

# 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

1	分布式文件管理系统
2	文件上传
3	文件下载
4	文件删除
5	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](http://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：

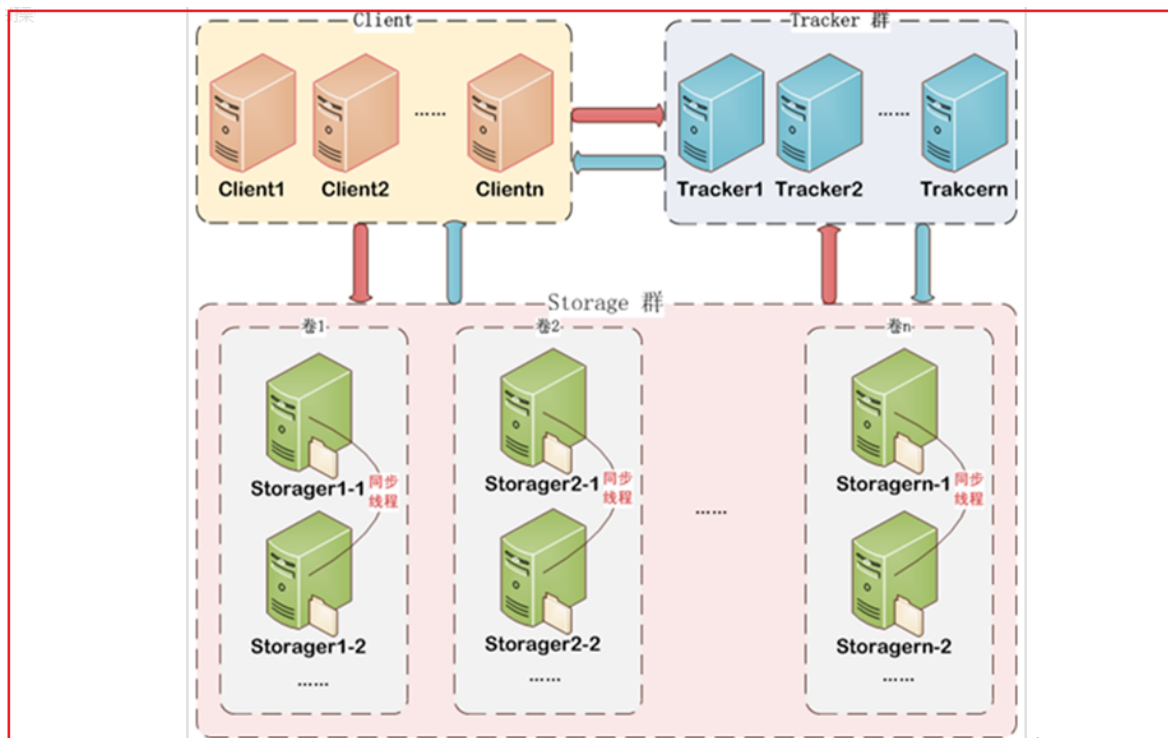
FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

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，Storage server 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：

- 1 **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。

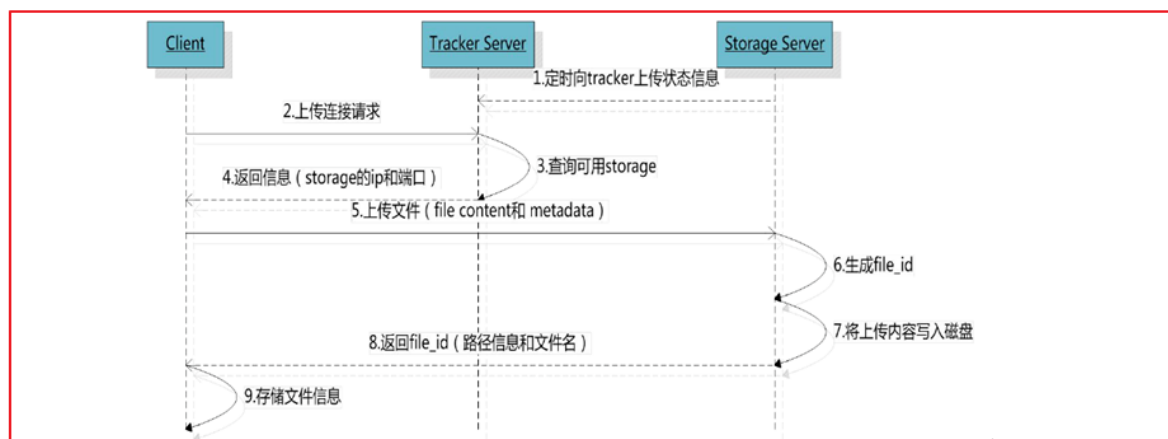
FastDFS组件：

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

FastDFS特性：

- 1 | **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。

**group1 /M00 /02/44/ wKgDrE34E8wAAAAAAAAAGkEYJK42378.sh**

**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)

- 1 | `docker pull morunchang/fastdfs`

运行tracker

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

运行storage

```
1 | 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

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

进入后

```
1 | vi /etc/nginx/conf/nginx.conf
```

添加以下内容:

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

上图配置如下:

```
1 | location ~ /M00 {  
2 |     ngx_fastdfs_module;  
3 | }
```

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:

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

退出容器:

```
1 | exit
```

重启storage容器:

```
1 | docker restart storage
```

查看启动容器 `docker ps`

```
1 9f2391f73d97 morunchang/fastdfs "sh storage.sh" 12 minutes ago Up 12 seconds storage
2 e22a3c7f95ea morunchang/fastdfs "sh tracker.sh" 13 minutes ago Up 13 minutes tracker
```

开启启动设置:

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

安装Nginx目的:

```
1 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

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>changgou-service</artifactId>
8         <groupId>com.changgou</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>changgou-service-file</artifactId>
13    <description>文件上传工程</description>
14
15    <!-- 依赖包 -->
16    <dependencies>
17        <dependency>
18            <groupId>net.oschina.zcx7878</groupId>
19            <artifactId>fastdfs-client-java</artifactId>
20            <version>1.27.0.0</version>
21        </dependency>
22        <dependency>
23            <groupId>com.changgou</groupId>
24            <artifactId>changgou-common</artifactId>
25            <version>1.0-SNAPSHOT</version>
26        </dependency>
27    </dependencies>
```

### 1.3.2 FastDFS配置

在resources文件夹下创建fasDFS的配置文件fdfs\_client.conf

```
1 charset=UTF-8
2 http.tracker_http_port=8080
3 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

```
1 spring:
2   servlet:
3     multipart:
4       max-file-size: 10MB
5       max-request-size: 10MB
6   application:
7     name: file
8 server:
9   port: 18082
10 eureka:
11   client:
12     service-url:
13       defaultZone: http://127.0.0.1:7001/eureka
14   instance:
15     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

```
1 @SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
2 @EnableEurekaClient
3 public class FileApplication {
4
5     public static void main(String[] args) {
6         SpringApplication.run(FileApplication.class);
7     }
8 }
```

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射解析记录，让自己访问域名 `file-changgou-java.itheima.net` 的时候能够访问虚拟机：

```
1 | 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：

```
1  #图片上传路径
2  server {
3      listen 80;
4      server_name file-changgou-java.itheima.net;
5
6      location / {
7          proxy_pass http://myip.com:18082;
8      }
9  }
```

## 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称、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：

```
1  public class FastDFSFile implements Serializable {
2
3      //文件名字
4      private String name;
5      //文件内容
```

```

6     private byte[] content;
7     //文件扩展名
8     private String ext;
9     //文件MD5摘要值
10    private String md5;
11    //文件创建作者
12    private String author;
13
14    public FastDFSFile(String name, byte[] content, String ext, String md5,
String author) {
15        this.name = name;
16        this.content = content;
17        this.ext = ext;
18        this.md5 = md5;
19        this.author = author;
20    }
21
22    public FastDFSFile(String name, byte[] content, String ext) {
23        this.name = name;
24        this.content = content;
25        this.ext = ext;
26    }
27
28    public FastDFSFile() {
29    }
30
31    //..get..set..toString
32 }

```

### 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:

```

1  /**
2   * 初始化tracker信息
3   */
4  static {
5      try {
6          //获取tracker的配置文件fdfs_client.conf的位置
7          String filePath = new
ClassPathResource("fdfs_client.conf").getPath();
8          //加载tracker配置信息
9          ClientGlobal.init(filePath);
10     } catch (Exception e) {
11         e.printStackTrace();
12     }
13 }

