**畅购电商系统开发**

**第3天**

传智播客.黑马程序员.深圳

# 学习目标

* SPU与SKU概念理解

SPU：某一款商品的公共属性

SKU:某款商品的不同参数对应的商品信息[某个商品]

* 新增商品、修改商品

增加：增加SPU和SKU

修改：修改SPU和SKU

* 商品审核、上架、下架

审核：修改审核状态

上架下架：修改上架下架状态

* 删除商品

逻辑删除：修改了删除状态

物理删除：真实删除了数据

* 找回商品

找回商品：一定是属于逻辑删除的商品

# SPU与SKU

## SPU与SKU概念

**SPU = Standard Product Unit （标准产品单位）**

* 概念 : SPU 是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。
* 通俗点讲，属性值、特性相同的货品就可以称为一个 SPU

例如：**华为P30**就是一个 SPU

**SKU=Stock Keeping Unit( 库存量单位)**

* SKU 即库存进出计量的单位， 可以是以件、盒、托盘等为单位。
* SKU 是物理上不可分割的最小存货单元。在使用时要根据不同业态，不同管理模式来处理。
* 在服装、鞋类商品中使用最多最普遍。

例如：**华为P30 红色 64G** 就是一个 SKU

## 表结构分析

tb\_spu 表 （SPU表）类似于对商品的分类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 字段类型 | 字段长度 | 备注 |
| id | 主键 | BIGINT |  |  |
| sn | 货号 | VARCHAR |  |  |
| name | SPU名 | VARCHAR |  |  |
| caption | 副标题 | VARCHAR |  |  |
| brand\_id | 品牌ID | INT |  |  |
| category1\_id | 一级分类 | INT |  |  |
| category2\_id | 二级分类 | INT |  |  |
| category3\_id | 三级分类 | INT |  |  |
| template\_id | 模板ID | INT |  |  |
| freight\_id | 运费模板id | INT |  |  |
| image | 图片 | VARCHAR |  |  |
| images | 图片列表 | VARCHAR |  |  |
| sale\_service | 售后服务 | VARCHAR |  |  |
| introduction | 介绍 | TEXT |  |  |
| spec\_items | 规格列表 | VARCHAR |  |  |
| para\_items | 参数列表 | VARCHAR |  |  |
| sale\_num | 销量 | INT |  |  |
| comment\_num | 评论数 | INT |  |  |
| is\_marketable | 是否上架 | CHAR |  |  |
| is\_enable\_spec | 是否启用规格 | CHAR |  |  |
| is\_delete | 是否删除 | CHAR |  |  |
| status | 审核状态 | CHAR |  |  |

tb\_sku 表（SKU商品表）

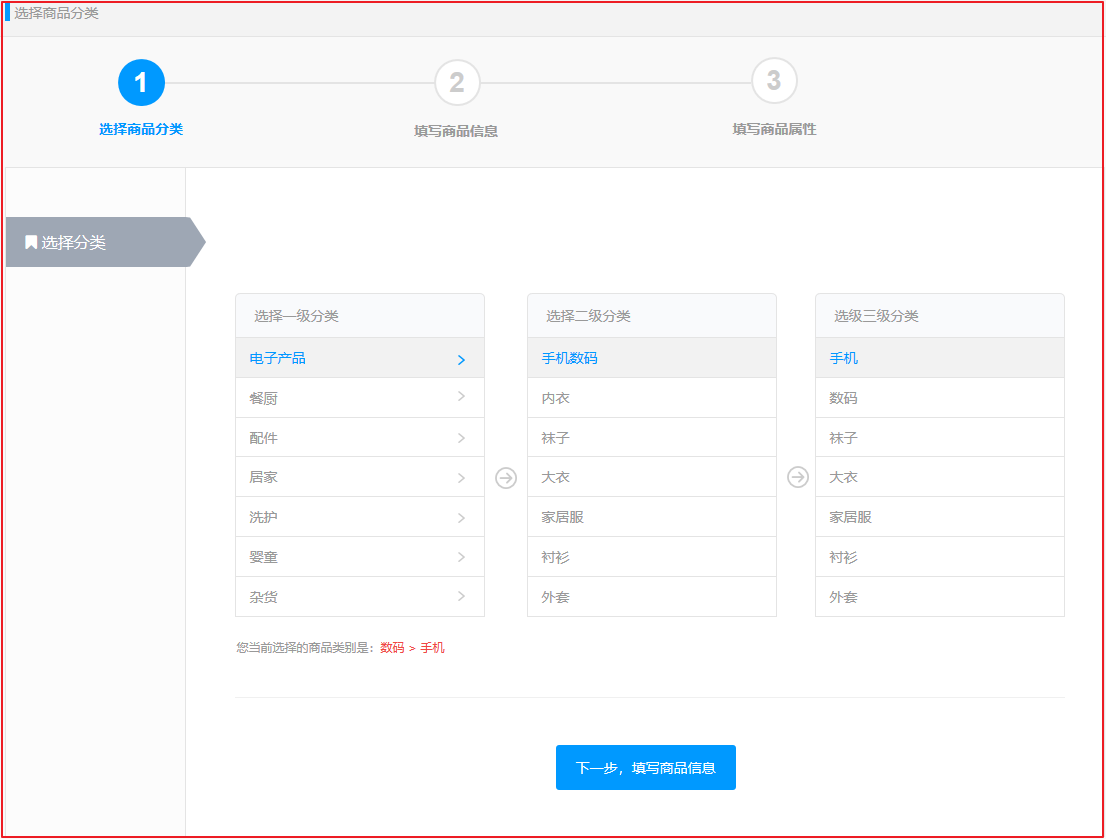
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 字段含义 | 字段类型 | 字段长度 | 备注 |
| id | 商品id | BIGINT |  |  |
| sn | 商品条码 | VARCHAR |  |  |
| name | SKU名称 | VARCHAR |  |  |
| price | 价格（分） | INT |  |  |
| num | 库存数量 | INT |  |  |
| alert\_num | 库存预警数量 | INT |  |  |
| image | 商品图片 | VARCHAR |  |  |
| images | 商品图片列表 | VARCHAR |  |  |
| weight | 重量（克） | INT |  |  |
| create\_time | 创建时间 | DATETIME |  |  |
| update\_time | 更新时间 | DATETIME |  |  |
| spu\_id | SPUID | BIGINT |  |  |
| category\_id | 类目ID | INT |  |  |
| category\_name | 类目名称 | VARCHAR |  |  |
| brand\_name | 品牌名称 | VARCHAR |  |  |
| spec | 规格 | VARCHAR |  |  |
| sale\_num | 销量 | INT |  |  |
| comment\_num | 评论数 | INT |  |  |
| status | 商品状态 1-正常，2-下架，3-删除 | CHAR |  |  |

