Apache ActiveMQ 技术文档

说明：本文档在我们最大努力范围之内确保其正确性、 实效性和可观性， 但并不代表所有的观点都是正确的，而仅代表个人看法。如发现不当之处，请多指教，谢谢！

联系方式：

1、技术概述

JMS 是指 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（ MOM ）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。

ActiveMQ 是 Apache 软件基金下的一个开源软件，它遵循 JMS1.1 规范，是消息驱动中间软件，为企业消息传递提供高可用，出色性能，可扩展，稳定和安全保障。 ActiveMQ 使用 Apache 许可协议，因此任何人都可以使用和修改它而不必反馈任何改变。 ActiveMQ 的目标是在尽可能多的平台和语言上提供一个标准的，消息驱动的应用集成。 ActiveMQ 实现 JMS 规范并在此之上提供大量额外的特性。

2、ActiveMQ 特性

1.遵循 JMS 规范：包括同步和异步消息传递， 对于预订者的持久消息等等。 依附于 JMS 意味着不论 JMS 消息提供者是谁，同样的基本特性都是有效的

2.连接：ActiveMQ 提供各种连接选择， 包括 HTTP、HTTPS、IP 多点传送， SSL, STOMP , TCP，UDP， XMPP 等。大量的连接协议支持使之具有更好的灵活性。很多现有的系统使用一种特定协议并且不能改变，所以一个支持多种协议的消息平台降低了使用的门槛。

3.可插拔的持久性和安全： ActivemQ 提供多种持久性方案可供选择，也可以完全按自己需求定制验证和授权。

4.用 java 建立消息驱动应用： ActiveMQ 最常用在 java 应用中，用于发送和接收消息。

5.与应用服务器集成： ActiveMQ 与 java 应用服务器集成是很常见的。

6.客户端 APIs :ActiveMQ 对多种语言提供客户端 API , 除了 Java 之外还有 C/C++ , .NET , Perl，PHP，Python，Ruby 等。这使得 ActiveMQ 能用在 Java 之外的其它语言中。很多其它语言都可以通过 ActiveMQ 提供的客户端 API 使用 ActiveMQ 的全部特性。 当然，ActiveMQ 代理器（ broker）仍然是运行在 java 虚拟机上，但是客户端能够使用其它的被支持的语言。

7.代理器集群（ Broker clustering ）：为了利于扩展，多个 ActiveMQ broker 能够联合工作。这个方式就是 network of brokers 并且能支持多种拓扑结构。

8.高级代理器特性和客户端选项： ActiveMQ 为代理器和客户端连接提供很多高级的特性。 ActiveMQ 也可以通过代理器的 XML 配置文件支持 Apache Camel。

9.简单的管理： ActiveMQ 是为开发者设计的。它并不需要专门的管理工具，因为它提

供各种易用且强大的管理特性。有很多方法去监控 ActiveMQ 的各个方面，可以通过 JMX 使用 JConsole 或 ActiveMQ web console ;可以运行 ActiveMQ 消息报告； 可以用命令行脚本；可以通过日志。

3、松耦合与 ActiveMQ

ActiveMQ 提供松耦合的应用架构。 松耦合一般是为了减轻经典 RPC（Remote Procedure Calls）调用的紧耦合架构而被引入的。 该松耦合以异步形式存在， 任何一个应用对 ActiveMQ 的调用不依赖于任何其它应用， 没有任何依赖或者时序要求。 应用依赖于 ActiveMQ 的能力保证消息传递。因此，我们把应用发送消息的形式称之为触发和忘记（ fire-and-forget ） -- 应用发送消息到 ActiveMQ 之后并不关心消息如何或者什么时候被传递。 同样的消息的接收者也不关心消息从哪里或者如何到来。 在不同的环境中这样做的好处是允许客户端使用不同的语言编写甚至使用不同的线路协议。 ActiveMQ 作为中间人存在，允许不同环境的集成和异步交互。

