转载：[http://www.cnblogs.com/jjj250/archive/2012/08/08/2628552.html](http://www.cnblogs.com/jjj250/archive/2012/08/08/2628552.html" \t "_blank)

[JMS入门教程](http://blog.csdn.net/zhangzikui/article/details/24837999)

**基础篇**

JMS是应用系统或组件之间相互通信的应用程序接口，利用它，我们可以轻易实现在不同JVM之间相互的远程通信。要实现远程通信，RPC同样也能做到，但RPC却不可避免地增加了不同系统之间的耦合度，JMS能极大地降低不同的应用系统之间的耦合。

要学习JMS，有几个概念必须要搞清楚：

      Messaging (消息通知、消息通信)

一种应用系统或组件之间相互通信的方式。

      Message (消息)

消息即为消息通信的载体，消息包括Message Headers, Message properties, Message bodies

      JMS有两种方式进行消息通信：Point-to-Point (P2P) 和 Publish/Subscriber (PUB/SUB)

P2P方式是一对一的，一条消息只有一个接收者，默认情况下P2P消息是持久的，也就是说发送者（sender）产生的一条消息（message）发送到消息队列（queue）之上后，只有等到消息接收者（receiver）接收到它，才会从消息队列中删除，没有被接收的消息会一直存在JMS容器里（jms容器主要管理jms）。这种方式有点像邮政通信，信件只有一个接收者，信件在接收之前，会一直存放在信箱里。

PUB/SUB方式的工作流程，首先subscriber（订阅者）向JMS容器订阅(Listen to)自己感兴趣的topic（主题），多个订阅者可以同时对一个主题进行订阅，消息发布者发布一条消息，所有订阅了该主题的订阅者都能收到这个消息。默认情况下，pub/sub方式下的消息不是持久的，这意味着，消息一经发出，不管有没有人接收，都不会保存下来，而且订阅者只能接收到自已订阅之后发布者发出的消息。这种方式有点像订阅报刊杂志，一种报刊可以有多人同时订阅，但订阅者只能收到开始订阅之后的报社发行的期刊。

      JMS（[**Java**](http://lib.csdn.net/base/java) Messaging Service）

是**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "_blank)**EE中的一种技术，它定义一套完整的接口（me：之所以编写了这个接口就是为了统一，以便于可移植，类似面向接口编程），来实现不同系统或应用之间的消息通信。这意味着：我们针对JMS接口编写的应用程序（客户程序），在任何一个实现了标准JMS接口的容器下都能运行起来，我们的应用程序与容器实现了真正的解耦，这也就是面向接口编程的好处之一吧。这点类似JDBC编程。

      JMS提供者（JMS Provider）

JMS提供者，也叫JMS服务器或JMS容器，也就是JMS服务的提供者，主流的J2EE容器一般都提供JMS服务（比如JBoss，BEA WebLogic，IBM WebSphere，**[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "_blank)** Application Server等都支持）

      连接工厂（Connection Factories）

连接工厂是用来创建客户端到JMS容器之间JMS连接的工厂，连接工厂有两种：(QueueConnectionFactory和TopicConnectionFactory)，分别用来创建QueueConnection 和 TopicConnection的。

Context ctx = new InitialContext();

QueueConnectionFactory queueConnectionFactory =

                    (QueueConnectionFactory) ctx.lookup("QueueConnectionFactory");

TopicConnectionFactory topicConnectionFactory =

                    (TopicConnectionFactory) ctx.lookup("TopicConnectionFactory");

      目的地（Destinations）

目的地是消息生产者(producer)消息发住的目的地，也是消费者(consumer)接收消息的来源地，它有点像信箱，邮递员把信件投往信箱，收件人从信箱取信件。对P2P方式来说，目的地就是Queue，对pub/sub方式来说，目的地就是Topic。我们要得到这个目的地的引用，只能通过JNDI查找(lookup)的方式得到，因为目的地是注册在JMS服务器的（后面的章节会讲到如何注册一个目的地）

Topic myTopic = (Topic) ctx.lookup("MyTopic");

Queue myQueue = (Queue) ctx.lookup("MyQueue");

