**畅购电商系统开发**

**第2天**

传智播客.黑马程序员.深圳

# 学习目标

* 理解FastDFS工作流程
* 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)

# FastDFS

## FastDFS简介

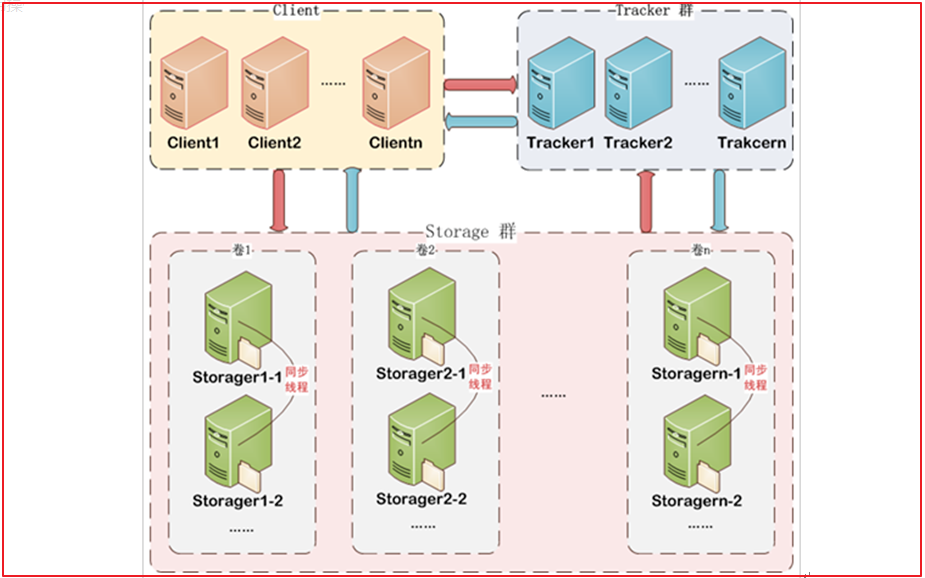
### FastDFS体系结构

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 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。

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**。



### 上传流程(时序图)



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。

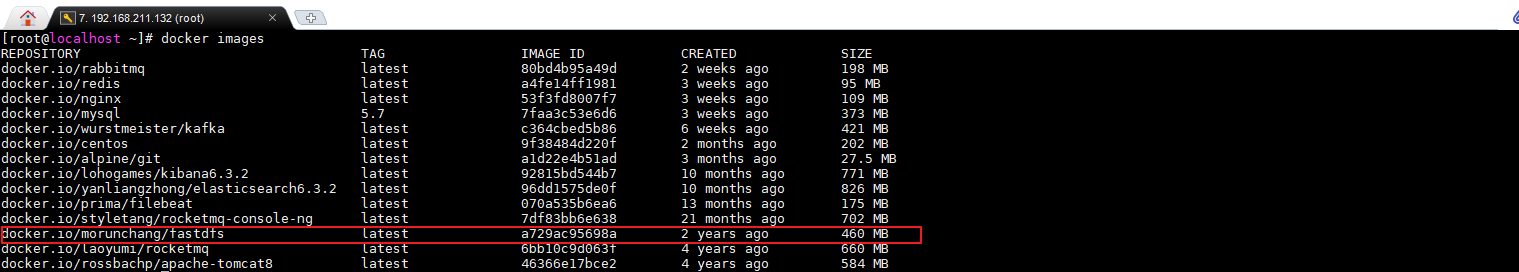


* **组名**：文件上传后所在的storage组名称，在文件上传成功后有storage服务器返回，需要客户端自行保存。
* **虚拟磁盘路径**：storage配置的虚拟路径，与磁盘选项store\_path\*对应。如果配置了store\_path0则是M00，如果配置了store\_path1则是M01，以此类推。
* 数据两级目录：storage服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。两级目录的范围都是 00~FF，备注(16进制中FF等于255，加上从00开始，也就是说第一级目录可以放255+1个目录，二级目录可放个数为：256\*256 =65,536个文件夹目录，为保证文件访问速度，粗略按照一个目录2,000个文件存放<00-FF>总共可保存文件: 131,072,000个)。
* 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。

## FastDFS搭建

### 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：



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, 192.168.211.132是宿主机的IP
* 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