使用 RPC，当一个应用调用另一个应用， 调用者将被阻塞知道被调用者返回结果。 图 1.1 描述了这个过程。

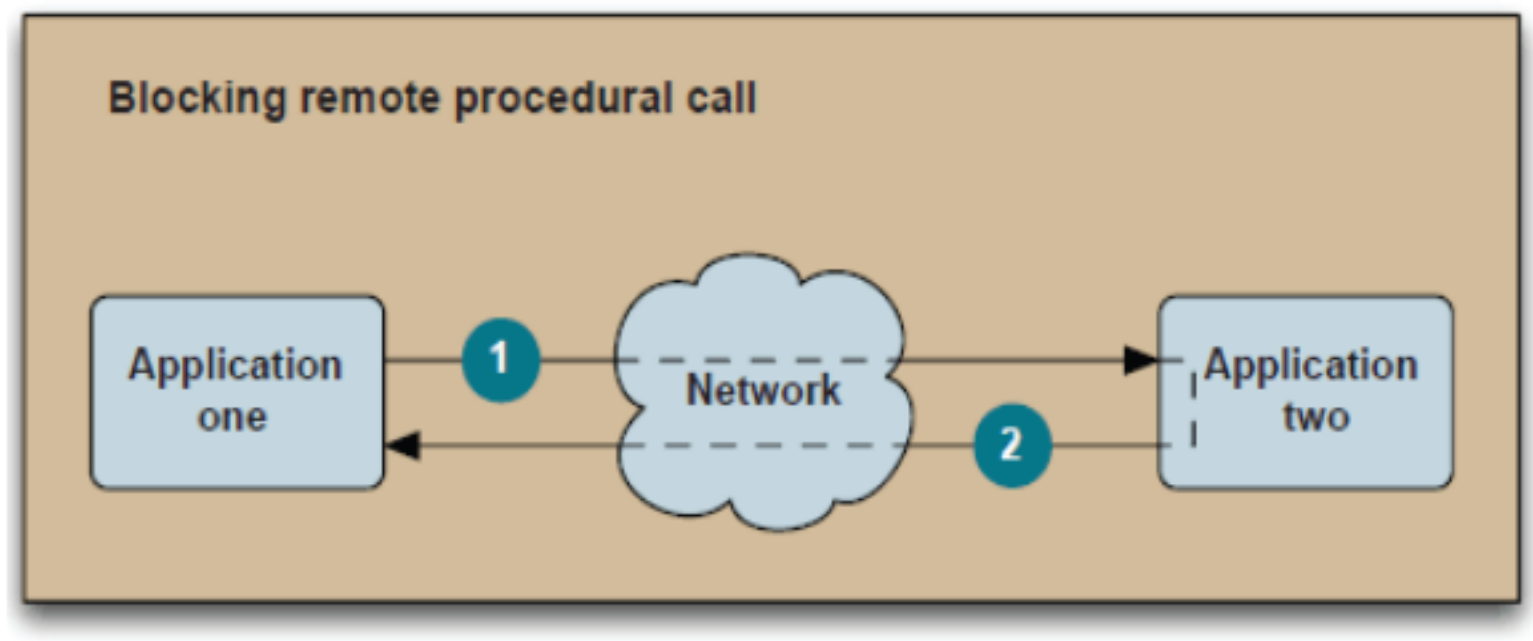


Figure 1.1 Two tightly coupled applications using remote procedure calls to communicate

调用方（ Application one ）将被阻塞直到被调用方（ Application two ）返回控制权。很多系统使用 RPC 并且成功了。但是对于这样一个紧耦合系统确实有很多缺点：最显著的缺点是，即使很小的一个改变都要较高的维护代价 ;正确的时机也很重要， 当请求从应用 1 发到应用 2 时，两个系统都必须正常工作， 同样的， 响应从应用 2 发送到应用 1 时，两个系统也必须正常工作。 这样的时序要求有点麻烦， 使得系统稳定性降低。 现在我们把这个紧耦合系统和图 1.2 的系统进行比较。

