# 淘淘商城-商品详情页展示

## 内容回顾

1. 搜索系统的实现
   1. 发布搜索服务，pc、移动端都可以调用。
   2. 在portal中调用服务，实现搜索功能。
2. Solr提供的搜索服务。
   1. Solr的集群
   2. Solr集群的使用。
   3. 单机版使用HttpSolrServer。
   4. 集群版使用CloudSolrServer。

## 课程计划

1. 商品详情页面的展示
   1. 伪静态化（使用jsp动态展示）
   2. 添加缓存，redis
2. 网页静态化freemaker+nginx

## 商品详情页面展示

### 分析

在搜索结果中点击商品图片或者title打开商品详情页面。



在Controller中从url中把商品id取出来，根据商品id查询对应的商品详细信息。

商品信息应该是调用服务来获得。需要在taotao-rest中发布服务。

Taotao-rest服务中根据商品id查询商品信息。

需要用到的表：

商品表：tb\_item

商品详细信息：tb\_item\_desc

商品的规格参数：tb\_item\_param\_item

应该发布三个服务：**商品基本信息、取商品详情、取商品的规格参数**

### 发布服务

#### 商品基本信息服务

根据商品id取数据。商品表：tb\_item

##### Dao层

使用逆向工程

##### Service层

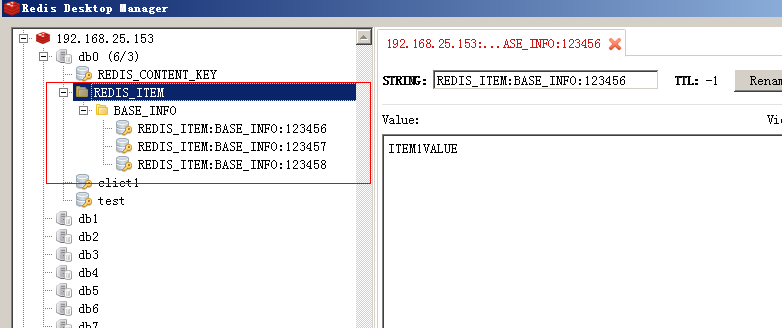
接收商品id，根据商品id查询商品基本信息。返回商品基本信息的pojo——TbItem。

为了减轻数据库的压力，需要添加缓存。由于商品数据比较多，需要把用户访问的数据添加缓存中，同时需要设置商品数据的过期时间，根据时间情况调整，需要把过期时间放到配置文件中。

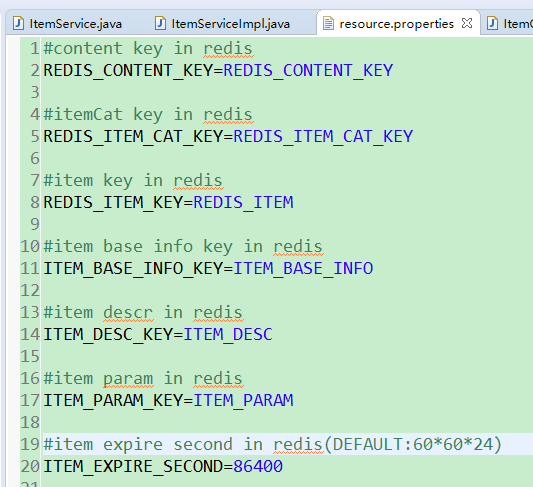
要设置过期时间redis中只能在key上设置。

使用String数据类型，为了便于分组可以使用“:”分隔的命名方式。（使用工具便于查看）

REDIS\_ITEM:BASE\_INFO:{ITEM\_ID}



|  |
| --- |
| **public** **interface** ItemService {  **public** TbItem getItemById(Long itemId);  }  ItemServiceImpl ：  @Service  **public** **class** ItemServiceImpl **implements** ItemService {  @Autowired  **private** TbItemMapper itemMapper;  @Autowired  **private** JedisClient jedisClient;  @Value("${REDIS\_ITEM\_KEY}")  **private** String REDIS\_ITEM\_KEY;  @Value("${ITEM\_BASE\_INFO\_KEY}")  **private** String ITEM\_BASE\_INFO\_KEY;  @Value("${ITEM\_EXPIRE\_SECOND}")  **private** Integer ITEM\_EXPIRE\_SECOND;  @Override  **public** TbItem getItemById(Long itemId) {  //查询缓存，如果有缓存，直接返回  **try** {  String json = jedisClient.get(REDIS\_ITEM\_KEY + ":" + ITEM\_BASE\_INFO\_KEY + ":" + itemId);  //判断数据是否存在  **if** (StringUtils.*isNotBlank*(json)) {  // 把json数据转换成java对象  TbItem item = JsonUtils.*jsonToPojo*(json, TbItem.**class**);  **return** item;  }  } **catch** (Exception e) {  e.printStackTrace();  }  //根据商品id查询商品基本信息  TbItem item = itemMapper.selectByPrimaryKey(itemId);  //向redis中添加缓存。  //添加缓存原则是不能影响正常的业务逻辑  **try** {  //向redis中添加缓存  jedisClient.set(REDIS\_ITEM\_KEY + ":" + ITEM\_BASE\_INFO\_KEY + ":" + itemId  , JsonUtils.*objectToJson*(item));  //设置key的过期时间  jedisClient.expire(REDIS\_ITEM\_KEY + ":" + ITEM\_BASE\_INFO\_KEY + ":" + itemId, ITEM\_EXPIRE\_SECOND);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** item;  }  } |



##### Controller

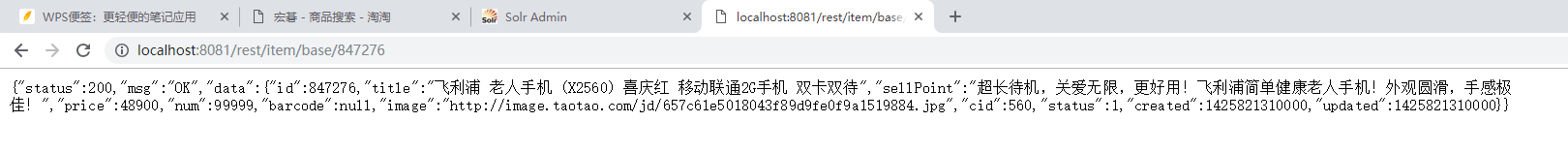
接收商品id，根据商品id查询商品基本信息，调用Service查询。直接响应json数据。

返回结果：使用TaotaoResult包装TbItem对象。

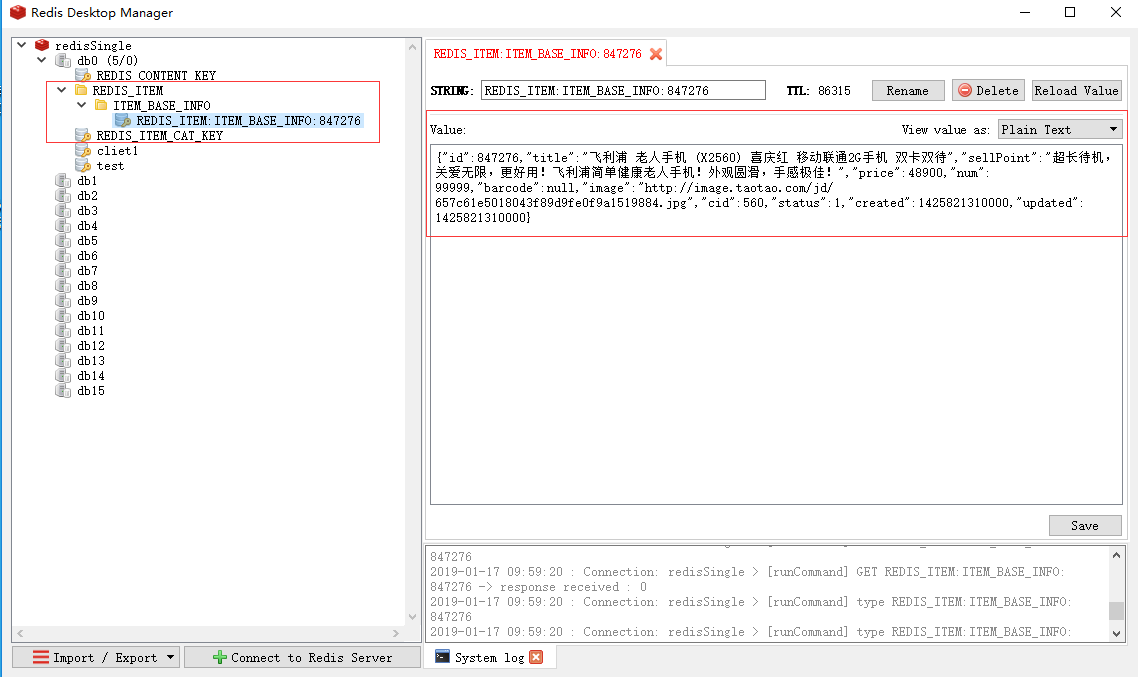
请求的url：/rest/item/base/{itemId}

|  |
| --- |
| @Controller  @RequestMapping("/item")  **public** **class** ItemController {  @Autowired  **private** ItemService itemService;  /\*\*  \* 查询商品基本信息  \* <p>Title: getItemById</p>  \* <p>Description: </p>  \* **@param** itemId  \* **@return**  \*/  @RequestMapping("/base/{itemId}")  @ResponseBody  **public** TaotaoResult getItemById(@PathVariable Long itemId) {  **try** {  TbItem item = itemService.getItemById(itemId);  **return** TaotaoResult.*ok*(item);  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

