谷粒商城

版本：V 1.0

# 电商的业务简介

## 1 整体业务简介



**当前的位置**

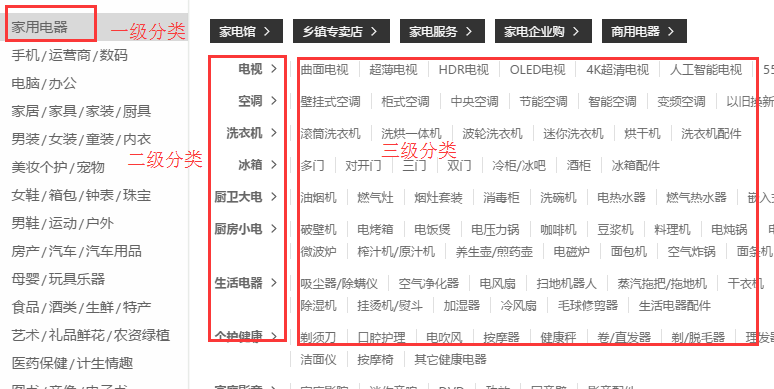
|  |  |
| --- | --- |
| 首页 | 静态页面，包含了商品分类，搜索栏，商品广告位。 |
| 全文搜索 | 通过搜索栏填入的关键字进行搜索，并列表展示 |
| 分类查询 | 根据首页的商品类目进行查询 |
| 商品详情 | 商品的详细信息展示 |
| 购物车 | 将有购买意向的商品临时存放的地方 |
| 单点登录 | 用户统一登录的管理 |
| 结算 | 将购物车中勾选的商品初始化成要填写的订单 |
| 下单 | 填好的订单提交 |
| 支付服务 | 下单后，用户点击支付，负责对接第三方支付系统。 |
| 订单服务 | 负责确认订单是否付款成功，并对接仓储物流系统。 |
| 仓储物流 | 独立的管理系统，负责商品的库存。 |
| 后台管理 | 主要维护类目、商品、库存单元、广告位等信息。 |

