火龙果•整理
uml.org.cn

目录

目录 3
1 引言 8
1.1 摘要 8
1.2 概述 8
1.2.1 是 Mail API 吗？ 8
1.2.2 现存的消息系统 8
1.2.3 JMS 目标 9
1.2.4 JMS 不包含什么 10
1.3 JMS 的要求是什么 10
1.4 与其他 Java API 的关系 10
1.4.1 JDBC 软件 10
1.4.2 JavaBean 组件 10
1.4.3 EJB 组件模型 11
1.4.4 Java 事务 API（JTA） 11
1.4.5 Java 事务服务（JTS） 11
1.4.6 Java 命名和目录接口 API（JNDI） 11
1.4.7 J2EE 平台 11
1.4.8 JMS 和 EJB 组件的集成 12
1.5 JMS1.1 的新特性是什么？ 12
2 架构 12
2.1 概述 12
2.2 什么是 JMS 应用 12
2.3 管理 13
2.4 两种消息风格 13
2.5 JMS 接口 14
2.6 开发一个 JMS 应用 15
2.6.1 开发一个 JMS 客户端 15
2.7 安全 15
2.8 多线程 15
2.9 触发式客户端 16
2.10 请求/回复 16
3 JMS 消息模型 16
3.1 背景 16
3.2 目标 17
3.3 JMS 消息 17
3.4 消息头字段 17
3.4.1 JMSDestination 17
3.4.2 JMSDeliveryMode 17
3.4.3 JMSMessageID 18
3.4.4 JMSTimestamp 18
3.4.5 JMSCorrelationID 18
3.4.6 JMSReplyTo 19

JANTY

火龙果•整理
uml.org.cn

3.4.7 JMSRedelivered 19
3.4.8 JMSType 19
3.4.9 JMSExpiration 19
3.4.10 JMSPriority 20
3.4.11 如何设置消息头的值 20
3.4.12 重载消息头字段 20
3.5 消息属性 20
3.5.1 属性名 21
3.5.2 属性值 21
3.5.3 属性的使用 21
3.5.4 属性值转换 21
3.5.5 属性值作为对象 22
3.5.6 属性迭代 22
3.5.7 清除消息属性的值 22
3.5.8 不存在的属性 22
3.5.9 JMS 定义的属性 22
3.5.10 提供商专有的属性 23
3.6 消息确认 24
3.7 Message 接口 24
3.8 消息选择 24
3.8.1 消息选择器 24
3.9 访问已发送的消息 27
3.10 改变收到的消息的值 28
3.11 JMS 消息体 28
3.11.1 清除消息体 28
3.11.2 只读消息体 28
3.11.3 由 StreamMessage 和 MapMessage 提供的转换 29
3.11.4 用于非 JMS 客户端的消息 29
3.12 JMS Message 接口的提供商实现 30
4 JMS 公共工具 30
4.1 概述 30
4.2 受管理的对象 30
4.2.1 Destination 31
4.2.2 ConnectionFactory 31
4.3 Connection 31
4.3.1 授权 32
4.3.2 客户端标识 32
4.3.3 Connection 设置 32
4.3.4 中止消息的转发 33
4.3.5 关闭 Connection 33
4.3.6 会话(Session) 34
4.3.7 ConnectionMetaData 34
4.3.8 ExceptionListener 34
4.4 Session 34

火龙果•整理
uml.org.cn

4.4.1 关闭会话 35
4.4.2 创建 MessageProducer 和 MessageConsumer 35
4.4.3 创建临时目的地 35
4.4.4 创建目的地对象 35
4.4.5 优化消息实现 36
4.4.6 使用会话的约定 36
4.4.7 事务 37
4.4.8 分布式事务 37
4.4.9 多会话 37
4.4.10 消息排序 37
4.4.11 消息确认 38
4.4.12 消息的重复转发 39
4.4.13 消息的重复产生 39
4.4.14 客户端代码的有序执行 39
4.4.15 并行消息转发 39
4.5 MessageConsumer 39
4.5.1 同步转发 40
4.5.2 异步转发 40
4.6 MessageProducer 40
4.7 消息转发模式 41
4.8 消息的生存时间 41
4.9 异常 41
4.10 可靠性 41
4.11 方法跨消息域继承 42
5 JMS 点对点模型 42
5.1 概述 42
5.2 队列管理 43
5.3 Queue 43
5.4 TemporaryQueue 43
5.5 QueueConnectionFactory 43
5.6 QueueConnection 44
5.7 QueueSession 44
5.8 QueueReceiver 44
5.9 QueueBrowser 44
5.10 QueueRequestor 44
5.11 可靠性 45
6 JMS 发布/订阅模型 45
6.1 概述 45
6.2 Pub/Sub 延时 45
6.3 永久订阅 46
6.4 主题（Topic）管理 46
6.5 Topic 46
6.6 TemporaryTopic 46
6.7 TopicConnectionFactory 47

JANTY

火龙果•整理
uml.org.cn

6.8 TopicConnection 47
6.9 TopicSession 47
6.10 TopicPublisher 47
6.11 TopicSubscriber 47
6.11.1 永久 TopicSubscriber 48
6.12 恢复和重发 48
6.13 管理订阅 48
6.14 TopicRequestor 48
6.15 可靠性 48
7 JMS 异常 49
7.1 概述 49
7.2 JMSException 49
7.3 标准异常 49
8 JMS 应用服务器工具 50
8.1 概述 50
8.2 并发处理订阅的消息 51
8.2.1 Session 51
8.2.2 ServerSession 51
8.2.3 ServerSessionPool 51
8.2.4 ConnectionConsumer 52
8.2.5 ConnectionConsumer 如何使用 ServerSession 52
8.2.6 应用服务器如何实现 ServerSession 52
8.2.7 结果 52
8.3 XAConnectionFactory 54
8.4 XAConnection 54
8.5 XASession 54
8.6 JMS 应用服务器接口 54
9 JMS 样例代码 55
9.1 准备发送和接收消息 55
9.1.1 获取 ConnectionFactory 55
9.1.2 获取 Destination 56
9.1.3 创建 Connection 56
9.1.4 创建 Session 56
9.1.5 创建 MessageProducer 56
9.1.6 创建 MessageConsumer 56
9.1.7 启动消息转发 57
9.1.8 使用 TextMessage 57
9.2 发送和接收消息 57
9.2.1 发送消息 57
9.2.2 同步接收消息 58
9.2.3 解包 TextMessage 58
9.3 其他消息特性 58
9.3.1 异步接收消息 58
9.3.2 使用消息选择器 59

火龙果•整理
uml.org.cn

9.3.3 使用永久订阅 59
9.4 JMS 消息类型 61
9.4.1 创建 TextMessage 61
9.4.2 解包 TextMessage 61
9.4.3 创建 BytesMessage 61
9.4.4 解包 BytesMessage 61
9.4.5 创建 MapMessage 62
9.4.6 解包 MapMessage 62
9.4.7 创建 StreamMessage 63
9.4.8 解包 StreamMessage 63
9.4.9 创建 ObjectMessage 64
9.4.10 解包 ObjectMessage 64
10 问题 65
10.1 已解决的问题 65
10.1.1 JDK1.1.x 兼容性 65
10.1.2 分布式 Java 事件模型 65
10.1.3 可以合并 JMS 的两个域 PTP 和 Pub/Sub 吗？ 65
10.1.4 JMS 应当指定一个 JMS JavaBean 集合吗？ 65
10.1.5 与 CORBA 通知服务对齐 65
10.1.6 JMS 应当提供端对端的同步消息转发和转发通知吗？ 66
10.1.7 JMS 应当提供发送到列表的机制吗？ 66
10.1.8 JMS 应当提供订阅通知吗？ 66
11 变更历史 66

JANTY

火龙果•整理
uml.org.cn

1 引言

1.1 摘要

本规范描述了 JMS 的目标和功能。

JMS 给 java 程序员提供了一种通用的方式来创建、发送、接收和查看企业消息系统消息。

1.2 概述

企业消息产品（或者有时称为面向消息的中间件产品）正逐渐成为公司内操作集成的关键组件。这些产品可以将分离的业务组件组合成一个可靠灵活的系统。

除了传统的 MOM 供应商，企业消息产品也可以由数据库供应商和许多与网络相关的公司来提供。

Java 语言的客户端和 Java 语言的中间层服务必须能够使用这些消息系统。JMS 为 Java 语言程序提供了一个通用的方式来获取这些系统。

JMS 是一个接口和相关语义的集合，那些语义定义了 JMS 客户端如何获取企业消息产品的功能。

由于消息是点对点的，所以 JMS 的所有用户都称为客户端（clients）。JMS 应用由定义消息的应用和一系列与他们交互的客户端组成。

1.2.1 是 Mail API 吗？

术语“消息”在计算机领域到处都有。它用于描述各种操作系统概念；用于描述邮件和传真系统；但在这里用于描述企业应用间的异步通讯。

这里描述的消息是由企业应用而不是人来处理的异步请求、报告或事件。他们包含了协同这些应用所必需的信息。他们包含了描述特定业务动作的格式化的数据。应用通过交换消息来跟踪企业的过程。

1.2.2 现存的消息系统

消息系统是点对点的工具。通常情况下，每个客户端可以发送消息到另一个客户端，也可以从任何客户端接收消息。每个客户端连接到提供创建、发送和接受消息的消息代理。

每个系统都提供了定位消息的方式。每个系统都提供了创建消息并给他填充数据的途径。

有些系统可以想多个目的地广播消息。其他的系统也可以只支持向一个目的地发送消息。

某些系统提供了异步接收消息的功能（当消息到达时被转发到客户端）。其他的系统可以支持同步接收（客户端必须请求每个消息）。

每个消息系统通常提供多种服务供不同的消息来选择。重要的问题是系统能保证转发的长度是多少。它们可能不是一次就能转发完全的。其他重要的问题是消息是有时效、有优先级和是否要求响应。

火龙果•整理
uml.org.cn

1.2.3 JMS 目标

如果 JMS 提供了现有消息系统的所有特性，那么对用户来讲它就太复杂了。在另一个方面，JMS 更多是所有消息产品公共特性的交集。重要的是 JMS 要包含实现专业企业应用需要的功能。

JMS 定义了一系列通用的企业消息概念和工具。它试图最小化 Java 语言程序员使用企业消息产品而必须了解的概念集。它致力于最大化消息应用的可移植性。

1.2.3.1JMS 提供商

正如前面提到的，JMS 提供商是一个在消息产品实现 JMS 的实体。

理想情况下，JMS 提供商用纯 Java 来实现消息产品，这样它就能运行在 applet 中，简化安装，并且可以架构和 OS 工作。

JMS 的一个重要目标是最小化实现一个提供商所需要的工作。

1.2.3.2JMS 消息

JMS 定义了一系列消息接口。

客户端使用由 JMS 提供商提供的消息实现。

JMS 的一个主要目标是客户端使用统一的 API 来创建和与独立于 JMS 提供商的消息一起工作。

1.2.3.3JMS 域

消息产品可以广义上可以分为点对点或发布-订阅系统。

点对点（PTP）产品围绕着消息队列创建。每个消息被放置在一个特定的队列中；客户端从队列中取出消息。

发布和订阅（Pub/Sub）客户端将消息放置到某个内容继承层次上的节点上。发布者和订阅者通常都是匿名的，通常可以动态的发布或订阅内容层级。系统关注将来自一个节点的多个发布者的消息分发到这个节点的多个订阅者。

JMS 提供了一系列的接口来让客户端在两种域下发送和接收消息，同时支持每个域的语义。JMS 也为每个域提供了相应的客户端接口。JMS1.1 规范以前的版本，只有对应于每个域的客户端接口。这些接口继续被支持以提高向后的兼容性。实现客户端的最好方式是使用不依赖域的接口。这些接口称为“通用接口”是域特有接口的父类。

1.2.3.4可移植性

最主要的可移植目标是新的只有 JMS 的应用可以在同一个消息域内可以跨产品。

另外，JMS 客户端跨机器架构和操作系统是期望的可移植性（当时有同一个 JMS 提供商时）。

火龙果•整理
uml.org.cn

尽管设计 JMS 的目的是让客户端可以和现存的在混合语言应用中使用的消息格式一起工作，但是这种客户端通常是不可移植的（将一个混合语言应用从一个产品移植到另一个产品超出了 JMS 的范围）。

1.2.4 JMS 不包含什么

JMS 没有包含下列功能：

- 负载均衡/容错（Load Balancing/Fault Tolerance）——许多产品为对多个协同的客户端提供了关键的服务。JMS API 没有指定这种客户端协同如何作为一个单独的统一的服务出现。
- 错误/劝告通知（Error/Advisory Notification）——多数消息产品定义系统消息来向客户端提供问题或系统事件的异步通知。JMS 没有试图标准化这些消息。通过遵循由 JMS 定义的指南，客户端可以避开这些消息，这样就可以防止由于使用这些消息而引入的可移植性问题。
- 管理——JMS 没有定义管理消息产品的 API。
- 安全——JMS 没有定义用于控制消息私密性和完整性的 API。它也没有指定数字签名或密钥如何被分发到客户端。安全是由 JMS 提供商要考虑的问题——由管理员配置这些特有的特性，而不是由客户端使用 JMS API 来控制。
- 通讯协议（Wire Protocal）——JMS 没有定义消息的通讯协议。
- 消息类型存储池——JMS 没有定义存储消息类型定义的存储池，也没有定义创建消息类型定义的语言。

1.3 JMS 的要求是什么

在这个规范内讨论的功能都是对 JMS 提供商的要求，除非它被显式的指出。

JMS 点对点功能的提供商不要求提供发布/订阅功能，反之亦然。

JMS 也可以用在 J2EE 平台。参加章节 1.4 ，“与其他 Java API 的关系”来了解当 JMS 被集成到那种软件环境时对 JMS 的附加要求。

1.4 与其他 Java API 的关系

1.4.1 JDBC 软件

JMS 客户端也可以使用 JDBC API。他们可能希望在同一个事务内使用 JDBC API 和 JMS API。大多数情况下，通过将这些客户端实现为 EJB 组件可以自动实现这个愿望。也可能直接使用 JTA 来实现这个愿望。

1.4.2 JavaBean 组件

JavaBean 组件可以使用 JMS 会话来发送/接收消息。JMS 本身是一个 API，它定义的接口没有打算直接作为 JavaBean 组件来使用。

火龙果•整理
uml.org.cn

1.4.3 EJB 组件模型

JMS API 是 EJB 组件开发者可以获得的重要资源。它可以和类似 JDBC 的资源一起使用来实现企业服务。

EJB2.0 规范定义了通过来自 EJB 客户端调用的方法被同步调用的 bean。也定义了一种异步 bean，当一个 JMS 客户端向他发送消息时它被调用，称为消息驱动 bean。EJB 规范支持同步和异步消息消费。另外，EJB2.0 指定 JMS API 如何参与 bean 管理的或容器管理的事务。EJB2.0 规范限制了当实现 EJB 客户端时如何使用 JMS 接口。参考 EJB2.0 规范来了解更详细的内容。

1.4.4 Java 事务 API（JTA）

Javax.transaction 包为划分分布式事务提供了客户端 API，并提供了获取要参与到分布式事务中的资源的 API。

JMS 客户端可以使用 JTA 来划分分布式事务；但是，这是运行客户端的事务环境的功能。不是 JMS 的功能。

JMS 提供商可以通过 JTA 支持分布式事务，也可以不支持。

1.4.5 Java 事务服务（JTS）

JMS 可以和 JTS 联合使用来组成分布式事务，这个分布式事务将消息的发送和接收与数据更新及其他 JTS 感知的服务结合起来。当 JMS 客户端在应用服务器中（如 EJB 服务器）被运行时，分布式事务应当被自动处理；但是对 JMS 客户端来说，显式的通过程序使用 JTS 也是可能的。

1.4.6 Java 命名和目录接口 API（JNDI）

JMS 客户端使用 JNDI API 查找已配置的 JMS 对象。JMS 管理员使用提供商提供的工具来创建和配置这些对象。

通过将特定提供商的工作代理给管理员最大化了客户端的可移植性。它也导致了更多的可管理应用，因为客户端不需要将管理用的值嵌入到他们的代码中。

1.4.7 J2EE 平台

J2EE 平台规范（版本 1.3）要求将 JMS API 作为 J2EE 平台的一部分。J2EE 平台规范对 JMS 的实现提出了附加的要求，这些要求超出了 JMS 规范中描述的要求，包括既要支持点对点域又要支持发布/订阅域。

火龙果•整理
uml.org.cn

1.4.8 JMS 和 EJB 组件的集成

J2EE 平台和 EJB 规范描述了要集成到 J2EE 平台的 JMS 提供商实现的附加要求。一个关键的需求集是 JMS 消息生产和 JMS 消息消费如何与容器管理事务的事务需求进行交互。参考这两个规范来来了解 JMS 集成的所有要求。

JMS API 规范没有说明实现这些集成要求的模型。因此，不同的 JMS 提供商实现可以使用不同的方式来实现与 J2EE 平台的集成，以及支持 EJB 的要求。

将来，JMS 集成到 J2EE 平台的集成点将用 J2EE 连接器架构来提供。

1.5 JMS1.1 的新特性是什么？

在 JMS 的以前版本中，用于点对点和 Pub/Sub 域的客户端编程都是类似的，但使用不同的类层次。在 JMS1.1 中，现在有一个不依赖域的方式来编写客户端应用。这有以下几个好处：

- 对于客户端程序员，简化了编程模型。
- 提供了在同一事务中使用队列（Queue）和主题（topic）的能力，现在可以在同一个会话内创建它们。
- 对于 JMS 提供商，通过线程池管理增加了优化实现的机会。

为使用这些优点，JMS 客户端开发者需要使用不依赖域的或“通用”的 API。将来，某些域专有的 API 可能被废弃。

在 JMS1.1 中，所有来自 JMS1.0.2b 的类和方法仍然被保留以提供向后的兼容性。两种消息域的语义也被保留；PTP 域和 Pub/Sub 域的行为仍然是相同的，正如在第 5 章“JMS 点对点模型”和第 6 章“JMS 发布/订阅模型”描述的一样。