# 新增和修改商品

## 需求分析

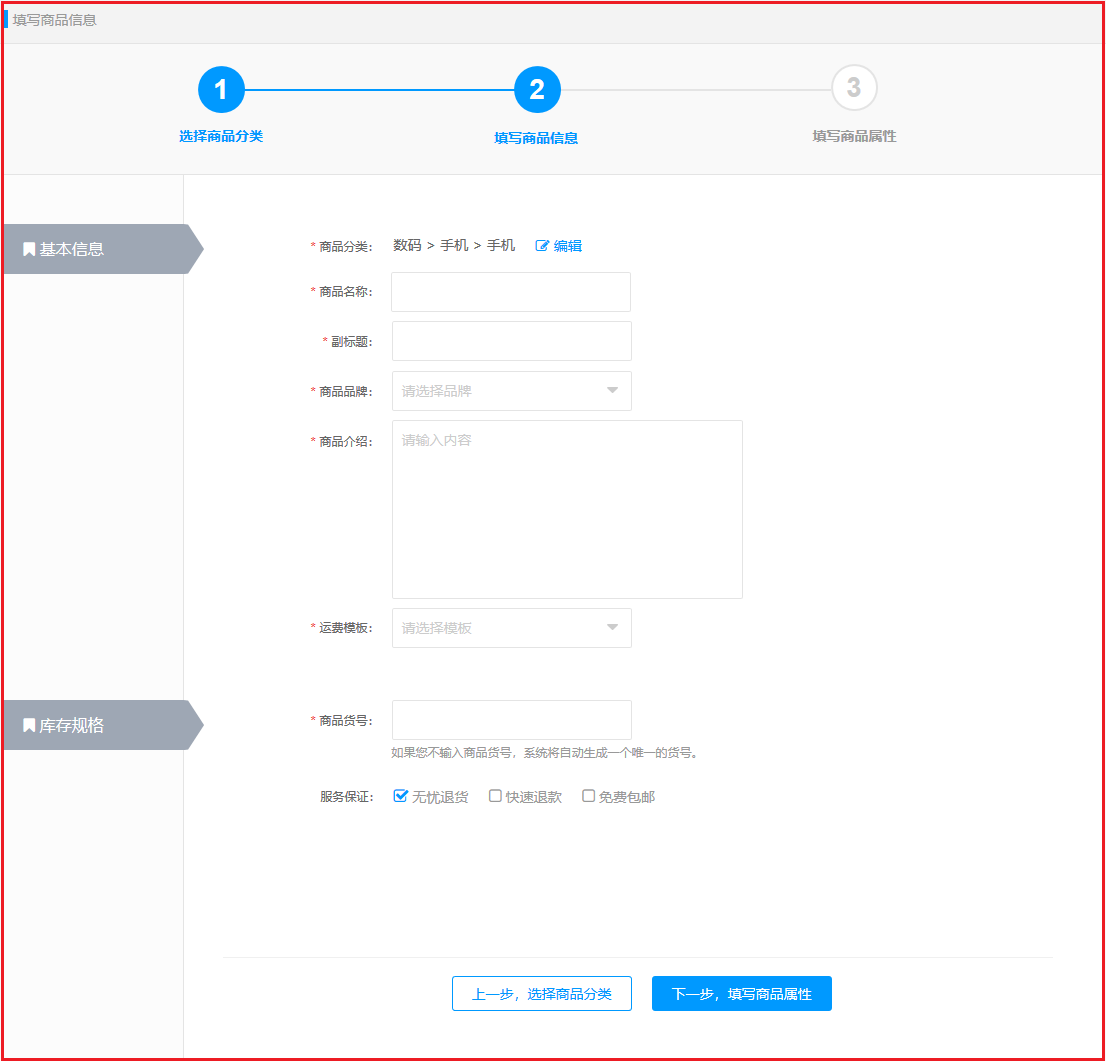
实现商品的新增与修改功能。

### 选择添加的商品所属分类

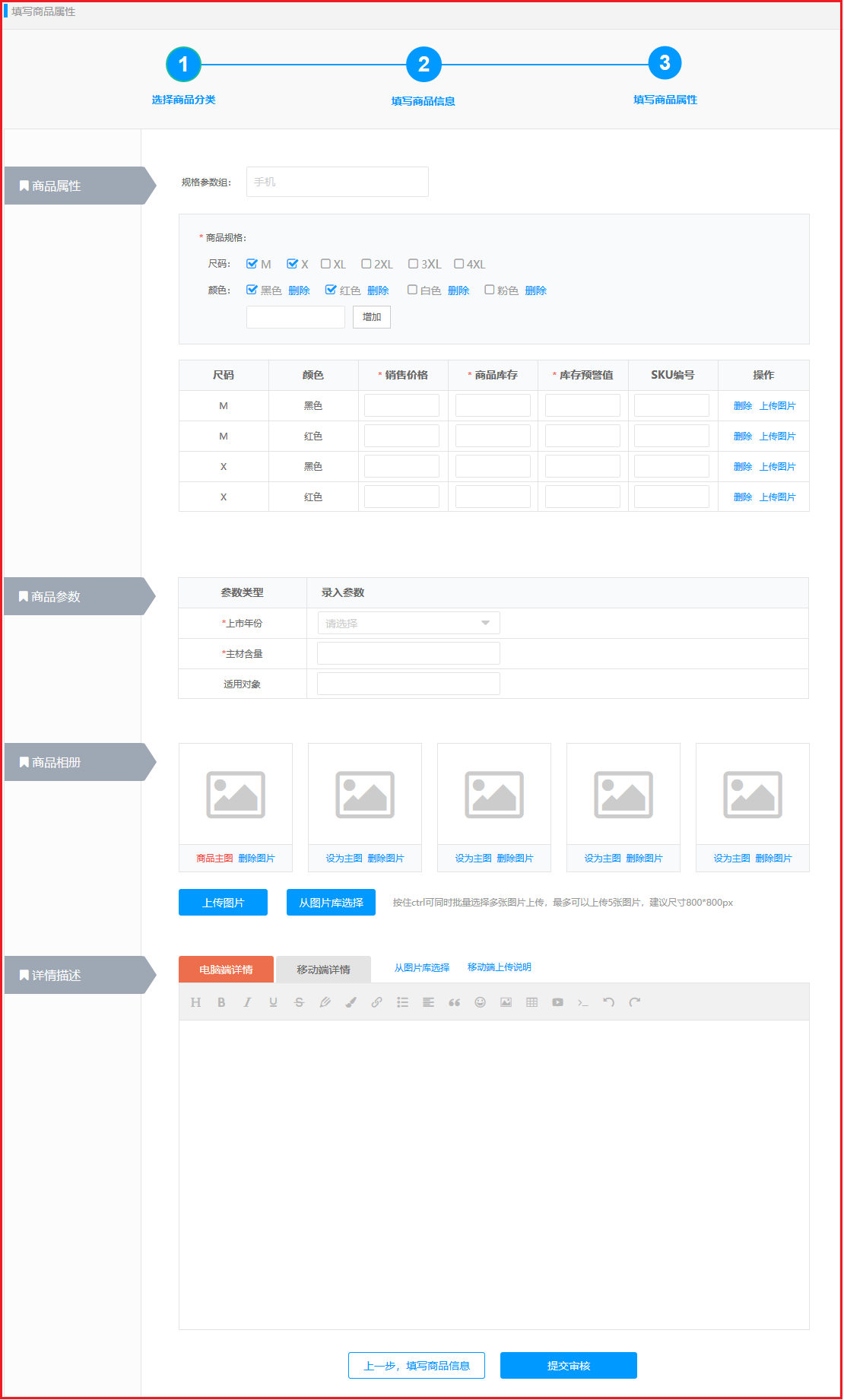


这块在第2天的代码中已经有一个根据父节点ID查询分类信息的方法，参考第2天的4.3.4的findByPrantId方法，首先查询顶级分类，也就是pid=0，然后根据用户选择的分类，将选择的分类作为pid查询子分类。

