淘淘商城第三天

# 第二天内容回顾

1. 展示商品列表。
   1. 分页插件的使用PageHelper。
   2. easyUIDataGrid的使用方法
2. 商品类目选择
   1. easyUI 异步tree控件的使用。
3. 图片上传
   1. 为什么要有图片服务器
   2. FastDFS
   3. http服务器。Nginx。
4. Nginx
   1. http服务器、虚拟主机、反向代理及负载均衡。
   2. 虚拟主机
      1. 基于端口
      2. 基于域名

# 课程计划

1. nginx反向代理和负载均衡
2. FastDFS的使用
3. 实现图片上传功能

# Nginx的反向代理

## 什么是反向代理

正向代理

Pc机

Pc机

Pc机

代理服务器

（代理客户端）

互联网

反向代理：

Pc机

反向代理服务器

[www.taobao.com](http://www.taobao.com)

功能就是请求的转发

Nginx实现

（代理服务器）

淘宝首页

服务器

淘宝首页

服务器

淘宝首页

服务器

反向代理服务器是引用在服务端。决定哪台服务器提供服务。

## 反向代理的模拟

### 反向代理

应该有一个nginx服务器有多个应用服务器（可以是tomcat）

可以使用一台虚拟机，安装一个nginx，多个tomcat，来模拟。

Nginx的配置文件：

|  |
| --- |
| upstream tomcats{  server 192.168.25.148:8080;  server 192.168.25.148:8081;  }  server {  listen 80;  server\_name tomcat.taotao.com;  #charset koi8-r;  #access\_log logs/host.access.log main;  location / {  proxy\_pass http://tomcats;  index index.html index.htm;  }  } |

### 负载均衡

只需要在upstream的server后面添加一个weight即可代表权重。权重越高，分配请求的数量就越多。默认权重是1



# 图片服务器的搭建

## 什么是FastDFS

FastDFS是用c语言编写的一款开源的分布式文件系统。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。



## 文件上传流程



## 文件下载流程



## 上传文件的文件名

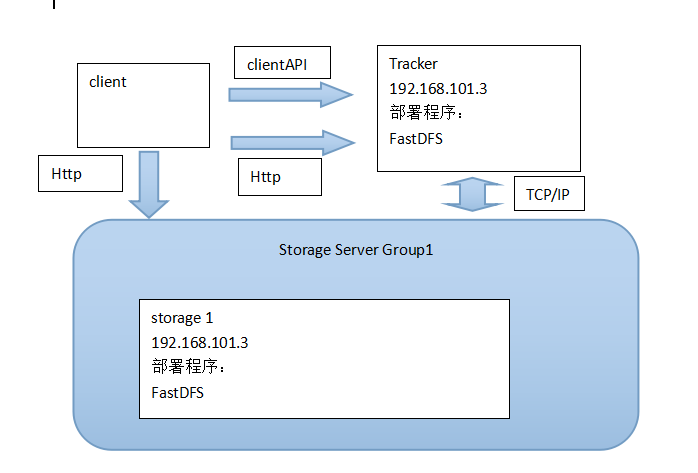
客户端上传文件后存储服务器将文件ID返回给客户端，此文件ID用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。



* 组名：文件上传后所在的storage组名称，在文件上传成功后有storage服务器返回，需要客户端自行保存。
* 虚拟磁盘路径：storage配置的虚拟路径，与磁盘选项store\_path\*对应。如果配置了store\_path0则是M00，如果配置了store\_path1则是M01，以此类推。
* 数据两级目录：storage服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。

文件名：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器IP地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

## FastDFS搭建



可以使用一台虚拟机来模拟，只有一个Tracker、一个Storage服务。

配置nginx访问图片。

### 搭建步骤

第一步：把fastDFS都上传到linux系统。

第二步：安装FastDFS之前，先安装libevent工具包。

yum -y install libevent

第三步：安装libfastcommonV1.0.7工具包。

1. 解压缩
2. ./make.sh
3. ./make.sh install
4. 把/usr/lib64/libfastcommon.so文件向/usr/lib/下复制一份

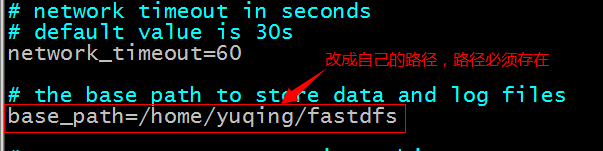
第四步：安装Tracker服务。

1. 解压缩
2. ./make.sh
3. ./make.sh install

安装后在/usr/bin/目录下有以fdfs开头的文件都是编译出来的。

配置文件都放到/etc/fdfs文件夹

1. 把/root/FastDFS/conf目录下的所有的配置文件都复制到/etc/fdfs下。
2. 配置tracker服务。修改/root/FastDFS/conf/tracker.conf文件。

