尚品汇商城

版本：V 2.0

商品（spu）管理

# 一 业务介绍

## 1 SPU与SKU

SPU(Standard Product Unit)：标准化产品单元。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。

‘Stock Keeping Unit（库存量单位）。即库存进出计量的基本单元，可以是以件，盒，托盘等为单位。

SKU这是对于大型连锁超市DC（配送中心）物流管理的一个必要的方法。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的SKU号。

比如，咱们购买一台iPhoneX手机，iPhoneX手机就是一个SPU，但是你购买的时候，不可能是以iPhoneX手机为单位买的，商家也不可能以iPhoneX为单位记录库存。必须要以什么颜色什么版本的iPhoneX为单位。比如，你购买的是一台银色、128G内存的、支持联通网络的iPhoneX ，商家也会以这个单位来记录库存数。那这个更细致的单位就叫库存单元（SKU）。

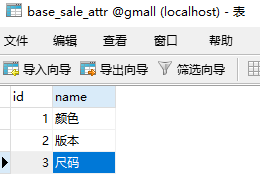


销售属性与平台属性

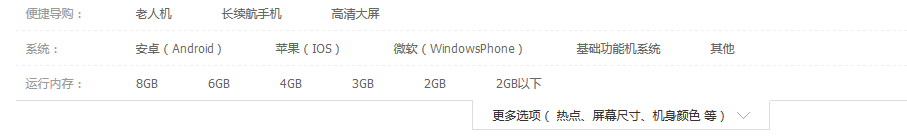
销售属性，就是商品详情页右边，可以通过销售属性来定位一组spu下的哪款sku。可以让当前的商品详情页，跳转到自己的“兄弟”商品。

一般每种商品的销售属性不会太多，大约1-4种。整个电商的销售属性种类也不会太多，大概10种以内。比如：颜色、尺寸、版本、套装等等。不同销售属性的组合也就构成了一个spu下多个sku的结构。





平台属性， 就是之前分类下面，辅助搜索的，类似于条件的属性。



销售属性与平台属性各自独立。一个SPU会决定一个商品都有哪些销售属性，比如iPhonx会有颜色、版本、内存的销售属性，某个T桖衫只有尺寸这个销售属性。

而某个商品有什么平台属性，由他的3级分类决定。比如笔记本包括：运行内存、cpu、显卡、硬盘、屏幕尺寸等等。

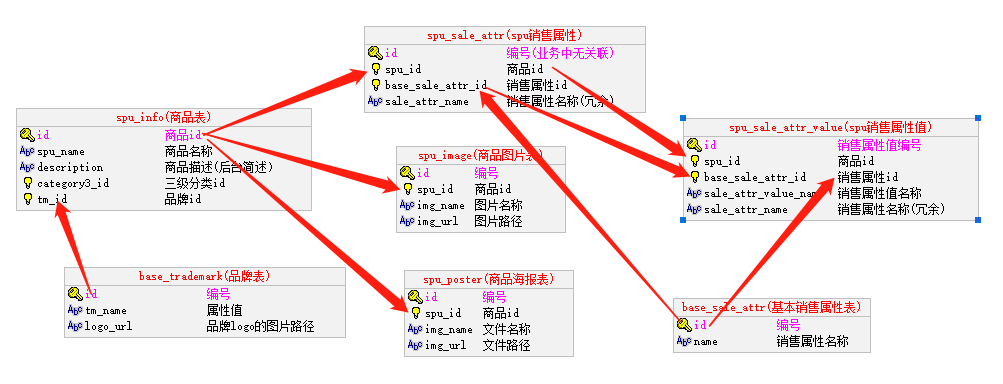
## 2 SKU与SPU的图片资源

另外同一个SPU下的SKU可以共用一些资源，比如商品图片，海报等等。毕竟同一种商品，大部分图片都是共用的只有因为颜色尺寸等，很少的差别。那么一般来说商品图片都是在新增SPU时上传的，在新增SKU时从该SPU已上传的图片中选择。

