就是 MQConnectionManager，则返回空。

addConnectionPoolToken

```
public static void addConnectionPoolToken(MQPoolToken token)
```

将所提供的 MQPoolToken 添加到令牌集。缺省 ConnectionManager 可以使用它作为提示；通常，只有当令牌集中至少有一个令牌时才启用它。

参数:

token 要添加到令牌集的 MQPoolToken。

addConnectionPoolToken

```
public static MQPoolToken addConnectionPoolToken()
```

构造一个 MQPoolToken 并把它添加到令牌集中。MQPoolToken 被返回到应用程序，随后将被传递到 removeConnectionPoolToken()。

removeConnectionPoolToken

```
public static void removeConnectionPoolToken(MQPoolToken token)
```

从令牌集中删除指定的 MQPoolToken。如果 MQPoolToken 不在令牌集中，则无操作。

参数:

token 要从令牌集中删除的 MQPoolToken。

MQException

```
java.lang.Object
   |
   └─ java.lang.Throwable
         |
         └─ java.lang.Exception
               |
               └─ com.ibm.mq.MQException
```

public class **MQException**
extends **Exception**

无论何时发生 MQSeries 错误时，都掷出 MQException。可以通过设置 MQException.log 值来更改记录的异常情况的输出流。缺省值是 System.err。这个类包含完成码和错误码常量的定义。以 MQCC_ 开始的常数是 MQSeries 完成码，以 MQRC_ 开始的常数是 MQSeries 原因码。*MQSeries Application Programming Reference* 中完整地说明了这些错误以及引发这些错误可能的原因。

变量

log `public static java.io.outputStreamWriter log`

记录异常的流。(缺省值是 System.err。) 如果将这设置为空，则不进行记录。

completionCode
`public int completionCode`

给出产生错误的 MQSeries 完成码。可能的值是:
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

reasonCode
`public int reasonCode`

描述错误的 MQSeries 原因码。原因码的完整说明，请参阅 *MQSeries Application Programming Reference*。

exceptionSource
`public Object exceptionSource`

产生异常的对象实例。当确定错误原因时，可以将它用作诊断的一部分。

构造器

MQException

```
public MQException(int completionCode,
                   int reasonCode,
                   Object source)
```

构造一个新的 MQException 对象。

参数

completionCode
MQSeries 完成码。

reasonCode
MQSeries 原因码。

source 发生错误的对象。

MQGetMessageOptions

```
java.lang.Object
   |
   └─ com.ibm.mq.MQGetMessageOptions
```

public class **MQGetMessageOptions**
extends **Object**

该类包含控制 MQQueue.get() 行为的选项。

注: 该类中可用的一些选项的行为取决于使用它们的环境。这些元素用 * 标记。有关详细信息，请参阅第71页的『第8章 取决于环境的行为』。

变量

options

```
public int options
```

控制 MQQueue.get 操作的选项。可以指定下列任何值，或不指定下列值。如果需要多个选项，可以使用按位的“或”运算符添加或组合这些值。

MQC.MQGMO_NONE

MQC.MQGMO_WAIT
等待消息到来。

MQC.MQGMO_NO_WAIT
如果没有适用的消息，则立即返回。

MQC.MQGMO_SYNCPOINT
获取同步点控制下的消息；标记消息为对其它应用程序不可用，仅当提交工作单元时才将它从队列中删除。如果工作单元被逆序恢复，那么消息将再次可用。

MQC.MQGMO_NO_SYNCPOINT
获取无同步点控制的消息。

MQC.MQGMO_BROWSE_FIRST
从队列开始处浏览。

MQC.MQGMO_BROWSE_NEXT
从队列当前位置浏览。

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
从浏览游标处浏览消息。

MQC.MQGMO_MSG_UNDER_CURSOR
获取浏览游标处的消息。

MQC.MQGMO_LOCK*
锁定被浏览的消息。

MQC.MQGMO_UNLOCK*
解锁以前锁定的消息。

MQC.MQGMO_ACCEPT_TRUNCATED_MSG
允许截断消息数据。

MQC.MQGMO_FAIL_IF_QUIESCING
如果队列管理器停顿，则失败。

MQC.MQGMO_CONVERT
在将数据复制到消息缓冲区前，请求把应用程序数据转换为与 MQMessage 的 characterSet 和 encoding 属性一致。因为数据转换也适用于从消息缓冲区检索出的数据，所以应用程序通常不设置这个选项。

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
如果消息一直存在，则使用同步点控制获取消息。

MQC.MQGMO_MARK_SKIP_BACKOUT*
允许工作单元不恢复队列上的消息而逆序恢复。

分段与分组 MQSeries 消息能作为单一实体被发送或接收，能为发送和接收分割成几个段，也能与组中的其它消息链接。

发送的每段数据都称为一个*物理*消息，它可以是完整的*逻辑*消息，或一个较长的逻辑消息中的一段。

每个物理消息通常有不同的 `MsgId`。单一逻辑消息的所有段都有相同的 `groupId` 值和 `MsgSeqNumber` 值，但是每个段的 `Offset` 值是不同的。`Offset` 字段给出了从逻辑消息的开始处到物理消息中数据的偏移量。这些段通常有不同的 `MsgId` 值，因为它们是独立的物理消息。

组成组的一部分的逻辑消息具有相同的 `groupId` 值，但是组中的每条消息都具有不同的 `MsgSe qNumber` 值。组中的消息也可被分段。

下列选项可用于处理被分段或分组的消息。

MQC.MQGMO_LOGICAL_ORDER*
以逻辑次序返回组和逻辑消息段的消息。

MQC.MQGMO_COMPLETE_MSG*
仅检索完整的逻辑消息。

MQC.MQGMO_ALL_MSGS_AVAILABLE*
仅当组中所有消息都可用时才从组检索消息。

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
仅当组中所有段都可用时才检索逻辑消息的段。

waitInterval

```
public int waitInterval
```

MQQueue.get 调用等待适用的消息到来的最大时间（以毫秒）（与 MQC.MQGMO_WAIT 一起使用）。值 MQC.MQWI_UNLIMITED 表明需要一个不受限制的等待。

resolvedQueueName

```
public String resolvedQueueName
```

这是队列管理器设置的检索出消息的队列本地名的输出字段。如果打开别名队列或模型队列，这将与用来打开队列的名称不同。

matchOptions*

```
public int matchOptions
```

确定检索什么消息的选择标准。可以设置下列匹配选项:

MQC.MQMO_MATCH_MSG_ID
要匹配的消息标识。

MQC.MQMO_MATCH_CORREL_ID
要匹配的关联标识。

MQC.MQMO_MATCH_GROUP_ID
要匹配的组标识。

MQC.MQMO_MATCH_MSG_SEQ_NUMBER
匹配消息序号。

MQC.MQMO_NONE
不需要匹配。

groupStatus*

```
public char groupStatus
```

这是一个输出字段，表明组中是否有检索消息，如果有的话是否是组中的最后一个消息。可能的值有:

MQC.MQGS_NOT_IN_GROUP
消息不在组中。

MQC.MQGS_MSG_IN_GROUP
消息在组中，但不是组中的最后一个。

MQC.MQGS_LAST_MSG_IN_GROUP
消息是组中的最后一个。这也是当组仅由一个消息组成时返回的值。

segmentStatus*

```
public char segmentStatus
```

这是表明检索消息是否是逻辑消息段的输出字段。如果消息是段，标志将表明它是否是最后一个段。可能的值有:

MQC.MQSS_NOT_A_SEGMENT
消息不是段。

MQC.MQSS_SEGMENT
消息是段，但不是逻辑消息的最后一个段。

MQC.MQSS_LAST_SEGMENT
消息是逻辑消息的最后一个段。这也是当逻辑消息仅由一个段组成时返回的值。

segmentation*

```
public char segmentation
```

这是表明检索的消息（它是逻辑消息的一个分段）是否允许分段的输出字段。可能的值有:

MQC.MQSEG_INHIBITED
不允许分段。

MQC.MQSEG_ALLOWED
允许分段。

构造器

MQGetMessageOptions

```
public MQGetMessageOptions()
```

使用选项设置成 MQC.MQGMO_NO_WAIT、等待间隔 0 和空白组成的队列名称来构造新的 MQGetMessageOptions 对象。

MQManagedObject

```
java.lang.Object
   │
   └─ com.ibm.mq.MQManagedObject
```

public class **MQManagedObject**
extends **Object**

MQManagedObject 是 MQQueueManager、MQQueue 和 MQProcess 的超类。它提供查询和设置这些资源属性的能力。

变量

alternateUserId

```
public String alternateUserId
```

打开资源时指定的替代用户标识（如果有的话）。设置这个属性没有作用。

name

```
public String name
```

这个资源的名称（访问方式提供的名称，或队列管理器为动态队列分配的名称）。设置这个属性没有作用。

openOptions

```
public int openOptions
```

当打开该资源时指定的选项。设置这个属性没有作用。

isOpen

```
public boolean isOpen
```

表明该资源是否当前已打开。这个属性是*不建议的*并且设置它没有作用。

connectionReference

```
public  MQQueueManager connectionReference
```

该资源属于的队列管理器。设置这个属性没有作用。

closeOptions

```
public int closeOptions
```

设置该属性以控制关闭资源的方法。缺省值是 MQC.MQCO_NONE，而且这是除永久动态队列外所有资源的唯一可允许的值，并且创建它们的对象将访问临时动态队列。对于这些队列，下列附加的值是可允许的:

MQC.MQCO_DELETE
如果没有消息，则删除队列。

MQC.MQCO_DELETE_PURGE
删除队列，清除其上的任何消息。

构造器

MQManagedObject

```
protected MQManagedObject()
```

构造器方法。

方法

getDescription

```
public String getDescription()
```

掷出 MQException。

返回队列管理器上保留的该资源的描述。

如果在关闭资源后调用此方法，将掷出 MQException。

inquire

```
public void inquire(int selectors[],
                    int intAttrs[],
                    byte charAttrs[])
```

掷出 MQException。

返回整数数组和包含对象（队列、进程或队列管理器）属性的字符串组。

要查询的属性在选择器数组中指定。有关可允许的选择器以及它们相应的整数值，请参考 *MQSeries Application Programming Reference*。

注意，可以使用 MQManagedObject、MQQueue、MQQueueManager 和 MQProcess 中定义的 getXXX() 方法来查询更多的公共属性。

参数

selectors
: 标识要查询值的属性的整数数组。

intAttrs
: 整数属性值返回到的数组。整数属性值以与选择器数组中整数属性选择器相同的次序返回。

charAttrs
: 字符属性返回到的缓冲区，字符属性在其中是并置的。字符属性以与选择器数组中字符属性选择器相同的次序返回。对于每个属性，每个属性字符串的长度是固定的。

如果查询失败，则掷出 MQException。

isOpen

```
public boolean isOpen()
```

返回 isOpen 变量的值。

set

```
public synchronized void set(int selectors[],
                             int intAttrs[],
                             byte charAttrs[])
```

掷出 MQException。

设置选择器向量中定义的属性。

要设置的属性在选择器数组中指定。有关可允许的选择器以及它们相应的整数值，请参考 *MQSeries Application Programming Reference*。

注意可以使用 MQQueue 中定义的 setXXX() 方法设置某些队列属性。

参数

selectors
标识要设置值的属性的整数数组。

intAttrs 要设置的整数属性值数组。这些值必须与选择器数组中整数属性选择器的次序相同。

charAttrs
要在其中设置字符属性的缓冲区是并置的。这些值必须与选择器数组中字符属性选择器的次序相同。每个字符属性的长度都是固定的。

如果设置失败，掷出 MQException。

close

```
public synchronized void close()
```

掷出 MQException。

关闭该对象。在调用这个方法后，不允许对这个资源再进行任何进一步操作。可以通过设置 closeOptions 属性改变关闭方法的操作。

如果 MQSeries 调用失败，则掷出 MQException。

MQMessage

```
java.lang.Object
   │
   └─ com.ibm.mq.MQMessage
```

public class **MQMessage**
implements **DataInput**, **DataOutput**

MQMessage 表示了 MQSeries 消息的消息描述符和数据。有一组从消息读数据的 readXXX 方法，以及一组将数据写入消息的 writeXXX 方法。这些读和写的方法使用的数字和字符串格式可以由编码和字符集成员变量控制。剩余的成员变量包含了当消息在发送和接收应用程序之间传递时，伴随应用程序数据的控制信息。应用程序能在将消息放入队列前在成员变量中设置值，并在从队列检索消息后读取值。

变量

report `public int report`

报告是关于另一个消息的消息。该成员变量使应用程序能发送原始消息，以指定需要哪些报告消息、应用程序消息数据是否要包含在其中以及如何设置报告或应答中的消息和相关标识。可以请求以下任一、全部报告类型，或都不请求:

- 异常
- 失效
- 到达时确认
- 发送时确认

对于每种类型，根据报告消息中是否要包含应用程序消息数据，指定下列三种相应值中的一种。

注: MVS 队列管理器不支持下列列表中使用 ** 标记的值，所以如果应用程序想访问 MVS 管理器，不管应用程序正运行在什么平台上，则不应该使用这些值。

有效值有:

- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**
- MQC.MQRO_COA
- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

可以指定下列值之一来控制如何为报告或应答消息生成消息标识。

- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

可以指定下列值之一来控制如何设置报告或应答消息的关联标识。

- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

可以指定下列值之一，以便在原始消息无法传递到目标队列时来控制它们的配置:

- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **

如果没有指定报告选项，则缺省值是:

```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

可以指定下列值之一，或同时指定两者，来请求接收应用程序发送一个正向操作或逆向操作报告消息:

- MQRO_PAN
- MQRO_NAN

messageType

```
public int messageType
```

指定消息类型。下列值是系统当前定义的:

- MQC.MQMT_DATAGRAM
- MQC.MQMT_REQUEST
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT

同样也可使用应用程序定义的数值。这些应该在 MQC.MQMT_APPL_FIRST 至 MQC.MQMT_APPL_LAST 范围之中。

该字段的缺省值是 MQC.MQMT_DATAGRAM。

expiry

```
public int expiry
```

由放入该消息的应用程序设置的失效时间，以十分之一秒计算。在到达消息失效时间后，它就可以被队列管理器废弃。如果消息指定了一个 MQC.MQRO_EXPIRATION 标志，则当废弃消息时会生成一个报告。

缺省值是 MQC.MQEI_UNLIMITED，表示消息永不会失效。

feedback

```
public int feedback
```

这个值与 MQC.MQMT_REPORT 消息类型一起表明报告的性质。下列反馈码是由系统定义的:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW

- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

也可使用 MQC.MQFB_APPL_FIRST 至 MQC.MQFB_APPL_LAST 范围中应用程序定义的反馈值。

该字段的缺省值是 MQC.MQFB_NONE，表明没有提供任何反馈。

encoding

```
public int encoding
```

该成员变量指定应用程序消息数据中数值的表示法；它适用于二进制、压缩十进制和浮点数据。这些数值格式的读和写操作会相应发生改变。

下列编码是为二进制整数定义的:

MQC.MQENC_INTEGER_NORMAL
如 Java 中的大尾数法整数

MQC.MQENC_INTEGER_REVERSED
如 PC 使用的小尾数法整数。

下列编码是为压缩十进制整数定义的:

MQC.MQENC_DECIMAL_NORMAL
如 System/390 使用的大尾数法压缩十进制数。

MQC.MQENC_DECIMAL_REVERSED
小尾数法压缩十进制

下列编码是为浮点数定义的:

MQC.MQENC_FLOAT_IEEE_NORMAL
如 Java 中的大尾数法 IEEE 浮点数。

MQC.MQENC_FLOAT_IEEE_REVERSED
如 PC 使用的小尾数法浮点数。

MQC.MQENC_FLOAT_S390
System/390 格式浮点数。

encoding 字段值应该通过添加分别来自这三个部分（或使用按位“或”运算符）的值来构造。缺省值是:

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

为了方便起见，也可由 MQC.MQENC_NATIVE 来表示该值。这种设置使 writeInt() 写一个大尾数法整数，使 readInt() 读一个大尾数法整数。如果设置了标志 MQC.MQENC_INTEGER_REVERSED 作为替换，则 writeInt() 将写一个小尾数法整数，而 readInt() 将读一个小尾数法整数。

注意当从 IEEE 格式浮点转换至 System/390 格式浮点时，会发生精度损失。

characterSet

```
public int characterSet
```

它指定了应用程序消息数据中字符数据的编码字符集标识。readString、readLine 和 writeString 方法的操作会相应发生改变。

该字段的缺省值是 MQC.MQCCSI_Q_MGR，指定应用程序消息数据中的字符数据是在队列管理器的字符集中。支持表13 中显示的附加字符集值。

表 13. 字符集标识

characterSet	描述
819	iso-8859-1 / latin1 / ibm819
912	iso-8859-2 / latin2 / ibm912
913	iso-8859-3 / latin3 / ibm913
914	iso-8859-4 / latin4 / ibm914
915	iso-8859-5 / cyrillic / ibm915
1089	iso-8859-6 / arabic / ibm1089
813	iso-8859-7 / greek / ibm813
916	iso-8859-8 / hebrew / ibm916
920	iso-8859-9 / latin5 / ibm920
37	ibm037
273	ibm273
277	ibm277
278	ibm278
280	ibm280
284	ibm284
285	ibm285
297	ibm297
420	ibm420
424	ibm424
437	ibm437 / PC 原码
500	ibm500
737	ibm737 / PC 希腊语
775	ibm775 / PC 波罗的海语
838	ibm838
850	ibm850 / PC 拉丁语 1
852	ibm852 / PC 拉丁语 2
855	ibm855 / PC 西里尔语
856	ibm856
857	ibm857 / PC 土耳其语
860	ibm860 / PC 葡萄牙语
861	ibm861 / PC 冰岛语
862	ibm862 / PC 希伯来语
863	ibm863 / PC 加拿大法语
864	ibm864 / PC 阿拉伯语
865	ibm865 / PC 北欧语
866	ibm866 / PC 俄语
868	ibm868
869	ibm869 / PC 现代希腊语
870	ibm870
871	ibm871
874	ibm874
875	ibm875
918	ibm918
921	ibm921
922	ibm922
930	ibm930
933	ibm933
935	ibm935
937	ibm937
939	ibm939

表 13. 字符集标识 (续)

characterSet	描述
942	ibm942
948	ibm948
949	ibm949
950	ibm950 / Big 5 繁体中文
964	ibm964 / CNS 11643 繁体中文
970	ibm970
1006	ibm1006
1025	ibm1025
1026	ibm1026
1097	ibm1097
1098	ibm1098
1112	ibm1112
1122	ibm1122
1123	ibm1123
1124	ibm1124
1381	ibm1381
1383	ibm1383
2022	JIS
932	PC 日语
954	EUCJIS
1250	Windows 拉丁语 2
1251	Windows 西里尔语
1252	Windows 拉丁语 1
1253	indows 希腊语
1254	Windows 土耳其语
1255	Windows 希伯来语
1256	Windows 阿拉伯语
1257	Windows 波罗的语
1258	Windows 越南语
33722	ibm33722
5601	ksc-5601 韩语
1200	Unicode
1208	UTF-8

format

```
public String format
```

消息发送人使用格式名向接收者表明消息中数据的性质。可以使用自己的格式名，但是以字母“MQ”开始的名称具有队列管理器定义的特殊含义。队列管理器内置的格式是:

MQC.MQFMT_NONE
无格式名。

MQC.MQFMT_ADMIN
命令服务器请求／回答消息。

MQC.MQFMT_COMMAND_1
类型 1 命令回答消息。

MQC.MQFMT_COMMAND_2
类型 2 命令回答消息。

MQC.MQFMT_DEAD_LETTER_HEADER
死信队列头。

MQC.MQFMT_EVENT
事件消息。

MQC.MQFMT_PCF
可编程命令格式的用户定义消息。

MQC.MQFMT_STRING
完全包含字符的消息。

MQC.MQFMT_TRIGGER
触发器消息

MQC.MQFMT_XMIT_Q_HEADER
传递队列头。

缺省值是 MQC.MQFMT_NONE。

priority

```
public int priority
```

信息优先级。也可以在出站消息中设置特殊值 MQC.MQPRI_PRIORITY_AS_Q_DEF，在这种情况下，消息的优先级是从目的队列的缺省优先级属性获得的。

缺省值是 MQC.MQPRI_PRIORITY_AS_Q_DEF。

persistence

```
public int persistence
```

消息持续性。已定义了下列值:

- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

缺省值是 MQC.MQPER_PERSISTENCE_AS_Q_DEF，表明消息的持续性应该从目的队列的缺省持续性属性获取。

messageId

```
public byte messageId[]
```

对于 MQQueue.get() 调用，该字段指定要检索的消息的消息标识。通常队列管理器返回第一个带消息标识的消息以及与指定相匹配的关联标识。特殊值 MQC.MQMI_NONE 允许*任何*消息标识与之匹配。

对于 MQQueue.put() 调用，这指定要使用的消息标识。如果指定了 MQC.MQMI_NONE，当放入消息时队列管理器会生成一个唯一的消息标识。放入后该成员变量的值会更新，以表明已使用的消息标识。

缺省值是 MQC.MQMI_NONE。

correlationId

```
public byte correlationId[]
```

对于 MQQueue.get() 调用，该字段指定要检索的消息的显式说明标识。通常队列管理器返回第一个带消息标识的消息，以及与指定相匹配的关联标识。特殊值 MQC.MQMI_NONE 允许*任何*相关标识与之匹配。

对于 MQQueue.put() 调用，这指定要使用的相关标识。

缺省值是 MQC.MQCI_NONE。

backoutCount

```
public int backoutCount
```

作为作业单元的一部分以前已由 MQQueue.get() 调用返回，后来又被复原的时间计数值。

缺省值为 0 。

replyToQueueName

```
public String replyToQueueName
```

消息队列的名称，为消息发出获取消息请求的应用程序应该向该队列发送 MQC.MQMT_REPLY 和 MQC.MQMT_REPORT 消息。

缺省值是 ″″。

replyToQueueManagerName

```
public String replyToQueueManagerName
```

回答和报告消息应该发送至的队列管理器的名称。

缺省值是 ″″。

如果值是 MQQueue.put() 调用上的 ″″ ，则 QueueManager 会填充该值。

userId

```
public String userId
```

消息的标识上下文的一部分，它标识了产生该消息的用户。

缺省值是 ″″。

accountingToken

```
public byte accountingToken[]
```

消息的标识上下文的一部分，只要对消息适当地计费，应用程序会使工作得以完成。

缺省值是 ″MQC.MQACT_NONE″。

applicationIdData

```
public String applicationIdData
```

消息的标识上下文的一部分；这个信息由应用程序套件定义，可用来提供消息或消息始发者的附加信息。

缺省值是 ″″。

putApplicationType

```
public int putApplicationType
```

放入消息的应用程序类型。它可以是系统定义或用户定义的值。下列值由系统定义:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

缺省值是特殊值 MQC.MQAT_NO_CONTEXT，表明消息中没有指出上下文信息。

putApplicationName

```
public String putApplicationName
```

放入消息的应用程序名称。缺省值是 ″″。

putDateTime

```
public GregorianCalendar putDateTime
```

放入消息的时间和日期。

applicationOriginData

```
public String applicationOriginData
```

由应用程序定义的信息，能提供有关消息起始的附加信息。

缺省值是 ″″。

groupId

```
public byte[] groupId
```

标识物理消息从属的消息组的字节字符串。

缺省值是 ″MQC.MQGI_NONE″。

messageSequenceNumber

```
public int messageSequenceNumber
```

组内逻辑消息的序号。

offset

```
public int offset
```

在分段消息中，从逻辑消息开始处在物理消息中数据的偏移量。

messageFlags

```
public int messageFlags
```

控制消息的分段和状态的标志。

originalLength

```
public int originalLength
```

分段消息的原始长度。

构造器

MQMessage

```
public MQMessage()
```

使用缺省消息描述符信息和空消息缓冲区创建新的消息。

方法

getTotalMessageLength

```
public int getTotalMessageLength()
```

存储在检索（或尝试检索）该消息的消息队列上的消息总计字节数。当 MQQueue.get() 方法失败，返回消息截断错误代码时，该方法指出队列上消息的总计大小。

另见第131页的『MQQueue.get』。

getMessageLength

```
public int getMessageLength
```

掷出 IOException。

该 MQMessage 对象中消息数据的字节数。

getDataLength

```
public int getDataLength()
```

掷出 MQException。

剩下要读取的消息数据的字节数。

seek

```
public void seek(int pos)
```

掷出 IOException。

将游标移动到由 *pos* 给出的消息缓冲区中的绝对位置。后续的读和写将在缓冲区中的这个位置上操作。

如果 pos 超出了消息数据的长度，则掷出 EOFException。

setDataOffset

```
public void setDataOffset(int offset)
```

掷出 IOException。

将游标移动到消息缓冲区中的绝对位置。该方法是 seek() 的同义词，是为使用其它 MQSeries API 的跨语言兼容性而提供的。

getDataOffset

```
public int getDataOffset()
```

掷出 IOException。

返回消息数据内的当前游标位置(读和写操作生效点)。

clearMessage

```
public void clearMessage()
```

掷出 IOException。

废弃消息缓冲区中的任何数据，并将数据偏移量重设成零。

getVersion

```
public int getVersion()
```

返回使用中的结构版本。

resizeBuffer

```
public void resizeBuffer(int size)
```

掷出 IOException。

暗示 MQMessage 对象有关后继获取操作可能需要的缓冲区大小。如果消息当前包含消息数据，且新的大小小于当前大小，则会截断消息数据。

readBoolean

```
public boolean readBoolean()
```

掷出 IOException。

从消息缓冲区的当前位置读（带符号）字节。

readChar

```
public char readChar()
```

掷出 IOException，EOFException。

从消息缓冲区的当前位置读一个 Unicode 字符。

readDouble

```
public double readDouble()
```

掷出 IOException、EOFException。

从消息缓冲区的当前位置读一个双精度数。该方法的行为由编码成员变量值确定。

MQC.MQENC_FLOAT_IEEE_NORMAL 和 MQC.MQENC_FLOAT_IEEE_REVERSED 的值分别读取一个大尾数法和小尾数法格式的 IEEE 标准双精度数。

MQC.MQENC_FLOAT_S390 值读取 System/390 格式浮点数。

readFloat

```
public float readFloat()
```

掷出 IOException、EOFException。

从消息缓冲区的当前位置读浮点数。该方法的行为由编码成员变量值确定。

MQC.MQENC_FLOAT_IEEE_NORMAL 和 MQC.MQENC_FLOAT_IEEE_REVERSED 的值分别读取大尾数法和小数法格式的 IEEE 标准浮点数。

MQC.MQENC_FLOAT_S390 值读取 System/390 格式浮点数。

readFully

```
public void readFully(byte b[])
```

掷出 Exception、EOFException。

使用来自消息缓冲区的数据填充字节数组 b。

readFully

```
public void readFully(byte b[],
                      int off,
                      int len)
```

掷出 IOException、EOFException。

使用来自消息缓冲区的数据，从偏移量 *off* 开始填充字节数组 b 的 *len* 元素。

readInt

```
public int readInt()
```

擲出 IOException，EOFException。

从消息缓冲区的当前位置读一个整数。该方法的行为由编码成员变量值确定。

MQC.MQENC_INTEGER_NORMAL 值读取大尾数法整数，MQC.MQENC_INTEGER_REVERSED 值读取小尾数法整数。

readInt4

```
public int readInt4()
```

擲出 IOException，EOFException。

readInt() 的同义词，为跨语言 MQSeries API 兼容性而提供。

readLine

```
public String readLine()
```

擲出 IOException。

从字符集成员变量中标识的代码集转换到 Unicode，然后读取以 \n、\r、\r\n 或 EOF 结尾的行。

readLong

```
public long readLong()
```

擲出 IOException，EOFException。

从消息缓冲区的当前位置读取一个长整数。该方法的行为由编码成员变量值确定。

MQC.MQENC_INTEGER_NORMAL 值读取大尾数法的长整数，MQC.MQENC_INTEGER_REVERSED 值读取小尾数法的长整数。

readInt8

```
public long readInt8()
```

擲出 IOException、EOFException。

readLong() 的同义词，为跨语言 MQSeries API 兼容性而提供。

readObject

```
public Object readObject()
```

擲出 OptionalDataException、ClassNotFoundException、IOException。

从消息缓冲区读取对象。读取对象类，类的特征以及类的非短暂和非静态字段。

readShort

```
public short readShort()
```

掷出 IOException，EOFException。

readInt2

```
public short readInt2()
```

掷出 IOException，EOFException。

readShort() 的同义词，为跨语言 MQSeries API 兼容性而提供。

readUTF

```
public String readUTF()
```

掷出 IOException。

从消息缓冲区中的当前位置读取带有 2 字节长度字段前缀的 UTF 字符串。

readUnsignedByte

```
public int readUnsignedByte()
```

掷出 IOException，EOFException。

从消息缓冲区的当前位置读无符号字字节。

readUnsignedShort

```
public int readUnsignedShort()
```

掷出 IOException，EOFException。

从消息缓冲区的当前位置读无符号短整数。该方法的行为由编码成员变量值确定。

MQC.MQENC_INTEGER_NORMAL 值读取大尾数法的无符号短整数，MQC.MQENC_INTEGER_REVERSED 值读取小尾数法的无符号短整数。

readUInt2

```
public int readUInt2()
```

掷出 IOException、EOFException。

readUnsignedShort() 的同义词，为跨语言 MQSeries API 兼容性而提供。

readString

```
public String readString(int length)
```

掷出 IOException、EOFException。

读取由字符集成员变量标识的代码集中的字符串，并将它转换成 Unicode。

参数:

length 要读取的字符数（可能根据代码集不同，字节数也不同，因为一些代码集每字符使用一个以上字节）。

readDecimal2

```
public short readDecimal2()
```

掷出 IOException，EOFException。

读取 2 字节压缩十进制数 (-999..999)。该方法的行为由编码成员变量值控制。MQC.MQENC_DECIMAL_NORMAL 值读大尾数法压缩十进制，MQC.MQENC_DECIMAL_REVERSED 值读小尾数法压缩十进制数。

readDecimal4

```
public int readDecimal4()
```

掷出 IOException、EOFException。

读取 4 字节压缩十进制数 (-9999999..9999999)。该方法的行为由编码成员变量值控制。 MQC.MQENC_DECIMAL_NORMAL 值读大尾数法压缩十进制，MQC.MQENC_DECIMAL_REVERSED 值读小尾数法压缩十进制数。

readDecimal8

```
public long readDecimal8()
```

掷出 IOException、EOFException。

读取 8 字节压缩十进制数（-999999999999999 至 999999999999999）。该方法的行为由编码成员变量控制。MQC.MQENC_DECIMAL_NORMAL 值读大尾数法压缩十进制， MQC.MQENC_DECIMAL_REVERSED 值读小尾数法压缩十进制数。

setVersion

```
public void setVersion(int version)
```

指定要使用什么结构版本。可能的值有:
- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

通常不需要调用该方法，除非当连接到能处理版本 2 结构的队列管理器时，想强制客户使用版本 1 结构。在其它所有情况中，客户通过查询队列管理器的能力来确定要使用的结构的正确版本。

skipBytes

```
public int skipBytes(int n)
```

掷出 IOException，EOFException。

在消息缓冲区中向前移动 n 字节。

该方法阻塞，直到发生下列情况之一:
- 跳过所有字节
- 检测到消息缓冲区结束
- 产生异常

返回跳过的字节数，总是 n。

write

```
public void write(int b)
```

掷出 IOException。

在当前位置将字节写入消息缓冲区。

write

```
public void write(byte b[])
```

掷出 IOException。

在当前位置将字节数组写入消息缓冲区。

write

```
public void write(byte b[],
                      int off,
                      int len)
```

掷出 IOException。

在当前位置将一系列字节数组写入消息缓冲区。将写入从数组 b 偏移量 *off* 开始取出的 *len* 字节。

writeBoolean

```
public void writeBoolean(boolean v)
```

掷出 IOException。

在当前位置将一个 boolean 写入消息缓冲区。

writeByte

```
public void writeByte(int v)
```

掷出 IOException。

在当前位置将字节写入消息缓冲区。

writeBytes

```
public void writeBytes(String s)
```

擲出 IOException。

将字符串作为字节序列写入消息缓冲区。按舍弃其高 8 位的顺序写出字符串中的每个字符。

writeChar

```
public void writeChar(int v)
```

擲出 IOException。

在当前位置将 Unicode 字符写入消息缓冲区。

writeChars

```
public void writeChars(String s)
```

擲出 IOException。

在当前位置将作为 Unicode 字符序列的字符串写入消息缓冲区。

writeDouble

```
public void writeDouble(double v)
```

擲出 IOException

在当前位置将双精度数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_FLOAT_IEEE_NORMAL 和 MQC.MQENC_FLOAT_IEEE_REVERSED 的值分别写大尾数法和小尾数法格式的 IEEE 标准浮点数。

MQC.MQENC_FLOAT_S390 值写 System/390 格式浮点数。注意， IEEE 双字节范围比 S/390® 双字节精度浮点数范围大，所以不能转换非常大的数。

writeFloat

```
public void writeFloat(float v)
```

擲出 IOException。

在当前位置将浮点数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_FLOAT_IEEE_NORMAL 和 MQC.MQENC_FLOAT_IEEE_REVERSED 的值分别写大尾数法和小尾数法格式的 IEEE 标准浮点数。

MQC.MQENC_FLOAT_S390 值将写 System/390 格式浮点数。

writeInt

```
public void writeInt(int v)
```

掷出 IOException。

在当前位置将整数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_INTEGER_NORMAL 值写大尾数法整数，MQC.MQENC_INTEGER_REVERSED 值写小尾数法整数。

writeInt4

```
public void writeInt4(int v)
```

掷出 IOException。

writeInt() 的同义词，为跨语言 MQSeries API 兼容性而提供。

writeLong

```
public void writeLong(long v)
```

掷出 IOException。

在当前位置将一个长整数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_INTEGER_NORMAL 值写一个大尾数法的长整数，MQC.MQENC_INTEGER_REVERSED 值写一个小尾数法的长整数。

writeInt8

```
public void writeInt8(long v)
```

掷出 IOException。

writeLong() 的同义词，为跨语言 MQSeries API 兼容性而提供。

writeObject

```
public void writeObject(Object obj)
```

掷出 IOException。

将被指定的对象写入消息缓冲区。对象的类，类的签名，类的非短暂和非静态字段值以及其所有超类型全被写入。

writeShort

```
public void writeShort(int v)
```

掷出 IOException。

在当前位置将短整数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_INTEGER_NORMAL 值写大尾数法的短整数，MQC.MQENC_INTEGER_REVERSED 值写小尾数法的短整数。

writeInt2

```
public void writeInt2(int v)
```

掷出 IOException。

writeShort() 的同义词，为跨语言 MQSeries API 兼容性而提供。

writeDecimal2

```
public void writeDecimal2(short v)
```

掷出 IOException。

在当前位置将 2 字节压缩十进位格式数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_DECIMAL_NORMAL 值写大尾数法压缩十进制，MQC.MQENC_DECIMAL_REVERSED 值写小尾数法压缩十进制。

参数

v 可以在 -999 至 999 的范围内。

writeDecimal4

```
public void writeDecimal4(int v)
```

掷出 IOException。

在当前位置将 4 字节压缩十进位格式数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_DECIMAL_NORMAL 值写大尾数法压缩十进制，MQC.MQENC_DECIMAL_REVERSED 值写小尾数法压缩十进制。

参数

v 可以在 -9999999 至 9999999 的范围内。

writeDecimal8

```
public void writeDecimal8(long v)
```

掷出 IOException。

在当前位置将 8 字节压缩十进位格式数写入消息缓冲区。该方法的行为由编码成员变量值确定。

MQC.MQENC_DECIMAL_NORMAL 值写大尾数法压缩十进制，MQC.MQENC_DECIMAL_REVERSED 值写小尾数法压缩十进制。

参数:

v 可以在 -999999999999999 至 999999999999999 的范围内。

writeUTF

```
public void writeUTF(String str)
```

掷出 IOException。

将带有 2 字节长度字段前缀的 UTF 字符串写入消息缓冲区中的当前位置。

writeString

```
public void writeString(String str)
```

掷出 IOException。

在当前位置将字符串写入消息缓冲区，并将它转换为字符集成员变量标识的代码集。

MQMessageTracker

```
java.lang.Object
   │
   └─ com.ibm.mq.MQMessageTracker
```

public abstract class **MQMessageTracker**
extends **Object**

注: 只有在连接到 MQSeries 版本 5（或更高版）队列管理器时才能使用该类。

该类由 MQDistributionListItem（第 85 页）继承，用于针对分发列表中给定的目的地剪裁消息参数。

变量

feedback

```
public int feedback
```

这个值与 MQC.MQMT_REPORT 消息类型一起表明报告的性质。下列反馈码是由系统定义的:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

也可使用 MQC.MQFB_APPL_FIRST 至 MQC.MQFB_APPL_LAST 范围中应用程序定义的反馈值。

该字段的缺省值是 MQC.MQFB_NONE，表明没有提供任何反馈。

messageId

```
public byte messageId[]
```

这指定了当放入消息时要使用的消息标识。如果指定了 MQC.MQMI_NONE，当放入消息时队列管理器会生成一个唯一的消息标识。放入后该成员变量的值会更新，以表明已使用的消息标识。

缺省值是 MQC.MQMI_NONE。

correlationId

```
public byte correlationId[]
```

这指定了当放入消息时要使用的关联标识。

缺省值是 MQC.MQCI_NONE。

accountingToken

```
public byte accountingToken[]
```

这是消息的标识上下文的一部分。只要对消息适当地计费，应用程序会使工作得以完成。

缺省值是 MQC.MQCI_NONE。

groupId

```
public byte[] groupId
```

标识物理消息所属的消息组的字节字符串。

缺省值是 MQC.MQCI_NONE。

MQPoolServices

```
java.lang.Object
   │
   └─ com.ibm.mq.MQPoolServices
```

public class **MQPoolServices**
extends **Object**

注: 通常，应用程序不使用该类。

想要用作 MQSeries 连接的缺省 ConnectionManager 的 ConnectionManager 实现可以使用 MQPoolService 类。

ConnectionManager 可以构造 MQPoolServices 对象并通过它注册侦听器。该侦听器接收与 MQEnvironment 管理的 MQPoolToken 集合相关的事件。ConnectionManager 可使用该信息执行任何必要的启动或清除工作。

另见第122页的『MQPoolServicesEvent』和第151页的『MQPoolServicesEventListener』。

构造器

MQPoolServices

```
public MQPoolServices()
```

构造新的 MQPoolService 对象。

方法

addMQPoolServicesEventListener

```
public void addMQPoolServicesEventListener
                       (MQPoolServicesEventListener listener)
```

添加 MQPoolServicesEventListener。任何时候将令牌添加到 MQEnvironment 控制的 MQPoolToken 集合或从其中除去，或更改了缺省 ConnectionManager，则侦听器都将接收到事件。

removeMQPoolServicesEventListener

```
public void removeMQPoolServicesEventListener
                       (MQPoolServicesEventListener listener)
```

除去 MQPoolServicesEventListener。

getTokenCount

```
public int getTokenCount()
```

返回当前用 MQEnvironment 注册的 MQPoolToken 数。

MQPoolServicesEvent

```
java.lang.Object
   │
   └─ java.util.EventObject
         │
         └─ com.ibm.mq.MQPoolServicesEvent
```

注: 通常，应用程序不使用该类。

任何时候将 MQPoolToken 添加到 MQEnvironment 控制的令牌集或从其中除去都将生成 MQPoolServicesEvent。更改缺省 ConnectionManager 时也会生成事件。

另见第121页的『MQPoolServices』和第151页的『MQPoolServicesEventListener』。

变量

TOKEN_ADDED

```
public static final int TOKEN_ADDED
```

将 MQPoolToken 添加到集合时使用的事件标识。

TOKEN_REMOVED

```
public static final int TOKEN_REMOVED
```

从集合中除去 MQPoolToken 时使用的事件标识。

DEFAULT_POOL_CHANGED

```
public static final int DEFAULT_POOL_CHANGED
```

更改缺省 ConnectionManager 时使用的事件标识。

ID

```
protected int ID
```

事件标识。有效值有:

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

token

```
protected MQPoolToken token
```

令牌。事件标识为 DEFAULT_POOL_CHANGED 时它为空。

构造器

MQPoolServicesEvent

```
public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)
```

构造基于事件标识和令牌的 MQPoolServicesEvent。

MQPoolServicesEvent

```
public MQPoolServicesEvent(Object source, int eid)
```

构造基于事件标识的 MQPoolServicesEvent。

方法

getId

```
public int getId()
```

获取事件标识。

返回

带有下列值之一的事件标识:

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

getToken

```
public MQPoolToken getToken()
```

返回添加到集合或从其中除去的令牌。如果标识是 DEFAULT_POOL_CHANGED，则它为空。

MQPoolToken

```
java.lang.Object
   │
   └─ com.ibm.mq.MQPoolToken
```

public class **MQPoolToken**
extends **Object**

可使用 MQPoolToken 启用连接池。应用程序组件连接到 MQSeries 前，使用 MQEnvironment 类注册 MQPoolToken。之后，当组件完成 MQSeries 使用后撤销注册它们。典型的情况是已注册的 MQPoolToken 非空时缺省 ConnectionManager 是活动的。

MQPoolToken 不提供方法或变量。ConnectionManager 供应商可选择扩展 MQPoolToken，这样就可将提示传递到 ConnectionManager。

参阅第90页的『MQEnvironment.addConnectionPoolToken』和第90页的『MQEnvironment.removeConnectionPoolToken』。

构造器

MQPoolToken

```
public MQPoolToken()
```

构造新的 MQPoolToken 对象。

MQProcess

```
java.lang.Object
   │
   └─ com.ibm.mq.MQManagedObject
         │
         └─ com.ibm.mq.MQProcess
```

public class **MQProcess**
extends **MQManagedObject**。(参见第 97 页。)

MQProcess 提供 MQSeries 进程的查询操作。

构造器

MQProcess

```
public MQProcess(MQQueueManager qMgr,
                                 String processName,
                                 int openOptions,
                                 String queueManagerName,
                                 String alternateUserId)
                     throws MQException
```

访问队列管理器 qMgr 上的进程。参阅第138页的『MQQueueManager』中的 accessProcess，以获得剩余参数的详细信息。

方法

getApplicationId

```
public String getApplicationId()
```

标识要启动的应用程序的字符串。该消息由在初始队列中处理消息的触发器监视器使用；该消息将作为触发器消息的一部分发送给初始队列。

如果在关闭进程后调用该方法，则掷出 MQException。

getApplicationType

```
public int getApplicationType()
```

掷出 MQException（参阅第 91 页）。

在响应触发器消息的收条中，这标识了要启动的程序的本质。应用程序可以有任何值，但推荐用下列标准类型值:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT
- MQC.MWQAT_USER_FIRST（用户定义的应用程序类型的最低值）
- MQC.MQAT_USER_LAST（用户定义的应用程序类型的最高值）

MQProcess

getEnvironmentData

```
public String getEnvironmentData()
```

掷出 MQException。

这是个包含环境相关信息的字符串，该信息指向所要启动的应用程序。

getUserData

```
public String getUserData()
```

掷出 MQException。

这是个包含与所要启动的应用程序相关的用户信息的字符串。

关闭

```
public synchronized void close()
```

掷出 MQException。

覆盖第99页的『MQManagedObject.close』。

MQPutMessageOptions

```
java.lang.Object
   |
   └─ com.ibm.mq.MQPutMessageOptions
```

public class **MQPutMessageOptions**
extends **Object**

该类包含控制 MQQueue.put() 操作的选项。

注: 该类中可用的一些选项的操作取决于使用它们的环境。使用 * 来标记这些元素。有关详细信息，请参阅第74页的『运行于其它环境下的版本 5 扩展』。

变量

options

```
public int options
```

控制 MQQueue.put 操作的选项。可以指定下列任何值，或不指定下列值。如果需要多个选项，可以使用按位“或”运算符添加或组合这些值。

MQC.MQPMO_SYNCPOINT
使用同步点控制放入消息。该消息在工作单元外部是不可见的，直到提交工作单元为止。如果工作单元被逆序恢复，那么将删除消息。

MQC.MQPMO_NO_SYNCPOINT
不使用同步点控制放入消息。注意，如果没有指定同步点控制选项，则假设为缺省值 ‘no syncpoint’。这适用于所有平台，包括 OS/390。

MQC.MQPMO_NO_CONTEXT
没有与消息关联的上下文。

MQC.MQPMO_DEFAULT_CONTEXT
将缺省上下文与消息关联。

MQC.MQPMO_SET_IDENTITY_CONTEXT
从应用程序设置标识上下文。

MQC.MQPMO_SET_ALL_CONTEXT
从应用程序设置所有上下文。

MQC.MQPMO_FAIL_IF_QUIESCING
如果队列管理器停顿，则失败。

MQC.MQPMO_NEW_MSG_ID*
为每条发送的消息生成新的消息标识。

MQC.MQPMO_NEW_CORREL_ID*
为每条发送的消息生成新的关联标识。

MQC.MQPMO_LOGICAL_ORDER*
将消息组中的逻辑消息和段按它们的逻辑次序放入。

MQC.MQPMO_NONE
没有指定选项。不要与其它选项一起使用。

MQC.MQPMO_PASS_IDENTITY_CONTEXT
从输入队列句柄传送标识上下文。

MQC.MQPMO_PASS_ALL_CONTEXT
从输入队列句柄传送所有上下文。

contextReference

```
public MQQueue ContextReference
```

这是表明上下文信息源的输入字段。

如果 `options` 字段包含 MQC.MQPMO_PASS_IDENTITY_CONTEXT 或 MQC.MQPMO_PASS_ALL_CONTEXT，则设置该字段以指向获取上下文消息的 MQQueue。

该字段的初始值是空。

recordFields *

```
public int recordFields
```

一种标志，指示将消息放置到分发列表时，哪些字段需要按每个队列定制。可以指定下列标志中的一个或多个:

MQC.MQPMRF_MSG_ID
使用 MQDistributionListItem 中的 messageId 属性。

MQC.MQPMRF_CORREL_ID
使用 MQDistributionListItem 中的 correlationId 属性。

MQC.MQPMRF_GROUP_ID
使用 MQDistributionListItem 中的 groupId 属性。

MQC.MQPMRF_FEEDBACK
使用 MQDistributionListItem 中的 feedback 属性。

MQC.MQPMRF_ACCOUNTING_TOKEN
使用 MQDistributionListItem 中的 accountingToken 属性。

特殊值 MQC.MQPMRF_NONE 表明没有字段将要定制。

resolvedQueueName

