# 淘淘商城-前台系统搭建、相关功能实现

## 内容回顾

商品添加功能的实现

1. 富文本编辑器的使用KindEditor。
   1. 引入js
   2. 使用js代码初始化KindEditor，需要一个textarea控件。
   3. 提交表单之前，需要调用sync方法，同步数据。是富文本编辑器和textArea控件同步数据。
2. 商品添加功能实现。
   1. 向商品表插入数据
   2. 向商品描述表插入数据
3. 商品的规格参数。
   1. 使用模板的思路。
   2. 为类商品创建一个规格参数模板。
   3. 添加商品时，读取模板，基于模板生成一个表单，供用户输入规格参数。
   4. 使用js把规格参数转换成json数据，保存到数据库
   5. 读取规格参数，转换成html展示到页面。

## 学习计划

1. 前台系统搭建
   1. 服务层
   2. 表现层
2. 商城首页展示
3. 实现商品分类展示功能。
4. 首页内容管理思路及cms系统。

# 前台系统搭建

## 系统架构分析



查询商品信息，pc端可以查询，移动端也可以查看商品。

下单：pc端可以下单，移动端也可以下单。

可以把业务逻辑提取出来，发布服务，供pc、移动端公用。

好处：

1. 提高代码复用度
2. 可以灵活的进行分布式部署。
3. 系统之间耦合度低
4. 团队之间开发不冲突。

缺点：

1. 需要发布Webservice，系统之间需要远程调用。
2. 工作量增加。
3. 需要系统之间协作才能完成整个业务流程，系统之间调用变的复杂。

需要创建两个工程

服务层：taotao-rest：没有表现层，只有业务逻辑。需要发布服务。

表现层：taotao-portal：只有表现层，没有业务逻辑，不需要连接数据库。

表现层和服务层通信，使用Webservice进行通信。Restful形式的Webservice。http+json数据。

## 创建工程

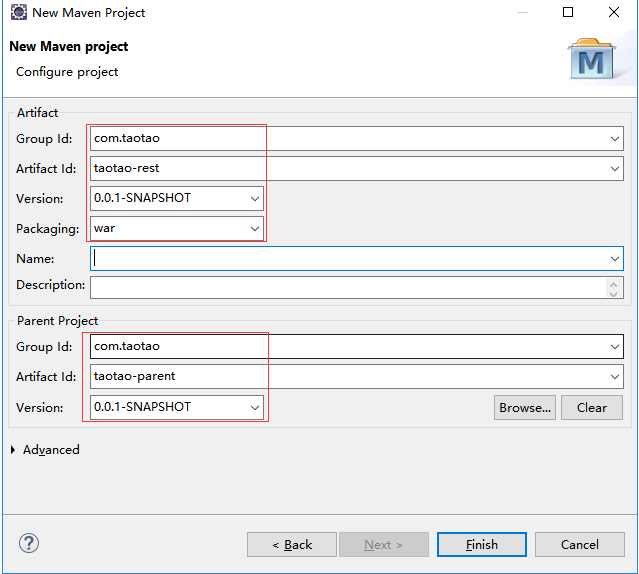
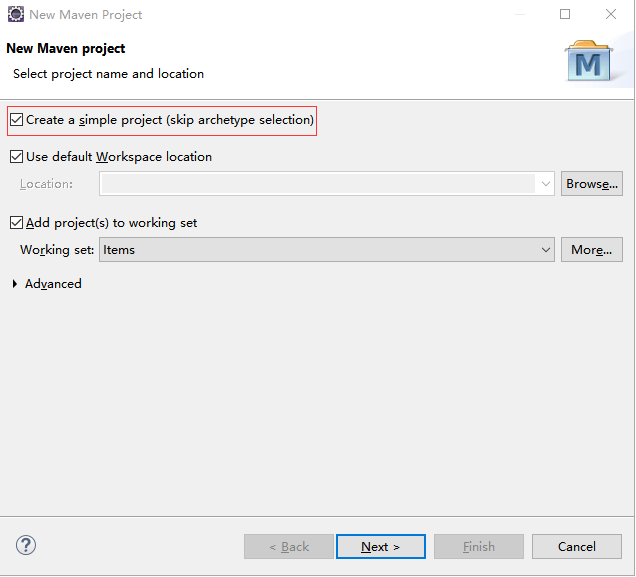
### 服务层taotao-rest

