# 淘淘商城-商品规格参数保存说明

## 课程计划

内容复习：

1. 商品添加功能
   1. 商品类目的选择。easyUI异步tree的使用
   2. 图片上传
      1. 图片服务器。http服务、ftp服务
      2. 使用kindEditor的图片上传插件实现上传，返回结果是一个json形式的字符串。
   3. 富文本编辑器的使用
      1. 引用富文本编辑器的js
      2. Jsp中添加一个textarea控件
      3. 基于textarea控件创建富文本编辑器。调用kindEditor的create方法。
      4. 表单提交前，需要把富文本编辑器的内容和textarea同步，Sync方法。

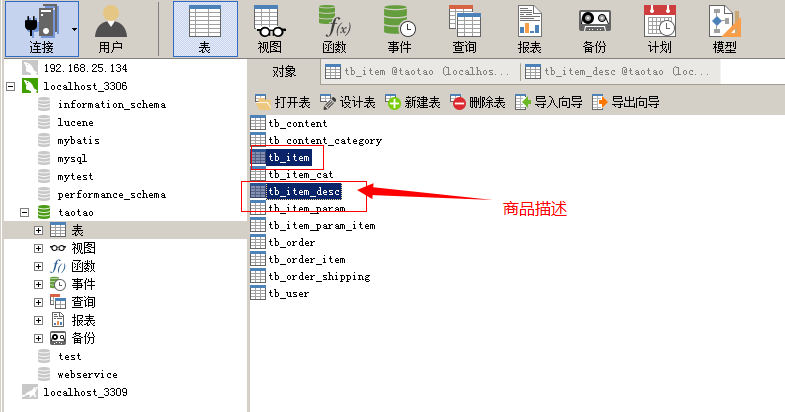
今天的内容：

1. 商品描述的保存
2. 商品规格的添加及使用
3. 使用多个表实现商品规格
4. 使用模板实现商品规格

## 商品描述的保存

后台要接收前台页面提交的商品信息，及商品描述。商品信息保存还要保存商品描述。

数据库中商品信息和商品描述是分开存储的。



### Dao层

把商品描述信息保存到tb\_item\_desc表中。可以使用逆向工程生成的代码

### Service层

接收商品描述调用dao把商品描述插入到表中。

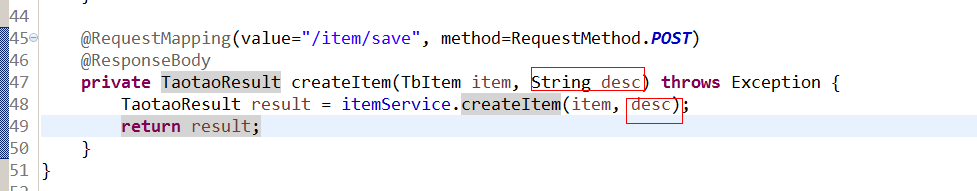
参数：String 商品描述

返回值：TaotaoResult

|  |
| --- |
| @Override  **public** TaotaoResult createItem(TbItem item, String desc) **throws** Exception {  //item补全：生成商品ID  Long itemId = IDUtils.*genItemId*();  item.setId(itemId);  // '商品状态，1-正常，2-下架，3-删除',  item.setStatus((**byte**) 1);  item.setCreated(**new** Date());  item.setUpdated(**new** Date());  //插入到数据库  itemMapper.insert(item);  //添加商品描述信息  TaotaoResult result = insertItemDesc(itemId, desc);  **if** (result.getStatus() != 200) {  **throw** **new** Exception();  }  **return** TaotaoResult.*ok*();  }  /\*\*  \* 添加商品描述  \* <p>Title: insertItemDesc</p>  \* <p>Description: </p>  \* **@param** desc  \*/  **private** TaotaoResult insertItemDesc(Long itemId, String desc) {  TbItemDesc itemDesc = **new** TbItemDesc();  itemDesc.setItemId(itemId);  itemDesc.setItemDesc(desc);  itemDesc.setCreated(**new** Date());  itemDesc.setUpdated(**new** Date());  itemDescMapper.insert(itemDesc);  **return** TaotaoResult.*ok*();  } |