```
public String resolvedQueueName
```

这是由队列管理器设置为放置消息的队列名称的输出字段。如果打开的队列是别名或模型队列，这将与用于打开队列的名称不同。

resolvedQueueManagerName

```
public String resolvedQueueManagerName
```

这是由队列管理器设置为队列管理器名称的输出字段，该队列管理器拥有远程队列名称指定的队列。这可能与队列管理器（如果队列是远程队列，则可以从该队列管理器访问队列）的名称不同。

knownDestCount *

```
public int knownDestCount
```

这是由队列管理器设置为消息数（消息是当前调用已成功发送到解析为本地队列的队列的消息）的输出字段。当打开一个不是分发列表一部分的单一队列时，也将设置该字段。

unknownDestCount *

```
public int unknownDestCount
```

这是由队列管理器设置为消息数（消息是当前调用已成功发送到解析为远程队列的队列的消息）的输出字段。当打开一个不是分发列表一部分的单一队列时，也将设置该字段。

invalidDestCount *

```
public int invalidDestCount
```

这是由队列管理器设置为消息数（消息不能被发送到分发列表中的队列）的输出字段。计数值包含不能成功打开的队列，也包含能成功打开但放入操作失败的队列。当打开一个不是分发列表一部分的单一队列时，也将设置该字段。

构造器

MQPutMessageOptions

```
public MQPutMessageOptions()
```

构造一个没有设置选项的新的 MQPutMessageOptions 对象，以及空白的 resolvedQueueName 和 resolvedQueueManagerName。

MQQueue

```
java.lang.Object
   │
   └─ com.ibm.mq.MQManagedObject
           │
           └─ com.ibm.mq.MQQueue
```

public class **MQQueue**
extends **MQManagedObject**。(参见第 97 页。)

MQQueue 提供 MQSeries 队列的查询、设置、放入和获取操作。查询和设置能力是从 MQ.MQManagedObject 继承来的。

另见第143页的『MQQueueManager.accessQueue』。

构造器

MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
                             String alternateUserId)
               throws MQException
```

访问队列管理器 qMgr 上的队列。

有关保留参数的详细信息'，请参阅第143页的『MQQueueManager.accessQueue』。

方法

get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions,
                             int MaxMsgSize)
```

掷出 MQException。

从队列中检索消息，达到最大指定消息大小。

此方法接受一个 MQMessage 对象作为参数。它将该对象的某些字段用作输入参数 - 特别是 messageId 和 correlationId，所以保证已按需要设置它们十分重要。（请参阅第257页的『Message』。）

如果获取失败，则 MQMessage 对象就不会更改。如果成功，则 MQMessage 的消息描述符（成员变量）和消息数据部分将完全用来自进入消息的消息描述符和消息数据替换。

注意，所有来自给定 MQQueueManager 的对 MQSeries 的调用都是同步的。因此，如果使用 wait 执行 get 操作，则使用相同 MQQueueManager 的所有其它线程都将被阻塞，不能执行进一步 MQSeries 调用直到 get 操作完成。如果需要多个线程同步访问 MQSeries，则每个线程都必须创建它自己的 MQQueueManager 对象。

参数

message
包含消息描述符信息和返回的消息数据的输入／输出参数。

getMessageOptions
控制获取操作的选项。（请参阅第93页的『MQGetMessageOptions』。）

MaxMsgSize
该调用能接收到的最大消息。如果队列上的消息大于这个大小，则可能发生下列情况之一:

1. 如果已在 MQGetMessageOptions 对象的选项成员变量中设置 MQC.MQGMO_ACCEPT_TRUNCATED_MSG 标志，则消息中会填充适合指定缓冲区大小的消息数据，且掷出异常，返回完成码 MQException.MQCC_WARNING 和原因码 MQException.MQRC_TRUNCATED_MSG_ACCEPTED。
2. 如果没有设置 MQC.MQGMO_ACCEPT_TRUNCATED_MSG 标志，则消息被留在队列上，且掷出 MQException，返回完成码 MQException.MQCC_WARNING 和原因码 MQException.MQRC_TRUNCATED_MSG_FAILED。

如果获取失败，则掷出 MQException。

get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions)
```

掷出 MQException。

从队列中检索消息，无论信息大小。对于较大的消息，get 方法必须代表您对 MQSeries 发出两个调用，一个建立所需的缓冲区大小，一个获取消息数据本身。

此方法接受一个 MQMessage 对象作为参数。它将该对象的某些字段用作输入参数 - 特别是 messageId 和 correlationId，所以保证已按需要设置它们十分重要。（请参阅第257页的『Message』。）

如果 get 失败，则 MQMessage 对象就不会更改。如果成功，则 MQMessage 的消息描述符（成员变量）和消息数据部分将完全用自进入信息的消息描述符和消息数据替换。

注意，所有来自给定 MQQueueManager 的对 MQSeries 的调用都是同步的。因此，如果使用 wait 执行 get 操作，则使用相同 MQQueueManager 的所有其它线程都将被阻塞，不能执行进一步 MQSeries 调用直到 get 操作完成。如果需要多个线程同步访问 MQSeries，则每个线程都必须创建它自己的 MQQueueManager 对象。

参数

message
包含消息描述符信息和返回的消息数据的输入／输出参数。

getMessageOptions
控制获取操作的选项。（详细信息，请参阅第93页的『MQGetMessageOptions』。）

如果 get 失败，则掷出 MQException。

get

```
public synchronized void get(MQMessage message)
```

这是以前描述过的获取方法的简化版本。

参数

MQMessage
包含消息描述符信息和返回的消息数据的输入／输出参数。

该方法使用缺省 MQGetMessageOptions 实例执行 get 操作。使用的消息选项是 MQGMO_NOWAIT。

put

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions)
```

掷出 MQException。

将消息放入队列。

此方法接受一个 MQMessage 对象作为参数。该对象的消息描述符属性可能因该方法的结果而改变。该方法完成后立即获得的值是放入 MQSeries 队列的值。

完成放入后对 MQMessage 对象的修改不会影响 MQSeries 队列上的实际消息。

put 更新了消息标识与相关标识。当使用相同的 MQMessage 对象进一步调用 put/get 时必须考虑这一点。而且，调用 put 不会清除消息数据，所以:

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

放入两个消息。第一个包含 ″a″，第二个包含 ″ab″。

参数

message
包含要发送的消息描述符数据和消息的消息缓冲区。

putMessageOptions
控制放入操作的选项。(请参阅第127页的『MQPutMessageOptions』)

如果 put 操作失败，则掷出 MQException。

put

```
public synchronized void put(MQMessage message)
```

这是以前描述过的 put 方法的简化版本。

参数

MQMessage
包含要发送的消息描述符数据和消息的消息缓冲区。

该方法使用缺省 MQPutMessageOptions 实例执行放入。

注: 如果在关闭队列后调用方法，则所有下列方法掷出 MQException。

getCreationDateTime

```
public GregorianCalendar getCreationDateTime()
```

掷出 MQException。

创建该队列的日期和时间。

getQueueType

```
public int getQueueType()
```

掷出 MQException

返回: 使用下列值之一的队列类型:
- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

getCurrentDepth

```
public int getCurrentDepth()
```

掷出 MQException。

获取当前队列上的消息数。在 put 调用和调用逆序恢复 get 期间，该值会递增。而在非浏览 get 和逆序恢复 put 调用期间减小。

getDefinitionType

```
public int getDefinitionType()
```

掷出 MQException。

表明了队列是如何定义的。

返回: 下列值之一:
- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

getMaximumDepth

```
public int getMaximumDepth()
```

掷出 MQException。

任何时候队列上存在的最大消息数量。尝试把消息放入已包含了这些消息的队列将失败，原因码为 MQException.MQRC_Q_FULL。

getMaximumMessageLength

```
public int getMaximumMessageLength()
```

掷出 MQException。

这是队列上的每个消息中能存在的最大应用程序数据长度。尝试放入大于该值的消息会失败，原因码为 MQException.MQRC_MSG_TOO_BIG_FOR_Q。

getOpenInputCount

```
public int getOpenInputCount()
```

掷出 MQException。

对从队列删除消息当前有效的句柄数。这是本地队列管理器所知的*所有*句柄数，而不仅仅是 MQSeries classes for Java（使用 accessQueue）创建的那些。

getOpenOutputCount

```
public int getOpenOutputCount()
```

掷出 MQException。

对将消息添加到队列当前有效的句柄数。这是本地队列管理器所知的*所有*句柄数，而不仅仅是 MQSeries classes for Java（使用 accessQueue）创建的那些。

getShareability

```
public int getShareability()
```

掷出 MQException。

它表明了是否可以多次打开队列以输入。

返回: 下列值之一:
- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

getInhibitPut

```
public int getInhibitPut()
```

掷出 MQException。

表明该队列是否允许放入操作。

返回: 下列值之一:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

setInhibitPut

```
public void setInhibitPut(int inhibit)
```

掷出 MQException。

控制该队列是否允许放入操作。可允许的值是:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

getInhibitGet

```
public int getInhibitGet()
```

掷出 MQException。

表明该队列是否允许获取操作。

返回: 可能的值是:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

setInhibitGet

```
public void setInhibitGet(int inhibit)
```

掷出 MQException。

控制该队列是否允许获取操作。可允许的值是:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

getTriggerControl

```
public int getTriggerControl()
```

掷出 MQException。

表明触发器消息是否被写入了一个启动队列，以便启动应用程序服务于队列。

返回: 可能的值是:
- MQC.MQTC_OFF
- MQC.MQTC_ON

setTriggerControl

```
public void setTriggerControl(int trigger)
```

掷出 MQException。

控制触发器消息是否被写入了一个启动队列，以便启动应用程序以服务于队列。可允许的值是:
- MQC.MQTC_OFF
- MQC.MQTC_ON

getTriggerData

```
public String getTriggerData()
```

掷出 MQException。

当到达队列的某个消息导致一个触发器消息被写入启动队列时，队列管理器将在触发器消息中插入的自由格式数据。

setTriggerData

```
public void setTriggerData(String data)
```

掷出 MQException。

当到达队列的某个消息导致一个触发器消息被写入启动队列时，设置队列管理器将在触发器消息中插入的自由格式数据。字符串的最大允许长度由 MQC.MQ_TRIGGER_DATA_LENGTH 给出。

getTriggerDepth

```
public int getTriggerDepth()
```

掷出 MQException。

当触发器设置成 MQC.MQTT_DEPTH，在触发器消息被写入之前必须在队列上存在消息数。

setTriggerDepth

```
public void setTriggerDepth(int depth)
```

掷出 MQException。

当触发器设置成 MQC.MQTT_DEPTH，在触发器消息被写入之前设置的必须存在于队列上的消息数。

getTriggerMessagePriority

```
public int getTriggerMessagePriority()
```

掷出 MQException。

这是指消息的优先级，低于该优先级，消息将不能用于产生触发器消息（也就是说，当队列管理器在决定生成哪个触发器时将忽略这些信息）。当值为零时，将导致所有消息都用于生成触发器消息。

setTriggerMessagePriority

```
public void setTriggerMessagePriority(int priority)
```

掷出 MQException。

设置消息的优先级，低于该优先级，消息将不能用于产生触发器消息(也就是说，当队列管理器在决定生成哪个触发器时将忽略这些信息)。当值为零时，将导致所有消息都用于生成触发器消息。

getTriggerType

```
public int getTriggerType()
```

掷出 MQException。

在该条件下，触发器消息被作为到达该队列的消息的结果而写入。

返回: 可能的值是:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

setTriggerType

```
public void setTriggerType(int type)
```

掷出 MQException。

设置条件，在该条件下触发器消息被作为到达该队列的消息的结果而写入。可能的值是:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

close

```
public synchronized void close()
```

掷出 MQException。

覆盖第99页的『MQManagedObject.close』。

MQQueueManager

```
java.lang.Object
   │
   └─ com.ibm.mq.MQManagedObject
         │
         └─ com.ibm.mq.MQQueueManager
```

public class **MQQueueManager**
extends **MQManagedObject**。（参见第 97 页。）

注: 该类中可用的一些选项的操作取决于使用它们的环境。使用 * 来标记这些元素。有关详细信息，请参阅第71页的『第8章 取决于环境的行为』。

变量

isConnected

```
public boolean isConnected
```

如果至队列管理器的连接仍打开着，则为真。

构造器

MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

掷出 MQException。

创建至给定名称队列的连接。

注: 当使用 MQSeries classes for Java 时，在连接请求期间要使用的主机名、通道名和端口将在 MQEnvironment 类中指定。必须在调用该构造器*之前*完成。

下列示例显示了运行在主机名为 fred.mq.com 的机器上的队列管理器 ″MYQM″ 的连接。

```
MQEnvironment.hostname = "fred.mq.com";  // 要连接的主机
MQEnvironment.port     = 1414;           // 要连接的端口。
                                         // 如果不设置，
                                         // 则缺省为 1414
                                         // （缺省的 MQSeries 端口）
MQEnvironment.channel  = "channel.name"; // 队列管理器上
                                         // SVR CONN 通
                                         // 道“区分大
                                         // 小写”的名称
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

如果队列管理器名称为空白（空或“”），则连接至缺省队列管理器。

另见第86页的『MQEnvironment』。

| **MQQueueManager**

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager cxManager)
```

掷出 MQException。

该构造器使用 MQEnvironment 中的特性连接到指定的“队列管理器”。指定的 MQConnectionManager 管理了连接。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager cxManager)
```

掷出 MQException。

该构造器使用 MQEnvironment 中的特性连接到指定的“队列管理器”。指定的 ConnectionManager 管理了连接。

这个方法需要安装带 JAAS 1.0 或更新版的 Java 2 v1.3 或更新版本的 JVM。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options)
```

掷出 MQException。

构造器的这个版本仅为绑定方式使用，它使用扩展的连接 API (MQCONNX) 连接队列管理器。 *options* 参数允许您选择快速或标准绑定。可能的值有:
- MQC.MQCNO_FASTPATH_BINDING 用于快速绑定 *。
- MQC.MQCNO_STANDARD_BINDING 用于标准绑定。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      MQConnectionManager cxManager)
```

掷出 MQException。

这个构造器执行一个 MQCONNX，传递提供的选项。指定的 MQConnectionManager 管理了连接。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

掷出 MQException。

这个构造器执行一个 MQCONNX，传递提供的选项。指定的 ConnectionManager 管理了连接。

这个方法需要安装带 JAAS 1.0 或更新版的 Java 2 v1.3 或更新版本的 JVM。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
java.util.Hashtable properties)
```

MQQueueManager

该特性参数采用一系列描述特定队列管理器的 MQSeries 环境的键／值对。这里指定的这些特性忽略 MQEnvironment 类设置的值，并允许按队列管理器为队列管理器设置个别特性。请参阅第87页的『"MQEnvironment.properties"』。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      MQConnectionManager cxManager)
```

掷出 MQException。

该构造器通过使用所提供的特性散列表覆盖 MQEnvironment，连接到命名过的“队列管理器”。指定的 MQConnectionManager 管理了连接。

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      ConnectionManager cxManager)
```

掷出 MQException。

该构造器通过使用所提供的特性散列表覆盖 MQEnvironment，连接到命名过的“队列管理器”。指定的 ConnectionManager 管理了连接。

这个方法需要安装带 JAAS 1.0 或更新版的 Java 2 v1.3 或更新版本的 JVM。

方法

getCharacterSet

```
public int getCharacterSet()
```

掷出 MQException。

返回队列管理器代码集的 CCSID（编码字符集标识）。这为应用程序编程接口中所有字符串字段定义了队列管理器使用的字符集。

如果从队列管理器断开后调用该方法，则掷出 MQException。

getMaximumMessageLength

```
public int getMaximumMessageLength()
```

掷出 MQException。

返回队列管理器能处理的最大消息长度（以字节）。不能使用大于该最大消息长度的长度定义队列。

如果从队列管理器断开后调用该方法，则掷出 MQException。

getCommandLevel

```
public int getCommandLevel()
```

掷出 MQException。

指定了队列管理器支持的系统控制命令级别。与特定命令级别相适应的系统控制命令集根据平台（其上运行队列管理器）体系结构的不同而不同。请参阅 MQSeries 文档，以获得平台的进一步细节。

如果从队列管理器断开后调用该方法，则掷出 MQException。

返回: 一个 MQC.MQCMDL_LEVEL_xxx 常数

getCommandInputQueueName

```
public String getCommandInputQueueName()
```

掷出 MQException。

返回队列管理器上定义的命令输入队列的名称。如果授权这样做，这是一个应用程序能将命令发送至的队列。

如果从队列管理器断开后调用该方法，则掷出 MQException。

getMaximumPriority

```
public int getMaximumPriority()
```

掷出 MQException。

返回队列管理器支持的最大消息优先级。优先级范围从零（最低）到该值。

如果从队列管理器断开后调用该方法，则掷出 MQException。

getSyncpointAvailability

```
public int getSyncpointAvailability()
```

掷出 MQException。

表示本地队列管理器是否支持工作单元，并与 MQQueue.get 和 MQQueue.put 方法同步。

返回:
- MQC.MQSP_AVAILABLE，如果能同步。
- MQC.MQSP_NOT_AVAILABLE，如果不能同步。

如果从队列管理器断开后调用该方法，则掷出 MQException。

getDistributionListCapable

```
public boolean getDistributionListCapable()
```

表明队列管理器是否支持分发列表。

disconnect

```
public synchronized void disconnect()
```

掷出 MQException。

终止至队列管理器的连接。队列管理器访问的所有打开队列和进程都已关闭，所以该连接不可用。当从队列管理器断开时，重新连接的唯一方法是创建一个新的 MQQueueManager 对象。

通常，任何作为工作单元一部分执行的工作将被提交。但是，如果是由 ConnectionManager 管理连接，而不是 MQConnectionManager，则工作单元可能要逆序恢复。

commit

```
public synchronized void commit()
```

掷出 MQException。

调用该方法是对队列管理器表明应用程序已到达同步点，且所有自最近一次同步点以来发生的消息的获取和放入都为永久的。作为工作单元一部分放入的消息（MQPutMessageOptions 的选项字段中设置了 MQC.MQPMO_SYNCPOINT 标志）对其它应用程序是可用的。删除了作为工作单元一部分检索的消息（MQGetMessageOptions 的选项字段中设置了 MQC.MQGMO_SYNCPOINT 标志）。

也可参阅下面的 ″backout″ 描述。

backout

```
public synchronized void backout()
```

掷出 MQException。

调用该方法是对队列管理器表明，所有自最近一次同步点以来发生的消息的获取和放入都被逆序恢复。作为工作单元一部分放入的消息（MQPutMessageOptions 的选项字段中设置了 MQC.MQPMO_SYNCPOINT 标志）被删除；作为工作单元一部分检索的消息（MQGetMessageOptions 的选项字段中设置了 MQC.MQGMO_SYNCPOINT 标志）在队列上逆序恢复。

另见上面 ″commit″ 中的描述。

accessQueue

```
public synchronized MQQueue accessQueue
                            (
                            String queueName, int openOptions,
                                   String queueManagerName,
                            String dynamicQueueName,
                            String alternateUserId
                            )
```

掷出 MQException。

建立至队列管理器上 MQSeries 队列的访问，以便获取或浏览消息、放入消息、查询队列属性或设置队列属性。

如果队列名称是模型队列，则创建动态的本地队列。可以通过检查 MQQueue 对象的 name 属性来确定已创建的队列的名称。

参数

queueName
要打开的队列名称。

openOptions
控制队列打开的选项。有效选项为:

MQC.MQOO_BROWSE
打开以浏览消息。

MQC.MQOO_INPUT_AS_Q_DEF
使用队列定义的缺省值打开以获取消息。

MQC.MQOO_INPUT_SHARED
打开以共享访问的方式来获取消息。

MQC.MQOO_INPUT_EXCLUSIVE
打开以互斥访问的方式来获取消息。

MQC.MQOO_OUTPUT
打开以放入消息。

MQC.MQOO_INQUIRE
打开以便查询 - 如果想查询特性则是必需的。

MQC.MQOO_SET
打开以设置属性。

MQC.MQOO_SAVE_ALL_CONTEXT
当检索消息 * 时，保存上下文。

MQC.MQOO_SET_IDENTITY_CONTEXT
允许设置标识上下文。

MQC.MQOO_SET_ALL_CONTEXT
允许设置所有上下文。

MQC.MQOO_ALTERNATE_USER_AUTHORITY
使用指定的用户标识验证。

MQC.MQOO_FAIL_IF_QUIESCING
如果队列管理器停顿，则失败。

MQC.MQOO_BIND_AS_QDEF
使用队列的缺省绑定。

MQC.MQOO_BIND_ON_OPEN
当打开队列时，将句柄绑定到目的地。

MQC.MQOO_BIND_NOT_FIXED
不要绑定到特定目的地。

MQC.MQOO_PASS_ALL_CONTEXT
允许传递所有上下文。

MQC.MQOO_PASS_IDENTITY_CONTEXT
允许标识要传递的上下文。

如果需要多个选项，可以使用按位“或”运算符添加或组合这些值。有关这些选项的完整描述，请参阅 MQSeries *MQSeries Application Programming Reference*。

queueManagerName
其上定义队列的队列管理器的名称。整个空白或空的名称表示 MQQueueManager 对象连接至的队列管理器。

dynamicQueueName
忽略该参数，除非 queueName 指定模型队列的名称。如果这样的话，该参数指定要创建的动态队列的名称。如果 queueName 指定模型队列的名称，则空白或空的名称是无效的。如果名称中的最后一个非空白字符是星号(*)，队列管理器会将该星号替换为管理队列（在队列管理器上是唯一的）产生的名称的字符串字符。

alternateUserId
如果在 openOptions 参数中指定了 MQOO_ALTERNATE_USER_AUTHORITY，则该参数指定将用于检查打开权限的替代用户标识。如果没有指定 MQOO_ALTERNATE_USER_AUTHORITY，则该参数为空白（或空）。

返回: 已成功打开的 MQQueue。

如果打开失败，掷出 MQException。

另见第145页的『"accessProcess"』。

accessQueue

```
public synchronized MQQueue accessQueue
                          (
                          String queueName,
                          int openOptions
                          )
```

如果从队列管理器断开后调用该方法，则掷出 MQException。

参数

queueName
: 要打开的队列名称

openOptions
: 控制队列打开的选项

有关参数的详细信息，请参阅第143页的『MQQueueManager.accessQueue』。

queueManagerName、*dynamicQueueName* 和 *alternateUserId* 被设置成 ""。

accessProcess

```
public synchronized MQProcess accessProcess
                        (
                        String processName,
                        int openOptions,
                        String queueManagerName,
                        String alternateUserId
                        )
```

掷出 MQException。

建立至该队列管理器上 MQSeries 进程的访问，以便查询进程属性。

参数

processName
: 要打开的进程名称。

openOptions
: 控制进程打开的选项。查询会自动地添加到指定的选项，所以不需要明确地指定它。

有效选项为:

MQC.MQOO_ALTERNATE_USER_AUTHORITY
: 使用指定的用户标识验证。

MQC.MQOO_FAIL_IF_QUIESCING
: 如果队列管理器停顿，则失败

如果需要多个选项，可以使用按位“或”运算符添加或组合这些值。有关这些选项的完整描述，请参阅 *MQSeries Application Programming Reference*。

queueManagerName
: 其上定义进程的队列管理器的名称。 应用程序应该使该参数为空白或空。

alternateUserId

如果在 openOptions 参数中指定了 MQOO_ALTERNATE_USER_AUTHORITY，则该参数指定将用于检查打开权限的替代用户标识。如果没有指定 MQOO_ALTERNATE_USER_AUTHORITY，则该参数为空白（或空）。

返回: 已成功打开的 MQProcess。

如果打开失败，掷出 MQException。

另见第143页的『MQQueueManager.accessQueue』。

accessProcess

这是以前描述过的 AccessProcess 方法的简化版本。

```
public synchronized MQProcess accessProcess
                          (
                         String processName,
                         int openOptions
                         )
```

这是以前描述过的 AccessQueue 方法的简化版本。

参数

processName

要打开的进程名称。

openOptions

控制进程打开的选项。

有关选项的详细信息，请参阅第145页的『"accessProcess"』。

queueManagerName 和 *alternateUserId* 被设置为 ""。

accessDistributionList

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems, int openOptions,
           String alternateUserId
          )
```

掷出 MQException。

参数

litems 在分发列表中包含的项目。

openOptions

控制分发列表打开的选项。

alternateUserId
如果在 openOptions 参数中指定了 MQOO_ALTERNATE_USER_AUTHORITY，则该参数指定将用于检查打开权限的替代用户标识。如果没有指定 MQOO_ALTERNATE_USER_AUTHORITY，则该参数为空白（或空）。

返回: 新创建的为放入操作而打开和就绪的 MQDistributionList。

如果打开失败，掷出 MQException。

另见第143页的『MQQueueManager.accessQueue』。

accessDistributionList

这是以前描述过的 AccessDistributionList 方法的简化版本。

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems,
           int openOptions,
          )
```

参数

litems 要包含进分发列表的项目。

openOptions
控制分发列表打开的选项。

有关参数的详细信息，请参阅第146页的『accessDistributionList』。

alternateUserId 被设置为 ""。

begin*（仅绑定连接）

```
public synchronized void begin()
```

掷出 MQException。

只有使用绑定方式的 MQSeries classes for Java 才支持此方式，并且它将向队列管理器发信号给正在启动新的工作单元。

不要在使用本地一阶段事务的应用程序中使用此方法。

isConnected

```
public boolean isConnected()
```

返回 isConnected 变量值。

MQSimpleConnectionManager

```
java.lang.Object      com.ibm.mq.MQConnectionManager
   │                          │
   └─ com.ibm.mq.MQSimpleConnectionManager
```

public class **MQSimpleConnectionManager**
implements **MQConnectionManager**（参见第 152 页。）

MQSimpleConnectionManager 提供了基本连接合用功能。 MQSimpleConnectionManager 可用作缺省的“连接管理器”，或作为 MQQueueManager 构造器的参数。在构造 MQQueueManager 时，将使用池中最近使用的连接。

当连接在一个指定周期内没有被使用或是池中不使用的连接数超出了指定的数目，则摧毁它（通过独立的线程）。可以指定超时周期以及最大不使用连接数。

变量

MODE_AUTO

`public static final int MODE_AUTO`。请参阅『setActive』。

MODE_ACTIVE

`public static final int MODE_ACTIVE`。请参阅『setActive』。

MODE_INACTIVE

`public static final int MODE_INACTIVE`。请参阅『setActive』。

构造器

MQSimpleConnectionManager

`public MQSimpleConnectionManager()`

构造一个 MQSimpleConnectionManager。

方法

setActive

`public void setActive(int mode)`

设置连接池的活动方式.

参数

mode 连接池必需的活动方式。有效值有:

MODE_AUTO

当“连接管理器”是缺省的“连接管理器”并且 MQEnvironment 含有的 MQPoolTokens 集中至少有一个令牌时，连接池活动。这是缺省方式。

MODE_ACTIVE

连接池一直是活动的。当调用 MQQueueManager.disconnect() 时，基本连接是合用的，同时也是 MQQueueManager 对象构造

的下一次潜在可用的。如果连接不被使用的时间超过了“超时”时间段，或是池的大小超过了 HighThreshold，将摧毁连接。

MODE_INACTIVE

连接池总是不活动的。当进入这种方式时，将清除连接到 MQSeries 的池。当调用 MQQueueManager.disconnect() 时，将根据任何活动 MQQueueManager 对象来摧毁连接。

getActive

```
public int getActive()
```

获取连接池的方式.

返回:

当前连接池的活动方式，返回的值可以是（见第148页的『setActive』）:

MODE_AUTO

MODE_ACTIVE

MODE_INACTIVE

setTimeout

```
public void setTimeout(long timeout)
```

设置 Timeout 值，其中持续这段时间不用的连接将由单独的线程摧毁。

参数

timeout Timeout 值，单位毫秒。

getTimeout

```
public long getTimeout()
```

返回“超时”值。

setHighThreshold

```
public void setHighThreshold(int threshold)
```

设置 HighThreshold。如果池中不使用的连接数超过了该值，将摧毁池中最旧的不使用连接。

参数

threshold
池中最大不使用连接数。

getHighThreshold

```
public int getHighThreshold ()
```

返回 HighThreshold 值。

MQC

public interface **MQC**
extends **Object**

MQC 接口定义了 MQ Java 编程接口使用的所有常量（除完成码常量和错误代码常量）。要从程序参阅这些常量之一，请在常量名称前加前缀 ″MQC.″。例如，可以如下为队列设置关闭选项:

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // 在队列
                                      // 关闭时,
                                      // 删除它。
 ...
```

MQSeries Application Programming Reference 中提供了这些常量的全部信息。

在 MQException 类中定义了完成码与错误码常量。请参阅第91页的『MQException』。

MQPoolServicesEventListener

public interface **MQPoolServicesEventListener**
extends **Object**

注: 通常，应用程序不使用该接口。

MQPoolServicesEventListener 是 ConnectionManager 供应商实现的。使用 MQPoolService 对象注册 MQPoolServicesEventListener 时，任何时候将 MQPoolToken 添加到 MQEnvironment 管理的 MQPoolToken 集合还是从中除去，事件侦听器都将接收到事件。任何时候更改缺省 ConnectionManager，它也将接收到事件。

另见第121页的『MQPoolServices』和第122页的『MQPoolServicesEvent』。

方法

tokenAdded

```
public void tokenAdded(MQPoolServicesEvent event)
```

将 MQPoolToken 添加到集合时调用。

tokenRemoved

```
public void tokenRemoved(MQPoolServicesEvent event)
```

从集合中除去 MQPoolToken 时调用。

defaultConnectionManagerChanged

```
public void defaultConnectionManagerChanged(MQPoolServicesEvent event)
```

设置缺省 ConnectionManager 时调用。将已经清除 MQPoolToken 集合。

MQConnectionManager

这是一个专用接口，不能由应用程序来实现。MQSeries classes for Java 为这个接口 (MQSimpleConnectionManager) 提供了一个实现，可以在 MQQueueManager 构造器上指定它，或通过 MQEnvironment.setDefaultConnectionManager 来指定。

请参阅第148页的『MQSimpleConnectionManager』。

希望提供它们自己的 ConnectionManager 的应用程序或中间件应实现 javax.resource.spi.ConnectionManager。这就需要安装带有 JAAS 1.0 的 Java 2 v1.3。

MQReceiveExit

public interface **MQReceiveExit**
extends **Object**

接收出口接口允许您通过 MQSeries classes for Java 检查和改变从队列管理器接收到的数据。

注: 当以绑定方式直接连接到 MQSeries 时，此接口不适用。

要提供您自己的接收出口，请定义实现该接口的类。创建该类的新的实例，并在构造 MQQueueManager 对象前将 MQEnvironment.receiveExit 变量赋给它。例如:

```
// 在 MyReceiveExit.java 中
class MyReceiveExit implements MQReceiveExit {
  // 必须提供 receiveExit
  // 方法的一个实现
  public byte[] receiveExit(
   MQChannelExit       channelExitParms,
   MQChannelDefinition channelDefinition,
   byte[]              agentBuffer)
  {
    // 这里放出口码...
  }
}
// 在主程序中...
MQEnvironment.receiveExit = new MyReceiveExit();
 ...    // 其它初始化
MQQueueManager qMgr        = new MQQueueManager("");
```

方法

receiveExit

```
public abstract byte[] receiveExit(MQChannelExit channelExitParms,
                           byte agentBuffer[])
```

类必须提供的接收出口方法。无论何时 MQSeries classes for Java 从队列管理器接收数据时，都将调用该方法。

参数

channelExitParms
包含有关正被调用的出口的上下文的信息。 exitResponse 成员变量是用来告诉 MQSeries classes for Java 下一步采取什么操作的输出参数。详细信息，请参阅第80页的『MQChannelExit』。

channelDefinition
包含通道细节，所有的通信都通过该通道与队列管理器联系。

agentBuffer
如果 channelExitParms.exitReason 是 MQChannelExit.MQXR_XMIT，则 agentBuffer 包含从队列管理器接收到的数据，否则 agentBuffer 为空。

返回:

如果设置了出口响应代码（在 channelExitParms 中）以便 MQSeries classes for Java 现在能处理数据 (MQXCC_OK)，则接收出口方法必须返回要处理的数据。所以，最简单的接收出口由单行 ″return agentBuffer;″ 组成。

另见:
- 第150页的『MQC』
- 第78页的『MQChannelDefinition』

MQSecurityExit

public interface **MQSecurityExit**
extends **Object**

安全性出口接口允许您定制当试图连接到队列管理器时发生的安全性流。

注: 当以绑定方式直接连接到 MQSeries 时，此接口不适用。

要提供您自己的安全性出口，请定义实现该接口的类。创建该类的新的实例，并在构造 MQQueueManager 对象前将 MQEnvironment.securityExit 变量赋给它。例如:

```
// 在 MySecurityExit.java 中
class MySecurityExit implements MQSecurityExit {
  // 必须提供 securityExit
  // 方法的一个实现
  public byte[] securityExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // 这里放出口码...
  }
}
// 在主程序中...
MQEnvironment.securityExit = new MySecurityExit();
 ...    // 其它初始化
MQQueueManager qMgr       = new MQQueueManager("");
```

方法

securityExit

```
public abstract byte[] securityExit(MQChannelExit channelExitParms,
                                    byte agentBuffer[])
```

类必须提供的安全性出口方法。

参数

channelExitParms
包含有关正被调用的出口的上下文的信息。 exitResponse 成员变量是用来告诉 MQSeries 客户机 Java 版下一步采取什么操作的输出参数。详细信息，请参阅第80页的『MQChannelExit』.

channelDefinition
包含通道细节，所有的通信都通过该通道与队列管理器联系。

agentBuffer
如果 channelExitParms.exitReason 是 MQChannelExit.MQXR_SEC_MSG, 则 agentBuffer 包含从队列管理器接收到的安全性消息，否则 agentBuffer 为空。

返回:

如果设置了出口响应代码（在 cha nnelExitParms 中）以便消息能被发送到队列管理器，则安全性出口方法必须返回要发送的数据。

另见:

- 第150页的『MQC』
- 第78页的『MQChannelDefinition』

MQSendExit

public interface **MQSendExit**
extends **Object**

发送出口接口允许您通过 MQSeries 客户机 Java 版检查和改变发送到队列管理器的数据。

注: 当以绑定方式直接连接到 MQSeries 时，此接口不适用。

要提供您自己的发送出口，请定义实现该接口的类。创建该类的新的实例，并在构造 MQQueueManager 对象前将 MQEnvironment.sendExit 变量赋给它。例如:

```
// 在 MySendExit.java 中
class MySendExit implements MQSendExit {
  // 必须提供 sendExit 方法的一个实现
  public byte[] sendExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // 这里放出口码...
  }
}
// 在主程序中...
MQEnvironment.sendExit = new MySendExit();
 ...    // 其它初始化
MQQueueManager qMgr       = new MQQueueManager("");
```

方法

sendExit

```
public abstract byte[] sendExit(MQChannelExit channelExitParms,
                                byte agentBuffer[])
```

类必须提供的发送出口方法。无论 MQSeries classes for Java 何时希望将一些数据发送给队列管理器时，都调用这个方法。

Parameters

channelExitParms
包含有关正被调用的出口的上下文的信息。 exitResponse 成员变量是用来告诉 MQSeries classes for Java 下一步采取什么操作的输出参数。详细信息，请参阅第80页的『MQChannelExit』。

channelDefinition
包含通道细节，所有的通信都通过该通道与队列管理器联系。

agentBuffer
如果 channelExitParms.exitReason 是 MQChannelExit.MQXR_XMIT，则 agentBuffer 包含要发送到队列管理器的数据，否则 agentBuffer 为空。

返回:

如果设置了出口响应代码（在 channelExitParms 中）以便消息能被发送到队列管理器(MQXCC_OK)，则发送出口方法必须返回要发送的数据。所以，最简单的发送出口由行 ″return agentBuffer;″ 组成。

另见:

- 第150页的『MQC』
- 第78页的『MQChannelDefinition』

ManagedConnection

public interface **javax.resource.spi.ManagedConnection**

注: 通常，应用程序不使用这个类；打算由 ConnectionManager 的实现使用。

MQSeries classes for Java 提供了从 ManagedConnectionFactory.createManagedConnection 返回的 ManagedConnection 的实现。此对象表示了至 MQSeries“队列管理器”的连接。

方法

getConnection

```
public Object getConnection(javax.security.auth.Subject subject,
                            ConnectionRequestInfo cxRequestInfo)
```

掷出 ResourceException。

为 ManagedConnection 对象所表示的物理连接创建新的连接句柄。对于 MQSeries classes for Java，将返回 MQQueueManager 对象。ConnectionManager 通常返回来自 allocateConnection 的该对象。

忽略 subject 参数。如果 cxRequestInfo 参数不适用，则掷出 ResourceException。每一个 ManagedConnection 都能同时使用多个连接句柄。

destroy

```
public void destroy()
```

掷出 ResourceException。

摧毁至 MQSeries“队列管理器”的物理连接。任何暂挂的本地事务被提交。更详细信息，请参阅第160页的『getLocalTransaction』。

cleanup

```
public void cleanup()
```

掷出 ResourceException。

关闭所有打开的连接句柄，并把物理连接复位成准备合用的初始状态。任何暂挂的本地事务都被复原。更详细信息，请参阅第160页的『getLocalTransaction』。

associateConnection

```
public void associateConnection(Object connection)
```

掷出 ResourceException。

MQSeries classes for Java 当前不支持这个方法。掷出一个 javax.resource.NotSupportedException。

ManagedConnection

addConnectionEventListener

```
public void addConnectionEventListener(ConnectionEventListener listener)
```

将 ConnectionEventListener 添加到 ManagedConnection 实例。

如果在 ManagedConnection 上发生错误，或者当在与该 ManagedConnection 关联的连接句柄上调用 MQQueueManager.disconnect() 时，将通知侦听器。关于本地事务事件将不通知侦听器（请参阅『getLocalTransaction』）。

removeConnectionEventListener

```
public void removeConnectionEventListener(ConnectionEventListener listener)
```

删除注册的 ConnectionEventListener。

getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

掷出 ResourceException。

MQSeries classes for Java 当前不支持这个方法。掷出一个 javax.resource.NotSupportedException。

getLocalTransaction

```
public LocalTransaction getLocalTransaction()
```

MQSeries classes for Java 当前不支持这个方法。掷出一个 javax.resource.NotSupportedException。

当前，ConnectionManager 不管理 MQSeries 本地事务，并且与本地事务相关的事件不通知注册的 ConnectionEventListeners。当 cleanup() 发生时，任何正在运行的工作单元都将复原。当 destroy() 发生时，将提交任何正在运行的工作单元。

现有的 API 行为是在 MQQueueManager.disconnect() 上提交了一个正在运行的工作单元。这个现有的行为只有在 MQConnectionManager（而非 ConnectionManager）管理连接时才被保留。

getMetaData

```
public ManagedConnectionMetaData getMetaData()
```

掷出 ResourceException。

获取基本“队列管理器”的元数据信息。请参阅第164页的『ManagedConnectionMetaData』。

setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

掷出 ResourceException。

为该 ManagedConnection 设置日志写入器。在创建 ManagedConnection 时，它将从 ManagedConnectionFactory 继承日志写入器。

MQSeries classes for Java 当前不支持日志写入器的使用。有关记录的更详细信息，请参阅第91页的『MQException.log』。

getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

掷出 ResourceException。

为该 ManagedConnection 返回日志写入器。

MQSeries classes for Java 当前不支持日志写入器的使用。有关记录的更详细信息，请参阅第91页的『MQException.log』。

ManagedConnectionFactory

public interface **javax.resource.spi.ManagedConnectionFactory**

注: 通常，应用程序不使用这个类。

MQSeries classes for Java 提供了至 ConnectionManagers 的接口的实现。ManagedConnectionFactory 用于构造 ManagedConnections，并用于从候选集中选出合适的 ManagedConnections。有关这个接口的详细信息，请参见"J2EE 连接器体系结构规范"（参考 Sun 的 Web 站点 `http://java.sun.com`）。

方法

createConnectionFactory

```
public Object createConnectionFactory()
```

掷出 ResourceException。

MQSeries classes for Java 当前不支持 createConnectionFactory 方法。这个方法将产生 javax.resource.NotSupportedException。

createConnectionFactory

```
public Object createConnectionFactory(ConnectionManager cxManager)
```

掷出 ResourceException。

MQSeries classes for Java 当前不支持 createConnectionFactory 方法。这个方法将掷出 javax.resource.NotSupportedException。

createManagedConnection

```
public ManagedConnection createManagedConnection
                                (javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

掷出 ResourceException。

创建一个新的至"MQSeries 队列管理器"的物理连接，并返回表示此连接的 ManagedConnection 对象。MQSeries 将忽略 subject 参数。

matchManagedConnection

```
public ManagedConnection matchManagedConnection
                                (java.util.Set connectionSet,
                                 javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

掷出 ResourceException。

为适当的 ManagedConnection 搜索提供的 ManagedConnections 候选集。返回空，或符合连接标准的集合中适用的 ManagedConnection。

setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

掷出 ResourceException。

为该 ManagedConnectionFactory 设置日志写入器。在创建 ManagedConnection 时，它将从 ManagedConnectionFactory 继承日志写入器。

MQSeries classes for Java 当前不支持日志写入器的使用。有关记录的更详细信息，请参见第91页的『MQException.log』。

getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

掷出 ResourceException。

为该 ManagedConnectionFactory 返回日志写入器。

MQSeries classes for Java 当前不支持日志写入器的使用。有关记录的更详细信息，请参见第91页的『MQException.log』。

hashCode

```
public int hashCode()
```

返回这个 ManagedConnectionFactory 的 hashCode。

equals

```
public boolean equals(Object other)
```

检查该 ManagedConnectionFactory 是否和另一个 ManagedConnectionFactory 相同。如果这两个 ManagedConnectionFactories 都描述了同一个目标“队列管理器”，则返回 true。

ManagedConnectionMetaData

public interface **javax.resource.spi.ManagedConnectionMetaData**

注: 通常，应用程序不使用这个类；打算由 ConnectionManager 的实现使用。

ConnectionManager 可使用这个类来检索与连接到“队列管理器”的基本物理连接相关的元数据。这个类的实现从 ManagedConnection.getMetaData() 返回。

方法

getEISProductName

```
public String getEISProductName()
```

掷出 ResourceException。

返回“IBM MQSeries”。

getProductVersion

```
public String getProductVersion()
```

掷出 ResourceException。

返回描述 ManagedConnection 所连接的 MQSeries“队列管理器”的命令级别。

getMaxConnections

```
public int getMaxConnections()
```

掷出 ResourceException。

返回 0。

getUserName

```
public String getUserName()
```

掷出 ResourceException。

如果 ManagedConnection 表示一个到“队列管理器”的“客户机”，则返回该连接使用的用户标识。否则，返回空字符串。

第3部分 用 MQ JMS 进行编程

第10章 编写 MQ JMS 程序 167
JMS 模型. 167
构建连接 168
从 JNDI 检索工厂 168
使用工厂创建连接 169
在运行时创建工厂 169
启动连接 169
选择客户机或绑定传输 170
获取会话 170
发送消息 171
使用 'set' 方法设置特性 172
消息类型 173
接收消息 173
消息选择器 174
异步传递 175
关闭 175
关闭时“Java 虚拟机”挂起. 175
处理错误 175
异常侦听器 176

第11章 编写“发布/订阅”应用程序 177
编写简单的“发布/订阅”应用程序 177
导入需要的包 177
获取或创建 JMS 对象 177
发布消息 179
接收订阅 179
关闭不想要的资源 179
使用主题 179
主题名称 179
在运行时创建主题 180
订户选项 181
创建非长期订户. 182
创建长期订户 182
使用消息选择器. 182
抑制本地出版物. 182
合并订户选项 183
配置基本订户队列 183
缺省配置 183
配置非长期订户. 183
配置长期订户 184
对于长期订户的重新创建和迁移问题 185
解决发布/订阅问题 185
非完整“发布/订阅”关闭 185
订户清除实用程序 185
处理代理报告 186

第12章 JMS 消息. 187
消息选择器 187
将 JMS 消息映射到 MQSeries 消息 191
MQRFH2 头. 192
带有相应 MQMD 字段的 JMS 字段和特性. . . 195
将 JMS 字段映射到 MQSeries 字段(外出消息) 196
在 send()/publish() 映射 JMS 头 197
映射 JMS 特性字段 198
映射 JMS 供应商特定的字段 199
将 MQSeries 字段映射到 JMS 字段(进入消息) 200
将 JMS 映射到本地 MQSeries 应用程序 . . . 201
消息主体 202

第13章 MQ JMS 应用程序服务器设施. 205
ASF 类和函数 205
ConnectionConsumer 205
规划应用程序 206
点到点消息传递的常规规则 206
发布/订阅消息传递的常规原则 207
处理有害消息 207
从队列中除去消息 208
错误处理 209
从错误情况恢复. 209
原因和反馈代码. 210
应用程序服务器样本代码. 211
MyServerSession.java 213
MyServerSessionPool.java 213
MessageListenerFactory.java 214
ASF 使用示例 214
Load1.java 215
CountingMessageListenerFactory.java 216
ASFClient1.java 216
Load2.java 218
LoggingMessageListenerFactory.java 218
ASFClient2.java 218
TopicLoad.java 219
ASFClient3.java 220
ASFClient4.java 220

第14章 JMS 接口与类 223
Sun Java Message Service 类和接口 223
MQSeries JMS 类 225
BytesMessage 227
方法 227
Connection 235
方法 235
ConnectionConsumer 238
方法 238
ConnectionFactory 239
MQSeries 构造器 239
方法 239
ConnectionMetaData 243
MQSeries 构造器 243
方法 243
DeliveryMode 245
字段 245