### 填写SPU的信息



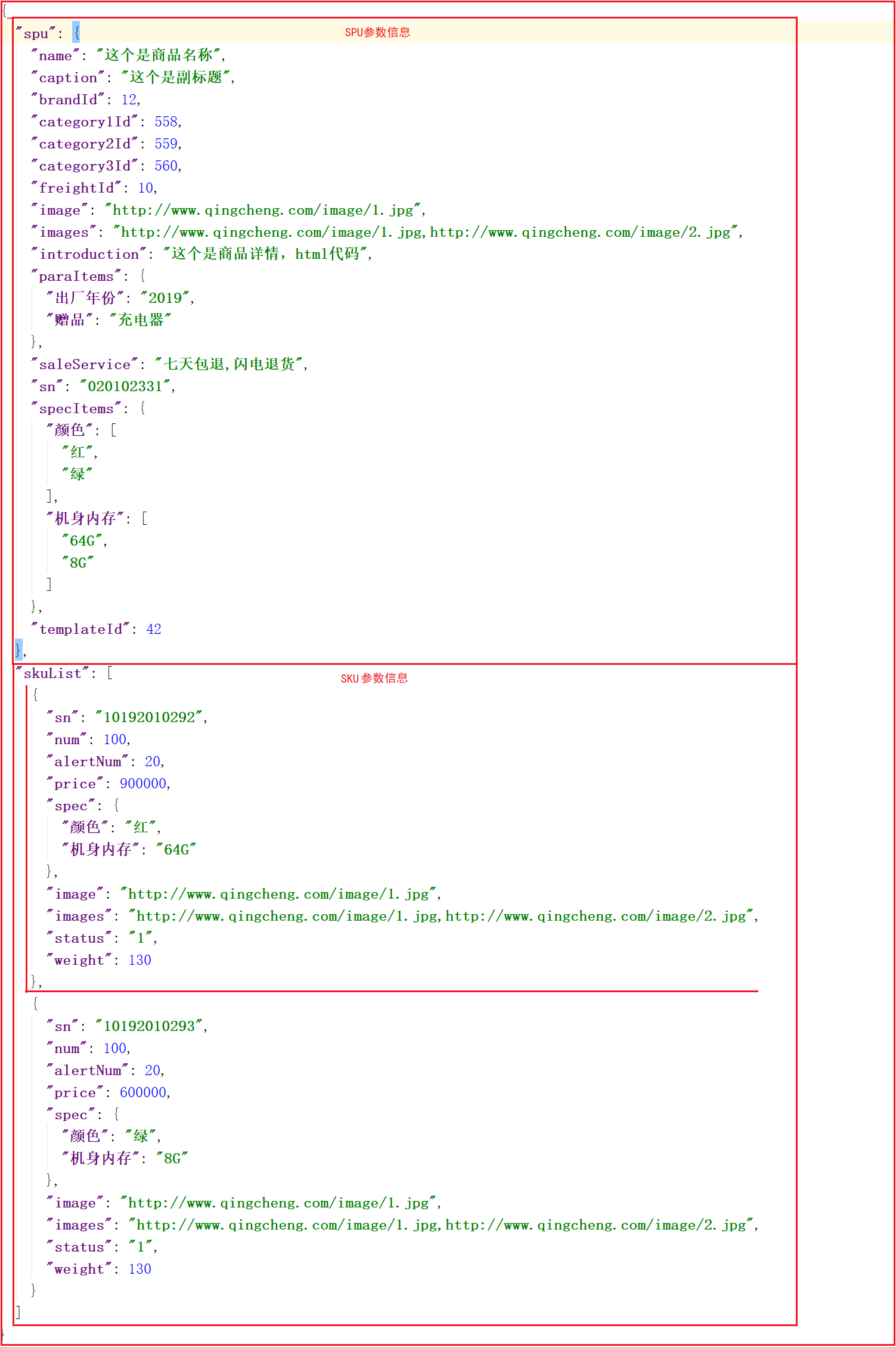
### 填写SKU信息



先进入选择商品分类 再填写商品的信息 填写商品的属性添加商品。

## 实现思路

前端传递给后端的数据格式 是一个spu对象和sku列表组成的对象,如下图:



上图JSON数据如下：

{

"**spu**": {

"**name**": "这个是商品名称",

"**caption**": "这个是副标题",

"**brandId**": 12,

"**category1Id**": 558,

"**category2Id**": 559,

"**category3Id**": 560,

"**freightId**": 10,

"**image**": "http://www.qingcheng.com/image/1.jpg",

"**images**": "http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jpg",

"**introduction**": "这个是商品详情，html代码",

"**paraItems**": {

"**出厂年份**": "2019",

"**赠品**": "充电器"

},

"**saleService**": "七天包退,闪电退货",

"**sn**": "020102331",

"**specItems**": {

"**颜色**": [

"红",

"绿"

],

"**机身内存**": [

"64G",

"8G"

]

},

"**templateId**": 42

},

"**skuList**": [

{

"**sn**": "10192010292",

"**num**": 100,

"**alertNum**": 20,

"**price**": 900000,

"**spec**": {

"**颜色**": "红",

"**机身内存**": "64G"

},

"**image**": "http://www.qingcheng.com/image/1.jpg",

"**images**": "http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jpg",

"**status**": "1",

"**weight**": 130

},

{

"**sn**": "10192010293",

"**num**": 100,

"**alertNum**": 20,

"**price**": 600000,

"**spec**": {

"**颜色**": "绿",

"**机身内存**": "8G"

},

"**image**": "http://www.qingcheng.com/image/1.jpg",

"**images**": "http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jpg",

"**status**": "1",

"**weight**": 130

}

]

}

## 代码生成器

准备工作：为了更快的实现代码编写，我们可以采用《黑马代码生成器》来批量生成代码，这些代码就已经实现了我们之前的增删改查功能。

《黑马代码生成器》一款由传智播客教育集团JavaEE教研团队开发的基于Freemarker模板引擎的“代码生成神器”。即便是一个工程几百个表，也可以瞬间完成基础代码的构建！用户只需建立数据库表结构，运行main方法就可快速生成可以运行的一整套代码，可以极大地缩短开发周期，降低人力成本。《黑马代码生成器》的诞生主要用于迅速构建生成微服务工程的Pojo、Dao、Service、Controller各层、并且可以生成swagger API模板等。 用户通过自己开发模板也可以实现生成php、python、C# 、c++、数据库存储过程等其它编程语言的代码。

《黑马代码生成器》目前已经开源，目录：资料\代码生成器

大家可以根据需求自行更新，或者上传到github开源都行。

## 功能实现

一会儿会用到ID生成，我们可以使用IdWorker，在启动类GoodsApplication中添加如下代码,用于创建IdWorker，并将IdWorker交给Spring容器，代码如下：

*/\*\*  
 \* 注册IdWorker实例对象  
 \** ***@return*** *\*/*@Bean  
**public** IdWorker getIdWorker(){  
 *//IdWorker(工作机器id,数据中心id)* **return new** IdWorker(0, 0);  
}

### 查询分类



在实现商品增加之前，需要先选择对应的分类，选择分类的时候，首选选择一级分类，然后根据选中的分类，将选中的分类作为查询的父ID，再查询对应的子分类集合，因此我们可以在后台编写一个方法，根据父类ID查询对应的分类集合即可。

需要操作的表：tb\_category

#### Service层

修改com.changgou.goods.service.CategoryService添加根据父类ID查询所有子节点，代码如下：

*/\*\*\*  
 \* 根据分类的父ID查询子分类节点集合  
 \*/*List<Category> findByParentId(Integer pid);

