参考: <http://blog.csdn.net/bestlove12345/article/details/51880092>

[**深入浅出JMS(一)--JMS基本概念**](http://blog.csdn.net/bestlove12345/article/details/51880092)

摘要：The [**Java**](http://lib.csdn.net/base/17) Message Service (JMS) API is a messaging standard(消息标准) that allows application components based on the [**Java**](http://lib.csdn.net/base/java)Platform Enterprise Edition ([**Java EE**](http://lib.csdn.net/base/17)) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

JMS（[**java**](http://lib.csdn.net/base/java)Message Service,[**Java**](http://lib.csdn.net/base/java)消息服务）API是一个消息服务的标准或者说是规范，允许应用程序组件基于JavaEE平台创建、发送、接收和读取消息。它使分布式通信耦合度更低，消息服务更加可靠以及异步性。

这篇博文我们主要介绍J2EE中的一个重要规范JMS，因为这个规范在企业中的应用十分的广泛，也比较重要，我们主要介绍JMS的基本概念和它的模式，消息的消费以及JMS编程步骤。

1. 基本概念

JMS是java的消息服务，JMS的客户端之间可以通过JMS服务进行异步的消息传输。

1. 消息模型（两种消息传递方式）

Point-to-Point(P2P)

Publish/Subscribe(Pub/Sub)

即点对点和发布订阅模型

1. P2P
   1. P2P模式图   
      
   2. 涉及到的概念
      1. 消息队列（Queue）
      2. 发送者(Sender)
      3. 接收者(Receiver)
      4. 每个消息都被发送到一个特定的队列，接收者从队列中获取消息。队列保留着消息，直到他们被消费或超时。
   3. P2P的特点
      1. 每个消息只有一个消费者（Consumer）(即一旦被消费，消息就不再在消息队列中)
      2. 发送者和接收者之间在时间上没有依赖性，也就是说当发送者发送了消息之后，不管接收者有没有正在运行，它不会影响到消息被发送到队列；同样，消费者消费消息也不管发送者是否正在运行。
      3. 接收者在成功接收消息之后需向队列应答成功
      4. 相对可靠性高

如果你希望发送的每个消息都应该被成功处理的话，那么你需要P2P模式。

1. Pub/Sub
   1. Pub/Sub模式图   
      
   2. 涉及到的概念
      1. 主题（Topic）
      2. 提供者（容器）
      3. 发布者（Publisher）
      4. 订阅者（Subscriber）   
         客户端将消息发送到主题。多个发布者将消息发送到Topic,系统将这些消息传递给多个订阅者。
   3. Pub/Sub的特点
      1. 每个消息可以有多个消费者
      2. 发布者和订阅者之间有时间上的依赖性。针对某个主题（Topic）的订阅者，它必须创建一个订阅者之后，才能消费发布者的消息，而且为了消费消息，订阅者必须保持运行的状态（而且对于订阅某个主题的消费者来说，在订阅时间之前的该主题的消息是不能被消费的）。
      3. 为了缓和这样严格的时间相关性，JMS允许订阅者创建一个可持久化的订阅。这样，即使订阅者没有被激活（运行），它也能接收到发布者的消息。
      4. 相对于P2P模型来说可靠性低，但是吞入量大

如果你希望发送的消息可以不被做任何处理、或者被一个消息者处理、或者可以被多个消费者处理的话，那么可以采用Pub/Sub模型

1. 消息的消费

在JMS中，消息的产生和消费是同步也可以是异步的。对于消费来说，JMS的消息者可以通过两种方式来消费消息。   
○ 同步   
订阅者或接收者调用receive方法来接收消息，receive方法在能够接收到消息之前（或超时之前）将一直阻塞   
○ 异步   
订阅者或接收者可以注册为一个消息监听器。当消息到达之后，系统自动调用监听器的onMessage方法。

1. JMS编程模型

(1) ConnectionFactory

创建Connection对象的工厂，针对两种不同的jms消息模型，分别有QueueConnectionFactory和TopicConnectionFactory两种。可以通过JNDI来查找ConnectionFactory对象。

(2) Destination（生产者的发送目标和消费者的消息来源）

Destination的意思是消息生产者的消息发送目标或者说消息消费者的消息来源。对于消息生产者来说，它的Destination是某个队列（Queue）或某个主题（Topic）;对于消息消费者来说，它的Destination也是某个队列或主题（即消息来源）。

所以，Destination实际上就是两种类型的对象：Queue、Topic可以通过JNDI来查找Destination。

(3) Connection

Connection表示在客户端和JMS系统之间建立的链接（对TCP/IP socket的包装）。Connection可以产生一个或多个Session。跟ConnectionFactory一样，Connection也有两种类型：QueueConnection和TopicConnection。

