# 今日大纲

1. 搭建前台系统
2. 实现商品分类展示

# 搭建前台系统

## 所使用的技术

前台技术：html、CSS、Jquery

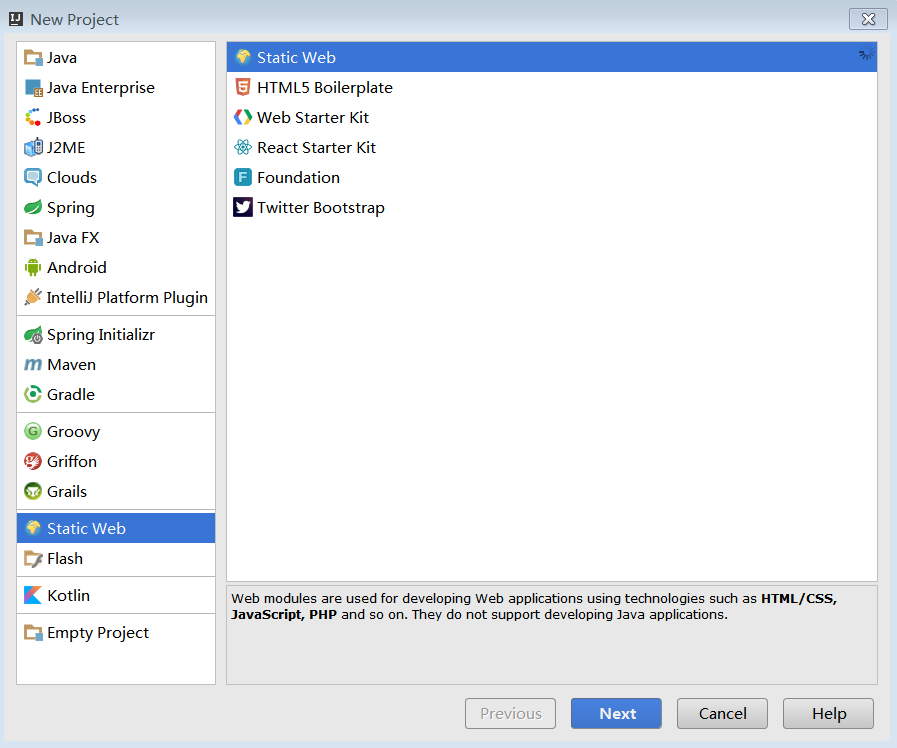
如果不使用Mybatis，商品的数据从何而来？ --来源于Mysql数据库

获取数据的途径：

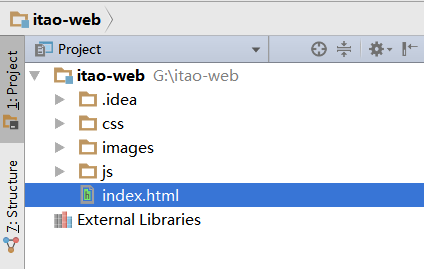
1. 从JDBC获取
   1. 优点
      1. 直接，获取的途径较短，简单
   2. 缺点
      1. 对后台系统团队而言，数据不安全（只要开放的账户是只读的账户即可）
      2. 前端系统团队需要有学习的成本，才能使用数据库
      3. 依赖、耦合度太高，后端团队将数据库结构修改，那么其他团队必须跟着修改逻辑，才能使用
      4. 直接走数据库查询，无法添加缓存逻辑
2. 通过后台系统接口获取
   1. 优点
      1. 耦合度降低，后端团队只要保证接口的返回数据格式不变化，其他团队就无需升级
      2. 数据安全
      3. 前端团队无需了解学习后端团队的底层数据库结构
      4. 后端团队可以在接口处添加缓存逻辑

