## 分类管理模块介绍

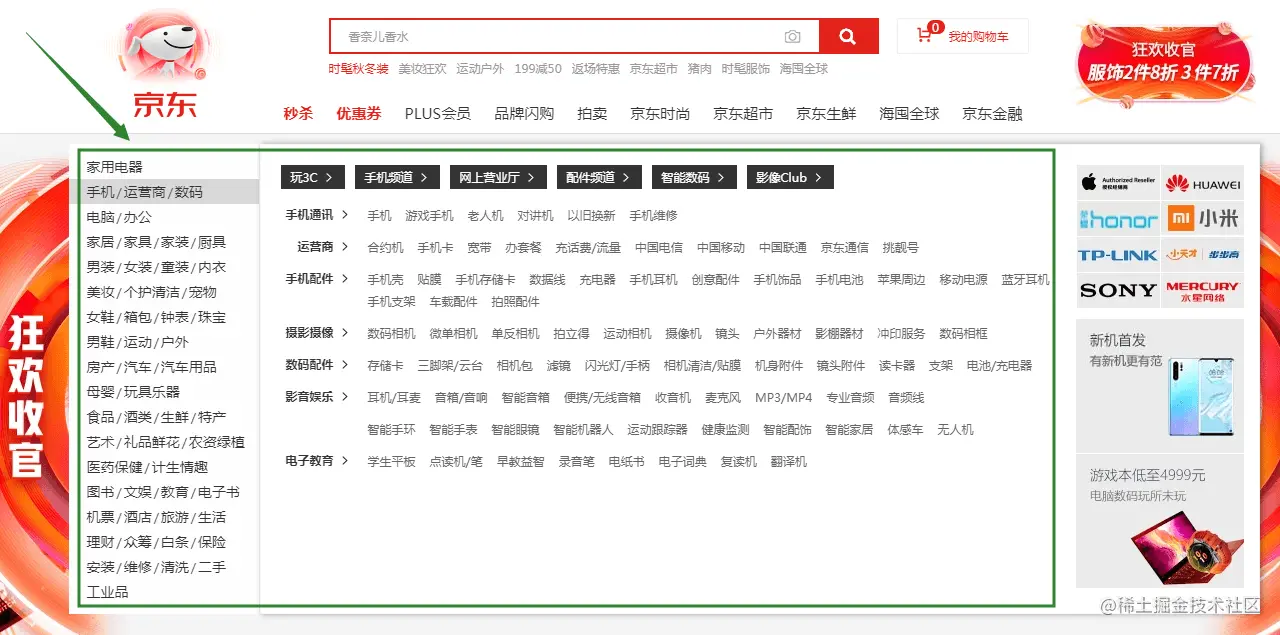
#### 商品分类

顾名思义，分类就是将商品进行分门别类，那部分商品是衣服，这部分商品是数码产品，另外一部分商品是美妆或者护理产品，这样处理的好处就是方便用户进行筛选和辨别，以天猫商城和京东商城为例，在商城首屏中很大一部分版面都可以进行分类的选择，在这里我们可以通过分类的设置快速的进入对应的商品列表页面进行商品选择。

* 天猫商城分类选择



* 京东商城分类选择



#### 分类层级

通过观察以上两个商城的分类模块，我们看出它们对于分类层级的设计，分类层级的不同就表示我们需要在商品分类上需要继续做归类操作，因为规模和业务的不同，分类的层级肯定也会出现截然不同的效果。天猫商城和京东商城在分类层级的设计思路上是相同的，都是将分类设置成三级。

如果不设置分类层级，过多的商品分类也会造成用户筛选的困难，而进行分类层级设置后，我们在查找时可以遵循 ****先大类后小类**** 的原则，比如我们需要买一部手机，我们可以现在左侧的一级分类中筛选并定位到“手机/数码”这一类目中，之后在这个类目的子分类下再去进行筛选，此时你想象一下如果没有分类层级的设计，我们就需要在众多的子分类中去逐一查找我们需要的分类条目，这种体验就很反人类。所以分类层级的设置使得用户的目的性更强，且用户在进行商品类别查找时有迹可循，也能够更加快速的定位到需要的分类中。