      连接（Connection）

这里说的连接是指客户端与JMS提供者（容器）之间的连接。连接也分两种：QueueConnection和TopicConnection，分别对应于P2P连接和Pub/Sub连接。

QueueConnection queueConnection = queueConnectionFactory.createQueueConnection();

TopicConnection topicConnection = topicConnectionFactory.createTopicConnection();

连接用完之后必须记得关闭，否则连接资源不会被释放掉。关闭连接的同时会自动把会话、产生者、消费者都关闭掉。

      会话（Session）

会话是用来创建消息产生者和消息消费者的单线程环境，你可以它来创建消息生产者、消费者、消息，用它来维持消息监听。

TopicSession topicSession = topicConnection.createTopicSession(false, Session.AUTO\_ACKNOWLEDGE);

QueueSession queueSession = queueConnection.createQueueSession(true, 0);

      消息生产者（Message Producers）

消息生产者也就是消息的产生者或发送者，在P2P方式下它是**QueueSender**，在Pub/Sub方式下它是**TopicPublisher**。它是一个由session创建的，用来把把消息发送到目的地的对象。

QueueSender queueSender = queueSession.createSender(myQueue);

TopicPublisher topicPublisher = topicSession.createPublisher(myTopic);

一旦你创建好生产者，你就可以用它来发送消息

queueSender.send(message);

topicPublisher.publish(message);

      消息消费者（Message Consumer）

消息消费者也就是消息的接收者或使用者，在P2P方式下这是**QueueReceiver**，在Pub/Sub方式下它是**TopicSubscriber**。这是一个由session来创建的，用来接收来自目的地消息的对象。JMS容器来负责把消息从目的地投递到注册了该目的地的消息消费者。

QueueReceiver queueReceiver = queueSession.createReceiver(myQueue);

TopicSubscriber topicSubscriber = topicSession.createSubscriber(myTopic);

一旦创建好消息消费者，它就是活动的，你可以用它来接收消息，你也可以用close()方法来使它失效（Inactive）。当你调用Connection的start()方法之前，消费者是不会接收到任何消息的。两种接收者都有一个receive方法，这是一个同步的方法，也就是说程序执行到这个方法会被阻塞，直到收到消息为止。

queueConnection.start();

Message m = queueReceiver.receive();

topicConnection.start();

Message m = topicSubscriber.receive(1000); // time out after a second

如果我们不想它被阻塞，就需要异步的接收消息，这时我们得用消息临听器（Message Listener）了。

      消息监听器（Message Listener）

消息监听器是一个充当消息的异步事件处理器的对象，它实现了MessageListener接口，这个接口只有一个方法onMessage，在这个方法里，你可以定义当接收到消息之后的要做的操作。你可以用setMessageListener方法为消息消费者注册一个监听器。

MessageListener listener = new MessageListener( {

public void onMessage(Message msg) {

    //

}

});

topicSubscriber.setMessageListener(listener); //注册监听

topicConnection.start();

有一点要注意，如果你先调用Connection的start，然后再调用setMessageListener，消息很可能接收不到，正确的做法是先注册监听，再启动Connection。

注册监听之后，一旦JMS容器有消费投递过来，消息消费（接收）者就会自动调用监听器的onMessage方法。这个方法的带有一个参数Message，这就接收到的消息。

      消息选择器（Message Selectors）

假如你只需要一个过滤器来过滤收到的消息，那么你可以使用消息选择器，它允许消费者指定只对特定的消息感兴趣。消息选择器只能是工作在JMS容器的，而不是我们的应用程序上。消息选择器是一个包含一个表达式的字符串，这个表达式的语法类似SQL的条件表达式，在createReceiver, createSubscriber这些方法里有一个参数让你指定一个消息选择器，由这些方法创建的消费者就只能收到与消息选择器匹配的消息了。

      消息（Messages）

JMS消息包括三个部分：消息头（Header），属性（Properties），消息体（Body）

其中消息头是必须的，后两个是可选的。

1）消息头里你可以指定JMSMessageID, JMSCorrelationID, JMSReplyTo, JMSType等信息。

2）属性指定一些消息头没有包括的附加信息，比如可以在属性里指定消息选择器。

3）消息体是消息的内容，有5种消息类型：TextMessage，MapMessage，BytesMessage，StreamMessage，ObjectMessage=-