(4) Session

Session是我们操作消息的接口。可以通过session创建生产者、消费者、消息等。Session提供了事务的功能。当我们需要使用session发送/接收多个消息时，可以将这些发送/接收动作放到一个事务中。同样，也分QueueSession和TopicSession。

(5) 消息的生产者

消息生产者由Session创建，并用于将消息发送到Destination。同样，消息生产者分两种类型：QueueSender和TopicPublisher。可以调用消息生产者的方法（send或publish方法）发送消息。

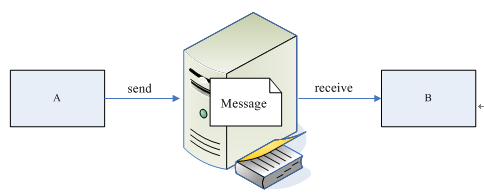
(6) 消息消费者

消息消费者由Session创建，用于接收被发送到Destination的消息。两种类型：QueueReceiver和TopicSubscriber。可分别通过session的createReceiver(Queue)或createSubscriber(Topic)来创建。当然，也可以session的creatDurableSubscriber方法来创建持久化的订阅者。

(7) MessageListener

消息监听器。如果注册了消息监听器，一旦消息到达，将自动调用监听器的onMessage方法。EJB中的MDB（Message-Driven Bean）就是一种MessageListener。

1. 企业消息系统的好处

我们先来看看下图，应用程序A将Message发送到服务器上，然后应用程序B从服务器中接收A发来的消息，通过这个图我们一起来分析一下JMS的好处：   


1. 提供消息灵活性
2. 松散耦合
3. 异步性

对于JMS的基本概念我们就介绍这么多，下篇博文介绍一种JMS的实现。

# [深入浅出JMS（二）——JMS的组成](http://blog.csdn.net/aking21alinjuju/article/details/6071123)

##### JMS Provider

      实现了JMS规范的消息系统，该系统还提供必须的用于管理和控制全方位的功能，如这里的ActiveMQ，kafka等。

##### Administered Objects（被管理的对象）

      Administered Objects是预先配置的JMS对象，由系统管理员为使用JMS的客户端（生产者或者是消费者）创建，如ConnectionFactory、Destination。JMS被管理的对象是包含JMS配置信息的对象，这些对象由JMS管理者创建，并且最终由JMS客户端使用。

      由于有很多JMS消息系统，它们的底层实现技术各不相同，比如Sun MQ、IBM MQ、BEA MQ、Apache ActiveMQ，那么如何使得JMS客户端针对这些消息系统编程时能够隔离这些产品的变化而具有跨平台特性呢？就是通过定义被管理的对象来实现。被管理的对象是由管理员通过使用JMS系统提供者的管理工具创建和定制，然后被JMS客户端使用。JMS客户端通过接口（面向接口编程）来调用这些被管理的对象，从而具备跨平台特性。

      主要有两个被管理的对象：ConnectionFactory、Destination

###### ConnectionFactory

      这是客户端（可能是生产者，也可能是消费者）用来创建同JMS服务提供者之间的连接的对象。

###### Destination

      这个对象是客户端用来指明消息被发送的目的地以及客户端接收消息的来源。被管理的对象一般被管理员放在JNDI名字空间中，通常在JMS客户端应用的文档中说明它所需要的JMS被管理对象，以及应以何种JNDI名字来提供这些JMS被管理对象。

env.put(Context.*PROVIDER\_URL*, "localhost:1099");

##### JMS客户端

      这些客户端程序通过使用JMS提供的API，来创建发送和接收消息的[**Java**](http://lib.csdn.net/base/java)语言程序，即消息生产者和消息消费者。

###### 消息生产者

      客户端使用MessageProducer向目的地发送消息。用Queue或者Topic对象作为参数来调用session对象的createProducer方法来创建MessageProducer。

      客户端也可以选择创建没有目的地的生产者。这种情况下，目的地对象必须传给每个发送操作。这种方式的一个典型用法就是生产者被用来发送回复请求时，使用请求的JMSReplyTo目的地。

      客户端可指定由生产者发出的消息的缺省的传送模式、优先级、存活周期。每次客户端创建MessageProducer，它就定义了新的消息系列，这些消息与以前发送的消息没有顺序关系。

###### 消息消费者

      客户端使用MessageConsumer接受来自于目的地的消息，MessageConsumer通过向Session的createConsumer方法传递Queue或Topic来创建。

      消费者可以被带有消息选择器的方式来创建。这使得客户端可以限制传送给消费者的消息必须同选择器相匹配。客户端既可以同步获取消费者的消息，也可以使提供者在消息到达时异步传送消息。

**Synchronous Delivery 同步传送**

      客户端可以使用MessageConsumer的receive方法请求下一个来自于MessageConsumer的消息。Receive有几种变化允许客户端poll或者wait下一个消息。