© Copyright IBM Corp. 1997, 2001

Destination 246
MQSeries 构造器 246
方法 246
ExceptionListener 248
方法 248
MapMessage 249
方法 249
Message 257
字段 257
方法 257
MessageConsumer 270
方法 270
MessageListener 272
方法 272
MessageProducer 273
MQSeries 构造器 273
方法 273
MQQueueEnumeration * 277
方法 277
ObjectMessage 278
方法 278
Queue 279
MQSeries 构造器 279
方法 279
QueueBrowser 281
方法 281
QueueConnection 283
方法 283
QueueConnectionFactory 285
MQSeries 构造器 285
方法 285
QueueReceiver 287
方法 287
QueueRequestor 288
构造器. 288
方法 288
QueueSender 290
方法 290
QueueSession. 293
方法 293
Session. 296
字段 296
方法 296
StreamMessage 301
方法 301
TemporaryQueue. 309
方法 309
TemporaryTopic 310
MQSeries 构造器 310
方法 310
TextMessage 311
方法 311
Topic 312
MQSeries 构造器 312
方法 312
TopicConnection. 314
方法 314
TopicConnectionFactory 316
MQSeries 构造器 316
方法 316
TopicPublisher 319
方法 319
TopicRequestor 322
构造器. 322
方法 322
TopicSession 323
MQSeries 构造器 323
方法 323
TopicSubscriber 327
方法 327
| XAConnection 328
| XAConnectionFactory 329
| XAQueueConnection 330
| 方法 330
| XAQueueConnectionFactory 331
| 方法 331
| XAQueueSession 333
| 方法 333
| XASession 334
| 方法 334
| XATopicConnection. 336
| 方法 336
| XATopicConnectionFactory 337
| 方法 337
| XATopicSession 339
| 方法 339

第10章 编写 MQ JMS 程序

本章提供了有助于编写 MQ JMS 应用程序的信息。它提供了 JMS 模型的简要介绍以及应用程序编程可能要执行的某些公共任务。

JMS 模型

JMS 定义了消息传递服务的类属视图。重要的是理解该视图，以及它是如何映射到基本 MQSeries 传输上的。

类属 JMS 模型基于在 Sun 的 `javax.jms` 包中定义的下列接口:

Connection
: 提供访问基本传输并用于创建**Session**。

Session
: 提供生成和使用消息的环境，包括用于创建 **MessageProducer** 和 **MessageConsumer** 的方法。

MessageProducer
: 用于发送消息。

MessageConsumer
: 用于接收消息。

注意，**Connection** 是线程安全，但**Session**、 **MessageProducer** 和 **MessageConsumer** 不是。建议策略是每个应用程序线程使用一个 Session。

在 MQSeries 术语中:

Connection
: 提供临时队列的作用域。它还提供了存储控制如何连接 MQSeries 参数的空间。如果使用 MQSeries Java 客户机连接，则这些参数的示例就是队列管理器名称和远程主机名称。

Session
: 包含定义事务性作用域的 `HCONN`。

MessageProducer 和 MessageConsumer
: 包含定义写入或读出的特定队列的 `HOBJ`。

注意正常的 MQSeries 规则应用于:

- 在任何给定时间每个 HCONN 只能处理一个单一操作。因此不能并行调用与 Session 关联的 MessageProducer 或 MessageConsumer。这与每个 Session 只能并行处理一个单一线程的 JMS 限制一致。
- PUT 可使用远程队列，但 GET 只能应用于本地队列管理器上的队列。

类属 JMS 接口可归属于“点到点”和“Publish/Subscribe”行为的更特定版本的子类。

点到点版本是:

- QueueConnection

© Copyright IBM Corp. 1997, 2001

- QueueSession
- QueueSender
- QueueReceiver

JMS 的关键思想是，可能并且强烈建议，编写仅使用 javax.jms 中接口引用的应用程序。下列实现中封装了所有供应商指定的信息:

- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

这些称为“受管理对象”，即可使用供应商提供的管理工具构建并存储在 JNDI 名称空间的对象。JMS 应用程序可从名称空间中检索这些对象并使用它们而无需知道提供该实现的供应商。

构建连接

连接不是直接创建的，而是使用连接工厂建立的。可在 JNDI 名称空间中存储工厂对象，这样可隔离 JMS 应用程序与供应商指定信息。关于创建和存储对象的详细信息，请参阅第29页的『第5章 使用 MQ JMS 管理工具』。

如果没有可用的 JNDI 名称空间，请参阅第169页的『在运行时创建工厂』。

从 JNDI 检索工厂

从 JNDI 名称空间中检索对象，必须如从 IVTRun 样本文件取出的该片段所示设置初始环境:

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

其中:

icf 定义初始环境的工厂类

url 定义环境特定 URL

关于 JNDI 用法的详细信息，请参阅 Sun JNDI 文档。

注: 某些 JNDI 包和 LDAP 服务供应商的组合可能导致 LDAP 错误 84。要解决该问题，请在调用 InitialDirContext 前插入下列行。

```
environment.put(Context.REFERRAL, "throw");
```

一旦获得了初始环境，则使用 lookup() 方法从名称空间中检索对象。下列代码从基于 LDAP 的名称空间中检索名为 ivtQCF 的 QueueConnectionFactory。

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

使用工厂创建连接

使用该工厂对象上的 createQueueConnection() 方法来创建 Connection，如下列代码所示:

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

在运行时创建工厂

如果 JNDI 名称空间不可用，则可能在运行时创建工厂对象。然而，使用该方法减少了 JMS 应用程序的可移植性，因为它需要 MQSeries 特定类的引用。

下列代码创建带有全部缺省设置的 QueueConnectionFactory:

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

（如果导入 com.ibm.mq.jms 包，则可以省略 com.ibm.mq.jms. 前缀。）

从以上工厂创建的连接使用 Java 绑定来连接本地机器上的缺省队列管理器。表14中显示的 set 方法可用来定制带有 MQSeries 特定信息的工厂。

启动连接

JMS 规范定义了应该在“停止”状态中创建的连接。直到启动连接后，与连接关联的 MessageConsumer 才能接收消息。要启动连接，请发出下列命令:

```
connection.start();
```

表 14. MQQueueConnectionFactory 上的设置方法

方法	描述
setCCSID(int)	用来设置 MQEnvironment.CCSID 属性
setChannel(String)	客户机连接的通道名称
setHostName(String)	客户机连接的主机名称
setPort(int)	客户机连接端口
setQueueManager(String)	队列管理器名称
setTemporaryModel(String)	用来生成作为 QueueSession.createTemporaryQueue() 调用结果的临时目的地的模型队列名称。建议这是临时动态队列名称而不是永久动态队列名称。
setTransportType(int)	指定如何连接 MQSeries。当前可用的选项是: • JMSC.MQJMS_TP_BINDINGS_MQ （缺省值） • JMSC.MQJMS_TP_CLIENT_MQ_TCPIP。 JMSC 在包 com.ibm.mq.jms 中

表 14. MQQueueConnectionFactory 上的设置方法 (续)

方法	描述
setReceiveExit(String) setSecurityExit(String) setSendExit(String) setReceiveExitInit(String) setSecutityExitInit(String) setSendExitInit(String)	这些方法的存在允许使用基本 MQSeries Classes for Java 的基本 MQSeries 类”提供的发送、接收以及安全性出口。set*Exit 方法采用实现相关出口方法的类名称。(请参阅 MQSeries 5.1 产品文档以获取详细信息。) 该类还必须实现带有单个 String 参数的构造器。该字符串提供任何出口可能需要的初始化数据，并设置成相应的 set*ExitInit 方法中提供的值。

选择客户机或绑定传输

MQ JMS 可使用客户机或绑定传输来与 MQSeries 通信。如果使用 Java 绑定，JMS 应用程序和 MQSeries 队列管理器必须在相同的机器上。如果使用客户机，则队列管理器可能在与应用程序在不同的机器上。

连接工厂对象的内容确定要使用哪一个传输。 第29页的『第5章 使用 MQ JMS 管理工具』描述如何定义用于客户机或绑定传输的工厂对象。

下列代码段说明如何定义应用程序中的传输:

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

获取会话

一旦构建连接，则使用 QueueConnection 上的 createQueueSession 方法来获取会话。

该方法采用两个参数:

1. 定义会话是 ’transacted’ 还是 ’non-transacted’ 的 boolean。
2. 确定 ’acknowledge’ 方式的参数。

最简单的情况是带有 AUTO_ACKNOWLEDGE 的 'non-transacted'，如下列代码段所示:

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

注: 连接是线程安全，但会话（及从他们创建的对象）不是。推荐的多线程应用程序实践是每个线程使用独立的会话。

发送消息

使用 MessageProducer 发送消息。对于点到点这是用 QueueSession 上的 createSender 方法创建的 QueueSender。QueueSender 通常是为特定队列创建的，因此将使用该发送器发送的所有消息都发送到相同的目的地。目的地是使用 Queue 对象指定的。队列对象可在运行时创建或者构建并存储在 JNDI 名称空间中。

用下列方法从 JNDI 检索对象:

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

MQ JMS 在 com.ibm.mq.jms.MQQueue 中提供了 Queue 的实现。它包含控制 MQSeries 特定行为细节的特性，但在大多数情况下可能使用缺省值。JMS 定义了目的地的标准方式，它使应用程序中的 MQSeries 特定代码最小化。该机制使用 QueueSession.createQueue 方法，它采用描述目的地的字符串参数。该字符串仍为供应商指定格式，但这是比直接引用供应商类要灵活的方法。

MQ JMS 接受两种 createQueue() 的字符串参数格式。

- 第一种是 MQSeries 队列名称，如从 samples 目录中 IVTRun 程序取出的下列代码片段所示:

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- 第二种功能更强的格式是以“统一资源标识”(URI) 为基础的。该格式允许指定远程队列（不是您连接的队列管理器上的队列）。它还允许您设置 com.ibm.mq.jms.MQQueue 对象中包含的其它特性。

  队列的 URI 以序列 queue:// 开头，带有驻留队列的队列管理器名称。它带有另一个 '/'、队列名称和可选的设置剩余 Queue 特性的名值对列表。例如，上列的 URI 等价形式是:

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

  注意省略了队列管理器名称。将它解释成使用 Queue 对象时本身的 QueueConnection 连接的队列管理器。

下列示例连接到队列管理器 'HOST1.QM1' 上的队列 'Q1' 并导致将所有消息作为非持续消息发送并具有优先级 5:

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

第172页的表15列出了可在 URL 的名称值部分使用的名称。这一格式的缺点是它不支持值的符号名，所以在适当的地方该表还表示“特殊”值。注意可能要更改这些特殊

值。(请参阅『使用 'set' 方法设置特性』以获取设置特性的替代方法。)

表 15. 队列 URI 的特性名

特性	描述	值
expiry	消息存活时间以秒计算。	0 表示无限，正整数表示超时 (ms)
priority	消息优先级	0 到 9，-1=QDEF，-2=APP
persistence	是否应该将消息“固化”到磁盘上	1=non-persistent，2=persistent，-1=QDEF，-2=APP
CCSID	目的地字符集	整数-基本 MQSeries 文档中列出的有效值
targetClient	接收的应用程序是否适应 JMS	0=JMS，1=MQ
encoding	如何表示数值字段	基本 MQSeries 文档中描述的整数值

QDEF - 表示应该由 MQSeries 队列配置确定的特性的特殊值。

APP - 表示 JMS 应用程序可以控制这个特性的特殊值

一旦获得 Queue 对象(使用上述 createQueue 或从 JNDI)，则它必须传递到 createSender 中以创建 QueueSender:

```
QueueSender queueSender = session.createSender(ioQueue);
```

使用生成的 queueSender 对象通过 send 方法发送消息:

```
queueSender.send(outMessage);
```

使用 'set' 方法设置特性

可通过首先使用缺省构造器创建 com.ibm.mq.jms.MQQueue 实例来设置 Queue 特性。然后可使用公共设置方法填充需要的值。这个方法表示可以对于特性值使用符号名。但是因为该值是供应商指定的且嵌在代码中，所以降低了应用程序可移植性。

下列代码段显示了带有 set 方法的队列特性集合。

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

表16 显示了提供给 MQ JMS 以使用 set 方法的符号特性值。

表 16. 队列特性的符号值

特性	管理工具关键字	值
expiry	UNLIM APP	JMSC.MQJMS_EXP_UNLIMITED JMSC.MQJMS_EXP_APP
priority	APP QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
persistence	APP QDEF PERS NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON

表 16. 队列特性的符号值（续）

特性	管理工具关键字	值
targetClient	JMS MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
encoding	Integer(N) Integer(R) Decimal(N) Decimal(R) Float(N) Float(R) Native	JMSC.MQJMS_ENCODING_INTEGER_NORMAL JMSC.MQJMS_ENCODING_INTEGER_REVERSED JMSC.MQJMS_ENCODING_DECIMAL_NORMAL JMSC.MQJMS_ENCODING_DECIMAL_REVERSED JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED JMSC.MQJMS_ENCODING_NATIVE

请参阅第39页的『ENCODING 特性』以获取关于编码的讨论。

消息类型

JMS 提供了几种消息类型，它们每个均包含其本身内容的一些信息。为避免引用供应商特定的该消息的类名，在 Session 对象上提供了创建消息的方法。

在样本程序中，用下列方式创建文本消息:

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

可使用的信息类型是:
- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

关于这些类型的详细信息在第223页的『第14章 JMS 接口与类』中。

接收消息

使用 QueueReceiver 接收消息。这是使用 `createReceiver()` 方法从 Session 中创建的。该方法采用定义从何处接收消息的 Queue 参数。参阅第171页的『发送消息』以获取如何创建 Queue 对象的详细信息。

样本程序用下列代码创建接收方并读回测试消息:

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

接收调用中的参数是以毫秒计算的超时。该参数定义如果没有立即可用的消息时该方法应该等待的时间。可省略该参数，在这种情况下调用无限地阻塞。如果不希望任何延迟，请使用 `receiveNoWait()` 方法。

该接收方法返回适当类型的消息。例如如果将 `TextMessage` 放置在队列上，接收到消息时返回的对象是 `TextMessage` 实例。

要从消息主体中抽取内容，有必要从类属 Message 类（它是接收方法的说明的返回类型）强制转换成更具体的子类，例如 TextMessage。如果不知道接收的消息类型，可使用 ’instanceof’ 运算符确定它的类型。好的作法是在强制类型转换前总是测试消息类，这样就可适当处理意外错误。

下列代码说明 ’instanceof’ 的使用以及从 TextMessage 抽取的内容:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

消息选择器

JMS 提供了选择队列上的消息子集的机制，因此接收调用就可返回该子集。创建 QueueReceiver 时提供一个包含确定要检索哪些消息的 SQL（结构化查询语言）表达式的字符串。选择器可以是指 JMS 消息头中的字段和消息特性中的字段（这些是有效地应用程序指定的头字段）。有关头字段名和 SQL 选择器语法的详细信息在第187页的『第12章 JMS 消息』中。

下列示例显示如何选择用户定义的特性名称 myProp:

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

注: JMS 规范不允许更改与接收方关联的选择器。一旦创建了选择器，选择器就固定了接收方的存活时间。这意味着如果需要不同的选择器，则必须创建新的接收方。

异步传递

调用 `QueueReceiver.receive()` 的替代方法是注册一个当有适当消息可用时自动调用的方法。下列代码片段说明了该机制:

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // 当消息不可用时, JMS 将调用
  // 的方法。
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // 这里是应用程序的特定处理
    .
    .
    .
  }
}

 .
 .
 .
  // 在“主”程序（或另一些类）中
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // 主程序现在可以使用其它应用程序特定行为
  // 继续执行。
```

注: 使用异步传递的 QueueReceiver 将整个 Session 标记为异步。显式调用与使用异步传递的 Session 关联的 QueueReceiver `receive` 方法将产生错误。

关闭

无用信息收集本身不能及时释放所有 MQSeries 资源。这在应用程序需要创建许多 Session 或更低级别的短存活时间 JMS 对象时更加明显。因此当不再需要资源时调用各种类（QueueConnection、QueueSession、 QueueSender 和 QueueReceiver）的 `close()` 方法是很重要的。

关闭时“Java 虚拟机”挂起

如果 MQ JMS 应用程序在未调用 `Connection.close()` 的情况下完成，则将中止某些 JVM。如果出现该问题，可编辑应用程序来包含 `Connection.close()` 调用或使用 `Ctrl-C` 键终止 JVM。

处理错误

将通过异常来报告 JMS 应用程序中的任何运行时错误。JMS 中的大多数方法将产生掷出错误的 JMSException。捕捉这些异常并将它们显示到合适的输出是一种良好的编程习惯。

与正常“Java 异常”不同，JMSException 可能包含一个嵌入其中的更深层异常。对于 JMS，这可能是从基本传输重要详细信息的有效方式。在 MQ JMS 中，MQSeries 生成 MQException 时通常将该异常作为嵌入的异常包含在 JMSException 中。

JMSException 的实现在其 toString() 方法输出中不包含嵌入异常。因此，有必要显式检查嵌入的异常并打印，如下列代码片段所示:

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

异常侦听器

对于异步消息传递，应用程序代码不能捕捉接收消息故障生成的异常。这是因为应用程序代码未显式调用 receive() 方法。要应付该情况，可以注册一个实现 onException() 方法的类实例 ExceptionListener。发生严重错误时，使用作为唯一参数传递的 JMSException 调用该方法。进一步的详细信息在 Sun 的 JMS 文档中。

第11章 编写“发布／订阅”应用程序

本部分介绍用于编写使用 MQSeries Classes for Java Message Service 的“发布／订阅”应用程序的编程模型。

编写简单的“发布／订阅”应用程序

本部分提供简单 MQ JMS 应用程序的“预排”。

导入需要的包

至少要用包含下列语句的 input 语句来开始 MQSeries classes for Java Message Service 应用程序:

```
import javax.jms.*;                 // JMS 接口
import javax.naming.*;              // 用于 JNDI 查表
import javax.naming.directory.*;    // 受管理对象
```

获取或创建 JMS 对象

下一步是获取或创建许多 JMS 对象:

1. 获取 TopicConnectionFactory
2. 创建 TopicConnection
3. 创建 TopicSession
4. 从 JNDI 获取 Topic
5. 创建 TopicPublisher 和 TopicSubscriber

这些过程中的许多都与用于点到点中的过程相似，如下所示:

获取 TopicConnectionFactory

完成该操作的更可取方式是使用 JNDI 查表，这样就维护了应用程序代码的可移植性。下列代码初始化 JNDI 环境:

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://server.company.com/o=company_us,c=us";

Java.util.Hashtable env = new java.util.Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, CTX_FACTORY );
env.put( Context.PROVIDER_URL,            INIT_URL );
env.put( Context.REFERRAL,                "throw" );

Context ctx = null;
try {
   ctx = new InitialDirContext( env );
} catch( NamingException nx ) {
   // 添加代码以处理无法连接到 JNDI 上下文
}
```

注: 需要定制 CTX_FACTORY 和 INIT_URL 变量以适应安装和 JNDI 服务供应商。

JNDI 初始化需要的特性在传递到 InitialDirContext 构造器的散列表中。如果该连接失败，将产生异常表示在应用程序中以后需要的管理对象不可用。

© Copyright IBM Corp. 1997, 2001

现在使用管理器已经定义的查表密钥来获取 TopicConnectionFactory:

```
TopicConnectionFactory factory;
factory = (TopicConnectionFactory)lookup("cn=sample.tcf");
```

如果 JNDI 名称空间不可用，可以在运行时创建 TopicConnectionFactory。用与第169页的『在运行时创建工厂』中描述的 QueueConnectionFactory 方法类似的方式创建新的 com.ibm.mq.jms.MQTopicConnectionFactory。

创建 TopicConnection

这是从 TopicConnectionFactory 对象创建的。总是在停止状态初始化连接并且必须用下列代码启动:

```
TopicConnection conn;
conn = factory.createTopicConnection();
conn.start();
```

创建 TopicSession

这是用 TopicConnection 创建的。这个方法采用两个参数；一个指定是否处理会话，另一个指定确认方式:

```
TopicSession session = conn.createTopicSession( false,
                                            Session.AUTO_ACKNOWLEDGE );
```

获得 Topic

可从 JNDI 获取该对象，要用于 TopicPublisher 和 TopicSubscriber 需要以后创建。下列代码检索 Topic:

```
Topic topic = null;
 try {
    topic = (Topic)ctx.lookup( "cn=sample.topic" );
 } catch( NamingException nx ) {
   // 添加代码以处理无法从 JNDI 检索 Topic
 }
```

如果 JNDI 名称空间不可用，可以在运行时创建 Topic，如第180页的『在运行时创建主题』所述。

创建出版物的消费者和生产者

根据编写的 JMS 客户机应用程序的性质，必须创建订阅者、发布者或二者都创建。使用下列 `createPublisher` 和 `createSubscriber` 方法:

```
// 创建发布者，发布给定主题
 TopicPublisher pub = session.createPublisher( topic );
// 创建订户，订阅给定主题
 TopicSubscriber sub = session.createSubscriber( topic );
```

发布消息

TopicPublisher 对象 pub 用于发布消息，有点类似于在点到点域中使用的 QueueSender。下列段使用会话创建 TextMessage，然后发布该消息:

```
// 创建 TextMessage 并包一些数据放入其中
TextMessage outMsg = session.createTextMessage();
outMsg.setText( "This is a short test string!" );

// 使用发布者发布消息
pub.publish( outMsg );
```

接收订阅

订户必须能阅读用下列代码传递到它们的订阅:

```
// 检索下一个正在等待的订阅
TextMessage inMsg = (TextMessage)sub.receive();

// 获取消息内容
String payload = inMsg.getText();
```

这个代码段执行“边等边获取”，它意味着将阻塞接收调用直到消息可用。接收调用的替代版本是可用的。有关详细信息，参见第327页的『TopicSubscriber』。

关闭不想要的资源

发布／订阅应用程序终止后释放其所有资源是很重要的。在能关闭的对象（发布者、订户、会话和连接）上使用 close() 方法:

```
// 关闭发布者和订户
pub.close();
sub.close();

// 关闭会话和连接
session.close();
conn.close();
```

使用主题

本节讨论 MQSeries classes for Java Message Service 应用程序中的 JMS Topic 对象的使用。

主题名称

本节描述 MQSeries classes for Java Message Service 中主题名称的使用。

注: JMS 规范不指定使用和维护主题层次结构的精确细节。因此，该区域对于每个供应商可能都非常不同。

MQ JMS 中的主题名称按类似于树的层次结构排列，它的示例在第180页的图3中显示。

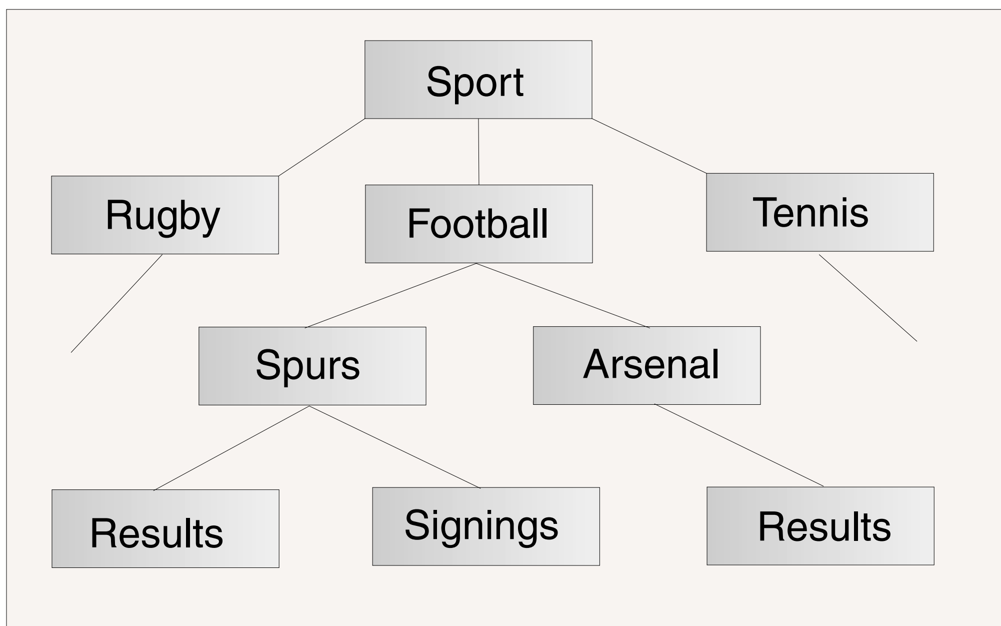


图 3. 主题名称层次结构

主题名称中，树的层次用 '/' 字符隔开。这意味着表示 'Signings' 节点的是主题名称。

```
Sport/Football/Spurs/Signings
```

MQSeries classes for Java Message Service 中主题系统的强大功能是使用了通配符。这些允许订户一次订阅多个主题。'*' 通配符匹配 0 或更多字符，'?' 通配符匹配单一字符。

如果订户订阅下列主题名称表示的“主题”：

```
Sport/Football/*/Results
```

主题上接收的出版物包括:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

如果订阅主题是:

```
Sport/Football/Spurs/*
```

主题上接收的出版物包括:
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

不必管理系统中代理方显式使用的主题层次结构。生成给定主题上的第一个发布或订阅后，代理自动创建当前正在发布和订阅的主题状态。

注: 发布者无法发布一个名称中包含通配符的主题。

在运行时创建主题

可用 4 种方法创建“主题”对象:

1. 使用单参数 MQTopic 构造器构造主题

2. 使用缺省 MQTopic 构造器构造主题，然后调用 setBaseTopicName(..) 方法
3. 使用会话的 createTopic(..) 方法
4. 使用会话的 createTemporaryTopic() 方法

Method 1：使用 MQTopic(..)

该方法需要 JMS Topic 接口的 MQSeries 实现的引用，并因此交付不可移植的代码。

构造器采用一个应该是统一资源标识 (URI) 的自变量。对于 MQSeries classes for Java Message Service Topic，它应该是如下形式:

```
topic://TopicName[?property=value[&property=value]*]
```

关于 URI 和允许的名值对，请参阅第171页的『发送消息』。

下列代码创建非持续性优先级 5 消息的主题:

```
// 使用单变量 MQTopic 构造器创建 Topic
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

方法 2：使用 MQTopic() 然后 setBaseTopicName(..)

该方法使用缺省 MQTopic 构造器，因此交付不可移植的代码。

创建了对象后，使用 setBaseTopicName 方法设置 baseTopicName 特性，用需要的主题名称传递。

注: 其中 使用的主题名称是非 URI 格式，并且不能包括名值对。使用第172页的『使用 'set' 方法设置特性』中描述的 'set' 方法设置这些。下列代码使用该方法创建主题:

```
// 使用缺省 MQTopic 构造器创建 Topic
Topic rtTopic = new MQTopic();

// 使用 setter 方法设置对象特性
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

方法 3：使用 session.createTopic(..)

也可使用 TopicSession 的 createTopic 方法创建 Topic 对象，它将主题 URI 作为下列:

```
// 使用工厂方法创建 Topic
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

方法 4：使用会话 session.createTemporaryTopic()

TemporaryTopic 是只可能由相同的 TopicConnection 创建的订户使用的 Topic。TemporaryTopic 创建如下:

```
// 使用工厂方法创建 TemporaryTopic
Topic rtTopic = session.createTemporaryTopic();
```

订户选项

使用 JMS 订户有许多不同的方法。本节描述一些使用它们的示例。

JMS 提供两种类型的订户:

非长期订户

只有当订户活动时发布消息，这些订户才接收发布在选中主题上的消息。

长期订户

这些订户接收所有发布在主题上的消息，包括那些订户为非活动时发布的消息。

创建非长期订户

第178页的『创建使用者和出版物生成者』中创建的订户是非长期的并用下列代码创建:

```
// 创建订户，订阅给定主题
 TopicSubscriber sub = session.createSubscriber( topic );
```

创建长期订户

创建长期订户与创建非长期订户非常相似，但是必须还要提供唯一标识订户的名称:

```
// 创建长期订户，提供唯一标识的名称
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

调用其 close() 方法（或它们退出作用域）时，非长期订户自动地撤销它们本身。但是，如果终止长期订阅必须显式地通知系统。要完成该操作，使用会话的 unsubscribe() 方法并用创建订户的唯一名称传递:

```
// 取消上面创建的长期订户的订阅
session.unsubscribe( "D_SUB_000001" );
```

在用 MQTopicConnectionFactory 队列管理器参数指定的队列管理器上创建长期订户。如果在不同的队列管理器上试图用相同的名称创建长期订户，则返回一个完全独立的新的长期订户。

使用消息选择器

可使用消息选择器过滤出不满足给定标准的消息。有关消息选择器的详细信息，请参阅第174页的『消息选择器』。消息选择器与订户关联如下:

```
// 将消息选择器与非长期订户关联起来
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

抑制本地出版物

可能创建忽略发布到订户本身连接上的出版物的订户。将 createSubscriber 调用设置成 true，如下:

```
// 使用 noLocal 选项集创建非长期订户
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

合并订户选项

可以合并订户变更，因此如果愿意可以创建一个应用选择器忽略本地出版物的长期订户。下列代码段显示了组合选项的使用:

```
// 使用所用的选择器创建长期的非本地订户
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                       selector, true );
```

配置基本订户队列

使用 MQ JMS V5.2，有两种方法配置订户:

- 多队列方法

  每个订户都有指派到其上的独占队列，它从中检索所有消息。JMS 为每个订户创建新的队列。使用 MQ JMS V1.1 时这是唯一可用的方法。

- 共享式队列方法

  订户使用共享式队列，并与其它订户一起从中检索它们的消息。该方法仅需要一个队列服务多个订户。这是 MQ JMS V5.2 使用的缺省方法。

在 MQ JMS V5.2 中，可选择要使用的方法并配置要使用的队列。

共享式队列通常提供较好性能优势。因为对于带有高吞吐量的系统显著降低了需要的队列数，它还提供了强大的结构和管理优势。

在某些情况下，仍需使用多个队列:

- 消息存储器的理论物理容量更大。

  MQSeries 队列无法保持多于 640000 条消息，在共享队列方法中必须将它分到共享该队列的所有订户。 这对于长期订户更加重要，因为长期订户的存活时间通常比非长期订户的长。因此，对于长期订户可能累积更多消息。

- 订阅队列的外部管理更容易。

  对于某些应用程序类型，管理员可能想监视状态和特定的订户队列深度。订户与队列之间存在一对一映射时，更易于完成任务。

缺省配置

缺省配置使用下列共享式订阅队列。

- 对于非长期订阅使用 SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE
- 对于长期订阅使用 SYSTEM.JMS.D.SUBSCRIPTION.QUEUE

运行 MQJMS_PSQ.MQSC 脚本时创建它们。

如果需要可指定替代替代物理队列。还可以更改配置来使用多个队列方法。

配置非长期订户

可用下列方法中的一种设置非长期订户队列名称特性:

- 使用 MQ JMS 管理工具（对于 JNDI 检索的对象）设置 BROKERSUBQ 特性
- 在程序中使用 setBrokerSubQueue() 方法

对于非长期订阅，提供的队列名称应该以下列字符开头:

SYSTEM.JMS.ND.

订户选项

要选择共享队列方法，请指定显式队列名称，已命名的队列用于共享队列。指定的队列必须是创建订阅前已经物理存在的队列。

要选择多个队列方法，指定以 * 字符结尾的队列名称。随后每个用该队列名称创建的订户创建供特定订户单独使用的适当动态队列。MQ JMS 使用其本身的内部的模型队列创建这样的队列。 因此，用多队列方法动态创建所有需要的队列。

使用多队列方法时无法指定显式队列名称。但是可以指定队列前缀。这使您能够创建不同的订户队列域。例如，可使用:

SYSTEM.JMS.ND.MYDOMAIN.*

* 字符之前的字符用作前缀，所以与该订阅关联的所有动态队列将含有以 SYSTEM.JMS.ND.MYDOMAIN 开头的队列名称。

配置长期订户

如前所述，对于长期订阅仍有必要使用多队列方法。长期订阅可能有更长的存活时间，因此许多未检索消息可能累积在队列中。

因此可在“主题”对象中设置长期订户队列名称特性（即比 TopicConnectionFactory 更高的管理级别）。这使您能指定许多不同的订户队列名称，而不需重新创建以 TopicConnectionFactory 开头的多个对象。

可用下列方法中的一种设置长期订户队列名称:

- 使用 MQ JMS 管理工具（对于 JNDI 检索的对象）设置 BROKERDURSUBQ 属性
- 在程序中使用 setBrokerDurSubQueue() 方法:

```
// 使用多队列方法设置 MQTopic 长期订
// 户队列名称
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

  一旦初始化“主题”对象，则将它传递到 TopicSession createDurableSubscriber() 方法以创建指定的订阅:

```
// 使用我们早前的 Topic 创建一个长期订户
TopicSubscriber sub = new session.createDurableSubscriber
                                            (sportsTopic, "D_SUB_SPORT_001");
```

对于长期订阅，提供的队列名称应该以下列字符开头:

SYSTEM.JMS.D.

要选择共享式队列方法，请指定显式队列名称，已命名的队列用于共享式队列。指定的队列必须是创建订阅前已经物理存在的队列。

要选择多队列方法，指定以 * 字符结尾的队列名称。随后每个用该队列名称创建的订户创建供特定订户单独使用的适当动态队列。MQ JMS 使用其本身的内部的模型队列创建这样的队列。 因此，用多队列方法动态创建所有需要的队列。

使用多队列方法时无法指定显式队列名称。但是可以指定队列前缀。这使您能够创建不同的订户队列域。例如，可使用:

SYSTEM.JMS.D.MYDOMAIN.*

* 字符之前的字符用作前缀，所以与该订阅关联的所有动态队列将具有以 SYSTEM.JMS.D.MYDOMAIN 开头的队列名称。

对于长期订户的重新创建和迁移问题

对于长期订户，直到删除订户后才能尝试重新配置订户队列名称。即执行 unsubscribe() 并随后从新的消息创建队列（记住删除任何老订户消息）。

但是，如果使用 MQ JMS V1.1 创建订户，则移植到当前级别时将认可该订户。不需要删除预订。使用多个队列继续操作订阅。

解决发布／订阅问题

本节描述开发使用发布／订阅域的 JMS 客户机应用程序是可能出现的某些问题。注意本节讨论特定于发布／订阅域的问题。请参考第175页的『处理错误』和第27页的『解决问题』以获取更多常规疑难解答指南。

非完整“发布／订阅”关闭

JMS 客户机应用程序终止时释放所有外部资源是重要的。要完成该操作，请在所有可关闭的对象上（一旦不再需要它们时）调用 `close()` 方法。对于“发布／订阅”域，这些对象为:

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

MQSeries classes for Java Message Service 通过使用“层叠关闭”实现该任务的简化。使用该进程， TopicConnection 上的 ’close’ 调用将引起它创建的每个 TopicSession 上的 ’close’ 调用。这会依次引起会话创建的所有 TopicSubscriber 和 TopicPublisher 上的 ’close’ 调用。

因此要确保外部资源的正确的释放，重要的是调用应用程序创建的每个会话的 `connection.close()`。

有一些情况下 ’close’ 过程可能无法完成。这些情况包括:

- MQSeries 客户机连接丢失
- 应用程序意外终止

在这些情况下，不会调用 `close()`，仍保留代表终止的应用程序的外部资源。它的主要结果是:

代理状态不一致

“MQSeries 消息代理”可能包含不再存在的订户和发布者的注册信息。这意味着代理可能继续将消息转发到永远不可能接收到它们的订户。

订户消息和队列剩余

订户注册过程有一部分是除去订户消息。如果适当，还会除去用于接收订阅的基本 MQSeries 队列。如果未出现正常关闭，则仍保留这些消息和队列。如果代理状态不一致，则队列继续填充无法读取的消息。

订户清除实用程序

要避免与订户对象的非完全关闭相关联的问题，MQ JMS 包含了订户清除实用程序。初始化第一个使用物理队列管理器的 TopicConnection 时，在队列管理器上运行该实用程

序。如果初始化该队列管理器的下一个 TopicConnection 时关闭了给定队列管理器上的所有 TopicConnection，则再次运行该实用程序。

清除实用程序试图检测其它应用程序中已发生的 MQ JMS 发布／订阅问题。如果它检测到问题，则通过下列方法清除关联的资源:

- 撤销“MQSeries 消息代理”存储
- 清除任何与订阅关联的非恢复消息队列

清除实用程序透明地在后台运行并且只维持很短的时间。它将不影响其它 MQ JMS 操作。如果检测到大量有关给定队列管理器的问题，清除资源后在初始化时可能发生小延迟。

注: 强烈建议在任何可能的时候完全关闭所有订户对象以避免订户构建问题。

处理代理报告

MQ JMS 实现使用从代理来的报告消息确认注册和撤销注册命令。通常是 MQSeries classes for Java Message Service 实现使用这些消息，但在某些错误条件下它们会保留在队列上。将这些消息发送到本地队列管理器上的 SYSTEM.JMS.REPORT.QUEUE 队列。

MQSeries classes for Java Message Service 提供了将该队列的内容转储为纯文本格式的 Java 应用程序 PSReportDump。然后可由用户或 IBM 支持人员分析该信息。也可在诊断或解决问题后使用应用程序清除消息队列。

工具的已编译形式安装在 <MQ_JAVA_INSTALL_PATH>/bin 目录中。要调用该工具，请更改到该目录然后使用下列命令:

```
java PSReportDump [-m queueManager] [-clear]
```

其中:

-m queueManager
= 指定要使用的队列管理器名称

-clear
= 转储其内容后清除消息队列

将输出发送到屏幕或将它重定向到文件。

第12章 JMS 消息

“JMS 消息”由下列部分组成:

头
所有消息都支持相同的头页字段集。头字段包含客户机和供应商所使用的值来标识和路由消息。

特性
每个消息包含支持应用程序定义的特性值的内置设施。特性提供了过滤应用程序定义的消息的有效机制。

主体
JMS 定义覆盖当前使用的大部分消息传递样式的几种消息类型。

JMS 定义 5 种消息主体类型:

流
Java 原始值流。顺序地填充并读取它。

映射
成对名值集合，其中名称是“字符串”，值是 Java 原始类型。可通过名称顺序或随机地访问该项。未定义项的顺序。

文本
包含 java.util.String 的消息

对象
包含 Serializable java 对象的消息

字节
未解释字节流。这个信息类型逐字地编码主体以匹配现存的的消息格式。

使用 JMSCorrelationID 头字段将一条消息与另一条链接。典型情况是将回答消息与它的请求消息链接。JMSCorrelationID 可以保持供应商定义消息标识，应用程序特定 String 或供应商固有 byte[] 值。

消息选择器

消息包含支持应用程序定义的特性值的内置设施。它有效提供了将应用程序定义的头字段添加到消息的机制。特性允许应用程序通过消息选择器拥有一个 JMS 供应商选择或使用应用程序定义的标准代表它过滤消息。应用程序定义的特性必须遵守下列规则:

- 特性名称必须遵守消息选择器标识规则。
- 特性值可能是 boolean、byte、short、int、long、float、double 和 string。
- 保留下列名称前缀: JMSX、JMS_。

发送消息前设置特性值。客户机接收到的消息特性是只读的。如果客户机试图在此时设置特性则产生 MessageNotWriteableException。如果调用 clearProperties，则特性是可读写的。

特性值可以重复消息主体值，也可以不重复。JMS 不定义将哪些值作为特性的策略。但应用程序开发商应该注意到 JMS 供应商将可能在处理消息主体中的数据时比处理消息特性中的数据更有效。为了获得最好性能，应用程序需要定制消息头时应该仅使用消息特性。这样做的主要原因是支持定制的消息选择。

© Copyright IBM Corp. 1997, 2001

消息选择器

JMS 消息选择器允许客户机通过使用消息头指定它感兴趣的消息。仅传递那些消息头与选择器匹配的消息。

消息选择器无法引用消息主体值。

用选择器中相应的标识替代消息头字段和特性值后选择器求值为真时消息选择器与消息匹配。

消息选择器是 String，它的语法以 SQL92 条件表达式语法子集为基础。消息选择器的求值顺序是在同一优先级中从左向右进行的。可使用括号来更改顺序。预定义的选择器文字和运算符名称是大写形式，但它们是不区分大小写的。

选择器可以包含:

- 文字
  - 单引号中的字符串文字。双引号表示单引号。例如 'literal' 和 'literal''s'。和 Java 字符串文字一样，这些使用 Unicode 字符编码。
  - 准确的数值文字是不带十进制小数点的数值，例如 57、-957、+62。支持在 Java long 范围内的数。
  - 大约的数字文字是科学记数法中的确数值，例如 7E3 或 -57.9E2 或带有小数点的数值，例如 7.、-95.7 或 +6.2。支持在 Java double 范围内的数。
  - boolean 文字 TRUE 和 FALSE。
- 标识:
  - 标识是不限制长度的 Java 字母和 Java 数字的序列，但它的第一个字符必须是 Java 字母。字母可以是 Character.isJavaLetter 方法返回为真的任何字符。这包括 '_' 和 '$'。字母或数字可以是 Character.isJavaLetterOrDigit 方法返回为真的任何字符。
  - 标识不能是名称 NULL、TRUE 或 FALSE。
  - 标识不能是 NOT、AND、OR、BETWEEN、LIKE、IN 和 IS。
  - 标识可以是头字段引用或特性引用。
  - 标识是区分大小写的。
  - 消息头字段参考引用受限于:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、JMSCorrelationID 和 JMSType 值可以为空，如果是这样将作为 NULL 值。
  - 任何以 'JMSX' 开头的名称都是 JMS 定义的特性名称。
  - 任何以 'JMS_' 开头的名称都是供应商定义的特性名称。
  - 任何不是以 'JMS' 开头的名称都是应用程序定义的特性名称。如果引用了消息中不存在的特性，它的值将为 NULL。如果它不存在，则它的值是相应的特性值。
- 为 Java 定义了相同的空白：空格、水平制表符、换页符和行终止符。
- 表达式:

  - 选择器是条件表达式。求值为真的选择器匹配，求值为假或未知的不匹配。
  - 算术表达式由它们自身、算术运算、标识（其值作为数字文字）和数字文字组成。
  - 条件表达式由它们自身、比较运算和逻辑运算组成。
- 支持用标准括号 () 设置表达式求值顺序。
- 逻辑运算符的优先顺序：NOT、AND、OR。
- 比较运算符：=、>、>=、<、<=、<>（不等）。
  - 只能比较相同类型的值。例外情况是可以比较精确数值和近似数值。（需要的类型转换是由 Java 数值提升规则定义。）如果试图比较不同的类型，选择器总为假。
  - 字符串和布尔型比较只限于 = 和 <>。只有两个 string 包含相同的字符序列时二者才相等。
- 算术运算符中的优先权顺序：
  - +、- 一元。
  - *、/ ，乘法和除法。
  - +、-，加法和减法。
  - 不支持算术运算 NULL 值。如果试图这样做，完成选择器总是为空。
  - 算术运算必须使用 Java 数值提升。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比较运算符：
  - age BETWEEN 15 and 19 等于 age >= 15 AND age <= 19。
  - age NOT BETWEEN 15 and 19 等于 age < 15 OR age > 19。
  - 如果任何 BETWEEN 运算中的表达式为 NULL，则运算值为假。如果任何 NOT BETWEEN 运算中的表达式为 NULL，则运算值为真。
- 标识 [NOT] IN (string-literal1、string-literal2,...) 是用于值为 String 或 NULL 值的标识的比较运算符。
  - Country IN (' UK', 'US', 'France') 对于 'UK' 是真，对于 'Peru' 是假。他与表达式 (Country = ' UK') OR (Country = ' US') OR (Country = ' France') 等价。
  - Country NOT IN (' UK', 'US', 'France') 对于 'UK' 为假，对于 'Peru' 为真。它与表达式 NOT ((Country = ' UK') OR (Country = ' US') OR (Country = ' France')) 等价。
  - 如果 IN 或 NOT IN 运算标识为 NULL，则运算值未知。
- 标识 [NOT] LIKE 模式值 [ESCAPE 换码字符] 比较运算符，其中 标识具有 String 值。模式值是字符串文字， '_' 表示任何单个字符 '%' 表示任何字符序列（包含空序列）。所有其它字符表示它们自身。可选的换码是单一字符串文字，它的字符用于转义模式值中 '_' 和 '%' 的特殊含义。
  - phone LIKE '12%3' 对于 '123' '12993' 为真，对于 '1234' 为假。
  - word LIKE 'l_se' 对于 'lose' 为真，对于 'loose' 为假。
  - underscored LIKE '_%' ESCAPE '\' 对于 '_foo' 为真，对于 'bar' 为假。
  - phone NOT LIKE '12%3' 对于 '123' '12993' 为假，对于 '1234' 为真。
  - 如果 LIKE 或 NOT LIKE 运算为年 NULL，则该运算的值未知。
- 标识 IS NULL 比较运算符测试空头字段值或丢失的特性值。
  - prop_name IS NULL。
- 标识 IS NOT NULL 比较运算符测试是否存在非空头字段值或特性值。

– prop_name IS NOT NULL。

下列消息选择器选择带有 car 消息类型，颜色为 blue 并且重量大于 2500 镑的消息:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

如上所述，特性值可以为 NULL。SQL 92 NULL 语义定义了包含 NULL 值的选择器表达式。下列就是这些语义的摘要描述:

- SQL 将 NULL 视为未知。
- 对于未知值的比较或算术运算总是产生未知值。
- IS NULL 和 IS NOT NULL 运算符将未知值转换成相应的 TRUE 和 FALSE 值。

虽然 SQL 支持固定十进制比较和算术运算，但 JMS 消息选择器却不支持。这就是限制精确数值文字不带小数的原因。也说明了为什么将带小数的数值作为近似数值的备用表示法。

不支持 SQL 注释。

将 JMS 消息映射到 MQSeries 消息

这一节描述了如何将本章第一部分所述的 JMS 消息映射到 MQSeries 消息。这是为了那些想在 JMS 和传统的应用程序之间传输消息的程序员而着想的。也为想操纵在两个 JMS 应用程序（例如，在一个消息代理实现中）之间传输的消息的人服务。

MQSeries 消息由下列组件组成:
- MQSeries 消息描述符 (MQMD)
- MQSeries MQRFH2 头
- 消息主体。

MQRFH2 是可选的，JMS Destination 类中的标志管理它在外出消息中包含的信息。可使用 JMS 管理工具设置该标志。因为 MQRFH2 带有 JMS 指定信息，发送方知道接收目的地是 JMS 应用程序时通常将它包含在消息中。通常在将消息发送到非 JMS 应用程序时省略 MQRFH2。这是因为这样的应用程序不能预见 MQSeries 消息中的 MQRFH2。图4 显示了结构转换:

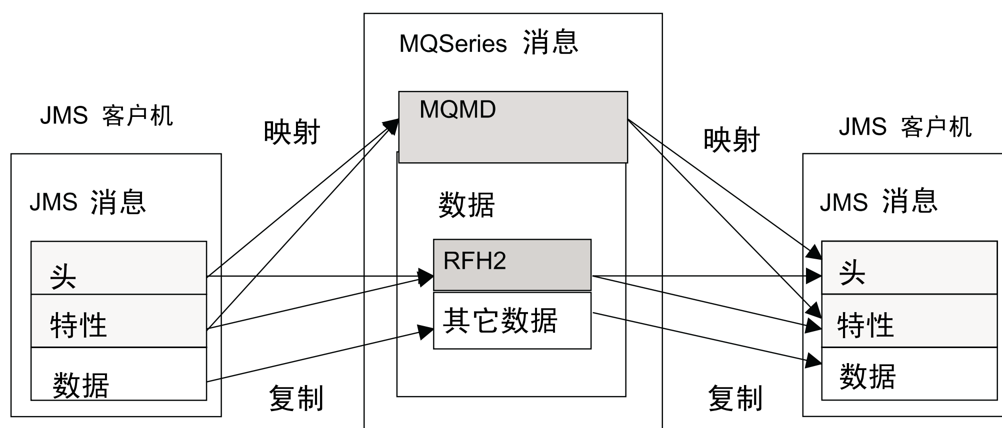


图 4. JMS 到 MQSeries 的映射模型

以两种方法转换结构:

映射 其中 MQMD 包含与 JMS 字段等价的字段，将 JMS 字段映射到 MQMD 字段。将附加的 MQMD 字段作为 JMS 特性显示，因为 JMS 应用程序与非 JMS 应用程序通信时需要获取或设置这些字段。

复制 如果没有等价的 MQMD，将传递 JMS 头或特性，可能变换成 MQRFH2 中的字段。

MQRFH2 头

这个部分描述 MQRFH 版本 2 头，它带有与消息内容关联的 JMS 指定数据。MQRFH2 版本 2 是可扩展头，也可以带有不与 JMS 直接关联的附加信息。但是，这一节只讨论它用于 JMS 的情况。

头有两部分，固定部分和可变部分。

固定部分

固定部分是以“标准” MQSeries 头模式建模并包含下列字段:

StrucId (MQCHAR4)

结构标识。

必须是 MQRFH_STRUC_ID （值: ″RFH ″）（初始值）

用常规方法定义 MQRFH_STRUC_ID_ARRAY （值: ’R’、’F’、’H’、’ ’)。

Version (MQLONG)

结构版本号。

必须是 MQRFH_VERSION_2 （值: 2）（初始值）

StrucLength (MQLONG)

MQRFH2 总长度，包括 NameValueData 字段。

在 StrucLength 中设置的值必须是 4（为达到此目的可用空格字符填充 NameValueData 字段中的数据）。

Encoding (MQLONG)

数据编码。

对 MQRFH2 后继消息部分中任何数值数据的编码（下一个头或该头后跟的消息数据）。

CodedCharSetId (MQLONG)

编码字符集标识。

MQRFH2 后续消息部分中任何字符数据的表示法（下一个头或该头后的消息数据）。

Format (MQCHAR8)

格式名。

MQRFH2 后续消息部分的模式名。

标志 (MQLONG)

标志。

MQRFH_NO_FLAGS =0。未设置标志。

NameValueCCSID (MQLONG)

该头包含的 NameValueData 字符串编码字符集标识 (CCSID)。可能在一个与包含在头 (StrucID 和 Format）中的其它字符串不同的字符集中编码 NameValueData。

如果 NameValueCCSID 是 2 字节 Unicode CCSID (1200、13488 或 17584），Unicode 的字节次序与 MQRFH2 数值字段中的字节次序相同。（例如，Version、StrucLength、NameValueCCSID 自身。）

NameValueCCSID 可以仅从下列列表中取值:

1200	可扩充的 UCS2
1208	UTF8
13488	UCS2 2.0 子集
17584	UCS2 2.1 子集（包含欧元符号）

可变部分

可变部分在固定部分之后。可变部分包含可变数量的 MQRFH2“文件夹”。每个文件夹包含可变数量的元素或特性。文件夹将相关特性组在一起。 JMS 创建的 MQRFH2 头可以最多包含 3 个文件夹:

<mcd> 文件夹
它包含描述该消息的“形状”或“格式”的特性。例如， msd 特性将消息标识为 Text、Bytes 和 Stream。 “映射”、 “对象”或“空”。总是在 JMS MQRFH2 中出现该文件夹。

<jms> 文件夹
它用来传递 JMS 头字段和不能在 MQMD 中完整表示的 JMSX 特性。通常在 JMS MQRFH2 中显示文件夹。

<usr> 文件夹
用它来传递任何与消息关联的应用程序定义特性。只有应用程序已经设置了某些应用程序定义特性时才出现该文件夹。

表17 显示了特性名称的完整列表。

表 17. JMS 使用的 MQRFH2 文件夹和特性

JMS 字段		MQRFH2 字段		
名称	**Java 类型**	**文件夹名称**	**特性名称**	**类型／值**
JMSDestination	Destination	jms	Dst	string
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	Cid	string
JMSReplyTo	Destination	jms	Rto	string
JMSType	String	mcd	Type	string
JMSXGroupID	String	jms	Gid	string
JMSXGroupSeq	int	jms	Seq	i4
xxx （用户定义）	Any	usr	xxx	any
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

用于表示可变部分特性的语法如下:

NameValueLength (MQLONG)

立即跟在该长度字段（不包括本身长度）后的 NameValueData 字符串字节长度。在 NameValueLength 中设置的值总是 4（要实现它可用空格字符填充 NameValueData 字段中的数据）。

NameValueData (MQCHARn)

前述 NameValueLength 字段给定其字符串长度的单个字符串。它包含保持“特性”序列的“文件夹”。每个特性都是包含在下列文件夹名称的 XML 元素中的“名称／类型／值”三元组。