为了详细的了解本规范的变更情况，参见第 11 章“变更历史”。

2 架构

2.1 概述

本章描述基于消息的应用的环境和 JMS 在环境中扮演的角色。

2.2 什么是 JMS 应用

JMS 应用由以下部分组成：

- JMS 客户端——发送和接收消息的 Java 语言程序。
- 非 JMS 客户端——使用消息系统的本地客户端 API 而不是 JMS 的 API，如果应用先于 JMS，那么它很可能既包含 JMS 又包含非 JMS 客户端。
- 消息——每个应用定义一系列的消息，这些消息用于在客户端之间交换信息。
- JMS 提供商——它是一个消息系统，它实现了 JMS 以及其他的完整消息产品所需要的管理和控制功能。

火龙果•整理
uml.org.cn

- 被管理的对象——被管理的对象是预先配置好的 JMS 对象，它由管理员为客户端使用而创建。

2.3 管理

期望 JMS 提供商和它们的后台消息技术有大的区别。也期望在如何安装和管理提供商的系统也有大的区别。

如果 JMS 客户端是可移植的，那么他们必须与提供商专有的方面隔离开来。通过定义 JMS 被管理的对象来做到隔离，这些对象由提供商的管理员创建和客户化，接下来由客户端使用。通过 JMS 接口来使用它们的客户端是可移植的。管理员使用提供商专有的工具来创建它们。

有两种类型的 JMS 被管理对象：

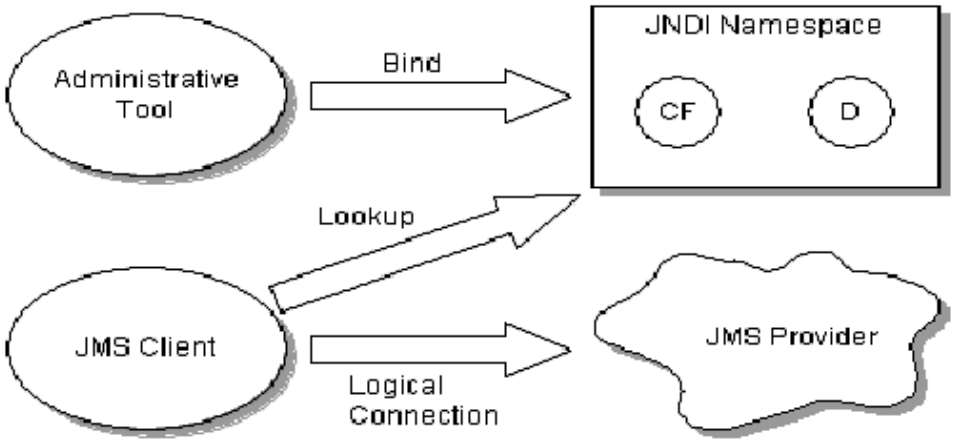
- ConnectionFactory——这个对象用于客户端来创建和提供商的连接。
- Destination——这个对象用于客户端来指定发送消息的目的地，也是接收消息的来源。

被管理对象被管理员放置在 JNDI 命名空间。JMS 客户端通常在它的文档中注上它要求的 JMS 被管理对象和这些对象的 JNDI 名字应当如何提供给它。

图 2-1 解释了 JMS 管理如何工作。

图 2-1 JMS 管理

Figure 2-1 JMS Administration



2.4 两种消息风格

JMS 应用既可以使用 PTP 风格又可以使用 Pub/Sub 消息风格，在后面的章节中会有更详细的描述。应用也可以将两种风格组合到一个应用中。消息的这两种风格常被称为消息域。JMS 提供了这两种消息域，因为它们代表了两种通用的消息模型。

当使用 JMS API 时，开发人员可以使用两种消息模型的接口和方法。当使用接口时，消息系统的行为可能会稍有不同，因为两种消息域有不同的语义。这些语义的差别在第 5 章“JMS 点对点模型”和第 6 章“JMS 发布/订阅模型”中描述。

火龙果•整理
uml.org.cn

2.5 JMS 接口

JMS 基于一系列通用的消息概念。每个 JMS 消息域—PTP 和 Pub/Sub—也为这些概念定义了各自的接口集。

表 2-1 PTP 和 Pub/Sub 接口的关系

JMS 公共接口	PTP 专有接口	Pub/Sub 专有接口
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver，QueueBrowser	TopicConsumer

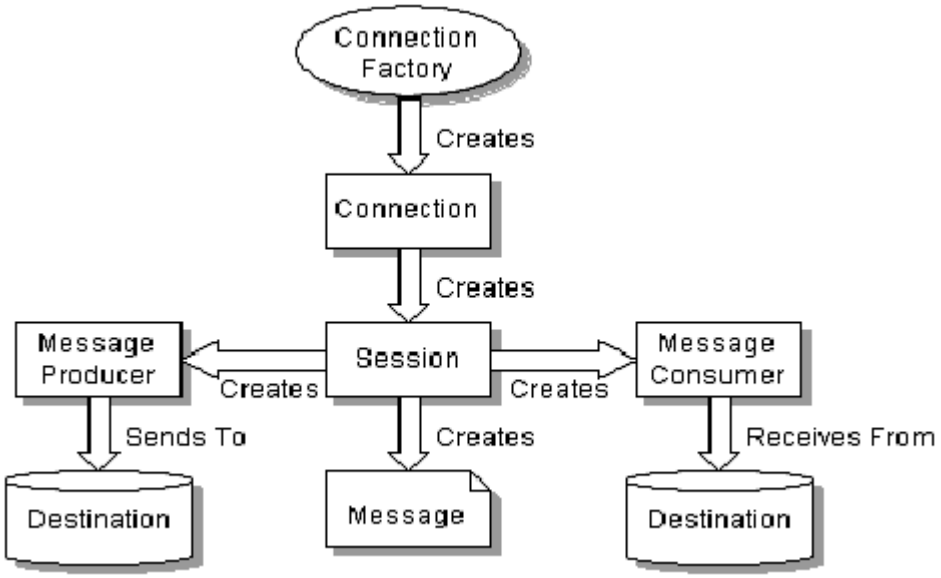
JMS 通用接口提供了一个独立于 PTP 和 Pub/Sub 消息域的域视图。鼓励 JMS 客户端程序员使用这些接口来创建他们的客户端程序。

下面列出了这些 JMS 概念的简要定义。参见第 4 章“JMS 通用工具”来详细了解这些概念。

对于两种消息域的差别的详细内容，参见第 5 章“JMS 点对点模型”和第 6 章“JMS 发布/订阅模型”。

- ConnectionFactory——客户端使用这个被管理对象来创建一个 Connection。
- Connection——一个到 JMS 提高商的活动连接。
- Destination——封装了消息目的地标识的被管理对象。
- Session——一个用于发送和接收消息的单线程上下文。
- MessageProducer——一个由 Session 创建用于往目的地发送消息的对象。
- MessageConsumer——一个由 Session 创建用于接收发送到目的地的消息的对象。

图 2-1 JMS 对象间关系概览



在这个文档中使用的术语“消费”是指通过 JMS 客户端接收消息；也就是说，一个 JMS 提供商已经收到一个消息并将它给了它的客户端。由于 JMS 支持同步和异步接收消息，因

火龙果•整理
uml.org.cn

此术语“消费”在不需要区分它们的时候使用。

术语“生产”用作发送消息的最通用的术语。它指给予 JMS 提供商一个消息以转发到一个目的地。

2.6 开发一个 JMS 应用

广义的讲，一个 JMS 应用是一个或多个交换消息的 JMS 客户端。应用也涉及非 JMS 客户端；但是，这些非 JMS 客户端使用 JMS 提供商的本地 API 而不是 JMS 的 API。

一个 JMS 应用可以被架构和部署为一个单元。在许多情况下，JMS 客户端是被逐渐增加到线程的应用中的。

应用使用的消息定义可以来源于 JMS，或者他们可能已经由应用的非 JMS 部分定义了。

2.6.1 开发一个 JMS 客户端

典型的 JMS 客户端执行下面的 JMS 设置过程：

- 使用 JNDI 来发现 ConnectionFactory 对象。
- 使用 JNDI 来发现一个或多个 Desitination 对象。
- 使用 ConnectionFactory 来创建一个具有消息转发约束的 JMS Connection。
- 使用 Connection 来创建一个或多个 JMS Session。
- 使用 Session 和 Destination 来创建需要的 MessageProducer 和 MessageConsumer。
- 告诉 Connection 开始转发消息。

此时，客户端有了生产和消费消息的基本的 JMS 设置。

2.7 安全

JMS 没有提供控制或配置消息完整或消息私密的特性。

这期望由 JMS 提供商来提供。也期望由提供商专有的管理工具来配置这些服务。客户端将得到正确的安全配置作为他们使用的被管理对象的一部分。

2.8 多线程

JMS 已经要求它的所有对象都支持并发誓要。由于支持并非访问通常会增加一些成本和复杂性，所以 JMS 限制了那些很自然要被多线程客户端并发访问的对象的要求。其他的都被设计成在一个时间只能被一个逻辑线程访问。

表 2-2 支持并发使用的 JMS 对象

JMS 对象	支持并非使用
Destination	YES
ConnectionFactory	YES
Connection	YES
Session	NO
MessageProducer	NO

火龙果•整理
uml.org.cn

MessageConsumer	NO

JMS 定义了一些特殊的规则来限制 Session 的并发使用。由于他们要求比在这里呈现的更多的 JMS 知识，所以他们将在后面描述。这里我们将描述他们必须遵守的基本原理。

有两个原因要限制并发访问 Session。第一个，Session 是支持事务的 JMS 实体。实现多线程的事务是非常困难的。第二，Session 支持异步的消费消息。重要的是 JMS 不要求用于消费异步消息的客户端代码有处理多个并发的消息的能力。另外，如果 Session 被设置有多个异步的消费者，那么不强迫客户端处理这些并发执行的分散的消费者。这些约束使得通常的客户端更加容易地使用 JMS。更专业的客户端可以通过使用多个会话来获得他们期望的并非性。

2.9 触发式客户端

某些客户端被设计成被周期性的唤醒来处理等待它们的消息。一个基于消息的应用触发机制常和客户端的风格一起使用。触发器通常是等待消息的起点，等等。

JMS 没有提供触发客户端执行的机制。某些提供商可以通过它们的管理工具提供这样的触发机制。

2.10 请求/回复

JMS 提供了 JMSReplyTo 消息头字段来指定回复的消息应当被发送到的目的地。回复的 JMSCorrelationID 头字段可以被用于引用原始的请求。参见 3.4 章节“消息头字段”了解详细信息。

另外，JMS 提供了创建临时队列和主题的功能，它可以用作回复的唯一目的地。

企业消息产品支持许多请求/回复风格，从简单的“一个消息请求产生一个消息回复”到“一个消息请求产生一串回复，回复消息来自多个应答者”。JMS 不仅架构了一个特定的请求/回复抽象，而且提供了基本的工具，许多企业消息产品可以基于它来构建。

为了方便，JMS 为 PTP 和 Pub/Sub 域定义了请求/响应帮助类（这些类用 JMS 实现），它实现了请求/回复的基本形式。JMS 提供商和客户端可以提供更加专业的实现。

3 JMS 消息模型

3.1 背景

企业消息产品把消息看作是轻量级的实体，它由消息头和消息体组成。消息头包含了用于消息路由和标识的字段；消息体包含了被发送的应用数据。

通常情况下，消息的定义随产品的不同有很大的差别。主要的差别是消息头的内容和语义。某些产品使用自描述的规范的消息数据编码；其他的可能将数据认为是完全不透明的。某些产品为能被用于标识和解释消息内容的消息描述提供了存储池；其他的就没有。

对 JMS 来讲捕获偶尔冲突的消息模型组合的范围是非常困难的。

火龙果•整理
uml.org.cn

3.2 目标

JMS 消息模型有下列目标：

- 提供一个单一的统一的消息 API。
- 提供一个 API，它能创建匹配现存的非 JMS 应用使用的格式的消息。
- 支持异构应用的开发，异构应用涵盖操作系统、机器架构和计算机语言。
- 支持包含 java 对象的消息。
- 支持包含可扩展标签语言（XML）页面的消息（参见 http://www.w3.org/XML）。（译者注：就是消息是 XML）

3.3 JMS 消息

JMS 消息由以下部分组成：

- 消息头——所有的消息都支持相同的头字段集。头字段包含了客户端和提供商都要使用的用于标识和路由消息的值。
- 属性——除了标准的头字段外，消息提供了一个内置的功能来向消息增加可选的头字段。
  - 应用专有属性—为消息增加应用专有的头字段提供的机制。
  - 标准属性—JMS 定义的一些标准属性，它们相当于可选的头字段。
  - 提供商专有属性—在集成 JMS 客户端和 JMS 提供商本地客户端时可能会用到提供商专有的属性。JMS 为这些属性定义了命名规范。
- 消息体——JMS 定义了几个消息体类型，这些类型覆盖了大部分当前使用的消息风格。

3.4 消息头字段

下面的子章节描述了所有的消息头字段。消息的头被完整地转发到所有的 JMS 客户端。JMS 没有定义那些转发到非 JMS 客户端的头字段。

3.4.1 JMSDestination

JMSDestination 包含了消息被发往的目的地。

当消息被发送时，忽略这个字段。在发送完成后，这个字段容纳由发送方法指定的目的地对象。

当消息被接收时，它的目的地的值必须等于发送时赋给的值。

3.4.2 JMSDeliveryMode

JMSDeliveryMode 包含了消息发送时指定的转发模式。

当消息被发送时，这个字段被忽略。在完成发送后，它包含了由发送方法指定的转发模式。

火龙果•整理
uml.org.cn

参见 4.7 章节“消息转发模式”了解更详细的信息。

3.4.3 JMSMessageID

JMSMessageID 包含了一个用于唯一标识由提供商发送的每个消息。

当消息被发送时，JMSMessageID 被忽略。当发送方法返回时，这个字段包含了一个提供商赋予的值。

JMSMessageID 是一个 String 值，它用于在历史存储池中唯一标识消息的主键。唯一性的确切范围由提供商定义。它至少应当覆盖一个提供商安装的所有的消息，这里一个安装被连接到一系列消息路由器。

所有的 JMSMessageID 值必须以前缀“ID:”开始，没有要求跨不同提供商的消息 ID 值的唯一性。

由于消息 ID 需要花费时间来创建并增加了消息的长度，因此如果消息已经被给了一个暗示说英语不使用消息 ID，那么 JMS 提供商可以优化过长的消息。JMSMessageProducer 提供取消消息 ID 的暗示。当客户端设置消息生产者不使用消息 ID 时，它就是说它生产的消息不依赖消息 ID 的值。如果 JMS 提供商接受这个暗示，那么这些消息的消息 ID 必须设置为 null；如果提供商不接受这个暗示，那么必须为消息 ID 设置一个唯一的值。

3.4.4 JMSTimestamp

JMSTimestamp 包含了消息被发送的时间。但不是消息被真正转发的时间，因为真正的发送可能由于事务或其他的客户端消息排队而比较晚。

当一个消息被发送时，JMSTimestamp 被忽略。当发生方法返回时，这个字段包含了调用和返回之间的某个时间值。它的格式是通常的 java 毫秒时间值。

由于时间戳需要花时间来创建，并增加了消息的长度，如果应用暗示不使用时间戳，那么某些 JMS 提供商可以优化消息的过载。JMS MessageProducer 可以暗示不使用时间戳。当客户端设置生产者不使用时间戳，那么它就是说它生产的消息不依赖于时间戳的值。如果 JMS 提供商接受了这种暗示，那么这些消息的时间戳必须被设置成 0；如果提供商不接受这种暗示，则必须设置时间戳的值。

3.4.5 JMSCorrelationID

客户端可以使用 JMSCorrelationID 来链接消息。典型的用法就是将响应消息和请求消息链接起来。

JMSCorrelationID 可以容纳下列的类型值：

- 提供商专有的消息 ID
- 应用专有的 String
- 提供商本地的 byte[]值

由于每个由 JMS 提供商发送的消息都被赋予了一个消息 ID 值，所以通过消息 ID 来链接消息是非常方便的。所有的消息 ID 值都必须以“ID:”作为前缀。

在某些情况下，应用（有几个客户端组成）需要使用应用专有的值来链接消息。例如，应用可以使用 JMSCorrelationID 来容纳一个引用一些外部信息的值。应用专有的值不必以“ID:”

火龙果•整理
uml.org.cn

为前缀；这是提供商生成消息 ID 值时的保留字符。

如果提供商支持关联 ID 的本地概念（native），那么 JMS 客户端可能需要给 JMSCorrelationID 赋特定的值来匹配非 JMS 客户端期望的值。Byte[]值就是用于这个目的。没有本地关联 ID 的 JMS 提供商不要求支持 byte[]值（注：setJMSCorrelationIDAsBytes()和 getJMSCorrelationIDAsBytes() 可以抛出 java.lang.UnsupportedOperationException）。为 JMSCorrelationID 使用 byte[]值是不可移植的。

3.4.6 JMSReplyTo

当消息被发送时，JMSReplyTo 包含一个由客户支持的目的地。它是回复消息应当被发送到的目的地。

JMSReplyTo 为 null 的消息可能是某个事件的通知消息或它们仅仅是发送者认为是有兴趣的数据。

JMSReplyTo 有值得消息通常是期望响应的消息。响应是可选的；由客户端来决定。

3.4.7 JMSRedelivered

如果客户端收到一个设置了 JMSRedelivered 指示的消息，那么很可能但不能保证这个消息被转发过但没有确认。通常情况下，提供商必须设置 JMSRedelivered，无论它是否正在重新转发一个消息。如果 JMSRedelivered 设置为 true，那么它告诉消费应用这个消息可能已经被转发过，应用应当引起注意以免重复处理。参加 4.4.11 章节“消息确认”了解更详细的信息。

这个头字段对发送没有意义，不会被发送方法赋值。

3.4.8 JMSType

JMSType 包含了由客户端在发送消息时提供的消息类型标识。

某些 JMS 提供商使用消息存储池，这个池包含了由应用发送的消息的定义。Type 字段可以引用提供商池内的消息定义。

JMS 没有定义标准的消息定义池，也没有定义定义的命名策略。

某些消息系统要求为每个应用消息创建消息类型定义并指定每个消息的类型。为了和这样的 JMS 提供商一起工作，无论应用是否使用它 JMS 客户端都应当为 JMSType 赋值。这保证为那些需要这个字段的提供商提供正确的设置。