开启开机启动设置，让tracker与storage随linux启动而启动

docker **update** --restart=always tracker

docker **update** --restart=always storage

### 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 {

root /data/fast\_data/data;

ngx\_fastdfs\_module;

}

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

## 文件存储微服务

### 创建工程与依赖

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。

*<?***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**>  
  
 *<!--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**>

### FastDFS配置

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

*# 连接超时时间，单位为秒。*connect\_timeout=**60***# 通信超时时间，单位为秒。发送或接收数据时。假设在超时时间后还不能发送或接收数据，则本次网络通信失败*network\_timeout=**60***# 字符集*charset=**UTF-8***# tracker的http端口*http.tracker\_http\_port=**8080***# tracker服务器TCP的IP和端口设置*tracker\_server=**192.168.211.132:22122**

### application.yml配置

在resources文件夹下创建application.yml

**spring**:  
 **servlet**:  
 **multipart**:  
 *# 单个文件大小* **max-file-size**: 10MB  
 *# 总上传的数据大小，除了文件外还有附加表单类参数* **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  
feign**:  
 **hystrix**:  
 **enabled**: **true**

### 启动引导类

创建启动类com.changgou.FileApplication

*/\*\*  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou  
 \* exclude：排除加载的自动配置  
 \*/*@SpringBootApplication(exclude = DataSourceAutoConfiguration.**class**)  
@EnableEurekaClient  
**public class** FileApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(FileApplication.**class**, args);  
 }  
}

## 文件上传

### 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。

创建com.changgou.file.FastDFSFile代码如下：

*/\*\*  
 \* 文件上传内容对象  
 \*/***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*}

### 文件操作

创建com.changgou.util.FastDFSClient类,在该类中实现FastDFS信息获取以及文件上传、文件下载、文件删除的相关操作。

FastDFS的核心对象有两个分别是：TrackerServer、StorageClient。我们先创建好这两个对象，后续直接使用。

API完整步骤:

//1、获取配置文件路径-filePath = new ClassPathResource("fdfs\_client.conf").getPath()

//2、加载配置文件-ClientGlobal.init(配置文件路径)

//3、创建一个TrackerClient对象。直接new一个。

//4、使用TrackerClient对象创建连接，getConnection获得一个TrackerServer对象。

//5、创建一个StorageClient对象，直接new一个，需要两个参数TrackerServer对象、null

#### 初始化Tracker信息

在com.changgou.util.FastDFSClient类中初始化FastDFS信息,在类中添加静态初始化块，实现加载配置：

*/\*\*  
 \* FastDFS操作工具  
 \* 实现信息获取、文件上传、文件下载、文件删除的相关操作  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.util  
 \*/***public class** FastDFSClient {  
 *//初始化Tracker配置信息* **static** {  
 **try** {  
 *//1、获取配置文件路径-filePath = new ClassPathResource("fdfs\_client.conf").getPath()* String filePath = **new** ClassPathResource(**"fdfs\_client.conf"**).getPath();  
 *//2、加载配置文件-ClientGlobal.init(配置文件路径)* ClientGlobal.*init*(filePath);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

#### 创建TrackerServer对象

在com.changgou.util.FastDFSClient类中添加获取TrackerServer方法。

*/\*\*  
 \* 创建TrackerServer对象  
 \** ***@return*** *TrackerServer  
 \*/***public static** TrackerServer getTrackerServer(){  
 TrackerServer trackerServer = **null**;  
 **try** {  
 *//3、创建一个TrackerClient对象。直接new一个。* TrackerClient trackerClient = **new** TrackerClient();  
 *//4、使用TrackerClient对象创建连接，getConnection获得一个TrackerServer对象。* trackerServer = trackerClient.getConnection();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** trackerServer;  
}

#### 创建StorageClient对象

在com.changgou.util.FastDFSClient类中添加获取StorageClient方法。

*/\*\*  
 \* 创建StorageClient对象  
 \** ***@return*** *StorageClient  
 \*/***public static** StorageClient getStorageClient(){  
 *//5、创建一个StorageClient对象，直接new一个，需要两个参数TrackerServer对象、null* StorageClient storageClient = **new** StorageClient(*getTrackerServer*(),**null**);  
 **return** storageClient;  
}