当然，也有人会提出设计出更深层级的分类，比如四级分类、五级分类等等，这种层级太深也不是很好的设计，太深的话一是对用户不太友好，层级太深反而会不利于寻找；二是对于后台管理人员也不友好，层级太深不方便管理，所以我们就直接参考天猫商城和京东商城的类目层级设计原则，将其设置成三级类目。

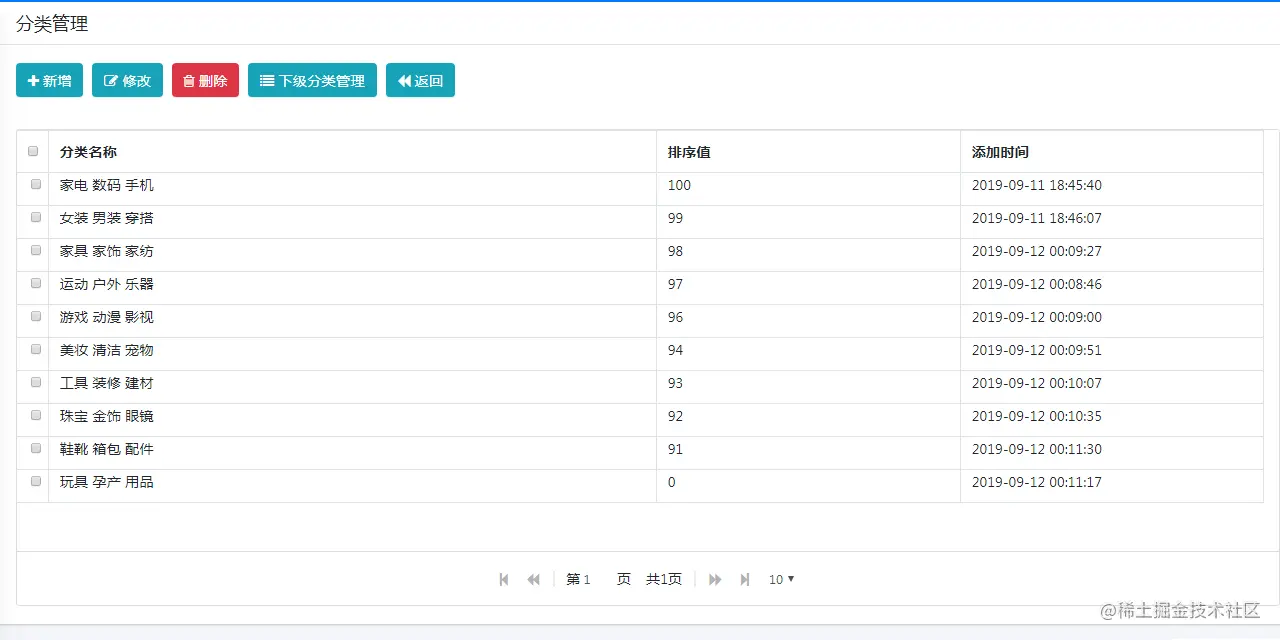


#### 分类模块的主要功能

新蜂商城中分类模块的主要功能如下：

* 分类数据的设置
* 商品与分类的挂靠和关联
* 分类信息展示
* 根据分类搜索商品

分类设置中包括分类信息的添加修改等操作，这些数据的存在才使得用户可以在商城端进行筛选。商品与分类的挂靠和关联是指将商品信息与分类信息建立联系，比如在商品表中创建一个分类 id 字段，这样商品与分类二者之间就产生了关联关系，我们就能够通过对应的分类搜索到对应的商品列表，分类数据的设置、商品与分类的挂靠和关联这两个功能点是在后台管理系统中完成的，大家可以在后台管理系统中体验并进行数据的配置：



