# 天猫后台管理员系统后端开发手册

V 1.0

2Executioner

service-category模块搭建

config：

CategoryConfig.java

mapper：

CategoryInfoMapper.java

PropertyMapper.java

xml：

CategoryInfoMapper.xml

PropertyMapper.xml

service：

CategoryInfoService.java

PropertyService.java

impl：

CategoryInfoServiceImpl.java

PropertyServiceImpl.java

controller：

CategoryInfoController.java

PropertyController.java

application.yml

**server**:  
 **port**: 8201  
  
**spring**:  
 **application**:  
 **name**: service-category  
 **cloud**:  
 **nacos**:  
 **discovery**:  
 **server-addr**: 192.168.123.130:8848  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.123.130:3306/tmall\_category?useUnicode=true&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=GMT%2B8  
 **username**: root  
 **password**: 123321  
  
*# 指定mapper的.xml文件路径***mybatis-plus**:  
 **mapper-locations**: classpath:com/study/tmall/category/mapper/xml/\*.xml

启动类

@SpringBootApplication  
@EnableDiscoveryClient  
@ComponentScan(basePackages = "com.study")  
@MapperScan("com.study.tmall.category.mapper")  
**public class** ServiceCategoryApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ServiceCategoryApplication.**class**, args);  
 }  
}

分类管理

控制层

CategoryInfoController

url：/admin/category/categoryInfo

分页条件显示分类

**前端传入参数：**current起始页）limit（每页显示数量）categoryQueryVo（前端分类查询条件实体类）

**请求方式：**get

**请求url：**/list/{current}/{limit}

**后端方法：**public Result findPageCategoryInfo(@PathVariable Long current, @PathVariable Long limit, CategoryQueryVo categoryQueryVo)

**后端返回数据参数：**IPage<CategoryInfo>（包含分页参数的集合）

CategoryInfoController

// 分页条件显示分类  
@ApiOperation(value = "分页条件显示分类")  
@GetMapping("/list/{current}/{limit}")  
**public** Result findPageCategoryInfo(  
 @ApiParam(name = "current", value = "起始页", required = **true**)  
 @PathVariable Long current,  
  
 @ApiParam(name = "limit", value = "每页记录数", required = **true**)  
 @PathVariable Long limit,  
  
 @ApiParam(name = "categoryQueryVo", value = "前端分类查询对象")  
 CategoryQueryVo categoryQueryVo){  
  
 Page<CategoryInfo> page = **new** Page<>(current, limit);  
 IPage<CategoryInfo> pageModule = categoryInfoService.findPageCategoryInfo(page, categoryQueryVo);  
 **return** Result.*ok*(pageModule);  
}

CategoryInfoService

// 分页条件显示分类  
IPage<CategoryInfo> findPageCategoryInfo(Page<CategoryInfo> page, CategoryQueryVo categoryQueryVo);

CategoryInfoServiceImpl

*/\*\*  
 \* 分页条件显示分类  
 \** ***@param*** *page  
 \** ***@param*** *categoryQueryVo  
 \** ***@return*** *\*/*@Override  
**public** IPage<CategoryInfo> findPageCategoryInfo(Page<CategoryInfo> page, CategoryQueryVo categoryQueryVo) {  
 QueryWrapper<CategoryInfo> wrapper = **new** QueryWrapper<>();  
 String name = categoryQueryVo.getName();  
 Date createTimeBegin = categoryQueryVo.getCreateTimeBegin();  
 Date createTimeEnd = categoryQueryVo.getCreateTimeEnd();  
  
 // 如果查询名称为空，则模糊匹配分类名称  
 **if** (!StringUtils.*isEmpty*(name)){  
 wrapper.like("name", name);  
 }  
 // 如果开始日期不为空  
 **if** (createTimeBegin != **null**){  
 wrapper.ge("create\_time", createTimeBegin);  
 }  
 // 如果结束日期不为空  
 **if** (createTimeEnd != **null**){  
 wrapper.le("create\_time", createTimeEnd);  
 }  
  
 // 查询并返回结果  
 IPage<CategoryInfo> categoryInfoPage = baseMapper.selectPage(page, wrapper);  
 **return** categoryInfoPage;  
}

批量删除分类

**前端传入参数：**idList（要批量删除的分类id集合）

**请求方式：**delete

**请求url：**/batchRemove

**后端方法：**public Result batchRemove(@RequestBody List<String> idList)

**后端返回数据参数：**ok()

CategoryInfoController

// 批量删除分类  
@ApiOperation(value = "批量删除分类")  
@DeleteMapping("/batchRemove")  
**public** Result batchRemove(  
 @ApiParam(name = "idList", value = "分类id集合", required = **true**)  
 @RequestBody List<String> idList){  
 categoryInfoService.batchRemoveCategory(idList);  
 **return** Result.*ok*();  
}

CategoryInfoService

// 批量删除分类  
**void** batchRemoveCategory(List<String> idList);

CategoryInfoServiceImpl

*/\*\*  
 \* 批量删除分类  
 \** ***@param*** *idList  
 \*/*@Override  
**public void** batchRemoveCategory(List<String> idList) {  
 idList.stream().forEach(item-> {  
 **this**.removeCategoryById(item);  
 });  
}

删除分类

**前端传入参数：**id（要删除的分类id）

**请求方式：**delete

**请求url：**/{id}

**后端方法：**public Result remove(@PathVariable String id)

**后端返回数据参数：**ok()

CategoryInfoController

// 删除分类  
@ApiOperation(value = "删除分类")  
@DeleteMapping("/{id}")  
**public** Result remove(  
 @ApiParam(name = "id", value = "分类id", required = **true**)  
 @PathVariable String id){  
  
 categoryInfoService.removeCategoryById(id);  
 **return** Result.*ok*();  
}

CategoryInfoService

// 删除分类  
**void** removeCategoryById(String id);

CategoryInfoServiceImpl

*/\*\*  
 \* 删除分类  
 \** ***@param*** *id  
 \*/*@Override  
**public void** removeCategoryById(String id) {  
 // 根据id查询出分类  
 CategoryInfo categoryInfo = **this**.getById(id);  
 **if** (categoryInfo == **null**){ // 如果为空，抛出异常  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
  
 // 不能删除分类下的商品，商品数据多且重要  
  
 // 删除分类下的属性  
 propertyService.removeByCid(id);  
  
 // 删除图片  
 **this**.deleteFastImage(categoryInfo.getImageUrl());  
  
 // 数据库中逻辑删除分类  
 baseMapper.deleteById(categoryInfo);  
}

*/\*\*  
 \* 删除fastDFS中的图片  
 \** ***@param*** *imageUrl  
 \*/***private void** deleteFastImage(String imageUrl) {  
 **if** (!StringUtils.*isEmpty*(imageUrl)) {  
 // 取得图片 服务器协议+ip group fastDFS中的路径  
 String[] strings = ImageUtil.*splitUrl*(imageUrl);  
 // 传入group和fastDFS中的路径 删除fastDFS中的图片  
 Integer deleteResult = FastDFSUtil.*delete*(strings[1], strings[2]);  
 // 返回0表示删除成功，否则删除失败  
 **if** (!Objects.*equals*(deleteResult, 0)) {  
 **throw new** TmallException(ResultCodeEnum.*DELETE\_FAIL*);  
 }  
 }  
}

PropertyService

// 删除分类下的属性  
**void** removeByCid(String cid);

PropertyServiceImpl

*/\*\*  
 \* 删除分类下的属性  
 \** ***@param*** *cid  
 \*/*@Override  
**public void** removeByCid(String cid) {  
 QueryWrapper<Property> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("category\_id", cid);  
 baseMapper.delete(wrapper);  
}

添加分类图片

**前端传入参数：**file（分类对象，无id，包含上传图片后得到的imageUrl）

**请求方式：**post

**请求url：**/saveImage

**后端方法：**public Result saveImage(MultipartFile file)

**后端返回数据参数：**返回imageUrl和imageName（图片url和图片上传时的名称）

CategoryInfoController

// 添加分类图片  
@ApiOperation(value = "添加分类")  
@PostMapping("/saveImage")  
**public** Result saveImage(  
 @ApiParam(name = "file", value = "分类图片")  
 MultipartFile file){  
  
 Map<String, String> map = categoryInfoService.saveImage(file);  
 **return** Result.*ok*(map);  
}

CategoryInfoService

// 添加分类图片  
Map<String, String> saveImage(MultipartFile file);

CategoryInfoServiceImpl

*/\*\*  
 \* 添加分类图片  
 \** ***@param*** *file  
 \** ***@return*** *\*/*@Override  
**public** Map<String, String> saveImage(MultipartFile file) {  
 // 上传到fastDFS中  
 **try** {  
 Map<String, String> map = **new** HashMap<>();  
 String filename = file.getOriginalFilename();  
 String[] upload = FastDFSUtil.*upload*(file.getBytes(), ImageUtil.*getFileExtName*(file));  
 map.put("filename", filename); // 文件名称，包括拓展名  
 map.put("imageUrl", ImageUtil.*compoundUrl*(upload)); // 图片url地址  
 **return** map;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 **return null**;  
 }  
}

