**品优购电商系统开发**

**第14章**

**消息中间件解决方案JMS**

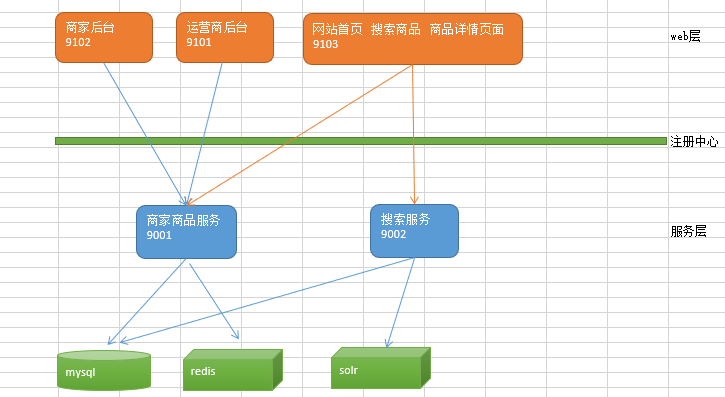
传智播客.黑马程序员

# 1. JMS入门

## 1.1消息中间件

### 1.1.1品优购系统模块调用关系分析

我们现在讲品优购已经完成的功能模块梳理如下：



我们已经完成了5个web模块和4个服务模块。其中运营商后台的调用关系最多，用到了商家商品服务、广告内容服务、搜索服务和页面生成服务。这种模块之间的依赖也称之为耦合。而耦合越多，之后的维护工作就越困难。那么如果改善系统模块调用关系、减少模块之间的耦合呢？我们接下来就介绍一种解决方案----消息中间件。

### 1.1.2什么是消息中间件

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F)的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。对于消息中间件，常见的角色大致也就有Producer（生产者）、Consumer（消费者）

常见的消息中间件产品:

**（1）ActiveMQ**

ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现。我们在本次课程中介绍 ActiveMQ的使用。

（2）RabbitMQ

AMQP协议的领导实现，支持多种场景。淘宝的MySQL集群内部有使用它进行通讯，OpenStack开源云平台的通信组件，最先在金融行业得到运用。

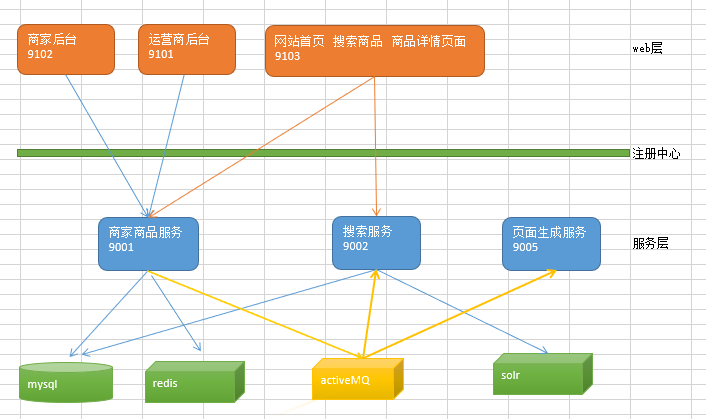
（3）ZeroMQ

史上最快的消息队列系统

（4）Kafka

Apache下的一个子项目 。特点：高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率；完全的分布式系统。适合处理海量数据。

### 1.1.3改造系统模块调用关系



我们通过引入消息中间件activeMQ,使得运营商系统与搜索服务、页面生成服务解除了耦合。

## 1.2 JMS简介

### 1.2.1什么是JMS

JMS（[**Java**](http://lib.csdn.net/base/java)Messaging Service）是Java平台上有关面向消息中间件的技术规范，它便于消息系统中的Java应用程序进行消息交换,并且通过提供标准的产生、发送、接收消息的接口简化企业应用的开发。