启动taotao-rest进行测试：<http://localhost:8081/rest/item/base/847276>



借助redis工具查看保存的数据信息：



#### 商品详情信息查询服务

根据商品id查询商品详情。tb\_item\_desc

##### Dao层

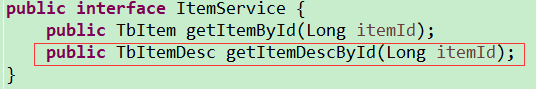
可以使用逆向工程

##### Service层

接收商品id，根据商品id查询商品详情，返回TbItemDesc对象。

需要添加缓存，需要设置过期时间。

此处调整redis名称命名，使之显示在一个商品id下显示相应的属性！



|  |
| --- |
| **package** com.taotao.rest.service.impl;  **import** org.apache.commons.lang3.StringUtils;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.utils.JsonUtils;  **import** com.taotao.mapper.TbItemDescMapper;  **import** com.taotao.mapper.TbItemMapper;  **import** com.taotao.pojo.TbItem;  **import** com.taotao.pojo.TbItemDesc;  **import** com.taotao.rest.component.JedisClient;  **import** com.taotao.rest.service.ItemService;  @Service  **public** **class** ItemServiceImpl **implements** ItemService {  @Autowired  **private** TbItemMapper itemMapper;  @Autowired  **private** TbItemDescMapper itemDescMapper;  @Autowired  **private** JedisClient jedisClient;  @Value("${REDIS\_ITEM\_KEY}")  **private** String REDIS\_ITEM\_KEY;  @Value("${ITEM\_BASE\_INFO\_KEY}")  **private** String ITEM\_BASE\_INFO\_KEY;  @Value("${ITEM\_EXPIRE\_SECOND}")  **private** Integer ITEM\_EXPIRE\_SECOND;  @Value("${ITEM\_DESC\_KEY}")  **private** String ITEM\_DESC\_KEY;  @Override  **public** TbItem getItemById(Long itemId) {  //查询缓存，如果有缓存，直接返回  **try** {  String json = jedisClient.get(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_BASE\_INFO\_KEY);  //判断数据是否存在  **if** (StringUtils.*isNotBlank*(json)) {  // 把json数据转换成java对象  TbItem item = JsonUtils.*jsonToPojo*(json, TbItem.**class**);  **return** item;  }  } **catch** (Exception e) {  e.printStackTrace();  }  //根据商品id查询商品基本信息  TbItem item = itemMapper.selectByPrimaryKey(itemId);  //向redis中添加缓存。  //添加缓存原则是不能影响正常的业务逻辑  **try** {  //向redis中添加缓存  jedisClient.set(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_BASE\_INFO\_KEY  , JsonUtils.*objectToJson*(item));  //设置key的过期时间  jedisClient.expire(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_BASE\_INFO\_KEY, ITEM\_EXPIRE\_SECOND);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** item;  }    @Override  **public** TbItemDesc getItemDescById(Long itemId) {  //查询缓存  //查询缓存，如果有缓存，直接返回  **try** {  String json = jedisClient.get(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_DESC\_KEY);  //判断数据是否存在  **if** (StringUtils.*isNotBlank*(json)) {  // 把json数据转换成java对象  TbItemDesc itemDesc = JsonUtils.*jsonToPojo*(json, TbItemDesc.**class**);  **return** itemDesc;  }  } **catch** (Exception e) {  e.printStackTrace();  }  //根据商品id查询商品详情  TbItemDesc itemDesc = itemDescMapper.selectByPrimaryKey(itemId);  //添加缓存  **try** {  //向redis中添加缓存  jedisClient.set(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_DESC\_KEY  , JsonUtils.*objectToJson*(itemDesc));  //设置key的过期时间  jedisClient.expire(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_DESC\_KEY, ITEM\_EXPIRE\_SECOND);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** itemDesc;  }  } |

##### Controller层

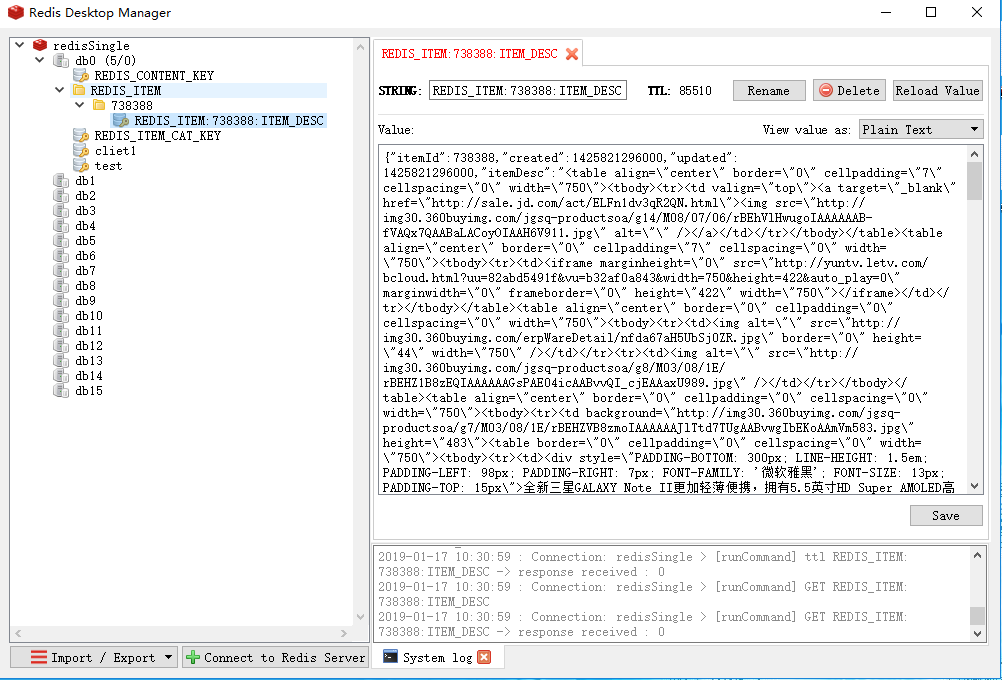
接收商品id，调用Service查询商品详情。返回json数据。使用TaotaoResult包装一个商品详情pojo返回。

请求的url：/rest/item/desc/{itemId}

|  |
| --- |
| @RequestMapping("/desc/{itemId}")  @ResponseBody  **public** TaotaoResult getItemDescById(@PathVariable Long itemId) {  **try** {  TbItemDesc itemDesc = itemService.getItemDescById(itemId);  **return** TaotaoResult.*ok*(itemDesc);  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  } |

重启taotao-rest测试：<http://localhost:8081/rest/item/desc/738388>





##### 问题分析：

报错原因，说是相关的参数没有找到，需要查看resource.properties下参数命名的格式、类型是否与定义的参数类型匹配，不匹配则无法查找，导致报错

#### 商品规格参数查询服务

根据商品id查询规格参数，响应的是json数据。同样需要添加缓存，设置缓存的过期时间。

tb\_item\_param\_item

##### Dao层

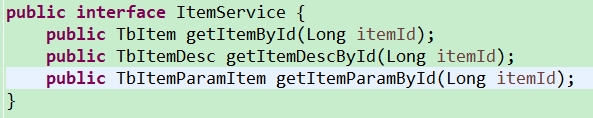
可以使用逆向工程

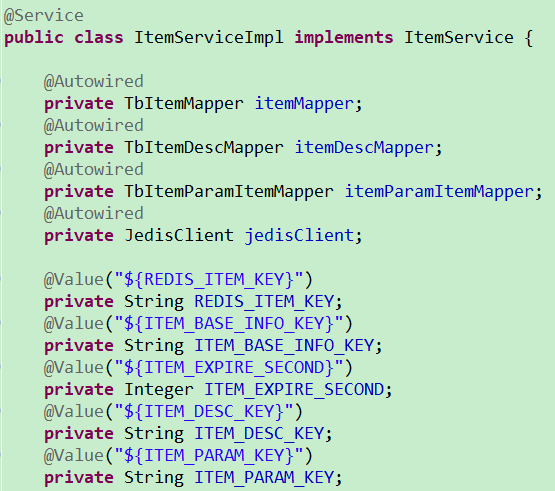
##### Service层

根据商品id查询规格参数，添加缓存。

参数：itemId

返回值：TbItemParamItem





|  |
| --- |
| @Override  **public** TbItemParamItem getItemParamById(Long itemId) {  //添加缓存逻辑  //查询缓存  //查询缓存，如果有缓存，直接返回  **try** {  String json = jedisClient.get(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_PARAM\_KEY);  //判断数据是否存在  **if** (StringUtils.*isNotBlank*(json)) {  // 把json数据转换成java对象  TbItemParamItem itemParamitem = JsonUtils.*jsonToPojo*(json, TbItemParamItem.**class**);  **return** itemParamitem;  }  } **catch** (Exception e) {  e.printStackTrace();  }  // 根据商品id查询规格参数  TbItemParamItemExample example = **new** TbItemParamItemExample();  Criteria criteria = example.createCriteria();  criteria.andItemIdEqualTo(itemId);  List<TbItemParamItem> list = itemParamItemMapper.selectByExampleWithBLOBs(example);  //取规格参数  **if** (list != **null** && list.size() > 0) {  TbItemParamItem itemParamItem = list.get(0);  //添加缓存  **try** {  //向redis中添加缓存  jedisClient.set(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_PARAM\_KEY  , JsonUtils.*objectToJson*(itemParamItem));  //设置key的过期时间  jedisClient.expire(REDIS\_ITEM\_KEY + ":" + itemId + ":" + ITEM\_PARAM\_KEY, ITEM\_EXPIRE\_SECOND);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** itemParamItem;  }  **return** **null**;  } |

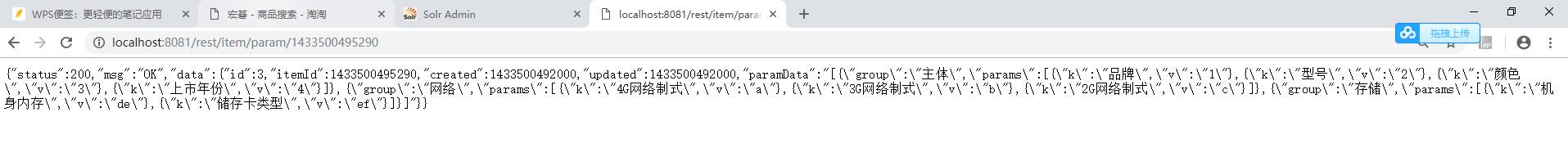
##### Controller层

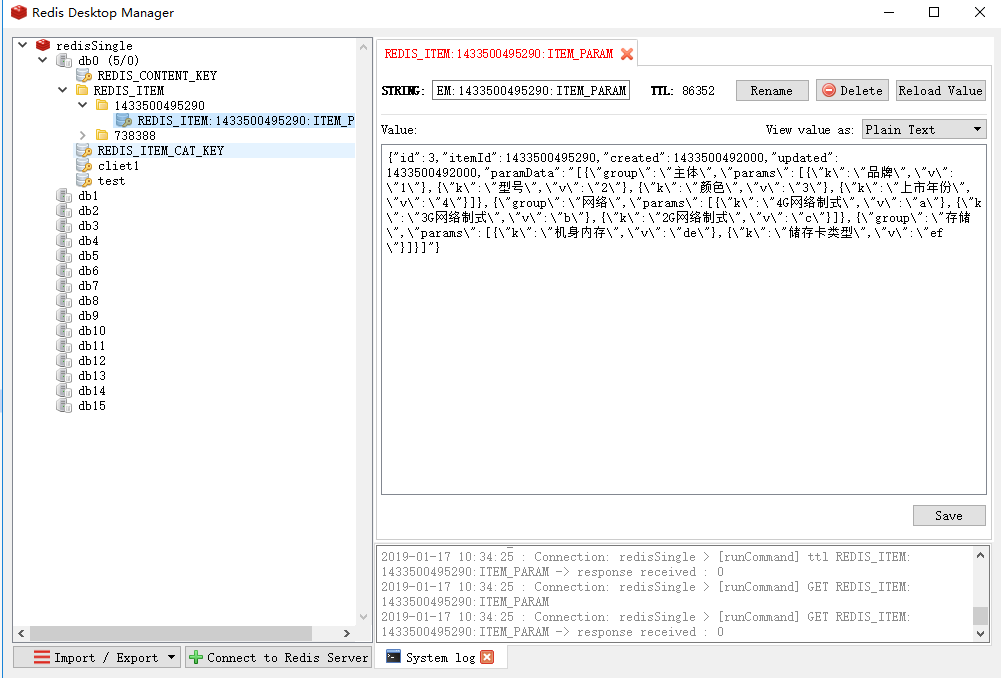
接收商品id，调用Service查询规格参数，返回json数据。

请求的url：/rest/item/param/{itemId}

|  |
| --- |
| @RequestMapping("/param/{itemId}")  @ResponseBody  **public** TaotaoResult getItemParamById(@PathVariable Long itemId) {  **try** {  TbItemParamItem itemParamItem = itemService.getItemParamById(itemId);  **return** TaotaoResult.*ok*(itemParamItem);  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  } |

