首先 我做的是一个***的一个电商购物项目,是那种没有第三方商家入驻
专门卖**的一个网站.
用户可以在线购买商品 加入购物车 下单 商品秒杀
可以评论已购买商品
管理员可以在后台管理商品的上下架、促销活动
管理员可以监控商品销售状况
客服可以在后台处理退款操作

后台系统主要包含以下功能:

商品管理，包括商品分类、品牌、商品规格等信息的管理销售管理，包括订单统计、订单退款处理、

促销活动生成等

用户管理，包括用户控制、冻结、解锁等

权限管理，整个网站的权限控制，采用 JWT 鉴权方案，对用户及 API 进行权限控制

统计，各种数据的统计分析展示

后台系统会采用前后端分离开发，而且整个后台管理系统会使用 Vue.js 框架搭建出单页应用

（SPA）。

前台门户面向的是客户，包含与客户交互的一切功能。例如:

搜索商品 加入购物车 下单 评价商品等等

无论是前台还是后台系统，都共享相同的微服务集群，包括:

商品微服务：商品及商品分类、品牌、库存等的服务

搜索微服务：实现搜索功能

订单微服务：实现订单相关

购物车微服务：实现购物车相关功能

用户中心：用户的登录注册等功能

Eureka 注册中心

Zuul 网关服务

Spring Cloud Config 配置中心

...

中台

nginx

前台发送请求,请求的是 nginx,nginx 做反向代理(端口转发

[manage.mrshop.com:80-->127.0.0.1:9001 | api.mrshop.com:80 --> 127.0.0.1:8088{zuul}])

zuul

/api-xxx/** --> xxx-server 服务

hosts

127.0.0.1 --> manage.mrshop.com

127.0.0.1 --> api.mrshop.com

www.baidu.com --> 万维网 .com | .cn | .com.cn

Nigex

```
server {
    listen       80;
    server_name manage.mrshop.com;

    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_


    location / {
        proxy_pass http://127.0.0.1:9001;
        proxy_connect_timeout 600;
        proxy_read_timeout 600;
    }

}
```

1. 浏览器准备发起请求，访问 http://mamage.mrshop.com，但需要进行域名解析

2. 优先进行本地域名解析，因为我们修改了 hosts，所以解析成功，得到地址：127.0.0.1

3. 请求被发往解析得到的 ip，并且默认使用 80 端口：http://127.0.0.1:80

本机的 nginx 一直监听 80 端口，因此捕获这个请求

4. nginx 中配置了反向代理规则，将 manage.mrshop.com 代理到 127.0.0.1:9001，因此请求被转发

5. 后台系统的 webpack server 监听的端口是 9001，得到请求并处理，完成后将响应返回到 nginx

6. nginx 将得到的结果返回到浏览器

商品分类

商品分类查询

通过父类目 parentId 进行查询

客户端发送请求 请求到接口

```
@Api(tags = "商品分类的接口")
public interface CategoryService {

    @ApiOperation(value = "通过父类id查取所有商品")
    @GetMapping(value="category/list")
    Result<List<CategoryEntity>> getCategoryByPid(Integer pid);

}
```

然后接口的实现类 实现这个方法,把封装的 mapper

```
public interface CategoryMapper extends Mapper<CategoryEntity> {
}
```

给注入进来

```java
@RestController
public class CategoryServiceImpl extends BaseApiService implements CategoryService {

    @Resource
    private CategoryMapper categoryMapper;


    @Override
    public Result<List<CategoryEntity>> getCategoryByPid(Integer pid) {
        CategoryEntity categoryEntity = new CategoryEntity();
        categoryEntity.setParentId(pid);
        List<CategoryEntity> list = categoryMapper.select(categoryEntity);

        return this.setResultSuccess(list);
    }
}
```

下面重写这个方法之后,进行编码

商品分类删除:

内容 为 父节点 子节点 叶子 大概是个三级节点
叶子在被删除的还剩一个时 那么 就把父节点的 isprent 也就是状态改成可删除的状态

1. 首先 我会判断一下传来的值 是否是不合法数据,
2. 我会通过这个传来的值 进行 查询
3. 将查询出的数据 进行非空判断
4. 判断这条数据是否为父节点 是父节点 就 return
5. 判断当前分类是否被品牌绑定 如果绑定的话,那么不可被删除,通过 id 去中间表查询,如果 size>0 那么就不可被删除
6. 查询出该数据的父节点是否还有其他子节点
7. 如果没有的话,那么把他的状态改成可删除状态

```
        //判断传来的id是否有异常
        if(id != null && id < 0){ return this.setResultError("当前数据不正确"); }
        //通过id查询出子节点的数据
        CategoryEntity categoryEntity = categoryMapper.selectByPrimaryKey(id);
        //判断当前节点是否是父级节点(安全)
        if (categoryEntity.getIsParent()==1) {return this.setResultError("父类节点不可删除");}
        //如果当前分类被品牌绑定的话不能被删除 --> 通过分类id查询中间表是否有数据 true : 当前分类不能被删除 false:继续执行
      Example example1 = new Example(CategoryBrandEntity.class);
          example1.createCriteria().andEqualTo("categoryId",id);
      List<CategoryBrandEntity> categoryBrandEntityList = categoryBrandMapper.selectByExample(example1);
      if (categoryBrandEntityList.size()!=0){return this.setResultError("已绑定品牌 删除失败");};
    //如果当前分类被品牌绑定的话不能被删除 --> 通过分类id查询中间表是否有数据 true : 当前分类不能被删除 false:继续执行
    List<CategoryBrandEntity> categoryBrandList = categoryBrandMapper.selectByCategoryId(id);
        if(categoryBrandList.size()>0)return this.setResultError("被绑定 无法删除");

    //判断当前父类节点除了当前子节点是否还有其他子节点
        Example example = new Example(CategoryEntity.class);
        example.createCriteria().andEqualTo( property: "parentId",categoryEntity.getParentId());
        List<CategoryEntity> list = categoryMapper.selectByExample(example);

        if (list.size()==1){
            //判断通过父类id查询出的子类是否有其他子类 如果 只有一条 那么改掉父类节点的isprent = 0
            CategoryEntity categoryEntity1 = new CategoryEntity();
            categoryEntity1.setId(categoryEntity.getParentId());
            categoryEntity1.setIsParent(0);

         categoryMapper.updateByPrimaryKeySelective(categoryEntity1);
        }
    categoryMapper.deleteByPrimaryKey(id);
    return this.setResultSuccess();
}
```

商品分类添加:

1.首先要把传过来的值 进行非空判断
2.然后判断这个值是否是父节点
3.如果判断他不是父级节点 那么把他的状态改成父级状态
(添加的是一个默认的子节点 需要手动修改他的 name)

```
    if(categoryEntity == null){return this.setResultError("为空");};
    //判断 这个新增的子节点的父节点 是否是父类
    CategoryEntity parentCategoryEntity1 = new CategoryEntity();
    parentCategoryEntity1.setId(categoryEntity.getParentId());
    CategoryEntity select = categoryMapper.selectByPrimaryKey(parentCategoryEntity1);
    if(select.getIsParent() == 0){
        select.setIsParent(1);
        categoryMapper.updateByPrimaryKeySelective(select);
    }

    //如果不是 那么把他的状态改成1

    categoryMapper.insertSelective(categoryEntity);
    return this.setResultSuccess();
}
```

商品分类的修改:

修改的话没有业务

```
public Result<CategoryEntity> updateById(CategoryEntity categoryEntity) {
    try {
        categoryMapper.updateByPrimaryKeySelective(categoryEntity);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return this.setResultSuccess();
}
```

品牌

品牌查询:

1.首先把前台传来的 分页信息通过 PageHelper.startpage("page","rows")这个方法 进行分页
PageHelper.*startPage*(brandDTO.getPage(),brandDTO.getRows());
2. 用来接收前台数据的 brandDTO 里面的属性使用 BeanUtils.*copyProperties*(),*进行 copy(这里我们是把这个方法进行了一个封装*

```
public static <T> T copyBean(Object source,Class<T> clazz)  {
    T t = null;//创建实例 类似 为为传来的参数new 一个实例
    try {
        t = clazz.newInstance();
        BeanUtils.copyProperties(source,t);
        return t;
    } catch (InstantiationException e) {
```