#### 文件上传

##### FastDFS上传文件

*/\*\*  
 \* 文件上传  
 \** ***@param fastDFSFile*** *上传包装对象  
 \** ***@return*** *String[存储的组名，存储的路径]  
 \*/***public static** String[] upload(FastDFSFile fastDFSFile){  
 String[] uploadResult = **null**;  
 **try** {  
 *//附加参数* NameValuePair[] meta\_list = **new** NameValuePair[1];  
 *//文件作者* meta\_list[0] = **new** NameValuePair(**"author"**,fastDFSFile.getAuthor());  
 *//上传文件  
 //upload\_file(文件字节数组，文件扩展名,附加参数)* uploadResult = *getStorageClient*().upload\_file(fastDFSFile.getContent(), fastDFSFile.getExt(), meta\_list);  
 **return** uploadResult;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** uploadResult;  
}

##### SpringMVC接收用户上传文件

@RestController  
@CrossOrigin  
**public class** FileController {  
 @RequestMapping(**"upload"**)  
 **public** Result upload(MultipartFile file) **throws** IOException {  
 *//1、包装文件上传对象* FastDFSFile dfsFile = **new** FastDFSFile(  
 file.getOriginalFilename(), *//文件原来的名字* file.getBytes(), *//文件字节数组* StringUtils.*getFilenameExtension*(file.getOriginalFilename()) *//获取文件后缀名* );  
 *//2、调用FastDFS上传文件* String[] upload = FastDFSClient.*upload*(dfsFile);  
 *//3、返回上传结果* String url = **"http://192.168.211.132:8080/"** + upload[0] + **"/"** + upload[1];  
 **return new** Result(**true**, StatusCode.***OK***, url);  
 }  
}

##### Postman测试文件上传

1、选择post请求方式，输入请求地址 <http://localhost:18082/upload>

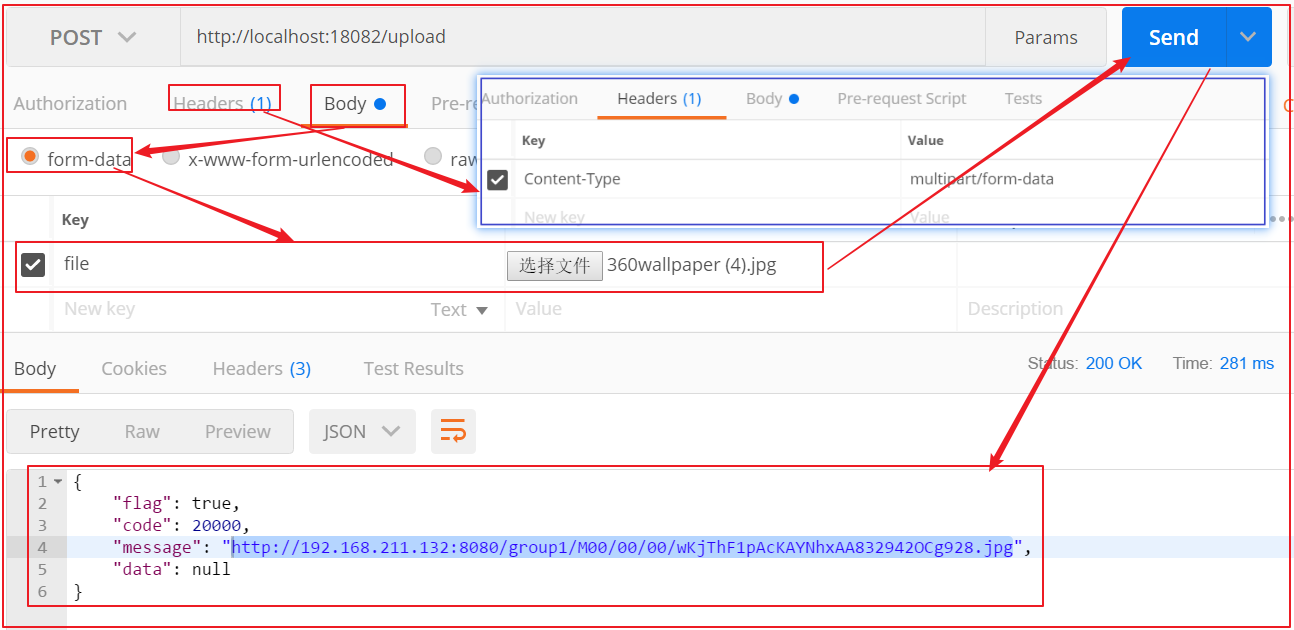
2、填写Headers(可以不填写)