```

#### (2)文件上传



在类中添加如下方法实现文件上传：

```
1  /**
2   * 文件上传
3   * @param file : 要上传的文件信息封装->FastDFSFile
4   * @return String[]
5   *      1:文件上传所存储的组名
6   *      2:文件存储路径
7   */
8  public static String[] upload(FastDFSFile file){
9      //获取文件作者
10     NameValuePair[] meta_list = new NameValuePair[1];
11     meta_list[0] =new NameValuePair(file.getAuthor());
12
13     /**
14      * 文件上传后的返回值
15      * uploadResults[0]:文件上传所存储的组名, 例如:group1
16      * uploadResults[1]:文件存储路径,例如:
17      M00/00/00/wKjThF0DBzaAP23MAAXz2mMp9oM26.jpeg
18      */
19     String[] uploadResults = null;
20     try {
21         //创建TrackerClient客户端对象
22         TrackerClient trackerClient = new TrackerClient();
23         //通过TrackerClient对象获取TrackerServer信息
24         TrackerServer trackerServer = trackerClient.getConnection();
25         //获取StorageClient对象
26         StorageClient storageClient = new StorageClient(trackerServer,
27         null);
28         //执行文件上传
29         uploadResults = storageClient.upload_file(file.getContent(),
30         file.getExt(), meta_list);
31     } catch (Exception e) {
32         e.printStackTrace();
33     }
34     return uploadResults;
35 }
```

### (3)获取文件信息

在类中添加如下方法实现获取文件信息：

```
1  /**
2   * 获取文件信息
3   * @param groupName:组名
4   * @param remoteFileName: 文件存储完整名
5   */
6  public static FileInfo getFile(String groupName,String remoteFileName){
7      try {
8          //创建TrackerClient对象
9          TrackerClient trackerClient = new TrackerClient();
10         //通过TrackerClient获得TrackerServer信息
11         TrackerServer trackerServer =trackerClient.getConnection();
12         //通过TrackerServer获取StorageClient对象
13         StorageClient storageClient = new StorageClient(trackerServer,null);
14         //获取文件信息
15         return storageClient.get_file_info(groupName,remoteFileName);
16     }
```

```

16     } catch (Exception e) {
17         e.printStackTrace();
18     }
19     return null;
20 }

```

#### (4)文件下载

在类中添加如下方法实现文件下载：

```

1  /**
2   * 文件下载
3   * @param groupName:组名
4   * @param remoteFileName: 文件存储完整名
5   * @return
6   */
7  public static InputStream downFile(String groupName,String remoteFileName){
8      try {
9          //创建TrackerClient对象
10         TrackerClient trackerClient = new TrackerClient();
11         //通过TrackerClient对象创建TrackerServer
12         TrackerServer trackerServer = trackerClient.getConnection();
13         //通过TrackerServer创建StorageClient
14         StorageClient storageClient = new StorageClient(trackerServer,null);
15         //通过StorageClient下载文件
16         byte[] fileByte = storageClient.download_file(groupName,
remoteFileName);
17         //将字节数组转换成字节输入流
18         return new ByteArrayInputStream(fileByte);
19     } catch (Exception e) {
20         e.printStackTrace();
21     }
22     return null;
23 }

```

#### (5)文件删除实现

```

1  /**
2   * 文件删除实现
3   * @param groupName:组名
4   * @param remoteFileName: 文件存储完整名
5   */
6  public static void deleteFile(String groupName,String remoteFileName){
7      try {
8          //创建TrackerClient对象
9          TrackerClient trackerClient = new TrackerClient();
10         //通过TrackerClient获取TrackerServer对象
11         TrackerServer trackerServer = trackerClient.getConnection();
12         //通过TrackerServer创建StorageClient
13         StorageClient storageClient = new StorageClient(trackerServer,null);
14         //通过StorageClient删除文件
15         storageClient.delete_file(groupName,remoteFileName);
16     } catch (Exception e) {
17         e.printStackTrace();
18     }
19 }

```

## (6)获取组信息

```
1  /**
2   * 获取组信息
3   * @param groupName :组名
4   */
5  public static StorageServer getStorages(String groupName){
6      try {
7          //创建TrackerClient对象
8          TrackerClient trackerClient = new TrackerClient();
9          //通过TrackerClient获取TrackerServer对象
10         TrackerServer trackerServer = trackerClient.getConnection();
11         //通过trackerClient获取Storage组信息
12         return trackerClient.getStoreStorage(trackerServer,groupName);
13     } catch (Exception e) {
14         e.printStackTrace();
15     }
16     return null;
17 }
```

## (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

```
1  /**
2   * 根据文件组名和文件存储路径获取Storage服务的IP、端口信息
3   * @param groupName :组名
4   * @param remoteFileName : 文件存储完整名
5   */
6  public static ServerInfo[] getServerInfo(String groupName, String
remoteFileName){
7      try {
8          //创建TrackerClient对象
9          TrackerClient trackerClient = new TrackerClient();
10         //通过TrackerClient获取TrackerServer对象
11         TrackerServer trackerServer = trackerClient.getConnection();
12         //获取服务信息
13         return
trackerClient.getFetchStorages(trackerServer,groupName,remoteFileName);
14     } catch (Exception e) {
15         e.printStackTrace();
16     }
17     return null;
18 }
```

## (8)获取Tracker服务地址

```
1  /**
2   * 获取Tracker服务地址
3   */
4  public static String getTrackerUrl(){
5      try {
6          //创建TrackerClient对象
7          TrackerClient trackerClient = new TrackerClient();
8          //通过TrackerClient获取TrackerServer对象
9          TrackerServer trackerServer = trackerClient.getConnection();
10         //获取Tracker地址
```

```

11         return
12         "http://" + trackerServer.getInetSocketAddress().getHostString() + ":" + ClientGlobal.getG_tracker_http_port();
13     } catch (IOException e) {
14         e.printStackTrace();
15     }
16     return null;
17 }

```

#### (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：

```

1  /**
2   * 获取TrackerServer
3   */
4  public static TrackerServer getTrackerServer() throws Exception{
5      //创建TrackerClient对象
6      TrackerClient trackerClient = new TrackerClient();
7      //通过TrackerClient获取TrackerServer对象
8      TrackerServer trackerServer = trackerClient.getConnection();
9      return trackerServer;
10 }
11
12 /**
13 * 获取StorageClient
14 * @return
15 * @throws Exception
16 */
17 public static StorageClient getStorageClient() throws Exception{
18     //获取TrackerServer
19     TrackerServer trackerServer = getTrackerServer();
20     //通过TrackerServer创建StorageClient
21     StorageClient storageClient = new StorageClient(trackerServer, null);
22     return storageClient;
23 }

```

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：

```

1  public class FastDFSClient {
2
3      /**
4       * 初始化tracker信息
5       */
6      static {
7          try {
8              //获取tracker的配置文件fdfs_client.conf的位置
9              String filePath = new
10 ClassPathResource("fdfs_client.conf").getPath();
11              //加载tracker配置信息
12              ClientGlobal.init(filePath);
13          } catch (Exception e) {
14              e.printStackTrace();
15          }
16      }
17
18 }

```

```

15     }
16
17     /****
18     * 文件上传
19     * @param file : 要上传的文件信息封装->FastDFSFile
20     * @return String[]
21     *         1:文件上传所存储的组名
22     *         2:文件存储路径
23     */
24     public static String[] upload(FastDFSFile file){
25         //获取文件作者
26         NameValuePair[] meta_list = new NameValuePair[1];
27         meta_list[0] =new NameValuePair(file.getAuthor());
28
29         /****
30         * 文件上传后的返回值
31         * uploadResults[0]:文件上传所存储的组名, 例如:group1
32         * uploadResults[1]:文件存储路径, 例如:
33         M00/00/00/wkjtHf0DBZaAP23MAAXz2mMp9oM26.jpeg
34         */
35         String[] uploadResults = null;
36         try {
37             //获取StorageClient对象
38             StorageClient storageClient = getStorageClient();
39             //执行文件上传
40             uploadResults = storageClient.upload_file(file.getContent(),
41 file.getExt(), meta_list);
42         } catch (Exception e) {
43             e.printStackTrace();
44         }
45         return uploadResults;
46     }
47
48     /****
49     * 获取文件信息
50     * @param groupName:组名
51     * @param remoteFileName: 文件存储完整名
52     */
53     public static FileInfo getFile(String groupName,String remoteFileName)
54     {
55         try {
56             //获取StorageClient对象
57             StorageClient storageClient = getStorageClient();
58             //获取文件信息
59             return storageClient.get_file_info(groupName,remoteFileName);
60         } catch (Exception e) {
61             e.printStackTrace();
62         }
63         return null;
64     }
65
66     /****
67     * 文件下载
68     * @param groupName:组名
69     * @param remoteFileName: 文件存储完整名
70     * @return
71     */

