**淘淘商城**

**第十一天**

# 课程计划

1. 商品详情页动态展示
2. freemarker介绍
3. freemarker使用
4. 商品详情页静态化

# 商品详情页动态展示

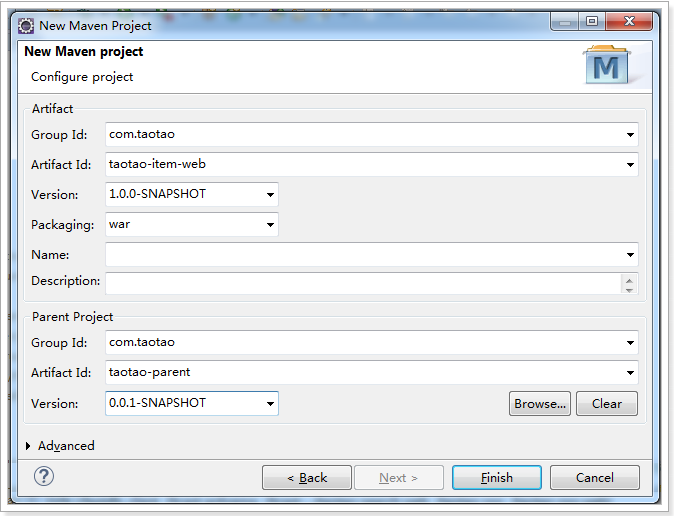
查看京东和天猫，发现主流电商的商品详情页都是独立的域名展示。

因为商品数量大，访问的频率很高，需要单独把商品详情分离出来，方便对商品详情功能进行扩容(增加集群数量)。

我们搭建表现层taotao-item-web即可，调用taotao-manager的服务，获取商品数据。

## 创建taotao-item-web

### 创建工程



### 添加依赖

需要的技术

springMVC

jsp相关，通用工具类等

dubbo

需要展现商品描述，和taotao-portal很像，可以参考

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-item-web</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- JSP相关 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8089</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

### 加入配置文件和静态资源

#### web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-item-web</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<!-- 配置解决post提交乱码的问题 -->

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置springMVC -->

<servlet>

<servlet-name>taotao-item-web</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 设置springMVC配置文件 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- 配置springMVC映射 -->

<servlet-mapping>

<servlet-name>taotao-item-web</servlet-name>

<!-- 所有以html结尾的请求进入springMVC -->

<!-- 伪静态化,SEO搜索引擎优化 -->

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

</web-app>

#### springmvc.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:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* 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://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 加入properties配置文件 -->

<context:property-placeholder location=*"classpath:* *resource/env.properties"* />

<!-- 配置注解驱动 -->

<mvc:annotation-driven />

<!-- 配置controller扫描 -->

<context:component-scan base-package=*"com.taotao.item.controller"* />

<!-- Example: prefix="/WEB-INF/jsp/", suffix=".jsp", viewname="test" ->

"/WEB-INF/jsp/test.jsp" -->

<!-- 配置视图解析器 ,配置前缀和后缀 -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

<!-- 配置dubbo服务 -->

<dubbo:application name=*"taotao-item-web"* />

<!-- 使用广播 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

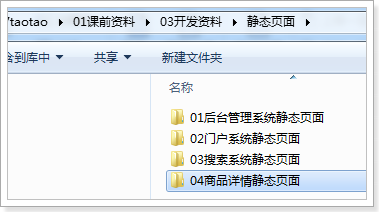
<!-- 声明要调用的服务，timeout是设置连接超时最长时间,如果不设置，超时时间默认是3秒 -->