修改com.changgou.goods.service.impl.CategoryServiceImpl添加上面的实现，代码如下：

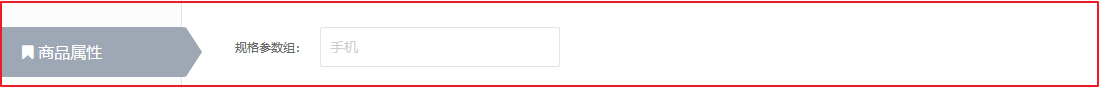
*/\*\*\*  
 \* 根据分类的父节点ID查询所有子节点  
 \** ***@param pid*** *\** ***@return*** *\*/*@Override  
**public** List<Category> findByParentId(Integer pid) {  
 *//SELECT \* FROM tb\_category WHERE parent\_id=?* Category category = **new** Category();  
 category.setParentId(pid);  
 *//select(查询对象)方法，会根据传入的查询对象判断属性，如果不为空时，拼接等值查询条件* **return categoryMapper**.select(category);  
}

#### Controller层

修改com.changgou.goods.controller.CategoryController添加根据父ID查询所有子类集合，代码如下：

*/\*\*\*\*  
 \* 根据节点ID查询所有子节点分类集合  
 \*/*@GetMapping(value = **"/list/{pid}"**)  
**public** Result<List<Category>> findByParentId(@PathVariable(value = **"pid"**)Integer pid){  
 *//调用Service实现查询* List<Category> categories = **categoryService**.findByParentId(pid);  
 **return new** Result<List<Category>>(**true**,StatusCode.***OK***,**"查询成功！"**,categories);  
}

### 模板查询(规格参数组)



如上图，当用户选中了分类后，需要根据分类的ID查询出对应的模板数据，并将模板的名字显示在这里

需要操作的表：tb\_category、tb\_template

#### Service层

修改com.changgou.goods.service.TemplateService接口，添加如下方法根据分类ID查询模板：

*/\*\*  
 \* 根据分类ID查询模板信息  
 \** ***@param id*** *\** ***@return*** *\*/*Template findByCategoryId(Integer id);

修改com.changgou.goods.service.impl.TemplateServiceImpl添加上面方法的实现：

@Autowired  
**private** CategoryMapper **categoryMapper**;  
*/\*\*\*  
 \* 根据分类ID查询模板信息  
 \** ***@param id*** *\** ***@return*** *\*/*@Override  
**public** Template findByCategoryId(Integer id) {  
 *//查询分类信息* Category category = **categoryMapper**.selectByPrimaryKey(id);  
  
 *//根据模板Id查询模板信息* **return templateMapper**.selectByPrimaryKey(category.getTemplateId());  
}

#### Controller层

修改com.changgou.goods.controller.TemplateController，添加根据分类ID查询模板数据：

*/\*\*\*  
 \* 根据分类查询模板数据  
 \** ***@param id****:分类ID  
 \*/*@GetMapping(value = **"/category/{id}"**)  
**public** Result<Template> findByCategoryId(@PathVariable(value = **"id"**)Integer id){  
 *//调用Service查询* Template template = **templateService**.findByCategoryId(id);  
 **return new** Result<Template>(**true**, StatusCode.***OK***,**"查询成功"**,template);  
}

### 查询分类品牌数据



用户每次选择了分类之后，可以根据用户选择的分类到tb\_category\_brand表中查询指定的品牌集合ID,然后根据品牌集合ID查询对应的品牌集合数据，再将品牌集合数据拿到这里来展示即可实现上述功能。

需要操作的表：tb\_category\_brand 、tb\_brand。

**实现方式**：

1. 先跟据分类id查询到List<CategoryBrand>，再根据List<CategoryBrand>中的品牌id查询所有品牌
2. 使用表连接查询(推荐)

#### Dao实现

修改com.changgou.goods.dao.BrandMapper添加根据分类ID查询对应的品牌数据，代码如下：

**public interface** BrandMapper **extends** Mapper<Brand> {  
 */\*\*\*  
 \* 查询分类对应的品牌集合  
 \*/* @Select(**"SELECT tb.\* FROM tb\_category\_brand tcb,tb\_brand tb WHERE tcb.category\_id=#{categoryid} AND tb.id=tcb.brand\_id"**)  
 List<Brand> findByCategory(Integer categoryid);  
}

#### Service层

修改com.changgou.goods.service.BrandService，添加根据分类ID查询指定的品牌集合方法，代码如下：

*/\*\*\*  
 \* 根据分类ID查询品牌集合  
 \** ***@param categoryid****:分类ID  
 \*/*List<Brand> findByCategory(Integer categoryid);

修改com.changgou.goods.service.impl.BrandServiceImpl添加上面方法的实现，代码如下：

*/\*\*\*  
 \* 根据分类ID查询品牌集合  
 \** ***@param categoryid****:分类ID  
 \** ***@return*** *\*/*@Override  
**public** List<Brand> findByCategory(Integer categoryid) {  
 *//根据当前分类，连表(tb\_category\_brand与tb\_brand)查询品牌列表* **return brandMapper**.findByCategory(categoryid);  
}

#### Controller层

修改,添加根据分类ID查询对应的品牌数据代码如下：

*/\*\*\*  
 \* 根据分类实现品牌列表查询  
 \* /brand/category/{id} 分类ID  
 \*/*@GetMapping(value = **"/category/{id}"**)  
**public** Result<List<Brand>> findBrandByCategory(@PathVariable(value = **"id"**)Integer categoryId){  
 *//调用Service查询品牌数据* List<Brand> categoryList = **brandService**.findByCategory(categoryId);  
 **return new** Result<List<Brand>>(**true**,StatusCode.***OK***,**"查询成功！"**,categoryList);  
}

### 规格查询



用户选择分类后，需要根据所选分类对应的模板ID查询对应的规格。

需要操作的表：tb\_category 、tb\_spec。

#### Service层

修改com.changgou.goods.service.SpecService添加根据分类ID查询规格列表，代码如下：

*/\*\*\*  
 \* 根据分类ID查询规格列表  
 \** ***@param categoryid*** *\** ***@return*** *\*/*List<Spec> findByCategoryId(Integer categoryid);

修改com.changgou.goods.service.impl.SpecServiceImpl添加上面方法的实现，代码如下：

@Autowired  
**private** CategoryMapper **categoryMapper**;  
*/\*\*\*  
 \* 根据分类ID查询规格列表**\** ***@return*** *\*/*@Override  
**public** List<Spec> findByCategoryId(Integer categoryid) {  
 *//先根据分类id查询分类，获取模板id* Category category = **categoryMapper**.selectByPrimaryKey(categoryid);  
 *//再根据分类的模板ID查询规格* Spec spec = **new** Spec();  
 spec.setTemplateId(category.getTemplateId());  
 **return specMapper**.select(spec);  
}

#### Controller层

修改com.changgou.goods.controller.SpecController添加根据分类ID查询规格数据，代码如下：

*/\*\*\*  
 \* 根据分类ID查询对应的规格列表  
 \*/*@GetMapping(value = **"/category/{id}"**)  
**public** Result<List<Spec>> findByCategoryId(@PathVariable(value = **"id"**)Integer categoryid){  
 *//调用Service查询* List<Spec> specs = **specService**.findByCategoryId(categoryid);  
 **return new** Result<List<Spec>>(**true**, StatusCode.***OK***,**"查询成功"**,specs);  
}

