第4章 商品管理后端逻辑

学习目标

- 掌握分布式ID生成器（snowflake）的使用
- 完成新增和修改商品的后端逻辑
- 完成商品审核与上下架的后端逻辑
- 完成商品删除与还原的后端逻辑

1. 分布式ID生成解决方案

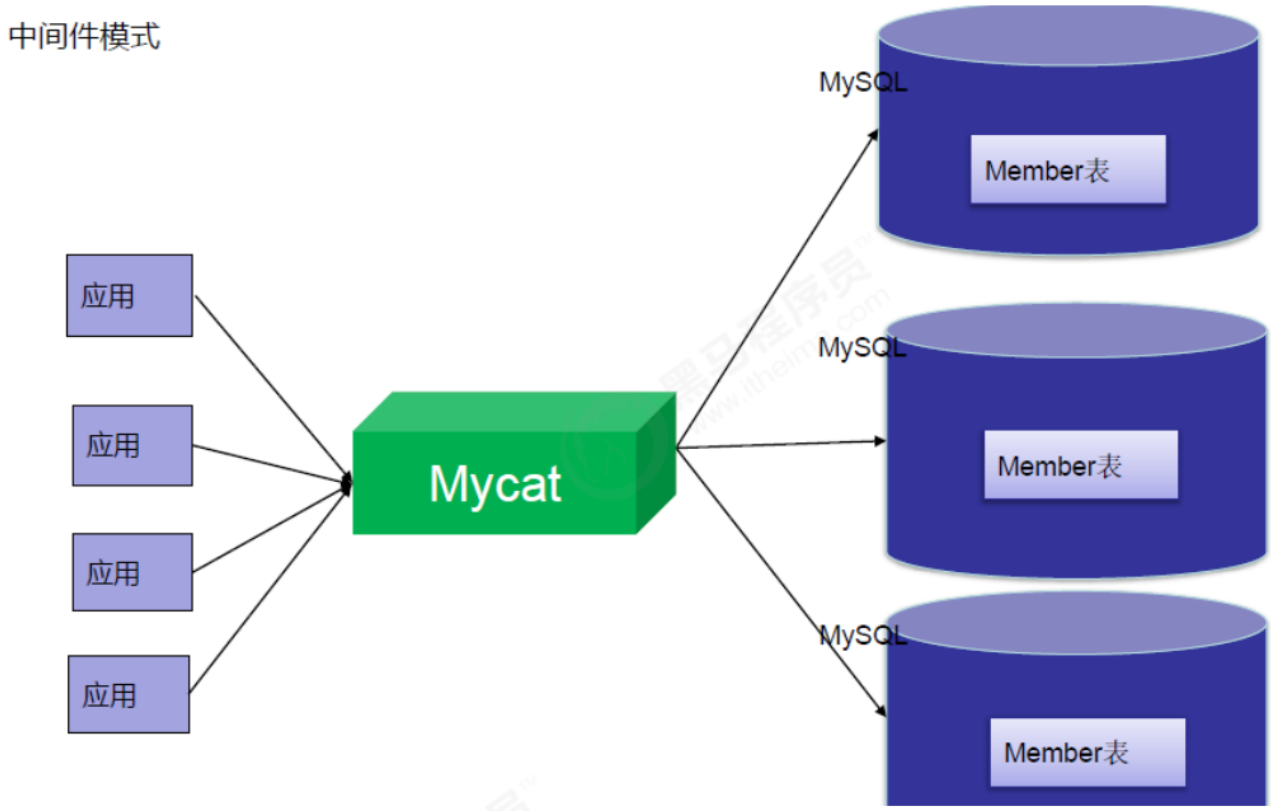
1.1 数据库分片（了解）

如今随着互联网的发展，数据的量级也是呈指数的增长，从GB到TB到PB。对数据的各种操作也是愈加的困难，如何解决这个问题呢？此时就需要做数据库集群，为了提高查询性能将一个数据库的数据分散到不同的数据库中存储，这就是我们通常所说的数据库分片。