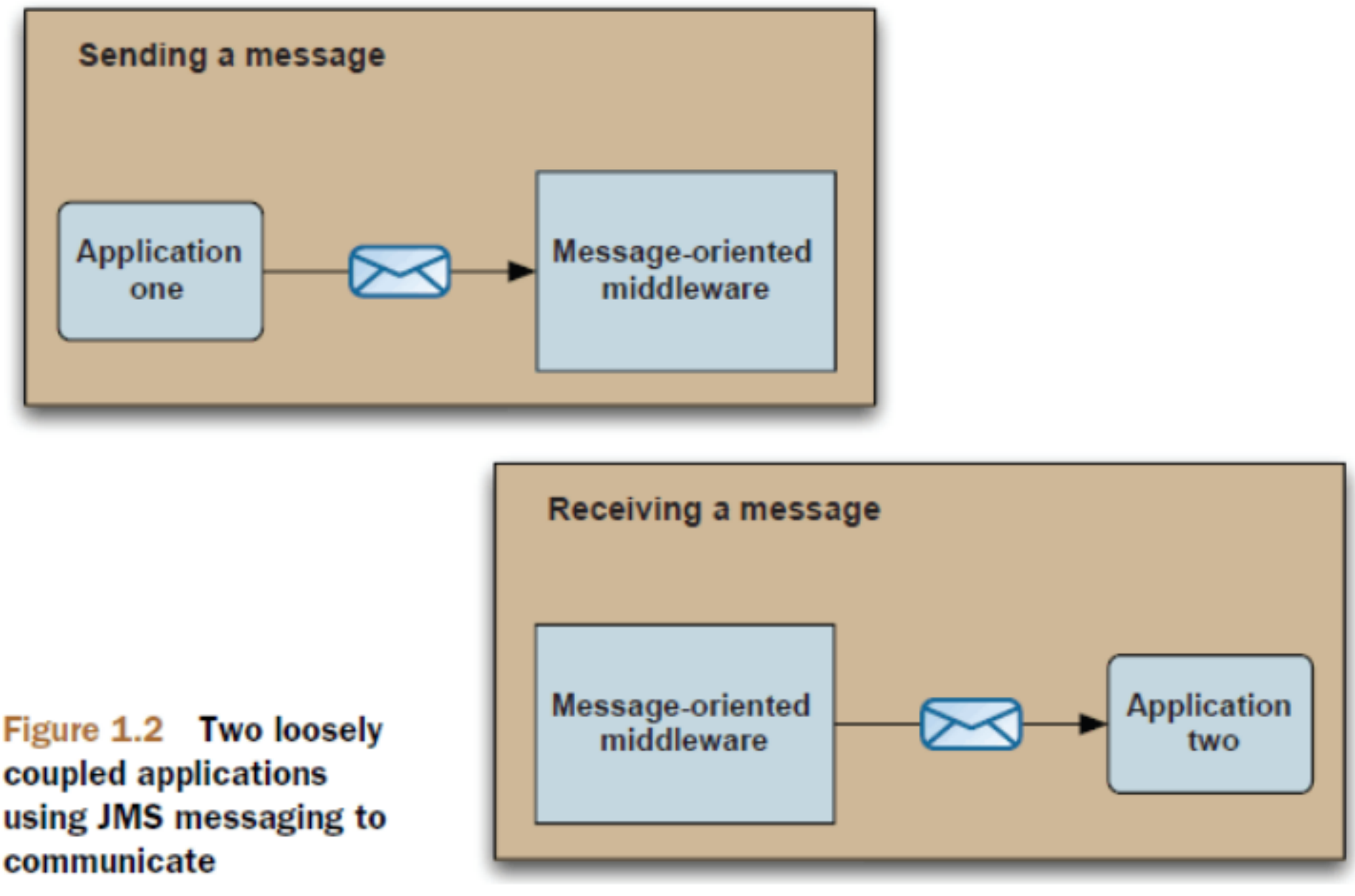


Figure 1.2 Two loosely coupled applications using JMS messaging to communicate

在图 1.2 中，应用 1 发送消息到 MOM 只是一个单方行为。 可能一段时间后， 应用 2 从 MOM 接收消息， 这也是一个单方行为。 任何一方都不需要知道另一方的存在， 它们之间也没有任何时序要求。 所以在分布式系统设计时， 松耦合系统比紧耦合系统有巨大的优势。 如图所示，这就是 ActiveMQ 存在的地方。

考虑现在其中的一个应用必须搬到一个新的地方。这可能在新硬件引入或应用需要移动时发生。 如果是一个紧耦合系统， 这样的迁移会很困难， 因为系统的其它部分都必须停止工作等待迁移完成。 如果是松耦合系统， 系统的各个部分能够自由迁移而不影响其它部分。 考虑这样一个场景，应用 A 和 B 各有很多个实例，其中各个实例分布在不同的机器上。 ActiveMQ 安装在另外的机器上。在这种情况下，任何一个应用实例都可以自由移动而不影响其它应用。事实上， 多个 ActiveMQ 实例也可以通过 network of brokers 配置联合使用。这就允许 ActiveMQ 实例自由迁移而不影响应用 A 或应用 B。采用这种价构， 系统的任何一部分在任何时间都可以停机进行维护而不影响整个系统。更多的介绍将放在第 10 章。