       JMS本身只定义了一系列的接口规范，是一种与厂商无关的 API，用来访问消息收发系统。它类似于 JDBC([**java**](http://lib.csdn.net/base/java)Database Connectivity)：这里，JDBC 是可以用来访问许多不同关系[**数据库**](http://lib.csdn.net/base/mysql)的 API，而 JMS 则提供同样与厂商无关的访问方法，以访问消息收发服务。许多厂商目前都支持 JMS，包括 IBM 的 MQSeries、BEA的 Weblogic JMS service和 Progress 的 SonicMQ，这只是几个例子。 JMS 使您能够通过消息收发服务（有时称为消息中介程序或路由器）从一个 JMS 客户机向另一个 JML 客户机发送消息。消息是 JMS 中的一种类型对象，由两部分组成：报头和消息主体。报头由路由信息以及有关该消息的元数据组成。消息主体则携带着应用程序的数据或有效负载。

JMS 定义了五种不同的消息正文格式，以及调用的消息类型，允许你发送并接收以一

些不同形式的数据，提供现有消息格式的一些级别的兼容性。

· TextMessage--一个字符串对象

· MapMessage--一套名称-值对

· ObjectMessage--一个序列化的 Java 对象

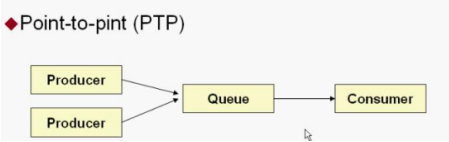
· BytesMessage--一个字节的数据流

· StreamMessage -- Java 原始值的数据流

### 1.2.2 JMS消息传递类型

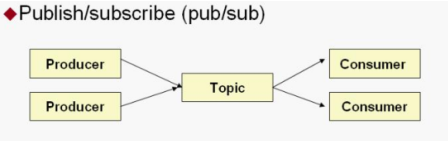
对于消息的传递有两种类型：

一种是点对点的，即一个生产者和一个消费者一一对应；



另一种是发布/ 订阅模式，即一个生产者产生消息并进行发送后，可以由多个消费者进

行接收。



## 1.3ActiveMQ下载与安装

### 1.3.1下载

官方网站下载：<http://activemq.apache.org/>

### 1.3.2安装（Linux）

1. 将apache-activemq-5.12.0-bin.tar.gz 上传至服务器
2. 解压此文件

|  |
| --- |
| tar zxvf apache-activemq-5.12.0-bin.tar.gz |

1. 为apache-activemq-5.12.0目录赋权

|  |
| --- |
| chmod 777 apache-activemq-5.12.0 |

1. 进入apache-activemq-5.12.0\bin目录

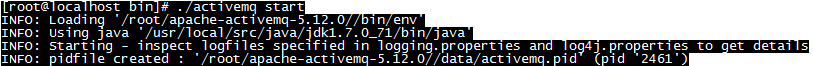
（5）赋与执行权限

|  |
| --- |
| chmod 755 activemq  -------------------------------------- **知识点小贴士** --------------------------  linux 命令chmod 755的意思  chmod是[**Linux**](http://lib.csdn.net/base/linux)下设置文件权限的命令，后面的数字表示不同用户或用户组的权限。  一般是三个数字： 第一个数字表示文件所有者的权限 第二个数字表示与文件所有者同属一个用户组的其他用户的权限 第三个数字表示其它用户组的权限。        权限分为三种：读（r=4），写（w=2），执行（x=1） 。 综合起来还有可读可执行（rx=5=4+1）、可读可写（rw=6=4+2）、可读可写可执行(rwx=7=4+2+1)。        所以，chmod 755 设置用户的权限为：  1.文件所有者可读可写可执行                                   --7  2.与文件所有者同属一个用户组的其他用户可读可执行 --5  3.其它用户组可读可执行                                        --5 |

