## 一、消息服务

消息服务指的是两个应用程序之间进行异步通信的API，它为标准消息协议和消息服务提供了一组通用接口，包括创建、发送、读取消息等，用于支持应用程序开发。

在Java中，当两个应用程序使用JMS进行通信时，它们之间并不是直接相连的，而是通过一个共同的消息收发服务连接起来，可以达到解耦的效果。

## 二、JMS

### 2.1简介

JMS即Java消息服务（Java Message Service）应用程序接口，是一个Java平台中关于面向消息中间件Message-Oriented Middleware（MOM-分布式系统的集成）的API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

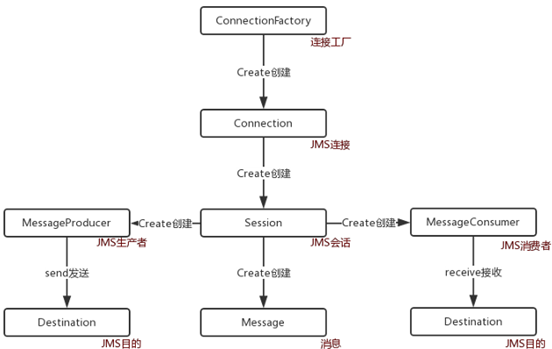
JMS是一种与厂商无关的API，用来访问消息收发系统消息，它类似于JDBC（Java Database Connectivity）。

### 2.2体系架构

JMS由以下元素组成：

|  |  |
| --- | --- |
| JMS提供者 | 连接面向消息中间件的，JMS接口的一个实现。提供者可以是Java平台的JMS实现，也可以是非Java平台的面向消息中间件的适配器。 |
| JMS客户 | 生产或消费基于消息的Java的应用程序或对象。 |
| JMS生产者 | 创建并发送消息的JMS客户。 |
| JMS消费者 | 接收消息的JMS客户 |
| JMS消息 | 包括可以在JMS客户之间传递的数据的对象。 |
| JMS队列 | 一个容纳那些被发送的等待阅读的消息的区域。与队列名字所暗示的意思不同，消息的接受顺序并不一定要与消息的发送顺序相同。一旦一个消息被阅读，该消息将被从队列中移走。 |
| JMS主题 | 一个支持发送消息给多个订阅者的机制。 |

### 2.3JMS对象模型



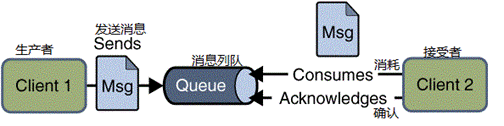
|  |  |
| --- | --- |
| **ConnectionFactory** | 创建Connection对象的工厂，针对两种不同的JMS消息模型，分别有QueueConnectionFactory和TopicConnectionFactory两种。可以通过JNDI来查找ConnectionFactory对象。 |
| **Connection** | Connection表示在客户端和JMS系统之间建立的链接（对TCP/IP socket的包装）。Connection可以产生一个或多个Session。跟ConnectionFactory一样，Connection也有两种类型：QueueConnection和TopicConnection。 |
| **Session** | Session是操作消息的接口。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。 |
| **MessageProducer** | 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。 |
| **MessageConsumer** | 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。 |
| **Destination** | 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）。 |

### 2.4JMS消息模型（PTP,Pub/Sub）

在JMS标准中，有两种消息模型PTP（Point to Point），Publish/Subscribe(Pub/Sub)。

PTP模式-点对点消息传送模型

在点对点消息传送模型中，应用程序由消息队列，发送者，接受者组成。每一个消息发送给一个特殊的消息队列，该队列保存了所有发送给它的消息（除了被接受者消费掉的和过期的消息）。



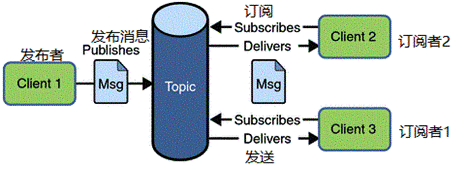
PTP特点：

1. 每个消息只有一个消费者(Consumer)即一旦被消费，消息就不在消息队列中。
2. 发送者和接受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，不管接受者有没有正在运行，它不会影响到消息被发送到队列。
3. 接受者在成功接收消息之后需向队列发送确认收到通知(acknowledgement)。