<!-- <dubbo:reference interface="com.taotao.content.service.ContentService" -->

<!-- id="contentService" timeout="1000000" /> -->

</beans>

#### 加入静态资源



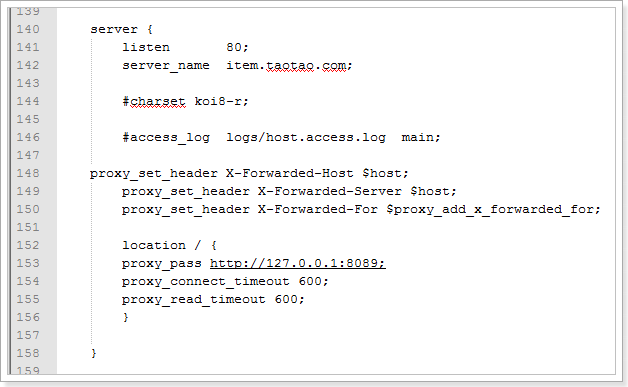
### 修改域名访问

增加item.taotao.com域名的映射



### 修改Nginx反向代理服务器

增加item.taotao.com的Server配置



## 前端实现

### 搜索系统页面

用户可以使用搜索系统搜索商品，然后点击搜索结果，进入商品详情页

商品详情页请求地址携带了商品id，可以根据商品id进行查询



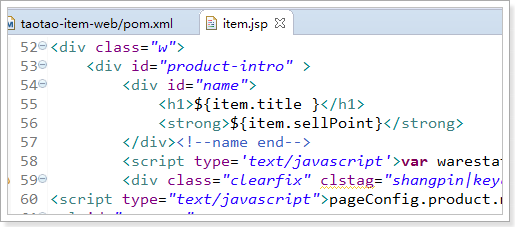
### 商品详情页

商品描述页在taotao-item-web，查看其中的item.jsp进行分析，

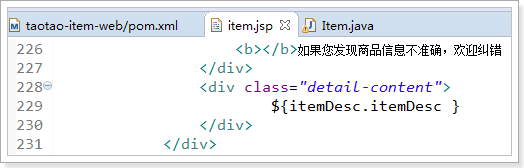
页面需要商品数据${item}和商品描述数据${itemDesc}

可以使用Item对象和ItemDesc对象进行数据传递

显示商品数据



显示描述数据



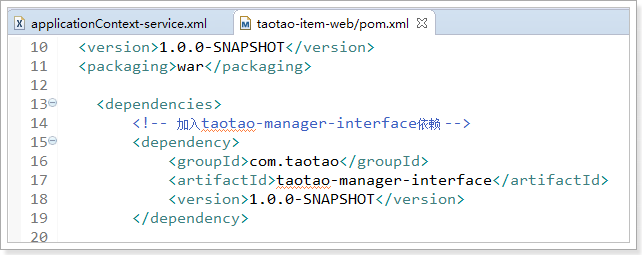
## 后端实现

### taotao-item-web实现

#### 添加依赖

在taotao-item-web添加

需要调用taotao-manager的商品服务，需要加入taotao-manager-interface依赖



#### 声明服务调用

在taotao-item-web声明服务调用

需要给商品和商品详情的服务



#### 实现Controller

在taotao-item-web实现

@Controller

@RequestMapping("item")

**public** **class** ItemController {

@Autowired

**private** ItemService itemService;

@Autowired

**private** ItemDescService itemDescService;

// http://item.taotao.com/item/${item.id }.html

@RequestMapping(value = "{itemId}", method = RequestMethod.***GET***)

**public** String toItem(@PathVariable("itemId") Long itemId, Model model) {

// 根据商品id从服务中查询商品数据

Item item = **this**.itemService.queryById(itemId);

// 根据商品id从服务中查询商品描述数据

ItemDesc itemDesc = **this**.itemDescService.queryById(itemId);

// 保存商品数据到模型中

model.addAttribute("item", item);

// 保存商品描述数据到模型中

model.addAttribute("itemDesc", itemDesc);

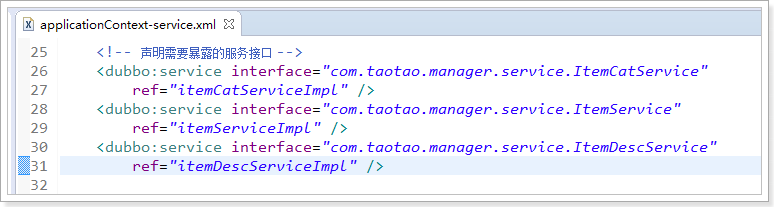
**return** "item";

}

}

#### 服务层声明服务的发布

在taotao-manager-servicce发布服务



## 实现效果



# freemarker介绍

## 为什么要用静态化

商品详情页是消费者了解商品的主要途径，访问的频率非常高。

所以需要对商品详情页进行优化，提高访问的速度。

优化方案：

1. 使用redis添加缓存

redis的访问速度快，能够较大的提升查询数据的速度，减轻MySQL数据库的访问压力。

如果使用缓存的方式，需要注意数据同步的问题

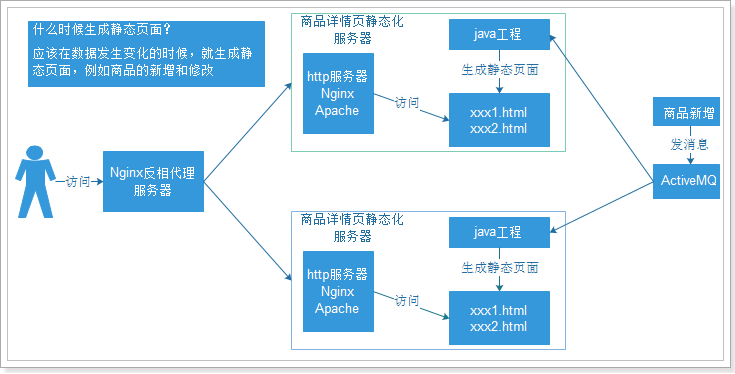
1. 使用静态化

把动态页面（jsp，php，asp等)，转变成静态页面（html）

好处：

1. 访问静态页面不需要经过程序处理，可以提高速度。（处理速度，和数据库访问速度）
2. 稳定性高。
3. 从安全角度讲，静态网页不易遭到黑客攻击。
4. 静态页面相对于动态页面更容易被搜索引擎收录。（SEO）