本文我们主要介绍的是新峰商城 Vue 版本中的分类页面：

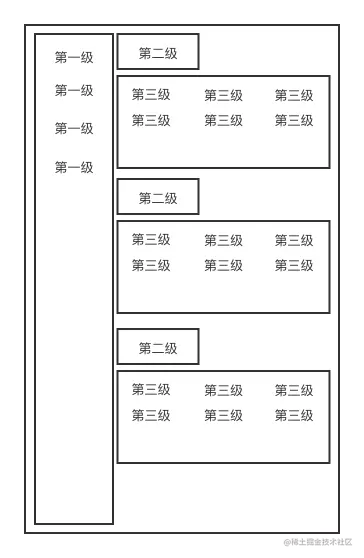


讲一讲该页面中展示数据的获取逻辑，后端的分类列表接口的设计和实现。

## 数据格式定义

页面布局和交互实现依然由前端代码来实现，后端只需要将页面所需的分类数据，通过接口响应回去即可，分类页面中的所有数据都是由后端返回的。

接下来我们定义一下分类接口中的返回数据，这里结合实际页面来讲解：



如上图所示，我们需要返回配置的多条一级分类展示在页面左侧并固定，这里我们就可以定义一个分类 VO 对象，返回的数据类型就是一个 voList 对象。理论上这样设计就可以，但是，通过对上图的观察，我们知道分类信息是有层级关系，一级、二级和三级所展示的并不相同。比如，我们要返回一级分类列表，但是页面上展示的不止是一级分类，还包括每个一级分类的二级分类列表，而二级分类列表中的每个二级分类也有一个三级分类列表。

因此新峰商城 API 项目中定义了三个 VO 对象，并做了层级关系，代码如下：

ThirdLevelCategoryVO：

*/\*\**

*\* 分类数据VO(第三级)*

*\*/***@Datapublic** **class** **ThirdLevelCategoryVO** **implements** **Serializable** {

**@ApiModelProperty("当前三级分类id")**

**private** Long categoryId;

**@ApiModelProperty("当前分类级别")**

**private** Byte categoryLevel;

**@ApiModelProperty("当前三级分类名称")**

**private** String categoryName;

}

SecondLevelCategoryVO：

*/\*\**

*\* 分类数据VO(第二级)*

*\*/***@Datapublic** **class** **SecondLevelCategoryVO** **implements** **Serializable** {

**@ApiModelProperty("当前二级分类id")**

**private** Long categoryId;

**@ApiModelProperty("父级分类id")**

**private** Long parentId;

**@ApiModelProperty("当前分类级别")**

**private** Byte categoryLevel;

**@ApiModelProperty("当前二级分类名称")**

**private** String categoryName;

**@ApiModelProperty("三级分类列表")**

**private** List<ThirdLevelCategoryVO> thirdLevelCategoryVOS;

}

NewBeeMallIndexCategoryVO：

*/\*\**

*\* 分类数据VO*

*\*/***@Datapublic** **class** **NewBeeMallIndexCategoryVO** **implements** **Serializable** {

**@ApiModelProperty("当前一级分类id")**

**private** Long categoryId;

**@ApiModelProperty("当前分类级别")**

**private** Byte categoryLevel;

**@ApiModelProperty("当前一级分类名称")**

**private** String categoryName;

**@ApiModelProperty("二级分类列表")**

**private** List<SecondLevelCategoryVO> secondLevelCategoryVOS;

}

因为都是分类信息，所以三个 VO 对象中都有分类名称和分类名称。虽然字段类似，不过我们商品分类是有层级关系的，在 VO 对象中还需要定义其中的层级关系，因此又定义了分类层级字段，并且在一级分类 VO 和二级分类 VO 对象中定义了下级分类列表字段。比如二级分类中，不仅仅要包含二级分类的信息，还要包含该二级分类下所有的三级分类的信息。

