第2章 分布式文件存储FastDFS

学习目标

- 理解FastDFS工作流程:工作的整体流程 组件

```
分布式文件管理系统
        文件上传
        文件下载
        文件删除
        文件缓存控制
```

- 搭建文件上传微服务
- 相册管理(实战)
- 规格参数管理(实战)
- 商品分类管理(实战)

1 FastDFS

目标

- 了解FastDFS
- 使用FastDFS实现文件上传下载 文件信息查看

路径

- FastDFS结构介绍:组件,组件的工作流程(面试)
- FastDFS上传流程讲解
- FastDFS搭建讲解
- FastDFS文件访问Nginx配置
- 文件服务器搭建--创建文件管理的微服务(文件进行增删查)
- 文件上传
- 图片域名配置(通过IP地址+端口号+文件的名字=访问图片--192.168.211.132:80/1.jpg)==（域名+文件名=访问的图片---www.itgheima.com/1.jpg）
- 品牌图片上传实现

讲解

1.1 FastDFS简介

1.1.1 FastDFS体系结构

FastDFS是一个开源的轻量级分布式文件系统，它对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。

FastDFS优点：

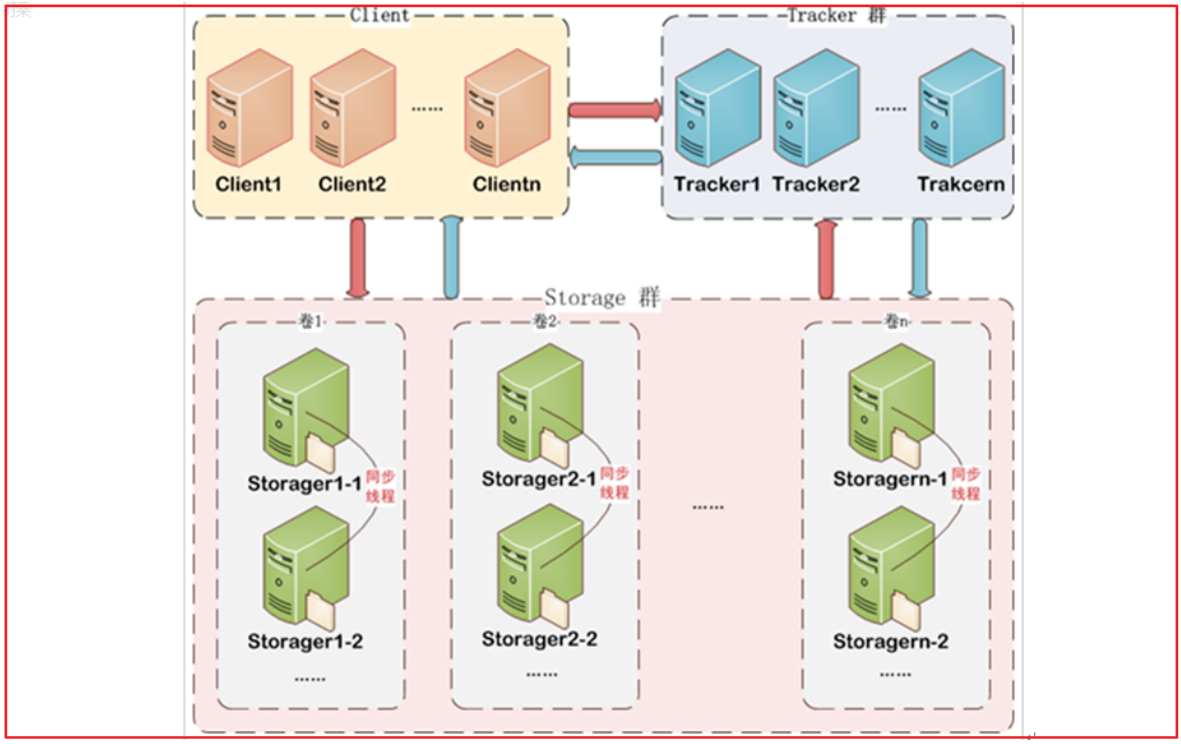
FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

FastDFS的组件：

FastDFS 架构包括 Tracker server 和 Storage server。客户端请求 Tracker server 进行文件上传、下载，通过Tracker server 调度最终由 Storage server 完成文件上传和下载。

FastDFS组件的职能：

Tracker server 作用是负载均衡和调度，通过 Tracker server 在文件上传时可以根据一些策略找到 Storage server 提供文件上传服务。可以将 tracker 称为追踪服务器或调度服务器。Storage server 作用是文件存储，客户端上传的文件最终存储在 Storage 服务器上，Storageserver 没有实现自己的文件系统而是利用操作系统的文件系统来管理文件。可以将storage称为存储服务器。



同步线程: 数据的备份同步

多个组: 数据分片

总结：

FastDFS：

```
FastDFS是一个开源的轻量级分布式文件管理系统，提供了文件管理功能，例如文件存储、文件上传、文件下载、文件同步、文件信息访问等功能，支持服务集群，解决了水平扩容问题，实现了文件下载和上传的负载均衡问题。
```

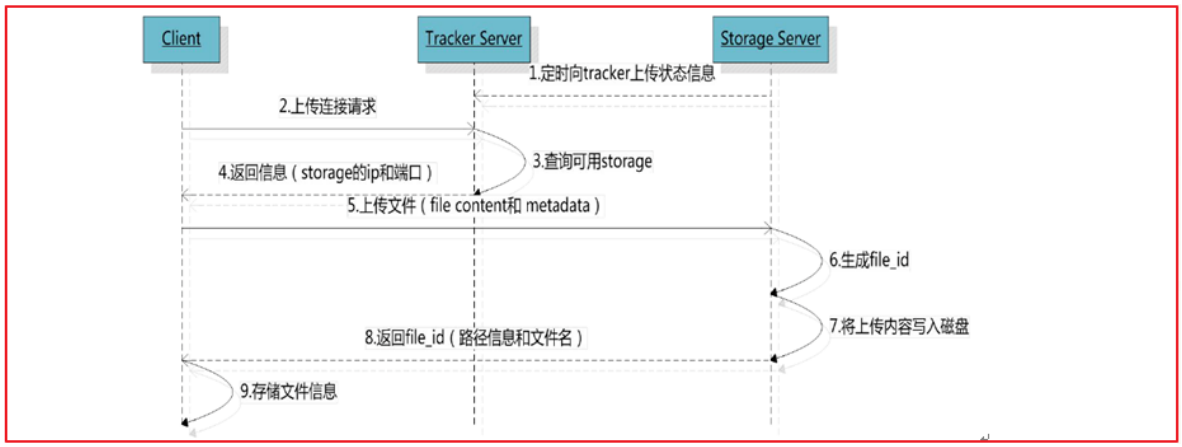
FastDFS组件：

```
1.Tracker:
        负责任务调度，负载均衡管理以及Storage的注册中心功能
2.Storage:
        负责文件上传和下载以及文件的删除等管理功能
```

FastDFS特性：

```
FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。
```

1.1.2 上传流程



客户端上传文件后存储服务器将文件 ID 返回给客户端，此文件 ID 用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

组名：文件上传后所在的 storage 组名称，在文件上传成功后有storage 服务器返回，需要客户端自行保存。

虚拟磁盘路径：storage 配置的虚拟路径，与磁盘选项store_path*对应。如果配置了

store_path0 则是 M00，如果配置了 store_path1 则是 M01，以此类推。

数据两级目录：storage 服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据

文件。

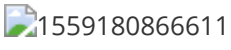
文件名：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储

服务器 IP 地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

1.2 FastDFS搭建

1.2.1 安装FastDFS镜像

我们使用Docker搭建FastDFS的开发环境,虚拟机中已经下载了fastdfs的镜像，可以通过 `docker images` 查看，如下图：


1559180866611

拉取镜像(已经下载了该镜像，大家无需下载了)

```
docker pull morunchang/fastdfs
```

运行tracker

```
docker run -d --name tracker --net=host morunchang/fastdfs sh tracker.sh
```

运行storage

```
docker run -d --name storage --net=host -e TRACKER_IP=192.168.211.132:22122 -e GROUP_NAME=group1 morunchang/fastdfs sh storage.sh
```

- 使用的网络模式是–net=host, 此时会将宿主机的网络应用于容器，链接容器就可以直接使用宿主机的IP192.168.211.132
- `sh tracker.sh`运行tracker.sh脚本文件
- group1是组名，即storage的组
- 如果想要增加新的storage服务器，再次运行该命令，注意更换 新组名

1.2.2 配置Nginx

Nginx在这里主要提供对FastDFS图片访问的支持，Docker容器中已经集成了Nginx，我们需要修改nginx的配置,进入storage的容器内部，修改nginx.conf

```
docker exec -it storage  /bin/bash
```

进入后

```
vi /etc/nginx/conf/nginx.conf
```

添加以下内容：

```
location ~ /M00 {
        ngx_fastdfs_module;
}
```

上图配置如下：

```
location ~ /M00 {
     ngx_fastdfs_module;
}
```

访问图片的时候，浏览器通常都会对图片进行缓存，如果有禁止缓存，可以设置nginx配置添加禁止缓存即可。

禁止缓存：

```
add_header Cache-Control no-store;
```

退出容器：

```
exit
```

重启storage容器：

```
docker restart storage
```

查看启动容器`docker ps`

```
9f2391f73d97 morunchang/fastdfs "sh storage.sh" 12 minutes ago Up 12 seconds storage
e22a3c7f95ea morunchang/fastdfs "sh tracker.sh" 13 minutes ago Up 13 minutes tracker
```

开启启动设置：

```
docker update --restart=always tracker
docker update --restart=always storage
```

安装Nginx目的：

```
nginx集成了FastDFS，可以通过它的ngx_fastdfs_module模块，可以通过该模块访问Tracker获取图片所存储的Storage信息，然后访问Storage信息获取图片信息。
```

1.3 文件存储微服务

创建文件管理微服务changgou-service-file，该工程主要用于实现文件上传以及文件删除等功能。

1.3.1 pom.xml依赖

修改pom.xml，引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>changgou-service</artifactId>
        <groupId>com.changgou</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>changgou-service-file</artifactId>
    <description>文件上传工程</description>

    <!--依赖包-->
    <dependencies>
        <dependency>
            <groupId>net.oschina.zcx7878</groupId>
            <artifactId>fastdfs-client-java</artifactId>
            <version>1.27.0.0</version>
        </dependency>
        <dependency>
            <groupId>com.changgou</groupId>
            <artifactId>changgou-common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
```

