# 天猫前台系统后端开发手册

V 1.0

2Executioner

home：首页

主要业务

**展示所有分类及其分类产品8\*8的小标题首段（需要64个产品）**

**每个分类展示五个热销产品**

**把查询结果存入redis便于数据二次使用**

控制层

CategoryInfoApi

url: /api/category/categoryInfo

ProductInfoApi

url: /api/product/productInfo

显示分类列表和热销商品

**前端传入参数：**无

**请求方式：**get

**请求url：**/list

**后端方法：**public Result listCategoryInfo()

**后端返回数据参数：**List<CategoryInfo> （首页分类菜单，包括分类名，分类id，8\*8小标题首段，和小标题首段对应的商品id）和 热销商品

CategoryInfoApi

// 首页显示分类列表和热销商品  
@ApiOperation(value = "首页显示分类列表和热销商品")  
@GetMapping("/list")  
**public** Result listCategoryInfo() {  
 List<CategoryInfo> list = categoryInfoService.homeListCategoryInfo();  
 **return** Result.*ok*(list);  
}

CategoryInfoService

// 首页显示分类列表和热销商品  
List<CategoryInfo> homeListCategoryInfo();

CategoryInfoServiceImpl

*/\*\*  
 \* 首页显示分类列表和热销商品  
 \*/*@Override  
**public** List<CategoryInfo> homeListCategoryInfo() {  
 // 查询出所有分类  
 List<CategoryInfo> categoryInfos = baseMapper.selectList(**null**);  
  
 **if** (categoryInfos == **null**) {  
 **return null**;  
 }  
  
 // 为了减少后面的远程调用次数，把所有分类的id封装到一个list集合中  
 List<String> idList = **new** ArrayList<>();  
 categoryInfos.stream().forEach(item -> {  
 idList.add(item.getId());  
 });  
  
 // 根据分类分别按销量查询出热销的前5个商品信息，包含第一张缩略图url（远程调用）  
 Map<String, List<ProductInfo>> hotMap = productInfoFeignClient.listProductInfoHot(idList);  
  
 // 根据分类分别按销量排序查询出前64个商品的小标题和id（远程调用）  
 Map<String, List<ProductInfo>> productInfoSubTitleList = productInfoFeignClient.listProductInfoSubTitle(idList);  
  
 // 封装数据  
 **for** (**int** x = 0; x < idList.size(); x++) {  
 String id = idList.get(x);  
  
 // 热销产品  
 List<ProductInfo> hotList = hotMap.get(id);  
 categoryInfos.get(x).getParams().put("hotList", hotList);  
  
 // 把这64个商品的小标题做包装，以 8\*8 的方式存储  
 List<ProductInfo> productInfoList = productInfoSubTitleList.get(id);  
 // 如果查询出来 productInfoList 为空则说明该分类没有商品，那么就跳过则此循环  
 **if** (productInfoList == **null**) {  
 **continue**;  
 }  
 // 浮动菜单  
 List<List<Map<String, String>>> floatMenu = **new** ArrayList<>();  
 List<Map<String, String>> rows = **null**; // 行  
 **for** (**int** i = 0; i < productInfoList.size(); i++) {  
 ProductInfo productInfo = productInfoList.get(i);  
 **if** (i % 8 == 0) { // 一行8个，循环到8个就new一个新的行  
 rows = **new** ArrayList<>(); // 新的行  
 floatMenu.add(rows);  
 }  
 String pid = productInfo.getId(); // productId  
 String subTitle = productInfo.getSubTitle(); // 小标题  
 // 把小标题进行拆分，取首段  
 **if** (StringUtils.*isEmpty*(subTitle)) { // 如果为空则直接置为空字符串  
 subTitle = "";  
 } **else** {  
 subTitle = subTitle.split(" ")[0]; // 取第一段  
 }  
 // 存入rows的map中  
 Map<String, String> item = **new** HashMap<>();  
 item.put("id", pid);  
 item.put("subTitle", subTitle);  
 rows.add(item);  
 }  
 // 封装到 categoryInfos  
 categoryInfos.get(x).getParams().put("floatMenu", floatMenu);  
 }  
  