Pub/Sub-发布/订阅消息传递模型

在发布/订阅消息模型中，发布者发布一个消息，该消息通过topic传递给所有客户端。在这种模型中，发布者和订阅者彼此不知道对方，是匿名的且可以动态发布和订阅topic。

在发布/订阅消息模型中，目的地被称为主题（topic），topic主要用于保存和传递消息，且会一致保存消息直到消息被传递给客户端。



Pub/Sub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，它必须创建一个或多个订阅之后，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。
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。

### 2.5接收消息

JMS中，消息的接收可以使用以下两种方式：

|  |  |
| --- | --- |
| **同步** | 使用同步方式接收消息的话，消息订阅者调用receive()方法。在receive()中，消息未达到或在到达指定时间之前，方法会阻塞，直到消息可用。 |
| **异步** | 使用异步方式接收消息的话，消息订阅者需注册一个消息监听者，类似于事件监听器，只要消息到达，JMS服务提供者会通过调用监听器的onMessage()递送消息。 |

### 2.6JMS消息结构（Message）

Message主要由三部分组成，分别是Header，Properties，Body，详细如下：

|  |  |
| --- | --- |
| **Header** | 消息头，所有类型的这部分格式都是一样的。 |
| **Properties** | 属性，按类型可以分为应用设置的属性，标准属性和消息中间件定义的属性。 |
| **Body** | 消息正文，指我们具体需要消息传输的内容。 |

Header

header中的各个属性，可以分为两大类：

1. 自动分配的消息头：

这里这些JMS消息头是自动分配的。

在传送消息时，消息头的值由JMS提供者来设置，因此开发者使用setJMSxxx()方法分配的值就被忽略了。换句话说，对于大多数自动分配的消息头来说，使用赋值函数方法显然是徒劳的。不过，这并非意味着开发者无法控制这些消息头的值。一些自动分配的消息头可以在创建Session和MessageProducer（也就是TopicPublisher）时，由开发者通过编程方式来设置。

|  |  |  |
| --- | --- | --- |
| 属性名称 | 说明 | 设置者 |
| **JMSDeliveryMode**  **消息发送模式** | 分为NON\_PERSISTENT和PERSISTENT，非持久性模式和持久性模式。默认设置为PERSISTENT。  一条持久性消息应该被传送一次（就一次），这就意味着如果JMS提供者出现故障，该消息并不会丢失； 它会在服务器恢复正常之后再次传送。  一条非持久性消息最多只会传送一次，这意味着如果JMS提供者出现故障，该消息可能会永久丢失。 | send |
| **JMSMessageID**  **消息ID** | 需要以ID:开头，用于唯一地标识了一条消息。 | send |
| **JMSTimestamp**  **消息发送的时间** | 这条消息头用于确定发送消息和它被消费者实际接收的时间间隔。时间戳是一个以毫秒来计算的Long类型时间值（自1970年1月1日算起）。 | send |
| **JMSExpiration**  **消息过期时间** | 以毫秒为单位，用来防止把过期时间的消息传送给消费者。任何直接通过编程方式来调用setJMSExpiration()犯法都会被忽略。 | send |
| **JMSRedelivered**  **消息是否重复发送** | 如果该消息之前发送过，那么这个属性的值需要被设置为true，客户端可以根据这个属性的值来确认是否重复发送过，以避免重复处理。 | provider |
| **JMSPriority**  **消息优先级** | 0-4为普通的优化级，而5-9位高优先级，通常情况下，高优化级的消息需要优先发送，任何直接通过编程方式调用setJMSPriority()方法都将被忽略。 | send |
| **JMSDestination**  **消息发送目的地** | 是一个Topic或Queue。 | send |

1. 开发者分配的消息头：

|  |  |  |
| --- | --- | --- |
| 属性名称 | 说明 | 设置者 |
| **JMSCorrelationID**  **关联的息ID** | 通常用在需要回传消息的时候。 | client |
| **JMSReplyTo**  **消息回复目的地** | 其值为一个Topic或Queue，这个由发送者设置，但是接受者可以决定是否响应。 | client |
| **JMSType**  **消息类型** | 代表消息的结构，有的消息中间件可能会用到这个，但这个并不是批消息的种类，比如TextMessage之类的。 | client |