如何实现数据库分片？我们通常会使用mycat数据库中间件来解决。

MyCat是一个开源的分布式数据库系统，是一个实现了MySQL协议的服务器，前端用户可以把它看作是一个数据库代理，用MySQL客户端工具和命令行访问，而其后端可以用MySQL原生协议与多个MySQL服务器通信，也可以用JDBC协议与大多数主流数据库服务器通信，其核心功能是分表分库，即将一个大表水平分割为N个小表，存储在后端MySQL服务器里或者其他数据库里。

MyCat发展到目前的版本，已经不是一个单纯的MySQL代理了，它的后端可以支持MySQL、SQL Server、Oracle、DB2、PostgreSQL等主流数据库，也支持MongoDB这种新型NoSQL方式的存储，未来还会支持更多类型的存储。而在最终用户看来，无论是那种存储方式，在MyCat里，都是一个传统的数据库表，支持标准的SQL语句进行数据的操作，这样一来，对前端业务系统来说，可以大幅降低开发难度，提升开发速度。



关于mycat具体的使用，我们是以补充资料的形式发给同学们，不作为课程内容讲解。

1.2 分布式ID生成解决方案

1.2.1 UUID

常见的方式。可以利用数据库也可以利用程序生成，一般来说全球唯一。

优点：

1）简单，代码方便。

2）生成ID性能非常好，基本不会有性能问题。

3）全球唯一，在遇见数据迁移，系统数据合并，或者数据库变更等情况下，可以从容应对。

缺点：

1）没有排序，无法保证趋势递增。

2）UUID往往是使用字符串存储，查询的效率比较低。

3）存储空间比较大，如果是海量数据库，就需要考虑存储量的问题。

4）传输数据量大

5）不可读。

1.2.2 Redis生成ID

当使用数据库来生成ID性能不够要求的时候，我们可以尝试使用Redis来生成ID。这主要依赖于Redis是单线程的，所以也可以用生成全局唯一的ID。可以用Redis的原子操作INCR和INCRBY来实现。

优点：

1）不依赖于数据库，灵活方便，且性能优于数据库。

2）数字ID天然排序，对分页或者需要排序的结果很有帮助。

缺点：

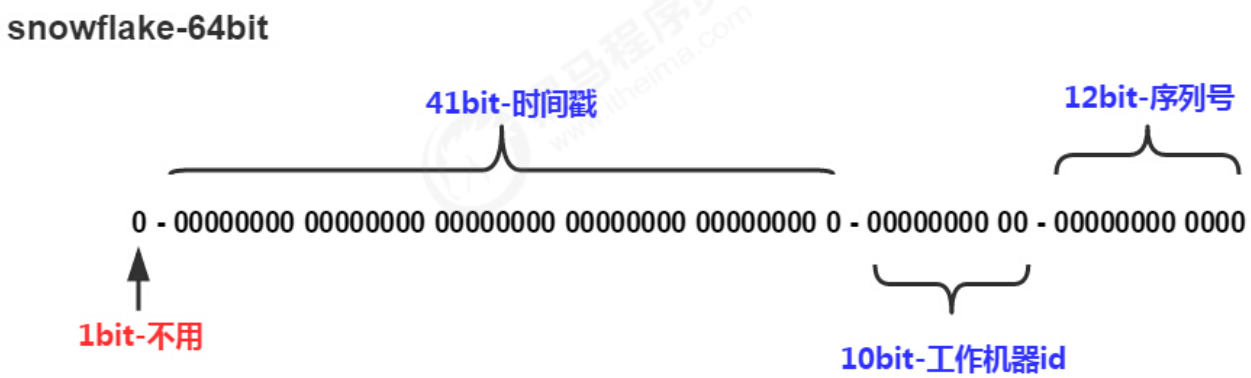
1）如果系统中没有Redis，还需要引入新的组件，增加系统复杂度。

2）需要编码和配置的工作量比较大。

3）网络传输造成性能下降。

1.2.3 开源算法snowflake

snowflake是Twitter开源的分布式ID生成算法，结果是一个long型的ID。其核心思想是：使用41bit作为毫秒数，10bit作为机器的ID（5个bit是数据中心，5个bit的机器ID），12bit作为毫秒内的流水号（意味着每个节点在每毫秒可以产生 4096 个 ID），最后还有一个符号位，永远是0



