尚品汇商城

一、spu 相关业务介绍

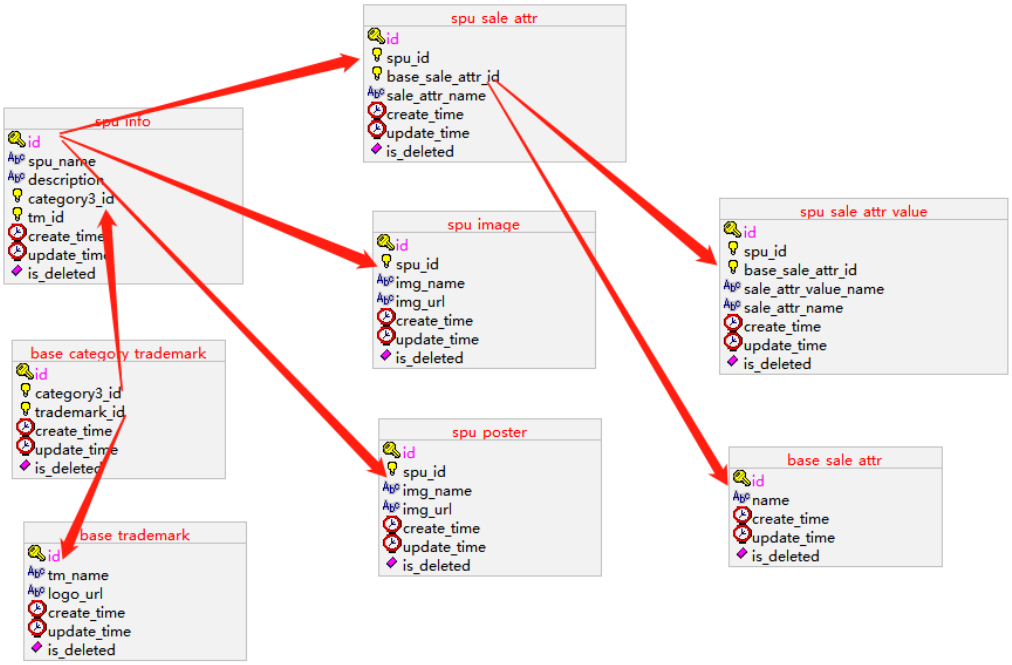
1.1 销售属性

销售属性，就是商品详情页右边，可以通过销售属性来定位一组 spu 下的哪款 sku。可以让当前的商品详情页，跳转到自己的“兄弟”商品。

一般每种商品的销售属性不会太多，大约 1-4 种。整个电商的销售属性种类也不会太多，大概 10 种以内。比如：颜色、尺寸、版本、套装等等。不同销售属性的组合也就构成了一个 spu 下多个 sku 的结构。

因此，在制作 spu 之前要先确定当前商品有哪些销售属性！

1.2 spu 数据结构图



二、列表查询功能开发

2.1 创建 mapper

```
@Mapper
public interface SpuInfoMapper extends BaseMapper<SpuInfo> {

}
```

2.2 创建接口 ManageService

```
/**
 * spu 分页查询
```

```
 * @param pageParam
 * @param spuInfo
 * @return
 */
IPage<SpuInfo>  getSpuInfoPage(Page<SpuInfo>  pageParam,  SpuInfo
spuInfo);
```

2.3 创建实现类 ManageServiceImpl

```
@Autowired
private SpuInfoMapper spuInfoMapper;


@Override
public  IPage<SpuInfo>  getSpuInfoPage(Page<SpuInfo>  pageParam,
SpuInfo spuInfo) {
    QueryWrapper<SpuInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("category3_id", spuInfo.getCategory3Id());
    queryWrapper.orderByDesc("id");
    return spuInfoMapper.selectPage(pageParam, queryWrapper);
}
```

2.4 创建控制器 SpuManageController

```
@RestController // @ResponseBody + @Controller
@RequestMapping("admin/product")
public class SpuManageController {

    @Autowired
    private ManageService manageService;

    // 根据查询条件封装控制器
 // springMVC 的时候，有个叫对象属性传值 如果页面提交过来的参数与实体类的参数
一致，
 // 则可以使用实体类来接收数据
 // http://api.gmall.com/admin/product/1/10?category3Id=61
    // @RequestBody 作用 将前台传递过来的 json{"category3Id":"61"}  字符串
变为 java 对象。
 @GetMapping("{page}/{size}")
    public Result getSpuInfoPage(@PathVariable Long page,
                                 @PathVariable Long size,
                                 SpuInfo spuInfo){
        // 创建一个 Page 对象
   Page<SpuInfo> spuInfoPage = new Page<>(page,size);
        // 获取数据
   IPage<SpuInfo>                    spuInfoPageList                    =
manageService.getSpuInfoPage(spuInfoPage, spuInfo);
```

```
        // 将获取到的数据返回即可!
    return Result.ok(spuInfoPageList);
     }
}
```

