谷粒商城

版本：V 2.0

# pms的业务简介

## 1 pms整体业务简介



**当前的位置**

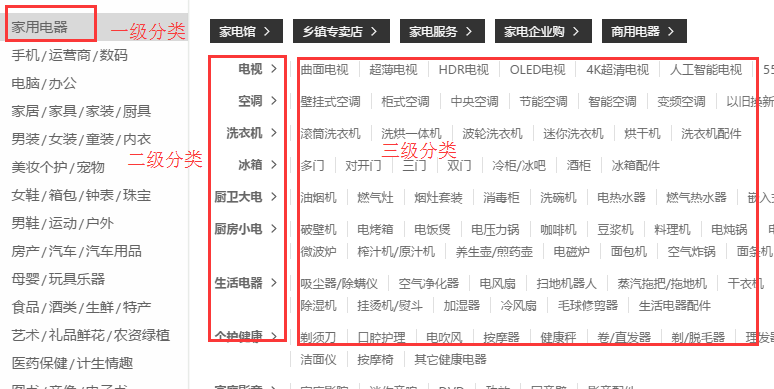
|  |
| --- |
| 1 三级分类的查询 |
| 2 商品的平台属性列表的增删改查 |
| 3 商品spu的添加  Spu列表查询  spu的销售属性、属性值、Fastdfs图片上传 |
| 4 商品sku的添加  sku信息、sku关联的销售属性、sku关联的平台属性、sku图片 |

## pms的数据结构设计思路

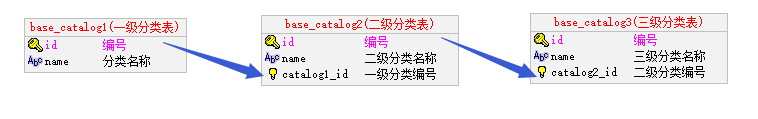
### 2.1基本信息—分类

一般情况可以分为两级或者三级。咱们的项目一共分为三级，即一级分类、二级分类、三级分类。

比如：家用电器是一级分类，电视是二级分类，那么超薄电视就是三级分类。



数据库结构

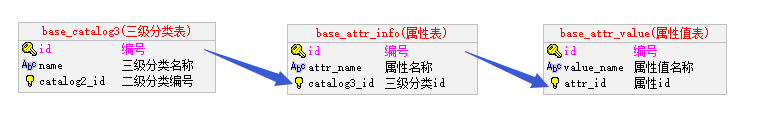


### 2.2基本信息—平台属性

平台属性和平台属性值



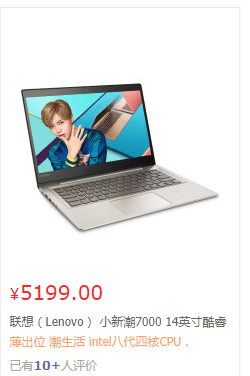
平台属性和平台属性值主要用于商品的检索，每个三级分类对应的**属性**都不同。



而每个商品对应的每种属性都有对应的**属性值。**

比如电脑整机的一级分类下，有笔记本、游戏本、台式机、一体机的二级分类。

笔记本这个二级分类又包含了处理器、屏幕尺寸、内存容量、硬盘容量、显卡类别这些属性。

那么针对联想某个型号的笔记本，它作为笔记本这种分类，每个分类属性都有对应的值，cpu(属性)是i7(属性值)的，内存(属性)是8G(属性值)的，屏幕尺寸(属性)是14寸(属性值)的。

### 2.3基本信息—销售属性

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。



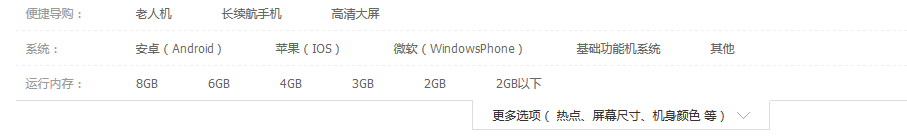
销售属性与平台属性

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。



### 2.3销售属性与平台属性的关系

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，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。整个平台的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。