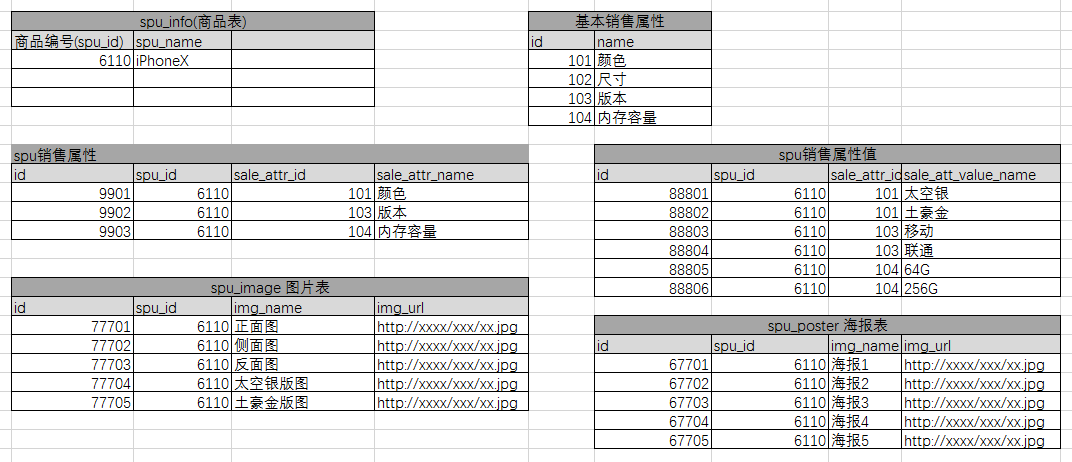
而海报几乎是所有SPU下的SKU都一样。

## 3 数据结构图

根据以上的需求，以此将SPU关联的数据库表结构设计为如下：



## 4 数据示例：



# 二、接口分析

## 1 spu分页列表接口



参考接口文档《电商管理系统API接口文档 .doc》

Spu接口：1

## 2 添加spu



参考接口文档《电商管理系统API接口文档 .doc》

Spu接口：2,3,4

## 3 品牌管理

参考接口文档《电商管理系统API接口文档 .doc》

品牌接口

# 三、列表查询功能开发

## 1 后台代码

### 1.1 Model

|  |
| --- |
| @Data @ApiModel(description = **"SpuInfo"**) @TableName(**"spu\_info"**) **public class** SpuInfo **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"商品名称"**)  @TableField(**"spu\_name"**)  **private** String **spuName**;   @ApiModelProperty(value = **"商品描述(后台简述）"**)  @TableField(**"description"**)  **private** String **description**;   @ApiModelProperty(value = **"三级分类id"**)  @TableField(**"category3\_id"**)  **private** Long **category3Id**;   @ApiModelProperty(value = **"品牌id"**)  @TableField(**"tm\_id"**)  **private** Long **tmId**;  } |

### 1.2 mapper

|  |
| --- |
| @Mapper **public interface** SpuInfoMapper **extends** BaseMapper<SpuInfo> {  } |

### 1.3 ManageService

|  |
| --- |
| */\*\*  \* spu分页查询  \** ***@param pageParam*** *\** ***@param spuInfo*** *\** ***@return*** *\*/* IPage<SpuInfo> selectPage(Page<SpuInfo> pageParam, SpuInfo spuInfo); |
| ManageServiceImpl  @Override **public** IPage<SpuInfo> selectPage(Page<SpuInfo> pageParam, SpuInfo spuInfo) {  QueryWrapper<SpuInfo> queryWrapper = **new** QueryWrapper<>();  queryWrapper.eq(**"category3\_id"**, spuInfo.getCategory3Id());  queryWrapper.orderByDesc(**"id"**);  **return spuInfoMapper**.selectPage(pageParam, queryWrapper); } |

注：ManageServiceImpl类添加如下：

@Autowired  
**private** SpuInfoMapper **spuInfoMapper**;