## 创建itao-web

静态web项目



## 导入静态文件



引入 css images js文件夹，和商品首页index.html

## 配置hosts和nginx

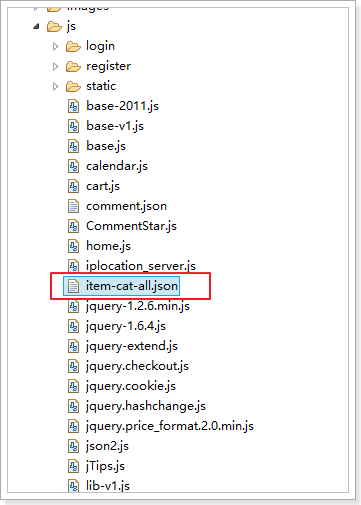
## 测试

# 首页Banner显示

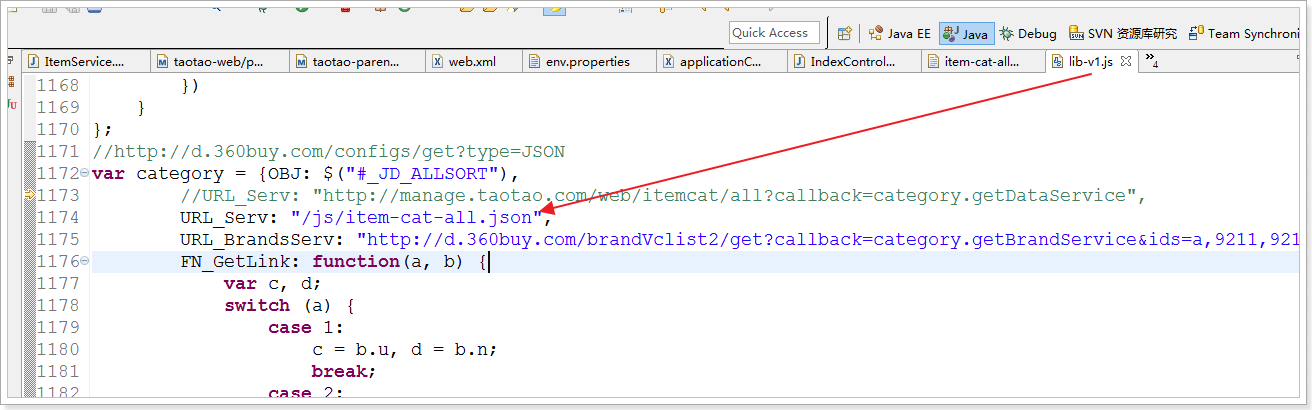
## 功能



## 构造数据



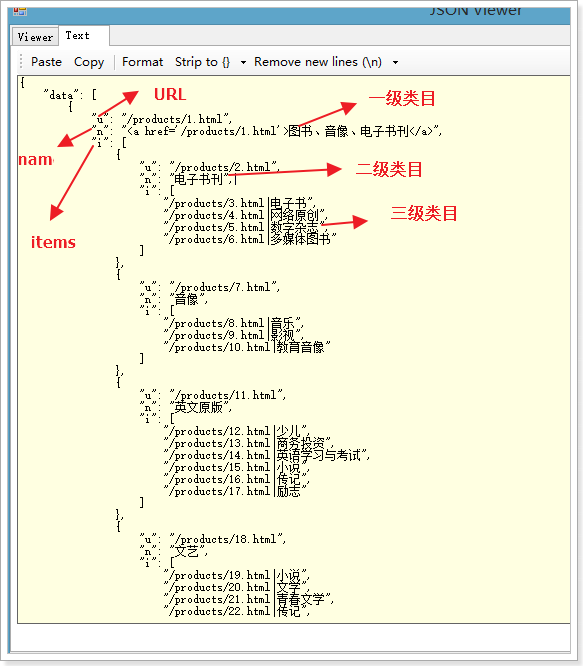
## 修改JS



## 测试



## 数据结构



## 后台系统开发接口返回数据

### ItemCatData

package com.zhiyou100.manage.api.model;  
  
import java.util.List;  
  
*/\*\*  
 \* Created by lijike 2016/12/25.  
 \*/*public class ItemCatData {  
 private String url;  
 private String name;  
 private List<ItemCatData> 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<ItemCatData> getItems() {  
 return items;  
 }  
  
 public void setItems(List<ItemCatData> items) {  
 this.items = items;  
 }  
}

### Service

### Controller



### Service（强烈建议自己写一遍，不是抄一遍）

/\*\*

\* 全部查询，并且生成树状结构

\*

\* **@return**

\*/

**public** ItemCatResult queryAllToTree() {

ItemCatResult result = **new** ItemCatResult();

// 全部查出，并且在内存中生成树形结构

List<ItemCat>cats = **super**.queryAll();

// 转为map存储，key为父节点ID，value为数据集合

Map<Long, List<ItemCat>>itemCatMap = **new** HashMap<Long, List<ItemCat>>();

**for** (ItemCat itemCat : cats) {

**if** (!itemCatMap.containsKey(itemCat.getParentId())) {

itemCatMap.put(itemCat.getParentId(), **new** ArrayList<ItemCat>());

}

itemCatMap.get(itemCat.getParentId()).add(itemCat);

}

// 封装一级对象

List<ItemCat>itemCatList1 = itemCatMap.get(0L);

**for** (ItemCat itemCat : itemCatList1) {

ItemCatData itemCatData = **new** ItemCatData();

itemCatData.setUrl("/products/" + itemCat.getId() + ".html");

itemCatData.setName("<a href='" + itemCatData.getUrl() + "'>" + itemCat.getName() + "</a>");

result.getItemCats().add(itemCatData);