使用maven管理工程：War包。

#### 使用的技术

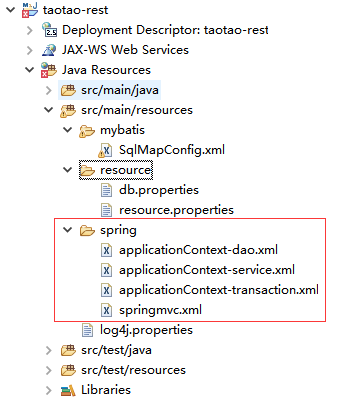
* Mybatis
* Spring
* 发布服务：cxf、springmvc

#### 创建工程



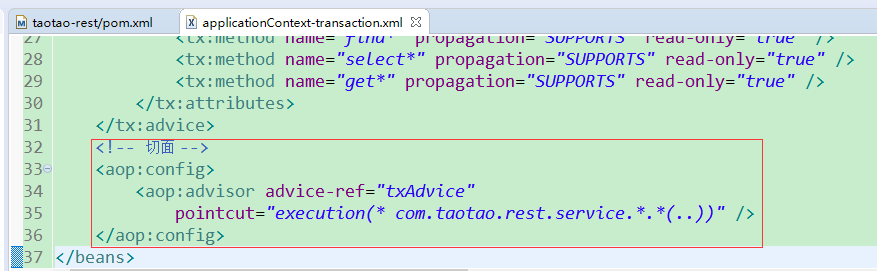
#### 框架整合

整合ssm框架（可以参考taotao-manager工程）



修改说明：





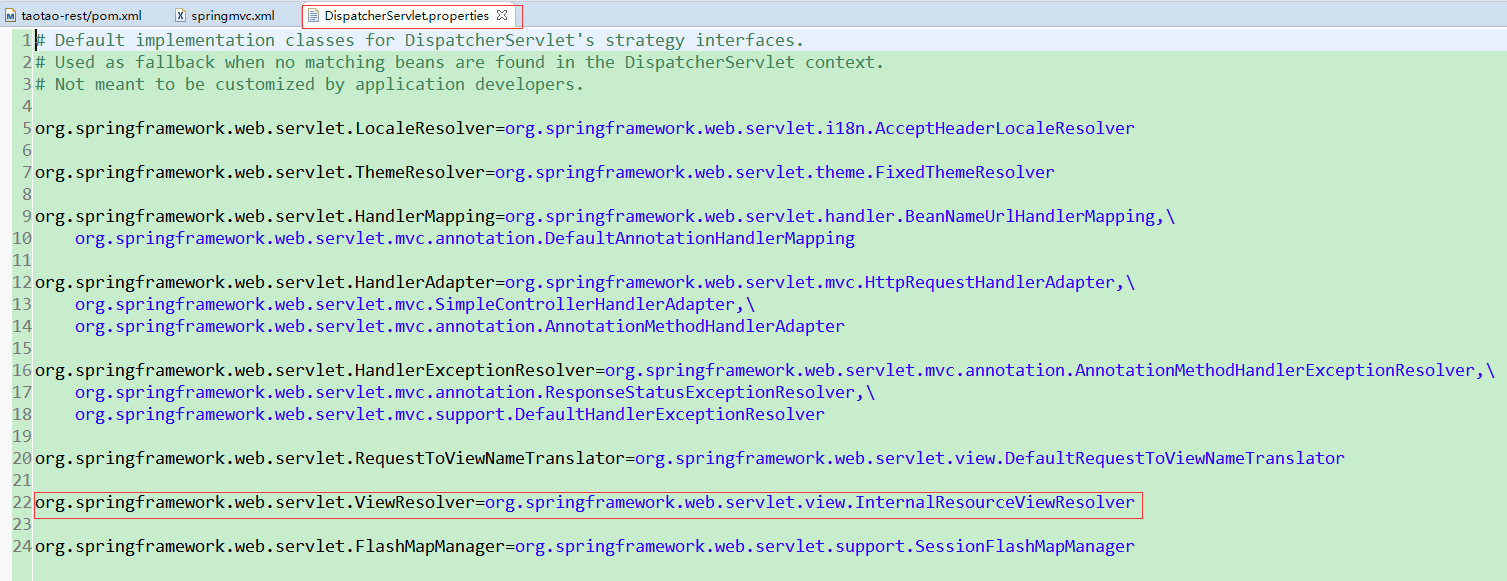


Spring默认配置了视图解析器，如果springmvc.xml中没有配置，默认也可以使用jsp

在对应springmvc的依赖jar中查看相关的信息

org\springframework\spring-webmvc\4.1.3.RELEASE\spring-webmvc-4.1.3.RELEASE.jar

org.springframework.web.servlet包下有DispatcherServlet.properties，默认配置了视图解析器



Web.xml文件复制：修改两个内容--项目名称、拦截路径、控制器

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

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

id=*"taotao"* version=*"2.5"*>

<display-name>taotao-rest</display-name>

<welcome-file-list>

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

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

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

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

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<!-- 加载spring容器 -->

<context-param>

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

<param-value>classpath:spring/applicationContext-\*.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 解决post乱码 -->

<filter>

<filter-name>CharacterEncodingFilter</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>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

<!-- springmvc的前端控制器 -->

<servlet>

<servlet-name>taotao-rest</servlet-name>

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

<!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->

<init-param>

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

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

</init-param>

<!-- load-on-startup元素标记容器是否应该在web应用程序启动的时候就加载这个servlet -->

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

</servlet>

<!-- 配置拦截url的请求:/表示拦截所有的请求(包括静态资源) -->

<servlet-mapping>

<servlet-name>taotao-rest</servlet-name>

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

</servlet-mapping>

</web-app>

#### Pom文件

添加对taotao-manager-dao的依赖

添加spring的依赖

配置tomcat插件

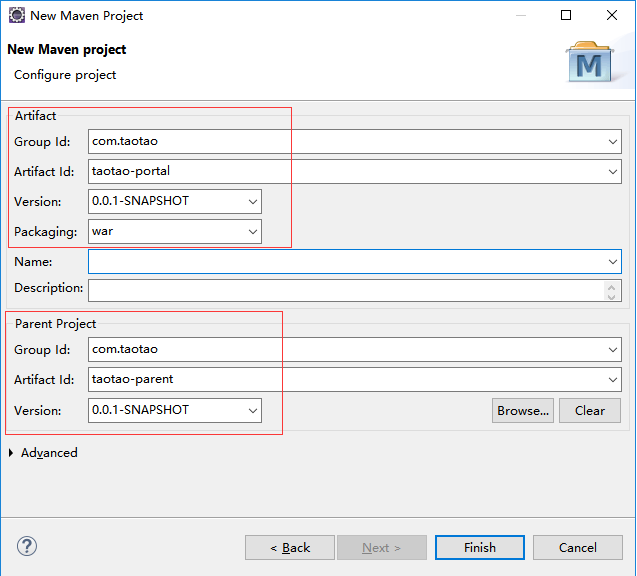
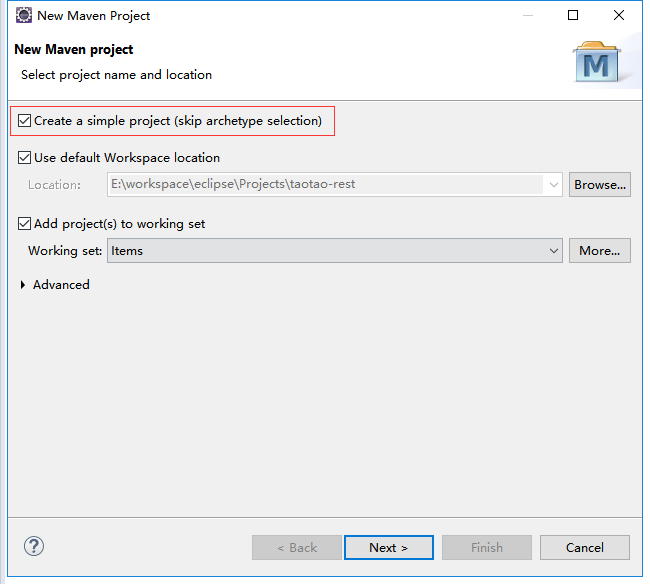
|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.taotao</groupId>  <artifactId>taotao-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.taotao</groupId>  <artifactId>taotao-rest</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>com.taotao</groupId>  <artifactId>taotao-manager-mapper</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <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>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>  <!-- solr客户端 -->  <dependency>  <groupId>org.apache.solr</groupId>  <artifactId>solr-solrj</artifactId>  </dependency>  </dependencies>  <!-- 添加tomcat插件 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8081</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