总之， ActiveMQ 提供一个令人难以置信的灵活性允许松耦合思想变成现实。对于某些情况不能使用异步方式实现， ActiveMQ 也提供消息的请求 /回复模式支持。

4、何时使用 ActiveMQ

有很多情况 ActiveMQ 和异步消息能对一个系统的架构产生有意义的作用。 下面列举一些场景。

1.不同语言应用集成 ----ActiveMQ 使用 java 编写，并且提供一个 java 客户端 API。但 ActiveMQ 也为 C/C++， .NET,Perl,PHP,Python,Ruby 等提供客户端。当你考虑在不同平台不同语言下的各个应用进行集成时，这将是一个巨大的优势。不同语言的客户端 API 使各种

不同的语言能够通过 ActiveMQ 发送和接收消息。对于 ActiveMQ 提供的多语言兼容，还有一个好处是相对于 RPC 调用，它能帮助系统各应用间的解耦。

2.RPC 的替代者 ----应用广泛的使用 RPC 模式的同步调用。想一下，现在大量使用 RPC 调用的客户端服务器模式的应用，它们包括 ATMs，大多数 web 应用，信用卡系统，销售点（point-of-sale）系统等。尽管它们大多数是成功的，但是转换到异步消息模式能够在保证正确响应的情况下带来一些好处。 使用同步请求的系统在规模上有较大的限制， 因为请求会被阻塞， 从而导致整个系统变慢。 如果使用异步消息替代， 可以很容易增加额外的消息接收者，使得消息能被并发消耗，从而加快请求处理。当然，必须你系统应用间是解耦的。

3.应用间解耦 ---- 就是上面讨论过的，紧耦合系统能带来很多问题，特别是在应用是分布式的情况下。 松耦合系统，也就是依赖性小的系统，可以更好地适应未知变化。 不只是系统某部分的改变不会影响整个系统， 而且部件间的交互也更简单。 相比使用同步的系统 （调用者必须等待被调用者返回信息） ，异步系统（调用方发送消息后就不管， 即 fire-and-forget ）能够给我们带来事件驱动架构（ event-driven architecture EDA ）.