三、品牌管理

3.1 品牌列表

3.1.1 创建 mapper

```
@Mapper
public interface BaseTrademarkMapper extends
BaseMapper<BaseTrademark> {

}
```

3.1.2 创建接口 BaseTrademarkService

```
public interface BaseTrademarkService extends
IService<BaseTrademark> {

    /**
     * Banner 分页列表
   * @param pageParam
     * @return
     */
    IPage<BaseTrademark> getPage(Page<BaseTrademark> pageParam);

}
```

3.1.3 创建实现类 BaseTrademarkServiceImpl

```
@Service
public class BaseTrademarkServiceImpl extends ServiceImpl<BaseTrademarkMapper, BaseTrademark> implements BaseTrademarkService {

    @Autowired
    private BaseTrademarkMapper baseTrademarkMapper;

    @Override
    public IPage<BaseTrademark> getPage(Page<BaseTrademark> pageParam) {
        QueryWrapper<BaseTrademark> queryWrapper = new QueryWrapper<>();
        queryWrapper.orderByAsc("id");

        IPage<BaseTrademark> page = baseTrademarkMapper.selectPage(pageParam, queryWrapper);
        return page;
    }
}
```

3.1.4 创建控制器 BaseTrademarkController

```
@RestController
@RequestMapping("/admin/product/baseTrademark")
public class BaseTrademarkController {

    @Autowired
    private BaseTrademarkService baseTrademarkService;

    @ApiOperation(value = "分页列表")
    @GetMapping("{page}/{limit}")
    public Result index(@PathVariable Long page,
                        @PathVariable Long limit) {

        Page<BaseTrademark> pageParam = new Page<>(page, limit);
        IPage<BaseTrademark> pageModel = baseTrademarkService.getPage(pageParam);
        return Result.ok(pageModel);
    }

    @ApiOperation(value = "获取 BaseTrademark")
    @GetMapping("get/{id}")
    public Result get(@PathVariable String id) {
```

```
        BaseTrademark                    baseTrademark                    =
baseTrademarkService.getById(id);
        return Result.ok(baseTrademark);
    }

    @ApiOperation(value = "新增 BaseTrademark")
    @PostMapping("save")
    public Result save(@RequestBody BaseTrademark banner) {
            baseTrademarkService.save(banner);
        return Result.ok();
    }

    @ApiOperation(value = "修改 BaseTrademark")
    @PutMapping("update")
    public Result updateById(@RequestBody BaseTrademark banner) {
            baseTrademarkService.updateById(banner);
        return Result.ok();
    }

    @ApiOperation(value = "删除 BaseTrademark")
    @DeleteMapping("remove/{id}")
    public Result remove(@PathVariable Long id) {
            baseTrademarkService.removeById(id);
        return Result.ok();
    }

}
```

3.2 分类品牌列表

3.2.1 创建 mapper

```
@Mapper
public          interface          BaseCategoryTrademarkMapper          extends
BaseMapper<BaseCategoryTrademark> {
}
```

3.2.2 创建接口

```
package com.atguigu.gmall.product.service;
```

```
public interface BaseCategoryTrademarkService extends IService<BaseCategoryTrademark> {

    /**
     * 根据三级分类获取品牌
     * @param category3Id
     * @return
     */
    List<BaseTrademark> findTrademarkList(Long category3Id);

    /**
     * 保存分类与品牌关联
     * @param categoryTrademarkVo
     */
    void save(CategoryTrademarkVo categoryTrademarkVo);

    /**
     * 获取当前未被三级分类关联的所有品牌
     * @param category3Id
     * @return
     */
    List<BaseTrademark> findCurrentTrademarkList(Long category3Id);

    /**
     * 删除关联
     * @param category3Id
     * @param trademarkId
     */
    void remove(Long category3Id, Long trademarkId);
}
```

