第13天

品优购电商系统开发

# 上节内容回顾

1. freemarker的使用及语法
2. Spring + freemarker的使用
3. 模版的编辑
4. SKU数据的展示

# 今日学习内容(学习内容按视频的顺序进行归纳总结)

1. ActiveMQ的点对点模式和发布订阅模式
2. ActiveMQ与Spring整合
3. 商品详细页静态化
4. 商品审核发送消息的功能

# 课程介绍及需求(视频01) [了解]

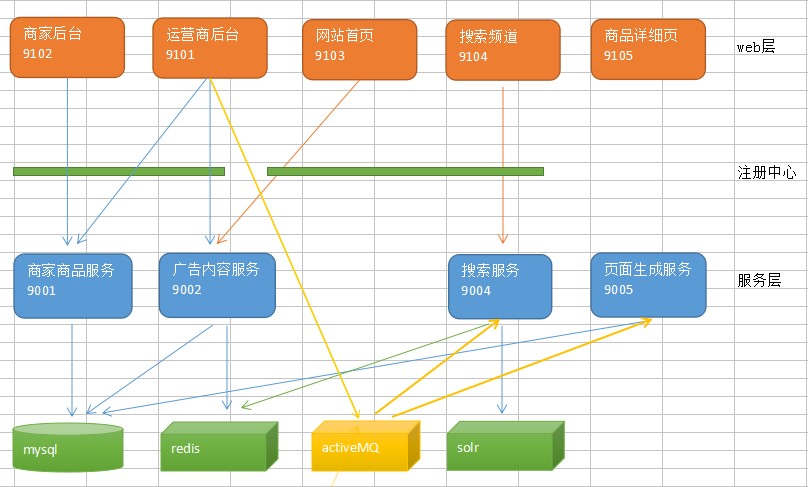
## 今日目标

昨天学习了网页静态化：直接一个问题。什么时候产生静态网页呢？

今天主要学习ActiveMQ，这个来帮助我们解决我们的业务逻辑问题

## 系统模块分析.avi

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。

## 消息中间件与JMS

我们上次课已经讲解了通过 Freemarker技术实现商品详细页的静态化，那么**在什么时候去做静态化操作呢？**

我们定义的业务规则是**商品审核后**才能展示，所以我们**需要在商品审核后去静态化生成**。由于我们采用的是**分布式架构和负载均衡**，为了分担访问压力，我们会部署多个商品详细页工程，这些工程是完全相同的。我们应该怎么去同时让这几台服务器来生成网页呢？这就需要一个消息中间件来做消息的中转操作。

**我们的商家商品服务将消息（商品ID）发送给消息中间件，另外的商品详细页生成的工程监听消息中间件的消息，一旦有消息立刻接收过来，然后按照一定的逻辑来处理**。

## JMS介绍

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。根据有效负载 的类型来划分，可以将消息分为几种类型，它们分别携带：简单文本 (TextMessage)、可序列化的对象 (ObjectMessage)、属性集合 (MapMessage)、字节流 (BytesMessage)、原始值流 (StreamMessage)，还有无有效负载的消息 (Message)。

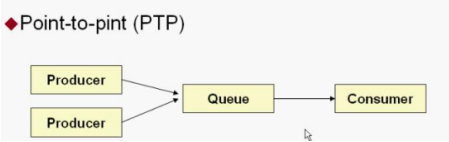
# activeMq的两消息方式 (视频05~10) [掌握]

## 05.ActiveMQ消息传递类型

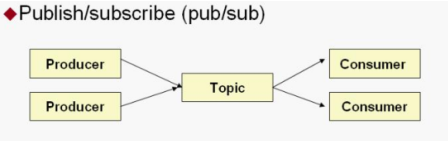
ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现，尽管JMS规范出台已经是很久的事情了，但是JMS在当今的J2EE应用中间仍然扮演着特殊的地位。

对于消息的传递有两种类型：

1、一种是点对点的，即一个**生产者**和一个**消费者**一一对应；



2、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。



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。

· StreamMessage -- Java 原始值的数据流

· MapMessage--一套名称-值对

· **TextMessage--一个字符串对象**

· ObjectMessage--一个序列化的 Java 对象

· BytesMessage--一个字节的数据流

## 06、ActiveMQ安装

先要有JDK、上传包、解压、进入bin

./activemq start

访问地址：http://192.168.25.168:8161/admin/

用户名密码：admin/admin



## 07、入门小Demo-点对点-消息生产者

用Java代码连接ActiveMQ，它的链接地址是：**tcp://192.168.25.168:61616**

**这个类里面引入类型，大部分是JMS包下面的**

|  |
| --- |
| // 4.获取session  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***); |

