**品优购电商系统开发**

**第12章**

**网页静态化解决方案-Freemarker**

传智播客.黑马程序员

# 课程目标

目标1：掌握Freemarker常用的指令与内建函数

目标2：完成商品详细页的数据显示

目标3：完成商品详细页的动态效果

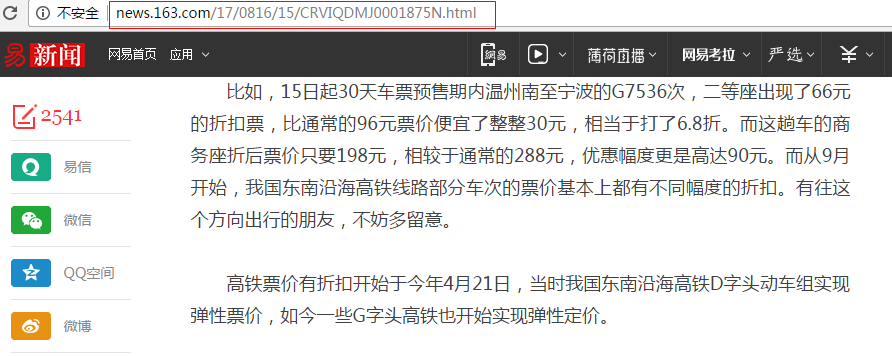
目标4：完成商品详细页读取SKU信息的业务逻辑

目标5：完成商品审核调用功能

# 1.网页静态化技术Freemarker

## 1.1为什么要使用网页静态化技术

网页静态化解决方案在实际开发中运用比较多，例如新闻网站，门户网站中的新闻频道或者是文章类的频道。



对于电商网站的商品详细页来说，至少几百万个商品，每个商品又有大量的信息，这样的情况同样也适用于使用网页静态化来解决。

网页静态化技术和缓存技术的共同点都是为了减轻数据库的访问压力，但是具体的应用场景不同，缓存比较适合小规模的数据，而网页静态化比较适合大规模且相对变化不太频繁的数据。另外网页静态化还有利于SEO。

另外我们如果将网页以纯静态化的形式展现，就可以使用Nginx这样的高性能的web服务器来部署。Nginx可以承载5万的并发，而Tomcat只有500。

什么是网页静态化：

将一些动态的页面，通过一些技术手段，转成静态的页面，这就是网页静态化。

为甚么要网页静态化：

为了提升系统的响应。（主要的目的）

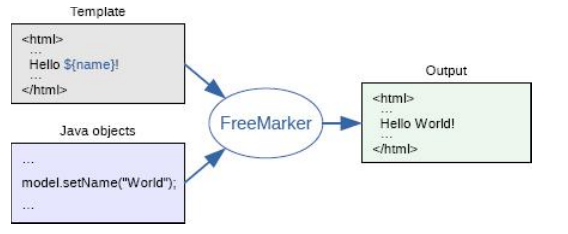
怎么实现网页静态化，就可以使用freemarker.

## 1.2什么是 Freemarker

模板引擎（velocity, autocode,thymleaf）

FreeMarker 是一个用 Java 语言编写的模板引擎(jar)，它基于模板来生成文本输出。FreeMarker与 Web 容器无关，即在 Web 运行时，它并不知道 Servlet 或 HTTP。它不仅可以用作表现层的实现技术，而且还可以用于生成 XML，JSP 或 Java 等。

模板 + 数据集 = html



## 1.3 Freemarker入门

### 1.3.1工程引入依赖

|  |
| --- |
| <dependency>  <groupId>org.freemarker</groupId>  <artifactId>freemarker</artifactId>  <version>2.3.23</version>  </dependency> |

### 1.3.2创建模板文件

模板文件中四种元素

  1、文本，直接输出的部分  
  2、注释，即<#--...-->格式不会输出  
  3、插值（Interpolation）：即${..}部分,将使用数据模型中的部分替代输出  
  4、FTL指令：FreeMarker指令，和HTML标记类似，名字前加#予以区分，不会输出。

我们现在就创建一个简单的创建模板文件test.ftl

|  |
| --- |
| <html>  <head>  <meta charset="utf-8">  <title>Freemarker入门小DEMO </title>  </head>  <body>  <#--我只是一个注释，我不会有任何输出 -->  ${name},你好。${message}  </body>  </html> |

这里有文本、插值和注释

### 1.3.3生成文件

使用步骤：

第一步：创建一个 Configuration 对象，直接 new 一个对象。构造方法的参数就是 freemarker的版本号。

第二步：设置模板文件所在的路径。

第三步：设置模板文件使用的字符集。一般就是 utf-8.