添加分类

**前端传入参数：**categoryInfo（分类对象，无id，包含上传图片后得到的imageUrl）

**请求方式：**post

**请求url：**/save

**后端方法：**public Result save(@RequestBody CategoryInfo categoryInfo)

**后端返回数据参数：**ok()

CategoryInfoController

// 添加分类  
@ApiOperation(value = "添加分类")  
@PostMapping("/save")  
**public** Result save(  
 @ApiParam(name = "categoryInfo", value = "分类", required = **true**)  
 @RequestBody CategoryInfo categoryInfo){ // 应该有分类名和分类图片的地址  
  
 categoryInfoService.save(categoryInfo);  
 **return** Result.*ok*();  
}

编辑分类

**前端传入参数：**categoryInfo（分类对象，包含id）

**请求方式：**put

**请求url：**/update

**后端方法：**public Result update(@RequestBody CategoryInfo categoryInfo)

**后端返回数据参数：**ok()

CategoryInfoController

// 编辑分类  
@ApiOperation(value = "编辑分类")  
@PutMapping("/update")  
**public** Result update(  
 @ApiParam(name = "categoryInfo", value = "分类信息（包含id和图片地址）", required = **true**)  
 @RequestBody CategoryInfo categoryInfo){  
  
 // 更新分类  
 categoryInfoService.updateCategoryById(categoryInfo);  
 **return** Result.*ok*();  
}

CategoryInfoService

// 编辑分类  
**void** updateCategoryById(CategoryInfo categoryInfo, MultipartFile file);

CategoryInfoServiceImpl

*/\*\*  
 \* 编辑分类  
 \** ***@param*** *categoryInfo  
 \*  
 \*/*@Override  
**public void** updateCategoryById(CategoryInfo categoryInfo) {  
 // 从数据库中取得分类  
 CategoryInfo original = **this**.getById(categoryInfo);  
 **if** (original == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 如果传来的分类名称为空则用以前的名称  
 **if** (StringUtils.*isEmpty*(categoryInfo.getName())) {  
 categoryInfo.setName(original.getName());  
 }  
 // 如果图片地址为空也用原来的  
 **if** (StringUtils.*isEmpty*(categoryInfo.getImageUrl())){  
 categoryInfo.setImageUrl(original.getImageUrl());  
 } **else** {  
 // 否则删除原来的图片  
 **this**.deleteFastImage(original.getImageUrl());  
 }  
 // 更新数据库  
 baseMapper.updateById(categoryInfo);  
}

属性管理

控制层

PropertyController

url：/admin/category/property

分页条件显示属性

**前端传入参数：**current（起始页）limit（每页显示数量）propertyQueryVo（前端属性查询条件实体类）

**请求方式：**get

**请求url：**/list/{current}/{limit}

**后端方法：**public Result findPageProperty(@PathVariable Integer current, @PathVariable Integer limit, PropertyQueryVo propertyQueryVo)

**后端返回数据参数：**IPage<Property> （包含分页参数的集合）

PropertyController

// 分页条件显示属性  
@ApiOperation(value = "分页条件显示属性")  
@GetMapping("/list/{current}/{limit}")  
**public** Result findPageProperty(  
 @ApiParam(name = "current", value = "起始页", required = **true**)  
 @PathVariable Integer current,  
  
 @ApiParam(name = "limit", value = "每页记录数", required = **true**)  
 @PathVariable Integer limit,  
  
 @ApiParam(name = "propertyQueryVo", value = "条件查询vo")  
 PropertyQueryVo propertyQueryVo){  
  
 Page<Property> page = **new** Page<>(current, limit);  
 IPage<Property> pageModule = propertyService.findPageProperty(page, propertyQueryVo);  
 **return** Result.*ok*(pageModule);  
}

PropertyService

// 分页条件显示属性  
IPage<Property> findPageProperty(Page<Property> page, PropertyQueryVo propertyQueryVo);

PropertyServiceImpl

*/\*\*  
 \* 分页条件显示属性  
 \** ***@param*** *page  
 \** ***@param*** *propertyQueryVo  
 \** ***@return*** *\*/*@Override  
**public** IPage<Property> findPageProperty(Page<Property> page, PropertyQueryVo propertyQueryVo) {  
 QueryWrapper<Property> wrapper = **new** QueryWrapper<>();  
 String categoryId = propertyQueryVo.getCategoryId();  
 String name = propertyQueryVo.getName();  
 Date createTimeBegin = propertyQueryVo.getCreateTimeBegin();  
 Date createTimeEnd = propertyQueryVo.getCreateTimeEnd();  
  
 **if** (StringUtils.*isEmpty*(categoryId)){  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 wrapper.eq("category\_id", categoryId);  
 **if** (!StringUtils.*isEmpty*(name)){  
 wrapper.like("name", name);  
 }  
 **if** (createTimeBegin != **null**){  
 wrapper.ge("create\_time", createTimeBegin);  
 }  
 **if** (createTimeEnd != **null**){  
 wrapper.le("create\_time", createTimeEnd);  
 }  
  
 IPage<Property> propertyPage = baseMapper.selectPage(page, wrapper);  
  
 **return** propertyPage;  
}

批量删除属性

**前端传入参数：**idList（要批量删除属性的id）

**请求方式：**delete

**请求url：**/batchRemove

**后端方法：**public Result batchRemove(@RequestBody List<String> idList)

**后端返回数据参数：**ok()

PropertyController

// 批量删除属性  
@ApiOperation(value = "批量删除属性")  
@DeleteMapping("/batchRemove")  
**public** Result batchRemove(  
 @ApiParam(name = "idList", value = "要批量删除属性的id", required = **true**)  
 @RequestBody List<String> idList){  
  
 propertyService.batchRemove(idList);  
 **return** Result.*ok*();  
}

PropertyService

// 批量删除属性  
**void** batchRemove(List<String> idList);

PropertyServiceImpl

*/\*\*  
 \* 批量删除属性  
 \** ***@param*** *idList  
 \*/*@Override  
**public void** batchRemove(List<String> idList) {  
 idList.stream().forEach(item -> {  
 baseMapper.deleteById(item);  
 });  
}

删除属性

**前端传入参数：**id（要删除属性的id）

**请求方式：**delete

**请求url：**/{id}

**后端方法：**public Result remove(@PathVariable String id)

**后端返回数据参数：**ok()

PropertyController

// 删除属性  
@ApiOperation(value = "删除属性")  
@DeleteMapping("/{id}")  
**public** Result remove(  
 @ApiParam(name = "id", value = "属性id", required = **true**)  
 @PathVariable String id){  
  
 propertyService.removeById(id);  
 **return** Result.*ok*();  
}

添加属性

**前端传入参数：**property（要保存的属性，无id）

**请求方式：**post

**请求url：**/save

**后端方法：**public Result save(@RequestBody Property property)

**后端返回数据参数：**ok()

PropertyController

// 添加属性  
@ApiOperation(value = "添加属性")  
@PostMapping("/save")  
**public** Result save(  
 @ApiParam(name = "property", value = "属性对象", required = **true**)  
 @RequestBody Property property){  
  
 propertyService.save(property);  
 **return** Result.*ok*();  
}

编辑属性

**前端传入参数：**property（要编辑的属性，有id）

**请求方式：**put

**请求url：**/update

**后端方法：**public Result update(@RequestBody Property property)

**后端返回数据参数：**ok()

PropertyController

// 编辑属性  
@ApiOperation(value = "编辑属性")  
@PutMapping("/update")  
**public** Result update(  
 @ApiParam(name = "property", value = "属性对象", required = **true**)  
 @RequestBody Property property){  
  
 propertyService.updateById(property);  
 **return** Result.*ok*();  
}

service-product模块搭建

mapper：

ProductInfoMapper.java

ProductImageMapper.java

PropertyValueMapper.java

ReviewMapper.java

xml：

ProductInfoMapper.xml

ProductImageMapper.xml

PropertyValueMapper.xml

ReviewMapper.xml

service：

ProductInfoService.java

ProductImageService.java

PropertyValueService.java

ReviewService.java

impl：

ProductInfoServiceImpl.java

ProductImageServiceImpl.java

PropertyValueServiceImpl.java

ReviewServiceImpl.java

listener：

ProductInfoListener

controller：

ProductInfoController.java

ProductImageController.java

PropertyValueController.java

application.yml

**server**:  
 **port**: 8202  
  
**spring**:  
 **application**:  
 **name**: service-product  
 **cloud**:  
 **nacos**:  
 **discovery**:  
 **server-addr**: 192.168.123.130:8848  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.123.130:3306/tmall\_product?useUnicode=true&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=GMT%2B8  
 **username**: root  
 **password**: 123321  
  
*# 指定mapper的.xml文件路径***mybatis-plus**:  
 **mapper-locations**: classpath:com/study/tmall/product/mapper/xml/\*.xml

启动类

@SpringBootApplication  
@ComponentScan(basePackages = "com.study.tmall")  
@MapperScan("com.study.tmall.product.mapper")  
@EnableFeignClients(basePackages = "com.study.tmall")  
@EnableDiscoveryClient  
**public class** ServiceProductApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ServiceProductApplication.**class**, args);  
 }  
}

产品管理

控制层

ProductInfoController

url：/admin/product/productInfo

分页条件显示产品

**前端传入参数：**current（起始页）limit（每页显示数量）productQueryVo（前端分类查询条件实体类，此实体类中包含分类id）

**请求方式：**get

**请求url：**/list/{current}/{limit}

**后端方法：**public Result findPageProductInfo(@PathVariable Long current, @PathVariable Long limit, ProductQueryVo productQueryVo)