### Controller

接收商品描述信息。



## 商品规格

### 什么是商品规格



规格参数：

规格组

|-规格项：规格值

规律：

1、同一类商品的规格项分组相同。

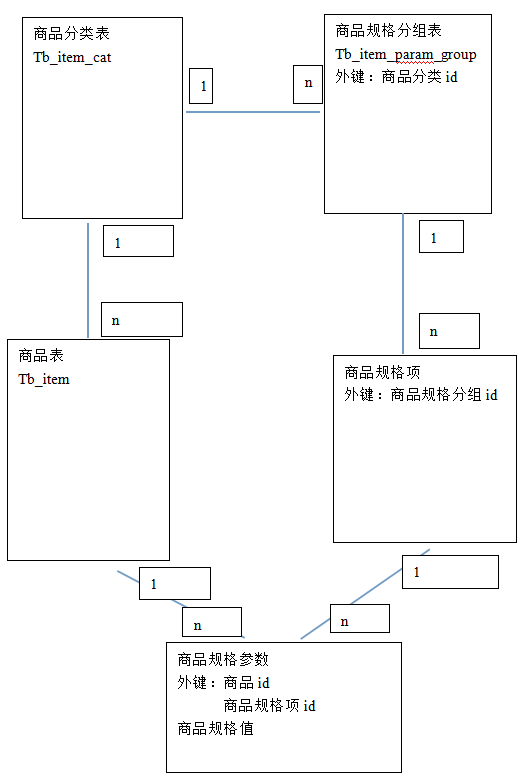
2、同一类商品的规格项目是相同的。规格项目是跟商品关联。

3、不同商品规格参数的值是不同的

### 实现方案

#### 方案一：使用多个表来存储

* 每一类商品有多个分组
* 每个分组下有多个项
* 每个商品对应不同的规格参数



使用二维表来维护规格数据

**表一：规格组信息**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **列名** | **类型** | **长度** | **可以null** | **键** | **说明** |
| Id | Int |  | 否 | P | 主键（自增长） |
| group\_name | varchar | 20 | 否 |  | 规格分组名称 |
| item\_cat\_id | Int |  | 否 | F | 商品分类id（外键） |

**表二：规格项信息**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **列名** | **类型** | **长度** | **可以null** | **键** | **说明** |
| Id | Int |  | 否 | P | 主键（自增长） |
| param\_name | varchar | 20 | 否 |  | 规格项目名称 |
| group\_id | Int |  | 否 | F | 规格分组id（外键） |

**表三：商品规格信息**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **列名** | **类型** | **长度** | **可以null** | **键** | **说明** |
| item\_id | Int |  | 否 | P | 商品id（联合主键） |
| param\_id | varchar |  | 否 | P | 规格项id（联合主键） |
| param\_value | varchar | 500 | 否 |  | 规格信息 |

##### Sql语句

|  |
| --- |
| SELECT  pg.group\_name,pk.param\_name,pv.param\_value  FROM  tb\_item\_param\_value pv  LEFT JOIN tb\_item\_param\_key pk ON pv.param\_id = pk.id  LEFT JOIN tb\_item\_param\_group pg ON pk.group\_id = pg.id  WHERE  item\_id = 855739 |

##### 方案一存在的问题：

* 需要创建多张表来描述规格参数之间的关系
* 查询时需要复杂的sql语句查询
* 规格参数数据量是商品信息的几十倍，数据量十分庞大。查询时效率很低
* 如果要求新添加的商品规格项发生改变，之前的商品不变是不能实现的

#### 方案二：使用模板进行管理

##### 方案分析

可以使用模板的思路来解决此问题，每一个商品分类对一个规格参数模板。

[

{

"group": "主体", //组名称

"params": [ // 记录规格成员

"品牌",

"型号",

"颜色",

"上市年份",

"上市月份"

]

}，

{

"group": "网络", //组名称

"params": [ // 记录规格成员

"4G",

"3G,

"2G"

]

}

]

使用模板

每个商品对应一唯一的规格参数。在添加商品时，可以根据规格参数的模板。生成一个表单。保存规格参数时。还可以生成规格参数的json数据。保存到数据库中。

[

{

"group": "主体",

"params": [

{

"k": "品牌",

"v": "苹果（Apple）"

},

{

"k": "型号",

"v": "iPhone 6 A1589"

},

{

"k": "智能机",

"v": "是 "

}

]

}

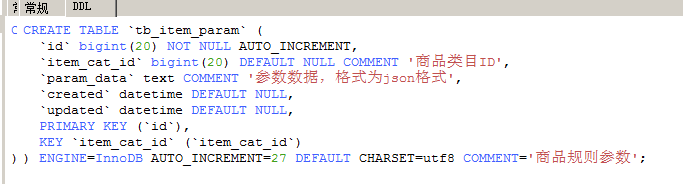
]