### 表现层taotao-portal

#### 使用的技术

* Spring
* Springmvc

#### 创建工程



#### 依赖的jar包

1. taotao-manager-pojo
2. Spring
3. Tomcat插件

Pom文件：

|  |
| --- |
| <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-portal</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>com.taotao</groupId>  <artifactId>taotao-common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>com.taotao</groupId>  <artifactId>taotao-manager-pojo</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- 单元测试 -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <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>  <!-- 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>  </dependencies>  <!-- 添加tomcat插件 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8082</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

添加Web.xml文件：

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

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

id=*"taotao"* version=*"2.5"*>

<display-name>taotao-portal</display-name>

<welcome-file-list>

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

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

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

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

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<!-- 加载spring容器 -->

<context-param>

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

<param-value>classpath:spring/applicationContext-\*.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 解决post乱码 -->

<filter>

<filter-name>CharacterEncodingFilter</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>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

<!-- springmvc的前端控制器 -->

<servlet>

<servlet-name>taotao-portal</servlet-name>

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

<!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->

<init-param>

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

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

</init-param>

<!-- load-on-startup元素标记容器是否应该在web应用程序启动的时候就加载这个servlet -->

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

</servlet>

<!-- 配置拦截url的请求:/表示拦截所有的请求(包括静态资源) -->

<servlet-mapping>

<servlet-name>taotao-portal</servlet-name>

<!-- 伪静态化 -->

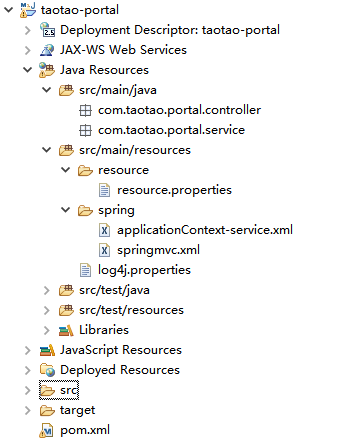
<url-pattern>\*.html</url-pattern>

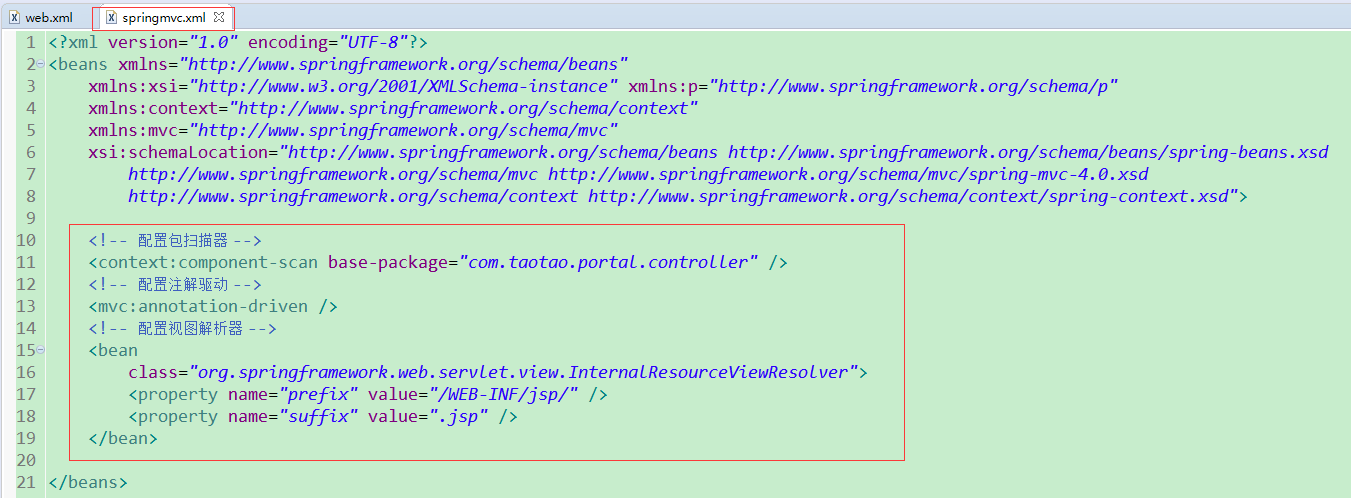
</servlet-mapping>

</web-app>

#### 框架整合

Spring和springmvc

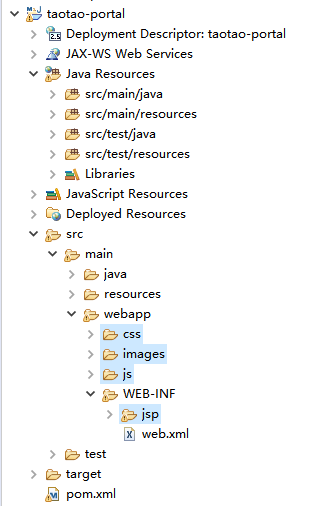




运行过程中可能会报错，要先将相应的工程导入仓库（taotao-manager-pojo、taotao-manager），Maven安装工程到本地仓库，跳过测试：mvn clean install -DskipTests

