深入掌握JMS

[深入掌握JMS（一）：JSM基础](http://blog.csdn.net/zhangxs_3/article/details/4034713)

1. JMS基本概念  
     JMS([**Java**](http://lib.csdn.net/base/java) Message Service) 即[**Java**](http://lib.csdn.net/base/javase)消息服务。它提供标准的产生、发送、接收消息的接口简化[企业](http://blog.csdn.net/hanjiaming/article/details/7821160)应用的开发。它支持两种消息通信模型：点到点（point-to-point）（P2P）模型和发布/订阅（Pub/Sub）模型。P2P 模型规定了一个消息只能有一个接收者;Pub/Sub 模型允许一个消息可以有多个接收者。  
    对于点到点模型，消息生产者产生一个消息后，把这个消息发送到一个Queue（队列）中，然后消息接收者再从这个Queue中读取，一旦这个消息被一个接收者读取之后，它就在这个Queue中消失了，所以一个消息只能被一个接收者消费。  
    与点到点模型不同，发布/订阅模型中，消息生产者产生一个消息后，把这个消息发送到一个Topic中，这个Topic可以同时有多个接收者在监听，当一个消息到达这个Topic之后，所有消息接收者都会收到这个消息。

简单的讲，点到点模型和发布/订阅模型的区别就是前者是一对一，后者是一对多。

2. 几个重要概念   
   **Destination** ：消息发送的目的地，也就是前面说的Queue和Topic。创建好一个消息之后，只需要把这个消息发送到目的地，消息的发送者就可以继续做自己的事情，而不用等待消息被处理完成。至于这个消息什么时候，会被哪个消费者消费，完全取决于消息的接受者。  
    **Message** ：从字面上就可以看出是被发送的消息。它有下面几种类型：  
        StreamMessage：Java 数据流消息，用标准流操作来顺序的填充和读取。  
        MapMessage：一个Map类型的消息；名称为 string 类型，而值为 Java 的基本类型。  
        TextMessage：普通字符串消息，包含一个String。  
        ObjectMessage：对象消息，包含一个可序列化的Java 对象  
        BytesMessage：二进制数组消息，包含一个byte[]。  
        XMLMessage:  一个XML类型的消息。  
    最常用的是TextMessage和ObjectMessage。  
   **Session**： 与JMS提供者所建立的会话，通过Session我们才可以创建一个Message。  
   **Connection**： 与JMS提供者建立的一个连接。可以从这个连接创建一个会话，即Session。  
   **ConnectionFactory:** 那如何创建一个Connection呢？这就需要下面讲到的ConnectionFactory了。通过这个工厂类就可以得到一个与JMS提供者的连接，即Conection。  
   **Producer**： 消息的生产者，要发送一个消息，必须通过这个生产者来发送。  
   **MessageConsumer**： 与生产者相对应，这是消息的消费者或接收者，通过它来接收一个消息。  
   前面多次提到JMS提供者，因为JMS给我们提供的只是一系列接口，当我们使用一个JMS的时候，还是需要一个第三方的提供者，它的作用就是真正[管理](http://blog.csdn.net/hanjiaming/article/details/7821160)这些Connection，Session，Topic和Queue等。  
  
    通过下面这个简图可以看出上面这些概念的关系。  
  
  ConnectionFactory---->Connection--->Session--->Message  
  Destination + Session------------------------------------>Producer  
  Destination + Session------------------------------------>MessageConsumer   
   
    那么可能有人会问: ConnectionFactory和Destination 从哪儿得到?  
    这就和JMS提供者有关了. 如果在一个JavaEE环境中, 可以通过JNDI查找得到, 如果在一个非JavaEE环境中, 那只能通过JMS提供者提供给我们的接口得到了.

