尚品汇商城

版本：V 2.0

# 一 电商的业务简介

## 1 整体业务简介



**当前的位置**

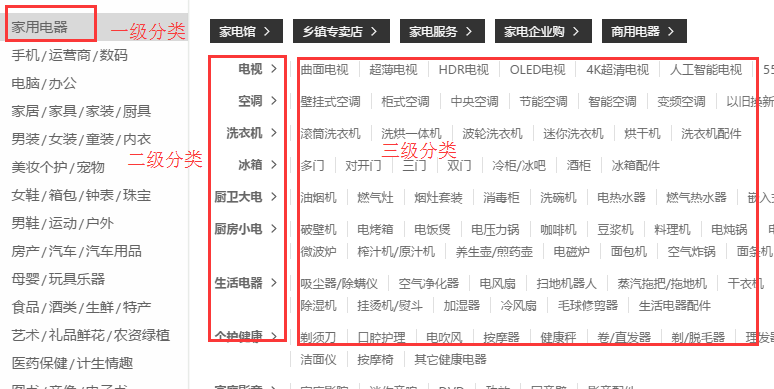
|  |  |
| --- | --- |
| 首页 | 静态页面，包含了商品分类，搜索栏，商品广告位。 |
| 全文搜索 | 通过搜索栏填入的关键字进行搜索，并列表展示 |
| 分类查询 | 根据首页的商品类目进行查询 |
| 商品详情 | 商品的详细信息展示 |
| 购物车 | 将有购买意向的商品临时存放的地方 |
| 单点登录 | 用户统一登录的管理 |
| 结算 | 将购物车中勾选的商品初始化成要填写的订单 |
| 下单 | 填好的订单提交 |
| 支付服务 | 下单后，用户点击支付，负责对接第三方支付系统。 |
| 订单服务 | 负责确认订单是否付款成功，并对接仓储物流系统。 |
|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

## 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

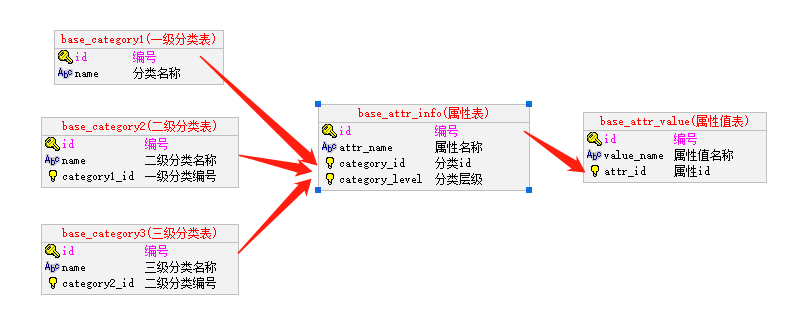


### 2.2 基本信息—平台属性

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，每个分类对应的**属性**都不同，分类包含一级分类、二级分类和三级分类，分类层级区分对应分类。