```

```

70     public static InputStream downFile(String groupName,String
remoteFileName){
71         try {
72             //获取StorageClient
73             StorageClient storageClient = getStorageClient();
74             //通过StorageClient下载文件
75             byte[] fileByte = storageClient.download_file(groupName,
remoteFileName);
76             //将字节数组转换成字节输入流
77             return new ByteArrayInputStream(fileByte);
78         } catch (Exception e) {
79             e.printStackTrace();
80         }
81         return null;
82     }
83
84     /**
85      * 文件删除实现
86      * @param groupName:组名
87      * @param remoteFileName: 文件存储完整名
88      */
89     public static void deleteFile(String groupName,String remoteFileName){
90         try {
91             //获取StorageClient
92             StorageClient storageClient = getStorageClient();
93             //通过StorageClient删除文件
94             storageClient.delete_file(groupName,remoteFileName);
95         } catch (Exception e) {
96             e.printStackTrace();
97         }
98     }
99
100
101     /**
102      * 获取组信息
103      * @param groupName :组名
104      */
105     public static StorageServer getStorages(String groupName){
106         try {
107             //创建TrackerClient对象
108             TrackerClient trackerClient = new TrackerClient();
109             //通过TrackerClient获取TrackerServer对象
110             TrackerServer trackerServer = trackerClient.getConnection();
111             //通过trackerClient获取Storage组信息
112             return trackerClient.getStoreStorage(trackerServer,groupName);
113         } catch (Exception e) {
114             e.printStackTrace();
115         }
116         return null;
117     }
118
119     /**
120      * 根据文件组名和文件存储路径获取Storage服务的IP、端口信息
121      * @param groupName :组名
122      * @param remoteFileName : 文件存储完整名
123      */
124     public static ServerInfo[] getServerInfo(String groupName, String
remoteFileName){

```

```

125         try {
126             //创建TrackerClient对象
127             TrackerClient trackerClient = new TrackerClient();
128             //通过TrackerClient获取TrackerServer对象
129             TrackerServer trackerServer = trackerClient.getConnection();
130             //获取服务信息
131             return
trackerClient.getFetchStorages(trackerServer,groupName,remoteFileName);
132         } catch (Exception e) {
133             e.printStackTrace();
134         }
135         return null;
136     }
137
138     /**
139      * 获取Tracker服务地址
140      */
141     public static String getTrackerUrl(){
142         try {
143             //创建TrackerClient对象
144             TrackerClient trackerClient = new TrackerClient();
145             //通过TrackerClient获取TrackerServer对象
146             TrackerServer trackerServer = trackerClient.getConnection();
147             //获取Tracker地址
148             return
"http://" + trackerServer.getInetAddress().getHostString() + ":" + ClientG
lobal.getG_tracker_http_port();
149         } catch (IOException e) {
150             e.printStackTrace();
151         }
152         return null;
153     }
154
155     /**
156      * 获取TrackerServer
157      */
158     public static TrackerServer getTrackerServer() throws Exception{
159         //创建TrackerClient对象
160         TrackerClient trackerClient = new TrackerClient();
161         //通过TrackerClient获取TrackerServer对象
162         TrackerServer trackerServer = trackerClient.getConnection();
163         return trackerServer;
164     }
165
166     /**
167      * 获取StorageClient
168      * @return
169      * @throws Exception
170      */
171     public static StorageClient getStorageClient() throws Exception{
172         //获取TrackerServer
173         TrackerServer trackerServer = getTrackerServer();
174         //通过TrackerServer创建StorageClient
175         StorageClient storageClient = new
StorageClient(trackerServer,null);
176         return storageClient;
177     }
178 }

```

### 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：

```
1  @RestController
2  @CrossOrigin
3  public class FileController {
4
5      /**
6       * 文件上传
7       * @return
8       */
9      @PostMapping(value = "/upload")
10     public String upload(@RequestParam("file")MultipartFile file) throws
Exception {
11         //封装一个FastDFSFile
12         FastDFSFile fastDFSFile = new FastDFSFile(
13             file.getOriginalFilename(), //文件名字
14             file.getBytes(),           //文件字节数组
15
16             StringUtils.getFilenameExtension(file.getOriginalFilename())); //文件扩展名
17
18         //文件上传
19         String[] uploads = FastDFSClient.upload(fastDFSFile);
20         //组装文件上传地址
21         return "http://192.168.211.132:8080"+"/"+uploads[0]+"/"+uploads[1];
22     }
23 }
```

## 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。

```
1 | docker exec -it storage /bin/bash
2 | 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;
    }
}
```

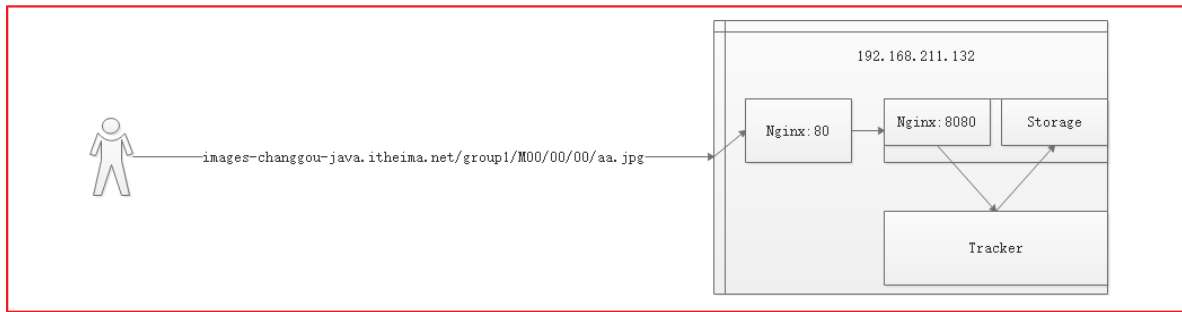
修改后重启storage即可根据自己修改的端口访问图片了。

## FastDFS总结

```
1 | 文件上传: storageClient.upload_file("文件内容", "扩展名", "自定义属性");
2 | 获取文件信息: storageClient.get_file_info("storage组名", "文件的存路径详细名字");
3 | 文件下载: storageClient.download_file("storage组名", "文件的存路径详细名字");
4 | 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

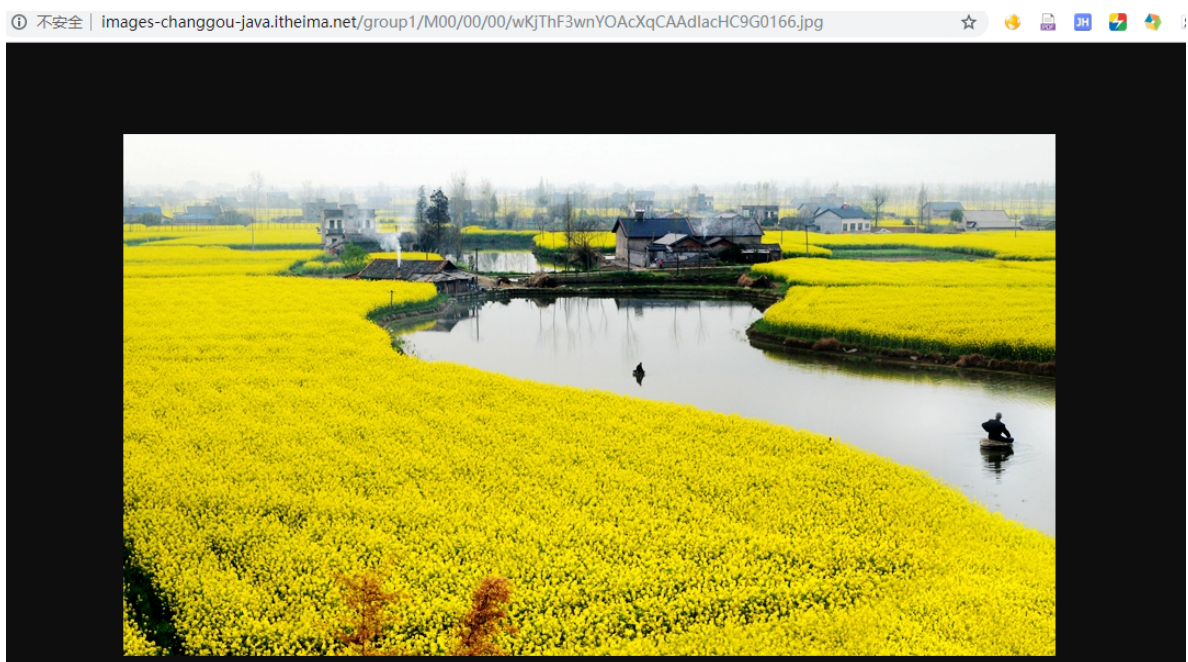
图片的域名为 `images-changgou-java.itheima.net`，用户每次请求的时候，如果不想加端口号，则可以使用80端口的Nginx进行转发操作，如下图：



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：