```
<foldername> triplet1 triplet2 .....   tripletn </foldername>
```

结尾的 `</foldername>` 标记可跟空格作为填充字符。使用 XML 类似语法编码每个三位字节。

```
<name dt='datatype'>value</name>
```

`dt='datatype'` 元素是可选的并且许多特性省略它，因为它们的数据类型是预定义的。如果包含它，则必须在该 `dt=` 标记前包含一个或多个空格字符。

`name` 是特性名称 - 参见第193页的表17。

折叠后 `datatype` 必须匹配表18 中的一个文字值。

`value` 是如表18 所示要传送的值的字符串表示法。

使用下列语法编码空值:

```
<name/>
```

表 18. 特性数据类型和值

数据类型	值
string	除 < 和 & 外的任何字符序列
boolean	字符 0 或 1 (1 = ″true″)
bin.hex	表示八位元的十六进制数
i1	用数字 0..9 表示的带符号数（无小数或指数）。必须在 -128 到 127 之间。
i2	用数字 0..9 表示的带符号数（无小数或指数）。必须在 -32768 到 32767 之间
i4	用数字 0..9 表示的带符号数（无小数或指数）。必须在 -2147483648 到 2147483647 之间
i8	用数字 0..9 表示的带符号数（无小数或指数）。必须在 -9223372036854775808 到 92233720368547750807 之间
int	用数字 0..9 表示的带符号数（无小数或指数）。必须与 ’i8’ 在相同的范围内。如果发送方不想对特性使用特定精度可用它替代一种 ’i*’ 类型。
r4	<= 3.40282347E+38 且 >= 1.175E-37，用数字 0..9、可选的正负数、可选的小数数字以及可选的指数表示的浮点数
r8	<= 1.7976931348623E+308 且 >= 2.225E-307，用数字 0..9、可选的正负数、可选的小数数字以及可选的指数表示的浮点数

string 值可能包含空格。必须在 string 值中使用下列换码序列:

`&` 替换 & 字符
`<` 替换 < 字符

可使用下列换码序列，但它们不是必需的:

> 替换 > 字符
' 替换 ' 字符
" 替换 " 字符

带有相应 MQMD 字段的 JMS 字段和特性

表19 列出了直接映射到 MQMD 字段的特性。

表 19. 映射到 MQMD 字段的 JMS 特性

JMS 字段		**MQMD 字段**	
头	**Java 类型**	字段	**C 类型**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiry	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	String	MessageID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	String	CorrelId	MQBYTE24
特性			
JMSXUserID	String	UserIdentifier	MQCHAR12
JMSXAppID	String	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	String	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG
供应商定义			
JMS_IBM_Report_Exception	int	Report	MQLONG
JMS_IBM_Report_Expiration	int	Report	MQLONG
JMS_IBM_Report_COA	int	Report	MQLONG
JMS_IBM_Report_COD	int	Report	MQLONG
JMS_IBM_Report_PAN	int	Report	MQLONG
JMS_IBM_Report_NAN	int	Report	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Report	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Report	MQLONG
JMS_IBM_Report_Discard_Msg	int	Report	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	String	Format	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	String	CodedCharacterSetId	MQLONG

将 JMS 字段映射到 MQSeries 字段（外出消息）

表20 显示了如何在 send() 或publish() 时将头／特性字段映射到 MQMD/RFH2 字段。

| 对于标记了“由 Message 对象设置”的字段，发送的值是在 send/publish() 前保存到“JMS 消息”中的值。send/publish() 将不更改“JMS 消息”中的值。

对于标记了“由 Send 方法设置”的字段，执行 send/publish() 时指派值（忽略“JMS 消息”中保持的值）。更新 JMS 消息中的值以显示使用的值。

未发送标记为“只接收”的字段而是通过 send() 或 publish() 将它们不变地保留在消息中。

表 20. 外出消息字段映射

JMS 字段	在...中传输		由...设置
名称	MQMD 字段	头	
JMSDestination		MQRFH2	Send 方法
JMSDeliveryMode	Persistence	MQRFH2	Send 方法
JMSExpiration	Expiry	MQRFH2	Send 方法
JMSPriority	Priority	MQRFH2	Send 方法
JMSMessageID	MessageID		Send 方法
JMSTimestamp	PutDate/PutTime		Send 方法
JMSCorrelationID	CorrelId	MQRFH2	Message 对象
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Message 对象
JMSType		MQRFH2	Message 对象
JMSRedelivered			Receive-only
特性			
JMSXUserID	UserIdentifier		Send 方法
JMSXAppID	PutApplName		Send 方法
JMSXDeliveryCount			仅接收
JMSXGroupID	GroupId	MQRFH2	Message 对象
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Message 对象
供应商特定			
JMS_IBM_Report_Exception	Report		Message 对象
JMS_IBM_Report_Expiration	Report		Message 对象
JMS_IBM_Report_COA/COD	Report		Message 对象
JMS_IBM_Report_NAN/PAN	Report		Message 对象
JMS_IBM_Report_Pass_Msg_ID	Report		Message 对象
JMS_IBM_Report_Pass_Correl_ID	Report		Message 对象
JMS_IBM_Report_Discard_Msg	Report		Message 对象
JMS_IBM_MsgType	MsgType		Message 对象
JMS_IBM_Feedback	Feedback		Message 对象
JMS_IBM_Format	Format		Message 对象
JMS_IBM_PutApplType	PutApplType		Send 方法
JMS_IBM_Encoding	Encoding		Message 对象

表 20. 外出消息字段映射（续）

JMS 字段	在...中传输		由...设置
名称	MQMD 字段	头	
JMS_IBM_Character_Set	CodedCharacterSetId		Message 对象

在 send()/publish() 映射 JMS 头

下列注释与在 send()/publish() 上映射 JMS 字段有关:

- **到 MQRFH2 的 JMS 目的地:** 将它作为序列化 Destination 对象特征的字符串存储，这样接收的 JMS 就可以重新构成等价 Destination 对象。将 MQRFH2 字段编码成 URI（参阅第171页的统一资源标识以获取有关 URI 表示法的详细信息）。
- **JMSReplyTo 到 MQMD ReplyToQ、ReplyToQMgr、MQRFH2:** 分别将 Queue 和 QueueManager 名称复制到 MQMD ReplyToQ 和 ReplyToQMgr 字段。将目的地扩展名信息（保存在"Destination 对象"中的其它有用信息）复制到 MQRFH2 字段。将 MQRFH2 字段编码成 URI（参阅第171页的统一资源标识以获取有关 URI 表示法的详细信息）。

- **JMSDeliveryMode 到“MQMD 持续性”**：由 send/publish() Method 或 MessageProducer 设置 JMSDeliveryMode 值，除非 Destination 对象覆盖它。将 MSDeliveryMode 映射到“MQMD 持续性”字段，如下所示:
  - JMS 值 PERSISTENT 等价于 MQPER_PERSISTENT
  - JMS 值 NON_PERSISTENT 等价于 MQPER_NOT_PERSISTENT

  如果将 JMSDeliveryMode 设置成非缺省值，也将在 MQRFH2 中编码传递方式值。
- **JMSExpiration 到／自“MQMD 终止”，MQRFH2**: JMSExpiration 存储终止时间（当前时间和要存活的时间之和），而 MQMD 存储要存活的时间。JMSExpiration 以毫秒计算而 MQMD.expiry 以厘秒计算。
  - 如果 send() 方法将存活时间设置为无限长，则将“MQMD 终止”设置到 MQEI_UNLIMITED，在 MQRFH2 中不编码 JMSExpiration。
  - 如果 send() 方法将存活时间设置为少于 214748364.7 秒（大约 7 年），则将它存储在 MQMD 中。将终止和失效时间（以毫秒为单位）编码为 MQRFH2 中的 i8 值。
  - 如果 send() 方法将存活时间设置为大于 214748364.7 秒，则将 MQMD.Expiry 设置到 MQEI_UNLIMITED。将真实时间（以秒为单位）编码为 MQRFH2 中的 i8 值。
- **JMSPriority 到“MQMD 优先级”**：直接将 JMSPriority 值 (0-9) 映射成 MQMD 优先级值 (0-9)。如果将 JMSPriority 设置成非缺省值，也将在 MQRFH2 中编码优先级别。
- **JMSMessageID 自 MQMD MessageID**: 所有已经从 JMS 发送的消息都有一个 MQSeries 指派的独立标识。指派的值在调用 MQPUT 后返回 MQMD messageId 字段，并传回 JMSMessageID 字段中的应用程序。MQSeries messageId 是 24 字节的二进制值，而 JMSMessageID 是 String。JMSMessageID 由占 48 个十六进制字符带有 'ID:' 前缀的二进制 messageId 值组成。JMS 提供了能设置禁用生成消息标识的提示。忽略该提示，将为所有情况指定唯一的标识。覆盖 send() 之前设置到 JMSMessageId 字段的任何值。
- **JMSTimestamp 自 MQMD PutDate、PutTime**: 1 秒后，将 JMSTimestamp 字段设置成等于 MQMD PutDate 和 PutTime 字段给定的日期／时间。覆盖 send() 之前设置到 JMSMessageId 字段的任何值。
- **JMSType 到 MQRFH2**: 将该字符串设置到 MQRFH2。
- **JMSCorrelationID 到 MQMD CorrelId、MQRFH2**: JMSCorrelationID 可以保持下列之一:
  - **供应商特定的消息标识**: 它是以前发送或接收消息的消息标识，应该是带有 'ID:' 前缀的 48 位十六进制数字符串.除去前缀，将剩余的字符转换为二进制然后将它们设置到 MQMD CorrelId 字段。在 MQRFH2 中不编码 correlid 值。
  - **供应商固有的 byte[] 值**: 将值复制到 MQMD CorrelId 字段 - 如有必要，可用空格填充或截断成 24 个字节。在 MQRFH2 中不编码 correlid 值。
  - **应用程序特定的字符串**: 将该值复制到 MQRFH2。将 UTF8 格式的前 24 个字节字符串写入 MQMD CorrelID。

映射 JMS 特性字段

这些注释指映射 MQSeries 消息中的 JMS 特性:

- **JMSXUserID 自 MQMD UserIdentifier**: 将 JMSXUserID 设置成从发送调用返回。

- **JMSXAppID 自 MQMD PutApplName**: 将 JSMXAppID 设置成从发送调用返回。
- **JMSXGroupID 到 MQRFH2 （点到点）**: 对于点到点消息，将 JMSXGroupID 复制到 MQMD GroupID 字段。将以前缀 'ID:' 开头的 JMSXGroupID 转换为二进制。否则，将它编码为 UTF8 字符串。如果有必要使用 24 字节则可填充或截断该值。设置 MQF_MSG_IN_GROUP 标志。
- **JMSXGroupID 到 MQRFH2（发布／订阅）**: 对于发布／订阅消息，将 JMSXGroupID 作为字符串复制到 MQRFH2 中。
- **JMSXGroupSeq MQMD MsgSeqNumber（点到点）**: 对于点到点消息，将 JMSXGroupSeq 复制到 MQMD MsgSeqNumber 字段中。设置 MQF_MSG_IN_GROUP 标志。
- **JMSXGroupSeq MQMD MsgSeqNumber（发布／订阅）**: 对于发布／订阅消息，将 JMSXGroupSeq 作为 i4 复制到 MQRFH2。

映射 JMS 供应商特定的字段

下列注释是指将“JMS 供应商”特定的字段映射到 MQSeries 消息:

- **JMS_IBM_Report_<name> 到“MQMD 报告”:** JMS 应用程序可使用下列 JMS_IBM_Report_XXX 特性设置“MQMD 报告”选项。将单个 MQMD 映射到几个 JMS_IBM_Report_XXX 特性。应用程序应该将这些特性的值设置成标准 MQSeries MQRO_ 常量（包括 com.ibm.mq.MQC）。因此，例如要请求带有完整“数据”的 COD，应用程序应该将 JMS_IBM_Report_COD 设置成值 MQC.MQRO_COD_WITH_FULL_DATA。

  JMS_IBM_Report_Exception

  MQRO_EXCEPTION 或
  MQRO_EXCEPTION_WITH_DATA 或
  MQRO_EXCEPTION_WITH_FULL_DATA

  JMS_IBM_Report_Expiration

  MQRO_EXPIRATION 或
  MQRO_EXPIRATION_WITH_DATA 或
  MQRO_EXPIRATION_WITH_FULL_DATA

  JMS_IBM_Report_COA

  MQRO_COA 或
  MQRO_COA_WITH_DATA 或
  MQRO_COA_WITH_FULL_DATA

  JMS_IBM_Report_COD

  MQRO_COD 或
  MQRO_COD_WITH_DATA 或
  MQRO_COD_WITH_FULL_DATA

  JMS_IBM_Report_PAN
  MQRO_PAN

  JMS_IBM_Report_NAN
  MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

- **JMS_IBM_MsgType 到 MQMD MsgType**: 值直接映射到 MQMD MsgType。如果应用程序还未设置 JMS_IBM_MsgType 值，则使用缺省值。缺省值按如下确定:
  - 如果将 JMSReplyTo 设置成 MQSeries 队列目的地，则将 MSGType 设置成值 MQMT_REQUEST
  - 如果未设置 JMSReplyTo 或设置成非 MQSeries 队列目的地，则将 MsgType 设置到值 MQMT_DATAGRAM
- **JMS_IBM_Feedback 到 MQMD Feedback**: 值直接反馈到 MQMD Feedback。
- **JMS_IBM_Format 到 MQMD Format**: 值直接映射到 MQMD Format。
- **JMS_IBM_Encoding 到 MQMD Encoding**: 如果设置，该特性覆盖“目的队列”或“主题”的数字编码。
- **JMS_IBM_Character_Set 到 MQMD CodedCharacterSetId**: 如果设置，该特性覆盖“目的队列”或“主题”的编码字符集特性。

将 MQSeries 字段映射到 JMS 字段（进入消息）

表21 显示了如何在 send() 或 publish() 时将头／特性字段映射到 MQMD/RFH2 字段。

表 21. 进入消息字段映射

JMS 字段	**从...检索**	
名称	**MQMD 字段**	**MQRFH2**
JMS 头		
JMSDestination		jms.Dst
JMSDeliveryMode	Persistence	
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MessageID	
JMSTimestamp	PutDate PutTime	
JMSCorrelationID	CorrelId	jms.Cid
JMSReplyTo	ReplyToQ ReplyToQMgr	jms.Rto
JMSType		mcd.Type
JMSRedelivered	BackoutCount	
JMS 特性		
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId	jms.Gid
JMSXGroupSeq	MsgSeqNumber	jms.Seq
JMS 供应商特定		

表 21. 进入消息字段映射（续）

JMS 字段	从...检索	
名称	MQMD 字段	MQRFH2
JMS_IBM_Report_Exception	Report	
JMS_IBM_Report_Expiration	Report	
JMS_IBM_Report_COA	Report	
JMS_IBM_Report_COD	Report	
JMS_IBM_Report_PAN	Report	
JMS_IBM_Report_NAN	Report	
JMS_IBM_Report_ Pass_Msg_ID	Report	
JMS_IBM_Report_Pass_Correl_ID	Report	
JMS_IBM_Report_Discard_Msg	Report	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding[1]	Encoding	
JMS_IBM_Character_Set[1]	CodedCharacterSetId	
1. 仅当进入消息是“字节消息”时设置。		

将 JMS 映射到本地 MQSeries 应用程序

本节描述了如果将消息从“JMS 客户机应用程序”发送到对于 MQRFH2 头一无所知的传统 MQSeries 应用程序时发生的情况。第202页的图5 是映射表。

管理员通过将“MQSeries 目的地”的 TargetClient 值设置为 JMSC.MQJMS_CLIENT_NONJMS_MQ 表示 “JMS 客户机”正在与这样一个应用程序通信。这表示不生成 MQRFH2 字段。

从 JMS 到 MQMD 以“本地 MQSeries”应用程序为目标的映射与从 JMS 到 MQMD 以真实的 JMS 客户机为目标的映射相同。如果 JMS 接收到一个将“MQMD 格式”字段设置为非 MQFMT_RFH2 的 MQSeries 消息，便可知道我们正在从非 JMS 应用程序接收数据。如果格式是 MQFMT_STRING，接收的消息将作为“JMS 文本消息”。否则，将作为“JMS 字节”消息。因为没有 MQRFH2，只能恢复那些在 MQMD 中传输的 JMS 特性。

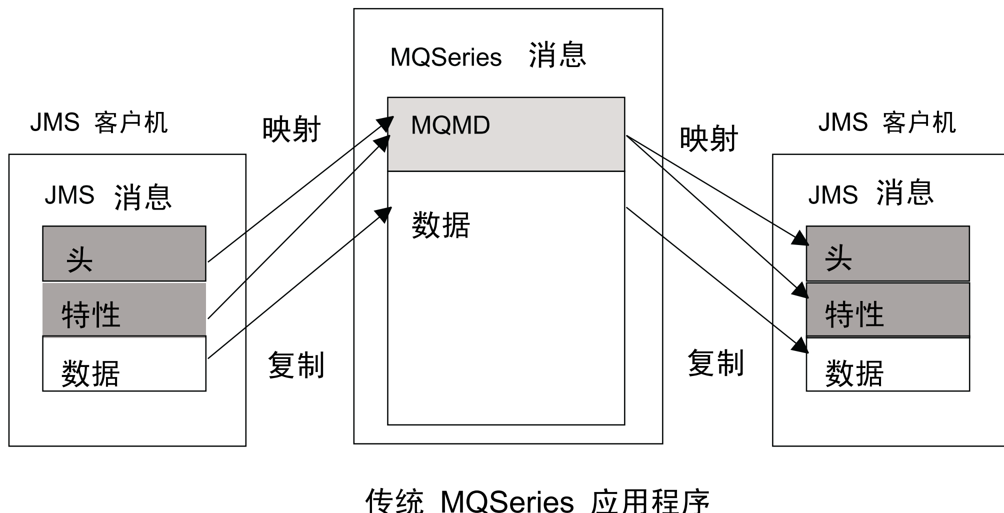


图 5. JMS 到 MQSeries 映射模型

消息主体

该节讨论了消息主体自身的编码。编码取决于 JMS 消息的类型:

ObjectMessage
: 是一个“Java 运行时”以正常方式序列化的对象。

TextMessage
: 是编码字符串。对于外出消息，在 Destination 对象给定的字符集中编码字符串。缺省情况是 UTF8 编码（UTF8 编码从消息的第一个字符开始 - 开头处没有长度字段）。可能指定 MQ Java 支持的任何其它字符集。主要在将消息发送到非 JMS 应用程序时使用这样的字符集。

如果字符集是双字节集（包括 UTF16），Destination 对象的整数编码规范确定字节顺序。

使用字符集并编码消息本身指定的信息来解释进入消息。这些规范在最右边的 MQSeries 头中（没有头时则在 MQMD 中）。对于大多数 JMS 消息，最右边的头通常是 MQRFH2。

BytesMessage
: 缺省情况下是由 JMS 1.0.2 规范和关联的 Java 文档定义的字节序列。

对于应用程序自组装的外出消息，可以使用 Destination 对象的编码特性覆盖消息中包含的整数和浮点字段。例如，可以请求以 S/390 格式而非 IEEE 格式的存储浮点值。

使用消息本身指定的数字编码解释进入消息。该规范在最右边的 MQSeries 头中（没有头时则在 MQMD 中）。对于大多数 JMS 消息，最右边的头通常是 MQRFH2。

如果接收到 BytesMessage 并不作修改地再发送，将按接收的消息逐字节传输其主体。Destination 对象的编码特性对主体不起作用。可在 BytesMessage 中显式

发送的类字符串的实体只有 UTF8 字符串。它以Java UTF8 格式编码，并以 2 字节长度字段开头。 Destination 对象的字符集特性对于输出 BytesMessage 的编码不起作用。进入 MQSeries 消息中的字符集值对作为 JMS BytesMessage 的消息的解释不起作用。

非 Java 应用程序不可能识别 Java UTF8 编码。因此，对于发送包含包含文本的 BytesMessage 的 JMS 应用程序，应用程序本身必须将它的字符串转换成字节数组，并将这些字节数组写入 BytesMessage。

MapMessage

是包含一组 XML 名称／类型／值 三元组的字符串，编码如下:

```
<map><elementName1 dt='datatype'>value</elementName1>
<elementName2 dt='datatype'>value</elementName2>.....
</map>
```

其中:

数据类型可取第194页的表18 中描述的值。

字符串是缺省数据类型，所以省略 dt='string'。

遵循应用于 TextMessage 的规则确定用来编码或解释组成 MapMessage 主体的 XML 字符串的字符集。

StreamMessage

类似于映射，但不带元素名称:

```
<stream><elt dt='datatype'>value</elt>
<elt dt='datatype'>value</elt>.....</stream>
```

发送的每个元素使用相同的标记名 (elt)。缺省类型是 string，所以对于字符串元素省略 dt='string'。

遵循应用于 TextMessage 的规则确定用来编码或解释组成 StreamMessage 主体的 XML 字符串的字符集。

按如下设置 MQRFH2.format 字段:

MQFMT_NONE

对于 ObjectMessage、BytesMessage 或不带主体的消息。

MQFMT_STRING

对于 TextMessage、StreamMessage 或 MapMessage。

第13章 MQ JMS 应用程序服务器设施

MQ JMS V5.2 支持 Java Message Service 1.0.2 规范中指定的“应用程序服务器设施”(ASF)（参阅 Sun 的 Java Web 站点 `http://java.sun.com`）。该规范标识编程模型中的三种角色:

- **JMS 供应商**提供 ConnectionConsumer 和高级 Session 功能。
- **应用程序服务器**提供 ServerSessionPool 和 ServerSession 功能。
- **客户机应用程序**使用 JMS 供应商和应用程序服务器提供的功能。

下列章节包含 MQ JMS 如何实现 ASF 的详细信息:

- 『ASF 类和函数』描述 MQ JMS 如何实现 ConnectionConsumer 类和 Session 类中的高级功能。
- 第211页的『应用程序服务器样本代码』描述样本 MQ JMS 提供的 ServerSessionPool 和 ServerSession 代码.
- 第214页的『ASF 使用示例』描述提供的 ASF 样本和从客户机应用程序透视使用的 ASF 样本。

注: ASF 的 Java Message Service 1.0.2 规范还描述了使用 X/Open XA 协议的分布式事务 JMS 支持。关于 MQ JMS 提供 XA 支持的详细信息，请参见第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

ASF 类和函数

MQ JMS 实现 ConnectionConsumer 类和 Session 类中的高级功能。有关详细信息，请参见:

- 第121页的『MQPoolServices』
- 第122页的『MQPoolServicesEvent』
- 第124页的『MQPoolToken』
- 第151页的『MQPoolServicesEventListener』
- 第238页的『ConnectionConsumer』
- 第283页的『QueueConnection』
- 第296页的『Session』
- 第314页的『TopicConnection』

ConnectionConsumer

JMS 规范使用 `ConnectionConsumer` 接口使应用程序服务器能够紧密集成 JMS 实现。该功能提供消息的并行处理。典型情况是应用程序服务器创建线程池，JMS 实现使得这些线程可使用消息。支持 JMS 的应用程序服务器可使用该特性提供高级消息传递功能，例如消息处理 bean。

普通应用程序不使用 ConnectionConsumer，但专门 JMS 客户机可能使用它。对于这样的客户机，ConnectionConsumer 提供高性能方法将消息并发传递到线程池。消息到达队

© Copyright IBM Corp. 1997, 2001

列或主题后，JMS 从池中选择线程并向其传递一批消息。要完成该操作，JMS 运行关联的 MessageListener 的 onMessage() 方法。

可通过构造多个 Session 和 MessageConsumer 对象取得相同效果，每个对象均带有已注册的 MessageListener 的。但是 ConnectionConsumer 提供了更好的性能、更少的资源使用和更大的灵活性。特别是需要更少的 Session 对象。

MQ JMS 提供一个完整功能示例的池实现来帮助您开发使用 ConnectionConsumer 的应用程序。可不作任何更改地使用该实现或调整它以适应应用程序的特定需要。

规划应用程序

点到点消息传递的常规规则

应用程序从 QueueConnection 对象创建 ConnectionConsumer 后，指定“JMS 队列”和选择器字符串。然后 ConnectionConsumer 开始接收消息（或更准确地说向关联的 ServeressionPool 中的 Session 提供消息）。消息到达队列，如果它们匹配选择器将被传递到相关联的 ServerSessionPool。

在 MQSeries 术语中 Queue 对象指的是本地“队列管理器”上的 QLOCAL 或 QALIAS。如果是 QALIAS，则 QALIAS 必须是指 QLOCAL。完全解析的 MQSeries QLOCAL 称为*基本 QLOCAL*。如果未关闭 ConnectionConsumer 或启动了其父代 QueueConnection，则称它为*活动的*。

多个 ConnectionConsumer，每个带有不同选择器， 可能对同一个基本 QLOCAL 运行。为维护性能，不应该将不需要的消息累加到队列中。不需要的消息是那些没有活动的 ConnectionConsumer 提供匹配选择器的消息。可设置 QueueConnectionFactory 将这些不需要的消息从队列中除去（有关详细信息，参阅第208页的『从队列中除去消息』）。可用一种或两种方法设置该属性:

- 使用“JMS 管理”工具将 QueueConnectionFactory 设置为 MRET(NO)。
- 在程序中，使用:

```
MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
```

如果不更改该设置，缺省设置是将不需要的消息保留在队列中。

可能从多个 QueueConnection 对象中创建以相同的基本 QLOCAL 为目标的 ConnectionConsumer。但由于性能原因，建议多个 JVM 不应针对同一个基本 QLOCAL 创建 ConnectionConsumer。

设置“MQSeries 队列管理器”时，考虑以下几点:

- 必须为共享输入启用基本 QLOCAL。要做到这一点，请使用下列 MQSC 命令:

  ```
  ALTER QLOCAL(your.qlocal.name) SHARE GET(ENABLED)
  ```

- 队列管理器必须已经有一个启用的死信队列。如果 ConnectionConsumer 将消息放入死信队列时出错，将停止从基本 QLOCAL 传递消息。要定义死信队列，请使用:

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- 运行 ConnectionConsumer 的用户必须有用 MQOO_SAVE_ALL_CONTEXT 和 MQOO_PASS_ALL_CONTEXT 执行 MQOPEN 的权限。详细信息请参阅针对您的特定计算机平台的 MQSeries 文档。
- 如果不需要的消息留在队列中，它们将降低系统性能。因此，规划消息选择器，在它们之间，ConnectionConsumer 将从队列中除去所有消息。

关于 MQSC 命令的详细信息，请参阅 *MQSeries MQSC 命令参考手册*。

发布／订阅消息传递的常规原则

应用程序从 TopicConnection 对象创建 ConnectionConsumer 时，它指定 Topic 对象和选择器字符串。然后 ConnectionConsumer 开始接收与选择器匹配的 Topic 上的消息。

或者，应用程序可以创建与特定名称关联的长期的 ConnectionConsumer。该 ConnectionConsumer 接收从长期的 ConnectionConsumer 最后一次有效以来已发布在 Topic 上的消息。它接收 Topic 上所有与选择器匹配的消息。

对于非长期订阅，ConnectionConsumer 订阅使用单独队列。TopicConnectionFactory 上的 CCSUB 配置选项指定要使用的队列。通常，CCSUB 应该指定单个队列给使用相同的 TopicConnectionFactory 的所有 ConnectionConsumer 使用。可能通过指定带有 ‘*’ 前缀的队列名使每个 ConnectionConsumer 生成临时队列。

对于长期订阅， Topic 的 CCDSUB 特性指定要使用的队列。它可以是已存在的队列或带有 ‘*’ 的队列名前缀。如果指定已存在的队列，所有订阅到 Topic 的长期 ConnectionConsumer 都使用该队列。如果指定带有 ‘*’ 的队列名前缀，则第一次用给定名称创建长期 ConnectionConsumer 时生成一个队列。以后用相同的名称创建长期 ConnectionConsumer 时再次使用该队列。

设置“MQSeries 队列管理器”时，考虑以下几点:

- 您的队列管理器必须已经有一个启用的死信队列。如果 ConnectionConsumer 将消息放入死信队列时出错，将停止从基本 QLOCAL 传递消息。要定义死信队列，请使用:

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```
- 运行 ConnectionConsumer 的用户必须有使用 MQOO_SAVE_ALL_CONTEXT 和 MQOO_PASS_ALL_CONTEXT 执行 MQOPEN 的权限。详细信息请参阅针对您的特定计算机平台的 MQSeries 文档。
- 可为单个 ConnectionConsumer 创建单独的专用队列以优化性能。这将以使用额外资源为代价。

处理有害消息

有时，错误格式的消息到达了队列。这样的消息可能导致接收应用程序故障并逆序恢复消息回条。这种情况下，将重复接收消息再返回队列。这些消息称为*有害消息*。ConnectionConsumer 必须能检测有害消息并将它们重新路由到替代目的地。

应用程序使用 ConnectionConsumer 时，逆序恢复消息的详情取决于应用程序服务器提供的 Session：

- Session 是带有 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE 的非处理会话时，只在发生系统错误或应用程序意外终止时才逆序恢复消息。
- Session 是带有 CLIENT_ACKNOWLEDGE 的非处理会话时，应用程序服务器调用 `Session.recover()` 可能逆序恢复未认可的消息。

  典型的情况是 MessageListener 的实现或应用程序服务器调用 `Message.acknowledge()`。目前`Message.acknowledge()` 认可传递到会话上的所有消息。
- Session 是事务型的会话时，典型情况是应用程序服务器提交会话。如果应用程序服务器检测到错误，它可能选择逆序恢复一个或多个消息。

- 如果应用程序服务器提供 XASession，将根据分布式事务提交或逆序恢复消息。应用程序服务器负责完成事务。

“MQSeries 队列管理器”记录了每条消息被逆序恢复的次数。当该次数达到可配置阈值时，ConnectionConsumer 在已命名的“逆序恢复队列”上重新排队消息。如果重新排队失败，将从队列中除去消息并重新排队到死信队列或废弃。参阅『从队列中除去消息』以获取更多详细信息。

在大多数平台上，阈值和重新排队队列是 MQSeries QLOCAL 的特性。对于点到点消息传递，这应该是基本的 QLOCAL。对于发布／订阅消息传递，这是在 TopicConnectionFactory 上定义的 CCSUB 队列或在 Topic 上定义的 CCDSUB 队列。要设置阈值并重新排队 Queue 特性，发出下列 MQSC 命令:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

对于发布／订阅消息传递，如果您的系统为每个订阅创建动态队列，这些设置将从 MQ JMS 模型队列中获取。要改变这些设置，可使用:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

如果阈值为 0，将禁用有害消息处理，有害消息将保留在输入队列中。否则，逆序恢复计数值到达阈值时消息将被发送到已命名的重新排队队列。如果逆序恢复计数值达到阈值，但消息不能转至重新排队队列，消息将被发送到死信队列或废弃。如果未定义重新排队队列，或 ConnectionConsumer 不能将消息发送到重新排队队列时将发生这种情况。在某些平台上，不能指定阈值和重新排队队列特性。在这些平台上，逆序恢复计数达到 20 时消息被发送到死信队列或废弃。参阅『从队列中除去消息』以获取详细信息。

从队列中除去消息

应用程序使用 ConnectionConsumer 时，在某些情况下 JMS 可能需要从队列中除去消息:

错误格式的消息
: 到达的消息可能是 JMS 无法语法分析的。

有害消息
: 消息可能到达逆序恢复阈值，但 ConnectionConsumer 在逆序恢复队列上对它重新排队时失败。

不感兴趣的 ConnectionConsumer
: 对于点到点消息传递，由于设置了 QueueConnectionFactory 后它不再保留不需要的消息，到达的消息是任何 ConnectionConsumer 都不需要的。

在这些情况下，ConnectionConsumer 试图从队列中除去消息。消息的 MQMD 报告字段中的配置选项设置额外特性。这些选项是:

MQRO_DEAD_LETTER_Q
: 该消息被重新排队到队列管理器的死信队列。这是缺省值。

MQRO_DISCARD_MSG
: 废弃该消息。

ConnectionConsumer 还生成报告消息，它也取决于消息的 MQMD 报告字段。该消息被发送到 ReplyToQmgr 上消息的 ReplyToQ。如果正在发送报告消息时出错，消息将被发送到死信队列。消息的 MQMD 报告字段中的异常报告选项设置报告消息的详细信息。这些选项是:

MQRO_EXCEPTION
生成了一个包含原始消息的 MQMD 的报告消息。它不包含任何消息主体数据。

MQRO_EXCEPTION_WITH_DATA
生成了一个包含 MQMD，任何 MQ 头和 100 字节的主体数据。

MQRO_EXCEPTION_WITH_FULL_DATA
生成了一个包含原始消息所有数据的报告消息。

缺省 不生成报告消息。

生成报告消息时，遵从下列选项:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

如果 ConnectionConsumer 无法遵从配置选项或消息的 MQMD 中的异常报告选项，则它的操作将取决于消息的持续性。如果消息是非持续的，则废弃消息并且不生成报告消息。如果消息是持续的，则停止从 QLOCAL 传递所有消息。

因此，定义死信队列和定期检查以确保不发生问题是重要的。特别应确保死信队列未达到其最大深度，并且它的最大消息大小对于所有消息是足够大的。

将消息重新排队到死信队列时，将在它们之前加上一个 MQSeries 死信队列头 (MQDLH)。参阅 *MQSeries Application Programming Reference* 以获取关于 MQDLH 格式的详细信息。可以在下列字段中标识 ConnectionConsumer 已放置在死信队列上的消息，或报告 ConnectionConsumer 已生成的消息:

- PutApplType 是 `MQAT_JAVA (0x1C)`
- PutApplName 是 "`MQ JMS ConnectionConsumer`"

这些字段在死信队列消息的 MQDLH 和报告消息的 MQMD 中。MQMD 的反馈字段和 MQDLH 的 Reason 字段包含描述错误的代码。关于这些代码的详细信息，请参阅『错误处理』。其它字段在 *MQSeries Application Programming Reference* 中描述。

错误处理

从错误情况恢复

如果 ConnectionConsumer 出现严重错误，将消息传递到对于相同的 QLOCAL 停止感兴趣的所有 ConnectionConsumer。典型情况下，如果 ConnectionConsumer 无法将消息重新排队到死信队列或从 QLOCAL 读取消息时出错，则出现这种情况。

出现这种情况时，使用下列方法通知应用程序和应用程序服务器:

- 通知任何使用受影响的 Connection 注册的 ExceptionListener。

可使用它们来标识问题的原因。在某种情况下，系统管理员必须干预以解决问题。

应用程序可用两种方法从这些错误条件恢复:

- 在所有受影响的 ConnectionConsumer 上调用 `close()`。应用程序只有在关闭了所有受影响的 ConnectionConsumer 并且解决了系统问题后才能创建新的 ConnectionConsumer。
- 在所有受影响的 Connection 上调用 `stop()`。一旦停止了所有 Connection 并解决了系统问题后，应用程序应该能够成功地 `start()` 所有 Connection。

原因和反馈代码

要确定错误原因可使用:

- 任何报告消息中的反馈代码
- 死信队列中任何消息的 MQDLH 中的原因代码。

ConnectionConsumer 生成下列原因代码:

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

原因
: 该消息达到 QLOCAL 上定义的“逆序恢复阈值”，但不定义“逆序恢复阈值”。

  在不能定义“逆序恢复”的平台上，该消息达到 JMS 定义的逆序恢复阈值 20。

操作
: 要避免这种情况，请确保使用队列的 ConnectionConsumer 提供处理所有消息的一组选择器或设置 QueueConnectionFactory 来保留消息。

  或者调查消息的源码。

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

原因
: 在点到点消息传递中，有一个与任何正在监控队列的 ConnectionConsumer 的选择器都不匹配的消息。要维护性能，将该消息重新排队到死信队列。

操作
: 要避免这种情况，请确保使用队列的 ConnectionConsumer 提供处理所有消息的一组选择器或设置 QueueConnectionFactory 来保留消息。

  或者调查消息的源码。

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

原因
: JMS 无法解释队列上的消息。

操作
: 调查消息源。JMS 经常传递 BytesMessage 或 TextMessage 意外格式的消息。偶尔情况下消息格式严重损坏，这也可能失败。

这些字段中出现的其它代码是由要将消息重新排队到“逆序恢复队列”的失败尝试引起的。在这种情况下，代码描述重新排队失败的原因。要诊断这些错误的原因，请参考 *MQSeries Application Programming Reference*。

如果无法将报告消息放入 ReplyToQ，则将它放入死信队列。在这种情况下，将按以上所述填充 MQMD 的反馈字段。MQDLH 中的原因字段说明不能将报告消息放入 ReplyToQ 的原因。

应用程序服务器样本代码

第212页的图6总结了 ServerSessionPool 和 ServerSession 功能的原则。

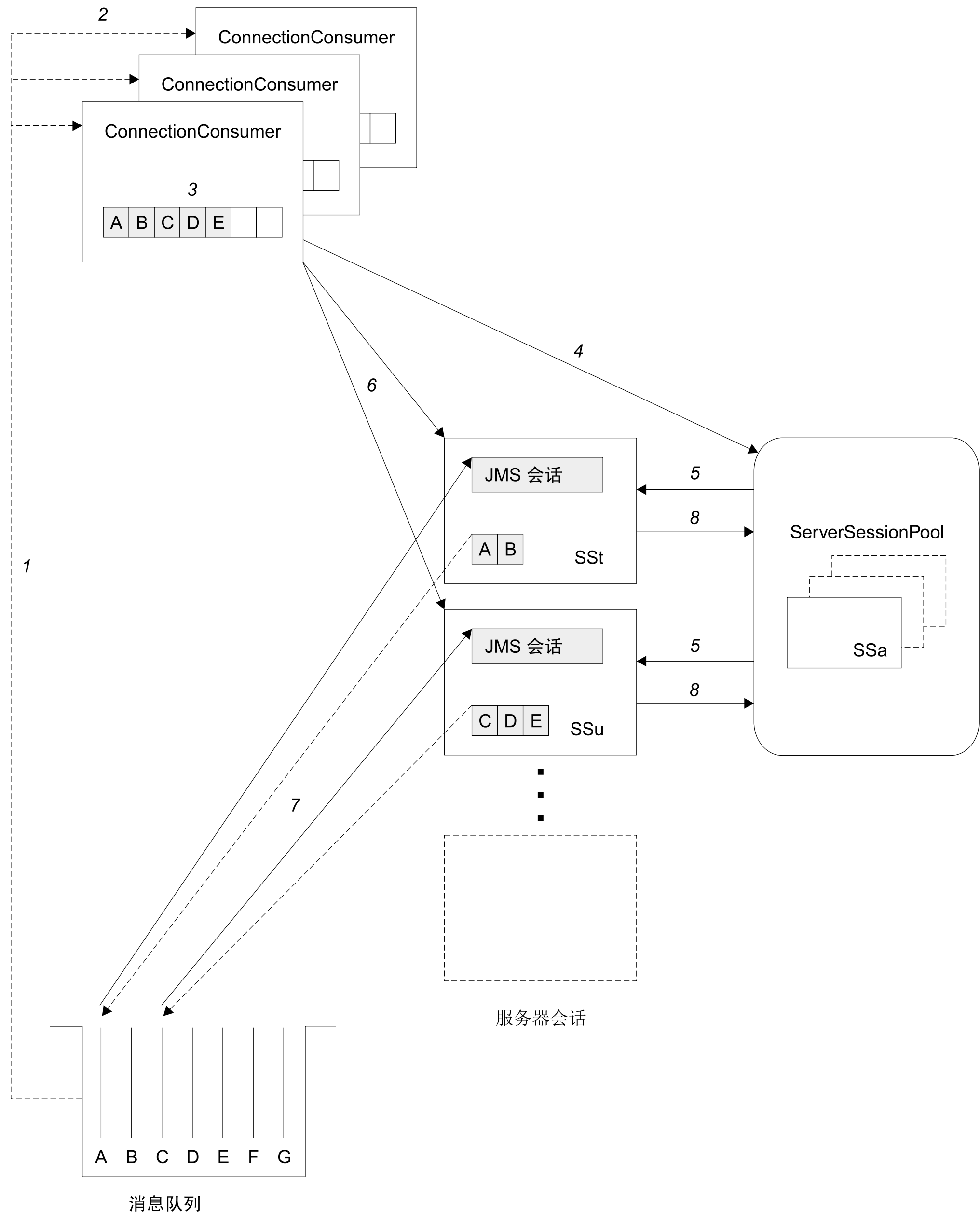


图 6. ServerSessionPool 和 ServerSession 功能

1. ConnectionConsumer 从队列获取消息引用。
2. 每个 ConnectionConsumer 选择特定的消息引用。
3. ConnectionConsumer 缓冲区保持已选中的消息引用。
4. ConnectionConsumer 从 ServerSessionPool 请求一个或多个 ServerSession。
5. 从 ServerSessionPool 分配 ServerSession.
6. ConnectionConsumer 将消息引用指派到 ServerSessions 并且启动运行 ServerSession 线程。
7. 每个 ServerSession 从队列检索它引用的消息。它从与 JMS Session 关联的 MessageListener 将它们传递到 `onMessage` 方法。
8. 完成处理后，将 ServerSession 返回池。

应用程序服务器通常提供 ServerSessionPool 和 ServerSession 功能。但是， MQ JMS 提供了附有程序源码的这些接口的简单实现。这些样本在下列目录中，其中 <install_dir> 是 MQ JMS 的安装目录:

```
<install_dir>/samples/jms/asf
```

这些样本允许您在单机环境下使用 MQ JMS ASF（即不需要合适的应用程序服务器）。它们还提供了如何实现这些接口和利用 MQ JMS ASF 的示例。这些示例意在帮助 MQ JMS 用户和其它应用程序服务器供应商。

MyServerSession.java

这个类实现 `javax.jms.ServerSession` 接口。它的基本功能是将线程与一个 JMS Session 关联。这个类与 ServerSessionPool 合用该实例（参阅『MyServerSessionPool.java』）。作为 ServerSession，它必须实现下列两个方法:

- 将与 ServerSession 关联的该方法返回 JMS Session 的 `getSession()`。
- 启动 ServerSession 的线程和导致调用 JMS Session 的 `run()` 方法的 `start()`。

MyServerSession 还实现 `Runnable` 接口。因此， ServerSession 的线程创建以这个类为基础而不需要一个单独类。

该类使用基于两种 boolean 标志值（ready 和 quit）的机制 `wait()-notify()` 。该机制意味着 ServerSession 在构建期间创建并启动与它关联的线程。但它不能自动执行 `run()` 方法的主体。仅当 `start()` 方法将 ready 标志设置为 true 时才执行 `run()` 方法的主体。有必要将消息传递到关联的 JMS Session 时 ASF 调用 `start()` 方法。

对于传递，调用 JMS Session 的 `run()` 方法。MQ JMS ASF 将对消息的 `run()` 方法加载。

传递完成后，将 ready 标志复位成 false，通知自己的 ServerSessionPool 完成传递。然后 ServerSession 保持等待状态直到再次调用 `start()` 方法或调用 `close()` 方法并结束这个 ServerSession 的线程。

MyServerSessionPool.java

这个类实现 `javax.jms.ServerSessionPool` 接口，存在以用来创建和控制访问 ServerSessions 池。

在这种简单实现中，该池由在构建池期间创建的 ServerSession 对象组成。将下列四个参数传递到构造器中:

- `javax.jms.Connection` *connection*

  用来创建 JMS Session 的连接。
- `int` *capacity*

  MyServerSession 对象的数组大小。
- `int` *ackMode*

  JMS Session 所需的确认方式。
- `MessageListenerFactory` *mlf*

  将创建消息侦听器的 MesssageListenerFactory 提供给了 JMS Session 。请参阅第214页的『MessageListenerFactory.java』。

池构造器使用这些参数来创建 MyServerSession 对象数组。使用提供的连接创建给定确认方式的 JMS Session 并改正域（点到点的 QueueSession 和发布／订阅的 TopicSession）。给消息侦听器提供了 Session。最后，创建基于 JMS Session 的 ServerSession 对象。

该样本实现是静态模型。即创建池时就创建了池中的所有 ServerSession，之后池不能增大或缩小。这种方法只是为了简单。ServerSessionPool 可能使用复杂的算法以按需动态创建 ServerSession。

MyServerSessionPool 通过保持成为 inUse 的 boolean 数组来记录当前使用的 ServerSession。这些初始 boolean 均为 false。调用 getServerSession 方法并从池请求 ServerSession 时，搜索 inUse 数组中的第一个 false 值。找到时，将该 boolean 设置为 true 并返回相应的 ServerSession。如果在 inUse 数组中不存在 false 值，则 getServerSession 方法必须 wait() 直到发生通知。

在下列情况下发生通知:
- 调用池的 close() 方法，表示应该关闭该池。
- 当前使用的 ServerSession 完成其工作负载并调用 serverSessionFinished 方法。serverSessionFinished 方法将 ServerSession 返回池，并将相应的 inUse 标志设置为 false。然后 ServerSession 可供重使用。

MessageListenerFactory.java

该样本中，消息侦听器与每个 ServerSessionPool 实例关联。MessageListenerFactory 类表示用来获取实现 javax.jms.MessageListener 接口的类实例的非常简单的接口。该类包含单个方法:

```
javax.jms.MessageListener createMessageListener();
```

构造 ServerSessionPool 时提供了这个接口的实现。使用该对象来创建 JMS Session 的消息侦听器并备份池中的 ServerSession。该体系结构表示每个 MessageListenerFactory 接口的独立实现都必须有自己的 ServerSessionPool。

MQ JMS 包含样本 MessageListenerFactory 实现，将在第216页的『CountingMessageListenerFactory.java』中讨论。

ASF 使用示例

在 <install_dir>/samples/jms/asf 中有一组带有源码的类（其中 <install_dir> 是 MQ JMS 的安装目录）。在样本单机应用程序服务器环境中（在第211页的『应用程序服务器样本代码』中描述），这些类使用 MQ JMS 应用程序服务器设施（在第205页的『ASF 类和函数』中描述）。

这些样本提供了三个从客户机应用程序观点的 ASF 使用示例:
- 简单的点到点示例使用:
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- 更复杂的的点到点示例使用:
  - ASFClient2.java

  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- 简单发布／订阅示例使用:
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- 更复杂的发布／订阅示例使用:
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java

下列章节依次描述每个类。

Load1.java

这个类是装入带有几条消息的给定队列然后终止的简单类属 JMS 应用程序。它能从 JNDI 名称空间检索需要的受管理对象，或使用实现这些接口的 MQ JMS 类来显式地创建它们。需要的受管理对象是 QueueConnectionFactory 和 Queue。可使用命令行选项设置用来装入队列的消息数和单独消息放入之间的睡眠时间。

该应用程序有两个版本的命令行语法。

对于使用 JNDI，语法是:

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

对于不使用 JNDI，语法是:

```
java Load1 -nojndi [-qm qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

表22 描述了参数并给出了缺省值。

表 22. Load1 参数和缺省值

参数	含义	缺省值
jndiICF	JNDI 使用的初始环境工厂类	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	JNDI 使用的供应商 URL	ldap://localhost/o=ibm,c=us
qcfLookup	QueueConnectionFactory 使用的 JNDI 查表键	cn=qcf
qLookup	Queue 使用的 JNDI 查表键	cn=q
qMgrName	要连接的队列管理器名称	″″ （使用缺省队列管理器）
qName	要装入的队列名称	SYSTEM.DEFAULT.LOCAL.QUEUE
sleepTime	消息放入之间的暂停时间（以毫秒为单位）	0 （无暂停）
numMsgs	要放入的消息数	1000

如果有任何错误，将显示有害消息并终止应用程序。

可使用该应用程序来模拟 MQSeries 队列上的消息装入。消息装入将依次触发下列部分中描述的应用程序。放入队列的消息是简单的 JMS TextMessage 对象。这些对象不包含用户定义消息特性，那些特性在使用不同消息侦听器时非常有用。提供源代码，以便在必要时修改该装入应用程序。

CountingMessageListenerFactory.java

该文件包含两个类的定义:

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener 是 `javax.jms.MessageListener` 接口的简单实现。它记录已调用其 `onMessage` 方法的次数，但对它传递的消息不作任何操作。

CountingMessageListenerFactory 是 CountingMessageListener 的工厂类。它是 `MessageListenerFactory` 接口的实现（在第214页的『MessageListenerFactory.java』中描述）。该工厂记录它生成的所有消息数。它还包含显示每个侦听器的使用统计信息的 `printStats()` 方法。

ASFClient1.java

该应用程序作为 MQ JMS ASF 的客户机。它设置单个 ConnectionConsumer 以使用单个 MQSeries 队列中的消息。它显示了使用的每个消息侦听器的吞吐量统计信息，并在一分钟后终止。

应用程序可以从 JNDI 名称空间中检索需要的管理对象或使用实现这些接口的 MQ JMS 类来显式地创建它们。需要的管理对象是 QueueConnectionFactory 和 Queue。

该应用程序具有两个版本的命令行语法:

对于使用 JNDI，语法是:

```
java ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

对于不使用 JNDI，语法是:

```
java ASFClient1 -nojndi [-qm qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

表23 描述了参数并给出了缺省值。

表 23. ASFClient1 参数和缺省值

参数	含义	缺省值
jndiICF	JNDI 使用的初始环境工厂类	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	JNDI 使用的供应商 URL	ldap://localhost/o=ibm,c=us
qcfLookup	QueueConnectionFactory 使用的 JNDI 查表键	cn=qcf
qLookup	用于 Queue 的 JNDI 查表密钥	cn=q
qMgrName	要连接的队列管理器名称	""（使用缺省队列管理器）
qName	要使用的队列名称	SYSTEM.DEFAULT.LOCAL.QUEUE

表 23. ASFClient1 参数和缺省值（续）

参数	含义	缺省值
poolSize	正在使用的 ServerSessionPool 中创建的 ServerSession 数	5
batchSize	每次可指派到 ServerSession 的最大消息数	10

应用程序从 QueueConnectionFactory 中获取 QueueConnection。

使用下列项构造 MyServerSessionPool 形式的 ServerSessionPool:
- 以前创建的 QueueConnection
- 需要的 poolSize
- 确认方式，AUTO_ACKNOWLEDGE
- 第216页的『CountingMessageListenerFactory.java』中描述的 CountingMessageListenerFactory 实例

然后调用正在下列项中传递的连接的 `createConnectionConsumer` 方法:

- 早期获取的 Queue
- 空消息选择选择器（表示所有应该接受的消息）
- 刚创建的 ServerSessionPool
- 需要的 batchSize

然后通过调用连接的 `start()` 方法启动消息使用。

客户机应用程序每隔 10 秒显示使用的每个消息侦听器的吞吐量统计信息。一分钟后，关闭连接，停止服务器会话并终止应用程序。

Load2.java

这个类是装入带有许多消息的给定队列然后终止的 JMS 应用程序，与 Load1.java 相似。命令行语法也与 Load1.java 的相似（在语法用 Load2 替代 Load1）。有关详细信息，请参阅第215页的『Load1.java』。

不同之处是每个消息包含称为 value 的用户特性，它在 0 到 100 范围内随机选择整数值。该特性表示您可对该消息应用消息选择器。从而可在客户机应用程序中创建的两个使用者（在『ASFClient2.java』中描述）之间共享消息。

LoggingMessageListenerFactory.java

该文件包含两个类的定义:

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener 是 `javax.jms.MessageListener` 接口的实现。它接受传递到其上的消息并将一项写入日志文件。缺省日志文件是 `./ASFClient2.log`。可以检查该文件和发送到使用这个消息侦听器的连接使用者的消息。

LoggingMessageListenerFactory 是 LoggingMessageListener 的工厂类。它是第214页的『MessageListenerFactory.java』中描述的 `MessageListenerFactory` 接口的实现。

ASFClient2.java

ASFClient2.java 是一个比 ASFClient1.java 略微复杂的客户机应用程序。它创建两个供给相同队列但应用不同消息选择器的 ConnectionConsumer。应用程序对于一个使用者使用 CountingMessageListenerFactory，对于另一个使用 LoggingMessageListenerFactory。使用两个不同的消息侦听器工厂表示每个使用者必须有其自己的服务器会话池。

应用程序在屏幕上显示一个 ConnectionConsumer 相关的统计信息并将与另一个 ConnectionConsumer 相关的统计信息写入日志文件。

命令行语法与第216页的『ASFClient1.java』 的命令行语法相似（在语法中用 ASFClient2 替代 ASFClient1）。每个服务器会话池包含 poolSize 参数设置的 ServerSession 数。

应该有一个不均匀的消息分发。不均匀并随机地分布由 Load2 装入源队列包含值为 0 到 100 的用户特性的消息。将消息选择器 `value>75` 应用到 highConnectionConsumer，将消息选择器 `value≤75` 应用到 normalConnectionConsumer。将 highConnectionConsumer

的消息（大约全部负载的 25%）发送到 LoggingMessageListener。将 normalConnectionConsumer 的消息（大约全部负载的 75%）发送到 CountingMessageListener。

运行客户机应用程序时，每隔 10 秒将在屏幕上显示与 normalConnectionConsumer 及其关联的 CountingMessageListenerFactory 相关的统计信息。将与 highConnectionConsumer 及其关联的 LoggingMessageListenerFactory 相关的统计信息写入日志文件。

可检查屏幕和日志文件来查看消息的真实目的地。将每个 CountingMessageListener 累加到总数。只要客户机应用程序在使用完所有消息前未终止，这就应该占负载的 75%。日志文件项数应该占据其余负载。（如果在使用完所有消息前终止客户机应用程序，可以增加应用程序超时。）

TopicLoad.java

这个类是 JMS 应用程序，即 load2 队列装入器（在第218页的『Load2.java』中描述）的发布／订阅版本。它在给定主题之下发布需要的消息数然后终止。每个消息包含称为 value 的用户特性，它是从 0 到 100 范围内随机选择的整数值。

要使用该应用程序，请确保正在运行代理并完成了需要的设置。有关详细信息，请参见第20页的『“发布／订阅”方式的附加设置』。

该应用程序有两个版本的命令行语法。

对于使用 JNDI，语法是:

```
java TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

对于不使用 JNDI，语法是:

```
java TopicLoad -nojndi [-qm qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

表24 描述了参数并给出了缺省值。

表 24. TopicLoad 参数和缺省值

参数	含义	缺省值
jndiICF	JNDI 使用的初始环境工厂类	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	JNDI 使用的供应商 URL	ldap://localhost/o=ibm,c=us
tcfLookup	TopicConnectionFactory 使用的 JNDI 查表键	cn=tcf
tLookup	Topic 使用的的 JNDI 查表键	cn=t
qMgrName	要连接的队列管理器，要向其发布消息的代理队列管理器	"" （使用缺省队列管理器）
tName	要向其发布的主题名称	MQJMS/ASF/TopicLoad
sleepTime	消息放入之间的暂停时间（以毫秒为单位）	0 （无暂停）
numMsgs	要放入的消息数	200

如果有任何错误，将显示有害消息并终止应用程序。

ASFClient3.java

ASFClient3.java 是客户机应用程序，即第216页的『ASFClient1.java』的发布／订阅版本。它设置单个 ConnectionConsumer 以使用在单个 Topic 上发布的消息。它显示了使用的每个消息侦听器的吞吐量统计信息，并在一分钟后终止。

该应用程序有两个版本的命令行语法。

对于使用 JNDI，语法是:

```
java ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize poolSize] [-batchsize batchSize]
```

对于不使用 JNDI，语法是:

```
java ASFClient3 -nojndi [-qm qMgrName] [-t tName]
                        [-poolsize poolSize] [-batchsize batchSize]
```

表25 描述了参数并给出了缺省值。

表 25. ASFClient3 参数和缺省值

参数	含义	缺省值
jndiICF	JNDI 使用的初始上下文工厂类	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	JNDI 使用的供应商 URL	ldap://localhost/o=ibm,c=us
tcfLookup	TopicConnectionFactory 使用的 JNDI 查表键	cn=tcf
tLookup	Topic 使用的 JNDI 查表键	cn=t
qMgrName	要连接的队列管理器，要向其发布消息的代理队列管理器	"" （使用缺省队列管理器）
tName	要从其中使用的主题名称	MQJMS/ASF/TopicLoad
poolSize	正在使用的 ServerSessionPool 中创建的 ServerSession 数	5
batchSize	每次可指派到 ServerSession 的最大消息数	10

与 ASFClient1 相似，客户机应用程序每隔 10 秒显示使用的每个消息侦听器的吞吐量统计信息。一分钟后，关闭连接，停止服务器会话并终止应用程序。

ASFClient4.java

ASFClient4.java 是一个更复杂的发布／订阅客户机应用程序。它创建三个供给相同主题但每个都应用不同消息选择器的 ConnectionConsumer。

前两个使用 ‘high’ 和 ‘normal’ 消息选择器，与第218页的『ASFClient2.java』应用程序相同。第三个使用者不使用任何消息选择器。应用程序对于两个基于选择器的使用者使用两个 CountingMessageListenerFactory，对于第三个使用者使用 LoggingMessageListenerFactory。因为应用程序使用不同的消息侦听器工厂，所以每个使用者都必须拥有其自己的服务器会话池。

应用程序在屏幕上显示两个基于选择器的使用者的相关统计信息。它将第三个 ConnectionConsumer 的相关统计信息写入日志文件。

命令行语法与第220页的『ASFClient3.java』相似（在语法中用 ASFClient4 替代 ASFClient3）。这三个服务器会话池中的每一个都包含 poolSize 参数设置的 ServerSession 数。

运行客户机应用程序时，每隔 10 秒将在屏幕上显示与 normalConnectionConsumer 和 highConnectionConsumer 及其关联的 CountingMessageListenerFactory 相关的统计信息。将与第三个 ConnectionConsumer 及其关联的 LoggingMessageListenerFactory 相关的统计信息写入日志文件。

可检查屏幕和日志文件来查看消息的真实目的地。将每个 CountingMessageListener 累加到总数并检查日志文件项数。

消息分发应该与相同的应用程序 (ASFClient2.java) 的点到点版本所获取的分发不同。这是因为在发布／订阅域中，主题的每个使用者都为其自身获取发布在主题上的每个消息的副本。在这个应用程序中，对于给定的负载，‘high’ 和 ‘normal’ 使用者将分别接收大约 25% 和 75% 的负载。第三个使用者将仍接收 100% 负载。因此接收的消息总数将大于初始发布到主题上的 100% 负载。

第14章 JMS 接口与类

MQSeries classes for Java Message Service 由许多基于接口和类的 Sun javax.jms 包的 java 类和接口组成。应该用下列 Sun 接口和类以及下列章节中的详细描述来编写客户机。实现 Sun 接口和类的 MQSeries 对象名称带有前缀 'MQ'（除非在对象描述中另有说明）。描述包括任何与标准 JMS 定义有偏差的 MQSeries 对象的详细信息。用 '*' 标记这些偏差。

Sun Java Message Service 类和接口

下表列出了包含在软件包 **javax.jms**. 中的 JMS 对象。带有 '*' 标记的"接口"由的应用程序服务器实现。带有 '**' 标记的接口由的应用程序服务器实现。

表 26. 接口摘要

接口	描述
BytesMessage	使用 BytesMessage 发送包含未解释字节流的消息。
Connection	JMS Connection 是客户机到其 JMS 供应商的活动连接。
ConnectionConsumer	对于应用程序服务器， Connection 提供了创建 ConnectionConsumer 的特殊设施。
ConnectionFactory	ConnectionFactory 封装管理员已经定义的一组连接配置参数。
ConnectionMetaData	ConnectionMetaData 提供描述 Connection 的信息。
DeliveryMode	JMS 支持的传递方式。
Destination	Queue 和 Topic 的父接口。
ExceptionListener*	用来接收 Connection 异步传递线程产生的异常的侦听器。
MapMessage	MapMessage 用来发送名称是 String 并且值是 Java 原始类型的名值对的集合。
Message	Message 接口是所有 JMS 消息的根接口。
MessageConsumer	所有消息使用者的父接口。
MessageListener*	MessageListener 用来接收异步传递的消息。
MessageProducer	客户机使用 MessageProducer 将消息发送到 Destination。
ObjectMessage	使用 ObjectMessage 发送包含 Java 对象的消息。
Queue	Queue 对象封装供应商指定的队列名称。
QueueBrowser	客户机使用 QueueBrowser 查看队列上的消息，而不需除去它们。
QueueConnection	QueueConnection 是到 JMS 点到点供应商的活动连接。
QueueConnectionFactory	使用 QueueConnectionFactory 创建带有点到点 JMS 供应商的 QueueConnection 的客户机。
QueueReceiver	客户机使用 QueueReceiver 接收已经传递到队列的消息。
QueueSender	客户机使用 QueueSender 将消息发送到队列。
QueueSession	QueueSession 提供创建 QueueReceiver、QueueSender、QueueBrowser 和 TemporaryQueue 的方法。

© Copyright IBM Corp. 1997, 2001

表 26. 接口摘要 (续)

接口	描述
ServerSession **	ServerSession 是通过应用程序服务器实现的对象。
ServerSessionPool **	ServerSessionPool 是为处理 ConnectionConsumer 消息而提供 ServerSession 池的应用程序服务器实现的对象。
Session	JMS Session 是生成和使用消息的单线程化上下文。
StreamMessage	使用 StreamMessage 发送 Java 原始流。
TemporaryQueue	TemporaryQueue 是为 QueueConnection 的持续时间而创建的唯一 Queue 对象。
TemporaryTopic	TemporaryTopic 是为 TopicConnection 的持续时间而创建的唯一 Topic 对象。
TextMessage	使用 TextMessage 发送包含 `java.lang.String` 的消息。
Topic	Topic 对象封装供应商指定的主题名称。
TopicConnection	TopicConnection 是到 JMS Pub/Sub 供应商的活动连接。
TopicConnectionFactory	客户机使用 TopicConnectionFactory 创建带有 JMS 发布／订阅供应商的 TopicConnection。
TopicPublisher	客户机使用 TopicPublisher 在主题上发布消息。
TopicSession	TopicSession 提供创建 TopicPublisher、TopicSubscriber 和 TemporaryTopic 的方法。
TopicSubscriber	客户机使用 TopicSubscriber 接收已经发布到主题上的消息。
XAConnection	XAConnection 通过提供 XASession 扩展 Connection 能力。
XAConnectionFactory	某些应用程序服务器提供在分布式事务中使用的支持“Java Transaction Service (JTS) 的资源的分组支持。
XAQueueConnection	XAQueueConnection 提供与 QueueConnection 相同的创建选项。
XAQueueConnectionFactory	XAQueueConnectionFactory 提供与 QueueConnectionFactory 相同的创建选项。
XAQueueSession	XAQueueSession 提供可用来创建 QueueReceiver、QueueSender 和 QueueBrowser 的常规 QueueSession。
XASession	XASession 通过访问 Java Transaction API (JTA) 的 JMS 供应商支持来扩展 Session 能力。
XATopicConnection	XATopicConnection 提供与 TopicConnection 相同的创建选项。
XATopicConnectionFactory	XATopicConnectionFactory 提供与 TopicConnectionFactory 相同的创建选项。
XATopicSession	XATopicSession 提供可用于创建 TopicSubscriber 和 TopicPublisher 的常规 TopicSession。

表 27. 类摘要

类	描述
QueueRequestor	JMS 提供 QueueRequestor 帮助程序类以简化发出服务请求。
TopicRequestor	JMS 提供 TopicRequestor 帮助程序类以简化发出服务请求。

MQSeries JMS 类

下表列出了包含实现 SUN 接口的 MQSeries 类的 **com.ibm.mq.jms** 和 **com.ibm.jms** 包。

表 28. 包 'com.ibm.mq.jms' 类摘要

类	实现
MQConnection	Connection
MQConnectionConsumer	ConnectionConsumer
MQConnectionFactory	ConnectionFactory
MQConnectionMetaData	ConnectionMetaData
MQDestination	Destination
MQMessageConsumer	MessageConsumer
MQMessageProducer	MessageProducer
MQQueue	Queue
MQQueueBrowser	QueueBrowser
MQQueueConnection	QueueConnection
MQQueueConnectionFactory	QueueConnectionFactory
MQQueueEnumeration	来自 QueueBrowser 的 java.util.Enumeration
MQQueueReceiver	QueueReceiver
MQQueueSender	QueueSender
MQQueueSession	QueueSession
MQSession	Session
MQTemporaryQueue	TemporaryQueue
MQTemporaryTopic	TemporaryTopic
MQTopic	Topic
MQTopicConnection	TopicConnection
MQTopicConnectionFactory	TopicConnectionFactory
MQTopicPublisher	TopicPublisher
MQTopicSession	TopicSession
MQTopicSubscriber	TopicSubscriber
MQXAConnection	XAConnection
MQXAConnectionFactory	XAConnectionFactory
MQXAQueueConnection	XAQueueConnection
MQXAQueueConnectionFactory	XAQueueConnectionFactory
MQXAQueueSession	XAQueueSession
MQXASession	XASession
MQXATopicConnection	XATopicConnection
MQXATopicConnectionFactory	XATopicConnectionFactory
MQXATopicSession	XATopicSession

表 29. 包 'com.ibm.jms' 类摘要

类	实现
JMSBytesMessage	BytesMessage
JMSMapMessage	MapMessage
JMSMessage	Message
JMSObjectMessage	ObjectMessage
JMSStreamMessage	StreamMessage
JMSTextMessage	TextMessage

该 MQSeries classes for Java Message Service 发行版中提供下列 JMS 接口的样本实现.

- ServerSession
- ServerSessionPool

请参阅第211页的『应用程序服务器样本代码』。

BytesMessage

public interface **BytesMessage**
extends **Message**

MQSeries 类: **JMSBytesMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

BytesMessage 用于发送包含未说明字节流的消息。它继承了 **Message** 并添加了字节消息主体。消息的接收方提供对字节的说明。

注: 此信息类型用于现有消息格式的客户机编码。如果可能，将使用另一种自定义的消息类型来取代它。

另见: **MapMessage**、**Message**、**ObjectMessage**、 **StreamMessage** 和 **TextMessage**。

方法

readBoolean

```
public boolean readBoolean() throws JMSException
```

从字节消息中读取 boolean 数据。

返回: 读取的 boolean 值。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果是消息字节的结尾。

readByte

```
public byte readByte() throws JMSException
```

从字节消息中读取带符号的 8 位值。

返回: 字节消息中下一个字节作为带符号的 8 位 byte。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readUnsignedByte

```
public int readUnsignedByte() throws JMSException
```

从字节消息中读取无符号的 8 位数。

返回: 字节消息中的下一个字节，按无符号的 8 位数解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部 JMS 错误而无法读取消息。

readShort

```
public short readShort() throws JMSException
```

从字节消息中读取带符号的 16 位数。

返回: 字节消息中下两个字节，按带符号的 16 位数解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部 JMS 错误而无法读取消息。

readUnsignedShort

```
public int readUnsignedShort() throws JMSException
```

从字节消息中读取无符号的 16 位数。

返回: 字节消息中下两个字节，按无符号的 16 位数解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readChar

```
public char readChar() throws JMSException
```

从字节消息中读取 Unicode 字符值。

返回: 字节消息中下两个字节，作为 Unicode 字符。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readInt

```
public int readInt() throws JMSException
```

从字节消息中读取带符号的 32 位整数。

返回: 字节消息中下四个字节，按 `int` 解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readLong

```
public long readLong() throws JMSException
```

从字节消息中读取带符号的 64 位整数。

返回: 字节消息中下八个字节，按 `long` 解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readFloat

```
public float readFloat() throws JMSException
```

从字节消息中读取一个 `float`。

返回: 字节消息中下四个字节，按 `float` 解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

BytesMessage

readDouble

```
public double readDouble() throws JMSException
```

从字节消息中读取一个 double。

返回: 字节消息中下八个字节，按 double 解释。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readUTF

```
public java.lang.String readUTF() throws JMSException
```

从字节消息读取已使用修改后的 UTF-8 格式编码的 String。前两个字节按 2 个字节长的字段解释。

返回: 字节消息中的 Unicode String。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- MessageEOFException - 如果是消息字节的结尾。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readBytes

```
public int readBytes(byte[] value) throws JMSException
```

从字节消息中读取一个字节数组。如果流中保留了足够的字节，则整个缓冲区将填满，如果没有，则缓冲区将部分填满。

参数: value - 从中读取数据的缓冲区。

返回: 读入缓冲区的字节总数，如果没有更多数据（因为已遇到了结束字节），则返回 -1。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

readBytes

```
public int readBytes(byte[] value, int length)
                                        throws JMSException
```

读取字节消息的一部分。

参数:

- value - 从中读取数据的缓冲区。
- length - 要读取的字节数。

返回: 读入缓冲区的字节总数，如果没有更多数据（因为已遇到了结束字节），则返回 -1。

掷出:

- MessageNotReadableException - 如果消息处于只写方式。

- IndexOutOfBoundsException - 如果 length 为负，或小于数组值的长度
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

将 boolean 作为 1 字节值写入字节消息。值 true 以值（字节）1 写出；值 false 以（字节）0 写出。

参数: value - 要写入的 boolean 值。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeByte

```
public void writeByte(byte value) throws JMSException
```

将 byte 作为 1 字节值写到字节消息中。

参数: value - 要写入的 byte 值。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeShort

```
public void writeShort(short value) throws JMSException
```

将 short 作为两个字节写入字节消息。

参数: value - 要写入的 short。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeChar

```
public void writeChar(char value) throws JMSException
```

将 char 作为 2 字节值写入字节消息，高字节在前。

参数: value - 要写入的 char 值。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeInt

```
public void writeInt(int value) throws JMSException
```

将 int 作为四个字节写入字节消息。

参数: value - 要写入的 int。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeLong

```
public void writeLong(long value) throws JMSException
```

将 long 作为八个字节写入字节消息。

参数: value - 要写入的 long。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeFloat

```
public void writeFloat(float value) throws JMSException
```

使用类 Float 中的 floatToIntBits 方法将 float 变量转换为 int，然后将该 int 值作为 4 字节数值写入字节消息。

参数: value - 要写入的 float 的值。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeDouble

```
public void writeDouble(double value) throws JMSException
```

使用类 Double 中的 doubleToLongBits 方法将 double 变量转换成 long，然后将该 long 的值作为 8 字节数值写入字节消息。

参数: value - 要写入的 double 的值。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeUTF

```
public void writeUTF(java.lang.String value)
                              throws JMSException
```

将使用与机器无关方式的 UTF-8 编码将 String 写入字节消息。该 UTF-8 String 被写入以长度为 2 字节的字段开始的缓冲区。

参数: value - 要写入的 String 的值。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

将 byte 数组写入字节消息。

参数: value - 要写入的 byte 数组。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeBytes

```
public void writeBytes(byte[] value,
                    int length) throws JMSException
```

将 byte 数组的一部分写入 byte 数组。

参数:

- value - 要写入的 byte 数组值。
- offset - byte 数组内的初始偏移量。
- length - 要使用的字节数。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

writeObject

```
public void writeObject(java.lang.Object value)
                                     throws JMSException
```

将 Java 对象写入字节消息。

注: 此方法仅对简单对象类型（如 Integer、Double 和 Long）、String 以及 byte 数组起作用。

参数: value - 要写入的 Java 对象。

掷出:

- MessageNotWriteableException - 如果消息处于只读方式。
- MessageFormatException - 如果对象是无效类型。
- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

复位

```
public void reset() throws JMSException
```

让消息主体处于只读方式，并将字节重新配置成字节开始时的位置。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法复位消息。
- MessageFormatException - 如果消息含有无效格式。

Connection

public interface **Connection**
子接口：**QueueConnection**、**TopicConnection**、**XAQueueConnection** 和 **XATopicConnection**

MQSeries 类：**MQConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

JMS Connection 是至 JMS 供应商的客户机活动连接。

另见：**QueueConnection**、**TopicConnection**、**XAQueueConnection** 和 **XATopicConnection**

方法

getClientID

```
public java.lang.String getClientID()
                                    throws JMSException
```

获取该连接的客户机标识。客户机标识可以是管理员在 ConnectionFactory 中预先配置的，或是通过调用 setClientId 指定的。

返回： 唯一的客户机标识。

掷出： JMSException - 如果由于内部错误 JMS 实现无法返回此 Connection 的客户机标识。

setClientID

```
public void setClientID(java.lang.String clientID)
                                    throws JMSException
```

设置该连接的客户机标识。

注： 忽略这个点到点连接的客户机标识。

参数： clientID - 唯一的客户机标识。

掷出：

- JMSException - 如果由于内部错误 JMS 实现无法设置此 Connection 的客户机标识。
- InvalidClientIDException - 如果 JMS 客户机指定了无效或重复的客户机标识。
- IllegalStateException - 如果在错误的时间尝试设置连接的客户机标识，或者，如果已经被配置管理。

getMetaData

```
public ConnectionMetaData getMetaData() throws JMSException
```

获取该连接的元数据。

返回： 连接元数据。

抛出: JMSException - 常规异常，如果 JMS 实现无法获取该 Connection 的 Connection 元数据。

另见: **ConnectionMetaData**

getExceptionListener

```
public ExceptionListener getExceptionListener()
                                     throws JMSException
```

获取该 Connection 的 ExceptionListener。

返回: 该 Connection 的 ExceptionListener

抛出: JMSException - 常规异常，如果 JMS 实现无法获取该 Connection 的 Exception 侦听器。

setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
                                     throws JMSException
```

设置该连接的异常侦听器。

参数: Listener - 异常侦听器。

抛出: JMSException - 常规异常，如果 JMS 实现无法设置该 Connection 的 Exception 侦听器。

start

```
public void start() throws JMSException
```

启动（或重新启动） Connection 的进入消息的发送。忽略正在启动的已启动会话。

抛出: JMSException - 如果由于内部错误 JMS 实现无法启动消息发送。

另见: stop

stop

```
public void stop() throws JMSException
```

用于临时性停止 Connection 的进入消息的发送。 可以使用它的 `start` 方法重新启动它。停止后，将禁止发送给该 Connection 的所有消息使用者。同步接收被阻止，而且不把消息发送到消息侦听器。

停止会话不会影响其发送消息的能力。忽略正在停止的已停止的会话。

抛出: JMSException - 如果由于内部错误 JMS 实现无法停止消息发送。

另见: start

close

```
public void close() throws JMSException
```

因为供应商可能会代表 Connection 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。不需要关闭一个已关闭连接的会话制造商和使用者。

关闭连接将导致其会话中任何一个正在运行的事务被逆序恢复。当会话是在外部事务协调下工作的情况中，在使用 XASession 时，将不使用会话的提交和逆序恢复方法并且已关闭会话的工作结果将在稍后由事务管理器来决定。关闭连接“不”会强迫对客户机确认会话进行确认。

MQ JMS 保留着一个 Session 可用的 MQSeries hConns 池。在某些情况下，Connection.close() 将清除该池。如果一个应用程序继续使用多个 Connection，则可能需要强迫该池在 JMS 连接之间保持活动。要这样做，应为您的 JMS 应用程序的生存期注册一个带 com.ibm.mq.MQEnvironment 的 MQPoolToken。详细信息，请参阅第62页的『连接合用』和第86页的『MQEnvironment』。

掷出: JMSException - 如果由于内部错误 JMS 实现无法关闭连接。例如，释放资源失败或关闭套接字连接失败。

ConnectionConsumer

public interface **ConnectionConsumer**

MQSeries 类: **MQConnectionConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionConsumer
```

对于应用程序服务器，Connection 提供特殊的设施以创建 ConnectionConsumer。Destination 和 Property Selector 指定要使用的消息。而且，ConnectionConsumer 必须被给出一个 ServerSessionPool 以用于处理它的消息。

另件: **QueueConnection** 和 **TopicConnection**。

方法

close()

```
public void close() throws JMSException
```

因为供应商可能会代表 ConnectionConsumer 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

掷出: JMSException - 如果 JMS 实现无法代表 ConnectionConsumer 释放资源，或者，如果它无法关闭连接的使用者。

getServerSessionPool()

```
public ServerSessionPool getServerSessionPool()
                                    throws JMSException
```

获取与该连接使用者关联的服务器会话。

返回: 该连接使用者使用的服务器会话池。

掷出: JMSException - 如果由于内部错误 JMS 实现无法获取与该连接使用者关联的服务器会话池。

ConnectionFactory

public interface **ConnectionFactory**
子接口：**QueueConnectionFactory**、**TopicConnectionFactory**、
| **XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

MQSeries 类：**MQConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

ConnectionFactory 封装了一组连接配置参数，这些参数是由管理员定义的。客户机使用它来创建与 JMS 供应商的 Connection。

| 另见：**QueueConnectionFactory**、**TopicConnectionFactory**、
| **XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

MQSeries 构造器

MQConnectionFactory

```
public MQConnectionFactory()
```

方法

setDescription *

```
public void setDescription(String x)
```

对象的简短描述。

getDescription *

```
public String getDescription()
```

检索对象描述。

setTransportType *

```
public void setTransportType(int x) throws JMSException
```

设置要使用的传送类型。可以是 JMSC.MQJMS_TP_BINDINGS_MQ 或 JMSC.MQJMS_TP_CLIENT_MQ_TCPIP。

getTransportType *

```
public int getTransportType()
```

检索传送类型。

setClientId *

```
public void setClientId(String x)
```

为使用该 Connection 创建的所有连接设置客户机标识。

getClientId *

```
public String getClientId()
```

为使用该 ConnectionFactory 创建的所有连接获取用于它们的客户机标识。

setQueueManager *

```
public void setQueueManager(String x) throws JMSException
```

设置要连接到的队列管理器的名称。

getQueueManager *

```
public String getQueueManager()
```

获取队列管理器的名称。

setHostName *

```
public void setHostName(String hostname)
```

仅用于客户机，要连接到的主机名。

getHostName *

```
public String getHostName()
```

检索主机名。

setPort *

```
public void setPort(int port) throws JMSException
```

设置客户机连接的端口。

参数: port - 要使用的新值。

掷出: JMSException，如果端口是负数。

getPort *

```
public int getPort()
```

仅用于客户机连接，获取端口号。

setChannel *

```
public void setChannel(String x) throws JMSException
```

仅用于客户机，设置要使用的通道。

getChannel *

```
public String getChannel()
```

仅用于客户机，获取所使用的通道。

setCCSID *

```
public void setCCSID(int x) throws JMSException
```

| 设置连接到队列管理器时要使用的字符集。关于允许的值列表，请参阅第103
| 页的表13。建议您在多数情况下使用缺省值（819)。

getCCSID *

```
public int getCCSID()
```

获取队列管理器的字符集。

setReceiveExit *

```
public void setReceiveExit(String receiveExit)
```

实现接收出口的类名。

getReceiveExit *

```
public String getReceiveExit()
```

获取接收出口类名。

setReceiveExitInit *

```
public void setReceiveExitInit(String x)
```

传递给接收出口类构造器的初始化字符串。

getReceiveExitInit *

```
public String getReceiveExitInit()
```

获取传递给接收出口类的初始化字符串。

setSecurityExit *

```
public void setSecurityExit(String securityExit)
```

实现安全性出口的类名。

getSecurityExit *

```
public String getSecurityExit()
```

获取安全性出口类名。

setSecurityExitInit *

```
public void setSecurityExitInit(String x)
```

传递给安全性出口构造器的初始化字符串。

getSecurityExitInit *

```
public String getSecurityExitInit()
```

获取安全性出口的初始化字符串。

setSendExit *

```
public void setSendExit(String sendExit)
```

实现发送出口的类名。

getSendExit *

```
public String getSendExit()
```

获取发送出口类名。

setSendExitInit *

```
public void setSendExitInit(String x)
```

传递给发送出口构造器的初始化字符串。

getSendExitInit *

```
public String getSendExitInit()
```

获取发送出口的初始化字符串。

ConnectionMetaData

public interface **ConnectionMetaData**

MQSeries 类: **MQConnectionMetaData**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

ConnectionMetaData 提供描述连接的信息。

MQSeries 构造器

MQConnectionMetaData

```
public MQConnectionMetaData()
```

方法

getJMSVersion

```
public java.lang.String getJMSVersion() throws JMSException
```

获取 JMS 版本。

返回: JMS 版本。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

getJMSMajorVersion

```
public int getJMSMajorVersion() throws JMSException
```

获取 JMS 的主版本号。

返回: JMS 主版本号。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

getJMSMinorVersion

```
public int getJMSMinorVersion() throws JMSException
```

获取 JMS 的辅版本号。

返回: JMS 辅版本号。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

getJMSXPropertyNames

```
public java.util.Enumeration getJMSXPropertyNames()
                                          throws JMSException
```

获取列举的该连接支持的“JMSX 特性”名。

返回: 列举的 JMSX 特性名。

掷出: JMSException - 如果在特性名称检索期间 JMS 实现中发生内部错误。

getJMSProviderName

```
public java.lang.String getJMSProviderName()
                                          throws JMSException
```

获取 JMS 供应商名称。

返回: JMS 供应商名称。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

getProviderVersion

```
public java.lang.String getProviderVersion()
                                 throws JMSException
```

获取 JMS 供应商版本。

返回: JMS 供应商版本。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

getProviderMajorVersion

```
public int getProviderMajorVersion() throws JMSException
```

获取 JMS 供应商主版本号。

返回: JMS 供应商主版本号。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

getProviderMinorVersion

```
public int getProviderMinorVersion() throws JMSException
```

获取 JMS 供应商辅版本号。

返回: JMS 供应商辅版本号。

掷出: JMSException - 如果在元数据检索期间 JMS 实现中发生内部错误。

toString *

```
public String toString()
```

覆盖: 类 Object 中的 toString。

DeliveryMode

public interface **DeliveryMode**

JMS 支持的传递方式.