为了保证可移植性，JMS 客户端应当使用抽象符合为 JMSType 赋值，以便它能在安装时被配置成当前提供商消息存储池中定义的值。如果使用字符串，对某些 JMS 提供商来说他们可以不是有效的类型名。

3.4.9 JMSExpiration

当消息被发送时，它的到期时间是在发生方法中指定的存活时间和当前 GMT 值之和。在从发送方法返回时，消息的 JMSExpiration 头字段包含这个到期的值。当消息被接收时，它的 JMSExpiration 中的值就是发送时的值。

火龙果•整理
uml.org.cn

如果存活时间指定为 0，那么表明消息没有到期时间。

当当前 GMT 已经晚于未转发消息的到期时间时，应当销毁这个消息。JMS 没有定义消息到期时的通知。

3.4.10JMSPriority

JMSPriority 头字段包含了消息的优先级。

当消息被发送时，这个字段被忽略。在完成发送后，它容纳了由发送方法指定的优先级值。

JMS 定义了十个优先级值，0 是最低的优先级，9 是最高的优先级。另外，客户端应当将 0-4 看作普通优先级，5-9 看作加急优先级。

JMS 没有要求提供商严格地实现消息的优先级顺序；但是，它应当尽力在普通消息之前转发加急消息。

3.4.11如何设置消息头的值

表 3-1 发送时设置消息头字段的值

头字段	设置者
JMSDestination	Send Method
JMSDeliveryMode	Send Method
JMSExpiration	Send Method
JMSPriority	Send Method
JMSMessageID	Send Method
JMSTimestamp	Client
JMSCorrelationID	Client
JMSReplyTo	Client
JMSType	Client
JMSRedelivered	Provider

3.4.12重载消息头字段

JMS 可以让管理员配置 JMS 来重载客户端特有的 JMSDeliveryMode、JMSExpiration 和 JMSPriority 的值。如果重载了，头字段的值必须是管理员指定的值。

JMS 没有特别定义管理员如何重装这些头字段的值。没有要求支持 JMS 提供商支持这个管理选项。

3.5 消息属性

除了这里定义的头字段外，Message 接口有一个内置的功能，这个功能支持属性值。这个功能为消息增加可选头字段提供了一种机制。

属性可以让客户端通过消息选择器（参见章节 3.8“消息选择”）来让 JMS 提供商根据

火龙果•整理
uml.org.cn

应用特有的规则来选择消息。

3.5.1 属性名

属性名必须遵循消息选择器标识的规则。参见章节 3.8.1.1“消息选择器语法”了解更详细信息。

3.5.2 属性值

属性的值可以是 boolean，byte，short，int，long，float，double 和 String。

3.5.3 属性的使用

在发送消息之前设置属性值。当客户端接收到一个消息时，它的属性是只读模式。如果客户端企图修改属性，那么抛出 MessageNotWriteableException。

可以在消息体中保存一份属性值的副本，也可以不保存。尽管 JMS 没有定义属性哪些作为属性或哪些不应当作为属性的策略，但应用开发者应当注意 JMS 提供商很可能在消息体内处理数据的效率比处理在属性中的数据高。为了提高性能，应用应当只在需要客户化消息头时才使用消息属性。客户化消息头的主要原因是为了支持客户化的消息选择。

参见章节 3.8“消息选择”来了解 JMS 消息属性的详细信息。

3.5.4 属性值转换

属性支持下述的转换表。打标记的情况必须支持。没有打标记的情况必须抛出 MessageFormatException。如果数值的 valueOf 方法认为传入的 String 不是有效的数值，那么 String 到数值的转换必须抛出 java.lang.NumberFormatException。企图读取一个 null 值作为 java 原始类型必须看作是调用原始类型的 valueOf(String)转换方法来转换 null 值。

设置为行总的类型，读作列中的类型。

表 3-2 属性值转换

	boolean	byte	short	int	long	float	double	String
boolean	×							×
byte		×	×	×	×			×
short			×	×	×			×
int				×	×			×
long					×			×
float						×	×	×
double							×	×
String	×	×	×	×	×	×	×	×

火龙果•整理
uml.org.cn

3.5.5 属性值作为对象

除了特定类型的属性有 set/get 方法外，JMS 提供了 setObjectProperty/getObjectProperty 方法。这两个方法支持设置使用对象化的原始类型的属性。它们用于在运行时来决定属性类型而不是在编译时决定。它们支持同样的属性值转换。

setObjectProperty 方法接受 Boolean，Byte，Short，Integer，Long，Float，Double 和 String 的值。企图使用其他的类必须抛出 MessageFormatException。

getObjectProperty 方法只返回 null，Boolean，Byte，Short，Integer，Long，Float，Double 和 String 的值。如果指定的属性名不存在则返回 null。

3.5.6 属性迭代

没有定义属性值的排序。为了通过消息的属性值来迭代，那么使用 getPropertyNames 来取得所有的属性名，然后使用不同的属性 get 方法来取出它们的值。

方法 getPropertyNames 没有返回 JMS 标准头字段的名字。

3.5.7 清除消息属性的值

通过 clearProperties 方法来删除消息的属性。这会使消息有一个空的属性集。然后可以创建和读取新的属性条目。

清除消息的属性条目不会清理消息体的值。

一旦私有属性条目被添加到消息中，则 JMS 没有提供删除它的途径。

3.5.8 不存在的属性

按名字获取一个还没有被赋值的属性值则按照属性被赋了 null 值来处理。

3.5.9 JMS 定义的属性

JMS 保留了“JMSX”作为 JMS 属性名的前缀。这些属性都在表 3-3 中。新的 JMS 定义的属性可能在后续版本中增加。

除非说明否则支持这些属性是可选的。ConnectionMetaData.getJMSXPropertyNames()方法返回所有连接支持的 JMSX 属性的名字。

无论连接是否支持 JMSX 属性，它们都可以在消息选择器中使用。如果消息中没有这些属性，那么它们与其他缺席属性一样看待。

在特定消息中，存在 JMS 定义的属性，它们是由 JMS 提供商根据如何控制属性的使用来设置的。根据管理或其它规则，可以在某些消息中包含它们在其他消息中忽略它们。

表 3-3 JMS 定义的属性

名字	类型	设置者	用法
JMSXUserID	String	发送时提供商设置	发送消息的用户标识
JMSXAppID	String	发送时提供商设置	发送消息的应用标识

火龙果•整理
uml.org.cn

JMSXDeliveryCount	int	发送时提供商设置	转发消息重试次数；第一次是 1，第二次是 2，…
JMSXGroupID	String	客户端	消息所在消息组的标识
JMSXGroupSeq	int	客户端	组内消息的序号；第一个消息是 1，第二个是 2，…
JMSXProducerTXID	String	发送时提供商设置	产生消息的事务的事务标识
JMSXConsumerTXID	String	接收时提供商设置	消费消息的事务的事务标识
JMSXRcvTimestamp	long	接收时提供商设置	JMS 转发消息到消费者的时间
JMSXState	int	提供商	假定存在一个消息仓库，它存储了每个消息的单独拷贝，且这些消息从原始消息被发送时开始。 每个拷贝的状态有：1（等待），2（准备），3（到期）或 4（保留）。由于状态与生产者和消费者无关，所以它不是由它们来提供。它只和在仓库中查找消息相关，因此 JMS 没有提供这种 API。

消息生产者和消费者都可以获取由提供商在发送时设置的 JMSX 属性。在接收时由提供商设置的 JSMX 属性只能由消费者获取。

如果客户端希望对消息进行分组，那么 JMSXGroupID 和 JMSXGroupSeq 是标准的属性。所有的提供商必须支持他们。

如果要使用 JMSX 属性，则必须用上表中定义的名字。

除非特殊说明，否则 JMSX 属性的值和语义是未下定义的。

3.5.10提供商专有的属性

JMS 为提供商专有的属性保留了“JMS_<vendor_name>”属性名前缀。每个提供商定义了他们自己的<vendor_name>值。JMS 提供商使用这个机制来让 JMS 客户端可以获取它的所有消息服务。

提供商专有属性的目的是为提供商本地客户端提供特殊功能。他们不应当用于 JMS。

火龙果•整理
uml.org.cn

3.6 消息确认

当客户端指定 JMS 消费者的消息要显式地被确认时，所有的 JMS 消息使用 acknowledge 方法来确认消息。

如果客户端使用自动确认，那么忽略对确认方法的调用。

参见 4.4.11 章节“消息确认”了解更详细的信息。

3.7 Message 接口

Message 接口是所有 JMS 消息的根接口。它定义了 JMS 消息头字段、属性工具和 acknowledge 方法。

3.8 消息选择

许多消息应用需要过滤和分类它们生产的消息。

在消息被发送到单个接收者的情况下，通过将标准放入消息中并在接收客户端丢弃它不感兴趣的消息来实现。

当消息被广播到许多客户端时，将选择标准放入消息头中是非常有用的，这样 JMS 提供商就可以知道这些标准。这样提供商就可以处理更多的过滤和路由工作，否则就需要应用来做这些工作。

JMS 提供了让客户端将消息选择代理给 JMS 提供商的功能。这简化了客户端的工作，也可以让 JMS 提供商减少时间和带宽，否则它们将会将客户端不需要的消息发送给客户端。

客户端使用消息属性将应用专有的选择标准附加到消息中。客户端用消息选择器表达式来指定消息选择标准。

3.8.1 消息选择器

JMS 消息选择器让客户端通过消息头指定它感兴趣的消息。只有头和属性匹配了选择器的消息才会被转发。“不转发”的语义随着使用的 MessageConsumer 的不同而有所不同。参见 5.8 章节“QueueReceiver”和 6.11 章节“TopicSubscriber”了解更详细的信息。

消息选择器不能引用消息体内的值。

当消息头字段和属性值与它们在选择器中对应的标识符匹配时，如果选择器计算值为 true，那么消息选择器匹配了一个消息。

3.8.1.1消息选择器语法

消息选择器是一个 String，它的语法是 SQL92 条件表达式语法的子集。（注：参见 X/Open CAE 规范数据管理：结构化查询语言（SQL），版本 2，ISBN:1-85912-151-9 1996 年 3 月）

如果消息选择器的值是空串，那么值被看作是 null，表示消息消费者没有消息选择器。

消息选择器的计算顺序是从左到右。圆括号可以改变这个顺序。

预定义的选择器文法和操作符名字用大写字符出现在这里；但是它们是大小写敏感的。

火龙果•整理
uml.org.cn

选择器可以包含：

- 文法
  - 字符串用单引号括起来，在字符串内出现的单引号使用两个单引号表示（即用单引号做转义符）；例如'literal'和'literal''s'。类似 java 的 String 文法，它们使用 Unicode 字符编码。
  - 精确数值文法是没有小数点的数值，例如 57，-957，+62；支持 Java 的 long 取值范围。精确数值文法使用 java 的 Integer 文法语法。
  - 近似数值文法是科学计数法，例如 7E3 和-57.9E2，或带小数的数值，例如 7.，-95.7 和+6.2；支持 double 的取值范围。近似数值文法使用 java 浮点数文法语法。
  - 布尔文法是 TRUE 和 FALSE。
- 标识符
  - 标识符是无限长的字符串，它必须以 java 标识符开始字符开头；下述的所有字符必须是 java 的标识符局部字符（part character）。标识符开头字符可以是方法 Character.isJavaIdentifierStart 返回 true 的任何字符。包括'_'和'$'。标识符局部字符是方法 Character.isJavaIdentifierPart 返回 true 的任何字符。
  - 标识符不能是 NULL，TRUE 或 FALSE。
  - 标识符不能是 NOT，AND，OR，BETWEEN，LIKE，IN，IS 或 ESCAPE。
  - 标识符可以是头字段引用或属性引用。消息选择器中的属性值类型与用于设置属性的类型一致。如果引用了消息中不存在的属性，那么它的值是 NULL。在选择器重 NULL 值的计算在节 3.8.1.2“Null 值”描述。
  - 应用到属性 get 方法的转换不应用到那些消息选择器表达式使用的属性。例如，假定你为一个属性设置了字符串值，如下：myMessage.setStringProperty(“NumberOfOrders”,”2”);那么下面的在消息选择器中的表达式计算值将是 false，因为字符串不能用在数学表达式中：”NumverOfOrders > 1”
  - 标识符是大小写敏感的。
  - 消息头字段引用仅限于 JMSDeliveryMode，JMSPriority，JMSMessageID，JMSTimestamp，JMSCorrelationID 和 JMSType。JMSMessageID，JMSCorrelationID 和 JMSType 值可以是 null，且如果是 null 则被看作是 NULL 值。
  - 任何以“JMSX”开头的名字是 JMS 定义的属性名。
  - 任何以“JMS_”开头的名字是提供商专有的属性名。
  - 任何不以“JMS”开头的名字是应用专有的属性名。
- 空格和 Java 定义的一样：空格，水平 tab，格式填充符（form feed）和行结束符。
- 表达式：
  - 选择器是条件表达式；选择器计算为 true 则匹配，计算为 false 获未知则不匹配。
  - 算术表达式由算术表达式，算术运算符，值是数值的标识符和数值文法组成。
  - 条件表达式由条件表达式，比较运算符，逻辑运算符，值是布尔的标识符和布尔文法组成。
- （）用于改变表达式计算的顺序。
- 逻辑运算符的顺序优先级：NOT，AND，OR
- 比较运算符：=，>，>=，<，<=，<>（不等于）
  - 只有类似类型的值才可以进行比较。用于比较精确数值和近似数值的表达式是

火龙果•整理
uml.org.cn

有效的表达式（类型转换的要求由 Java 数值提升规则（译者注：即低精度向高精度自动转换）定义）。如果比较非类似类型的值，那么返回 false。如果类型的值是 NULL，那么表达式的值是未知。

- String 和 Boolean 比较只能使用=和<>。有且只有两个字符串有相同的字符序列，它们才是相等的。

- 算术运算符的优先顺序：
  - +,-（一元）
  - *,/（乘和除）
  - +,-（加法和减法）
  - 算术运算符必须使用 java 数值提升
- *arithmetic-exp1* [NOT] BETWEEN *arithmetic-exp2* AND *arithmetic-exp3*(比较操作)
  - “age BETWEEN 15 AND 19”等价于”age >= 15 AND age <= 19”
  - “age NOT BETWEEN 15 AND 19”等价于”age <15 OR age > 19”
- *identifier* [NOT] IN (*string_literal1,string_literal2,…*)（比较操作，*identifier* 是 String 或 NULL）
  - “Country IN (‘UK’,’US’,’France’)”，对于’UK’返回 true，对于’Peru’返回 false；等价于表达式”(Country = ‘UK’) OR (Country = ‘US’) OR (Country = ‘France’)”
  - “Country NOT IN (‘UK’,’US’,’France’)”，对于’UK’返回 false，对于’Peru’返回 true；等价于表达式”NOT ((Country = ‘UK’) OR (Country = ‘US’) OR (Country = ‘France’))”
  - 如果 IN 或 NOT IN 操作的 *identifier* 是 NULL，那么操作的结果是未知的。
- *identifier* [NOT] LIKE *pattern-value* [ESCAPE *escape-character*] （比较操作，*identifier* 是 String；*pattern-value* 是字符串文法，其中’_’代表任意单个字符；’%’代表任何字符串，包括空串；其他的字符代表自己。选项 *escape-character* 是单字符的字符串文法，它用于忽略在 *pattern-value* 中’_’和’%’的特定含义。）
  - “phone LIKE ‘12%3’” 对于’123’或’12993’是 true，对于’1234’是 false
  - “word LIKE ‘l_se’” 对于’lose’是 true，对于’loose’是 false
  - “underscored LIKE ‘_%’ ESCAPE ‘\’” 对于’_foo’是 true，对于’bar’是 false
  - “phone NOT LIKE ‘12%3’” 对于’123’或’12993’是 false，对于’1234’是 true
  - 如果 LIKE 或 NOT LIKE 操作的 *identifier* 是 NULL，那么操作的结果是未知的
- *identifier* IS NULL（测试头字段值是否为空或属性值是否缺失的比较操作）
  - “prop_name IS NULL”
- *identifier* IS NOT NULL（测试非空头字段值或属性值是否存在的比较操作）
  - “prop_name IS NOT NULL”

JMS 提供者需要在呈现消息选择器时验证语法的正确性。提供语法错误的选择器的方法必须产生 JMS InvalidSelectorException。JMS 提供者也可以可选的在呈现选择器时提供语义检查。并非所有的语义检查都能在呈现消息选择器时执行，因为属性的类型还不知道。

下面的消息选择器用消息类型是 car，颜色是 blue 和重量大于 2500lbs 的消息选择器来选择消息：

“JMSType = ‘car’ AND color = ‘blue’ AND weight > 2500”

火龙果•整理
uml.org.cn

3.8.1.2 Null 值

如上所述，头字段和属性值都可以是 NULL。包含 NULL 值的选择器表达式的计算由 SQL 92 NULL 语义定义。这里提供了这些语义的简要描述。

SQL 认为 NULL 值是未知值。用未知值比较或运算仍然是未知的。

IS NULL 和 IS NOT NULL 操作符将未知的头或属性值分别转换成 TRUE 或 FALSE 值。

布尔操作符使用下表定义的三值逻辑：

表 3-4 AND 操作符的定义

AND	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

表 3-5 OR 操作的定义

OR	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

表 3-6 NOT 操作的定义

NOT	
T	F
F	T
U	U

3.8.1.3 特别注意

当在消息选择器中使用 JMSDeliveryMode 时，它被看作值是 'PERSISTENT' 和'NON_PERSISTENT'。

日期和时间值应当使用标准的 Java long 毫秒值。当消息选择器中包含日期或时间文法时，它的值应当是整型的。产生毫秒值得标准方式是使用 java.util.Calendar。