3.2.3 创建实现类

```
package com.atguigu.gmall.product.service.impl;

@Service
public class BaseCategoryTrademarkServiceImpl extends ServiceImpl<BaseCategoryTrademarkMapper,BaseCategoryTrademark> implements BaseCategoryTrademarkService {

    //  调用mapper 层!
  @Autowired
    private BaseTrademarkMapper baseTrademarkMapper;

    @Autowired
    private BaseCategoryTrademarkMapper baseCategoryTrademarkMapper;
```

```
    @Override
    public List<BaseTrademark> findTrademarkList(Long category3Id) {
        //  根据分类Id 获取到品牌Id 集合数据
   //  select * from base_category_trademark where category3_id = ?;
        QueryWrapper<BaseCategoryTrademark>
baseCategoryTrademarkQueryWrapper = new QueryWrapper<>();
        baseCategoryTrademarkQueryWrapper.eq("category3_id",category3Id);
        List<BaseCategoryTrademark>       baseCategoryTrademarkList       =
baseCategoryTrademarkMapper.selectList(baseCategoryTrademarkQueryWrapper);

        //  判断baseCategoryTrademarkList 这个集合
   if(!CollectionUtils.isEmpty(baseCategoryTrademarkList)){
            //  需要获取到这个集合中的品牌Id 集合数据
     List<Long>                        tradeMarkIdList                    =
baseCategoryTrademarkList.stream().map(baseCategoryTrademark -> {
                return baseCategoryTrademark.getTrademarkId();
            }).collect(Collectors.toList());
            //  正常查询数据的话... 需要根据品牌Id 来获取集合数据!
     return baseTrademarkMapper.selectBatchIds(tradeMarkIdList);
        }
        //  如果集合为空，则默认返回空
   return null;
    }

    @Override
    public  void  removeBaseCategoryTrademarkById(Long  category3Id,  Long
trademarkId) {
        //  逻辑删除: 本质更新操作 is_deleted
        //  更新:  update base_category_trademark set is_deleted = 1 where
category3_id=? and trademark_id=?;
        QueryWrapper<BaseCategoryTrademark>
baseCategoryTrademarkQueryWrapper = new QueryWrapper<>();
        baseCategoryTrademarkQueryWrapper.eq("category3_id",category3Id);
        baseCategoryTrademarkQueryWrapper.eq("trademark_id",trademarkId);

baseCategoryTrademarkMapper.delete(baseCategoryTrademarkQueryWrapper);

    }

    @Override
    public List<BaseTrademark> findCurrentTrademarkList(Long category3Id) {
        //  哪些是关联的品牌Id
        QueryWrapper<BaseCategoryTrademark>
baseCategoryTrademarkQueryWrapper = new QueryWrapper<>();
        baseCategoryTrademarkQueryWrapper.eq("category3_id",category3Id);
        List<BaseCategoryTrademark>       baseCategoryTrademarkList       =
baseCategoryTrademarkMapper.selectList(baseCategoryTrademarkQueryWrapper);

        //  判断
   if (!CollectionUtils.isEmpty(baseCategoryTrademarkList)){
            //  找到关联的品牌Id 集合数据 {1,3}
            List<Long>                    tradeMarkIdList                    =
baseCategoryTrademarkList.stream().map(baseCategoryTrademark -> {
                return baseCategoryTrademark.getTrademarkId();
            }).collect(Collectors.toList());
            //  在所有的品牌Id 中将这些有关联的品牌Id 给过滤掉就可以!
```

```
    //          select      *      from      base_trademark;      外      面
baseTrademarkMapper.selectList(null) {1,2,3,5}
            List<BaseTrademark>              baseTrademarkList              =
baseTrademarkMapper.selectList(null).stream().filter(baseTrademark -> {
                return !tradeMarkIdList.contains(baseTrademark.getId());
            }).collect(Collectors.toList());
            //  返回数据
     return baseTrademarkList;
        }
        //  如果说这个三级分类Id 下没有任何品牌！则获取到所有的品牌数据！
   return baseTrademarkMapper.selectList(null);
    }

    @Override
    public void save(CategoryTrademarkVo categoryTrademarkVo) {
        /*
        private Long category3Id;
        private List<Long> trademarkIdList;

        category3Id 61 tmId 2;
        category3Id 61 tmId 5;
         */
        //  获取到品牌Id 集合数据
   List<Long> trademarkIdList = categoryTrademarkVo.getTrademarkIdList();

        //  判断
   if (!CollectionUtils.isEmpty(trademarkIdList)){
           //  做映射关系
     List<BaseCategoryTrademark>          baseCategoryTrademarkList          =
trademarkIdList.stream().map((trademarkId) -> {
                //  创建一个分类Id 与品牌的关联的对象
      BaseCategoryTrademark        baseCategoryTrademark        =        new
BaseCategoryTrademark();

baseCategoryTrademark.setCategory3Id(categoryTrademarkVo.getCategory3Id());
                baseCategoryTrademark.setTrademarkId(trademarkId);
                //  返回数据
      return baseCategoryTrademark;
            }).collect(Collectors.toList());

            //  集中保存到数据库     baseCategoryTrademarkList
            this.saveBatch(baseCategoryTrademarkList);
        }
    }
}
```