## 静态化访问的流程



## freemarker介绍

freemarker是java语言开发

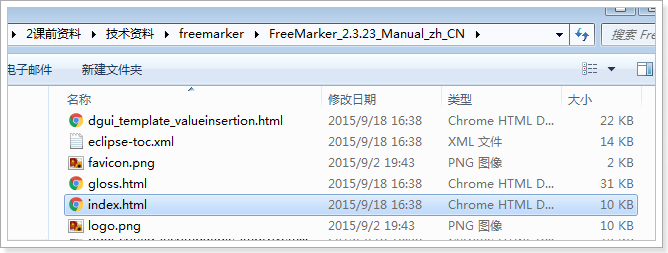
freemarker是一个模板引擎，可以基于模板生成文本（html）

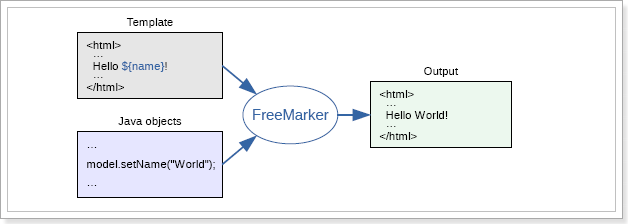
freemarker和应用服务器（Tomcat、jetty）是没有关联的，不需要使用jsp和servlet技术

freemarker除了前端页面展示的功能外，还可以生成xml，java

企业一般用freemarker做生成静态页面功能

freemarker文档地址





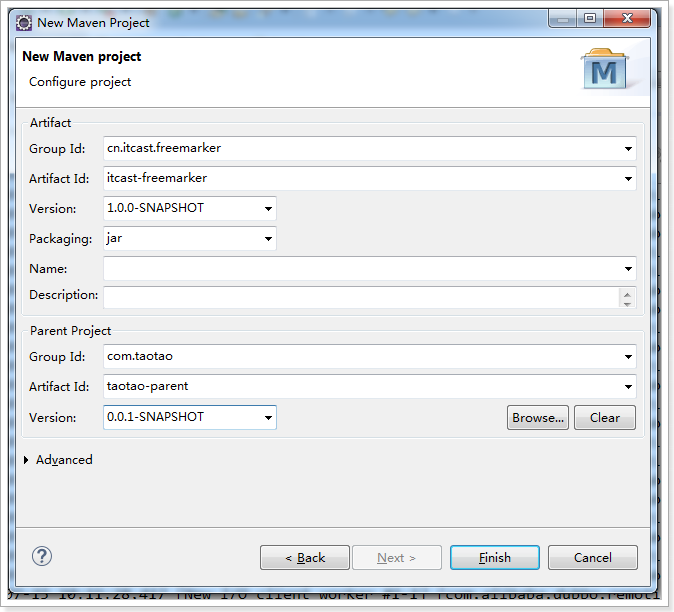
模板 + 数据 = 静态页面

其实我们要学习的是如何编写模板

# freemarker使用

## 创建工程

创建itcast-freemarker



## 加入依赖

加入freemarker依赖

依赖只有一个，freemarker可以单独使用

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-freemarker</artifactId>

<version>1.0.0-SNAPSHOT</version>

<dependencies>

<!-- 静态化freemarker -->

<dependency>

<groupId>org.freemarker</groupId>

<artifactId>freemarker</artifactId>

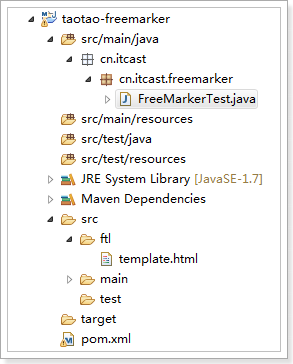
</dependency>

</dependencies>

</project>

## 最简单的使用

需要编写测试逻辑和模板文件



### 测试逻辑

**public** **class** FreeMarkerTest {

**public** **static** **void** main(String[] args) **throws** Exception {

// 模板 + 数据 = 静态页面

// 创建核心类Configuration

Configuration cfg = **new** Configuration(Configuration.***VERSION\_2\_3\_23***);

// 设置模板所在的位置

System.***out***.println(System.*getProperty*("user.dir"));

cfg.setDirectoryForTemplateLoading(**new** File(System.*getProperty*("user.dir") + "/src/ftl"));

// 获取模板,参数是模板名称

// 模板类型，官方要求ftl后缀,jsp,html,java,itcast后缀都可以

Template template = cfg.getTemplate("template.html");