TextMessage message = queueSession.createTextMessage();

message.setText(msg\_text);     // msg\_text is a String

queueSender.send(message);

在消费者端，接收到的总是一个通用的Message对象，你需要把它转型成特定的类型才能提取出里面的内容。

Message m = queueReceiver.receive();

if (m instanceof TextMessage) {

    TextMessage message = (TextMessage) m;

    System.out.println("Reading message: " + message.getText());

} else {

    // Handle error}

**实战篇**

前面对JMS概念的作了一个基本介绍，下面我们看一个具体的例子程序

Pub/sub方式的消息传递的例子:

         HelloPublisher.java

**package** com.jms.test;

**import** java.util.Hashtable;

**import** javax.jms.JMSException;

**import** javax.jms.Session;

**import** javax.jms.TextMessage;

**import** javax.jms.Topic;

**import** javax.jms.TopicConnection;

**import** javax.jms.TopicConnectionFactory;

**import** javax.jms.TopicPublisher;

**import** javax.jms.TopicSession;

**import** javax.naming.Context;

**import** javax.naming.InitialContext;

**import** javax.naming.NamingException;

/\*\*

 \* pub/sub方式的消息发送程序

 \*/

**public class** HelloPublisher {

    TopicConnection topicConnection;// JMS连接，属于Pub/Sub方式的连接

    TopicSession topicSession; //JMS会话，属于Pub/Sub方式的会话

    TopicPublisher topicPublisher; //消息发布者

    Topic topic; //主题

**public** HelloPublisher(String factoryJNDI, String topicJNDI)

**throws** JMSException, NamingException {

       Hashtable<String, String> env = **new** Hashtable<String, String>();

       //设置好连接JMS容器的属性，不同的容器需要的属性可能不同，需要查阅相关文档

       env.put(Context.*INITIAL\_CONTEXT\_FACTORY*,

              "org.jnp.interfaces.NamingContextFactory");

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

       env.put("java.naming.rmi.security.manager", "yes");

       env.put(Context.*URL\_PKG\_PREFIXES*, "org.jboss.naming");

       //创建连接JMS容器的上下文(context)

       Context context = **new** InitialContext(env);

       //通过连接工厂的JNDI名查找ConnectionFactory

       TopicConnectionFactory topicFactory =

           (TopicConnectionFactory) context.lookup(factoryJNDI);

       //用连接工厂创建一个JMS连接

       topicConnection = topicFactory.createTopicConnection();

       //通过JMS连接创建一个Session

       topicSession = topicConnection.createTopicSession(**false**,

              Session.*AUTO\_ACKNOWLEDGE*);

       //通过上下文查找到一个主题(topic)

       topic = (Topic) context.lookup(topicJNDI);

       //用session来创建一个特定主题的消息发送者

       topicPublisher = topicSession.createPublisher(topic);

    }

    /\*\*

     \* 发布一条文本消息

     \* **@param** msg 待发布的消息

     \* **@throws** JMSException

     \*/

**public void** publish(String msg) **throws** JMSException {

       //用session来创建一个文本类型的消息

       TextMessage message = topicSession.createTextMessage();

       message.setText(msg);//设置消息内容

       topicPublisher.publish(topic, message);//消息发送，发送到特定主题

    }

**public void** close() **throws** JMSException {

       topicSession.close();//关闭session

       topicConnection.close();//关闭连接

    }

**public static void** main(String[] args)

**throws** JMSException, NamingException {

       HelloPublisher publisher =

**new** HelloPublisher("ConnectionFactory", "topic/testTopic");

**try** {

**for** (**int** i = 1; i < 11; i++) {

              String msg = "Hello World no. " + i;

              System.*out*.println("Publishing message: " + msg);

              publisher.publish(msg);

           }

           publisher.close();//session和connection用完之后一定记得关闭

       } **catch** (Exception ex) {

           ex.printStackTrace();