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

**public class** QueueProducer {  
 **public static void** main(String[] args) **throws** JMSException {  
 *//1.创建连接工厂* ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory(**"tcp://192.168.25.168:61616"**);  
 *//2.创建连接* Connection connection = connectionFactory.createConnection();  
 *//3.启动连接* connection.start();  
 *//4.获取session(会话对象) 参数1：是否启动事务 参数2：消息确认方式* Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  
 *//5.创建队列对象* Queue queue = session.createQueue(**"test-queue"**);  
 *//6.创建消息生产者对象* MessageProducer producer = session.createProducer(queue);  
 *//7.创建消息对象（文本消息）* TextMessage textMessage = session.createTextMessage(**"欢迎来到品优购世界"**);  
 *//8.发送消息* producer.send(textMessage);  
 *//9.关闭资源* producer.close();  
 session.close();  
 connection.close();  
 }  
}

## 08、入门小Demo-点对点-消息消费者

我们接收到的信息要进行处理

**public class** QueueConsumer {  
 **public static void** main(String[] args) **throws** JMSException, IOException {  
 *//1.创建连接工厂* ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory(**"tcp://192.168.25.168:61616"**);  
 *//2.创建连接* Connection connection = connectionFactory.createConnection();  
 *//3.启动连接* connection.start();  
 *//4.获取session(会话对象) 参数1：是否启动事务 参数2：消息确认方式* Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  
 *//5.创建队列对象* Queue queue = session.createQueue(**"test-queue"**);  
 *//6.创建消息消费者对象* MessageConsumer consumer = session.createConsumer(queue);  
 *//7.设置监听* consumer.setMessageListener(**new** MessageListener() {  
   
 **public void** onMessage(Message message) {  
 TextMessage textMessage=(TextMessage)message;  
 **try** {  
 System.***out***.println(**"提取的消息："**+ textMessage.getText() );  
 } **catch** (JMSException e) {   
 e.printStackTrace();  
 }  
   
 }  
 });  
 *//8.等待键盘输入* System.***in***.read();  
 *//9.关闭资源* consumer.close();  
 session.close();  
 connection.close();  
 }  
}

这里面取出信息过后，activeMq里面就没有了，所以说点对点是一个生成者一个消费者，默认会缓存到消息队列上，但一但被消费了，就没有了。（只有一个消费者能够消费这一条消息）

## 09、入门小demo-发布订阅-消息生产者

发布订阅模式又叫广播，我们现在就发个广播

**public class** TopicProducer {  
 **public static void** main(String[] args) **throws** JMSException {  
 *//1.创建连接工厂* ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory(**"tcp://192.168.25.168:61616"**);  
 *//2.创建连接* Connection connection = connectionFactory.createConnection();  
 *//3.启动连接* connection.start();  
 *//4.获取session(会话对象) 参数1：是否启动事务 参数2：消息确认方式* Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  
 *//5.创建主题对象* Topic topic = session.createTopic(**"test-topic"**);   
 *//6.创建消息生产者对象* MessageProducer producer = session.createProducer(topic);  
 *//7.创建消息对象（文本消息）* TextMessage textMessage = session.createTextMessage(**"欢迎来到申请的品优购世界"**);  
 *//8.发送消息* producer.send(textMessage);  
 *//9.关闭资源* producer.close();  
 session.close();  
 connection.close();  
 }  
}

这个广播发完就完了，有没有人听到呢？没有，再听还能听到吗？不行。因为不会默认保存到消息队列上。

## 10、入门小Demo-发布订阅-消息消费者

打开我们的收音机：

**public class** TopicConsumer {  
 **public static void** main(String[] args) **throws** JMSException, IOException {  
 *//1.创建连接工厂* ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory(**"tcp://192.168.25.168:61616"**);  
 *//2.创建连接* Connection connection = connectionFactory.createConnection();  
 *//3.启动连接* connection.start();  
 *//4.获取session(会话对象) 参数1：是否启动事务 参数2：消息确认方式* Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  
 *//5.创建主题对象* Topic topic = session.createTopic(**"test-topic"**);   
 *//6.创建消息消费者对象* MessageConsumer consumer = session.createConsumer(topic);  
 *//7.设置监听* consumer.setMessageListener(**new** MessageListener() {  
   
 **public void** onMessage(Message message) {  
 TextMessage textMessage=(TextMessage)message;  
 **try** {  
 System.***out***.println(**"提取的消息："**+ textMessage.getText() );  
 } **catch** (JMSException e) {   
 e.printStackTrace();  
 }  
 }  
 });  
 *//8.等待键盘输入* System.***in***.read();  
 *//9.关闭资源* consumer.close();  
 session.close();  
 connection.close();  
 }  
}