**后端返回数据参数：**IPage<ProductInfo>（包含分页参数的集合）

ProductInfoController

// 分页条件显示商品  
@ApiOperation(value = "分页条件显示商品")  
@GetMapping("/list/{current}/{limit}")  
**public** Result findPageProductInfo(  
 @ApiParam(name = "current", value = "起始页", required = **true**)  
 @PathVariable Long current,  
  
 @ApiParam(name = "limit", value = "每页记录数", required = **true**)  
 @PathVariable Long limit,  
  
 @ApiParam(name = "productQueryVo", value = "条件查询商品vo")  
 ProductQueryVo productQueryVo){  
  
 Page<ProductInfo> page = **new** Page<>(current, limit);  
 IPage<ProductInfo> pageModule = productInfoService.findPageProductInfo(page, productQueryVo);  
 **return** Result.*ok*(pageModule);  
}

ProductInfoService

// 分页条件显示商品  
IPage<ProductInfo> findPageProductInfo(Page<ProductInfo> page, ProductQueryVo productQueryVo);

ProductInfoServiceImpl

*/\*\*  
 \* 分页条件显示商品  
 \** ***@param*** *page  
 \** ***@param*** *productQueryVo  
 \** ***@return*** *\*/*@Override  
**public** IPage<ProductInfo> findPageProductInfo(Page<ProductInfo> page, ProductQueryVo productQueryVo) {  
 QueryWrapper<ProductInfo> wrapper = **new** QueryWrapper<>();  
 String categoryId = productQueryVo.getCategoryId(); // 分类id  
 String keyword = productQueryVo.getKeyword(); // 商品名 或 商品小标题  
 BigDecimal lowPrice = productQueryVo.getLowPrice(); // 最低价格  
 BigDecimal highPrice = productQueryVo.getHighPrice(); // 最高价格  
 Date createTimeBegin = productQueryVo.getCreateTimeBegin(); // 创建时间区间  
 Date createTimeEnd = productQueryVo.getCreateTimeEnd(); // 创建时间区间  
  
 **if** (StringUtils.*isEmpty*(categoryId)){ // 不能让分类id为空，分类id为空则抛出异常  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 wrapper.eq("category\_id", categoryId);  
 **if** (!StringUtils.*isEmpty*(keyword)){  
 //wrapper.like("promote\_price", lowPrice);  
 // 这种写法相当于 有括号  
 // select \* from user where status = '1' and (name like '%张%' or phone like '%张%')  
 wrapper.and(qw->qw.like("name", keyword).or().like("sub\_title", keyword));  
 // 而下面这个相当于 无括号  
 // select \* from user where status = '1' and name like '%张%' or phone like '%张%'  
 //wrapper.like("name", keyword).or().like("sub\_title", keyword);  
 }  
 **if** (lowPrice != **null**){  
 wrapper.ge("promote\_price", lowPrice);  
 }  
 **if** (highPrice != **null**){  
 wrapper.le("promote\_price", highPrice);  
 }  
 **if** (createTimeBegin != **null**){  
 wrapper.ge("create\_time", createTimeBegin);  
 }  
 **if** (createTimeEnd != **null**){  
 wrapper.le("create\_time", createTimeEnd);  
 }  
  
 IPage<ProductInfo> productInfoIPage = baseMapper.selectPage(page, wrapper);  
 productInfoIPage.getRecords().stream().forEach(item -> {  
 // 把第一张缩略图装进去  
 **this**.packImage(item);  
 });  
 **return** productInfoIPage;  
}

// 把第一张缩略图装进去  
**private** ProductInfo packImage(ProductInfo productInfo) {  
 // 从数据库查询产品第一张缩略图的url  
 String imageUrl = productImageService.getFirstThumbnailImage(productInfo.getId());  
 productInfo.getParams().put("imageUrl", imageUrl);  
 **return** productInfo;  
}

ProductImageService

// 从数据库查询产品第一张缩略图的url  
String getFirstThumbnailImage(String id);

ProductImageServiceImpl

*/\*\*  
 \* 从数据库查询产品第一张缩略图的url  
 \** ***@param*** *id  
 \** ***@return*** *\*/*@Override  
**public** String getFirstThumbnailImage(String id) {  
 ProductImage firstThumbnailImage = baseMapper.getFirstThumbnailImage(id);  
 **if** (firstThumbnailImage == **null**){  
 **return** "";  
 }  
 **return** firstThumbnailImage.getUrl();  
}

ProductImageMapper

// 获取第一张缩略图  
ProductImage getFirstThumbnailImage(@Param("pid") String pid);

ProductImageMapper.xml

<**resultMap** id="productImageBean" type="com.study.tmall.model.product.ProductImage">  
 <**id** column="id" property="id"/>  
 <**result** column="product\_id" property="productId"/>  
 <**result** column="type" property="type"/>  
 <**result** column="url" property="url"/>  
 <**result** column="create\_time" property="createTime"/>  
 <**result** column="update\_time" property="updateTime"/>  
 <**result** column="is\_deleted" property="isDeleted"/>  
</**resultMap**>  
  
<**select** id="getFirstThumbnailImage" resultMap="productImageBean">  
 select  
 id,  
 product\_id,  
 type,  
 url,  
 is\_deleted  
 from  
 product\_image  
 where  
 product\_id=#{pid}  
 and is\_deleted=0  
 and type=0  
 limit 0,1  
</**select**>

批量删除产品

**前端传入参数：**idList（要批量删除的产品id集合）

**请求方式：**delete

**请求url：**/batchRemove

**后端方法：**public Result batchRemove(@RequestBody List<String> idList)

**后端返回数据参数：**ok()

ProductInfoController

// 批量删除商品  
@ApiOperation(value = "批量删除商品")  
@DeleteMapping("/batchRemove")  
**public** Result batchRemove(  
 @ApiParam(name = "idList", value = "商品id集合", required = **true**)  
 @RequestBody List<String> idList){  
  
 productInfoService.batchRemove(idList);  
 **return** Result.*ok*();  
}

ProductInfoService

// 批量删除商品  
**void** batchRemove(List<String> idList);

ProductInfoServiceImpl

*/\*\*  
 \* 批量删除商品  
 \** ***@param*** *idList  
 \*/*@Override  
**public void** batchRemove(List<String> idList) {  
 idList.stream().forEach(item -> {  
 **this**.removeProductById(item);  
 });  
}

删除产品

**前端传入参数：**id（要删除的产品id）

**请求方式：**delete

**请求url：**/{id}

**后端方法：**public Result remove(@PathVariable String id)

**后端返回数据参数：**ok()

ProductInfoController

// 删除商品  
@ApiOperation(value = "删除商品")  
@DeleteMapping("/{id}")  
**public** Result remove(  
 @ApiParam(name = "id", value = "商品id", required = **true**)  
 @PathVariable String id){  
  
 productInfoService.removeProductById(id);  
 **return** Result.*ok*();  
}

ProductInfoService

// 删除商品  
**void** removeProductById(String id);

ProductInfoServiceImpl

*/\*\*  
 \* 删除商品  
 \** ***@param*** *id  
 \*/*@Override  
**public void** removeProductById(String id) {  
 // 如果没有这个商品则抛出参数异常  
 ProductInfo productInfo = baseMapper.selectById(id);  
 **if** (productInfo == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
  
 // 删除商品下的属性值  
 propertyValueService.removeByPid(id);  
  
 // 删除商品下的图片  
 productImageService.removeImageByPid(id);  
  
 // 不能删除商品下的评论，因为评论属于用户  
 // 不能删除商品对应的订单，因为订单属于用户  
  
 // 删除商品  
 baseMapper.deleteById(id);  
}

PropertyValueService

// 根据商品id删除商品下的属性值  
**void** removeByPid(String productId);

PropertyValueServiceImpl

*/\*\*  
 \* 根据商品id删除商品下的属性值  
 \** ***@param*** *productId  
 \*/*@Override  
**public void** removeByPid(String productId) {  
 QueryWrapper<PropertyValue> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("product\_id", productId);  
 baseMapper.delete(wrapper);  
}

ProductImageService

// 根据商品id删除图片  
**void** removeImageByPid(String pid);

ProductImageServiceImpl

*/\*\*  
 \* 根据商品id删除图片  
 \** ***@param*** *pid  
 \*/*@Override  
**public void** removeImageByPid(String pid) {  
 // 查询数据库中是否存在数据  
 QueryWrapper<ProductImage> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("product\_id", pid);  
 List<ProductImage> productImages = baseMapper.selectList(wrapper);  
  
 **if** (productImages != **null**) { // 如果不为空则开始删除  
 productImages.stream().forEach(item -> {  
 // 删除fastDFS中的图片  
 String[] splitUrl = ImageUtil.*splitUrl*(item.getUrl());  
 FastDFSUtil.*delete*(splitUrl[1], splitUrl[2]);  
 });  
 // 逻辑删除数据库中的记录  
 baseMapper.delete(wrapper);  
 }  
}

添加产品

**前端传入参数：**productInfo（json格式的产品对象，无id）

**请求方式：**post

**请求url：**/save

**后端方法：**public Result save(@RequestBody ProductInfo productInfo)

**后端返回数据参数：**ok()

**备注：**添加产品应当创建对应的空的属性值，需要查询cateogyr库，用openFeign

ProductInfoController

// 添加商品  
@ApiOperation(value = "添加商品")  
@PostMapping("/save")  
**public** Result save(  
 @ApiParam(name = "productInfo", value = "商品实体类", required = **true**)  
 @RequestBody ProductInfo productInfo){  
  
 productInfoService.saveProduct(productInfo);  
 **return** Result.*ok*();  
}

ProductInfoService

// 添加商品  
**void** saveProduct(ProductInfo productInfo);