```
1  修改nginx.conf文件
2  cd /usr/local/openresty/nginx/conf
3
4  添加如下配置:
5  server {
6      listen 80;
7      server_name images-changgou-java.itheima.net;
8
9      location / {
10         proxy_pass http://127.0.0.1:8080;
11     }
12 }
13
14 重启Nginx
15 /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：

```

/**
 * 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：

POST /upload 文件上传

文件上传方法详情

Parameters

Cancel

Name	Description
file * required	
file	需要上传的文件
(formData)	

选择文件 12.png

Execute Clear

Responses

Response content type text/plain; charset=utf-8

Curl

```
curl -X POST "http://file-changgou-java.itheima.net/upload" -H "accept: text/plain; charset=utf-8" -H "Content-Type: multipart/form-data" -F "file=@12.png;type=image/png"
```

Request URL

http://file-changgou-java.itheima.net/upload

Server response

Code	Details
200	<div>Response body</div> <div>http://images-changgou-java.itheima.net/group1/M00/00/00/s5TlF3mCSAbTlAAAF9lPtwE903.png</div> <div>Download</div> <div>Response headers</div> <div>content-type: text/plain; charset=utf-8</div>

Responses

Code	Description
200	<div>文件上传成功</div> <div>Example Value   Model</div> <div>string</div>
400	Invalid status value(无效的状态值)
403	403 Forbidden(请求被拒绝)
404	not found(没有找到相关资源)
405	Invalid input(无效的输入)
500	服务器内部错误

## 1.7 品牌文件上传测试

畅购

商品

Jay Lin

商品管理

商品配置

商品分类

品牌参数

品牌管理

添加品牌

品牌名称: 传智播客IT培训高端品牌

品牌前字母: C

品牌LOGO: 

11.png

提交

效果如下：

<input type="checkbox"/>	325425	传智播客IT培训高端品牌	C		编辑 删除
--------------------------	--------	--------------	---	--	-------

## 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：

```
1  [
2    {
3      "url": "http://localhost:9101/img/1.jpg",
4      "uid": 1548143143154,
5      "status": "success"
6    },
7    {
8      "url": "http://localhost:9101/img/7.jpg",
9      "uid": 1548143143155,
10     "status": "success"
11   }
12 ]
```

## 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：

```
1  @Table(name="tb_album")
2  public class Album implements Serializable{
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      @Column(name = "id")
7      private Long id;//编号
8
9      @Column(name = "title")
10     private String title;//相册名称
11
12     @Column(name = "image")
13     private String image;//相册封面
14
15     @Column(name = "image_items")
16     private String imageItems;//图片列表
17
18     //get...set...toString..
19 }
```

### 2.3.2 Dao

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

```
1  public interface AlbumMapper extends Mapper<Album> {
2  }
```

### 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：

```
1  public interface AlbumService {
2
3      /**
4       * Album多条件分页查询
5       * @param album
6       * @param page
7       * @param size
8       * @return
9       */
10     PageInfo<Album> findPage(Album album, int page, int size);
11
12     /**
13      * Album分页查询
14      * @param page
15      * @param size
16      * @return
17      */
18     PageInfo<Album> findPage(int page, int size);
19
20     /**
21      * Album多条件搜索方法
22      * @param album
23      * @return
24      */
25     List<Album> findList(Album album);
26
27     /**
28      * 删除Album
29      * @param id
30      */
31     void delete(Long id);
32
33     /**
34      * 修改Album数据
35      * @param album
36      */
37     void update(Album album);
38
39     /**
40      * 新增Album
41      * @param album
42      */
43     void add(Album album);
44
45     /**
46      * 根据ID查询Album
47      * @param id
48      * @return
49      */
50     Album findById(Long id);
51
52     /**
```

```

53     * 查询所有Album
54     * @return
55     */
56     List<Album> findAll();
57 }

```

## (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：

```

1  @Service
2  public class AlbumServiceImpl implements AlbumService {
3
4      @Autowired
5      private AlbumMapper albumMapper;
6
7
8      /**
9       * Album条件+分页查询
10      * @param album 查询条件
11      * @param page 页码
12      * @param size 页大小
13      * @return 分页结果
14      */
15      @Override
16      public PageInfo<Album> findPage(Album album, int page, int size){
17          //分页
18          PageHelper.startPage(page,size);
19          //搜索条件构建
20          Example example = createExample(album);
21          //执行搜索
22          return new PageInfo<Album>(albumMapper.selectByExample(example));
23      }
24
25      /**
26      * Album分页查询
27      * @param page
28      * @param size
29      * @return
30      */
31      @Override
32      public PageInfo<Album> findPage(int page, int size){
33          //静态分页
34          PageHelper.startPage(page,size);
35          //分页查询
36          return new PageInfo<Album>(albumMapper.selectAll());
37      }
38
39      /**
40      * Album条件查询
41      * @param album
42      * @return
43      */
44      @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
77                 criteria.andEqualTo("imageItems",album.getImageItems());
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

```



```

102     * @param album
103     */
104     @Override
105     public void add(Album album){
106         albumMapper.insert(album);
107     }
108
109     /**
110     * 根据ID查询Album
111     * @param id
112     * @return
113     */
114     @Override
115     public Album findById(Long id){
116         return albumMapper.selectByPrimaryKey(id);
117     }
118
119     /**
120     * 查询Album全部数据
121     * @return
122     */
123     @Override
124     public List<Album> findAll() {
125         return albumMapper.selectAll();
126     }
127 }

```

### 2.3.4 控制层

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

```

1  @RestController
2  @RequestMapping("/album")
3  @CrossOrigin
4  public class AlbumController {
5
6      @Autowired
7      private AlbumService albumService;
8
9      /**
10     * Album分页条件搜索实现
11     * @param album
12     * @param page
13     * @param size
14     * @return
15     */
16     @PostMapping(value = "/search/{page}/{size}" )
17     public Result<PageInfo> findPage(@RequestBody(required = false) Album
18     album, @PathVariable int page, @PathVariable int size){
19         //执行搜索
20         PageInfo<Album> pageInfo = albumService.findPage(album, page,
21         size);
22         return new Result(true,StatusCode.OK,"查询成功",pageInfo);
23     }
24 }

```

```

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

```

```

77     */
78     @PostMapping
79     public Result add(@RequestBody Album album){
80         albumService.add(album);
81         return new Result(true,StatusCode.OK,"添加成功");
82     }
83
84     /**
85      * 根据ID查询Album数据
86      * @param id
87      * @return
88      */
89     @GetMapping("/{id}")
90     public Result<Album> findById(@PathVariable Long id){
91         //根据ID查询
92         Album album = albumService.findById(id);
93         return new Result<Album>(true,StatusCode.OK,"查询成功",album);
94     }
95
96     /**
97      * 查询Album全部数据
98      * @return
99      */
100    @GetMapping
101    public Result<Album> findAll(){
102        List<Album> list = albumService.findAll();
103        return new Result<Album>(true, StatusCode.OK,"查询成功",list) ;
104    }
105 }

```

添加相册示例数据如下：

```

1  {
2      "title": "北京冬天的雪景",
3      "image": "http://localhost:9101/img/1.jpg",
4      "imageItems": "[ {\"url\": \"http://localhost:9101/img/1.jpg\", \"uid\": 1548143143154, \"status\": \"success\"}, {\"url\": \"http://localhost:9101/img/7.jpg\", \"uid\": 1548143143155, \"status\": \"success\"}]"
5  }

```

用Swagger测试上面的功能:

AlbumController AlbumController	
GET	/album 查询所有Album
POST	/album 添加Album
DELETE	/album/{id} 根据ID删除Album
PUT	/album/{id} 根据ID修改Album
GET	/album/{id} 根据ID查询Album
POST	/album/search 不带分页条件搜索Album
GET	/album/search/{page}/{size} 分页列表查询Album
POST	/album/search/{page}/{size} 分页条件搜索Album

# 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：