[深入掌握JMS（二）：一个JMS例子](http://blog.csdn.net/zhangxs_3/article/details/4034775)

        前一讲简单的介绍了一下JMS的基本概念, 这一讲结合一个例子让大家深入理解前一讲的基本概念. 首先需要做的是选择一个JMS提供者, 如果在JavaEE环境中可以不用考虑这些. 我们选择ActiveMQ, 官方地址: http://activemq.apache.org/. 网上有很多介绍ActiveMQ的文档, 所以在这里就不介绍了.

按照上一讲的这个简图,

  ConnectionFactory---->Connection--->Session--->Message  
  Destination + Session------------------------------------>Producer  
  Destination + Session------------------------------------>MessageConsumer

首先需要得到ConnectionFactoy和Destination，这里创建一个一对一的Queue作为Destination。  
  ConnectionFactory factory = new ActiveMQConnectionFactory("vm://localhost");  
  Queue queue = new ActiveMQQueue("testQueue");

然后又ConnectionFactory创建一个Connection, 再启动这个Connection:  
  Connection connection = factory.createConnection();  
  connection.start();

接下来需要由Connection创建一个Session:  
  Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE)  
    现在暂且不用管参数的含义, 以后会详细讲到.

下面就可以创建Message了,这里创建一个TextMessage。  
  Message message = session.createTextMessage("Hello JMS!");

要想把刚才创建的消息发送出去，需要由Session和Destination创建一个消息生产者：  
  MessageProducer producer = session.createProducer(queue);

下面就可以发送刚才创建的消息了：  
  producer.send(message);

消息发送完成之后，我们需要创建一个消息消费者来接收这个消息：  
  MessageConsumer comsumer = session.createConsumer(queue);  
  Message recvMessage = comsumer.receive();

消息消费者接收到这个消息之后，就可以得到它的内容：  
  System.out.println(((TextMessage)recvMessage).getText());

至此，一个简单的JMS例子就完成了。下面是全部源码 ：

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. **import** javax.jms.Connection;
2. **import** javax.jms.Message;
3. **import** javax.jms.MessageConsumer;
4. **import** javax.jms.MessageProducer;
5. **import** javax.jms.Queue;
6. **import** javax.jms.Session;
7. **import** javax.jms.TextMessage;
8. **import** org.apache.activemq.ActiveMQConnectionFactory;
9. **import** org.apache.activemq.command.ActiveMQQueue;
10. **public** **class** MessageSendAndReceive {
11. **public** **static** **void** main(String[] args) **throws** Exception {
12. ConnectionFactory factory = **new** ActiveMQConnectionFactory("vm://localhost");
14. Connection connection = factory.createConnection();
15. connection.start();
17. Queue queue = **new** ActiveMQQueue("testQueue");
19. **final** Session session = connection.createSession(**false**, Session.AUTO\_ACKNOWLEDGE);
20. Message message = session.createTextMessage("Hello JMS!");
22. MessageProducer producer = session.createProducer(queue);
23. producer.send(message);
25. System.out.println("Send Message Completed!");
27. MessageConsumer comsumer = session.createConsumer(queue);
28. Message recvMessage = comsumer.receive();
29. System.out.println(((TextMessage)recvMessage).getText());
30. }
31. }

[深入掌握JMS（三）：MessageListener](http://blog.csdn.net/zhangxs_3/article/details/4034788)

消息的消费者接收消息可以采用两种方式：

  1、consumer.receive() 或 consumer.receive(int timeout)；  
  2、注册一个MessageListener。

  采用第一种方式，消息的接收者会一直等待下去，直到有消息到达，或者超时。后一种方式会注册一个监听器，当有消息到达的时候，会回调它的onMessage()方法。下面举例说明：

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. MessageConsumer comsumer = session.createConsumer(queue);
2. comsumer.setMessageListener(**new** MessageListener(){
3. @Override
4. **public** **void** onMessage(Message m) {
5. TextMessage textMsg = (TextMessage) m;
6. **try** {
7. System.out.println(textMsg.getText());
8. } **catch** (JMSException e) {
9. e.printStackTrace();
10. }
11. }
13. });