在运行收音机后，再次执行上面的广播：“中国中央电视台，中国中央电视台,欢迎您收2019年春节联欢晚会！”，收音机就可以听到信息了。

# activeMq与Spring整合(视频11~12) [重点]

## 11、spring与activeMQ整合-点对点-消息生产者

与spring整合要先进行配置文件applicationContext-jms-producer.xml

*<!-- 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://192.168.25.168:61616"**/>  
</**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**>   
*<!--这个是队列目的地，点对点的 文本信息-->*<**bean id="queueTextDestination" class="org.apache.activemq.command.ActiveMQQueue"**>   
 <**constructor-arg value="queue\_text"**/>   
</**bean**>

测试

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext-jms-producer.xml"**)  
**public class** QueueProducer {  
  
 @Autowired  
 **private** JmsTemplate **jmsTemplate**;  
 @Autowired  
 **private** Destination **queueTextDestination**;  
 */\*\*  
 \* 发送文本消息  
 \*/* @Test  
 **public void** sendTextMessage(){  
 String text = **"spring JMS 点对点"**;  
 **jmsTemplate**.send(**queueTextDestination**, **new** MessageCreator() {  
 **public** Message createMessage(Session session) **throws** JMSException {  
 **return** session.createTextMessage(text);  
 }  
 });  
 }  
}

## 12、springJMS-点对点-消息消费者

spring文件

*<!-- 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://192.168.25.168:61616"**/>  
</**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**>   
*<!--这个是队列目的地，点对点的 文本信息-->*<**bean id="queueTextDestination" class="org.apache.activemq.command.ActiveMQQueue"**>   
 <**constructor-arg value="queue\_text"**/>   
</**bean**>   
*<!-- 我的监听类 -->*<**bean id="myMessageListener" class="cn.itcast.demo.MyMessageListener"**></**bean**>  
*<!-- 消息监听容器 -->*<**bean class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>  
 <**property name="connectionFactory" ref="connectionFactory"** />  
 <**property name="destination" ref="queueTextDestination"** />  
 <**property name="messageListener" ref="myMessageListener"** />  
</**bean**>

其中我的监听类：是我们自己的类实现了MessageListener接口

**public class** MyMessageListener **implements** MessageListener {  
 **public void** onMessage(Message message) {  
 TextMessage textMessage=(TextMessage)message;  
 **try** {  
 System.***out***.println(**"接收到消息："**+textMessage.getText());  
 } **catch** (JMSException e) {  
 e.printStackTrace();  
 }  
 }  
}

//消费者消费消息

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext-jms-consumer-queue.xml"**)  
**public class** TestQueue {  
 @Test  
 **public void** testQueue(){  
 **try** {  
 System.***in***.read();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

## 13、spring与activeMQ整合-发布订阅模式-消息生产者

在applicationContext-jms-producer.xml配置文件上，添加一个目的地：

*<!--这个是订阅模式 文本信息-->*<**bean id="topicTextDestination" class="org.apache.activemq.command.ActiveMQTopic"**>   
 <**constructor-arg value="topic\_text"**/>   
</**bean**>

**发送消息**

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext-jms-producer.xml"**)  
**public class** TopicProducer {  
 @Autowired  
 **private** JmsTemplate **jmsTemplate**;  
 @Autowired  
 **private** Destination **topicTextDestination**;  
 */\*\*  
 \* 发送文本消息  
 \*/* @Test  
 **public void** sendTextMessage(){  
 String text = **"spring JMS 发布订阅"**;  
 **jmsTemplate**.send(**topicTextDestination**, **new** MessageCreator() {  
   
 **public** Message createMessage(Session session) **throws** JMSException {  
   
 **return** session.createTextMessage(text);  
 }  
 });  
 }  
}

## 14、spring与activeMQ整合-发布订阅模式-消息消费者

在applicationContext-jms-consumer-topic.xml配置文件里面再添加一个广播的监听当然还有目的地

*<!-- 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://192.168.25.168:61616"**/>  
</**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**>   
*<!--这个是队列目的地，点对点的 文本信息-->*<**bean id="topicTextDestination" class="org.apache.activemq.command.ActiveMQTopic"**>   
 <**constructor-arg value="topic\_text"**/>   
</**bean**>   
*<!-- 我的监听类 -->*<**bean id="myMessageListener" class="cn.itcast.demo.MyMessageListener"**></**bean**>  
*<!-- 消息监听容器 -->*<**bean class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>  
 <**property name="connectionFactory" ref="connectionFactory"** />  
 <**property name="destination" ref="topicTextDestination"** />  
 <**property name="messageListener" ref="myMessageListener"** />  
</**bean**>