// 指定数据模型

Map<String, Object> root = **new** HashMap<String, Object>();

root.put("hello", "world");

// 指定输出

Writer out = **new** FileWriter(**new** File("c:/result.html"));

// 使用模板输出

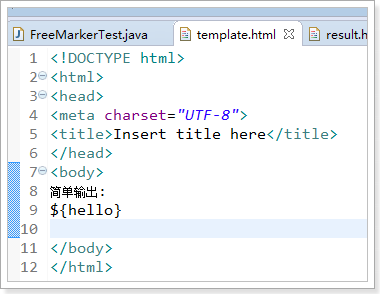
template.process(root, out);

out.close();

}

}

### 编写模板

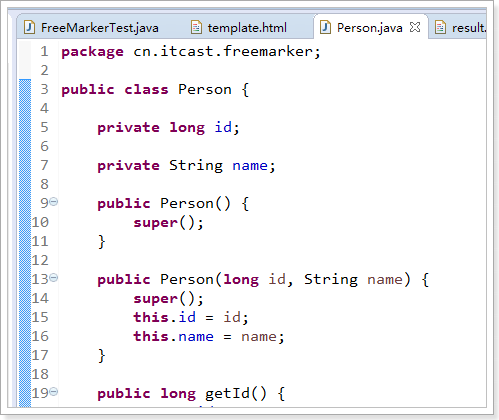


## 常用的模板语法

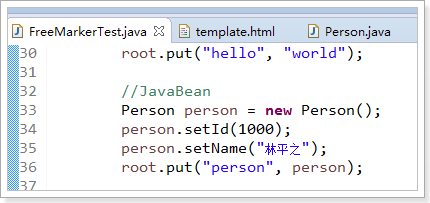
### JavaBean

java代码输出数据

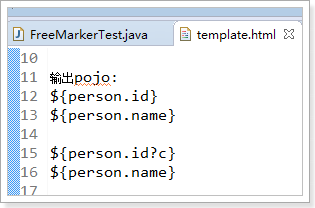
创建JavaBean



编写输出逻辑



模板展示

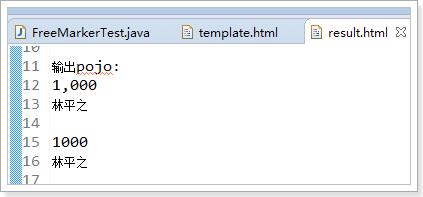


Long型会自动加千分位标识符，

去除千分位加?c 即可。

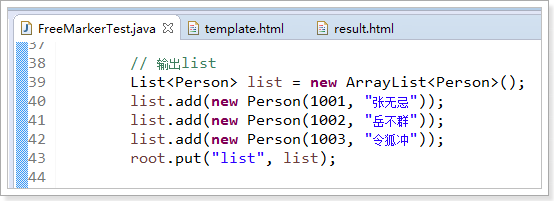
freemarker是java开发的，大小写敏感

效果

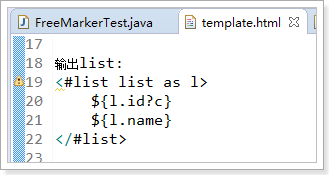


### List

java代码输出数据



模板展示



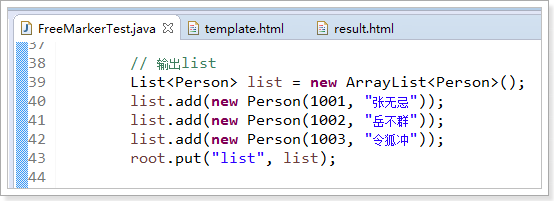
效果



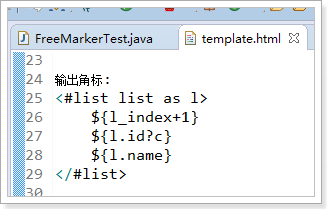
### 角标

java代码输出数据

同上面的List输出



模板展示



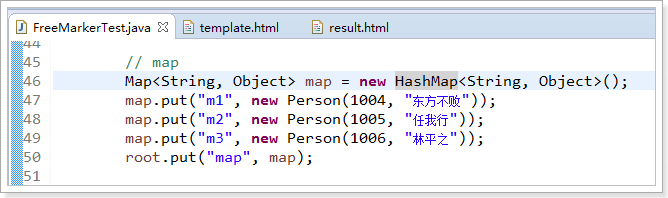
角标从0开始，可以加任意数字，这里是加1

效果



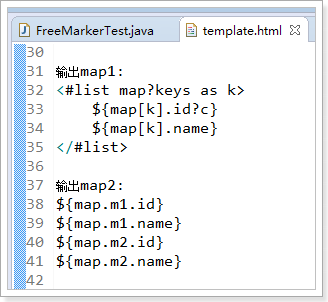
### Map

java代码输出数据

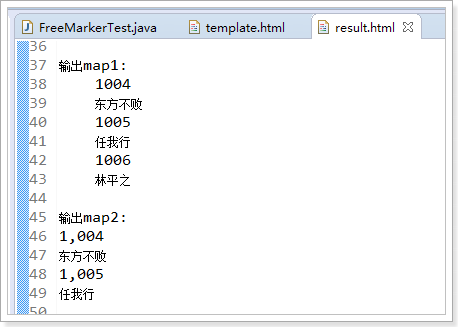


模板展示

map输出有两种方式

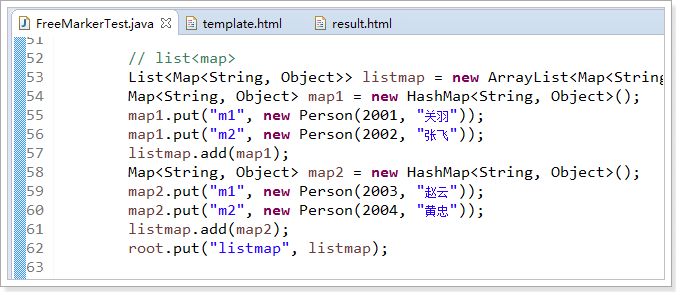


效果

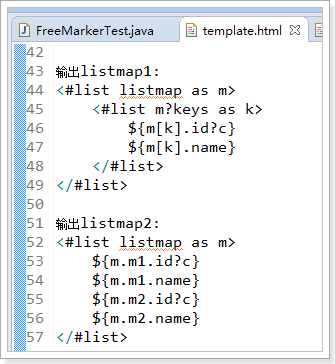


### List<Map>

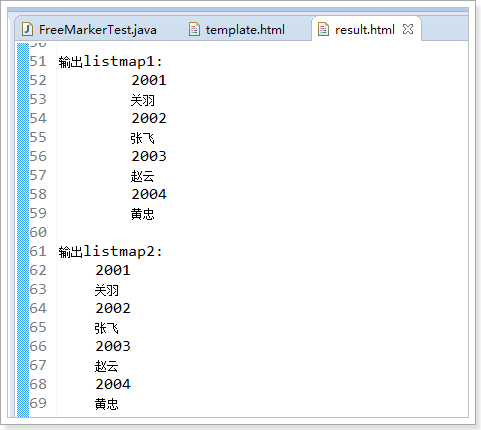
java代码输出数据



模板展示



效果