ProductInfoServiceImpl

*/\*\*  
 \* 添加商品  
 \** ***@param*** *productInfo  
 \*/*@Override  
**public void** saveProduct(ProductInfo productInfo) {  
 // 添加商品  
 baseMapper.insert(productInfo);  
  
 // 根据分类属性创建商品属性值，默认值空字符串  
 // 远程调用category查询分类属性  
 List<Property> propertyList = propertyFeignClient.showByCid(productInfo.getCategoryId());  
 propertyList.stream().forEach(item -> {  
 // 调用 PropertyValueService 保存商品属性值  
 PropertyValue propertyValue = **new** PropertyValue();  
 propertyValue.setPropertyId(item.getId());  
 propertyValue.setProductId(productInfo.getId());  
 propertyValueService.save(propertyValue);  
 });  
}

excel批量导入产品

**前端传入参数：**excel 文件

**请求方式：**post

**请求url：**/importData

**后端方法：**public Result batchSave(@RequestBody MultipartFile file)

**后端返回数据参数：**ok()

ProductInfoListener

**public class** ProductInfoListener **extends** AnalysisEventListener<ProductInfoEeVo> {  
 **private** ProductInfoMapper productInfoMapper; // 商品信息的mapper  
  
  
 // 有参构造，以便于在invoke方法中添加向数据库添加数据  
 **public** ProductInfoListener (ProductInfoMapper productInfoMapper) {  
 **this**.productInfoMapper = productInfoMapper;  
 }  
  
  
 // 一行一行的读，从第二行开始读取，第一行是表头  
 @Override  
 **public void** invoke(ProductInfoEeVo productInfoEeVo, AnalysisContext analysisContext) {  
 ProductInfo productInfo = **new** ProductInfo();  
 BeanUtils.*copyProperties*(productInfoEeVo, productInfo);  
 productInfoMapper.insert(productInfo); // 一行一行的向数据库添加商品信息  
 }  
  
 // 读取表头  
 @Override  
 **public void** invokeHeadMap(Map<Integer, String> headMap, AnalysisContext context) {  
 }  
  
 // 读取之后执行  
 @Override  
 **public void** doAfterAllAnalysed(AnalysisContext analysisContext) {  
  
 }  
}

ProductInfoController

// 根据上传的excel文档添加商品数据到数据库  
@ApiOperation(value = "利用excel批量添加商品信息")  
@PostMapping("/importData")  
**public** Result batchSave(  
 @ApiParam(name = "file", value = "excel文件", required = **true**)  
 @RequestBody MultipartFile file) {  
  
 productInfoService.importData(file);  
 **return** Result.*ok*();  
}

ProductInfoService

// 根据上传的excel文档添加商品数据到数据库  
**void** importData(MultipartFile file);

ProductInfoServiceImpl

*/\*\*  
 \* 根据上传的excel文档添加商品数据到数据库  
 \** ***@param*** *file  
 \*/*@Override  
**public void** importData(MultipartFile file) {  
 **try** {  
 EasyExcel.*read*(file.getInputStream(), ProductInfoEeVo.**class**, **new** ProductInfoListener(baseMapper)).sheet().doRead();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
}

产品信息导出为excel

**前端传入参数：**无

**请求方式：**get

**请求url：**/exportData

**后端方法：**public void exportDict(HttpServletResponse response)

**后端返回数据参数：**void，让用户点击后直接下载excel文件

ProductInfoController

// 把商品信息导出到excel文件中  
 @ApiOperation(value = "把商品信息导出到excel文件中", produces="application/octet-stream")  
 @GetMapping("/exportData")  
 **public void** exportDict(  
 @ApiParam(name = "response", value = "响应包")  
 HttpServletResponse response){  
  
 productInfoService.exportDictData(response);  
 }  
}

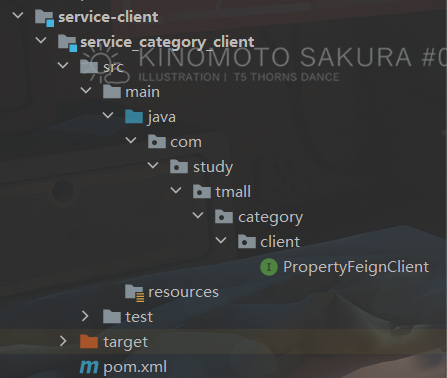
ProductInfoService

// 把商品信息导出到excel文件中  
**void** exportDictData(HttpServletResponse response);

ProductInfoServiceImpl

*/\*\*  
 \* 把商品信息导出到excel文件中  
 \** ***@param*** *response  
 \*/*@Override  
**public void** exportDictData(HttpServletResponse response) {  
 response.setContentType("application/vnd.ms-excel");  
 response.setCharacterEncoding("utf-8");  
 **try** {  
 // 设置文件名  
 String fileName = URLEncoder.*encode*("商品信息表", "utf-8");  
 // 参数1：设置头信息以下载方式打开  
 // 参数2：文件名  
 response.setHeader("Content-disposition", "attachment;filename="+ fileName + ".xlsx");  
 // 根据分类id查询商品信息  
 QueryWrapper<ProductInfo> wrapper = **new** QueryWrapper<>();  
 List<ProductInfo> productInfoList = baseMapper.selectList(wrapper);  
 List<ProductInfoEeVo> productInfoEeVos = **new** ArrayList<>();  
 **for** (ProductInfo productInfo : productInfoList) {  
 ProductInfoEeVo productInfoEeVo = **new** ProductInfoEeVo();  
 BeanUtils.*copyProperties*(productInfo, productInfoEeVo);  
 productInfoEeVos.add(productInfoEeVo);  
 }  
 EasyExcel.*write*(response.getOutputStream(), ProductInfoEeVo.**class**).sheet("productInfo").doWrite(productInfoEeVos);  
 } **catch** (UnsupportedEncodingException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
}

在server-client下创建server\_category\_client 模块



PropertyFeignClient

// 内部调用，显示对应分类下的所有属性  
@GetMapping("/admin/category/property/inner/show/{categoryId}")  
List<Property> showByCid(@PathVariable("categoryId") String categoryId);

PropertyController（server-category模块）

@Component  
@FeignClient(value = "service-category")  
**public interface** PropertyFeignClient {  
 // 内部调用，显示对应分类下的所有属性  
 @GetMapping("/admin/category/property/inner/show/{categoryId}")  
 List<Property> showByCid(@PathVariable("categoryId") String categoryId);  
}

PropertyService

// 根据分类id显示所有属性  
List<Property> showByCid(String categoryId);

PropertyServiceImpl

// 内部调用，显示对应分类下的所有属性  
@ApiOperation(value = "内部调用，显示对应分类下的所有属性")  
@GetMapping("/inner/show/{categoryId}") // inner 内部调用  
**public** List<Property> showByCid(  
 @ApiParam(name = "categoryId", value = "分类id", required = **true**)  
 @PathVariable String categoryId){  
  
 **return** propertyService.showByCid(categoryId);  
}

编辑产品

**前端传入参数：**productInfo（json格式的产品对象，有id）

**请求方式：**put

**请求url：**/update

**后端方法：**public Result update (@RequestBody ProductInfo productInfo)

**后端返回数据参数：**ok()

ProductInfoController

// 编辑商品  
@ApiOperation(value = "编辑商品")  
@PutMapping("/update")  
**public** Result update (  
 @ApiParam(name = "productInfo", value = "商品实体类", required = **true**)  
 @RequestBody ProductInfo productInfo){  
  
 productInfoService.updateById(productInfo);  
 **return** Result.*ok*();  
}

产品图片管理

控制层

ProductImageController

url：/admin/product/productImage

显示产品图片

**前端传入参数：**pid（产品的id）

**请求方式：**get

**请求url：**/show/{pid}

**后端方法：**public Result show(@PathVariable String pid)

**后端返回数据参数：**Map<String, List<ProductImage>> （根据图片类型进行分组，用stream实现list按属性分组转map）

ProductImageController

// 显示商品图片  
@ApiOperation(value = "显示商品图片")  
@GetMapping("/show/{productId}")  
**public** Result show(  
 @ApiParam(name = "productId", value = "商品id", required = **true**)  
 @PathVariable String productId){  
  
 Map<String, List<ProductImageReturnVo>> map = productImageService.showByProductId(productId);  
 **return** Result.*ok*(map);  
}

ProductImageService

// 显示商品图片  
Map<String, List<ProductImageReturnVo>> showByProductId(String productId);

ProductImageServiceImpl

*/\*\*  
 \* 显示商品图片  
 \** ***@param*** *productId  
 \** ***@return*** *\*/*@Override  
**public** Map<String, List<ProductImageReturnVo>> showByProductId(String productId) {  
 // 根据商品id查询出对应的所有商品图片  
 QueryWrapper<ProductImage> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("product\_id", productId);  
  
 List<ProductImage> productImages = baseMapper.selectList(wrapper);  
 // 让图片list集合 按 type 进行分组  
 Map<Integer, List<ProductImage>> collect = productImages.stream().collect(Collectors.*groupingBy*(ProductImage::getType));  
  