TopicLinstener就和原来的Linstener一样

@RunWith(SpringJUnit4ClassRunner.**class**)  
@ContextConfiguration(locations=**"classpath:applicationContext-jms-consumer-topic.xml"**)  
**public class** TestTopic {  
 @Test  
 **public void** testTopic(){  
 **try** {  
 System.***in***.read();  
 } **catch** (IOException e) {e.printStackTrace();  
 }  
 }  
}

# 用activeMq运营商后台与搜索服务的零耦合

**(视频15~18) [撑握]**

运用消息中间件 activeMQ 实现运营商后台与搜索服务的零耦合。运营商执行商品审核后，向 activeMQ 发送消息（SKU 列表），搜索服务从 activeMQ 接收到消息并导入到 solr 索引库。

## 15、商品审核导入索引库-消息生产者-1

**解除耦合**

修改 pinyougou-manager-web，移除搜索服务接口依赖：

<**dependency**>  
 <**groupId**>com.pinyougou</**groupId**>  
 <**artifactId**>pinyougou-search-interface</**artifactId**>  
 <**version**>0.0.3-SNAPSHOT</**version**>  
</**dependency**>

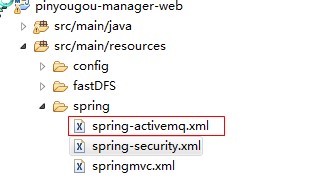
GoodsController.java 中删除调用搜索服务接口的相关代码

|  |
| --- |
| //itemSearchService.deleteByGoodsIds(Arrays.asList(ids));  //@Reference  //private ItemSearchService itemSearchService;  //itemSearchService.importList(itemList); |

修改 pinyougou-manager-web 的 pom.xml,引入依赖

<**dependency**>  
 <**groupId**>org.apache.activemq</**groupId**>  
 <**artifactId**>activemq-client</**artifactId**>  
 <**version**>5.13.4</**version**>  
</**dependency**>

pinyougou-sellergoods-service 工程添加 applicationContext-jms-producer.xml



改名为 spring-activemq.xml 内容如下：

*<!-- 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://192.168.25.168:61616"**/>  
</**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**>   
*<!--这个是队列目的地，导入索引库-->*<**bean id="queueSolrDestination" class="org.apache.activemq.command.ActiveMQQueue"**>   
 <**constructor-arg value="pinyougou\_queue\_solr"**/>   
</**bean**>

修改 web.xml

<**context-param**>  
<**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring/spring-\*.xml</**param-value**>

</**context-param**>

## 16.商品审核导入索引库-消息生产者-2.avi

修改 pinyougou-manager-web 工程的 GoodsController.java

@Autowired  
**private** JmsTemplate **jmsTemplate**;  
@Autowired  
**private** Destination **queueSolrDestination**;*//用于导入solr索引库的消息目标（点对点）*@RequestMapping(**"/updateStatus"**)  
**public** Result updateStatus(Long[] ids,String status){  
 **try** {  
 **goodsService**.updateStatus(ids, status);  
 **if**(**"2"**.equals(status)){*//如果是审核通过   
 //\*\*\*\*\*导入到索引库  
 //得到需要导入的SKU列表* List<TbItem> itemList = **goodsService**.findItemListByGoodsIdListAndStatus(ids, status);  
 *//导入到solr  
 //itemSearchService.importList(itemList);* **final** String jsonString = JSON.*toJSONString*(itemList);*//转换为json传输* **jmsTemplate**.send(**queueSolrDestination**, **new** MessageCreator() {  
 @Override  
 **public** Message createMessage(Session session) **throws** JMSException {  
 **return** session.createTextMessage(jsonString);  
 }  
 });  
 }   
 **return new** Result(**true**, **"修改状态成功"**);   
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**, **"修改状态失败"**);  
 }  
}

## 17.商品审核导入索引库-消息消费者

**修改 pinyougou-search-service ，在 pom.xml 中添加 activemq 依赖**

<**dependency**>  
 <**groupId**>org.apache.activemq</**groupId**>  
 <**artifactId**>activemq-client</**artifactId**>  
 <**version**>5.13.4</**version**>  
</**dependency**>

**添加 spring 配置文件 applicationContext-jms-consumer.xml**

*<!-- 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://192.168.25.168:61616"**/>   
</**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**>   
*<!--这个是队列目的地，导入索引库-->*<**bean id="queueSolrDestination" class="org.apache.activemq.command.ActiveMQQueue"**>   
 <**constructor-arg value="pinyougou\_queue\_solr"**/>   
</**bean**>  
*<!-- 消息监听容器 导入索引库-->*<**bean class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>  
 <**property name="connectionFactory" ref="connectionFactory"** />  
 <**property name="destination" ref="queueSolrDestination"** />  
 <**property name="messageListener" ref="itemSearchListener"** />  
</**bean**>