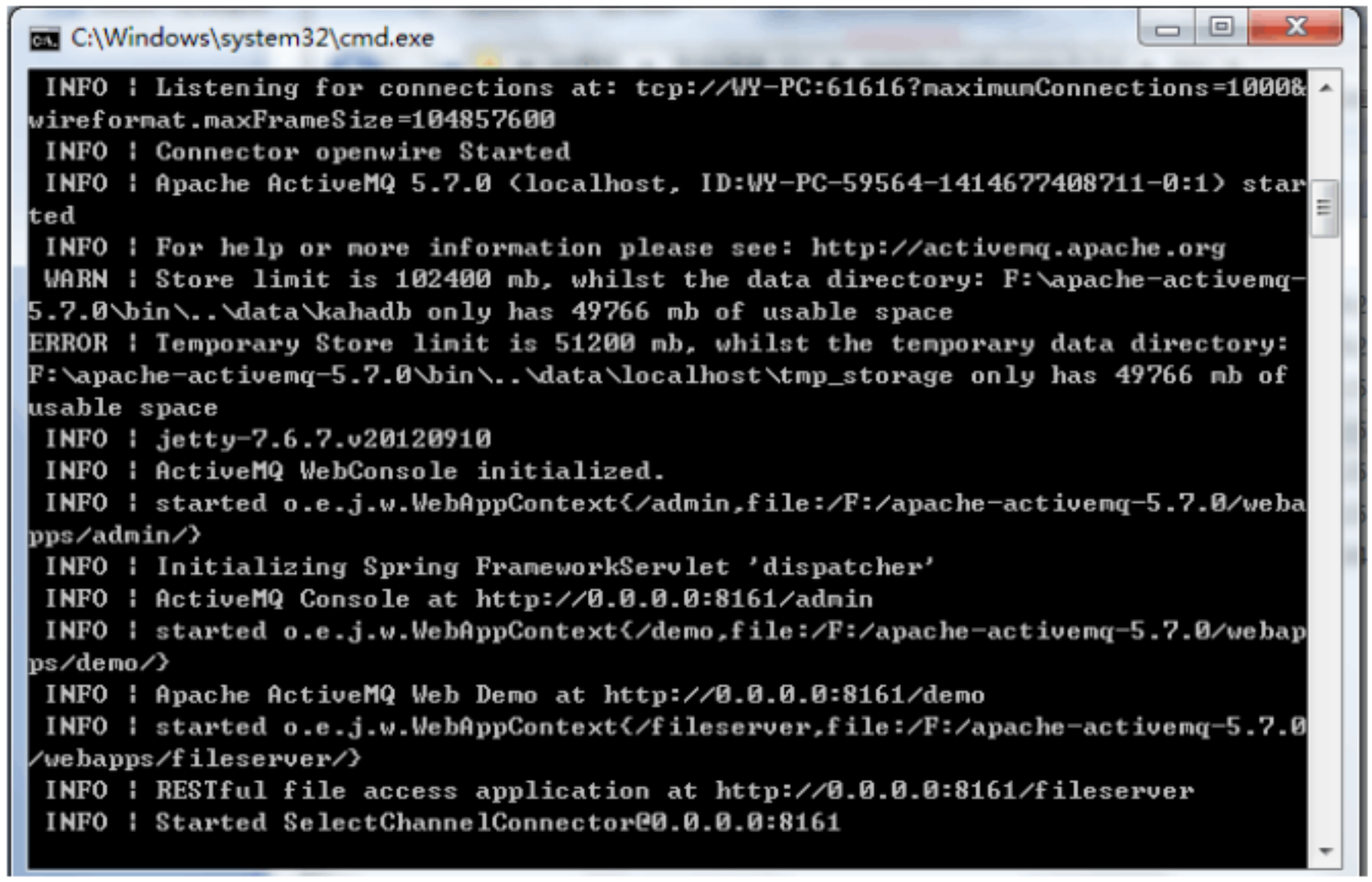
4.作为事件驱动架构的骨架 ---- 解耦，异步架构的系统允许通过代理器自己配置更多的客户端， 内存等（即 vertical scalability ）来扩大系统， 而不是增加更多的代理器 （即 horizontal scalability ）。考虑如亚马逊这样繁忙的电子商务系统。当用户购买物品，事实上系统需要很多步骤去处理，包括下单，创建发票，付款，执行订单，运输等。但是用户下单后，会立即返回“ 谢谢你下单 ”的界面。 不只是没有延迟， 而且用户还会受到一封邮件表明订单已经收到。 在亚马逊下单的例子就是一个多步处理的例子。 每一步都由单独的服务去处理。 当用户下单是， 有一个同步的体积表单动作， 但整个处理流程并不通过浏览器同步处理。 相反地，订单马上被接受和反馈。 而剩下的步骤就通过异步处理。 如果在处理过程中出错， 用户会通过邮件收到通知。这样的异步处理能提供高负载和高可用性。

5.提高系统扩展性。很多使用事件驱动设计的系统是为了获得高可扩展性，例如电子商务，政府，制造业，线上游戏等。通过异步消息分开商业处理步骤给各个应用，能够带来很多可能性。考虑设计一个应用来完成一项特殊的任务。这就是面向服务的架构（service-oriented architecture SOA ）。每一个服务完成一个功能并且只有一个功能。应用就通过服务组合起来， 服务间使用异步消息和最终一致性。 这样的设计便可以引入一个复杂事件处理概念（ complex event processing CEP）。使用 CEP，部件间的交互可以被记录追踪。在异步消息系统中，可以很容易在部件间增加一层处理。