### 1.4 controller

|  |
| --- |
| @Api(tags = **"商品SPU接口"**) @RestController @RequestMapping(**"admin/product"**) **public class** SpuManageController {   @Autowired  **private** ManageService **manageService**;   */\*\*  \* SPU分页查询  \** ***@param page*** *\** ***@param limit*** *\** ***@param spuInfo*** *\** ***@return*** *\*/* @GetMapping(**"{page}/{limit}"**)  **public** Result<IPage<SpuInfo>> index(  @ApiParam(name = **"page"**, value = **"当前页码"**, required = **true**)  @PathVariable(**"page"**) Long page,   @ApiParam(name = **"limit"**, value = **"每页记录数"**, required = **true**)  @PathVariable(**"limit"**) Long limit,   @ApiParam(name = **"spuInfo"**, value = **"查询对象"**, required = **false**)  SpuInfo spuInfo) {  Page<SpuInfo> pageParam = **new** Page<>(page, limit);  IPage<SpuInfo> pageModel = **manageService**.selectPage(pageParam,spuInfo);  **return** Result.*ok*(pageModel);  }  } |

# 四、spu的保存功能中的图片上传

## 1文件服务器

文件，图片存储需要注意的：

图片名称不能重复

图片的后缀名，保存跟源文件后缀名一致

文件服务器-长期开着的

现在咱们实现了文件从客户端提交，并展示的功能。服务器端要做的就是接收文件流，保存起来，并且返回给客户端文件的访问地址。

传统的用io流保存到web服务器本地的方式，可以直接用当前web服务的路径+图片名称来访问。

但是类似于商品图片这种海量级文件，光靠web服务器的硬盘是无法满足的。

另外如果，web服务器是集群的那么A服务器是没法访问B服务器的本地文件的。

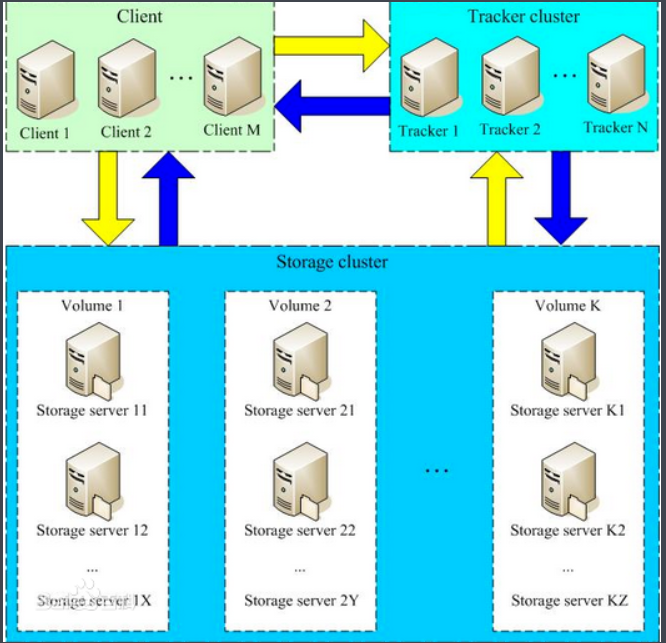
所以需要把文件服务单独管理起来，成为文件服务器。

实现方式就是nginx+FastDFS

### 1.1 FastDFS介绍

FastDFS 是一个由 C 语言实现的开源轻量级**分布式文件系统**，作者余庆(happyfish100)，支持 Linux、FreeBSD、AID 等 Unix 系统，解决了大数据存储和读写负载均衡等问题，适合存储 4KB~500MB 之间的小文件，如图片网站、短视频网站、文档、app 下载站等，UC、京东、支付宝、迅雷、酷狗等都有使用。