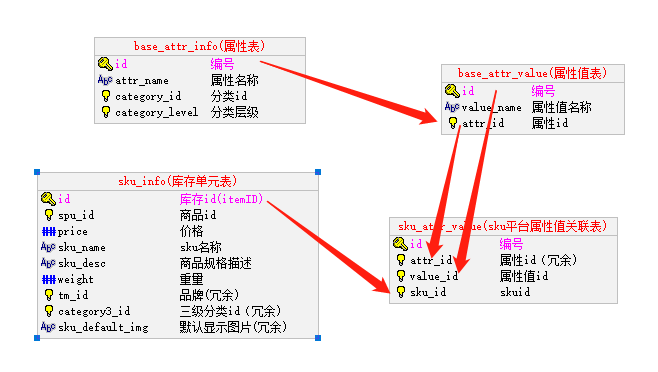


手机 属性，属性值

电脑 属性，属性值

手机与电脑 属性，属性值一样么？

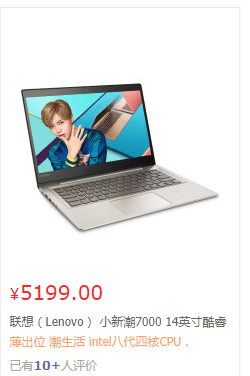
而每个**商品**对应的每种**属性**都有对应的**属性值。**



比如

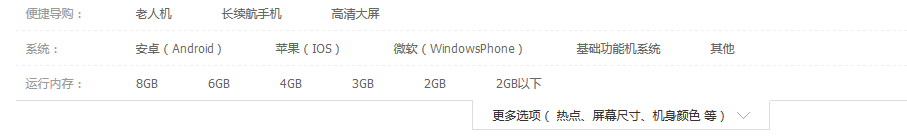
电脑整机的一级分类下，有笔记本、游戏本、台式机、一体机的二级分类。

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—销售属性与销售属性值

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。





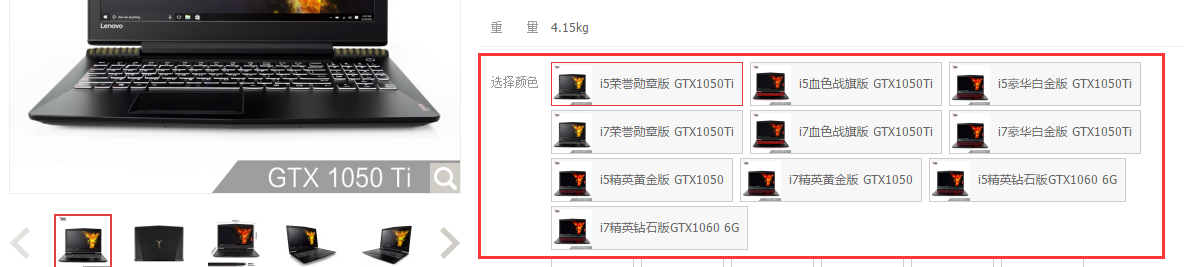
### 2.4 基本信息—spu与 sku

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。

# 二 接口分析

## 1分类接口



参考接口文档《电商管理系统API接口文档 .doc》

分类与平台属性接口：1,2,3

## 2获取平台属性接口



参考接口文档《电商管理系统API接口文档 .doc》

分类与平台属性接口：4

## 3添加平台属性接口



参考接口文档《电商管理系统API接口文档 .doc》

分类与平台属性接口：5

## 4修改平台属性接口





参考接口文档《电商管理系统API接口文档 .doc》

分类与平台属性接口：6,7

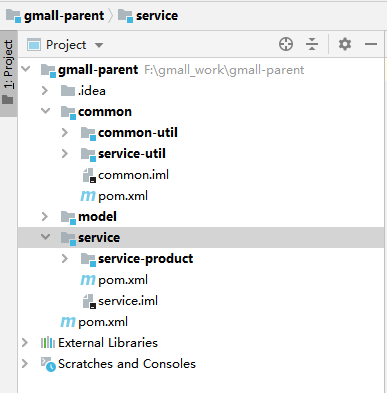
# 三 商品管理模块开发

## 1搭建service-product模块

### 1.1 搭建service-product

搭建过程同common-util

如图



### 1.2 修改配置

修改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**>  <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>service</**artifactId**>  <**version**>1.0</**version**>  </**parent**>   <**artifactId**>service-product</**artifactId**>  <**version**>1.0</**version**>   <**packaging**>jar</**packaging**>  <**name**>service-product</**name**>  <**description**>service-product</**description**>   <**dependencies**>  <**dependency**>  <**groupId**>net.oschina.zcx7878</**groupId**>  <**artifactId**>fastdfs-client-java</**artifactId**>  </**dependency**>  </**dependencies**>   <**build**>  <**finalName**>service-product</**finalName**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>   </**project**> |

添加配置文件application.yml

|  |
| --- |
| **spring**:  **application**:  **name**: service-product  **profiles**:  **active**: dev  **cloud**:  **nacos**:  **discovery**:  **server-addr**: 127.0.0.1:8848 |

注：很多时候，我们项目在开发环境和生成环境的环境配置是不一样的，例如，数据库配置，在开发的时候，我们一般用测试数据库，而在生产环境的时候，我们是用正式的数据，这时候，我们可以利用profile在不同的环境下配置用不同的配置文件或者不同的配置；

spring boot允许你通过命名约定按照一定的格式(application-{profile}.yml)来定义多个配置文件，然后通过在application.yml通过spring.profiles.active来具体激活一个或者多个配置文件，如果没有没有指定任何profile的配置文件的话，spring boot默认会启动application.yml。

添加配置文件application-dev.yml

|  |
| --- |
| **server**:  **port**: 8206  **mybatis-plus**:  **configuration**:  **log-impl**: org.apache.ibatis.logging.stdout.StdOutImpl  **mapper-locations**: classpath:com/atguigu/service/\*/mapper/\*.xml  **spring**:  **cloud**:  **sentinel**:  **transport**:  **dashboard**: http://127.0.0.1:8080  **rabbitmq**:  **host**: 47.93.118.241  **port**: 9672  **username**: admin  **password**: admin  **redis**:  **host**: localhost  **port**: 6379  **database**: 0  **timeout**: 1800000  **password**:  **lettuce**:  **pool**:  **max-active**: 20 *#最大连接数* **max-wait**: -1 *#最大阻塞等待时间(负数表示没限制)* **max-idle**: 5 *#最大空闲* **min-idle**: 0 *#最小空闲* **datasource**:  **type**: com.zaxxer.hikari.HikariDataSource  **driver-class-name**: com.mysql.jdbc.Driver  **url**: jdbc:mysql://localhost:3306/gmall\_product?characterEncoding=utf-8&useSSL=false  **username**: root  **password**: root  **hikari**:  **connection-test-query**: SELECT 1  **connection-timeout**: 60000  **idle-timeout**: 500000  **max-lifetime**: 540000  **maximum-pool-size**: 12  **minimum-idle**: 10  **pool-name**: GuliHikariPool  **jackson**:  **date-format**: yyyy-MM-dd HH:mm:ss  **time-zone**: GMT+8 |