第四步：加载一个模板，创建一个模板对象。

第五步：创建一个模板使用的数据集，可以是 pojo 也可以是 map。一般是 Map。

第六步：创建一个 Writer 对象，一般创建一 FileWriter 对象，指定生成的文件名。

第七步：调用模板对象的 process 方法输出文件。

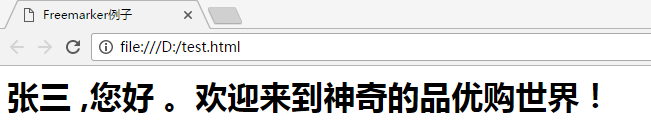
第八步：关闭流

代码：

创建Test类 main方法如下：

|  |
| --- |
| //1.创建配置类  Configuration configuration=**new** Configuration(Configuration.*getVersion*());  //2.设置模板所在的目录  configuration.setDirectoryForTemplateLoading(**new** File("D:/pinyougou\_work/freemarkerDemo/src/main/resources/"));  //3.设置字符集  configuration.setDefaultEncoding("utf-8");  //4.加载模板  Template template = configuration.getTemplate("test.ftl");  //5.创建数据模型  Map map=**new** HashMap();  map.put("name", "张三 ");  map.put("message", "欢迎来到神奇的品优购世界！");  //6.创建Writer对象  Writer out =**new** FileWriter(**new** File("d:\\test.html"));  //7.输出  template.process(map, out);  //8.关闭Writer对象  out.close(); |

执行后，在D盘根目录即可看到生成的test.html ，打开看看



## 1.4 FTL指令

### 1.4.1 assign指令

此指令用于在页面上定义一个变量

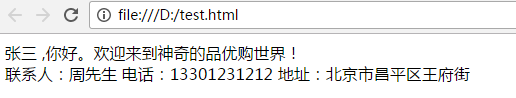
（1）定义简单类型：

|  |
| --- |
| <#assign linkman="周先生">  联系人：${linkman} |

（2）定义对象类型：

|  |
| --- |
| <#assign info={"mobile":"13301231212",'address':'北京市昌平区王府街'} >  电话：${info.mobile} 地址：${info.address} |

运行效果：



### 1.4.2 include指令

此指令用于模板文件的嵌套

创建模板文件head.ftl

|  |
| --- |
| <h1>黑马信息网</h1> |

我们修改test.ftl，在模板文件中使用include指令引入刚才我们建立的模板

|  |
| --- |
| <#include "head.ftl"> |

### 1.4.3 if指令

在模板文件上添加

|  |
| --- |
| <#if success==true>  你已通过实名认证  <#else>  你未通过实名认证  </#if> |

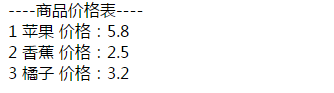
在代码中对str变量赋值

|  |
| --- |
| map.put("success", **true**); |

在freemarker的判断中，可以使用= 也可以使用==

### 1.4.4 list指令

需求，实现商品价格表，如下图：



（1）代码中对变量goodsList赋值

|  |
| --- |
| List goodsList=**new** ArrayList();  Mapgoods1=**new** HashMap();  goods1.put("name", "苹果");  goods1.put("price", 5.8);  Mapgoods2=**new**HashMap();  goods2.put("name", "香蕉");  goods2.put("price", 2.5);  Mapgoods3=**new**HashMap();  goods3.put("name", "橘子");  goods3.put("price", 3.2);  goodsList.add(goods1);  goodsList.add(goods2);  goodsList.add(goods3);  map.put("goodsList", goodsList); |

（2）在模板文件上添加

|  |
| --- |
| ----商品价格表----<br>  <#list goodsList as goods>  ${goods\_index+1} 商品名称： ${goods.name} 价格：${goods.price}<br>  </#list> |

如果想在循环中得到索引，使用循环变量+\_index就可以得到。

## 1.5 内建函数

内建函数语法格式： 变量+?+函数名称

### 1.5.1获取集合大小

我们通常要得到某个集合的大小，如下图：



我们使用size函数来实现，代码如下：

|  |
| --- |
| 共 ${goodsList?size} 条记录 |

### 1.5.2转换JSON字符串为对象

我们通常需要将json字符串转换为对象，那如何处理呢？看代码

|  |
| --- |
| <#assign text="{'bank':'工商银行','account':'10101920201920212'}" />  <#assign data=text?eval />  开户行：${data.bank} 账号：${data.account} |

### 1.5.3日期格式化

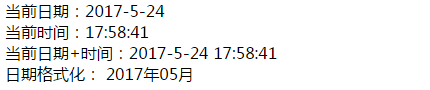
代码中对变量赋值：

|  |
| --- |
| dataModel.put("today", **new** Date()); |

在模板文件中加入

|  |
| --- |
| 当前日期：${today?date} <br>  当前时间：${today?time} <br>  当前日期+时间：${today?datetime} <br>  日期格式化： ${today?string("yyyy年MM月")} |