我们在《青橙》系统中采用的就是开源算法snowflake

1.3 snowflake快速入门

1.3.1 快速入门

（1）新建工程，将资料/工具类下的IdWorker.java拷贝到工程中。

（2）编写代码

```
        IdWorker idWorker=new IdWorker(1,1);

        for(int i=0;i<10000;i++){
            long id = idWorker.nextId();
            System.out.println(id);
        }
```

1.3.2 配置分布式ID生成器

（1）将IdWorker.java拷贝到qingcheng_common_service工程com.qingcheng.util包中

（2）在qingcheng_service_goods工程resources下新增applicationContext-service.xml

```
    <!--雪花ID生成器-->
    <bean id="idWorker" class="com.qingcheng.util.IdWorker">
        <constructor-arg index="0" value="1"></constructor-arg>
        <constructor-arg index="1" value="1"></constructor-arg>
    </bean>
```

2. 新增和修改商品

2.1 概念与表结构分析

2.1.1 SPU与SKU

SPU = Standard Product Unit （标准产品单位）
SPU是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。
通俗点讲，属性值、特性相同的商品就可以称为一个SPU。

例如：
华为P20 pro就是一个SPU，与商家，与颜色、款式、套餐都无关。

SKU=stock keeping unit(库存量单位**)**
SKU即库存进出计量的单位， 可以是以件、盒、托盘等为单位。
SKU是物理上不可分割的最小存货单元。在使用时要根据不同业态，不同管理模式来处理。在服装、鞋类商品中使用最多最普遍。

例如：
华为P20 pro 宝石蓝 64G

华为P20 pro 亮黑色 64G

华为P20 pro 宝石蓝 128G

华为P20 pro 亮黑色 128G

2.1.2 表结构分析

tb_spu 表

字段名称	字段含义	字段类型	字段长度	备注
id	主键	VARCHAR		
sn	货号	VARCHAR		
name	SPU名	VARCHAR		
caption	副标题	VARCHAR		
brand_id	品牌ID	INT		
category1_id	一级分类	INT		
category2_id	二级分类	INT		
category3_id	三级分类	INT		
template_id	模板ID	INT		
freight_id	运费模板id	INT		
image	图片	VARCHAR		
images	图片列表	VARCHAR		
sale_service	售后服务	VARCHAR		
introduction	介绍	TEXT		
spec_items	规格列表	VARCHAR		
para_items	参数列表	VARCHAR		
sale_num	销量	INT		
comment_num	评论数	INT		
is_marketable	是否上架	CHAR		
is_enable_spec	是否启用规格	CHAR		
is_delete	是否删除	CHAR		
status	审核状态	CHAR		

tb_sku 表

字段名称	字段含义	字段类型	字段长度	备注
id	商品id	VARCHAR		
sn	商品条码	VARCHAR		
name	SKU名称	VARCHAR		
price	价格（分）	INT		
num	库存数量	INT		
alert_num	库存预警数量	INT		
image	商品图片	VARCHAR		
images	商品图片列表	VARCHAR		
weight	重量（克）	INT		
create_time	创建时间	DATETIME		
update_time	更新时间	DATETIME		
spu_id	SPUID	VARCHAR		
category_id	类目ID	INT		
category_name	类目名称	VARCHAR		
brand_name	品牌名称	VARCHAR		
spec	规格	VARCHAR		
sale_num	销量	INT		
comment_num	评论数	INT		
status	商品状态 1-正常，2-下架，3-删除	CHAR		