 // 对分组结果进行简化包装，前端只需要productImage 的 id 和 url，type可以根据map得到  
 Map<String, List<ProductImageReturnVo>> map = **new** HashMap<>();  
 Iterator<Integer> iterator = collect.keySet().iterator();  
 **while** (iterator.hasNext()){  
 Integer type = iterator.next();  
 List<ProductImageReturnVo> list = **new** ArrayList<>();  
 **for** (ProductImage pi : collect.get(type)) {  
 ProductImageReturnVo vo = **new** ProductImageReturnVo();  
 vo.setId(pi.getId());  
 vo.setUrl(pi.getUrl());  
 list.add(vo);  
 }  
 map.put(ImageTypeEnum.*getNameByType*(type), list);  
 }  
  
 **return** map;  
}

批量上传图片

**前端传入参数：**pid（产品的id） MultipartFile[]（文件流，数组）

**请求方式：**post

**请求url：**/batchUpload/{pid}

**后端方法：**public Result batchUploadImage (@PathVariable String pid, MultipartFile[] files)

**后端返回数据参数：**ok()

ProductImageController

// 批量上传图片  
@ApiOperation(value = "批量上传图片")  
@PostMapping("/batchUpload/{productId}/{type}")  
**public** Result batchUploadImage (  
 @ApiParam(name = "productId", value = "商品id", required = **true**)  
 @PathVariable String productId,  
  
 @ApiParam(name = "type", value = "图片类型", required = **true**)  
 @PathVariable Integer type,  
  
 @ApiParam(name = "files", value = "批量文件", required = **true**)  
 MultipartFile[] files){  
  
 ProductImage productImage = **new** ProductImage();  
 productImage.setProductId(productId);  
 productImage.setType(type);  
 productImageService.batchUploadImage(productImage, files);  
 **return** Result.*ok*();  
}

ProductImageService

// 批量上传图片  
**void** batchUploadImage(ProductImage productImage, MultipartFile[] files);

ProductImageServiceImpl

*/\*\*  
 \* 批量上传图片  
 \** ***@param*** *productImage  
 \** ***@param*** *files  
 \*/*@Override  
**public void** batchUploadImage(ProductImage productImage, MultipartFile[] files) {  
 // 先查询数据库中是否有这个商品  
 **try** {  
 ProductInfo productInfo = productInfoService.getById(productImage.getProductId());  
 **if** (files == **null** || productInfo == **null**){  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 上传图片  
 **for** (MultipartFile file : files) {  
 ProductImage newProductImage = **new** ProductImage(); // 这里重新创建新的对象，是为了生成新的id  
 newProductImage.setProductId(productImage.getProductId());  
 newProductImage.setType(productImage.getType());  
  
 String[] upload = FastDFSUtil.*upload*(file.getBytes(), ImageUtil.*getFileExtName*(file));  
 newProductImage.setUrl(ImageUtil.*compoundUrl*(upload));  
  
 // 保存到数据库  
 baseMapper.insert(newProductImage);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
}

批量删除图片

**前端传入参数：**idList（要批量删除图片的id）

**请求方式：**delete

**请求url：**/batchRemove

**后端方法：**public Result batchRemove(@RequestBody List<String> idList)

**后端返回数据参数：**ok()

ProductImageController

// 批量删除图片  
@ApiOperation(value = "批量删除图片")  
@DeleteMapping("/batchRemove")  
**public** Result batchRemove(  
 @ApiParam(name = "idList", value = "批量删除的图片id", required = **true**)  
 @RequestBody List<String> idList){  
  
 productImageService.batchRemove(idList);  
 **return** Result.*ok*();  
}

ProductImageService

// 批量删除图片  
**void** batchRemove(List<String> idList);

ProductImageServiceImpl

*/\*\*  
 \* 批量删除图片  
 \** ***@param*** *idList  
 \*/*@Override  
**public void** batchRemove(List<String> idList) {  
 idList.stream().forEach(item -> {  
 **this**.removeImageById(item);  
 });  
}

删除图片

**前端传入参数：**id（要删除图片的id）

**请求方式：**delete

**请求url：**/{id}

**后端方法：**public Result remove(@PathVariable String id)

**后端返回数据参数：**ok()

ProductImageController

// 删除图片  
@ApiOperation(value = "删除图片")  
@DeleteMapping("/{id}")  
**public** Result remove(  
 @ApiParam(name = "id", value = "图片id", required = **true**)  
 @PathVariable String id){  
  
 productImageService.removeImageById(id);  
 **return** Result.*ok*();  
}

ProductImageService

// 删除图片  
**void** removeImageById(String id);

ProductImageServiceImpl

*/\*\*  
 \* 删除图片  
 \** ***@param*** *id  
 \*/*@Override  
**public void** removeImageById(String id) {  
 // 查询数据库中是否存在该数据  
 ProductImage productImage = baseMapper.selectById(id);  
 **if** (productImage == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
  
 // 删除fastDFS中的图片  
 String[] splitUrl = ImageUtil.*splitUrl*(productImage.getUrl());  
 FastDFSUtil.*delete*(splitUrl[1], splitUrl[2]);  
  
 // 逻辑删除数据库中的记录  
 baseMapper.deleteById(productImage);  
}

产品属性管理

控制层

PropertyValueController

url：/admin/product/propertyValue

显示产品属性值

**前端传入参数：**pid（产品的id）

**请求方式：**get

**请求url：**/show/{pid}

**后端方法：**public Result show (@PathVariable String pid)

**后端返回数据参数：**Map<String, List<Property or PropertyValue>>，如果map的key为property，value就为List<Property>；如果map的key为propertyValue，value就为List<PropertyValue>

**备注：**要访问tmall\_cateogry库，要用到openFeign

PropertyValueController

// 显示商品属性值  
@ApiOperation(value = "显示商品属性值")  
@GetMapping("/show/{productId}")  
**public** Result show(  
 @ApiParam(name = "productId", value = "商品id", required = **true**)  
 @PathVariable String productId){  
  
 List<PropertyAndValueVo> list = propertyValueService.show(productId);  
 **return** Result.*ok*(list);  
}

PropertyValueService

// 显示商品属性值  
List<PropertyAndValueVo> show(String productId);

PropertyValueServiceImpl

*/\*\*  
 \* 显示商品属性值  
 \** ***@param*** *productId  
 \** ***@return*** *\*/*@Override  
**public** List<PropertyAndValueVo> show(String productId) {  
 // 查询出categoryId  
 ProductInfo productInfo = productInfoService.getById(productId);  
 **if** (productInfo == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
  
 // 用openFeign查询属性  
 List<Property> propertyList = propertyFeignClient.showByCid(productInfo.getCategoryId());  
 **if** (propertyList == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PROPERTY\_NULL*); // 未录入属性  
 }  
  
 // 查询出产品对应的propertyValue  
 QueryWrapper<PropertyValue> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("product\_id", productId);  
 List<PropertyValue> propertyValues = baseMapper.selectList(wrapper);  
  
 // 封装成List<PropertyAndValueVo>  
 List<PropertyAndValueVo> list = **new** ArrayList<>();  
 propertyList.stream().forEach(item -> {  
 list.add(**this**.packPropertyValue(propertyValues, item));  
 });  
  
 **return** list;  
}

*/\*\*  
 \* 封装属性名和属性值  
 \** ***@param*** *propertyValues  
 \** ***@param*** *property  
 \** ***@return*** *\*/***private** PropertyAndValueVo packPropertyValue(List<PropertyValue> propertyValues, Property property) {  
 PropertyAndValueVo vo = **new** PropertyAndValueVo();  
 **for** (**int** i = 0; i < propertyValues.size(); i++) {  
 PropertyValue propertyValue = propertyValues.get(i);  
 **if** (property.getId().equals(propertyValue.getPropertyId())) {  
 vo.setPropertyName(property.getName());  
 vo.setPropertyValue(propertyValue.getValue());  
 vo.setPropertyValueId(propertyValue.getId());  
 }  
 }  
 **return** vo;  
}

异步更新产品属性值

**前端传入参数：**propertyValue（属性值，含id）

**请求方式：**put

**请求url：**/update

**后端方法：**public Result update (@RequestBody PropertyValue propertyValue)

**后端返回数据参数：**ok()

PropertyValueController

// 异步更新商品属性值  
@ApiOperation(value = "异步更新产品属性值")  
@PutMapping("/update")  
**public** Result update (  
 @ApiParam(name = "propertyAndValueVo", value = "属性和属性值vo", required = **true**)  
 @RequestBody PropertyAndValueVo propertyAndValueVo){  
  
 propertyValueService.updatePropertyValueById(propertyAndValueVo);  
 **return** Result.*ok*();  
}

PropertyValueService

// 异步更新商品属性值  
**void** updatePropertyValueById(PropertyAndValueVo propertyAndValueVo);

PropertyValueServiceImpl

*/\*\*  
 \* 异步更新商品属性值  
 \** ***@param*** *propertyAndValueVo  
 \*/*@Override  
**public void** updatePropertyValueById(PropertyAndValueVo propertyAndValueVo) {  
 String value = propertyAndValueVo.getPropertyValue();  
 String id = propertyAndValueVo.getPropertyValueId();  
 PropertyValue propertyValue = **new** PropertyValue();  
 propertyValue.setId(id);  
 propertyValue.setValue(value);  
 baseMapper.updateById(propertyValue);  
}

service-user模块搭建

mapper：

UserInfoMapper.java

UserLoginRecordMapper.java

xml：

UserInfoMapper.xml

UserLoginRecordMapper.xml

service：

UserInfoService.java

UserLoginRecordService.java

impl：

UserInfoServiceImpl.java

UserLoginRecordServiceImpl.java

controller：

UserInfoController.java

application.yml

**server**:  
 **port**: 8203  
  
**spring**:  
 **application**:  
 **name**: service-user  
 **cloud**:  
 **nacos**:  
 **discovery**:  
 **server-addr**: 192.168.123.130:8848  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.123.130:3306/tmall\_user?useUnicode=true&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=GMT%2B8  
 **username**: root  
 **password**: 123321  
  