### 1.3.3启动

|  |
| --- |
| ./activemq start |

出现下列提示表示成功！



假设服务器地址为192.168.200.128 ，打开浏览器输入地址

<http://192.168.200.128:8161/> 即可进入ActiveMQ管理页面



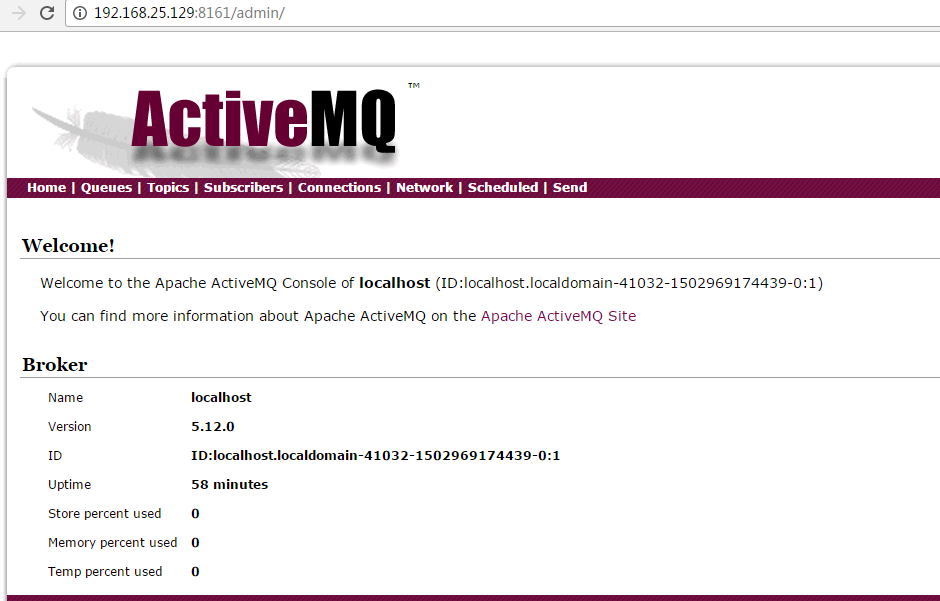
点击进入管理页面



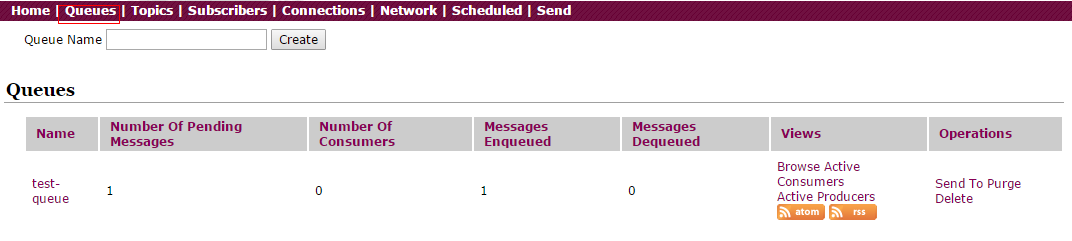
输入用户名和密码 均为 admin



进入主界面



点对点消息列表：



列表各列信息含义如下：

[**Number Of Pending Messages**](http://192.168.25.135:8161/admin/queues.jsp)**：**等待消费的消息 这个是当前未出队列的数量。

[**Number Of Consumers**](http://192.168.25.135:8161/admin/queues.jsp)**：**消费者 这个是消费者端的消费者数量

[**Messages Enqueued**](http://192.168.25.135:8161/admin/queues.jsp)**：**进入队列的消息  进入队列的总数量,包括出队列的。

[**Messages Dequeued**](http://192.168.25.135:8161/admin/queues.jsp)**：**出了队列的消息  可以理解为是消费这消费掉的数量。