### 参数列表查询



当用户选中分类后，需要根据分类的模板ID查询对应的参数列表。

需要操作的表：tb\_category 、tb\_para。

#### Service层

修改com.changgou.goods.service.ParaService添加根据分类ID查询参数列表，代码如下：

*/\*\*\*  
 \* 根据分类ID查询参数列表  
 \** ***@param id*** *\** ***@return*** *\*/*List<Para> findByCategoryId(Integer id);

修改com.changgou.goods.service.impl.ParaServiceImpl添加上面方法的实现，代码如下：

@Autowired  
**private** CategoryMapper **categoryMapper**;  
*/\*\*\*  
 \* 根据分类ID查询参数列表  
 \** ***@return*** *\*/*@Override  
**public** List<Para> findByCategoryId(Integer id) {  
 *//先根据分类id查询分类信息，获取模板id* Category category = **categoryMapper**.selectByPrimaryKey(id);  
 *//再根据模板ID查询参数列表* Para para = **new** Para();  
 para.setTemplateId(category.getTemplateId());  
 **return paraMapper**.select(para);  
}

#### Controller层

修改com.changgou.goods.controller.ParaController，添加根据分类ID查询参数列表，代码如下：

*/\*\*  
 \* 根据分类ID查询参数列表  
 \** ***@return*** *\*/*@GetMapping(value = **"/category/{id}"**)  
**public** Result<List<Para>> getByCategoryId(@PathVariable(value = **"id"**)Integer id){  
 *//根据分类ID查询对应的参数信息* List<Para> paras = **paraService**.findByCategoryId(id);  
 **return new** Result<List<Para>>(**true**,StatusCode.***OK***,**"查询分类对应的参数成功！"**,paras);  
}

### SPU+SKU保存

保存商品数据的时候，需要保存Spu和Sku，一个Spu对应多个Sku，我们可以先构建一个Goods对象，将Spu和List<Sku>组合到一起,前端将2者数据提交过来，再实现添加操作。

需要操作的表：tb\_spu 、tb\_sku。

#### Pojo改造

修改changgou-service-goods-api工程创建组合实体类，创建com.changgou.goods.pojo.Goods,代码如下：

**public class** Goods **implements** Serializable {  
 *//SPU* **private** Spu **spu**;  
 *//SKU集合* **private** List<Sku> **skuList**;  
 *//..get..set..toString*}

#### 业务层

修改com.changgou.goods.service.SpuService接口，添加保存Goods方法，代码如下：

*/\*\*  
 \* 保存商品  
 \** ***@param goods*** *\*/***void** saveGoods(Goods goods);

修改com.changgou.goods.service.impl.SpuServiceImpl类，添加保存Goods的方法实现，代码如下：

@Autowired  
**private** IdWorker **idWorker**;  
@Autowired  
**private** CategoryMapper **categoryMapper**;  
@Autowired  
**private** BrandMapper **brandMapper**;  
@Autowired  
**private** SkuMapper **skuMapper**;  
*/\*\*\*  
 \* 保存Goods  
 \** ***@param goods*** *\*/*@Override  
**public void** saveGoods(Goods goods) {  
 *//增加Spu* Spu spu = goods.getSpu();  
 spu.setId(**idWorker**.nextId());  
 **spuMapper**.insertSelective(spu);  
  
 *//增加Sku* Date now = **new** Date();  
 *//查询分类* Category category = **categoryMapper**.selectByPrimaryKey(spu.getCategory3Id());  
 *//查询品牌* Brand brand = **brandMapper**.selectByPrimaryKey(spu.getBrandId());  
 *//获取Sku集合* List<Sku> skus = goods.getSkuList();  
 *//循环将数据加入到数据库* **for** (Sku sku : skus) {  
 *//构建SKU名称，采用SPU+规格值组装* **if**(StringUtils.*isEmpty*(sku.getSpec())){  
 sku.setSpec(**"{}"**);  
 }  
 *//获取Spu的名字* String name = spu.getName();  
 *//将规格转换成Map* Map<String,String> specMap = JSON.*parseObject*(sku.getSpec(), Map.**class**);  
 *//循环组装Sku的名字* **for** (Map.Entry<String, String> entry : specMap.entrySet()) {  
 name+=**" "**+entry.getValue();  
 }  
 sku.setName(name);  
  
 *//分布式ID* sku.setId(**idWorker**.nextId());  
 *//SpuId* sku.setSpuId(spu.getId());  
 *//创建日期* sku.setCreateTime(now);  
 *//修改日期* sku.setUpdateTime(now);  
 *//商品分类ID* sku.setCategoryId(spu.getCategory3Id());  
 *//分类名字* sku.setCategoryName(category.getName());  
 *//品牌名字* sku.setBrandName(brand.getName());  
 *//增加* **skuMapper**.insertSelective(sku);  
 }  
}

#### 控制层

修改com.changgou.goods.controller.SpuController，增加保存Goods方法，代码如下：

*/\*\*\*  
 \* 添加Goods  
 \** ***@param goods*** *\** ***@return*** *\*/*@PostMapping(**"save"**)  
**public** Result save(@RequestBody Goods goods){  
 **spuService**.saveGoods(goods);  
 **return new** Result(**true**,StatusCode.***OK***,**"保存成功"**);  
}