### 1.3 添加启动类

|  |
| --- |
| **package** com.atguigu.gmall.product;  @SpringBootApplication @ComponentScan({**"com.atguigu.gmall"**}) @EnableDiscoveryClient **public class** ServiceProductApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(ServiceProductApplication.**class**, args);  }  } |

## 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

model模块统一存放实体类,该实体放在：com.atguigu.gmall.model.product包下面

BaseEntity

|  |
| --- |
| **package** com.atguigu.gmall.model.base;   **import** java.io.Serializable;  @Data **public class** BaseEntity **implements** Serializable {   @ApiModelProperty(value = **"id"**)  @TableId(type = IdType.***AUTO***)  **private** Long **id**; } |

BaseCategory1

|  |
| --- |
| @Data @ApiModel(description = **"商品一级分类"**) @TableName(**"base\_category1"**) **public class** BaseCategory1 **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"分类名称"**)  @TableField(**"name"**)  **private** String **name**;  } |

BaseCategory2

|  |
| --- |
| @Data @ApiModel(description = **"商品二级分类"**) @TableName(**"base\_category2"**) **public class** BaseCategory2 **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"二级分类名称"**)  @TableField(**"name"**)  **private** String **name**;   @ApiModelProperty(value = **"一级分类编号"**)  @TableField(**"category1\_id"**)  **private** Long **category1Id**;  } |

BaseCategory3

|  |
| --- |
| @Data @ApiModel(description = **"商品三级分类"**) @TableName(**"base\_category3"**) **public class** BaseCategory3 **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"三级分类名称"**)  @TableField(**"name"**)  **private** String **name**;   @ApiModelProperty(value = **"二级分类编号"**)  @TableField(**"category2\_id"**)  **private** Long **category2Id**;  } |

BaseAttrInfo

|  |
| --- |
| @Data @ApiModel(description = **"平台属性"**) @TableName(**"base\_attr\_info"**) **public class** BaseAttrInfo **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"属性名称"**)  @TableField(**"attr\_name"**)  **private** String **attrName**;   @ApiModelProperty(value = **"分类id"**)  @TableField(**"category\_id"**)  **private** Long **categoryId**;   @ApiModelProperty(value = **"分类层级"**)  @TableField(**"category\_level"**)  **private** Integer **categoryLevel**;   @TableField(exist = **false**)  **private** List<BaseAttrValue> **attrValueList**;  } |

BaseAttrValue

|  |
| --- |
| @Data @ApiModel(description = **"平台属性值"**) @TableName(**"base\_attr\_value"**) **public class** BaseAttrValue **extends** BaseEntity {   **private static final long *serialVersionUID*** = 1L;   @ApiModelProperty(value = **"属性值名称"**)  @TableField(**"value\_name"**)  **private** String **valueName**;   @ApiModelProperty(value = **"属性id"**)  @TableField(**"attr\_id"**)  **private** Long **attrId**; } |

#### 2.1.2 service-product中创建Mapper

包名：com.atguigu.gmall.product.mapper

BaseCategory1Mapper

|  |
| --- |
| @Mapper **public interface** BaseCategory1Mapper **extends** BaseMapper<BaseCategory1> { } |

BaseCategory2Mapper

|  |
| --- |
| @Mapper **public interface** BaseCategory2Mapper **extends** BaseMapper<BaseCategory2> { } |

BaseCategory1Mapper

|  |
| --- |
| @Mapper **public interface** BaseCategory3Mapper **extends** BaseMapper<BaseCategory3> { } |

BaseAttrInfoMapper

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

BaseAttrValueMapper

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

#### 2.1.3 在service-product中 增加service接口

包名：com.atguigu.gmall.product.service

接口方法是根据页面得来

|  |
| --- |
| **public interface** ManageService {  */\*\*  \* 查询所有的一级分类信息  \** ***@return*** *\*/* List<BaseCategory1> getCategory1();  */\*\*  \* 根据一级分类Id 查询二级分类数据  \** ***@param category1Id*** *\** ***@return*** *\*/* List<BaseCategory2> getCategory2(Long category1Id);  */\*\*  \* 根据二级分类Id 查询三级分类数据  \** ***@param category2Id*** *\** ***@return*** *\*/* List<BaseCategory3> getCategory3(Long category2Id);  */\*\*  \* 根据分类Id 获取平台属性数据  \* 接口说明：  \* 1，平台属性可以挂在一级分类、二级分类和三级分类  \* 2，查询一级分类下面的平台属性，传：category1Id，0，0； 取出该分类的平台属性  \* 3，查询二级分类下面的平台属性，传：category1Id，category2Id，0；   \* 取出对应一级分类下面的平台属性与二级分类对应的平台属性  \* 4，查询三级分类下面的平台属性，传：category1Id，category2Id，category3Id；  \* 取出对应一级分类、二级分类与三级分类对应的平台属性  \*  \** ***@param category1Id*** *\** ***@param category2Id*** *\** ***@param category3Id*** *\** ***@return*** *\*/*  List<BaseAttrInfo> getAttrInfoList(Long category1Id, Long category2Id, Long category3Id);  } |