重启taotao-rest测试：<http://localhost:8081/rest/item/param/1433500495290>





### 商品详情页面展示

#### 分析

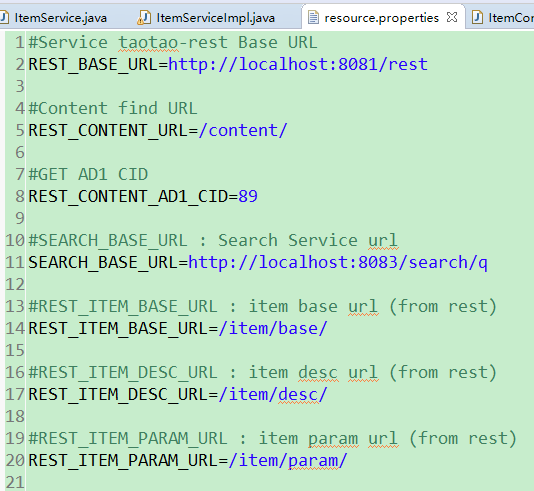
请求的url：/item/{itemId}.html

根据商品id查询商品的信息，需要调用taotao-rest发布的服务。

把获得商品数据传递给jsp。

1. 先取商品的基本信息展示给用户
2. 当页面加载完成后使用ajax调用服务查询商品的详情。延迟1秒加载。
3. 商品规格参数按需加载，当用户点击规格参数tab页时加载。

#### 展示商品基本信息



##### Service层

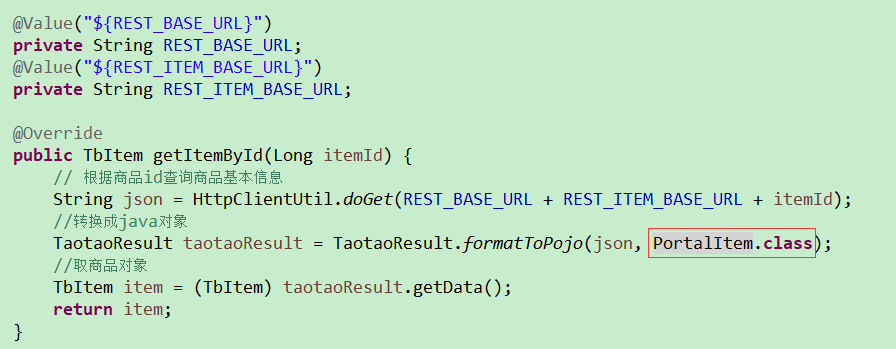
根据商品id调用服务获得商品基本信息数据。需要把数据转换成java对象。返回TbItem对象。

参数：itemId

返回值：TbItem

服务的url：服务url+/rest/item/base/{itemId}

|  |
| --- |
| **public** **interface** ItemService {  **public** TbItem getItemById(Long itemId);  }    @Service  **public** **class** ItemServiceImpl **implements** ItemService {  @Value("${REST\_BASE\_URL}")  **private** String REST\_BASE\_URL;  @Value("${REST\_ITEM\_BASE\_URL}")  **private** String REST\_ITEM\_BASE\_URL;  @Override  **public** TbItem getItemById(Long itemId) {  // 根据商品id查询商品基本信息  String json = HttpClientUtil.*doGet*(REST\_BASE\_URL + REST\_ITEM\_BASE\_URL + itemId);  //转换成java对象  TaotaoResult taotaoResult = TaotaoResult.*formatToPojo*(json, TbItem.**class**);  //取商品对象  TbItem item = (TbItem) taotaoResult.getData();  **return** item;  }  } |



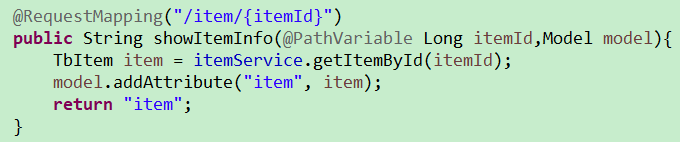
##### Controller层

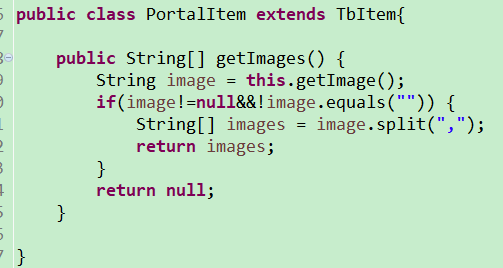
接收商品id，从url取。调用Service，把查询结果传递给jsp

请求的url：<http://localhost:8082/item/{itemId}.html>

把TbItem对象传递给jsp。Key就是“item”

以下访问出错，前端jsp请求的是images，需要封装一个新的





访问测试：<http://localhost:8082/item/154684233038024.html>



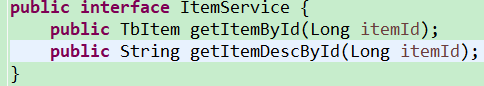
#### 展示商品描述

请求的url：/item/desc/{itemId}.html

响应的内容就是商品描述的html片段。

参数：商品ID

##### Service层



|  |
| --- |
| @Override  **public** String getItemDescById(Long itemId) {  //根据商品id调用taotao-rest的服务获得数据  //http://localhost:8081/rest/item/desc/144766336139977  String json = HttpClientUtil.*doGet*(REST\_BASE\_URL + REST\_ITEM\_DESC\_URL + itemId);  //转换成java对象  TaotaoResult taotaoResult = TaotaoResult.*formatToPojo*(json, TbItemDesc.**class**);  //取商品描述  TbItemDesc itemDesc = (TbItemDesc) taotaoResult.getData();  String desc = itemDesc.getItemDesc();  **return** desc;  } |

##### Controller层

接收商品id，从url中获得，调用Service获得商品描述数据，响应给浏览器。响应一个字符串。

参数：商品id

返回值：String（html片段）

|  |
| --- |
| @RequestMapping(value="/item/desc/{itemId}", produces=MediaType.***TEXT\_HTML\_VALUE***+";charset=utf-8")  @ResponseBody  **public** String getItemDesc(@PathVariable Long itemId) {  String desc = itemService.getItemDescById(itemId);  **return** desc;  } |

访问测试:http://localhost:8082/item/562379.html

此处选择数据库中存在的数据进行分析（选择descr内容存在的商品id进行测试）



#### 展示规格参数

当点击tab标签时，触发一个单击事件。Js做ajax请求，请求规格参数 的数据。把响应的内容展示到页面。

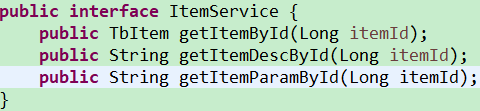
请求的url：/item/param/{itemId}.html

响应的内容：响应一个html片段。在java代码中生成。

参数：商品id。

##### Service

接收商品id，调用taotao-rest的服务，获得规格参数数据。转换成java对象，转换成html片段，返回。



|  |
| --- |
| @Override  **public** String getItemParamById(Long itemId) {  // 根据商品id获得对应的规格参数  String json = HttpClientUtil.*doGet*(REST\_BASE\_URL + REST\_ITEM\_PARAM\_URL + itemId);  // 转换成java对象  TaotaoResult taotaoResult = TaotaoResult.*formatToPojo*(json, TbItemParamItem.**class**);  // 取规格参数  TbItemParamItem itemParamItem = (TbItemParamItem) taotaoResult.getData();  String paramJson = itemParamItem.getParamData();  // 把规格参数的json数据转换成java对象  // 转换成java对象  List<Map> mapList = JsonUtils.*jsonToList*(paramJson, Map.**class**);  // 遍历list生成html  StringBuffer sb = **new** StringBuffer();  sb.append("<table cellpadding=\"0\" cellspacing=\"1\" width=\"100%\" border=\"0\" class=\"Ptable\">\n");  sb.append(" <tbody>\n");  **for** (Map map : mapList) {  sb.append(" <tr>\n");  sb.append(" <th class=\"tdTitle\" colspan=\"2\">" + map.get("group") + "</th>\n");  sb.append(" </tr>\n");  // 取规格项  List<Map> mapList2 = (List<Map>) map.get("params");  **for** (Map map2 : mapList2) {  sb.append(" <tr>\n");  sb.append(" <td class=\"tdTitle\">" + map2.get("k") + "</td>\n");  sb.append(" <td>" + map2.get("v") + "</td>\n");  sb.append(" </tr>\n");  }  }  sb.append(" </tbody>\n");  sb.append("</table>");  **return** sb.toString();  } |

##### Controller层

接收商品id，调用Service获得规格参数的html片段，直接响应。

参数:商品id

返回值：String

|  |
| --- |
| @RequestMapping(value="/item/param/{itemId}", produces=MediaType.***TEXT\_HTML\_VALUE***+";charset=utf-8")  @ResponseBody  **public** String getItemParam(@PathVariable Long itemId) {  String paramHtml = itemService.getItemParamById(itemId);  **return** paramHtml;  } |