尽管 SQL 支持固定小数比较和算术运算，但 JMS 消息选择器不支持。这是因为精确数值是不能有小数的（而且，带小数的数值的加法是作为近似数值的备选呈现方式）。

不支持 SQL 注释。

3.9 访问已发送的消息

在发送消息后，客户端可以保留和更改这个消息，但不影响已发送的消息。同一个消息对象可以被多次发送。

在发送方法执行时，消息不能被客户端修改。如果修改了，则没有定义发送的结果是什么。

火龙果•整理
uml.org.cn

3.10 改变收到的消息的值

当接收到消息时，可以改变消息的头字段；但是他的属性和消息体是只读的，正如本章所述。

限制只读是因为为了给 JMS 提供商在如何实现管理接收到的消息方面更多的自由。例如，它们可以返回一个消息对象，这个对象引用位于内部消息缓冲区的属性条目和消息体的值，而不需要返回一个拷贝。

在调研 clearBody 或 clearProperties 方法使得消息体或属性可写后，消费者可以修改接收到的消息。如果消费者更改了接收到的消息，且消息随后被转发，那么转发的消息必须是原始的没有被改变的消息（除了被 JMS 提供商为转发而更改的头和属性外，例如 JMSRedilvered 头和 JMSXDeliveryCount 属性）。

3.11 JMS 消息体

JMS 提供了五种形式的消息体。每种形式都由一个消息接口来定义：

- StreamMessage——消息体包含的是 java 原始值流。它连续的填充和读。
- MapMessage——消息体包含一系列名字-值对儿，其中名字是 String，值是 Java 原始类型。条目可以被枚举器连续获取也可以按名字随机获取。条目的顺序没有定义。
- TextMessage——消息体包含的是 java.lang.String。这个消息类型是基于一个假设：String 消息被广泛的使用。这是因为 XML 很可能变成一个代表 JMS 消息内容的流行机制。
- ObjectMessage——消息包含了可序列化的 Java 对象。如果需要 java 对象的集合，那么可以使用在 JDK1.2 中提供的集合类。
- BytesMessage——消息包含了一个未解释的字节流。这个消息类型用于按字面编码的消息体去匹配一个存在的消息格式。在许多情况下，它可能用于一种其他未定义的消息类型。***尽管 JMS 允许消息属性使用字节消息，但通常不会使用，因为这样可能影响消息的格式。***

3.11.1清除消息体

Message 的 clearBody 方法重设消息体的值为'空'的初始化消息值，这个值由消息类型的由 Session 提供的创建方法设置。清除消息体不会清除它的属性条目。

3.11.2只读消息体

当消息被接收时，它的消息体是只读的。如果企图改变消息体，那么必须抛出 MessageNotWriteableException。如果消息体随后被清除，那么消息体的状态和新创建消息时的空消息体的状态一样。

火龙果•整理
uml.org.cn

3.11.3 由 StreamMessage 和 MapMessage 提供的转换

StreamMessage 和 MapMessage 都支持相同的原始数据类型集合。

这些类型可以被显式地使用类型对应的方法来读写。它们也可以都作为对象被读写。例如，调用 MapMessage.setInt("foo",6)等价于 MapMessage.setObject("foo",new Integer(6))。提供两种形式是因为显式的方式对静态的编程更方便，但对象形式在编译时不知道类型时是需要的。

StreamMessage 和 MapMessage 都支持下面的转换表。

标记为大写的必须支持。其他的必须抛出 MessageFormatException。如果传入数值的 valueOf 方法的 String 是无效的，那么 String 到数值的转换必须抛出 java.lang.NumberFormatException。

StreamMessage 和 MapMessage 必须实现 String 到布尔的转换，那和在 Java 语言中定义的 Boolean 的 valueOf(String)转换一样。

企图读取 null 值作为 java 的原始类型必须看作用 null 值调用原始类型对应的 valueOf(String)方法。由于 char 不支持 String 转换，因此企图读取 null 值作为 char 必须抛出 NullPointerException。

用名字获取还没有被设值的 MapMessage 字段按照 null 值处理。

如果 StreamMessage 好哦 BytesMessage 的读方法抛出 MessageFormatException 或 NumberFormatException，那么不能增加读指针所在的位置。随后的读必须能够通过重读数据作为不同的类型而从异常中恢复。

用行类型写的值可能被读作列类型。

表 3-7 用于 StreamMessage 和 MapMessage 的转换

	boolean	byte	short	char	int	long	float	double	String	Byte[]
boolean	×								×	
byte		×	×		×	×			×	
short			×		×	×			×	
char				×					×	
int					×	×			×	
long						×			×	
float							×	×	×	
double								×	×	
String	×	×	×		×	×	×	×	×	
byte[]										×

3.11.4 用于非 JMS 客户端的消息

很多企业消息系统支持自定义的 Stream 和/或 map 本地消息类型。尽管客户端可以使用 BytesMessage 来构造这种形式的本地消息，但 JMS 为 StreamMessage 和 MapMessage 类型提供了更加方便的 API。

例如，当客户端使用支持本地消息的 JMS 提供商，且它希望发送既能被 JMS 客户端又能被本地客户端读取的 map 消息时，它是由 MapMessage。当消息被发送时，提供商将它转换成本地格式。本地客户端然后就可以接收它。如果 JMS 提供商接收它，那么提供商将它

火龙果•整理
uml.org.cn

转换回 MapMessage。

即使实现使用新定义的消息的新 JMS 应用时，应用可以选择使用 StreamMessage 和 MapMessage 来保证非 JMS 客户端能够读取消息。

如果 JMS 客户端发送 StreamMessage 或 MapMessage，那么它必须被接收 JMS 提供商转换成一个等价的 StreamMessage 或 MapMessage。当在 JMS 客户端间传递时，消息必须保持它全部的格式。例如，作为 MapMessage 发生的消息不能作为 BytesMessage 消息被接收。

如果 JMS 提供商接收一个由本地客户端创建的消息，那么提供商应当将它尽力转换成“最佳的”JMS 消息类型。例如，如果是本地流消息，那么它应当被转换成 StreamMessage。如果转换不了，那么提供商总能够将它转换成 BytesMessage。

3.12 JMS Message 接口的提供商实现

JMS 提供了一系列消息接口，这些接口定义了 JMS 消息模型。它没有提供这些接口的实现。

每个 JMS 提供商提供它自己的会话的消息创建方法的实现。这可以让提供商来使用满足它需要的消息实现。

提供商必须能够从客户端接收不是它自己的消息实现。提供商可以不高效的处理外来消息实现，但它必须处理。

当提供商处理外来消息实现时需要注意下面的例外情况。如果外来消息实现包含 JMSReplyTo 头字段，这个字段设置了外来地址实现，那么不要求提供商处理或保存这个头字段的值。

JMS 消息接口提供了写/set 方法来设置消息体内和消息属性内的对象的值。所有的这些方法必须复制它们输入的对象，然后放进消息中。输入对象的值可以是 null，当获取时返回 null。一个例外是 BytesMessage 不支持空流的概念，因此企图向他写 null 则必须抛出 java.lang.NullPointerException。

JMS 消息接口提供读/get 方法来访问消息体和消息属性内的对象。所有的这些方法必须返回要获取对象的拷贝。

4 JMS 公共工具

4.1 概述

本章描述了在 PTP 和 Pub/Sub 域间共享的 JMS 工具。

4.2 受管理的对象

JMS 受管理的对象是那些包含 JMS 配置信息的对象，这些配置信息由 JMS 管理员创建然后由客户端使用。它们实际管理企业中的 JMS 应用。

尽管受管理对象的接口不显式的依赖 JNDI，但 JMS 为了 JMS 客户端通过 JNDI 查找的方便而设置了这个方便性。

管理员可以将受管理对象放到命名空间的任何位置。JMS 没有定义命名策略。

火龙果•整理
uml.org.cn

区分 JMS 和管理提供了几个好处：

- 它对客户端隐藏了提供商特有的配置信息。
- 他将 JMS 可管理的信息抽象成容易被通用管理控制台组织和管理的 java 对象。
- 由于有 JNDI 提供商提供所有流行的命名服务，那么这意味着 JMS 提供商可以转发一个受管理对象的实现，这些对象可以运行在任何地方。

受管理对象不能持有任何远程资源。它的 lookup 不应当使用远程资源，除了有 JNDI 自己使用的资源以外。

客户端应当把受管理的对象看作是本地 java 对象。查找它们不应有任何隐藏边际影响或使用大量的本地资源。

JMS 定义了两个手管理对象，Destination 和 ConnectionFactory。

期望 JMS 提供商为管理员提供创建和在 JNDI 命名空间中配置受管理对象的工具。JMS 提供商提供的手管理对象的实现应当是既 javax.naming.Referencealbe 又 java.io.Serializable 的，以便它们能够被存储在所有的 JNDI 命名上下文中。另外，建议这些实现遵循 JavaBean 的设计模式。

4.2.1 Destination

JMS 没有定义标准的地址语法。尽管考虑到了，但在现存的地址语义有很大差别的企业消息产品间寻求一个单一的语法是不可能的。因此，JMS 定义了 Destination 对象来封装提供商特有的地址作为替代方案。

由于 Destination 是一个受管理对象，因此它可以包含除了地址外的提供商特有的配置信息。

JMS 也支持客户端使用提供商特有的地址名称。参加节 4.4.4“创建 Destination 对象”了解更详细的信息。

Destination 支持并发使用。

4.2.2 ConnectionFactory

ConnectionFactory 封装了一系列连接配置参数，这些参数由管理员定义。客户端使用它来创建和 JMS 提供商的连接。

ConnectionFactory 对象支持并发使用。

4.3 Connection

JMS Connection 是一个客户端到 JMS 提供商间的活动连接。它通常分配在 Java 虚拟机外部的提供商的资源。

Connection 对象支持并发使用。

Connection 用作几个目的：

- 它封装了一个与 JMS 提供商的连接。他通常代表一个在提供商服务域和客户端间的打开的 TCP/IP Socket。
- 在客户端授权时创建它。
- 它可以指定一个唯一的客户端标识。

火龙果•整理
uml.org.cn

- 它创建 Session 对象。
- 它提供 ConnectionMetaData。
- 它支持可选的 ExceptionListener。

由于在创建 Connection 时设置授权和通讯，因此 Connection 相对是重量级的 JMS 对象。大多数客户端使用一个连接来执行所有的消息动作。其他更先进的应用可以使用多个 Connection。JMS 使用多个连接不是架构方面的原因（除了当客户端作为两个不同提供商间的网关时）；但是可能有操作方面的原因。

4.3.1 授权

当创建连接时，客户端可以指定它的信任状如用户名/密码。

如果没有指定信任状，那么使用当前线程的信任状。这一点上，JDK 没有定义线程缺省信任状的概念。但是在不久的将来可能会定义。目前，可以使用 JMS 客户端运行使用的用户标识。

4.3.2 客户端标识

设置客户端标识的最好方式是将它配置在一个客户端特有的 ConnectionFactory 中，然后透明的将它赋给它创建的连接。另一个方法就是客户端可以用提供商特有的值来设置连接的客户端标识。显式地设置连接的客户端标识的工具不是一个用于覆盖已经被配置的标识的机制。它只是用于管理员没有配置标识的情况。如果标识已经存在，那么企图更改它必须抛出 IllegalStateException。

如果客户端显式地设置标识，那么它必须在创建连接后和在连接上发生其他动作之前立即设置。如果在这个时刻后再设置，则设置客户端标识就是程序性错误，将抛出 IllegalStateException。

客户端标识的目的是将连接和它的对象与由提供商按客户端维护的状态关联起来。按照定义，被客户端标识标识的客户端状态在一个时刻只能由一个客户端“在用”。JMS 提供商必须防止并发执行的客户端来使用它。

阻止方式可以是当企图这样使用时抛出各种 JMSException 的形式；也可以阻塞后续的客户端；或者是其他的解决方案。JMS 提供商必须保证这种共享单个客户端状态的企图不能引起消息的丢失或重复处理。

由 JMS 标识的唯一的单个客户端状态要支持长期订阅。

4.3.3 Connection 设置

JMS 客户端通常创建一个 Connection，一个或多个 Session 和大量的 MessageProducer 和 MessageConsumer。当 Connection 被创建时，它处于 stopped 模式。意思是没有消息可以转发给它。

通常会让昂 Connection 处于停止模式直到设置完成。设置完成后调用 Connection.start() 方法，且消息开始到达 Connection 的消费者。这种设置约定减小了客户端还在设置它自己时就有异步消息转发的问题。

Connection 可以立即启动然后再进行设置。这样，客户端必须做好它们在设置过程中处

火龙果•整理
uml.org.cn

理异步消息转发的准备。

MessageProducer 可以在 Connection 是停止时发送消息。

要注意，客户端必须在连接启动后它才能转发消息。JMS 提供商必须保证这种情况。

4.3.4 中止消息的转发

连接对接收到的消息的转发可以使用 stop()方法被临时停止。可以使用 start()方法重新启动它。当连接被停止时，禁止向所有连接的 MessageConsumer 进行转发：同步接收被阻塞，消息不被转发到 MessageListener。

停止连接对发送消息没有影响。停止一个已经停止的连接和启动一个已经启动的连接被忽略。

在消息的转发被停止后才能返回 stop 方法。这意味着停止后，不会调用任何消息监听器，并且所有的等待接收消息的控制线程在连接重新启动前不会返回消息。用于已停止连接的的接收计时器仍然有效，因此接收可能会超时并返回一个空消息。

如果 stop 方法被调用时消息监听器(MessageListener)仍在运行，则必须在所有的监听器都返回后才能停止。当这些消息监听器将要完成时，他们必须拥有连接的所有服务。

4.3.5 关闭 Connection

由于提供商通常为 Connection 分配 JVM 外部的资源，因此客户端应当在它们不需要的时候关闭这些资源。依赖垃圾回收来回收这些资源可能不是非常及时。

关闭操作将终止所有连接会话消费者接收的未处理消息。接收可能返回一个消息也可能返回 null，这要根据关闭时刻是否可以获得一个消息。

注意，在这种情况下，如果消息消费者企图用这个关闭的连接来处理最后的消息，它很可能会得到一个异常。开发者在写消息消费者的时候必须考虑这种“最后的消息”。必须再次强调，消息消费者不能依靠空返回值来判断“最后的消息”。

如果一个或多个消息监听器在连接关闭被调用时正在处理消息，那么在控制没有返回给 JMS 提供商之前就必须保持连接和会话的所有功能都可以被那些监听器获得。

当连接被关闭时，直到消息处理按顺序都被停止后关闭方法才能返回。这意味着所有正在运行的消息监听器已经返回且所有未处理的接收已经返回。

如果连接被关闭，不需要关闭它的组成对象。连接关闭足以通知 JMS 提供商应当释放连接的所有资源。

关闭连接必须回滚处于处理的交易会话中的事务（注：术语“交易会话”指的是会话的提交和回滚方法用于分割会话的本地事务。在这种情况下会话的工作和外部的事务管理器的工作是一致的，此时不使用会话的提交和回滚方法，被关闭的会话的工作的结果随后由事务管理器决定。）。关闭连接不会强迫客户端确认会话的确认。调用来自已关闭的会话的已接收消息的 acknowledge 方法必须抛出 IllegalStateException。对于那些要求由 JMS 客户端连续可靠处理的队列和长期订阅来说，这些语义保证关闭连接不会造成消息的丢失。

一旦连接被关闭，企图使用它或它的会话或它们的消息消费者和生产者必须抛出 IllegalStateException（必须忽略对这些对象的 close 方法的调用）。但继续使用通过这个连接创建或接收的消息对象仍是有效的，除了已接收消息的 acknowledge 方法。

关闭一个已关闭的连接不必抛出异常。

火龙果•整理
uml.org.cn

4.3.6 会话(Session)

Connection 是 Session 的工厂，它底层使用对 JMS 提供商的连接来生产和消费消息。

4.3.7 ConnectionMetaData

Connection 提供 ConnectionMetaData 对象。这个对象提供了提供商支持的 JMS 的最新版本和提供商的产品名和版本。

它也提供了连接支持的 JMS 定义的属性名列表。

4.3.8 ExceptionListener

如果 JMS 提供商检测到与连接有关的问题，那么它将通知连接的 ExceptionListener，如果它已经被注册。为了取出 ExceptionListener，JMS 提供商调用连接的 getExceptionListener()方法。这个方法返回这个连接的 ExceptionListener。如果没有注册 ExceptionListener，则返回 null。连接然后可以通过调用监听器的 onException()方法来使用监听器，传入描述问题的 JMSException。

这可以让客户端异步通知问题。某些连接只消费消息，因此它们没有其他途径来获知它们的连接已经失败。

连接序列化 ExceptionListener 的执行。

JMS 提供商应当在通知客户端之前尽力解决连接问题。

转发到 ExceptionListener 的异常是那些没有其他地方可报告的异常。如果在 JMS 调用连接时抛出异常，按照定义不会将这个异常转发给 ExceptionListener（换句话说，ExceptionListener 不是用于监听由连接抛出的所有异常）。

4.4 Session

JMS Session 是一个单线程的上下文（对能够使用会话对象或它创建的线程的数量没有限制。限制是会话的资源不应当被多个线程并发使用。这由用户来保证满足这个限制。最简单的方式是使用一个线程。在异步转发的情况下，使用一个线程来设置停止模式然后启动异步转发。在更复杂的情况下，用户必须提供显式的同步。），用于生产和消费消息。尽管会话可以给提供商分配 JVM 外部的资源，但是它被看作是轻量 JMS 对象。

Seesion 用于几个目的：

- 它是 MessageProducer 和 MessageConsumer 的工厂。
- 它是 TemporaryTopic 和 TemporaryQueue 的工厂。
- 它为需要动态操纵提供商专有目的地名字的客户端提供了一种创建 Queue 或 Topic 对象的途径。
- 它提供了提供商优化后的消息工厂。
- 它支持事务串，这些事务将跨会话生产者和消费者的工作组合成原子单元。
- 它为它消费的消息和它生产的消息定义了一个连续的顺序。
- 它保留它消费的消息直到这些消息被确认。

火龙果•整理
uml.org.cn

- 它序列化注册到它的 MessageListener 的执行。
- 它是 QueueBrowser 的工厂。

4.4.1 关闭会话

由于提供商可以代表会话分配位于 JVM 之外的资源，客户端应当在这些资源不需要的时候关闭它们。依靠垃圾回收来最终回收它们不是非常及时的。对于由会话创建的 MessageProducer 和 MessageConsumer 来说同样需要这样做。

