**谷粒商城**

**版本：V 1.0**

**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）。

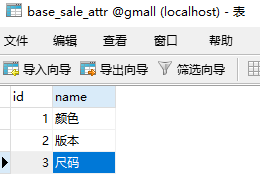


销售属性与平台属性

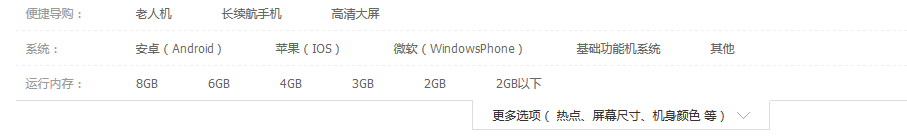
销售属性，就是商品详情页右边，可以通过销售属性来定位一组spu下的哪款sku。可以让当前的商品详情页，跳转到自己的“兄弟”商品。

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。

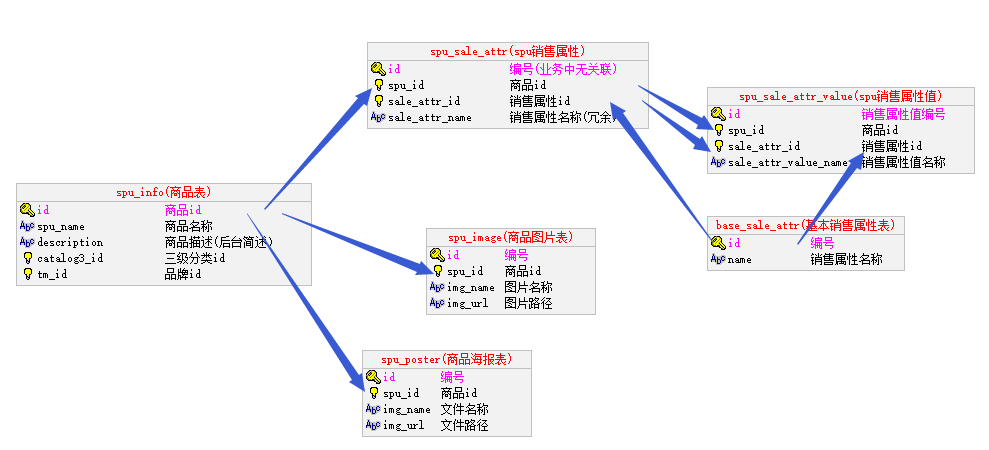
## 2 SKU与SPU的图片资源

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。

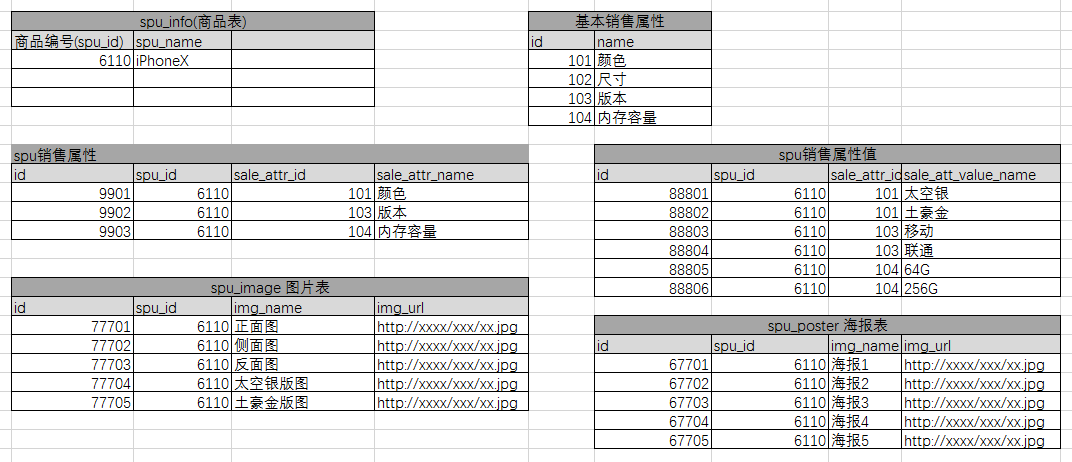
而海报几乎是所有SPU下的SKU都一样。

## 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、列表查询功能开发

## 后台代码

### bean

|  |
| --- |
| **public class** SpuInfo **implements** Serializable { @Column @Id @GeneratedValue(strategy = GenerationType.***IDENTITY***) **private** String **id**;  @Column **private** String **spuName**;  @Column **private** String **description**;  @Column **private** String **catalog3Id**;  } |

### mapper

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

### ManageService

|  |
| --- |
| List<SpuInfo> getSpuInfoList(SpuInfo spuInfo); |
| ManageServiceImpl  @Override **public** List<SpuInfo> getSpuInfoList(SpuInfo spuInfo) {  **return spuInfoMapper**.select(spuInfo); } |

### controller

|  |
| --- |
| @Controller @CrossOrigin **public class** SpuManageController {  @RequestMapping(**"spuList"**) @ResponseBody **public** List<SpuInfo> spuList(String catalog3Id){  SpuInfo spuInfo = **new** SpuInfo();  spuInfo.setCatalog3Id(catalog3Id);  List<SpuInfo> spuInfoList = **manageService**.getSpuInfoList(spuInfo);  **return** spuInfoList; }  } |