**if** (!itemCat.getIsParent()) {

**continue**;

}

// 封装二级对象

List<ItemCat>itemCatList2 = itemCatMap.get(itemCat.getId());

List<ItemCatData>itemCatData2 = **new** ArrayList<ItemCatData>();

itemCatData.setItems(itemCatData2);

**for** (ItemCat itemCat2 : itemCatList2) {

ItemCatData id2 = **new** ItemCatData();

id2.setName(itemCat2.getName());

id2.setUrl("/products/" + itemCat2.getId() + ".html");

itemCatData2.add(id2);

**if** (itemCat2.getIsParent()) {

// 封装三级对象

List<ItemCat>itemCatList3 = itemCatMap.get(itemCat2.getId());

List<String>itemCatData3 = **new** ArrayList<String>();

id2.setItems(itemCatData3);

**for** (ItemCat itemCat3 : itemCatList3) {

itemCatData3.add("/products/" + itemCat3.getId() + ".html|" + itemCat3.getName());

}

}

}

**if** (result.getItemCats().size() >= 14) {

**break**;

}

}

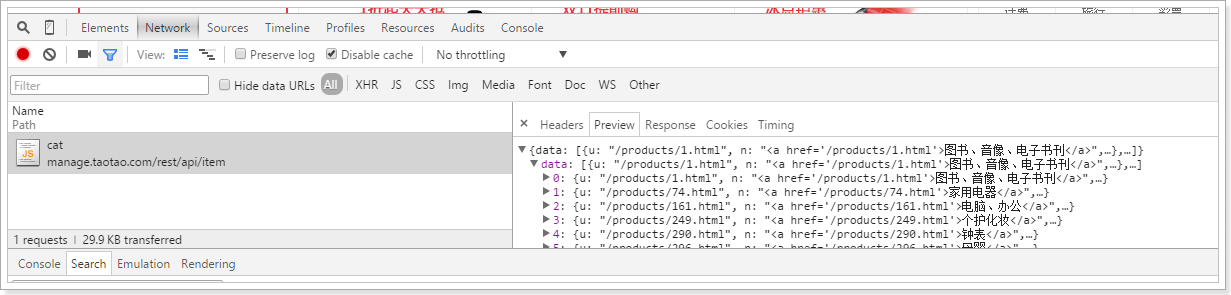
**return**result;

}

### 测试



### 集成到前台系统



可以看到数据是以及获取到。

但是，出错（JS解析出错）：



## 跨域问题

浏览器对ajax请求的限制，不允许跨域请求资源。

<http://www.a.com>🡺<http://www.b.com>是跨域

<http://www.a.com>🡺<http://www.a.com>:8080是跨域

<http://a.a.com>🡺<http://b.a.com>是跨域

<http://www.a.com>🡺<http://www.a.com/api>不是

总结：

不同的域名或不同的端口都是跨域请求。

如何解决跨域问题？ --jsonp

## Jsonp

### 编写json.jsp

在itao-manager-web模块webapp下创建json.jsp

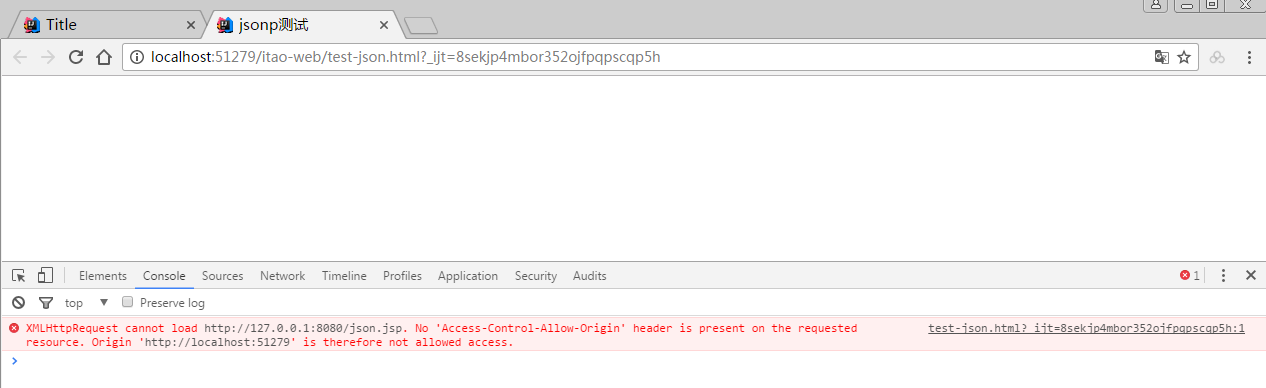
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
**<%** out.print("{\"abc\":123}");  
**%>**