字段

NON_PERSISTENT

```
public static final int NON_PERSISTENT
```

这是开销最低的传递方式，因为它不需要把消息记录到稳定存储器上。

PERSISTENT

```
public static final int PERSISTENT
```

此方式要求 JMS 供应商将消息作为客户机发送操作的一部分记录到稳定存储器上。

Destination

public interface **Destination**
子接口：**Queue**、**TemporaryQueue**、**TemporaryTopic** 和 **Topic**

MQSeries 类：**MQDestination**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

Destination 对象封装了供应商特定的地址。

另见：**Queue**、**TemporaryQueue**、**TemporaryTopic** 和 **Topic**

MQSeries 构造器

MQDestination

```
public MQDestination()
```

方法

setDescription *

```
public void setDescription(String x)
```

对象的简短描述。

getDescription *

```
public String getDescription()
```

获取对象的描述。

setPriority *

```
public void setPriority(int priority) throws JMSException
```

用于重新设定发送到目的地的所有消息的优先级。

getPriority *

```
public int getPriority()
```

获取重新设定的优先级值。

setExpiry *

```
public void setExpiry(int expiry) throws JMSException
```

用于重新设定发送到目的地的所有消息的失效时间。

getExpiry *

```
public int getExpiry()
```

获取目的地失效值。

setPersistence *

```
public void setPersistence(int persistence)
                                throws JMSException
```

用于重新设定发送到目的地的所有消息的持续性。

getPersistence *

```
public int getPersistence()
```

获取目的地的持续性值。

setTargetClient *

```
public void setTargetClient(int targetClient)
                                throws JMSException
```

指示远程应用程序是否遵从 JMS 的标志。

getTargetClient *

```
public int getTargetClient()
```

获取遵从 JMS 的指示符标志。

setCCSID *

```
public void setCCSID(int x) throws JMSException
```

用于编码发送到目的地的消息中的文本字符串的字符集。关于允许的值列表，请参阅第103页的表13。缺省值是 1208 (UTF8)。

getCCSID *

```
public int getCCSID()
```

获取该目的地使用的字符集名称。

setEncoding *

```
public void setEncoding(int x) throws JMSException
```

指定用于发送到目的地的消息中数值字段的编码。关于允许的值列表，请参阅第103页的表13。

getEncoding *

```
public int getEncoding()
```

获取该目的地使用的编码。

ExceptionListener

public interface **ExceptionListener**

如果 JMS 供应商检测到 Connection 中存在严重问题，那么，如果已注册了一个 ExceptionListener，它将通知 Connection 的这个 ExceptionListener。它是通过调用侦听器的 onException() 方法，并将它传递到描述此问题的 JMSException 来实现的。

这将使客户机能够异步地被通知存在的问题。有些 Connection 只使用消息，因此它们将无法了解它们的 Connection 已发生了故障。

以下情况下将发送异常:
- 在接收异步消息时出现故障
- 消息产生一个运行时异常

方法

onException

```
public void onException(JMSException exception)
```

通知用户 JMS 异常。

参数: exception - JMS 异常。这些是因异步消息的发送而导致的异常。通常，它们表示了在从队列管理器接收消息时存在的问题，或可能是 JMS 实现中的内部错误。

MapMessage

public interface **MapMessage**
extends **Message**

MQSeries 类: **JMSMapMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

使用 MapMessage 发送名值对集合，其中名称是 String，值是 Java 原始类型。可通过名称顺序或随机地访问该项。未定义项的顺序。

另见：**BytesMessage**、**Message**、 **ObjectMessage**、**StreamMessage** 和 **TextMessage**。

方法

getBoolean

```
public boolean getBoolean(java.lang.String name)
                                          throws JMSException
```

使用给定的名称返回 boolean 值。

参数: name - boolean 的名称

返回: 带有给定名称的 boolean 值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getByte

```
public byte getByte(java.lang.String name)
                                    throws JMSException
```

返回带有给定名称的 byte 值。

参数: name - byte 名称。

返回: 带有给定名称的 byte 值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getShort

```
public short getShort(java.lang.String name) throws JMSException
```

返回带有给定名称的 short 值。

参数: name - short 名称。

返回: 带有给定名称的 short 值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getChar

```
public char getChar(java.lang.String name)
                                        throws JMSException
```

返回带有给定名称的 Unicode 字符。

参数: name - Unicode 字符名称。

返回: 带有给定名称的 Unicode 字符。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getInt

```
public int getInt(java.lang.String name)
                                              throws JMSException
```

返回带有给定名称的整数值。

参数: name - 整数名称。

返回: 带有给定名称的整数值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getLong

```
public long getLong(java.lang.String name)
                                            throws JMSException
```

返回带有给定名称的 long 值。

参数: name - long 名称。

返回: 带有给定名称的 long 值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getFloat

```
public float getFloat(java.lang.String name) throws JMSException
```

返回带有给定名称的 float 值。

参数: name - float 名称。

返回: 带有给定名称的 float 值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getDouble

```
public double getDouble(java.lang.String name) throws JMSException
```

返回带有给定名称的 double 值。

参数: name - double 名称。

返回: 带有给定名称的 double 值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getString

```
public java.lang.String getString(java.lang.String name)
                                              throws JMSException
```

返回带有给定名称的 String 值。

参数: name - String 名称。

返回: 带有给定名称的 String 值。如果没有以该名称命名的项将返回空值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getBytes

```
public byte[] getBytes(java.lang.String name) throws JMSException
```

返回带有给定值的 byte 数组值。

参数: name - byte 数组名称。

返回: 带有给定名称的 byte 数组值副本。如果没有以该名称命名的项将返回空值。

掷出:

- JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。
- MessageFormatException - 如果该类型转换无效。

getObject

```
public java.lang.Object getObject(java.lang.String name)
                                    throws JMSException
```

返回带有给定名称的Java 对象值。该方法以对象格式返回已使用 setObject 方法调用或等价的原始设置方法在 Map 中存储的值。

参数: name -Java 对象名称。

返回: 带有给定名称的 Java 对象名称，以对象格式（如果将它设置为 int，则将返回整数）。如果没有以该名称命名的项将返回空值。

掷出: JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。

getMapNames

```
public java.util.Enumeration getMapNames() throws JMSException
```

返回所有 Map 消息名称的 Enumeration。

返回: Map 消息中所有名称的枚举。

掷出: JMSException - 如果因为内部的 JMS 错误导致 JMS 读取消息失败。

setBoolean

```
public void setBoolean(java.lang.String name,
                       boolean value) throws JMSException
```

使用给定名称在 Map 中设置 boolean 值。

参数:

- name - boolean 名称。
- value - 要在 Map 中设置的 boolean 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setByte

```
public void setByte(java.lang.String name,
                    byte value) throws JMSException
```

使用给定名称在 Map 中设置 byte 值。

参数:

- name - byte 名称。
- value - 要在 Map 中设置的 byte 值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setShort

```
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

使用给定名称在 Map 中设置 short 值。

参数:

- name - short 名称。
- value - 要在 Map 中设置的 short 值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setChar

```
public void setChar(java.lang.String name,
                    char value) throws JMSException
```

使用给定名称在 Map 中设置 Unicode 字符值。

参数:

- name - Unicode 字符名称。
- value - 要在 Map 中设置的 Unicode 字符值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setInt

```
public void setInt(java.lang.String name,
                   int value) throws JMSException
```

使用给定名称在 Map 中设置整数值。

参数:

- name - 整数名称。
- value - 要在 Map 中设置的整数值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。

- MessageNotWriteableException - 如果消息处于只读方式。

setLong

```
public void setLong(java.lang.String name,
                    long value) throws JMSException
```

使用给定名称在 Map 中设置 long 值。

参数:

- name - long 名称。
- value - 要在 Map 中设置的 long 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setFloat

```
public void setFloat(java.lang.String name,
                     float value) throws JMSException
```

使用给定名称在 Map 中设置 float 值。

参数:

- name - float 名称。
- value - 要在 Map 中设置的 float 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setDouble

```
public void setDouble(java.lang.String name,
                      double value) throws JMSException
```

使用给定名称在 Map 中设置 double 值。

参数:

- name - double 名称。
- value - 要在 Map 中设置的 double 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setString

```
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

使用给定名称在 Map 中设置 String 值。

参数:

- name - String 名称。
- value - 要在 Map 中设置的 String 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value) throws JMSException
```

使用给定名称在 Map 中设置 byte 数组值。

参数:

- name - byte 数组名称。
- value - 要在 Map 中设置的 byte 数组值。
  复制了数组，因此随后对数组的修改不会改变映射中的值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value,
                     int offset,
                     int length) throws JMSException
```

使用给定的名称在 Map 中设置 byte 数组值的一部分。

复制了数组，因此随后对数组的修改不会改变映射中的值。

参数:

- name - byte 数组名称。
- value - 要在 Map 中设置的 byte 数组值。
- offset - byte 数组中的初始的偏移量。
- length - 要复制的 byte 数目。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

setObject

```
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

使用给定的名称在 Map 中设置 Java 对象值。此方法仅对对象原始类型（如 Integer、Double、Long）、String 以及 byte 数组起作用。

参数:

- name -Java 对象名称。
- value - 要在 Map 中设置的Java 对象值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageFormatException - 如果对象无效。
- MessageNotWriteableException - 如果消息处于只读方式。

itemExists

```
public boolean itemExists(java.lang.String name)
                                      throws JMSException
```

检查在该 MapMessage 中是否存在项。

参数: name - 要测试的项名。

返回: true，如果存在项。

掷出: JMSException - 如果发生 JMS 错误。

Message

public interface **Message**
子接口：**BytesMessage**、**MapMessage**、**ObjectMessage**、
StreamMessage 和 **TextMessage**

MQSeries 类：**JMSMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.MQJMSMessage
```

Message 接口是所有 JMS 消息的根接口。它定义 JMS 头和用于所有消息的确认方法。

字段

DEFAULT_DELIVERY_MODE

```
public static final int DEFAULT_DELIVERY_MODE
```

缺省传递方式值。

DEFAULT_PRIORITY

```
public static final int DEFAULT_PRIORITY
```

缺省优先级值。

DEFAULT_TIME_TO_LIVE

```
public static final long DEFAULT_TIME_TO_LIVE
```

缺省生存时间值。

方法

getJMSMessageID

```
public java.lang.String getJMSMessageID()
                                              throws JMSException
```

获取消息标识。

返回： 消息标识。

抛出： JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取消息标识。

另见：
setJMSMessageID()

setJMSMessageID

```
public void setJMSMessageID(java.lang.String id)
                                              throws JMSException
```

设置消息标识。

发送该消息时将忽略任何用这个方法设置的值，但是可使用该方法更改已接收消息中的值。

参数：
id - 该消息的标识。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置消息标识。

另见：
getJMSMessageID()

getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException
```

获取消息时间戳。

返回: 消息时间戳。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取时间戳 。

另见：
setJMSTimestamp()

setJMSTimestamp

```
public void setJMSTimestamp(long时间戳)
                               throws JMSException
```

设置消息时间戳。

发送消息时将忽略任何用该方法设置的值，但可使用该方法更改已接收消息中的值。

参数: timestamp - 该消息的时间戳。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置时间戳。

另见：
getJMSTimestamp()

getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes()
                                       throws JMSException
```

获取作为该消息的 byte 数组的相关标识。

返回: 作为 byte 数组的消息相关标识。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取相关标识。

另见：
setJMSCorrelationID()、getJMSCorrelationID()、setJMSCorrelationIDAsBytes()

setJMSCorrelationIDAsBytes

```
public void setJMSCorrelationIDAsBytes(byte[]
                                        correlationID)
                                        throws JMSException
```

将相关标识设置为消息的 byte 数组。客户机可使用该调用将 correlationID 设置为以前消息的 messageID 或应用程序指定的字符串。应用程序指定的必须以字符标识开头:

参数: correlationID - 作为字符串的相关标识或正在引用的消息的消息标识。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置相关标识。

另见 :
setJMSCorrelationID()、getJMSCorrelationID()、getJMSCorrelationIDAsBytes()

getJMSCorrelationID

```
public java.lang.String getJMSCorrelationID()
                                        throws JMSException
```

获取该消息的相关标识。

返回: 作为 String 的相关消息标识.

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取相关标识。

另见 :
setJMSCorrelationID()、getJMSCorrelationIDAsBytes()、setJMSCorrelationIDAsBytes()

setJMSCorrelationID

```
public void setJMSCorrelationID
                (java.lang.String correlationID)
                                        throws JMSException
```

设置该消息的相关标识。

客户机可使用 JMSCorrelationID 头字段链接一个消息与另一个消息。典型用法是链接响应消息及其请求消息。

注: JMSCorrelationID 的 byte[] 值的使用是不可移植的。

参数: correlationID - 正在引用消息的消息标识。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置相关标识。

另见: getJMSCorrelationID()、getJMSCorrelationIDAsBytes()、setJMSCorrelationIDAsBytes()

getJMSReplyTo

```
public Destination getJMSReplyTo() throws JMSException
```

获取将该消息的回答发送到何处。

返回: 将消息响应发送到何处

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 ReplyTo Destination。

另见：

setJMSReplyTo()

setJMSReplyTo

```
public void setJMSReplyTo(Destination replyTo)
                                           throws JMSException
```

设置应该将该消息发送到何处。

参数: replyTo - 将该消息响应发送何处。空值表示不希望回答。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 “ReplyTo Destination”。

另见：

getJMSReplyTo()

getJMSDestination

```
public Destination getJMSDestination() throws JMSException
```

获取该消息的 Destination。

返回: 该消息的 Destination。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 JMS Destination。

另见：

setJMSDestination()

setJMSDestination

```
public void setJMSDestination(Destination destination)
                                           throws JMSexception
```

设置该消息的 Destination。

| 发送消息时将忽略任何用该方法设置的值，但可使用该方法更改已接收消息中
| 的值。

参数: destination - 该消息的 Destination。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 JMS Destination。

另见：

getJMSDestination()

getJMSDeliveryMode

```
public int getJMSDeliveryMode() throws JMSException
```

获取该消息的传递方式。

返回: 该消息的传递方式。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 JMS DeliveryMode。

另见：

setJMSDeliveryMode()、DeliveryMode

setJMSDeliveryMode

```
public void setJMSDeliveryMode(int deliveryMode)
                                                throws JMSException
```

设置该消息的传递方式。

| 发送消息时将忽略任何用该方法设置的值，但可使用该方法更改已接收消息中
| 的值。

| 要在发送消息时改变传递方式，请使用 QueueSender 或 TopicPublisher 上的
| setDeliveryMode 方法（该方法是从 MessageProducer 中继承的）。

参数: deliveryMode - 该消息的传递方式。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 JMS DeliveryMode。

另见：

getJMSDeliveryMode()、DeliveryMode

getJMSRedelivered

```
public boolean getJMSRedelivered() throws JMSException
```

获取是否重新传递该消息的指示。

如果客户机接收到带有重新传递指示符集的消息，则该消息可能（但不确认）在以前就已传递到客户机但那时客户机未确认其接收。

返回: 如果正在重新传递该消息，则设置为 true。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 JMS Redelivered 标志。

另见：

setJMSRedelivered()

setJMSRedelivered

```
public void setJMSRedelivered(boolean redelivered)
                                                throws JMSException
```

设置为表示是否正在重新传递该消息。

发送消息时将忽略任何用该方法设置的值，但可使用该方法更改已接收消息中的值。

参数: redelivered - 是否正在重新传递消息的标志。

掷出: JMSException - 如果 JMS 由于内部的 JMSRedelivered 错误而无法设置 JMSRedelivered 标志。

另见：

getJMSRedelivered()

getJMSType

```
public java.lang.String getJMSType() throws JMSException
```

获取消息类型。

返回: 消息类型。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 JMS 消息类型。

另见：

setJMSType()

setJMSType

```
public void setJMSType(java.lang.String type)
                                        throws JMSException
```

设置消息类型。

不论应用程序是否使用它，JMS 客户机都应该指派一个类型值。这将确保为那些需要它的供应商正确地设置它。

参数: type - 消息的类。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 JMS 消息类型。

另见：

getJMSType()

getJMSExpiration

```
public long getJMSExpiration() throws JMSException
```

获取消息失效值。

返回: 消息失效的时间。它是客户机指定的存活时间数值和发送时间的 GMT。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 JMS 消息失效值。

另见：

setJMSExpiration()

setJMSExpiration

```
public void setJMSExpiration(long expiration)
                                         throws JMSException
```

设置消息的失效值。

发送消息时将忽略任何用该方法设置的值，但可使用该方法更改已接收消息中的值。

参数: expiration - 消息失效时间。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 JMS 消息失效值。

另见：

getJMSExpiration()

getJMSPriority

```
public int getJMSPriority() throws JMSException
```

获取消息优先级。

返回: 消息优先级。

抛出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取 JMS 消息优先级。

另见：

setJMSPriority() for priority levels

setJMSPriority

```
public void setJMSPriority(int priority)
                                    throws JMSException
```

设置消息的优先级。

JMS 定义了 10 种优先级值，0 表示最低优先级而 9 表示最高优先级。另外，客户机应该认为优先级 0-4 是与正常的优先级，优先级 5-9 是高级的优先级。

参数: priority - 该消息的优先级。

抛出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 JMS 消息的优先级。

另见：

getJMSPriority()

clearProperties

```
public void clearProperties() throws JMSException
```

清除消息的特性。不清除头字段和消息主体。

抛出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法清除 JMS 消息的优先级。

propertyExists

```
public boolean propertyExists(java.lang.String name)
                                            throws JMSException
```

检查是否存在特性值。

参数: name - 要测试的特性名称。

返回: true，如果不存在特性。

抛出: JMSException - 如果 JMS 由于内部的 JMS 错误而检无法查是否存在特性。

getBooleanProperty

```
public boolean getBooleanProperty(java.lang.String name)
                                                throws JMSException
```

使用给定的名称返回 boolean 特性值。

参数: name - boolean 特性名称。

返回: 带有给定名称的 boolean 特性值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getByteProperty

```
public byte getByteProperty(java.lang.String name)
                                              throws JMSException
```

返回带有给定名称的 byte 特性值。

参数: name - byte 特性的名称。

返回: 带有给定名称的 byte 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getShortProperty

```
public short getShortProperty(java.lang.String name)
                                                throws JMSException
```

返回带有给定名称的 short 特性值。

参数: name - short 特性的名称。

返回: 带有给定名称的 short 特性值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getIntProperty

```
public int getIntProperty(java.lang.String name)
                                            throws JMSException
```

返回带有给定名称的整数特性值。

参数: name - 整数特性的名称。

返回: 带有给定名称的整数特性值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getLongProperty

```
public long getLongProperty(java.lang.String name)
                                              throws JMSException
```

返回带有给定名称的 long 特性值。

参数: name - long 特性的名称。

返回: 带有给定名称的 long 特性值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getFloatProperty

```
public float getFloatProperty(java.lang.String name)
                                                throws JMSException
```

返回带有给定名称的 float 特性值。

参数: name - float 特性的名称。

返回: 带有给定名称的 float 特性值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getDoubleProperty

```
public double getDoubleProperty(java.lang.String name)
                                                throws JMSException
```

返回带有给定名称的 double 特性值。

参数: name - double 特性的名称。

返回: 带有给定名称的 double 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getStringProperty

```
public java.lang.String getStringProperty (java.lang.String name)
                                                throws JMSException
```

返回带有给定名称的 String 特性值。

参数: name - String 特性的名称。

返回: 带有给定名称的 String 特性值。如果没有以该名称命名的特性将返回空值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。
- MessageFormatException - 如果该类型转换无效。

getObjectProperty

```
public java.lang.Object getObjectProperty (java.lang.String name)
                                                throws JMSException
```

返回带有给定名称的 Java 对象特性值。

参数: name -Java 对象特性的名称。

返回: 带有给定名称的 Java 对象特性值，以对象格式（如果将它设置为 int，则将返回整数）。如果没有以该名称命名的特性将返回空值。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性。

getPropertyNames

```
public java.util.Enumeration getPropertyNames()
                                        throws JMSException
```

返回所有特性名称的 Enumeration。

返回: 所有特性值名称的枚举。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取特性名称。

setBooleanProperty

```
public void setBooleanProperty(java.lang.String name,
                            boolean value) throws JMSException
```

使用给定名称在 Message 中设置 boolean 特性值。

参数:

- name - boolean 特性名称。

- value - 要在 Message 中设置的 boolean 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 Property。
- MessageNotWriteableException - 如果特性处于只读方式。

setByteProperty

```
public void setByteProperty(java.lang.String name,
                            byte value) throws JMSException
```

使用给定名称在 Message 中设置 byte 特性值。

参数:

- name - byte 特性的名称。
- value - 要在 Message 中设置的 byte 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 Property。
- MessageNotWriteableException - 如果特性处于只读方式。

setShortProperty

```
public void setShortProperty(java.lang.String name,
                             short value) throws JMSException
```

使用给定名称在 Message 中设置 short 特性值。

参数:

- name - short 特性的名称。
- value - 要在 Message 中设置的 short 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 Property。
- MessageNotWriteableException - 如果特性处于只读方式。

setIntProperty

```
public void setIntProperty(java.lang.String name,
                           int value) throws JMSException
```

使用给定名称在 Message 中设置整数特性值。

参数:

- name - 整数特性的名称。
- value - 要在 Message 中设置的整数特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 Property。
- MessageNotWriteableException - 如果特性处于只读方式。

setLongProperty

```
public void setLongProperty(java.lang.String name,
                            long value) throws JMSException
```

使用给定名称在 Message 中设置 long 特性值。

参数:

- name - long 特性的名称。
- value - 要在 Message 中设置的 long 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 Property。
- MessageNotWriteableException - 如果特性处于只读方式。

setFloatProperty

```
public void setFloatProperty(java.lang.String name,
                             float value) throws JMSException
```

使用给定名称在 Message 中设置 float 特性值。

参数：

- name - float 特性的名称。
- value - 要在 Message 中设置的 float 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置特性。
- MessageNotWriteableException - 如果特性处于只读方式。

setDoubleProperty

```
public void setDoubleProperty(java.lang.String name,
                              double value) throws JMSException
```

使用给定名称在 Message 中设置 double 特性值。

参数：

- name - double 特性的名称。
- value - 要在 Message 中设置的 double 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置特性。
- MessageNotWriteableException - 如果特性处于只读方式。

setStringProperty

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value) throws JMSException
```

使用给定名称在 Message 中设置 String 特性值。

参数：

- name - String 特性的名称。
- value - 要在 Message 中设置的 String 特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置特性。
- MessageNotWriteableException - 如果特性处于只读方式。

setObjectProperty

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value) throws JMSException
```

使用给定名称在 Message 中设置特性值。

参数：

- name -Java 对象特性的名称。
- value - 要在 Message 中设置的 Java 对象特性值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置 Property。
- MessageFormatException - 如果对象无效。
- MessageNotWriteableException - 如果特性处于只读方式。

acknowledge

```
public void acknowledge() throws JMSException
```

确认它和所有会话以前接收的消息。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法确认。

clearBody

```
public void clearBody() throws JMSException
```

清除消息主体。该消息的所有其它组件保留不变。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法清除。

MessageConsumer

public interface **MessageConsumer**
子接口: **QueueReceiver** 和 **TopicSubscriber**

MQSeries 类: **MQMessageConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

所有消息使用者的父接口。 客户机使用消息使用者接收 Destination 的消息。

方法

getMessageSelector

```
public java.lang.String getMessageSelector()
                                 throws JMSException
```

获取消息使用者的消息选择器表达式。

返回: 获取消息使用者的消息选择器。

掷出: JMSException - 如果由于 JMS 错误 JMS 而无法获取消息选择器。

getMessageListener

```
public MessageListener getMessageListener()
                                 throws JMSException
```

获取消息使用者的 MessageListener。

返回: 消息使用者的侦听器, 如果没有设置侦听器, 则返回空。

掷出: JMSException - 如果由于 JMS 错误 JMS 而无法获取消息侦听器。

另见: setMessageListener

setMessageListener

```
public void setMessageListener(MessageListener listener)
                                 throws JMSException
```

设置消息使用者的 MessageListener。

参数: messageListener - 发送到侦听器的消息。

掷出: JMSException - 如果由于 JMS 错误 JMS 而无法设置消息侦听器。

另见: getMessageListener

receive

```
public Message receive() throws JMSException
```

接收该消息使用者掷出的下一条消息。

返回: 消息使用者产生的下一条消息。

掷出: JMSException - 如果 JMS 由于错误而无法接收下一条消息。

receive

```
public Message receive(long timeOut) throws JMSException
```

接收在指定的超时间隔内到达的下一条消息。如果超时值为零，将导致调用等待的时间是不确定的，将一直等待消息的到达。

参数: timeout - 超时值（单位：毫秒）

返回: 消息使用者产生的下一条消息。

掷出: JMSException - 如果 JMS 由于错误而无法接收下一条消息。

receiveNoWait

```
public Message receiveNoWait() throws JMSException
```

如果消息可立即获取，则接收下一条消息。

返回: 消息使用者产生的下一条消息，如果消息不可用，则返回空。

掷出: JMSException - 如果 JMS 由于错误而无法接收下一条消息。

close

```
public void close() throws JMSException
```

因为供应商可能会代表 MessageConsumer 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

只有当接收完毕或消息侦听器处理完毕后才执行此调用。

掷出: JMSException - 如果 JMS 由于错误而无法关闭消息使用者。

MessageListener

public interface **MessageListener**

MessageListener 用于接收异步发送的消息。

方法

onMessage

```
public void onMessage(Message message)
```

传递消息至 Listener。

参数: message - 传递到侦听器的消息。

另见 Session.setMessageListener

MessageProducer

public interface **MessageProducer**
子接口: **QueueSender** 和 **TopicPublisher**

MQSeries 类: **MQMessageProducer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

客户机使用消息生产者将消息发送到 Destination。

MQSeries 构造器

MQMessageProducer

```
public MQMessageProducer()
```

方法

setDisableMessageID

```
public void setDisableMessageID(boolean value)
                                          throws JMSException
```

设置是否禁用消息标识。

缺省情况下，消息标识是启用的。

注: 此方法在 MQSeries classes for Java Message Service 实现中被忽略。

参数: value - 表示是否禁用消息标识。

掷出: JMSException - 如果 JMS 由于内部错误而无法设置禁用的消息标识。

getDisableMessageID

```
public boolean getDisableMessageID() throws JMSException
```

获取是否禁用消息标识的指示。

返回: 是否禁用消息标识的指示。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取禁用的消息标识。

setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value)
                                                 throws JMSException
```

设置是否禁用消息时间戳。

缺省情况下，消息时间戳是启用的。

注: 此方法在 MQSeries classes for Java Message Service 实现中被忽略。

参数: value - 表示是否禁用消息时间戳。

掷出: JMSException - 如果 JMS 由于内部错误而无法设置禁用的消息时间戳。

getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp()
                                    throws JMSException
```

获取是否禁用消息时间戳的指示。

返回: 是否禁用消息标识的指示。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取禁用的消息时间戳。

setDeliveryMode

```
public void setDeliveryMode(int deliveryMode)
                                    throws JMSException
```

设置消息生产者的缺省发送方式。

缺省情况下，发送方式设置成 PERSISTENT。

参数: deliveryMode - 该消息生产者的消息发送方式。

掷出: JMSException - 如果 JMS 由于内部错误而无法设置发送方式。

另见: getDeliveryMode

getDeliveryMode

```
public int getDeliveryMode() throws JMSException
```

获取生产者的缺省发送方式。

返回: 该消息生产者的消息发送方式。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取发送方式。

另见: setDeliveryMode

setPriority

```
public void setPriority(int priority) throws JMSException
```

设置生产者的缺省优先级。

缺省情况下，优先级设置为 4。

参数: priority - 该消息生产者的消息优先级。

掷出: JMSException - 如果 JMS 由于内部错误而无法设置优先级。

另见: getPriority

getPriority

```
public int getPriority() throws JMSException
```

获取生产者的缺省优先级。

返回: 此消息生产者的消息优先级。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取优先级。

另见: setPriority

setTimeToLive

```
public void setTimeToLive(long timeToLive)
                                  throws JMSException
```

从分派时间中设置单位为毫秒的缺省时间长度，这是消息系统将保留所生成消息的时间。

缺省情况下，存活时间设置为 0。

参数: timeToLive - 消息的存活时间，单位：秒；零表示无限制。

掷出: JMSException - 如果 JMS 由于内部错误而无法设置“存活时间”。

另见: getTimeToLive

getTimeToLive

```
public long getTimeToLive() throws JMSException
```

从分派时间中获取单位为毫秒的缺省时间长度，这是消息系统将保留所生成消息的时间。

返回: 消息的存活时间，单位：秒；零表示无限制。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取“存活时间”。

另见: setTimeToLive

close

```
public void close() throws JMSException
```

因为供应商可能会代表 MessageProducer 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

掷出: JMSException - 如果 JMS 由于错误而无法关闭生产者。

MQQueueEnumeration *

public class **MQQueueEnumeration**
extends **Object**
implements **Enumeration**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

队列上消息的 Enumeration。在 JMS 规格中未定义该类，它是通过调用 MQQueueBrowser 的 getEnumeration 方法创建的。该类包含保持浏览游标的基本 MQQueue 实例。一旦游标移出队列结尾，则关闭该队列。

无法复位该类的实例 - 它使用“一次使用”机制。

另见：**MQQueueBrowser**

方法

hasMoreElements

```
public boolean hasMoreElements()
```

表示是否可以返回另一个消息。

nextElement

```
public Object nextElement() throws NoSuchElementException
```

返回当前消息。

如果 hasMoreElements() 返回 'true'，则 nextElement() 总是返回消息。已返回的消息可能在 hasMoreElements() 和 nextElement 调用之间超过其失效日期。

ObjectMessage

public interface **ObjectMessage**
extends **Message**

MQSeries 类: **JMSObjectMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

用 ObjectMessage 发送包含可串行化 Java 对象的消息。它继承自 Message 并添加了包含单个 Java 引用的主体。只有“可串行化”Java 对象才能被使用。

另见: **BytesMessage**、**MapMessage**、**Message**、**StreamMessage** 和 **TextMessage**

方法

setObject

```
public void setObject(java.io.Serializable object)
                                        throws JMSException
```

设置包含消息数据的可串行化对象。ObjectMessage 包含了调用 setObject() 时对象的快照。对象随后的修改不影响 ObjectMessage 主体。

参数: object - 消息数据。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置对象。
- MessageFormatException - 如果对象连载失败。
- MessageNotWriteableException - 如果消息处于只读方式。

getObject

```
public java.io.Serializable getObject()
                                throws JMSException
```

获取包含消息数据的可串行化对象。缺省值是空。

返回: 包含消息数据的可串行化对象。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取对象。
- MessageFormatException - 如果对象取消连载失败。

Queue

public interface **Queue**
extends **Destination**
子接口: **TemporaryQueue**

MQSeries 类: **MQQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

Queue 对象封装供应商指定的队列名称。这是客户机向 JMS 方法指定队列标识的方法。

MQSeries 构造器

MQQueue *

```
public MQQueue()
```

管理工具使用的缺省构造器。

MQQueue *

```
public MQQueue(String URIqueue)
```

创建一个新的 MQQueue 实例。字符串采用 URI 格式，第171页中有所描述。

MQQueue *

```
public MQQueue(String queueManagerName,
               String queueName)
```

方法

getQueueName

```
public java.lang.String getQueueName()
                                  throws JMSException
```

获取队列的名称。

依赖此名称的客户机不可移植。

返回: 队列名称

掷出: JMSException - 如果 Queue 的 JMS 实现因内部错误而无法返回队列名称。

toString

```
public java.lang.String toString()
```

返回打印良好的队列名称版本。

返回: 供应商指定的队列的标识值。

覆盖: 类 java.lang.Object 中的 toString

getReference *

```
public Reference getReference() throws NamingException
```

创建对该队列的引用。

返回: 对该对象的引用

掷出: NamingException

setBaseQueueName *

```
public void setBaseQueueName(String x) throws JMSException
```

设置 MQSeries 队列名称的值

注: 只有管理工具才能使用这个方法。不要尝试译码 queue:qmgr:queue 格式的字符串。

getBaseQueueName *

```
public String getBaseQueueName()
```

返回: MQSeries 队列名称的值。

setBaseQueueManagerName *

```
public void setBaseQueueManagerName(String x) throws JMSException
```

设置 MQSeries 队列管理器的名称。

注: 只有管理工具才能使用这个方法。

getBaseQueueManagerName *

```
public String getBaseQueueManagerName()
```

返回: MQSeries 队列管理器名称的值。

QueueBrowser

public interface **QueueBrowser**

MQSeries 类: **MQQueueBrowser**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

客户机使用 QueueBrowser 查看队列上的消息，但不除去它们。

注: MQSeries 类 **MQQueueEnumeration** 用来保持浏览游标。

另见: **QueueReceiver**

方法

getQueue

```
public Queue getQueue() throws JMSException
```

获取与队列浏览器关联的队列。

返回: 该队列。

抛出: JMSException - 如果由于 JMS 错误 JMS 而无法获取与该 Browser 关联的队列。

getMessageSelector

```
public java.lang.String getMessageSelector() throws JMSException
```

获取该队列浏览器的消息选择器表达式。

返回: 该队列浏览器的消息选择器。

抛出: JMSException - 如果由于 JMS 错误 JMS 而无法获取该浏览器的消息选择器。

getEnumeration

```
public java.util.Enumeration getEnumeration() throws JMSException
```

获取按将要接收的顺序浏览当前队列消息的枚举。

返回: 浏览消息的枚举。

抛出: JMSException - 如果由于 JMS 错误 JMS 而无法获取该浏览器的枚举。

注: 如果为不存在的队列创建了浏览器，则直到第一次调用 getEnumeration 时才会检测到它。

关闭

```
public void close() throws JMSException
```

因为供应商可能会代表 QueueBrowser 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

掷出: JMSException - 如果由于 JMS 错误 JMS 而无法关闭该 Browser。

QueueConnection

public interface **QueueConnection**
extends **Connection**
子接口: **XAQueueConnection**

MQSeries 类: **MQQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
```

QueueConnection 是到 JMS 点到点供应商的活动连接。客户机使用 QueueConnection 创建一个或多个生成和使用消息的 QueueSession。

另见: **Connection**、**QueueConnectionFactory** 和 **XAQueueConnection**

方法

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                  throws JMSException
```

创建 QueueSession。

参数:

- transacted - 如果是 true，则该会话是事务性的。
- acknowledgeMode - 表明消息使用者或客户机是否将确认接收到的任何消息。可能的值有:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  如果会话是事务性的，则忽略此参数。

返回: 新创建的队列会话。

掷出: JMSException - 如果由于内部错误或缺少特定事务支持和确认方式 JMS Connection 无法创建会话。

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                                    (Queue queue,
                                     java.lang.String messageSelector,
                                     ServerSessionPool sessionPool,
                                     int maxMessages)
                                  throws JMSException
```

创建该连接的连接使用者。这是不由常规 JMS 客户机使用的专门设施。

参数:

- queue - 要访问的队列。
- messageSelector - 仅传递带有与消息选择器表达式匹配的特性的消息。
- sessionPool - 与该连接使用者关联的服务器会话池。
- maxMessages - 在某一时刻可以分配给服务器会话的最大消息数。

返回: 连接使用者。

掷出:

- JMSException - 如果由于内部错误或 sessionPool 和 messageSelector 的无效变量 JMS Connection 无法创建连接使用者。
- InvalidSelectorException - 如果消息选择器无效。

另见: ConnectionConsumer

close *

```
public void close() throws JMSException
```

覆盖: 类 MQConnection 中的 close。

QueueConnectionFactory

public interface **QueueConnectionFactory**
extends **ConnectionFactory**
子接口: **XAQueueConnectionFactory**

MQSeries 类: **MQQueueConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
```

使用 QueueConnectionFactory 创建带有点到点 JMS 供应商的 QueueConnection 的客户机。

另见: **ConnectionFactory** 和 **XAQueueConnectionFactory**

MQSeries 构造器

MQQueueConnectionFactory

```
public MQQueueConnectionFactory()
```

方法

createQueueConnection

```
public QueueConnection createQueueConnection()
                                            throws JMSException
```

创建使用缺省用户标识的队列连接。使用停止方式来创建该连接。直到当显式地调用了 Connection.start 方法后，才能发送消息。

返回: 新创建的队列连接。

掷出:

- JMSException - 如果由于内部错误 JMS Provider 无法创建“队列连接”。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

createQueueConnection

```
public QueueConnection createQueueConnection
                                 (java.lang.String userName,
                                  java.lang.String password)
                                                  throws JMSException
```

创建使用指定用户标识的队列连接。

注: 该方法只能用于传递 类型 JMSC.MQJMS_TP_CLIENT_MQ_TCPIP （参阅 ConnectionFactory）。使用停止方式来创建该连接。直到显式地调用了 Connection.start 方法后，才能发送消息。

参数:

- userName - 调用方的用户名。
- password - 调用方的口令。

返回: 新创建的队列连接。

掷出:

- JMSException - 如果由于内部错误 JMS Provider 无法创建“队列连接”。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

setTemporaryModel *

```
public void setTemporaryModel(String x) throws JMSException
```

getTemporaryModel *

```
public String getTemporaryModel()
```

getReference *

```
public Reference getReference() throws NamingException
```

创建对该连接工厂的引用。

返回: 对该对象的引用。

掷出: NamingException。

setMessage Retention*

```
public void setMessageRetention(int x) throws JMSException
```

messageRetention 属性的设置方法。

参数: 有效的值有:

- JMSC.MQJMS_MRET_YES - 不需要的消息保留在输入队列中。
- JMSC.MQJMS_MRET_NO - 根据配置选项来处理不需要的消息。

getMessage Retention*

```
public void getMessageRetention()
```

获取 messageRetention 属性的方法。

返回:

- JMSC.MQJMS_MRET_YES - 不需要的消息保留在输入队列中。
- JMSC.MQJMS_MRET_NO - 根据配置选项来处理不需要的消息。

QueueReceiver

public interface **QueueReceiver**
extends **MessageConsumer**

MQSeries 类: **MQQueueReceiver**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

客户机使用 QueueReceiver 接收已经传递到队列的消息。

另见: **MessageConsumer**

该类从 **MQMessageConsumer** 继承了下列方法。
- receive
- receiveNoWait
- close
- getMessageListener
- setMessageListener

方法

getQueue

```
public Queue getQueue() throws JMSException
```

获取与该队列接收方关联的队列。

返回: 队列。

抛出: JMSException - 如果 JMS 由于内部的错误而无法获取该队列接收方的队列。

QueueRequestor

public class **QueueRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.QueueRequestor
```

JMS 提供了 QueueRequestor 帮助程序类以简化对服务的请求。QueueRequestor 构造器给出了一个非事务性的 QueueSession 以及一个目的 Queue。它为响应创建了一个 TemporaryQueue，并提供 request() 方法用以意发送请求消息并等待其回答。用户可以自由创建更高级的版本。

另见：**TopicRequestor**

构造器

QueueRequestor

```
public QueueRequestor(QueueSession session,
                      Queue queue)
                      throws JMSException
```

这个实现假设，会话参数是非事务性的，并且是 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE。

参数:

- session - 队列所属的队列会话。
- queue - 要执行请求／响应调用的队列。

掷出: JMSException - 如果发生 JMS 错误。

方法

request

```
public Message request(Message message)
                       throws JMSException
```

发出请求并等待回答。对 replyTo 使用临时队列，并且每个请求只有一个回答。

参数: message - 要发送的消息。

返回: 回答消息。

掷出: JMSException - 如果发生 JMS 错误。

close

```
public void close() throws JMSException
```

因为供应商可能会代表 QueueRequestor 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

注: 这个方法关闭了传递给 QueueRequestor 构造器的 Session 对象。

掷出: JMSException - 如果发生 JMS 错误。

QueueSender

public interface **QueueSender**
extends **MessageProducer**

MQSeries 类: **MQQueueSender**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

客户机使用 QueueSender 将消息发送到队列。

QueueSender 通常与特定的 Queue 关联。但是，也可以创建一个不与任何给定 Queue 关联的未标识的 QueueSender。

另见: **MessageProducer**

方法

getQueue

```
public Queue getQueue() throws JMSException
```

获取与该队列发送方关联的队列。

返回: 队列。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取该队列接收方的队列。

send

```
public void send(Message message) throws JMSException
```

发送消息到队列。使用 QueueSender 的缺省发送方式、存活时间和优先级。

参数：

message - 要发送的消息。

掷出:

- JMSException - 如果 JMS 由于出错而无法发送消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机同对这个方法使用了带有无效队列的 Queue 发送方。

send

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

将消息按指定的发送方式、优先级和存活时间发送到队列。

参数：

- message - 要发送的消息。

- deliveryMode - 要使用的发送方式。
- priority - 该消息的优先级。
- timeToLive - 消息的生命周期（单位: 毫秒）。

掷出:

- JMSException - 如果 JMS 由于内部的错误而无法发送消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机同时使用了这个方法以及带有无效队列的 QueueSender。

send

```
public void send(Queue queue,
                 Message message) throws JMSException
```

使用 QueueSender 的缺省发送方式、存活时间和优先级将消息发送到指定的队列。

注: 这个方法只能和未标识的 QueueSenders 一起使用。

参数:

- queue - 消息将发送到的队列。
- message - 要发送的消息。

掷出:

- JMSException - 如果 JMS 由于内部的错误而无法发送消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机同时使用了这个方法以及无效队列。

send

```
public void send(Queue queue,
                 Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

使用发送方式、优先级和存活时间将消息发送到指定的队列。

注: 这个方法只能和未标识的 QueueSenders 一起使用。

参数:

- queue - 消息将发送到的队列。
- message - 要发送的消息。
- deliveryMode - 要使用的发送方式。
- priority - 该消息的优先级。
- timeToLive - 消息的生命周期（单位: 毫秒）。

掷出:

- JMSException - 如果 JMS 由于内部的错误而无法发送消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机同时使用了这个方法以及无效队列。

close *

```
public void close() throws JMSException
```

| 因为供应商可能会代表 QueueSender 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

掷出: JMSException，如果 JMS 因某些错误而无法关闭生产者。

覆盖: 类 MQMessageProducer 中的 close。

QueueSession

public interface **QueueSession**
extends **Session**

MQSeries 类: **MQQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

QueueSession 提供创建 QueueReceiver、QueueSender、QueueBrowser 和 TemporaryQueue 的方法。

另见: **Session**

以下方法从 **MQSession** 继承:
- close
- commit
- rollback
- recover

方法

createQueue

```
public Queue createQueue(java.lang.String queueName)
                                                     throws JMSException
```

用给定“队列名”创建 Queue。这将允许用供应商的特定名称来创建队列。字符串采用 URI 格式，第171页中有所描述。

注: 依赖此性能的客户机不可移植。

参数:
queueName - 队列的名称。

返回: 使用给定名称的 Queue。

掷出: JMSException - 如果由于 JMS 错误会话无法创建队列。

createReceiver

```
public QueueReceiver createReceiver(Queue queue)
                                                     throws JMSException
```

创建 QueueReceiver 从指定队列接收消息。

参数: queue - 要访问的队列。

掷出:
- JMSException - 如果由于 JMS 错误会话无法创建接收方。
- InvalidDestinationException - 如果指定了无效的 Queue。

createReceiver

```
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
                                    throws JMSException
```

创建 QueueReceiver 从指定队列接收消息。

参数:

- queue - 要访问的队列。
- messageSelector - 只传递带有与消息选择器表达式匹配的特性的消息。

掷出:

- JMSException - 如果由于 JMS 错误会话无法创建接收方。
- InvalidDestinationException - 如果指定了无效的 Queue。
- InvalidSelectorException - 如果消息选择器无效。

createSender

```
public QueueSender createSender(Queue queue)
                                throws JMSException
```

创建 QueueSender 从指定队列发送消息。

参数: queue - 要访问的队列，如果生产者未经确认，则为空。

掷出:

- JMSException - 如果由于 JMS 错误会话无法创建发送方。
- InvalidDestinationException - 如果指定了无效的 Queue。

createBrowser

```
public QueueBrowser createBrowser(Queue queue)
                                  throws JMSException
```

创建 QueueBrowser 以查看指定队列上的消息。

参数: queue - 要访问的队列。

掷出:

- JMSException - 如果由于 JMS 错误会话无法创建浏览器。
- InvalidDestinationException - 如果指定了无效的 Queue。

createBrowser

```
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
                                  throws JMSException
```

创建 QueueBrowser 以查看指定队列上的消息。

参数:

- queue - 要访问的队列。
- messageSelector - 只传递带有与消息选择器表达式匹配的特性的消息。

掷出:

- JMSException - 如果由于 JMS 错误会话无法创建浏览器。
- InvalidDestinationException - 如果指定了无效的 Queue。

- InvalidSelectorException - 如果消息选择器无效。

createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                    throws JMSException
```

创建一个临时队列。它的生命周期将和 QueueConnection 一样，除非在之前删除了它。

返回: 一个临时队列。

抛出: JMSException - 如果由于 JMS 错误以致会话无法创建一个“临时队列”。

Session

public interface **Session**
extends **java.lang.Runnable**
子接口：**QueueSession**、**TopicSession**、**XAQueueSession**、**XASession** 和 **XATopicSession**

MQSeries 类：**MQSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

JMS Session 是生成和使用消息的单线程上下文。

另见：**QueueSession**、**TopicSession**、**XAQueueSession**、**XASession** 和 **XATopicSession**

字段

AUTO_ACKNOWLEDGE

```
public static final int AUTO_ACKNOWLEDGE
```

使用这种确认方式，当消息是从要接收的调用成功返回，或是由调用它的侦听器来处理消息的成功返回时，会话将自动确认消息。

CLIENT_ACKNOWLEDGE

```
public static final int CLIENT_ACKNOWLEDGE
```

使用确认方式，客户机通过调用消息的确认方式来确认消息。

DUPS_OK_ACKNOWLEDGE

```
public static final int DUPS_OK_ACKNOWLEDGE
```

该确认方式指示会话将缓慢地确认消息的发送。

方法

createBytesMessage

```
public BytesMessage createBytesMessage()
                                throws JMSException
```

创建一个 BytesMessage。 BytesMessage 用于发送包含未解释字节流的消息。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

createMapMessage

```
public MapMessage createMapMessage() throws JMSException
```

创建一个 MapMessage。MapMessage 用于发送自定义的名-值对，其中名称为 String，值为 Java 原始类型。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

createMessage

```
public Message createMessage() throws JMSException
```

创建一个 Message。Message 接口是所有 JMS 消息的根接口。它含有所有的标准消息头信息。当消息仅包含头信息就够时，可以发送它。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

createObjectMessage

```
public ObjectMessage createObjectMessage()
                                 throws JMSException
```

创建一个 ObjectMessage。 ObjectMessage 用于发送包含可串行化 Java 对象的消息。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

createObjectMessage

```
public ObjectMessage createObjectMessage
                                (java.io.Serializable object)
                                 throws JMSException
```

创建一个初始的 ObjectMessage。ObjectMessage 用于发送包含可串行化 Java 对象的消息。

参数 :

object - 将用于初始化消息的对象。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

createStreamMessage

```
public StreamMessage createStreamMessage()
                                  throws JMSException
```

创建一个 StreamMessage。 StreamMessage 用于发送自定义的 Java 原始流。

掷出: JMSException，如果 JMS 由于内部错误而无法创建消息。

createTextMessage

```
public TextMessage createTextMessage() throws JMSException
```

创建一个 TextMessage。TextMessage 用于发送包含 String 的消息。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

createTextMessage

```
public TextMessage createTextMessage
                       (java.lang.String string)
                       throws JMSException
```

创建一个初始的 TextMessage。TextMessage 用于发送包含 String 的消息。

参数 :

string - 用于初始化消息的字符串。

掷出: JMSException - 如果 JMS 由于内部错误而无法创建消息。

getTransacted

```
public boolean getTransacted() throws JMSException
```

会话是否处于事务性方式?

返回: true，如果消息处于事务性方式。

掷出: JMSException - 如果 JMS 由于 JMS 供应商内部的错误而无法返回事务性方式。

commit

```
public void commit() throws JMSException
```

提交在该事务中执行的所有消息，并解开当前拥有的任何锁。

掷出:

- JMSException - 如果 JMS 由于内部错误而无法提交事务。
- TransactionRolledBackException - 如果事务由于提交期间的内部错误而复原。

rollback

```
public void rollback() throws JMSException
```

逆序恢复在该事务中执行的任何消息，并解开当前拥有的任何锁。

掷出: JMSException - 如果由于内部错误 JMS 实现无法复原事务。

close

```
public void close() throws JMSException
```

因为供应商可能会代表 Session 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

关闭事务性会话并复原正在处理的事务。关闭会话将自动关闭它的消息生产者和使用者，所以没有必要单独关闭它们。

掷出: JMSException - 如果由于内部错误 JMS 实现无法关闭 Session。

recover

```
public void recover() throws JMSException
```

停止该会话中的消息发送，并重新发送最老的未确认消息。

掷出: JMSException - 如果由于内部错误 JMS 实现无法停止消息发送以及重新开始消息发送。

getMessageListener

```
public MessageListener getMessageListener()
                                throws JMSException
```

| 返回会话的专有消息侦听器。

返回: 与该会话关联的消息侦听器。

掷出: JMSException - 如果 JMS 由于 JMS 供应商内部的错误而无法获取消息侦听器。

另见：

setMessageListener

setMessageListener

```
public void setMessageListener(MessageListener listener)
                                throws JMSException
```

设置会话的专有消息侦听器。设置后，会话中将不能使用其它任何形式的收据。但仍然支持发送消息的所有形式。

| 这是非常规 JMS 客户机使用的专门设施。

参数：

listener - 与该会话关联的消息侦听器。

掷出: JMSException - 如果 JMS 由于 JMS 供应商内部的错误而无法设置消息侦听器。

另见：

| getMessageListener、ServerSessionPool、ServerSession

run

```
public void run()
```

| 只有应用程序服务器才能使用这个方法。

| **指定:** 在接口 java.lang.Runnable 中运行

另见：

ServerSession

StreamMessage

public interface **StreamMessage**
extends **Message**

MQSeries 类: **JMSStreamMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

StreamMessage 用于发送 Java 原语流。

另见: **BytesMessage**、**MapMessage**、**Message**、**ObjectMessage** 和 **TextMessage**

方法

readBoolean

```
public boolean readBoolean() throws JMSException
```

从流消息中读取 boolean。

返回: 读取的 boolean 值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readByte

```
public byte readByte() throws JMSException
```

从流消息中读取字节值。

返回: 流消息中下一个字节作为带符号的 8 位字节。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readShort

```
public short readShort() throws JMSException
```

从流消息中读取 16 位数字。

返回: 流消息中的 16 位数字。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readChar

```
public char readChar() throws JMSException
```

从流消息中读取 Unicode 字符值。

返回: 流消息中的 Unicode 字符。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readInt

```
public int readInt() throws JMSException
```

从流消息中读取 32 位数字。

返回: 流消息中的 32 位整数值，按 int 解释。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readLong