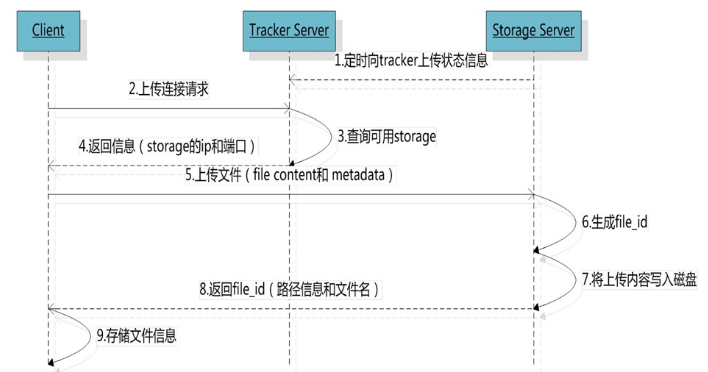
该软件作者是**阿里巴巴**大牛、chinaUnix版主**余庆**个人独立开发的。



FastDFS是一个开源的轻量级[分布式文件系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F/1250388" \t "https://baike.baidu.com/item/fastdfs/_blank)，它对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和[负载均衡](https://baike.baidu.com/item/%E8%B4%9F%E8%BD%BD%E5%9D%87%E8%A1%A1/932451" \t "https://baike.baidu.com/item/fastdfs/_blank)的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。

FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

### 1.2 FastDFS上传下载的流程



只要 storage 返回图片的路径图片名称，我们就能通过浏览器来访问图片了？

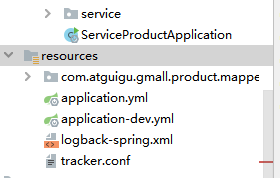
图片服务器在linux ？ nginx 做反向代理{图片服务器}！

### 1.3 安装步骤参见《FastDFS安装说明》

添加依赖

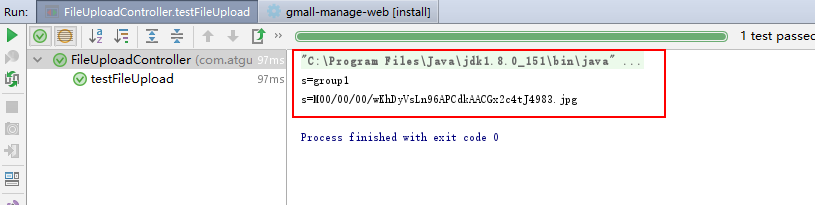
|  |
| --- |
| *<!--添加fdfs依赖-->* <**dependency**>  <**groupId**>net.oschina.zcx7878</**groupId**>  <**artifactId**>fastdfs-client-java</**artifactId**> </**dependency**> |

在项目resource中添加tracker.conf 配置文件



|  |
| --- |
| **tracker\_server=**47.93.118.241**:22122**  # 连接超时时间，针对socket套接字函数connect，默认为30秒 connect\_timeout=30000  # 网络通讯超时时间，默认是60秒 network\_timeout=60000 |
| @Test **public void** textFileUpload() **throws** IOException, MyException {  String file = **this**.getClass().getResource(**"/tracker.conf"**).getFile();  ClientGlobal.*init*(file);  TrackerClient trackerClient=**new** TrackerClient();  TrackerServer trackerServer=trackerClient.getConnection();  StorageClient storageClient=**new** StorageClient(trackerServer,**null**);  String orginalFilename=**"e://victor.jpg"**;  String[] upload\_file = storageClient.upload\_file(orginalFilename, **"jpg"**, **null**);  **for** (**int** i = 0; i < upload\_file.**length**; i++) {  String s = upload\_file[i];  System.***out***.println(**"s = "** + s);  } } |