### 在test-json.html

在itao-web模块编写test-json.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>jsonp测试</title>  
 <script type="text/javascript" src="js/jquery.js"></script>  
 <script type="text/javascript">  
 alert("aaa");  
 $(document).ready(**function** () {  
 $.ajax({  
 type: "GET",  
 url: "http://127.0.0.1:8080/json.jsp",  
 dataType: "json",  
 success: **function** (data) {  
 alert(data.abc);  
 }  
 });  
 });  
 </script>  
</head>  
<body>  
  
</body>  
</html>

### 测试



发现:

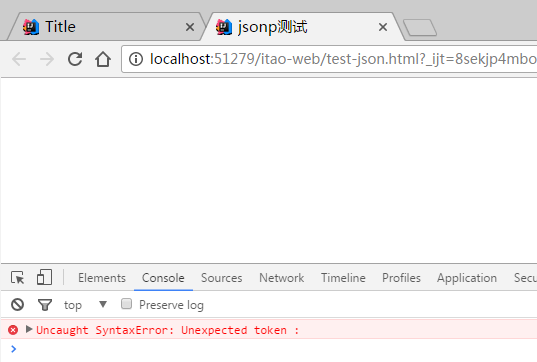
1. alert(‘aaa’) 可以正常弹出
2. alert(data.abc) 不能够正常的弹出，出现跨域问题

结论：script标签的src可以跨域请求资源，但是ajax请求不可以跨域请求。

疑问：能否借助script标签的src进行加载数据？ --可以的。

### 借助script的src跨域加载资源

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>jsonp测试</title>  
 <script type="text/javascript" src="js/jquery.js"></script>  
 <script type="text/javascript" src="http://127.0.0.1:8080/json.jsp"></script>  
 </head>  
<body>  
  
</body>  
</html>



发现：

请求资源可以正常请求，但是，报js解析出错。

原因：

Script标签加载到资源后，会将资源当做是js脚本解析，但是我们返回的是json数据，所以导致解析失败。

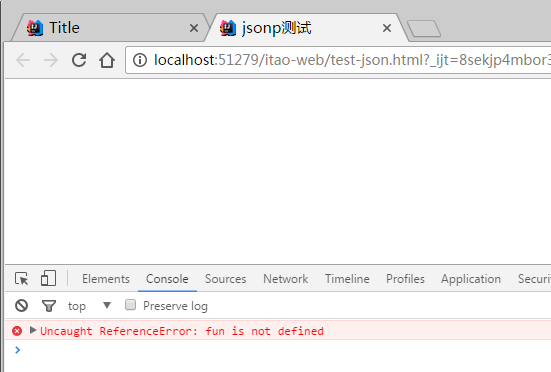
解决：只需要返回js脚本即可。

### 后端系统返回js脚本

test.jsp修改为：

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
**<%** out.print("fun({\"abc\":123})");  
**%>**

测试：



发现:

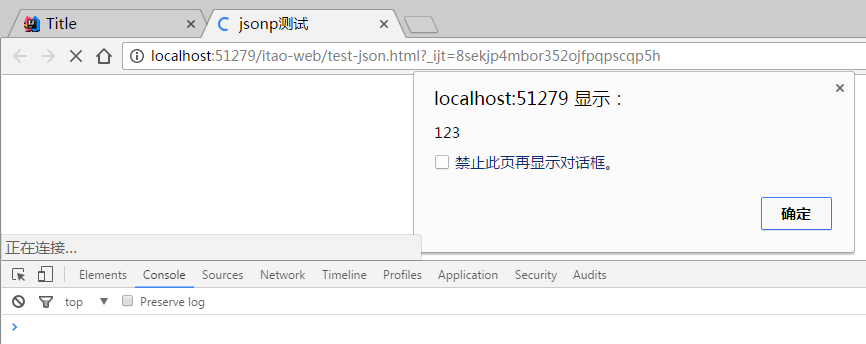
返回的js脚本成功解析，但是，fun没有定义。

解决：定义个一个fun方法即可。

### 定义fun方法

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>jsonp测试</title>  
 <script type="text/javascript" src="js/jquery.js"></script>  
 <script>  
 **function** fun(data) {  
 alert(data.abc)  
 }  
 </script>  
 <script type="text/javascript" src="http://127.0.0.1:8080/json.jsp"></script>  
</head>  
<body>  
  
</body>  
</html>

测试：



### 总结

Jsonp的原理：

1. jsonp通过script标签的src可以跨域请求的特性，加载资源
2. 将加载的资源（通过一个方法名将数据进行包裹）当做是js脚本解析
3. 定义一个回调函数，获取传入的数据