在 pinyougou-search-service 的 com.pinyougou.search.service.impl 新增监听类

@Component  
**public class** ItemSearchListener **implements** MessageListener {  
 @Autowired  
 **private** ItemSearchService **itemSearchService**;  
 @Override  
 **public void** onMessage(Message message) {  
 TextMessage textMessage=(TextMessage)message;  
 **try** {  
 String text = textMessage.getText();*//json字符串* System.***out***.println(**"监听到消息:"**+text);  
 List<TbItem> itemList = JSON.*parseArray*(text, TbItem.**class**);  
 **itemSearchService**.importList(itemList);  
 System.***out***.println(**"导入到solr索引库"**);  
 } **catch** (JMSException e) {  
 e.printStackTrace();  
 }  
 }  
}

## 18.商品删除移除索引库记录

通过消息中间件实现在商品删除时也同时移除索引库记录的功能

修改 pinyougou-manager-web 工程的 spring-activemq.xml，添加 bean 配置

*<!--这个是队列目的地，导入索引库-->*<**bean id="queueSolrDeleteDestination" class="org.apache.activemq.command.ActiveMQQueue"**>   
 <**constructor-arg value="pinyougou\_queue\_solr\_delete"**/>   
</**bean**>

修改 GoodsController.java

@Autowired  
**private** Destination **queueSolrDeleteDestination**;

*/\*\*  
 \* 批量删除  
 \** ***@param ids*** *\** ***@return*** *\*/*@RequestMapping(**"/delete"**)  
**public** Result delete(**final** Long [] ids){  
 **try** {  
 **goodsService**.delete(ids);  
 *//从索引库中删除  
 //itemSearchService.deleteByGoodsIds(Arrays.asList(ids));* **jmsTemplate**.send(**queueSolrDeleteDestination**, **new** MessageCreator() {  
 @Override  
 **public** Message createMessage(Session session) **throws** JMSException {  
 **return** session.createObjectMessage(ids);  
 }  
 });   
 **return new** Result(**true**, **"删除成功"**);   
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**, **"删除失败"**);  
 }  
}

修改 pinyougou-search-service 的 applicationContext-activemq-consumer.xml

*<!--这个是队列目的地，导入索引库-->*<**bean id="queueSolrDeleteDestination" class="org.apache.activemq.command.ActiveMQQueue"**>   
 <**constructor-arg value="pinyougou\_queue\_solr\_delete"**/>   
</**bean**>   
*<!-- 消息监听容器 导入索引库-->*<**bean class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>  
 <**property name="connectionFactory" ref="connectionFactory"** />  
 <**property name="destination" ref="queueSolrDeleteDestination"** />  
 <**property name="messageListener" ref="itemDeleteListener"** />  
</**bean**>

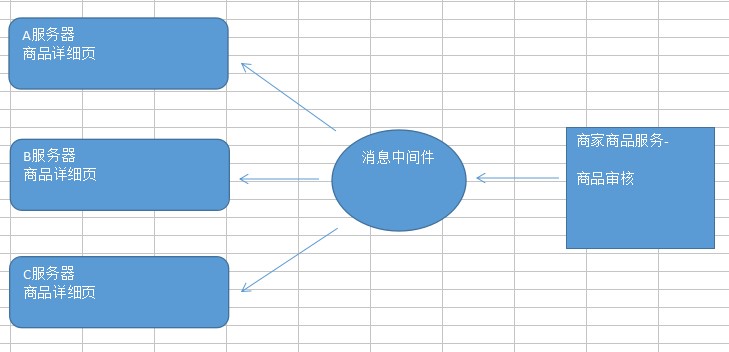
com.pinyougou.search.service.impl 包下创建监听类 ItemDeleteListener

@Component  
**public class** ItemDeleteListener **implements** MessageListener {  
 @Autowired  
 **private** ItemSearchService **itemSearchService**;  
 @Override  
 **public void** onMessage(Message message) {  
 ObjectMessage objectMessage =(ObjectMessage)message;  
 **try** {  
 Long[] goodsIds= (Long[]) objectMessage.getObject();  
 System.***out***.println(**"监听获取到消息："**+goodsIds);  
 **itemSearchService**.deleteByGoodsIds(Arrays.*asList*(goodsIds));  
 System.***out***.println(**"执行索引库删除"**);  
 } **catch** (JMSException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 }  
}