访问测试 ：<http://localhost:8082/item/154684383340439.html>

选择存在的参数进行测试：



## 网页静态化

### 分析

每个商品要对应生成一个html文件，保存到磁盘，访问文件就不一定使用tomcat，可以使用nginx访问。

可以使用freemaker工具生成

### Freemarker

#### 什么是freemarker

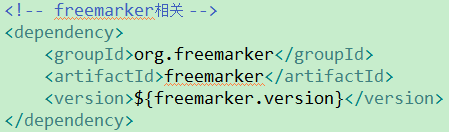
FreeMarker是一个用Java语言编写的模板引擎，它基于模板来生成文本输出。FreeMarker与Web容器无关，即在Web运行时，它并不知道Servlet或HTTP。它不仅可以用作表现层的实现技术，而且还可以用于生成XML，JSP或Java 等。

#### Freemarker的使用方法

##### 第一步：把freemarker的jar包添加到工程中

（此前taotao-parent如果没有该依赖引用，此处直接进行添加）





##### 第二步：freemarker的运行不依赖web容器，可以在java工程中运行。创建一个测试方法进行测试。

**package** com.taotao.freemarket;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.Writer;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.junit.Test;

**import** freemarker.template.Configuration;

**import** freemarker.template.Template;

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

@Test

**public** **void** testFreeMarker() **throws** Exception {

// 第一步：把freemarker的jar包添加到工程中

// 第二步：freemarker的运行不依赖web容器，可以在java工程中运行。创建一个测试方法进行测试。

// 第三步：创建一个Configuration对象

Configuration configuration = **new** Configuration(Configuration.*getVersion*());

// 第四步：告诉config对象模板文件存放的路径。

configuration.setDirectoryForTemplateLoading(**new** File("E:\\workspace\\eclipse\\Projects\\taotao-portal\\src\\main\\webapp\\WEB-INF\\ftl"));

// 第五步：设置config的默认字符集。一般是utf-8

configuration.setDefaultEncoding("utf-8");

// 第六步：从config对象中获得模板对象。需要制定一个模板文件的名字。

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

// 第七步：创建模板需要的数据集。可以是一个map对象也可以是一个pojo，把模板需要的数据都放入数据集。

Map root = **new** HashMap<>();

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

// 第八步：创建一个Writer对象，指定生成的文件保存的路径及文件名。

Writer out = **new** FileWriter(**new** File("E:\\temp\\html\\hello.html"));

// 第九步：调用模板对象的process方法生成静态文件。需要两个参数数据集和writer对象。

template.process(root, out);

// 第十步：关闭writer对象。

out.flush();

out.close();

}

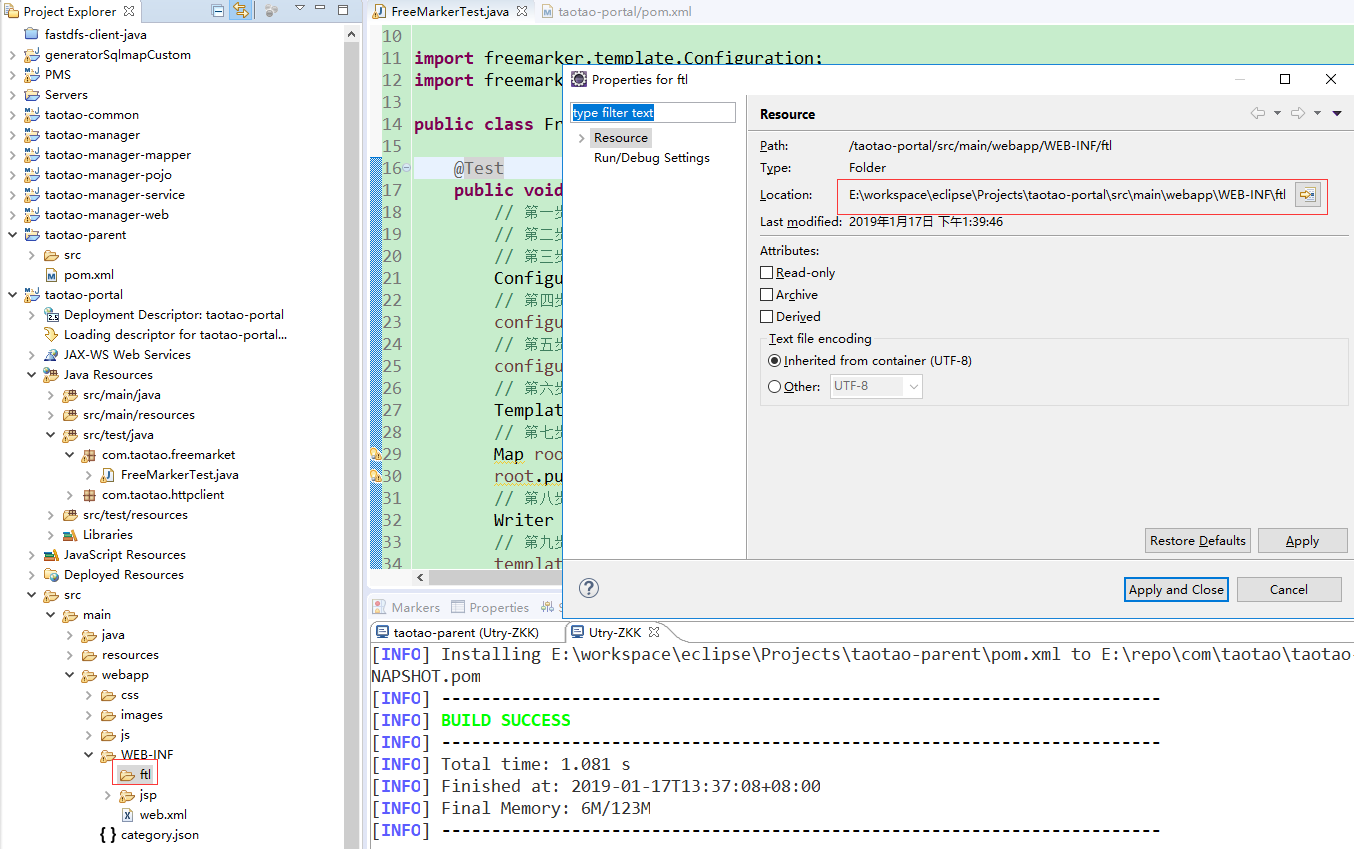
}

##### 第三步：创建一个Configration对象

##### 第四步：告诉config对象模板文件存放的路径。

在WEB-INF下创建ftl文件夹，右键点击ftl文件夹，随后选择properties，在mysql中选择查询，利用工具将/进行转义（选择含引号复制）：

"E:\\workspace\\eclipse\\Projects\\taotao-portal\\src\\main\\webapp\\WEB-INF\\ftl"

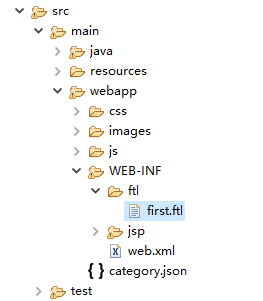


##### 第五步：设置config的默认字符集，一般是utf-8

##### 第六步：从config对象中获得模板对象。需要制定一个模板文件的名字。

在ftl文件上创建first.ftl文件，但由于eclipse不支持ftl格式的文件，依次此处需要加载相关的插件信息：<https://www.cnblogs.com/mophy/p/9299001.html>（在线导入，如果仅仅只是需要ide则通过在线导入即可）





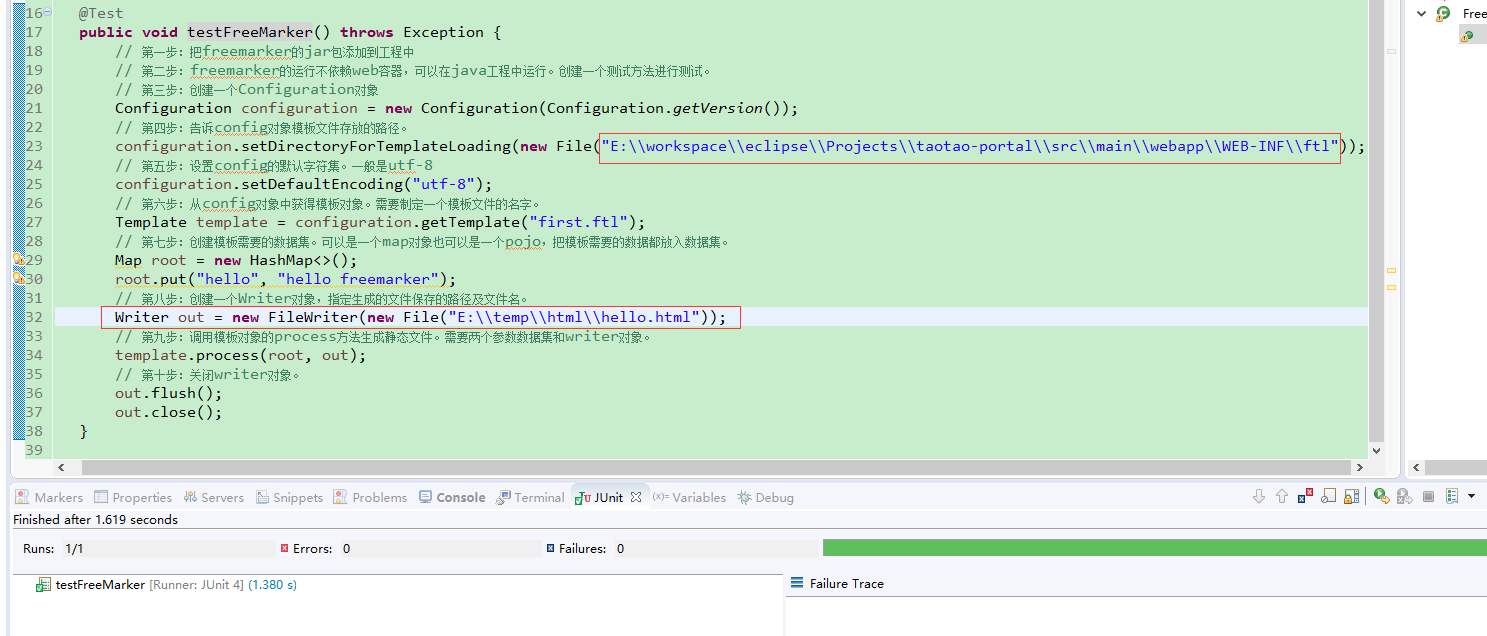
##### 第七步：创建模板需要的数据集。可以是一个map对象也可以是一个pojo，把模板需要的数据都放入数据集。