[深入掌握JMS（四）：实战Queue](http://blog.csdn.net/zhangxs_3/article/details/4034801)

  Queue实现的是点到点模型，在下面的例子中，启动2个消费者共同监听一个Queue，然后循环给这个Queue中发送多个消息，我们依然采用ActiveMQ。

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. **import** javax.jms.Connection;
2. **import** javax.jms.DeliveryMode;
3. **import** javax.jms.JMSException;
4. **import** javax.jms.Message;
5. **import** javax.jms.MessageConsumer;
6. **import** javax.jms.MessageListener;
7. **import** javax.jms.MessageProducer;
8. **import** javax.jms.Queue;
9. **import** javax.jms.Session;
10. **import** javax.jms.TextMessage;
11. **import** org.apache.activemq.ActiveMQConnectionFactory;
12. **import** org.apache.activemq.command.ActiveMQQueue;
14. **public** **class** QueueTest {
15. **public** **static** **void** main(String[] args) **throws** Exception {
16. ActiveMQConnectionFactory factory = **new** ActiveMQConnectionFactory("vm://localhost");
18. Connection connection = factory.createConnection();
19. connection.start();

22. //创建一个Queue
23. Queue queue = **new** ActiveMQQueue("testQueue");
24. //创建一个Session
25. Session session = connection.createSession(**false**, Session.AUTO\_ACKNOWLEDGE);

28. //注册消费者1
29. MessageConsumer comsumer1 = session.createConsumer(queue);
30. comsumer1.setMessageListener(**new** MessageListener(){
31. **public** **void** onMessage(Message m) {
32. **try** {
33. System.out.println("Consumer1 get " + ((TextMessage)m).getText());
34. } **catch** (JMSException e) {
35. e.printStackTrace();
36. }
37. }
38. });

41. //注册消费者2
42. MessageConsumer comsumer2 = session.createConsumer(queue);
43. comsumer2.setMessageListener(**new** MessageListener(){
44. **public** **void** onMessage(Message m) {
45. **try** {
46. System.out.println("Consumer2 get " + ((TextMessage)m).getText());
47. } **catch** (JMSException e) {
48. e.printStackTrace();
49. }
50. }
52. });

55. //创建一个生产者，然后发送多个消息。
56. MessageProducer producer = session.createProducer(queue);
57. **for**(**int** i=0; i<10; i++){
58. producer.send(session.createTextMessage("Message:" + i));
59. }
60. }
61. }

 运行这个例子会得到下面的输出结果：

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. Consumer1 get Message:0
2. Consumer2 get Message:1
3. Consumer1 get Message:2
4. Consumer2 get Message:3
5. Consumer1 get Message:4
6. Consumer2 get Message:5
7. Consumer1 get Message:6
8. Consumer2 get Message:7
9. Consumer1 get Message:8
10. Consumer2 get Message:9

  可以看出每个消息直被消费了一次，但是如果有多个消费者同时监听一个Queue的话，无法确定一个消息最终会被哪一个消费者消费。

[深入掌握JMS（五）：实战Topic](http://blog.csdn.net/zhangxs_3/article/details/4034811)