#### 2.1.4 在service-product添加实现类

包名：com.atguigu.gmall.product.service.impl

增加实现类

|  |
| --- |
| @Service **public class** ManageServiceImpl **implements** ManageService {  @Autowired  **private** BaseCategory1Mapper **baseCategory1Mapper**;   @Autowired  **private** BaseCategory2Mapper **baseCategory2Mapper**;   @Autowired  **private** BaseCategory3Mapper **baseCategory3Mapper**;   @Autowired  **private** BaseAttrInfoMapper **baseAttrInfoMapper**;   @Autowired  **private** BaseAttrValueMapper **baseAttrValueMapper**;   @Override  **public** List<BaseCategory1> getCategory1() {  **return baseCategory1Mapper**.selectList(**null**);  }   @Override  **public** List<BaseCategory2> getCategory2(Long category1Id) {  *// select \* from baseCategory2 where Category1Id = ?* QueryWrapper queryWrapper = **new** QueryWrapper<BaseCategory2>();  queryWrapper.eq(**"category1\_id"**,category1Id);  List<BaseCategory2> baseCategory2List = **baseCategory2Mapper**.selectList(queryWrapper);  **return** baseCategory2List;  }   @Override  **public** List<BaseCategory3> getCategory3(Long category2Id) {  *// select \* from baseCategory3 where Category2Id = ?* QueryWrapper queryWrapper = **new** QueryWrapper<BaseCategory3>();  queryWrapper.eq(**"category2\_id"**,category2Id);  **return baseCategory3Mapper**.selectList(queryWrapper);  }   @Override  **public** List<BaseAttrInfo> getAttrInfoList(Long category1Id, Long category2Id, Long category3Id) {  *// 调用mapper：* **return baseAttrInfoMapper**.selectBaseAttrInfoList(category1Id, category2Id, category3Id);  } } |

#### 2.1.5 BaseAttrInfoMapper类添加方法

|  |
| --- |
| */\*\*  \* 根据分类Id 查询平台属性集合对象 | 编写xml 文件  \** ***@param category1Id*** *\** ***@param category2Id*** *\** ***@param category3Id*** *\** ***@return*** *\*/* List<BaseAttrInfo> selectBaseAttrInfoList(@Param(**"category1Id"**)Long category1Id, @Param(**"category2Id"**)Long category2Id, @Param(**"category3Id"**)Long category3Id); |

#### 2.1.6 BaseAttrInfoMapper.xml添加查询方法

在resources目录添加com.atguigu.gmall.product.mapper文件夹，添加 BaseAttrInfoMapper.xml文件

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>*** *<!--namespace 定义接口的全路径-->* <**mapper namespace="com.atguigu.gmall.product.mapper.BaseAttrInfoMapper"**>  *<!--  resultMap：表示返回的映射结果集  id ： 表示唯一标识  type：表示返回结果集的数据类型  autoMapping : 自动映射  -->* <**resultMap id="baseAttrInfoMap" type="com.atguigu.gmall.model.product.BaseAttrInfo" autoMapping="true"**>  *<!--id:表示主键 property：表示实体类的属性名 column：表示通过sql 执行以后查询出来的字段名-->* <**id property="id" column="id"**></**id**>  *<!--result : 表示映射普通字段-->  <!--<result property="" column=""></result>-->  <!--mybatis 如何配置一对多-->  <!--ofType : 返回的数据类型-->* <**collection property="attrValueList" ofType="com.atguigu.gmall.model.product.BaseAttrValue" autoMapping="true"**>  *<!--如果有字段重复则起别名-->* <**id property="id" column="attr\_value\_id"**></**id**>  </**collection**>  </**resultMap**>  *<!--id:表示方法名-->* <**select id="selectBaseAttrInfoList" resultMap="baseAttrInfoMap"**>  SELECT  bai.id,  bai.attr\_name,  bai.category\_id,  bai.category\_level,  bav.id attr\_value\_id,  bav.value\_name,  bav.attr\_id  FROM  base\_attr\_info bai  INNER JOIN base\_attr\_value bav ON bai.id = bav.attr\_id  <**where**>  <**if test="category1Id != null and category1Id != 0"**>  or (bai.category\_id = #{category1Id} and bai.category\_level = 1)  </**if**>  <**if test="category2Id != null and category2Id != 0"**>  or (bai.category\_id = #{category2Id} and bai.category\_level = 2)  </**if**>  <**if test="category3Id != null and category3Id != 0"**>  or (bai.category\_id = #{category3Id} and bai.category\_level = 3)  </**if**>  </**where**>  order by bai.category\_level, bai.id </**select**>  </**mapper**> |