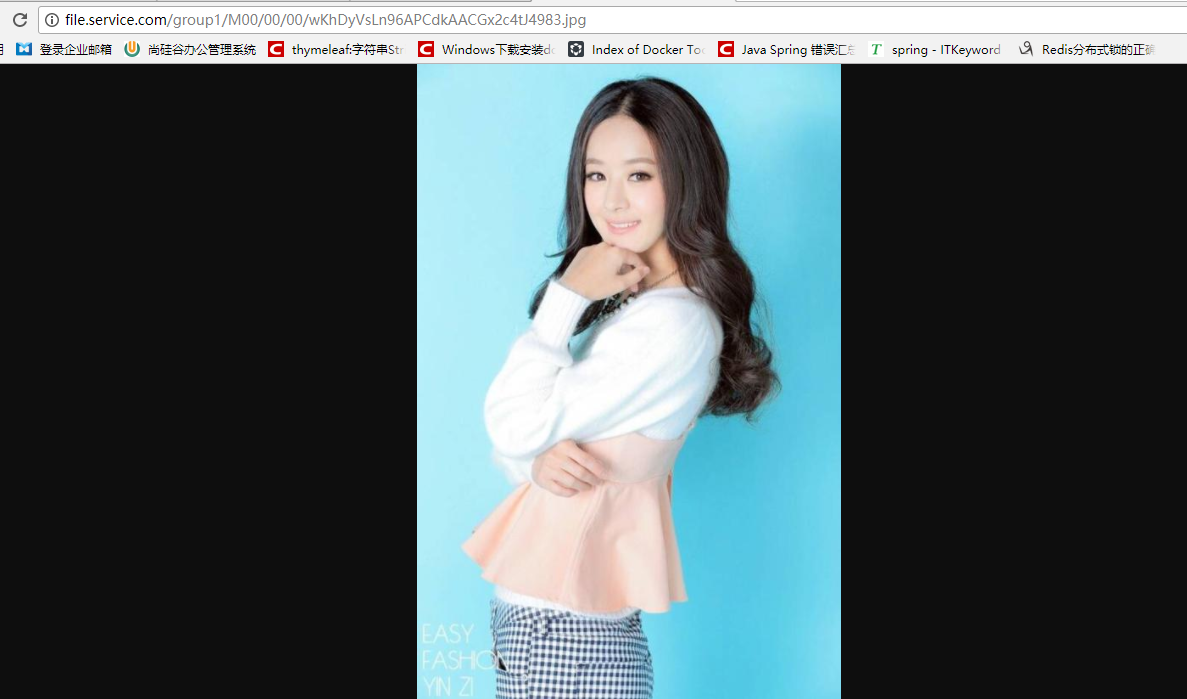
打印结果



这个打印结果实际上就是我们访问的路径，加上服务器地址我们可以拼接成一个字符串

|  |
| --- |
| http://file.gmall.com/group1/M00/00/00/wKhDyVsLn96APCdkAACGx2c4tJ4983.jpg |

直接放到浏览器去访问



上传成功！

对接到业务模块中

在修改FileUploadController的方法

服务器的地址：要实现软编码！硬编码！

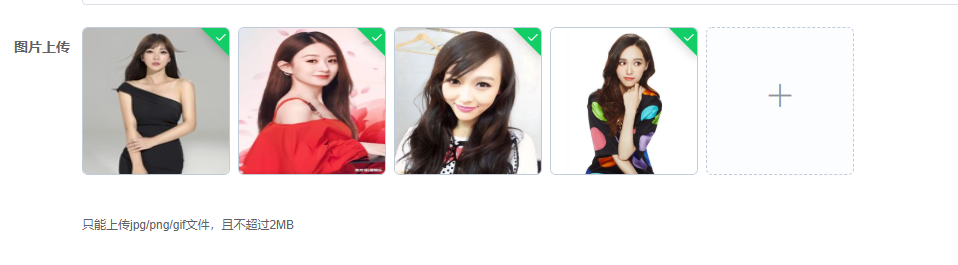
|  |
| --- |
| @Api(tags = **"上传管理接口"**) @RestController @RequestMapping(**"admin/product"**) **public class** FileUploadController {   @Value(**"${fileServer.url}"**)  **private** String **fileUrl**; *// http://192.168.67.224* @RequestMapping(**"fileUpload"**)  **public** Result<String> fileUpload(MultipartFile file) **throws** IOException, MyException {   String imgUrl = **""**;  **if** (file!=**null**){  String configFile = **this**.getClass().getResource(**"/tracker.conf"**).getFile();  ClientGlobal.*init*(configFile);  TrackerClient trackerClient=**new** TrackerClient();  TrackerServer trackerServer=trackerClient.getConnection();  StorageClient storageClient=**new** StorageClient(trackerServer,**null**);  *// String orginalFilename="e://img//zly.jpg";* String originalFilename = file.getOriginalFilename();  *// 获取文件后缀名* String extName = StringUtils.*substringAfterLast*(originalFilename, **"."**);  imgUrl = **fileUrl**; *// imgUrl=http://192.168.67.224* String[] upload\_file = storageClient.upload\_file(file.getBytes(), extName, **null**);  **for** (**int** i = 0; i < upload\_file.**length**; i++) {  String path = upload\_file[i]; *// System.out.println("s = " + s);* imgUrl+=**"/"**+path;  */\*  s = group1  s = M00/00/00/wKhD4F3PpKaAScz3AACGx2c4tJ4432.jpg  \*/* System.***out***.println(imgUrl);  }  }   *// return "http://192.168.67.224/group1/M00/00/00/wKhD4F3PpKaAScz3AACGx2c4tJ4432.jpg";* **return** Result.*ok*(imgUrl);  } } |