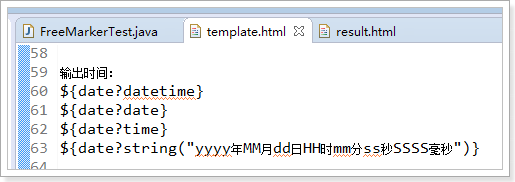


### 时间

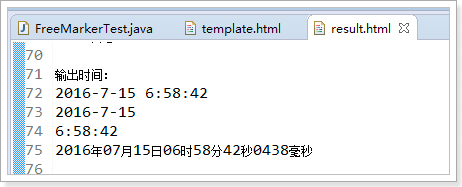
java代码输出数据



模板展示



效果

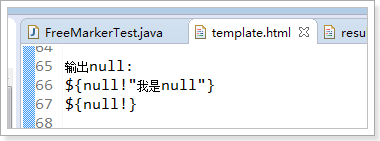


### null

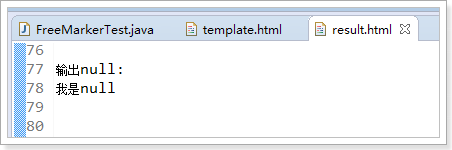
java代码输出数据



模板展示



效果



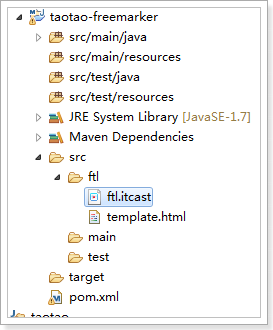
### <#include>

包含标签，可以把其他的文档包含进来

java代码输出数据

模板展示

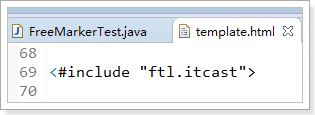
建立ftl.itcast



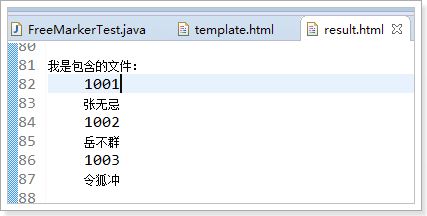
内容



编辑模板文件



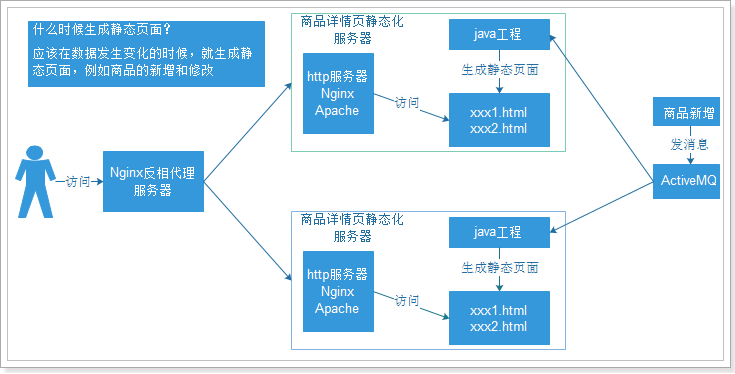
效果



# 商品详情静态化改造

## 流程图

用户访问静态化页面流程图



## 实现静态化

需要在新增商品时发送消息给ActiveMQ，商品详情系统接收消息，生成静态页面

商品新增发送消息给ActiveMQ，已经实现，需要实现商品详情系统接收消息，生成静态页面

### 实现静态化逻辑

需要接受消息，根据消息内容生成静态化页面

需要做两件事，

1. 接受消息

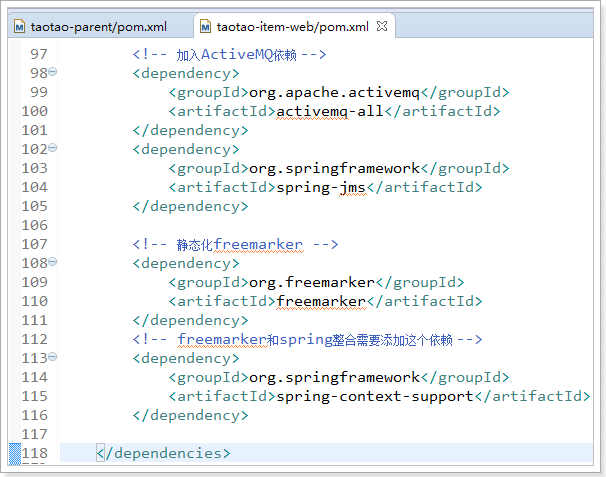
消费消息：

1. 编写静态化方法（数据）
2. 编写商品模板（模板）

### 商品详情集成ActiveMQ

#### 添加依赖

在taotao-item-web添加



#### 配置文件

freemarker和spring整合，添加配置文件

springmvc-freemarker.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:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* 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://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置freemarker -->

<bean id=*"freeMarkerConfigurer"*

class=*"org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer"*>