运行效果如下：



### 1.5.4数字去除千分位

代码中对变量赋值：

|  |
| --- |
| map.put("point", 102920122); |

修改模板：

|  |
| --- |
| 累计积分：${point} |

页面显示：



我们会发现数字会以每三位一个分隔符显示，有些时候我们不需要这个分隔符，就需要将数字转换为字符串,使用内建函数c

|  |
| --- |
| 累计积分：${point?c} |

页面显示效果如下：



## 1.6空值处理运算符

如果你在模板中使用了变量但是在代码中没有对变量赋值，那么运行生成时会抛出异常。但是有些时候，有的变量确实是null，怎么解决这个问题呢？

### 1.6.1判断某变量是否存在:“??”

用法为:variable??,如果该变量存在,返回true,否则返回false

|  |
| --- |
| <#if aaa??>  aaa变量存在  <#else>  aaa变量不存在  </#if> |

### 1.6.2缺失变量默认值:“!”

我们除了可以判断是否为空值，也可以使用!对null值做转换处理

在模板文件中加入

|  |
| --- |
| ${aaa!'-'} |

在代码中不对aaa赋值，也不会报错了 ，当aaa为null则返回！后边的内容-

## 1.7运算符

### 1.7.1算数运算符

FreeMarker表达式中完全支持算术运算,FreeMarker支持的算术运算符包括:+, - , \* , / , %

### 1.7.2逻辑运算符

逻辑运算符有如下几个:   
逻辑与:&&   
逻辑或:||   
逻辑非:!   
逻辑运算符只能作用于布尔值,否则将产生错误

### 1.7.3比较运算符

表达式中支持的比较运算符有如下几个:   
1=或者==:判断两个值是否相等.   
2!=:判断两个值是否不等.   
3>或者gt:判断左边值是否大于右边值   
4>=或者gte:判断左边值是否大于等于右边值   
5<或者lt:判断左边值是否小于右边值   
6<=或者lte:判断左边值是否小于等于右边值

**注意**:=和!=可以用于字符串,数值和日期来比较是否相等,但=和!=两边必须是相同类型的值,否则会产生错误,而且FreeMarker是精确比较,"x","x ","X"是不等的.其它的运行符可以作用于数字和日期,但不能作用于字符串,大部分的时候,使用gt等字母运算符代替>会有更好的效果,因为 FreeMarker会把>解释成FTL标签的结束字符,当然,也可以使用括号来避免这种情况,如:<#if (x>y)>

#### 品优购的商品详情的网页静态化的解决方案

参考：33.xlsx

思路：

创建一个工程pingyougou-page-service：用于生产静态页面

在producer方（pingyougou-manager-web）修改商品时发送消息

在pinyougou-page-service工程中 消费消息

接收消息 之后 获取最新数据 利用freemarker生成静态页面

实现：

加入依赖：

|  |
| --- |
| <**dependency**>  <**groupId**>org.freemarker</**groupId**>  <**artifactId**>freemarker</**artifactId**> </**dependency**> <**dependency**>  <**groupId**>org.apache.activemq</**groupId**>  <**artifactId**>activemq-client</**artifactId**>  <**version**>5.14.5</**version**> </**dependency**> |

pingyougou-page-service工程中核心代码：

|  |
| --- |
| **package** com.pinyougou.page.service.impl;  **import** com.alibaba.dubbo.config.annotation.Service; **import** com.pinyougou.mapper.TbGoodsDescMapper; **import** com.pinyougou.mapper.TbGoodsMapper; **import** com.pinyougou.page.service.PageService; **import** com.pinyougou.pojo.TbGoods; **import** com.pinyougou.pojo.TbGoodsDesc; **import** com.pinyougou.pojo.TbGoodsExample; **import** freemarker.template.Configuration; **import** freemarker.template.Template; **import** org.apache.http.impl.conn.Wire; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer;  **import** java.io.File; **import** java.io.FileWriter; **import** java.io.IOException; **import** java.util.HashMap; **import** java.util.List; **import** java.util.Map;  /\*\*  \* 在MQ的消费者端 监听消息 调用该方法生成静态页面  \*  \* **@author** 三国的包子  \* **@version** 1.0  \* **@package** com.pinyougou.page.service.impl  \* **@since** 1.0  \*/ @Service **public class** PageServiceImpl **implements** PageService {   @Autowired  **private** TbGoodsMapper **goodsMapper**;  @Autowired  **private** TbGoodsDescMapper **tbGoodsDescMapper**;   @Autowired  **private** FreeMarkerConfigurer **configurer**;   *//从数据库查询数据 并生成静态页面* @Override  **public boolean** genHtml(Long goodsid) {  *//3.从数据库中获取SPU的信息* TbGoodsExample example = **new** TbGoodsExample();  example.createCriteria().andIdEqualTo(goodsid).andIsDeleteEqualTo(**false**).andAuditStatusEqualTo(**"1"**);  List<TbGoods> tbGoods = **goodsMapper**.selectByExample(example);   **if**(tbGoods!=**null** && tbGoods.size()>0) {  TbGoodsDesc goodsDesc = **tbGoodsDescMapper**.selectByPrimaryKey(goodsid);  *//生成静态页面* genFreemarkerhtml(tbGoods.get(0),goodsDesc);  }  **return true**;  }   **private void** genFreemarkerhtml(TbGoods goods, TbGoodsDesc desc) {  FileWriter writer=**null**;  **try** {  *//模板 + 数据集 =html* Configuration configuration = **configurer**.getConfiguration();  Template template = configuration.getTemplate(**"item.ftl"**);   *//数据集* Map model = **new** HashMap();  model.put(**"goods"**,goods);  model.put(**"goodsDesc"**,desc);  *//输出* writer = **new** FileWriter(**new** File(**"G:\\freemarker\\"**+goods.getId()+**".html"**));  template.process(model,writer);  } **catch** (Exception e) {  e.printStackTrace();  }**finally** {  **if**(writer!=**null**){  **try** {  writer.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } } |

配置文件applicationContext-service.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  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 http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"**>    *<!--<context:component-scan base-package="com.pinyougou.sellergoods.service"></context:component-scan>-->* <**dubbo:application name="pinyougou-page-service"**></**dubbo:application**>  <**dubbo:registry address="192.168.25.128:2181" protocol="zookeeper"**></**dubbo:registry**>  <**dubbo:protocol name="dubbo" port="20883"**></**dubbo:protocol**>   <**dubbo:annotation package="com.pinyougou.page.service"**></**dubbo:annotation**>   <**dubbo:provider timeout="100000"**></**dubbo:provider**>   *<!--配置freemarker和spring的整合-->* <**bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"**>  <**property name="templateLoaderPath" value="/WEB-INF/template"**></**property**>  <**property name="defaultEncoding" value="utf-8"**></**property**>  </**bean**>  </**beans**> |

配置消费方配置文件：applicationContext-activemq-consumer.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.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://192.168.25.133:61616"**/>  </**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**>   <**bean id="solr\_update\_item\_queue" class="org.apache.activemq.command.ActiveMQQueue"**>  <**constructor-arg value="solr\_update\_item\_queue"**/>  </**bean**>     *<!--配置监听容器-->* <**bean id="solr\_update\_Listener" class="com.pinyougou.search.listener.SolrUpdateListener"**></**bean**>  *<!--配置监听容器 这个就是一个消费者-->* <**bean class="org.springframework.jms.listener.DefaultMessageListenerContainer"**>  <**property name="connectionFactory" ref="connectionFactory"** />  <**property name="destination" ref="solr\_update\_item\_queue"** />  <**property name="messageListener" ref="solr\_update\_Listener"** />  </**bean**> </**beans**> |

监听器代码 ：

|  |
| --- |
| **package** com.pinyougou.page.listener;  **import** com.pinyougou.page.service.PageService; **import** org.springframework.beans.factory.annotation.Autowired;  **import** javax.jms.JMSException; **import** javax.jms.Message; **import** javax.jms.MessageListener; **import** javax.jms.ObjectMessage; **import** java.io.Serializable;  /\*\*  \* 描述  \*  \* **@author** 三国的包子  \* **@version** 1.0  \* **@package** com.pinyougou.page.listener  \* **@since** 1.0  \*/ **public class** GenHtmlListener **implements** MessageListener{  @Autowired  **private** PageService **pageService**;  @Override  **public void** onMessage(Message message) {  **if**(message **instanceof** ObjectMessage) {  *//1.接收消息* **try** {  ObjectMessage message1 = (ObjectMessage)message;  Long[] ids = (Long[]) message1.getObject();  *//2.调用pageService的方法生成静态页面* **for** (Long id : ids) {  System.***out***.println(**"id:>>>"**+id);  **pageService**.genHtml(id);  }  } **catch** (JMSException e) {  e.printStackTrace();  }   }   } } |

生产者：pinyougou-manager-web项目中：goodsControlle.java中添加如下黄色部分。

|  |
| --- |
| @Autowired **private** JmsTemplate **jmsTemplate**; @Autowired **private** Destination **solr\_update\_item\_queue**;  @Autowired **private** Destination **gen\_html\_topic**; *//审核* @RequestMapping(**"/updateStatus"**) **public** Result updateStatus(Long[] ids,String status){  **try** {  **goodsService**.updateStatus(ids,status);   **if**(status.equals(**"1"**)){  *//审核  //发送消息 消息内容就是ID数组* **jmsTemplate**.send(**solr\_update\_item\_queue**, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createObjectMessage(ids);  }  });   *//发送消息 生成静态页面 发送商品（SPU）的 数组对象* **jmsTemplate**.send(**gen\_html\_topic**, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createObjectMessage(ids);  }  });    }   **return new** Result(**true**,**"更新成功"**);  } **catch** (Exception e) {  e.printStackTrace();  **return new** Result(**false**,**"更新失败"**);  } }  }   **return new** Result(**true**,**"更新成功"**);  } **catch** (Exception e) {  e.printStackTrace();  **return new** Result(**false**,**"更新失败"**);  } } |