### 2.4基本信息—spu与 sku

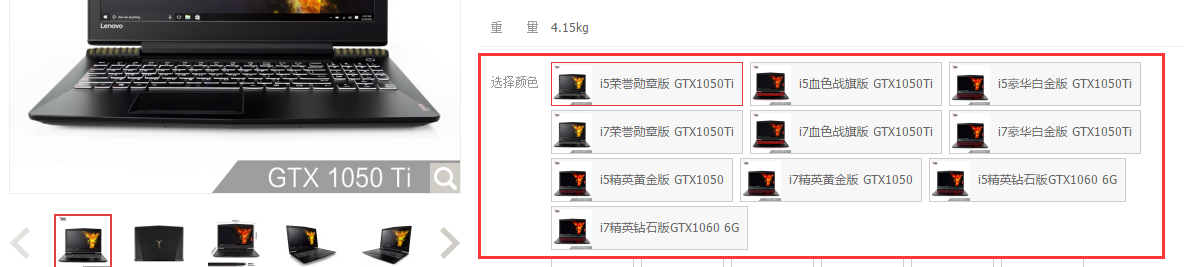
SPU(Standard Product Unit)：标准化产品单元。是商品信息聚合的最小单位，是一组**可复用、易检索**的标准化信息的集合，该集合描述了一个产品的特性。