##### 第八步：创建一个Writer对象，指定生成的文件保存的路径及文件名（必须保证路径有效！）

##### 第九步：调用模板对象的process方法生成静态文件。需要两个参数数据集和writer对象。

##### 第十步：关闭writer对象。

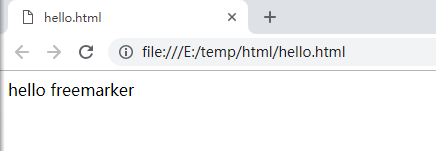
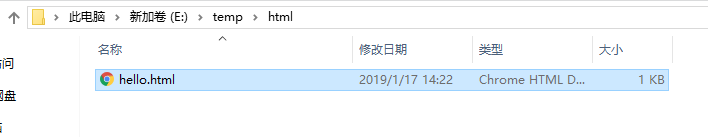
#### 代码实现



模板：

${hello}

测试结果：在对应路径下生成了一个html文件



#### Freemarker模板的写法

##### 取简单数据类型数据

使用EL表达式：${hello}

##### 包装数据类型

模板：（在WEB-INF下创建second.ftl进行测试）

|  |
| --- |
| <html>  <head>  <title>${title}</title>  </head>  <body>  <label>学号：</label>${student.id}<br>  <label>姓名：</label>${student.name}<br>  <label>住址：</label>${student.address}<br>  </body>  </html> |

测试结果：

**package** com.taotao.freemarket;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.Writer;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.junit.Test;

**import** freemarker.template.Configuration;

**import** freemarker.template.Template;

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

**public** **class** Student{

**private** **int** id;

**private** String name;

**private** String address;

**public** Student() {

}

**public** Student(**int** id, String name, String address) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.address = address;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

};

}

@Test

**public** **void** testFreeMarker2() **throws** Exception {

Configuration configuration = **new** Configuration(Configuration.*getVersion*());

configuration.setDirectoryForTemplateLoading(**new** File("E:\\workspace\\eclipse\\Projects\\taotao-portal\\src\\main\\webapp\\WEB-INF\\ftl"));

configuration.setDefaultEncoding("utf-8");

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

Map root = **new** HashMap<>();

root.put("title", "hello freemarker");

root.put("student", **new** Student(1,"hahabibu","杭州"));

Writer out = **new** FileWriter(**new** File("E:\\temp\\html\\second.html"));

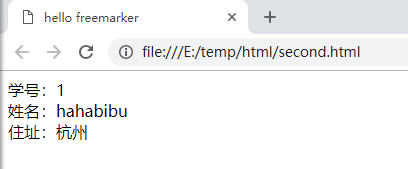
template.process(root, out);

out.flush();

out.close();

}

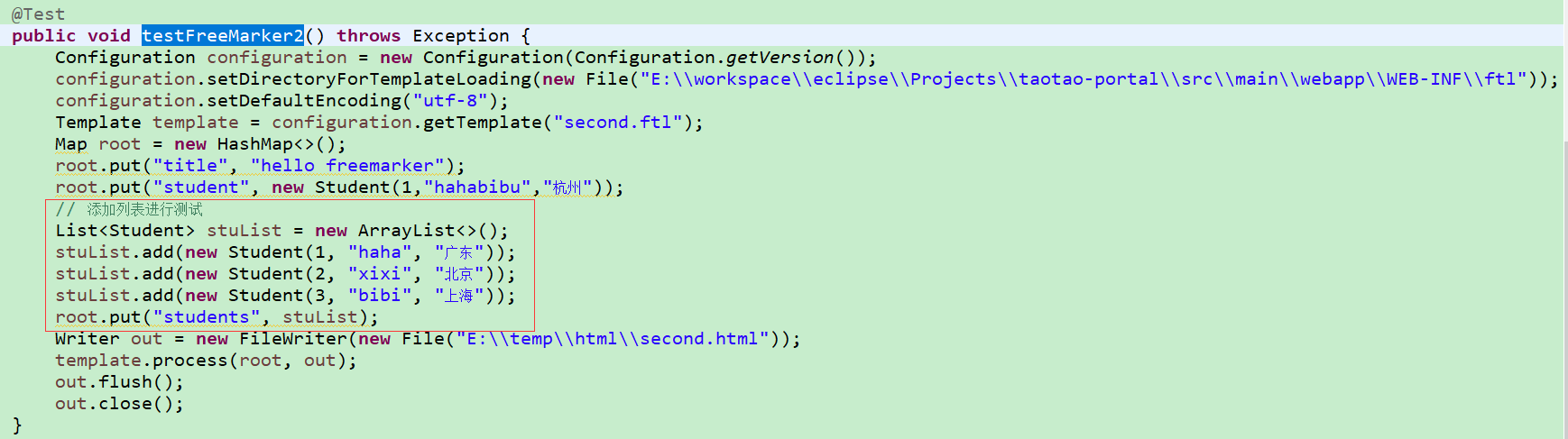
}



##### 历遍集合/数组

在上述基础上简单修改进行测试

List<Person> persons = new ArrayList<Person>();

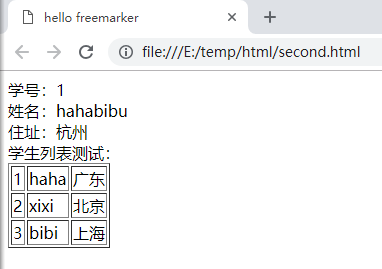


页面中内容

<#list persons as p>

${p.id}/${p.name}

</#list>



##### 获得当前迭代的索引

List<Person> list = new ArrayList<Person>();