## 商品管理

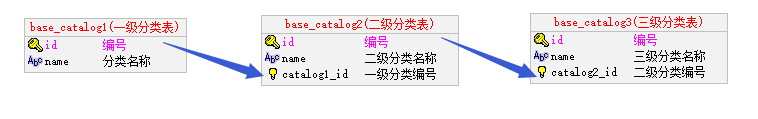
### 2.1 基本信息—分类

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

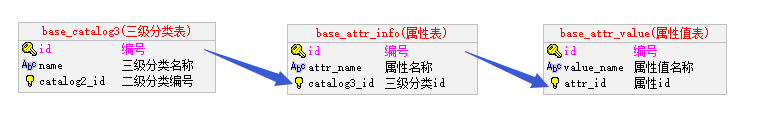


### 2.2 基本信息—平台属性

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。

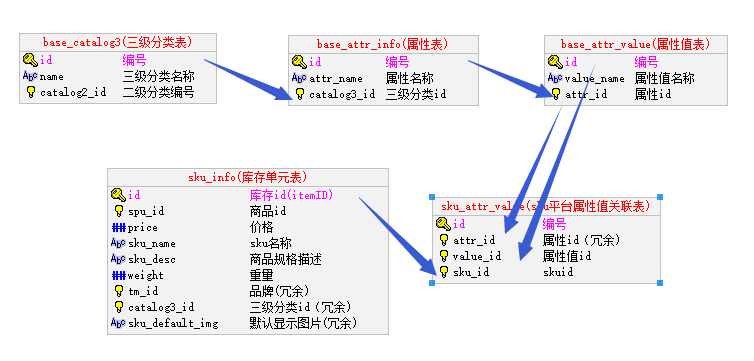


手机 属性，属性值

电脑 属性，属性值

手机与电脑 属性，属性值一样么？

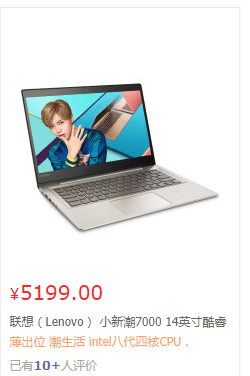
而每个**商品**对应的每种**属性**都有对应的**属性值。**



比如

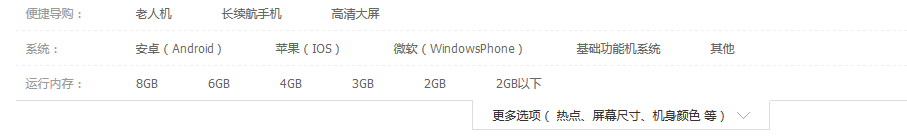
电脑整机的一级分类下，有笔记本、游戏本、台式机、一体机的二级分类。

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 商品信息—销售属性与平台属性

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。





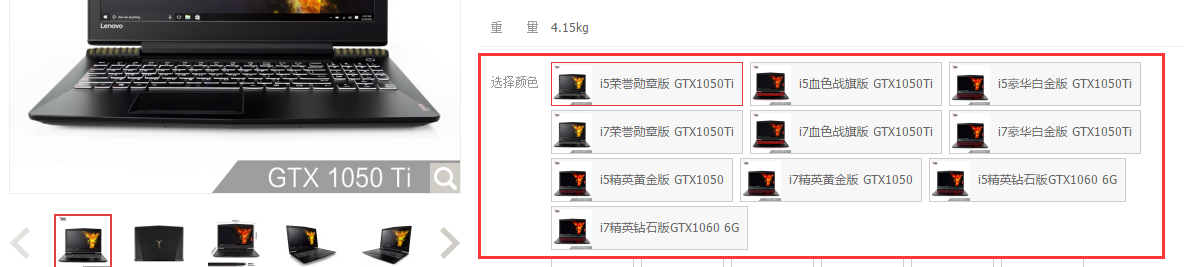
### 2.4 基本信息—spu与 sku

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。

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。

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：

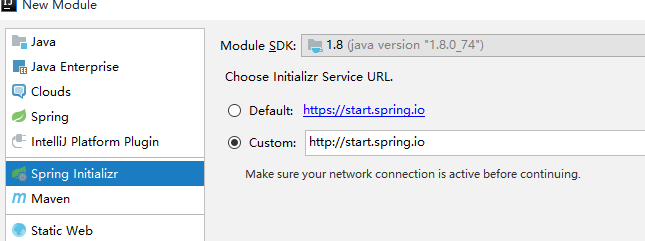
图中有两个图片信息表，其中spu\_image表示整个spu相关下的所有图片信息，而sku\_image表示这个spu下的某个sku使用的图片。sku\_image中的图片是从spu\_image中选取的。

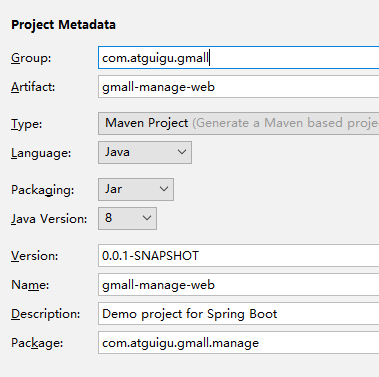
但是由于一个spu下的所有sku的海报都是一样，所以只存一份spu\_poster就可以了。