## 分类接口实现

介绍完返回数据的数据格式和字段定义后，我们来看一下商品分类表结构的设计和接口实现。

#### 商品分类表结构设计

虽然是三级分类，但是并没有设计成三张表，因为大部分字段都是一样的，所以就选择增加一个 category\_level 字段来区分是哪一级的分类，同时使用 parent\_id 字段来进行上下级类目之间的关联，分类表的字段设计如下：

**DROP** **TABLE** IF **EXISTS** `tb\_newbee\_mall\_goods\_category`;**CREATE** **TABLE** `tb\_newbee\_mall\_goods\_category` (

`category\_id` **bigint**(20) **NOT** **NULL** AUTO\_INCREMENT COMMENT '分类id',

`category\_level` tinyint(4) **NOT** **NULL** **DEFAULT** 0 COMMENT '分类级别(1-一级分类 2-二级分类 3-三级分类)',

`parent\_id` **bigint**(20) **NOT** **NULL** **DEFAULT** 0 COMMENT '父分类id',

`category\_name` **varchar**(50) **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci **NOT** **NULL** **DEFAULT** '' COMMENT '分类名称',

`category\_rank` **int**(11) **NOT** **NULL** **DEFAULT** 0 COMMENT '排序值(字段越大越靠前)',

`is\_deleted` tinyint(4) **NOT** **NULL** **DEFAULT** 0 COMMENT '删除标识字段(0-未删除 1-已删除)',

`create\_time` datetime(0) **NOT** **NULL** **DEFAULT** CURRENT\_TIMESTAMP COMMENT '创建时间',

`create\_user` **int**(11) **NOT** **NULL** **DEFAULT** 0 COMMENT '创建者id',

`update\_time` datetime(0) **NOT** **NULL** **DEFAULT** CURRENT\_TIMESTAMP COMMENT '修改时间',

`update\_user` **int**(11) **NULL** **DEFAULT** 0 COMMENT '修改者id',

**PRIMARY** KEY (`category\_id`) **USING** BTREE

) ENGINE = InnoDB AUTO\_INCREMENT = 107 **CHARACTER** **SET** = utf8 **COLLATE** = utf8\_general\_ci ROW\_FORMAT = **Dynamic**;

商品类目表的字段以及每个字段对应的含义都在上面的 SQL 中有介绍，大家可以对照 SQL 进行理解，接下来我们进行编码工作。

#### 商品分类数据读取

主要的实现逻辑在 NewBeeMallCategoryService 业务类的 getCategoriesForIndex() 方法中，代码如下：

**public** List<NewBeeMallIndexCategoryVO> **getCategoriesForIndex**() {

List<NewBeeMallIndexCategoryVO> newBeeMallIndexCategoryVOS = **new** **ArrayList**<>();

*//获取一级分类的固定数量的数据*

List<GoodsCategory> firstLevelCategories = goodsCategoryMapper.selectByLevelAndParentIdsAndNumber(Collections.singletonList(0L), NewBeeMallCategoryLevelEnum.LEVEL\_ONE.getLevel(), Constants.INDEX\_CATEGORY\_NUMBER);

**if** (!CollectionUtils.isEmpty(firstLevelCategories)) {

List<Long> firstLevelCategoryIds = firstLevelCategories.stream().map(GoodsCategory::getCategoryId).collect(Collectors.toList());

*//获取二级分类的数据*

List<GoodsCategory> secondLevelCategories = goodsCategoryMapper.selectByLevelAndParentIdsAndNumber(firstLevelCategoryIds, NewBeeMallCategoryLevelEnum.LEVEL\_TWO.getLevel(), 0);

**if** (!CollectionUtils.isEmpty(secondLevelCategories)) {

List<Long> secondLevelCategoryIds = secondLevelCategories.stream().map(GoodsCategory::getCategoryId).collect(Collectors.toList());

*//获取三级分类的数据*

List<GoodsCategory> thirdLevelCategories = goodsCategoryMapper.selectByLevelAndParentIdsAndNumber(secondLevelCategoryIds, NewBeeMallCategoryLevelEnum.LEVEL\_THREE.getLevel(), 0);