# 三、spu的保存功能中的图片上传

## 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。

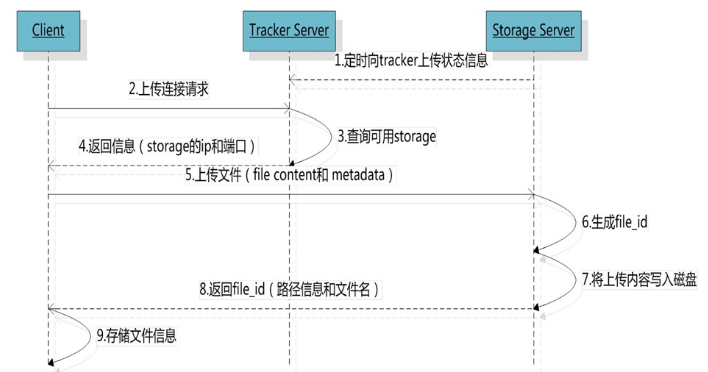
实现方式就是nginx+FastDFS

### 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}！

### 安装步骤参见《FastDFS安装说明》

### 利用Java客户端调用FastDFS

服务器安装完毕后，咱们通过Java调用fastdfs。

**加载Maven依赖**

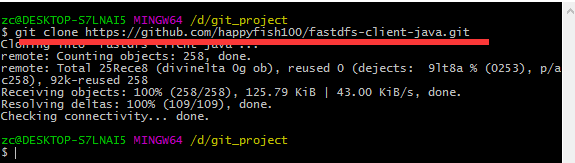
fastdfs 没有在中心仓库中提供获取的依赖坐标。

只能自己通过源码方式编译，打好jar 包，安装到本地仓库。

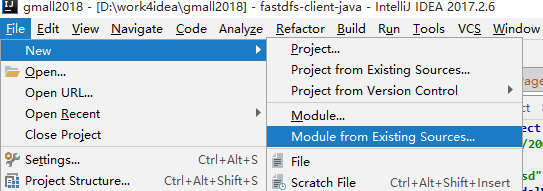
官方仓库地址：

<https://github.com/happyfish100/fastdfs-client-java>

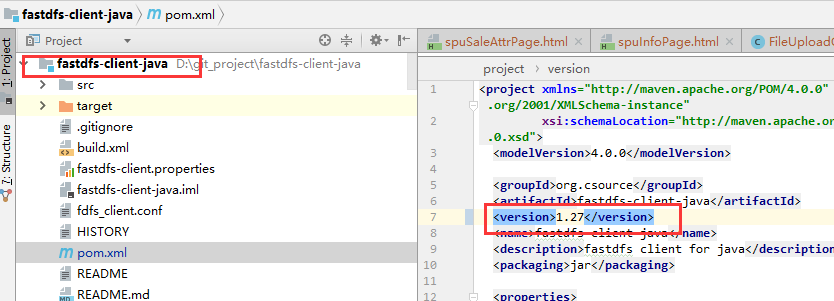
在**项目的根目录**下使用git clone 命令，将代码下载



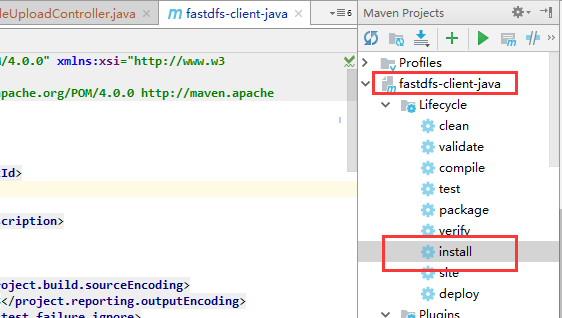
直接用idea 直接把这个源码作为模块导入工程



别的不用改，只把pom.xml中的版本改成1.27。



然后右边 执行install 就好了

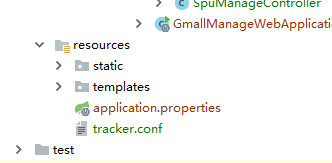


安装好了 ，在gmall-manage-web模块就可以直接使用这个坐标了。

|  |
| --- |
| *<!--添加fdfs依赖-->* <**dependency**>  <**groupId**>org.csource</**groupId**>  <**artifactId**>fastdfs-client-java</**artifactId**>  <**version**>1.27</**version**> </**dependency**> |

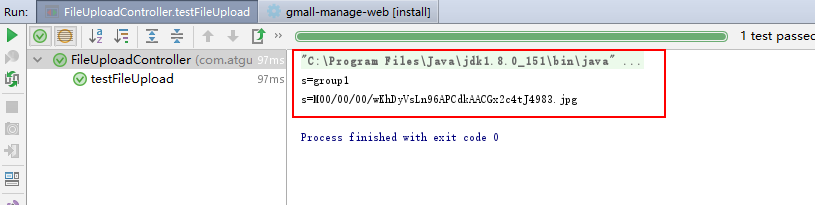
然后可以进行一下**上传的测试**

在项目resource中添加tracker.conf 配置文件



|  |
| --- |
| **tracker\_server=192.168.67.162:22122**  # 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);  } } |