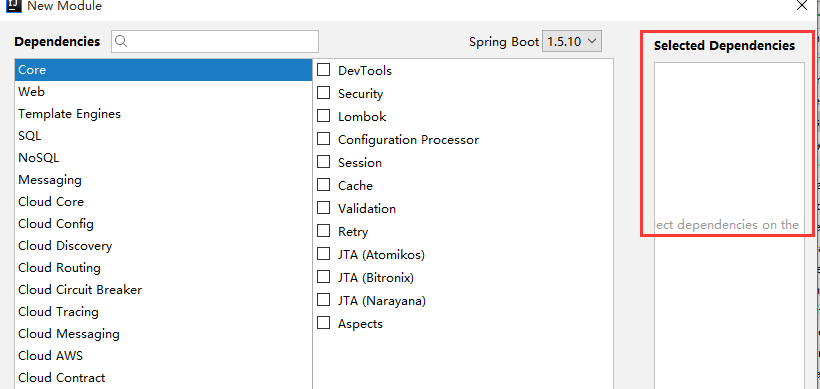
# 第二章 后台管理模块开

## 1 后台的Web模块搭建

### 1.1 建module







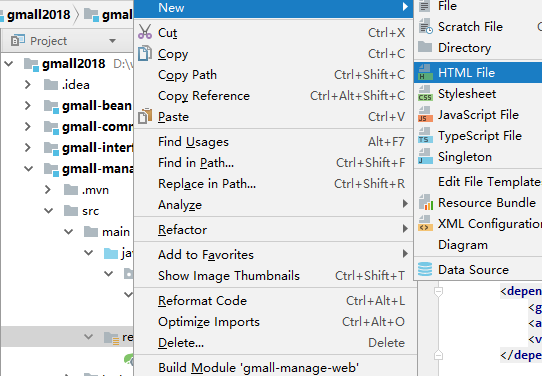
pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-manage-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-manage-web</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-parent</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>  <dependencies>  <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-web-util</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

### 1.2 页面开发

如何在web模块中加入网页：

在resources中的templates文件夹，加入一个index.html的页面



index.html代码

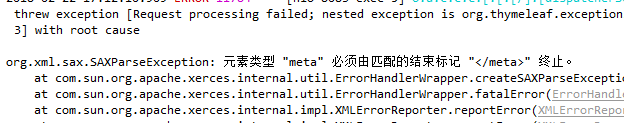
|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> hello world ! </**body**> </**html**> |

这时如果你启动服务，然后用浏览器访问其实是访问不到的。index.html放到templates 只是一个模板，必须要经过controller层渲染，才能被访问。

增加ManageController

|  |
| --- |
| @Controller **public class** ManageController {  @RequestMapping(value = **"index"** )  **public** String index(){  **return "index"**;  } } |

如果后台出现



这是因为springboot默认使用themeleaf渲染，而这种模板语言对html的标签约束非常严格，所有标签必须有开有闭，比如 <br><br/> 或者<br/> 是可以的，而<br> 是会报错的。

不能使用单行标签！<br> <hr> 等。

解决页面松校验的方式

application.properties中增加

|  |
| --- |
| **server.port**=**8082** |

在gmall-parent模块中的pom.xml增加依赖管理

|  |
| --- |
| <**nekohtml.version**>1.9.20</**nekohtml.version**> <**xml-apis.version**>1.4.01</**xml-apis.version**> <**batik-ext.version**>1.9.1</**batik-ext.version**>  <**dependency**>  <**groupId**>net.sourceforge.nekohtml</**groupId**>  <**artifactId**>nekohtml</**artifactId**>  <**version**>${nekohtml.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>xml-apis</**groupId**>  <**artifactId**>xml-apis</**artifactId**>  <**version**>${xml-apis.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.apache.xmlgraphics</**groupId**>  <**artifactId**>batik-ext</**artifactId**>  <**version**>${batik-ext.version}</**version**> </**dependency**> |

在web-util模块中的pom.xml增加

|  |
| --- |
| <**dependency**>  <**groupId**>net.sourceforge.nekohtml</**groupId**>  <**artifactId**>nekohtml</**artifactId**> </**dependency**> <**dependency**>  <**groupId**>xml-apis</**groupId**>  <**artifactId**>xml-apis</**artifactId**> </**dependency**> <**dependency**>  <**groupId**>org.apache.xmlgraphics</**groupId**>  <**artifactId**>batik-ext</**artifactId**> </**dependency**> |

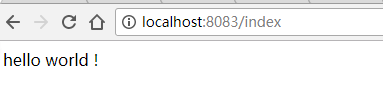
请在application.properties中增加

|  |
| --- |
| **spring.thymeleaf.cache**=**false**  **spring.thymeleaf.mode**=**LEGACYHTML5** |