       }

    }

}

程序在控制台输出：

Publishing message: Hello World no. 1

Publishing message: Hello World no. 2

Publishing message: Hello World no. 3

Publishing message: Hello World no. 4

Publishing message: Hello World no. 5

Publishing message: Hello World no. 6

Publishing message: Hello World no. 7

Publishing message: Hello World no. 8

Publishing message: Hello World no. 9

Publishing message: Hello World no. 10

        HelloSubscriber.java

**package** com.jms.test;

**import** javax.jms.JMSException;

**import** javax.jms.Message;

**import** javax.jms.MessageListener;

**import** javax.jms.Session;

**import** javax.jms.TextMessage;

**import** javax.jms.Topic;

**import** javax.jms.TopicConnection;

**import** javax.jms.TopicConnectionFactory;

**import** javax.jms.TopicSession;

**import** javax.jms.TopicSubscriber;

**import** javax.naming.Context;

**import** javax.naming.InitialContext;

**import** javax.naming.NamingException;

/\*\*

 \* pub/sub方式下的消息接收器。注意，这个消息接收器可以与上面的消息发送器可以工作

\* 在不同的JVM中，只要保证它们各自能够连通JMS容器(JMS Provider)

 \*

 \*/

**public class** HelloSubscriber **implements** MessageListener {

    TopicConnection topicConnection;

    TopicSession topicSession;

    TopicSubscriber topicSubscriber;

    Topic topic;

**public** HelloSubscriber(String factoryJNDI, String topicJNDI)

**throws** JMSException, NamingException {

Hashtable<String, String> env = **new** Hashtable<String, String>();

       //设置好连接JMS容器的属性，不同的容器需要的属性可能不同，需要查阅相关文档

       env.put(Context.*INITIAL\_CONTEXT\_FACTORY*,

              "org.jnp.interfaces.NamingContextFactory");

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

       env.put("java.naming.rmi.security.manager", "yes");

       env.put(Context.*URL\_PKG\_PREFIXES*, "org.jboss.naming");

       Context context = **new** InitialContext();

       TopicConnectionFactory topicFactory =

           (TopicConnectionFactory) context.lookup(factoryJNDI);

       //创建连接

       topicConnection = topicFactory.createTopicConnection();

       topicSession = topicConnection.createTopicSession(**false**,

              Session.*AUTO\_ACKNOWLEDGE*);//创建session

       topic = (Topic) context.lookup(topicJNDI);//查找到主题

       //用session创建一个特定queue的消息接收者

       topicSubscriber = topicSession.createSubscriber(topic);

       //注册监听，这里设置的监听是自己，因为本类已经实现了MessageListener接口，

       //一旦queueReceiver接收到了消息，就会调用本类的onMessage方法

       topicSubscriber.setMessageListener(**this**);

       System.*out*.println("HelloSubscriber subscribed to topic: "

              + topicJNDI);

       topicConnection.start();//启动连接，这时监听器才真正生效

    }

**public void** onMessage(Message msg) {

**try** {

**if** (msg **instanceof** TextMessage) {

              //把Message 转型成 TextMessage 并提取消息内容

              String msgTxt = ((TextMessage) msg).getText();

              System.*out*.println("HelloSubscriber got message: " +

                  msgTxt);

           }

       } **catch** (JMSException ex) {

           System.*err*.println("Could not get text message: " + ex);

           ex.printStackTrace();

       }

    }

**public void** close() **throws** JMSException {

       topicSession.close();

       topicConnection.close();

    }

**public static void** main(String[] args) {

**try** {

**new** HelloSubscriber("TopicConnectionFactory",

              "topic/testTopic");

       } **catch** (Exception ex) {

           ex.printStackTrace();