从上表中我们可以看到，系统提供的标准头信息一共有10个属性，其中有6个是由send方法在调用时设置的，有三个是由客户端（client）设置的，还有一个是由消息中间件（Provider）设置的。

需要注意的是，这里的客户端（client）不是指消费者，而是指使用JMS的客户端，即开发者所写的应用程序，即在生产消息时，这三个属性是可以由应用程序来设定的，而其它的header要么由消息中间件设置，要么由发送方法来决定，开发者即使设置了，也是无效的。测试如下：

生产者：

//创建文本消息

TextMessage textMessage = session.createTextMessage("消息内容" + (i + 1 ));

//消息发送的目的地

textMessage.setJMSDestination(new Queue(){

@Override

public String getQueueName() throws JMSException {

return name;

}

});

//消息的发送模式

textMessage.setJMSDeliveryMode(DeliveryMode.NON\_PERSISTENT);

//消息ID

textMessage.setJMSMessageID("ID:JMSMessageID");

//消息发送时的时间

textMessage.setJMSTimestamp(1000);

//关联的消息ID

textMessage.setJMSCorrelationID("100:JMSCorrelationID");

//消息回复的目的地

textMessage.setJMSReplyTo(new Queue(){

@Override

public String getQueueName() throws JMSException {

return name;

}

});

//消息是否重复发送过

textMessage.setJMSRedelivered(true);

//消息类型，代表消息的结构

textMessage.setJMSType("type");

//消息的过期时间，以毫秒为单位

textMessage.setJMSExpiration(36000);

//消息的优先级,0-4为普通的优化级，而5-9为高优先级

textMessage.setJMSPriority(5);

只有红框的JMSType，ReplayTo，CorrelationId可以显示设置，其它设置了都无效。

public interface Message {

public Destination getJMSDestination() throws JMSException;

public void setJMSDestination(Destination destination) throws JMSException;

public int getJMSDeliveryMode() throws JMSException

public void setJMSDeliveryMode(int deliveryMode) throws JMSException;

public String getJMSMessageID() throws JMSException;

public void setJMSMessageID(String id) throws JMSException;

public long getJMSTimestamp() throws JMSException'

public void setJMSTimestamp(long timestamp) throws JMSException;

public long getJMSExpiration() throws JMSException;

public void setJMSExpiration(long expiration) throws JMSException;

public boolean getJMSRedelivered() throws JMSException;

public void setJMSRedelivered(boolean redelivered) throws JMSException;

public int getJMSPriority() throws JMSException;

public void setJMSPriority(int priority) throws JMSException;

public Destination getJMSReplyTo() throws JMSException;

public void setJMSReplyTo(Destination replyTo) throws JMSException;

public String getJMScorrelationID() throws JMSException;

public void setJMSCorrelationID(String correlationID) throws JMSException;

public byte[] getJMSCorrelationIDAsBytes() throws JMSException;

public void setJMSCorrelationIDAsBytes(byte[] correlationID) throws JMSException;

public String getJMSType() throws JMSException;

public void setJMSType(String type) throws JMSException;

}

消息属性

消息的属性就像可以分配给一条消息的附加消息头一样。它们允许开发者添加有关消息的不透明附加信息。它们还用于暴露消息选择器在消息过滤时使用的数据。

Message接口为读取和写入属性提供了若干个取值函数和赋值函数方法。消息的属性值可以是String,boolean,byte,short,double,int,long或float型。Message接口为每种类型的属性值都提供了取值函数和赋值方法。如下：

public interface Message {

public String getStringProperty(String name) throws JMSException,MessageFormatException;

public void setStringProperty(String name,String value) throws JMSException,MessageNotWriteableException;

public int getIntProperty(String name) throws JMSException,MessageFormatException;

public void setIntProperty(String name,int value) throws JMSException,MessageNotWriteableException;

public boolean getBooleanProperty(String name) throws JMSException,MessageFormatException;

public void setBooleanProperty(String name,boolean value) throws JMSException,MessageNotWriteableException;

public double getDoubleProperty(String name) throws JMSException,MessageFormatException;

public void setDoubleProperty(String name) throws JMSException,MessageFormatException;