这个是关闭了springboot的页面缓存，如果不关闭会影响开发调试过程中的热部署，如果系统上线可以再把这个缓存打开。

热部署：ctrl+shift+F9

然后测试页面



## 2 属性管理功能

### 2.1 分类信息及属性的查询

#### 2.1.1 添加实体类

首先是 bean,以下代码由于节省篇幅没有生产getter,setter方法，请自行用idea生成。

BaseCatalog1

|  |
| --- |
| **public class** BaseCatalog1 **implements** Serializable {  @Id  @Column  **private** String **id**;  @Column  **private** String **name**; } |

BaseCatalog2

|  |
| --- |
| **public class** BaseCatalog2 **implements** Serializable {  @Id  @Column  **private** String **id**;  @Column  **private** String **name**;  @Column  **private** String **catalog1Id**; } |

BaseCatalog3

|  |
| --- |
| **public class** BaseCatalog3 **implements** Serializable {  @Id  @Column  **private** String **id**;  @Column  **private** String **name**;  @Column  **private** String **catalog2Id**; } |

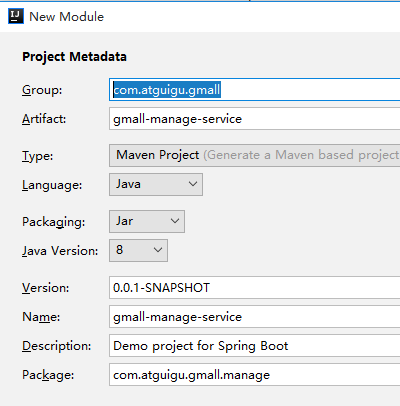
BaseAttrInfo

|  |
| --- |
| **public class** BaseAttrInfo **implements** Serializable {  @Id  @Column  **private** String **id**;  @Column  **private** String **attrName**;  @Column  **private** String **catalog3Id**;  } |

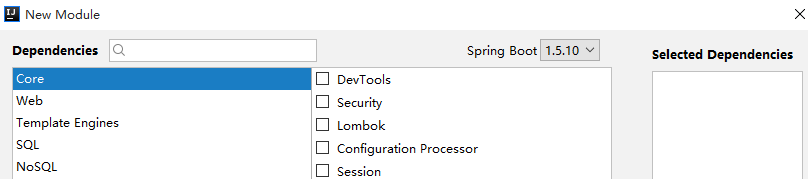
BaseAttrValue

|  |
| --- |
| **public class** BaseAttrValue **implements** Serializable {  @Id  @Column  **private** String **id**;  @Column  **private** String **valueName**;  @Column  **private** String **attrId**; } |

#### 2.1.2 创建manage-service模块



**不用添加任何依赖**



pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-manage-service</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-manage-service</name>   <parent>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-parent</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>  <dependencies>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-service-util</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> </project> |

manage-service中创建Mapper

BaseCatalog1Mapper

|  |
| --- |
| **public interface** BaseCatalog1Mapper **extends** Mapper<BaseCatalog1> { } |

BaseCatalog2Mapper

|  |
| --- |
| **public interface** BaseCatalog2Mapper **extends** Mapper<BaseCatalog2> { } |

BaseCatalog3Mapper

|  |
| --- |
| **public interface** BaseCatalog3Mapper **extends** Mapper<BaseCatalog3> { } |

BaseAttrInfoMapper

|  |
| --- |
| **public interface** BaseAttrInfoMapper **extends** Mapper<BaseAttrInfo> { } |

BaseAttrValueMapper

|  |
| --- |
| **public interface** BaseAttrValueMapper **extends** Mapper<BaseAttrValue> { } |

#### 2.1.3 在gamll-interface中 增加service接口

接口方法是根据页面得来

|  |
| --- |
| **public interface** ManageService {  **public** List<BaseCatalog1> getCatalog1();  **public** List<BaseCatalog2> getCatalog2(String catalog1Id);  **public** List<BaseCatalog3> getCatalog3(String catalog2Id);  **public** List<BaseAttrInfo> getAttrList(String catalog3Id);  } |