2.2 需求与实现思路

2.2.1 需求分析

需求详见静态原型

2.2.2 实现思路

前端传递给后端的数据格式

```
{
    "spu": {
        "name": "这个是商品名称",
        "caption": "这个是副标题",
        "brandId": 12,
        "category1Id": 558,
        "category2Id": 559,
        "category3Id": 560,
        "freightId": 10,
        "image": "http://www.qingcheng.com/image/1.jpg",
         "images":
"http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jp
g",
        "introduction": "这个是商品详情，html代码",
         "paraItems": {"出厂年份":"2019","赠品":"充电器"},
        "saleService": "七天包退,闪电退货",
        "sn": "020102331",
        "specItems":  {"颜色":["红","绿"],"机身内存":["64G","8G"]},
        "templateId": 42
    },
    "skuList": [{
        "sn": "10192010292",
         "num": 100,
         "alertNum": 20,
        "price": 900000,
         "spec": {"颜色":"红","机身内存":"64G"},
         "image": "http://www.qingcheng.com/image/1.jpg",
         "images":
"http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jp
g",
        "status": "1",
        "weight": 130
    },
    {
        "sn": "10192010293",
         "num": 100,
         "alertNum": 20,
         "price": 600000,
         "spec": {"颜色":"绿","机身内存":"8G"},

         "image": "http://www.qingcheng.com/image/1.jpg",
```

```
          "images":
"http://www.qingcheng.com/image/1.jpg,http://www.qingcheng.com/image/2.jp
g",
        "status": "1",
        "weight": 130
    }
  ]
}
```

2.3 代码实现

2.3.1 SPU与SKU列表的保存

代码实现：

（1）qingcheng_pojo工程创建组合实体类

```java
/**
 * 商品组合实体类
 */
public class Goods implements Serializable {

    private Spu spu;
    private List<Sku> skuList;

    public Spu getSpu() {
        return spu;
    }

    public void setSpu(Spu spu) {
        this.spu = spu;
    }

    public List<Sku> getSkuList() {
        return skuList;
    }

    public void setSkuList(List<Sku> skuList) {
        this.skuList = skuList;
    }
}
```

（2）qingcheng_interface工程SpuService新增方法定义

```java
    /**
     *  保存商品
     * @param goods 商品组合实体类
     */
    public void saveGoods(Goods goods);
```

（3）qingcheng_service_goods工程SpuServiceImpl实现此方法

```
    @Autowired
    private SkuMapper skuMapper;

    @Autowired
    private IdWorker idWorker;

    @Autowired
    private CategoryMapper categoryMapper;

    /**
     * 保存商品
     * @param goods
     */
    @Transactional
    public void saveGoods(Goods goods) {

        //保存一个spu的信息
        Spu spu = goods.getSpu();
        spu.setId(idWorker.nextId()+"");
        spuMapper.insert(spu);
        //保存sku列表的信息

        Date date=new Date();
        //分类对象
        Category category =
categoryMapper.selectByPrimaryKey(spu.getCategory3Id());

        List<Sku> skuList = goods.getSkuList();
        for (Sku sku:skuList){

            sku.setId(idWorker.nextId()+"");
            sku.setSpuId(spu.getId());

            //sku名称  =spu名称+规格值列表
            String name=spu.getName();
            //sku.getSpec()  {"颜色":"红","机身内存":"64G"}
            Map<String,String> specMap = JSON.parseObject(sku.getSpec(),
Map.class);
            for(String value:specMap.values()){

                name+=" "+value;
```

```
            }
            sku.setName(name);//名称
            sku.setCreateTime(date);//创建日期
            sku.setUpdateTime(date);//修改日期
            sku.setCategoryId(spu.getCategory3Id());//分类id
            sku.setCategoryName(category.getName());//分类名称
            sku.setCommentNum(0);//评论数
            sku.setSaleNum(0);//销售数量

            skuMapper.insert(sku);
        }
    }
```