SKU=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）。

首先通过检索搜索出来的商品列表中，每个商品都是一个sku。每个sku都有自己独立的库存数。也就是说每一个商品详情展示都是一个sku。

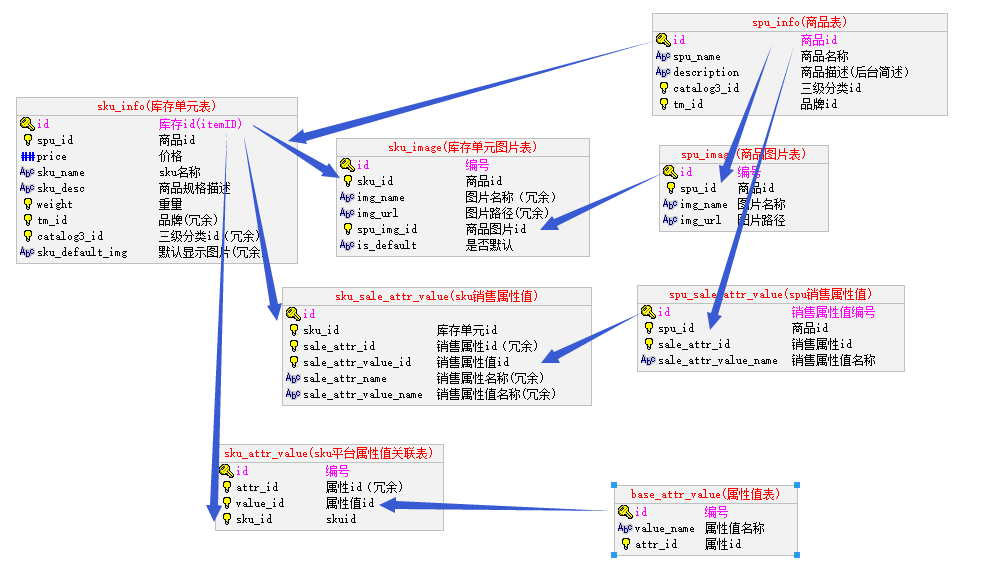
那spu又是干什么的呢？



如上图，一般的电商系统你点击进去以后，都能看到这个商品关联了其他好几个类似的商品，而且这些商品很多的信息都是共用的，比如商品图片，海报、销售属性等。

那么系统是靠什么把这些sku识别为一组的呢，那是这些sku都有一个公用的spu信息。而它们公共的信息，都放在spu信息下。

所以，sku与spu的结构如下：

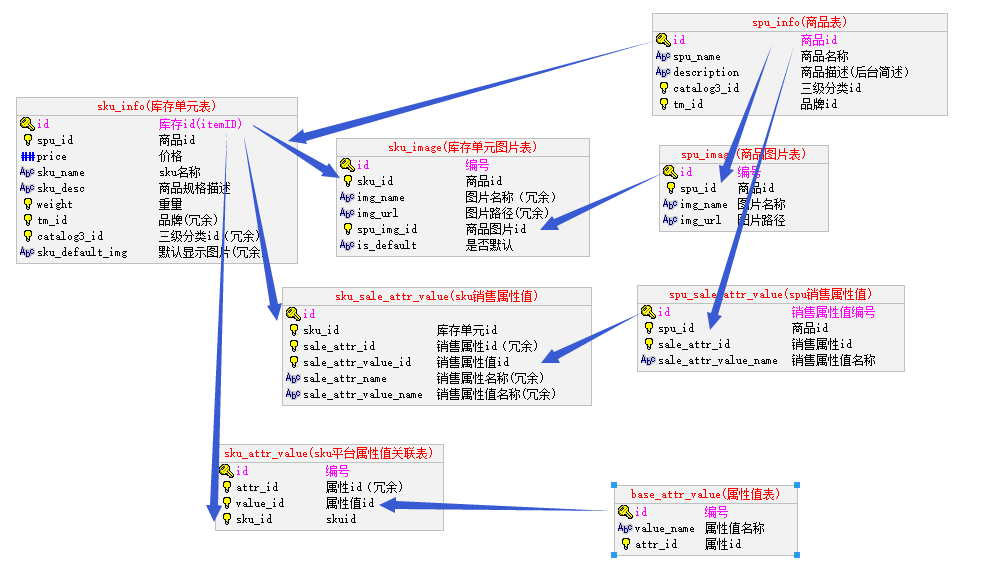


图中有两个图片信息表，其中spu\_image表示整个spu相关下的所有图片信息，而sku\_image表示这个spu下的某个sku使用的图片。sku\_image中的图片是从spu\_image中选取的。

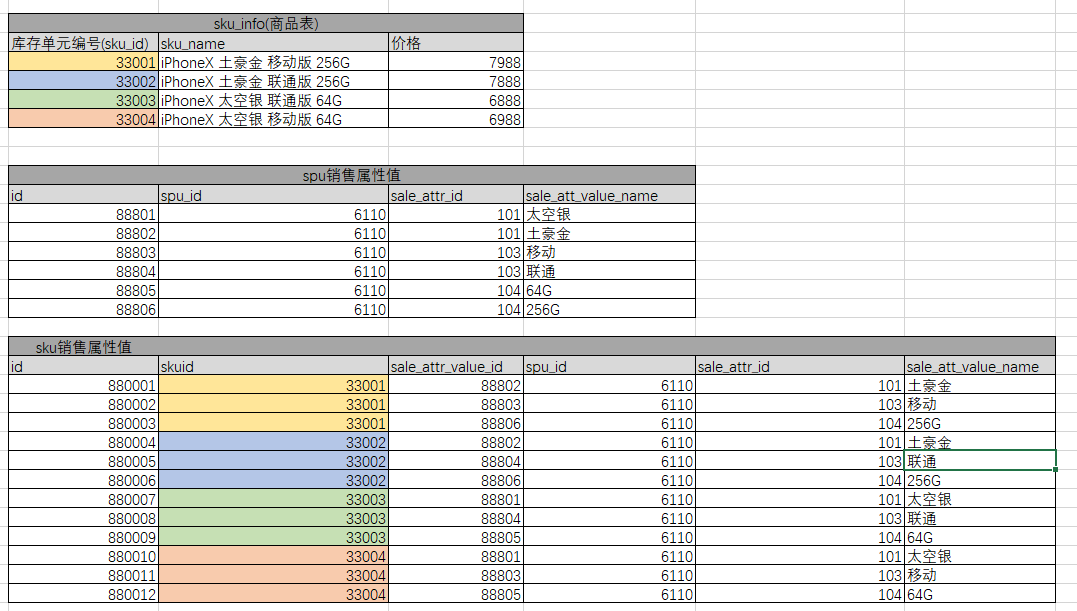
但是由于一个spu下的所有sku的海报都是一样，所以只存一份spu\_poster就可以了。