# 2. JMS入门小Demo

## 2.1点对点模式

点对点的模式主要建立在一个队列上面，当连接一个列队的时候，发送端不需要知道接收端是否正在接收，可以直接向ActiveMQ发送消息，发送的消息，将会先进入队列中，如果有接收端在监听，则会发向接收端，如果没有接收端接收，则会保存在activemq服务器，直到接收端接收消息，点对点的消息模式可以有多个发送端，多个接收端，但是一条消息，只会被一个接收端给接收到，哪个接收端先连上ActiveMQ，则会先接收到，而后来的接收端则接收不到那条消息。

### 2.1.1消息生产者

（1）创建工程jmsDemo ，引入依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-client</artifactId>  <version>5.13.4</version>  </dependency>  </dependencies>  <build>  <plugins>  <!-- java编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.2</version>  <configuration>  <source>1.7</source>  <target>1.7</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build> |

（2）创建类QueueProducer main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.200.128:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建队列对象  Queue queue = session.createQueue("test-queue");  //6.创建消息生产者  MessageProducer producer = session.createProducer(queue);  //7.创建消息  TextMessage textMessage = session.createTextMessage("欢迎来到神奇的品优购世界");  //8.发送消息  producer.send(textMessage);  //9.关闭资源  producer.close();  session.close();  connection.close(); |

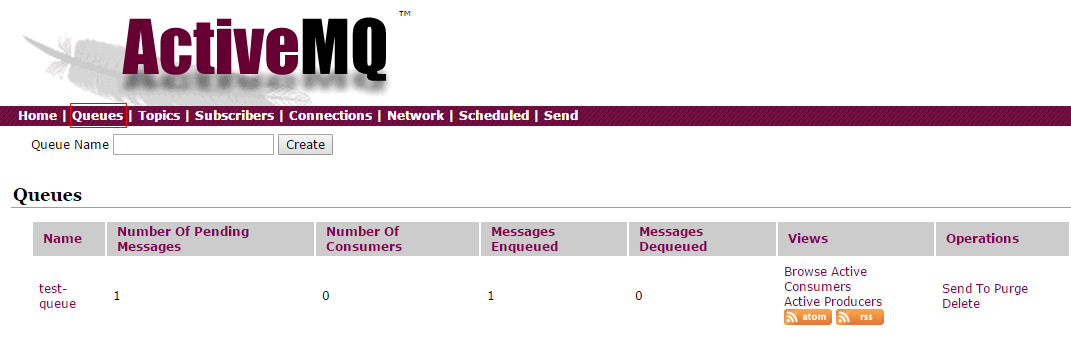
上述代码中第4步创建session 的两个参数：

第1个参数 是否使用事务

第2个参数 消息的确认模式

* AUTO\_ACKNOWLEDGE = 1    自动确认
* CLIENT\_ACKNOWLEDGE = 2    客户端手动确认
* DUPS\_OK\_ACKNOWLEDGE = 3    自动批量确认
* SESSION\_TRANSACTED = 0    事务提交并确认

运行后通过ActiveMQ管理界面查询

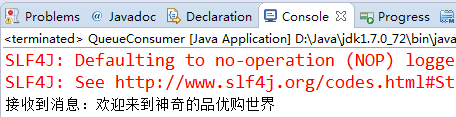


### 2.1.2消息消费者

创建类QueueConsumer ，main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.200.128:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建队列对象  Queue queue = session.createQueue("test-queue");  //6.创建消息消费  MessageConsumer consumer = session.createConsumer(queue);    //7.监听消息  consumer.setMessageListener(**new** MessageListener() {  **public** **void** onMessage(Message message) {  TextMessage textMessage=(TextMessage)message;  **try** {  System.***out***.println("接收到消息："+textMessage.getText());  } **catch** (JMSException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  });  //8.等待键盘输入  System.***in***.read();  //9.关闭资源  consumer.close();  session.close();  connection.close(); |