    与Queue不同的是，Topic实现的是发布/订阅模型，在下面的例子中，启动2个消费者共同监听一个Topic，然后循环给这个Topic中发送多个消息。

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. **import** javax.jms.Connection;
2. **import** javax.jms.JMSException;
3. **import** javax.jms.Message;
4. **import** javax.jms.MessageConsumer;
5. **import** javax.jms.MessageListener;
6. **import** javax.jms.MessageProducer;
7. **import** javax.jms.Session;
8. **import** javax.jms.TextMessage;
9. **import** javax.jms.Topic;
10. **import** org.apache.activemq.ActiveMQConnectionFactory;
11. **import** org.apache.activemq.command.ActiveMQTopic;
13. **public** **class** TopicTest {
14. **public** **static** **void** main(String[] args) **throws** Exception {
15. ActiveMQConnectionFactory factory = **new** ActiveMQConnectionFactory("vm://localhost");
17. Connection connection = factory.createConnection();
18. connection.start();
20. //创建一个Topic
21. Topic topic= **new** ActiveMQTopic("testTopic");
22. Session session = connection.createSession(**false**, Session.AUTO\_ACKNOWLEDGE);
24. //注册消费者1
25. MessageConsumer comsumer1 = session.createConsumer(topic);
26. comsumer1.setMessageListener(**new** MessageListener(){
27. **public** **void** onMessage(Message m) {
28. **try** {
29. System.out.println("Consumer1 get " + ((TextMessage)m).getText());
30. } **catch** (JMSException e) {
31. e.printStackTrace();
32. }
33. }
34. });
36. //注册消费者2
37. MessageConsumer comsumer2 = session.createConsumer(topic);
38. comsumer2.setMessageListener(**new** MessageListener(){
39. **public** **void** onMessage(Message m) {
40. **try** {
41. System.out.println("Consumer2 get " + ((TextMessage)m).getText());
42. } **catch** (JMSException e) {
43. e.printStackTrace();
44. }
45. }
47. });
49. //创建一个生产者，然后发送多个消息。
50. MessageProducer producer = session.createProducer(topic);
51. **for**(**int** i=0; i<10; i++){
52. producer.send(session.createTextMessage("Message:" + i));
53. }
54. }
55. }

运行后得到下面的输出结果：

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. Consumer1 get Message:0
2. Consumer2 get Message:0
3. Consumer1 get Message:1
4. Consumer2 get Message:1
5. Consumer1 get Message:2
6. Consumer2 get Message:2
7. Consumer1 get Message:3
8. Consumer2 get Message:3
9. Consumer1 get Message:4
10. Consumer2 get Message:4
11. Consumer1 get Message:5
12. Consumer2 get Message:5
13. Consumer1 get Message:6
14. Consumer2 get Message:6
15. Consumer1 get Message:7
16. Consumer2 get Message:7
17. Consumer1 get Message:8
18. Consumer2 get Message:8
19. Consumer1 get Message:9
20. Consumer2 get Message:9

说明每一个消息都会被所有的消费者消费。

[深入掌握JMS（六）：消息头](http://blog.csdn.net/zhangxs_3/article/details/4034834)

    一个消息对象分为三部分：消息头(Headers)，属性（Properties）和消息体（Payload）。对于StreamMessage和MapMessage，消息本身就有特定的结构，而对于TextMessage，ObjectMessage和BytesMessage是无结构的。一个消息可以包含一些重要的数据或者仅仅是一个事件的通知。

    消息的Headers部分通常包含一些消息的描述信息，它们都是标准的描述信息。包含下面一些值：

**JMSDestination**   
       消息的目的地，Topic或者是Queue。

**JMSDeliveryMode**   
        消息的发送模式：persistent或nonpersistent。前者表示消息在被消费之前，如果JMS提供者DOWN了，重新启动后消息仍然存在。后者在这种情况下表示消息会被丢失。可以通过下面的方式设置：  
       Producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);

**JMSTimestamp**   
       当调用send()方法的时候，JMSTimestamp会被自动设置为当前事件。可以通过下面方式得到这个值：  
       long timestamp = message.getJMSTimestamp();

**JMSExpiration**  
       表示一个消息的有效期。只有在这个有效期内，消息消费者才可以消费这个消息。默认值为0，表示消息永不过期。可以通过下面的方式设置：  
       producer.setTimeToLive(3600000); //有效期1小时 （1000毫秒 \* 60秒 \* 60分）  
  