关闭会话会中止这个会话上的所有消息处理。它必须处理由会话消费者未处理的接收或正在运行的消息监听器的关闭，正如在节 4.3.5“关闭连接”中所述。

关闭会话的方法可以从一个不同于当前控制这个会话的控制线程调用。

当会话关闭被调用时，在消息处理被按顺序停止前关闭方法不能返回。这意味着不会有消息监听器正在运行，且如果有未处理的接收，则它返回 null 或者一个消息。

当会话被关闭时，不需要关闭它的消息生产者和消费者。会话关闭足够通知 JMS 提供商释放会话的所有资源。

关闭有事务的会话必须回滚处理中的事务。关闭客户端确认的会话不强迫确认。

一旦会话被关闭，那么企图使用它或它的消费者和生产者必须跑出 IllegalStateException（必须忽略对这些对象 close 方法的调用）。继续使用通过这个会话创建或接收的消息对象是有效的，除了已接收消息的 acknoledge 方法。

关闭一个已关闭的会话不能抛出异常。

4.4.2 创建 MessageProducer 和 MessageConsumer

会话可以创建和服务于多个 MessageProducer 和 MessageConsumer。参见节 4.5“MessageConsumer”和节 4.6“MessageProducer”了解创建和使用它们的详细信息。

尽管会话可以创建多个生产者和消费者，它们必须依次使用。在效果上，只有一个逻辑控制线程可以使用它们。这在后面会详细的解释。

4.4.3 创建临时目的地

尽管会话被用于创建临时目的地，这只是为了方便。它们的范围实际上是整个连接。它们的生命周期与连接相同，连接的所有会话都可以创建临时目的地的 MessageConsumer。

临时目的地（TemporaryQueue 或 TemporaryTopic 对象）是系统为连接生成的唯一的目的地。只有它们自己的连接才可以为它们创建 MessageConsumer。

临时目的地的通常用法师作为 JMSReplyTo 的目的地。

每个 TemporaryQueue 或 TemporaryTopic 对象都是唯一的。不能够被复制。

由于临时目的地可以分配 JVM 外部的资源，如果这些资源不再使用，应当及时删除它们。当它们被垃圾回收或连接关闭时，这些资源将被自动删除。

4.4.4 创建目的地对象

大多数的客户端都将使用 Destination，Destination 是 JMS 的受管理对象，通过 JNDI 查

火龙果•整理
uml.org.cn

找它们。这是最方便的方法。

某些特殊的客户端可能需要动态操纵使用提供商特有的目的地名来创建 Destination。会话为 JMS 提供商提供了实现这个功能的提供商专用的方法。

4.4.5 优化消息实现

会话提供使用提供商优化过的实现来创建消息的方法。这可以让提供商最小化处理消息的负荷。

会话必须能够发生所有的 JMS 消息而不管它们如何被实现的。

4.4.6 使用会话的约定

会话被设计成一个时间只能由一个线程使用。一个例外情况是在会话或它的连接依次关闭时可以多个线程使用。参见节 4.3.5“关闭连接”和节 4.4.1“关闭会话”了解更详细的信息。

一个典型的用法是让一个线程阻塞在同步的 MessageConsumer 上，直到有消息到达。这个线程然后使用一个或多个 MessageProducer。

如果已经有一个客户端控制线程正在这个会话中等待接收消息，那么另一个客户端控制线程不能同步接收消息。

另一个典型的用法是让一个线程通过创建会话的生产者和一到多个异步消费者来设置会话。在这种情况下，消息生产者不能独占消费者消息监听器。由于会话按顺序执行消费者的 MessageListener，因此这些监听器可以安全地共享会话的资源。

如果在会话正在被设置时连接处于停止模式，那么客户端在完全准备好处理消息之前不需要处理到达的消息。这是最好的策略，因为它降低了设置和消息处理间出现冲突的可能性。当连接正在接收消息时可以创建和设置会话。在这种情况下，要注意保证会话的 MessageProducer、MessageConsumer 和 MessageListener 要按正确的顺序创建。例如，错误的顺序可能引起 MessageListener 使用还没有创建号的 MessageProducer；或由于 MessageListener 注册的顺序不正确而使得消息按错误的顺序到达。

如果客户端希望在其他线程消费消息时有一个线程产生消息，那么客户端应当为生产消息的线程使用独立的会话。

一旦连接被启动，则它的所有带有消息监听器的会话都致力于给它们转发消息的控制线程。从另外一个控制线程中使用这些会话的客户端代码是错误的。唯一的例外是这种非那个是可以用于会话或连接的关闭方法。

对会话增加单线程控制的限制的结果是带消息监听器的会话也不能同步接收消息。会话要么致力于用作转发消息到消息监听器的控制线程，要么致力于由客户端初始化的控制线程。不能在同一个会话中同时使用。

另一个结果是，连接必须在停止模式下对带多个消息监听器的会话进行设置。原因是当连接正在转发消息，一旦第一个消息监听器被注册，则会话就由向这个监听器转发消息的控制线程控制。此时，客户端的控制线程不能进一步配置这个会话。

对大多数客户端来说，将它们的工作分派到多个会话是很正常的。这个模型可以让客户端简单的启动和随着并发增长的需要而逐渐地增加消息处理。

火龙果•整理
uml.org.cn

4.4.7 事务

Session 可选地可以被指定为事务性的。每个事务性的会话支持单序列的事务。每个事务将一系列生成的消息和一系列消费的消息分组到一个原子工作单元。效果是，事务将会话的输入消息流和输出消息流组织称一系列的原子单元。当事务提交时，输入原子单元被确认，输出原子单元被发送。如果事务回滚，则它生产的消息被销毁，它消费的消息被自动恢复。关于会话恢复的详细信息，参见节 4.4.11“消息确认”。

会话使用它的 commit()或 rollback()方法来完成会话。当前事务完成时会自动启动下一个事务。因此，事务性的会话当前总会有一个事务。

JTS 或一些其他的事务监控器工具可以被用于将会话的事务和其它资源（数据库，其他的 JMS 会话等等）上的事务组合在一起。由于 Java 分布式事务由 JTA 事务分割 API 控制，因此在这种上下文中使用会话的 commit 和 rollback 方法将抛出 JMS 的 TransactionInProgressException。

4.4.8 分布式事务

JMS 不要求提供商支持分布式事务；但是如果提供商支持了，则应当通过 JTA XAResource API 来提供分布式事务。

JMS 提供商也可以是分布式事务监听器。如果它是，那么它应当通过 JTA API 提供事务控制。

尽管对 JMS 客户端来说它可以直接处理分布式事务，但 JMS 客户端尽量不要这样做。使用基于 XA 接口的 JMS 客户端在第 8 章“JMS 应用服务器工具”中描述，但这种客户端不能跨不同的 JMS 实现进行移植，因为这些接口是可选的。在 JMS 中支持 JTA 是为了系统提供商能够将 JMS 集成到他们的应用服务器产品中。参见第 8 章“JMS 应用服务器工具”了解更详细的信息。

4.4.9 多会话

一个客户端可以创建多个会话。每个会话都是一个独立的消息生产者和消费者。

对于 Pub/Sub，如果两个回合都有一个 TopicSubscriber，且都订阅同一个 Topic，那么每个订阅者都会收到消息。它们之间不会相互阻塞。

对于 PTP，JMS 没有指定对同一个 Queue 并发 QueueReceiver 的语义。但是，JMS 没有禁止提供商提供这个功能。因此，向多个 QueueReceiver 转发消息将依赖 JMS 提供商的实现。但这是不可移植的。

4.4.10消息排序

JMS 客户端不需要理解它们什么时候可以依靠消息排序，什么时候不能。

火龙果•整理
uml.org.cn

4.4.10.1 消息接收的顺序

由会话消费的消息定义了一系列的顺序。这个顺序是重要的，因为它定义了消息确认的顺序。参见 4.4.11“消息确认”了解详细信息。每个会话消费者交叉读取会话输入消息流中的消息。

JMS 定义了由会话发送到目的地的消息必须按它们发送的顺序被接收（参见节 4.4.10.2“消息发送的顺序”了解限制条件）。这节定义了一部分在会话输入消息流上排序的约束。

JMS 没有定义跨目的地接收消息的顺序或从多个会话发送的跨目的地的消息的顺序。会话输入消息流的顺序依赖于时间。不在应用的控制之下。

4.4.10.2 消息发送的顺序

尽管客户端松散地查看在会话中生产的消息，这些消息形成了一个有序的发送消息流，对流进行整体排序是没有意义的。对接收客户端唯一可见的排序是会话发送到一个特定目的地的消息的顺序。有几个事情可以影响整个顺序：

- 高优先级的消息可以跳到低优先级消息的前面。
- 客户端可以不接收 NON_PERSISTENT 的消息，因为 JMS 提供商失败造成。
- 如果 PERSISTENT 和 NON_PERSISTENT 消息被发送到一个目的地，那么只在转发模式中保证顺序。也就是说，稍晚的 NON_PERSISTENT 消息可以在稍早的 PERSISTENT 之间到达；但是不会在稍早的同优先级的 NON_PERSISTENT 消息之前到达。
- 客户端使用事务性的会话将会话发送的消息分组到原子单元（一个 JMS 事务的生产者组件）。发送到特定目的地的消息的事务顺序是有意义的。跨目的地发送的消息的顺序是没有意义的。参见 4.4.7“事务”了解更详细的信息。

4.4.11 消息确认

如果会话是事务性的，那么消息确认自动由 commit 处理，且恢复自动由 rollback 处理。

如果会话不是事务性的，有三个确认选择，且手工处理恢复：

- DUPS_OK_ACKNOWLEDGE——这个选项告诉会话懒惰确认消息的传递。如果 JMS 失败，这很可能造成传递重复消息，因此这个选项只用于可以忍受重复消息的消费者。它的好处是减少了会话为防止重复所要做的工作。
- AUTO_ ACKNOWLEDGE——使用这个选项，当消息被成功地从调用接收返回或处理消息的 MessageListener 成功返回时，会话自动确认客户端的消息接收。
- CLIENT_ ACKNOWLEDGE——使用这个选项，客户端通过调用消息的 acknowledge 方法来确认消息。确认一个被消费的消息会自动确认被该会话转发的所有消息。

当使用 CLIENT_ ACKNOWLEDGE 模式时，客户端可以在处理它们时产生大量未确认消息。JMS 提供商应当为管理员提供限制客户端超量运行的途径，以便客户端不会造成资源耗尽并保证当它们使用的资源被临时阻塞时造成失败。

会话的 recover 方法用于停止一个会话然后使用第一个未确认消息来重新启动它。事实上，会话的被转发消息序列被重新设置到最后一个确认消息之后。现在转发的消息序列可以与起初转发的消息序列不同，因为消息到期和收到更高优先级的消息。

会话必须设置消息的 redelivered 标记，表示它是由于恢复而被重新转发的。

火龙果•整理
uml.org.cn

4.4.12消息的重复转发

JMS 提供商不能重复转发已确认消息。

当客户端使用 AUTO_ACKNOWLEDGE 模式时，它不会直接控制消息的确认。由于这种客户端不能确切知道某个消息是否已经被确认，因此它们必须做好再次收到最后消费的消息的准备。这可能由于客户端完成它的工作恰好在防止消息确认发生失败之前引起。只有会话测最后消费的消息会遇到这种情况。JMSRedelivered 消息头字段将用于这种情况下被重发的消息。

4.4.13消息的重复产生

JMS 提供商不能生产重复的消息。这意味着生产消息的客户端可以依赖 JMS 提供商来保证消息的消费者一个消息只会接收一次。客户端的错误不会引起提供商重复一个消息。

如果在客户端提交和提交方法返回期间出现错误，那么客户端不能决定事务是否被提交还是被回滚。当非事务性的发送一个 PERSISTENT 消息和发送方法返回之间产生错误时，会产生同样的不确定性问题。

这种不确定性由 JMS 应用来处理。在某些情况下，这可能会造成客户端生产重复消息。

由于恢复被重发的消息不认为是重复消息。

4.4.14客户端代码的有序执行

尽管 java 语言本身提供多线程，但写多线程程序仍然比写单线程程序困难。

因此，JMS 不会引起客户端代码的并非执行，除非客户端显式地要求这样做。做到这一点的一种途径是一个会话对所有消息的异步转发进行排序。

为了异步接收消息，客户端向 MessageConsumer 注册实现了 JMS MessageListener 接口的对象。事实上，会话使用一个单线程来运行所有的 MessageListener。当线程正在执行一个监听器时，所有其他被异步转发的消息必须等待。

4.4.15并行消息转发

希望并行转发的客户端可以使用多会话。事实上，每个会话的监听器线程并行的运行。当会话上的监听器正在执行时，在另一个会话上的监听器也可以被执行。

注意，JMS 本省不提供并行处理主题消息集合的功能（这些消息被转发到单个消费者）。客户端可以使用单个消费者但实现多线程逻辑来并行处理这些消息；但是，这样做是不可靠的，因为 JMS 没有事务功能来处理这种方式需要的并发事务。

4.5 MessageConsumer

客户端使用 MessageConsumer 来接收来自目的地的消息。通过向 Session 的 createConsumer 方法传入 Queue 或 Topic 来创建 MessageConsumer。

消费者可以用消息选择器来创建。这可以让客户端限制转发到消费者的消息，只有符合

火龙果•整理
uml.org.cn

选择器的消息才能被转发到该消费者。参见节 3.8.1“消息选择器”了解更详细的信息。

客户端可以同步接收消费者的消息，也可以让提供商在消息到达时异步地转发消息。

4.5.1 同步转发

客户端可以要求来自 MessageConsumer 的下一个消息使用某个 receiver 方法。有几个接收变量可以让客户端获取或等待下一个消息。

4.5.2 异步转发

客户端可以向 MessageConsumer 注册一个实现了 JMS MessageListener 接口的对象。当消息到达消费者时，提供商通过调用监听器的 onMessage 方法来转发它们。

监听器可能会抛出 RuntimeException；但是这被看作是客户端程序错误。好的监听器应当捕获这种异常并尽量将产生异常的消息转发到应用的‘未处理消息’目的地。

监听器抛出 RuntimeException 的结果依赖于会话的确认模式。

- AUTO_ACKNOWLEDGE 或 DUPS_ACKNOWLEDGE——消息被立即重发。JMS 提供商在放弃之前重发同一个消息次数由提供商决定。在这种情况下，将为重发的消息设置 JMSRedelivered 消息头字段。
- CLIENT_ACKNOWLEDGE——为监听器转发下一个消息。如果客户端希望让前一个未确认的消息被重发，那么它必须手工恢复会话。
- 事务性的会话——为监听器转发下一个消息。客户端可以提交或回滚这个会话（换句话说，RuntimeException 不自动回滚这个会话）。

JMS 提供商应当对抛出 RuntimeException 作为可能故障的具有消息监听器的客户端进行标记。

参见节 4.4.14“客户端代码的顺序执行”了解 onMessage 如何被会话有序的调用。

4.6 MessageProducer

客户端使用 MessageProducer 来向 Destination 发送消息。通过向会话的 createProducer 方法传入 Queue 或 Topic 来创建 MessageProducer。

客户端也可以不提供目的地来创建消息生产者。在这种情况下，必须在每次发送操作时提供目的地。这种风格的生产者的通常用于使用请求的 JMSReplyTo 目的地来发送请求的回复。

客户端可以指定一个缺省的转发模式、优先级和消息的生存时间。它也可以为每个消息指定转发模式、优先级和消息的生存时间。

客户端每次创建一个 MessageProducer，它定义了一个新的消息序列，这些消息和以前发送的消息没有顺序关系。

参见节 3.4.9“JMSExpiration”进一步了解生存时间。参见节 3.4.10“JMSPriority”进一步了解优先级。

火龙果•整理
uml.org.cn

4.7 消息转发模式

JMS 支持两种消息转发模式。

- NON_PERSISTENT 模式是最小符合的转发模式。因为它不要求将消息记录到稳定存储器中。JMS 提供商失败可能导致 NON_PERSISTENT 消息丢失。
- PERSISTENT 模式告诉 JMS 提供商要保证在转发期间消息不能由于 JMS 提供商失败而造成消息丢失。

JMS 提供商必须“最多一次的”转发 NON_PERSISTENT 消息。这意味着它可能丢失消息，但不会转发两次。

JMS 提供商必须“有且只有一次的”转发 PERSISTENT 消息。这意味着 JMS 提供商的失败不能引起消息的丢失，但不会转发两次。

PERSISTENT 和 NON_PERSISTENT 消息转发对 JMS 客户端来说是两种转发技术的选择，一种是在 JMS 提供商不工作时可以丢失消息，另一种是尽力保证消息在 JMS 提供商失败时还要存在。这种选择暗含了性能/可靠性的平衡。当客户端选择 NON_PERSISTENT 转发模式时，它表示它更看重性能而不是可靠性；选择 PERSISTENT 则相反。

使用 PERSISTENT 消息不保证所有的消息总是被转发到每个合格的消费者。参见节 4.10“可靠性”做进一步了解。

4.8 消息的生存时间

客户端可以为它发送的每个消息以毫秒为单位指定生存时间。它定义了消息的到期时间，到期时间是消息的生存时间和发送的 GMT 的和（对于事务性发生，这个时间是客户端发送消息的时间，不是事务提交的时间）。

JMS 提供商应当尽力做到精确的终止消息；但是，JMS 没有定义如何提供精确性。简单地忽略生存时间是不可接受的。

参见 3.4.9“JMSExpiration”了解消息到期的更详细信息。

4.9 异常

JMSException 是所有 JMS 异常的基类。参见第 7 章“JMS 异常”了解更详细的信息。

4.10 可靠性

大多数客户端应当使用生产 PERSISTENT 消息的生产者。这样可以保证有且只有一次的转发来自队列或永久订阅的消息。

在某些情况下，应用只可以要求最多一次的消息转发。通常在发布 NON_PERSISTENT 消息时使用。这些消息通常有较低的负荷；但是，这些消息可能在 JMS 提供商失败时被丢失。PERSISTENT 和 NON_PERSISTENT 消息都能被发布到同一个目的地。