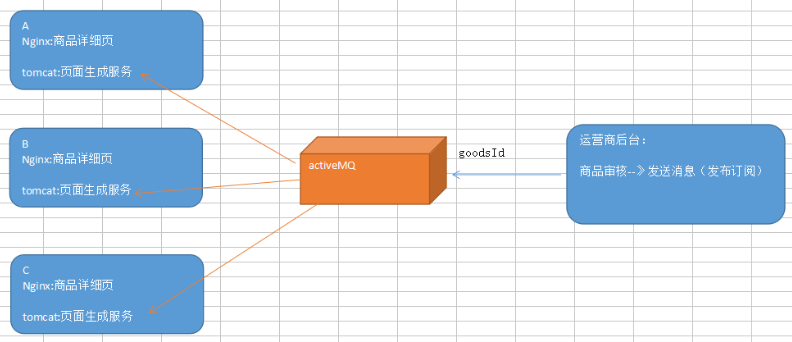
# 用activeMq运营商后台与页面静态化服务的零耦合

**(视频19~23) [撑握]**

运用消息中间件 activeMQ 实现运营商后台与网页生成服务的零耦合。运营商执行商品审核后，向 activeMQ 发送消息（商品 ID），网页生成服务从 activeMQ 接收到消息后执行网页生成操作。



## 19.商品审核执行网页静态化-架构分析



## 20.商品审核生成详细页-消息生产者

修改 pinyougou-manager-web，移除网页生成服务接口依赖

<**dependency**>  
 <**groupId**>com.pinyougou</**groupId**>  
 <**artifactId**>pinyougou-page-interface</**artifactId**>  
 <**version**>0.0.3-SNAPSHOT</**version**>  
</**dependency**>

GoodsController.java 中删除调用网页生成服务接口的相关代码

*//@Reference(timeout=40000)  
//private ItemPageService itemPageService;*

*//itemPageService.genItemHtml(goodsId);*

修改配置文件 spring-activemq.xml，添加配置

*<!--这个是订阅模式 生成商品详细页-->*<**bean id="topicPageDestination" class="org.apache.activemq.command.ActiveMQTopic"**>   
 <**constructor-arg value="pinyougou\_topic\_page"**/>   
</**bean**>

修改 pinyougou-manager-web 的 GoodsController.java

@RequestMapping(**"/updateStatus"**)  
**public** Result updateStatus(Long[] ids,String status){  
 **try** {  
 **goodsService**.updateStatus(ids, status);  
   
 **if**(**"1"**.equals(status)){*//如果是审核通过   
 //\*\*\*\*\*导入到索引库  
 //得到需要导入的SKU列表* …   
 *//\*\*\*\*生成商品详细页* **for**(**final** Long goodsId:ids){  
 *// itemPageService.genItemHtml(goodsId);* **jmsTemplate**.send(**topicPageDestination**, **new** MessageCreator() {  
 @Override  
 **public** Message createMessage(Session session) **throws** JMSException {   
 **return** session.createTextMessage(goodsId+**""**);  
 }  
 });   
 }  
 }   
 **return new** Result(**true**, **"修改状态成功"**);   
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**, **"修改状态失败"**);  
 }  
}

## 21.商品审核生成详细页-消息消费者

1、修改工程 pinyougou-page-service ，删除 dubbox 相关依赖

*<!-- dubbo相关 -->*<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>dubbo</**artifactId**>   
</**dependency**>  
<**dependency**>  
 <**groupId**>org.apache.zookeeper</**groupId**>  
 <**artifactId**>zookeeper</**artifactId**>   
</**dependency**>  
<**dependency**>  
 <**groupId**>com.github.sgroschupf</**groupId**>  
 <**artifactId**>zkclient</**artifactId**>   
</**dependency**>

2、修改 applicationContext-service.xml，删除 dubbox 相关配置

<**dubbo:protocol name="dubbo" port="20885"**></**dubbo:protocol**>  
<**dubbo:application name="pinyougou-page-service"**/>  
<**dubbo:registry address="zookeeper://192.168.25.168:2181"**/>  
<**dubbo:annotation package="com.pinyougou.page.service.impl"** />

3、修改 ItemPageServiceImpl 类的@Service 注解 为 org.springframework.stereotype.Service 包下的@Service 注解

4、修改 applicationContext-service.xml，添加配置

<**context:component-scan base-package="com.pinyougou.page.service.impl"**/>

5、pom.xml 中引入 activeMQ 客户端的依赖

<**dependency**>  
 <**groupId**>org.apache.activemq</**groupId**>  
 <**artifactId**>activemq-client</**artifactId**>  
 <**version**>5.13.4</**version**>  
</**dependency**>

6、添加 spring 配置文件 applicationContext-jms-consumer.xml

*<!-- 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://192.168.25.168:61616"**/>  
</**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**>   
*<!--这个是主题目的 生成页面-->*<**bean id="topicPageDestination" class="org.apache.activemq.command.ActiveMQTopic"**>   
 <**constructor-arg value="pinyougou\_topic\_page"**/>   
</**bean**>  
*<!-- 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="topicPageDestination"** />  
 <**property name="messageListener" ref="pageListener"** />  
</**bean**>

