JMS规范1.1（中文版）

徐 岚

JMS规范1.1（中文版）

徐 岚

目录

插图清单

表格清单

第 1 章 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。

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。

1.2.3.1 JMS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。

1.2.3.2 JMS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。

1.2.3.3 JMS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）。

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。

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。

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”。

第 2 章 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。
- 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. JMS管理

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。

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 2.2. 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
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。

第 3 章 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。

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

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。

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:”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。

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。

3.4.10 JMSPriority

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

头字段	设置者
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。

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 3.2. 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
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

名字	类型	设置者	用法
			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。

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。

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”

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。

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。

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使用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。

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。

第 4 章 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。

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。
- 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。

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。

Session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。
- 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。

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 XAResourceAPI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。

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，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。

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。

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。

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

接口	方法
	createTopic
	unsubscribe
TopicSession	createQueueBrowser
	createQueue
	createTemporaryQueue

第 5 章 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。

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 PT 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。

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。

第 6 章 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。

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。

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。

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

第 7 章 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。

第 8 章 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。

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。

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。

图 8.1.

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

图 8.2.

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

JMS公共接口	PTP接口	Pub/Sub接口
XAConnectionFactory	XAQueueConnectionFactory	XATopicConnectionFactory
XAConnection	XAQueueConnection	XATopicConnection
XASession	XAQueueSession	XATopicSession