需要注意的是：不要将fileUrl返回去

利用@Value 标签可以引用application.yml中的值

|  |
| --- |
| **fileServer**:  **url**: http://47.93.118.241 |

测试结果：



至此我们解决了文件上传的功能。

注意：application.yml ：fastdfs的路径是有http://

测试图片上传的时候，放过自己！

tracker.conf！

# 五、商标品牌管理

## 1 实体

|  |
| --- |
| @Data @ApiModel(description = **"商标品牌"**) @TableName(**"base\_trademark"**) **public class** BaseTrademark **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"属性值"**)  @TableField(**"tm\_name"**)  **private** String **tmName**;   @ApiModelProperty(value = **"品牌logo的图片路径"**)  @TableField(**"logo\_url"**)  **private** String **logoUrl**;  } |

## 2 controller

|  |
| --- |
| @RestController @RequestMapping(**"/admin/product/baseTrademark"**) **public class** BaseTrademarkController {   @Autowired  **private** BaseTrademarkService **baseTrademarkService**;   @ApiOperation(value = **"分页列表"**)  @GetMapping(**"{page}/{limit}"**)  **public** Result index(  @ApiParam(name = **"page"**, value = **"当前页码"**, required = **true**)  @PathVariable Long page,   @ApiParam(name = **"limit"**, value = **"每页记录数"**, required = **true**)  @PathVariable Long limit) {   Page<BaseTrademark> pageParam = **new** Page<>(page, limit);  IPage<BaseTrademark> pageModel = **baseTrademarkService**.selectPage(pageParam);  **return** Result.*ok*(pageModel);  }   @ApiOperation(value = **"获取BaseTrademark"**)  @GetMapping(**"get/{id}"**)  **public** Result get(@PathVariable String id) {  BaseTrademark baseTrademark = **baseTrademarkService**.getById(id);  **return** Result.*ok*(baseTrademark);  }   @ApiOperation(value = **"新增BaseTrademark"**)  @PostMapping(**"save"**)  **public** Result save(@RequestBody BaseTrademark banner) {  **baseTrademarkService**.save(banner);  **return** Result.*ok*();  }   @ApiOperation(value = **"修改BaseTrademark"**)  @PutMapping(**"update"**)  **public** Result updateById(@RequestBody BaseTrademark banner) {  **baseTrademarkService**.updateById(banner);  **return** Result.*ok*();  }   @ApiOperation(value = **"删除BaseTrademark"**)  @DeleteMapping(**"remove/{id}"**)  **public** Result remove(@PathVariable Long id) {  **baseTrademarkService**.removeById(id);  **return** Result.*ok*();  }  } |