创建消息监听类 PageListener

@Component  
**public class** PageListener **implements** MessageListener {  
 @Autowired  
 **private** ItemPageService **itemPageService**;  
 @Override  
 **public void** onMessage(Message message) {  
 TextMessage textMessage=(TextMessage)message;  
 **try** {  
 String text = textMessage.getText();  
 System.***out***.println(**"接收到消息："**+text);  
 **boolean** b = **itemPageService**.genItemHtml(Long.*parseLong*(text));  
 System.***out***.println(**"网页生成结果："**+b);  
 } **catch** (JMSException e) {  
 e.printStackTrace();  
 }  
 }  
}

## 22.商品审核生成详细页-测试

## 23.商品删除-删除商品详细页

执行商品删除后，同时删除每个服务器上的商品详细页

运营商后台修改 spring-activemq.xml，添加配置

*<!--这个是订阅模式 删除商品详细页-->*<**bean id="topicPageDeleteDestination" class="org.apache.activemq.command.ActiveMQTopic"**>   
 <**constructor-arg value="pinyougou\_topic\_page\_delete"**/>   
</**bean**>

修改 GoodsController.java

@Autowired  
**private** Destination **topicPageDeleteDestination**;

@RequestMapping(**"/delete"**)  
**public** Result delete(**final** Long [] ids){  
 **try** {  
 **goodsService**.delete(ids);  
   
 *//从索引库中删除  
 //itemSearchService.deleteByGoodsIds(Arrays.asList(ids));* **jmsTemplate**.send(**queueSolrDeleteDestination**, **new** MessageCreator() {  
   
 @Override  
 **public** Message createMessage(Session session) **throws** JMSException {  
 **return** session.createObjectMessage(ids);  
 }  
 });  
   
 *//删除每个服务器上的商品详细页* **jmsTemplate**.send(**topicPageDeleteDestination**, **new** MessageCreator() {  
   
 @Override  
 **public** Message createMessage(Session session) **throws** JMSException {  
 **return** session.createObjectMessage(ids);  
 }  
 });  
   
 **return new** Result(**true**, **"删除成功"**);   
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**, **"删除失败"**);  
 }  
}

修改 pinyougou-page-service 的 applicationContext-activemq-consumer.xml

*<!--这个是主题目的 删除页面-->*<**bean id="topicPageDeleteDestination" class="org.apache.activemq.command.ActiveMQTopic"**>   
 <**constructor-arg value="pinyougou\_topic\_page\_delete"**/>   
</**bean**>   
*<!-- 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="topicPageDeleteDestination"** />  
 <**property name="messageListener" ref="pageDeleteListener"** />  
</**bean**>

修改 pinyougou-page-interface 的 ItemPageService.java

@Override  
**public boolean** deleteItemHtml(Long[] goodsIds) {  
 **try** {  
 **for**(Long goodsId:goodsIds){  
 **new** File(**pagedir**+goodsId+**".html"**).delete();   
 }  
 **return true**;  
 } **catch** (Exception e) {e.printStackTrace();  
 **return false**;  
 }   
}