```
</project>
```

1.3.2 FastDFS配置

在resources文件夹下创建fasfDFS的配置文件fdfs_client.conf

```
charset=UTF-8
http.tracker_http_port=8080
tracker_server=192.168.211.132:22122
```

charset： 字符集

http.tracker_http_port:tracker的Http端口号

tracker_server： tracker服务器IP和端口设置

1.3.3 application.yml配置

在resources文件夹下创建application.yml

```
spring:
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB
  application:
    name: file
server:
  port: 18082
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:7001/eureka
  instance:
    prefer-ip-address: true
```

max-file-size是单个文件大小，max-request-size是设置总上传的数据大小

1.3.4 启动类

创建com.changgou包，创建启动类FileApplication

```
@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
@EnableEurekaClient
public class FileApplication {

    public static void main(String[] args) {
        SpringApplication.run(FileApplication.class);
    }
}
```

这里禁止了DataSource的加载创建。

1.3.5 文件微服务域名配置

我们给文件服务器配置一下域名`file-changgou-java.itheima.net`，用户访问`file-changgou-java.itheima.net`的时候，我们需要把请求路由到本地电脑，流程如下图：

1)在自己的电脑的`C:\Windows\System32\drivers\etc\hosts`文件中添加一条域名的映解析记录，让自己访问域名`file-changgou-java.itheima.net`的时候能够访问虚拟机：

```
192.168.211.132 file-changgou-java.itheima.net
```

2)修改`/usr/local/openresty/nginx/conf/nginx.conf`，添加`file-changgou-java.itheima.net`域名的配置，配置如下：

```
#图片上传路径
 server {
     listen 80;
     server_name file-changgou-java.itheima.net;

     location / {
       proxy_pass http://myip.com:18082;
     }
 }
```

1.4 文件上传

1.4.1 文件信息封装

文件上传一般都有文件的名字、文件的内容、文件的扩展名、文件的md5值、文件的作者等相关属性，我们可以创建一个对象封装这些属性，代码如下：

创建`com.changgou.file.FastDFSFile`代码如下：

```
public class FastDFSFile implements Serializable {

    //文件名字
    private String name;
    //文件内容
```

```
    private byte[] content;
    //文件扩展名
    private String ext;
    //文件MD5摘要值
    private String md5;
    //文件创建作者
    private String author;

    public FastDFSFile(String name, byte[] content, String ext, String md5,
String author) {
        this.name = name;
        this.content = content;
        this.ext = ext;
        this.md5 = md5;
        this.author = author;
    }

    public FastDFSFile(String name, byte[] content, String ext) {
        this.name = name;
        this.content = content;
        this.ext = ext;
    }

    public FastDFSFile() {
    }

    //..get..set..toString
}
```

1.4.2 文件操作

创建`com.changgou.util.FastDFSClient`类,在该类中实现FastDFS信息获取以及文件的相关操作，代码如下：

(1)初始化Tracker信息

在`com.changgou.util.FastDFSClient`类中初始化Tracker信息,在类中添加如下静态块：

```
/***
 * 初始化tracker信息
 */
static {
    try {
        //获取tracker的配置文件fdfs_client.conf的位置
        String filePath = new
ClassPathResource("fdfs_client.conf").getPath();
        //加载tracker配置信息
        ClientGlobal.init(filePath);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(2)文件上传

在类中添加如下方法实现文件上传：

```
/****
 * 文件上传
 * @param file : 要上传的文件信息封装->FastDFSFile
 * @return String[]
 *          1:文件上传所存储的组名
 *          2:文件存储路径
 */
public static String[] upload(FastDFSFile file){
    //获取文件作者
    NameValuePair[] meta_list = new NameValuePair[1];
    meta_list[0] =new NameValuePair(file.getAuthor());

    /***
     * 文件上传后的返回值
     * uploadResults[0]:文件上传所存储的组名，例如:group1
     * uploadResults[1]:文件存储路径,例如:
M00/00/00/wKjThF0DBzaAP23MAAXz2mMp9oM26.jpeg
     */
    String[] uploadResults = null;
    try {
        //创建TrackerClient客户端对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient对象获取TrackerServer信息
        TrackerServer trackerServer = trackerClient.getConnection();
        //获取StorageClient对象
        StorageClient storageClient = new StorageClient(trackerServer,
null);
        //执行文件上传
        uploadResults = storageClient.upload_file(file.getContent(),
file.getExt(), meta_list);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return uploadResults;
}
```

(3)获取文件信息

在类中添加如下方法实现获取文件信息：

```
/***
 * 获取文件信息
 * @param groupName:组名
 * @param remoteFileName：文件存储完整名
 */
public static FileInfo getFile(String groupName,String remoteFileName){
    try {
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient获得TrackerServer信息
        TrackerServer trackerServer =trackerClient.getConnection();
        //通过TrackerServer获取StorageClient对象
        StorageClient storageClient = new StorageClient(trackerServer,null);
        //获取文件信息
        return storageClient.get_file_info(groupName,remoteFileName);
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

(4)文件下载

在类中添加如下方法实现文件下载：

```
/***
 * 文件下载
 * @param groupName:组名
 * @param remoteFileName：文件存储完整名
 * @return
 */
public static InputStream downFile(String groupName,String remoteFileName){
    try {
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient对象创建TrackerServer
        TrackerServer trackerServer = trackerClient.getConnection();
        //通过TrackerServer创建StorageClient
        StorageClient storageClient = new StorageClient(trackerServer,null);
        //通过StorageClient下载文件
        byte[] fileByte = storageClient.download_file(groupName,
remoteFileName);
        //将字节数组转换成字节输入流
        return new ByteArrayInputStream(fileByte);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

(5)文件删除实现

```
/***
 * 文件删除实现
 * @param groupName:组名
 * @param remoteFileName：文件存储完整名
 */
public static void deleteFile(String groupName,String remoteFileName){
    try {
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient获取TrackerServer对象
        TrackerServer trackerServer = trackerClient.getConnection();
        //通过TrackerServer创建StorageClient
        StorageClient storageClient = new StorageClient(trackerServer,null);
        //通过StorageClient删除文件
        storageClient.delete_file(groupName,remoteFileName);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(6)获取组信息

```
/***
 * 获取组信息
 * @param groupName :组名
 */
public static StorageServer getStorages(String groupName){
    try {
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient获取TrackerServer对象
        TrackerServer trackerServer = trackerClient.getConnection();
        //通过trackerClient获取Storage组信息
        return trackerClient.getStoreStorage(trackerServer,groupName);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

(7)根据文件组名和文件存储路径获取Storage服务的IP、端口信息

```
/***
 * 根据文件组名和文件存储路径获取Storage服务的IP、端口信息
 * @param groupName :组名
 * @param remoteFileName : 文件存储完整名
 */
public static ServerInfo[] getServerInfo(String groupName, String
remoteFileName){
    try {
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient获取TrackerServer对象
        TrackerServer trackerServer = trackerClient.getConnection();
        //获取服务信息
        return
trackerClient.getFetchStorages(trackerServer,groupName,remoteFileName);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

(8)获取Tracker服务地址

```
/***
 * 获取Tracker服务地址
 */
public static String getTrackerUrl(){
    try {
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient获取TrackerServer对象
        TrackerServer trackerServer = trackerClient.getConnection();
        //获取Tracker地址
```

```
        return "http://"+trackerServer.getInetSocketAddress().getHostString()+":"+ClientGlobal.getG_tracker_http_port();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

(9)优化

我们可以发现，上面所有方法中都会涉及到获取TrackerServer或者StorageClient，我们可以把它们单独抽取出去，分别在类中添加如下2个方法：

```
/***
 * 获取TrackerServer
 */
public static TrackerServer getTrackerServer() throws Exception{
    //创建TrackerClient对象
    TrackerClient trackerClient = new TrackerClient();
    //通过TrackerClient获取TrackerServer对象
    TrackerServer trackerServer = trackerClient.getConnection();
    return trackerServer;
}

/***
 * 获取StorageClient
 * @return
 * @throws Exception
 */
public static StorageClient getStorageClient() throws Exception{
    //获取TrackerServer
    TrackerServer trackerServer = getTrackerServer();
    //通过TrackerServer创建StorageClient
    StorageClient storageClient = new StorageClient(trackerServer,null);
    return storageClient;
}
```

修改其他方法，在需要使用TrackerServer和StorageClient的时候，直接调用上面的方法,完整代码如下：

```
public class FastDFSClient {

    /***
     * 初始化tracker信息
     */
    static {
        try {
            //获取tracker的配置文件fdfs_client.conf的位置
            String filePath = new ClassPathResource("fdfs_client.conf").getPath();
            //加载tracker配置信息
            ClientGlobal.init(filePath);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    /****
     * 文件上传
     * @param file : 要上传的文件信息封装->FastDFSFile
     * @return String[]
     *          1:文件上传所存储的组名
     *          2:文件存储路径
     */
    public static String[] upload(FastDFSFile file){
        //获取文件作者
        NameValuePair[] meta_list = new NameValuePair[1];
        meta_list[0] =new NameValuePair(file.getAuthor());

        /***
         * 文件上传后的返回值
         * uploadResults[0]:文件上传所存储的组名，例如:group1
         * uploadResults[1]:文件存储路径,例如:
M00/00/00/wKjThF0DBzaAP23MAAXz2mMp9oM26.jpeg
         */
        String[] uploadResults = null;
        try {
            //获取StorageClient对象
            StorageClient storageClient = getStorageClient();
            //执行文件上传
            uploadResults = storageClient.upload_file(file.getContent(),
file.getExt(), meta_list);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return uploadResults;
    }


    /***
     * 获取文件信息
     * @param groupName:组名
     * @param remoteFileName：文件存储完整名
     */
    public static FileInfo getFile(String groupName,String remoteFileName)
{
        try {
            //获取StorageClient对象
            StorageClient storageClient = getStorageClient();
            //获取文件信息
            return storageClient.get_file_info(groupName,remoteFileName);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /***
     * 文件下载
     * @param groupName:组名
     * @param remoteFileName：文件存储完整名
     * @return
     */
```

```
    public static InputStream downFile(String groupName,String remoteFileName){
        try {
            //获取StorageClient
            StorageClient storageClient = getStorageClient();
            //通过StorageClient下载文件
            byte[] fileByte = storageClient.download_file(groupName, remoteFileName);
            //将字节数组转换成字节输入流
            return new ByteArrayInputStream(fileByte);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /***
     * 文件删除实现
     * @param groupName:组名
     * @param remoteFileName：文件存储完整名
     */
    public static void deleteFile(String groupName,String remoteFileName){
        try {
            //获取StorageClient
            StorageClient storageClient = getStorageClient();
            //通过StorageClient删除文件
            storageClient.delete_file(groupName,remoteFileName);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    /***
     * 获取组信息
     * @param groupName :组名
     */
    public static StorageServer getStorages(String groupName){
        try {
            //创建TrackerClient对象
            TrackerClient trackerClient = new TrackerClient();
            //通过TrackerClient获取TrackerServer对象
            TrackerServer trackerServer = trackerClient.getConnection();
            //通过trackerClient获取Storage组信息
            return trackerClient.getStoreStorage(trackerServer,groupName);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /***
     * 根据文件组名和文件存储路径获取Storage服务的IP、端口信息
     * @param groupName :组名
     * @param remoteFileName ： 文件存储完整名
     */
    public static ServerInfo[] getServerInfo(String groupName, String remoteFileName){
```

```
        try {
            //创建TrackerClient对象
            TrackerClient trackerClient = new TrackerClient();
            //通过TrackerClient获取TrackerServer对象
            TrackerServer trackerServer = trackerClient.getConnection();
            //获取服务信息
            return
trackerClient.getFetchStorages(trackerServer,groupName,remoteFileName);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /***
     * 获取Tracker服务地址
     */
    public static String getTrackerUrl(){
        try {
            //创建TrackerClient对象
            TrackerClient trackerClient = new TrackerClient();
            //通过TrackerClient获取TrackerServer对象
            TrackerServer trackerServer = trackerClient.getConnection();
            //获取Tracker地址
            return
"http://"+trackerServer.getInetSocketAddress().getHostString()+":"+ClientG
lobal.getG_tracker_http_port();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    /***
     * 获取TrackerServer
     */
    public static TrackerServer getTrackerServer() throws Exception{
        //创建TrackerClient对象
        TrackerClient trackerClient = new TrackerClient();
        //通过TrackerClient获取TrackerServer对象
        TrackerServer trackerServer = trackerClient.getConnection();
        return trackerServer;
    }

    /***
     * 获取StorageClient
     * @return
     * @throws Exception
     */
    public static StorageClient getStorageClient() throws Exception{
        //获取TrackerServer
        TrackerServer trackerServer = getTrackerServer();
        //通过TrackerServer创建StorageClient
        StorageClient storageClient = new
StorageClient(trackerServer,null);
        return storageClient;
    }
}
```