执行后看到控制台输出



### 2.1.3运行测试

同时开启2个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现只有一个消费者会接收到消息。

## 2.2 发布/订阅模式

### 2.2.1消息生产者

创建类TopicProducer ，main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.200.128:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建主题对象  Topic topic = session.createTopic("test-topic");  //6.创建消息生产者  MessageProducer producer = session.createProducer(topic);  //7.创建消息  TextMessage textMessage = session.createTextMessage("欢迎来到神奇的品优购世界");  //8.发送消息  producer.send(textMessage);  //9.关闭资源  producer.close();  session.close();  connection.close(); |

运行效果如下：



### 2.2.2消息消费者

创建类TopicConsumer ,main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.200.128:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建主题对象  //Queue queue = session.createQueue("test-queue");  Topic topic = session.createTopic("test-topic");  //6.创建消息消费  MessageConsumer consumer = session.createConsumer(topic);    //7.监听消息  consumer.setMessageListener(**new** MessageListener() {  **public** **void** onMessage(Message message) {  TextMessage textMessage=(TextMessage)message;  **try** {  System.***out***.println("接收到消息："+textMessage.getText());  } **catch** (JMSException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  });  //8.等待键盘输入  System.***in***.read();  //9.关闭资源  consumer.close();  session.close();  connection.close(); |

### 2.2.3运行测试

同时开启2个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现每个消费者会接收到消息。

# 3.商品审核-导入Solr索引库和静态化

## 3.1需求分析

1. manager-web运营商项目审核商品, 调用sellergoods-service项目中的方法

2. sellergoods-service项目更改数据库中商品的审核状态

3. sellergoods-service项目向activeMq服务器发送消息(发布订阅模式), 消息为审核商品的id

4. search-service接收商品id, 根据id查询商品详细数据放入solr索引库

5.page-service接收商品id, 根据id查询商品详细数据生成静态化页面

## 3.2运营商后台

Manager-web项目的GoodsController.java中使用dubbox调用sellergoods-service项目

|  |
| --- |
| @Reference  **private** GoodsService goodsService;    @RequestMapping("/updateStatus")  **public** Result updateStatus(Long[] ids, String status) **throws** Exception {  **try** {  goodsService.updateStatus(ids, status);  **return** **new** Result(**true**, "审核成功!");  } **catch** (Exception e) {  e.printStackTrace();  **return** **new** Result(**false**, "审核失败!");  }  } |

## 3.3消息生产者

### 3.3.1配置文件

sellergoods-service项目src/main/resources/spring文件夹下加入applicationContext-jms.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:amq=*"http://activemq.apache.org/schema/core"*  xmlns:jms=*"http://www.springframework.org/schema/jms"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <bean id=*"targetConnectionFactory"* class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <property name=*"brokerURL"* value=*"tcp://192.168.200.128:61616"*/>  </bean>    <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <bean id=*"connectionFactory"* class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <property name=*"targetConnectionFactory"* ref=*"targetConnectionFactory"*/>  </bean>    <!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->  <bean id=*"jmsTemplate"* class=*"org.springframework.jms.core.JmsTemplate"*>  <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->  <property name=*"connectionFactory"* ref=*"connectionFactory"*/>  </bean>    <!-- 发布订阅模式, 商品导入索引库和生成静态页面 -->  <bean id=*"topicPageAndSolrDestination"* class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-arg value=*"pinyougou\_topic\_page\_solr"*/>  </bean>  </beans> |

### 3.3.2代码实现

GoodsServiceImpl.java中修改商品审核状态, 发布消息到activeMq

|  |
| --- |
| @Autowired  **private** JmsTemplate jmsTemplate;    @Autowired  **private** Destination topicPageAndSolrDestination;  @Override  **public** **void** updateStatus(Long[] ids, String status) {  **if**(ids != **null**){  **for**(**final** Long id : ids){  Goods goods = **new** Goods();  goods.setId(id);  goods.setAuditStatus(status);  goodsDao.updateByPrimaryKeySelective(goods);    //发布订阅模式, 发布审核通过的商品id  **if**("1".equals(status)){  jmsTemplate.send(topicPageAndSolrDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  TextMessage message = session.createTextMessage(String.*valueOf*(id));  **return** message;  }  });  }  }  }  } |