获取当前选代的索引:<br/>

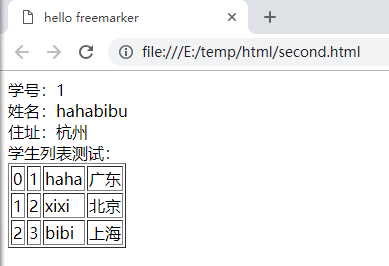
<#list persons as p>

${p\_index}

</#list>

测试方法不变，修改页面，添加索引数据显示：





##### 模板中判断条件

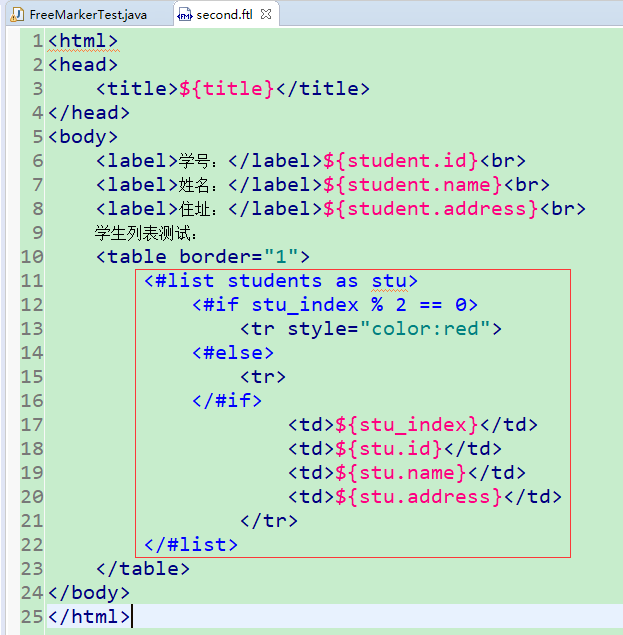
<#if 判断条件>

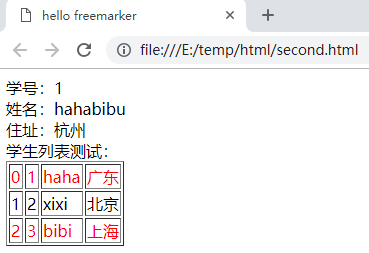
<#else>

</#if>

逻辑运算符（== != || &&）

测试方法不变，修改页面进行测试：





##### 日期类型格式化

默认格式

1：date

${cur\_time?date}

2：datetime

${cur\_time?datetime}

3：time

${cur\_time?time}

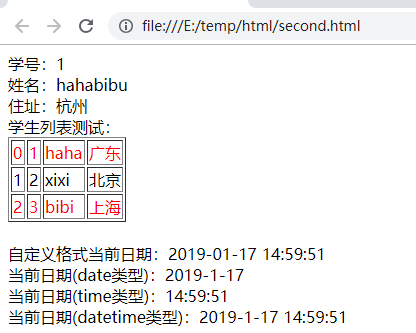
自定义格式

${cur\_time?string("yyyy-MM-dd HH:mm:ss")}

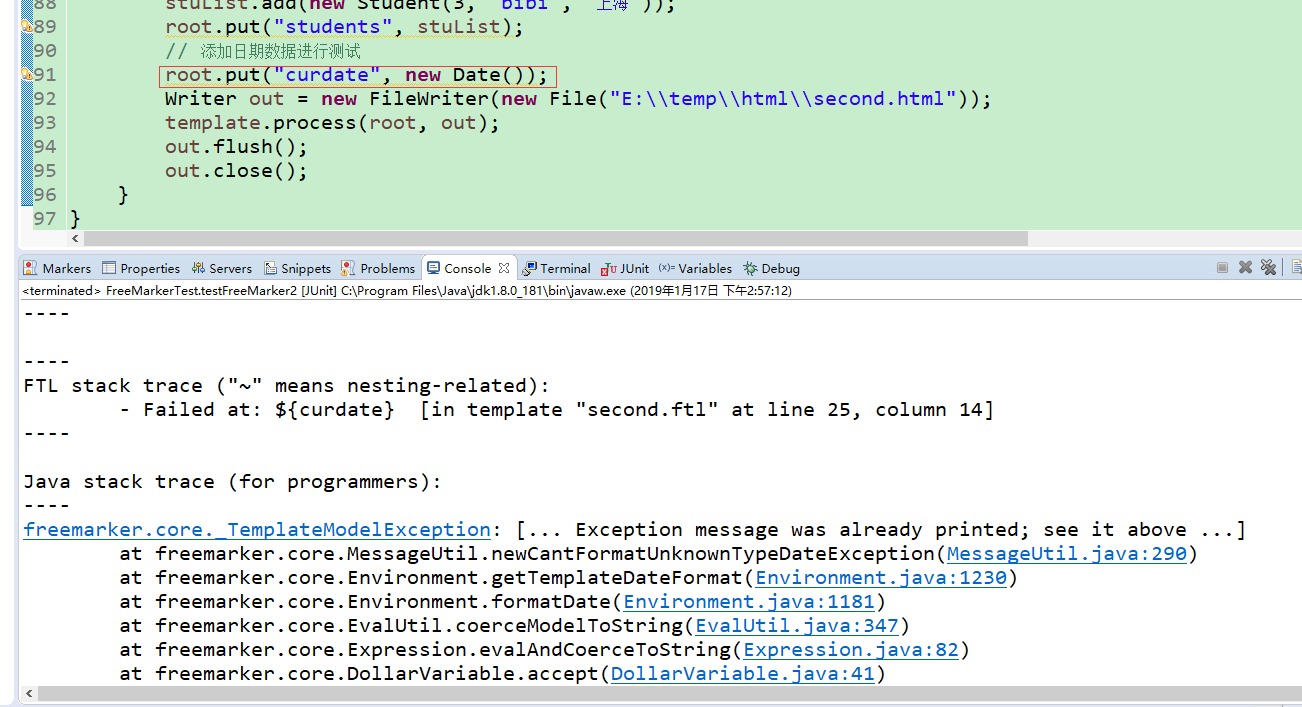
页面：



测试结果：

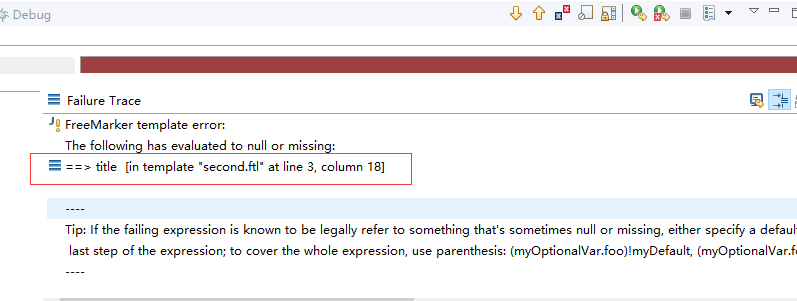


如果直接获取，没有指定格式会报错：



##### 处理null值

在jsp中如果没有传入相关数据，而页面中需要用到，即页面获取没有的数据也不会报错，但针对freemarker而言，如果页面没有获取需要的数据则相应的会报错，因此此处需要进行null值的处理



root.put(“val”,null);

解决办法

1:null 变 空串

${val!} ${val!"这里是空"}

2:为Null时给默认值

${val!“我是默认值"}

3、<#if curdate ??>

当前日期:${curdate?string("yyyy/MM/dd HH:mm:ss")}

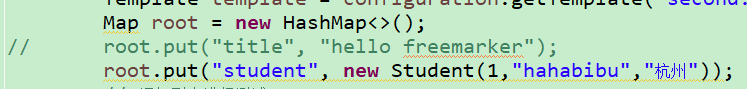
<#else>

curdate属性为null

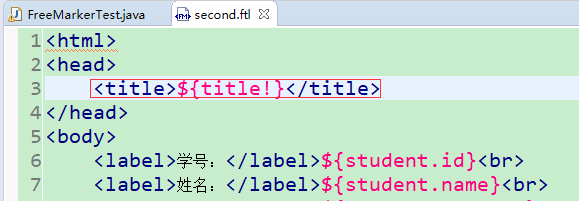
</#if>

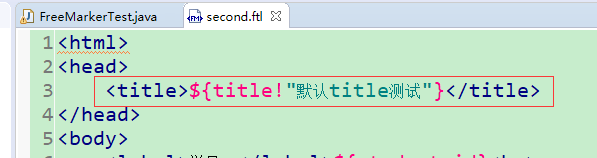
###### 普通数据格式化：

后台测试方法注释掉titile注入的内容：

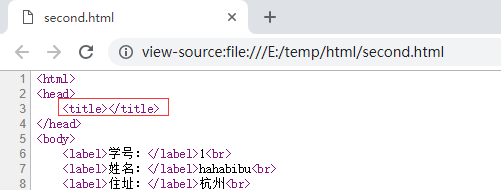


页面修改为，也可以指定默认值





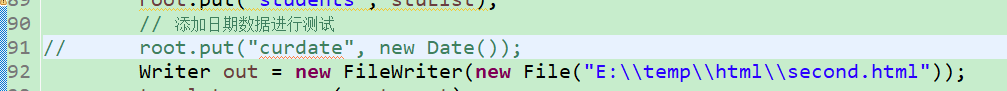
测试时查看源代码，如果没有指定默认值则title为空字符串，不会报错！



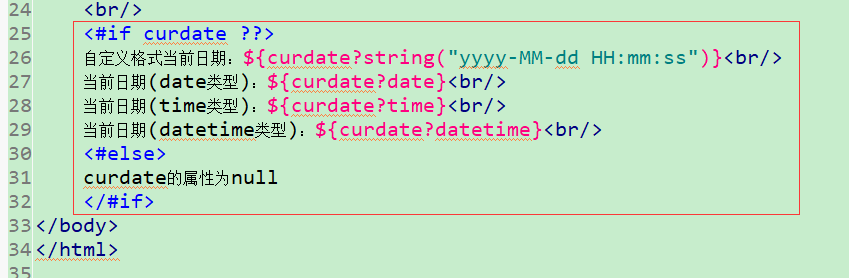


###### 日期数据格式化：

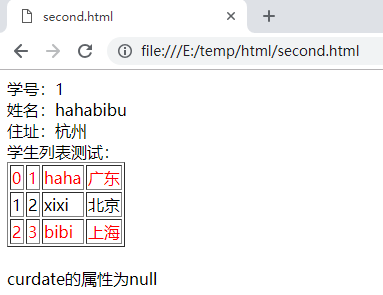
后台测试方法注释掉curdate的注入：



页面修改为如下：



测试结果：



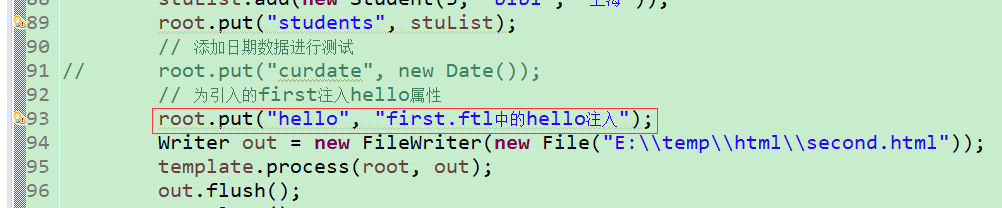
##### Include

将另一个页面引入本页面时可用以下命令完成

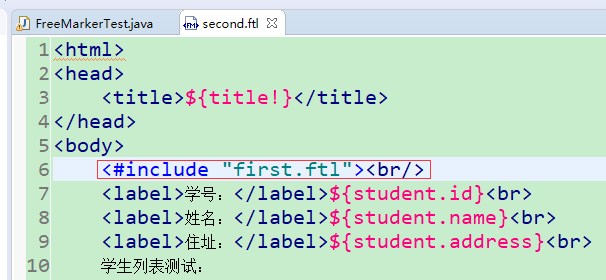
<#include "/include/head.html">

此处测试在second.ftl中引入first.ftl

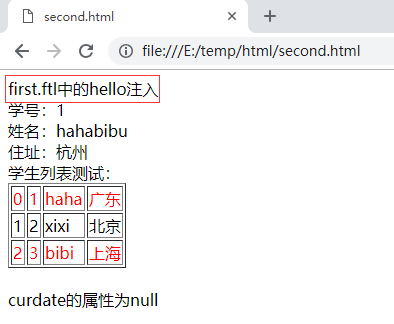
后台测试方法修改：



页面修改：



测试结果：



### 项目中使用freemarker

使用freemarker整合spring。把Configuration交给spring容器管理。

依赖的jar包：

<dependency>

<groupId>org.springframework</groupId>

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

<version>4.1.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.freemarker</groupId>