*# 指定mapper的.xml文件路径***mybatis-plus**:  
 **mapper-locations**: classpath:com/study/tmall/user/mapper/xml/\*.xml

启动类

@SpringBootApplication  
@ComponentScan(basePackages = {"com.study.tmall"})  
@MapperScan("com.study.tmall.user.mapper")  
@EnableDiscoveryClient  
**public class** ServiceUserApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ServiceUserApplication.**class**, args);  
 }  
}

用户管理

控制层

UserController

url：/admin/user/userInfo

分页条件显示用户

**前端传入参数：**current（起始页）limit（每页显示数量）userQueryVo（前端用户查询条件实体类，此实体类中包含分类id）

**请求方式：**get

**请求url：**/list/{current}/{limit}

**后端方法：**public Result findPageUserInfo(@PathVariable Long current, @PathVariable Long limit, UserQueryVo userQueryVo)

**后端返回数据参数：**IPage<UserInfo>（包含分页参数的集合）

UserInfoController

// 分页条件显示用户  
@ApiOperation(value = "分页条件显示用户")  
@GetMapping("/list/{current}/{limit}")  
**public** Result findPageUserInfo(  
 @ApiParam(name = "current", value = "起始页", required = **true**)  
 @PathVariable Long current,  
  
 @ApiParam(name = "limit", value = "每页记录数", required = **true**)  
 @PathVariable Long limit,  
  
 @ApiParam(name = "userQueryVo", value = "查询条件vo", required = **true**)  
 UserQueryVo userQueryVo){  
  
 Page<UserInfo> page = **new** Page<>(current, limit);  
 IPage<UserInfo> pageModule = userInfoService.findPageUserInfo(page, userQueryVo);  
 **return** Result.*ok*(pageModule);  
}

UserInfoService

// 分页条件显示用户  
IPage<UserInfo> findPageUserInfo(Page<UserInfo> page, UserQueryVo userQueryVo);

UserInfoServiceImpl

*/\*\*  
 \* 分页条件显示用户  
 \** ***@param*** *page  
 \** ***@param*** *userQueryVo  
 \** ***@return*** *\*/*@Override  
**public** IPage<UserInfo> findPageUserInfo(Page<UserInfo> page, UserQueryVo userQueryVo) {  
 String id = userQueryVo.getId(); // 用户id  
 String keyword = userQueryVo.getKeyword(); // 用户手机号 昵称 姓名  
 Integer status = userQueryVo.getStatus(); // 用户状态  
 Integer authStatus = userQueryVo.getAuthStatus(); // 用户认证状态  
  
 QueryWrapper<UserInfo> wrapper = **new** QueryWrapper<>();  
 **if** (!StringUtils.*isEmpty*(id)){  
 wrapper.eq("id", id);  
 }  
 **if** (!StringUtils.*isEmpty*(keyword)){ // 根据 用户手机号 昵称 姓名 做模糊查询  
 wrapper.and(wr->wr.like("name", keyword).or()  
 .like("nick\_name", keyword).or()  
 .like("phone", keyword));  
 }  
 **if** (status != **null**){  
 wrapper.eq("status", status);  
 }  
 **if** (authStatus != **null**){  
 wrapper.eq("auth\_status", authStatus);  
 }  
  
 IPage<UserInfo> userInfoIPage = baseMapper.selectPage(page, wrapper);  
 // 参数封装  
 userInfoIPage.getRecords().stream().forEach(item -> {  
 **this**.packageUserInfo(item);  
 });  
 **return** userInfoIPage;  
}

// 参数封装  
**private** UserInfo packageUserInfo(UserInfo userInfo) {  
 Integer authStatus = userInfo.getAuthStatus();  
 Integer status = userInfo.getStatus();  
 // 把用户状态的中文字符串包装进去  
 userInfo.getParams().put("authStatusStr", AuthStatusEnum.*getStatusNameByStatus*(authStatus));  
 userInfo.getParams().put("statusStr", UserLockStatusEnum.*getStatusNameByStatus*(status));  
 **return** userInfo;  
}

锁定\解锁 用户

**前端传入参数：**id（用户id），status（锁定状态 0：锁定 1：正常）

**请求方式：**put

**请求url：**/lock/{id}/{status}

**后端方法：**public Result lock(@PathVariable String id, @PathVariable Integer status)

**后端返回数据参数：**ok()

UserInfoController

// 锁定\解锁 用户  
@ApiOperation(value = "锁定 or 解锁 用户")  
@PutMapping("/lock/{id}/{status}")  
**public** Result lock(  
 @ApiParam(name = "id", value = "用户id", required = **true**)  
 @PathVariable String id,  
  
 @ApiParam(name = "status", value = "锁定状态", required = **true**)  
 @PathVariable Integer status){  
  
 userInfoService.lock(id, status);  
 **return** Result.*ok*();  
}

UserInfoService

// 锁定\解锁 用户  
**void** lock(String id, Integer status);

UserInfoServiceImpl

*/\*\*  
 \* 锁定\解锁 用户（0：锁定 1：正常）  
 \** ***@param*** *id  
 \** ***@param*** *status  
 \*/*@Override  
**public void** lock(String id, Integer status) {  
 // 查询数据库是否存在该用户  
 UserInfo userInfo = baseMapper.selectById(id);  
 **if** (userInfo == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 查询传来的状态是否合法  
 **if** (status!= UserLockStatusEnum.*UNLOCK*.getStatus().intValue()  
 && status != UserLockStatusEnum.*LOCK*.getStatus().intValue()){  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 如果传来的status和数据库中一直就不用更新数据库  
 **if** (userInfo.getStatus() == status){  
 **return**;  
 }  
 // 否则更新锁定状态  
 userInfo.setStatus(status);  
 baseMapper.updateById(userInfo);  
}

审批用户认证

**前端传入参数：**id（用户id），authStatus（认证状态 2：认证成功 -1：认证失败）

**请求方式：**update

**请求url：**/authUser/{id}/{authStatus}

**后端方法：**public Result authUser (@PathVariable String id, @PathVariable Integer authStatus)

**后端返回数据参数：**ok()

UserInfoController

// 审批用户认证  
@ApiOperation(value = "审批用户认证")  
@PutMapping("/authUser/{id}/{authStatus}")  
**public** Result authUser(  
 @ApiParam(name = "id", value = "用户id", required = **true**)  
 @PathVariable String id,  
  
 @ApiParam(name = "authStatus", value = "认证状态", required = **true**)  
 @PathVariable Integer authStatus){  
  
 userInfoService.authUser(id, authStatus);  
 **return** Result.*ok*();  
}

UserInfoService

// 审批用户认证  
**void** authUser(String id, Integer authStatus);

UserInfoServiceImpl

*/\*\*  
 \* 审批用户认证  
 \** ***@param*** *id  
 \** ***@param*** *authStatus  
 \*/*@Override  
**public void** authUser(String id, Integer authStatus) {  
 // 查询数据库是否存在该用户  
 UserInfo userInfo = baseMapper.selectById(id);  
 **if** (userInfo == **null**) {  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 查询传来的认证状态值是否合法  
 **if** (authStatus != AuthStatusEnum.*AUTH\_SUCCESS*.getStatus().intValue() // 通过认证的值  
 && authStatus != AuthStatusEnum.*AUTH\_FAIL*.getStatus().intValue()){ // 不通过认证的值  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 如果传来的authStatus和数据库中一致就不用更新数据库  
 **if** (userInfo.getAuthStatus() == authStatus){  
 **return**;  
 }  
 // 否则更新认证状态  
 userInfo.setAuthStatus(authStatus);  
 baseMapper.updateById(userInfo);  
}

service-order模块搭建

mapper：

OrderInfoMapper.java

OrderItemMapper.java

PaymentInfoMapper.java

RefundInfoMapper.java

xml：

OrderInfoMapper.xml

OrderItemMapper.xml

PaymentInfoMapper.xml

RefundInfoMapper. xml

service：

OrderInfoService.java

OrderItemService.java

PaymentInfoService.java

RefundInfoService.java

impl：

OrderInfoServiceImpl.java

OrderItemServiceImpl.java

PaymentInfoServiceImpl.java

RefundInfoServiceImpl.java

controller：

OrderInfoController.java

OrderItemController.java

application.yml

**server**:  
 **port**: 8204  
  
**spring**:  
 **application**:  
 **name**: service-order  
 **cloud**:  
 **nacos**:  
 **discovery**:  
 **server-addr**: 192.168.123.130:8848  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.123.130:3306/tmall\_order?useUnicode=true&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=GMT%2B8  
 **username**: root  
 **password**: 123321  
  
*# 指定mapper的.xml文件路径***mybatis-plus**:  
 **mapper-locations**: classpath:com/study/tmall/order/mapper/xml/\*.xml  
  
