Apache ActiveMQ 教程

一、特性及优势

1、实现 JMS1.1 规范，支持 J2EE1.4 以上

2、可运行于任何 jvm 和大部分 web 容器（ActiveMQ works great in any JVM）

3、支持多种语言客户端（java, C, C++, AJAX, ACTIONSCRIPT 等等）

4、支持多种协议（stomp，openwire，REST）

5、良好的 spring 支持（ActiveMQ has great Spring Support）

6、速度很快，JBossMQ 的十倍（ActiveMQ is very fast; often 10x faster than JBossMQ.）

7、与 OpenJMS、JbossMQ 等开源 jms provider 相比，ActiveMQ 有 Apache 的支持，持续发展的优势明显。

二、下载部署

1、下载

http://activemq.apache.org/activemq-510-release.html，下载 5.1.0 Windows Distribution版本

2、安装

直接解压至任意目录（如：d:\ apache-activemq-5.1.0）

3、启动 ActiveMQ 服务器

方法 1：

直接运行 bin\activemq.bat

方法 2（在 JVM 中嵌套启动）：

cd example

ant embedBroker

4、ActiveMQ 消息管理后台系统：

http://localhost:8161/admin

三、运行附带的示例程序

1、Queue 消息示例：

* 启动 Queue 消息消费者

  cd example

ant consumer

* 启动 Queue 消息生产者

cd example

ant producer

简要说明：生产者（producer）发消息，消费者（consumer）接消息，发送/接收 2000 个消息后自动关闭

2、Topic 消息示例：

* 启动 Topic 消息消费者

cd example

ant topic-listener

* 启动 Topic 消息生产者

cd example

ant topic-publisher

简要说明：重复 10 轮，publisher 每轮发送 2000 个消息，并等待获取 listener 的处理结果报告，然后进入下一轮发送，最后统计全局发送时间。

四、Queue 与 Topic 的比较

1、JMS Queue 执行 load balancer 语义：

一条消息仅能被一个 consumer 收到。如果在 message 发送的时候没有可用的 consumer，那么它将被保存一直到能处理该 message 的 consumer 可用。如果一个 consumer 收到一条 message 后却不响应它，那么这条消息将被转到另一个 consumer 那儿。一个 Queue 可以有很多 consumer，并且在多个可用的 consumer 中负载均衡。

2、Topic 实现 publish 和 subscribe 语义：

一条消息被 publish 时，它将发到所有感兴趣的订阅者，所以零到多个 subscriber 将接收到消息的一个拷贝。但是在消息代理接收到消息时，只有激活订阅的 subscriber 能够获得消息的一个拷贝。

3、分别对应两种消息模式：

Point-to-Point (点对点),Publisher/Subscriber Model (发布/订阅者)

其中在 Publicher/Subscriber 模式下又有 Nondurable subscription（非持久订阅）和 durable subscription (持久化订阅)2 种消息处理方式。

五、Point-to-Point (点对点)消息模式开发流程

1、生产者（producer）开发流程（ProducerTool.java）：

1.1 创建 Connection：

根据 url，user 和 password 创建一个 jms Connection。

1.2 创建 Session：

在 connection 的基础上创建一个 session，同时设置是否支持事务和 ACKNOWLEDGE 标识。

1.3 创建 Destination 对象：

需指定其对应的主题（subject）名称，producer 和 consumer 将根据 subject 来发送/接收对应的消息。

1.4 创建 MessageProducer：

根据 Destination 创建 MessageProducer 对象，同时设置其持久模式。

1.5 发送消息到队列（Queue）：

封装 TextMessage 消息，使用 MessageProducer 的 send 方法将消息发送出去。

2、消费者（consumer）开发流程（ConsumerTool.java）：

2.1 实现 MessageListener 接口：

消费者类必须实现 MessageListener 接口，然后在 onMessage()方法中监听消息的到达并处理。

2.2 创建 Connection：

根据 url，user 和 password 创建一个 jms Connection，如果是 durable 模式，还需要给 connection 设置一个 clientId。