```

1  @Table(name="tb_template")
2  public class Template implements Serializable{
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      @Column(name = "id")
7      private Integer id;//ID
8
9      @Column(name = "name")
10     private String name;//模板名称
11
12     @Column(name = "spec_num")
13     private Integer specNum;//规格数量
14
15     @Column(name = "para_num")
16     private Integer paraNum;//参数数量
17     //..get..set..toString
18 }

```

### 3.3.2 Dao

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

```
1 public interface TemplateMapper extends Mapper<Template> {  
2 }
```

### 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:

```
1 public interface TemplateService {  
2  
3     /**  
4      * Template多条件分页查询  
5      * @param template  
6      * @param page  
7      * @param size  
8      * @return  
9      */  
10    PageInfo<Template> findPage(Template template, int page, int size);  
11  
12    /**  
13     * Template分页查询  
14     * @param page  
15     * @param size  
16     * @return  
17     */  
18    PageInfo<Template> findPage(int page, int size);  
19  
20    /**  
21     * Template多条件搜索方法  
22     * @param template  
23     * @return  
24     */  
25    List<Template> findList(Template template);  
26  
27    /**  
28     * 删除Template  
29     * @param id  
30     */  
31    void delete(Integer id);  
32  
33    /**  
34     * 修改Template数据  
35     * @param template  
36     */  
37    void update(Template template);  
38  
39    /**
```

```

40     * 新增Template
41     * @param template
42     */
43     void add(Template template);
44
45     /**
46     * 根据ID查询Template
47     * @param id
48     * @return
49     */
50     Template findById(Integer id);
51
52     /**
53     * 查询所有Template
54     * @return
55     */
56     List<Template> findAll();
57 }

```

## (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：

```

1  @Service
2  public class TemplateServiceImpl implements TemplateService {
3
4      @Autowired
5      private TemplateMapper templateMapper;
6
7
8      /**
9      * Template条件+分页查询
10     * @param template 查询条件
11     * @param page 页码
12     * @param size 页大小
13     * @return 分页结果
14     */
15     @Override
16     public PageInfo<Template> findPage(Template template, int page, int
size){
17         //分页
18         PageHelper.startPage(page, size);
19         //搜索条件构建
20         Example example = createExample(template);
21         //执行搜索
22         return new PageInfo<Template>
(templateMapper.selectByExample(example));
23     }
24
25     /**
26     * Template分页查询
27     * @param page
28     * @param size
29     * @return

```

```

30     */
31     @Override
32     public PageInfo<Template> findPage(int page, int size){
33         //静态分页
34         PageHelper.startPage(page,size);
35         //分页查询
36         return new PageInfo<Template>(templateMapper.selectAll());
37     }
38
39     /**
40      * Template条件查询
41      * @param template
42      * @return
43      */
44     @Override
45     public List<Template> findList(Template template){
46         //构建查询条件
47         Example example = createExample(template);
48         //根据构建的条件查询数据
49         return templateMapper.selectByExample(example);
50     }
51
52
53     /**
54      * Template构建查询对象
55      * @param template
56      * @return
57      */
58     public Example createExample(Template template){
59         Example example=new Example(Template.class);
60         Example.Criteria criteria = example.createCriteria();
61         if(template!=null){
62             // ID
63             if(!StringUtils.isEmpty(template.getId())){
64                 criteria.andEqualTo("id",template.getId());
65             }
66             // 模板名称
67             if(!StringUtils.isEmpty(template.getName())){
68                 criteria.andLike("name", "%"+template.getName()+"%");
69             }
70             // 规格数量
71             if(!StringUtils.isEmpty(template.getSpecNum())){
72                 criteria.andEqualTo("specNum",template.getSpecNum());
73             }
74             // 参数数量
75             if(!StringUtils.isEmpty(template.getParaNum())){
76                 criteria.andEqualTo("paraNum",template.getParaNum());
77             }
78         }
79         return example;
80     }
81
82     /**
83      * 删除
84      * @param id
85      */
86     @Override
87     public void delete(Integer id){

```



```

88         templateMapper.deleteByPrimaryKey(id);
89     }
90
91     /**
92      * 修改Template
93      * @param template
94      */
95     @Override
96     public void update(Template template){
97         templateMapper.updateByPrimaryKey(template);
98     }
99
100    /**
101     * 增加Template
102     * @param template
103     */
104    @Override
105    public void add(Template template){
106        templateMapper.insert(template);
107    }
108
109    /**
110     * 根据ID查询Template
111     * @param id
112     * @return
113     */
114    @Override
115    public Template findById(Integer id){
116        return templateMapper.selectByPrimaryKey(id);
117    }
118
119    /**
120     * 查询Template全部数据
121     * @return
122     */
123    @Override
124    public List<Template> findAll() {
125        return templateMapper.selectAll();
126    }
127 }

```

### 3.3.4 控制层

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

```

1  @RestController
2  @RequestMapping("/template")
3  @CrossOrigin
4  public class TemplateController {
5
6      @Autowired
7      private TemplateService templateService;
8
9      /**
10       * Template分页条件搜索实现

```

```

11     * @param template
12     * @param page
13     * @param size
14     * @return
15     */
16     @PostMapping(value = "/search/{page}/{size}" )
17     public Result<PageInfo> findPage(@RequestBody(required = false)
Template template, @PathVariable int page, @PathVariable int size){
18         //执行搜索
19         PageInfo<Template> pageInfo = templateService.findPage(template,
page, size);
20         return new Result(true,StatusCode.OK,"查询成功",pageInfo);
21     }
22
23     /**
24     * Template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<Template> pageInfo = templateService.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 template
39     * @return
40     */
41     @PostMapping(value = "/search" )
42     public Result<List<Template>> findList(@RequestBody(required = false)
Template template){
43         List<Template> list = templateService.findList(template);
44         return new Result<List<Template>>(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 Integer id){
54         templateService.delete(id);
55         return new Result(true,StatusCode.OK,"删除成功");
56     }
57
58     /**
59     * 修改Template数据
60     * @param template
61     * @param id

```

```

62     * @return
63     */
64     @PutMapping(value="/{id}")
65     public Result update(@RequestBody Template template,@PathVariable
Integer id){
66         //设置主键值
67         template.setId(id);
68         //修改数据
69         templateService.update(template);
70         return new Result(true,StatusCode.OK,"修改成功");
71     }
72
73     /**
74      * 新增Template数据
75      * @param template
76      * @return
77      */
78     @PostMapping
79     public Result add(@RequestBody Template template){
80         templateService.add(template);
81         return new Result(true,StatusCode.OK,"添加成功");
82     }
83
84     /**
85      * 根据ID查询Template数据
86      * @param id
87      * @return
88      */
89     @GetMapping("/{id}")
90     public Result<Template> findById(@PathVariable Integer id){
91         //根据ID查询
92         Template template = templateService.findById(id);
93         return new Result<Template>(true,StatusCode.OK,"查询成
功",template);
94     }
95
96     /**
97      * 查询Template全部数据
98      * @return
99      */
100    @GetMapping
101    public Result<Template> findAll(){
102        List<Template> list = templateService.findAll();
103        return new Result<Template>(true, StatusCode.OK,"查询成功",list) ;
104    }
105 }

```

Swagger测试:

TemplateController	
GET	/template 查询所有Template
POST	/template 添加Template
DELETE	/template/{id} 根据ID删除Template
PUT	/template/{id} 根据ID修改Template
GET	/template/{id} 根据ID查询Template
POST	/template/search 不带分页条件搜索Template
GET	/template/search/{page}/{size} 分页列表查询Template
POST	/template/search/{page}/{size} 分页条件搜索Template

## 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:

```

1  @Table(name="tb_spec")
2  public class Spec implements Serializable{
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      @Column(name = "id")
7      private Integer id;//ID
8
9      @Column(name = "name")
10     private String name;//名称
11
12     @Column(name = "options")
13     private String options;//规格选项
14
15     @Column(name = "seq")
16     private Integer seq;//排序
17
18     @Column(name = "template_id")
19     private Integer templateId;//模板ID
20
21     //get..set..toString
22 }

```

### 3.4.2 Dao

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