1.4.3 文件上传

创建一个FileController，在该控制器中实现文件上传操作，代码如下：

```
@RestController
@CrossOrigin
public class FileController {

    /***
     * 文件上传
     * @return
     */
    @PostMapping(value = "/upload")
    public String upload(@RequestParam("file")MultipartFile file) throws
Exception {
        //封装一个FastDFSFile
        FastDFSFile fastDFSFile = new FastDFSFile(
                file.getOriginalFilename(), //文件名字
                file.getBytes(),            //文件字节数组

 StringUtils.getFilenameExtension(file.getOriginalFilename()));//文件扩展名

        //文件上传
        String[] uploads = FastDFSClient.upload(fastDFSFile);
        //组装文件上传地址
        return "http://192.168.211.132:8080"+"/"+uploads[0]+"/"+uploads[1];
    }
}
```

1.5 Postman测试文件上传

上传失败时,在宿主机的nginx中添加client_max_body_size 200m;

步骤：

1、选择post请求方式，输入请求地址 http://file-changgou-java.itheima.net/upload

2、填写body

选择form-data 然后选择文件file 点击添加文件，最后发送即可。

访问 `http://192.168.211.132:8080/group1/M00/00/00/wKjThF3wnYOAcXqCAAdIacHC9G0166.jpg` 如下图

注意，这里每次访问的端口是8080端口，访问的端口其实是storage容器的nginx端口，如果想修改该端口可以直接进入到storage容器，然后修改即可。

```
docker exec -it storage  /bin/bash
vi /etc/nginx/conf/nginx.conf
```

```
server {
    listen       8080;
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        root   html;
        index  index.html index.htm;
    }

    location ~ /M00 {
                root /data/fast_data/data;
                ngx_fastdfs_module;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   html;
    }
```

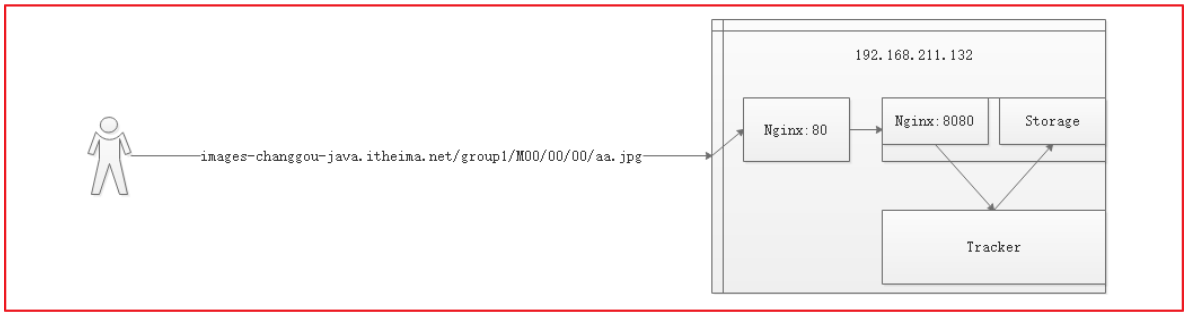
修改后重启storage即可根据自己修改的端口访问图片了。

FastDFS总结

```
文件上传：storageClient.upload_file("文件内容", "扩展名", "自定义属性");
获取文件信息:storageClient.get_file_info("storage组名","文件的存路径详细名字");
文件下载：storageClient.download_file("storage组名","文件的存路径详细名字");
文件删除：storageClient.delete_file("storage组名","文件的存路径详细名字");
```

1.6 图片域名配置

图片的域名为`images-changgou-java.itheima.net`，用户每次请求的时候，如果不想加端口号，则可以使用80端口的Nginx进行转发操作，如下图：



修改虚拟机的80端口Nginx，添加`images-changgou-java.itheima.net`域名的解析配置，配置如下：

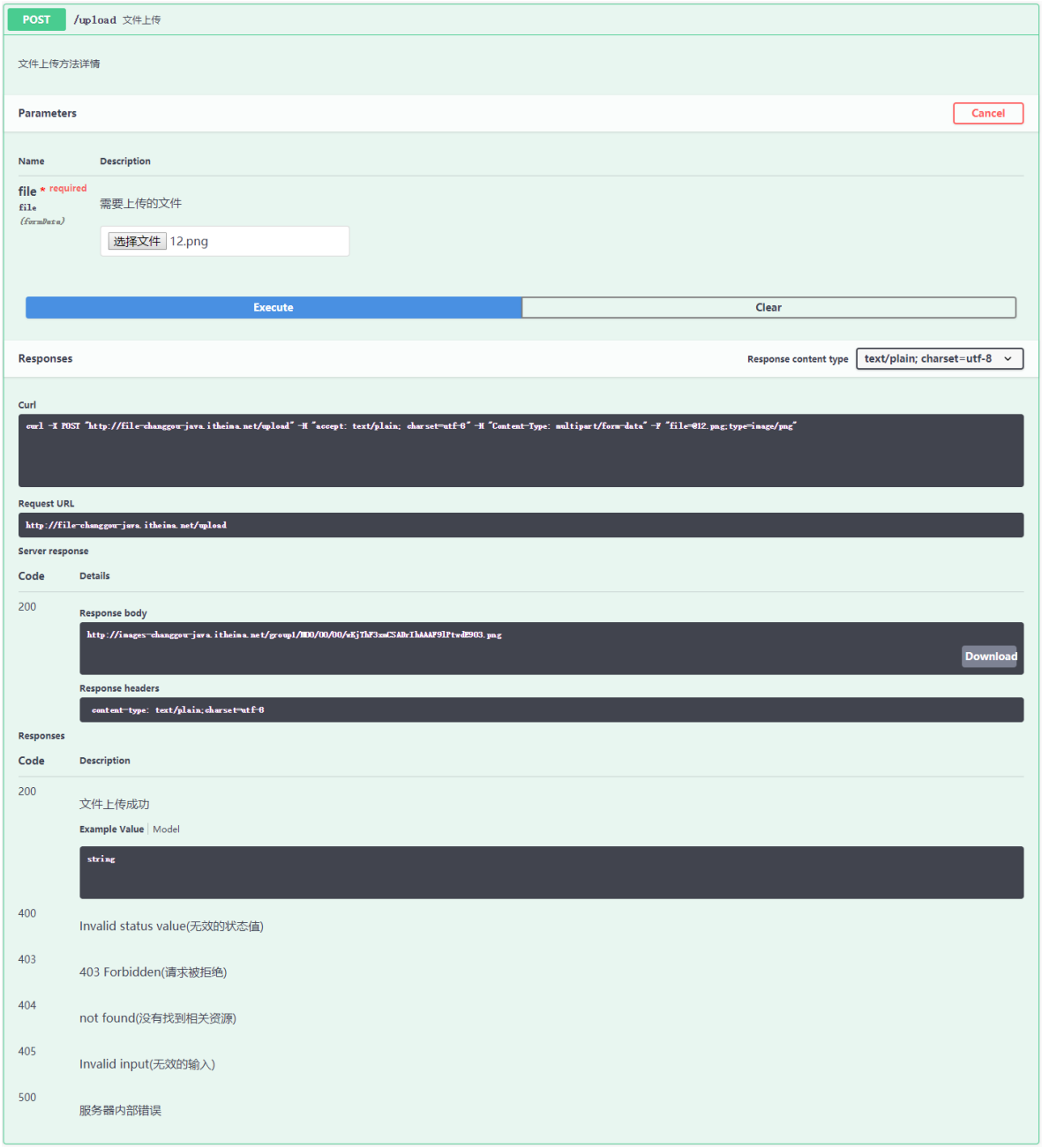
```
修改nginx.conf文件
cd /usr/local/openresty/nginx/conf

添加如下配置：
server {
    listen 80;
    server_name images-changgou-java.itheima.net;

    location / {
      proxy_pass http://127.0.0.1:8080;
    }
}

重启Nginx
/usr/local/openresty/nginx/sbin/nginx -s reload
```

访问图片：`<http://images-changgou-java.itheima.net/group1/M00/00/00/wKjThF3wnYOAcXqCAAdIacHC9G0166.jpg>`