2.3 创建 Session 和 Destination：

与 ProducerTool.java 中的流程类似，不再赘述。

2.4 创建 replyProducer【可选】：

可以用来将消息处理结果发送给 producer。

2.5 创建 MessageConsumer：

根据 Destination 创建 MessageConsumer 对象。

2.6 消费 message：

在 onMessage()方法中接收 producer 发送过来的消息进行处理，并可以通过 replyProducer 反馈信息给 producer

```
if (message.getJMSReplyTo() != null) {
  replyProducer.send(message.getJMSReplyTo(),
    session.createTextMessage("Reply: " + message.getJMSMessageID()));
}
```

六、Publisher/Subscriber(发布/订阅者)消息模式开发流程

1、订阅者（Subscriber）开发流程（TopicListener.java）：

1.1 实现 MessageListener 接口：

在 onMessage()方法中监听发布者发出的消息队列，并做相应处理。

1.2 创建 Connection：

根据 url，user 和 password 创建一个 jms Connection。

1.3 创建 Session：

在 connection 的基础上创建一个 session，同时设置是否支持事务和 ACKNOWLEDGE 标识。

1.4 创建 Topic：

创建 2 个Topic， topictest.messages用于接收发布者发出的消息，topictest.control用于向发布者发送消息，实现双方的交互。

1.5 创建 consumer 和 producer 对象：

根据topictest.messages创建consumer，根据topictest.control创建producer。

1.6 接收处理消息：

在 onMessage()方法中，对收到的消息进行处理，可直接简单在本地显示消息，或者根据消息内容不同处理对应的业务逻辑（比如：数据库更新、文件操作等等），并且可以使用 producer 对象将处理结果返回给发布者。

2、发布者（Publisher）开发流程（TopicPublisher.java）：

2.1 实现 MessageListener 接口：

在 onMessage()方法中接收订阅者的反馈消息。

2.2 创建 Connection：

根据 url，user 和 password 创建一个 jms Connection。

2.3 创建 Session：

在 connection 的基础上创建一个 session，同时设置是否支持事务和 ACKNOWLEDGE 标识。

2.4 创建 Topic：

创建 2 个Topic，topictest.messages用于向订阅者发布消息，topictest.control用于接收订阅者反馈的消息。这 2 个topic与订阅者开发流程中的topic是一一对应的。

2.5 创建 consumer 和 producer 对象：

根据topictest.messages创建publisher；

根据topictest.control创建consumer，同时监听订阅者反馈的消息。

2.6 给所有订阅者发送消息，并接收反馈消息：

示例代码中，一共重复 10 轮操作。

每轮先向所有订阅者发送 2000 个消息；

然后堵塞线程，开始等待；

最后通过 onMessage()方法，接收到订阅者反馈的“REPORT”类信息后，才 print 反馈信息并解除线程堵塞，进入下一轮。

注：可同时运行多个订阅者测试查看此模式效果

七、ActiveMQ 与 Tomcat 整合

说明：Tomcat 示例版本 6.0.14，其它版本在配置上可能有一些差异

1、准备 jar 包：

将 ActiveMQ lib 目录下的 5 个 jar 包复制到 Tomcat lib 目录下：

activemq-core-5.1.0.jar
activemq-web-5.1.0.jar
geronimo-j2ee-management_1.0_spec-1.0.jar
geronimo-jms_1.1_spec-1.1.1.jar
geronimo-jta_1.0.1B_spec-1.0.1.jar