#### 2.1.6 service-prodect的controller中BaseManageController 中增加方法

ManageController : 针对平台属性的操作！

|  |
| --- |
| @Api(tags = **"商品基础属性接口"**) @RestController @RequestMapping(**"admin/product"**) **public class** BaseManageController {   @Autowired  **private** ManageService **manageService**;   */\*\*  \* 查询所有的一级分类信息  \** ***@return*** *\*/* @GetMapping(**"getCategory1"**)  **public** Result<List<BaseCategory1>> getCategory1() {  List<BaseCategory1> baseCategory1List = **manageService**.getCategory1();  **return** Result.*ok*(baseCategory1List);  }   */\*\*  \* 根据一级分类Id 查询二级分类数据  \** ***@param category1Id*** *\** ***@return*** *\*/* @GetMapping(**"getCategory2/{category1Id}"**)  **public** Result<List<BaseCategory2>> getCategory2(@PathVariable(**"category1Id"**) Long category1Id) {  List<BaseCategory2> baseCategory2List = **manageService**.getCategory2(category1Id);  **return** Result.*ok*(baseCategory2List);  }   */\*\*  \* 根据二级分类Id 查询三级分类数据  \** ***@param category2Id*** *\** ***@return*** *\*/* @GetMapping(**"getCategory3/{category2Id}"**)  **public** Result<List<BaseCategory3>> getCategory3(@PathVariable(**"category2Id"**) Long category2Id) {  List<BaseCategory3> baseCategory3List = **manageService**.getCategory3(category2Id);  **return** Result.*ok*(baseCategory3List);  }    */\*\*  \* 根据分类Id 获取平台属性数据  \** ***@param category1Id*** *\** ***@param category2Id*** *\** ***@param category3Id*** *\** ***@return*** *\*/* @GetMapping(**"attrInfoList/{category1Id}/{category2Id}/{category3Id}"**)  **public** Result<List<BaseAttrInfo>> attrInfoList(@PathVariable(**"category1Id"**) Long category1Id,  @PathVariable(**"category2Id"**) Long category2Id,  @PathVariable(**"category3Id"**) Long category3Id) {  List<BaseAttrInfo> baseAttrInfoList = **manageService**.getAttrInfoList(category1Id, category2Id, category3Id);  **return** Result.*ok*(baseAttrInfoList);  }  } |

#### 2.1.7 service-prodect中添加启动类

包名：com.atguigu.gmall.product

ServiceProductApplication类

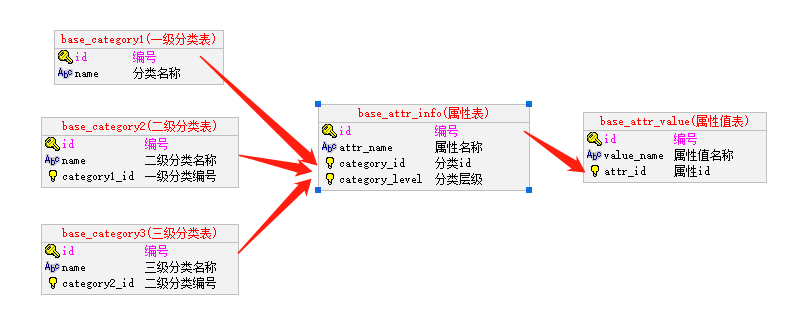
|  |
| --- |
| @SpringBootApplication @ComponentScan({**"com.atguigu.gmall"**}) @EnableDiscoveryClient **public class** ServiceProductApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(ServiceProductApplication.**class**, args);  }  } |

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 |

BaseManageController

|  |
| --- |
| */\*\*  \* 保存平台属性方法  \** ***@param baseAttrInfo*** *\** ***@return*** *\*/* @PostMapping(**"saveAttrInfo"**) **public** Result saveAttrInfo(@RequestBody BaseAttrInfo baseAttrInfo) {  *// 前台数据都被封装到该对象中baseAttrInfo* **manageService**.saveAttrInfo(baseAttrInfo);  **return** Result.*ok*(); } |

注意：在实体类BaseAttrInfo类中添加属性值集合字段。

|  |
| --- |
| @TableField(exist = **false**) **private** List<BaseAttrValue> **attrValueList**; |

在ManageService添加service接口

|  |
| --- |
| */\*\*  \* 保存平台属性方法  \** ***@param baseAttrInfo*** *\*/* **void** saveAttrInfo(BaseAttrInfo baseAttrInfo); |