## 3 service

|  |
| --- |
| **public interface** BaseTrademarkService **extends** IService<BaseTrademark> {   */\*\*  \* Banner分页列表  \** ***@param pageParam*** *\** ***@return*** *\*/* IPage<BaseTrademark> selectPage(Page<BaseTrademark> pageParam);  } |
| @Service **public class** BaseTrademarkServiceImpl **extends** ServiceImpl<BaseTrademarkMapper, BaseTrademark> **implements** BaseTrademarkService {   @Autowired  **private** BaseTrademarkMapper **baseTrademarkMapper**;   @Override  **public** IPage<BaseTrademark> selectPage(Page<BaseTrademark> pageParam) {  QueryWrapper<BaseTrademark> queryWrapper = **new** QueryWrapper<>();  queryWrapper.orderByAsc(**"id"**);   IPage<BaseTrademark> page = **baseTrademarkMapper**.selectPage(pageParam, queryWrapper);  **return** page;  } } |

## 3 mapper

|  |
| --- |
| @Mapper **public interface** BaseTrademarkMapper **extends** BaseMapper<BaseTrademark> {  } |

# 六spu保存

## 1加载销售属性

创建实体类：

BaseSaleAttr

|  |
| --- |
| @Data @ApiModel(description = **"销售属性"**) @TableName(**"base\_sale\_attr"**) **public class** BaseSaleAttr **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"销售属性名称"**)  @TableField(**"name"**)  **private** String **name**;  } |

Mapper

|  |
| --- |
| @Mapper **public interface** BaseSaleAttrMapper **extends** BaseMapper<BaseSaleAttr> { } |

|  |
| --- |
| 接口  */\*\*  \* 查询所有的销售属性数据  \** ***@return*** *\*/* List<BaseSaleAttr> getBaseSaleAttrList(); |
| 实现类  @Override **public** List<BaseSaleAttr> getBaseSaleAttrList() {  **return baseSaleAttrMapper**.selectList(**null**); } |
| **在**SpuManageController**中 完成'baseSaleAttrList'**控制器  @Autowired **private** BaseSaleAttrMapper **baseSaleAttrMapper**;  */\*\*  \* 查询所有的销售属性数据  \** ***@return*** *\*/* @GetMapping(**"baseSaleAttrList"**) **public** Result<List<BaseSaleAttr>> getBaseSaleAttrList(){  List<BaseSaleAttr> spuInfoList = **manageService**.getBaseSaleAttrList();  **return** Result.*ok*(spuInfoList); } |

## 2加载品牌

|  |
| --- |
| BaseTrademarkController  */\*\*  \* 查询全部品牌  \** ***@return*** *\*/* @GetMapping(**"getTrademarkList"**) **public** Result<List<BaseTrademark>> getTrademarkList() {  List<BaseTrademark> baseTrademarkList = **baseTrademarkService**.getTrademarkList();  **return** Result.*ok*(baseTrademarkList); } |
| BaseTrademarkService  */\*\*  \* 查询全部品牌  \** ***@return*** *\*/* List<BaseTrademark> getTrademarkList(); |
| BaseTrademarkServiceImpl  @Override **public** List<BaseTrademark> getTrademarkList() {  **return baseTrademarkMapper**.selectList(**null**); } |

## 3保存后台代码

创建实体类：

在spuInfo 实体类中添加如下属性：

|  |
| --- |
| *// 销售属性集合* @TableField(exist = **false**) **private** List<SpuSaleAttr> **spuSaleAttrList**;  @TableField(exist = **false**) **private** List<SpuImage> **spuImageList**; |

销售属性表

|  |
| --- |
| @Data @ApiModel(description = **"销售属性"**) @TableName(**"spu\_sale\_attr"**) **public class** SpuSaleAttr **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"商品id"**)  @TableField(**"spu\_id"**)  **private** Long **spuId**;   @ApiModelProperty(value = **"销售属性id"**)  @TableField(**"base\_sale\_attr\_id"**)  **private** Long **baseSaleAttrId**;   @ApiModelProperty(value = **"销售属性名称(冗余)"**)  @TableField(**"sale\_attr\_name"**)  **private** String **saleAttrName**;   *// 销售属性对象集合* @TableField(exist = **false**)  List<SpuSaleAttrValue> **spuSaleAttrValueList**;  } |