#### 2.1.4 在manage-service添加实现类

增加实现类

|  |
| --- |
| @com.alibaba.dubbo.config.annotation.Service **public class** ManageServiceImpl **implements** ManageService {   @Autowired  BaseAttrInfoMapper **baseAttrInfoMapper**;   @Autowired  BaseAttrValueMapper **baseAttrValueMapper**;   @Autowired  BaseCatalog1Mapper **baseCatalog1Mapper**;   @Autowired  BaseCatalog2Mapper **baseCatalog2Mapper**;   @Autowired  BaseCatalog3Mapper **baseCatalog3Mapper**;   @Override  **public** List<BaseCatalog1> getCatalog1() {  List<BaseCatalog1> baseCatalog1List = **baseCatalog1Mapper**.selectAll();  **return** baseCatalog1List;  }   @Override  **public** List<BaseCatalog2> getCatalog2(String catalog1Id) {  BaseCatalog2 baseCatalog2=**new** BaseCatalog2();  baseCatalog2.setCatalog1Id(catalog1Id);   List<BaseCatalog2> baseCatalog2List = **baseCatalog2Mapper**.select(baseCatalog2);  **return** baseCatalog2List;  }   @Override  **public** List<BaseCatalog3> getCatalog3(String catalog2Id) {  BaseCatalog3 baseCatalog3=**new** BaseCatalog3();  baseCatalog3.setCatalog2Id(catalog2Id);   List<BaseCatalog3> baseCatalog3List = **baseCatalog3Mapper**.select(baseCatalog3);  **return** baseCatalog3List;  }   @Override  **public** List<BaseAttrInfo> getAttrList(String catalog3\_id) {  BaseAttrInfo baseAttrInfo = **new** BaseAttrInfo();  baseAttrInfo.setCatalog3Id(catalog3\_id);   List<BaseAttrInfo> baseAttrInfoList = **baseAttrInfoMapper**.select(baseAttrInfo);  **return** baseAttrInfoList;   } } |

注意在启动类前加上注解

|  |
| --- |
| @MapperScan(basePackages = **"com.atguigu.gmall.manage.mapper"**) |

配置manage-service的application.properties

|  |
| --- |
| **server.port**=**8083 logging.level.root**=**debug spring.dubbo.application.name**=**manage-service spring.dubbo.registry.protocol**=**zookeeper spring.dubbo.registry.address**=**192.168.67.201:2181 spring.dubbo.base-package**=**com.atguigu.gmall spring.dubbo.protocol.name**=**dubbo spring.datasource.url**=**jdbc:mysql://localhost:3306/gmall?characterEncoding=UTF-8 spring.datasource.username**=**root spring.datasource.password**=**root spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver** |

#### 2.1.5 manage-web的controller中AttrManageController中增加方法

ManageController : 针对平台属性的操作！

|  |
| --- |
| @Controller @CrossOrigin **public class** ManageController {   @Reference  **private** ManageService **manageService**;   @RequestMapping(**"getCatalog1"**)  @ResponseBody  **public** List<BaseCatalog1> getCatalog1(){  **return manageService**.getCatalog1();  }   @RequestMapping(**"getCatalog2"**)  @ResponseBody  **public** List<BaseCatalog2> getCatalog2(String catalog1Id){  **return manageService**.getCatalog2(catalog1Id);  }   @RequestMapping(**"getCatalog3"**)  @ResponseBody  **public** List<BaseCatalog3> getCatalog3(String catalog2Id){  **return manageService**.getCatalog3(catalog2Id);  }   @RequestMapping(**"attrInfoList"**)  @ResponseBody  **public** List<BaseAttrInfo> attrInfoList(String catalog3Id){  **return manageService**.getAttrList(catalog3Id);  } } |

配置manage-web的application.properties

|  |
| --- |
| **server.port**=**8082 spring.thymeleaf.cache**=**false spring.thymeleaf.mode**=**LEGACYHTML5 spring.dubbo.application.name**=**manage-web spring.dubbo.registry.protocol**=**zookeeper spring.dubbo.registry.address**=**192.168.67.201:2181 spring.dubbo.base-package**=**com.atguigu.gmall spring.dubbo.protocol.name**=**dubbo spring.dubbo.consumer.timeout**=**10000 spring.dubbo.consumer.check**=**false** |