Key：Content-Type

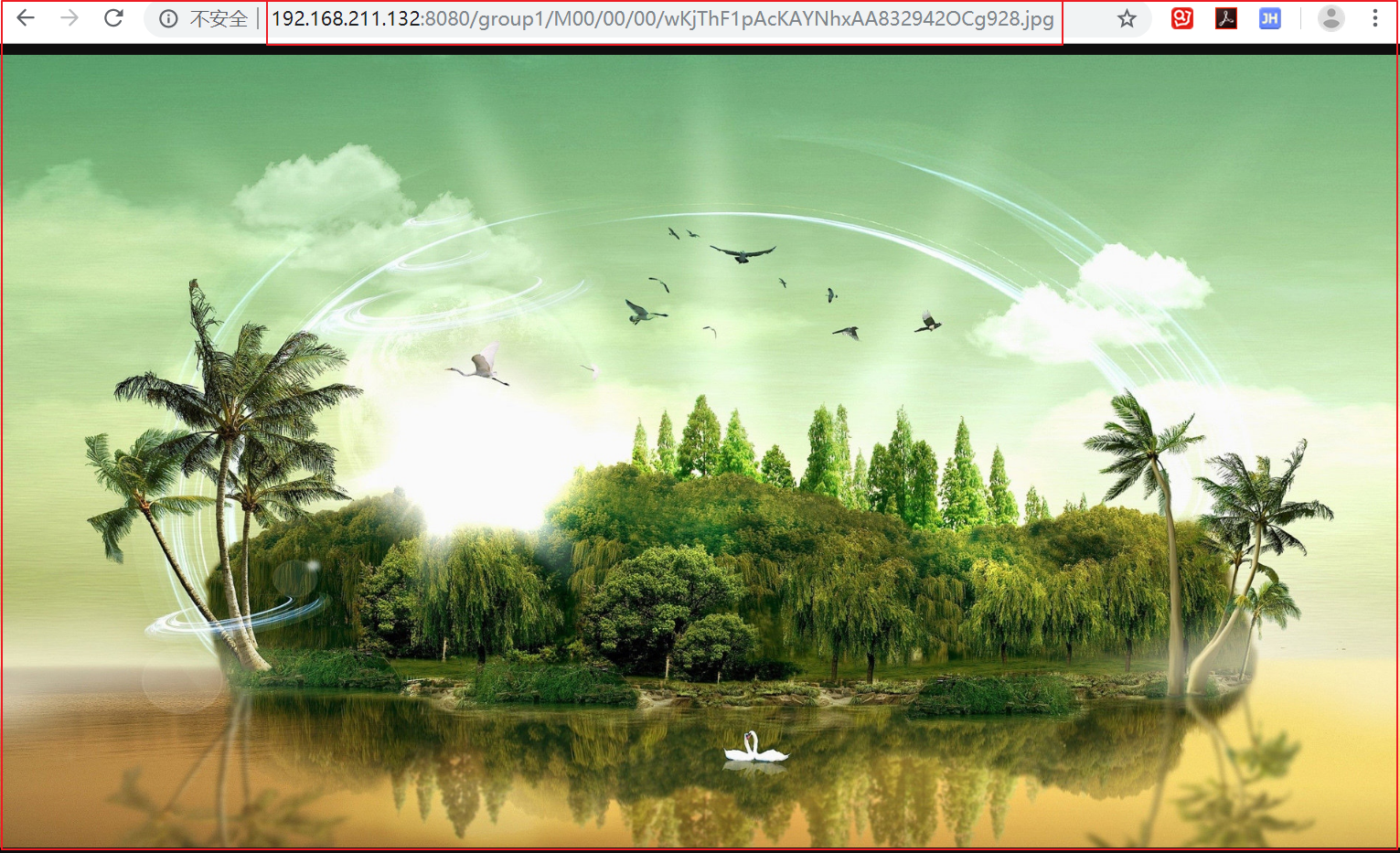
Value：multipart/form-data

3、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/wKjThF1pAcKAYNhxAA832942OCg928.jpg如下图



路径解析

<http://192.168.211.132:8080/group1/M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg>