销售属性值表

|  |
| --- |
| @Data @ApiModel(description = **"销售属性值"**) @TableName(**"spu\_sale\_attr\_value"**) **public class** SpuSaleAttrValue **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"商品id"**)  @TableField(**"spu\_id"**)  **private** Long **spuId**;   @ApiModelProperty(value = **"销售属性id"**)  @TableField(**"base\_sale\_attr\_id"**)  **private** Long **baseSaleAttrId**;   @ApiModelProperty(value = **"销售属性值名称"**)  @TableField(**"sale\_attr\_value\_name"**)  **private** String **saleAttrValueName**;   @ApiModelProperty(value = **"销售属性名称(冗余)"**)  @TableField(**"sale\_attr\_name"**)  **private** String **saleAttrName**;} |

商品图片实体类

|  |
| --- |
| @Data @ApiModel(description = **"Spu图片"**) @TableName(**"spu\_image"**) **public class** SpuImage **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"商品id"**)  @TableField(**"spu\_id"**)  **private** Long **spuId**;   @ApiModelProperty(value = **"图片名称"**)  @TableField(**"img\_name"**)  **private** String **imgName**;   @ApiModelProperty(value = **"图片路径"**)  @TableField(**"img\_url"**)  **private** String **imgUrl**;  } |

建立对应的mapper 文件

|  |
| --- |
| @Mapper **public interface** SpuImageMapper **extends** BaseMapper<SpuImage> { }  @Mapper **public interface** SpuSaleAttrMapper **extends** BaseMapper<SpuSaleAttr> { }  @Mapper **public interface** SpuSaleAttrValueMapper **extends** BaseMapper<SpuSaleAttrValue> { } |

|  |
| --- |
| 接口  */\*\*  \* 保存商品数据  \** ***@param spuInfo*** *\*/* **void** saveSpuInfo(SpuInfo spuInfo); |
| @Override @Transactional **public void** saveSpuInfo(SpuInfo spuInfo) { *// spuInfo 商品表* **spuInfoMapper**.insert(spuInfo); *// spuImage 商品图片表* List<SpuImage> spuImageList = spuInfo.getSpuImageList();  **if** (spuImageList != **null** && spuImageList.size() > 0) {  **for** (SpuImage spuImage : spuImageList) {  spuImage.setSpuId(spuInfo.getId());  **spuImageMapper**.insert(spuImage);  }  } *// spuSaleAttr 销售属性表* List<SpuSaleAttr> spuSaleAttrList = spuInfo.getSpuSaleAttrList();  **if** (spuSaleAttrList != **null** && spuSaleAttrList.size() > 0) {  **for** (SpuSaleAttr spuSaleAttr : spuSaleAttrList) {  spuSaleAttr.setSpuId(spuInfo.getId());  **spuSaleAttrMapper**.insert(spuSaleAttr);   *// spuSaleAttrValue 销售属性值表* List<SpuSaleAttrValue> spuSaleAttrValueList = spuSaleAttr.getSpuSaleAttrValueList();  **if** (spuSaleAttrValueList != **null** && spuSaleAttrValueList.size() > 0) {  **for** (SpuSaleAttrValue spuSaleAttrValue : spuSaleAttrValueList) {  spuSaleAttrValue.setSpuId(spuInfo.getId());  spuSaleAttrValue.setSaleAttrName(spuSaleAttr.getSaleAttrName());  **spuSaleAttrValueMapper**.insert(spuSaleAttrValue);  }  }  }  }  } |
| */\*\*  \* 保存spu  \** ***@param spuInfo*** *\** ***@return*** *\*/* @PostMapping(**"saveSpuInfo"**) **public** Result saveSpuInfo(@RequestBody SpuInfo spuInfo){  *// 调用服务层的保存方法* **manageService**.saveSpuInfo(spuInfo);  **return** Result.*ok*(); } |