 **return** categoryInfos;  
}

ProductInfoFeignClient（service\_product\_client模块）

// 返回商品小标题集合（内部调用）  
@PostMapping("/api/product/productInfo/inner/list/simple")  
Map<String, List<ProductInfo>> listProductInfoSubTitle(@RequestBody List<String> idList);  
  
// 显示每个分类前五个热销商品（内部调用）  
@PostMapping("/api/product/productInfo/inner/list/hot")  
Map<String, List<ProductInfo>> listProductInfoHot(@RequestBody List<String> idList);

ProductInfoApi（service-product模块）

// 返回商品小标题集合（内部调用）  
@ApiOperation(value = "内部调用，返回商品小标题集合")  
@PostMapping("/inner/list/simple")  
**public** Map<String, List<ProductInfo>> listProductInfoSubTitle(  
 @ApiParam(name = "idList", value = "分类id集合", required = **true**)  
 @RequestBody List<String> idList) {  
  
 **return** productInfoService.listProductInfoSubTitle(idList);  
}  
  
// 显示每个分类前五个热销商品（内部调用）  
@ApiOperation(value = "显示每个分类前五个热销商品（内部调用）")  
@PostMapping("/inner/list/hot")  
**public** Map<String, List<ProductInfo>> listProductInfoHot(  
 @ApiParam(name = "idList", value = "分类id集合", required = **true**)  
 @RequestBody List<String> idList) {  
  
 **return** productInfoService.listProductInfoHot(idList);  
}

ProductInfoService

// 返回商品小标题集合 内部调用  
Map<String, List<ProductInfo>> listProductInfoSubTitle(List<String> idList);  
  
// 显示每个分类前五个热销商品（内部调用）  
Map<String, List<ProductInfo>> listProductInfoHot(List<String> idList);

ProductInfoServiceImpl

*/\*\*  
 \* 返回商品小标题集合 内部调用  
 \** ***@param*** *idList  
 \** ***@return*** *\*/*@Override  
**public** Map<String, List<ProductInfo>> listProductInfoSubTitle(List<String> idList) {  
 List<ProductInfo> productInfoList = baseMapper.listProductInfoSubTitle(idList);  
 **if** (productInfoList == **null**) {  
 **return null**;  
 }  
 // 对查询出来的结果按categoryId进行分组  
 Map<String, List<ProductInfo>> collect = productInfoList.stream().collect(Collectors.*groupingBy*(ProductInfo::getCategoryId));  
 **return** collect;  
}  
  
*/\*\*  
 \* 显示每个分类前五个热销商品（内部调用）  
 \** ***@param*** *idList  
 \** ***@return*** *\*/*@Override  
**public** Map<String, List<ProductInfo>> listProductInfoHot(List<String> idList) {  
 // 查询五条记录  
 Page<ProductInfo> page = **new** Page<>(1, 5);  
 QueryWrapper<ProductInfo> wrapper = **new** QueryWrapper<>();  
  
 // 放入到map集合中，分类id作为key，查询结果的Records作为value  
 Map<String, List<ProductInfo>> map = **new** HashMap<>();  
 **for** (String str : idList) {  
 // 按月销量进行降序排序  
 wrapper.eq("category\_id", str);  
 wrapper.orderByDesc("monthly\_sales");  
 IPage<ProductInfo> pageModule = baseMapper.selectPage(page, wrapper);  
 wrapper.clear(); // 清空条件  
 // 对商品数据再进行处理  
 pageModule.getRecords().stream().forEach(item -> {  
 // 把第一张缩略图封装进去  
 **this**.packImage(item);  
 // 把每个商品的名称缩短，方便前端显示（显示商品名称的前25个字符）  
 item.setName(item.getName().substring(0, 25));  
 });  
 map.put(str, pageModule.getRecords());  
 }  
 **return** map;  
}