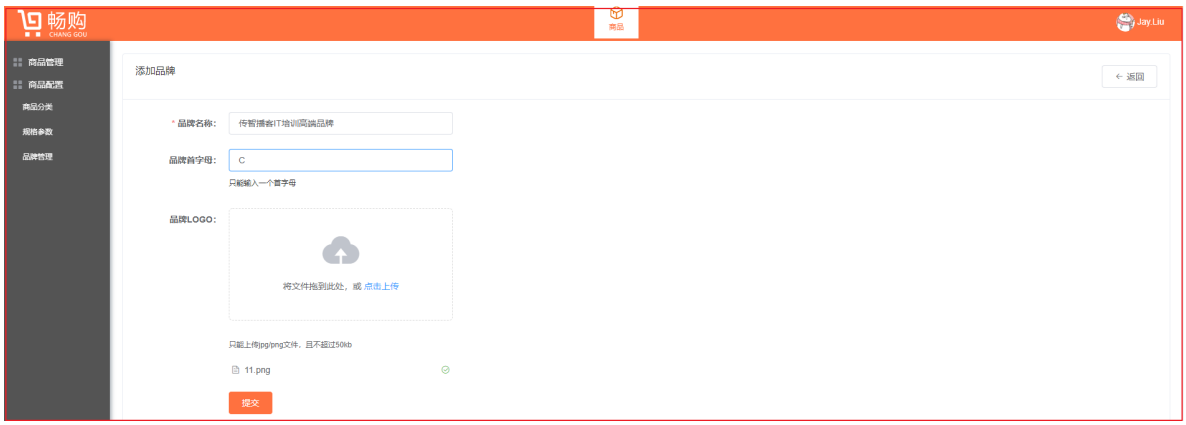
我们可以将文件上传中图片回显的地址换成域名，需改如下：

```java
/***
 * 文件上传
 * @return
 */
@PostMapping(value = "/upload")
public String upload(@RequestParam("file")MultipartFile file) throws Exception {
    //封装一个FastDFSFile
    FastDFSFile fastDFSFile = new FastDFSFile(
            file.getOriginalFilename(), //文件名字
            file.getBytes(),            //文件字节数组
            StringUtils.getFilenameExtension(file.getOriginalFilename()));//文件扩展名

    //文件上传
    String[] uploads = FastDFSClient.upload(fastDFSFile);
    //组装文件上传地址
    return "http://images-changgou-java.itheima.net/"+uploads[0]+"/"+uploads[1];
}
```

Swagger测试：



1.7 品牌文件上传测试



效果如下：

总结

2 相册管理(实战)

目标

- 实现相册的增删改查

路径

- 分析相册表结构
- 实现相册增删改查
- 使用Swagger测试相册功能

讲解

2.1 需求分析

相册是用于存储图片的管理单元，我们通常会将商品的图片先上传到相册中，在添加商品时可以直接在相册中选择，获取相册中的图片地址，保存到商品表中。

前端交互方式见管理后台的静态原型

2.2 表结构分析

tb_album 表（**相册表**）

字段名称	字段含义	字段类型	备注
id	编号	BIGINT(20)	主键
title	相册名称	VARCHAR(100)	
image	相册封面	VARCHAR(100)	
image_items	图片列表	TEXT	

表中image_items数据如下示例：

```
[
  {
    "url": "http://localhost:9101/img/1.jpg",
    "uid": 1548143143154,
    "status": "success"
  },
  {
    "url": "http://localhost:9101/img/7.jpg",
    "uid": 1548143143155,
    "status": "success"
  }
]
```

2.3 代码实现

2.3.1 Pojo

在changgou-service-goods-api工程中创建com.changgou.goods.pojo.Album，代码如下：

```
@Table(name="tb_album")
public class Album implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;//编号

    @Column(name = "title")
    private String title;//相册名称

    @Column(name = "image")
    private String image;//相册封面

    @Column(name = "image_items")
    private String imageItems;//图片列表

    //get...set...toString..
}
```

2.3.2 Dao

在changgou-service-goods中创建com.changgou.goods.dao.AlbumMapper接口，代码如下：

```
public interface AlbumMapper extends Mapper<Album> {
}
```

2.3.3 业务层

(1)业务层接口

在changgou-service-goods中创建com.changgou.goods.service.AlbumService接口，并添加常用方法，代码如下：

```
public interface AlbumService {

    /***
     * Album多条件分页查询
     * @param album
     * @param page
     * @param size
     * @return
     */
    PageInfo<Album> findPage(Album album, int page, int size);

    /***
     * Album分页查询
     * @param page
     * @param size
     * @return
     */
    PageInfo<Album> findPage(int page, int size);

    /***
     * Album多条件搜索方法
     * @param album
     * @return
     */
    List<Album> findList(Album album);

    /***
     * 删除Album
     * @param id
     */
    void delete(Long id);

    /***
     * 修改Album数据
     * @param album
     */
    void update(Album album);

    /***
     * 新增Album
     * @param album
     */
    void add(Album album);

    /**
     * 根据ID查询Album
     * @param id
     * @return
     */
     Album findById(Long id);

    /***
```

```
     * 查询所有Album
     * @return
     */
    List<Album> findAll();
}
```

(2)业务层实现类

在changgou-service-goods中创建com.changgou.goods.service.impl.AlbumServiceImpl，并实现接口方法，代码如下：

```
@Service
public class AlbumServiceImpl implements AlbumService {

    @Autowired
    private AlbumMapper albumMapper;


    /**
     * Album条件+分页查询
     * @param album 查询条件
     * @param page 页码
     * @param size 页大小
     * @return 分页结果
     */
    @Override
    public PageInfo<Album> findPage(Album album, int page, int size){
        //分页
        PageHelper.startPage(page,size);
        //搜索条件构建
        Example example = createExample(album);
        //执行搜索
        return new PageInfo<Album>(albumMapper.selectByExample(example));
    }

    /**
     * Album分页查询
     * @param page
     * @param size
     * @return
     */
    @Override
    public PageInfo<Album> findPage(int page, int size){
        //静态分页
        PageHelper.startPage(page,size);
        //分页查询
        return new PageInfo<Album>(albumMapper.selectAll());
    }

    /**
     * Album条件查询
     * @param album
     * @return
     */
    @Override
```

```
45      public List<Album> findList(Album album){
46          //构建查询条件
47          Example example = createExample(album);
48          //根据构建的条件查询数据
49          return albumMapper.selectByExample(example);
50      }
51
52
53      /**
54       * Album构建查询对象
55       * @param album
56       * @return
57       */
58      public Example createExample(Album album){
59          Example example=new Example(Album.class);
60          Example.Criteria criteria = example.createCriteria();
61          if(album!=null){
62              // 编号
63              if(!StringUtils.isEmpty(album.getId())){
64                      criteria.andEqualTo("id",album.getId());
65              }
66              // 相册名称
67              if(!StringUtils.isEmpty(album.getTitle())){
68                      criteria.andLike("title","%"+album.getTitle()+"%");
69              }
70              // 相册封面
71              if(!StringUtils.isEmpty(album.getImage())){
72                      criteria.andEqualTo("image",album.getImage());
73              }
74              // 图片列表
75              if(!StringUtils.isEmpty(album.getImageItems())){
76
    criteria.andEqualTo("imageItems",album.getImageItems());
77              }
78          }
79          return example;
80      }
81
82      /**
83       * 删除
84       * @param id
85       */
86      @Override
87      public void delete(Long id){
88          albumMapper.deleteByPrimaryKey(id);
89      }
90
91      /**
92       * 修改Album
93       * @param album
94       */
95      @Override
96      public void update(Album album){
97          albumMapper.updateByPrimaryKey(album);
98      }
99
100     /**
101      * 增加Album
```

```
     * @param album
     */
    @Override
    public void add(Album album){
        albumMapper.insert(album);
    }

    /**
     * 根据ID查询Album
     * @param id
     * @return
     */
    @Override
    public Album findById(Long id){
        return  albumMapper.selectByPrimaryKey(id);
    }

    /**
     * 查询Album全部数据
     * @return
     */
    @Override
    public List<Album> findAll() {
        return albumMapper.selectAll();
    }
}
```

2.3.4 控制层

在changgou-service-service工程中创建com.changgou.goods.controller.AlbumController，代码如下：

```
@RestController
@RequestMapping("/album")
@CrossOrigin
public class AlbumController {

    @Autowired
    private AlbumService albumService;

    /***
     * Album分页条件搜索实现
     * @param album
     * @param page
     * @param size
     * @return
     */
    @PostMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@RequestBody(required = false)  Album album, @PathVariable  int page, @PathVariable  int size){
        //执行搜索
        PageInfo<Album> pageInfo = albumService.findPage(album, page, size);
        return new Result(true,StatusCode.OK,"查询成功",pageInfo);
    }
```

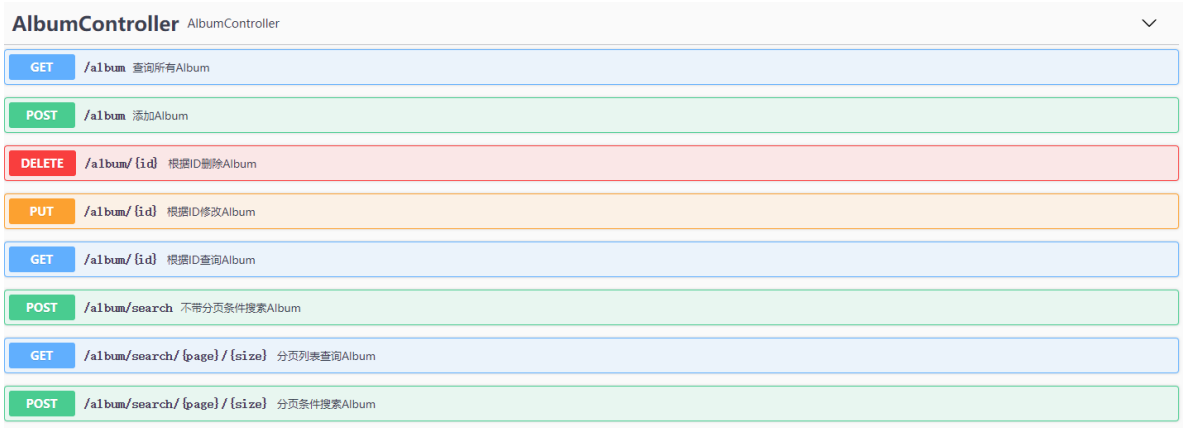
```
22
23      /***
24       * Album分页搜索实现
25       * @param page:当前页
26       * @param size:每页显示多少条
27       * @return
28       */
29      @GetMapping(value = "/search/{page}/{size}" )
30      public Result<PageInfo> findPage(@PathVariable  int page,
    @PathVariable  int size){
31          //分页查询
32          PageInfo<Album> pageInfo = albumService.findPage(page, size);
33          return new Result<PageInfo>(true,StatusCode.OK,"查询成
    功",pageInfo);
34      }
35
36      /***
37       * 多条件搜索品牌数据
38       * @param album
39       * @return
40       */
41      @PostMapping(value = "/search" )
42      public Result<List<Album>> findList(@RequestBody(required = false)
     Album album){
43          List<Album> list = albumService.findList(album);
44          return new Result<List<Album>>(true,StatusCode.OK,"查询成功",list);
45      }
46
47      /***
48       * 根据ID删除品牌数据
49       * @param id
50       * @return
51       */
52      @DeleteMapping(value = "/{id}" )
53      public Result delete(@PathVariable Long id){
54          albumService.delete(id);
55          return new Result(true,StatusCode.OK,"删除成功");
56      }
57
58      /***
59       * 修改Album数据
60       * @param album
61       * @param id
62       * @return
63       */
64      @PutMapping(value="/{id}")
65      public Result update(@RequestBody  Album album,@PathVariable Long id){
66          //设置主键值
67          album.setId(id);
68          //修改数据
69          albumService.update(album);
70          return new Result(true,StatusCode.OK,"修改成功");
71      }
72
73      /***
74       * 新增Album数据
75       * @param album
76       * @return
```

```
     */
    @PostMapping
    public Result add(@RequestBody   Album album){
        albumService.add(album);
        return new Result(true,StatusCode.OK,"添加成功");
    }

    /***
     * 根据ID查询Album数据
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Result<Album> findById(@PathVariable Long id){
        //根据ID查询
        Album album = albumService.findById(id);
        return new Result<Album>(true,StatusCode.OK,"查询成功",album);
    }

    /***
     * 查询Album全部数据
     * @return
     */
    @GetMapping
    public Result<Album> findAll(){
        List<Album> list = albumService.findAll();
        return new Result<Album>(true, StatusCode.OK,"查询成功",list) ;
    }
}
```