3.2.4 创建控制器

```
package com.atguigu.gmall.product.controller;

@RestController
```

```
@RequestMapping("admin/product/baseCategoryTrademark")
public class BaseCategoryTrademarkController {

    @Autowired
    private BaseCategoryTrademarkService baseCategoryTrademarkService;


    @PostMapping("save")
    public Result save(@RequestBody CategoryTrademarkVo categoryTrademarkVo){
        // 保存方法
        baseCategoryTrademarkService.save(categoryTrademarkVo);
        return Result.ok();
    }

    @DeleteMapping("remove/{category3Id}/{trademarkId}")
    public Result remove(@PathVariable Long category3Id, @PathVariable Long trademarkId){
        // 调用服务层方法
        baseCategoryTrademarkService.remove(category3Id, trademarkId);
        return Result.ok();
    }

    @GetMapping("findTrademarkList/{category3Id}")
    public Result findTrademarkList(@PathVariable Long category3Id){
        // select * from base_trademark
        List<BaseTrademark> list = baseCategoryTrademarkService.findTrademarkList(category3Id);
        // 返回
        return Result.ok(list);
    }

    @GetMapping("findCurrentTrademarkList/{category3Id}")
    public Result findCurrentTrademarkList(@PathVariable Long category3Id){
        List<BaseTrademark> list = baseCategoryTrademarkService.findCurrentTrademarkList(category3Id);
        // 返回
        return Result.ok(list);
    }
}
```

四、spu 的保存功能中的图片上传

4.1 MinIo 介绍

MinIO 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

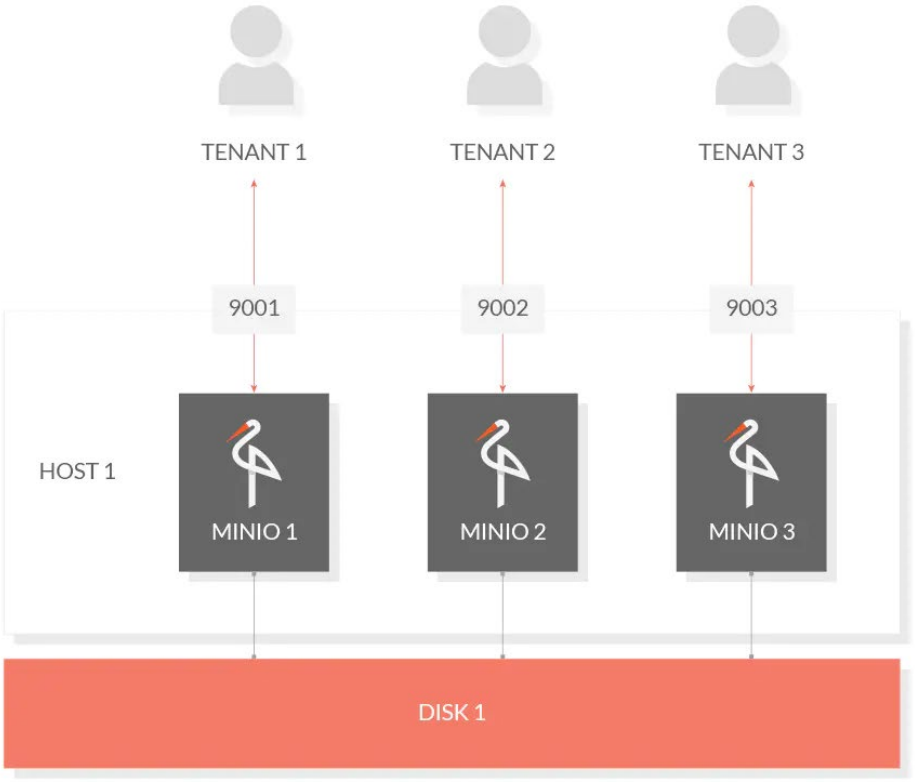
MinIO 是一个非常轻量的服务,可以很简单的和其他应用的结合，类似 NodeJS, Redis 或者 MySQL。