#### 测试数据

{

"**skuList**": [

{

"**alertNum**": 10,

"**brandName**": "华为",

"**categoryId**": 64,

"**commentNum**": 0,

"**image**": "http://www.baidu.com",

"**images**": "",

"**num**": 5,

"**price**": 1000,

"**saleNum**": 0,

"**sn**": "No1001",

"**spec**": "{\"颜色\":\"红色\",\"网络\":\"移动3G\"}",

"**weight**": 0

}

],

"**spu**": {

"**brandId**": 8557,

"**caption**": "华为手机大促销",

"**category1Id**": 1,

"**category2Id**": 59,

"**category3Id**": 64,

"**commentNum**": 0,

"**freightId**": 0,

"**images**": "http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jpg",

"**introduction**": "华为产品世界最强",

"**isEnableSpec**": "1",

"**isMarketable**": "1",

"**name**": "string",

"**specItems**": "{\"颜色\":[\"红\",\"绿\"],\"机身内存\":[\"64G\",\"8G\"]}",

"**paraItems**": "{\"赠品\":\"充电器\",\"出厂年份\":\"2019\"}",

"**saleNum**": 0,

"**saleService**": "一年包换",

"**sn**": "No10001",

"**status**": "1",

"**templateId**": 42

}

}

### 根据ID查询商品

需求：根据id 查询SPU和SKU列表 ，显示效果如下：

{

"**spu**": {

"**brandId**": 0,

"**caption**": "111",

"**category1Id**": 558,

"**category2Id**": 559,

"**category3Id**": 560,

"**commentNum**": null,

"**freightId**": null,

"**id**": 149187842867993,

"**image**": null,

"**images**": null,

"**introduction**": null,

"**isDelete**": null,

"**isEnableSpec**": "0",

"**isMarketable**": "1",

"**name**": "黑马智能手机",

"**paraItems**": null,

"**saleNum**": null,

"**saleService**": null,

"**sn**": null,

"**specItems**": null,

"**status**": null,

"**templateId**": 42

},

"**skuList**": [{

"**alertNum**": null,

"**brandName**": "金立（Gionee）",

"**categoryId**": 560,

"**categoryName**": "手机",

"**commentNum**": null,

"**createTime**": "2018-11-06 10:17:08",

"**id**": 1369324,

"**image**": null,

"**images**": "blob:http://localhost:8080/ec04d1a5-d865-4e7f-a313-2e9a76cfb3f8",

"**name**": "黑马智能手机",

"**num**": 100,

"**price**": 900000,

"**saleNum**": null,

"**sn**": "",

"**spec**": null,

"**spuId**": 149187842867993,

"**status**": "1",

"**updateTime**": "2018-11-06 10:17:08",

"**weight**": null

},{

"**alertNum**": null,

"**brandName**": "金立（Gionee）",

"**categoryId**": 560,

"**categoryName**": "手机",

"**commentNum**": null,

"**createTime**": "2018-11-06 10:17:08",

"**id**": 1369325,

"**image**": null,

"**images**": "blob:http://localhost:8080/ec04d1a5-d865-4e7f-a313-2e9a76cfb3f8",

"**name**": "黑马智能手机",

"**num**": 100,

"**price**": 900000,

"**saleNum**": null,

"**sn**": "",

"**spec**": null,

"**spuId**": 149187842867993,

"**status**": "1",

"**updateTime**": "2018-11-06 10:17:08",

"**weight**": null

}

]

}

#### 业务层

修改com.changgou.goods.service.SpuService接口,添加根据ID查找方法findGoodsById代码如下：

*/\*\*\*  
 \* 根据SPU的ID查找SPU以及对应的SKU集合  
 \** ***@param spuId*** *\*/*Goods findGoodsById(Long spuId);

修改com.changgou.goods.service.impl.SpuServiceImpl类，添加根据ID查找findGoodsById方法，代码如下：

*/\*\*\*  
 \* 根据SpuID查询goods信息  
 \** ***@param spuId*** *\** ***@return*** *\*/*@Override  
**public** Goods findGoodsById(Long spuId) {  
 *//封装Goods* Goods goods = **new** Goods();  
 *//查询Spu* Spu spu = **spuMapper**.selectByPrimaryKey(spuId);  
 goods.setSpu(spu);  
  
 *//查询List<Sku>* Sku sku = **new** Sku();  
 sku.setSpuId(spuId);  
 List<Sku> skus = **skuMapper**.select(sku);  
 goods.setSkuList(skus);  
 **return** goods;  
}

#### 控制层

修改com.changgou.goods.controller.SpuController，修改findById方法，代码如下：

@GetMapping(**"goods/{id}"**)  
**public** Result<Goods> findById(@PathVariable Long id){  
 *//调用SpuService实现根据主键查询Spu* Goods goods = **spuService**.findGoodsById(id);  
 **return new** Result<Goods>(**true**,StatusCode.***OK***,**"查询成功"**,goods);  
}

测试：http://localhost:18081/spu/goods/1088256029394866176

### 保存修改

思路：修改changgou-service-goods的SpuServiceImpl的saveGoods方法，修改添加SPU部分代码：

* 判断是否传入spuId，没有传入时，执行之前的保存逻辑
* 如果传入了spuId说明是修改操作，更新spu同时，删除原来所有的sku列表，再重新添加

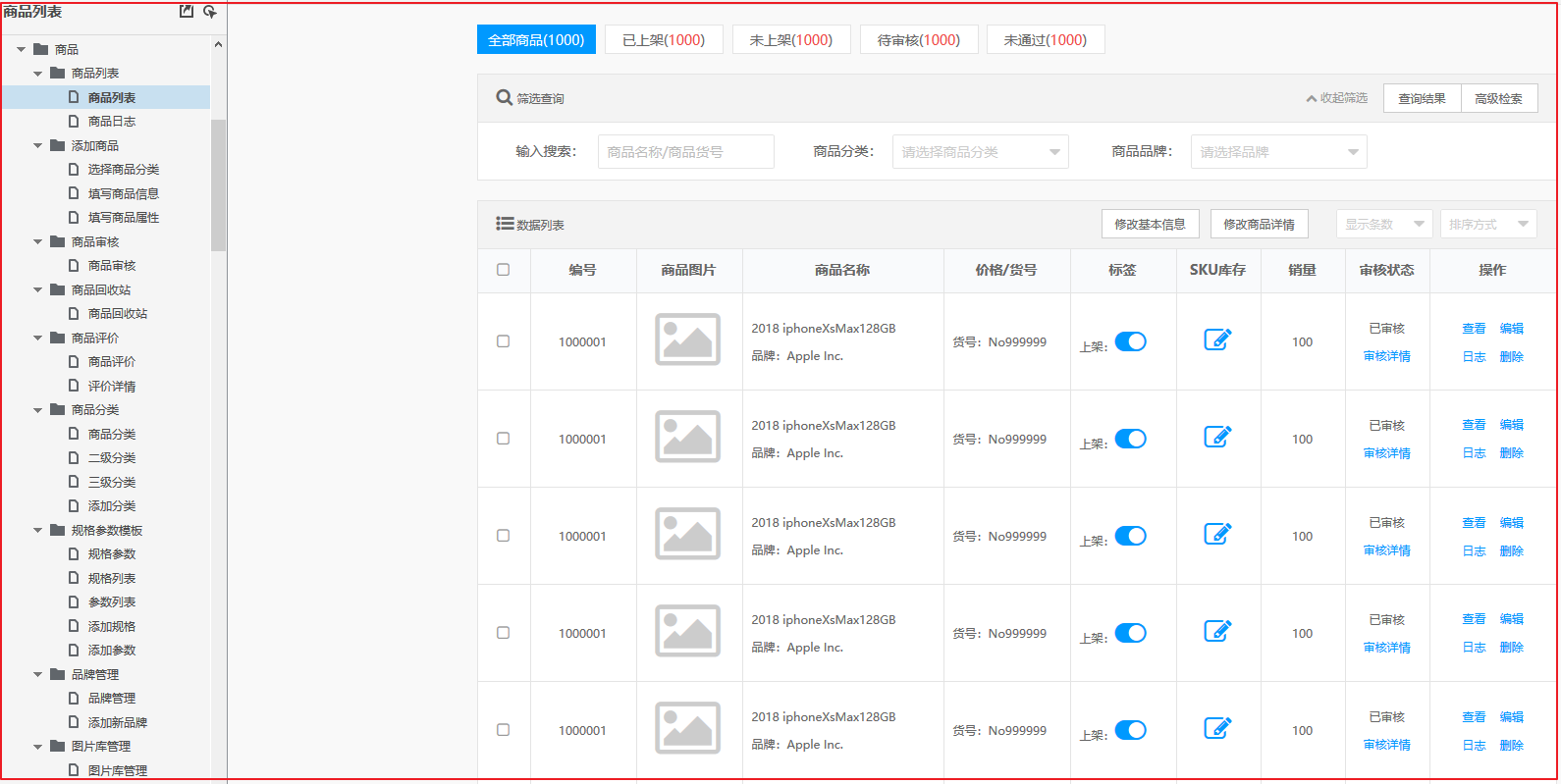
代码如下：

@Override  
**public void** saveGoods(Goods goods) {  
 *//增加Spu* Spu spu = goods.getSpu();  
 *//判断是否传入spuId，没有传入时，执行之前的保存逻辑* **if**(spu .getId() == **null**){  
 spu.setId(**idWorker**.nextId());  
 **spuMapper**.insertSelective(spu);  
 }**else**{ *//如果传入了spuId说明是修改操作，更新spu同时，删除原来所有的sku列表，再重新添加  
 //更新spu* **spuMapper**.updateByPrimaryKeySelective(spu);  
 *//删除原来的sku列表* Sku where = **new** Sku();  
 where.setSpuId(spu.getId());  
 **skuMapper**.delete(where);  
 }  
  