**Asynchronous Delivery 异步传送**

      客户端可以注册一个用MessageConsumer来实现JMS MessageListener接口的对象。当消息达到了消费者时，提供者通过调用监听器的onMessage方法来传送它们。可能监听器会抛出RuntimeException异常，但是这主要考虑到的是客户端程序错误。良好的监听器应当捕捉这些异常并且尝试将这些消息转向发给一些应用指定的某些形式的“不可处理消息”的目的地。

监听器抛出RuntimeException的结果取决于会话的确认模式：

n AUTO\_ACKNOWLEDGE or DUPS\_OK\_ACKNOWLEDGE

      消息将被立即重发。在放弃之前的重发的次数取决于提供商。在这种情况下，JMSRedelivered 消息头字段将设置在被重发的消息中。

n CLIENT\_ACKNOWLEDGE

      监听器的下一个消息将被传送。如果客户端希望是前面未确认的消息重新发送，它必须手工恢复会话。

n Transacted Session

      监听器的下一个消息被发送，客户端可以提交或者回滚会话。（换句话说，RuntimeException不会导致会话的自动回滚）JMS服务提供者应当将消息监听器抛出异常的客户端标记为“可能的障碍”。

##### Message

      用于在客户端之间进行通讯的消息，消息系统的核心当然是消息。JMS 为不同类型的内容提供了几种消息类型，但所有消息都是从 Message 接口派生出来的。

|  |  |  |
| --- | --- | --- |
| 消息类型 | 说明 | 常用方法概览 |
| TextMessage | 文本消息 | getText,setText |
| MapMessage | 映射消息 | setString,getString |
| BytesMessage | 字节消息 | writeBytes,readBytes |
| StreamMessage | 流消息 | writeString,readString |
| ObjectMessage | 对象消息 | setObject,getObject |

                                                         表 常用消息类

Message 分为三个组成部分：

###### Header

      是一组标准字段，客户机和提供者都用它们来标识和路由消息。

###### Properties

      提供了一个给消息添加可选标题字段的实用工具。如果应用程序需要用标准标题字段没有提供的方法对消息进行归类或分类，那么可以为消息添加一个属性来实现这种归类和分类；提供了 setProperty(...) 和 getProperty(...) 方法来设置和获得各种 Java 类型的属性，其中包括 Object。JMS 定义了提供者可以选择性提供的一组标准属性。

###### Body

      包含将发送到接收应用程序的内容。每一个消息接口都专用于它所支持的内容类型。

下面列出了 Message 的每一个标题字段的名称、它对应的 Java 类型和字段的描述：

**JMSMessageID——类型为 string**

      惟一标识提供者发送的每一条消息。这个字段是在发送过程中由提供者设置的，客户机只能在消息发送后才能确定消息的 JMSMessageID。

**JMSDestination——类型为 Destination**

      消息发送的 Destination，在发送过程中由提供者设置。

**JMSDeliveryMode—类型为 int**

      包含值 DeliveryMode.PERSISTENT 或者 DeliveryMode.NON\_PERSISTENT。持久性消息被传输并且只被传输一次，非持久性消息最多被传输一次。要知道“最多一次”包括根本不传输。非持久性消息在应用程序或者系统出故障时被提供者弄丢。因此要格外小心，确保持久性消息不受故障的影响。这笔开销通常被认为是发送持久性消息方面的开销，在决定消息的发送模式时，必须仔细考虑，在可靠性和性能之间进行权衡。

**JMSTimestamp——类型为 long**

      提供者发送消息的时间，由提供者在发送过程中设置。

**JMSExpiration——类型为 long**

      消息失效的时间。这个值是在发送过程中计算的，是发送方法的生存时间（time-to-live）值和当前时间值的和。提供者不应发送过期的消息。值 0 表明消息不会过期。

**JMSPriority——类型为 int**

      消息的优先级，由提供者在发送过程中设置。优先级 0 的优先级最低，优先级 9 的优先级最高。

**JMSCorrelationID——类型为 string**

      通常用来链接响应消息与请求消息，由发送消息的 JMS 程序设置。响应来自另一个 JMS 程序的消息的 JMS 程序将正响应消息的 JMSMessageID 拷贝到这个字段中，这样，正作出响应的程序就可以与它所发出的特定请求的响应相关联。

**JMSReplyTo—类型为 Destination**

      请求程序用它来指出回复消息应发送的地方，由发送消息的 JMS 程序设置。

**JMSType—类型为 string**

      JMS 程序用它来指出消息的类型。一些提供者维护着一个消息类型仓库，并用该字段引用仓库中的定义类型，在这里，JMS 程序不应该使用这个字段。

**JMSRedelivered——类型为 boolean**

      指出消息被过早地发送给了 JMS 程序，程序不知道消息的接收者是谁；由提供者在接收过程中设置。