## 访问首页

将静态文件导入到相应的webapp文件夹下



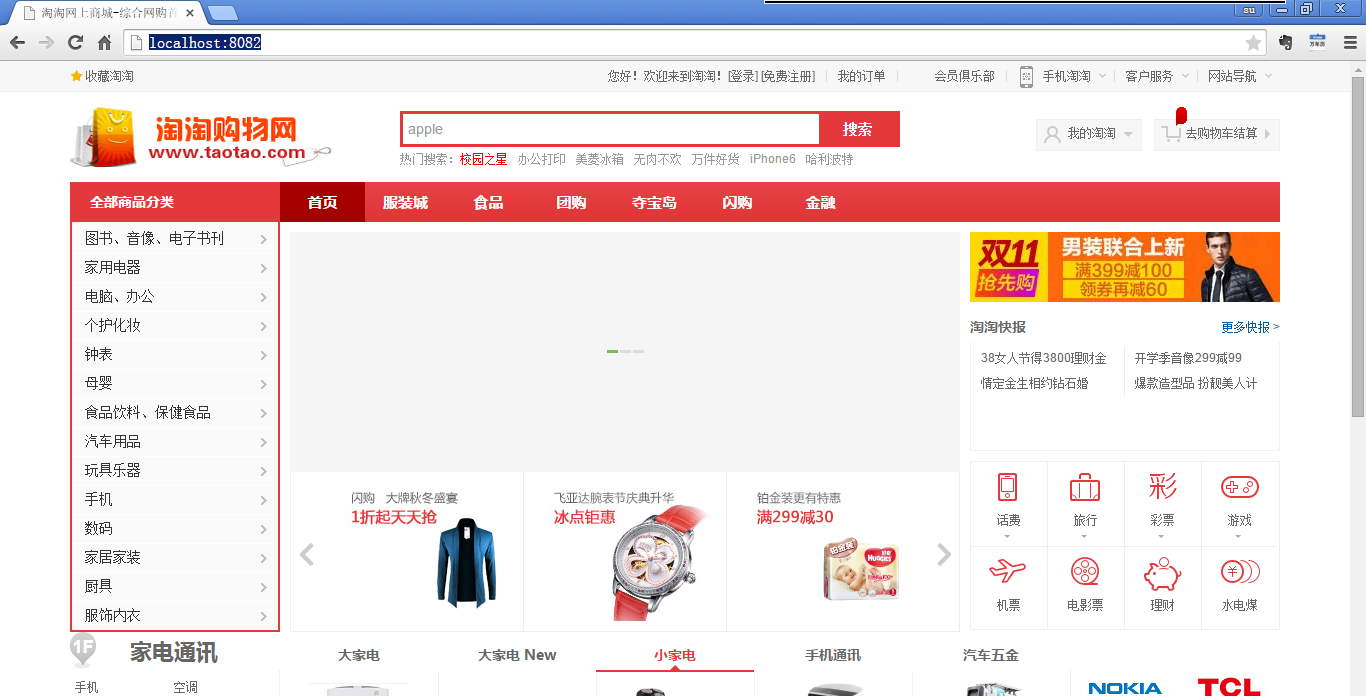
创建一个Controller，当访问http://localhost:8082/跳转到首页。

请求的url：/index

|  |
| --- |
| @Controller  **public** **class** IndexController {  @RequestMapping("/index")  **public** String showIndex() {  **return** "index";  }  } |

启动tomcat测试：clean tomcat7:run,请求路径：<http://localhost:8082/index.html>、http://localhost:8082/

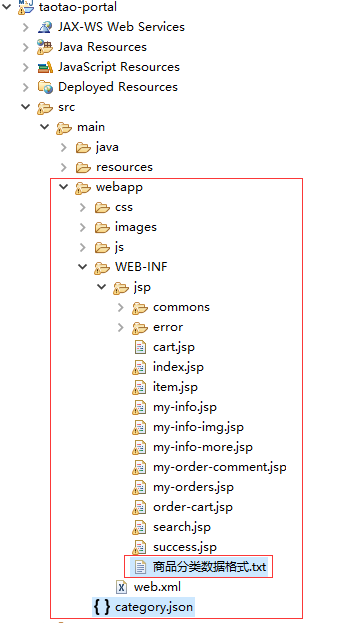
http://localhost:8082/index无法访问（拦截器）



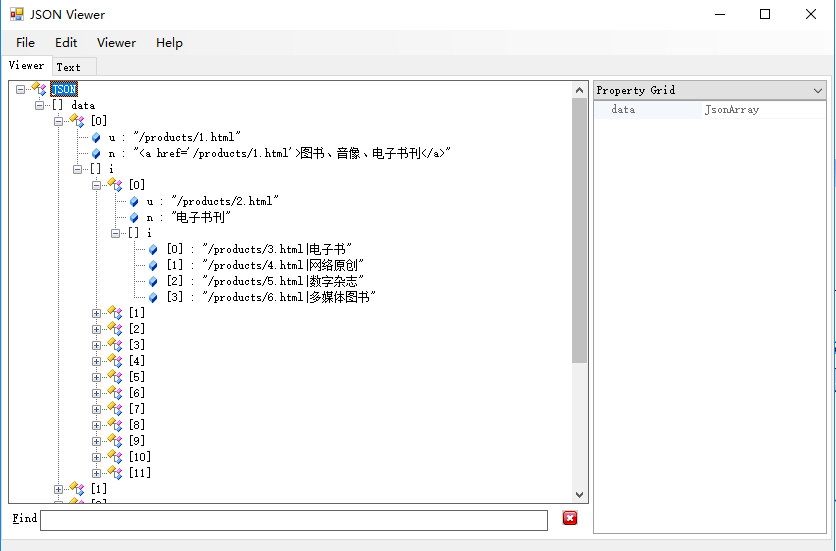
# 首页商品类目展示

## 分析

* 类目展示需要异步加载
* 商品类目从数据库中获得，调用taotao-rest发布的服务获得数据
* 可以在首页中使用js直接调用taotao-rest发布的服务获得数据