 *//增加Sku* Date now = **new** Date();  
 *//查询分类* Category category = **categoryMapper**.selectByPrimaryKey(spu.getCategory3Id());  
 *//查询品牌* Brand brand = **brandMapper**.selectByPrimaryKey(spu.getBrandId());  
 *//获取Sku集合* List<Sku> skus = goods.getSkuList();  
 *//循环将数据加入到数据库* **for** (Sku sku : skus) {  
 *//构建SKU名称，采用SPU+规格值组装* **if**(StringUtils.*isEmpty*(sku.getSpec())){  
 sku.setSpec(**"{}"**);  
 }  
 *//获取Spu的名字* String name = spu.getName();  
 *//将规格转换成Map* Map<String,String> specMap = JSON.*parseObject*(sku.getSpec(), Map.**class**);  
 *//循环组装Sku的名字* **for** (Map.Entry<String, String> entry : specMap.entrySet()) {  
 name+=**" "**+entry.getValue();  
 }  
 sku.setName(name);  
  
 *//分布式ID* sku.setId(**idWorker**.nextId());  
 *//SpuId* sku.setSpuId(spu.getId());  
 *//创建日期* sku.setCreateTime(now);  
 *//修改日期* sku.setUpdateTime(now);  
 *//商品分类ID* sku.setCategoryId(spu.getCategory3Id());  
 *//分类名字* sku.setCategoryName(category.getName());  
 *//品牌名字* sku.setBrandName(brand.getName());  
 *//增加* **skuMapper**.insertSelective(sku);  
 }  
}

# 商品列表

## 需求分析

如图所示 展示商品的列表。并实现分页。



**思路**：

* 根据查询的条件 分页查询 并返回分页结果即可。
* 分页查询 采用 pagehelper ，条件查询 通过map进行封装传递给后台即可。

**代码实现:**

在代码生成器生成的代码中已经包含了该实现，这里就省略了。

# 商品审核与上下架

## 需求分析

商品新增后，审核状态为0（未审核），默认为下架状态。

* 审核商品，需要校验是否是被删除的商品，如果未删除则修改审核状态为1，并自动上架
* 下架商品，需要校验是否是被删除的商品，如果未删除则修改上架状态为0
* 上架商品，需要审核通过的商品

## 实现思路

（1）按照ID查询SPU信息

（2）判断修改审核、上架和下架状态

（3）保存SPU

## 商品审核

实现审核通过，自动上架。

### 业务层

修改修改changgou-service-goods工程的com.changgou.goods.service.SpuService接口，添加审核方法，代码如下：

*/\*\*\*  
 \* 商品审核  
 \** ***@param spuId*** *\*/***void** audit(Long spuId);

修改changgou-service-goods工程的com.changgou.goods.service.impl.SpuServiceImpl类，添加audit方法，代码如下：

*/\*\*\*  
 \* 商品审核  
 \** ***@param spuId*** *\*/*@Override  
**public void** audit(Long spuId) {  
 *//查询商品* Spu spu = **spuMapper**.selectByPrimaryKey(spuId);  
 *//判断商品是否已经删除* **if**(spu.getIsDelete().equalsIgnoreCase(**"1"**)){  
 **throw new** RuntimeException(**"此商品已被删除，无法审核！"**);  
 }  
 *//实现上架和审核* spu.setStatus(**"1"**); *//审核通过* spu.setIsMarketable(**"1"**); *//上架* **spuMapper**.updateByPrimaryKeySelective(spu);  
}

### 控制层

修改com.changgou.goods.controller.SpuController，新增audit方法，代码如下：

*/\*\*  
 \* 审核商品  
 \** ***@param id*** *\** ***@return*** *\*/*@PutMapping(**"/audit/{id}"**)  
**public** Result audit(@PathVariable Long id){  
 **spuService**.audit(id);  
 **return new** Result(**true**,StatusCode.***OK***,**"审核成功"**);  
}

## 下架商品

### 业务层

修改com.changgou.goods.service.SpuService接口，添加pull方法，用于商品下架，代码如下：

*/\*\*\*  
 \* 商品下架  
 \** ***@param spuId*** *\*/***void** pull(Long spuId);

修改com.changgou.goods.service.impl.SpuServiceImpl，添加如下方法：

*/\*\*  
 \* 商品下架  
 \** ***@param spuId*** *\*/*@Override  
**public void** pull(Long spuId) {  
 Spu spu = **spuMapper**.selectByPrimaryKey(spuId);  
 **if**(spu.getIsDelete().equals(**"1"**)){  
 **throw new** RuntimeException(**"此商品已被删除，无法下架！"**);  
 }  
 spu.setIsMarketable(**"0"**);*//下架状态* **spuMapper**.updateByPrimaryKeySelective(spu);  
}

### 控制层

修改com.changgou.goods.controller.SpuController，添加pull方法，代码如下：

*/\*\*  
 \* 下架  
 \** ***@param id*** *\** ***@return*** *\*/*@PutMapping(**"/pull/{id}"**)  
**public** Result pull(@PathVariable Long id){  
 **spuService**.pull(id);  
 **return new** Result(**true**,StatusCode.***OK***,**"下架成功"**);  
}

## 上架商品

必须是通过审核的商品才能上架

### 业务层

修改com.changgou.goods.service.SpuService，添加put方法，代码如下：

*/\*\*\*  
 \* 商品上架  
 \** ***@param spuId*** *\*/***void** put(Long spuId);

修改com.changgou.goods.service.impl.SpuServiceImpl，添加put方法实现，代码如下：

*/\*\*\*  
 \* 商品上架  
 \** ***@param spuId*** *\*/*@Override  
**public void** put(Long spuId) {  
 Spu spu = **spuMapper**.selectByPrimaryKey(spuId);  
 *//检查是否删除的商品* **if**(spu.getIsDelete().equals(**"1"**)){  
 **throw new** RuntimeException(**"此商品已被删除，无法上架！"**);  
 }  
 **if**(!spu.getStatus().equals(**"1"**)){  
 **throw new** RuntimeException(**"未通过审核的商品无法上架！"**);  
 }  
 *//上架状态* spu.setIsMarketable(**"1"**);  
 **spuMapper**.updateByPrimaryKeySelective(spu);  
}

### 控制层

修改com.changgou.goods.controller.SpuController，添加put方法代码如下：

*/\*\*  
 \* 商品上架**\*/*@PutMapping(**"/put/{id}"**)  
**public** Result put(@PathVariable Long id){  
 **spuService**.put(id);  
 **return new** Result(**true**,StatusCode.***OK***,**"上架成功"**);  
}

## 批量上架

前端传递一组商品ID，后端进行批量上下架处理

### 业务层

修改com.changgou.goods.service.SpuService接口，代码如下：

*/\*\*  
 \* 批量上架  
 \** ***@param ids*** *\** ***@return*** *\*/***int** putMany(Long[] ids);