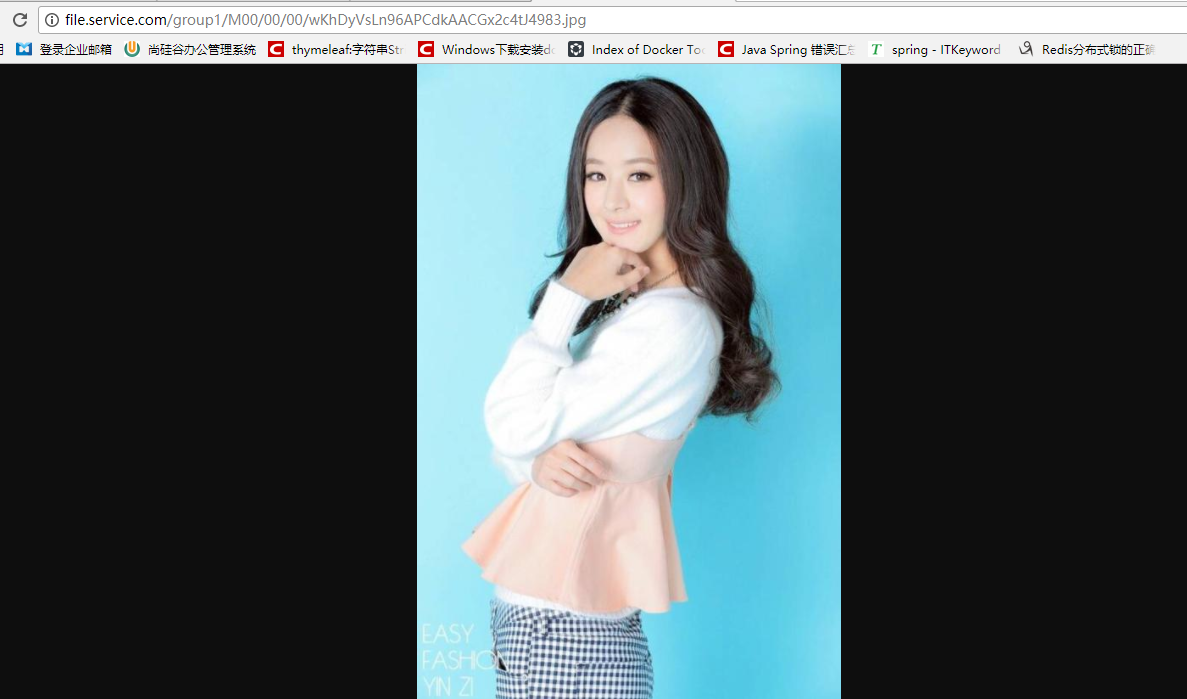
打印结果



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！

注意：控制器需要跨域！ @CrossOrigin

|  |
| --- |
| @Value("${fileServer.url}") String fileUrl;  @RequestMapping(value = "fileUpload",method = RequestMethod.*POST*) public String fileUpload(@RequestParam("file") MultipartFile file) throws IOException, MyException {  String imgUrl=fileUrl;  if(file!=null){  System.*out*.println("multipartFile = " + file.getName()+"|"+file.getSize());  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 filename= file.getOriginalFilename();  String extName = StringUtils.*substringAfterLast*(filename, ".");   String[] upload\_file = storageClient.upload\_file(**file.getBytes()**, extName, null);  imgUrl=fileUrl ;  for (int i = 0; i < upload\_file.length; i++) {  String path = upload\_file[i];  imgUrl+="/"+path;  }   }   return imgUrl; } |

需要注意的是：不要将fileUrl返回去

利用@Value 标签可以引用application.properties中的值

|  |
| --- |
| **fileServer.url**=**http://192.168.67.201** |

测试结果：



至此我们解决了文件上传的功能。

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

测试图片上传的时候，放过自己！

tracker.conf！

os7：设置开机启动nginx，fdfs ，一定失败！

解决方案：放过自己，手动！

**pid logs/nginx.pid;（把#删除）**

http://www.cnblogs.com/yufeng218/p/8215381.html

**chmod +x /etc/rc.d/rc.local**

<https://www.cnblogs.com/yufeng218/p/8215421.html>

# 四、 spu保存

## 加载销售属性

创建实体类：

BaseSaleAttr

|  |
| --- |
| **public class** BaseSaleAttr **implements** Serializable {  @Id  @Column  String **id** ;   @Column  String **name**;  } |

Mapper

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

|  |
| --- |
| 接口  *// 查询基本销售属性表* List<BaseSaleAttr> getBaseSaleAttrList(); |
| 实现类  @Override **public** List<BaseSaleAttr> getBaseSaleAttrList() {  **return baseSaleAttrMapper**.selectAll(); } |
| **在manageController 中 完成'baseSaleAttrList'**控制器  @RequestMapping(**"baseSaleAttrList"**) @ResponseBody **public** List<BaseSaleAttr> getBaseSaleAttrList(){  **return manageService**.getBaseSaleAttrList(); } |

## 保存后台代码

创建实体类：

在spuInfo 实体类中添加如下属性：

|  |
| --- |
| @Transient **private** List<SpuSaleAttr> **spuSaleAttrList**; @Transient **private** List<SpuImage> **spuImageList**; |