官方文档：http://docs.minio.org.cn/docs 旧一点

https://docs.min.io/ 新

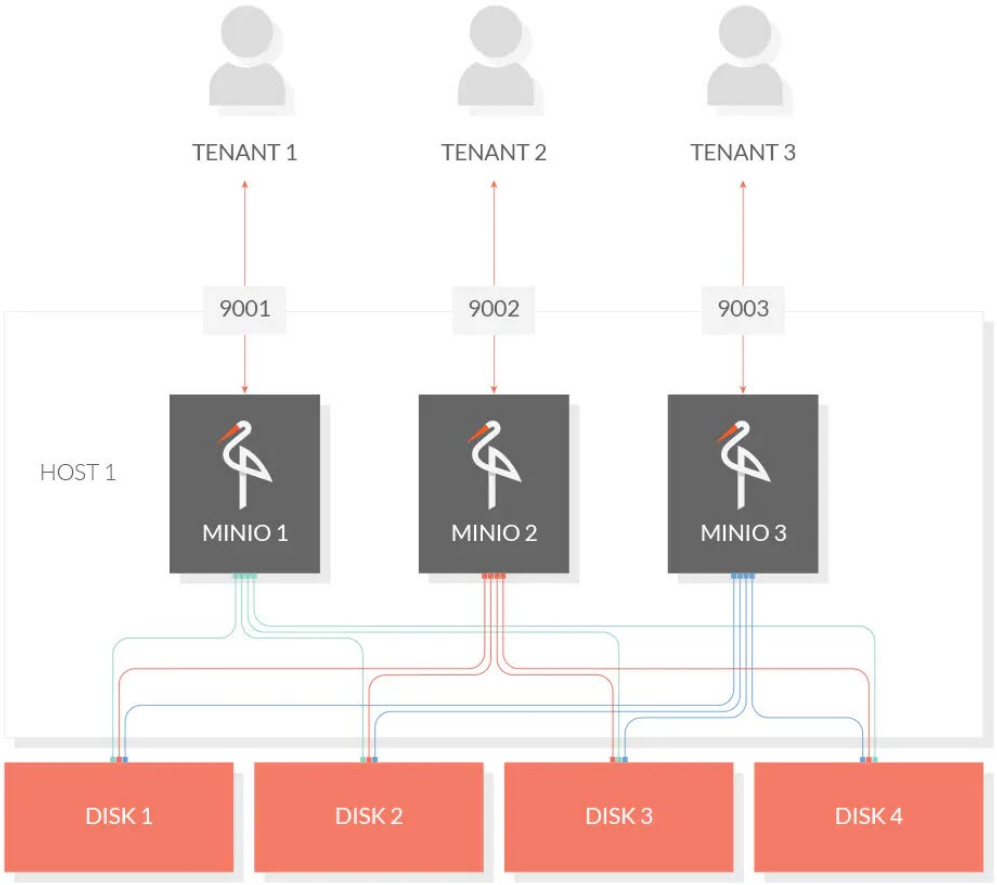
4.2 应用场景

4.2.1 单主机单硬盘模式



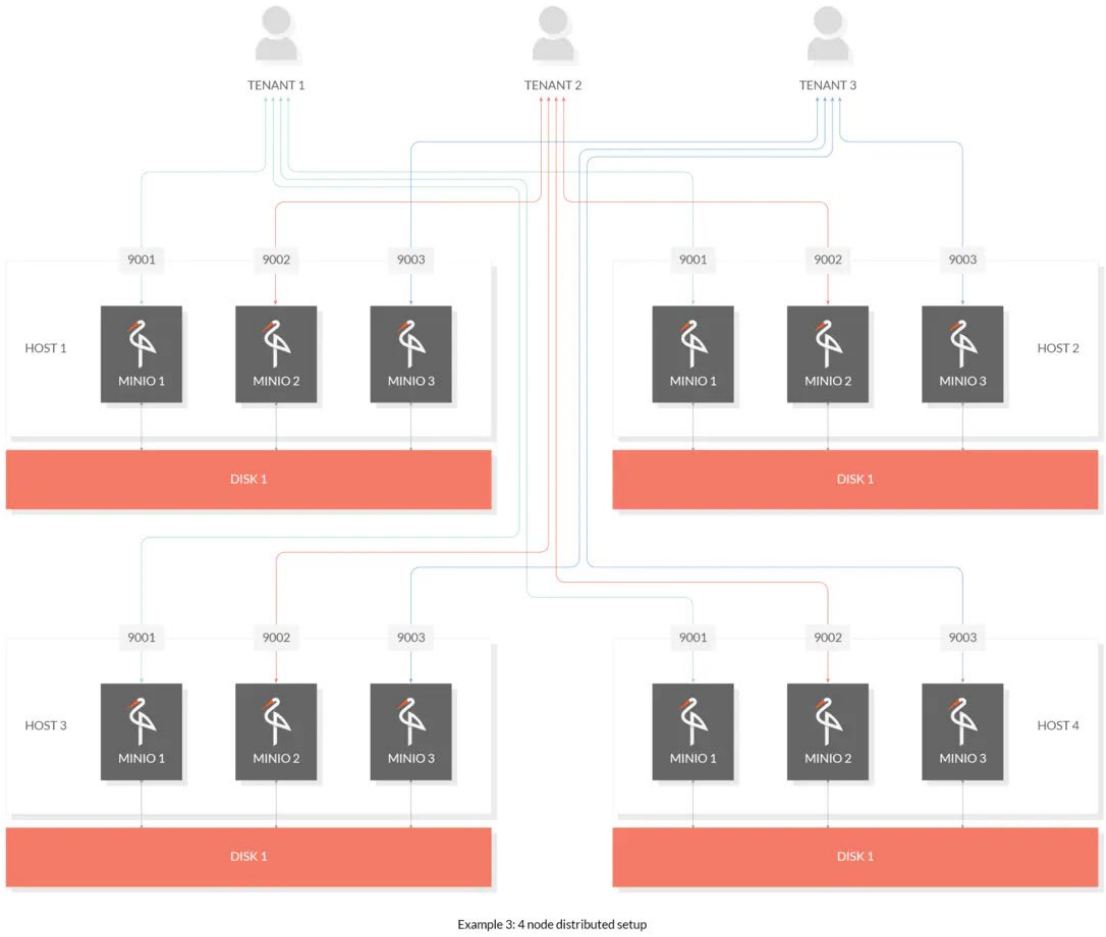
Example 1: 3 tenants on single host, single drive