实现类

|  |
| --- |
| @Override @Transactional **public void** saveAttrInfo(BaseAttrInfo baseAttrInfo) {  *// 什么情况下 是添加，什么情况下是更新，修改 根据baseAttrInfo 的Id  // baseAttrInfo* **if** (baseAttrInfo.getId() != **null**) {  *// 修改数据* **baseAttrInfoMapper**.updateById(baseAttrInfo);  } **else** {  *// 新增  // baseAttrInfo 插入数据* **baseAttrInfoMapper**.insert(baseAttrInfo);  }   *// baseAttrValue 平台属性值  // 修改：通过先删除{baseAttrValue}，在新增的方式！  // 删除条件：baseAttrValue.attrId = baseAttrInfo.id* QueryWrapper queryWrapper = **new** QueryWrapper<BaseAttrValue>();  queryWrapper.eq(**"attr\_id"**, baseAttrInfo.getId());  **baseAttrValueMapper**.delete(queryWrapper);   *// 获取页面传递过来的所有平台属性值数据* List<BaseAttrValue> attrValueList = baseAttrInfo.getAttrValueList();  **if** (attrValueList != **null** && attrValueList.size() > 0) {  *// 循环遍历* **for** (BaseAttrValue baseAttrValue : attrValueList) {  *// 获取平台属性Id 给attrId* baseAttrValue.setAttrId(baseAttrInfo.getId()); *// ?* **baseAttrValueMapper**.insert(baseAttrValue);  }  } } |

事务：

实现类添加：@Transactional

### 2.3 修改平台属性

控制器

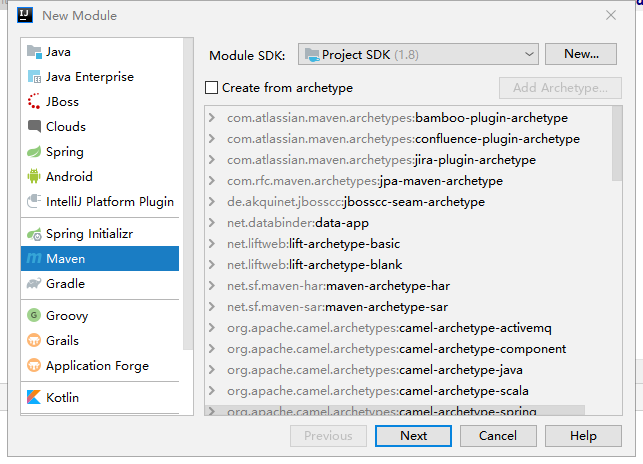
|  |
| --- |
| */\*\*  \* 根据attrId 查询平台属性对象  \** ***@param attrId*** *\** ***@return*** *\*/* @GetMapping(**"getAttrValueList/{attrId}"**) **public** Result<List<BaseAttrValue>> getAttrValueList(@PathVariable(**"attrId"**) Long attrId) {  BaseAttrInfo baseAttrInfo = **manageService**.getAttrInfo(attrId);  List<BaseAttrValue> baseAttrValueList = baseAttrInfo.getAttrValueList();  **return** Result.*ok*(baseAttrValueList); } |
| 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。  */\*\*  \* 根据attrId 查询平台属性对象  \** ***@param attrId*** *\** ***@return*** *\*/* BaseAttrInfo getAttrInfo(Long attrId); |
| 实现类  @Override **public** BaseAttrInfo getAttrInfo(Long attrId) {  BaseAttrInfo baseAttrInfo = **baseAttrInfoMapper**.selectById(attrId);  *// 查询到最新的平台属性值集合数据放入平台属性中！* baseAttrInfo.setAttrValueList(getAttrValueList(attrId));  **return** baseAttrInfo; }  */\*\*  \* 根据属性id获取属性值  \** ***@param attrId*** *\** ***@return*** *\*/* **private** List<BaseAttrValue> getAttrValueList(Long attrId) {  *// select \* from baseAttrValue where attrId = ?* QueryWrapper queryWrapper = **new** QueryWrapper<BaseAttrValue>();  queryWrapper.eq(**"attr\_id"**, attrId);  List<BaseAttrValue> baseAttrValueList = **baseAttrValueMapper**.selectList(queryWrapper);  **return** baseAttrValueList; } |