5、使用步骤

1.从 apache官网下载 activeMQ

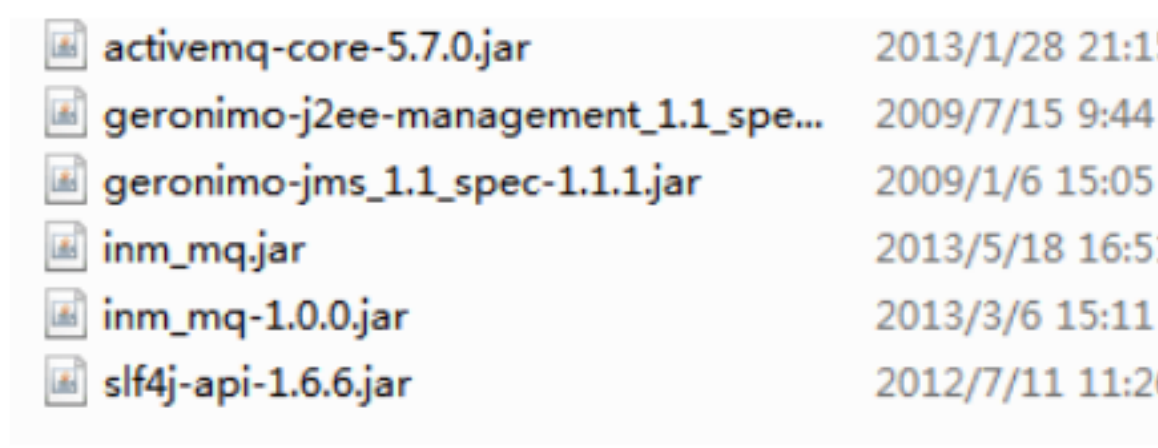
解压到任意目录之后进入 bin 文件夹，运行 activemq.bat,



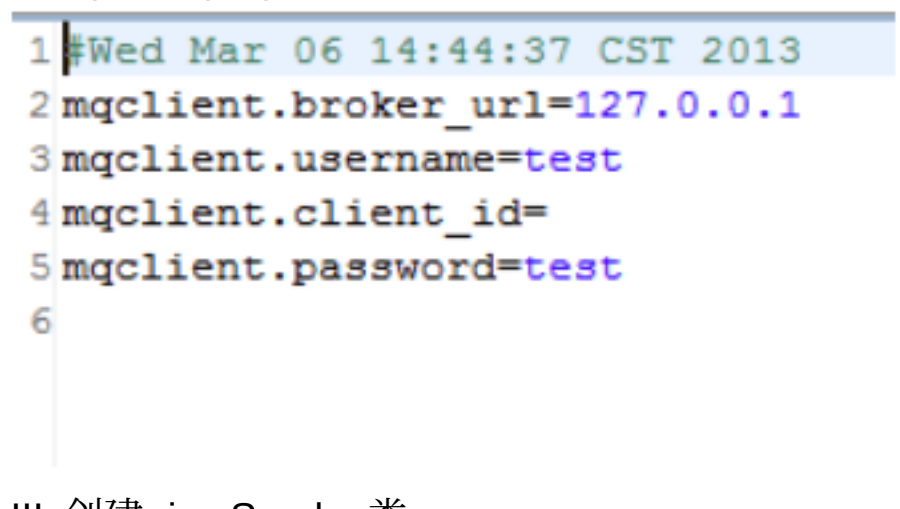
默认占用 8161 端口，访问 http://localhost:8161/admin/ 可以查看 activeMQ 首页

2.activeMQ 包括客户端和服务器，我们首先在客户端加入 activeMQ

I.导入 jar 包



II.mqclient.properties



```
#Wed Mar 06 14:44:37 CST 2013
mqclient.broker_url=127.0.0.1
mqclient.username=test
mqclient.client_id=
mqclient.password=test
```

III. 创建 jmsSender 类

```
public class JmsSender {

    /**
     *  @Description : 消息发送者
     *  @param zyflsbm 消息类型
     *  @param message 消息内容
     *  @return
```

```
 *  @date : 2013  - 3 - 11
 */
public  void  send(String topic,String message){

    try  {  //  使用默认的配置文件  ( 进程当前路径下的  mqclient.properties  文件 )
创建 MQClient  对象

        String  path  =
JmsSender.  class  .getClassLoader().getResource(  "mqclient.properties"  ).g
etPath();
        MQClient.  setConfigFilePath  (path);
        MQClient client = MQClient.  getInstance  ();
        //  创建与 Test  主题关联的  MQProducer  对象，  此对象用于向  Test  主题发送消
息
        MQProducer  producer  =
client.createTopicProducer(  "CLIENT_RESOURCE"  );
            //  创建一个消息对象
            MQMessage msg =  new  MQMessage();

            //  设置消息内容  (json  串)
             msg.setContent(message);
             try  {
                producer.send(msg);
            }  catch  (MQException e) {
                //  TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.  out  .println(  "send: "  + msg.getContent());

    }  catch  (MQException e1) {
        //  TODO Auto-generated catch block
        e1.printStackTrace();
    }
    }
    }
```