4.2.2 单主机多硬盘模式



Example 2: 3 tenants on single host, 4 drives (erasure code)

4.2.3 多主机多硬盘分布式



Example 3: 4 node distributed setup

4.3 特点

- 高性能：作为高性能对象存储，在标准硬件条件下它能达到 55GB/s 的读、35GB/s 的写速率
- 可扩容：不同 MinIO 集群可以组成联邦，并形成一个全局的命名空间，并跨越多个数据中心
- 云原生：容器化、基于 K8S 的编排、多租户支持
- Amazon S3 兼容：Minio 使用 Amazon S3 v2 / v4 API。可以使用 Minio SDK，Minio Client，AWS SDK 和 AWS CLI 访问 Minio 服务器。

- 可对接后端存储: 除了 Minio 自己的文件系统，还支持 DAS、 JBODs、NAS、Google 云存储和 Azure Blob 存储。
- SDK 支持: 基于 Minio 轻量的特点，它得到类似 Java、Python 或 Go 等语言的 sdk 支持
- Lambda 计算: Minio 服务器通过其兼容 AWS SNS / SQS 的事件通知服务触发 Lambda 功能。支持的目标是消息队列，如 Kafka，NATS，AMQP，MQTT，Webhooks 以及 Elasticsearch，Redis，Postgres 和 MySQL 等数据库。
- 有操作页面
- 功能简单: 这一设计原则让 MinIO 不容易出错、更快启动
- 支持纠删码：MinIO 使用纠删码、Checksum 来防止硬件错误和静默数据污染。在最高冗余度配置下，**即使丢失 1/2 的磁盘也能恢复数据！**

4.4 存储机制

Minio 使用纠删码 erasure code 和校验和 checksum。 即便丢失一半数量（N/2）的硬盘，仍然可以恢复数据。

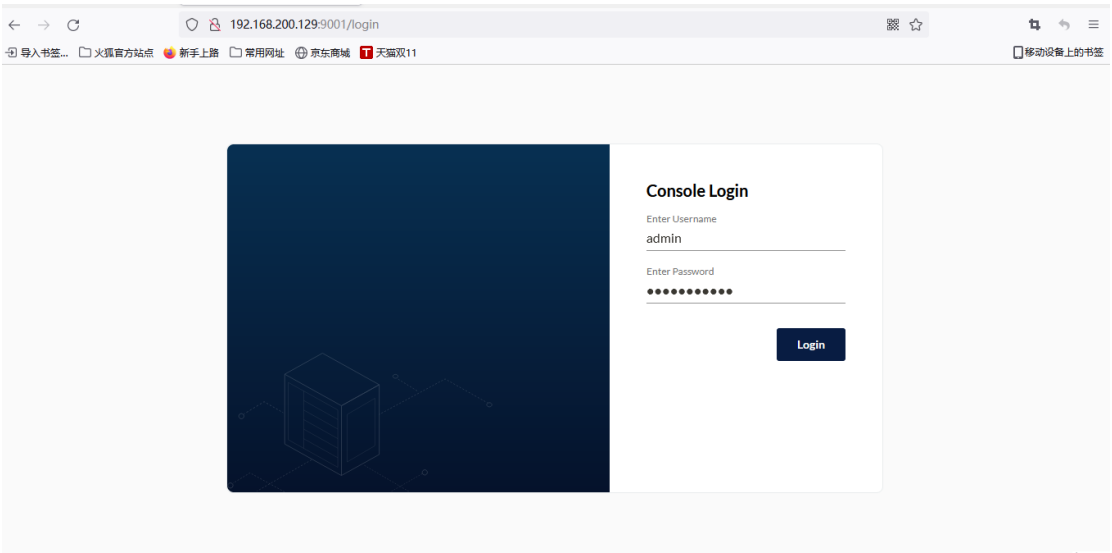
纠删码是一种恢复丢失和损坏数据的数学算法。

4.5 docker 安装 Minio

docker pull minio/minio
新版本: docker run \ -p 9000:9000 \ -p 9001:9001 \ --name minio \ -d --restart=always \ -e "MINIO_ROOT_USER=**admin**" \ -e "MINIO_ROOT_PASSWORD=**admin123456**" \ -v /home/data:/data \

```
-v /home/config:/root/.minio \
minio/minio server /data --console-address ":9001"
```

