参考：<http://blog.csdn.net/shandian534/article/details/8372822>

**1、JMS API 体系结构**

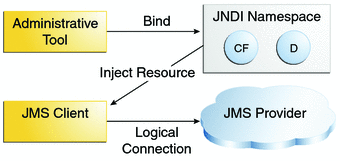
      1)、JMS Provider （JMS提供者）: 实现了JMS API ，提供管理和控制的组件。如[**Java**](http://lib.csdn.net/base/java)EE platform

      2)、JMS Client（JMS客户端）:  发送和接收消息。

      3)、Messages: 消息体。

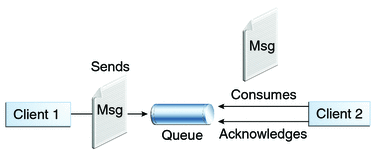
      4)、Administered Object : 管理对象，主要包括 destinations 和 connection factories

      如图：



**2、JMS 类型(Messaging Domain)：PTP和pub/sub**

      1)、PTP（P2P，点对点）：Point to Point  发送者和接收者中有一个消息队列(messages quene),发送者发送消息则把消息加入到队列中，接收者接收消息则把消息从队列中取出；如果接收者没有接收，则这条消息永远保存在队列中(除非已过期)。如下图：



            特性：

             A、每条消息只能有一个接收者

             B、发送者和接收者之间可以异步（不依赖于时间）（no timing dependencies）

             C、接收者成功接收答复机制

       2)、Publish/Subscribe   发送者把消息挂在一个主题下（类似电子公告板），接收者先订阅这个主题，当这个主题有新消息发布时，接收者就可以接收这个主题下的消息了，这个消息一直保持到所有订阅这个消息的人（在线的）都接收了才删除。如图：



            特性：

             A、一条消息可以有多个接收者接收

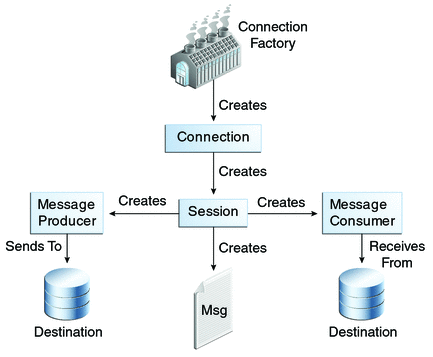
             B、接收者和发送者之间必须同步。

为了弥补这种类型的时间依赖（timing dependencies）劣势,JMS API 提供了创建持久订阅的机制（可靠但是影响性能），这样不管接收者是否在线，发送者都可以发送，接收者也可以接收。

**3、JMS API 程序设计模型，包括：**

* Administered objects: connection factories and destinations
* Connections
* Sessions
* Message producers
* Message consumers
* Messages

   如图：



    1)、JMS Connection Factories

       封装了使用者所有的配置，用来创建Connection 的对象，可以是ConnectionFactory,QueueConnectionFactory, or TopicConnectionFactory的一个实例。可以用JNDI命名空间来管理connection Factories

如：

@Resource(lookup = "jms/ConnectionFactory")

private static ConnectionFactory connectionFactory;

       2)、JMS Destinations

         消息发送的目的地和来源地，在PTP类型中，Destination 是quene,而在pub/sub类型中，Destination是topic,一个JMS应用可以有多个quenes或者topics。和connection factory一样也可以用JNDI命名空间来管理Destinations,如：

@Resource(lookup = "jms/Queue")

private static Queue queue;

@Resource(lookup = "jms/Topic")

private static Topic topic;

     3)、JMS Connections

Connections其实就是一个虚拟的TCP/IP链接，在客户端和JMS提供者之间建立链接，通过它来创建Session。在应用关闭之前，你必须关闭connection,否则会造成资源不会释放。

     4)、JMS Session

            session是一个单线程的实例，它可以创建以下对象：

* Message producers
* Message consumers
* Messages
* Queue browsers
* Temporary queues and topics

      session提供了消息事务管理功能，如：

Session session = connection.createSession(false,

Session.AUTO\_ACKNOWLEDGE);

      第一个参数表示非事务性，如果要使创建的消息具有事务性必须用true,第二个参数表示接收消息后会自动答复。

    5)、JMS Message Producers

           用法：

MessageProducer producer = session.createProducer(dest);

MessageProducer producer =session.createProducer(queue);

MessageProducer producer =session.createProducer(topic);

           也要以这样用

MessageProducer anon\_prod =session.createProducer(null);

anon\_prod.send(dest, message);

            在消息发送的时候指定目的地。

     6)、JMS Message Consumers

        用法：

MessageConsumer consumer = session.createConsumer(dest);

MessageConsumer consumer = session.createConsumer(queue);

MessageConsumer consumer = session.createConsumer(topic);

connection.start();

Message m = consumer.receive();

connection.start();