2、修改配置文件：

2.1 修改 Tomcat 的 conf/context.xml 文件：

在<context></context>节点中添加以下内容：

```
<Resource
```

```
    name="jms/FailoverConnectionFactory"
    auth="Container"
    type="org.apache.activemq.ActiveMQConnectionFactory"
    description="JMS Connection Factory"
    factory="org.apache.activemq.jndi.JNDIReferenceFactory"
    brokerURL="failover:(tcp://localhost:61616)?initialReconnectDelay=100&amp;maxReconnectAttempts=5"
   brokerName="localhost"
    useEmbeddedBroker="false"/>
<Resource
    name="jms/NormalConnectionFactory"
    auth="Container"
    type="org.apache.activemq.ActiveMQConnectionFactory"
    description="JMS Connection Factory"
    factory="org.apache.activemq.jndi.JNDIReferenceFactory"
    brokerURL="tcp://localhost:61616"
   brokerName="localhost"
    useEmbeddedBroker="false"/>
<Resource name="jms/topic/MyTopic"
    auth="Container"
    type="org.apache.activemq.command.ActiveMQTopic"
    factory="org.apache.activemq.jndi.JNDIReferenceFactory"
    physicalName="MY.TEST.FOO"/>

<Resource name="jms/queue/MyQueue"
    auth="Container"
    type="org.apache.activemq.command.ActiveMQQueue"
    factory="org.apache.activemq.jndi.JNDIReferenceFactory"
    physicalName="MY.TEST.FOO.QUEUE"/>
```

配置说明：以 JNDI 的方式定义了 ActiveMQ 的 broker 连接 url、Topic 和 Queue。

此处需**加以注意**的是 Listener 端的 borkerURL 使用了 failover 传输方式：

```
failover:(tcp://localhost:61616)?initialReconnectDelay=100&amp;maxReconnectAttempts=5
```

客户端使用普通传输方式：`tcp://localhost:61616`

failover transport 是一种重新连接机制，用于建立可靠的传输。此处配置的是一旦 ActiveMQ broker 中断，Listener 端将每隔 100ms 自动尝试连接，直至成功连接或重试 5 次连接失败为止。

failover 还支持多个 borker 同时提供服务，实现负载均衡的同时可增加系统容错性，格式：

failover:(uri1,...,uriN)?transportOptions

2.2 新建 web 应用(webapps/jms-test)，修改 WEB-INF/web.xml 文件：

增加一个自启动 Servlet，该 Servlet 实现了 MessageListener 接口，作为 Topic 消息的 Listener 端。

```
<servlet>
    <servlet-name>jms-listener</servlet-name>
    <servlet-class>
       com.flvcd.servlet.JMSListener
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

2.3 修改 activemq.xml 文件：

为了支持持久化消息，需修改 ActiveMQ 的配置文件如下，使用默认的 AMQ Message Store 方式（索引文件方式）存储消息，据官网介绍是快速、稳定的。数据库存储方式可参照官网相关文档。

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost"
persistent="true" useShutdownHook="false">
<persistenceAdapter>
<amqPersistenceAdapter directory="activemq-data" maxFileLength="32mb"/>
</persistenceAdapter>
</broker>
```

3、Listener 端（JMSListener.java）完整实现：

```
package com.flvcd.servlet;

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

import javax.naming.*;

import javax.jms.*;

import org.apache.activemq.ActiveMQConnectionFactory;

public class JMSListener extends HttpServlet implements MessageListener{
```

```
    /** 初始化 jms 连接，创建 topic 监听器 */
    public void init(ServletConfig config) throws
ServletException{
        try {
            InitialContext initCtx = new InitialContext();
            Context envContext = (Context)
initCtx.lookup("java:comp/env");
            ConnectionFactory connectionFactory = (ConnectionFactory)
envContext.lookup("jms/FailoverConnectionFactory");
            Connection connection =
connectionFactory.createConnection();
            connection.setClientID("MyClient");
            Session jmsSession = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            //普通消息订阅者，无法接收持久消息
            //MessageConsumer consumer = jmsSession.createConsumer((Destination)
envContext.lookup("jms/topic/MyTopic"));
            //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
            TopicSubscriber consumer = jmsSession.createDurableSubscriber((Topic)
envContext.lookup("jms/topic/MyTopic"), "MySub");

            consumer.setMessageListener(this);
            connection.start();
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
    /** 接收消息，做对应处理 */
    public void onMessage(Message message) {
```

```
        if (checkText(message, "RefreshArticleId") != null) {
            String articleId = checkText(message, "RefreshArticleId");
            System.out.println("接收刷新文章消息，开始刷新文章 ID=" +
articleId);
        }
        else if (checkText(message, "RefreshThreadId") != null) {
            String threadId = checkText(message, "RefreshThreadId");
            System.out.println("接收刷新论坛帖子消息，开始刷新帖子
ID=" + threadId);
        } else {
            System.out.println("接收普通消息，不做任何处理！");
        }
    }
    private static String checkText(Message m, String s) {
        try {
            return m.getStringProperty(s);
        } catch (JMSException e) {
            e.printStackTrace(System.out);
            return null;
        }
    }
}
```