192.168.211.132：虚拟机的路径

8080：nginx的端口 nginx\_fastdfs\_module这个模块包含了nginx访问tracker的功能：访问22122端口

/group1/M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg：存储的虚拟路径和文件夹

tracker\_server=192.168.211.132:22122：tracker的地址

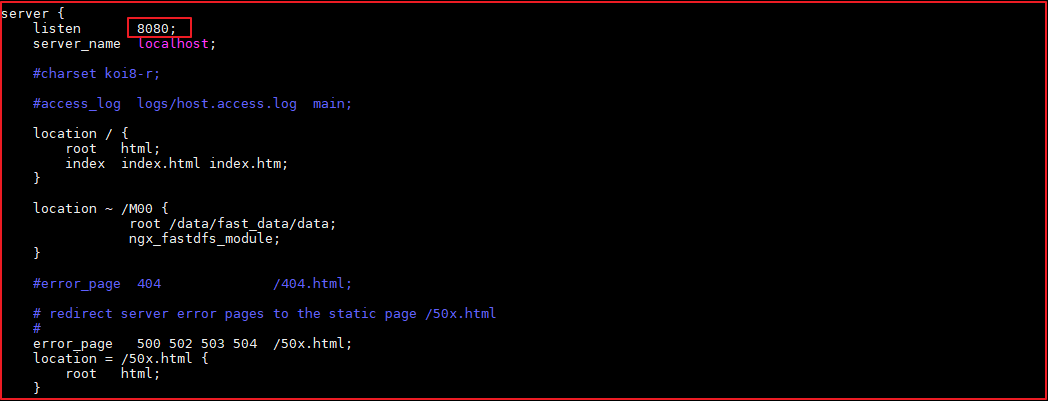
##### 服务器存储信息介绍

**1)访问端口8080问题**

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

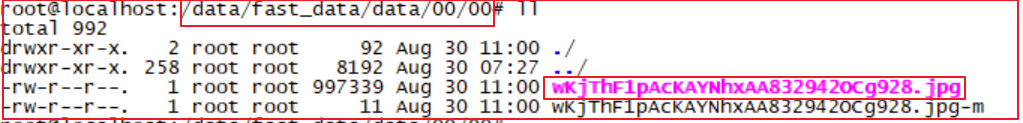
**vim** /etc/nginx/**conf**/nginx.**conf**



修改后重启storage即可根据自己修改的端口访问图片了。

**2)文件存储位置**

可以查询**storage容器**的目录：vim /etc/fdfs/storage.conf其中的 store\_path0=/data/fast\_data 就是我们文件存储的目录。

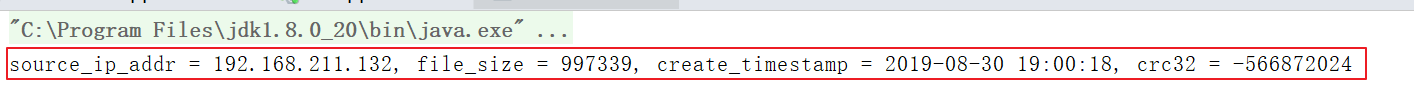


#### 获取文件信息

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  
 \** ***@return*** *\*/***public static** FileInfo getFileInfo(String groupName, String remoteFilename){  
 FileInfo info = **null**;  
 **try** {  
 *//获取文件信息* info = *getStorageClient*().get\_file\_info(groupName, remoteFilename);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** info;  
}  
  
*//测试方法***public static void** main(String[] args) {  
 *//group1/M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg* FileInfo info = *getFileInfo*(**"group1"**, **"M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg"**);  
 System.***out***.println(info);  
}

运行main函数测试，得到结果



#### 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 downloadFile(String groupName, String remoteFilename){  
 InputStream inputStream = **null**;  
 **try** {  
 *//下载文件* **byte**[] bytes = *getStorageClient*().download\_file(groupName, remoteFilename);  
 *//把字节数组转换为字节输入流* inputStream = **new** ByteArrayInputStream(bytes);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** inputStream;  
}  
  
*//测试方法***public static void** main(String[] args) {  
 *//获取文件信息-测试  
 //group1/M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg  
 /\*FileInfo info = getFileInfo("group1", "M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg");  
 System.out.println(info);\*/* **try** {  
 *//文件下载-测试* InputStream is = *downloadFile*(**"group1"**, **"M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg"**);  
 *//定义输出流对象* OutputStream out = **new** FileOutputStream(**"D:/1.jpg"**);  
 *//定义缓冲区* **byte**[] buff = **new byte**[1024];  
 *//读取输入流* **while** ((is.read(buff) > -1)) {  
 out.write(buff);  
 }  
 *//释放资源* out.close();  
 is.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
}