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

**4 、 实例:实现了JMS1.1 和J2EE1.4的开源框架:ActiveMq  (持久化用**[**MySQL**](http://lib.csdn.net/base/mysql)**)**

第一步：下载activemq 5.7.0.zip，解压放在一个目录下，如D:\activemq\_5.7.0

         第二步：持久化配置：在安装目录\conf下找到activemq.xml,

                        加入如下配置：

**[html]** [view plain](http://blog.csdn.net/shandian534/article/details/8372822) [copy](http://blog.csdn.net/shandian534/article/details/8372822)

[print?](http://blog.csdn.net/shandian534/article/details/8372822)

1. **<persistenceAdapter>**
2. **<jdbcPersistenceAdapter** dataDirectory="${activemq.data}" dataSource="#mysql-ds"**/>**
3. **</persistenceAdapter>**

**[html]** [view plain](http://blog.csdn.net/shandian534/article/details/8372822) [copy](http://blog.csdn.net/shandian534/article/details/8372822)

[print?](http://blog.csdn.net/shandian534/article/details/8372822)

1. **<bean** id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-           method="close"**>**
2. **<property** name="driverClassName" value="com.mysql.jdbc.Driver"**/>**
3. **<property** name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"**/>**
4. **<property** name="username" value="root"**/>**
5. **<property** name="password" value="root"**/>**
6. **<property** name="maxActive" value="200"**/>**
7. **<property** name="poolPreparedStatements" value="true"**/>**
8. **</bean>**

         第二步：启动：打开一个cmd窗口，把当前目录切换到activemq的安装目录：D:\activemq\_5.7.0

                        进入bin目录，输入activemq命令回车 OK了（要先配置好[**Java**](http://lib.csdn.net/base/java) 的环境变量java\_home、path、

                         classpath）,在浏览器窗口中输入<http://127.0.0.1:8161/admin/>**,**出现activemq的后台管理界

                                          面，表示安装成功！

第三步：在eclipse 创建一个web project testActiveMq,同时把activemq-all-5.7.0.jar

                        (在activemq的安装目录下)导入到项目中

                        1)、在com.jms.servlet包下创建MyPublish类

**[java]** [view plain](http://blog.csdn.net/shandian534/article/details/8372822) [copy](http://blog.csdn.net/shandian534/article/details/8372822)

[print?](http://blog.csdn.net/shandian534/article/details/8372822)

1. **package** com.jms.servlet;


5. **import** java.io.IOException;
7. **import** javax.jms.Connection;
9. **import** javax.jms.ConnectionFactory;
11. **import** javax.jms.DeliveryMode;
13. **import** javax.jms.Destination;
15. **import** javax.jms.JMSException;
17. **import** javax.jms.Message;
19. **import** javax.jms.MessageProducer;
21. **import** javax.jms.Session;
23. **import** javax.naming.Context;
25. **import** javax.naming.InitialContext;
27. **import** javax.naming.NamingException;
29. **import** javax.servlet.ServletException;
31. **import** javax.servlet.http.HttpServlet;
33. **import** javax.servlet.http.HttpServletRequest;
35. **import** javax.servlet.http.HttpServletResponse;
37. **import** org.apache.activemq.ActiveMQConnectionFactory;


41. **public** **class** MyPublish **extends** HttpServlet {


45. **private** **static** **final** **long** serialVersionUID = 8861449351626383534L;
47. **private** InitialContext initialContext;
49. **private** Context context;
51. **private** ConnectionFactory connectionFactory;
53. **private** Connection connection;
55. **private** Session session;
57. **private** Destination destination;
59. **private** MessageProducer messageProducer;


63. **public** **void** init() **throws** ServletException {
65. **try** {
67. //                  initialContext = new InitialContext();
68. //
69. //                  context = (Context) initialContext.lookup("java:comp/env");
70. //
71. //                  connectionFactory = (ConnectionFactory) context
72. //
73. //                              .lookup("jms/NormalConnectionFactory");
74. connectionFactory = **new** ActiveMQConnectionFactory(**null**, **null**, "failover:(tcp://localhost:61616)");
75. connection = connectionFactory.createConnection();
76. session = connection.createSession(**false**, Session.AUTO\_ACKNOWLEDGE);
77. destination = session.createTopic("TOOL.DEFAULT");
78. messageProducer = session.createProducer(destination);
79. } **catch** (JMSException e) {
80. e.printStackTrace();
81. }
82. }


86. **public** **void** destroy() {
87. **super**.destroy();
88. }


92. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
93. **throws** ServletException, IOException {
94. doPost(request, response);
95. }


99. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
100. **throws** ServletException, IOException {
101. String content = request.getParameter("content");
102. **try** {
104. // 设置持久方式
105. messageProducer.setDeliveryMode(DeliveryMode.PERSISTENT);
106. Message testMessage = session.createMessage();
107. // 发布刷新文章消息
108. testMessage.setStringProperty("RefreshArticleId", content);
109. messageProducer.send(testMessage);
110. // 发布刷新帖子消息
111. testMessage.clearProperties();
112. testMessage.setStringProperty("RefreshTopicId", content);
113. messageProducer.send(testMessage);
114. } **catch** (Exception e) {
115. e.printStackTrace();
116. }
118. }
120. }