##### 实现流程

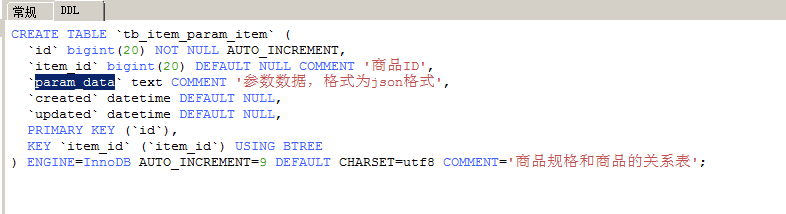


### 数据库存储

规格参数模板表：



商品的规格参数表：



优点：

1、不需要做多表管理。

2、如果要求新添加的商品规格项发生改变，之前的商品不变是很简单的。

缺点：

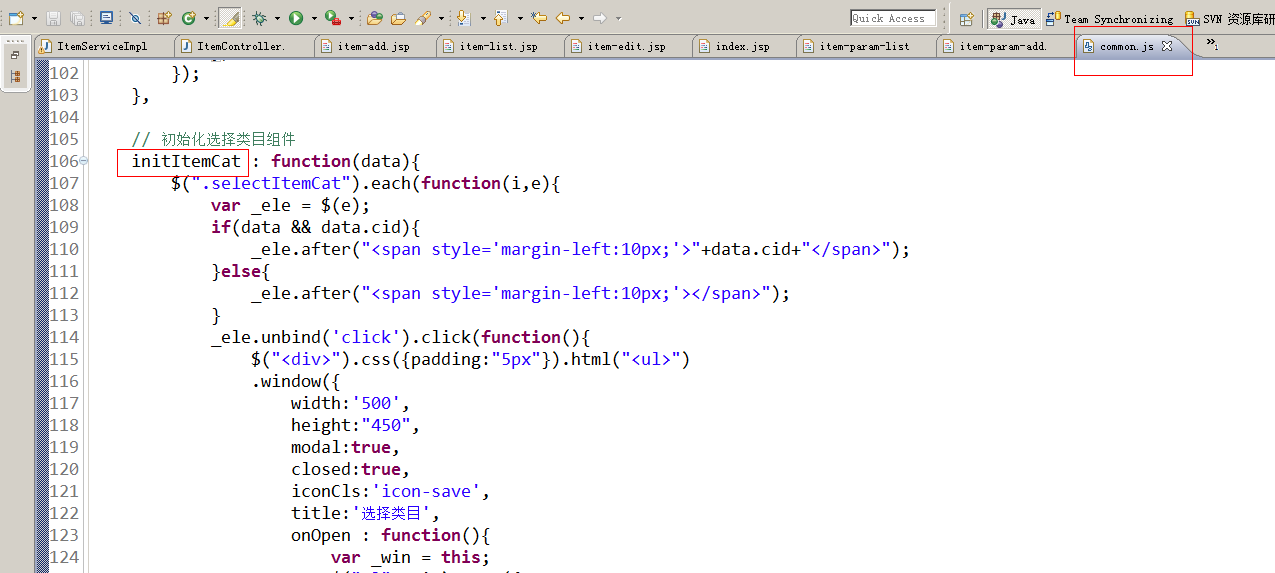
复杂的表单和json之间的转换。对js的编写要求很高。

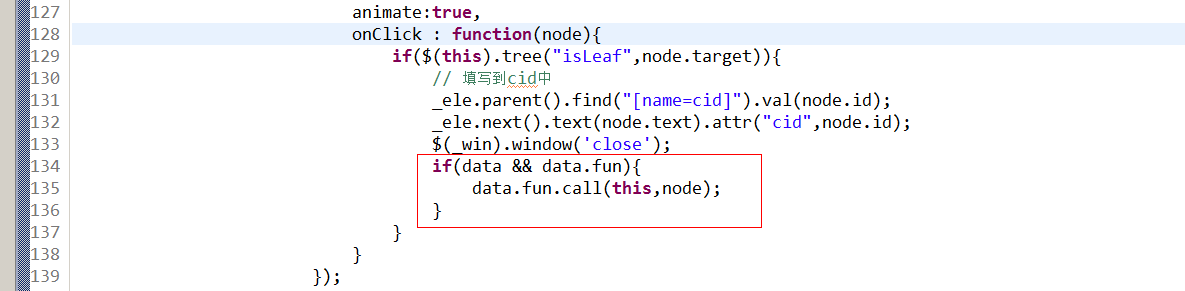
### 创建规格参数模板

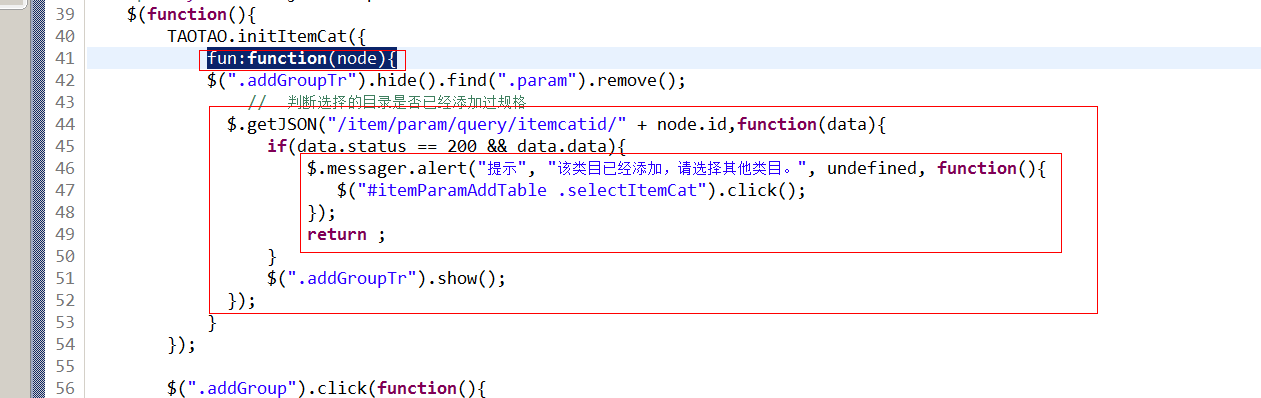
#### 选择商品分类

选择商品分类后根据选择的商品分类到tb\_item\_param规格参数模板表中取规格模板，取到了说明此商品分类的规格模板已经添加提示不能添加，如果没有取得正常添加。









##### 需求分析：判断模板是否存在

请求的url：

/item/param/query/itemcatid/{itemCatId}

参数：itemCatId，从url中获得

返回值：TaotaoResult

##### Dao层

从tb\_item\_param表中根据商品分类id查询内容。

单表操作，可以实现逆向工程的代码。

##### Service层

功能：接收商品分类id。调用mapper查询tb\_item\_param表，返回结果TaotaoResult。