**if** (!CollectionUtils.isEmpty(thirdLevelCategories)) {

*//根据 parentId 将 thirdLevelCategories 分组*

Map<Long, List<GoodsCategory>> thirdLevelCategoryMap = thirdLevelCategories.stream().collect(groupingBy(GoodsCategory::getParentId));

List<SecondLevelCategoryVO> secondLevelCategoryVOS = **new** **ArrayList**<>();

*//处理二级分类*

**for** (GoodsCategory secondLevelCategory : secondLevelCategories) {

**SecondLevelCategoryVO** secondLevelCategoryVO = **new** **SecondLevelCategoryVO**();

BeanUtil.copyProperties(secondLevelCategory, secondLevelCategoryVO);

*//如果该二级分类下有数据则放入 secondLevelCategoryVOS 对象中*

**if** (thirdLevelCategoryMap.containsKey(secondLevelCategory.getCategoryId())) {

*//根据二级分类的id取出thirdLevelCategoryMap分组中的三级分类list*

List<GoodsCategory> tempGoodsCategories = thirdLevelCategoryMap.get(secondLevelCategory.getCategoryId());

secondLevelCategoryVO.setThirdLevelCategoryVOS((BeanUtil.copyList(tempGoodsCategories, ThirdLevelCategoryVO.class)));

secondLevelCategoryVOS.add(secondLevelCategoryVO);

}

}

*//处理一级分类*

**if** (!CollectionUtils.isEmpty(secondLevelCategoryVOS)) {

*//根据 parentId 将 thirdLevelCategories 分组*

Map<Long, List<SecondLevelCategoryVO>> secondLevelCategoryVOMap = secondLevelCategoryVOS.stream().collect(groupingBy(SecondLevelCategoryVO::getParentId));

**for** (GoodsCategory firstCategory : firstLevelCategories) {

**NewBeeMallIndexCategoryVO** newBeeMallIndexCategoryVO = **new** **NewBeeMallIndexCategoryVO**();

BeanUtil.copyProperties(firstCategory, newBeeMallIndexCategoryVO);

*//如果该一级分类下有数据则放入 newBeeMallIndexCategoryVOS 对象中*

**if** (secondLevelCategoryVOMap.containsKey(firstCategory.getCategoryId())) {

*//根据一级分类的id取出secondLevelCategoryVOMap分组中的二级级分类list*

List<SecondLevelCategoryVO> tempGoodsCategories = secondLevelCategoryVOMap.get(firstCategory.getCategoryId());

newBeeMallIndexCategoryVO.setSecondLevelCategoryVOS(tempGoodsCategories);

newBeeMallIndexCategoryVOS.add(newBeeMallIndexCategoryVO);

}

}

}

}

}

**return** newBeeMallIndexCategoryVOS;

} **else** {

**return** null;

}

}

该方法的作用是返回后台管理系统中配置的分类数据并响应给前端，实现思路如下：

首先读取固定数量的一级分类数据，之后获取二级分类数据并设置到对应的一级分类下，最后是获取和设置每一个二级分类下的三级分类数据，这里的逻辑是一次性读取所有数据并返回，与[《新蜂商城商品类目三级联动功能实现》](https://juejin.cn/book/6844733814074245133/section/6844733814149742599" \o "https://juejin.cn/book/6844733814074245133/section/6844733814149742599" \t "/home/erfenjiao/文档\\x/_blank)中的多次获取数据的实现逻辑不同，多次获取的实现方式主要是后台管理系统中需要三级分类联动的效果，而新蜂商城 Vue 版本分类页面是将所有三级分类数据读取并渲染到页面上，所以这里就将所有的数据都读取出来并封装成一个对象返回给视图层。