## 3.4消息消费者（搜索服务）

### 3.4.1.配置文件

Search-service项目的src/main/resources/spring文件夹下加入配置文件applicationContext-jms-consumer.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:amq=*"http://activemq.apache.org/schema/core"*  xmlns:jms=*"http://www.springframework.org/schema/jms"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <bean id=*"targetConnectionFactory"* class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <property name=*"brokerURL"* value=*"tcp://192.168.200.128:61616"*/>  </bean>    <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <bean id=*"connectionFactory"* class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <property name=*"targetConnectionFactory"* ref=*"targetConnectionFactory"*/>  </bean>    <!--发布订阅模式, 将数据导入solr索引库-->  <bean id=*"topicPageAndSolrDestination"* class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-arg value=*"pinyougou\_topic\_page\_solr"*/>  </bean>    <!-- 发布订阅模式, 消息监听容器, 将数据导入solr索引库 -->  <bean class=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <property name=*"connectionFactory"* ref=*"connectionFactory"* />  <property name=*"destination"* ref=*"topicPageAndSolrDestination"* />  <property name=*"messageListener"* ref=*"pageAndSolrListener"* />  </bean>  <bean id=*"pageAndSolrListener"* class=*"cn.itcast.core.listener.ItemSearchListener"*></bean>  </beans> |

### 3.4.2.代码实现

Search-service项目下创建监听器类ItemSearchListener.java

|  |
| --- |
| **public** **class** ItemSearchListener **implements** MessageListener {    @Autowired  **private** ItemSearchService itemSearchService;  @Override  **public** **void** onMessage(Message message) {  ActiveMQTextMessage atm = (ActiveMQTextMessage)message;  **try** {  String text = atm.getText();  List<Item> list = itemSearchService.findItemListByGoodsIdAndStatus(Long.*parseLong*(text), "1");  itemSearchService.importList(list);  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

ItemSearchService.java中加入方法

|  |
| --- |
| @Override  **public** List<Item> findItemListByGoodsIdAndStatus(Long id, String status) {  ItemQuery itemQuery = **new** ItemQuery();  cn.itcast.core.pojo.item.ItemQuery.Criteria createCriteria = itemQuery.createCriteria();  createCriteria.andGoodsIdEqualTo(id);  createCriteria.andStatusEqualTo(status);  List<Item> list = itemDao.selectByExample(itemQuery);  **return** list;  } |

## 3.5消息消费者（静态化服务）

### 3.5.1.配置文件

Page-service项目的src/main/sources/spring文件夹下加入配置文件, applicationContext-jms-consumer.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:amq=*"http://activemq.apache.org/schema/core"*  xmlns:jms=*"http://www.springframework.org/schema/jms"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <bean id=*"targetConnectionFactory"* class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <property name=*"brokerURL"* value=*"tcp://192.168.200.128:61616"*/>  </bean>    <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <bean id=*"connectionFactory"* class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <property name=*"targetConnectionFactory"* ref=*"targetConnectionFactory"*/>  </bean>    <!--发布订阅模式, 生成页面-->  <bean id=*"topicPageAndSolrDestination"* class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-arg value=*"pinyougou\_topic\_page\_solr"*/>  </bean>    <!-- 发布订阅模式, 消息监听容器 生成页面 -->  <bean class=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <property name=*"connectionFactory"* ref=*"connectionFactory"* />  <property name=*"destination"* ref=*"topicPageAndSolrDestination"* />  <property name=*"messageListener"* ref=*"pageListener"* />  </bean>  <bean id=*"pageListener"* class=*"cn.itcast.core.listener.PageListener"*></bean>    </beans> |