添加相册示例数据如下：

```
{
  "title": "北京冬天的雪景",
  "image": "http://localhost:9101/img/1.jpg",
  "imageItems": "[ {\"url\": \"http://localhost:9101/img/1.jpg\",\"uid\": 1548143143154,\"status\": \"success\"},{\"url\": \"http://localhost:9101/img/7.jpg\",\"uid\": 1548143143155,\"status\": \"success\"}]"
}
```

用Swagger测试上面的功能：



总结

3 规格/参数/模板

目标

- 实现模板参数的增删改查

路径

- 了解模板、规格、参数的作用和关系
- 模板、规格、参数表结构讲解
- 模板、规格、参数的增删改查

讲解

3.1 需求分析

规格参数模板是用于管理规格参数的单元。规格是例如颜色、手机运行内存等信息，参数是例如系统：安卓（Android）后置摄像头像素：2000万及以上 热点：快速充电等信息 。

前端交互方式见管理后台的静态原型

3.2 表结构分析

规格参数模板相关的表有3个

tb_template 表（模板表）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	INT	10	
name	模板名称	VARCHAR	华为手机	
spec_num	规格数量	INT		
para_num	参数数量	INT		

tb_spec 表（ 规格表）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	INT		
name	名称	VARCHAR		
options	规格选项	VARCHAR		
seq	排序	INT		
template_id	模板ID	INT	10	

tb_para 表（参数表）

字段名称	字段含义	字段类型	字段长度	备注
id	id	INT		
name	名称	VARCHAR		
options	选项	VARCHAR		
seq	排序	INT		
template_id	模板ID	INT	10	

模板与规格是一对多关系， 模板与参数是一对多关系

3.3 模板管理

3.3.1 Pojo

在changgou-service-goods-api工程中创建com.changgou.goods.pojo.Template，代码如下：

```
@Table(name="tb_template")
public class Template implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;//ID

    @Column(name = "name")
    private String name;//模板名称

    @Column(name = "spec_num")
    private Integer specNum;//规格数量

    @Column(name = "para_num")
    private Integer paraNum;//参数数量
    //..get..set..toString
}
```

3.3.2 Dao

在changgou-service-goods中创建com.changgou.goods.dao.TemplateMapper,代码如下:

```
public interface TemplateMapper extends Mapper<Template> {
}
```

3.3.3 业务层

(1)业务层接口

在changgou-service-goods中创建com.changgou.goods.service.TemplateService接口，并添加相关方法，代码如下:

```
public interface TemplateService {

    /***
     * Template多条件分页查询
     * @param template
     * @param page
     * @param size
     * @return
     */
    PageInfo<Template> findPage(Template template, int page, int size);

    /***
     * Template分页查询
     * @param page
     * @param size
     * @return
     */
    PageInfo<Template> findPage(int page, int size);

    /***
     * Template多条件搜索方法
     * @param template
     * @return
     */
    List<Template> findList(Template template);

    /***
     * 删除Template
     * @param id
     */
    void delete(Integer id);

    /***
     * 修改Template数据
     * @param template
     */
    void update(Template template);

    /***
```

```
     * 新增Template
     * @param template
     */
    void add(Template template);

    /**
     * 根据ID查询Template
     * @param id
     * @return
     */
     Template findById(Integer id);

    /***
     * 查询所有Template
     * @return
     */
    List<Template> findAll();
}
```

(2)业务层接口实现类

在changgou-service-goods中创建com.changgou.goods.service.impl.TemplateServiceImpl实现类，并实现对应的方法，代码如下：

```
@Service
public class TemplateServiceImpl implements TemplateService {

    @Autowired
    private TemplateMapper templateMapper;


    /**
     * Template条件+分页查询
     * @param template 查询条件
     * @param page 页码
     * @param size 页大小
     * @return 分页结果
     */
    @Override
    public PageInfo<Template> findPage(Template template, int page, int size){
        //分页
        PageHelper.startPage(page,size);
        //搜索条件构建
        Example example = createExample(template);
        //执行搜索
        return new PageInfo<Template>(templateMapper.selectByExample(example));
    }

    /**
     * Template分页查询
     * @param page
     * @param size
     * @return
```

```
     */
    @Override
    public PageInfo<Template> findPage(int page, int size){
        //静态分页
        PageHelper.startPage(page,size);
        //分页查询
        return new PageInfo<Template>(templateMapper.selectAll());
    }

    /**
     * Template条件查询
     * @param template
     * @return
     */
    @Override
    public List<Template> findList(Template template){
        //构建查询条件
        Example example = createExample(template);
        //根据构建的条件查询数据
        return templateMapper.selectByExample(example);
    }


    /**
     * Template构建查询对象
     * @param template
     * @return
     */
    public Example createExample(Template template){
        Example example=new Example(Template.class);
        Example.Criteria criteria = example.createCriteria();
        if(template!=null){
            // ID
            if(!StringUtils.isEmpty(template.getId())){
                    criteria.andEqualTo("id",template.getId());
            }
            // 模板名称
            if(!StringUtils.isEmpty(template.getName())){
                    criteria.andLike("name","%"+template.getName()+"%");
            }
            // 规格数量
            if(!StringUtils.isEmpty(template.getSpecNum())){
                    criteria.andEqualTo("specNum",template.getSpecNum());
            }
            // 参数数量
            if(!StringUtils.isEmpty(template.getParaNum())){
                    criteria.andEqualTo("paraNum",template.getParaNum());
            }
        }
        return example;
    }

    /**
     * 删除
     * @param id
     */
    @Override
    public void delete(Integer id){
```

```
        templateMapper.deleteByPrimaryKey(id);
    }

    /**
     * 修改Template
     * @param template
     */
    @Override
    public void update(Template template){
        templateMapper.updateByPrimaryKey(template);
    }

    /**
     * 增加Template
     * @param template
     */
    @Override
    public void add(Template template){
        templateMapper.insert(template);
    }

    /**
     * 根据ID查询Template
     * @param id
     * @return
     */
    @Override
    public Template findById(Integer id){
        return  templateMapper.selectByPrimaryKey(id);
    }

    /**
     * 查询Template全部数据
     * @return
     */
    @Override
    public List<Template> findAll() {
        return templateMapper.selectAll();
    }
}
```

3.3.4 控制层

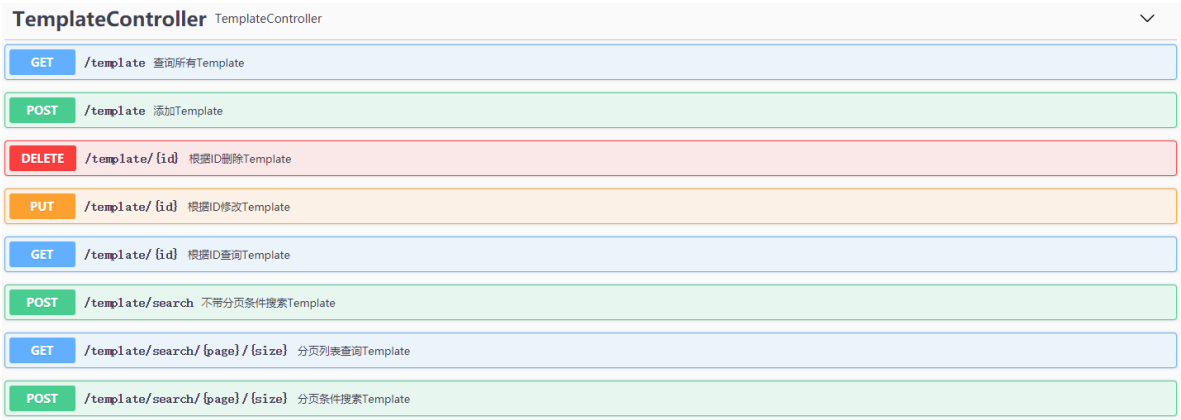
在changgou-service-goods中创建com.changgou.goods.controller.TemplateController，代码如下：

```
@RestController
@RequestMapping("/template")
@CrossOrigin
public class TemplateController {

    @Autowired
    private TemplateService templateService;

    /***
     * Template分页条件搜索实现
```

```
     * @param template
     * @param page
     * @param size
     * @return
     */
    @PostMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@RequestBody(required = false) Template template, @PathVariable  int page, @PathVariable  int size){
        //执行搜索
        PageInfo<Template> pageInfo = templateService.findPage(template, page, size);
        return new Result(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * Template分页搜索实现
     * @param page:当前页
     * @param size:每页显示多少条
     * @return
     */
    @GetMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@PathVariable  int page, @PathVariable  int size){
        //分页查询
        PageInfo<Template> pageInfo = templateService.findPage(page, size);
        return new Result<PageInfo>(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * 多条件搜索品牌数据
     * @param template
     * @return
     */
    @PostMapping(value = "/search" )
    public Result<List<Template>> findList(@RequestBody(required = false) Template template){
        List<Template> list = templateService.findList(template);
        return new Result<List<Template>>(true,StatusCode.OK,"查询成功",list);
    }

    /***
     * 根据ID删除品牌数据
     * @param id
     * @return
     */
    @DeleteMapping(value = "/{id}" )
    public Result delete(@PathVariable Integer id){
        templateService.delete(id);
        return new Result(true,StatusCode.OK,"删除成功");
    }

    /***
     * 修改Template数据
     * @param template
     * @param id
```

```
     * @return
     */
    @PutMapping(value="/{id}")
    public Result update(@RequestBody  Template template,@PathVariable Integer id){
        //设置主键值
        template.setId(id);
        //修改数据
        templateService.update(template);
        return new Result(true,StatusCode.OK,"修改成功");
    }

    /***
     * 新增Template数据
     * @param template
     * @return
     */
    @PostMapping
    public Result add(@RequestBody   Template template){
        templateService.add(template);
        return new Result(true,StatusCode.OK,"添加成功");
    }

    /***
     * 根据ID查询Template数据
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Result<Template> findById(@PathVariable Integer id){
        //根据ID查询
        Template template = templateService.findById(id);
        return new Result<Template>(true,StatusCode.OK,"查询成功",template);
    }

    /***
     * 查询Template全部数据
     * @return
     */
    @GetMapping
    public Result<Template> findAll(){
        List<Template> list = templateService.findAll();
        return new Result<Template>(true, StatusCode.OK,"查询成功",list) ;
    }
}
```