<artifactId>freemarker</artifactId>

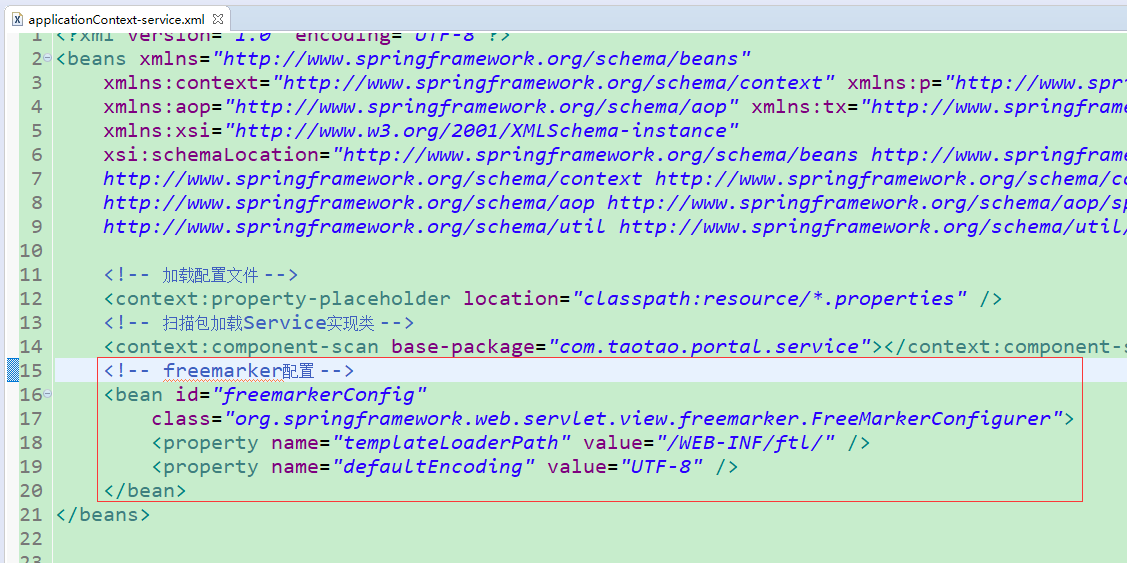
<version>2.3.23</version>

</dependency>

#### Spring配置文件

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

修改taotao-portal下的spring配置文件：



#### 静态文件生成的时机

1. 当用户第一次访问时生成静态文件。不推荐使用。
2. 当后台添加、编辑商品时生成静态网页。

此时，需要taotao-portal发布服务，供后台系统调用，当后台添加或者修改商品时调用此服务。此时taotao-portal的功能就是生成静态页面。

#### 发布生成页面的服务

1、获得数据的方式还是通过调用rest发布的服务获得数据。

2、生成静态页面。

3、响应生成成功。

#### Service

获得参数：商品id，根据商品id调用rest发布的服务，获得商品数据（商品基本信息、商品描述、商品规格参数），生成静态页面。返回成功。

参数：商品id

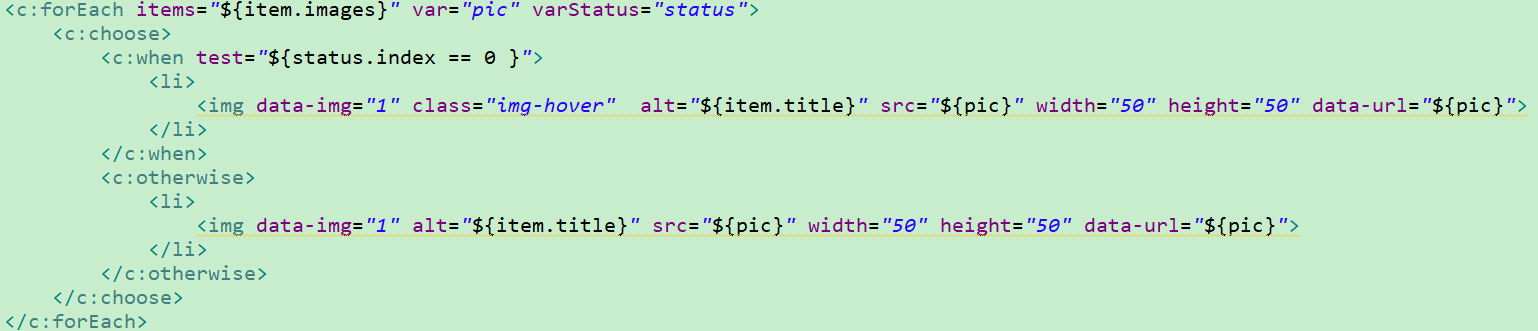
返回值：TaotaoResult

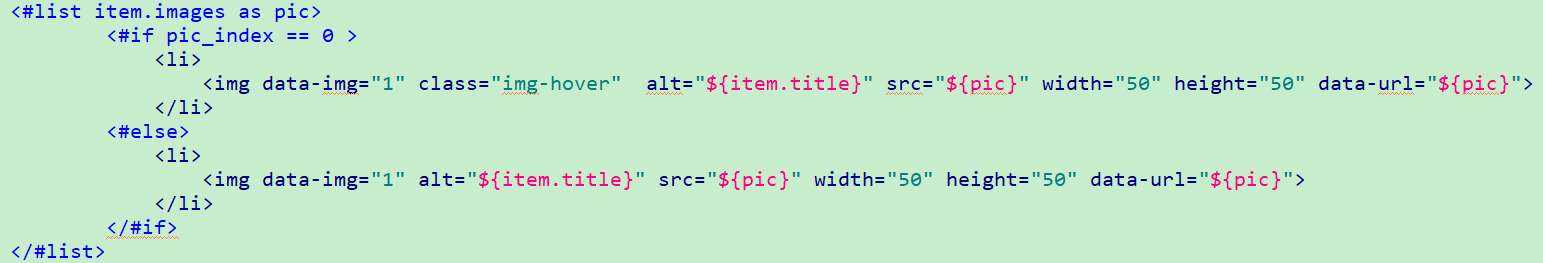
|  |
| --- |
| @Service  **public** **class** StaticPageServiceImpl **implements** StaticPageService {    @Autowired  **private** ItemService itemService;  @Autowired  **private** FreeMarkerConfigurer freeMarkerConfigurer;    @Value("${STATIC\_PAGE\_PATH}")  **private** String STATIC\_PAGE\_PATH;  @Override  **public** TaotaoResult genItemHtml(Long itemId) **throws** Exception {  //商品基本信息  TbItem tbItem = itemService.getItemById(itemId);  //商品描述  String itemDesc = itemService.getItemDescById(itemId);  //规格参数  String itemParam = itemService.getItemParamById(itemId);  //生成静态页面  Configuration configuration = freeMarkerConfigurer.getConfiguration();  Template template = configuration.getTemplate("item.ftl");  //创建一个数据集  Map root = **new** HashMap<>();  //向数据集中添加属性  root.put("item", tbItem);  root.put("itemDesc", itemDesc);  root.put("itemParam", itemParam);  //创建一个Writer对象  Writer out = **new** FileWriter(**new** File(STATIC\_PAGE\_PATH + itemId + ".html"));  //生成静态文件  template.process(root, out);  out.flush();  out.close();    **return** TaotaoResult.*ok*();  }      } |

#### Freemarker模板

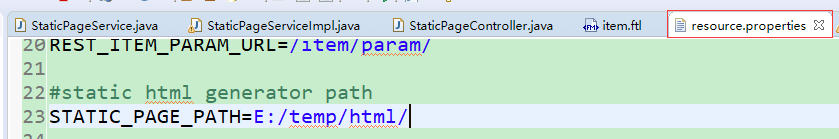
将文件夹WEB-INF/jsp相关的文件修改为ftl文件，ftl中要删除jsp特定的内容（只有在jsp文件中才有效的内容），其余内容均参考ftl定义标准进行定义

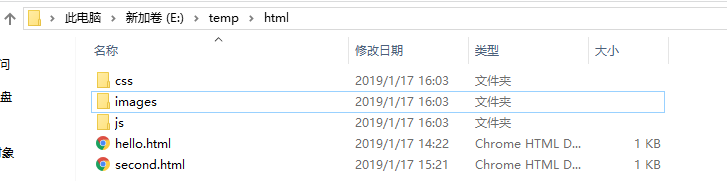
Forearch循环修改：

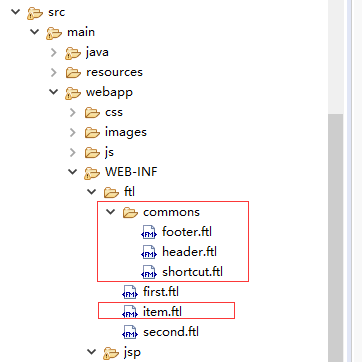




属性修改：（设置静态文件生成的路径，需要将相关的css、js、images复制到指定文件夹下供其访问）







#### Controller

接收商品id，调用Service生成静态页面。返回json数据。

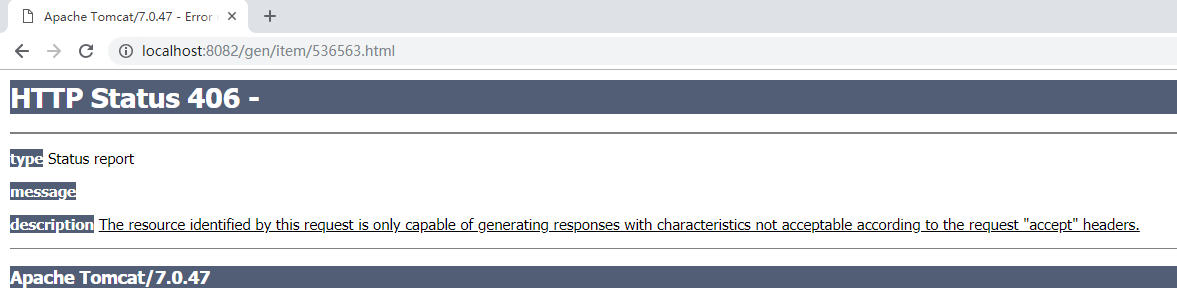
参数：商品id

返回值：TaotaoResult

|  |
| --- |
| @Controller  **public** **class** StaticPageController {  @Autowired  **private** StaticPageService staticPageService;  @RequestMapping("/gen/item/{itemId}")  @ResponseBody  **public** TaotaoResult genItemPage(@PathVariable Long itemId) {  **try** {  TaotaoResult result = staticPageService.genItemHtml(itemId);  **return** result;  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

#### 可能遇到的问题（406）：

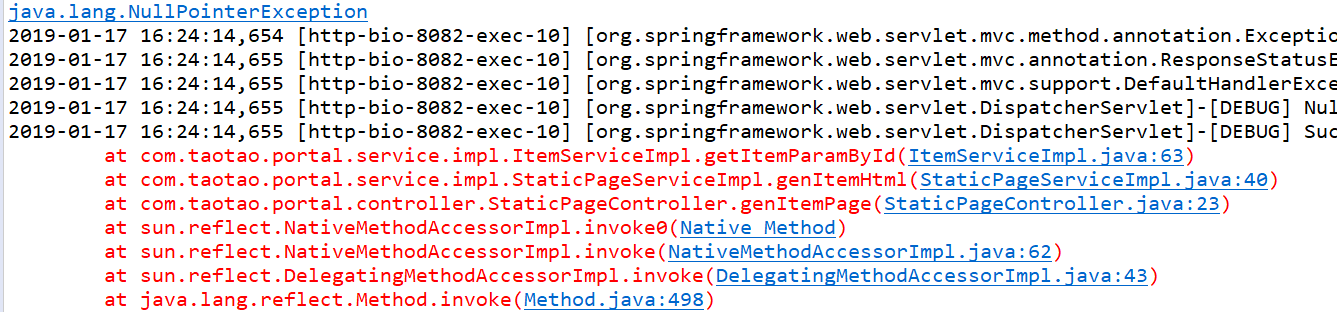
启动taotao-portal进行测试：



可能是没有导入jackon相关jar导致



需要启动taotao-rest、taotao-search相关的tomcat服务器



数据库不匹配导致，没有找到指定的参数数据集，导致空指针异常，此处选择自定义新增的数据进行参考，重新访问测试<http://localhost:8082/gen/item/154684383340439.html>