注意！！！这个方法要对两个表进行操作，需要添加事务。

```
@Service(interfaceClass=SpuService.class)
```

我们在类上添加@Transactional注解，并且在@Service注解中指定接口为SpuService.class

（3）qingcheng_web_manager工程SpuController修改add方法

```
    @PostMapping("/save")
    public Result save(@RequestBody Goods goods){
        spuService.saveGoods(goods);
        return new Result();
    }
```

2.3.2 建立分类与品牌的关联

需求：

（1）为什么要建立分类与品牌的关联？因为我们在前台搜索时需要通过分类找到品牌列表。

（2）分类与品牌是什么关系？多对多。

（3）在什么地方添加关系？我们不在后台单独实现分类与品牌的关联，而是在添加商品时自动添加关联。

实现思路：

（1）设计中间表tb_category_brand表

（2）创建实体类、数据访问接口

（3）在添加商品的saveGoods方法中添加代码逻辑 ，将SPU的品牌编号和分类编号 一起插入到（中间表）中。

代码实现：

（1）创建实体类

```
@Table(name="tb_category_brand")
public class CategoryBrand implements Serializable {

    @Id
    private Integer categoryId;

    @Id
    private Integer brandId;


    public Integer getCategoryId() {
        return categoryId;
    }

    public void setCategoryId(Integer categoryId) {
        this.categoryId = categoryId;
    }

    public Integer getBrandId() {
        return brandId;
    }

    public void setBrandId(Integer brandId) {
        this.brandId = brandId;
    }
}
```

这个表是联合主键，所以templateId和brandId都有@Id注解

（2）新建数据访问接口

```
public interface CategoryBrandMapper extends Mapper<CategoryBrand> {

}
```

（3）修改saveGoods方法，添加以下代码

```
        CategoryBrand categoryBrand =new CategoryBrand();
        categoryBrand.setBrandId(spu.getBrandId());
        categoryBrand.setCategoryId(spu.getCategory3Id());
        int count=categoryBrandMapper.selectCount(categoryBrand);
        if(count==0) {
            categoryBrandMapper.insert(categoryBrand);
        }
```

2.3.3 根据ID查询商品

需求：根据id 查询SPU和SKU列表 ， 显示效果如下：

```
{
    "spu": {
        "brandId": 0,
        "caption": "111",
        "category1Id": 558,
        "category2Id": 559,
        "category3Id": 560,
        "commentNum": null,
        "freightId": null,
        "id": 149187842867993,
        "image": null,
        "images": null,
        "introduction": null,
        "isDelete": null,
        "isEnableSpec": "0",
        "isMarketable": "1",
        "name": "黑马智能手机",
        "paraItems": null,
        "saleNum": null,
        "saleService": null,
        "sn": null,
        "specItems": null,
        "status": null,
        "templateId": 42
    },
    "skuList": [{
        "alertNum": null,
        "brandName": "金立（Gionee）",
        "categoryId": 560,
        "categoryName": "手机",
        "commentNum": null,
        "createTime": "2018-11-06 10:17:08",
        "id": 1369324,
        "image": null,
        "images": "blob:http://localhost:8080/ec04d1a5-d865-4e7f-a313-
2e9a76cfb3f8",
        "name": "黑马智能手机",
        "num": 100,
        "price": 900000,

        "saleNum": null,
```

```
            "sn": "",
            "spec": null,
            "spuId": 149187842867993,
            "status": "1",
            "updateTime": "2018-11-06 10:17:08",
            "weight": null
        },{
            "alertNum": null,
            "brandName": "金立（Gionee）",
            "categoryId": 560,
            "categoryName": "手机",
            "commentNum": null,
            "createTime": "2018-11-06 10:17:08",
            "id": 1369325,
            "image": null,
            "images": "blob:http://localhost:8080/ec04d1a5-d865-4e7f-a313-
2e9a76cfb3f8",
            "name": "黑马智能手机",
            "num": 100,
            "price": 900000,
            "saleNum": null,
            "sn": "",
            "spec": null,
            "spuId": 149187842867993,
            "status": "1",
            "updateTime": "2018-11-06 10:17:08",
            "weight": null
        }
    ]
}
```