销售属性表

|  |
| --- |
| **public class** SpuSaleAttr **implements** Serializable{   @Id  @Column  String **id** ;   @Column  String **spuId**;   @Column  String **saleAttrId**;   @Column  String **saleAttrName**;    @Transient  List<SpuSaleAttrValue> **spuSaleAttrValueList**;  } |

销售属性值表

|  |
| --- |
| **public class** SpuSaleAttrValue **implements** Serializable {   @Id  @Column  String **id** ;   @Column  String **spuId**;   @Column  String **saleAttrId**;   @Column  String **saleAttrValueName**;   @Transient  String **isChecked**;  } |

商品图片实体类

|  |
| --- |
| **public class** SpuImage **implements** Serializable{  @Column  @Id  **private** String **id**;  @Column  **private** String **spuId**;  @Column  **private** String **imgName**;  @Column  **private** String **imgUrl**;  } |

建立对应的mapper 文件

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

|  |
| --- |
| 接口  **public void** saveSpuInfo(SpuInfo spuInfo); |
| @Override **public void** saveSpuInfo(SpuInfo spuInfo) {  *// 什么情况下是保存，什么情况下是更新 spuInfo* **if** (spuInfo.getId()==**null** || spuInfo.getId().length()==0){  *//保存数据* spuInfo.setId(**null**);  **spuInfoMapper**.insertSelective(spuInfo);  }**else** {  **spuInfoMapper**.updateByPrimaryKeySelective(spuInfo);  }   *// spuImage 图片列表 先删除，在新增  // delete from spuImage where spuId =?* SpuImage spuImage = **new** SpuImage();  spuImage.setSpuId(spuInfo.getId());  **spuImageMapper**.delete(spuImage);   *// 保存数据，先获取数据* List<SpuImage> spuImageList = spuInfo.getSpuImageList();  **if** (spuImageList!=**null** && spuImageList.size()>0){  *// 循环遍历* **for** (SpuImage image : spuImageList) {  image.setId(**null**);  image.setSpuId(spuInfo.getId());  **spuImageMapper**.insertSelective(image);  }  }  *// 销售属性 删除，插入* SpuSaleAttr spuSaleAttr = **new** SpuSaleAttr();  spuSaleAttr.setSpuId(spuInfo.getId());  **spuSaleAttrMapper**.delete(spuSaleAttr);   *// 销售属性值 删除，插入* SpuSaleAttrValue spuSaleAttrValue = **new** SpuSaleAttrValue();  spuSaleAttrValue.setSpuId(spuInfo.getId());  **spuSaleAttrValueMapper**.delete(spuSaleAttrValue);   *// 获取数据* List<SpuSaleAttr> spuSaleAttrList = spuInfo.getSpuSaleAttrList();  **if** (spuSaleAttrList!=**null** && spuSaleAttrList.size()>0){  *// 循环遍历* **for** (SpuSaleAttr saleAttr : spuSaleAttrList) {  saleAttr.setId(**null**);  saleAttr.setSpuId(spuInfo.getId());  **spuSaleAttrMapper**.insertSelective(saleAttr);   *// 添加销售属性值* List<SpuSaleAttrValue> spuSaleAttrValueList = saleAttr.getSpuSaleAttrValueList();  **if** (spuSaleAttrValueList!=**null** && spuSaleAttrValueList.size()>0){  *// 循环遍历* **for** (SpuSaleAttrValue saleAttrValue : spuSaleAttrValueList) {  saleAttrValue.setId(**null**);  saleAttrValue.setSpuId(spuInfo.getId());  **spuSaleAttrValueMapper**.insertSelective(saleAttrValue);  }  }   }  } } |
| @RequestMapping(**"saveSpuInfo"**) @ResponseBody **public** String saveSpuInfo(@RequestBody SpuInfo spuInfo){  **manageService**.saveSpuInfo(spuInfo);  **return "OK"**; } |