```

1  public interface SpecMapper extends Mapper<Spec> {
2  }

```

### 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：

```
1 public interface SpecService {
2
3     /**
4      * Spec多条件分页查询
5      * @param spec
6      * @param page
7      * @param size
8      * @return
9      */
10    PageInfo<Spec> findPage(Spec spec, int page, int size);
11
12    /**
13     * Spec分页查询
14     * @param page
15     * @param size
16     * @return
17     */
18    PageInfo<Spec> findPage(int page, int size);
19
20    /**
21     * Spec多条件搜索方法
22     * @param spec
23     * @return
24     */
25    List<Spec> findList(Spec spec);
26
27    /**
28     * 删除Spec
29     * @param id
30     */
31    void delete(Integer id);
32
33    /**
34     * 修改Spec数据
35     * @param spec
36     */
37    void update(Spec spec);
38
39    /**
40     * 新增Spec
41     * @param spec
42     */
43    void add(Spec spec);
44
45    /**
46     * 根据ID查询Spec
47     * @param id
48     * @return
49     */
50    Spec findById(Integer id);
51
52    /**
```

```

53     * 查询所有Spec
54     * @return
55     */
56     List<Spec> findAll();
57 }

```

## (2)业务层实现类

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

```

1  @Service
2  public class SpecServiceImpl implements SpecService {
3
4      @Autowired
5      private SpecMapper specMapper;
6
7      @Autowired
8      private TemplateMapper templateMapper;
9
10     /**
11      * Spec条件+分页查询
12      * @param spec 查询条件
13      * @param page 页码
14      * @param size 页大小
15      * @return 分页结果
16      */
17     @Override
18     public PageInfo<Spec> findPage(Spec spec, int page, int size){
19         //分页
20         PageHelper.startPage(page,size);
21         //搜索条件构建
22         Example example = createExample(spec);
23         //执行搜索
24         return new PageInfo<Spec>(specMapper.selectByExample(example));
25     }
26
27     /**
28      * Spec分页查询
29      * @param page
30      * @param size
31      * @return
32      */
33     @Override
34     public PageInfo<Spec> findPage(int page, int size){
35         //静态分页
36         PageHelper.startPage(page,size);
37         //分页查询
38         return new PageInfo<Spec>(specMapper.selectAll());
39     }
40
41     /**
42      * Spec条件查询
43      * @param spec
44      * @return
45      */
46     @Override

```

```

47     public List<Spec> findList(Spec spec){
48         //构建查询条件
49         Example example = createExample(spec);
50         //根据构建的条件查询数据
51         return specMapper.selectByExample(example);
52     }
53
54
55     /**
56      * Spec构建查询对象
57      * @param spec
58      * @return
59      */
60     public Example createExample(Spec spec){
61         Example example=new Example(Spec.class);
62         Example.Criteria criteria = example.createCriteria();
63         if(spec!=null){
64             // ID
65             if(!StringUtils.isEmpty(spec.getId())){
66                 criteria.andEqualTo("id",spec.getId());
67             }
68             // 名称
69             if(!StringUtils.isEmpty(spec.getName())){
70                 criteria.andLike("name","%" +spec.getName()+"%");
71             }
72             // 规格选项
73             if(!StringUtils.isEmpty(spec.getOptions())){
74                 criteria.andEqualTo("options",spec.getOptions());
75             }
76             // 排序
77             if(!StringUtils.isEmpty(spec.getSeq())){
78                 criteria.andEqualTo("seq",spec.getSeq());
79             }
80             // 模板ID
81             if(!StringUtils.isEmpty(spec.getTemplateId())){
82
83                 criteria.andEqualTo("templateId",spec.getTemplateId());
84             }
85             return example;
86         }
87
88     /**
89      * 删除
90      * @param id
91      */
92     @Override
93     public void delete(Integer id){
94         //查询模板
95         Spec spec = specMapper.selectByPrimaryKey(id);
96         //变更模板数量
97         updateSpecNum(spec,-1);
98
99         //删除指定规格
100         specMapper.deleteByPrimaryKey(id);
101     }
102
103     /**

```

```

104     * 修改Spec
105     * @param spec
106     */
107     @Override
108     public void update(Spec spec){
109         specMapper.updateByPrimaryKey(spec);
110     }
111
112     /**
113     * 增加Spec
114     * @param spec
115     */
116     @Override
117     public void add(Spec spec){
118         specMapper.insert(spec);
119         //变更模板数量
120         updateSpecNum(spec,1);
121     }
122
123     /**
124     * 根据ID查询Spec
125     * @param id
126     * @return
127     */
128     @Override
129     public Spec findById(Integer id){
130         return specMapper.selectByPrimaryKey(id);
131     }
132
133     /**
134     * 查询Spec全部数据
135     * @return
136     */
137     @Override
138     public List<Spec> findAll() {
139         return specMapper.selectAll();
140     }
141
142
143     /**
144     * 修改模板统计数据
145     * @param spec:操作的模板
146     * @param count:变更的数量
147     */
148     public void updateSpecNum(Spec spec,int count){
149         //修改模板数量统计
150         Template template =
templateMapper.selectByPrimaryKey(spec.getTemplateId());
151         template.setSpecNum(template.getSpecNum()+count);
152         templateMapper.updateByPrimaryKeySelective(template);
153     }
154 }

```

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:

```
1  @RestController
2  @RequestMapping("/spec")
3  @CrossOrigin
4  public class SpecController {
5
6      @Autowired
7      private SpecService specService;
8
9      /**
10       * Spec分页条件搜索实现
11       * @param spec
12       * @param page
13       * @param size
14       * @return
15       */
16      @PostMapping(value = "/search/{page}/{size}" )
17      public Result<PageInfo> findPage(@RequestBody(required = false) Spec
18      spec, @PathVariable int page, @PathVariable int size){
19          //执行搜索
20          PageInfo<Spec> pageInfo = specService.findPage(spec, page, size);
21          return new Result<PageInfo>(true, StatusCode.OK, "查询成功", pageInfo);
22      }
23
24      /**
25       * Spec分页搜索实现
26       * @param page:当前页
27       * @param size:每页显示多少条
28       * @return
29       */
30      @GetMapping(value = "/search/{page}/{size}" )
31      public Result<PageInfo> findPage(@PathVariable int page,
32      @PathVariable int size){
33          //分页查询
34          PageInfo<Spec> pageInfo = specService.findPage(page, size);
35          return new Result<PageInfo>(true, StatusCode.OK, "查询成
36      功", pageInfo);
37      }
38
39      /**
40       * 多条件搜索品牌数据
41       * @param spec
42       * @return
43       */
44      @PostMapping(value = "/search" )
45      public Result<List<Spec>> findList(@RequestBody(required = false)
46      Spec spec){
47          List<Spec> list = specService.findList(spec);
48          return new Result<List<Spec>>(true, StatusCode.OK, "查询成功", list);
49      }
50
51      /**
52       * 根据ID删除品牌数据
53       * @param id
```

```

50     * @return
51     */
52     @DeleteMapping(value =("/{id}") )
53     public Result delete(@PathVariable Integer id){
54         specService.delete(id);
55         return new Result(true,StatusCode.OK,"删除成功");
56     }
57
58     /**
59     * 修改Spec数据
60     * @param spec
61     * @param id
62     * @return
63     */
64     @PutMapping(value="/{id}")
65     public Result update(@RequestBody Spec spec,@PathVariable Integer id)
66 {
67     //设置主键值
68     spec.setId(id);
69     //修改数据
70     specService.update(spec);
71     return new Result(true,StatusCode.OK,"修改成功");
72 }
73
74 /**
75 * 新增Spec数据
76 * @param spec
77 * @return
78 */
79 @PostMapping
80 public Result add(@RequestBody Spec spec){
81     specService.add(spec);
82     return new Result(true,StatusCode.OK,"添加成功");
83 }
84
85 /**
86 * 根据ID查询Spec数据
87 * @param id
88 * @return
89 */
90 @GetMapping("/{id}")
91 public Result<Spec> findById(@PathVariable Integer id){
92     //根据ID查询
93     Spec spec = specService.findById(id);
94     return new Result<Spec>(true,StatusCode.OK,"查询成功",spec);
95 }
96
97 /**
98 * 查询Spec全部数据
99 * @return
100 */
101 @GetMapping
102 public Result<Spec> findAll(){
103     List<Spec> list = specService.findAll();
104     return new Result<Spec>(true, StatusCode.OK,"查询成功",list) ;
105 }
106 }

```

Swagger测试：

SpecController SpecController	
GET	/spec 查询所有Spec
POST	/spec 添加Spec
DELETE	/spec/{id} 根据ID删除Spec
PUT	/spec/{id} 根据ID修改Spec
GET	/spec/{id} 根据ID查询Spec
POST	/spec/search 不带分页条件搜索Spec
GET	/spec/search/{page}/{size} 分页列表查询Spec
POST	/spec/search/{page}/{size} 分页条件搜索Spec

## 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：