### 优化

将回调函数名传递到服务端，返回：

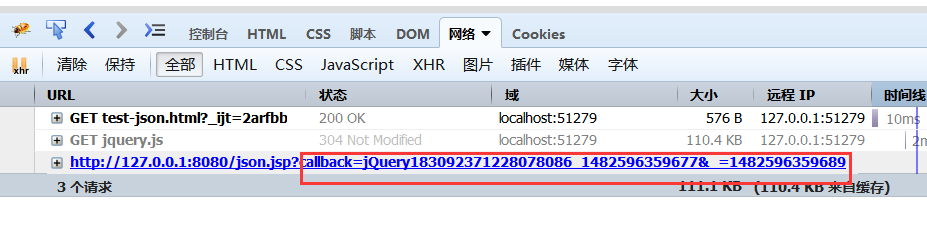
<%@ **page** language="java" contentType="text/html; charset=UTF-8"  
 pageEncoding="UTF-8" %>  
**<%** String callback = request.getParameter("callback");  
 if (callback != null) {  
 out.print(callback + "({\"abc\":123})");  
 } else {  
 out.print("{\"abc\":123}");  
 }  
**%>**

调用方：

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>jsonp测试</title>  
 <script type="text/javascript" src="js/jquery.js"></script>  
 <script>  
 **function** getJson(data) {  
 alert(data.abc)  
 }  
 </script>  
 <script type="text/javascript" src="http://127.0.0.1:8080/json.jsp?callback=getJson"></script>  
 </head>  
<body>  
  
</body>  
</html>

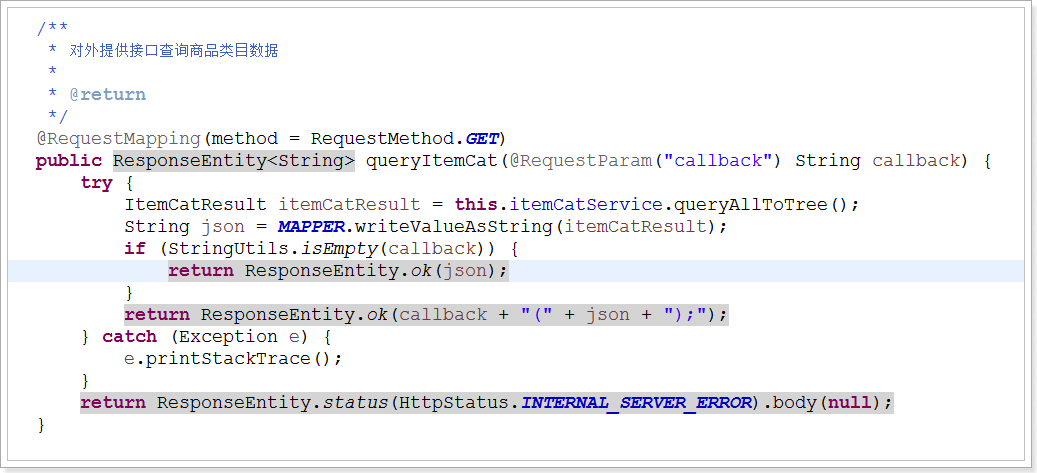
### jQuery使用jsonp

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>jsonp测试</title>  
 <script type="text/javascript" src="js/jquery.js"></script>  
 <script type="text/javascript">  
 alert($);  
 $(**function**(){  
 $.ajax({  
 type : "GET",  
 url : "http://127.0.0.1:8080/json.jsp",  
 dataType : "jsonp",  
 success : **function**(data){  
 alert(data.abc);  
 }  
 });  
 });  
 </script>  
</head>  
<body>  
  