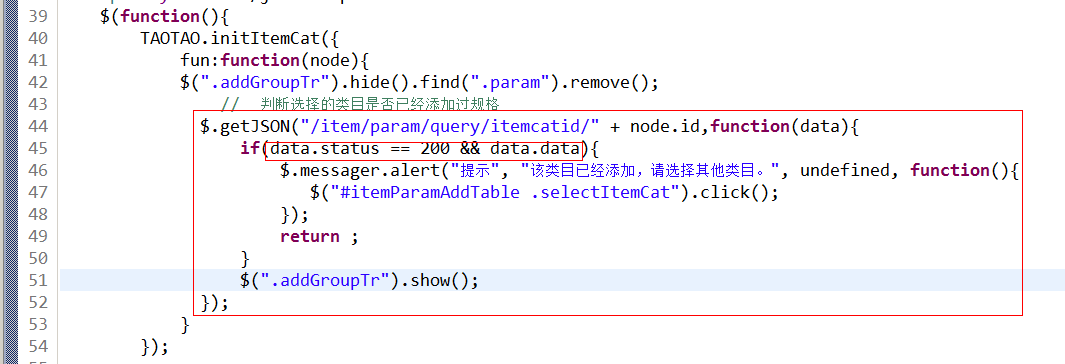
|  |
| --- |
| @Service  **public** **class** ItemParamServiceImpl **implements** ItemParamService {  @Autowired  **private** TbItemParamMapper itemParamMapper;    @Override  **public** TaotaoResult getItemParamByCid(**long** cid) {  TbItemParamExample example = **new** TbItemParamExample();  Criteria criteria = example.createCriteria();  criteria.andItemCatIdEqualTo(cid);  List<TbItemParam> list = itemParamMapper.selectByExample(example);  //判断是否查询到结果  **if** (list != **null** && list.size() > 0) {  **return** TaotaoResult.*ok*(list.get(0));  }    **return** TaotaoResult.*ok*();  }  } |