配置mq的配置文件中添加：

|  |
| --- |
| *<!--生成静态页面的topic-->* <**bean class="org.apache.activemq.command.ActiveMQTopic" id="gen\_html\_topic"**>  <**constructor-arg name="name" value="gen\_html\_topic"**></**constructor-arg**> </**bean**> |

# 2.商品详情页-数据显示

## 2.1需求分析

运用Freemarker技术来实现商品详细页的静态化。通过地址栏输入某地址，如下形式

<http://localhost:9101/gen_item.do?goodsId=149187842867952>

能在本地电脑某目录生成商品详细页，页面的名称为商品id.html.

浏览器发送请求，生成某一个商品的的商品的详细信息。

## 2.2工程搭建

### 2.2.1服务接口层

创建pinyougou-page-interface工程，创建com.pinyougou.page.service包,包下创建接口

|  |
| --- |
| /\*\*  \* 商品详细页接口  \* **@author** Administrator  \*  \*/  **public interface** ItemPageService {  /\*\*  \* 生成商品详细页  \* **@param** goodsId  \*/  **Public boolean** genItemHtml(Long goodsId);  } |

### 2.2.2服务实现层

（1）创建war工程pinyougou-page-service （设置tomcat7插件的端口为9005）

（2）pom.xml引入依赖 参见其它服务工程, 另外添加freemarker依赖

|  |
| --- |
| <dependency>  <groupId>org.freemarker</groupId>  <artifactId>freemarker</artifactId>  </dependency> |

（3）添加web.xml 参见其它服务工程

（4）参考其他的工程配置spring的各种配置文件 ，另外配置：

|  |
| --- |
| <bean id=*"freemarkerConfig"* class=*"org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"*>  <propertyname=*"templateLoaderPath"*value=*"/WEB-INF/ftl/"*/>  <propertyname=*"defaultEncoding"*value=*"UTF-8"*/>  </bean> |

1. 在resources下创建属性文件:properties/page.properties

内容为：

|  |
| --- |
| pagedir=g:\\item\\ |

用于配置网页生成目录

（6）建立com.pinyougou.page.service.impl包，包下建立类

|  |
| --- |
| @Service  **publicclass** ItemPageServiceImpl **implements** ItemPageService {  @Value("${pagedir}")  **private** String pagedir;    @Autowired  **private** FreeMarkerConfig freeMarkerConfig;    @Autowired  **private** TbGoodsMapper goodsMapper;    @Autowired  **private** TbGoodsDescMapper goodsDescMapper;    @Override  **publicboolean** genItemHtml(Long goodsId){  **try** {  Configuration configuration = freeMarkerConfig.getConfiguration();  Template template = configuration.getTemplate("item.ftl");  MapdataModel=**new** HashMap<>();  //1.加载商品表数据  TbGoods goods = goodsMapper.selectByPrimaryKey(goodsId);  dataModel.put("goods", goods);  //2.加载商品扩展表数据  TbGoodsDesc goodsDesc = goodsDescMapper.selectByPrimaryKey(goodsId);  dataModel.put("goodsDesc", goodsDesc);  Writer out=**new** FileWriter(pagedir+goodsId+".html");  template.process(dataModel, out);  out.close();  **returntrue**;  } **catch** (Exception e) {  e.printStackTrace();  **returnfalse**;  }  }  } |

（7）将item.html拷贝至web-inf/ftl下 ，修改扩展名为ftl ,将商品名称用插值代替

|  |
| --- |
| <div class="sku-name">  <h4>${goods.goodsName}</h4>  </div> |

（8）在G盘建立文件夹item,将必要的样式表和Js拷贝到此目录下，此目录为生成的目录

### 2.2.3运营商管理后台

（1）pinyougou-manager-web引入依赖pinyougou-page-interface

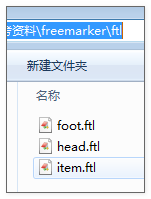
（2）在GoodsController.java中新增方法

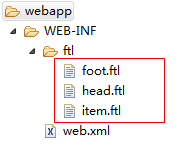
|  |
| --- |
| @Reference(timeout=40000)  **private** ItemPageService itemPageService;  /\*\*  \* 生成静态页（测试）  \* **@param** goodsId  \*/  @RequestMapping("/genHtml")  **public**void genHtml(Long goodsId){  itemPageService.genItemHtml(goodsId);  } |

## 2.3商品详情页模板构建

### 2.3.1模板模块化引入

此时我们的item.ftl内容较多，当我们编辑时不容易快速找到编辑的位置，所以我们将原来item.ftl中的头部信息分拆分到head.ftl ,将尾部信息拆分到foot.ftl ,用include指令在item.ftl中引入 。如下图copy即可