```
1  @Table(name="tb_para")
2  public class Para implements Serializable{
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      @Column(name = "id")
7      private Integer id;//id
8
9      @Column(name = "name")
10     private String name;//名称
11
12     @Column(name = "options")
13     private String options;//选项
14
15     @Column(name = "seq")
16     private Integer seq;//排序
17
18     @Column(name = "template_id")
19     private Integer templateId;//模板ID
20     //get..set..toString
21 }
```

### 3.5.2 Dao

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

```
1  public interface ParaMapper extends Mapper<Para> {
2  }
```

### 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：

```
1 public interface ParaService {
2
3     /**
4      * Para多条件分页查询
5      * @param para
6      * @param page
7      * @param size
8      * @return
9      */
10    PageInfo<Para> findPage(Para para, int page, int size);
11
12    /**
13     * Para分页查询
14     * @param page
15     * @param size
16     * @return
17     */
18    PageInfo<Para> findPage(int page, int size);
19
20    /**
21     * Para多条件搜索方法
22     * @param para
23     * @return
24     */
25    List<Para> findList(Para para);
26
27    /**
28     * 删除Para
29     * @param id
30     */
31    void delete(Integer id);
32
33    /**
34     * 修改Para数据
35     * @param para
36     */
37    void update(Para para);
38
39    /**
40     * 新增Para
41     * @param para
42     */
43    void add(Para para);
44
45    /**
46     * 根据ID查询Para
47     * @param id
48     * @return
49     */
```

```

50     Para findById(Integer id);
51
52     /**
53      * 查询所有Para
54      * @return
55      */
56     List<Para> findAll();
57 }

```

## (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:

```

1  @Service
2  public class ParaServiceImpl implements ParaService {
3
4      @Autowired
5      private ParaMapper paraMapper;
6
7      @Autowired
8      private TemplateMapper templateMapper;
9
10     /**
11      * Para条件+分页查询
12      * @param para 查询条件
13      * @param page 页码
14      * @param size 页大小
15      * @return 分页结果
16      */
17     @Override
18     public PageInfo<Para> findPage(Para para, int page, int size){
19         //分页
20         PageHelper.startPage(page,size);
21         //搜索条件构建
22         Example example = createExample(para);
23         //执行搜索
24         return new PageInfo<Para>(paraMapper.selectByExample(example));
25     }
26
27     /**
28      * Para分页查询
29      * @param page
30      * @param size
31      * @return
32      */
33     @Override
34     public PageInfo<Para> findPage(int page, int size){
35         //静态分页
36         PageHelper.startPage(page,size);
37         //分页查询
38         return new PageInfo<Para>(paraMapper.selectAll());
39     }
40
41     /**

```

```

42     * Para条件查询
43     * @param para
44     * @return
45     */
46     @Override
47     public List<Para> findList(Para para){
48         //构建查询条件
49         Example example = createExample(para);
50         //根据构建的条件查询数据
51         return paraMapper.selectByExample(example);
52     }
53
54
55     /**
56     * Para构建查询对象
57     * @param para
58     * @return
59     */
60     public Example createExample(Para para){
61         Example example=new Example(Para.class);
62         Example.Criteria criteria = example.createCriteria();
63         if(para!=null){
64             // id
65             if(!StringUtils.isEmpty(para.getId())){
66                 criteria.andEqualTo("id",para.getId());
67             }
68             // 名称
69             if(!StringUtils.isEmpty(para.getName())){
70                 criteria.andLike("name", "%"+para.getName()+"%");
71             }
72             // 选项
73             if(!StringUtils.isEmpty(para.getOptions())){
74                 criteria.andEqualTo("options",para.getOptions());
75             }
76             // 排序
77             if(!StringUtils.isEmpty(para.getSeq())){
78                 criteria.andEqualTo("seq",para.getSeq());
79             }
80             // 模板ID
81             if(!StringUtils.isEmpty(para.getTemplateId())){
82
83                 criteria.andEqualTo("templateId",para.getTemplateId());
84             }
85             return example;
86         }
87
88     /**
89     * 删除
90     * @param id
91     */
92     @Override
93     public void delete(Integer id){
94         //根据ID查询
95         Para para = paraMapper.selectByPrimaryKey(id);
96         //修改模板统计数据
97         updateParaNum(para,-1);
98

```

```
99         paramMapper.deleteByPrimaryKey(id);
100     }
101
102     /**
103      * 修改Para
104      * @param para
105      */
106     @Override
107     public void update(Para para){
108         paramMapper.updateByPrimaryKey(para);
109     }
110
111     /**
112      * 增加Para
113      * @param para
114      */
115     @Override
116     public void add(Para para){
117         paramMapper.insert(para);
118
119         //修改模板统计数据
120         updateParaNum(para,1);
121     }
122
123     /**
124      * 根据ID查询Para
125      * @param id
126      * @return
127      */
128     @Override
129     public Para findById(Integer id){
130         return paramMapper.selectByPrimaryKey(id);
131     }
132
133     /**
134      * 查询Para全部数据
135      * @return
136      */
137     @Override
138     public List<Para> findAll() {
139         return paramMapper.selectAll();
140     }
141
142     /**
143      * 修改模板统计数据
144      * @param para:操作的参数
145      * @param count:变更的数量
146      */
147     public void updateParaNum(Para para, int count){
148         //修改模板数量统计
149         Template template =
150         templateMapper.selectByPrimaryKey(para.getTemplateId());
151         template.setParaNum(template.getParaNum()+count);
152         templateMapper.updateByPrimaryKeySelective(template);
153     }
```

### 3.5.4 控制层

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

```
1  @RestController
2  @RequestMapping("/para")
3  @CrossOrigin
4  public class ParaController {
5
6      @Autowired
7      private ParaService paraService;
8
9      /**
10       * Para分页条件搜索实现
11       * @param para
12       * @param page
13       * @param size
14       * @return
15       */
16      @PostMapping(value = "/search/{page}/{size}" )
17      public Result<PageInfo> findPage(@RequestBody(required = false) Para
18      para, @PathVariable int page, @PathVariable int size){
19          //执行搜索
20          PageInfo<Para> pageInfo = paraService.findPage(para, page, size);
21          return new Result(true,StatusCode.OK,"查询成功",pageInfo);
22      }
23
24      /**
25       * Para分页搜索实现
26       * @param page:当前页
27       * @param size:每页显示多少条
28       * @return
29       */
30      @GetMapping(value = "/search/{page}/{size}" )
31      public Result<PageInfo> findPage(@PathVariable int page,
32      @PathVariable int size){
33          //分页查询
34          PageInfo<Para> pageInfo = paraService.findPage(page, size);
35          return new Result<PageInfo>(true,StatusCode.OK,"查询成功",pageInfo);
36      }
37
38      /**
39       * 多条件搜索品牌数据
40       * @param para
41       * @return
42       */
43      @PostMapping(value = "/search" )
44      public Result<List<Para>> findList(@RequestBody(required = false)
45      Para para){
46          List<Para> list = paraService.findList(para);
47          return new Result<List<Para>>(true,StatusCode.OK,"查询成功",list);
48      }
49
50      /**
```