浏览器访问：http://IP:9000/minio/login，如图：



说明：安装是指定了登录账号

4.6 利用 Java 客户端调用 Minio

参考文档：https://docs.min.io/docs/java-client-api-reference.html

4.6.1 引入依赖

在 service-product 模块中添加依赖

```
<dependency>
    <groupId>io.minio</groupId>
    <artifactId>minio</artifactId>
    <version>8.2.0</version>
</dependency>
```

4.6.2 添加配置文件

在 nacos 中配置好了！

```
minio:
  endpointUrl: http://IP:9000
  accessKey: admin
  secreKey: admin123456
  bucketName: gmall
```

4.6.3 创建 FileUploadController 控制器

```
package com.atguigu.gmall.product.controller;

@RestController
@RequestMapping("admin/product")
public class FileUploadController {

    //  获取文件上传对应的地址
    @Value("${minio.endpointUrl}")
    public String endpointUrl;

    @Value("${minio.accessKey}")
    public String accessKey;

    @Value("${minio.secreKey}")
    public String secreKey;

    @Value("${minio.bucketName}")
    public String bucketName;

    //  文件上传控制器
  @PostMapping("fileUpload")
    public Result fileUpload(MultipartFile file) throws Exception{
        //  准备获取到上传的文件路径!
    String url = "";

        // 使用 MinIO 服务的 URL，端口，Access key 和 Secret key 创建一个 MinioClient 对象
    // MinioClient minioClient = new MinioClient("https://play.min.io", "Q3AM3UQ867SPQQA43P2F", "zuf+tfteSlswRu7BJ86wekitnifILbZam1KYY3TG");
        MinioClient minioClient =
                MinioClient.builder()
                        .endpoint(endpointUrl)
                        .credentials(accessKey, secreKey)
                        .build();
        // 检查存储桶是否已经存在
    boolean isExist =
minioClient.bucketExists(BucketExistsArgs.builder().bucket(bucketName).
build());
        if(isExist) {
            System.out.println("Bucket already exists.");
        } else {
            // 创建一个名为asiatrip 的存储桶，用于存储照片的zip 文件。
```

```
        minioClient.makeBucket(MakeBucketArgs.builder()
                        .bucket(bucketName)
                        .build());
            }
            //  定义一个文件的名称 ： 文件上传的时候，名称不能重复！
        String          fileName          =          System.currentTimeMillis()+
UUID.randomUUID().toString();
            // 使用putObject 上传一个文件到存储桶中。
        //                  minioClient.putObject("asiatrip","asiaphotos.zip",
"/home/user/Photos/asiaphotos.zip");
            minioClient.putObject(

PutObjectArgs.builder().bucket(bucketName).object(fileName).stream(
                            file.getInputStream(), file.getSize(), -1)
                            .contentType(file.getContentType())
                            .build());
            //      System.out.println("/home/user/Photos/asiaphotos.zip   is
successfully uploaded as asiaphotos.zip to `asiatrip` bucket.");
            //  文件上传之后的路径：  http://39.99.159.121:9000/gmall/xxxxxx
            url = endpointUrl+"/"+bucketName+"/"+fileName;

            System.out.println("url:\t"+url);
            //  将文件上传之后的路径返回给页面！
        return Result.ok(url);
        }
}
```

注意：文件上传时，需要调整一下 linux 服务器的时间与 windows 时间一致！

第一步：安装 ntp 服务

```
yum -y install ntp
```

第二步：开启开机启动服务

```
systemctl enable ntpd
```

第三步：启动服务

```
systemctl start ntpd
```

第四步：更改时区

```
timedatectl set-timezone Asia/Shanghai
```

第五步：启用 ntp 同步

```
timedatectl set-ntp yes
```

第六步：同步时间

```
ntpq -p
```

五、spu 保存

5.1 加载销售属性

5.1.1 创建 mapper

```
@Mapper
public interface BaseSaleAttrMapper extends BaseMapper<BaseSaleAttr>
{
}
```

5.1.2 在 MangeService 添加接口

接口

```
/**
 * 查询所有的销售属性数据
 * @return
 */
List<BaseSaleAttr> getBaseSaleAttrList();
```

实现类

```
@Autowired
private BaseSaleAttrMapper baseSaleAttrMapper;
```

```
@Override
public List<BaseSaleAttr> getBaseSaleAttrList() {
    return baseSaleAttrMapper.selectList(null);
}
```