### 2.3.2生成基本数据

在模板中找到合适的位置，用插值替换静态文本

|  |
| --- |
| <divclass=*"news"*><span>${goods.caption}</span></div> |

|  |
| --- |
| <divclass=*"fl price"*><i>¥</i><em>${goods.price}</em><span>降价通知</span></div> |

|  |
| --- |
| <divclass=*"intro-detail"*><!-- 商品详情 --> ${goodsDesc.introduction}</div> |

|  |
| --- |
| <divid=*"two"*class=*"tab-pane"*><p>${goodsDesc.packageList}</p></div>  <divid=*"three"*class=*"tab-pane"*><p>${goodsDesc.saleService}</p></div> |

运行控制层代码，测试生成效果

<http://localhost:9101/goods/genHtml.do?goodsId=149187842867960>

149187842867964

### 2.3.3生成图片列表

编辑模板文件

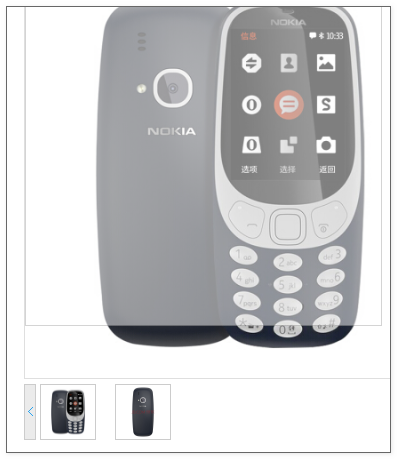
|  |
| --- |
| <#--图片列表 -->  <#assign imageList=goodsDesc.itemImages?eval /> |

这一句要转换图片列表的json字符串

图片部分的代码

|  |
| --- |
| <!--默认第一个预览-->  <div id="preview" class="spec-preview">  <span class="jqzoom">  <#if (imageList?size>0)>  <img jqimg="${imageList[0].url}"src="${imageList[0].url}"width="400px" height="400px" />  </#if>  </span>  </div>  <!--下方的缩略图--><div class="spec-scroll">  <div class="items">  <ul>  <#list imageList as item>  <li><img src="${item.url}"bimg="${item.url}"onmousemove="preview(this)" /></li>  </#list>  </ul>  </div>  </div> |

生成效果如下：



### 2.3.4生成扩展属性列表

修改模板 首先进行json转换

|  |
| --- |
| <#--扩展属性列表 -->  <#assign customAttributeList=goodsDesc.customAttributeItems?eval /> |

显示扩展属性数据，如果扩展属性为空则不显示此条数据

|  |
| --- |
| <#list customAttributeList as item>  <#if item.value??>  <li>${item.text} ：${item.value}</li>  </#if>  </#list> |

### 2.3.5生成规格列表

修改模板 转换规格列表

|  |
| --- |
| <#--规格列表 -->  <#assign specificationList=goodsDesc.specificationItems?eval /> |

此时，我们需要使用嵌套循环

|  |
| --- |
| <#list specificationList as specification>  <dl>  <dt>  <div class="fl title">  <i>${specification.attributeName}</i>  </div>  </dt>  <#list specification.attributeValue as item>  <dd><a href="javascript:;">${item}</a></dd>  </#list>  </dl>  </#list> |

### 2.3.6生成商品类型面包屑

修改ItemPageServiceImpl ，读取三级商品分类名称，加入到数据模型中

|  |
| --- |
| @Service  **publicclass** ItemPageServiceImpl **implements** ItemPageService {  @Autowired  **private** FreeMarkerConfig freeMarkerConfig;    @Autowired  **private** TbGoodsMapper goodsMapper;    @Autowired  **private** TbGoodsDescMapper goodsDescMapper;    @Autowired  **private** TbItemCatMapper itemCatMapper;    @Override  **publicboolean** genItemHtml(Long goodsId){    **try** {  Configuration configuration = freeMarkerConfig.getConfiguration();  Template template = configuration.getTemplate("item.ftl");  Map dataModel=**new** HashMap<>();    //1.加载商品表数据  TbGoods goods = goodsMapper.selectByPrimaryKey(goodsId);  dataModel.put("goods", goods);    //2.加载商品扩展表数据  TbGoodsDesc goodsDesc = goodsDescMapper.selectByPrimaryKey(goodsId);  dataModel.put("goodsDesc", goodsDesc);    //3.商品分类  String itemCat1 = itemCatMapper.selectByPrimaryKey(goods.getCategory1Id()).getName();  String itemCat2 = itemCatMapper.selectByPrimaryKey(goods.getCategory2Id()).getName();  String itemCat3 = itemCatMapper.selectByPrimaryKey(goods.getCategory3Id()).getName();  dataModel.put("itemCat1", itemCat1);  dataModel.put("itemCat2", itemCat2);  dataModel.put("itemCat3", itemCat3);    Writer out=**new** FileWriter("d:\\item\\"+goodsId+".html");  template.process(dataModel, out);  out.close();  **returntrue**;  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  **returnfalse**;  }  }  } |