Swagger测试：



3.4 规格管理

3.4.1 Pojo

在changgou-service-goods-api中创建com.changgou.goods.pojo.Spec,代码如下：

```
@Table(name="tb_spec")
public class Spec implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;//ID

    @Column(name = "name")
    private String name;//名称

    @Column(name = "options")
    private String options;//规格选项

    @Column(name = "seq")
    private Integer seq;//排序

    @Column(name = "template_id")
    private Integer templateId;//模板ID

    //get..set..toString
}
```

3.4.2 Dao

在changgou-service-goods中创建com.changgou.goods.dao.SpecMapper，代码如下：

```
public interface SpecMapper extends Mapper<Spec> {
}
```

3.4.3 业务层

(1)业务层接口

在changgou-service-goods中创建com.changgou.goods.service.SpecService接口，并实现对应的方法，代码如下：

```
public interface SpecService {

    /***
     * Spec多条件分页查询
     * @param spec
     * @param page
     * @param size
     * @return
     */
    PageInfo<Spec> findPage(Spec spec, int page, int size);

    /***
     * Spec分页查询
     * @param page
     * @param size
     * @return
     */
    PageInfo<Spec> findPage(int page, int size);

    /***
     * Spec多条件搜索方法
     * @param spec
     * @return
     */
    List<Spec> findList(Spec spec);

    /***
     * 删除Spec
     * @param id
     */
    void delete(Integer id);

    /***
     * 修改Spec数据
     * @param spec
     */
    void update(Spec spec);

    /***
     * 新增Spec
     * @param spec
     */
    void add(Spec spec);

    /**
     * 根据ID查询Spec
     * @param id
     * @return
     */
     Spec findById(Integer id);

    /***
```

```
     * 查询所有Spec
     * @return
     */
    List<Spec> findAll();
}
```

(2)业务层实现类

在changgou-service-goods中创建com.changgou.goods.service.impl.SpecServiceImpl,代码如下:

```
@Service
public class SpecServiceImpl implements SpecService {

    @Autowired
    private SpecMapper specMapper;

    @Autowired
    private TemplateMapper templateMapper;

    /**
     * Spec条件+分页查询
     * @param spec 查询条件
     * @param page 页码
     * @param size 页大小
     * @return 分页结果
     */
    @Override
    public PageInfo<Spec> findPage(Spec spec, int page, int size){
        //分页
        PageHelper.startPage(page,size);
        //搜索条件构建
        Example example = createExample(spec);
        //执行搜索
        return new PageInfo<Spec>(specMapper.selectByExample(example));
    }

    /**
     * Spec分页查询
     * @param page
     * @param size
     * @return
     */
    @Override
    public PageInfo<Spec> findPage(int page, int size){
        //静态分页
        PageHelper.startPage(page,size);
        //分页查询
        return new PageInfo<Spec>(specMapper.selectAll());
    }

    /**
     * Spec条件查询
     * @param spec
     * @return
     */
    @Override
```

```
    public List<Spec> findList(Spec spec){
        //构建查询条件
        Example example = createExample(spec);
        //根据构建的条件查询数据
        return specMapper.selectByExample(example);
    }


    /**
     * Spec构建查询对象
     * @param spec
     * @return
     */
    public Example createExample(Spec spec){
        Example example=new Example(Spec.class);
        Example.Criteria criteria = example.createCriteria();
        if(spec!=null){
            // ID
            if(!StringUtils.isEmpty(spec.getId())){
                    criteria.andEqualTo("id",spec.getId());
            }
            // 名称
            if(!StringUtils.isEmpty(spec.getName())){
                    criteria.andLike("name","%"+spec.getName()+"%");
            }
            // 规格选项
            if(!StringUtils.isEmpty(spec.getOptions())){
                    criteria.andEqualTo("options",spec.getOptions());
            }
            // 排序
            if(!StringUtils.isEmpty(spec.getSeq())){
                    criteria.andEqualTo("seq",spec.getSeq());
            }
            // 模板ID
            if(!StringUtils.isEmpty(spec.getTemplateId())){

criteria.andEqualTo("templateId",spec.getTemplateId());
            }
        }
        return example;
    }

    /**
     * 删除
     * @param id
     */
    @Override
    public void delete(Integer id){
        //查询模板
        Spec spec = specMapper.selectByPrimaryKey(id);
        //变更模板数量
        updateSpecNum(spec,-1);

        //删除指定规格
        specMapper.deleteByPrimaryKey(id);
    }

    /**
```

```
     * 修改Spec
     * @param spec
     */
    @Override
    public void update(Spec spec){
        specMapper.updateByPrimaryKey(spec);
    }

    /**
     * 增加Spec
     * @param spec
     */
    @Override
    public void add(Spec spec){
        specMapper.insert(spec);
        //变更模板数量
        updateSpecNum(spec,1);
    }

    /**
     * 根据ID查询Spec
     * @param id
     * @return
     */
    @Override
    public Spec findById(Integer id){
        return  specMapper.selectByPrimaryKey(id);
    }

    /**
     * 查询Spec全部数据
     * @return
     */
    @Override
    public List<Spec> findAll() {
        return specMapper.selectAll();
    }


    /**
     * 修改模板统计数据
     * @param spec:操作的模板
     * @param count:变更的数量
     */
    public void updateSpecNum(Spec spec,int count){
        //修改模板数量统计
        Template template =
templateMapper.selectByPrimaryKey(spec.getTemplateId());
        template.setSpecNum(template.getSpecNum()+count);
        templateMapper.updateByPrimaryKeySelective(template);
    }
}
```

这里注意，每次执行增加和删除的时候，需要调用模板，修改统计数据，另外大家思考下，如果是修改呢，是否会对模板统计数据造成变更呢?

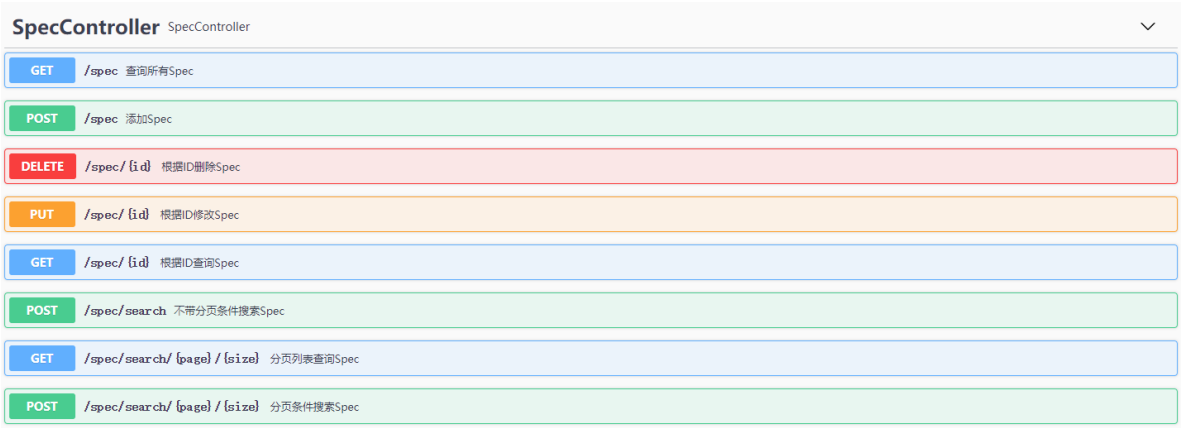
3.4.4 控制层

在changgou-service-goods中创建com.changgou.goods.controller.SpecController,代码如下:

```
@RestController
@RequestMapping("/spec")
@CrossOrigin
public class SpecController {

    @Autowired
    private SpecService specService;

    /***
     * Spec分页条件搜索实现
     * @param spec
     * @param page
     * @param size
     * @return
     */
    @PostMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@RequestBody(required = false)  Spec spec, @PathVariable  int page, @PathVariable  int size){
        //执行搜索
        PageInfo<Spec> pageInfo = specService.findPage(spec, page, size);
        return new Result(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * Spec分页搜索实现
     * @param page:当前页
     * @param size:每页显示多少条
     * @return
     */
    @GetMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@PathVariable  int page, @PathVariable  int size){
        //分页查询
        PageInfo<Spec> pageInfo = specService.findPage(page, size);
        return new Result<PageInfo>(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * 多条件搜索品牌数据
     * @param spec
     * @return
     */
    @PostMapping(value = "/search" )
    public Result<List<Spec>> findList(@RequestBody(required = false) Spec spec){
        List<Spec> list = specService.findList(spec);
        return new Result<List<Spec>>(true,StatusCode.OK,"查询成功",list);
    }

    /***
     * 根据ID删除品牌数据
     * @param id
```

```
     * @return
     */
    @DeleteMapping(value = "/{id}" )
    public Result delete(@PathVariable Integer id){
        specService.delete(id);
        return new Result(true,StatusCode.OK,"删除成功");
    }

    /***
     * 修改Spec数据
     * @param spec
     * @param id
     * @return
     */
    @PutMapping(value="/{id}")
    public Result update(@RequestBody  Spec spec,@PathVariable Integer id)
{
        //设置主键值
        spec.setId(id);
        //修改数据
        specService.update(spec);
        return new Result(true,StatusCode.OK,"修改成功");
    }

    /***
     * 新增Spec数据
     * @param spec
     * @return
     */
    @PostMapping
    public Result add(@RequestBody   Spec spec){
        specService.add(spec);
        return new Result(true,StatusCode.OK,"添加成功");
    }

    /***
     * 根据ID查询Spec数据
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Result<Spec> findById(@PathVariable Integer id){
        //根据ID查询
        Spec spec = specService.findById(id);
        return new Result<Spec>(true,StatusCode.OK,"查询成功",spec);
    }

    /***
     * 查询Spec全部数据
     * @return
     */
    @GetMapping
    public Result<Spec> findAll(){
        List<Spec> list = specService.findAll();
        return new Result<Spec>(true, StatusCode.OK,"查询成功",list) ;
    }
}
```