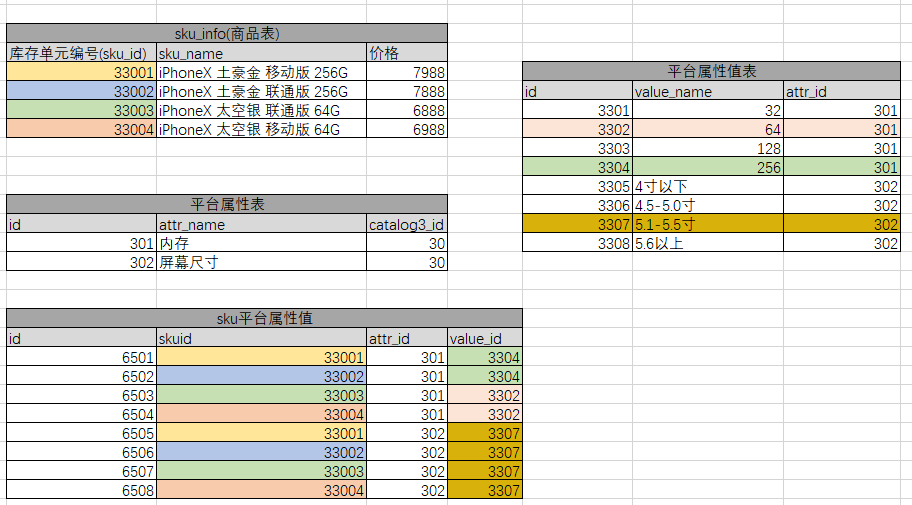
## 3 pms数据模型

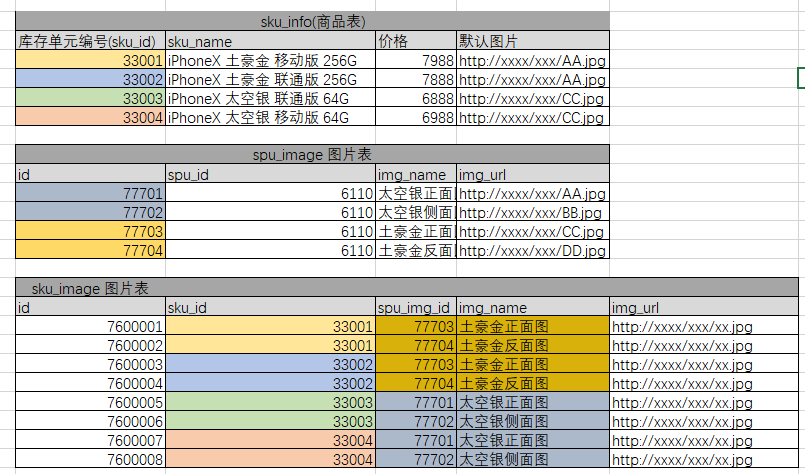
根据以上的需求，以此将关联的数据库表结构设计为如下：



数据示例：



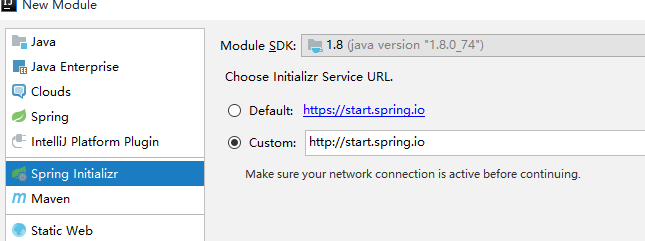


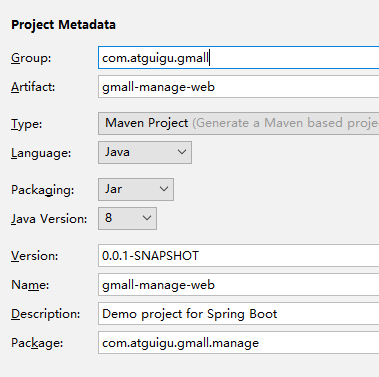


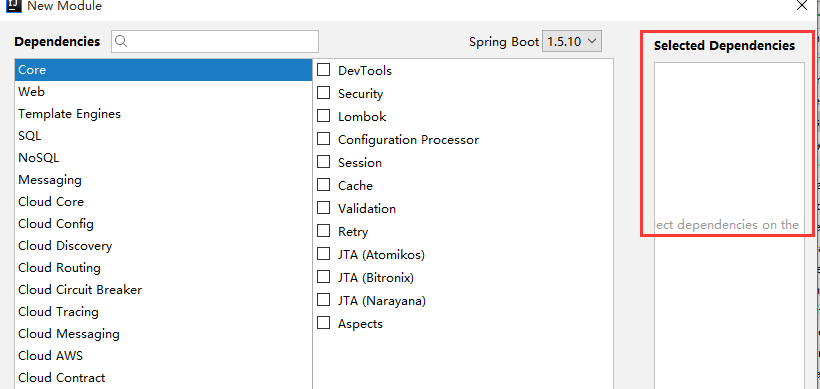
# 第二章 后台管理模块开发

## 1 后台的manage-web模块搭建

1.1 建module

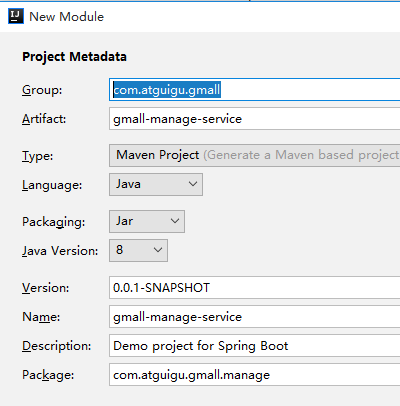




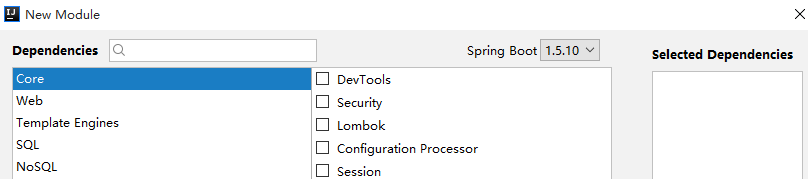


## 2 后台的manage-service模块的搭建

创建manage-service模块



**不用添加任何依赖**



## 后台的springboot的配置文件

配置manage-web的application.properties

|  |
| --- |
| server.port=8083  spring.thymeleaf.cache=false  spring.thymeleaf.mode=LEGACYHTML5  spring.dubbo.application.name=manage-web spring.dubbo.registry.protocol=zookeeper spring.dubbo.registry.address=192.168.67.159:2181 spring.dubbo.base-package=com.atguigu.gmall spring.dubbo.protocol.name=dubbo spring.dubbo.consumer.timeout=10000 spring.dubbo.consumer.check=false |

配置manage-service的application.properties

|  |
| --- |