修改模板，展示商品分类面包屑

|  |
| --- |
| <ul class="sui-breadcrumb">  <li><a href="#">${itemCat1}</a></li>  <li><a href="#">${itemCat2}</a></li>  <li><a href="#">${itemCat3}</a></li>  </ul> |

# 3.商品详情页-前端逻辑

## 3.1购买数量加减操作

### 3.1.1加入angularJS库

将angularJS库加入g:\item下



### 3.1.2前端控制层

将base.js拷贝到js目录下

在js目录下构建controller文件夹，创建itemController.js

|  |
| --- |
| //商品详细页（控制层）  app.controller('itemController',function($scope){  //数量操作  $scope.addNum=function(x){  $scope.num = parseInt($scope.num);  x=parseInt(x);  $scope.num=$scope.num+x;  if($scope.num<1){  $scope.num=1;  }  }  }); |

在方法中控制数量不能小于1

### 3.1.3模板中修改

引入js

|  |
| --- |
| <script type="text/javascript" src="plugins/angularjs/angular.min.js"></script>  <script type="text/javascript" src="js/base.js"></script>  <script type="text/javascript" src="js/controller/itemController.js"></script> |

添加指令

|  |
| --- |
| <body ng-app="pinyougou" ng-controller="itemController" ng-init="num=1"> |

调用操作数量的方法

|  |
| --- |
| <**div class="controls"**>  <**input autocomplete="off" type="text" value="{{num}}" ng-model="num" minnum="1" class="itxt"**/>  <**a href="javascript:void(0)" class="increment plus" ng-click="addNum(1)"**>+</**a**>  <**a href="javascript:void(0)" class="increment mins" ng-click="addNum(-1)"**>-</**a**> </**div**> |

## 3.2规格选择

最终我们需要实现的效果：



### 3.2.1前端控制层

修改itemController.js

|  |
| --- |
| $scope.specificationItems={};//记录用户选择的规格  //用户选择规格  $scope.selectSpecification=function(name,value){  $scope.specificationItems[name]=value;  }  //判断某规格选项是否被用户选中  $scope.isSelected=function(name,value){  if($scope.specificationItems[name]==value){  return true;  }else{  return false;  }  } |

### 3.2.2模板

页面调用控制器的方法

|  |
| --- |
| <dd>  <a class="{{isSelected('${specification.attributeName}','${item}')?'selected':''}}"  ng-click="selectSpecification('${specification.attributeName}','${item}')">  ${item}  <span title="点击取消选择">&nbsp;</span>  </a>  </dd> |

# 4.商品详情页-读取SKU信息

需求：当我们选择规格后，应该在页面上更新商品名称为SKU的商品标题，价格也应该为SKU的商品价格。

## 4.1页面生成SKU列表变量

### 4.1.1后端服务层

修改pinyougou-page-service的ItemPageServiceImpl.java

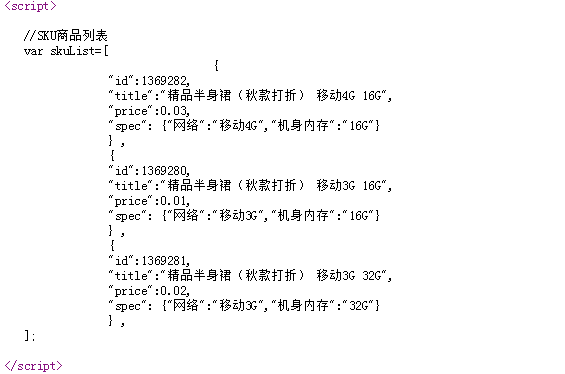
|  |
| --- |
| @Autowired  **private** TbItemMapper itemMapper;    @Override  **publicboolean** genItemHtml(Long goodsId){  **try** {  Configuration configuration = freeMarkerConfig.getConfiguration();  Template template = configuration.getTemplate("item.ftl");  Map dataModel=**new** HashMap<>();  //1.加载商品表数据  //2.加载商品扩展表数据  //3.商品分类  //4.SKU列表  TbItemExample example=**new** TbItemExample();  Criteria criteria = example.createCriteria();  criteria.andStatusEqualTo("1");//状态为有效  criteria.andGoodsIdEqualTo(goodsId);//指定SPU ID  example.setOrderByClause("is\_default desc");//按照状态降序，保证第一个为默认  List<TbItem> itemList = itemMapper.selectByExample(example);  dataModel.put("itemList", itemList);  Writer out=**new** FileWriter(pagedir+goodsId+".html");  template.process(dataModel, out);  out.close();  **returntrue**;  } **catch** (Exception e) {  e.printStackTrace();  **returnfalse**;  }  } |