```
public long readLong() throws JMSException
```

从流消息中读取 64 位数字。

返回: 流消息中的 64 位整数值，按 long 解释。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果是消息流的结束部分
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readFloat

```
public float readFloat() throws JMSException
```

从流消息中读取一个 float。

返回: 流消息中的一个 float。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果是消息流的结束部分
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readDouble

```
public double readDouble() throws JMSException
```

从流消息中读取一个 double。

返回: 流消息中的一个 double 的值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readString

```
public java.lang.String readString() throws JMSException
```

从流消息中读取一个 String。

返回: 流消息中的 Unicode String。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readBytes

```
public int readBytes(byte[] value)
              throws JMSExceptioneam message.
```

从流消息中读取 byte 数组字段到指定的 byte[] 对象（读缓冲区）。如果缓冲区大小小于或等于消息字段中数据的大小，则应用程序必须进一步调用此方法来检索余下的数据。一旦 readByte 第一次调用 byte[]字段值完毕，就必须在正确读取下一字段之前读取全部的字段值。如果尝试在调用完毕前就读取下一字段，将掷出 MessageFormatException。

参数: value - 从中读取数据的缓冲区。

返回: 读入缓冲区的字节总数，如果没有更多数据（因为已达到了结束字节），则返回 -1。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- MessageFormatException - 如果该类型转换无效。
- MessageNotReadableException - 如果消息处于只写方式。

readObject

```
public java.lang.Object readObject() throws JMSException
```

从流消息中读取一个 Java 对象。

返回: 流消息中指定了对象格式的一个 Java 对象（例如，如果设置为一个 int，则返回一个 Integer）。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageEOFException - 如果接收到消息流的结束部分。
- NotReadableException - 如果消息处于只写方式。

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

将一个 boolean 写入流消息。

参数: value - 要写入的 boolean 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法读取消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeByte

```
public void writeByte(byte value) throws JMSException
```

将字节写入流消息。

参数: value - 要写入的字节值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeShort

```
public void writeShort(short value) throws JMSException
```

将 short 写入流消息。

参数: value - 要写入的 short。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeChar

```
public void writeChar(char value) throws JMSException
```

将 char 写入流消息。

参数: value - 要写入的 char 值。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeInt

```
public void writeInt(int value) throws JMSException
```

将 int 写入流消息。

参数: value - 要写入的 int。

抛出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeLong

```
public void writeLong(long value) throws JMSException
```

将 long 写入流消息。

参数: value - 要写入的 long。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeFloat

```
public void writeFloat(float value) throws JMSException
```

将 float 写入流消息。

参数: value - 要写入的 float 的值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeDouble

```
public void writeDouble(double value) throws JMSException
```

将 double 写入流消息。

参数: value - 要写入的 double 的值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeString

```
public void writeString(java.lang.String value)
                                          throws JMSException
```

将 string 写入流消息。

参数: value - 要写入的 String 值。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

将字节数组写入流消息。

参数: value - 要写入的字节数组。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeBytes

```
public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException
```

将 byte 数组的一部分写入流消息。

参数:

- value - 要写入的 byte 数组值。
- offset - 字节数组内的初始偏移量。
- length - 要使用的字节数。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。

writeObject

```
public void writeObject(java.lang.Object value)
                                     throws JMSException
```

将 Java 对象写入流消息。此方法仅对对象原始类型（如 Integer、Double 和 Long）、String 以及 byte 数组起作用。

参数: value - 要写入的 Java 对象。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法写消息。
- MessageNotWriteableException - 如果消息处于只读方式。
- MessageFormatException - 如果对象无效。

reset

```
public void reset() throws JMSException
```

让消息主体处于只读方式，并将流重新定位到一开始。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法复位消息。
- MessageFormatException - 如果消息含有无效格式。

TemporaryQueue

public interface **TemporaryQueue**
extends **Queue**

MQSeries 类: **MQTemporaryQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

TemporaryQueue 是在 QueueConnection 持续期间而创建的唯一 Queue 对象。

方法

delete

```
public void delete() throws JMSException
```

删除临时队列。如果仍然存在发送方，或接收方仍然在使用它，则掷出一个 JMSException。

掷出: JMSException - 如果由于内部错误 JMS 实现无法删除 TemporaryQueue。

TemporaryTopic

public interface **TemporaryTopic**
extends **Topic**

MQSeries 类: **MQTemporaryTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

TemporaryTopic 是在 TopicConnection 持续期间创建的唯一 Topic 对象，并且只能通过连接使用者才能取得。

MQSeries 构造器

MQTemporaryTopic

```
MQTemporaryTopic() throws JMSException
```

方法

delete

```
public void delete() throws JMSException
```

删除临时主题。如果仍然存在发布者，或订户仍然在使用它，则掷出一个 JMSException。

掷出: JMSException - 如果由于内部错误导致 JMS 实现无法删除 TemporaryTopic。

TextMessage

public interface **TextMessage**
extends **Message**

MQSeries 类: **JMSTextMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

TextMessage 用于发送包含 java.lang.String 的消息。它继承自 Message 并添加了文本消息主体。

另见: **BytesMessage**、**MapMessage**、**Message**、**ObjectMessage** 和 **StreamMessage**

方法

setText

```
public void setText(java.lang.String string)
                                           throws JMSException
```

设置包含消息数据的字符串。

参数: string - 包含消息数据的“字符串”。

掷出:

- JMSException - 如果 JMS 由于内部的 JMS 错误而无法设置文本。
- MessageNotWriteableException - 如果消息处于只读方式。

getText

```
public java.lang.String getText() throws JMSException
```

获取包含消息数据的字符串。缺省值是空。

返回: 包含消息数据的“字符串”。

掷出: JMSException - 如果 JMS 由于内部的 JMS 错误而无法获取文本。

Topic

public interface **Topic**
extends **Destination**
子接口: **TemporaryTopic**

MQSeries 类: **MQTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

Topic 对象封装供应商指定的主题名称。这是客户机向 JMS 方法指定主题标识的方法。

另见: **Destination**

MQSeries 构造器

MQTopic

```
public MQTopic()
public MQTopic(string URItopic)
```

见 **TopicSession.createTopic**。

方法

getTopicName

```
public java.lang.String getTopicName() throws JMSException
```

获取 URI 格式的主题名称。(第180页的『在运行时创建主题』中描述了 URI 格式。)

注: 依赖此名称的客户机不能移植。

返回: 主题名称。

抛出: JMSException - 如果 Topic 的 JMS 实现因内部错误而无法返回主题名称。

toString

```
public String toString()
```

返回打印良好的 Topic 名称版本。

返回: 该 Topic 的供应商特定标识值。

覆盖: 类对象中的 toString。

getReference *

```
public Reference getReference()
```

创建对该主题的引用。

返回: 对该对象的引用。

掷出: NamingException。

setBaseTopicName *

```
public void setBaseTopicName(String x)
```

基本 MQSeries 主题名称的设置方法。

getBaseTopicName *

```
public String getBaseTopicName()
```

基本 MQSeries 主题名称的获取方法。

setBrokerDurSubQueue *

```
public void setBrokerDurSubQueue(String x) throws JMSException
```

brokerDurSubQueue 属性的设置方法。

参数: brokerDurSubQueue - 要使用的长期订阅队列的名称。

getBrokerDurSubQueue *

```
public String getBrokerDurSubQueue()
```

brokerDurSubQueue 属性的获取方法。

返回: 要使用的长期订阅队列（brokerDurSubQueue）的名称。

setBrokerCCDurSubQueue *

```
public void setBrokerCCDurSubQueue(String x) throws JMSException
```

brokerCCDurSubQueue 属性的设置方法。

参数: brokerCCDurSubQueue - 要用于 ConnectionConsumer 的长期订阅队列的名称。

getBrokerCCDurSubQueue *

```
public String getBrokerCCDurSubQueue()
```

brokerCCDurSubQueue 属性的获取方法。

返回: 要用于 ConnectionConsumer 的长期订阅队列 (brokerCCDurSubQueue) 的名称。

TopicConnection

public interface **TopicConnection**
extends **Connection**
子接口: **XATopicConnection**

MQSeries 类: **MQTopicConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
```

TopicConnection 是到 JMS Publish/Subscribe 供应商的活动连接。

另见: **Connection**、**TopicConnectionFactory** 和 **XATopicConnection**

方法

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                       throws JMSException
```

创建 TopicSession。

参数:

- transacted - 如果是 true，则该会话是事务性的。
- acknowledgeMode - 以下之一:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  表明消息使用者或客户机是否将确认接收到的任何消息。如果会话是事务性的，则忽略此参数。

返回: 新创建的主题会话。

掷出: JMSException - 如果由于内部错误 JMS Connection 无法创建会话，或是缺少对指定事务和确认方式的支持。

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                              (Topic topic,
                               java.lang.String messageSelector,
                               ServerSessionPool sessionPool,
                               int maxMessages)
                                        throws JMSException
```

创建该连接的连接使用者。这是不由常规 JMS 客户机使用的专门设施。

参数:

- topic - 要访问的主题。
- messageSelector - 只传送带有与消息选择器表达式匹配的特性的消息。
- sessionPool - 与该连接使用者关联的服务器会话池。
- maxMessages - 在某一时刻可以分配给服务器会话的最大消息数。

返回: 连接使用者。

抛出:

- JMSException - 如果由于内部错误或 sessionPool 的无效变量 JMS Connection 无法创建连接使用者。
- InvalidSelectorException - 如果消息选择器无效。

另见: ConnectionConsumer

createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer
                                (Topic topic,
                                 java.lang.String subscriptionName
                                 java.lang.String messageSelector,
                                 ServerSessionPool sessionPool,
                                 int maxMessages)
                                          throws JMSException
```

| 创建该连接的长期连接使用者。这是不由常规 JMS 客户机使用的专门设施。

参数:

- topic - 要访问的主题。
- subscriptionName - 长期订阅的名称。
- messageSelector - 只传送带有与消息选择器表达式匹配的特性的消息。
- sessionPool - 与该长期连接使用者关联的服务器会话池。
- maxMessages - 在某一时刻可以分配给服务器会话的最大消息数。

返回: 长期连接使用者。

抛出:

- JMSException - 如果由于内部错误或 sessionPool 和 messageSelector 的无效变量 JMS Connection 无法创建连接使用者。
- | InvalidSelectorException - 如果消息选择器无效。

另见: ConnectionConsumer

TopicConnectionFactory

public interface **TopicConnectionFactory**
extends **ConnectionFactory**
子接口： **XATopicConnectionFactory**

MQSeries 类： **MQTopicConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
```

客户机使用 TopicConnectionFactory 创建带有 JMS 发布／订阅供应商的 TopicConnection。

另见： **ConnectionFactory** 和 **XATopicConnectionFactory**

MQSeries 构造器

MQTopicConnectionFactory

```
public MQTopicConnectionFactory()
```

方法

createTopicConnection

```
public TopicConnection createTopicConnection()
                                     throws JMSException
```

创建使用缺省用户标识的主题连接。使用停止方式来创建该连接。直到当显式地调用了 Connection.start 方法后，才能发送消息。

返回: 新创建的主题连接。

掷出:

- JMSException - 如果由于内部错误 JMS Provider 无法创建“主题连接”。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

createTopicConnection

```
public TopicConnection createTopicConnection
                                     (java.lang.String userName,
                                     java.lang.String password)
                                     throws JMSException
```

创建使用指定用户标识的主题连接。使用停止方式来创建该连接。直到当显式地调用了 Connection.start 方法后，才能发送消息。

注: 此方法只有在传送类型为 IBM_JMS_TP_CLIENT_MQ_TCPIP 时才有效。见 ConnectionFactory。

参数:

- userName - 调用方的用户名。
- password - 调用方的口令。

返回: 新创建的主题连接。

抛出:

- JMSException - 如果由于内部错误 JMS Provider 无法创建“主题连接”。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

setBrokerControlQueue *

```
public void setBrokerControlQueue(String x) throws JMSException
```

brokerControlQueue 属性的设置方法。

参数: brokerControlQueue - 代理控制队列的名称。

getBrokerControlQueue *

```
public String getBrokerControlQueue()
```

brokerControlQueue 属性的获取方法。

返回: 代理的控制队列名

setBrokerQueueManager *

```
public void setBrokerQueueManager(String x) throws JMSException
```

brokerQueueManager 属性的设置方法。

参数: brokerQueueManager - 代理的“队列管理器”名称。

getBrokerQueueManager *

```
public String getBrokerQueueManager()
```

brokerQueueManager 属性的获取方法。

返回: 代理的队列管理器名称。

setBrokerPubQueue *

```
public void setBrokerPubQueue(String x) throws JMSException
```

brokerPubQueue 属性的设置方法。

参数: brokerPubQueue - 代理发布队列的名称。

getBrokerPubQueue *

```
public String getBrokerPubQueue()
```

brokerPubQueue 属性的获取方法。

返回: 代理的发布队列名称。

|
setBrokerSubQueue *

|
```
public void setBrokerSubQueue(String x) throws JMSException
```

| brokerSubQueue 属性的设置方法。

| **参数**: brokerSubQueue - 要使用的非长期订阅队列的名称。

| **getBrokerSubQueue ***

```
public String getBrokerSubQueue()
```

| brokerSubQueue 属性的获取方法。

| **返回**: 要使用的非长期订阅队列的名称。

| **setBrokerCCSubQueue ***

```
public void setBrokerCCSubQueue(String x) throws JMSException
```

| brokerCCSubQueue 属性的设置方法。

| **参数**: brokerSubQueue - 要用于 ConnectionConsumer 的非长期订阅队列的名
| 称。

| **getBrokerCCSubQueue ***

```
public String getBrokerCCSubQueue()
```

| brokerCCSubQueue 属性的获取方法。

| **返回**: 要用于 ConnectionConsumer 的非长期订阅队列的名称。

| **setBrokerVersion ***

```
public void setBrokerVersion(int x) throws JMSException
```

brokerVersion 属性的设置方法。

参数: brokerVersion - 代理的版本号。

getBrokerVersion *

```
public int getBrokerVersion()
```

brokerVersion 属性的获取方法。

返回: 代理的版本号。

getReference *

```
public Reference getReference()
```

返回对主题连接工厂的引用。

返回: 对主题连接工厂的引用。

掷出: NamingException。

TopicPublisher

public interface **TopicPublisher**
extends **MessageProducer**

MQSeries 类: **MQTopicPublisher**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

客户机使用 TopicPublisher 在主题上发布消息。TopicPublisher 是 JMS 消息生产者的 Pub/Sub 变体。

方法

getTopic

```
public Topic getTopic() throws JMSException
```

获取与此发布者关联的主题。

返回: 发布者的主题

抛出: JMSException - 如果 JMS 由于内部的错误而无法获取该主题发布者的主题。

publish

```
public void publish(Message message) throws JMSException
```

“使用”主题的缺省发送方式、存活时间和优先级将“消息”发布到主题。

参数: message - 要发布的消息

抛出:

- JMSException - 如果 JMS 由于内部错误而无法发布消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机同时使用了这个方法以及带有无效主题的“主题发布者”。

publish

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

将“消息”发布到指定了发送方式、优先级和存活时间的主题。

参数:

- message - 要发布的消息。
- deliveryMode - 要使用的发送方式。
- priority - 该消息的优先级。
- timeToLive - 消息的生命周期（单位: 毫秒）。

掷出:

- JMSException - 如果 JMS 由于内部错误而无法发布消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机同时使用了这个方法以及带有无效主题的 “主题发布者”。

publish

```
public void publish(Topic topic,
                    Message message) throws JMSException
```

将“消息”发布到一个未经确定的消息生产者的主题。使用主题的缺省发送方式、存活时间和优先级。

参数:

- topic - 要发布消息的主题。
- message - 要发送的消息。

掷出:

- JMSException - 如果 JMS 由于内部错误而无法发布消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机使用了带有无效主题的方法。

publish

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

将“消息”发布到指定了发送方式、优先级和存活时间的一个未经确定的消息生产者的主题。

参数:

- topic - 要发布消息的主题。
- message - 要发送的消息。
- deliveryMode - 要使用的发送方式。
- priority - 该消息的优先级。
- timeToLive - 消息的生命周期（单位: 毫秒）。

掷出:

- JMSException - 如果 JMS 由于内部错误而无法发布消息。
- MessageFormatException - 如果指定了无效的消息。
- InvalidDestinationException - 如果客户机使用了带有无效主题的方法。

close *

```
public void close() throws JMSException
```

| 因为供应商可能会代表 TopicPublisher 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

掷出: JMSException，如果 JMS 因错误而无法关闭生产者。

覆盖: 类 MQMessageProducer 中的 close。

TopicRequestor

public class **TopicRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.TopicRequestor
```

JMS 提供这个 TopicRequestor 类是为了帮助执行服务请求。

TopicRequestor 构造器给出了一个非事务性的 TopicSession 以及一个目的地 Topic。它为响应创建了一个 TemporaryTopic，并提供 request() 方法用以发送请求消息并等待其回答。用户可以自由创建更有效的版本。

构造器

TopicRequestor

```
public TopicRequestor(TopicSession session,
                      Topic topic) throws JMSException
```

TopicRequestor 类的构造器。这个实现假设，会话参数是非事务性的，并且是 AUTO_ACKNOWLEDGE 或 DUPS_OK_ACKNOWLEDGE。

参数:

- session - 主题所属的主题会话。
- topic - 要执行请求／响应调用的主题。

掷出: JMSException - 若发生 JMS 错误。

方法

request

```
public Message request(Message message) throws JMSException
```

发出请求并等待回答。

参数: message - 要发送的消息。

返回: 回答消息。

掷出: JMSException - 若发生 JMS 错误。

close

```
public void close() throws JMSException
```

因为供应商可能会代表 TopicRequestor 来分配 JVM 之外的一些资源，所以在不需要这些资源时，客户机应关闭它们。不能依赖无用信息收集到最后才回收这些资源，因为这样不够及时。

注: 这个方法关闭了传递给 TopicRequestor 构造器的 Session 对象。

掷出: JMSException - 若发生 JMS 错误。

TopicSession

public interface **TopicSession**
extends **Session**

MQSeries 类: **MQTopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

TopicSession 提供了创建 TopicPublishers、TopicSubscribers 和 TemporaryTopics 的方法。

另见: **Session**

MQSeries 构造器

MQTopicSession

```
public MQTopicSession(boolean transacted,
                      int acknowledgeMode) throws JMSException
```

见 **TopicConnection.createTopicSession**。

方法

createTopic

```
public Topic createTopic(java.lang.String topicName)
                                          throws JMSException
```

创建给出 URI 格式的 Topic 名称的 Topic。（第180页的『在运行时创建主题』中描述了 URI 格式。）这将允许用供应商的特定名称来创建主题。

注: 依赖此能力的客户机不可移植。

参数: topicName - 主题的名称。

返回: 使用给定名称的 Topic。

掷出: JMSException - 如果由于 JMS 错误而无法创建主题。

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic)
                                          throws JMSException
```

创建指定主题的非长期 Subscriber。

参数: topic - 要订阅的主题

掷出:

- JMSException - 如果由于 JMS 错误而无法创建订户。
- InvalidDestinationException - 如果指定了无效的 Topic。

createSubscriber

```
public TopicSubscriber createSubscriber
                                 (Topic topic,
                                  java.lang.String messageSelector,
                                  boolean noLocal) throws JMSException
```

创建指定主题的非长期 Subscriber。

参数:

- topic - 要订阅的主题。
- messageSelector - 只传送带有与消息选择器表达式匹配的特性的消息。这个值可能是空。
- noLocal - 如果设置，则禁止发送由其连接发布的消息。

掷出:

- JMSException - 如果由于 JMS 错误或无效的选择器而无法创建订户。
- InvalidDestinationException - 如果指定了无效的 Topic。
- InvalidSelectorException - 如果消息选择器无效。

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                        (Topic topic,
                                         java.lang.String name) throws JMSException
```

创建指定主题的长期 Subscriber。客户机可以通过使用相同的名称以及新的主题和／或消息选择器来创建“长期订户”，从而更改现有的长期订户。

参数:

- topic - 要订阅的主题。
- name - 用于标识该订阅的名称。

掷出:

- JMSException - 如果由于 JMS 错误而无法创建订户。
- InvalidDestinationException - 如果指定了无效的 Topic。

见 **TopicSession.unsubscribe**

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                        (Topic topic,
                                         java.lang.String name,
                                         java.lang.String messageSelector,
                                         boolean noLocal)  throws JMSException
```

创建指定主题的长期 Subscriber。

参数:

- topic - 要订阅的主题。
- name - 用于标识该订阅的名称。
- messageSelector - 只传送带有与消息选择器表达式匹配的特性的消息。这个值可能是空。
- noLocal - 如果设置，则禁止发送由其连接发布的消息。

掷出:

- JMSException - 如果由于 JMS 错误或无效的选择器而无法创建订户。
- InvalidDestinationException - 如果指定了无效的 Topic。
- InvalidSelectorException - 如果消息选择器无效。

createPublisher

```
public TopicPublisher createPublisher(Topic topic)
                                   throws JMSException
```

为指定主题创建一个 Publisher。

参数: topic - 要发布的主题，如果是未经确认的生产者，则为空。

掷出:

- JMSException - 如果由于 JMS 错误而无法创建发布者。
- InvalidDestinationException - 如果指定了无效的 Topic。

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                     throws JMSException
```

创建一个临时主题。它的生命周期将和 TopicConnection 一样，除非在之前删除了它。

返回: 临时主题。

掷出: JMSException - 如果由于 JMS 错误会话无法创建临时主题。

unsubscribe

```
public void unsubscribe(java.lang.String name)
                                      throws JMSException
```

取消一个已经由客户机创建的长期订阅。

| **注:** 如果存在活动订阅时，请不要使用这个方法。必须先 close() 订户。

参数: name - 用于标识该订阅的名称。

掷出:

- JMSException - 如果由于内部的 JMS 错误而无法取消长期订阅。
- InvalidDestinationException - 如果指定了无效的 Topic。

TopicSubscriber

public interface **TopicSubscriber**
extends **MessageConsumer**

MQSeries 类: **MQTopicSubscriber**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

客户机使用 TopicSubscriber 接收已经发布到主题上的消息。TopicSubscriber 是 JMS 消息使用者的 Pub/Sub 变体。

另见: **MessageConsumer** 和 **TopicSession.createSubscriber**

MQTopicSubscriber 从 MQMessageConsumer 继承了下列方法:
- close
- getMessageListener
- receive
- receiveNoWait
- setMessageListener

方法

getTopic

```
public Topic getTopic() throws JMSException
```

获取与该订户关联的主题。

返回: 订户的主题。

掷出: JMSException - 如果 JMS 由于内部的错误而无法获取主题订户。

getNoLocal

```
public boolean getNoLocal() throws JMSException
```

获取该 TopicSubscriber 的 NoLocal 属性。这个属性的缺省值为 false.。

返回: 如果要继承本地发布发的消息，则设置为 true。

掷出: JMSException - 如果 JMS 由于内部错误而无法获取主题订户的 NoLocal 属性。

XAConnection

public interface **XAConnection**
子接口：**XAQueueConnection** 和 **XATopicConnection**

MQSeries 类：**MQXAConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnection
```

XAConnection 通过提供 XASession 扩展 Connection 能力。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见：**XAQueueConnection** 和 **XATopicConnection**

XAConnectionFactory

public interface **XAConnectionFactory**
子接口：**XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

MQSeries 类：**MQXAConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnectionFactory
```

有些应用程序服务器为分组 JTS 可用资源以在分布式事务中使用提供支持。要在 JTS 事务中包含 JMS 事务，应用程序服务器需要一个 JTS 已知的供应商。JMS 供应商通过使用 JMS XAConnectionFactory 而受到了 JTS 的支持，应用程序服务器将用它来创建 XASessions。XAConnectionFactories 是由 JMS 管理对象，就如 ConnectionFactories 一样。要求应用程序服务器使用 JNDI 来查找它们。

有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见：**XAQueueConnectionFactory** 和 **XATopicConnectionFactory**

XAQueueConnection

public interface **XAQueueConnection**
extends **QueueConnection** and **XAConnection**

MQSeries 类: **MQXAQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
                   |
                   +----com.ibm.mq.jms.MQXAQueueConnection
```

XAQueueConnection 提供与 QueueConnection 相同的创建选项。根据定义，它们之间唯一不同的是，XAConnection 是事务性的。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见: **XAConnection** 和 **QueueConnection**

方法

createXAQueueSession

```
public XAQueueSession createXAQueueSession()
```

创建一个 XAQueueSession。

掷出: JMSException - 如果由于内部错误 JMS Connection 无法创建 XA 队列会话。

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                              throws JMSException
```

创建一个 QueueSession。

参数:

- transacted - 如果是 true，则该会话是事务性的。
- acknowledgeMode - 表明消息使用者或客户机是否将确认接收到的任何消息。可能的值有:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  如果会话是事务性的，则忽略此参数。

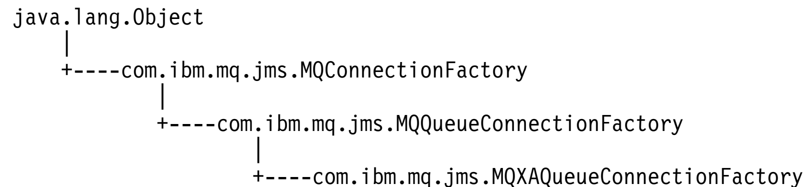
返回: 新建的队列会话（注意，这不是一个 XA 队列会话）。

掷出: JMSException - 如果由于内部错误 JMS Connection 无法创建队列会话。

XAQueueConnectionFactory

public interface **XAQueueConnectionFactory**
extends **QueueConnectionFactory** and **XAConnectionFactory**

MQSeries 类: **MQXAQueueConnectionFactory**



```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQQueueConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

XAQueueConnectionFactory 提供与 QueueConnectionFactory 相同的创建选项。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见: **QueueConnectionFactory** 和 **XAConnectionFactory**

方法

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection()
                                          throws JMSException
```

使用缺省的用户标识创建 XAQueueConnection。使用停止方式来创建该连接。直到显式地调用 Connection.start 方法后，才传送消息。

返回: 新创建的 XA 队列连接。

掷出:

- JMSException - 如果由于内部错误 JMS 供应商无法创建 XA 队列连接。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection
                              (java.lang.String userName,
                               java.lang.String password)
                                          throws JMSException
```

使用特定的用户标识来创建 XA 队列连接。使用停止方式来创建该连接。直到显式地调用 Connection.start 方法后，才传送消息。

参数:

- userName - 调用方的用户名。
- password - 调用方的口令。

返回: 新创建的 XA 队列连接。

掷出:

- JMSException - 如果由于内部错误 JMS 供应商无法创建 XA 队列连接。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

XAQueueSession

public interface **XAQueueSession**
extends **XASession**

MQSeries 类: **MQXAQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXAQueueSession
```

XAQueueSession 提供可用来创建 QueueReceivers、QueueSenders 和 QueueBrowsers 的常规 QueueSession。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

与 QueueSession 相对应的 XAResource 可以通过调用 getXAResource 方法来获取，它从 XASession 继承。

另见: **XASession**

方法

getQueueSession

```
public QueueSession  getQueueSession()
                                    throws JMSException
```

获取与该 XAQueueSession 关联的队列会话。

返回: 队列会话对象。

掷出: JMSException - 若发生 JMS 错误。

XASession

public interface **XASession**
extends **Session**
子接口: **XAQueueSession** 和 **XATopicSession**

MQSeries 类: **MQXASession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
```

XASession 通过访问 JTA 的 JMS 供应商支持扩展了 Session 的功能。该项支持采用了 javax.transaction.xa.XAResource 对象的格式。这个对象的功能性与用标准 “X/Open XA 资源” 接口定义的对象非常相似。

应用程序服务器通过获取 XASession 的 XAResource 来控制其事务性分配。它使用 XAResource 将会话分配给一个事务，准备并提交该事务上的工作等。

XAResource 提供了一些十分有效的设施，比如多个事务上的交叉工作以及恢复正在处理的事务列表。

JTA 已知的 JMS 供应商必须完全实现此项功能。要做到这一点，JMS 供应商可以使用支持 XA 的数据库服务，或是通过擦除来实现这项功能。

应用程序服务器的客户机作为一个常规的 JMS Session 出现。在幕后，应用程序服务器控制了基本 XASession 的事务管理。

有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见: **XAQueueSession** 和 **XATopicSession**

方法

getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

向调用方返回一个 XA 资源。

返回: 向调用方返回一个 XA 资源。

getTransacted

```
public boolean getTransacted()
                       throws JMSException
```

总是返回 true。

指定: 由 Session 接口中的 getTransacted 指定。

返回: true - 如果消息处于事务性方式。

掷出: JMSException - 如果 JMS 由于 JMS 供应商内部的错误而无法返回事务方式。

commit

```
public void commit()
            throws JMSException
```

不能对 XASession 对象调用此方法。如果调用，它将掷出一个 TransactionInProgressException。

指定: 由 Session 接口中的 commit 指定。

掷出: TransactionInProgressException - 如果在 XASession 上调用此方式。

rollback

```
public void rollback()
              throws JMSException
```

不能对 XASession 对象调用此方法。如果调用，它将掷出一个 TransactionInProgressException。

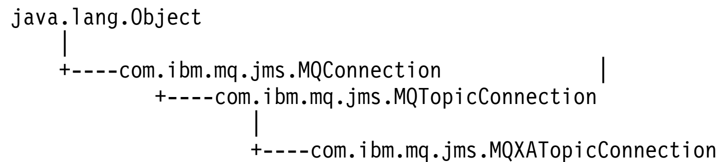
指定: 由 Session 接口中的 rollback 指定。

掷出: TransactionInProgressException - 如果在 XASession 上调用此方式。

XATopicConnection

public interface **XATopicConnection**
extends **TopicConnection** and **XAConnection**

MQSeries 类: **MQXATopicConnection**



```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQTopicConnection
                    |
                    +----com.ibm.mq.jms.MQXATopicConnection
```

XATopicConnection 提供了与 TopicConnection 相同的创建选项。根据定义，它们之间唯一不同的是，XAConnection 是事务性的。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见: **TopicConnection** 和 **XAConnection**

方法

createXATopicSession

```
public XATopicSession createXATopicSession()
                                            throws JMSException
```

创建 XATopicSession。

掷出: JMSException - 如果由于内部错误 JMS 连接无法创建主题会话。

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                        int acknowledgeMode)
                                  throws JMSException
```

创建 TopicSession。

指定: 由接口 TopicConnection 中的 createTopicSession 指定。

参数:

- transacted - 如果是 true，则该会话是事务性的。
- acknowledgeMode - 以下之一:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  表明消息使用者或客户机是否将确认接收到的任何消息。如果会话是事务性的，则忽略此参数。

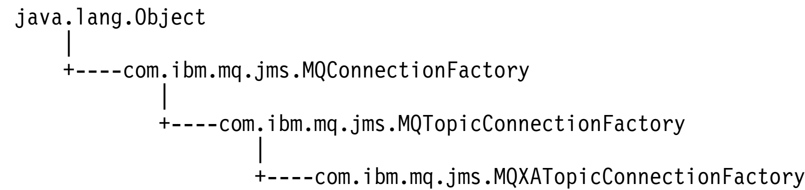
返回: 一个新建的主题会话（注意，这不是一个 XA 主题会话）。

掷出: JMSException - 如果由于内部错误 JMS 连接无法创建主题会话。

XATopicConnectionFactory

public interface **XATopicConnectionFactory**
extends **TopicConnectionFactory** and **XAConnectionFactory**

MQSeries 类: **MQXATopicConnectionFactory**



```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
                   |
                   +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

XATopicConnectionFactory 提供与 TopicConnectionFactory 相同的创建选项。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录E. 与 WebSphere 的 JMS JTA/XA 接口』。

另见: **TopicConnectionFactory** 和 **XAConnectionFactory**

方法

createXATopicConnection

```
public XATopicConnection createXATopicConnection()
                                          throws JMSException
```

使用缺省的用户标识创建 XA 主题连接。使用停止方式来创建该连接。直到显式地调用 Connection.start 方法后，才传送消息。

返回: 新创建的 XA 主题连接。

掷出:

- JMSException - 如果由于内部错误 JMS 供应商无法创建 XA 主题连接。
- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

createXATopicConnection

```
public XATopicConnection createXATopicConnection(java.lang.String userName,
                                                 java.lang.String password)
                                          throws JMSException
```

使用指定的用户标识创建 XA 主题连接。使用停止方式来创建该连接。直到显式地调用 Connection.start 方法后，才传送消息。

参数:

- userName - 调用方的用户名
- password - 调用方的口令

返回: 新创建的 XA 主题连接。

掷出:

- JMSException - 如果由于内部错误 JMS 供应商无法创建 XA 主题连接。

- JMSSecurityException - 如果由于无效的用户名或口令客户机认证失败。

XATopicSession

public interface **XATopicSession**
extends **XASession**

MQSeries 类: **MQXATopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXATopicSession
```

XATopicSession 提供了一个 TopicSession，可用于创建 TopicSubscribers 和 TopicPublishers。有关 MQ JMS 如何使用 XA 类的详细信息，请参考第351页的『附录 E. 与 WebSphere 的 JMS JTA/XA 接口』。

与 TopicSession 相对应的 XAResource 可以通过调用 getXAResource 方法来获取，它将自 XASession 继承。

另见: **TopicSession** 和 **XASession**

方法

getTopicSession

```
public TopicSession getTopicSession()
                          throws JMSException
```

获取与该 XATopicSession 关联的主题会话。

返回: 主题会话对象。

掷出:

- JMSException - 若发生 JMS 错误。

第4部分 附录

© Copyright IBM Corp. 1997, 2001

附录A. 管理工具特性与可编程特性之间的映射

MQSeries Classes for Java Message Service 提供了设置与查询使用 MQ JMS 管理工具或应用程序中的管理对象特性的设施。表30 显示了管理工具中使用的每个特性名与它所引用的相应的成员变量之间的映射。它还显示了工具中所使用的符号特性值与其等价程序之间的映射。

表 30. 管理工具内的特性以及可编程等价程序表示法之比较。

特性	成员变量名	特性值映射	
		工具	程序
DESCRIPTION	description		
TRANSPORT	transportType	• BIND • CLIENT	JMSC.MQJMS_TP_BINDINGS_MQ JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
CLIENTID	clientId		
QMANAGER	queueManager*		
HOSTNAME	hostName		
PORT	port		
CHANNEL	channel		
CCSID	CCSID		
RECEXIT	receiveExit		
RECEXITINIT	receiveExitInit		
SECEXIT	securityExit		
SECEXITINIT	securityExitInit		
SENDEXIT	sendExit		
SENDEXITINIT	sendExitInit		
TEMPMODEL	temporaryModel		
MSGRETENTION	messageRetention	• YES • NO	JMSC.MQJMS_MRET_YES JMSC.MQJMS_MRET_NO
BROKERVER	brokerVersion	• V1	JMSC.MQJMS_BROKER_V1
BROKERPUBQ	brokerPubQueue		
BROKERSUBQ	brokerSubQueue		
BROKERDURSUBQ	brokerDurSubQueue		
BROKERCCSUBQ	brokerCCSubQueue		
BROKERCCDSUBQ	brokerCCDurSubQueue		
BROKERQMGR	brokerQueueManager		
BROKERCONQ	brokerControlQueue		
EXPIRY	expiry	• APP • UNLIM	JMSC.MQJMS_EXP_APP JMSC.MQJMS_EXP_UNLIMITED
PRIORITY	priority	• APP • QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF

表 30. 管理工具内的特性以及可编程等价程序表示法之比较。 (续)

特性	成员变量名	特性值映射	
		工具	程序
PERSISTENCE	persistence	• APP • QDEF • PERS • NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON
TARGCLIENT	targetClient	• JMS • MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
ENCODING	encoding		
QUEUE	baseQueueName		
TOPIC	baseTopicName		
注: * 对于 MQQueue 对象，成员变量名为 baseQueueManagerName			

附录B. MQSeries classes for Java Message Service 提供的脚本

MQ JMS 安装的 bin 目录中提供了下列文件。提供脚本是为了帮助需要在安装或使用 MQ JMS 时执行的公共任务。表31 例出了脚本以及它们的用法。

表 31. MQSeries classes for Java Message Service 提供的实用程序

实用程序	用法
IVTRun.bat IVTTidy.bat IVTSetup.bat	用于运行点到点的安装验证测试程序，如第21页的『运行点到点 IVT』中所述。
PSIVTRun.bat	用于运行 Pub/Sub 安装验证测试程序，如第24页的『“发布／订阅”安装验证测试』中所述。
formatLog.bat	用于将日志文件转换成简单文本，如第28页的『日志记录』中所述。
JMSAdmin.bat	用于运行管理工具，如第29页的『第5章 使用 MQ JMS 管理工具』中所述。
JMSAdmin.config	管理工具的配置文件，如第30页的『配置』中所述。
runjms.bat	帮助 JMS 应用程序运行的实用脚本，如第27页的『运行您自己的 MQ JMS 程序』中所述。
PSReportDump.class	用于查看报告消息，如第186页的『处理代理报告』中所述。
注：在 UNIX 系统上，文件名中的扩展名 ’.bat’ 可省略。	

附录C. 针对 Java 对象的 LDAP 服务器配置

如果使用 JNDI 来存储由 MQ JMS 管理的对象，并使用 LDAP 服务器作为 JNDI 服务供应商，则该服务器必须是 LDAP v3（例如 SecureWay® eNetwork Directory v3.1），并且必须把它配置成能存储 Java 对象。

检查 LDAP 服务器配置

要检查 LDAP 服务器是否已经配置成可以接受 Java 对象，请以 LDAP 方式来运行 MQ JMS“管理工具”（请参阅第29页的『调用管理工具』）。

尝试使用以下命令来创建和显示测试对象:

```
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

如果没有发生异常情况，则表示服务器配置正确，并可以继续存储 JMS 对象。

| 如果返回 ’SchemaViolationException’，或出现“无法绑定到对象”这样一条消息，则表示服务器配置不正确。无论是服务器没有配置成可存储 Java 对象，还是对象的许可权或后缀不正确，下列过程都将帮助您完成配置任务。

配置过程

许多 LDAP 服务器都提供了允许您管理服务器的工具。有关使用这些工具的详细信息，请参考服务器文档。这些工具应该允许您查看和更新包含“属性”和“对象类”定义的模式。

请确保这些模式中包含了以下对象类定义，必要时，请添加它们:

```
( 1.3.6.1.4.1.42.2.27.4.2.1
  NAME 'javaContainer'
  DESC 'Container for a Java object'
  SUP top
  STRUCTURAL
  MUST ( cn )
)
( 1.3.6.1.4.1.42.2.27.4.2.4
  NAME 'javaObject'
  DESC 'Java object representation'
  SUP top
  ABSTRACT
  MUST ( javaClassName )
  MAY ( javaClassNames $
        javaCodebase $
        javaDoc $
        description )
)
```

© Copyright IBM Corp. 1997, 2001

```
( 1.3.6.1.4.1.42.2.27.4.2.5
  NAME 'javaSerializedObject'
  DESC 'Java serialized object'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)

( 1.3.6.1.4.1.42.2.27.4.2.7
  NAME 'javaNamingReference'
  DESC 'JNDI reference'
  SUP javaObject
  AUXILIARY
  MAY ( javaReferenceAddress $
        javaFactory )
)
```

此外，还应确保这些模式中包含了以下属性定义，必要时，请更新模式:

```
( 1.3.6.1.4.1.42.2.27.4.1.11
        NAME 'javaReferenceAddress'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.10
        NAME 'javaFactory'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.7
        NAME 'javaCodebase'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

更新完毕时，请停止并重新启动 LDAP 服务器，然后重复执行第347页的『检查 LDAP 服务器配置』中所描述的配置检查过程。

附录D. 连接到 MQSeries Integrator V2

可使用 MQSeries Integrator V2:

- 作为 MQ JMS 的发布／订阅代理
- 路由或转换 JMS 客户机应用程序创建的消息并且将消息发送或发布到 JMS 客户机。

发布／订阅

可使用 MQSeries Integrator V2 作为 MQ JMS 的发布／订阅代理。这需要下列设置活动:

- 基本 MQSeries

  首先，必须创建代理出版物队列。它是代理队列管理器上用来向代理提交出版物的 MQSeries 队列。可为该队列选择您自己的名称但它必须与 TopicConnectionFactory 的 BROKERPUBQ 特性中的队列名称匹配。缺省状态下，TopicConnectionFactory 的 BROKERPUBQ 特性被设置到 SYSTEM.BROKER.DEFAULT.STREAM 值中，除非要在 TopicConnectionFactory 中配置不同的名称，则应该命名队列 SYSTEM.BROKER.DEFAULT.STREAM。

- MQSeries Integrator V2

  下一步是在代理执行组中设置*消息流*。该消息流的目的是从代理出版物队列中读取消息。（如果愿意可设置多个出版物队列，每个队列都需要自己的 TopicConnectionFactory 和消息流。）

  基本消息流由输出连接到 Publication（或 MQOutput）节点输入的 MQInput 节点组成（配置为从 SYSTEM.BROKER.DEFAULT.STREAM 队列读取）。

  因此消息流图类似于以下:

图 7. MQSeries Integrator 消息流

部署了消息流并启动代理后，从 JMS 应用程序透视 MQSeries Integrator V2 代理类似于 MQSeries 发布／订阅代理。可使用 “MQSeries Integrator 控制中心” 查看当前订阅状态。

注:

1. Java Message Service 不需修改 MQSeries 类。
2. MQSeries 发布／订阅和 MQSeries Integrator V2 不能共存于相同的队列管理器上。
3. 关于 MQSeries Integrator V2 安装和设置过程的详细信息在 *MQSeries Integrator for Windows NT Version 2.0 Installation Guide* 一书中描述。

© Copyright IBM Corp. 1997, 2001

转换和路由

可使用 MQSeries Integrator V2 路由或转换 JMS 客户机应用程序创建的消息并且将消息发送或发布到 JMS 客户机。

MQSeries JMS 实现使用 MQRFH2 的 mcd 文件夹存储消息信息，如第192页的『MQRFH2 头』所述。缺省状态下，使用 Message Domain (Msd) 特性将消息标识为文本、字节、流、映射或对象消息。

JMS 应用程序创建文本或字节消息后，应用程序可覆盖 Msd 特性并设置其它 mcd 文件夹字段。它通过设置特殊 URL 格式的“JMS 类型”特性完成该操作，例如:

```
mcd://domain/set/type[?format=fmt]
```

将 *domain*、*set*、*type* 和 *fmt* 字段（*fmt* 为可选）中的值复制到外出 MQRFH2。这意味着应用程序能将这些字段设置为 MQSeries Integrator V2 消息流认可的值。

附录E. 与 WebSphere 的 JMS JTA/XA 接口

MQSeries classes for Java Message Service 包含 JMS XA 接口。这些允许 MQ JMS 参与由遵守 Java Transaction API (JTA) 的事务管理器协调的两阶段提交。

本节描述如何使用 WebSphere Application Server 高级版，因此，WebSphere 可在全局事务中协调 JMS 发送和接收操作和数据库更新。

与 WebSphere 一起使用 MQ JMS 和 XA 类之前，可能需要附加的安装或配置步骤。请参考 MQSeries 使用 Java SupportPac Web 页面上的 `Readme.txt` 以获取最新信息 (`www.ibm.com/software/ts/mqseries/txppacs/ma88.html`)。

与 WebSphere 的 JMS 接口

本节提供关于与 WebSphere Application Server，高级版一起使用 JMS 接口的指南。

您必须已经了解 JMS 程序、MQSeries 和 EJB bean 的基础。关于这些的详细情况，请参见 JMS 规范、EJB V2 规范（二者都可从 Sun 公司获取）、本手册、MQ JMS 提供的样本及 MQSeries 和 WebSphere 的其它手册。

受管理的对象

JMS 使用受管理的对象封装供应商指定的信息。它使供应商指定的详细信息对于最终用户应用程序的影响最小化。将受管理的对象存储到 JNDI 名称空间中，可不需要知道供应商特定的内容就可以移植方式来检索并使用它们。

对于单机使用，MQ JMS 提供下列类:

- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere 提供一对附加的管理对象，以便 MQ JMS 可与 WebSphere 集成:

- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

可用与 MQQueueConnectionFactory 和 MQTopicConnectionFactory 完全相同的方式使用这些对象。但是，在后台它们使用 JMS 类的 XA 版本，并在 WebSphere 事务列出 MQ XAResource。

容器管理的事务与 bean 管理的事务

容器管理的事务是 EJB bean 中由 EJB 存储器自动划分的事务。Bean 管理的事务是 EJB bean 中由程序（经由 UserTransaction 接口）划分的事务。

© Copyright IBM Corp. 1997, 2001

两阶段提交与一阶段优化

如果在特定事务中使用多个 XAResource，WebSphere 协调程序仅调用一个真实的两阶段提交。使用一阶段优化提交包含单个资源的事务。这将很大程度地除去分布式和非分布式事务使用不同 ConnectionFactory 的需要。

定义受管理的对象

可使用 MQ JMS 管理工具定义 WebSphere 指定的连接工厂并将它们存储在 JNDI 名称空间中。*MQ_install_dir*/bin 中的 admin.config 文件应该包含下列行:

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
PROVIDER_URL=iiop://hostname/
```

MQ_install_dir 是 MQ JMS 的安装目录，*hostname* 是运行 WebSphere 机器的名称或 IP 地址。

要访问 com.ibm.ejs.ns.jndi.CNInitialContextFactory，必须将 ejs.jar 文件从 WebSphere lib 目录添加到 CLASSPATH。

要创建新的工厂，请使用带有下列两种新类型的定义动词:

```
def WSQCF(name) [properties]
def WSTCF(name) [properties]
```

除了仅允许使用 BIND 传送类型的情况外，这些新类型使用等价于 QCF 或 TCF 类型的相同特性（因此无法配置客户机特性）。有关详细信息，请参阅第33页的『管理 JMS 对象』。