#### 文件删除实现

*/\*\*  
 \* 文件删除  
 \** ***@param groupName*** *组名  
 \** ***@param remoteFilename*** *文件存储完整名  
 \** ***@return*** *\*/***public static void** deleteFile(String groupName, String remoteFilename){  
 InputStream inputStream = **null**;  
 **try** {  
 *//文件删除  
 getStorageClient*().delete\_file(groupName, remoteFilename);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
}  
*//测试方法***public static void** main(String[] args) {  
 *//省略其它测试方法…………   
  
 //文件删除  
 deleteFile*(**"group1"**, **"M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg"**);  
  
}

(1)删除文件后可以查看storage容器文件目录，发现文件已被删除。

(2)访问<http://192.168.211.132:8080/group1/M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg>发现还可以看到图片，是因为缓存问题，解决方法：

* 1. 清除浏览器缓存即可
  2. 修改nginx配置，禁用缓存，修改配置后，记得重启容器

**vi** /etc/nginx/**conf**/nginx.**conf**

location ~ /M00 {

add\_header Cache-Control no-store;

root /data/fast\_data/data;

ngx\_fastdfs\_module;

}

#### 获取组(Storage)信息

*/\*\*  
 \* 获取组(Storage)信息  
 \** ***@param groupName*** *组名  
 \** ***@return*** *\*/***public static** StorageServer getStorageServer(String groupName){  
 **try** {  
 *//3、创建一个TrackerClient对象。直接new一个。* TrackerClient trackerClient = **new** TrackerClient();  
 *//4、使用TrackerClient对象创建连接，getConnection获得一个TrackerServer对象。* TrackerServer trackerServer = trackerClient.getConnection();  
 *//获取信息* StorageServer storageServer = trackerClient.getStoreStorage(trackerServer, groupName);  
 **return** storageServer;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
}  
  
*//测试方法***public static void** main(String[] args) {  
 *//省略其它测试方法…  
   
 //获取组(Storage)信息* StorageServer storageServer = *getStorageServer*(**"group1"**);  
 System.***out***.println(**"store的下标："** + storageServer.getStorePathIndex());  
 System.***out***.println(**"store的ip与端口："** + storageServer.getInetSocketAddress());  
}

#### 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  
 \** ***@return*** *\*/***public static** ServerInfo[] getServerInfo(String groupName, String remoteFilename){  
 **try** {  
 *//3、创建一个TrackerClient对象。直接new一个。* TrackerClient trackerClient = **new** TrackerClient();  
 *//4、使用TrackerClient对象创建连接，getConnection获得一个TrackerServer对象。* TrackerServer trackerServer = trackerClient.getConnection();  
 *//获取信息Storage服务的IP、端口信息* ServerInfo[] infos = trackerClient.getFetchStorages(trackerServer, groupName, remoteFilename);  
 **return** infos;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
}  
  
*//测试方法***public static void** main(String[] args) {  
 *//省略其它测试方法………  
 //根据文件组名和文件存储路径获取Storage服务的IP、端口信息* ServerInfo[] infos = *getServerInfo*(**"group1"**, **"M00/00/00/wKjThF1pAcKAYNhxAA832942OCg928.jpg"**);  
 **for** (ServerInfo info : infos) {  
 System.***out***.println(info.getIpAddr() + **":"** + info.getPort());  
 }  
}

#### 获取Tracker服务地址

*/\*\*  
 \* 获取Tracker服务地址  
 \** ***@return*** *\*/***public static** String getTrackerUrl(){  
 **try** {  
 *//3、创建一个TrackerClient对象。直接new一个。* TrackerClient trackerClient = **new** TrackerClient();  
 *//4、使用TrackerClient对象创建连接，getConnection获得一个TrackerServer对象。* TrackerServer trackerServer = trackerClient.getConnection();  
 *//拼接TrackerUrl* String url = **"http://"** + trackerServer.getInetSocketAddress().getHostString() + **":"** + ClientGlobal.*getG\_tracker\_http\_port*() + **"/"**;  
 **return** url;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
}