### 4.1.2模板

修改模板：

|  |
| --- |
| <**script**>  *//SKU商品列表* **var *skuList***=[  **<#list itemList as item>  <#if item\_index**==(**itemList**?**size**-1)**>** {  **"id"**:**${item**.**id**?**c}**,  **"title"**:**"${item**.**title**!**''}"**,  **"price"**:**${item**.**price**?**c}**,  **"spec"**: **${item**.**spec}** }  **<#else>** {  **"id"**:**${item**.**id**?**c}**,  **"title"**:**"${item**.**title**!**''}"**,  **"price"**:**${item**.**price**?**c}**,  **"spec"**: **${item**.**spec}** } ,  **</#if>   </#list>** ]; </**script**> |

测试生成，发现页面源代码中生成了变量



## 4.2显示SKU标题和价格

### 4.2.1加载默认SKU信息

修改itemController.js

|  |
| --- |
| //加载默认SKU  $scope.loadSku=function(){  $scope.sku=skuList[0];  $scope.specificationItems= JSON.parse(JSON.stringify($scope.sku.spec)) ;  } |

修改模板item.ftl

|  |
| --- |
| <body ng-app="pinyougou"ng-controller="itemController"ng-init="num=1;loadSku()"> |

修改模板，显示标题

|  |
| --- |
| <div class="sku-name"><h4>{{sku.title}}</h4></div> |

显示价格

|  |
| --- |
| <div class="summary-wrap">  <div class="fl title"><i>价　　格</i></div>  <div class="fl price"><i>¥</i><em>{{sku.price}}</em><span>降价通知</span></div>  </div> |

### 4.2.2选择规格更新SKU

修改itemController.js , 编写匹配对象的方法

|  |
| --- |
| //匹配两个对象  matchObject=function(map1,map2){  for(var k in map1){  if(map1[k]!=map2[k]){  return false;  }  }  for(var k in map2){  if(map2[k]!=map1[k]){  return false;  }  }  return true;  } |

编写方法，在SKU列表中查询当前用户选择的SKU

|  |
| --- |
| //查询SKU  searchSku=function(){  for(var i=0;i<skuList.length;i++ ){  if( matchObject(skuList[i].spec ,$scope.specificationItems ) ){  $scope.sku=skuList[i];  return ;  }  }  $scope.sku={id:0,title:'--------',price:0};//如果没有匹配的  } |

在用户选择规格后触发读取方法

|  |
| --- |
| //用户选择规格  $scope.selectSpecification=function(name,value){  $scope.specificationItems[name]=value;  searchSku();//读取sku  } |

## 4.3添加商品到购物车

修改itemController.js

|  |
| --- |
| //添加商品到购物车  $scope.addToCart=function(){  alert('skuid:'+$scope.sku.id);  } |

修改模板：

|  |
| --- |
| <li><a href="#" target="\_blank" class="sui-btnbtn-danger addshopcar"ng-click="addToCart()">加入购物车</a></li> |

# 5.系统模块对接

## 5.1运营商后台调用页面生成服务

在SPU商品被审核之后通过之后，应当就需要生成商品详情页。

修改pinyougou-manager-web的GoodsController.java

|  |
| --- |
| @RequestMapping("/updateStatus")  **public** Result updateStatus(Long[] ids,String status){  **try** {  goodsService.updateStatus(ids, status);  //按照SPU ID查询 SKU列表(状态为1)  **if**(status.equals("1")){//审核通过  List<TbItem> itemList = goodsService.findItemListByGoodsIdandStatus(ids, status);  //调用搜索接口实现数据批量导入  **if**(itemList.size()>0){  itemSearchService.importList(itemList);  }**else**{  System.***out***.println("没有明细数据");  }  //静态页生成  **for**(Long goodsId:ids){  itemPageService.genItemHtml(goodsId);  }  }  **returnnew** Result(**true**, "修改状态成功");  } **catch** (Exception e) {  e.printStackTrace();  **returnnew** Result(**false**, "修改状态失败");  }  } |

## 5.2创建商品详情web工程

说明：生成好了页面之后，需要通过http请求来访问页面，不能直接访问，所以需要部署一个http服务器，指定静态页面的路径。此处在开发环境下，使用web项目来临时保存，部署tomcat 作为http服务器，真正部署时，需要部署在nginx中。

创建war模块工程pinyougou-page-web ，将目标目录（g:\item）的文件拷贝到此工程

在pom.xml中添加tomcat7插件，指定端口为9105

## 5.3搜索系统修改跳转的连接

修改pinyougou-search-web 的search.html，修改点击图片的链接为

http://localhost:9105/{{item.id}}.html

说明：商品详细页是静态页，所以在开发阶段我们可以使用tomcat来进行测试。部署在生产环境是部署在Nginx中。