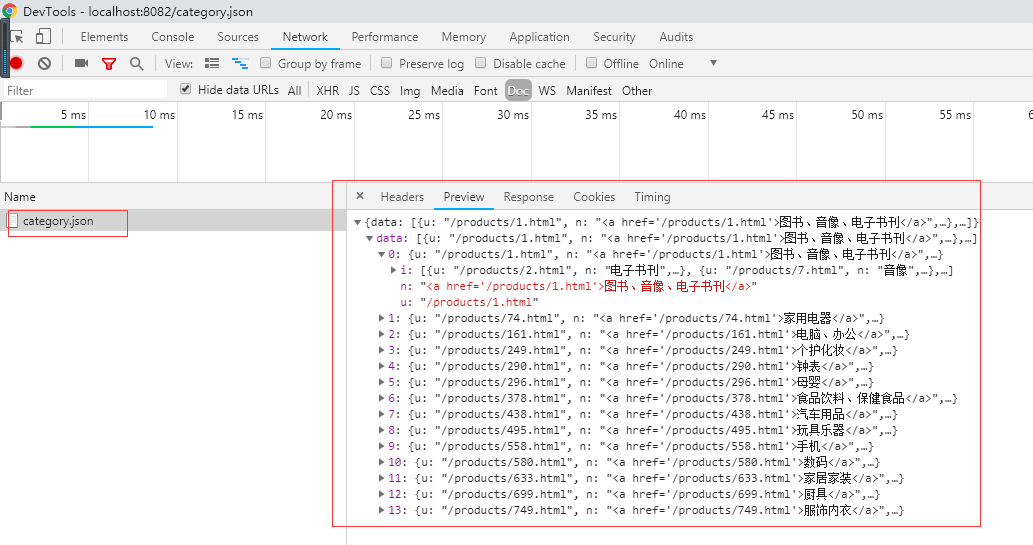
复制对应的内容，在webapp下创建category.json

需要使用到的json数据格式如下：（借助JSON Viewer：format格式化，viewer查看视图）

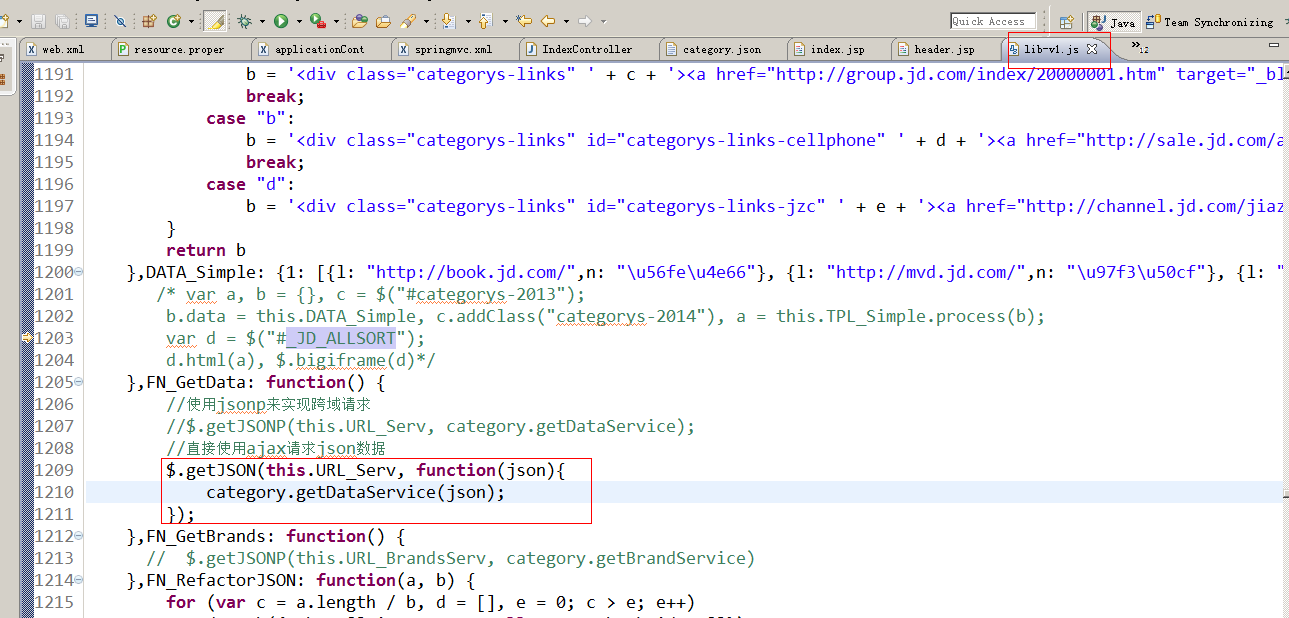


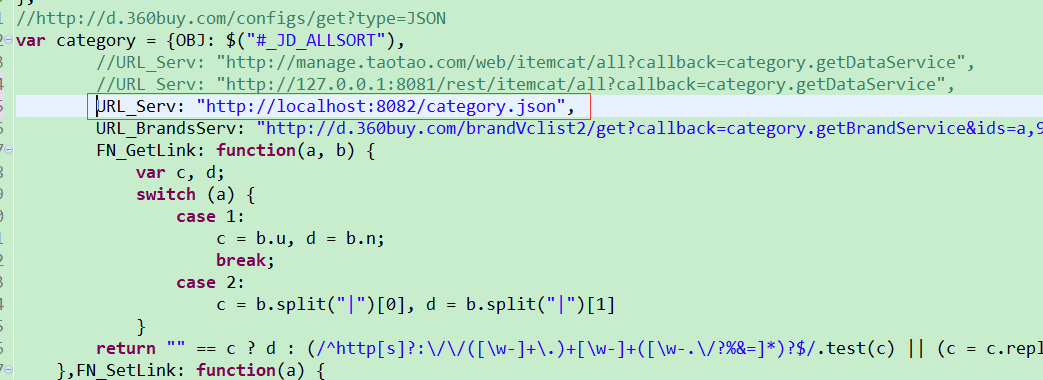
可以通过浏览器直接访问到category.json：<http://localhost:8082/category.json>