## 3搭建server-gateway模块

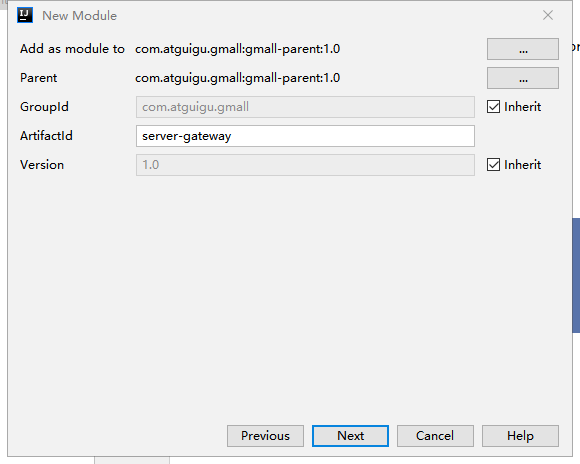
服务网关

### 3.1 搭建server-gateway

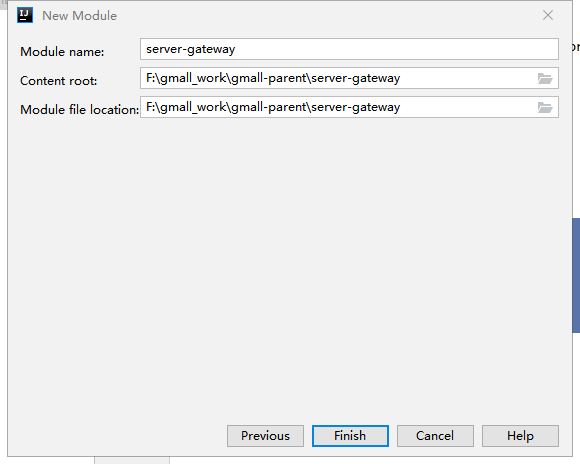
点击gmall-parent，选择New–>Module,操作如下



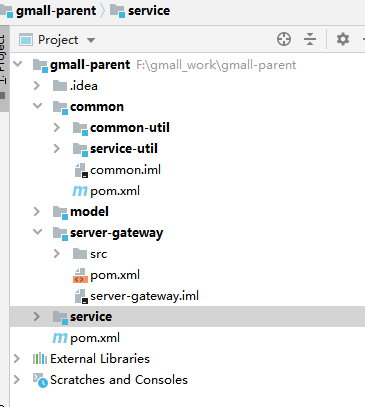
点击下一步



点击下一步



点击完成



### 3.2 修改配置pom.xml

修改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**>   <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-parent</**artifactId**>  <**version**>1.0</**version**>  </**parent**>   <**version**>1.0</**version**>  <**artifactId**>server-gateway</**artifactId**>   <**packaging**>jar</**packaging**>  <**name**>server-gateway</**name**>   <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>common-util</**artifactId**>  <**version**>1.0</**version**>  </**dependency**>   *<!-- 服务注册 -->* <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-alibaba-nacos-discovery</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-gateway</**artifactId**>  </**dependency**>    </**dependencies**>   <**build**>  <**finalName**>server-gateway</**finalName**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>  </**project**> |

### 3.3 在resources下添加配置文件

application.yml

|  |
| --- |
| **server**:  **port**: 80 **spring**:  **application**:  **name**: api-gateway  **cloud**:  **nacos**:  **discovery**:  **server-addr**: 127.0.0.1:8848  **gateway**:  **discovery**: *#是否与服务发现组件进行结合，通过 serviceId(必须设置成大写) 转发到具体的服务实例。默认为false，设为true便开启通过服务中心的自动根据 serviceId 创建路由的功能。* **locator**: *#路由访问方式：http://Gateway\_HOST:Gateway\_PORT/大写的serviceId/\*\*，其中微服务应用名默认大写访问。* **enabled**: **true  routes**:  - **id**: service-product  **uri**: lb://service-product  **predicates**:  - Path=/\*/product/\*\* *# 路径匹配* |

### 3.4 启动类

|  |
| --- |
| **package** com.atguigu.gmall.gateway;  @SpringBootApplication **public class** ServerGatewayApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(ServerGatewayApplication.**class**, args);  }  } |

### 3.4 跨域处理

跨域：浏览器对于javascript的同源策略的限制 。

以下情况都属于跨域：

| **跨域原因说明** | **示例** |
| --- | --- |
| 域名不同 | www.jd.com 与 www.taobao.com |
| 域名相同，端口不同 | www.jd.com:8080 与 www.jd.com:8081 |
| 二级域名不同 | item.jd.com 与 miaosha.jd.com |