public float getFloatProperty (String name) throws JMSException,MessageFormatExdeption;

public void setFloatProperty(String name,float value) throws JMSException,MessageNotWriteableException;

public byte getByteProperty(String name) throws JMSException,MessageFormatException;

public void setByteProperty(String name) throws JMSException,MessageNotWriteableException;

public long getLongProperty(String name) throws JMSException,MessageNotWriteableException;

public void setLongProperty(String name,long value) throws JMSException,MessageNotWriteableException;

public short getShortProperty(String name) throws JMSException,MessageFormatException;

public void setShortProperty(String name,short value) throws JMSException,MessageNotWriteableException;

public Object getObjectProperty(String name) throws JMSException,MessageNotWriteableException;

public void setObjectProperty(String name,Object value) throws JMSException,MessageNotWriteableException;

......

}

消息属性有3种基本类型：

1. 应用程序特定的属性

由应用程序开发者定义的所有属性都可以作为一个应用程序特定的属性。应用程序属性在消息传送之前进行设置。并不存在预先定义的应用程序属性，开发者可以自由定义能够满足它们需要的任何属性。例如，在一个应用中，可以添加一个特定的属性，该属性用于标识正在发送消息的用户：

TextMessage message = pubSession.createTextMessage();

message.setStringProperty("username",username);//自定义属性

publisher.publish(message);

作为一个应用程序的特定属性，username一旦离开该应用程序就变得毫无意义，它专门用于应用程序根据发布者身份对消息进行过滤。

一旦一条消息发布或发送以后，它就变成了只读属性；消费者或生产者都无法修改它的属性。如果消费者试图设置某个属性，该方法就会抛出一个javax.jms.MessageNotWriteableException。

1. JMS定义的属性

JMS定义的属性具有和应用程序属性相同的特性，除了前者大多数在消息发送时由JMS提供者来设置之外。JMS定义的属性可以作为可选的JMS消息头，下面是JMS定义的9个属性清单：

JMSXUserID

JMSXAppID

JMSXProducerTXID

JMSXConsumerTXID

JMSXRcvTimestamp

JMSXDeliveryCount

JMSXState

JMSXGroupID

JMSXGroupSeq

在这份清单中，只有JMSXGroupID和JMSXGroupSeq需要所有JMS提供者的支持。这些可选属性用于聚合消息。

请注意：在Message接口中，您将无法找到对应的setJMSX<PROPERTY>()和getJMSX<PROPERTY>()方法定义，在使用这些方法时，必须使用和应用程序特定属性相同的方法来设置它们，如下：

message.setStringProperty("JMSXGroupID","GroupID-001");

message.setIntProperty("JMSXGroupSeq",5);

1. 提供者特定的属性

每个JMS提供者都可以定义一组私有属性，这些属性可以由客户端或提供者自动设置。提供者特定的属性必须以前缀JMS开头，后面紧接着是属性名称（JMS<vendor-property-name>），例如：JMSUserID。提供者特定的属性，其作用就是支持厂商的私有特性。

消息体

为了适应不同场景下的消息，提高消息存储的灵活性，JMS定义了几种具体类型的消息，不同的子类型的消息体也不一样，需要注意的是，Message接口并没有提供一个统一的getBody之类的方法。消息子接口定义如下：

|  |  |
| --- | --- |
| **TextMessage** | 最简单的消息接口，用于发送文本类的消息，设置/获取其body的方法定义如下setText()/getText()。 |
| **StreamMessage** | 流式消息接口，里面定义了一系列的对基本类型的set/get方法，消息发送者可以通过这些方法写入基本类型的数据，消息接收者需要按发送者的写入顺序来读取相应的数据。 |
| **MapMessage** | 把消息内容存储在Map里，本接口定义了一系列对基本类型的set/get方法，与StreamMessage不同的是，每个值都对应了一个相应的key，所以消息接收者不必按顺序去读取数据。 |
| **ObjectMessage** | 将对象作为消息的接口，提供了一个set/get对象的方法，需要注意的是只能设置一个对象，这个对象可以是一个Collection，但必须是序列化的。 |
| **BytesMessage** | 以字节的形式来传递消息的接口，除了提供了对基本类型的set/get，还提供了按字节方式进行set/get。 |