### 3.5.2.代码实现

Page-service项目创建监听器类PageListener.java

|  |
| --- |
| **public** **class** PageListener **implements** MessageListener {    @Autowired  **private** ItemPageService itemPageService;  @Override  **public** **void** onMessage(Message message) {  ActiveMQTextMessage atm = (ActiveMQTextMessage)message;  **try** {  String goodsId = atm.getText();    itemPageService.createItemPage(Long.*parseLong*(goodsId));  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

# 4.商品删除-移除Solr索引库记录

## 4.1需求分析

1. manager-web运营商项目删除商品, 调用sellergoods-service项目中的方法

2. sellergoods-service项目更改数据库中商品的删除状态

3. sellergoods-service项目向activeMq服务器发送消息(点对点模式), 消息为删除商品的id

4. search-service项目接收商品id, 根据id删除solr索引库中对应的数据

## 4.2运营商后台

Manager-web项目的GoodsController.java中使用dubbox调用sellergoods-service项目

|  |
| --- |
| @RequestMapping("/delete")  **public** Result delete(**final** Long[] ids) **throws** Exception {  **try** {  goodsService.delAll(ids);  **return** **new** Result(**true**, "删除成功!");  } **catch** (Exception e) {  e.printStackTrace();  **return** **new** Result(**false**, "删除失败!");  }  } |

## 4.2消息生产者

### 4.2.1配置文件

Sellergoods-service项目中的applicationContext-jms.xml配置文件加入下面配置

|  |
| --- |
| <!-- 点对点模式，删除索引库-->  <bean id=*"queueSolrDeleteDestination"* class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-arg value=*"pinyougou\_queue\_solr\_delete"*/>  </bean> |

### 4.2.2代码实现

Sellergoods-service项目中的GoodsServiceImpl.java加入下面代码

|  |
| --- |
| @Autowired  **private** Destination queueSolrDeleteDestination;  @Override  **public** **void** delAll(Long[] ids) {  **if**(ids != **null**){  **for**(**final** Long id : ids){  Goods goods = **new** Goods();  goods.setId(id);  goods.setIsDelete("1");  goodsDao.updateByPrimaryKeySelective(goods);    //删除solr索引库中的内容  jmsTemplate.send(queueSolrDeleteDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  TextMessage textMessage = session.createTextMessage(String.*valueOf*(id));  **return** textMessage;  }  });  }  }  } |

## 4.3消息消费者（搜索服务）

### 4.3.1配置文件

Search-service项目中的applicationContext-jms-consumer.xml配置文件加入下面配置

|  |
| --- |
| <!-- 点对点模式，删除索引库-->  <bean id=*"queueSolrDeleteDestination"* class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-arg value=*"pinyougou\_queue\_solr\_delete"*/>  </bean>  <!-- 点对点模式, 消息监听容器 删除索引库-->  <bean class=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <property name=*"connectionFactory"* ref=*"connectionFactory"* />  <property name=*"destination"* ref=*"queueSolrDeleteDestination"* />  <property name=*"messageListener"* ref=*"itemDeleteListener"* />  </bean>  <bean id=*"itemDeleteListener"* class=*"cn.itcast.core.listener.ItemDeleteListener"*></bean> |

### 4.3.2代码实现

Search-service项目创建监听器类ItemDeleteListener.java

|  |
| --- |
| **public** **class** ItemDeleteListener **implements** MessageListener {    @Autowired  **private** ItemSearchService itemSearchService;    @Override  **public** **void** onMessage(Message message) {  ActiveMQTextMessage atm = (ActiveMQTextMessage)message;  **try** {  String idStr = atm.getText();  itemSearchService.deleteByGoodsIds(Long.*parseLong*(idStr));  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

Search-service项目的ItemSearchServiceImpl.java

|  |
| --- |
| @Override  **public** **void** deleteByGoodsIds(Long goodsId) {  Query query = **new** SimpleQuery();  Criteria criteria = **new** Criteria("item\_goodsid").is(goodsId);  query.addCriteria(criteria);  solrTemplate.delete(query);  solrTemplate.commit();  } |