*# 日志打印级别  
#logging:  
# level:  
# root: debug*

启动类

@SpringBootApplication  
@ComponentScan(basePackages = "com.study.tmall")  
@MapperScan("com.study.tmall.order.mapper")  
@EnableDiscoveryClient  
@EnableFeignClients(basePackages = "com.study.tmall")  
**public class** ServiceOrderApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ServiceOrderApplication.**class**, args);  
 }  
}

订单管理

控制层

OrderInfoController

url：/admin/order/orderInfo

分页条件显示订单

**前端传入参数：**current（起始页）limit（每页显示数量）orderQueryVo（前端用户查询条件实体类，此实体类中包含分类id）

**请求方式：**get

**请求url：**/list/{current}/{limit}

**后端方法：**public Result findPageOrderInfo(@PathVariable Long current, @PathVariable Long limit, OrderQueryVo orderQueryVo)

**后端返回数据参数：**IPage<OrderInfo>（包含分页参数的集合）

OrderInfoController

// 分页条件显示订单  
@ApiOperation(value = "分页条件显示订单")  
@GetMapping("/list/{current}/{limit}")  
**public** Result findPageOrderInfo(  
 @ApiParam(name = "current", value = "起始页", required = **true**)  
 @PathVariable Long current,  
  
 @ApiParam(name = "limit", value = "每页记录", required = **true**)  
 @PathVariable Long limit,  
  
 @ApiParam(name = "orderQueryVo", value = "查询条件vo")  
 OrderQueryVo orderQueryVo){  
  
 Page<OrderInfo> page = **new** Page<>(current, limit);  
 IPage<OrderInfo> pageModule = orderInfoService.findPageOrderInfo(page, orderQueryVo);  
 **return** Result.*ok*(pageModule);  
}

OrderInfoService

// 分页条件显示订单  
IPage<OrderInfo> findPageOrderInfo(Page<OrderInfo> page, OrderQueryVo orderQueryVo);

OrderInfoServiceImpl

*/\*\*  
 \* 分页条件显示订单  
 \** ***@param*** *page  
 \** ***@param*** *orderQueryVo  
 \** ***@return*** *\*/*@Override  
**public** IPage<OrderInfo> findPageOrderInfo(Page<OrderInfo> page, OrderQueryVo orderQueryVo) {  
 Integer orderStatus = orderQueryVo.getOrderStatus(); // 订单状态  
 String userId = orderQueryVo.getUserId(); // 用户id  
 String outTradeNo = orderQueryVo.getOutTradeNo(); // 订单编号  
 Date createDateEnd = orderQueryVo.getCreateDateEnd(); // 订单创建日期结束  
 Date createDateBegin = orderQueryVo.getCreateDateBegin(); // 订单创建日期开始  
  
 QueryWrapper<OrderInfo> wrapper = **new** QueryWrapper<>();  
 **if** (orderStatus != **null**) {  
 wrapper.eq("order\_status", orderStatus);  
 }  
 **if** (!StringUtils.*isEmpty*(userId)) {  
 wrapper.eq("user\_id", userId);  
 }  
 **if** (!StringUtils.*isEmpty*(outTradeNo)) {  
 wrapper.eq("out\_trade\_no", outTradeNo);  
 }  
 **if** (createDateBegin != **null**) {  
 wrapper.ge("create\_date", createDateBegin);  
 }  
 **if** (createDateEnd != **null**) {  
 wrapper.le("create\_date", createDateEnd);  
 }  
  
 IPage<OrderInfo> orderInfoIPage = baseMapper.selectPage(page, wrapper);  
 // 每个订单都要获得用户信息，为了减少远程调用量，把这一页所有订单的用户id封装到list集合中  
 List<String> userIdList = **new** ArrayList<>();  
 orderInfoIPage.getRecords().stream().forEach(item -> {  
 userIdList.add(item.getUserId());  
 });  
 // 通过远程调用获得用户信息集合  
 List<UserInfo> userInfoList = userInfoFeignClient.listUserInfoOfInner(userIdList);  
  
 // 封装参数  
 orderInfoIPage.getRecords().stream().forEach(item -> {  
 **this**.packOrderInfo(item, userInfoList);  
 });  
 **return** orderInfoIPage;  
}

*/\*\*  
 \* 封装订单参数  
 \** ***@param*** *orderInfo  
 \*/***private** OrderInfo packOrderInfo(OrderInfo orderInfo, List<UserInfo> userInfoList) {  
 Integer number = 0; // 商品总数量  
 BigDecimal totalMoney = **new** BigDecimal(0);  
 Integer orderStatus = orderInfo.getOrderStatus();  
 String statusNameByStatus = OrderStatusEnum.*getStatusNameByStatus*(orderStatus);  
 // 获取订单项信息  
 List<OrderItem> orderItems = orderItemService.showByOrderId(orderInfo.getId());  
 // 封装总数量 和 总金额  
 **for** (OrderItem obj : orderItems) {  
 ProductInfo productInfo = (ProductInfo) obj.getParams().get("productInfo");  
 number += obj.getNumber();  
 totalMoney = totalMoney.add(productInfo.getPromotePrice());  
 }  
  
 // 额外数据封装  
 orderInfo.getParams().put("number", number); // 商品总数量  
 orderInfo.getParams().put("totalMoney", totalMoney); // 商品总金额  
 orderInfo.getParams().put("orderItems", orderItems); // 订单项  
 orderInfo.getParams().put("orderStatusStr", statusNameByStatus); // 订单状态中文显示  
  
 // 把用户信息封装进去  
 **if** (orderInfo.getUserId() != **null**) {  
 Iterator<UserInfo> it = userInfoList.iterator();  
 **while** (it.hasNext()) {  
 UserInfo userInfo = it.next();  
 **if** (orderInfo.getUserId().equals(userInfo.getId())) {  
 **if** (!StringUtils.*isEmpty*(userInfo.getNickName())) { // 如果昵称不为空就把昵称做userName  
 orderInfo.getParams().put("customerName", userInfo.getNickName());  
 } **else if** (!StringUtils.*isEmpty*(userInfo.getPhone())) { // 如果手机号不为空就把昵称做手机号  
 orderInfo.getParams().put("customerName", userInfo.getPhone());  
 }  
 **break**;  
 }  
 }  
 }  
 // *TODO 后续根据前端需求继续完善* **return** orderInfo;  
}

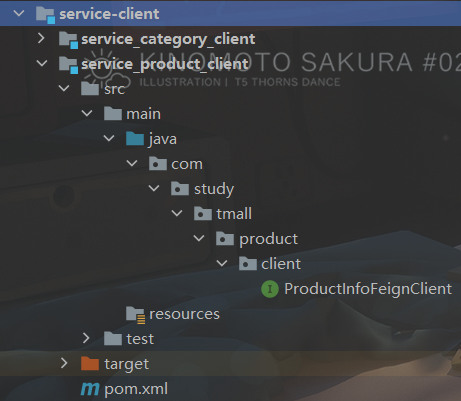
OrderItemService

// 显示订单项  
List<OrderItem> showByOrderId(String orderId);

OrderItemServiceImpl

*/\*\*  
 \* 显示订单项  
 \** ***@param*** *orderId  
 \** ***@return*** *\*/*@Override  
**public** List<OrderItem> showByOrderId(String orderId) {  
 // 根据订单id查询出订单项  
 QueryWrapper<OrderItem> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("order\_id", orderId);  
 List<OrderItem> orderItems = baseMapper.selectList(wrapper);  
  
 // 根据订单项中的商品id查询出产品，用远程调用访问product模块  
 List<String> productInfoIdList = **new** ArrayList<>();  
 orderItems.stream().forEach(item -> {  
 productInfoIdList.add(item.getProductId());  
 });  
  
 // 传入list集合是为了减少远程调用，提高处理速度  
 List<ProductInfo> productInfoList = productInfoFeignClient.listProductInfoById(productInfoIdList);  
  
 // 把商品信息放到 orderItems 中  
 **for** (**int** i = 0; i < orderItems.size(); i++) {  
 orderItems.get(i).getParams().put("productInfo", productInfoList.get(i));  
 }  
  
 **return** orderItems;  
}

在server-client下创建server\_product\_client 模块



ProductInfoFeignClient

// 根据id查询商品（内部调用）  
@PostMapping("/admin/product/productInfo/inner/list")  
List<ProductInfo> listProductInfoById(@RequestBody List<String> idList);

ProductInfoController（service-product模块）

// 根据id查询商品（内部调用）  
@ApiOperation(value = "根据id查询商品（内部调用）")  
@PostMapping("/inner/list")  
**public** List<ProductInfo> listProductInfoById(  
 @ApiParam(name = "idList", value = "商品id集合", required = **true**)  
 @RequestBody List<String> idList){  
  
 **return** productInfoService.listProductInfoById(idList);  
}

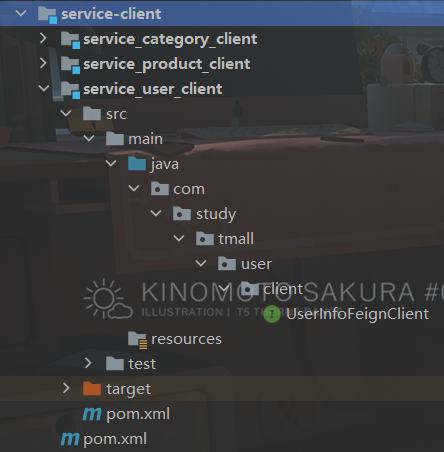
ProductInfoService

// 根据id查询商品（内部调用）  
List<ProductInfo> listProductInfoById(List<String> idList);