##### Controller

接收cid参数。调用Service查询规格参数模板。返回TaotaoResult。返回json数据。

|  |
| --- |
| @Controller  @RequestMapping("/item/param")  **public** **class** ItemParamController {  @Autowired  **private** ItemParamService itemParamService;  @RequestMapping("/query/itemcatid/{itemCatId}")  @ResponseBody  **public** TaotaoResult getItemParamByCid(@PathVariable Long itemCatId) {  TaotaoResult result = itemParamService.getItemParamByCid(itemCatId);  **return** result;  }  } |

##### Jsp



#### 提交规格参数模板

##### 需求分析：提交规格参数模板信息

请求的url：/item/param/save/{cid}

请求的参数：Long cid、String paramData

返回值：TaotaoResult

##### Dao层

向tb\_item\_param表中插入记录，使用逆向工程代码。

##### Service层

接收参数：cid、paramData两个参数，创建一个tb\_item\_param表对应的pojo对象。设置好对象中的属性。调用insert方法，添加一条记录。返回TaotaoResult。

|  |
| --- |
| @Override  **public** TaotaoResult insertItemParam(Long cid, String paramData) {  //创建一个pojo  TbItemParam itemParam = **new** TbItemParam();  itemParam.setItemCatId(cid);  itemParam.setParamData(paramData);  itemParam.setCreated(**new** Date());  itemParam.setUpdated(**new** Date());  //插入记录  itemParamMapper.insert(itemParam);  **return** TaotaoResult.*ok*();  } |

##### Controller层

接收两个参数，从url中取cid、从参数中取 paramData。调用Service插入记录，返回TaotaoResult（json），需要使用@ResourceBody

|  |
| --- |
| @RequestMapping("/save/{cid}")  @ResponseBody  **public** TaotaoResult insertItemParam(@PathVariable Long cid, String paramData) {  TaotaoResult result = itemParamService.insertItemParam(cid, paramData);  **return** result;  } |

#### 规格模板提交测试

##### 参数说明



##### 提交格式



#### 规格参数的编辑、删除：自主完成

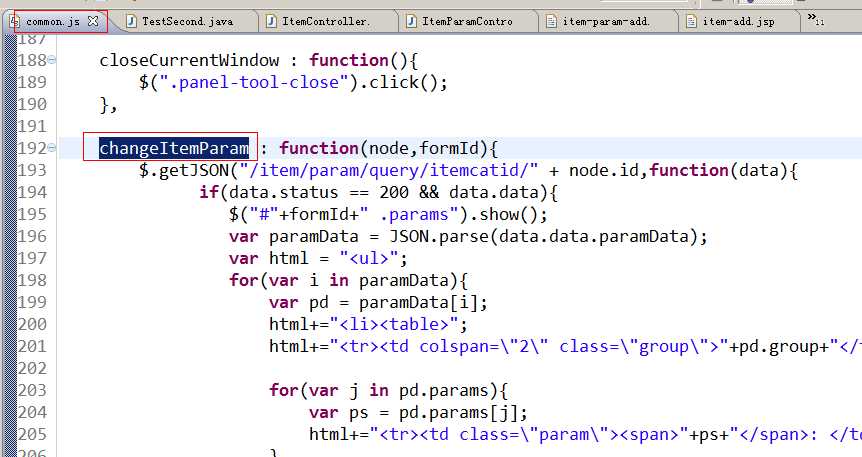
## 规格参数模板的使用

### 展示规格参数模板

应该在商品添加或者商品修改时，根据商品的分类id查询此商品分类对应的规格参数模板。根据规格参数模板，生成一个表单供用户使用。



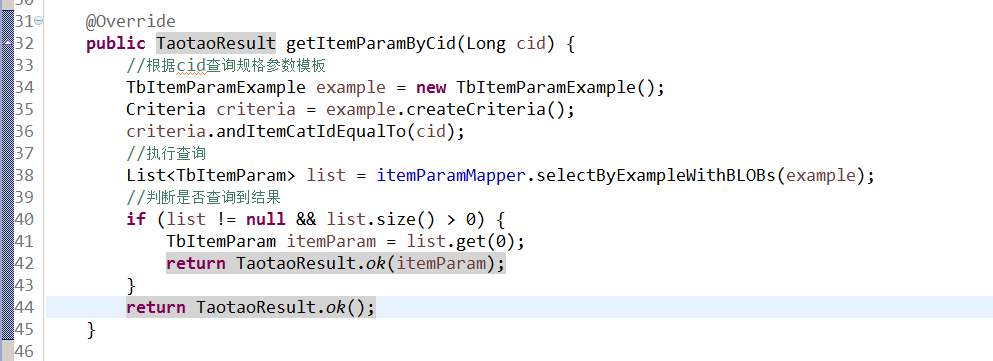
当商品类目选择完成后，调用此方法：



展示规格参数表单：

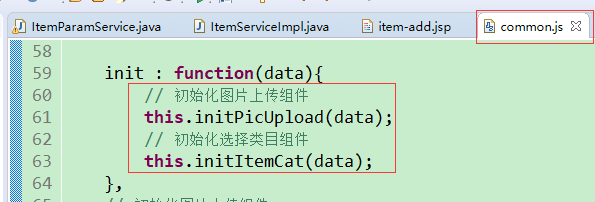
请求的url：/item/param/query/itemcatid/{cid}