编译 JMSListener.java 至 classes 目录：

javac

-cp .;D:\apache-tomcat-6.0.14\lib\servlet-api.jar;D:\apache-tomcat-6.0.14\lib\geronimo-jms_1.1_spec-1.1.1.jar;D:\apache-tomcat-6.0.14\lib\activemq-core-5.1.0.jar -d . JMSListener.java

注：D:\apache-tomcat-6.0.14 请替换成本地对应目录。

4、Publisher 端（publish.jsp）实现：

在 jms-test 目录下新建 publish.jsp 文件：

```
<%@ page language="java" import="javax.jms.*" pageEncoding="GBK"%>
<%@ page language="java" import="javax.naming.*"%>
<%@ page language="java" import="org.apache.activemq.ActiveMQConnectionFactory"%>
<%
    try {
        InitialContext initCtx = new InitialContext();
        Context envContext = (Context) initCtx.lookup("java:comp/env");
       ConnectionFactory connectionFactory = (ConnectionFactory)
envContext.lookup("jms/NormalConnectionFactory");
        Connection connection = connectionFactory.createConnection();
        Session jmsSession = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer = jmsSession.createProducer((Destination)
envContext.lookup("jms/topic/MyTopic"));

        //设置持久方式
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);
        Message testMessage = jmsSession.createMessage();
        //发布刷新文章消息
        testMessage.setStringProperty("RefreshArticleId", "2046");
        producer.send(testMessage);
        //发布刷新帖子消息
        testMessage.clearProperties();
        testMessage.setStringProperty("RefreshThreadId", "331");
        producer.send(testMessage);
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (JMSException e) {
        e.printStackTrace();
    }
%>
```

Publisher 和 Listner 之间通过 Message 的 setStringProperty 和 getStringProperty 方法，实现对应的业务逻辑。

上述示例代码中，RefreshArticleId 代表刷新某篇文章，RefreshThreadId 代表刷新某个帖子，property 值保持对应的 ID。当然用户可根据实际需求灵活地使用。

5、运行 Demo：

5.1 启动 ActiveMQ 服务器

5.2 启动 Tomcat 服务器：JMSListener 将自动连接 ActiveMQ broker，日志信息：

Successfully connected to tcp://localhost:61616

5.3 访问 http://localhost:8080/jms-test/publish.jsp

Tomcat 服务器日志将提示：

接收刷新文章消息，开始刷新文章 ID=2046

接收刷新论坛帖子消息，开始刷新帖子 ID=331

5.4 访问http://localhost:8161/admin/topics.jsp查看MY.TEST.FOO的消息日志，分别发送和接收 2 条。

至此，已成功完成 ActiveMQ 与 Tomcat 的基本整合！

Publisher 和 Listener 完全可以独立部署到不同的 Web 服务器上，并通过 ActiveMQ 来进行消息传递，实现用户所需的业务逻辑。

测试持久消息的具体步骤：

l 启动 Publisher 所在 Web 服务器

l 启动 ActiveMQ

l 访问 publish.jsp 发送消息，此时 Listener 还未启动，消息将保存在 ActiveMQ 的 bin\activemq-data 目录下，查看日志可以看到发送 2 条，接收 0 条

l 启动 Listener 所在 Web 服务器，将自动接收到 ActiveMQ 的持久消息并处理，查看日志：发送 2 条，接收 2 条，表明持久消息应用成功！