FileController.upload方法逻辑实现动态地址获取

*/\*\*  
 \* 文件上传控制器  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.controller  
 \*/*@RestController  
@CrossOrigin  
**public class** FileController {  
 @RequestMapping(**"upload"**)  
 **public** Result upload(MultipartFile file) **throws** IOException {  
 *//1、包装文件上传对象* FastDFSFile dfsFile = **new** FastDFSFile(  
 file.getOriginalFilename(), *//文件原来的名字* file.getBytes(), *//文件字节数组* StringUtils.*getFilenameExtension*(file.getOriginalFilename()) *//获取文件后缀名* );  
 *//2、调用FastDFS上传文件* String[] upload = FastDFSClient.*upload*(dfsFile);  
 *//3、返回上传结果  
 //String url = "http://192.168.211.132:8080/" + upload[0] + "/" + upload[1];* String url = FastDFSClient.*getTrackerUrl*() + upload[0] + **"/"** + upload[1];  
 **return new** Result(**true**, StatusCode.***OK***, url);  
 }  
}

重新通过PostMan上传一张图片测试。

# [相册管理（实战）](file:///D:\\BaiduNetdiskDownload\\%E8%AF%BE%E7%A8%8B%E4%BC%98%E5%8C%96\\%E8%AE%B2%E4%B9%89\\day02\\%E8%AE%B2%E4%B9%89\\%E7%AC%AC2%E5%A4%A9.html" \l "2-相册管理-实战-)

## 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

## 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"

}

]

## 代码实现

### 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..

}

### 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**> {

}

### 业务层

#### 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**();

}

#### 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

**public** List<Album> **findList**(Album album){

//构建查询条件

Example example = createExample(album);

//根据构建的条件查询数据

**return** albumMapper.selectByExample(example);

}

/\*\*

\* Album构建查询对象

\* @param album

\* @return

\*/

**public** Example **createExample**(Album album){

Example example=**new** Example(Album.class);

Example.Criteria criteria = example.createCriteria();

**if**(album!=**null**){

// 编号

**if**(!StringUtils.isEmpty(album.getId())){

criteria.andEqualTo("id",album.getId());

}

// 相册名称

**if**(!StringUtils.isEmpty(album.getTitle())){

criteria.andLike("title","%"+album.getTitle()+"%");

}

// 相册封面

**if**(!StringUtils.isEmpty(album.getImage())){

criteria.andEqualTo("image",album.getImage());

}

// 图片列表

**if**(!StringUtils.isEmpty(album.getImageItems())){

criteria.andEqualTo("imageItems",album.getImageItems());

}

}

**return** example;

}

/\*\*

\* 删除

\* @param id

\*/

@Override

**public** **void** **delete**(Long id){

albumMapper.deleteByPrimaryKey(id);

}

/\*\*

\* 修改Album

\* @param album

\*/

@Override

**public** **void** **update**(Album album){

albumMapper.updateByPrimaryKey(album);

}

/\*\*

\* 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();

}

}

### 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);

}

/\*\*\*

\* Album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<Album> pageInfo = albumService.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 album

\* @return

\*/

@PostMapping(value = "/search" )

**public** Result<List<Album>> findList(@RequestBody(required = **false**) Album album){

List<Album> list = albumService.findList(album);

**return** **new** Result<List<Album>>(**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 Long id){

albumService.delete(id);

**return** **new** **Result**(**true**,StatusCode.OK,"删除成功");

}

/\*\*\*

\* 修改Album数据

\* @param album

\* @param id

\* @return

\*/

@PutMapping(value="/{id}")

**public** Result **update**(@RequestBody Album album,@PathVariable Long id){

//设置主键值

album.setId(id);

//修改数据

albumService.update(album);

**return** **new** **Result**(**true**,StatusCode.OK,"修改成功");

}

/\*\*\*

\* 新增Album数据

\* @param album

\* @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) ;

}

}

# [规格参数模板（实战）](file:///D:\\BaiduNetdiskDownload\\%E8%AF%BE%E7%A8%8B%E4%BC%98%E5%8C%96\\%E8%AE%B2%E4%B9%89\\day02\\%E8%AE%B2%E4%B9%89\\%E7%AC%AC2%E5%A4%A9.html" \l "3-规格参数模板-实战-)