ProductInfoServiceImpl

*/\*\*  
 \* 根据id查询商品（内部调用）  
 \** ***@param*** *idList  
 \** ***@return*** *\*/*@Override  
**public** List<ProductInfo> listProductInfoById(List<String> idList) {  
 // 如果idList没有就返回空  
 **if** (idList == **null** || idList.size() == 0) **return null**;  
 List<ProductInfo> list = **new** ArrayList<>();  
 // 遍历封装  
 idList.stream().forEach(item -> {  
 ProductInfo productInfo = baseMapper.selectById(item);  
 **if** (productInfo != **null**) {  
 **this**.packImage(productInfo); // 把图片搞进去  
 }  
 // 即使查询出来为空也装入集合，保证数据不错位  
 list.add(productInfo);  
 });  
 **return** list;  
}

在server-client下创建server\_user\_client 模块



UserInfoFeignClient

// 获取用户基本信息，内部调用  
@ApiModelProperty(value = "获取用户基本信息，内部调用")  
@PostMapping("/admin/user/userInfo/inner/listUserInfo")  
List<UserInfo> listUserInfoOfInner(@RequestBody List<String> idList);

UserInfoController（service-user模块）

// 获取用户基本信息，内部调用  
@ApiModelProperty(value = "获取用户基本信息，内部调用")  
@PostMapping("/inner/listUserInfo")  
**public** List<UserInfo> listUserInfoOfInner(  
 @ApiParam(name = "idList", value = "用户id集合", required = **true**)  
 @RequestBody List<String> idList){  
  
 List<UserInfo> userInfoList = userInfoService.listUserInfo(idList);  
 **return** userInfoList;  
}

UserInfoService

// 获取用户基本信息，内部调用  
List<UserInfo> listUserInfo(List<String> idList);

UserInfoServiceImpl

*/\*\*  
 \* 获取用户基本信息，内部调用  
 \** ***@param*** *idList  
 \** ***@return*** *\*/*@Override  
**public** List<UserInfo> listUserInfo(List<String> idList) {  
 List<UserInfo> userInfoList = **new** ArrayList<>();  
 idList.stream().forEach(id -> {  
 UserInfo userInfo = baseMapper.selectById(id);  
 **if** (userInfo != **null**) { // 如果查出来的用户不为空，则把重要信息做置空处理  
 userInfo.setPassword(**null**);  
 // *TODO 根据未来需求把不需要的参数再进一步置空* }  
 userInfoList.add(userInfo);  
 });  
 **return** userInfoList;  
}

订单发货

**前端传入参数：**id（订单的id）

**请求方式：**put

**请求url：**/[deliverGoods](javascript:;) /{id}

**后端方法：**public Result [deliverGoods](javascript:;)(@PathVariable Integer id)

**后端返回数据参数：**Result.ok()（返回订单信息）

OrderInfoController

// 订单发货  
@ApiOperation(value = "订单发货")  
@PutMapping("/deliverGoods/{id}")  
**public** Result deliverGoods(  
 @ApiParam(name = "id", value = "订单id", required = **true**)  
 @PathVariable String id){  
  
 orderInfoService.deliverGoods(id);  
 **return** Result.*ok*();  
}

OrderInfoService

// 订单发货  
**void** deliverGoods(String id);

OrderInfoServiceImpl

*/\*\*  
 \* 订单发货  
 \** ***@param*** *id  
 \*/*@Override  
**public void** deliverGoods(String id) {  
 OrderInfo orderInfo = baseMapper.selectById(id);  
 **if** (orderInfo == **null**) { // 如果查不出来就抛出参数异常  
 **throw new** TmallException(ResultCodeEnum.*PARAM\_ERROR*);  
 }  
 // 检查订单是否已经付款，处于待发货状态  
 Integer orderStatus = orderInfo.getOrderStatus();  
 **if** (orderStatus != OrderStatusEnum.*WAIT\_SHIPMENTS*.getStatus()) {  
 **throw new** TmallException(ResultCodeEnum.*DELIVER\_GOODS\_FAIL*);  
 }  
 // 更新发货时间，更新订单状态  
 orderInfo.setDeliveryDate(**new** Date());  
 orderInfo.setOrderStatus(OrderStatusEnum.*WAIT\_TAKE\_GOODS*.getStatus());  
 baseMapper.updateById(orderInfo);  
}

订单项管理

控制层

OrderInfoController

url：/admin/order/orderItem

显示订单项

**前端传入参数：**id（订单的id）

**请求方式：**get

**请求url：**/show/{id}

**后端方法：**public Result getOrderItem(@PathVariable Integer id)

**后端返回数据参数：**List<OrderItem>（返回订单信息，即返回OrderItem）

**备注：**这里的查看订单详情只是显示这个订单下的订单项，需要远程调用product模块

OrderItemController

// 显示订单项  
@ApiOperation(value = "显示订单项")  
@GetMapping("/show/{orderId}")  
**public** Result getOrderItem(  
 @ApiParam(name = "orderId", value = "订单的id", required = **true**)  
 @PathVariable String orderId){  
  
 List<OrderItem> list = orderItemService.showByOrderId(orderId);  
 **return** Result.*ok*(list);  
}

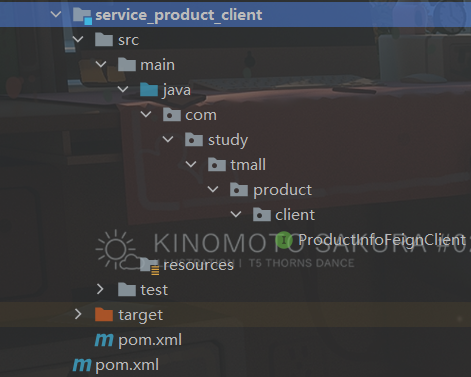
OrderItemService

// 显示订单项  
List<OrderItem> showByOrderId(String orderId);

OrderItemServiceImpl

*/\*\*  
 \* 显示订单项  
 \** ***@param*** *orderId  
 \** ***@return*** *\*/*@Override  
**public** List<OrderItem> showByOrderId(String orderId) {  
 // 根据订单id查询出订单项  
 QueryWrapper<OrderItem> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("order\_id", orderId);  
 List<OrderItem> orderItems = baseMapper.selectList(wrapper);  
  
 // 根据订单项中的商品id查询出产品，用远程调用访问product模块  
 List<String> productInfoIdList = **new** ArrayList<>();  
 orderItems.stream().forEach(item -> {  
 productInfoIdList.add(item.getProductId());  
 });  
  
 // 传入list集合是为了减少远程调用，提高处理速度  
 List<ProductInfo> productInfoList = productInfoFeignClient.listProductInfoById(productInfoIdList);  
  
 // 把商品信息放到 orderItems 中  
 **for** (**int** i = 0; i < orderItems.size(); i++) {  
 orderItems.get(i).getParams().put("productInfo", productInfoList.get(i));  
 }  
  
 **return** orderItems;  
}

在server-client下创建server\_product\_client 模块



ProductInfoFeignClient

@Component  
@FeignClient(value = "service-product")  
**public interface** ProductInfoFeignClient {  
 // 根据id查询商品（内部调用）  
 @PostMapping("/admin/product/productInfo/inner/list")  
 List<ProductInfo> listProductInfoById(@RequestBody List<String> idList);  
}

ProductInfoController（service-product模块）

// 根据id查询商品（内部调用）  
@ApiOperation(value = "根据id查询商品（内部调用）")  
@PostMapping("/inner/list")  
**public** List<ProductInfo> listProductInfoById(  
 @ApiParam(name = "idList", value = "商品id集合", required = **true**)  
 @RequestBody List<String> idList){  
  
 **return** productInfoService.listProductInfoById(idList);  
}

ProductInfoService

// 根据id查询商品（内部调用）  
List<ProductInfo> listProductInfoById(List<String> idList);

ProductInfoServiceImpl

*/\*\*  
 \* 根据id查询商品（内部调用）  
 \** ***@param*** *idList  
 \** ***@return*** *\*/*@Override  
**public** List<ProductInfo> listProductInfoById(List<String> idList) {  
 // 如果idList没有就返回空  
 **if** (idList == **null** || idList.size() == 0) **return null**;  
 List<ProductInfo> list = **new** ArrayList<>();  
 // 遍历封装  
 idList.stream().forEach(item -> {  
 ProductInfo productInfo = baseMapper.selectById(item);  
 **if** (productInfo != **null**) {  
 **this**.packImage(productInfo); // 把图片搞进去  
 }  
 // 即使查询出来为空也装入集合，保证数据不错位  
 list.add(productInfo);  
 });  
 **return** list;  
}

[service-statistics](https://github.com/Executioner2/ShangYiTong/tree/master/yygh-parent/service/service-statistics)模块搭建（数据视图）

controller：

StatisticsController.java

控制层

StatisticsController

url：/admin/statistics

按分类显示产品热销数据

**前端传入参数：** categoryCountQueryVo（分类热销产品查询）

**请求方式：**get

**请求url：**/getCountMap

**后端方法：**public Result getCountMap (@RequestBody CategoryCountQueryVo categoryCountQueryVo)

**后端返回数据参数：**做出前端效果再定

**备注：**要访问tmall\_cateogry、tmall\_product，要用到openFeign