<!-- 设置模板加载路径 -->

<property name=*"templateLoaderPath"* value=*"WEB-INF/ftl"* />

<!-- 设置编码 -->

<property name=*"defaultEncoding"* value=*"UTF-8"* />

</bean>

</beans>

spring和ActiveMQ整合配置文件

applicationContext-activemq.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:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:jms=*"http://www.springframework.org/schema/jms"* 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-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.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://192.168.37.161:61616"* />

</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>

<!--这个是主题目的地，一对多的 -->

<bean id=*"topicDestination"* class=*"org.apache.activemq.command.ActiveMQTopic"*>

<constructor-arg value=*"topic"* />

</bean>

<!-- messageListener实现类 -->

<bean id=*"myMessageListener"* class=*"com.taotao.item.listener.ItemMessageListener"*></bean>

<!-- 配置一个jsm监听容器 -->

<bean id=*"jmsContainer"*

class=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>

<property name=*"connectionFactory"* ref=*"connectionFactory"* />

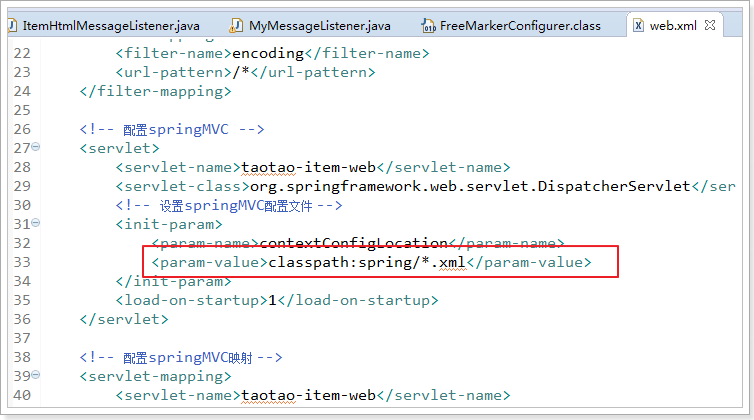
<property name=*"destination"* ref=*"topicDestination"* />