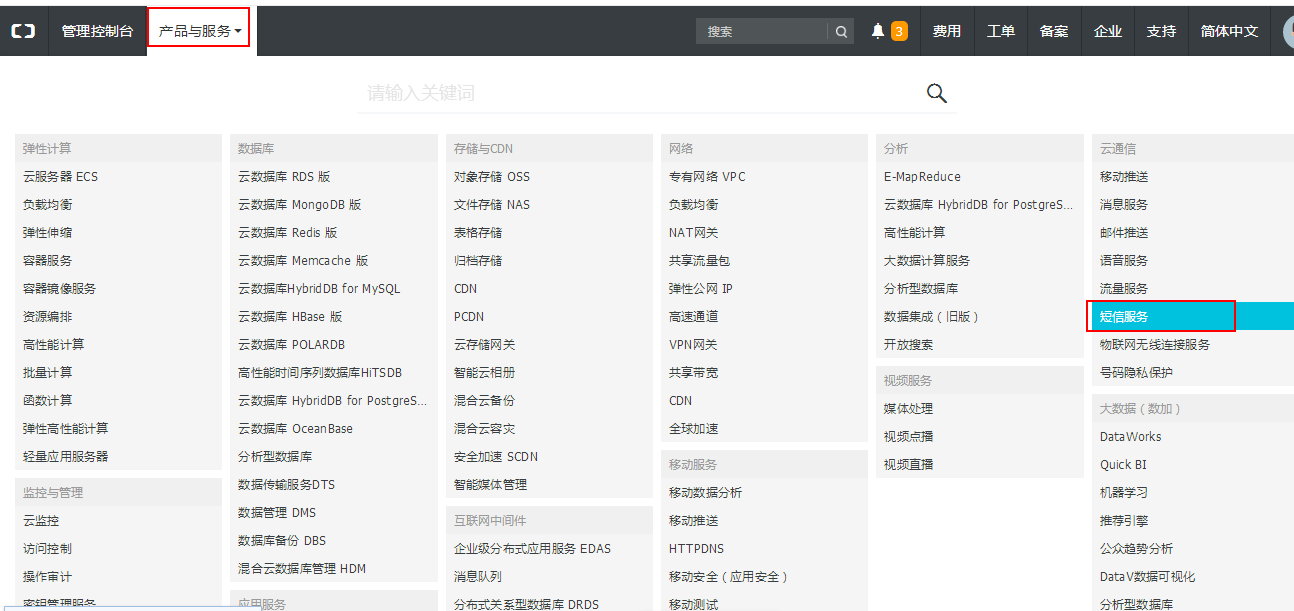
创建监听类 PageDeleteListener

@Component  
**public class** PageDeleteListener **implements** MessageListener {  
 @Autowired  
 **private** ItemPageService **itemPageService**;  
 @Override  
 **public void** onMessage(Message message) {  
 ObjectMessage objectMessage =(ObjectMessage)message;  
 **try** {  
 Long [] goodsIds= (Long[]) objectMessage.getObject();  
 System.***out***.println(**"接收到消息:"**+goodsIds);  
 **boolean** b = **itemPageService**.deleteItemHtml(goodsIds);   
 System.***out***.println(**"删除网页："**+b);  
 } **catch** (JMSException e) {  
 e.printStackTrace();  
 }  
 }  
}

# 下集预告

下回我们要学习：springboot,activemq，阿里大于，要自己注册一个帐号

[www.alidayu.com](http://www.baidu.com/baidu.php?url=ZzmK000GQJC8hW1POcJEP4KEa9MqOc64Kv9VaSYkPBGRtKKgrM9zjpMjkg0QSyMnQxKRQWKjPMwVPRtuZHdquplwvyTojNksmJ4fXyH6PR2oD2XMXttLlESx3RspZ_3aMmqjEv4UOpBl4_suPNxrqoJgxNO3dD62O8C6tR-EGupmJLpXA6.7D_iuTZ-GYkYApacrSapBsAprFMw6wKL9JuBBHwmEukmr_X5is1gsdeEQx3txQ_erMktJOAEWkl_LT8FB8zzEWgEvNSQQPLuy--rexduYt8FWojG4THZgZG4THZy7MHWuxJB4THAe_tpqOudEmtxZ_L2N9h9mLU2Srqf.U1Yk0ZDq_2Q0eeH1YSn0IjvsC_jEV55H3sKGUHYznWR0u1dBuHms0ZNG5yF9pywd0ZKGujYk0APGujY1rjR0UgfqnH0YP7tknjDLg1DsnH-xn1msnfKopHYs0ZFY5Hm4P0K-pyfqnHfdndtznHDzPdtkP10snNtznW01PNtznWDLPdtznWDdPsKBpHYznjf0UynqnHR4PWRzPWbdP-t1PWn4rHmknj9xn1mYPjbsnHDYg100TgKGujYs0Z7Wpyfqn0KzuLw9u1Ys0AqvUjYzPWm4Qywlg1cvPW-mQywlg1msn1nVn7t3PHbVnsK9mWYsg100ugFM5H00TZ0qn0K8IM0qna3snj0snj0sn0KVIZ0qn0KbuAqs5H00ThCqn0KbugmqTAn0uMfqn0KspjYs0Aq15H00mMTqnH00UMfqn0K1XWY0IZN15HDLPjnzPjnzP1b4rjnzP1RvPWD10ZF-TgfqnHf3rjcYPjDLPjmvP6K1pyfqmyPbryNBmH6snjK9ujRdnsKWTvYqrjuAwjNarDRdnH03rH7aPsK9m1Yk0ZK85H00TydY5H00Tyd15H00XMfqn0KVmdqhThqV5HDsPjuxn7tsg100uA78IyF-gLK_my4GuZnqn7tsg1Kxn0KbIA-b5H00ugwGujYVnfK9TLKWm1Ys0ZNspy4Wm1Ys0Z7VuWYs0AuWIgfqn0KhXh6qn0KlTAkdT1Ys0A7buhk9u1Yk0APzm1YznWnz&us=0.0.0.0.0.0.10&us=0.0.0.0.0.0.30&ck=0.0.0.0.0.0.0.0&shh=www.baidu.com)  （https://dayu.aliyun.com/）



# 总结：

1. activeMq发送消息的方式
2. actieMq的具体业务使用