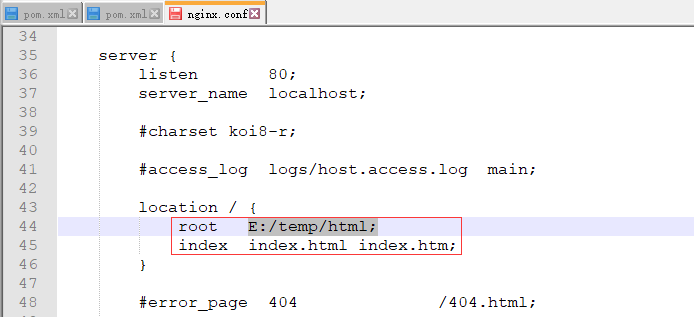


本地访问无法获取到css相关的内容，借助本地windows的nginx配置访问进行测试：

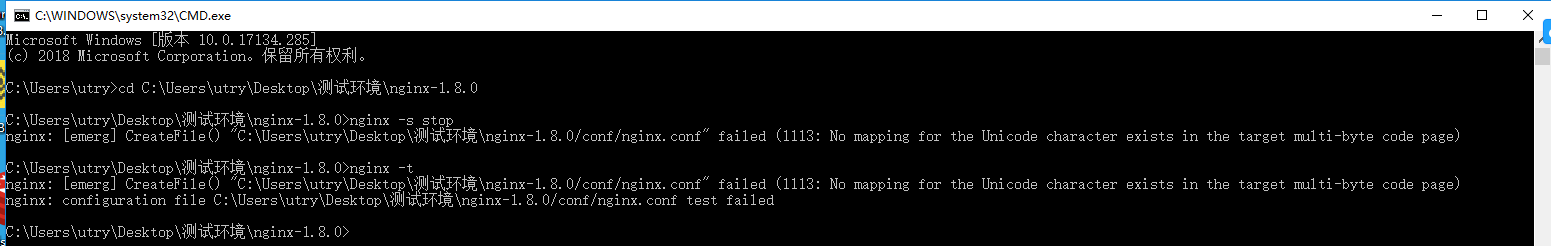


##### Windows版nginx环境安装

下载windows版的nginx包，解压后修改相关配置直接使用（修改路径为指定访问的路径）



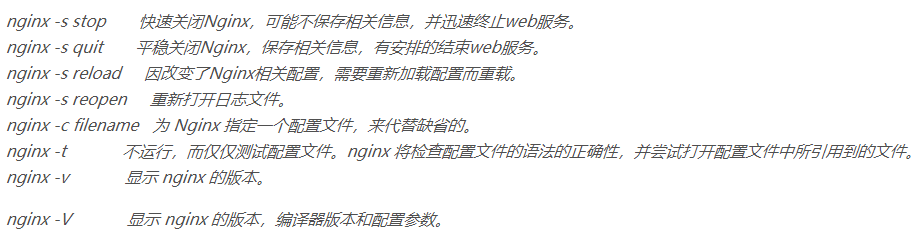
双击nginx.exe，启动本地的nginx进行测试：借助任务管理器查看本地nginx启动的状态

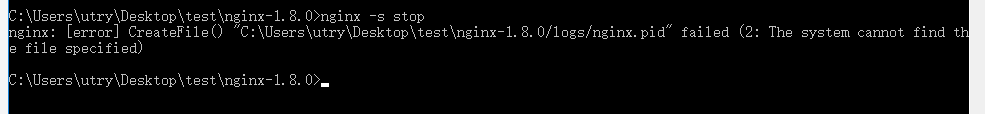


如果nginx无法启动，则可能是nginx配置文件配置出错，通过nginx -t测试配置文件，如果出现上述错误，则可能是nginx解压路径包含中文字符从而导致出错，修正后再次测试

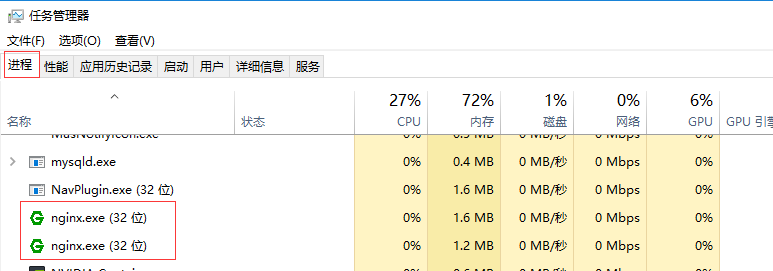
最好通过命令台的形式启动nginx，不要直接点击nginx.exe，可能会导致一些问题

常见的nginx命令说明：

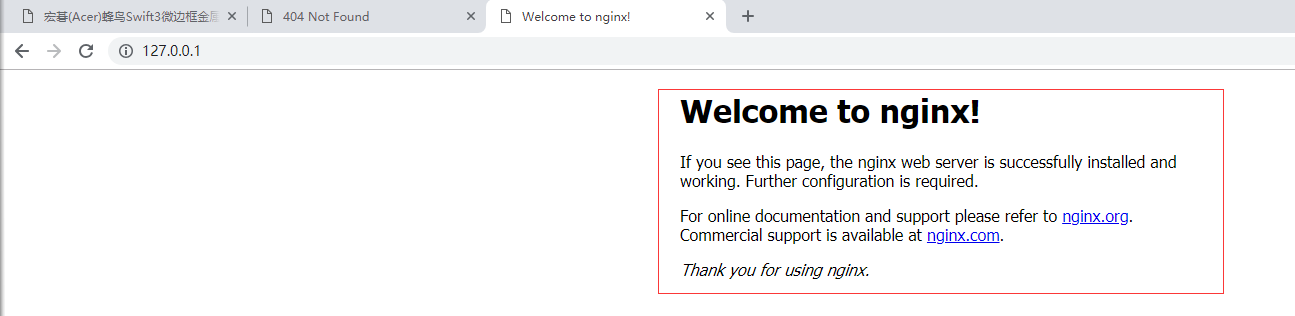




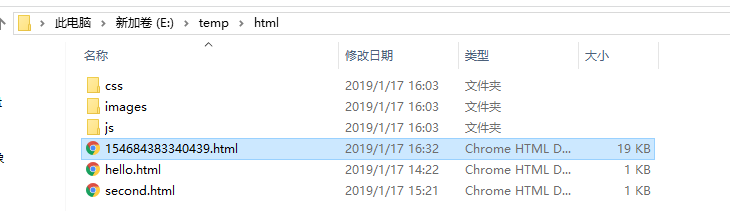
出现以下问题说明没有权限创建脚本文件，需要以管理员的方式运行cmd窗口随后再进行测试，服务正常启动。



或者通过浏览网页进行测试：127.0.0.1，如果指定了端口则需要加入指定端口

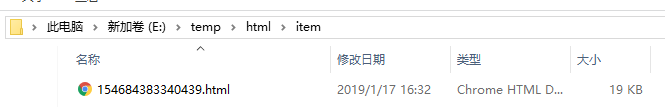


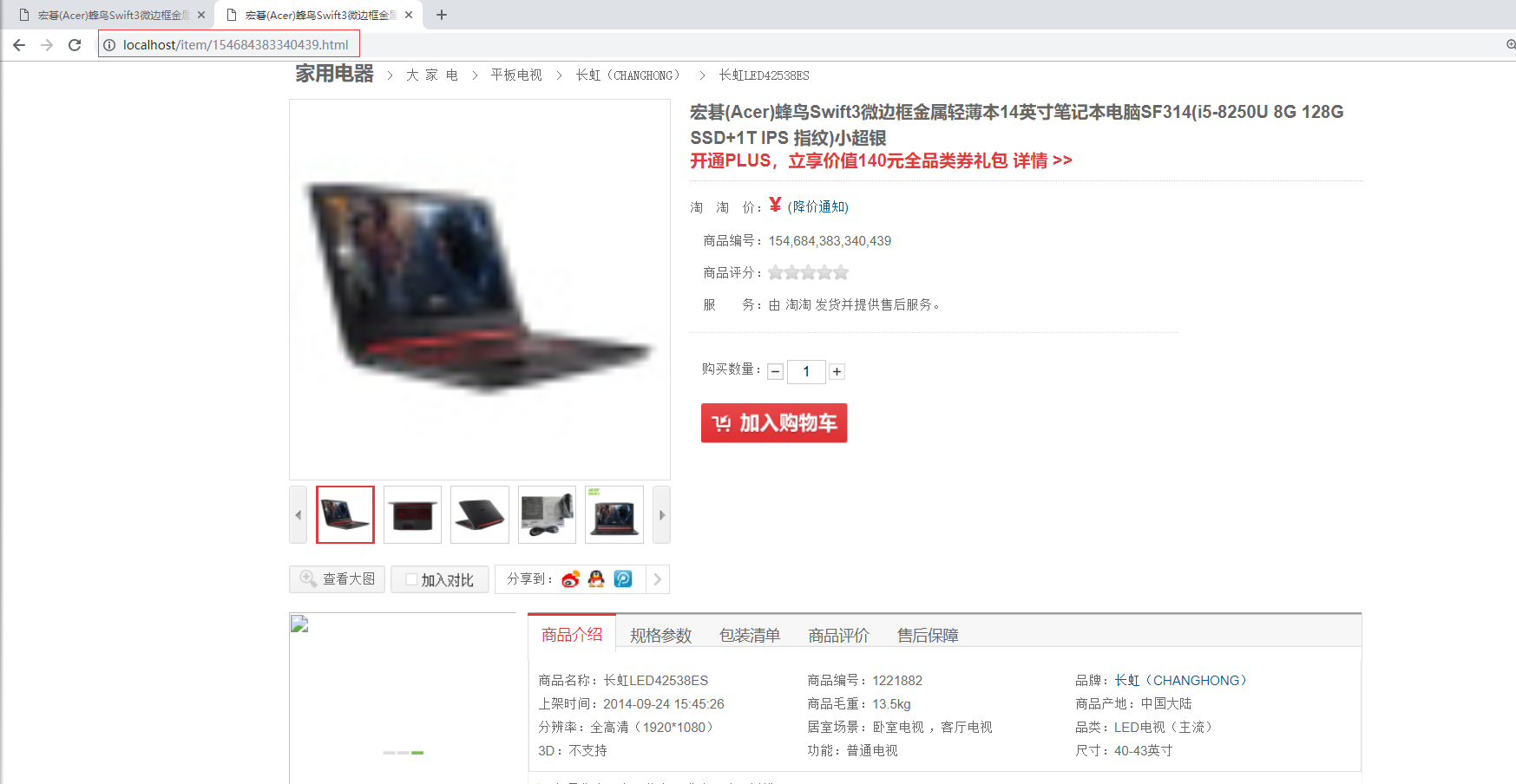
访问路径：http://localhost/154684383340439.html



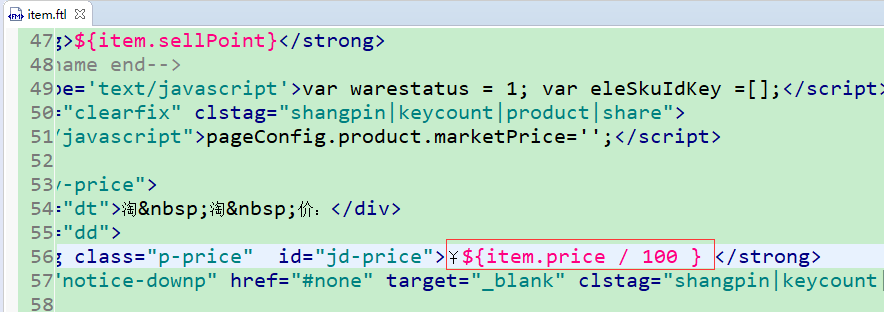


如果要指定目录级数，则添加一层目录即可：





**价格没有显示：格式化修改**



数据正常显示：



由于请求的url的后缀为.html导致406问题

解决方法：不要使用\*.html的后缀，修改web.xml文件如下所示



再次启动访问进行测试：<http://localhost:8082/gen/item/154684383340439.action>