)

3. 进行模糊查询,这里是用到的 tk 包下的方法

```
Example example = new Example(BrandEntity.class);
//进行模糊查询
example.createCriteria().andLike( property: "name", value: "%"+ brandEntity.getName()  +"%");
List<BrandEntity> brandEntityList = brandMapper.selectByExample(example);
```

4. 查出来之后放到 pageInfo 这个对象里面;

```
PageInfo<BrandEntity> pageInfo = new PageInfo<>(brandEntityList);
```

品牌新增：

1. 首先需要对前台传来的值进行非空判断

```
if(brandDTO ==null){return this.setResultError("---"); }
```

2. 把 DTO 里面的属性 copy 给我们需要操作数据库的实体类

```
//里面放着前台传来的值
BrandEntity brandEntity = zBeanUtils.copyBean(brandDTO, BrandEntity.class);
```

3. 处理 letter 新增时的首字母要大写 并且截取首字母

```
brandEntity.setLetter(PinyinUtil.getUpperCase(String.valueOf(brandEntity.getName().toCharArray()[0]), isFull: false).toCharArray()[0]);
```

4. 进行新增 但同时要获取新增 id 这里用到了一个注解(加入到实体类 Id 属性上)

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

```
//新增 获取到新增id
brandMapper.insert(brandEntity);
```

5. 因为前台可以选取多个商品分类,所以需要批量新增

```
//这是categories id
String categories = brandDTO.getCategories();
List<CategoryBrandEntity> categoryBrandEntities = new ArrayList<>();
if(categories.contains(",")){
    String[] split = categories.split( regex: ",");
    for (String s : split) {
        CategoryBrandEntity categoryBrandEntity = new CategoryBrandEntity();
        categoryBrandEntity.setCategoryId(Integer.valueOf(s));
        categoryBrandEntity.setBrandId(brandEntity.getId());
        categoryBrandEntities.add(categoryBrandEntity);
    }
    categoryBrandMapper.insertList(categoryBrandEntities);
}else{
    CategoryBrandEntity categoryBrandEntity = new CategoryBrandEntity();
    categoryBrandEntity.setCategoryId(Integer.valueOf(categories));
    categoryBrandEntity.setBrandId(brandEntity.getId());
    categoryBrandMapper.insert(categoryBrandEntity);
}
return this.setResultSuccess();
```

品牌上传图片：

图片上传注意跨域问题

```
**/
@Configuration
public class GlobalCorsConfig {

    @Bean
    public CorsFilter corsFilter() {
        final UrlBasedCorsConfigurationSource source = new
                UrlBasedCorsConfigurationSource();
        final CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true); // 允许cookies跨域
        config.addAllowedOrigin("*");// 允许向该服务器提交请求的URI，*表示全部允许。这里尽量限制来源域，比如http://xxxx:8080 ,以降低安全风险。.
        config.addAllowedHeader("*");// 允许访问的头信息,*表示全部
        config.setMaxAge(18000L);// 预检请求的缓存时间（秒），即在这个时间段里，对于相同的跨域请求不会再预检了
        config.addAllowedMethod("*");// 允许提交请求的方法，*表示全部允许，也可以单独置GET、PUT等
        config.addAllowedMethod("HEAD");
        config.addAllowedMethod("GET");// 允许Get的请求方法
        config.addAllowedMethod("PUT");
        config.addAllowedMethod("POST");
        config.addAllowedMethod("DELETE");
        config.addAllowedMethod("PATCH");
        source.registerCorsConfiguration( path: "/**", config);
//3.返回新的CorsFilter.
        return new CorsFilter(source);
    }
```

品牌修改:

1. 把 DTO 里的属性 copy 给要操作数据库的 Entity

```
BrandEntity brandEntity = zBeanUtils.copyBean(brandDTO, BrandEntity.class);
```