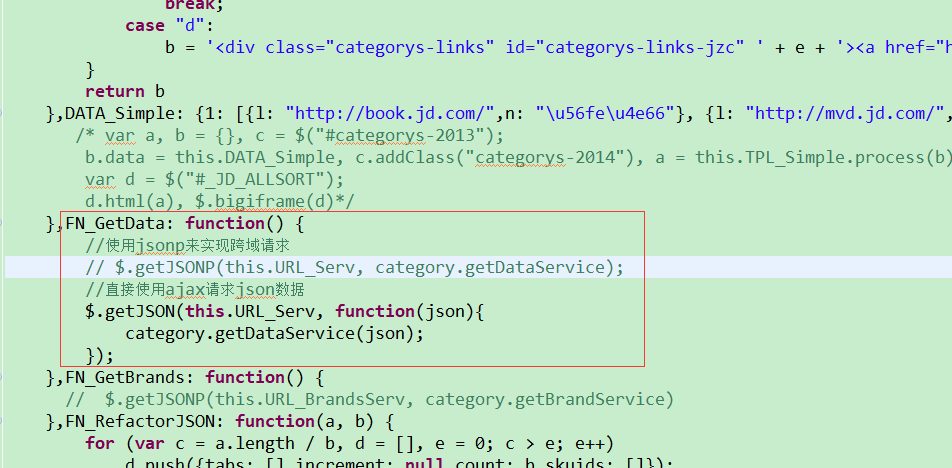




请求数据的js位置：index.jsp-->header.jsp-->lib-vl.js

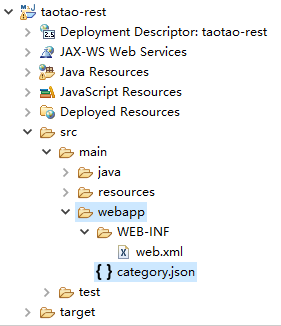




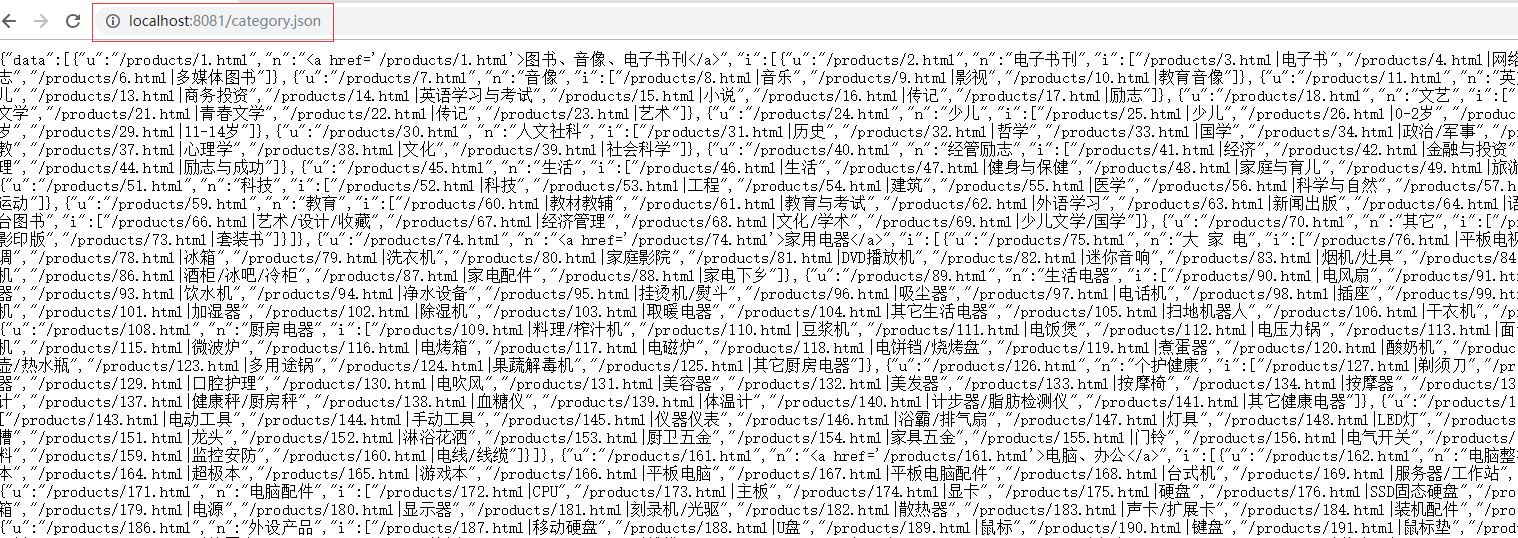




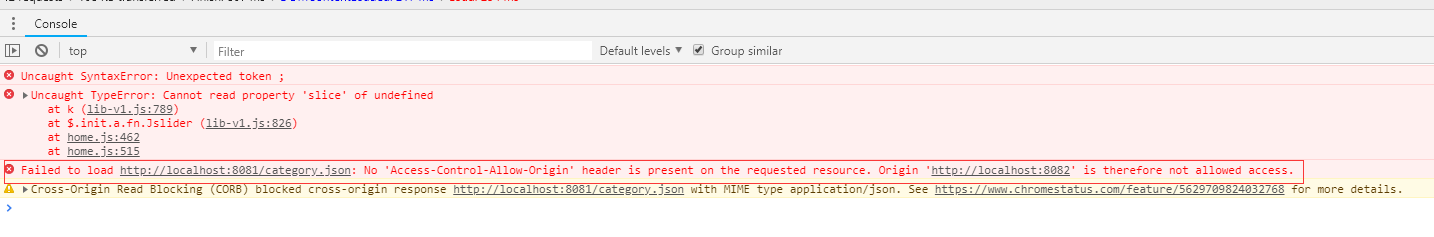
将category.json放到taotao-rest的webapp文件夹下，重启taotao-rest服务器访问



localhost：8081/category.json



此时再次修改lib-v1.js文件，实现由taotao-portal访问taotao-rest的category.json进行测试，出现如下问题：涉及到js跨域问题



## Js跨域

### 什么是跨域

Js为了安全有一个限制，不允许跨域访问。

* 如果两个url的域名不同
* Url相同，端口不同也是跨域
* Ip不同也是跨域

在taotao-portal中不能使用ajax直接调用taotao-rest的服务。

### 解决跨域的问题

可以使用jsonp解决跨域的问题。

1、在js中不能跨域请求数据，js可以跨域请求js片段。

2、可以把数据包装成js片段。可以把数据使用js方法来包装，形成一条方法的调用语句。

3、可以使用ajax请求js片段，当js判断到达浏览器会被立即执行。

4、在浏览器端，先创建好回调方法，在回调方法中通过参数可以获得请求的数据。

Taotao-portal

Taotao-rest

json

Ajax请求json数据

Ajax请求js

1、请求js数据

Js，其中包含json

json

js执行，调用js方法，要求此方法已经定义。

2、响应js

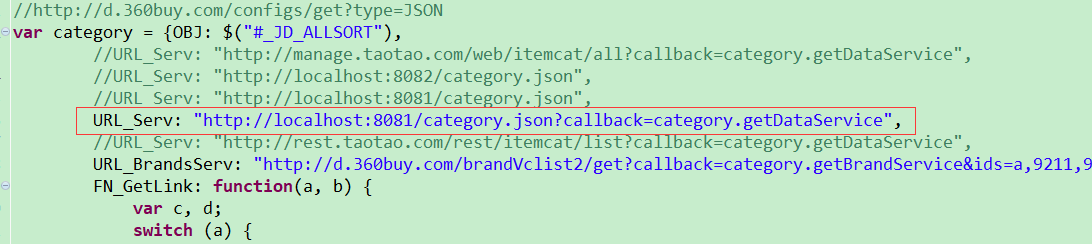
3、取出json

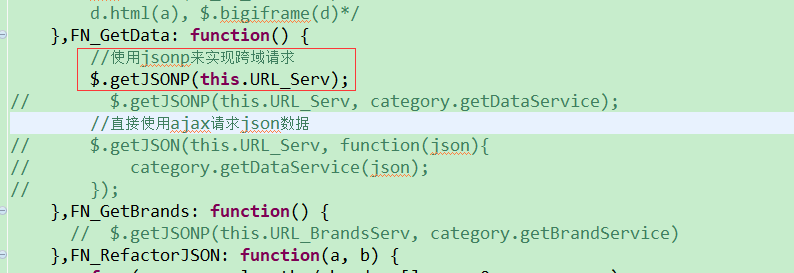
json

前期准备：

* 需要把js的回调方法先写好。（将json数据作为参数封装到某个方法）
* 做ajax请求时，需要把回调方法的方法名传递给服务端
* 服务端接收回调方法名，把数据包装好，响应给客户端