可以和其他功能公用同一个方法。





在common.js中查看相关的信息

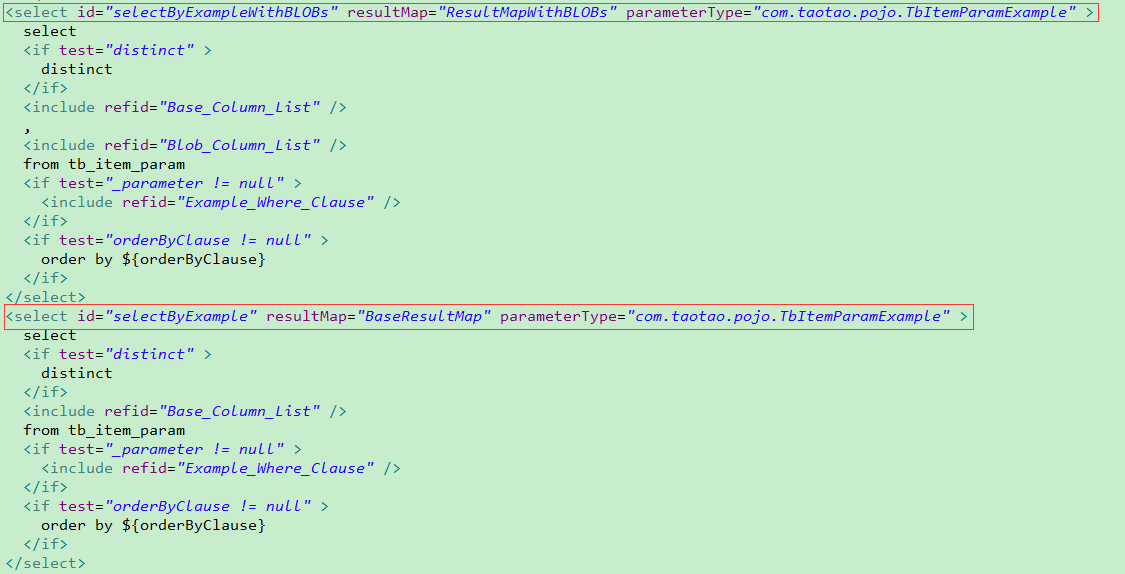


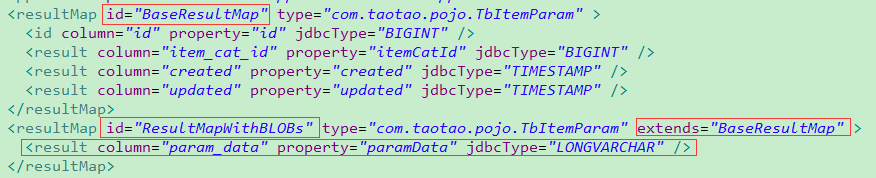


#### 可能存在问题分析：

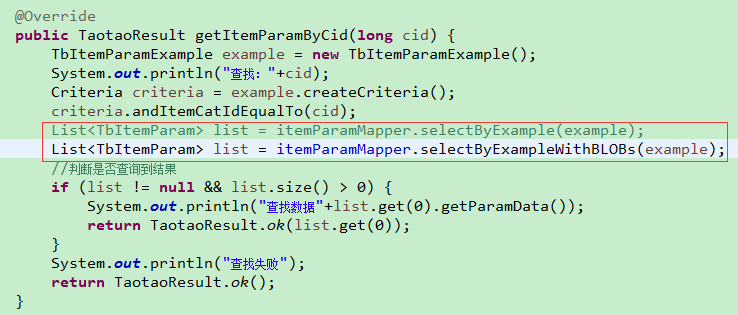
测试的时候发现，针对已有规格参数的商品分类，却没有显示其对应的参数模板，通过F12窗口查看相关信息，接口访问成功，但实际上接口的部分数据却没有封装到返回结果中，控制台输出查找也没有相关的数据，此处将service层查找做了如下调整，此时能够正常查找数据，原因分析:使用mybatis逆向工程生成的dao层代码，在单表查询的时候没有自信关注每个接口的实际应用，此处出了问题一步步回退查找出错点，发现是使用的接口方法并没有将大字段的paramData检索出来，从而导致数据没有封装，页面获取和显示的数据paramData为null，此处需要参考“MyBatis中selectByExample与selectByExampleWithBLOBs”代码区别