实际调用 DAO 层的查询方法为：

List<GoodsCategory> **selectByLevelAndParentIdsAndNumber**(**@Param("parentIds")** List<Long> parentIds, **@Param("categoryLevel")** **int** categoryLevel, **@Param("number")** **int** number);

selectByLevelAndParentIdsAndNumber() 方法中，定义了三个字段，分别是 parentIds、categoryLevel、number，即分类的父类 id 集合、分类级别和需要返回的查询数量，比如查询一级分类列表的代码为：

*//获取一级分类的固定数量的数据*

List<GoodsCategory> firstLevelCategories = goodsCategoryMapper.selectByLevelAndParentIdsAndNumber(Collections.singletonList(0L), NewBeeMallCategoryLevelEnum.LEVEL\_ONE.getLevel(), Constants.INDEX\_CATEGORY\_NUMBER);

因为一级分类是没有父类的，即父级分类的 id 为缺省值 0，同时 parentIds 参数为 List 类型，所以这里 parentIds 参数我们传的就是 Collections.singletonList(0L)，分类级别传的就是 1，这里用的是枚举类 NewBeeMallCategoryLevelEnum.LEVEL\_ONE，数量是 10，这里用的也是一个常量 Constants.INDEX\_CATEGORY\_NUMBER，该值默认为 10。当然，这里直接传数字 10 也是可以的。

之后是二级分类列表的查询，代码为 :

*//获取二级分类的数据*

List<GoodsCategory> secondLevelCategories = goodsCategoryMapper.selectByLevelAndParentIdsAndNumber(firstLevelCategoryIds, NewBeeMallCategoryLevelEnum.LEVEL\_TWO.getLevel(), 0);

因为上一步已经获取到所有的一级分类列表数据，所以我们把其中的 id 字段全部获取出来并放入到一个 List 对象中，这里 parentIds 参数我们传的就是 firstLevelCategoryIds，分类级别传的是 2，用的是枚举类 NewBeeMallCategoryLevelEnum.LEVEL\_ONE，数量 number 参数我们传的是 0，表示查询所有，并不是代表查询 0 条数据，三级分类查询方式与二级分类的查询逻辑类似，这里就不做赘述。

查询 SQL 语句为：

<select id="selectByLevelAndParentIdsAndNumber" resultMap="BaseResultMap">

select

<include refid="Base\_Column\_List"/>

from tb\_newbee\_mall\_goods\_category

where parent\_id in

<foreach item="parentId" collection="parentIds" open="(" separator="," close=")">

#{parentId,jdbcType=BIGINT}

</foreach>

and category\_level = #{categoryLevel,jdbcType=TINYINT}

and is\_deleted = 0

order by category\_rank desc

<if test="number>0">

limit #{number}

</if>

</select>

该方法名称为 selectByLevelAndParentIdsAndNumber()，类型为 SELECT 语句， 根据传入的参数查询固定数量的记录，如果传参 number 大于 0 我们就对数量进行过滤，否则会查询并返回所有条数的数据。

#### 接口实现

最终，我们的分类列表接口实现代码如下：

**@RestController@Api(value = "v1", tags = "新蜂商城分类页面接口")@RequestMapping("/api/v1")public** **class** **NewBeeMallGoodsCategoryAPI** {

**@Resource**

**private** NewBeeMallCategoryService newBeeMallCategoryService;

**@GetMapping("/categories")**

**@ApiOperation(value = "获取分类数据", notes = "分类页面使用")**

**public** Result<List<NewBeeMallIndexCategoryVO>> **getCategories**() {

List<NewBeeMallIndexCategoryVO> categories = newBeeMallCategoryService.getCategoriesForIndex();

**if** (CollectionUtils.isEmpty(categories)) {

NewBeeMallException.fail(ServiceResultEnum.DATA\_NOT\_EXIST.getResult());