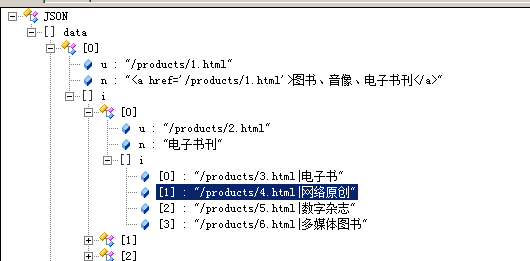




# 发布商品列表查询服务

**在测试的时候需要先将taotao-rest服务启动，随后再将taotao-portal启动进行测试**

## 分析（获取动态数据）



获得一个分类列表的数据，从tb\_item\_cat表中取数据。

可以创建一个pojo描述树形列表的节点，其中包含三个属性u\n\i。

U：url String

N：name String

I：items List

在taotao-rest创建com.taotao.rest.pojo包，添加CatNode类

|  |
| --- |
| **package** com.taotao.rest.pojo;  **import** java.util.List;  **import** com.fasterxml.jackson.annotation.JsonProperty;  **public** **class** CatNode {  @JsonProperty("u")  **private** String url;  @JsonProperty("n")  **private** String name;  @JsonProperty("i")  **private** List items;  **public** String getUrl() {  **return** url;  }  **public** **void** setUrl(String url) {  **this**.url = url;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** List getItems() {  **return** items;  }  **public** **void** setItems(List items) {  **this**.items = items;  }  } |

|  |
| --- |
| **package** com.taotao.rest.pojo;  **import** java.util.List;  **public** **class** ItemCatResult {  **private** List data;  **public** List getData() {  **return** data;  }  **public** **void** setData(List data) {  **this**.data = data;  }  } |

## Dao层

从tb\_item\_cat表中取数据，单表查询，可以使用逆向工程生成的代码。

## Service层

返回结果：ItemCatResult

参数：没有

业务逻辑：根据parentid查询子节点列表，并递归调用。

|  |
| --- |
| **public** **interface** ItemCatService {  // 获取分类列表  **public** ItemCatResult getItemCatList();  }  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.taotao.mapper.TbItemCatMapper;  **import** com.taotao.pojo.TbItemCat;  **import** com.taotao.pojo.TbItemCatExample;  **import** com.taotao.pojo.TbItemCatExample.Criteria;  **import** com.taotao.rest.pojo.CatNode;  **import** com.taotao.rest.pojo.ItemCatResult;  **import** com.taotao.rest.service.ItemCatService;  @Service  **public** **class** ItemCatServiceImpl **implements** ItemCatService {  @Autowired  **private** TbItemCatMapper itemCatMapper;    @Override  **public** ItemCatResult getItemCatList() {  //调用递归方法查询商品分类列表  List catList = getItemCatList(0l);  //返回结果  ItemCatResult result = **new** ItemCatResult();  result.setData(catList);  **return** result;  }    **private** List getItemCatList(Long parentId) {  //根据parentId查询列表  TbItemCatExample example = **new** TbItemCatExample();  Criteria criteria = example.createCriteria();  criteria.andParentIdEqualTo(parentId);  //执行查询  List<TbItemCat> list = itemCatMapper.selectByExample(example);  List resultList = **new** ArrayList<>();  **for** (TbItemCat tbItemCat : list) {  //如果是父节点  **if** (tbItemCat.getIsParent()) {  CatNode node = **new** CatNode();  node.setUrl("/products/"+tbItemCat.getId()+".html");  //如果当前节点为第一级节点  **if** (tbItemCat.getParentId() == 0) {  node.setName("<a href='/products/"+tbItemCat.getId()+".html'>"+tbItemCat.getName()+"</a>");  } **else** {  node.setName(tbItemCat.getName());  }  node.setItems(getItemCatList(tbItemCat.getId()));  //把node添加到列表  resultList.add(node);  } **else** {  //如果是叶子节点  String item = "/products/"+tbItemCat.getId()+".html|" + tbItemCat.getName();  resultList.add(item);  }  }  **return** resultList;  }  } |

## Controller

响应一个json数据。判断callback参数是否为空，如果为空正常返回json数据，如果不为空，支持jsonp调用。

需要使用@ResponseBody注解。

测试：

@Controller

@RequestMapping("/item/cat")

**public** **class** ItemCatController {

@Autowired

**private** ItemCatService itemCatService;

@RequestMapping(value="/list")

@ResponseBody

**public** ItemCatResult testGetItemCatList() {

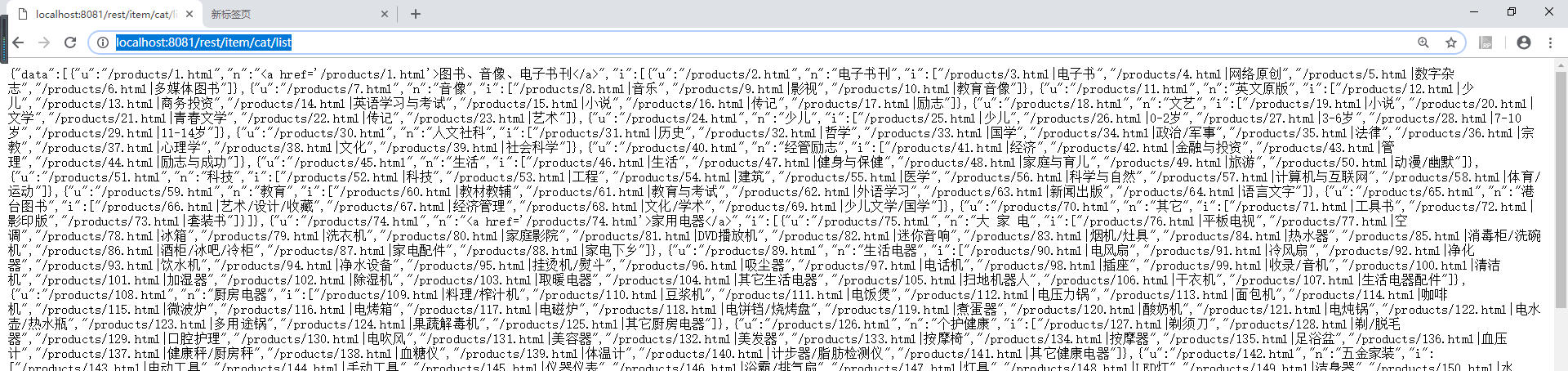
**return** itemCatService.getItemCatList();