2. 处理 letter 修改时的首字母要大写 并且截取首字母

```
brandEntity.setLetter(PinyinUtil.getUpperCase(String.valueOf(brandEntity.getName().toCharArray()[0]), isFull: false).toCharArray()[0]);
```

3. 进行修改

```
brandMapper.updateByPrimaryKeySelective(brandEntity);
```

5. 我们需要通过 brandId 来进行删除中间表的数据

```
Example example = new Example(CategoryBrandEntity.class);
example.createCriteria().andEqualTo( property: "brandId",brandEntity.getId());
categoryBrandMapper.deleteByExample(example);
```

6. 进行批量修改

```
String categories = brandDTO.getCategories();
List<CategoryBrandEntity> categoryBrandEntities = new ArrayList<>();
if(categories.contains(",")){
    String[] split = categories.split( regex: ",");
    for (String s : split) {
        CategoryBrandEntity categoryBrandEntity = new CategoryBrandEntity();
        categoryBrandEntity.setCategoryId(Integer.valueOf(s));
        categoryBrandEntity.setBrandId(brandEntity.getId());
        categoryBrandEntities.add(categoryBrandEntity);
    }
    categoryBrandMapper.insertList(categoryBrandEntities);
}else{
    CategoryBrandEntity categoryBrandEntity = new CategoryBrandEntity();
    categoryBrandEntity.setCategoryId(Integer.valueOf(categories));
    categoryBrandEntity.setBrandId(brandEntity.getId());
    categoryBrandMapper.insert(categoryBrandEntity);
}
```

品牌删除:

1. 直接进行删除

```
brandMapper.deleteByPrimaryKey(id);
```

2. 删除中间表数据

```
Example example = new Example(CategoryBrandEntity.class);
example.createCriteria().andEqualTo( property: "brandId" , id);
categoryBrandMapper.deleteByExample(example);
```

规格组:

规格查询:

普通查询 还是需要把 dto 里面的属性 copy 给 entity 通过前台传来 cid(分类 id)进行查询数据

```
Example example = new Example(SpecificationEntity.class);
example.createCriteria().andEqualTo( property: "cid", zBeanUtils.copyBean(specificationDTO, SpecificationEntity.class).getCid());

List<SpecificationEntity> cId = specificationMapper.selectByExample(example);

return this.setResultSuccess(cId);
```

规格新增:

普通新增

```
specificationMapper.insertSelective(zBeanUtils.copyBean(specificationDTO,SpecificationEntity.class));
return this.setResultSuccess();
```

规格修改:

普通修改

```
specificationMapper.updateByPrimaryKeySelective(zBeanUtils.copyBean(specificationDTO,SpecificationEntity.class));

return this.setResultSuccess();
```

规格删除:

1. 先把传来的 id 进行非空判断
2. 通过 groudId 查出相应的数据
3. 如果查出来数据 size!=0,那么就不得删除 因为他以被绑定规格参数.如果没有数据那么可以删除
4.

```
if(id != null){
    //通过id 查出数据      条件查询出的规格
    Example example = new Example(SpecParamEntity.class);
    //通过groupID 去tb_spec_param表里查询数据
    example.createCriteria().andEqualTo( property: "groupId",id);
    List<SpecParamEntity> list = specParamMapper.selectByExample(example);
    //如果有的话 那么无法删除
    if (list.size()!=0){
        return this.setResultError("被绑定 无法删除");
    }
```

规格参数:

规格参数查询:

先把 dto 属性 copy 给 entity 再通过 groudID 进行查询

```
    SpecParamEntity specParamEntity = zBeanUtils.copyBean(specParamDTO, SpecParamEntity.class);
    Example example = new Example(SpecParamEntity.class);
    example.createCriteria().andEqualTo( property: "groupId",specParamEntity.getGroupId());
    List<SpecParamEntity> specParamEntities = specParamMapper.selectByExample(example);
    return this.setResultSuccess(specParamEntities);
}
```

规格参数新增:

普通新增

```
if(specParamDTO != null ){
    specParamMapper.insertSelective(zBeanUtils.copyBean(specParamDTO,SpecParamEntity.class));
}
return this.setResultSuccess();
```

规格参数修改:

普通修改

```
if(specParamDTO != null){
    specParamMapper.updateByPrimaryKey(zBeanUtils.copyBean(specParamDTO, SpecParamEntity.class));
}
return this.setResultSuccess();
```

规格参数删除:

普通删除

```
if(id != null )specParamMapper.deleteByPrimaryKey(id);
```

商品列表:

商品查询:

首先 前台传来分页 限制每次查询的条数

```
if(ObjectUtils.isNotNull(spuDTO.getPage())&&ObjectUtils.isNotNull(spuDTO.getRows())){
    PageHelper.startPage(spuDTO.getPage(),spuDTO.getRows());
}
if(ObjectUtils.isNotNull((spuDTO.getSort()))&&ObjectUtils.isNotNull(spuDTO.getOrder())){
    PageHelper.orderBy(spuDTO.getOrderBy());
}
```

前台传来 saleable 判断的是 每次查询 状态为 2 以下的数据
通过 title 模糊查询

```
if (ObjectUtils.isNotNull(spuDTO.getSaleable())&&spuDTO.getSaleable() < 2){
    criteria.andEqualTo( property: "saleable",spuDTO.getSaleable());
}
if(!StringUtils.isEmpty(spuDTO.getTitle())){
    criteria.andLike( property: "title", value: "%"+ spuDTO.getTitle()  +"%");
}
```

```java
@Override
public Result<List<SkuDTO>> skuBySpuIdStock(Integer spuId) {
    //使用spuid 查询出sku里面的数据

    List<SkuDTO> skuDTOList = skuMapper.skuBySpuIdStock(spuId);

    return this.setResultSuccess(skuDTOList);
}

@Override
public Result<JsonObject> spuByIdGetDetail(Integer spuId) {

    SpuDetailEntity spuDetailEntity = spuDetailMapper.selectByPrimaryKey(spuId);

    return this.setResultSuccess(spuDetailEntity);
}
```

```java
//查询出 所有的标题 id brandId cid
List<SpuEntity> entities = spuMapper.selectByExample(example);
//用map进行遍历
List<Object> collect = entities.stream().map(spuEntity -> {

    //把属性copy给dto
    SpuDTO spuDTO1 = zBeanUtils.copyBean(spuEntity, SpuDTO.class);
    //可以使用集合进行查询的selectByIdList函数,Arrays.asList可以放置多个参数
    List<CategoryEntity> categoryEntities = categoryMapper.
            selectByIdList(Arrays.asList(spuEntity.getCid1(), spuEntity.getCid2(), spuEntity.getCid3()));
    // set进CategoryName
    spuDTO1
            .setCategoryName(categoryEntities
                    .stream()
                    .map(categoryEntity -> categoryEntity.getName())
                    .collect(Collectors.joining( delimiter: "/")));
    //通过brandId 进行查询
    BrandEntity brandEntity = brandMapper.selectByPrimaryKey(spuDTO1.getBrand_id());
    spuDTO1.setBrandName(brandEntity.getName());


    return spuDTO1;
}).collect(Collectors.toList());
```