Swagger测试：



3.5 参数管理

3.5.1 Pojo

在changgou-service-goods-api中创建com.changgou.goods.pojo.Para，代码如下：

```
@Table(name="tb_para")
public class Para implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;//id

    @Column(name = "name")
    private String name;//名称

    @Column(name = "options")
    private String options;//选项

    @Column(name = "seq")
    private Integer seq;//排序

    @Column(name = "template_id")
    private Integer templateId;//模板ID
    //get..set..toString
}
```

3.5.2 Dao

在changgou-service-goods中创建com.changgou.goods.dao.ParaMapper,代码如下：

```
public interface ParaMapper extends Mapper<Para> {
}
```

3.5.3 业务层

(1)业务层接口

在changgou-service-goods中创建com.changgou.goods.service.ParaService接口，并添加常用方法，代码如下：

```
public interface ParaService {

    /***
     * Para多条件分页查询
     * @param para
     * @param page
     * @param size
     * @return
     */
    PageInfo<Para> findPage(Para para, int page, int size);

    /***
     * Para分页查询
     * @param page
     * @param size
     * @return
     */
    PageInfo<Para> findPage(int page, int size);

    /***
     * Para多条件搜索方法
     * @param para
     * @return
     */
    List<Para> findList(Para para);

    /***
     * 删除Para
     * @param id
     */
    void delete(Integer id);

    /***
     * 修改Para数据
     * @param para
     */
    void update(Para para);

    /***
     * 新增Para
     * @param para
     */
    void add(Para para);

    /**
     * 根据ID查询Para
     * @param id
     * @return
     */
```

```
    Para findById(Integer id);

    /***
     * 查询所有Para
     * @return
     */
    List<Para> findAll();
}
```

(2)业务层接口实现类

在changgou-service-goods中创建com.changgou.goods.service.impl.ParaServiceImpl接口实现类，代码如下：

```
@Service
public class ParaServiceImpl implements ParaService {

    @Autowired
    private ParaMapper paraMapper;

    @Autowired
    private TemplateMapper templateMapper;

    /**
     * Para条件+分页查询
     * @param para 查询条件
     * @param page 页码
     * @param size 页大小
     * @return 分页结果
     */
    @Override
    public PageInfo<Para> findPage(Para para, int page, int size){
        //分页
        PageHelper.startPage(page,size);
        //搜索条件构建
        Example example = createExample(para);
        //执行搜索
        return new PageInfo<Para>(paraMapper.selectByExample(example));
    }

    /**
     * Para分页查询
     * @param page
     * @param size
     * @return
     */
    @Override
    public PageInfo<Para> findPage(int page, int size){
        //静态分页
        PageHelper.startPage(page,size);
        //分页查询
        return new PageInfo<Para>(paraMapper.selectAll());
    }

    /**
```

```
     * Para条件查询
     * @param para
     * @return
     */
    @Override
    public List<Para> findList(Para para){
        //构建查询条件
        Example example = createExample(para);
        //根据构建的条件查询数据
        return paraMapper.selectByExample(example);
    }


    /**
     * Para构建查询对象
     * @param para
     * @return
     */
    public Example createExample(Para para){
        Example example=new Example(Para.class);
        Example.Criteria criteria = example.createCriteria();
        if(para!=null){
            // id
            if(!StringUtils.isEmpty(para.getId())){
                    criteria.andEqualTo("id",para.getId());
            }
            // 名称
            if(!StringUtils.isEmpty(para.getName())){
                    criteria.andLike("name","%"+para.getName()+"%");
            }
            // 选项
            if(!StringUtils.isEmpty(para.getOptions())){
                    criteria.andEqualTo("options",para.getOptions());
            }
            // 排序
            if(!StringUtils.isEmpty(para.getSeq())){
                    criteria.andEqualTo("seq",para.getSeq());
            }
            // 模板ID
            if(!StringUtils.isEmpty(para.getTemplateId())){

criteria.andEqualTo("templateId",para.getTemplateId());
            }
        }
        return example;
    }

    /**
     * 删除
     * @param id
     */
    @Override
    public void delete(Integer id){
        //根据ID查询
        Para para = paraMapper.selectByPrimaryKey(id);
        //修改模板统计数据
        updateParaNum(para,-1);

```

```
        paraMapper.deleteByPrimaryKey(id);
    }

    /**
     * 修改Para
     * @param para
     */
    @Override
    public void update(Para para){
        paraMapper.updateByPrimaryKey(para);
    }

    /**
     * 增加Para
     * @param para
     */
    @Override
    public void add(Para para){
        paraMapper.insert(para);

        //修改模板统计数据
        updateParaNum(para,1);
    }

    /**
     * 根据ID查询Para
     * @param id
     * @return
     */
    @Override
    public Para findById(Integer id){
        return  paraMapper.selectByPrimaryKey(id);
    }

    /**
     * 查询Para全部数据
     * @return
     */
    @Override
    public List<Para> findAll() {
        return paraMapper.selectAll();
    }

    /**
     * 修改模板统计数据
     * @param para:操作的参数
     * @param count:变更的数量
     */
    public void updateParaNum(Para para, int count){
        //修改模板数量统计
        Template template =
templateMapper.selectByPrimaryKey(para.getTemplateId());
        template.setParaNum(template.getParaNum()+count);
        templateMapper.updateByPrimaryKeySelective(template);
    }
}
```

3.5.4 控制层

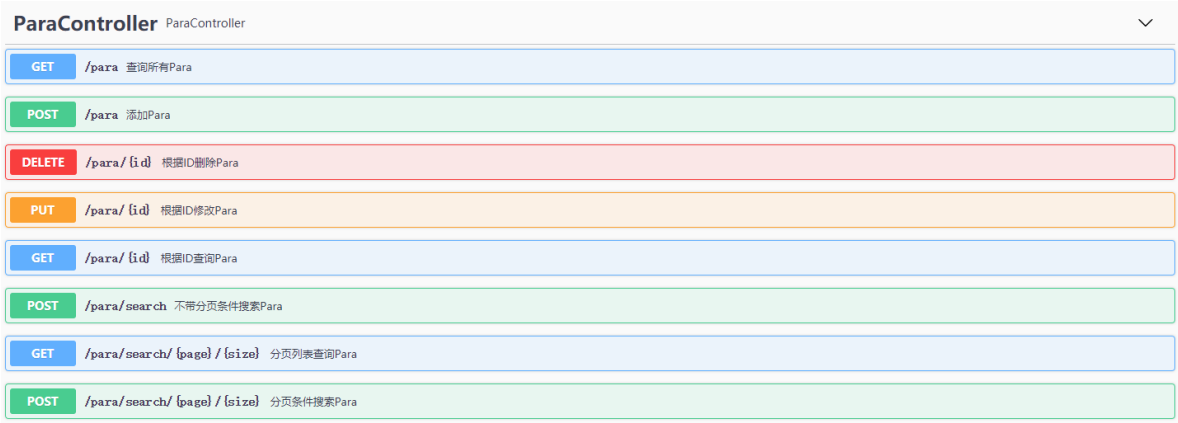
在changgou-service-goods下创建com.changgou.goods.controller.ParaController,代码如下:

```
@RestController
@RequestMapping("/para")
@CrossOrigin
public class ParaController {

    @Autowired
    private ParaService paraService;

    /***
     * Para分页条件搜索实现
     * @param para
     * @param page
     * @param size
     * @return
     */
    @PostMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@RequestBody(required = false)  Para para, @PathVariable  int page, @PathVariable  int size){
        //执行搜索
        PageInfo<Para> pageInfo = paraService.findPage(para, page, size);
        return new Result(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * Para分页搜索实现
     * @param page:当前页
     * @param size:每页显示多少条
     * @return
     */
    @GetMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@PathVariable  int page, @PathVariable  int size){
        //分页查询
        PageInfo<Para> pageInfo = paraService.findPage(page, size);
        return new Result<PageInfo>(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * 多条件搜索品牌数据
     * @param para
     * @return
     */
    @PostMapping(value = "/search" )
    public Result<List<Para>> findList(@RequestBody(required = false) Para para){
        List<Para> list = paraService.findList(para);
        return new Result<List<Para>>(true,StatusCode.OK,"查询成功",list);
    }

    /***
```

```
     * 根据ID删除品牌数据
     * @param id
     * @return
     */
    @DeleteMapping(value = "/{id}" )
    public Result delete(@PathVariable Integer id){
        paraService.delete(id);
        return new Result(true,StatusCode.OK,"删除成功");
    }

    /***
     * 修改Para数据
     * @param para
     * @param id
     * @return
     */
    @PutMapping(value="/{id}")
    public Result update(@RequestBody  Para para,@PathVariable Integer id)
{
        //设置主键值
        para.setId(id);
        //修改数据
        paraService.update(para);
        return new Result(true,StatusCode.OK,"修改成功");
    }

    /***
     * 新增Para数据
     * @param para
     * @return
     */
    @PostMapping
    public Result add(@RequestBody   Para para){
        paraService.add(para);
        return new Result(true,StatusCode.OK,"添加成功");
    }

    /***
     * 根据ID查询Para数据
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Result<Para> findById(@PathVariable Integer id){
        //根据ID查询
        Para para = paraService.findById(id);
        return new Result<Para>(true,StatusCode.OK,"查询成功",para);
    }

    /***
     * 查询Para全部数据
     * @return
     */
    @GetMapping
    public Result<Para> findAll(){
        List<Para> list = paraService.findAll();
        return new Result<Para>(true, StatusCode.OK,"查询成功",list) ;
    }
```

```
105 }
```