代码实现：

（1）qingcheng_interface工程SpuService新增方法定义

```
    /**
     * 根据ID查询商品
     * @param id
     * @return
     */
    public Goods findGoodsById(String id);
```

（2）qingcheng_service_goods工程SpuServiceImpl实现此方法

```
/**
     * 根据ID查询商品
     * @param id
     * @return
     */
    public Goods findGoodsById(String id){
        //查询spu
        Spu spu = spuMapper.selectByPrimaryKey(id);

        //查询SKU 列表
        Example example=new Example(Sku.class);
        Example.Criteria criteria = example.createCriteria();
        criteria.andEqualTo("spuId",id);
        List<Sku> skuList = skuMapper.selectByExample(example);

        //封装，返回
        Goods goods=new Goods();
        goods.setSpu(spu);
        goods.setSkuList(skuList);
        return goods;
    }
```

（3）qingcheng_web_manager工程SpuController新增方法

```
    @GetMapping("/findGoodsById")
    public Goods findGoodsById(Long id){
        return spuService.findGoodsById(id);
    }
```

2.3.4 保存修改

实现思路：

（1）修改与新增共用同一个方法

（2）通过spu的id判断当前操作是新增还是修改

（3）如果是新增需要设置spu的id，对spu执行的是insert操作

（4） 如果修改则需要删除原来的sku列表，对spu执行的是updateByPrimaryKeySelective操作。

（5）sku列表的插入部分的代码要判断sku是否有id,如果有id则不重新生成id

代码实现：

修改SpuServiceImpl的saveGoods方法，修改后代码如下：

```
    /**
     * 保存商品
     * @param goods
     */
    @Transactional
    public void saveGoods(Goods goods) {

        //保存一个spu的信息
        Spu spu = goods.getSpu();
        if(spu.getId()==null){//新增商品
            spu.setId(idWorker.nextId()+"");
            spuMapper.insert(spu);
        }else{ //修改
            //删除原来的sku列表
            Example example=new Example(Sku.class);
            Example.Criteria criteria = example.createCriteria();
            criteria.andEqualTo("spuId",spu.getId());
            skuMapper.deleteByExample(example);
            //执行spu的修改
            spuMapper.updateByPrimaryKeySelective(spu);
        }
        //保存sku列表的信息
        List<Sku> skuList = goods.getSkuList();
        for (Sku sku:skuList){

            if(sku.getId()==null){//新增
                sku.setId(idWorker.nextId()+"");
                sku.setCreateTime(date);//创建日期
            }
            //添加sku 。。。。。。（略）
        }
        //建立分类和品牌的关联
        //。。。。。。（略）
    }
```

2.3.5 未启用规格的sku处理

需求分析：

有些商品是没有区分规格的，也就是一个spu对应一个sku，这种情况下sku列表只传递一条记录，并且没有规格（spec）属性，我们要对其进行判断，避免因空值产生

实现思路：

在saveGoods方法的sku列表循环中添加代码，判断

```
//构建SKU名称，采用SPU+规格值组装
if(sku.getSpec()==null || "".equals(sku.getSpec())){
    sku.setSpec("{}");
}
```

3. 商品审核与上下架

3.1 商品审核

3.1.1 需求分析与实现思路

商品审核：新录入的商品是未审核状态，也是未上架状态。

实现思路：

（1）修改审核状态，如果审核状态为1则上架状态也修改为1

（2）记录商品审核记录，表自行设计，代码学员实现

（3）记录商品日志，表自行设计，代码学员实现

3.1.2 代码实现

（1）SpuService新增方法定义

```
    public void audit(String id,String status,String message);
```

（2）SpuServiceImpl实现方法

```
    /**
     * 商品审核
     * @param id
     * @param status
     * @param message
     */
    @Transactional
    public void audit(String id, String status, String message) {
        //1.修改状态   审核状态和上架状态
        Spu spu = new Spu();
        spu.setId(id);
        spu.setStatus(status);
        if("1".equals(status)){//审核通过
            spu.setIsMarketable("1");//自动上架
        }
        spuMapper.updateByPrimaryKeySelective(spu);

        //2.记录商品审核记录    学员实现
        //3.记录商品日志    学员实现

    }
```