category：分类页

主要业务

**展示分类图片**

**分页展示分类下所有产品**

**可按价格进行查询**

**可按条件进行排序**

控制层

CategoryInfoApi

url: /api/category/categoryInfo

ProductInfoApi

url: /api/product/productInfo

显示分类图片

**前端传入参数：**categoryId（分类id）

**请求方式：**get

**请求url：**/{id}

**后端方法：**public Result getCategoryInfo()

**后端返回数据参数：** CategoryInfo (分类信息)

CategoryInfoApi

// 根据id查询出分类  
@ApiOperation(value = "根据id查询出分类")  
@GetMapping("/{id}")  
**public** Result getCategoryInfo(  
 @ApiParam(value = "id", name = "分类id", required = **true**)  
 @PathVariable String id) {  
 CategoryInfo categoryInfo = categoryInfoService.getById(id);  
 **return** Result.*ok*(categoryInfo);  
}

条件查询商品

**前端传入参数：**current（起始页），limit（每页大小），searchObj（查询对象，包含分类id）

**请求方式：**get

**请求url：**/list/{current}/{limit}

**后端方法：**public Result listProductInfo(@PathVariable Long current, @PathVariable Long limit, @RequestBody ProductInfoFrontQueryVo productInfoFrontQueryVo)

**后端返回数据参数：**IPage<ProductInfo> （商品信息List集合）

ProductInfoApi

// 分类页，根据分类id和查询条件显示商品  
@ApiOperation(value = "分类页，根据分类id和查询条件显示商品")  
@GetMapping("/list/{current}/{limit}")  
**public** Result listProductInfo(  
 @ApiParam(name = "current", value = "起始页", required = **true**)  
 @PathVariable Long current,  
  
 @ApiParam(name = "limit", value = "每页大小", required = **true**)  
 @PathVariable Long limit,  
  
 @ApiParam(name = "productInfoFrontQueryVo", value = "前台分类页商品查询对象", required = **true**)  
 ProductInfoFrontQueryVo productInfoFrontQueryVo) {  
  
 Page<ProductInfo> page = **new** Page<>(current, limit);  
 IPage<ProductInfo> pageModule = productInfoService.listProductInfo(page, productInfoFrontQueryVo);  
 **return** Result.*ok*(pageModule);  
}

ProductInfoService

// 分类页，根据分类id和查询条件显示商品  
IPage<ProductInfo> listProductInfo(Page<ProductInfo> page, ProductInfoFrontQueryVo productInfoFrontQueryVo);

ProductInfoServiceImpl

*/\*\*  
 \* 分类页，根据分类id和查询条件显示商品  
 \** ***@param*** *page  
 \** ***@param*** *productInfoFrontQueryVo  
 \** ***@return*** *\*/*@Override  
**public** IPage<ProductInfo> listProductInfo(Page<ProductInfo> page, ProductInfoFrontQueryVo productInfoFrontQueryVo) {  
 String categoryId = productInfoFrontQueryVo.getCategoryId(); // 分类id  
 BigDecimal lowPrice = productInfoFrontQueryVo.getLowPrice(); // 最低价格  
 BigDecimal highPrice = productInfoFrontQueryVo.getHighPrice(); // 最高价格  
 Integer sortField = productInfoFrontQueryVo.getSortField(); // 排序字段  
 Integer lostSortField = productInfoFrontQueryVo.getLostSortField(); // 上一次排序字段  
 Integer sortType = productInfoFrontQueryVo.getSortType(); // 排序方式  
  