       }

    }

}

程序在控制台输出：

HelloSubscriber subscribed to topic: topic/testTopic

HelloSubscriber got message: Hello World no. 1

HelloSubscriber got message: Hello World no. 2

HelloSubscriber got message: Hello World no. 3

HelloSubscriber got message: Hello World no. 4

HelloSubscriber got message: Hello World no. 5

HelloSubscriber got message: Hello World no. 6

HelloSubscriber got message: Hello World no. 7

HelloSubscriber got message: Hello World no. 8

HelloSubscriber got message: Hello World no. 9

HelloSubscriber got message: Hello World no. 10

P2P方式下的消息传递

         HelloQueue.java

**package** com.jms.test;

**import** javax.naming.Context;

**import** javax.naming.InitialContext;

**import** javax.naming.NamingException;

**import** javax.jms.QueueConnectionFactory;

**import** javax.jms.QueueConnection;

**import** javax.jms.QueueSession;

**import** javax.jms.QueueSender;

**import** javax.jms.Queue;

**import** javax.jms.TextMessage;

**import** javax.jms.Session;

**import** javax.jms.JMSException;

**import** java.util.Hashtable;

**public class** HelloQueue {

    QueueConnection queueConnection; //queue方式的JMS连接

    QueueSession queueSession; //queue会话

    QueueSender queueSender; //queue消息发送者

    Queue queue; //消息队列

**public** HelloQueue(String factoryJNDI, String topicJNDI)

**throws** JMSException, NamingException {

        //连接JMS Provider的环境参数

        Hashtable<String, String> props = **new** Hashtable<String, String>();

        props.put(Context.*INITIAL\_CONTEXT\_FACTORY*,

                "org.jnp.interfaces.NamingContextFactory");

        //JMS provider的主机和端口

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

        props.put("java.naming.rmi.security.manager", "yes");

        props.put(Context.*URL\_PKG\_PREFIXES*, "org.jboss.naming");

        Context context = **new** InitialContext(props);

        //lookup到连接工厂

        QueueConnectionFactory queueFactory =

            (QueueConnectionFactory) context.lookup(factoryJNDI);

        queueConnection = queueFactory.createQueueConnection();//创建连接

        queueSession = queueConnection.createQueueSession(**false**,

                Session.*AUTO\_ACKNOWLEDGE*);//创建会话

        queue = (Queue) context.lookup(topicJNDI);//lookup到特定的消息队列

        queueSender = queueSession.createSender(queue);//创建队列消息的发送者

    }

**public void** send(String msg) **throws** JMSException {

        TextMessage message = queueSession.createTextMessage();

        message.setText(msg);

        queueSender.send(queue, message);

    }

**public void** close() **throws** JMSException {

        queueSession.close();

        queueConnection.close();

    }

**public static void** main(String[] args) {

**try** {

            HelloQueue queue = **new** HelloQueue("ConnectionFactory",

                    "queue/testQueue");

**for** (**int** i = 11; i < 21; i++) {

                String msg = "Hello World no. " + i;

                System.*out*.println("Hello Queue Publishing message: " + msg);

                queue.send(msg);

            }

            queue.close();

        } **catch** (Exception ex) {

            System.*err*.println("An exception occurred " +

"while testing HelloPublisher25: " + ex);

            ex.printStackTrace();

        }

    }

}

程序在控制台输出：

Hello Queue Publishing message: " Hello World no. 11

Hello Queue Publishing message: " Hello World no. 12

Hello Queue Publishing message: " Hello World no. 13

Hello Queue Publishing message: " Hello World no. 14

Hello Queue Publishing message: " Hello World no. 15

Hello Queue Publishing message: " Hello World no. 16

Hello Queue Publishing message: " Hello World no. 17

Hello Queue Publishing message: " Hello World no. 18

Hello Queue Publishing message: " Hello World no. 19

Hello Queue Publishing message: " Hello World no. 20

        HelloRecvQueue.java

**package** com.jms.test;

**import** javax.jms.JMSException;

**import** javax.jms.Message;

**import** javax.jms.MessageListener;

**import** javax.jms.Session;

**import** javax.jms.TextMessage;

**import** javax.jms.Queue;

**import** javax.jms.QueueConnection;

**import** javax.jms.QueueConnectionFactory;

**import** javax.jms.QueueSession;

**import** javax.jms.QueueReceiver;

**import** javax.naming.Context;

**import** javax.naming.InitialContext;

**import** javax.naming.NamingException;

**public class** HelloRecvQueue **implements** MessageListener {

    QueueConnection queueConnection;

    QueueSession queueSession;

    QueueReceiver queueReceiver;

    Queue queue;