同样：注意在启动类上加注解！

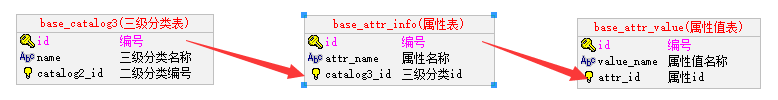
|  |
| --- |
| @ComponentScan(basePackages = **"com.atguigu.gmall"**) |

启动服务后，进行测试



### 2.2 属性的添加

数据库中的表！



#### 2.2.1 后台代码

保存时调用后台的请求：

|  |  |
| --- | --- |
| **saveAttrInfo** | 保存属性和属性值的请求 |

ManageController

|  |
| --- |
| @RequestMapping(**"saveAttrInfo"**) @ResponseBody **public void** saveAttrInfo(@RequestBody BaseAttrInfo baseAttrInfo){  *// 调用服务层做保存方法* **manageService**.saveAttrInfo(baseAttrInfo); } |

gmall-manage-service中的实现ManageServiceImpl

注意：在实体类BaseAttrInfo类中的id属性添加如下注解。

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

同时还需要添加属性值集合字段,添加get，set方法

|  |
| --- |
| @Transient **private** List<BaseAttrValue> **attrValueList**; |

实现类

|  |
| --- |
| @Override **public void** saveAttrInfo(BaseAttrInfo baseAttrInfo) {  *//如果有主键就进行更新，如果没有就插入* **if**(baseAttrInfo.getId()!=**null**&&baseAttrInfo.getId().length()>0){  **baseAttrInfoMapper**.updateByPrimaryKey(baseAttrInfo);  }**else**{  baseAttrInfo.setId(**null**);  **baseAttrInfoMapper**.insertSelective(baseAttrInfo);  }  *//把原属性值全部清空* BaseAttrValue baseAttrValue4Del = **new** BaseAttrValue();  baseAttrValue4Del.setAttrId(baseAttrInfo.getId());  **baseAttrValueMapper**.delete(baseAttrValue4Del);   *//重新插入属性值* **if**(baseAttrInfo.getAttrValueList()!=**null**&&baseAttrInfo.getAttrValueList().size()>0) {  **for** (BaseAttrValue attrValue : baseAttrInfo.getAttrValueList()) {  *//防止主键被赋上一个空字符串*  attrValue.setId(**null**);  attrValue.setAttrId(baseAttrInfo.getId());  **baseAttrValueMapper**.insertSelective(attrValue);  }  } } |

### 2.3 属性编辑

控制器

|  |
| --- |
| @RequestMapping(value = **"getAttrValueList"**,method = RequestMethod.***POST***) @ResponseBody **public** List<BaseAttrValue> getAttrValueList(String attrId){  BaseAttrInfo attrInfo = **manageService**.getAttrInfo(attrId);  **return** attrInfo.getAttrValueList(); } |
| 接口  *选中准修改数据 ， 根据该attrId 去查找AttrInfo，该对象下 List<BaseAttrValue> ！*  所以在返回的时候，需要返回BaseAttrInfo。  BaseAttrInfo getAttrInfo(String attrId); |
| 实现类  @Override **public** BaseAttrInfo getAttrInfo(String attrId) {  *// 创建属性对象* BaseAttrInfo attrInfo = **baseAttrInfoMapper**.selectByPrimaryKey(attrId);  *// 创建属性值对象* BaseAttrValue baseAttrValue = **new** BaseAttrValue();  *// 根据attrId字段查询对象* baseAttrValue.setAttrId(attrInfo.getId());  List<BaseAttrValue> attrValueList = **baseAttrValueMapper**.select(baseAttrValue);  *// 给属性对象中的属性值集合赋值* attrInfo.setAttrValueList(attrValueList);  *// 将属性对象返回* **return** attrInfo; } |