</body>  
</html>

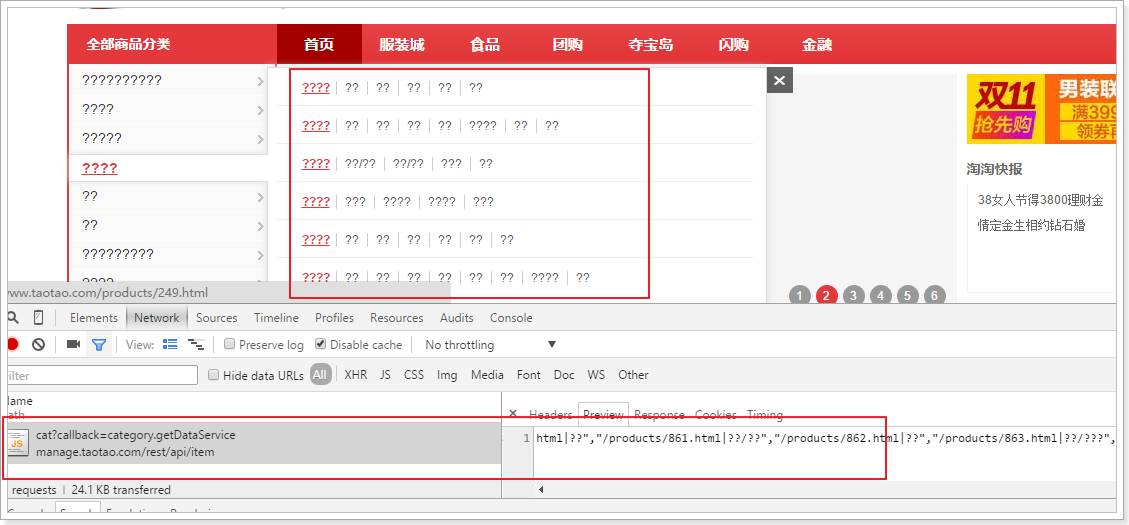


## 解决项目中跨域问题

### 后台系统Controller



### 测试



问题解决，但是带来了新问题，乱码问题。

## 解决乱码问题

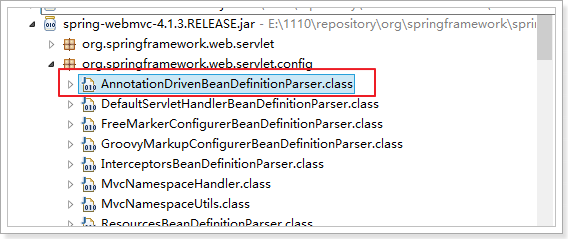
### 分析乱码产生的原因

在SpringMVC中产生的响应有2类：

1. ModelAndView
2. 返回数据响应
   1. 使用消息转化器完成

配置：





配置默认的消息转化器：

**private** ManagedList<?> getMessageConverters(Element element, Object source, ParserContext parserContext) {

Element convertersElement = DomUtils.*getChildElementByTagName*(element, "message-converters");

ManagedList<? **super** Object>messageConverters = **new** ManagedList<Object>();

**if** (convertersElement != **null**) {

messageConverters.setSource(source);

**for** (Element beanElement : DomUtils.*getChildElementsByTagName*(convertersElement, "bean", "ref")) {

Object object = parserContext.getDelegate().parsePropertySubElement(beanElement, **null**);

messageConverters.add(object);

}

}

**if** (convertersElement == **null** || Boolean.*valueOf*(convertersElement.getAttribute("register-defaults"))) {

messageConverters.setSource(source);

messageConverters.add(createConverterDefinition(ByteArrayHttpMessageConverter.**class**, source));

RootBeanDefinition stringConverterDef = createConverterDefinition(StringHttpMessageConverter.**class**, source);

stringConverterDef.getPropertyValues().add("writeAcceptCharset", **false**);

messageConverters.add(stringConverterDef);

messageConverters.add(createConverterDefinition(ResourceHttpMessageConverter.**class**, source));

messageConverters.add(createConverterDefinition(SourceHttpMessageConverter.**class**, source));

messageConverters.add(createConverterDefinition(AllEncompassingFormHttpMessageConverter.**class**, source));

**if** (*romePresent*) {

messageConverters.add(createConverterDefinition(AtomFeedHttpMessageConverter.**class**, source));

messageConverters.add(createConverterDefinition(RssChannelHttpMessageConverter.**class**, source));

}

**if** (***jackson2XmlPresent***) {

RootBeanDefinition jacksonConverterDef = createConverterDefinition(MappingJackson2XmlHttpMessageConverter.**class**, source);

GenericBeanDefinition jacksonFactoryDef = createObjectMapperFactoryDefinition(source);

jacksonFactoryDef.getPropertyValues().add("createXmlMapper", **true**);

jacksonConverterDef.getConstructorArgumentValues().addIndexedArgumentValue(0, jacksonFactoryDef);

messageConverters.add(jacksonConverterDef);

}

**elseif** (***jaxb2Present***) {

messageConverters.add(createConverterDefinition(Jaxb2RootElementHttpMessageConverter.**class**, source));

}

**if** (***jackson2Present***) {

RootBeanDefinition jacksonConverterDef = createConverterDefinition(MappingJackson2HttpMessageConverter.**class**, source);

GenericBeanDefinition jacksonFactoryDef = createObjectMapperFactoryDefinition(source);

jacksonConverterDef.getConstructorArgumentValues().addIndexedArgumentValue(0, jacksonFactoryDef);

messageConverters.add(jacksonConverterDef);

}

**elseif** (***gsonPresent***) {

messageConverters.add(createConverterDefinition(GsonHttpMessageConverter.**class**, source));