* ResultMapWithBLOBs 定义时，继承了BaseResultMap，并且自己特殊的字段，该字段通常是longvarchar类型，本例中content就为特殊字段。
* content字段类型为text。故如需检索的字段中包含大字段类型时，必须用selectByExampleWithBLOBs，不检索大字段时，用selectByExample就足够了。update同样如此。

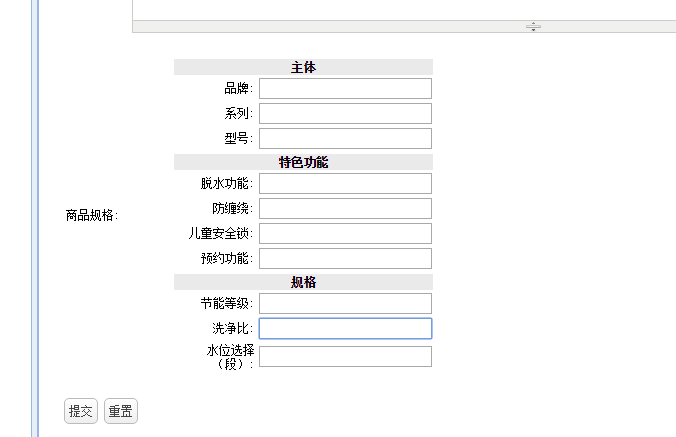




ResultMapWithBLOBs映射类型是在继承BaseResultMap的基础上对应配置另一个字段paramData，因此如果仅仅只是通过selectByExample查找，其返回的数据类型是BaseResultMap，不会检索大字段paramData，如此设计也是为了减轻数据访问压力！



测试结果：



### 规格参数的提交

#### 需求分析

在商品表单提交之前，先把规格参数的信息，转换成json数据。把json提交到后台插入到表中即可。



Json数据的格式：

1、生成的规格模板数据格式

[

{

"group": "主体", //组名称

"params": [ // 记录规格成员

"品牌",

"型号",

"颜色",

"上市年份",

"上市月份"

]

}，

{

"group": "网络", //组名称

"params": [ // 记录规格成员

"4G",

"3G,

"2G"

]

}

]

2、生成的规格数据格式

[

{

"group": "主体",

"params": [

{

"k": "品牌",

"v": "苹果（Apple）"

},

{

"k": "型号",

"v": "iPhone 6 A1589"

},

{

"k": "智能机",

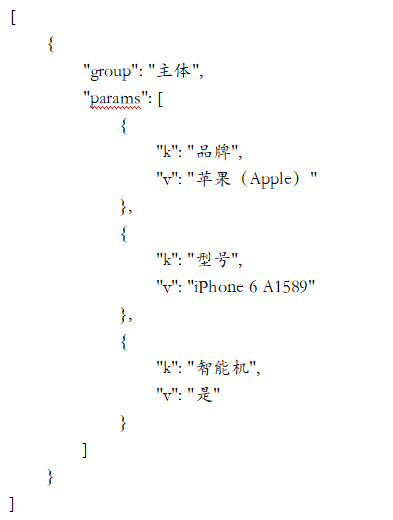
"v": "是"

}

]

}

]