2)、在com.jms.servlet包下创建JMSListener类

**[java]** [view plain](http://blog.csdn.net/shandian534/article/details/8372822) [copy](http://blog.csdn.net/shandian534/article/details/8372822)

[print?](http://blog.csdn.net/shandian534/article/details/8372822)

1. **package** com.jms.servlet;
3. **import** javax.servlet.\*;
4. **import** javax.servlet.http.\*;
5. **import** javax.naming.\*;
6. **import** javax.jms.\*;
8. **import** org.apache.activemq.ActiveMQConnectionFactory;
10. **public** **class** JMSListener **extends** HttpServlet **implements** MessageListener {
11. **private** **static** **final** **long** serialVersionUID = 5088494289145588596L;
13. **public** **void** init(ServletConfig config) **throws** ServletException {
14. **try** {
15. //            InitialContext initialContext = new InitialContext();
16. //            Context context = (Context) initialContext.lookup("java:comp/env");
17. //            ConnectionFactory connectionFactory = (ConnectionFactory) context
18. //                    .lookup("jms/FailoverConnectionFactory");
19. ActiveMQConnectionFactory connectionFactory = **new** ActiveMQConnectionFactory(**null**, **null**,
20. "failover:(tcp://localhost:61616)");
21. Connection connection = connectionFactory.createConnection();
22. connection.setClientID("MyClient");
23. connection.start();
24. Session session = connection.createSession(**false**,
25. Session.AUTO\_ACKNOWLEDGE);
26. // 普通消息订阅者，无法接收持久消息
27. // Destination destination = (Destination) context
28. // .lookup("jms/topic/MyTopic");
29. // MessageConsumer consumer = session.createConsumer(destination);
30. // 基于Topic创建持久的消息订阅者，前提：Connection必须指定一个唯一的clientId，当前为MyClient
31. // Topic topic = (Topic) context.lookup("jms/topic/MyTopic");
32. Topic topic=session.createTopic("TOOL.DEFAULT");
33. TopicSubscriber consumer = session.createDurableSubscriber(topic,"MySub");
34. consumer.setMessageListener(**this**);
35. } **catch** (JMSException e) {
36. e.printStackTrace();
37. }
38. }

41. **public** **void** onMessage(Message message) {
42. **if** (checkText(message, "RefreshArticleId") != **null**) {
43. String articleId = checkText(message, "RefreshArticleId");
44. System.out.println("refresh article, ID=" + articleId);
45. } **else** **if** (checkText(message, "RefreshTopicId") != **null**) {
46. String topicId = checkText(message, "RefreshTopicId");
47. System.out.println("refresh topic, ID=" + topicId);
48. } **else** {
49. System.out.println("it's normal message, no need to care");
50. }
51. }
53. **private** **static** String checkText(Message m, String s) {
54. **try** {
55. **return** m.getStringProperty(s);
56. } **catch** (JMSException e) {
57. e.printStackTrace(System.out);
58. **return** **null**;
59. }
60. }
61. }

   3)、配置web.xml

在web.xml 增加以下配置

1. **<span** style="font-size:18px;"**>**       **<servlet>**
2. **<servlet-name>**jms-listener**</servlet-name>**
3. **<servlet-class>**com.jms.servlet.JMSListener**</servlet-class>**
4. **<load-on-startup>**1**</load-on-startup>**
5. **</servlet>**
6. **<servlet>**
7. **<servlet-name>**MyPublish**</servlet-name>**
8. **<servlet-class>**com.jms.servlet.MyPublish**</servlet-class>**
9. **</servlet>**
10. **<servlet-mapping>**
11. **<servlet-name>**MyPublish**</servlet-name>**
12. **<url-pattern>**/myPublish.do**</url-pattern>**
13. **</servlet-mapping></span>**

4)、新增页面index.jsp

**[html]** [view plain](http://blog.csdn.net/shandian534/article/details/8372822) [copy](http://blog.csdn.net/shandian534/article/details/8372822)

[print?](http://blog.csdn.net/shandian534/article/details/8372822)

1. **<**%@ page language="java" import="java.util.\*" pageEncoding="utf-8"%**>**
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"**>**
3. **<html>**
4. **<body>**
5. **<form** action="myPublish.do"**>**
6. **<input** type="text" name="content" **/>**
7. **<input** type="submit" value="提交"**>**
8. **</form>**
9. **</body>**
10. **</html>**

 5)、把该项目发布到tomcat6下启动，在浏览器中输入<http://127.0.0.1:8080/testActiveMq/>

输入要发送的消息，如hello,jms 然后在 eclipse 的控制台会看到

refresh article, ID=hello,jms

refresh topic, ID=hello,jms

同时在本地[**数据库**](http://lib.csdn.net/base/mysql)activemq的activemq\_msgs表中会增加两条记录，表示项目[**测试**](http://lib.csdn.net/base/softwaretest)成功！