}

}

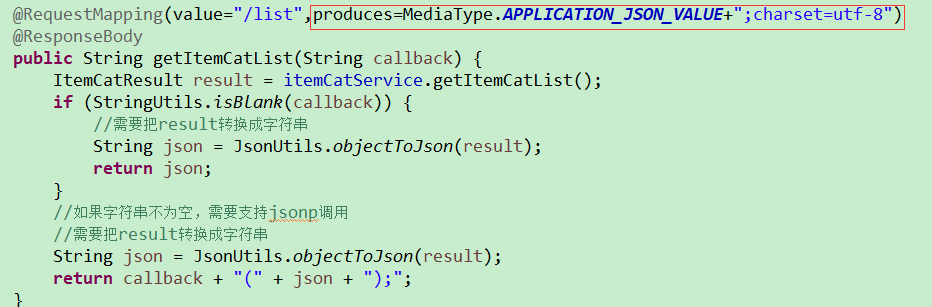
访问测试：

<http://localhost:8081/rest/item/cat/list>



支持jsonp两种方式：

第一种：直接响应字符串（乱码问题解决）



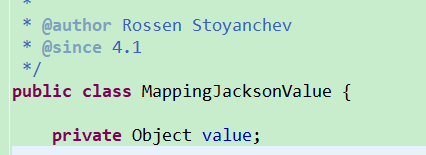
|  |
| --- |
| @RequestMapping(value="/list",produces=MediaType.***APPLICATION\_JSON\_VALUE***+";charset=utf-8")  @ResponseBody  **public** String getItemCatList(String callback) {  ItemCatResult result = itemCatService.getItemCatList();  **if** (StringUtils.*isBlank*(callback)) {  //需要把result转换成字符串  String json = JsonUtils.*objectToJson*(result);  **return** json;  }  //如果字符串不为空，需要支持jsonp调用  //需要把result转换成字符串  String json = JsonUtils.*objectToJson*(result);  **return** callback + "(" + json + ");";  } |

<http://localhost:8081/rest/item/cat/list>数据正常访问

第二种方法：

要求springmvc必须是4.1以上版本。

MappingJacksonValue



|  |
| --- |
| @RequestMapping(value="/list")  @ResponseBody  **public** Object getItemCatList(String callback) {  ItemCatResult result = itemCatService.getItemCatList();  **if** (StringUtils.*isBlank*(callback)) {  //需要把result转换成字符串  **return** result;  }  //如果字符串不为空，需要支持jsonp调用  MappingJacksonValue mappingJacksonValue = **new** MappingJacksonValue(result);  mappingJacksonValue.setJsonpFunction(callback);  **return** mappingJacksonValue;  } |

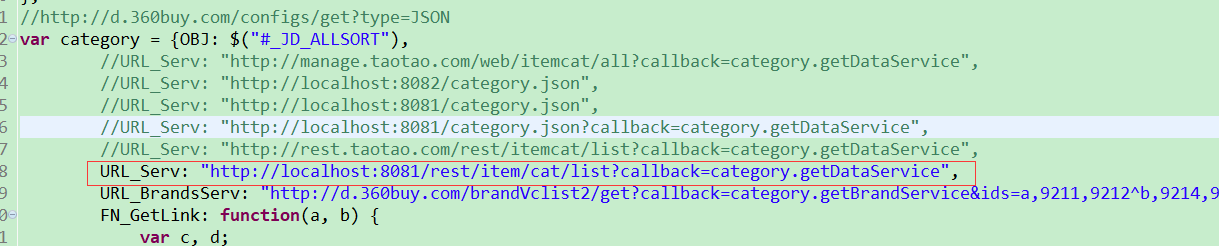
访问测试：

<http://localhost:8081/rest/item/cat/list?callback=function1>

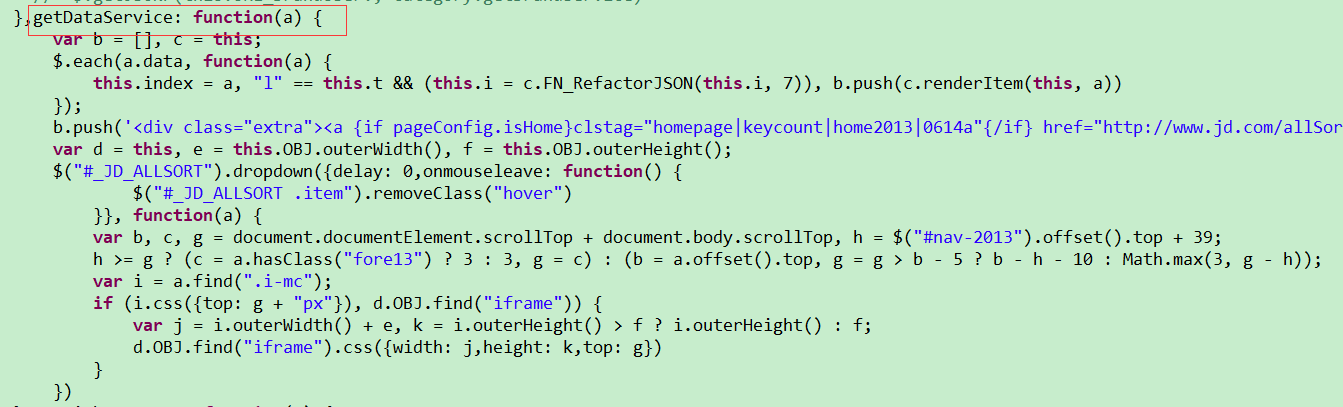
<http://localhost:8081/rest/item/cat/list?callback=functionn>

......

## Portal中js处理







限制节点个数，不然会默认将所有的分类节点查找出来：在taotao-rest服务端进行处理