　　**JMSPriority**  
       消息的优先级。0-4为正常的优先级，5-9为高优先级。可以通过下面方式设置：  
       producer.setPriority(9);  
  
　　**JMSMessageID**  
       一个字符串用来唯一标示一个消息。  
  
　　**JMSReplyTo**  
       有时消息生产者希望消费者回复一个消息，JMSReplyTo为一个Destination，表示需要回复的目的地。当然消费者可以不理会它。  
  
　　**JMSCorrelationID**  
       通常用来关联多个Message。例如需要回复一个消息，可以把JMSCorrelationID设置为所收到的消息的JMSMessageID。  
  
　　**JMSType**  
       表示消息体的结构，和JMS提供者有关。  
  
　　**JMSRedelivered**  
       如果这个值为true，表示消息是被重新发送了。因为有时消费者没有确认他已经收到消息或者JMS提供者不确定消费者是否已经收到。  
  
    除了Header，消息发送者可以添加一些属性(Properties)。这些属性可以是应用自定义的属性，JMS定义的属性和JMS提供者定义的属性。我们通常只适用自定义的属性。  
  
    后面会讲到这些Header和属性的用法。

[深入掌握JMS（七）：DeliveryMode例子](http://blog.csdn.net/zhangxs_3/article/details/4034837)

在下面的例子中，分别发送一个Persistent和nonpersistent的消息，然后关闭退出JMS。

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. **import** javax.jms.Connection;
2. **import** javax.jms.DeliveryMode;
3. **import** javax.jms.MessageProducer;
4. **import** javax.jms.Queue;
5. **import** javax.jms.Session;
6. **import** org.apache.activemq.ActiveMQConnectionFactory;
7. **import** org.apache.activemq.command.ActiveMQQueue;
8. **public** **class** DeliveryModeSendTest {
9. **public** **static** **void** main(String[] args) **throws** Exception {
10. ActiveMQConnectionFactory factory = **new** ActiveMQConnectionFactory("vm://localhost");
12. Connection connection = factory.createConnection();
13. connection.start();
15. Queue queue = **new** ActiveMQQueue("testQueue");
16. Session session = connection.createSession(**false**, Session.AUTO\_ACKNOWLEDGE);
18. MessageProducer producer = session.createProducer(queue);
19. producer.setDeliveryMode(DeliveryMode.PERSISTENT);
20. producer.send(session.createTextMessage("A persistent Message"));
22. producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);
23. producer.send(session.createTextMessage("A non persistent Message"));
25. System.out.println("Send messages sucessfully!");
26. }
27. }

   运行上面的程序，当输出“Send messages sucessfully!”时，说明两个消息都已经发送成功，然后我们结束它，来停止JMS Provider。  
  
    接下来我们重新启动JMS Provicer，然后添加一个消费者：

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. **import** javax.jms.Connection;
2. **import** javax.jms.JMSException;
3. **import** javax.jms.Message;
4. **import** javax.jms.MessageConsumer;
5. **import** javax.jms.MessageListener;
6. **import** javax.jms.Queue;
7. **import** javax.jms.Session;
8. **import** javax.jms.TextMessage;
9. **import** org.apache.activemq.ActiveMQConnectionFactory;
10. **import** org.apache.activemq.command.ActiveMQQueue;
11. **public** **class** DeliveryModeReceiveTest {
12. **public** **static** **void** main(String[] args) **throws** Exception {
13. ActiveMQConnectionFactory factory = **new** ActiveMQConnectionFactory("vm://localhost");
15. Connection connection = factory.createConnection();
16. connection.start();
18. Queue queue = **new** ActiveMQQueue("testQueue");
19. Session session = connection.createSession(**false**, Session.AUTO\_ACKNOWLEDGE);
21. MessageConsumer comsumer = session.createConsumer(queue);
22. comsumer.setMessageListener(**new** MessageListener(){
23. **public** **void** onMessage(Message m) {
24. **try** {
25. System.out.println("Consumer get " + ((TextMessage)m).getText());
26. } **catch** (JMSException e) {
27. e.printStackTrace();
28. }
29. }
30. });
31. }
32. }

运行上面的程序，可以得到下面的输出结果：  
  
Consumer get A persistent Message  
  
可以看出消息消费者只接收到一个消息，它是一个Persistent的消息。而刚才发送的non persistent消息已经丢失了。  
  
另外, 如果发送一个non persistent消息, 而刚好这个时候没有消费者在监听, 这个消息也会丢失.