进行普通查询,通过前台传来的分页条件,每次只能查出5条数据

这里把查出的数据进行map遍历

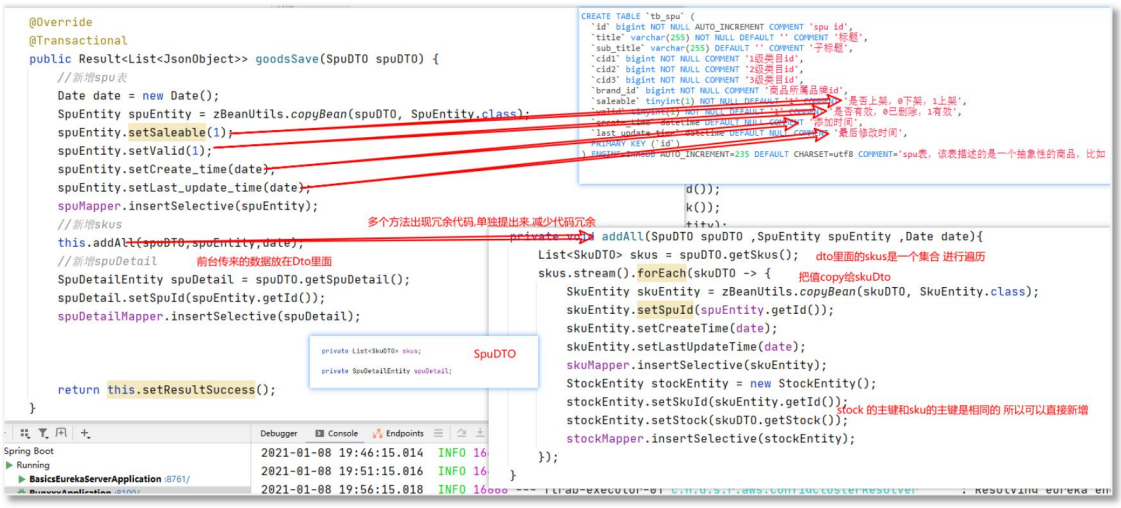
把spuEntity里面的值copy给SpuDTO

这里是用了selectByIdList(这个mapper方法可以放置集合作为条件)arrays.asList(可以存放多个参数)

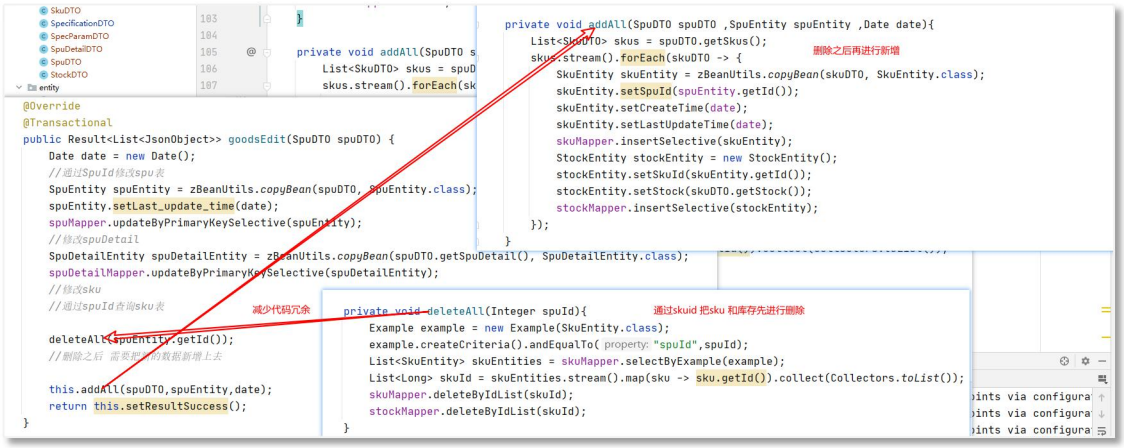
将查出来的分类内容通过/进行分割

通过brandId进行查询

商品新增:



商品修改:



商品删除:

上架下架:

前台:

```
methods: {
  editSaleable(item){
    let tishi = item.saleable==1?"下架":"上架"

    this.$message.confirm("是否"+tishi+ "?").then(()=>{
    this.$http.put("goods/xiajia",{
        id:item.id,
        saleable:item.saleable
    }).then(resp=>{
      this.closeForm();
      console.log(resp)
    })
    })
```

后台:

```java
@Override
public Result<JsonObject> goodsXiajia(SpuDTO spuDTO) {
    SpuEntity spuEntity = zBeanUtils.copyBean(spuDTO, SpuEntity.class);
    spuEntity.setSaleable(spuEntity.getSaleable()==1?0:1);
    spuMapper.updateByPrimaryKeySelective(spuEntity);


    return this.setResultSuccess();
}
```