| server.port=8073  logging.level.root=debug  spring.dubbo.application.name=manage-service spring.dubbo.registry.protocol=zookeeper spring.dubbo.registry.address=192.168.67.159:2181 spring.dubbo.base-package=com.atguigu.gmall spring.dubbo.protocol.name=dubbo spring.datasource.url=jdbc:mysql://59.110.141.236:3306/gmall?characterEncoding=UTF-8 spring.datasource.username=root spring.datasource.password=123123 spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

# 第三章 文件服务器(存储商品图片或者视频文件)

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。

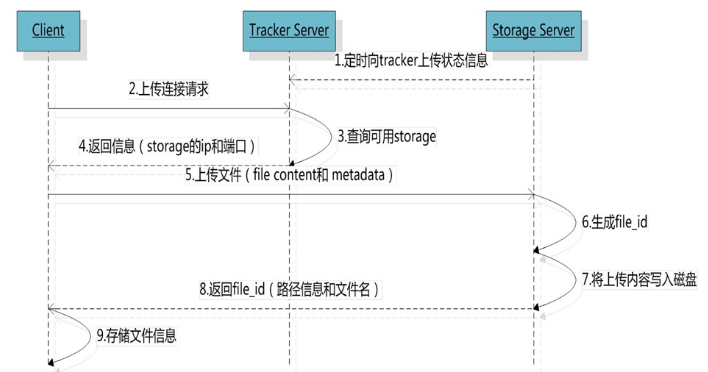
实现方式就是nginx+FastDFS

## 1 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。

## 2 FastDFS上传下载的流程



## 3 安装步骤参见《FastDFS安装说明》

## 4 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

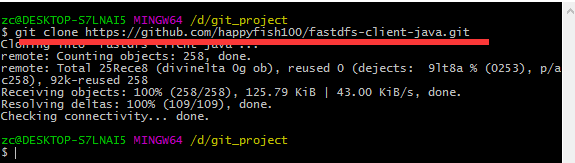
**加载Maven依赖**

fastdfs 没有在中心仓库中提供获取的依赖坐标。

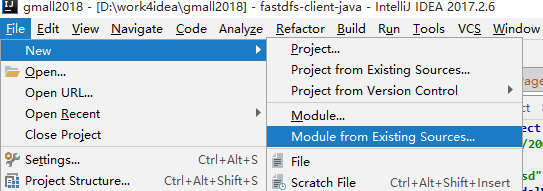
只能自己通过源码方式编译，打好jar 包，安装到本地仓库。

官方仓库地址：

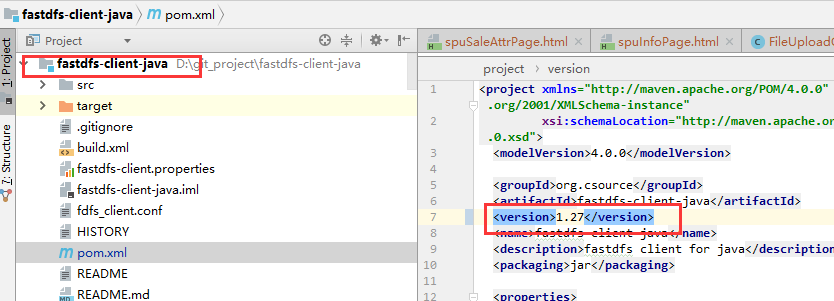
[https://github.com/happyfish100/fastdfs-client-java](https://github.com/happyfish100/fastdfs-client-java" \t "_blank)