修改com.changgou.goods.service.impl.SpuServiceImpl，添加批量上架方法实现，代码如下：

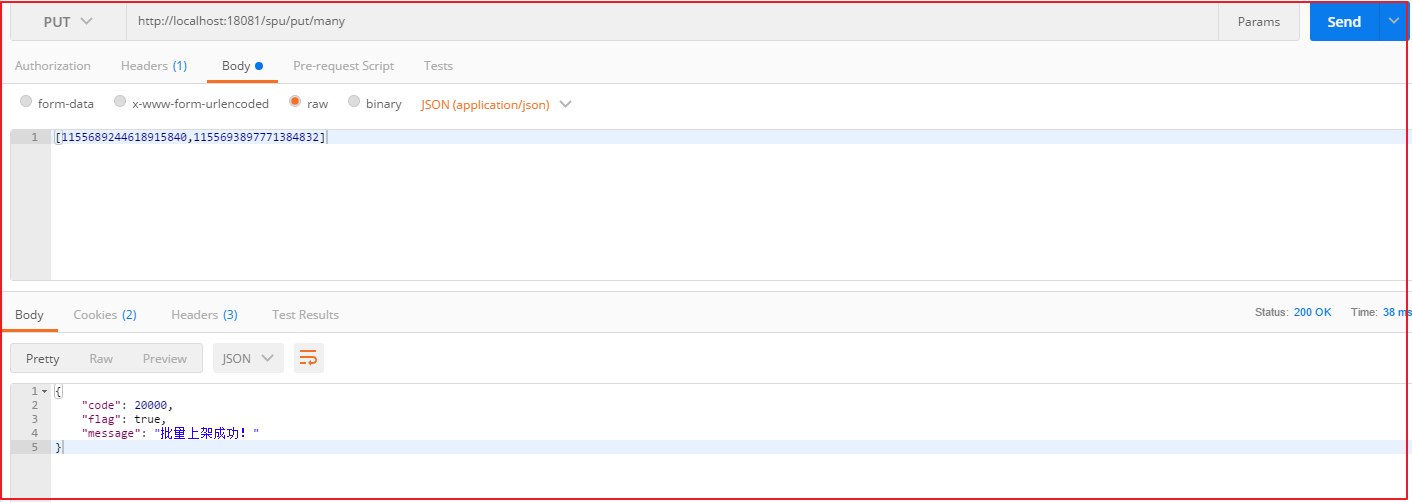
*/\*\*\*  
 \* 批量上架  
 \** ***@param ids****:需要上架的商品ID数组  
 \** ***@return*** *\*/*@Override  
**public int** putMany(Long[] ids) {  
 *//修改结果* Spu spu = **new** Spu();  
 spu.setIsMarketable(**"1"**);*//上架  
  
 //修改条件* Example example = **new** Example(Spu.**class**);  
 Example.Criteria criteria = example.createCriteria();  
 criteria.andIn(**"id"**, Arrays.*asList*(ids)); *//id* criteria.andEqualTo(**"isMarketable"**, **"0"**); *//下架商品才能上架* criteria.andEqualTo(**"status"**, **"1"**); *//审核通过的* criteria.andEqualTo(**"isDelete"**, **"0"**); *//非删除的  
   
 //执行更新* **return spuMapper**.updateByExampleSelective(spu, example);  
}

### 控制层

修改com.changgou.goods.controller.SpuController，天啊及批量上架方法，代码如下：

*/\*\*  
 \* 批量上架  
 \** ***@param ids*** *\** ***@return*** *\*/*@PutMapping(**"/put/many"**)  
**public** Result putMany(@RequestBody Long[] ids){  
 **int** count = **spuService**.putMany(ids);  
 **return new** Result(**true**,StatusCode.***OK***,**"上架"**+count+**"个商品"**);  
}

使用Postman测试：



## 批量下架

学员实现

# 删除与还原商品

## 需求分析

请看管理后台的静态原型

商品列表中的删除商品功能，并非真正的删除，而是将删除标记的字段设置为1，

在回收站中有恢复商品的功能，将删除标记的字段设置为0

在回收站中有删除商品的功能，是真正的物理删除。

## 实现思路

* 逻辑删除商品，修改spu表is\_delete字段为1
* 商品回收站显示spu表is\_delete字段为1的记录
* 回收商品，修改spu表is\_delete字段为0

## 逻辑删除商品

### 业务层

修改com.changgou.goods.service.SpuService接口，增加logicDelete方法，代码如下：

*/\*\*\*  
 \* 逻辑删除  
 \** ***@param spuId*** *\*/***void** logicDelete(Long spuId);

修改com.changgou.goods.service.impl.SpuServiceImpl，添加logicDelete方法实现，代码如下：

*/\*\*\*  
 \* 逻辑删除  
 \** ***@param spuId*** *\*/*@Override  
**public void** logicDelete(Long spuId) {  
 Spu spu = **spuMapper**.selectByPrimaryKey(spuId);  
 *//检查是否下架的商品* **if**(!spu.getIsMarketable().equals(**"0"**)){  
 **throw new** RuntimeException(**"必须先下架再删除！"**);  
 }  
 *//删除* spu.setIsDelete(**"1"**);  
 **spuMapper**.updateByPrimaryKeySelective(spu);  
}

### 控制层

修改com.changgou.goods.controller.SpuController，添加logicDelete方法，如下：

*/\*\*  
 \* 逻辑删除  
 \** ***@param id*** *\** ***@return*** *\*/*@DeleteMapping(**"/logic/delete/{id}"**)  
**public** Result logicDelete(@PathVariable Long id){  
 **spuService**.logicDelete(id);  
 **return new** Result(**true**,StatusCode.***OK***,**"逻辑删除成功！"**);  
}

## 还原被删除的商品

### 业务层

修改com.changgou.goods.service.SpuService接口，添加restore方法代码如下：

*/\*\*\*  
 \* 还原被删除商品  
 \** ***@param spuId*** *\*/***void** restore(Long spuId);

修改com.changgou.goods.service.impl.SpuServiceImpl类，添加restore方法，代码如下：

*/\*\*  
 \* 恢复数据  
 \** ***@param spuId*** *\*/*@Override  
**public void** restore(Long spuId) {  
 Spu spu = **spuMapper**.selectByPrimaryKey(spuId);  
 *//检查是否删除的商品* **if**(!**"1"**.equals(spu.getIsDelete())){  
 **throw new** RuntimeException(**"此商品未删除！"**);  
 }  
 spu.setIsDelete(**"0"**); *//恢复数据* **spuMapper**.updateByPrimaryKeySelective(spu);  
}

### 控制层

修改com.changgou.goods.controller.SpuController，添加restore方法，代码如下：

*/\*\*  
 \* 恢复数据  
 \** ***@return*** *\*/*@PutMapping(**"/restore/{id}"**)  
**public** Result restore(@PathVariable Long id){  
 **spuService**.restore(id);  
 **return new** Result(**true**,StatusCode.***OK***,**"数据恢复成功！"**);  
}

## 物理删除商品

修改com.changgou.goods.service.impl.SpuServiceImpl的delete方法,代码如下：

*/\*\*  
 \* 删除  
 \** ***@param id*** *\*/*@Override  
**public void** delete(Long id){  
 Spu spu = **spuMapper**.selectByPrimaryKey(id);  
 *//检查是否被逻辑删除 ,必须先逻辑删除后才能物理删除* **if**(!**"1"**.equals(spu.getIsDelete())){  
 **throw new** RuntimeException(**"此商品不能删除！"**);  
 }  
 **spuMapper**.deleteByPrimaryKey(id);  
}

其它代码不用修改，直接测试。