通常，一个消费者在确认之前完整处理每个消息。这保证 JMS 不会因为机器出问题等而导致丢弃部分处理的消息。消费者通过使用事务的或 CLIENT_ACKNOWLEDGE 会话来达到。JMS 提供商必须设置由于系统失败而导致重发的未确认的消息的 JMSRedelivered 消息头字段。

火龙果•整理
uml.org.cn

如果 NON_PERSISTENT 消息被转发到永久订阅或一个队列，那么如果永久订阅变成不活动的（也就是，如果它当前没有订阅者）或 JMS 提供商关闭然后被重新启动，则转发不受保证。

重要的消息期望使用 PERSISTENT 转发模式在事务内生产，并且在来自非临时队列或永久订阅的事务内被消费。

当这些都被做时，应用就拥有了最高级别的保证：消息已经被正确的生产，可靠的转发和精确的消费。非事务的生产和消费也可以达到相同的保障级别；但是这要求认真的编码。

JMS 提供商可以限制高容量目的地能处理的消息数量或不响应的客户端数量。如果消息由于消息限制被丢弃，则这是一个需要注意的严重的管理问题。JMS 要求的正确功能是客户端是响应的且有足够的资源服务于这些客户端。

正如本规范描述的一样，有且只有一次的的消息转发有重要的一点，就是它不会覆盖由于消息到期或其他管理原因毁坏的消息。它也不覆盖由于资源限制丢失的消息。为 JMS 应用配置足够的资源和处理能力是管理员的工作，他必须知道 JMS 提供商的可靠性特性。

NON_PERSISTENT 消息，非永久性订阅，和临时目的地都是不可靠的。JMS 提供商关闭或失败时很可能造成 NON_PERSISTENT 消息的丢失和临时目的地以及非永久性订阅持有的消息的丢失。终止应用很可能造成由非永久性订阅和临时目的地持有的消息的丢失。

4.11 方法跨消息域继承

由于统一了消息域，因此某些不适用于一个域的方法可以在域类中被继承。例如，Session 接口有方法 createQueueBrowser。由于 TopicSession 继承了 Session 接口，因此 TopicSession 继承了 createQueueBrowser 方法，尽管这个方法不能被主题使用，也就是主题不支持 QueueBrowser。表 4-1 列出了这些实例。

如果应用企图调用列出的方法，则 JMS 提供商必须抛出 IllegalStateException。

表 4-1 必须抛出 IllegalStateException 的方法

接口	方法
QueueConnection	createDurableConnectionConsumer
QueueSession	createDurableSubscriber
	createTemporaryTopic
	createTopic
	unsubscribe
TopicSession	createQueueBrowser
	createQueue
	createTemporaryQueue

5 JMS 点对点模型

5.1 概述

点对点系统是与消息队列一起工作的。它们是点对点的是因为客户端将消息发送到一个队列。某些 PTP 系统通过给客户端提供自动分发消息功能模糊了 PTP 和 Pub/Sub 间的差别。

火龙果•整理
uml.org.cn

对客户端来讲，通常是将它们的消息转发到单个队列中。

和常见的邮件箱一样，队列可以包含混合消息。但类似于实际的邮件箱，创建和维护每个队列的成本都是很高的。大多数队列都由管理员创建，并被客户端看作是静态资源。

JMS PTP 模型定义了客户端如何和队列工作；如何找到队列，客户端如何将消息发送给它们，以及如何从队列中接收消息。

本章描述了 PTP 模型的语义。支持 PTP 模型的 JMS 提供商必须实现这里描述的语义。

不管 JMS 客户端程序是否使用 PTP 域的接口，还是在第 4 章“JMS 公共工具”中描述的公共接口，客户端程序必须被保证有相同的行为。

表 5-1 展示了 PTP 域特有的接口和 JMS 的公共接口。公共接口是创建 JMS 应用程序的最佳方式，因为它们是独立于域的。

表 5-1 PTP 域接口和 JMS 公共接口

PTP 域接口	JMS 最佳的公共接口
QueueConnectionFactory	ConnectionFactory
QueueConnection	Connection
Queue	Destination
QueueSession	Session
QueueSender	MessageProducer
QueueReceiver	MessageConsumer

5.2 队列管理

JMS 没有定义创建、管理或删除长生命队列的工具（没有为 TemporaryQueue 提供这样的机制）。由于大多数客户的使用静态定义的队列，因此这不是问题。

5.3 Queue

Queue 对象封装了提供商特有的队列名称。这个名称是客户端为 JMS 方法指定队列标识的途径。

JMS 没有定义消息被队列持有的真正时间长度和资源溢出的后果。

参见节 4.2“受管理的对象”了解关于 JMS Destination 对象的更多信息。

5.4 TemporaryQueue

TemporaryQueue 是唯一的 Queue 对象，它为 Connection 或 QueueConnection 的永久性而创建。它是系统定义的队列，只能被创建它的 Connection 或 QueueConnection 来消费。

参见节 4.4.3“创建临时目的地”了解更详细的信息。

5.5 QueueConnectionFactory

客户端使用 QueueConnectionFactory 来创建具有 JMS PTP 提供商的 QueueConnection。

参见节 4.2“受管理对象”了解关于 JMS QueueConnectionFactory 对象更多的信息。

火龙果•整理
uml.org.cn

5.6 QueueConnection

QueueConnection 是一个对 JMS PTP 提供商的活动连接。客户端使用 QueueConnection 来创建一到多个用于生产和消费消息的 QueueSession。

参见节 4.3“Connection”了解更详细的信息。

5.7 QueueSession

QueueSession 提供了创建 QueueReceiver、QueueSender、QueueBrowser 和 TemporaryQueue 的方法。

如果在 QueueSession 终止时还有已被接收但还没被确认的消息，那么这些消息必须被保留，且在消费者下一次访问队列时被重发。

参见节 4.4“Session”了解更详细的信息。

5.8 QueueReceiver

客户端使用 QueueReceiver 来接收已被转发到队列的消息。

尽管里两个会话可以有连接到同一个队列的 QueueReceiver，但 JMS 没有定义在 QueueReceiver 间如何分发消息。

如果 QueueReceiver 指定了消息选择器，那么没被选中的消息仍然在队列中。根据定义，消息选择器可以让 QueueReceiver 跳过消息。这意味着当跳过的消息最终被读时，总的读排序不保留由每个消息生产者定义的部分排序。只有没有消息选择器的 QueueReceiver 才按照消息生产者指定的顺序读消息。

参见 4.5“MessageConsumer”了解更详细的信息。如果 MessageConsumer 正在从 Queue 中消费消息，那么它的行为必须和节 5.8“QueueReceiver”中描述的一样。

客户端使用 MessageProducer 或 QueueSender 来向 Queue 发送消息。

参见节 4.6“MessageProducer”了解更详细的信息。

5.9 QueueBrowser

客户端使用 QueueBrowser 来查看队列中的消息，但不删除它们。QueueBrowser 可以从 Session 或 QueueSession 来创建。

浏览方法返回 java.util.Enumeration，它用于遍历队列中的消息。它可以是队列的全部内容，或它只包含批评消息选择器的消息。

当在浏览消息时，消息可以到达和到期。JMS 不要求枚举的内容是对列内容的静态快照。这些变化是否可见依赖于 JMS 提供商。

5.10 QueueRequestor

JMS 提供了一个 QueueRequestor 帮助类来简化服务请求。

QueueRequestor 构造器需要一个 QueueSession 和目的地队列。它为响应创建

火龙果•整理
uml.org.cn

TemporaryQueue，并提供了一个 request 方法来发送请求信息和等待回复。

5.11 可靠性

队列通常由管理员创建，并长时间存在。它总是持有发送到它的消息，不管消费消息的客户端是否是活动的。因此，客户端不必担心它会错过消息。

6 JMS 发布/订阅模型

6.1 概述

JMS Pub/Sub 模型定义了 JMS 客户端如何发布消息到基于内容层次的众所周知的节点和如何从节点订阅消息。JMS 将这些节点称为主题（topic）。

在这一节，术语 publish 和 subscribe 用于替代前面更常用的术语 produce 和 consume。

注意可以被看作是一个小的消息代理，它收集和分发定位到它的消息。通过依靠主题作为中介，消息发布者和订阅者保持独立。主题随着发布者和订阅者的变化而自动适配。

当代表它们的 Java 对象存在时，发布者和订阅者是活动的。JMS 也支持可选的永久订阅者，这些订阅者在不活动时也会被记住是存在的。

本章描述 Pub/Sub 模型的语义。支持 Pub/Sub 模型的 JMS 提供商必须支持这里描述的语义。

不管 JMS 客户端程序使用 Pub/Sub 域特有的接口还是使用在第 4 章“JMS 公共工具”中描述的公共接口，客户端程序必须被保证有相同的行为。

表 6-1 展示了 Pub/Sub 域特有的接口和 JMS 的公共接口。公共接口是创建 JMS 应用程序的最佳方式，因为它们是独立于域的。

表 6-1 Pub/Sub 域接口和 JMS 公共接口

Pub/Sub 域接口	JMS 最佳的公共接口
TopicConnectionFactory	ConnectionFactory
TopicConnection	Connection
Topic	Destination
TopicSession	Session
TopicPublisher	MessageProducer
TopicSubscriber	MessageConsumer

6.2 Pub/Sub 延时

由于在所有的 Pub/Sub 系统中通常都会有一些延时，订阅者实际看见的消息的时间根据 JMS 提供商产生新的订阅者和提供商在转发过程中保留消息的时间长度会有很大的不同。

例如，由于新的订阅者传播到整个系统会花费一些时间，因此来自远方发布者的消息可能会被错过。当创建新的订阅者时，它可以接收在创建之间发送的消息，因为提供商可能还没有让订阅者可以获取这些消息。

火龙果•整理
uml.org.cn

JMS 没有定义在 pub/sub 提供商调整到新的客户端的间隔期内的语义。JMS 语义只应用到达到“稳定状态”的提供商。

6.3 永久订阅

非永久订阅维持着订阅者对象的生命。这意味着客户端将只能看见在订阅者是活动时发布到主题的消息。如果订阅者是不活动的，那么它将错过发布到主题的消息。

订阅者可以以高负荷的成本来使用永久订阅。永久订阅者用一个唯一标识向 JMS 注册一个永久订阅。使用同一个标识的后续订阅者对象重新使用前一个订阅者使用过的订阅，这个订阅的状态是前一个订阅者留下的。如果永久订阅没有活动的订阅者，则 JMS 保留订阅的消息直到它们被订阅接收或到期。

所有的 JMS 提高商必须能够动态创建和删除永久订阅的 JMS 应用。另外，某些 JMS 提供商可以提供管理配置永久订阅的工具。如果永久订阅已经被配置，它可以默默覆盖由客户端指定的订阅。

不活动的永久订阅是一个存在但当前还没有消息消费者向它订阅的永久订阅。

6.4 主题（Topic）管理

某些产品要求主题是静态定义、要与权限控制相关联，等等；其他的甚至没有主题管理的概念。

JMS 没有定义创建、管理或删除主题的工具。

一种特殊类型的主题是 TemporaryTopic，它用于创建一个对于 TopicConnection 来说是唯一的 Topic。参见节 6.6“临时主题（TemporaryTopic）”了解详细信息。

6.5 Topic

Topic 对象封装了提供商特有的主题名。它用于在 JMS 方法中指定主题标识。

许多 Pub/Sub 提供商将主题组织成层级形式，并提供很多选项来订阅层级的部分主题。JMS 对 Topic 对象代表什么没有限制。它可能是主题层级的叶子，或者可能是层级的较大部分（用于订阅信息的大类）。

主题的组织和订阅授权都是 Pub/Sub 应用架构很重要的部分。JMS 没有指定如何实现它们的策略。如果应用使用了提供上特有的分组机制，那么它应当记录它。如果应用使用不同的提供商，那么管理员负责构造相等的主题架构和创建相等的 Topic 对象。

6.6 TemporaryTopic

TemporaryTopic 是一个唯一的 Topic 对象，它为 Connection 或 TopicConnection 永久化而创建。它是系统定义的 Topic，只能被创建它的 Connection 或 TopicConnection 消费。

根据定义，向一个临时主题创建永久订阅是没有意义的。如果这么做就是程序错误，它可能会也可能不会被 JMS 提供商检测到。

参见节 4.4.3“创建临时目的地”了解详细信息。

火龙果•整理
uml.org.cn

6.7 TopicConnectionFactory

客户端使用 TopicConnectionFactory 来创建带 JMS Pub/Sub 提供商的 TopicConnection。

参见节 4.2“受管理对象”了解 JMS ConnectionFactory 对象的更多信息。

6.8 TopicConnection

TopicConnection 是一个连接到 JMS Pub/Sub 提供商的活动连接。客户端使用 TopicConnection 来创建一到多个用于生产和消费消息的 TopicSession。

参见节 4.3“Connection”了解更多信息。

6.9 TopicSession

TopicSession 提供了创建 TopicPublisher、TopicSubscriber 和 TemporaryTopic 的方法。它也提供了用于删除客户端永久订阅的 unsubscribe 方法。

如果已经接收到消息但消息在 TopicSession 终止时还没有被确认，则永久 TopicSubscriber 必须保留和重发这些消息；非永久订阅不需要这么做。

参见节 4.4“Session”了解更多信息。

6.10 TopicPublisher

客户端使用 TopicPublisher 向一个 topic 发布消息。TopicPublisher 是 JMS MessageProducer 的 Pub/Sub 变量。可以使用 MessageProducer 想一个 Topic 发送消息。参见节 4.6“MessageProducer”了解公共特性的描述。

6.11 TopicSubscriber

客户端使用 TopicSubscriber 来接收已发布到主题的消息。TopicSubscriber 是 JMS MessageConsumer 的 Pub/Sub 变量。参见节 4.5“MessageConsumer”了解更多信息。

普通的 TopicSubscriber 不是永久性的。它们只接收它们是活动时被发布的消息。

被订阅者的消息选择器过滤掉的消息永远不会被订阅者接收到。从订阅者的角度来看，这些消息是不存在的。

在某些情况下，连接可以既是到主题的发布又是到主题的订阅。订阅者的 NoLocal 属性可以让订阅者抑制转发由它自己的连接发布的消息。

TopicSession 可以为每个目的地创建多个 TopicSubscriber，它将目的地的每个消息都转发到符合接收条件的 TopicSubscriber。这些都工作在消息的复制品上，而不互相影响；确认一个消息不会确认其他的消息；一个消息可能被立刻转发，但另一个可能等待消费者处理完它前面的消息。

JANTY

火龙果•整理
uml.org.cn

6.11.1 永久 TopicSubscriber

如果客户端需要接收发布到主题上的所有消息，包括订阅者是不活动时发布的消息，那么它使用永久 TopicSubscriber。永久 TopicSubscriber 可以由 Session 或 TopicSession 创建。JMS 保留永久订阅的记录并保证来自 Topic 的发布者的所有消息都被保留，直到这些消息被永久订阅者确认或它们到期。

使用永久订阅的会话必须总是提供同一个客户端标识。另外，每个客户端必须指定唯一标识（在客户端标识符中）它创建的每个永久订阅的名字。一次只能有一个会话可以有一个特殊永久订阅的 TopicSubscriber。参见节 4.3.2“客户端标识”了解更多信息。

客户端可以通过创建一个具有相同名字和新主题和/或消息选择器或 NoLocal 属性的永久 TopicSubscriber 来改变一个存在的永久订阅。改变一个永久订阅相当于删除并重新创建它。

Session 和 TopicSession 提供了 unsubsribe 用于删除它们客户端创建的永久订阅。这回删除由提供商代表订阅者维护的状态。对客户端来说，当永久订阅有活动的 TopicSubscriber 或者当被接收到的消息是当前事务的一部分或者还没有在会话中确认时，删除一个永久订阅是错误的。

6.12 恢复和重发

非永久订阅者的未确认消息应当能够在非永久订阅者生存时被恢复。当非永久订阅者终止时，等待它的消息很可能被丢弃而不管是否被确认。

只有永久订阅可以可靠地恢复未确认消息。

向转发模式是 PERSISTNET 的主题发送消息不会改变恢复和重发的模式。为了保证转发，TopicSubscriber 应当设置一个永久订阅。

6.13 管理订阅

理想情况下，发布者和订阅者都是在它们被创建时由提供商动态注册。从客户端视角来看，总是这样的。从管理员的角度看，可能需要其他任务来支持发布者和订阅者的创建。

为消息存储分配的资源的数量和资源过量的结果在 JMS 中没有定义。

6.14 TopicRequestor

JMS 提供了 TopicRequestor 帮助类来简化服务请求。

TopicRequestor 构造器需要传入 TopicSession 和目的地主题。它为响应创建了一个 TemporaryTopic，并提供了发送请求消息的 request()方法然后等待回复。

这是最基本的请求/响应抽象，它可以满足大多数的用途。JMS 提供者和客户端可以自由地创建多种专业版本。

6.15 可靠性

当主题的所有消息必须被接收时，应当使用永久订阅。JMS 保证在永久订阅者不活动时

火龙果•整理
uml.org.cn

保留发布的消息，并在订阅者活动后将消息转发给它。

非永久订阅者只应当用于可以接受丢失的情况。

表 6-2 Pub/Sub 可靠性

如何发布	非永久订阅	永久订阅
NON_PERSISTENT	最多一次（如果不活动则丢失）	最多一次
PERSISTENT	一次只有一个(如果不活动则丢失）	一次只有一个

7 JMS 异常

7.1 概述

本章对 JMS 异常处理做了概述并定义了标准的 JMS 异常。

7.2 JMSException

JMS 定义了 JMSException 作为 JMS 方法抛出的异常的根类。JMSException 是一个受检查异常，捕捉它就可以处理所有 JMS 相关的异常。JMSException 提供了下面的信息：

- 提供商特有的描述错误的字符串——这个字符串是标准的 java 异常消息，可用 getMessage()获得这个消息。
- 提供商特有的字符串型错误代码。
- 对另一个异常的引用——一个 JMS 异常通常是由底层问题引起。如果合适，底层异常可以被关联到 JMS 异常。