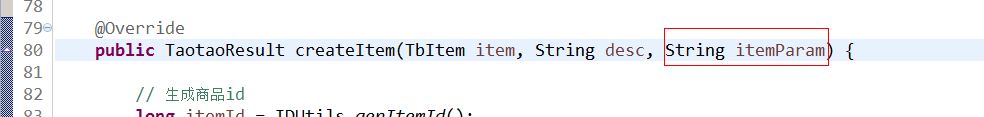
生成规格参数json字符串的处理：

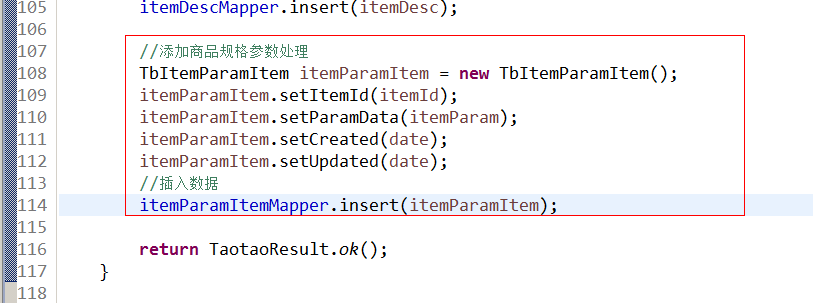


只需要在Controller中添加一个参数，接收商品的规格参数即可。

Service中也需要添加一个参数，增加插入规格参数表的处理。

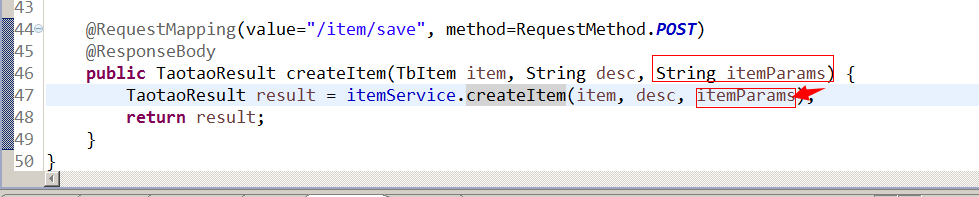
#### Service





#### Controller

添加一个接收商品规格参数的形参，完成规格参数的传入



## 展示规格参数

### 分析

从表中把规格参数json数据取处理，可以在java代码中解析json数据生成html展示到jsp页面。

根据商品id查询规格参数。

请求的url：/item/{itemId}

返回结果：String（逻辑视图）

### Dao层

单表查询，可以使用逆向工程

### Service

接收商品id，根据商品id取规格参数。可以把json转换成java对象。遍历java对象，生成html，返回。

参数：商品id

返回值：字符串（html片段）

|  |
| --- |
| @Override  **public** String getItemParamHtml(Long itemId) {  // 根据商品id查询规格参数  TbItemParamItemExample example = **new** TbItemParamItemExample();  com.taotao.pojo.TbItemParamItemExample.Criteria criteria = example.createCriteria();  criteria.andItemIdEqualTo(itemId);  // 执行查询  // List<TbItemParamItem> list = itemParamItemMapper.selectByExample(example);  List<TbItemParamItem> list = itemParamItemMapper.selectByExampleWithBLOBs(example);  **if** (list == **null** || list.isEmpty()) {  **return** "";  }  // 取规格参数  TbItemParamItem itemParamItem = list.get(0);  // 获取json数据  String paramData = itemParamItem.getParamData();  // 将json数据转换成java对象  List<Map> mapList = JsonUtils.*jsonToList*(paramData, Map.**class**);  // 遍历list生成html  StringBuffer sb = **new** StringBuffer();  sb.append("<table cellpadding=\"0\" cellspacing=\"1\" width=\"100%\" border=\"1\" 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();  } |

模板生成说明：

在某个网站中选择模板，右键选择“检查”查看源代码，提取相关属性，封装数据

字符转义技巧：可以通过mysql中查询编辑器完成转义，选中复制的代码，在编辑器中点击右键，选择“含引号复制”-->“JAVA/C#”，完成转义

### Controller层

接收商品id，调用Service查询规格参数，返回html片段。把html片段传递给jsp。

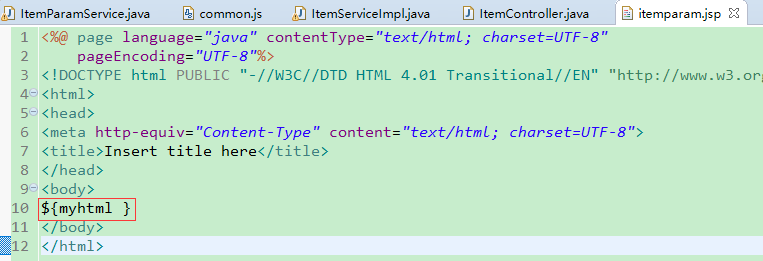
参数：

1、商品id

2、Model

|  |
| --- |
| @RequestMapping("/page/item/{itemId}")  **public** String showItemParam(@PathVariable Long itemId, Model model) {  String html = itemService.getItemParamHtml(itemId);  model.addAttribute("myhtml", html);  **return**"itemparam";  } |

创建jsp测试：controller层如果传入html，则可能存在于jsp内置属性冲突，导致出错，最好命名不要有冲突



访问连接测试：

http://localhost:8080/page/item/154684383340439