<property name=*"messageListener"* ref=*"myMessageListener"* />

</bean>

</beans>

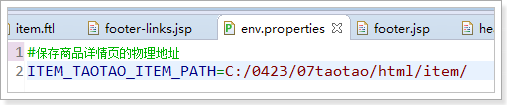
在taotao-item-web配置web.xml



#### 消费消息（静态化java实现）

在taotao-item-web实现

加入配置文件



接收消息，并消费消息，根据消息实现静态化

**public** **class** ItemMessageListener **implements** MessageListener {

**private** **static** **final** ObjectMapper ***MAPPER*** = **new** ObjectMapper();

@Override

**public** **void** onMessage(Message message) {

**if** (message **instanceof** TextMessage) {

// 获取消息的内容

TextMessage textMessage = (TextMessage) message;

**try** {

String json = textMessage.getText();

// 判断json为非空

**if** (StringUtils.*isNotBlank*(json)) {

// 使用MAPPER解析json格式的数据，获取jsonNode

JsonNode jsonNode = ***MAPPER***.readTree(json);

// 获取商品的操作符

String type = jsonNode.get("type").asText();

// 获取商品id

**long** itemId = jsonNode.get("itemId").asLong();

// 根据商品id生成静态化页面

**this**.genHtml(itemId);

}

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

@Autowired

**private** FreeMarkerConfigurer freeMarkerConfigurer;

@Autowired

**private** ItemService itemService;

@Autowired

**private** ItemDescService itemDescService;

@Value("${TAOTAO\_ITEM\_HTML\_PATH}")

**private** String TAOTAO\_ITEM\_HTML\_PATH;

**private** **void** genHtml(**long** itemId) **throws** Exception {

// 获取freemarker的核心对象，使用spring整合的对象获取

Configuration configuration = **this**.freeMarkerConfigurer.getConfiguration();

// 使用核心对象，获取模板

Template template = configuration.getTemplate("item.ftl");

// 设置模型数据

Map<String, Object> root = **new** HashMap<>();

// 把商品基础数据放到root中

root.put("item", **this**.itemService.queryById(itemId));

// 把商品描述数据放到root中

root.put("itemDesc", **this**.itemDescService.queryById(itemId));

// 使用模板输出静态页面

Writer out = **new** FileWriter(**new** File(**this**.TAOTAO\_ITEM\_HTML\_PATH + itemId + ".html"));

template.process(root, out);