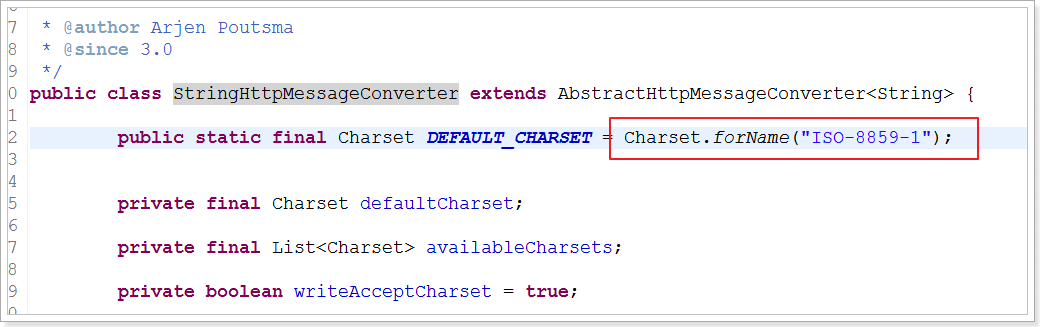
}

}

**return**messageConverters;

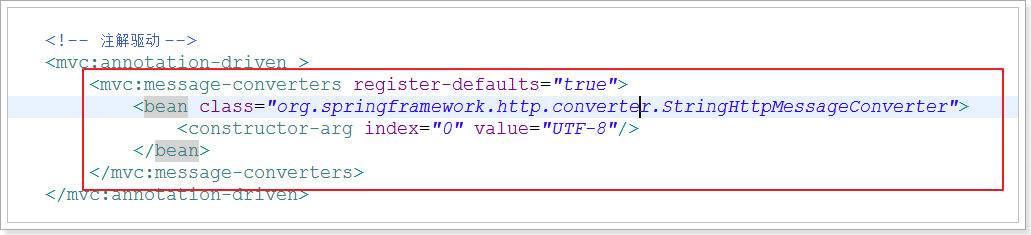
}

### 默认使用的字符串的消息转化器

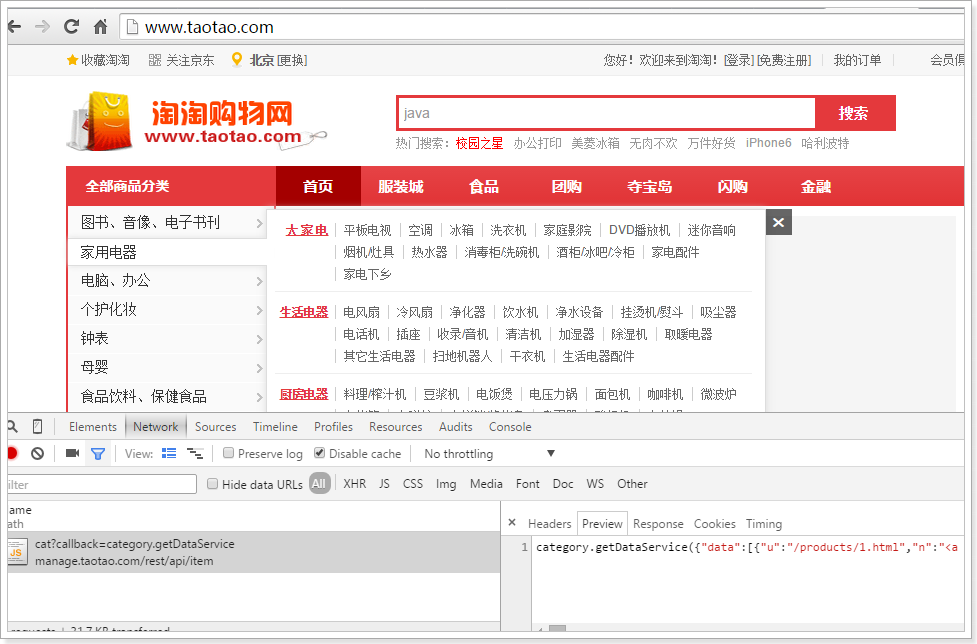


发现，默认使用ISO-8859-1，所以会产生乱码。

### 解决乱码



问题解决：