**public** HelloRecvQueue(String factoryJNDI, String topicJNDI)

**throws** JMSException, NamingException {

        Context context = **new** InitialContext();

        QueueConnectionFactory queueFactory =

            (QueueConnectionFactory) context.lookup(factoryJNDI);

        queueConnection = queueFactory.createQueueConnection();

        queueSession = queueConnection.createQueueSession(**false**,

                Session.*AUTO\_ACKNOWLEDGE*);

        queue = (Queue) context.lookup(topicJNDI);

        queueReceiver = queueSession.createReceiver(queue);

        queueReceiver.setMessageListener(**this**);

        System.*out*.println("HelloReceQueue receiver to queue: " + topicJNDI);

        queueConnection.start();

    }

**public void** onMessage(Message m) {

**try** {

            String msg = ((TextMessage) m).getText();

            System.*out*.println("HelloReceQueue got message: " + msg);

        } **catch** (JMSException ex) {

            System.*err*.println("Could not get text message: " + ex);

            ex.printStackTrace();

        }

    }

**public void** close() **throws** JMSException {

        queueSession.close();

        queueConnection.close();

    }

    Public ovid main(String[] args) {

    new HelloRecvQueue();

}

}

程序在控制台输出：

HelloReceQueue got message: Hello World no. 11

HelloReceQueue got message: Hello World no. 12

HelloReceQueue got message: Hello World no. 13

HelloReceQueue got message: Hello World no. 14

HelloReceQueue got message: Hello World no. 15

HelloReceQueue got message: Hello World no. 16

HelloReceQueue got message: Hello World no. 17

HelloReceQueue got message: Hello World no. 18

HelloReceQueue got message: Hello World no. 19

HelloReceQueue got message: Hello World no. 20

**配置篇**

下面我们来看看是JMS是在JBoss下如何配置的，首先JMS需要一个**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**来保存其持久化的消息，幸运的是JBoss自带有一个开源的JAVA数据库HSQL（[www.hsqldb.org](http://www.hsqldb.org/)）

在这里简单地介绍一下这个数据库，它支持标准的SQL语法和JDBC接口，是一个用纯JAVA编写的数据库，其实它只有一个jar文件而已：hsqldb.jar，在%JBOSS\_HOME%/server/default/lib目录下你能找到它。

启动这个数据库有三种模式：Server模式、进程模式和内存模式,在Server模式下，你可以用下面的命令让它启动起来：

$cd %JBOSS\_HOME%/server/default/lib

$ java -cp hsqldb.jar org.hsqldb.Server -database.0 mydb -dbname.0 demoDB

其中mydb是数据库名，demoDB是数据库别名，我们用JDBC连它是就用这个别名,用户名是sa,密码默认是空,我们下列语句就能创建表、插入数据了

create table employee (

  employee\_id int,

  employee\_name varchar(50),

  age int,

  hiredate date

)

insert into employee values(1, 'linyufa', 33, '2007-12-17')

insert into employee values(2, 'scott', 25, '2008-11-23')

insert into employee values(3, 'larry', 35, '2004-11-23')

想进一步了解HSQL的知识，网上有很多学习资料，好了，回到我们讨论的JMS话题，有了这个数据库，那我们就不必去找其他数据库了，JMS默认是用内存模式来启动它的，所以我们基本上不用去关心它是如何工作的。

1)   在 %JBOSS\_HOME%/server/default/deploy/jms目录下，

打开hsqldb-jdbc-state-service.xml文件，

<depends optional-attribute-name="ConnectionManager">

            jboss.jca:service= DataSourceBinding, name=**DefaultDS**

</depends>

**DefaultDS**这个名字就是JMS连接数据库的数据源，可以让其保持默认值。

2)   再在同一目录打开hsqldb-jdbc2-service.xml 文件，

<depends optional-attribute-name="ConnectionManager">

jboss.jca:service=DataSourceBinding,name=**DefaultDS**

      </depends>

**DefaultDS**这个名字保持和前面一致即可，也可以让其保持默认值。

3)   在同一目录打开jbossmq-destinations-service.xml文件，找到下面的代码段：

<mbean code="org.jboss.mq.server.jmx.Topic"