如果**域名和端口都相同，但是请求路径不同**，不属于跨域，如：

www.jd.com/item

www.jd.com/goods

http和https也属于跨域

而我们刚才是从localhost:1000去访问localhost:8888，这属于端口不同，跨域了。

**3.4.1 为什么有跨域问题？**

跨域不一定都会有跨域问题。

因为跨域问题是浏览器对于ajax请求的一种安全限制：**一个页面发起的ajax请求，只能是与当前页域名相同的路径**，这能有效的阻止跨站攻击。

因此：**跨域问题 是针对ajax的一种限制**。

但是这却给我们的开发带来了不便，而且在实际生产环境中，肯定会有很多台服务器之间交互，地址和端口都可能不同，怎么办？

**3.4.1 解决跨域问题的方案**

目前比较常用的跨域解决方案有3种：

- Jsonp

最早的解决方案，利用script标签可以跨域的原理实现。

限制：

- 需要服务的支持

- 只能发起GET请求

- nginx反向代理

思路是：利用nginx把跨域反向代理为不跨域，支持各种请求方式

缺点：需要在nginx进行额外配置，语义不清晰

- CORS

规范化的跨域请求解决方案，安全可靠。

优势：

- 在服务端进行控制是否允许跨域，可自定义规则

- 支持各种请求方式

缺点：

- 会产生额外的请求

我们这里会采用cors的跨域方案。

**3.4.1 什么是cors**

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。

它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

- 浏览器端：

目前，所有浏览器都支持该功能（IE10以下不行）。整个CORS通信过程，都是浏览器自动完成，不需要用户参与。

- 服务端：

CORS通信与AJAX没有任何差别，因此你不需要改变以前的业务逻辑。只不过，浏览器会在请求中携带一些头信息，我们需要以此判断是否允许其跨域，然后在响应头中加入一些信息即可。这一般通过过滤器完成即可。

**3.4.1 原理有点复杂**

预检请求

跨域请求会在正式通信之前，增加一次HTTP查询请求，称为"预检"请求（preflight）。

浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。

一个“预检”请求的样板：

OPTIONS /cors HTTP/1.1

Origin: http://localhost:1000

Access-Control-Request-Method: GET

Access-Control-Request-Headers: X-Custom-Header

User-Agent: Mozilla/5.0...

- Origin：会指出当前请求属于哪个域（协议+域名+端口）。服务会根据这个值决定是否允许其跨域。

- Access-Control-Request-Method：接下来会用到的请求方式，比如PUT

- Access-Control-Request-Headers：会额外用到的头信息

预检请求的响应

服务的收到预检请求，如果许可跨域，会发出响应：

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 01:15:39 GMT

Server: Apache/2.0.61 (Unix)

Access-Control-Allow-Origin: http://localhost:1000

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: GET, POST, PUT

Access-Control-Allow-Headers: X-Custom-Header

Access-Control-Max-Age: 1728000

Content-Type: text/html; charset=utf-8

Content-Encoding: gzip

Content-Length: 0

Keep-Alive: timeout=2, max=100

Connection: Keep-Alive

Content-Type: text/plain

如果服务器允许跨域，需要在返回的响应头中携带下面信息：

- Access-Control-Allow-Origin：可接受的域，是一个具体域名或者\*（代表任意域名）

- Access-Control-Allow-Credentials：是否允许携带cookie，默认情况下，cors不会携带cookie，除非这个值是true

- Access-Control-Allow-Methods：允许访问的方式

- Access-Control-Allow-Headers：允许携带的头

- Access-Control-Max-Age：本次许可的有效时长，单位是秒，过期之前的ajax请求就无需再次进行预检了

有关cookie：

要想操作cookie，需要满足3个条件：

- 服务的响应头中需要携带Access-Control-Allow-Credentials并且为true。

- 浏览器发起ajax需要指定withCredentials 为true

- 响应头中的Access-Control-Allow-Origin一定不能为\*，必须是指定的域名

**3.4.1 实现非常简单**

**方式一：**

全局配置类实现

包名：com.atguigu.gmall.gateway.config

CorsConfig类

|  |
| --- |
| @Configuration **public class** CorsConfig {   @Bean  **public** CorsWebFilter corsWebFilter(){   *// cors跨域配置对象* CorsConfiguration configuration = **new** CorsConfiguration();  configuration.addAllowedOrigin(**"\*"**);  configuration.setAllowCredentials(**true**);  configuration.addAllowedMethod(**"\*"**);  configuration.addAllowedHeader(**"\*"**);   *// 配置源对象* UrlBasedCorsConfigurationSource configurationSource = **new** UrlBasedCorsConfigurationSource();  configurationSource.registerCorsConfiguration(**"/\*\*"**, configuration);  *// cors过滤器对象* **return new** CorsWebFilter(configurationSource);  } } |

**方式二：**

使用网关的跨域配置来实现，修改application.yml ,在spring.cloud.gateway节点添加配置

|  |
| --- |
| globalcors:  cors-configurations:  '[/\*\*]': # 匹配所有请求  allowedOrigins: "\*" #跨域处理 允许所有的域  allowedMethods: # 支持的方法  - GET  - POST  - PUT  - DELETE |

# 四 搭建后台页面

1. 拷贝资料中的前段项目页面，放入一个没有中文目录的文件下
2. 在gmall-admin当前目录下cmd 回车
   1. npm install [安装依赖 node\_modules]
   2. npm run dev
   3. 直接访问浏览器
3. 讲两个配置文件
   1. dev.env.js [http://localhost](http://localhost:8082)
   2. index.js host: 'localhost', port: 8888

注：第一个为网关地址，第二个为项目访问地址