IV. 客户端调用 jmsSender 类的方法

```
    @RequestMapping  ( "/send.do"  )
    public  void  send(HttpServletRequest  request,HttpServletResponse
response)  throws  IOException{
        String json=(String)request.getAttribute(  "json"  );
        new  JmsSender().send(  " 客户端请求 " ,  "aa"  , json);
        String contextPath = request.getContextPath();
        response.sendRedirect(contextPath+  "/user/list.do"  );
```

}

3.服务端 ActiveMQ 配置

I. 首先就是导入相同的 jar 包以及相同的 activemq.properties 配置文件

II. 创建 listener 类，继承 ServletContextListener

```
public void contextInitialized(ServletContextEvent arg0) {
// TODO Auto-generated method stub
System. out .println( "------- 开始监听消息 " );
try {
// SelectDataService selectDataService = (SelectDataService)
ContextHolder.getBean("selectDataService");
String path =
ReceiveMessageListener. class .getClassLoader().getResource( "mqclient.p
roperties" ).getPath();
MQClient. setConfigFilePath (path);
MQClient client = MQClient. getInstance ();
// 获取要同步数据的专业网管系统

MQConsumer consumer
=client.createDurableTopicConsumer( "CLIENT_RESOURCE" , "CLIENT_RESOURCE
" );
consumer.setMessageListener( new JmsReceiver());
} catch (MQException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
```

III. 在 web.xml 中配置 listener 接收消息

```
<!-- 监听消息服务器，接收消息 -->
<listener >

<listener-class >com.baidu.jms.ReceiveMessageListener </ listener-class >
</ listener >
```

IV. 创建 JMSReceiver 类 并且实现 MQListener

```
/**
* 处理收到的资源数据
* @author Administrator
*
*/
public void onMessage(MQMessage msg)
{
```

```
try
{
    ApplicationContext ac = new ClassPathXmlApplicationContext( "/spring.xml" );
    SelectDataService selectDataService = (SelectDataService)ac.getBean( "selectDataService" );
    // 从消息中取出数据
    String jsonData= msg.getContent();
    System. out .println( " 接收方 : " + jsonData);

// 解析数据 ...
// 将 json 格式字符串转换成 map
JSONObject jsonObject=JSONObject. fromObject (jsonData);
Map<String,String> map= new HashMap<String, String>();
Iterator it = jsonObject.keys();
while (it.hasNext()){
    String key=String. valueOf (it.next());
    String value=(String)jsonObject.get(key);
    map.put(key, value);
}
// 拼接 sql 语句
String operate = map.get( "operate" );
    String sql= "" ;
    String date= new SimpleDateFormat( "yyyy-MM-dd HH:mm:ss" ).format( new Date());
    // 将更新信息保存到操作表中
    sql= "insert into t_operate(id,operate,updatetime) values('" +WebUtils. getRandomId ()+ "','" +operate+ "','" +date+ "')" ;
    selectDataService.update(sql);
    if ( "insert" .equals(operate)){
    sql= "insert into t_user(id,loginName,password,phone,mail) values('" +map.get( "id" )+ "','" +map.get( "loginName" )+ "','" +map.get( "password" )+ "','" +map.get( "phone" )+ "','" +map.get( "mail" )+ "')" ;
    } else if ( "update" .equals(operate)){
        sql= "update t_user set loginName='" +map.get( "loginName" )+ "',password='" +map.get( "password" )+ "',phone='" +map.get( "phone" )+ "',mail='" +map.get( "mail" )+ "' where id='" +map.get( "id" )+ "'" ;
    } else if ( "delete" .equals(operate)){
        sql= "delete from t_user where id='" +map.get( "id" )+ "'" ;
    } else {
        sql= null ;
    }
    if (sql!= null ){
```

```
        // 根据接收的 sql 语句更新数据库
        selectDataService.update(sql);
    }
    System. out .println( " 数据更新成功！！！ " );
```

----------Spring 集成

Pom文件

```
<dependency >
        <groupId >org.springframework </ groupId >
        <artifactId >spring-jms </ artifactId >
        <version >${spring.version} </ version >
    </ dependency >

<dependency >
        <groupId >org.apache.activemq </ groupId >
        <artifactId >activemq-core </ artifactId >
        <version >${activemq_version} </ version >
```

```
</ dependency >

<dependency >

<groupId >org.apache.activemq </ groupId >

<artifactId >activemq-pool </ artifactId >

<version >${activemq_version} </ version >

</ dependency >
```