Swagger测试：



总结

4 商品分类(实战)

目标

- 商品分类管理

路径

- 商品分类介绍
- 商品分类表结构讲解
- 商品分类增删改查

讲解

4.1 需求分析

商品分类一共分三级管理，主要作用是在网站首页中显示商品导航，以及在管理后台管理商品时使用。

前端交互方式见管理后台的静态原型

4.2 表结构分析

tb_category 表 （**商品分类**）

字段名称	字段含义	字段类型	字段长度	备注
id	分类ID	INT		
name	分类名称	VARCHAR		
goods_num	商品数量	INT		
is_show	是否显示	CHAR		0 不显示 1显示
is_menu	是否导航	CHAR		0 不时导航 1 为导航
seq	排序	INT		
parent_id	上级ID	INT		
template_id	模板ID	INT		

商品分类与模板是多对一关系

4.3 实现

4.3.1 Pojo

在changgou-service-goods-api中创建com.changgou.goods.pojo.Category,代码如下:

```
@Table(name="tb_category")
public class Category implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;//分类ID

    @Column(name = "name")
    private String name;//分类名称

    @Column(name = "goods_num")
    private Integer goodsNum;//商品数量

    @Column(name = "is_show")
    private String isShow;//是否显示

    @Column(name = "is_menu")
    private String isMenu;//是否导航

    @Column(name = "seq")
    private Integer seq;//排序

    @Column(name = "parent_id")
    private Integer parentId;//上级ID

    @Column(name = "template_id")
    private Integer templateId;//模板ID
    //..set..get..toString
}
```

4.3.2 Dao

在changgou-servicegoods中创建com.changgou.goods.dao.CategoryMapper接口，代码如下：

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

4.3.3 业务层

(1)业务层接口

修改changgou-service-goods，添加com.changgou.goods.service.CategoryService接口，代码如下：

```
public interface CategoryService {

    /***
     * Category多条件分页查询
     * @param category
     * @param page
     * @param size
     * @return
     */
    PageInfo<Category> findPage(Category category, int page, int size);

    /***
     * Category分页查询
     * @param page
     * @param size
     * @return
     */
    PageInfo<Category> findPage(int page, int size);

    /***
     * Category多条件搜索方法
     * @param category
     * @return
     */
    List<Category> findList(Category category);

    /***
     * 删除Category
     * @param id
     */
    void delete(Integer id);

    /***
     * 修改Category数据
     * @param category
     */
    void update(Category category);

    /***
```

```
     * 新增Category
     * @param category
     */
    void add(Category category);

    /**
     * 根据ID查询Category
     * @param id
     * @return
     */
     Category findById(Integer id);

    /***
     * 查询所有Category
     * @return
     */
    List<Category> findAll();

    /***
     * 根据父节点ID查询
     * @param pid:父节点ID
     */
    List<Category> findByParentId(Integer pid);
}
```

(2)业务层接口实现类

修改changgou-service-goods，添加com.changgou.goods.service.impl.CategoryServiceImpl接口实现类，代码如下：

```
@Service
public class CategoryServiceImpl implements CategoryService {

    @Autowired
    private CategoryMapper categoryMapper;


    /**
     * Category条件+分页查询
     * @param category 查询条件
     * @param page 页码
     * @param size 页大小
     * @return 分页结果
     */
    @Override
    public PageInfo<Category> findPage(Category category, int page, int size){
        //分页
        PageHelper.startPage(page,size);
        //搜索条件构建
        Example example = createExample(category);
        //执行搜索
        return new PageInfo<Category>(categoryMapper.selectByExample(example));
    }
```

```
24
25      /**
26       * Category分页查询
27       * @param page
28       * @param size
29       * @return
30       */
31      @Override
32      public PageInfo<Category> findPage(int page, int size){
33          //静态分页
34          PageHelper.startPage(page,size);
35          //分页查询
36          return new PageInfo<Category>(categoryMapper.selectAll());
37      }
38
39      /**
40       * Category条件查询
41       * @param category
42       * @return
43       */
44      @Override
45      public List<Category> findList(Category category){
46          //构建查询条件
47          Example example = createExample(category);
48          //根据构建的条件查询数据
49          return categoryMapper.selectByExample(example);
50      }
51
52
53      /**
54       * Category构建查询对象
55       * @param category
56       * @return
57       */
58      public Example createExample(Category category){
59          Example example=new Example(Category.class);
60          Example.Criteria criteria = example.createCriteria();
61          if(category!=null){
62              // 分类ID
63              if(!StringUtils.isEmpty(category.getId())){
64                      criteria.andEqualTo("id",category.getId());
65              }
66              // 分类名称
67              if(!StringUtils.isEmpty(category.getName())){
68                      criteria.andLike("name","%"+category.getName()+"%");
69              }
70              // 商品数量
71              if(!StringUtils.isEmpty(category.getGoodsNum())){
72
criteria.andEqualTo("goodsNum",category.getGoodsNum());
73              }
74              // 是否显示
75              if(!StringUtils.isEmpty(category.getIsShow())){
76                      criteria.andEqualTo("isShow",category.getIsShow());
77              }
78              // 是否导航
79              if(!StringUtils.isEmpty(category.getIsMenu())){
80                      criteria.andEqualTo("isMenu",category.getIsMenu());
```

```
81              }
82              // 排序
83              if(!StringUtils.isEmpty(category.getSeq())){
84                      criteria.andEqualTo("seq",category.getSeq());
85              }
86              // 上级ID
87              if(!StringUtils.isEmpty(category.getParentId())){
88
criteria.andEqualTo("parentId",category.getParentId());
89              }
90              // 模板ID
91              if(!StringUtils.isEmpty(category.getTemplateId())){
92
criteria.andEqualTo("templateId",category.getTemplateId());
93              }
94          }
95          return example;
96      }
97
98      /**
99       * 删除
100      * @param id
101      */
102     @Override
103     public void delete(Integer id){
104         categoryMapper.deleteByPrimaryKey(id);
105     }
106
107     /**
108      * 修改Category
109      * @param category
110      */
111     @Override
112     public void update(Category category){
113         categoryMapper.updateByPrimaryKey(category);
114     }
115
116     /**
117      * 增加Category
118      * @param category
119      */
120     @Override
121     public void add(Category category){
122         categoryMapper.insert(category);
123     }
124
125     /**
126      * 根据ID查询Category
127      * @param id
128      * @return
129      */
130     @Override
131     public Category findById(Integer id){
132         return  categoryMapper.selectByPrimaryKey(id);
133     }
134
135     /**
136      * 查询Category全部数据
```

```
     * @return
     */
    @Override
    public List<Category> findAll() {
        return categoryMapper.selectAll();
    }

    /***
     * 根据父节点ID查询
     * @param pid:父节点ID
     */
    @Override
    public List<Category> findByParentId(Integer pid) {
        Category category = new Category();
        category.setParentId(pid);
        return categoryMapper.select(category);
    }
}
```

4.3.4 控制层

修改changgou-service-goods，添加com.changgou.goods.controller.CategoryController,代码如下:

```
@RestController
@RequestMapping("/category")
@CrossOrigin
public class CategoryController {

    @Autowired
    private CategoryService categoryService;

    /***
     * Category分页条件搜索实现
     * @param category
     * @param page
     * @param size
     * @return
     */
    @PostMapping(value = "/search/{page}/{size}" )
    public Result<PageInfo> findPage(@RequestBody(required = false)
 Category category, @PathVariable  int page, @PathVariable  int size){
        //执行搜索
        PageInfo<Category> pageInfo = categoryService.findPage(category,
page, size);
        return new Result(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * Category分页搜索实现
     * @param page:当前页
     * @param size:每页显示多少条
     * @return
     */
    @GetMapping(value = "/search/{page}/{size}" )
```

```
    public Result<PageInfo> findPage(@PathVariable  int page, @PathVariable  int size){
        //分页查询
        PageInfo<Category> pageInfo = categoryService.findPage(page, size);
        return new Result<PageInfo>(true,StatusCode.OK,"查询成功",pageInfo);
    }

    /***
     * 多条件搜索品牌数据
     * @param category
     * @return
     */
    @PostMapping(value = "/search" )
    public Result<List<Category>> findList(@RequestBody(required = false) Category category){
        List<Category> list = categoryService.findList(category);
        return new Result<List<Category>>(true,StatusCode.OK,"查询成功",list);
    }

    /***
     * 根据ID删除品牌数据
     * @param id
     * @return
     */
    @DeleteMapping(value = "/{id}" )
    public Result delete(@PathVariable Integer id){
        categoryService.delete(id);
        return new Result(true,StatusCode.OK,"删除成功");
    }

    /***
     * 修改Category数据
     * @param category
     * @param id
     * @return
     */
    @PutMapping(value="/{id}")
    public Result update(@RequestBody  Category category,@PathVariable Integer id){
        //设置主键值
        category.setId(id);
        //修改数据
        categoryService.update(category);
        return new Result(true,StatusCode.OK,"修改成功");
    }

    /***
     * 新增Category数据
     * @param category
     * @return
     */
    @PostMapping
    public Result add(@RequestBody   Category category){
        categoryService.add(category);
        return new Result(true,StatusCode.OK,"添加成功");
```

```
    }

    /***
     * 根据ID查询Category数据
     * @param id
     * @return
     */
    @GetMapping("/{id}")
    public Result<Category> findById(@PathVariable Integer id){
        //根据ID查询
        Category category = categoryService.findById(id);
        return new Result<Category>(true,StatusCode.OK,"查询成功",category);
    }

    /***
     * 查询Category全部数据
     * @return
     */
    @GetMapping
    public Result<Category> findAll(){
        List<Category> list = categoryService.findAll();
        return new Result<Category>(true, StatusCode.OK,"查询成功",list) ;
    }

    /**
     * 根据父ID查询
     */
    @RequestMapping(value ="/list/{pid}")
    public Result<Category> findByPrantId(@PathVariable(value = "pid")Integer pid){
        //根据父节点ID查询
        List<Category> list = categoryService.findByParentId(pid);
        return new Result<Category>(true,StatusCode.OK,"查询成功",list);
    }
}
```

Swagger测试：

总结

自习的顺序:

1.停止第一天没有写完的代码!

2.搭建好文件微服务,实现文件的上传/下载/删除/查看

3.配置文件的上传的域名(把IP+端口号的上传方式,换成域名上传)

4.配置图片查看的域名(把IP+端口号的查看凡是,换成域名查看)

5.规格 参数 模板表的增删改查

6.必须要了解FastDFS的工作流程:至少能说出文件上传的流程

7.商品分类的增删改查 相册的增删改