5.1.4 添加控制器 SpuManageController

```
@RestController
@RequestMapping("admin/product")
public class SpuManageController {

    // 引入服务层
  @Autowired
    private ManageService manageService;

    // 销售属性 http://api.gmall.com/admin/product/baseSaleAttrList
    @GetMapping("baseSaleAttrList")
    public Result baseSaleAttrList(){
        // 查询所有的销售属性集合
    List<BaseSaleAttr>                          baseSaleAttrList            =
manageService.getBaseSaleAttrList();

        return Result.ok(baseSaleAttrList);
    }

}
```

5.2 加载品牌数据

```
BaseCategoryTrademarkController

/**
 * 查询全部品牌
 * @return
 */
@GetMapping("findTrademarkList/{category3Id}")
public Result findTrademarkList(@PathVariable Long category3Id){
    //  调用服务层方法
  List<BaseTrademark>             baseCategoryTrademarkList             =
baseCategoryTrademarkService.findTrademarkList(category3Id);
    //  返回数据
  return Result.ok(baseCategoryTrademarkList);
}
```

5.3 保存后台代码

5.3.1 创建 mapper

建立对应的 mapper 文件

```
@Mapper
public interface SpuImageMapper extends BaseMapper<SpuImage> {
}

@Mapper
public interface SpuSaleAttrMapper extends BaseMapper<SpuSaleAttr> {

}

@Mapper
public interface SpuSaleAttrValueMapper extends
BaseMapper<SpuSaleAttrValue> {
}

@Mapper
public interface SpuPosterMapper extends BaseMapper<SpuPoster> {
}
```

5.3.2 添加数据接口

```
/**
 * 保存商品数据
 * @param spuInfo
 */
void saveSpuInfo(SpuInfo spuInfo);
```

```
@Override
@Transactional(rollbackFor = Exception.class)
public void saveSpuInfo(SpuInfo spuInfo) {
    /*
        spuInfo;
        spuImage;
        spuSaleAttr;
        spuSaleAttrValue;
        spuPoster
     */
    spuInfoMapper.insert(spuInfo);
```

```
    //  获取到 spuImage 集合数据
    List<SpuImage> spuImageList = spuInfo.getSpuImageList();
    //  判断不为空
    if (!CollectionUtils.isEmpty(spuImageList)){
        //  循环遍历
        for (SpuImage spuImage : spuImageList) {
            //  需要将 spuId 赋值
            spuImage.setSpuId(spuInfo.getId());
            //  保存 spuImge
            spuImageMapper.insert(spuImage);
        }
    }
    //  获取销售属性集合
    List<SpuSaleAttr> spuSaleAttrList = spuInfo.getSpuSaleAttrList();
    //  判断
    if (!CollectionUtils.isEmpty(spuSaleAttrList)){
        //  循环遍历
        for (SpuSaleAttr spuSaleAttr : spuSaleAttrList) {
            //  需要将 spuId 赋值
            spuSaleAttr.setSpuId(spuInfo.getId());
            spuSaleAttrMapper.insert(spuSaleAttr);

            //  再此获取销售属性值集合
            List<SpuSaleAttrValue>          spuSaleAttrValueList          =
spuSaleAttr.getSpuSaleAttrValueList();
            //  判断
            if (!CollectionUtils.isEmpty(spuSaleAttrValueList)){
                //  循环遍历
                for        (SpuSaleAttrValue        spuSaleAttrValue        :
spuSaleAttrValueList) {
                    //   需要将 spuId,  sale_attr_name 赋值
                    spuSaleAttrValue.setSpuId(spuInfo.getId());

spuSaleAttrValue.setSaleAttrName(spuSaleAttr.getSaleAttrName());
                    spuSaleAttrValueMapper.insert(spuSaleAttrValue);
                }
            }
        }
    }

    //  获取到 posterList 集合数据
    List<SpuPoster> spuPosterList = spuInfo.getSpuPosterList();
    //  判断不为空
    if (!CollectionUtils.isEmpty(spuPosterList)){
        for (SpuPoster spuPoster : spuPosterList) {
            //  需要将 spuId 赋值
            spuPoster.setSpuId(spuInfo.getId());
            //  保存 spuPoster
```

```
                spuPosterMapper.insert(spuPoster);
            }
        }
}
```

5.3.3 添加控制器

```
/**
 * 保存spu
 * @param spuInfo
 * @return
 */
@PostMapping("saveSpuInfo")
public Result saveSpuInfo(@RequestBody SpuInfo spuInfo){
    // 调用服务层的保存方法
  manageService.saveSpuInfo(spuInfo);
    return Result.ok();
}
```