JMS 方法在它们的标识符中只包含 JMSException。JMS 方法可以抛出任意的标准 JMS 异常，以及 JMS 供应商特有的异常。**JMS 方法的 javadoc 只给出了必需的异常情况。**

7.3 标准异常

除了 JMSException 外，JMS 定义了几个其他异常，它们标准化了基本错误原因的报告。

只有几种情况必须抛出指定的 JMS 异常。这些情况在异常描述中用单词“必须”标出。**这些情况都是唯一的，在这些情况下客户端逻辑应当根据特定的问题抛出特定的 JMS 异常。**

在其他情况下，强烈建议 JMS 提供商尽可能使用标准的异常。JMS 提供商如果需要也可以从这些异常中提取供应商特有的异常。

JMS 定义了下面的标准异常：

- IllegalStateException：当在不合法或不合适的时间，或提供商没有处于处理请求操作的合适状态，则抛出这个异常。例如，如果 Session.commit()在非事务会话中被调用，则必须抛出这个异常。当调用与域不匹配的方法时，例如调用 TopicSession.CreateQueueBrowser()，必须抛出这个异常。
- JMSSecurityException：当提供商拒绝客户端提交的用户名/密码时必须抛出这个异

火龙果•整理
uml.org.cn

常。它也可以用于安全限制阻止方法完成的情况。

- InvalidClientException：当客户端企图设置连接的客户端标识，但提供商拒绝了这个标识的值时，就必须抛出这个异常。
- InvalidDestinationException：当提供商不能识别给定的目的地或目的地不再有效时，必须抛出这个异常。
- InvalidSelectorException：当 JMS 客户端企图用无效的语法向提供商提供消息选择器时，必须抛出这个异常。
- MessageEOFException：当正在读取 StreamMessage 或 ByteMessage 时到达了不期望的流结尾，则必须抛出这个异常。
- MessageRormatException：当 JMS 客户端企图使用消息不支持的数据类型或企图将消息的数据读作错误的类型时，必须抛出这个异常。当消息属性值发生同样的类型错误时也必须抛出这个异常。例如，如果 StreamMessage.writeObject()被给了不支持的类或如果 StreamMessage.readShort()用于读取布尔值，则必须抛出这个异常。如果提供商被给了不能接受的类型，则也必须抛出这个异常。注意，一个特殊的情况是当企图读取不正确格式的 String 数据作为数值型值时，必须抛出 java.lang.NumberFormatException。
- MessageNotReadableException：当 JMS 客户端企图读一个只可写的消息时必须抛出这个异常。
- MessageNotWriteableException：当 JMS 客户端企图写一个只可读的消息时必须抛出这个异常。
- ResourceAllocationException：当提供商不能定位方法请求的资源时必须抛出这个异常。例如，当由于 JMS 提供商资源缺失而导致调用 createTopicConnection 失败时应当抛出这个异常。
- TransactionProgressException：当由于事务处于处理中而导致操作无效时抛出这个异常。例如，当会话时分布式事务的一部分时调用 Session.commit()应当抛出这个异常。
- TransactionRolledBackException：当调用 Session.commit 导致当前事务回滚时必须抛出这个异常。

8 JMS 应用服务器工具

8.1 概述

本章描述用于并发处理订阅消息的 JMS 工具。也定义了 JMS 提供商如何支持 JTS 可感知的会话。这些工具主要由 JMS 提供商使用。

如果 JMS 客户端使用 JTS 可感知工具来进行客户端编程，则它可能是不可移植的，因为不要求 JMS 提供商支持这些接口。

在本章中描述的工具是 JMS 的一个特殊类别。它们是可选的，可能只有部分 JMS 提供商对它们提供支持。

火龙果•整理
uml.org.cn

8.2 并发处理订阅的消息

JMS 提供了一个特殊的工具用于创建 MessageConsumer，它可以并发的消费消息。这个工具将这项工作分成三个角色：

- JMS 提供商——用于转发消息。
- 应用服务器——用于创建消费者和管理由并发 MessageListener 对象使用的线程。
- 应用——用目的地和可选的消息选择器定义一个订阅，并提供一个单线程的 MessageListener 类来消费它的消息。应用服务器将构造这个类的多个对象来并发消费消息。

8.2.1 Session

会话提供了应用服务器使用的三个方法：

- setMessageListener() 和 getMessageListener()——会话的 MessageListener 通过 ConnectionConsumer 消费分配到这个会话的消息，如后面的几个段落描述。
- run()——会话的 MessageListener 依次处理由 ConnectionConsumer 分配到会话的消息。当监听器从最后一个消息处理返回时，run()返回。

应用服务器通常被给一个 MessageListener 类，它包含了由应用程序员编写的处理消息的单线程代码。应用服务器也被给监听器要消费消息的目的地和消息选择器。

应用服务器将负责创建处理消息处理的 JMS Connection，ConnectionConsumer 和 Session。应用服务器将按需创建 MessageListener 实例并将它们注册到它自己的会话中。

由于许多监听器需要使用会话的服务，因此监听器很可能要求将它的会话传入到它的构造器中。

8.2.2 ServerSession

ServerSession 是一个由应用服务器实现的对象。应用服务器使用它来将一个线程和一个 JMS 会话关联起来。

ServerSession 实现了两个方法：

- getSession()——返回 ServerSession 的 JMS 会话。
- start()——启动 ServerSession 线程的执行，并执行关联的 JMS 会话的 run 方法。

8.2.3 ServerSessionPool

ServerSessionPool 是一个由应用服务器实现的对象，它提供了用于处理 ConnectionConsumer 的消息的 ServerSession 池。

它只有一个方法是 getServerSession()。这个方法从池中删除一个 ServerSession 并将它返回给调用者（假定是 ConnectionConsumer），用于消费一个或多个消息。

JMS 没有说明如何实现池。它可能是一个静态的 ServerSession 池，或者它可以使用更专业的算法来按需动态创建 ServerSession。

如果 ServerSessionPool 没有了 ServerSession，那么 getServerSession()方法可能阻塞。如

火龙果•整理
uml.org.cn

果 ConnectionConsumer 被阻塞，那么它不能转发新的消息直到 ServerSession 返回。

8.2.4 ConnectionConsumer

对于应用服务器来说，连接提供了创建 ConnectionConsumer 的专用工具。它消费的消息由目的地和消息选择器指定。另外，ConnectionConsumer 必须交给 ServerSessionPool 用于处理它的消息。指定 maxMessages 值用于限制 ConnectionConsumer 一次可以加载到 ServerSession 会话的消息数量。

通常，当拥堵轻微时，ConnectionConsumer 从池中得到一个 ServerSession，用一个消息加载它的会话，然后启动它。当拥堵严重时，消息可能被退回。如果发生退回，ConnectionConsumer 可以用多个消息来加载每个 Session。这减少了线程上下文的切换，减小了某些消息处理排序时资源的使用。

8.2.5 ConnectionConsumer 如何使用 ServerSession

由 JMS 提供商实现的 ConnectionConsumer 使用 ServerSession 来处理一到多个到达的消息。按以下方式做这项工作：

- 从 ServerSessionPool 中得到一个 ServerSession。
- 得到 ServerSession 的 Session。
- 用一个或多个消息加载 Session。
- 然后其他 ServerSession 来消费这些消息。

用于 QueueConnection 的 ConnectionConsumer 将它的消息加载到 QueueSession，同样，用于 TopicConnection 的 ConnectionConsumer 加载 TopicSession。

注意，JMS 没有说明 ConnectionConsumer 如何用消息加载 Session。由于 ConnectionConsumer 和 Session 都由同一个 JMS 提供商实现，因此它们可以用私有机制完成这种加载。

8.2.6 应用服务器如何实现 ServerSession

JMS 没有说明 ServerSession 的实现。下面出现的普通实现解释了这个概念：

- 应用服务器为 ServerSession 创建一个线程，注册 ServerSession 的 runObject。runObject 的实现是应用服务器私有的。
- ServerSession 的 start 方法调用线程的 start 方法。和所有的 java 线程一样，调用 start 初始化线程的执行并调用线程的 runObject。ServerSession.start(ConnectionConsumer) 的调用者和 ServerSession 的 runObject 现在运行在不同的线程中。
- runObject 将做一些内部处理然后调用会话的 run()方法。在返回时，runObject 将 ServerSession 返回给 ServerSessionPool 然后返回。这终止了 ServerSession 线程的执行，并再次循环启动。

8.2.7 结果

JMS 定义了灵活的机制将并发消息消费的工作拆分成适合于每个参与者的角色。

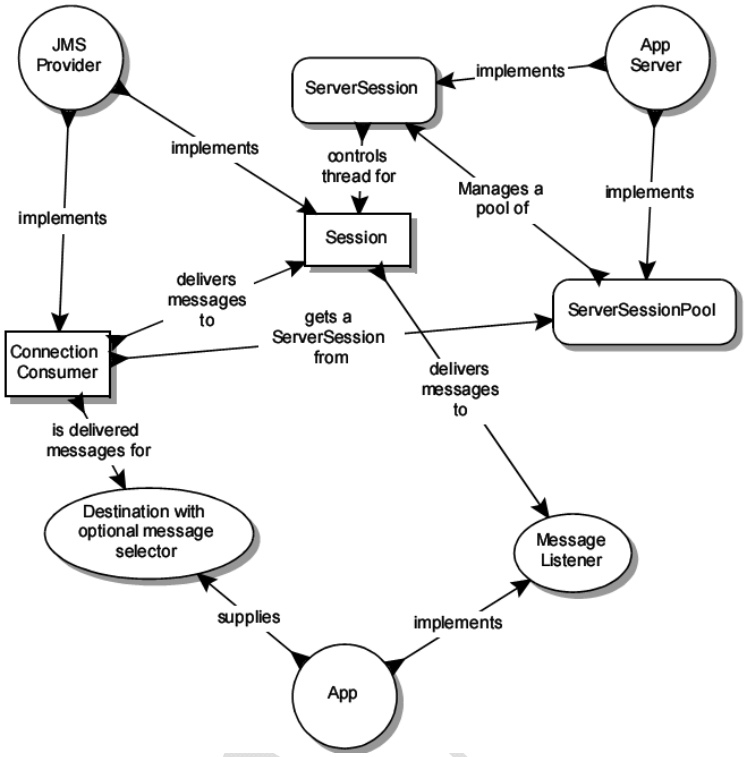
火龙果•整理
uml.org.cn

应用程序员提供一种易于书写单线程的 MessageListener 实现。

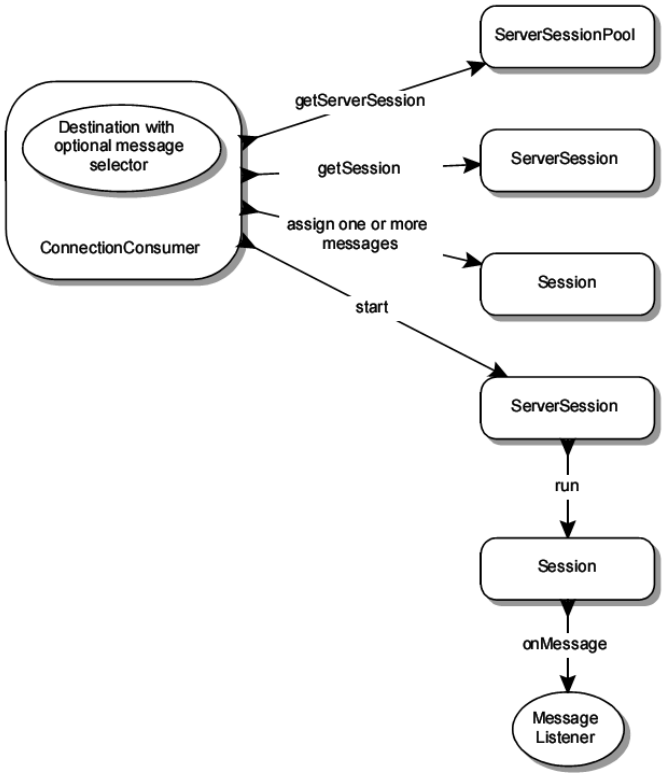
JMS 提供者保持对消息的控制直到它们被转发到 MessageListener。这保证它是在消息确认的直接控制之下。

应用服务器控制 ConnectionConsumer 的设置和管理用于执行 MessageListener 的线程。

下图解释了三个角色和它们实现的对象间的关系。



下图解释了 ConnectionConsumer 将消息转发给 MessageListener 的过程。



火龙果•整理
uml.org.cn

8.3 XAConnectionFactory

某些应用服务器提供对用于分布式事务的资源分组提供支持。为了在分布式事务中包含 JMS 事务，应用服务器要求 JTA 兼容 JMS 提供商。JMS 提供商用 JMS XAConnectionFactory 暴露它的 JTA 支持，应用服务器使用 XAConnectionFactory 来创建 XAConnection。

XAConnectionFactory 提供了和 ConnectionFactory 一样的授权选项。

XAConnectionFactory 对象是 JMS 受管理对象，就像 ConnectionFactory 对象。期望用应用服务器使用 JNDI 找到它们。

8.4 XAConnection

XAConnection 通过提供创建 XASession 扩展了 Connection 的能力。

8.5 XASession

XASession 提供了获取看起来象普通 Session 对象的对象和控制会话事务上下文的 javax.transaction.xa.XAResource 对象。XAResource 的功能非常类似于标准 X/Open XA 资源接口定义的功能。

应用服务器通过获取 XAResource 来控制 XASession 的事务分配。它使用 XAResource 来将会话分配给分布式事务，准备和提交事务上的工作等等。

XAResource 为多事务上的交叉工作、恢复处理中的事务列表等等提供了公平的专业工具。JTA 感知的 JMS 提供上必须全部实现这个功能。这可以通过使用支持 XA 的数据库服务来做到，或者 JMS 提供商可以选择从基础开始来实现这个功能。

将 XASession 的 Session 赋给应用服务器的客户端。之后，应用服务器控制后台 XASession 的事务管理。

但必须注意的是，分布式事务上下文不会随着消息流动；也就是说，接收消息的事务和和生产消息的事务不能是同一个。这是异步消息和同步处理间的基本差别。消息生产者和消费者使用两种方式来建立 JMS 提供商保证一次只有一个消息被转发的能力。

再次重申，在 Session 中生产和/或消费消息的行为都可以是事务性的。在不同会话间生产和消费一个特定消息的行为不能是事务性的。

8.6 JMS 应用服务器接口

PTP 和 Pub/Sub 两个域都提供了它们自己的 JTS 感知的 JMS 工具。

但是，应当优先使用公共接口。表 8-1 列出了 JMS 的公共接口。

表 8-1 域内可选接口的关系

JMS 公共接口	PTP 接口	Pub/Sub 接口
ServerSessionPool	没有域特定的接口	没有域特定的接口
ServerSession	没有域特定的接口	没有域特定的接口
ConnectionConsumer	没有域特定的接口	没有域特定的接口
XAConnectionFactory	XAQueueConnectionFactory	XATopicConnectionFactory

火龙果•整理
uml.org.cn

XAConnection	XAQueueConnection	XATopicConnection
XASession	XAQueueSession	XATopicSession

9 JMS 样例代码

本章给出了一些展示 JMS 客户端如何使用 JMS API 的代码样例。也解释了如何使用几个消息类型。这些例子使用了支持统一消息模型的方法：这些例子使用点对点或发布/订阅进行工作。用 JMS API 进行工作是推荐的方式。

在 JMS 规范的早期版本中，只支持消息域（点对点或发布/订阅）各自的接口，且客户端要么使用点对点要么使用发布/订阅来进行编程。现在，JMS 客户端可以使用 JMS 公共接口进行编程。

在例子中，客户端应用发送和接收股票报价信息。客户端应用从股票报价服务中接收消息。股票报价服务在这个例子中没有描述。

为了简化这个例子，没有对异常进行处理。

本章描述了为发送和接收消息而创建正确环境的步骤。

在描述了基本的功能后，本章描述如何执行一些其他的公共功能，例如使用消息选择器。

9.1 准备发送和接收消息

下面是建立连接然后准备发送和接收消息的基本步骤。

- 获得 ConnectionFactory 和 Destination。
- 创建 Connection 和 Session。
- 创建 MessageConsumer。
- 创建 MessageProducer。

9.1.1 获取 ConnectionFactory

消息生产者和消息消费者（发送者和接收者）都需要得到 ConnectionFactory，并使用它来建立 Connection 和 Session。

通常，管理员已经为 JMS 客户端创建和配置好了 ConnectionFactory。客户端程序通常使用 JNDI API 来查找 ConnectionFactory。

```
import javax.naming.*;
import javax.jms.*;
ConnectionFactory connectionFactory;
Context messaging = new InitialContext();
connectionFactory = (ConnectionFactory)
        messaging.lookup("ConnectionFactory");
```

火龙果•整理
uml.org.cn

9.1.2 获取 Destination

管理员已经创建和配置了一个名字为“StockQueue”的 Queue，它是股票报价消息被发送和接收的地方。同样，可以使用 JNDI API 来查找目的地。

```
Queue stockQueue;
stockQueue = (Queue)messaging.lookup("StockSource");
```

9.1.3 创建 Connection

在得到 ConnectionFactory 之后，客户端程序使用它来创建 Connection。

```
Connection connection;
connection = ConnectionFactory.createConnection();
```

9.1.4 创建 Session

在得到 Connection 之后，客户端程序使用它来创建 Session。这个 Session 用于创建 MessageProducer（发送消息）或 MessageConuser（接收消息）。

Connection.createSession 方法有两个参数：

- 一个表示会话是否是事务性的布尔。
- 消息确认模式。

```
Session session;
/* Session is not transacted,
*    uses AUTO_ACKNOWLEDGE for message
*    acknowledgement
*/
session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

9.1.5 创建 MessageProducer

在得到 Session 后，客户端程序使用 Session 来创建 MessageProducer。这个 MessageProducer 对象用于向目的地发送消息。通过使用 Session.createProducer 方法来创建 MessageProducer，参数是消息被发送的目的地。