}

**return** ResultGenerator.genSuccessResult(categories);

}

}

方法名称为 getCategories()，请求类型为 GET，映射的路径为 /api/v1/categories，响应结果类型为 Result，实际的 data 属性类型为 List<NewBeeMallIndexCategoryVO> 对象，即分类列表数据。实现逻辑就是分别调用分类业务实现类 NewBeeMallCategoryService 中的查询方法，将所需的数据查询并响应给前端，所有的实现逻辑都是在业务实现类中处理的，包括查询和字段设置，在控制层代码中只是将获得的数据结果设置到 Result 对象并返回。

还有一点需要注意，分类页面的数据也可以直接查看，并不需要登录认证，因此该接口中并没有使用权限认证的注解 @TokenToMallUser。

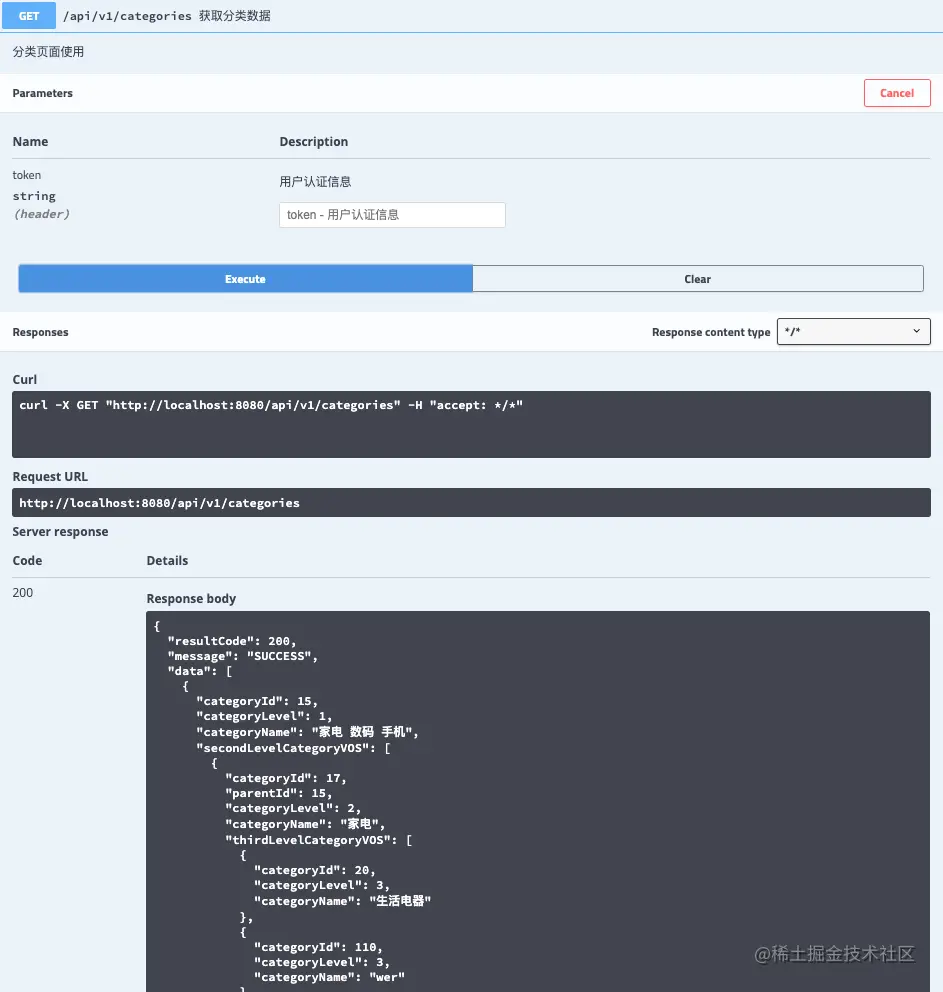
分类数据的获取是做成了单独的一个接口，当然也可以做三个接口，第一个接口拿所有的一级分类数据，第二个接口根据选择的一级分类 id 查询所有的二级分类数据，第三个接口类似。但是这样的话，不如一次性全部查出来，让前端可以一次性处理和渲染这些数据，而不是分多次去查询、多次去渲染页面。