检索管理对象

在 EJB bean 中，使用 InitialContext.lookup() 方法检索 JMS 管理的对象，例如:

```
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

可转换对象的类型并作为类属 JMS 接口。通常，不需要对应用程序代码中的 MQSeries 特定类编程。

样本

有 3 个阐明了与 WebSphere Application Server 高级版一起使用 MQ JMS 基础的样本。这些在子目录 *MQ_install_dir*/samples/ws 中，其中 *MQ_install_dir* 是 MQ JMS 的安装目录。

- 样本 1 演示通过使用容器管理的事务将消息放入队列或从队列获取消息的简单示例。
- 样本 2 演示通过使用 bean 管理的事务将消息放入队列或从队列获取消息的简单示例。
- 样本 3 说明发布／订阅 API 的使用。

关于如何构建和部署 EJB bean 的详细信息，请参考 WebSphere Application Server 文档。

每个样本目录中的自述文件都包含来自每个 EJB bean 的输出示例。该脚本假设缺省队列管理器本地机器上是可用的。如果您的安装与缺省不同，可以按需要编辑这些脚本。

样本 1

样本 1 目录中的 Sample1EJB.java 定义了使用 JMS 的两种方法:

- putMessage() 将 TextMessage 发送到队列，并返回发送消息的 MessageID
- getMessage() 从队列中读回带有指定 MessageID 的消息

运行样本之前，必须在WebSphere JNDI 名称空间中存储两个管理对象:

QCF1 WebSphere 指定的队列连接工厂

Q1 队列

两个对象必须都绑定在 jms/Samples 子上下文中。

要使用受管理对象，可以使用 MQ JMS 管理工具并手工设置它们，或使用提供的脚本。

必须将 MQ JMS 管理工具配置成访问 WebSphere 名称空间。关于如何配置管理工具的详细信息，请参考第31页的『配置 WebSphere』。

要使用典型缺省设置来设置管理对象，可输入下列命令运行脚本 admin.scp:

```
JMSAdmin < admin.scp
```

必须使用标记为 TX_REQUIRED 的 getMessage 和 putMessage 方法来部署 bean。这确保存储器在进入每个方法前启动事务，并在方法完成后提交事务。在方法中不需要任何与事务性状态相关的应用程序代码。但是应记住从 putMessage 发送的消息出现在同步点下，并且直到提交事务后才可用。

在样本 1 目录中存在一个调用 EJB bean 的简单客户程序 Sample1Client.java。还有一个简化运行该程序的脚本 runClient。

该客户程序（或脚本）取单一参数，它用作 EJB bean putMessage 方法发送的 TextMessage 主体。然后，调用 getMessage 从队列读中读出消息并将主体返回客户机显示。EJB bean 将进程消息发送到应用程序服务器的标准输出 (stdout)，这样就可以在运行期间监视该输出。

如果应用程序服务器在客户机的远程机器上，可能需要编辑 Sample1Client.java。如果不使用缺省值，可能需要编辑 runClient 脚本以匹配本地安装路径和部署的 jar 文件名称。

样本 2

样本 2 目录中的 Sample2EJB.java 与样本 1中的相应文件作用相同。与样本 1 不同的是，样本 2 使用 bean 管理的事务来控制事务性边界。

如果已经运行样本 1，请确保设置了如『样本 1』中所示的管理对象 QCF1 和 Q1。

通过获取 UserTransaction 实例来启动 putMessage 方法和 getMessage 方法。它们使用该实例通过 UserTransaction.begin() 方法创建事务。然后，每个方法代码的主体均与样本 1 中的相同，只是每个方法的结尾不同。在每个方法的结尾，UserTransaction.commit() 调用完成事务。

在样本 2 目录中存在一个调用 EJB bean 的简单客户程序 Sample2Client.java。还有一个简化运行该程序的脚本 runClient。可按『样本 1』中所述的相同方法来使用它们。

样本 3

样本 3 目录中的 Sample3EJB.java 演示了使用带有WebSphere 的发布／订阅 API。发布消息与点到点的情况十分相似。但在通过 TopicSubscriber 接收消息时有所不同。

发布／订阅程序通常使用非长期订户。这些非长期订户只为其自己的会话存在（或者说如果显式关闭订户会减少）。它们也只能在存活时间期间从代理接收消息。

要将样本 1 转换为发布／订阅，可以用 TopicPublisher 替换 putMessageg 中的 QueueSender，用非长期的 TopicSubscriber 替换 getMessage 中的 QueueReceiver。但是这将失败，因为发送该消息时，代理不知道主题的任何订户。因此将废弃该消息。

该解决方案在发布消息前创建长期订户。长期订户在超过会话存活时间后仍保持为咳传递的端点。因此在调用 getMessage() 期间，可检索该消息。

EJB bean 包含两种附加方法。

- createSubscription 创建长期订阅
- destroySubscription 删除长期订阅

必须用 TX_REQUIRED 属性来部署这些方法（与 putMessage 和 getMessage 一起）。

运行样本 3 之前，必须在WebSphere JNDI 名称空间中存储两个管理对象:

TCF1

T1

两个对象都必须绑定在 jms/Samples 子环境中。

要设置管理对象，可以使用 MQ JMS 管理工具并手工设置它们，或使用脚本。在样本 3 目录中提供了脚本 admin.scp。

必须配置 MQ JMS 管理工具来访问WebSphere 名称空间。关于如何配置管理工具的详细信息，请参考第31页的『配置 WebSphere』。

要设置带有典型缺省设置的管理对象，咳输入下列命令来运行脚本 admin.scp:

```
JMSAdmin < admin.scp
```

如果已经运行 admin.scp 为样本 1 和样本 2 设置对象，则运行样本 3 的 admin.scp 时将发出错误信息。(在试图创建 jms 和“样本”子环境时出现这些消息。)可安全忽略这些错误信息。

运行样本 3 前也必须确保已安装并正在运行MQSeries 发布／订阅代理 (SupportPac MA0C)。

在样本 3 目录中存在一个调用 EJB bean 的简单客户程序 Sample3Client.java。还有一个简化运行该程序的脚本 runClient。可按第353页的『样本 1』中所述的相同方法来使用它们。

附录F. 声明

本信息是为在美国提供的产品和服务而开发的。IBM 可能未在其它国家提供本信息中所讨论的产品、服务或功能。请咨询本地的 IBM 代理以获得当前您所在的区域可用的产品和服务的信息。所有对 IBM 产品、程序或服务的引用并不明示或暗示只可以使用 IBM 的产品、程序或服务。任何不侵犯 IBM 知识产权的具有相同功能的产品、程序或服务都可以代替使用。但是，对任何非 IBM 产品、程序或服务的操作评估和验证都由用户自行负责。

IBM 可能已经申请或正在申请与本信息有关的各项专利权。提供本信息并不表示允许您使用这些专利。您可以用书面方式将许可证查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

涉及双字节 (DBCS) 信息的许可证查询，请联系您所在国的 IBM 知识产权部门或以书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

下列篇幅不适用于英联邦国家或任何当地法律与本条款不一致的其它国家: 国际商业机器公司以“照原样”方式提供本信息，本出版物不带任何形式的担保，无论是明示的或暗示的，其中包括（但不限于），非侵害性或特定用途的适用性隐式担保。有些国家禁止对某些事物的明示的或暗示的担保推卸责任，因此上述条款对您或许不适用。

本资料可能会包含技术上的不精确性或印刷错误。此处提到的信息会定期更改；这些更改被合并至本资料的新版本中。IBM 可能会在不作声明的情况下，随时对本文中所说明的产品和（或）程序作改进和（或）更改。

本信息中引用的任何非 IBM Web 站点仅为方便起见，在任何情况下都不能作为对那些 Web 站点的认同。那些站点中的资料不是 IBM 产品的一部分，使用那些 Web 站点请风险自负。

IBM 也许会以它认为适当的方法使用或散发您提供的信息，而不必对您负担任何责任。

为了以下目的: (i) 允许在独立创建的程序和其它程序（包括本程序）之间进行信息交换 (ii) 允许对已经交换的信息进行相互使用，而希望获取本程序有关信息的合法用户请与下列地址联系:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,

© Copyright IBM Corp. 1997, 2001

England
SO21 2JN.

只要遵守适当的条件和条款，包括某些情形下的一定数量的付款，都可获取这方面的信息。

本信息中描述的特许程序及所有可用于它的特许资料都是 IBM 按我们之间的 IBM 客户协议、IBM 国际编程许可证协议及其它同等的协议的各项条款提供。

涉及非 IBM 产品的信息是从这些产品的供应商、他们出版的通告或其它公开可用的途径获得的。IBM 没有测试过这些产品，不能确认其性能的准确度、兼容性或任何与非 IBM 产品相关的其它声明。有关非 IBM 产品功能的问题应该与这些产品的供应商联系。

商标

下列术语是国际商业机器公司在美国和／或其它国家的商标:

AIX	AS/400	BookManager
CICS	IBM	IBMLink
语言环境	MQSeries	MVS/ESA
OS/2	OS/390	OS/400
SecureWay	SupportPac	System/390
S/390	VisualAge	VSE/ESA
WebSphere		

Java、HotJava、JDK 和所有基于 Java 的商标和徽标是 Sun 公司在美国和／或其它国家的商标或注册商标。

Microsoft、Windows 和 Windows NT 是微软公司在美国和／或其它国家的商标。

UNIX 是 Open Group 在美国和其它国家的注册商标。

其它公司、产品和服务名称，可能是其它公司的商标或服务标记。

术语与缩写词汇表

本词汇表描述了本书中使用的术语，以及与其日常含义不同的一些词用法。某些情况下，一个定义可能不是唯一应用于一条术语的，但是它给出了该词用于本书的大概意思。

若未找到要查找的术语，请查阅索引或 *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994。

| **Abstract Window Toolkit for Java (AWT).** 使用组件的本地化平台版本实现的图形用户界面（GUI）组件的集合。

小应用程序 (applet). Java 程序，设计为只在 Web 页面上运行。

API. 应用程序编程接口。

应用程序编程接口 (Application Programming Interface (API)). 应用程序编程接口由函数和变量构成，允许程序员在其程序中使用这些函数和变量。

| **AWT.** Abstract Window Toolkit for Java.

| **类型转换 (casting).** Java 中使用的术语，用于描述将对象值或原语类型显式地转换成另一种类型。

通道 (channel). 见 MQI 通道。

类 (class). 由数据与在数据上执行的方法封装而成的集合。一个类会产生一个对象作为该类的实例。

客户机 (client). 在 MQSeries 中，客户机是一种运行时组件，它为本地用户应用程序提供了对服务器上排队服务的访问。

| **EJB.** 企业 JavaBean。

封装 (encapsulation). 封装是一种面向对象的编程技术，它使对象的数据专用化或受到保护，并允许程序员只能通过方法调用访问和操纵数据。

| **企业 JavaBean (Enterprise JavaBeans (EJB)).** 服务器方组件体系结构，由 Sun 公司发行，用于编写可重新使用的商业逻辑和可移植的企业应用程序。企业 JavaBean 组件全部由 Java 编写并且能运行在任何 EJB 兼容的服务器上。

HTML. 超文本标记语言 (Hypertext Markup Language)

超文本标记语言 (Hypertext Markup Language (HTML)). 一种用于定义在万维网上显示的信息的语言。

| **IEEE.** 电子与电气工程师协会。

IIOP. 因特网内部 ORB 协议。

因特网内部 ORB 协议 (Internet Inter-ORB Protocol (IIOP)). 是不同供应商与 ORB 的 TCP/IP 通信标准。

实例 (instance). 一个实例就是一个对象。当类付诸于产生一个对象时，称对象是类的实例。

接口 (interface). 接口是一个类，它只包含抽象的方法，没有实例变量。接口提供一组公用方法，它们可由许多不同的子类实现。

因特网 (Internet). 因特网是共享信息的协作公用网络。在物理上，因特网使用了所有当前现有公用远程通信网络资源的子资源。技术上，因特网之所以成为协作公用网络，是因为它使用了一组称为 TCP/IP 的协议（传输控制协议／网际协议）。

| **JAAS.** Java 认证与授权服务。

| **Java 认证与授权服务 (Java Authentication and Authorization Service (JAAS)).** 一种 Java 服务，它提供了实体认证与访问控制。

Java 开发工具箱 (Java Developers Kit (JDK)). Sun 公司为 Java 开发者发布的一个软件包。它包括 Java 解释器、Java 类和 Java 开发工具：编译器、调试程序、反汇编器、小应用程序查看器、文件生产者以及文档生产者。

Java 命名与目录服务 (Java Naming and Directory Service (JNDI)). 一个在 Java 编程语言中指定的 API。它向使用 Java 编程语言编写的应用程序提供了命名与目录功能。

Java 消息服务 (Java Message Service (JMS)). Sun 公司的 API，用于从 Java 程序访问企业的消息传递系统。

| **Java 运行时环境 (Java Runtime Environment (JRE)).** Java 开发工具箱 (JDK) 的一个子集，它包含了构建标准 Java 平台的核心可执行体与文件。JRE 包含了 Java 虚拟机、核心类以及支持文件。

| **Java 事务 API (Java Transaction API (JTA)).** 一个 API，使应用程序和 J2EE 服务器能够访问事务。

Java 事务服务 (Java Transaction Service (JTS)). 一个事务管理器，支持 JTA 并实现了 API 级别下的 OMG 对象事务服务 1.1 规范的 Java 映射。

Java 虚拟机 (Java Virtual Machine (JVM)). 中央处理单元 (CPU) 的一种软件实现，用于运行编译过的 Java 代码（小应用程序和应用程序）。

Java 2 平台，企业版 (Java 2 Platform, Enterprise Edition (J2EE)). 一组服务、API 和协议，它们提供了开发多层的，基于 Web 的应用程序的功能。

JDK. Java 开发工具箱。

JNDI. Java 命名与目录服务。

JMS. Java 消息服务。

JRE. Java 运行时环境。

JTA. Java 事务 API。

JTS. Java 事务服务。

JVM. Java 虚拟机。

J2EE. Java 2 平台，企业版。

LDAP. 轻量级目录访问协议。

轻量级目录访问协议 (Lightweight Directory Access Protocol (LDAP)). 用于访问目录服务的一种客户机-服务器协议。

消息 (message). 在消息队列程序中，消息是在程序之间发送的通信。

消息队列 (message queue). 见队列。

消息排队 (message queuing). 一种编程技术，应用程序内的每个程序通过将消息放入队列来与其它程序通信。

方法 (method). “方法”是面向对象的编程术语，针对函数或过程。

MQDLH. MQSeries 死信头。请参阅*MQSeries Application Programming Reference*。

MQI 通道 (MQI channel). MQI 通道将 MQSeries 客户机连到服务器系统的队列管理器上，并以双向方式传送 MQI 呼叫和响应。

MQMD. MQSeries 消息描述符。

MQSC. MQSeries 命令。

MQSeries. MQSeries 是 IBM 特许程序的一个系列，能够提供消息排队服务。

MQSeries 命令 (MQSeries commands (MQSC)). 在所有平台上用户都可读的相同的命令，用以操纵 MQSeries 对象。

MQSeries 消息描述符 (MQSeries Message Descriptor (MQMD)). 描述消息格式与特性的控制信息，它将作为 MQSeries 消息的一部分来传递。

对象 (object). （1）在 Java 中，对象是类的实例。一个类要塑造一组内容；一个对象塑造该组的一个特定成员。（2）在 MQSeries 中，一个对象是一个队列管理器、一个队列或是一个通道。

对象请求代理 (Object Request Broker (ORB)). 应用程序框架，为分布在异构环境中对象提供多方共同操作，这些对象用不同语言建立，在不同机器上运行。

对象管理组 (Object Management Group (OMG)). 一个设置面向对象编程标准的协会。

OMG. 对象管理组。

ORB. 对象请求代理。

重载 (overloading). 指某一标识指向了同一作用域中多个项的情况。在 Java 中，方法可以重载，但变量或运算符不可以。

包 (package). Java 中的包是指使一段 Java 代码访问某一组类的一种方法。一个特定包中的 Java 代码能够访问包中的所有类和类中的所有非专用方法及字段。

专用 (private). 专用字段在自己所属类的外部不可见。

受保护的 (protected). 保护字段只能在其所属的类或包中可见，或在其子类中可见。

公用 (public). 公用类或接口无论在何处都可见。公用方法或变量在其类可见时都是可见的

队列 (queue). 队列是 MQSeries 对象。消息队列可将消息从队列中放入和取出

队列管理器 (queue manager). 队列管理器是一种系统程序，为应用程序提供消息排队服务。

Red Hat Package Manager (RPM). 一种在 Red Hat Linux 平台以及其它 Linux 和 UNIX 上使用的打包系统。

RPM. Red Hat Package Manager。

服务器 (server). （1）MQSeries 服务器是队列管理器，为在远程工作站上运行的客户程序提供消息队列。（2）更一般地讲，服务器是一个程序，用特定的客户/服务器的双程序对于流模型，对信息的请求作出响应。（3）运行服务器程序的计算机。

小服务程序 (servlet). 一种 Java 程序，只设计为在 Web 服务器上运行。

子类 (subclass). 扩展另一个类的类。子类继承了其超类的公用和受保护的方法以及变量。

超类 (superclass). 被一些其它类扩展的类。超类的公用和受保护的方法以及变量可用于子类。

TCP/IP. 传输控制协议／网际协议。

传输控制协议／网际协议 (Transmission Control Protocol/Internet Protocol (TCP/IP)). 一组通信协议，即支持本地同级连通性功能，也支持广域网络的同级连通性功能。

| **统一资源定位器 (Uniform Resource Locator (URL)).**
| 一个字符序列，表示了计算机或网络（例如因特网）中的
| 信息资源。

| **URL.** 统一资源定位器。

VisiBroker for Java. 一种用 Java 编写的对象请求代理 (ORB)。

Web. 见万维网 (World Wide Web)。

web 浏览器 (Web browser). 格式化并显示万维网上发布的信息的程序。

万维网 (World Wide Web (Web)). 万维网是一项因特网服务，它基于了一组公共协议，允许某个配置成服务器的计算机以标准方法在因特网中发布文档。

文献目录

这个部分描述了所有当前 MQSeries 产品的可用文档。

MQSeries 跨平台出版物

这些出版物中的大多数书（有时也称为 MQSeries『系列』丛书）都适用于所有的 MQSeries 级别 2 产品。MQSeries 级别 2 的最新产品有:

- MQSeries AIX 版，V5.2
- MQSeries AS/400 版，V5.2
- MQSeries for AT&T GIS UNIX，V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS，V2.2.1.1
- MQSeries Compaq Tru64 UNIX 版，V5.1
- MQSeries HP-UX 版，V5.2
- MQSeries Linux 版，V5.2
- MQSeries OS/2 Warp 版， V5.1
- MQSeries for OS/390，V5.2
- MQSeries for SINIX and DC/OSx，V2.2
- MQSeries Sun Solaris 版，V5.2
- MQSeries Sun Solaris，Intel Platform Edition，V5.1
- MQSeries for Tandem NonStop Kernel，V2.2.0.1
- MQSeries for VSE/ESA，V2.1.1
- MQSeries Windows 版，V2.0
- MQSeries Windows 版，V2.1
- MQSeries Windows NT 和 Windows 2000 版，V5.2

MQSeries 的跨平台出版物有:

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries 客户机*, GA84-0689
- *MQSeries 系统管理*, SC84-0688
- *MQSeries MQSC 命令参考手册*, SC84-0736
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries 可编程系统管理*, SA40-1728
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries 应用程序设计指南*, SA40-1726
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries 使用 Java*, SC84-0704
- *MQSeries Application Messaging Interface*, SC34-5604

MQSeries 特定平台出版物

除 MQSeries 系列丛书外，每个 MQSeries 都至少编制了一个特定平台的出版物。

MQSeries AIX 版，V5.2

MQSeries AIX 版快速入门, GC84-0600

MQSeries AS/400 版，V5.2

MQSeries AS/400 版快速入门, GC34-5557

MQSeries for AS/400 System Administration, SC34-5558

MQSeries for AS/400 Application Programming Reference (ILE RPG), SC34-5559

MQSeries for AT&T GIS UNIX，V2.2

MQSeries for AT&T GIS UNIX System Management Guide, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide, GC33-1791

MQSeries Compaq Tru64 UNIX 版，V5.1

MQSeries Compaq Tru64 UNIX Quick Beginnings, GC34-5684

MQSeries HP-UX 版，V5.2

MQSeries HP-UX 版快速入门, GC84-0602

MQSeries Linux 版，V5.2

MQSeries Linux 版快速入门, GB84-0154

© Copyright IBM Corp. 1997，2001

MQSeries OS/2 Warp 版， V5.1

MQSeries OS/2 Warp 快速入门, GC84-0601

MQSeries for OS/390，V5.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5892

MQSeries for OS/390 Messages and Codes, GC34-5891

MQSeries for OS/390 Licensed Program Specifications, GC34-5893

MQSeries for OS/390 Program Directory

MQSeries link for R/3，版本 1.2

MQSeries link for R/3 User's Guide, GC31-1926

MQSeries for SINIX and DC/OSx, V2.2

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries Sun Solaris 版，V5.2

MQSeries Sun Solaris 版快速入门, GC84-0603

MQSeries Sun Solaris，Intel Platform Edition, V5.1

MQSeries Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel，V2.2.0.1

MQSeries for Tandem NonStop Kernel System Management Guide, GC33-1893

MQSeries for VSE/ESA，V2.1.1

MQSeries for VSE/ESA™ Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA System Management Guide, GC34-5364

MQSeries Windows 版，V2.0

MQSeries for Windows User's Guide, GC33-1822

MQSeries Windows 版，V2.1

MQSeries for Windows User's Guide, GC33-1965

MQSeries Windows NT 和 Windows 2000 版，V5.2

MQSeries Windows NT 快速入门, GC84-0604

MQSeries Windows NT 版部件对象模型接口, SC84-0702

MQSeries LotusScript Extension, SC84-0689

软拷贝书籍

大多数 MQSeries 书籍同时以硬拷贝和软拷贝格式提供。

HTML 格式

以下这些 MQSeries 产品中提供了 HTML 格式的 MQSeries 相关文档:

- MQSeries AIX 版，V5.2
- MQSeries AS/400 版，V5.2
- MQSeries Compaq Tru64 UNIX 版，V5.1
- MQSeries HP-UX 版，V5.2
- MQSeries Linux 版，V5.2
- MQSeries OS/2 Warp 版， V5.1
- MQSeries for OS/390，V5.2
- MQSeries Sun Solaris 版，V5.2
- MQSeries Sun Solaris，Intel Platform Edition, V5.1
- MQSeries Windows NT 和 Windows 2000 版，V5.2（编译过的 HTML）
- MQSeries link for R/3，V1.2

您还可以从以下的 MQSeries 产品系列的 Web 站点上获取 HTML 格式的 MQSeries 书籍:

```
http://www.ibm.com/software/mqseries/
```

可移植文档格式 (PDF)

可用 Adobe Acrobat Reader 查看并打印 PDF 文件。

要获取 Adobe Acrobat Reader 或是想了解目前有哪些平台支 Acrobat Reader，请访问 Adobe 公司的 Web 站点:

```
http://www.adobe.com/
```

提供 PDF 版本的相关 MQSeries 书籍的 MQSeries 产品有:

- MQSeries AIX 版，V5.2

- MQSeries AS/400 版，V5.2
- MQSeries Compaq Tru64 UNIX 版，V5.1
- MQSeries HP-UX 版，V5.2
- MQSeries Linux 版，V5.2
- MQSeries OS/2 Warp 版， V5.1
- MQSeries for OS/390，V5.2
- MQSeries Sun Solaris 版，V5.2
- MQSeries Sun Solaris， Intel Platform Edition，V5.1
- MQSeries Windows NT 和 Windows 2000 版，V5.2
- MQSeries link for R/3，V1.2

当前所有 MQSeries 书籍的 PDF 版本还可以从 MQSeries 产品系列 Web 站点上获取:

http://www.ibm.com/software/mqseries/

BookManager® 格式

MQSeries 书库都是以 IBM BookManager 格式在多种联机书库集合工具包中提供，包括*事务处理与数据*集合工具包，SK2T-0730。可以用以下 IBM 特许程序查看 IBM BookManager 格式的软拷贝书籍:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ Windows 版

PostScript 格式

许多 MQSeries 版本 2 产品都提供了 PostScript (.PS) 格式的 MQSeries 书库。PostScript 格式的图书可在 PostScript 打印机上打印，或用适当的查看程序查看。

Windows 帮助格式

在 MQSeries Windows 版，版本 2.0 和 MQSeries Windows 版，版本 2.1 中提供了 Windows 帮助格式的 *MQSeries Windows 版用户指南*。

因特网上可用的 MQSeries 信息

MQSeries 产品系列 Web 站点位于:

http://www.ibm.com/software/mqseries/

通过该 Web 站点的链接，可以:

- 获取有关 MQSeries 产品系列的最新消信息。
- 访问 HTML 和 PDF 格式的 MQSeries 书籍。
- 下载 MQSeries SupportPac。

索引

[A]

安全性考虑 事项 31
安装
  发布/订阅安装验证测试程序 (PSIVT) 24
  目录 10
  设置 19
  验证 19
  AS/400 上 MQ base Java 9
  IVT 错误恢复 24
  Linux 上的 MQ Java 9
  MQSeries classes for Java 7
  MQSeries classes for Java Message Service 7
  PSIVT 错误恢复 26
  Unix 上的 MQ Java 8
  Windows 上的 MQ Java 10
安装验证测试程序 (IVT) 21

[B]

绑定
  连接 6
  连接,编程 50
  验证 15
  样本应用程序 54
绑定传递 , 选择 170
包 (package)
  com.ibm.jms 226
  com.mq.ibm.jms 225
  javax.jms 223
报告选项, 消息 100, 209
报告, 代理 186
本地出版物, 抑制 182
必备软件 6
编程
  绑定连接 50
  编译 67
  多线程 59
  跟踪 68
  客户机连接 49
  连接 49
  写 49
编程接口 45
编译程序 MQSeries classes for Java程序 67
不同环境中的区别 71

[C]

操纵子上下文 32
测试 MQSeries classes for Java 程序 68
长期订户 182
超文本标记语言 (HTML) 364
程序
  跟踪 27
  运行 27, 68
  “发布/订阅”, 写 177
  JMS, 编写 167
程序员, 介绍 45
出版物
  MQSeries 363
出版物 (发布/订阅), 本地抑制 182
处理错误
  出错 58
  消息 57
  JMS 运行时错误 175
传递 , 选择 170
创建
  连接 169
  运行时的工厂 169
  运行时的主题 180
  JMS 对象 35
词汇表 359
从队列管理器断开 56
从 JNDI 检索对象 168
存储器管理的事务
  样本程序 353
错误
  处理错误 58
  对象创建的情况 41
  恢复, IVT 24
  恢复, PSIVT 26
  记录 28
  运行时, 处理 175
错误消息 18
  LDAP 服务器 347

[D]

代理报告 186
代码样本 50
导入语句 177
点到点安装验证 21
订户选项 181
定义传递 170
定义连接类型 50
订阅, 接收 179
定制样本 小应用程序 15
动词, 支持的 MQSeries 45
独立程序, 运行 5
读字符串 58
队列
  对象 (object) 168
  接口 279
队列管理器
  操作 56
  断开 56
  连接到 56
  配置客户机 13
队列管理器上的操作 56
队列特性
  设置 171
  用设置方法设置 172
队列特性的统一资源标识 (URI) 171
队列特性的 URI 171
队列, 访问 57
对象
  从 JNDI 检索 168
  管理的 168
  消息 187
  JMS, 创建 35
  JMS, 管理 33
  JMS, 特性 36
对象与特性的有效组合 38
对象与特性, 有效组合 38
对象 创建, 错误条件 41
多线程程序 59

[F]

发布消息 179
发布/订阅安装验证测试程序 (PSIVT) 24
发布/订阅, 样本应用程序 354
发送消息 171
访问队列与进程 57
非长期订户 182

[G]

概述 3
跟踪
  程序 68
  样本小应用程序 17
  样本应用程序 17
  MQSeries for Java Message Service 27
跟踪, 缺省输出位置 27
工厂, 在运行时创建 169
功能, MQ Java 特别提供的 3
构建连接 168
关闭
  发布/订阅方式中的 JMS 资源 179
  应用程序 175

© Copyright IBM Corp. 1997, 2001

关闭 *(续)*
资源 175
关闭应用程序 175
管理
动词 32
命令 32
管理对象 34，168
管理工具
概述 29
配置 30
配置文件 30
启动 29
特性映射 343
管理工具和程序之间的 映射 343
管理 JMS 对象 33

[H]

函数，应用程序服务器设施 205
核心类 71
异常 72
V5 扩展 74
环境变量 10
配置 19
环境区别 71
会话接口 167
会话类 205
会话，获取 170
获取会话 170

[J]

记录错误 28
接口
JMS 167，223
MQSeries 167
接口，编程 45
接收
发布／订阅方式中的消息 179
消息 173
解决问题 17
常规 27
以发布／订阅方式 185
介绍
针对程序员 45
MQSeries classes for Java 3
MQSeries classes for Java Message Service 3
进程，访问 57

[K]

可移植文档格式 (PDF) 364
客户机
编程 49
连接 5
客户机 *(续)*
配置队列管 理器 13
验证 15
客户机传递 ，选择 170
客户机特性 39
库，Java 类 46

[L]

类库 46
类路径
设置 10
类，核心 71
类，应用程序服务器设施 205
类，JMS 223
类，MQSeries classes for Java 77
ManagedConnection 159
ManagedConnectionFactory 162
ManagedConnectionMetaData 164
MQC 150
MQChannelDefinition 78
MQChannelExit 80
MQConnectionManager 152
MQDistributionList 83
MQDistributionListItem 85
MQEnvironment 86
MQException 91
MQGetMessageOptions 93
MQManagedObject 97
MQMessage 100
MQMessageTracker 119
MQPoolServices 121
MQPoolServicesEvent 122
MQPoolServicesEventListener 151
MQPoolToken 124
MQProcess 125
MQPutMessageOptions 127
MQQueue 130
MQQueueManager 138
MQReceiveExit 153
MQSecurityExit 155
MQSendExit 157
MQSimpleConnectionManager 148
连接
创建 169
建立 168
接口（interface） 167
启动 169
选项 4
MQSeries，丢失 185
连接到队列管理器 56
连接到 MQSeries Integrator V2 349
连接合用 62
示例 62
连接类型,定义 50
两阶段提交，与 WebSphere 352
令牌，连接合用 62
流消息 187

[M]

名称，主题的 179
命令，管理 32
命名考虑事项，LDAP 35
模式，LDAP 服务器 347
模型，JMS 167
目录，安装 10

[P]

配置
对于发布／订阅 20
管理工具 30
环境变量 19
客户机的队列管理器 13
您的安装 19
您的类路径 19
为 WebSphere 31
LDAP 服务器 347
Web 服务器 12
配置文件，用于管理工具 30
配置选项，消息 101，208
平台区别 71

[Q]

启动管 理工具 29
启动连接 169
缺省的跟踪和记录输出的位置 27
缺省连接池 62
多个组件 64
确认传递报告选项，消息 100
确认到达报告选项，消息 100

[R]

日志文件
缺省输出位置 27
转换 29
容器管理的事务 351
入门 3
软件，必备 6
软拷贝书籍 364

[S]

设置
带有设置方法的队列特性 172
队列特性 171
设置方法
使用来设置队列特性 172
在 MQQueueConnectionFactory 上 170

失效报告选项，消息　100
使用
　小应用程序查看器　13
　CICS 事务处理服务器　16
　MQ base Java　13
使用 MQSeries　3
事务
　容器管理的　351
　样本程序　353
　bean 管理的　351
受管理的对象
　与 WebSphere　351
书目　363

[T]

特性与对象，有效组合　38
头，消息　187

[W]

文本消息　187
问题解答　27
问题解决　17
问题。以发布／订阅方式解决　185

[X]

相关，特性　39
消息
　处理错误　57
　传递，异步　175
　错误　18
　发布　179
　发布／订阅方式中的选择器　182
　发送　171
　接收　173
　类型　173，187
　特性　187
　头　187
　消息主体　202
　选择　174，187
　选择器　174，187
　选择器和 SQL　188
　用发布／订阅方式接收　179
　有害　207
　主体　187
　JMS　187
　JMS 和 MQSeries 之间的映射　191
消息选择器的 SQL　188
消息子集，选择　174，187
小应用程序
　对应用程序　49
　样本代码　50
　运行　67
小应用程序查看器
　使用　5，13
小应用程序查看器 *(续)*
　用样本小应用程序　14
小应用程序与应用程序之间的差异　49
写
　程序　49
　用户出口　61
　字符串　58
　“发布／订阅”应用程序　177
　JMS 程序　167
选择传递　170
选择消息子集　174，187

[Y]

验证
　不使用 JNDI（点到点）　21
　不使用 JNDI （发布／订阅）　25
　客户机方式安装　13
　您的安装　19
　使用 JNDI（点到点）　22
　使用 JNDI （发布／订阅）　26
　用样本程序　15
　用样本小应用程序　13
　TCP/IP 客户机　15
样本程序
　绑定方式　54
　存储器管理的事务　353
　带有 WebSphere 的 MQ JMS　352
　发布／订阅　177，354
　跟踪　17
　使用验证　15
　使用“应用程序服务器设施”　214
　bean 管理的事务　353
样本代码　50
　小应用程序　50
　ServerSession　211
　ServerSessionPool　211
样本类路径设置　10
样本小应用程序
　定制　15
　跟踪　17
　使用验证　13
　用小应用程序查看器　14
要求的软件版本　6
一阶段优化，与 WebSphere　352
异步消息传递　175
异常
　对核心类　72
　JMS　175
　MQSeries　175
异常报告选项，消息　100，209
异常，侦听器　176
抑制本地出版物　182
因为环境的区别　71
应用程序
　对小应用程序　49
　关闭　175
应用程序 *(续)*
　意外终止　185
　运行　68
　“发布／订阅”，写　177
应用程序范例　54
应用程序服务器设施　205
　类和函数　205
　样本代码　211
　样本客户机应用程序　214
应用程序意外终止　185
映射消息　187
用户出口,写　61
由 MQ Java 提供 的特别功能　3
有害消息　207
运行
　程序　27
　独立程序　5
　小应用程序　67
　用小应用程序查看器　5
　自己编写的程序　16
　CICS 事务处理服务器下的应用程序　68
　IVT　21
　MQSeries classes for Java 程序　68
　PSIVT　24
　Web 浏览器内　5
运行时
　创建工厂　169
　创建“主题”　180
　错误,处理　175

[Z]

侦听器，异常　176
终止，意外的　185
主体，消息　187
主题
　对象（object）　168
　接口　312
　接口（interface）　177
　名称　179
　名称，通配符　180
主题名称中的通配符　180
转换日志文件　29
资源，关闭　175
子上下文，操纵　32
字符串,读写　58
字节消息　187
组合，有效，对象与特性　38

A

AIX，安装 MQ Java　8
ASF（应用程序服务器设施）　205
ASFClient1.java　216
ASFClient2.java　218

ASFClient3.java 220
ASFClient4.java 220
AS/400，安装 MQ base Java 9

B

bean 管理的事务 351
  样本程序 353
BookManager 365
BROKERCCDSUBQ 对象特性 37，207，345
BROKERCCSUBQ 对象特性 37，207，345
BROKERCONQ 对象特性 37，345
BROKERDURSUBQ 对象特性 37，345
BROKERPUBQ 对象特性 37，345
BROKERQMGR 对象特性 37，345
BROKERSUBQ 对象特性 37，345
BROKERVER 对象特性 37，345
BytesMessage
  接口 227
  类型 173

C

CCSID 对象特性 37，345
CHANGE（管理动词） 32
CHANNEL 对象特性 37，345
CICS 事务处理服务器
  使用 16
  运行应用程序 68
classpath
  配置 19
CLIENTID 对象特性 37，345
com.ibm.jms 软件包 226
com.ibm.mqbind.jar 7
com.ibm.mqjms.jar 7
com.ibm.mq.iiop.jar 7
com.ibm.mq.jar 7
com.ibm.mq.jms package 225
Connection 接口 235
ConnectionConsumer 接口 238
ConnectionConsumer 类 205
ConnectionFactory 接口 239
ConnectionMetaData 接口 243
connector.jar 7
COPY（管理动 词） 32
CountingMessageListenerFactory.java 216
createQueueSession 方法 170
createReceiver 方法 173
createSender 方法 171

D

DEFINE（管理动词） 32
DELETE（管理动词） 32
DeliveryMode 接口 245
DESCRIPTION 对象特性 37，345
Destination 接口 246
DISPLAY（管理动 词） 32

E

ENCODING 对象特性 39
END（管理动词） 32
ExceptionListener 接口 248
exit string properties 39
EXPIRY 对象特性 37，345

F

formatLog 实用程序 29，345
fscontext.jar 7

H

HOSTNAME 对象特性 37，345
HP-UX，安装 MQ Java 8
HTML（超文本标记语言） 364

I

IIOP 连接，编程 49
INITIAL_CONTEXT_FACTORY 参数 30
inquire 和 set 59
IVT （安装验证测试程序） 21
IVTRun 实用程序 21，23
IVTrun 实用程序 345
IVTSetup 实用程序 22，345
IVTTidy 实用程序 24，345

J

J2EE 连接器体系结构 62
JAAS（Java 认证与授权服务） 62，152
jar 文件 7
Java 接口的优点 45
Java 接口，优点 45
Java 开发工具箱（Java Developers Kit (JDK)） 46
Java 类 46，77
Java 认证与授权服务 (JAAS) 62，152
Java 事务 API (JTA) 334
Java 2 平台，企业版 (J2EE) 62
Java Transaction API (JTA) 351
javax.jms 软件包 223
JDK（Java 开发工具箱） 46
JMS
  程序，编写 167
  对象，创建 35
  对象，管理 33
JMS *(续)*
  对象，特性 36
  发布 / 订阅的对象 177
  管理对象 168
  好处 3
  接口 167，223
  介绍 3
  类 223
  模型 167
  使用 MQMD 映射 195
  消息 187
  消息类型 173
  异常 175
  异常侦听器 176
  在发送 / 发布的字段映射 197
  资源，在发布 / 订阅方式中关闭 179
JMS 的好处 3
JMS 消息类型 173，187
JMS JTA/XA 接口 351
JMSAdmin 实用程序 345
JMSAdmin.config 实用程序 345
JMSBytesMessage 类 227
JMSCorrelationID 头字段 187
JMSMapMessage 类 249
JMSMessage 类 257
JMSStreamMessage 类 301
JMSTextMessage 类 311
jms.jar 7
JNDI
  安全性考虑事项 31
  检索 168
jndi.jar 7
JTA（Java 事务 API） 334
JTA（Java Transaction API） 351

L

LDAP 服务器 22
  模式 347
  配置 347
LDAP 命名考虑事项 35
ldap.jar 7
Linux，安装 MQ Java 9
Load1.java 215
Load2.java 218
LoggingMessageListenerFactory.java 218

M

ManagedConnection 159
ManagedConnectionFactory 162
ManagedConnectionMetaData 164
MapMessage
  接口（interface） 249
  类型 173
Message 接口 257

MessageConsumer 接口 167, 270
MessageListener 接口 272
MessageListenerFactory.java 214
MessageProducer 对象 171
MessageProducer 接口 167, 273
MOVE（管理动词） 32
MQC 150
MQChannelDefinition 78
MQChannelExit 80
MQConnection 类 235
MQConnectionConsumer 类 205, 238
MQConnectionFactory 类 239
MQConnectionManager 152
MQConnectionMetaData 类 243
MQDeliveryMode 类 245
MQDestination 类 246
MQDistributionList 83
MQDistributionListItem 85
MQEnvironment 50, 56, 86
MQException 91
MQGetMessageOptions 93
MQIVP
  跟踪 17
  列举 15
  样本应用程序 15
mqjavac
  跟踪 17
  使用验证 13
MQManagedObject 97
MQMD（MQSeries 消息描述符） 191
MQMessage 57, 100
MQMessageConsumer 类 270
MQMessageProducer 接口 273
MQMessageTracker 119
MQObjectMessage 类 278
MQPoolServices 121
MQPoolServicesEvent 122
MQPoolServicesEventListener 151
MQPoolToken 124
MQProcess 125
MQPutMessageOptions 127
MQQueue 57, 130
  对于验证 22
  类 279
  （JMS 对 象） 34
MQQueueBrowser class 281
MQQueueConnection class 283
MQQueueConnectionFactory
  对象（object） 168
  对于验证 22
  设置方法 170
  class 285
  interface 285
  （JMS 对 象） 34
MQQueueEnumeration class 277
MQQueueManager 57, 138
MQQueueReceiver class 287
MQQueueSender 接口 290
MQQueueSession 类 293
MQReceiveExit 153
MQRFH2 头 192
MQSecurityExit 155
MQSendExit 157
MQSeries
  接口 167
  连接, 丢失 185
  消息 191
  异常 175
MQSeries 出版物 363
MQSeries 消息描述符 (MQMD) 191
  使用 JMS 映射 195
MQSeries 支持的动词 45
MQSeries classes for Java 类 77
MQSeries classes for Java Message Service 提供的脚本 345
MQSeries classes for Java Message Service 提供的实用程序 345
MQSeries Integrator V2, 连接到 MQ JMS 349
MQSeriesV5 e扩展 74
MQSession 类 205, 296
MQSimpleConnectionManager 148
MQTemporaryQueue 类 309
MQTemporaryTopic 类 310
MQTopic
  类 312
  （JMS 对 象） 34
MQTopicConnection 类 314
MQTopicConnectionFactory
  对象（object） 168
  类 316
  （JMS 对 象） 34
MQTopicPublisher 类 319
MQTopicSession 类 323
MQTopicSubscriber 类 327
MQXAConnection 类 328
MQXAConnectionFactory 类 329
MQXAQueueConnection 类 330
MQXAQueueConnectionFactory 类 331
MQXAQueueSession 类 333
MQXASession 类 334
MQXATopicConnection 类 336
MQXATopicConnectionFactory 类 337
MQXATopicSession 类 339
MSGRETENTION 对象特性 37, 345
MyServerSessionPool.java 213
MyServerSession.java 213

N

Netscape Navigator, 使用 6

O

ObjectMessage
  接口 278
  类型 173
options
  订户 181
  连接 4

P

PDF（可移植文档格式） 364
PERSISTENCE 对象特性 37, 345
PORT 对象特性 37, 345
PostScript 格式 365
PRIORITY 对象特性 37, 345
properties
  出口字符串 39
  队列, 设置 171
  管理工具和程序之间的射映 343
  客户机 39
  相关性 39
  消息 187
  JMS 对象 36
providerutil.jar 7
PROVIDER_URL 参数 30
PSIVT（安装验证测试程序） 24
PSIVTRun 实用程序 25, 345
PSReportDump 应用程序 186

Q

QMANAGER 对象特性 37, 345
QUEUE 对象特性 37, 345
QueueBrowser interface 281
QueueConnection interface 283
QueueReceiver interface 287
QueueRequestor 类 288
QueueSender 接口 290
QueueSession 接口 293

R

RECEXIT 对象特性 37, 345
RECEXITINIT 对象特性 37, 345
runjms 实用程序 27, 345

S

Sample1EJB.java 353
Sample2EJB.java 353
Sample3EJB.java 354
SECEXIT 对象特性 37, 345
SECEXITINIT 对象特性 37, 345
SECURITY_AUTHENTICATION 参数 30
selectors
  发布／订阅方式中的消息 182

selectors *(续)*
  消息 174, 187
  消息, 和 SQL 188
SENDEXIT 对象特性 37, 345
SENDEXITINIT 对象特性 37, 345
ServerSession 样本代码 211
ServerSessionPool 样本代码 211
Session 接口 296
set 和 inquire 59
Solaris
  安装 MQ Java 8
StreamMessage
  接口 301
  类型 173
Sun 公司 Web 站点 3
Sun JMS 接口类 223
Sun Solaris
  安装 MQ Java 8
SupportPac 365

T

TARGCLIENT 对象特性 37, 345
TCP/IP
  客户机验证 15
  连接,编程 49
TEMPMODEL 对象特性 37, 345
TemporaryQueue 接口 309
TemporaryTopic 接口 310
TextMessage
  接口 311
  类型 173
TOPIC 对象特性 37, 345
TopicConnection 177
  接口 314
TopicConnectionFactory 177
  接口 316
TopicLoad.java 219
TopicPublisher 178
  接口 319
TopicRequestor 类 322
TopicSession 177
  接口 323
TopicSubscriber 178
  接口 327
TRANSPORT 对象特性 37, 345

U

UNIX没, 安装 MQ Java 8

V

V5 核心类的扩展 74
V5 扩展 74
VisiBroker
  连接 4, 50, 53
VisiBroker *(续)*
  配置队列管理器 14
  使用 4, 6, 16

W

Web 服务器, 配置 12
Web 浏览器（Web browser）
  使用 5
WebSphere
  配置 31
  CosNaming 名称空间 30
  CosNaming 资源库 30
WebSphere 应用程序服务器 211
WebSphere Application Server 351
  使用 JMS 351
Windows
  安装 MQ Java 10
Windows 帮助 365

X

XAConnection 接口 328
XAConnectionFactory 接口 329
XAQueueConnection 接口 330
XAQueueConnection interface 283
XAQueueConnectionFactory 接口 331
XAQueueConnectionFactory interface 285
XAQueueSession 接口 333
XAResource 334
XASession 接口 334
XATopicConnection 接口 336
XATopicConnectionFactory 接口 337
XATopicSession 接口 339

把您的意见发送给 IBM

如果您特别喜欢或不喜欢这本书中的某些内容，请使用下面列出的方法之一把您的意见发送给 IBM。

请大胆地提出您认为的特定错误或遗漏处，以及本书在准确性、组织、主题内容或完整性方面的意见。

请您只是就本书中的信息及信息表示的方法上提出您的意见。

若要对有关 IBM 产品或系统的功能提出意见，请告诉您的 IBM 代表或 IBM 的授权转售商。

当您向 IBM 发送意见时，就授予了 IBM 非专有权， IBM 对您所提供的意见，有权利以任何它认为适当的方式使用或分发，而不必对您负担任何责任。

您可以通过下列方式之一将意见发送到 IBM:

- 通过邮件，地址为:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- 通过传真:
  - 在 U.K. 之外，请在国际访问码后使用 44-1962-870229
  - 在 U.K.，请使用 01962-870229
- 通过电子发送，请使用适当的网络标识:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - 因特网: idrcf@hursley.ibm.com

不论您使用哪种方式，请确保您包含了:

- 出版物标题与订单号码
- 您的意见所适用的主题
- 您的名字和地址/电话号码/传真号码/网络标识。

© Copyright IBM Corp. 1997, 2001

Printed in China

SC84-0704-01