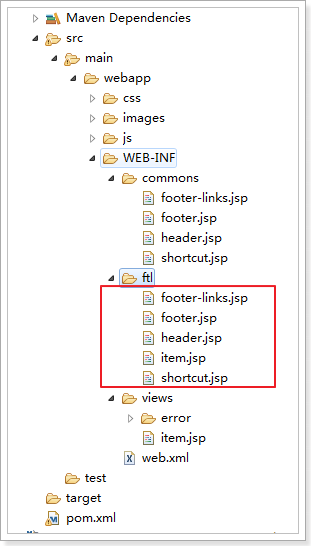
}

}

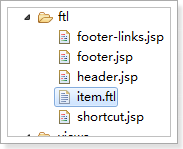
#### 编写商品模板

在taotao-item-web编写

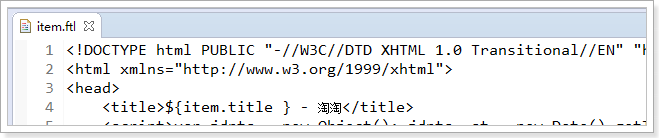
##### 创建ftl文件夹，并复制5个文件到ftl里面

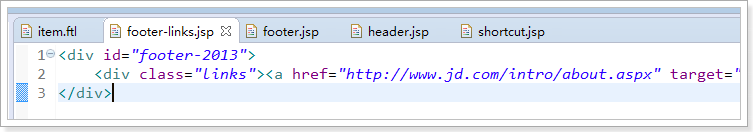


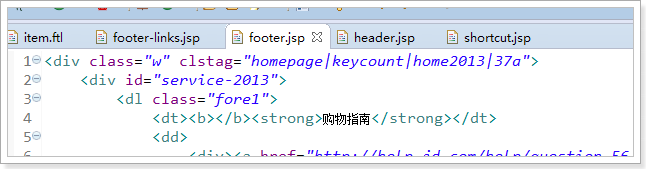
##### item.jsp改名为item.ftl

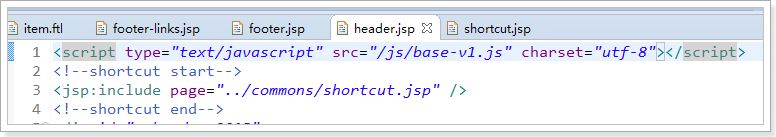


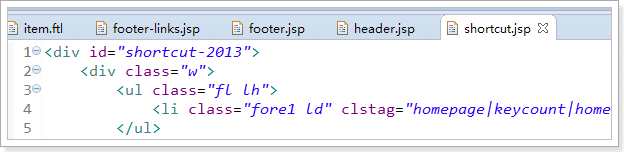
##### 修改5个文件去掉jsp头信息





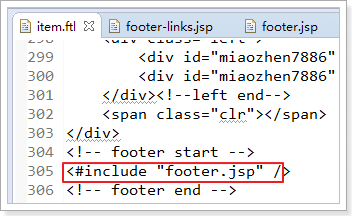




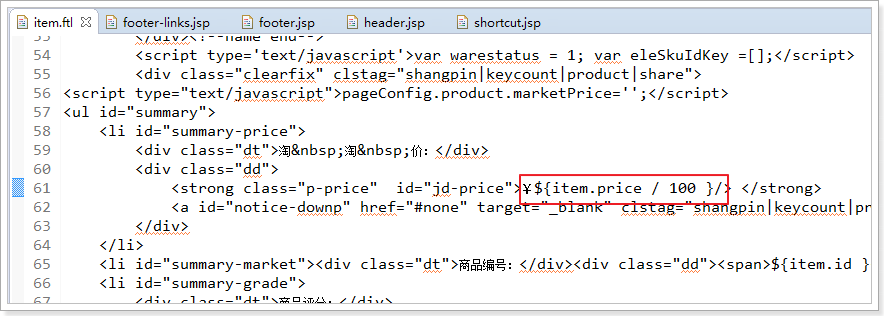


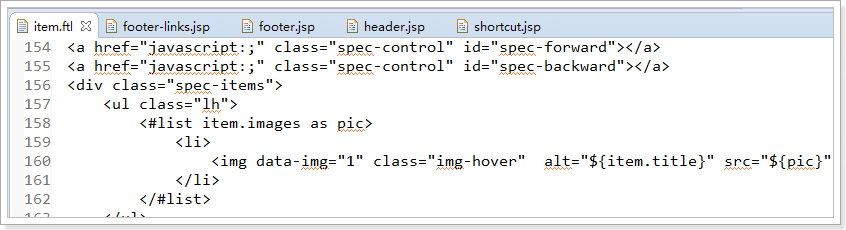
##### 修改所有文件中的include标签，为freemarker的include

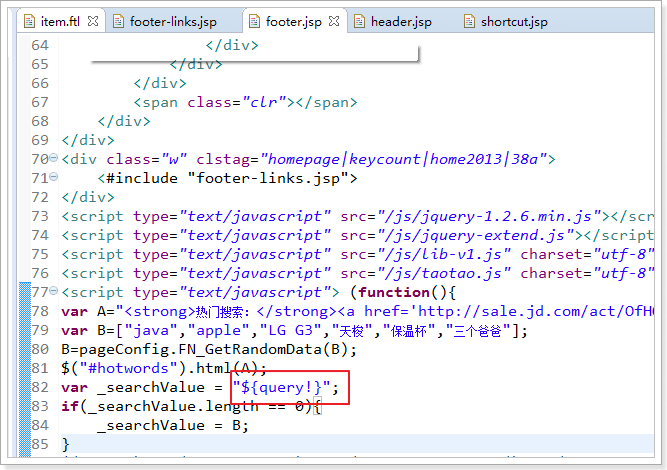
需要修改所ftl下所有的文件



#### 修改item.ftl符合freemarker语法

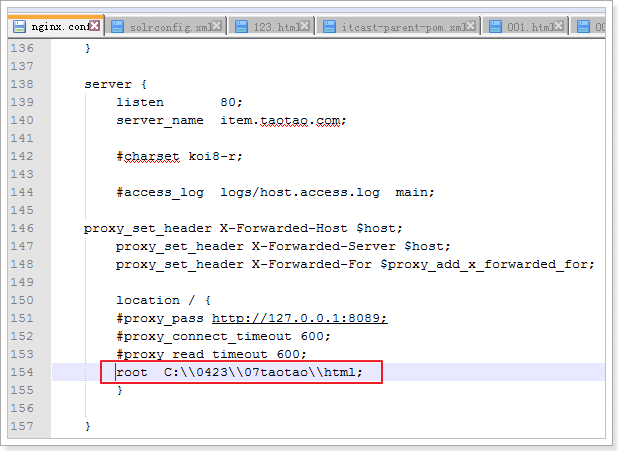




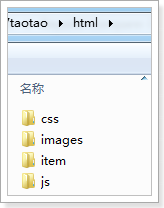


## 搭建静态资源服务器

使用Nginx搭建静态资源服务器



加入静态资源（js，css等）到对应的目录



测试：

先在后台系统新增一个商品

商品详情页进行访问，成功显示

PS：

因为只有生成了静态页的商品才可以显示，所有导入的商品全部生成静态页比较麻烦，所以这个功能实现以后，为了后续开发学习方便，仍旧切换为动态页面展示商品详情。

修改方式：只需要修改Nginx配置即可