```

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 Integer id){
54         paraService.delete(id);
55         return new Result(true,StatusCode.OK,"删除成功");
56     }
57
58     /**
59     * 修改Para数据
60     * @param para
61     * @param id
62     * @return
63     */
64     @PutMapping(value="/{id}")
65     public Result update(@RequestBody Para para,@PathVariable Integer id)
66 {
67     //设置主键值
68     para.setId(id);
69     //修改数据
70     paraService.update(para);
71     return new Result(true,StatusCode.OK,"修改成功");
72 }
73
74 /**
75 * 新增Para数据
76 * @param para
77 * @return
78 */
79 @PostMapping
80 public Result add(@RequestBody Para para){
81     paraService.add(para);
82     return new Result(true,StatusCode.OK,"添加成功");
83 }
84
85 /**
86 * 根据ID查询Para数据
87 * @param id
88 * @return
89 */
90 @GetMapping("/{id}")
91 public Result<Para> findById(@PathVariable Integer id){
92     //根据ID查询
93     Para para = paraService.findById(id);
94     return new Result<Para>(true,StatusCode.OK,"查询成功",para);
95 }
96
97 /**
98 * 查询Para全部数据
99 * @return
100 */
101 @GetMapping
102 public Result<Para> findAll(){
103     List<Para> list = paraService.findAll();
104     return new Result<Para>(true, StatusCode.OK,"查询成功",list) ;

```

Swagger测试:

ParaController ParaController	
GET	/para 查询所有Para
POST	/para 添加Para
DELETE	/para/{id} 根据ID删除Para
PUT	/para/{id} 根据ID修改Para
GET	/para/{id} 根据ID查询Para
POST	/para/search 不带分页条件搜索Para
GET	/para/search/{page}/{size} 分页列表查询Para
POST	/para/search/{page}/{size} 分页条件搜索Para

## 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:

```

1  @Table(name="tb_category")
2  public class Category implements Serializable{
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      @Column(name = "id")
7      private Integer id;//分类ID
8
9      @Column(name = "name")
10     private String name;//分类名称
11
12     @Column(name = "goods_num")
13     private Integer goodsNum;//商品数量
14
15     @Column(name = "is_show")
16     private String isShow;//是否显示
17
18     @Column(name = "is_menu")
19     private String isMenu;//是否导航
20
21     @Column(name = "seq")
22     private Integer seq;//排序
23
24     @Column(name = "parent_id")
25     private Integer parentId;//上级ID
26
27     @Column(name = "template_id")
28     private Integer templateId;//模板ID
29     //..set..get..toString
30 }

```

### 4.3.2 Dao

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

```
1 public interface CategoryMapper extends Mapper<Category> {  
2 }
```

### 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：

```
1 public interface CategoryService {  
2  
3     /**  
4      * Category多条件分页查询  
5      * @param category  
6      * @param page  
7      * @param size  
8      * @return  
9      */  
10    PageInfo<Category> findPage(Category category, int page, int size);  
11  
12    /**  
13     * Category分页查询  
14     * @param page  
15     * @param size  
16     * @return  
17     */  
18    PageInfo<Category> findPage(int page, int size);  
19  
20    /**  
21     * Category多条件搜索方法  
22     * @param category  
23     * @return  
24     */  
25    List<Category> findList(Category category);  
26  
27    /**  
28     * 删除Category  
29     * @param id  
30     */  
31    void delete(Integer id);  
32  
33    /**  
34     * 修改Category数据  
35     * @param category  
36     */  
37    void update(Category category);  
38  
39    /**
```

```

40     * 新增Category
41     * @param category
42     */
43     void add(Category category);
44
45     /**
46     * 根据ID查询Category
47     * @param id
48     * @return
49     */
50     Category findById(Integer id);
51
52     /**
53     * 查询所有Category
54     * @return
55     */
56     List<Category> findAll();
57
58     /**
59     * 根据父节点ID查询
60     * @param pid:父节点ID
61     */
62     List<Category> findByParentId(Integer pid);
63 }

```

## (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：

```

1  @Service
2  public class CategoryServiceImpl implements CategoryService {
3
4      @Autowired
5      private CategoryMapper categoryMapper;
6
7
8      /**
9      * Category条件+分页查询
10     * @param category 查询条件
11     * @param page 页码
12     * @param size 页大小
13     * @return 分页结果
14     */
15     @Override
16     public PageInfo<Category> findPage(Category category, int page, int
size){
17         //分页
18         PageHelper.startPage(page,size);
19         //搜索条件构建
20         Example example = createExample(category);
21         //执行搜索
22         return new PageInfo<Category>
(categoryMapper.selectByExample(example));
23     }

```

```

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
73                 criteria.andEqualTo("goodsNum",category.getGoodsNum());
74             }
75             // 是否显示
76             if(!StringUtils.isEmpty(category.getIsShow())){
77                 criteria.andEqualTo("isShow",category.getIsShow());
78             }
79             // 是否导航
80             if(!StringUtils.isEmpty(category.getIsMenu())){
81                 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
89             criteria.andEqualTo("parentId",category.getParentId());
90         }
91         // 模板ID
92         if(!StringUtils.isEmpty(category.getTemplateId())){
93             criteria.andEqualTo("templateId",category.getTemplateId());
94         }
95         }
96         return example;
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

```

```

137     * @return
138     */
139     @Override
140     public List<Category> findAll() {
141         return categoryMapper.selectAll();
142     }
143
144     /**
145      * 根据父节点ID查询
146      * @param pid:父节点ID
147      */
148     @Override
149     public List<Category> findByParentId(Integer pid) {
150         Category category = new Category();
151         category.setParentId(pid);
152         return categoryMapper.select(category);
153     }
154 }

```

#### 4.3.4 控制层

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

```

1  @RestController
2  @RequestMapping("/category")
3  @CrossOrigin
4  public class CategoryController {
5
6      @Autowired
7      private CategoryService categoryService;
8
9      /**
10       * Category分页条件搜索实现
11       * @param category
12       * @param page
13       * @param size
14       * @return
15       */
16      @PostMapping(value = "/search/{page}/{size}" )
17      public Result<PageInfo> findPage(@RequestBody(required = false)
18      Category category, @PathVariable int page, @PathVariable int size){
19          //执行搜索
20          PageInfo<Category> pageInfo = categoryService.findPage(category,
21          page, size);
22          return new Result(true,StatusCode.OK,"查询成功",pageInfo);
23      }
24
25      /**
26       * Category分页搜索实现
27       * @param page:当前页
28       * @param size:每页显示多少条
29       * @return
30       */
31      @GetMapping(value = "/search/{page}/{size}" )

```



```

30     public Result<PageInfo> findPage(@PathVariable int page,
    @PathVariable int size){
31         //分页查询
32         PageInfo<Category> pageInfo = categoryService.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 category
39      * @return
40      */
41     @PostMapping(value = "/search" )
42     public Result<List<Category>> findList(@RequestBody(required = false)
Category category){
43         List<Category> list = categoryService.findList(category);
44         return new Result<List<Category>>(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 Integer id){
54         categoryService.delete(id);
55         return new Result(true,StatusCode.OK,"删除成功");
56     }
57
58     /**
59      * 修改Category数据
60      * @param category
61      * @param id
62      * @return
63      */
64     @PutMapping(value="/{id}")
65     public Result update(@RequestBody Category category,@PathVariable
Integer id){
66         //设置主键值
67         category.setId(id);
68         //修改数据
69         categoryService.update(category);
70         return new Result(true,StatusCode.OK,"修改成功");
71     }
72
73     /**
74      * 新增Category数据
75      * @param category
76      * @return
77      */
78     @PostMapping
79     public Result add(@RequestBody Category category){
80         categoryService.add(category);
81         return new Result(true,StatusCode.OK,"添加成功");

```

```

82     }
83
84     /**
85      * 根据ID查询Category数据
86      * @param id
87      * @return
88      */
89     @GetMapping("/{id}")
90     public Result<Category> findById(@PathVariable Integer id){
91         //根据ID查询
92         Category category = categoryService.findById(id);
93         return new Result<Category>(true,StatusCode.OK,"查询成功",category);
94     }
95
96     /**
97      * 查询Category全部数据
98      * @return
99      */
100    @GetMapping
101    public Result<Category> findAll(){
102        List<Category> list = categoryService.findAll();
103        return new Result<Category>(true, StatusCode.OK,"查询成功",list) ;
104    }
105
106    /**
107     * 根据父ID查询
108     */
109    @RequestMapping(value = "/list/{pid}")
110    public Result<Category> findByPrantId(@PathVariable(value = "pid")Integer pid){
111        //根据父节点ID查询
112        List<Category> list = categoryService.findByParentId(pid);
113        return new Result<Category>(true,StatusCode.OK,"查询成功",list);
114    }
115 }

```

Swagger测试:

CategoryController CategoryController	
GET	/category 查询所有Category
POST	/category 添加Category
DELETE	/category/{id} 根据ID删除Category
PUT	/category/{id} 根据ID修改Category
GET	/category/{id} 根据ID查询Category
POST	/category/search 不带分页条件搜索Category
GET	/category/search/{page}/{size} 分页列表查询Category
POST	/category/search/{page}/{size} 分页条件搜索Category

## 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