（3）SpuController新增方法

```
    @GetMapping("/audit")
    public Result audit(Long id){
        spuService.audit(id);
        return new Result();
    }
```

3.2 下架商品

3.2.1 需求与实现思路

下架商品，修改上下架状态为下架。下架商品不修改审核状态。

下架商品需要记录商品日志。

3.2.2 代码实现

（1）SpuService新增方法定义

```
    /**
     * 下架商品
     * @param id
     */
    public void pull(String id);
```

（2）SpuServiceImpl实现方法

```
    /**
     * 下架商品
     * @param id
     */
    public void pull(String id) {
        Spu spu = spuMapper.selectByPrimaryKey(id);
        spu.setIsMarketable("0");//下架状态
        spuMapper.updateByPrimaryKeySelective(spu);
    }
```

（3）SpuController新增方法

```
    @GetMapping("/pull")
    public Result pull(String id){
        spuService.pull(id);
        return new Result();
    }
```

3.3 上架商品

3.3.1 需求分析

将商品修改为上架状态，需要验证该商品是否审核通过，未审核通过的商品不能上架。

上架商品需要记录商品日志。

3.3.2 代码实现

必须是通过审核的商品才能上架

（1）SpuService新增方法定义

```
    /**
     * 上架商品
     * @param id
     */
    public void put(String id);
```

（2）SpuServiceImpl实现方法

```
    /**
     * 商品上架
     * @param id
     */
    public void put(String id) {
        //1.修改状态
        Spu spu = spuMapper.selectByPrimaryKey(id);
        if(!"1".equals(spu.getStatus())){
            throw new RuntimeException("此商品未通过审核");
        }
        spu.setIsMarketable("1");
        spuMapper.updateByPrimaryKeySelective(spu);
        //2.记录商品日志（学员实现）

    }
```

（3）SpuController新增方法

```
    @GetMapping("/put")
    public Result put(String id){
        spuService.put(id);
        return new Result();
    }
```

3.4 批量上下架

3.4.1 需求分析

前端传递一组商品ID，后端进行批量上下架处理，处理后给前端返回处理的条数

3.4.2 代码实现

（1）SpuService新增方法定义

```
    /**
     * 批量上架商品
     * @param ids
     */
    public int putMany(Long[] ids);
```

（2）SpuServiceImpl实现方法

```
    /**
     * 批量上架商品
     * @param ids
     */
    public int putMany(Long[] ids) {
        Spu spu=new Spu();
        spu.setIsMarketable("1");//上架
        //批量修改
        Example example=new Example(Spu.class);
        Example.Criteria criteria = example.createCriteria();
        criteria.andIn("id", Arrays.asList(ids));//id
        criteria.andEqualTo("isMarketable","0");//下架
        criteria.andEqualTo("status","1");//审核通过的
        criteria.andEqualTo("isDelete","0");//非删除的
        return spuMapper.updateByExampleSelective(spu, example);
    }
```

（3）SpuController新增方法

```
    @GetMapping("/putMany")
    public Result putMany(Long[] ids){
        int count = spuService.putMany(ids);
        return new Result(0,"上架"+count+"个商品");
    }
```

批量下架代码略

4. 删除与还原商品（作业）

4.1 需求分析

删除商品并非物理删除（真正的执行删除数据），而是通过将表中某字段标记为删除状态。

还原商品实际就是将删除状态再修改回来。

如果商品需要物理删除，必须是先逻辑删除才能进行物理删除，删除前需要检查状态。

4.2 实现思路

逻辑删除商品，修改spu表is_delete字段为1

商品回收站显示spu表is_delete字段为1的记录

回收商品，修改spu表is_delete字段为0

代码由学员实现