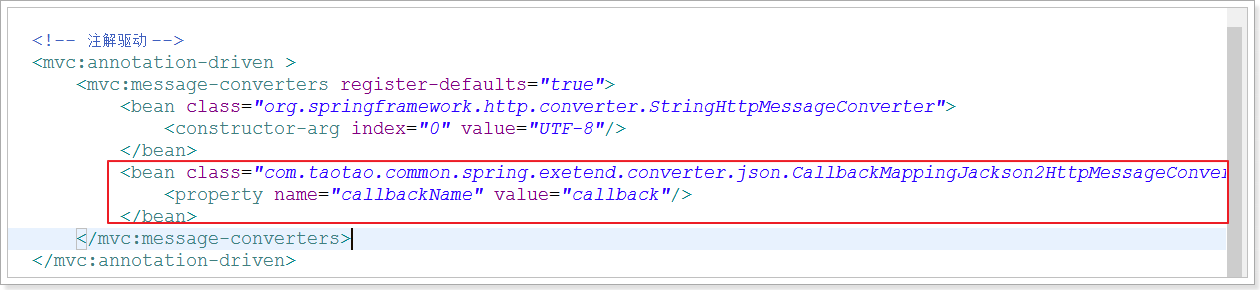


## 统一支持jsonp

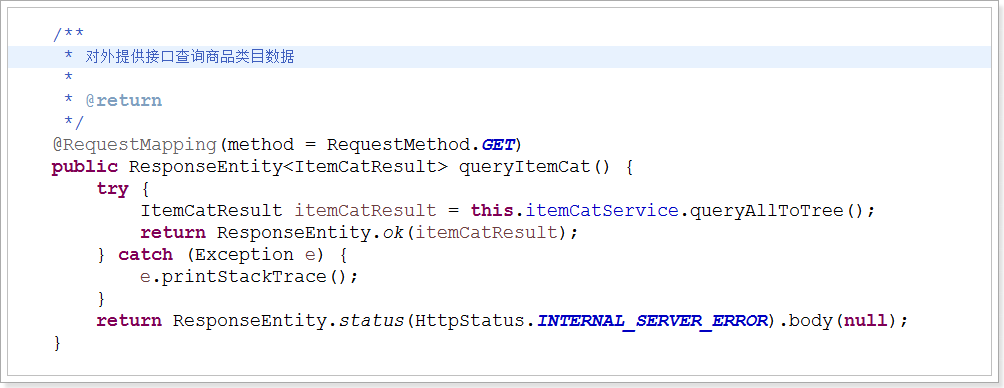
### 扩展CallbackMappingJackson2HttpMessageConverter



### 配置



### Controller实现

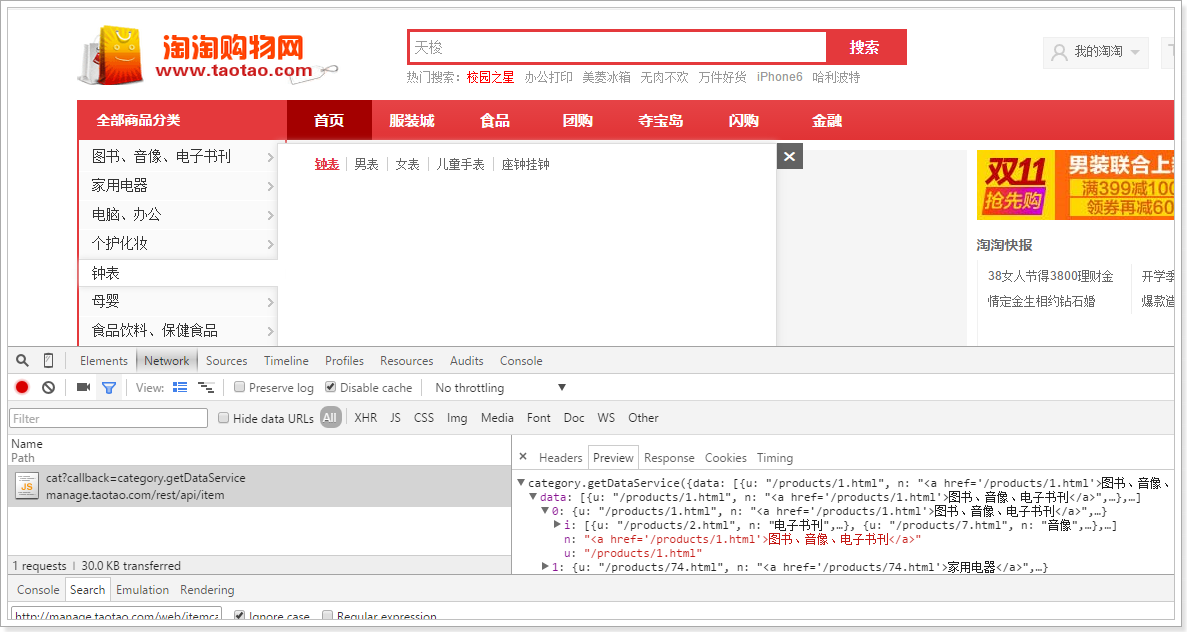


### 测试





效果：



至此，首页左侧商品类目的显示，完美实现。