这样做成单独一个接口，前端方便处理，既不用因为用户切换了不同的分类而重新改变页面 DOM，也不用根据页面选项卡的切换多次发送请求，一定程度上可以节省网络开销。

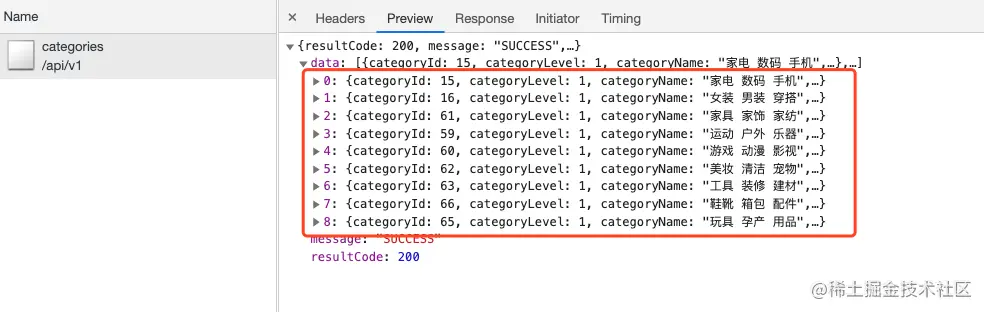
#### 分类列表接口测试

最后，我们通过 Swagger 页面来测试一下分类列表接口。

重启项目，打开 swagger-ui 页面，点开“获取分类数据”，由于不需要身份验证，所以 token 输入框中不填入 token 值也可以获取到数据，之后点击”Execute“按钮，即可获取到分类页面展示所需要的数据，如下图所示：

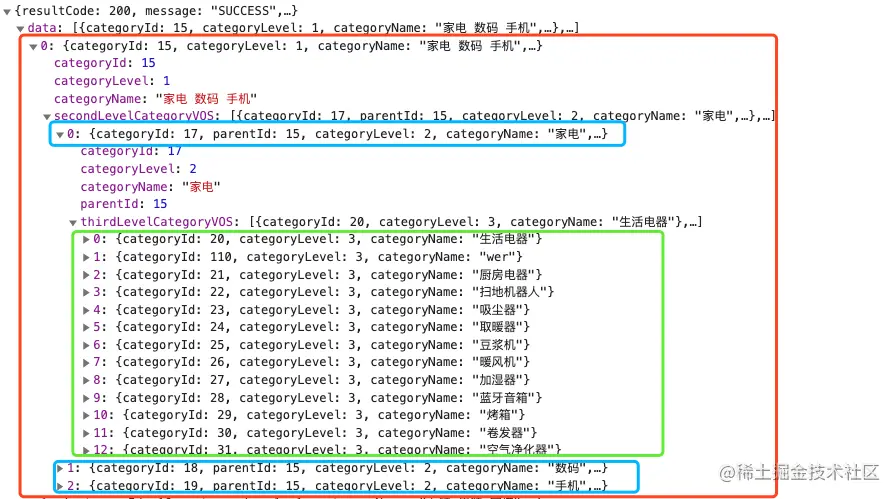


由于数据过多，截图无法截全，这里我们 F12 打开控制台，通过控制台中 Network 面板中的内容来看一下返回的数据结构，如下图所示，这里我们点开刚刚请求的 /api/v1/categories 链接，左侧点击 Preview 选项卡就可以看到返回的数据：



前端接收到的是一个 Result 类型的 json 格式数据，包括 resultCode、message、data 三个键值对，分类数据在 data 键中，截图中红色框中的数据就是 data 键中的所有一级分类数据。

当然，以上并不是所有的数据，我们一次点开每一条一级分类，可以看到一级分类下还有二级分类列表，每一个二级分类列表下还有三级分类数据列表，如下图所示：



前端获取到这些数据后，就可以一次性渲染到页面中，分类接口实现成功！

如果想要更改这部分配置内容，可以在新蜂商城后台管理系统中去操作。