 String typeBySort = SortTypeEnum.*getTypeBySort*(sortType); // 取得排序方式的字符  
 String sortFieldStr = ProductSortStatusEnum.*getFieldByNumber*(sortField); // 取得本次排序字段  
 String lostSortFieldStr = ProductSortStatusEnum.*getFieldByNumber*(lostSortField); // 取得上次排序字段  
  
 QueryWrapper<ProductInfo> wrapper = **new** QueryWrapper<>();  
 wrapper.eq("category\_id", categoryId);  
 // 设置价格区间  
 **if** (lowPrice != **null**) {  
 wrapper.ge("promote\_price", lowPrice);  
 }  
 **if** (highPrice != **null**) {  
 wrapper.le("promote\_price", highPrice);  
 }  
 // 定义排序条件  
 **if** (!StringUtils.*isEmpty*(sortFieldStr) && // 首要条件是本次排序字段不为空 并且 不是默认排序  
 sortField != ProductSortStatusEnum.*DEFAULT*.getNumber()) {  
  
 **if** (SortTypeEnum.*ASC*.getType().equals(typeBySort)) { // 是升序排序  
 wrapper.orderByAsc(sortFieldStr);  
 } **else** { // 否则就是降序排序  
 wrapper.orderByDesc(sortFieldStr);  
 }  
 } **else if**(!StringUtils.*isEmpty*(lostSortFieldStr) && // 如果本次排序条件为空，上次排序条件不为空也不是默认排序，则按上次排序条件排序  
 lostSortField != ProductSortStatusEnum.*DEFAULT*.getNumber()) {  
  
 **if** (SortTypeEnum.*ASC*.getType().equals(typeBySort)) { // 是升序排序  
 wrapper.orderByAsc(lostSortFieldStr);  
 } **else** { // 否则就是降序排序  
 wrapper.orderByDesc(lostSortFieldStr);  
 }  
 }  
 IPage<ProductInfo> productInfoIPage = baseMapper.selectProductInfoPageOrderBy(page, wrapper);  
  
 // 对商品数据再进行处理  
 productInfoIPage.getRecords().stream().forEach(item -> {  
 // 把第一张缩略图封装进去  
 **this**.packImage(item);  
 // 把每个商品的名称缩短，方便前端显示（显示商品名称的前25个字符）  
 item.setName(item.getName().substring(0, 25));  
 // 评价数量  
 QueryWrapper<Review> queryWrapper = **new** QueryWrapper<>();  
 queryWrapper.eq("product\_id", item.getId());  
 **int** count = reviewService.count(queryWrapper);  
 item.getParams().put("reviewNumber", count);  
 });  
 **return** productInfoIPage;  
}

ProductInfoMapper

// 查询出所有商品以及按指定方式排序，因为要查询出评价数所以自己写多表查询语句  
IPage<ProductInfo> selectProductInfoPageOrderBy(IPage<ProductInfo> page, @Param(Constants.*WRAPPER*) Wrapper<ProductInfo> queryWrapper);

ProductInfoMapper.xml

<**select** id="selectProductInfoPageOrderBy" resultMap="productInfoBean">  
 SELECT  
 id,  
 category\_id,  
 name,  
 sub\_title,  
 orignal\_price,  
 promote\_price,  
 stock,  
 sales\_volume,  
 monthly\_sales,  
 create\_time,  
 update\_time,  
 is\_deleted  
 FROM  
 product\_info p  
 LEFT JOIN (  
 SELECT  
 product\_id,  
 count(\*) AS reviews  
 FROM  
 review  
 GROUP BY  
 product\_id  
 ) r ON r.product\_id = p.id  
 ${ew.customSqlSegment}  
</**select**>

productInfo：产品页

search：搜索页

login：登录页

regist：注册页

[account](javascript:;)Set ：账户设置页

orderItem：购物车

forebudy：下单页

pay：支付页

paySucceed：支付成功页

orderInfo：订单页

confirmReceipt：确认收货页

[tradeSuccessfully](javascript:;)：交易成功页

review：产品评价页