## 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

## 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 |  |  |
| name | 模板名称 | VARCHAR |  |  |
| 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 |  |  |

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 |  |  |

模板与规格是一对多关系 ，模板与参数是一对多关系

## 模板管理

### 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

}

### Dao

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

**public** **interface** **TemplateMapper** **extends** **Mapper**<**Template**> {

}

### 业务层

#### 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**();

}

#### 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();

}

}

### 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");

}

/\* \*\*

\* 通过id查找要修改的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) ;

}

}

## 规格管理

### 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

}

### Dao

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

**public** **interface** **SpecMapper** **extends** **Mapper**<**Spec**> {

}

### 业务层

#### 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**();

}

#### 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？

### 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) ;

}

}

## 参数管理

### 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

}

### Dao

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

**public** **interface** **ParaMapper** **extends** **Mapper**<**Para**> {

}

### 业务层

#### 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**();

}

#### 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);

}

}

### 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) ;

}

}

# [商品分类（实战）](file:///D:\\BaiduNetdiskDownload\\%E8%AF%BE%E7%A8%8B%E4%BC%98%E5%8C%96\\%E8%AE%B2%E4%B9%89\\day02\\%E8%AE%B2%E4%B9%89\\%E7%AC%AC2%E5%A4%A9.html" \l "4-商品分类-实战-)

## 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

## 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

## 实现

### 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

}

### Dao

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

**public** **interface** **CategoryMapper** **extends** **Mapper**<**Category**> {

}

### 业务层

#### 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);

}

#### 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));

}

/\*\*

\* Category分页查询

\* @param page

\* @param size

\* @return

\*/

@Override

**public** PageInfo<Category> **findPage**(**int** page, **int** size){

//静态分页

PageHelper.startPage(page,size);

//分页查询

**return** **new** PageInfo<Category>(categoryMapper.selectAll());

}

/\*\*

\* Category条件查询

\* @param category

\* @return

\*/

@Override

**public** List<Category> **findList**(Category category){

//构建查询条件

Example example = createExample(category);

//根据构建的条件查询数据

**return** categoryMapper.selectByExample(example);

}

/\*\*

\* Category构建查询对象

\* @param category

\* @return

\*/

**public** Example **createExample**(Category category){

Example example=**new** Example(Category.class);

Example.Criteria criteria = example.createCriteria();

**if**(category!=**null**){

// 分类ID

**if**(!StringUtils.isEmpty(category.getId())){

criteria.andEqualTo("id",category.getId());

}

// 分类名称

**if**(!StringUtils.isEmpty(category.getName())){

criteria.andLike("name","%"+category.getName()+"%");

}

// 商品数量

**if**(!StringUtils.isEmpty(category.getGoodsNum())){

criteria.andEqualTo("goodsNum",category.getGoodsNum());

}

// 是否显示

**if**(!StringUtils.isEmpty(category.getIsShow())){

criteria.andEqualTo("isShow",category.getIsShow());

}

// 是否导航

**if**(!StringUtils.isEmpty(category.getIsMenu())){

criteria.andEqualTo("isMenu",category.getIsMenu());

}

// 排序

**if**(!StringUtils.isEmpty(category.getSeq())){

criteria.andEqualTo("seq",category.getSeq());

}

// 上级ID

**if**(!StringUtils.isEmpty(category.getParentId())){

criteria.andEqualTo("parentId",category.getParentId());

}

// 模板ID

**if**(!StringUtils.isEmpty(category.getTemplateId())){

criteria.andEqualTo("templateId",category.getTemplateId());

}

}

**return** example;

}

/\*\*

\* 删除

\* @param id

\*/

@Override

**public** **void** **delete**(Integer id){

categoryMapper.deleteByPrimaryKey(id);

}

/\*\*

\* 修改Category

\* @param category

\*/

@Override

**public** **void** **update**(Category category){

categoryMapper.updateByPrimaryKey(category);

}

/\*\*

\* 增加Category

\* @param category

\*/

@Override

**public** **void** **add**(Category category){

categoryMapper.insert(category);

}

/\*\*

\* 根据ID查询Category

\* @param id

\* @return

\*/

@Override

**public** Category **findById**(Integer id){

**return** categoryMapper.selectByPrimaryKey(id);

}

/\*\*

\* 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);

}

}

### 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);

}

}