```
MessageProducer sender;
/* Value in stockQueue previously looked up in the JNDI
*    createProducer takes a Destination
*/
sender = session.createProducer(stockQueue);
```

9.1.6 创建 MessageConsumer

消息可以被同步或异步的消费。这个例子展示了如何创建同步消费消息的消息消费者。

火龙果•整理
uml.org.cn

参见 9.3.1“异步接收消息”了解如何异步消费消息。

MessageConsumer 用于从目的地接收消息，在这个例子中是“StockQueue”。用 Session.createConsumer 方法来创建 MessageConsumer，参数是接收消息的目的地。

```
MessageConsumer receiver;
/* Value in stockQueue previously looked up in the JNDI
*    createConsumer takes a Destination
*/
receiver = session.createConsumer(stockQueue);
```

9.1.7 启动消息转发

到现在为止，消息的转发还是被禁止的，以便前面的设置不会被异步转发的消息打断。现在设置已经完成，可以告诉 Connection 开始向 MessageConsumer 转发消息了。

```
connection.start();
```

9.1.8 使用 TextMessage

有几个消息格式。在这个例子中，股票报价消息作为文本串被发送。客户端按文本串读取和展示这个消息。

下面的代码解释了如何创建这样的消息：

```
String          stockData;      /* Stock information as a string */
TextMessage     message;
/* Set the message's text to be the stockData string */
message = session.createTextMessage();
message.setText(stockData);
```

9.2 发送和接收消息

现在已经完成了 Session 的设置，你可以发送和接收消息了。本节描述了如何：

- 创建消息
- 发送消息
- 同步接收消息

9.2.1 发送消息

为了发送消息，使用 MessageProducer.send 方法，参数是 Message 对象。

```
/* Send the message */
sender.send(message);
```

JANTY

火龙果•整理
uml.org.cn

9.2.2 同步接收消息

为了接收 Queue 里的消息，你可以使用 MessageConsumer.receive 方法。这个方法调用在消息到达 Queue 之前可能阻塞。这个方法也可以用于从 Topic 中接收消息。

```
TextMessage stockMessage;
stockMessage = (TextMessage)receiver.receive();
```

为了限制客户端阻塞的时间长度，使用 receive 方法的超时参数。如果在超时到期时还没有消息到达，那么 receive 方法返回。超时参数按毫秒计时。

```
TextMessage stockMessage;
/* Wait 4 seconds for a message */
TextMessage = (TextMessage)receiver.receive(4000);
```

9.2.3 解包 TextMessage

股票报价信息使用 TextMessage 发送。为了从消息中得到这个信息，使用 TextMessage.getText 方法。它以字符串返回消息的内容。

```
String newStockData;      /* Stock information as a string */
newStockData = message.getText();
```

9.3 其他消息特性

本节不仅提供了基本的消息功能，而且还描述了如何执行某些其他的公共消息功能：

- 创建异步 MessageListener。
- 使用消息选择器过滤消息转发。
- 创建 Topic 的永久订阅。
- 对使用永久订阅的 Topic 进行重新连接。

9.3.1 异步接收消息

为了在消息被转发到消息消费者时异步的接收消息，客户端程序需要创建实现了 MessageListener 接口的消息监听器。本例中 MessageListener 接口的实现称为 StockListener.java，类似于下面的代码：

```
import javax.jms.*;
public class StockListener implements MessageListener
{
public void onMessage(Message message) {
  /* Unpack and handle the messages received */
...
}
 }
```

客户端程序按下面的方式将这个 MessageListener 对象注册到 MessageConsumer 对象：

火龙果•整理
uml.org.cn

```
StockListener myListener = new StockListener();
/* Receiver is MessageConsumer object */
receiver.setMessageListener(myListener);
```

Connection 必须被启动以迎接开始的消息转发。当消息被发布到 Queue 时会异步的通知 MessageListener。这通过 MessageListener 接口的 onMessage 方法实现。由客户端负责处理消息。

```
public void onMessage(Message message)
{
String newStockData;
/* Unpack and handle the messages received */
newStockData = message.getText();
if(...)
{
  /* Logic related to the data */
}
}
```

9.3.2 使用消息选择器

在这个例子中，客户端程序只对技术相关的股票感兴趣。消息的发送者向消息的属性中设置一个称为 StockSector 的属性。它的值包括“Technology”，“Financial”，“Manufacturing”，“Emerging”和“Global”。消息发送者通过使用 Message.setStringProperty 方法来设置这些属性值。

```
String stockData;        /* Stock information as a String */
TextMessage message;
/* Set the message's text to be the stockData string *
message = session.createTextMessage();
message.setText(stockData);
/* Set the message property 'StockSector'
message.setStringProperty("StockSector", "Technology");
```

当接收股票报价消息客户端程序 MessageConsumer 被创建时，它可以创建一个消息选择器字符串来决定它将接收哪些消息。

```
String selector;
selector = new String("(StockSector = 'Technology')"
```

这个字符串在创建 MessageConsumer 时指定。

```
MessageConsumer receiver;
receiver = session.createConsumer(stockQueue, selector);
```

客户端程序只接收和 Technology 相关的消息。

9.3.3 使用永久订阅

永久订阅用于从 Topic 接收消息。当 JMS 客户端创建永久订阅时，客户端能够从 Topic 断开连接。当客户端程序重新连接时，它可以收到断开连接后到达的消息。在这个例子中，

火龙果•整理
uml.org.cn

Destination 提供对消息更新的通知。

9.3.3.1创建永久订阅

下面的例子创建了从 Topic 获取消息的永久订阅。首先，客户端程序必须执行常规的设置步骤，查找 ConnectionFactory 和 Destination，创建 Connection 和 Session，如节 9.1“准备发送和接收消息”中所述。

```
import javax.naming.*;
import javax.jms.*;
/* Look up connection factory */
ConnectionFactory connectionFactory;
Context messaging = new InitialContext();
connectionFactory =
(ConnectionFactory) Messaging.lookup("ConnectionFactory")
/* Look up destination */
Topic newsFeedTopic;
newsFeedTopic = messaging.lookup("BreakingNews");
/* Create connection and session */
Connection connection;
Session session;
connection = ConnectionFactory.createConnection();
session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

执行完常规的设置后，客户端程序可以创建到目的地的永久订阅者。使用 session.createDurableSubscriber 来创建永久 TopicSubscriber。名字“mySubscription”用于标识永久订阅。

```
session.createDurableSubscriber(newsFeedTopic,"mySubscription");
```

此时，客户端程序可以启动连接和开始接收消息。

9.3.3.2重新连接使用永久订阅的 Topic

为了重新连接一个存在永久订阅的 Topic，客户端程序可以简单的再次调用 session.createDurableSubscriber，参数和以前的相同。客户端程序可以中断连接。使用永久订阅可以让从 Topic 消费消息的客户端程序始终能够得到消息，即使客户端程序不是持续连接的。

```
/* Reconnect to a durable subscription */
session.createDurableSubscriber(newsFeedTopic, "mySubscription");
```

这将会重新建立客户端程序到 Topic 的连接，并且转发客户端断开时到达的消息。但是需要知道几个重要的限制：

- 客户端必须使用同一个 Connection。
- Destination 和订阅的名字必须和以前的一样。
- 如果指定了消息选择器，则它的名字也必须和以前的一样。

如果不能满足这些条件，那么永久订阅被删除，并创建一个新的订阅。

火龙果•整理
uml.org.cn

9.4 JMS 消息类型

有五个 JMS 消息类型。本节提供了如何创建和解包每种消息类型的例子。在每个例子中，在消息中的数据都是与股票报价相关的数据。任何情况下，创建消息真正内容的代码都是可以忽略的。

9.4.1 创建 TextMessage

在这个例子中，股票报价信息作为 TextMessgae 被发送。TextMessage 携带了可以被客户端作为文本字符串读取的消息。

下面的代码解释了如何创建这样的消息：

```
String          stockData;      /* Stock information as a string */
TextMessage     message;
message = session.createTextMessage();
/* Set the stockData string to the message body */
message.setText(stockData);
```

9.4.2 解包 TextMessage

为了解包 TextMessage，客户端使用 Message.getText 方法。

```
String stockInfo;       /* String to hold stock info */
stockInfo = message.getText();
```

9.4.3 创建 BytesMessage

股票报价信息可以以字节格式发送，服务器知道如何构造它，客户端知道如何作为股票报价来解析和展现它。它作为 BytesMessage 被发送。

这样的消息可以按下面的方式来构造：

```
byte[]          stockData;      /* Stock information as a byte array */
BytesMessage    message;
message = session.createBytesMessage();
message.writeBytes(stockData);
```

9.4.4 解包 BytesMessage

当收到 BytesMessage 时，可以按照以下方式来解包：

```
byte[]      stockInfo; /* Byte array to hold stock information */
int         length;
length = message.readBytes(stockData);
```

消息体被复制到字节数组。客户端程序然后可以开始读和解析数据。

火龙果•整理
uml.org.cn

9.4.5 创建 MapMessage

由服务器发送的每个股票消息都可以是不同股票报价名字/值对的映射，使用 MapMessage。例如，它可以包含以下条目：

- 股票订价名称——String
- 当前的值——double
- 订价时间——long
- 最后更新——double
- 股票信息——String

为了构造 MapMessage，客户端程序使用不同的与 MapMessage 关联的 set 方法（setString，setLong 等等），并在 MapMessage 中设置命名的值。

```
String stockName;        /* Name of the stock */
double stockValue;       /* Current value of the stock */
long     stockTime;          /* Time the stock quote was updated */
double stockDiff;        /* the +/- change in the stock quote*/
String stockInfo;        /* Other information on this stock */
MapMessage message;
message = session.createMapMessage();
```

注意，下面的设置可以按任何顺序进行：

```
/* First parameter is the name of the map element,
*    second is the value
*/
message.setString("Name",    "SUNW");
message.setDouble("Value", stockValue);
message.setLong("Time",       stockTime);
message.setDouble("Diff",    stockDiff);
message.setString("Info",        "Recent    server    announcement    causes    market
interest");
```

9.4.6 解包 MapMessage

为了解包 MapMessage，客户端程序使用不同的与 MapMessage 相关的 get 方法来获取 MapMessage 各元素的值。在下面的例子中，客户端程序希望得到某些 MapMessage 元素。

```
String stockName;          /*Name of the stock */
double stockValue;         /* Current value of the stock */
long     stockTime;          /* Time of the stock update */
double stockDiff;          /* +/- change in the stock */
String stockInfo;          /* Information on this stock */
```

通过使用 get 方法和提供期望的值的名字从消息中取出的数据。可以按任意顺序从 MapMessage 中得到它的元素。

```
stockName    = message.getString("Name");
stockDiff    = message.getDouble("Diff");
stockValue = message.getDouble("Value");
```

火龙果•整理
uml.org.cn

```
stockTime    = message.getLong("Time");
```

如果客户端需要得到 MapMessage 的元素列表，那么它可以使用 MapMessage.getMapNames 方法。

9.4.7 创建 StreamMessage

与 MapMessage 类似，应用可以发送由按序写入消息的不同字段组成的消息，每个字段都是原始数据类型。使用 StreamMessage 来实现。下面的原始类型赋给股票报价消息的每一项。

- 股票订价名称——String
- 当前的值——double
- 订价时间——long
- 最后更新——double
- 股票信息——String

客户端可能只对某些消息字段感兴趣，但在使用 StreamMessage 的情况下，客户端必须按顺序读并暗地抛弃每个字段。

在下面的例子中，已经设置了下面字段的值：

```
String stockName;          /* Name of the stock */
double stockValue;         /* Current value of the stock */
long     stockTime;          /* Time of the stock update */
double stockDiff;          /* +/- change in the stock quote */
String stockInfo;          /* Information on this stock*/
StreamMessage message;
/* Create message */
message = session.createStreamMessage();
```

下面的元素必须按照它们被读取的顺序写入 StreamMessage。注意，它们不是单独命名的属性，和 MapMessage 中一样。

```
/* Set data for message */
message.writeString(stockName);
message.writeDouble(stockValue);
message.writeLong(stockTime);
message.writeDouble(stockDiff);
message.writeString(stockInfo);
```

9.4.8 解包 StreamMessage

StreamMessage 的元素必须按照写入的顺序被读取。

```
String stockName;          /* Name of the stock quote */
double stockValue;         /* Current value of the stock */
long     stockTime;          /* Time of the stock update */
double stockDiff;          /* +/- change in the stock quote */
String stockInfo;          /* Information on this stock */
stockName    = message.readString();
```

火龙果•整理
uml.org.cn

```
stockValue = message.readDouble();
stockTime    = message.readLong();
stockDiff    = message.readDouble();
stockInfo    = message.readString();
```

9.4.9 创建 ObjectMessage

股票信息可以以特殊的 StockObject 对象的形式被发送。这个对象然后作为 ObjectMessage 的消息体被发送。ObjectMessage 可以用于发送 java 对象。

使用那些 StockObject 实现中互不重复的方法来设置这些值。例如，StockObject 可以有设置不同数据值的方法。使用 StockObject 的应用可能看起来类似如下代码：

```
String stockName;          /* Name of the stock quote */
double stockValue;         /* Current value of the stock */
long     stockTime;          /* Time of the stock update */
double stockDiff;          /* +/- change in the stock quote */
String stockInfo;          /* Information on this stock */
/* Create a StockObject */
StockObject stockObject = new StockObject();
/* Establish the values for the StockObject */
stockObject.setName(stockName);
stockObject.setValue(stockValue);
stockObject.setTime(stockTime);
stockObject.setDiff(stockDiff);
stockObject.setInfo(stockInfo);
```

为了创建 ObjectMessage，将 StockObject 传入消息，你将按下面的方式来实现：

```
/* Create an ObjectMessage */
ObjectMessage message;
message = session.createObjectMessage();
/* Set the body of the message to the StockObject */
message.setObject(stockObject);
```

9.4.10解包 ObjectMessage

为了解包 ObjectMessage，使用 ObjectMessage.getObject 方法来得到对象。一旦对象被取出，客户端应用使用与对象类型匹配的方法从对象中取出数据。

```
StockObject stockObject;
/* Retrieve the StockObject from the message */
stockObject = (StockObject)message.getObject();
/* Extract data from the StockObject by using StockObject methods */
String stockName;          /* Name of the stock quote */
double stockValue;         /* Current value of the stock */
long     stockTime;          /* Time of the stock update */
double stockDiff;          /* +/- change in the stock quote */
```

火龙果•整理
uml.org.cn

```
String stockInfo;          /* Information on this stock */
stockName =    stockObject.getName();
stockValue = stockObject.getValue();
stockTime =    stockObject.getTime();
stockDiff =    stockObject.getDiff();
stockInfo =    stockObject.getInfo();
```

10 问题

10.1 已解决的问题

10.1.1JDK1.1.x 兼容性

JMS 兼容 JDK1.1.x。

10.1.2分布式 Java 事件模型

一般情况下，JMS 可以被用作通知服务；但是它没有定义 java 事件的分布版本。

实现分布式 Java 事件的一个替代方案是将事件作为 JavaBean 通过 JMS 透明地分发它，这种分发对事件的生产者和监听器 bean 是透明的。

10.1.3可以合并 JMS 的两个域 PTP 和 Pub/Sub 吗？

尽管有许多类似点，提供分离的域似乎仍然是重要的。

这意味着供应商不必支持超出域的那些工具，且产品完全支持一个域会使客户端代码有更好的移植性（相对于对合并域的子集做比较少的支持）。

10.1.4JMS 应当指定一个 JMS JavaBean 集合吗？

JMS 是一个底层 API，和其它的 Java 底层 API 一样，它不直接作为 JavaBean 出现。

10.1.5与 CORBA 通知服务对齐

通知服务增加了过滤、转发保证语义、永久连接和对 CORBA 事件服务的事件网络的组装。它从 CORBA 消息服务得到它的转发保证语义（这个服务定义了异步 CORBA 方法调用）。

Java 技术可以和 CORBA 很好的集成。它提供了 Java IDL 和 COS 命名。另外，OMG 最近定义了基于 IIOP 的 RMI。

在 Java 中期望通过 RMI 来使用 IIOP。希望通过 JNDI 来使用 COS 命名。JMS 是 Java API，

火龙果•整理
uml.org.cn

它设计的目标是架构在现存的和将来的 MOM 系统之上的一层 API（就像 JNDI 是位于现存的命名和目录服务至上）。

10.1.6JMS 应当提供端对端的同步消息转发和转发通知吗？

某些消息系统将向目的地同步转发作为实现可靠应用的机制来提供。某些系统向客户端提供不同形式的转发通知来让客户端可以检测丢弃或忽略的消息。这都不是 JMS 定义的模型。

JMS 通过 PERSISTNECT 消息的一次只有一个的转发语义提供了转发保证。另外，消息消费者通过使用 CLIENT_ACKNOWLEDGE 模式或事务性会话可以保证消息的可靠处理。

这样就可以使用最小的同步达到可靠的转发，并且这是大多数供应商和开发者喜欢的企业消息模型。

JMS 没有定义系统消息的模式（Schema）（例如转发通知）。如果应用需要对这些消息接收进行确认，那么它可以定义应用级的消息确认。

当在 Pub/Sub 应用的上下文中检验这些问题时，它们可以更好理解。在这个上下文中，同步转发和/或接收的系统确认都不是实现可靠应用有效的机制（因为按照定义，生产者不负责也不想负责端对端（End to End）的消息转发）。

10.1.7JMS 应当提供发送到列表的机制吗？

目前，JMS 提供了大量的消息发送选项；但是消息一次只能被发送到一个目的地。

发送到列表的好处是简化了程序员的工作，且潜在地可以让 JMS 提供商优化向多个目的地发送同一个消息。

发送到列表机制的缺点是列表效果上是客户端实现和维护的一个组。这复杂了 JMS 客户端的管理。

JMS 提供发送到列表的机制的替代方案，推荐提供商支持代表一个组的目的地。这可以让客户端只要一个发送就可以到达多个消费者，但要保证组受到妥善的管理。

10.1.8JMS 应当提供订阅通知吗？

如果发布者能够检测到订阅者什么时候存在，那么它可以禁止想未订阅的主题上发布消息。

尽管为发布者提供禁止向未订阅的主题上发布消息的机制有一些好处，但这将增加 JMS 的复杂性，且它给提供商增加的工作量大于了它得到的好处。反而，JMS 提供商应当保证最小化处理发布到未订阅主题的消息的负载。

11 变更历史

略。