    name="jboss.mq.destination:service=Topic,name=**testTopic**">

    <depends optional-attribute-name="DestinationManager">

       jboss.mq:service=DestinationManager

    </depends>

    <depends optional-attribute-name="SecurityManager">

       jboss.mq:service=SecurityManager

    </depends>

    <attribute name="SecurityConf">

       <security>

       <role name="guest" read="true" write="true"/>

       <role name="publisher" read="true" write="true" create="false"/>

       <role name="durpublisher" read="true" write="true" create="true"/>

       </security>

    </attribute>

</mbean>

这是定义一个名叫testTopic的示例，如果你要定义一个新的topic，只需要复制这段代码，改一下name属性即可。

同样找到下面这段的代码，这是定义一个名叫testQueue的示例，如果要定义一个新的queue，复制这段代码，改一下名字即可。

<mbean code="org.jboss.mq.server.jmx.Queue"

    name="jboss.mq.destination:service=Queue,name=**testQueue**">

    <depends optional-attribute-name="DestinationManager">

       jboss.mq:service=DestinationManager

    </depends>

    <depends optional-attribute-name="SecurityManager">

        jboss.mq:service=SecurityManager

    </depends>

    <attribute name="MessageCounterHistoryDayLimit">-1</attribute>

    <attribute name="SecurityConf">

      <security>

       <role name="guest" read="true" write="true"/>

       <role name="publisher" read="true" write="true" create="false"/>

       <role name="noacc" read="false" write="false" create="false"/>

      </security>

    </attribute>

</mbean>

4)   启动Jboss后在控制台看到如下输出，即说明JMS正常启动了

09:50:28,390 INFO  [A] Bound to JNDI name: queue/A

09:50:28,406 INFO  [B] Bound to JNDI name: queue/B

09:50:28,406 INFO  [C] Bound to JNDI name: queue/C

09:50:28,406 INFO  [D] Bound to JNDI name: queue/D

09:50:28,421 INFO  [ex] Bound to JNDI name: queue/ex

09:50:28,437 INFO  [testTopic] Bound to JNDI name: **topic/testTopic**

09:50:28,484 INFO  [securedTopic] Bound to JNDI name: topic/securedTopic

09:50:28,484 INFO  [testDurableTopic] Bound to JNDI name: topic/testDurableTopic

09:50:28,500 INFO  [testQueue] Bound to JNDI name: **queue/testQueue**

5)   如果是Jboss4.2或以上的版本，在启动Jboss时必须指定 –b 0.0.0.0参数，否则本机之外的任何主机都无法连接JMS。可以修改run.bat或run.sh文件，也可以在运行命令时附带上这个参数，如下 sh run.sh –b 0.0.0.0

从上面介绍可以看出，在Jboss下配置JMS是非常简单的，仅需要copy一段代码，改个名字即可。如果在WebLogic下，你就要依次配置JMS Module, ConnectionFactory, Topic, Queue, Template，不过好在console都有向导，非常直观，所以配置起来也不是什么难事。

**JMS编程其他注意事项**

创建一个JMS Connection、查找ConnectionFactory和Destination都是需要很大的系统开销的操作，所以我们的应用程序应避免频繁地去做这些操作。一般情况下，我们可以把ConnectionFactory，Connection, Topic, Queue定义成类的成员变量，并在类的构造函数里初始化他们，避免在每次接收和发送JMS消息时去做这些工作。但是因此也带了一个问题，就是说当Connection不可用了（比如JMS Server重启了），我们的应用程序就会开始不工作了，所以我们要有一种机制去检测我们的Connection是否有效，如果已经断掉，应该试图去重新连接，并通知系统管理员。

JMS的Connection和JDBC的Connection类似，不再使用后应该关闭，不管是正常退出，还是异常退出，否则别的客户程序可能就再也取不到连接了。Session也是如此。

因为JMS工作模式是异步的，我们要意识到调用Connection.start()这个方法，系统已经启动了一个新的线程在工作，也就是说退出了这行语句所在的方法之后，这个线程还在工作，它会不断地去侦听有没有新的JMS消息，直到这个Connection被关闭或不可用。