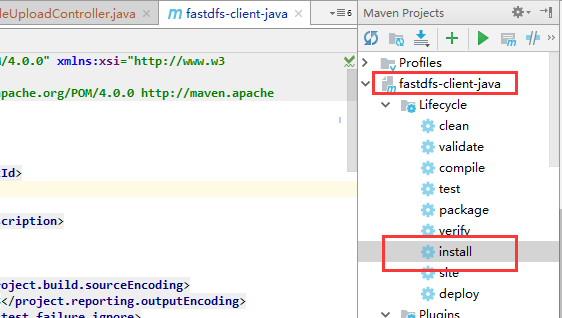
直接用idea 直接把这个源码作为模块导入工程



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 ，别的模块就可以直接使用这个坐标了。

|  |
| --- |
| <**groupId**>org.csource</**groupId**> <**artifactId**>fastdfs-client-java</**artifactId**> <**version**>1.27</**version**> |

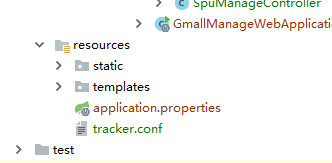
而这个fastdfs-client-java模块可以从idea 中删除。

然后可以进行一下**上传的测试**

@RunWith(SpringRunner.**class**)  
@SpringBootTest

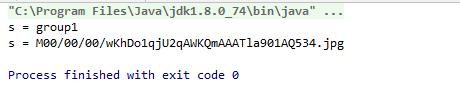
|  |
| --- |
| @Test **public void** textFileUpload() **throws** IOException, MyException {  String file = **this**.getClass().getResource(**"/tracker.conf"**).getFile();  ClientGlobal.*init*(file);  TrackerClient trackerClient=**new** TrackerClient();  TrackerServer trackerServer=trackerClient.getConnection();  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);  }  } |

加入tracker.conf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 |

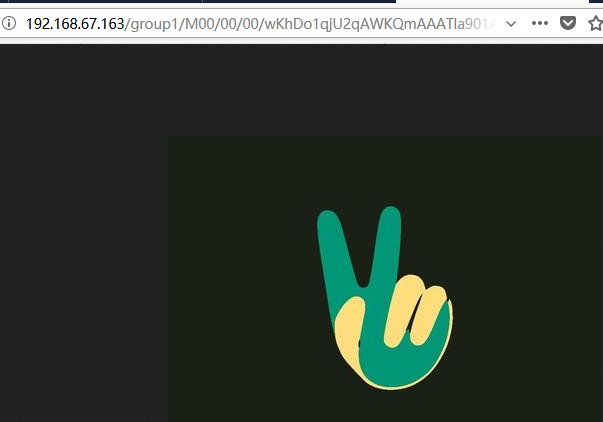
打印结果



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://192.168.67.163/group1/M00/00/00/wKhDo1qjU2qAWKQmAAATla901AQ534.jpg |

直接放到浏览器去访问



上传成功！

对接到业务模块中

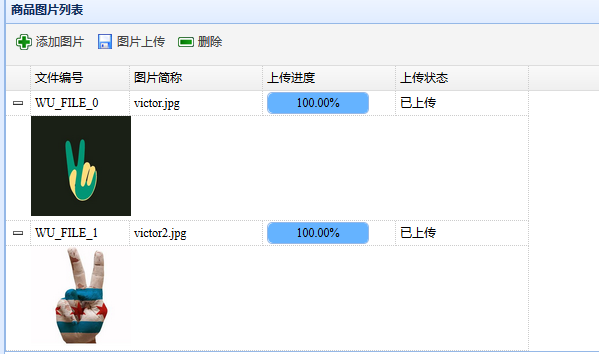
在修改FileUploadController的方法

|  |
| --- |
| @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; } |

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

|  |
| --- |
| **fileServer.url**=**http://192.168.67.163** |

测试结果：



至此我们解决了文件上传的功能。