[深入掌握**JMS**（八）：**JMSReplyTo**](http://blog.csdn.net/zhangxs_3/article/details/4034847)

在下面的例子中，首先创建两个**Queue**，发送者给一个**Queue**发送，接收者接收到消息之后给另一个**Queue**回复一个**Message**，然后再创建一个消费者来接受所回复的消息。

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. **import javax.jms.Connection;**
2. **import javax.jms.JMSException;**
3. **import javax.jms.Message;**
4. **import javax.jms.MessageConsumer;**
5. **import javax.jms.MessageListener;**
6. **import javax.jms.MessageProducer;**
7. **import javax.jms.Queue;**
8. **import javax.jms.Session;**
9. **import javax.jms.TextMessage;**
10. **import org.apache.activemq.ActiveMQConnectionFactory;**
11. **import org.apache.activemq.command.ActiveMQQueue;**
12. **public class MessageSendReceiveAndReply {**
13. **public static void main(String[] args) throws Exception {**
14. **ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("vm://localhost");**
16. **Connection connection = factory.createConnection();**
17. **connection.start();**
19. **//**消息发送到这个**Queue**
20. **Queue queue = new ActiveMQQueue("testQueue");**
21. **//**消息回复到这个**Queue**
22. **Queue replyQueue = new ActiveMQQueue("replyQueue");**
24. **final Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);**
25. **//**创建一个消息，并设置它的**JMSReplyTo**为**replyQueue**。
26. **Message message = session.createTextMessage("Andy");**
27. **message.setJMSReplyTo(replyQueue);**
29. **MessageProducer producer = session.createProducer(queue);**
30. **producer.send(message);**
32. **//**消息的接收者
33. **MessageConsumer comsumer = session.createConsumer(queue);**
34. **comsumer.setMessageListener(new MessageListener(){**
35. **public void onMessage(Message m) {**
36. **try {**
37. **//**创建一个新的**MessageProducer**来发送一个回复消息。
38. **MessageProducer producer = session.createProducer(m.getJMSReplyTo());**
39. **producer.send(session.createTextMessage("Hello " + ((TextMessage) m).getText()));**
40. **} catch (JMSException e1) {**
41. **e1.printStackTrace();**
42. **}**
43. **}**
45. **});**
47. **//**这个接收者用来接收回复的消息
48. **MessageConsumer comsumer2 = session.createConsumer(replyQueue);**
49. **comsumer2.setMessageListener(new MessageListener(){**
50. **public void onMessage(Message m) {**
51. **try {**
52. **System.out.println(((TextMessage) m).getText());**
53. **} catch (JMSException e) {**
54. **e.printStackTrace();**
55. **}**
56. **}**
57. **});**
58. **}**
59. **}**

首先消息生产者发送一个消息，内容为**“Andy”**，然后消费者收到这个消息之后根据消息的**JMSReplyTo**，回复一个消息，内容为**“Hello Andy‘**。最后在回复的**Queue**上创建一个接收回复消息的消费者，它输出所回复的内容。运行上面的程序，可以得到下面的输出结果：

**[java]** [view plain](http://blog.csdn.net/hanjiaming/article/details/7821160) [copy](http://blog.csdn.net/hanjiaming/article/details/7821160)

1. Hello Andy

<http://blog.csdn.net/hanjiaming/article/details/7821160>

<http://blog.csdn.net/zhangxs_3/article/details/4034713>