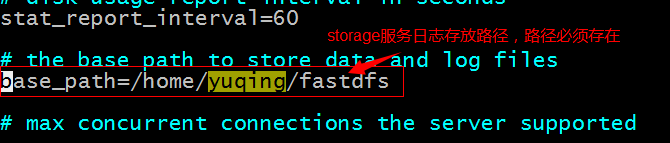


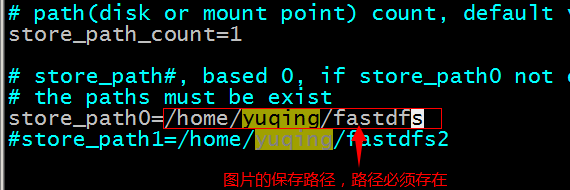
1. 启动tracker。/usr/bin/fdfs\_trackerd /etc/fdfs/tracker.conf

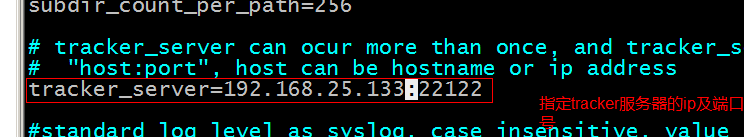
重启使用命令：/usr/bin/fdfs\_trackerd /etc/fdfs/tracker.conf restart

第五步：安装storage服务。

1. 如果是在不同的服务器安装，第四步的1~4需要重新执行。
2. 配置storage服务。修改/root/FastDFS/conf/storage.conf文件





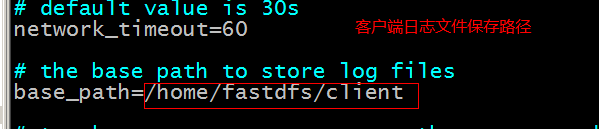


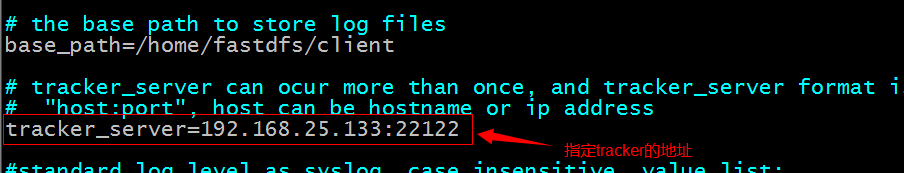
1. 启动storage服务。

/usr/bin/fdfs\_storaged /etc/fdfs/storage.conf **restart**

第六步：测试服务。

1. 修改配置文件/etc/fdfs/client.conf





1. 测试

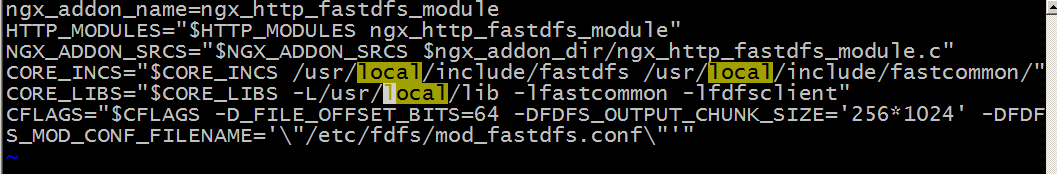
/usr/bin/fdfs\_test /etc/fdfs/client.conf upload anti-steal.jpg

第七步：搭建nginx提供http服务。

可以使用官方提供的nginx插件。要使用nginx插件需要重新编译。

fastdfs-nginx-module\_v1.16.tar.gz

1. 解压插件压缩包
2. 修改/root/fastdfs-nginx-module/src/config文件，把其中的local去掉。



1. 对nginx重新config

./configure \

--prefix=/usr/local/nginx \

--pid-path=/var/run/nginx/nginx.pid \

--lock-path=/var/lock/nginx.lock \

--error-log-path=/var/log/nginx/error.log \

--http-log-path=/var/log/nginx/access.log \

--with-http\_gzip\_static\_module \

--http-client-body-temp-path=/var/temp/nginx/client \

--http-proxy-temp-path=/var/temp/nginx/proxy \

--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \

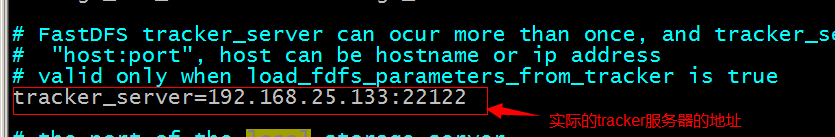
--http-uwsgi-temp-path=/var/temp/nginx/uwsgi \

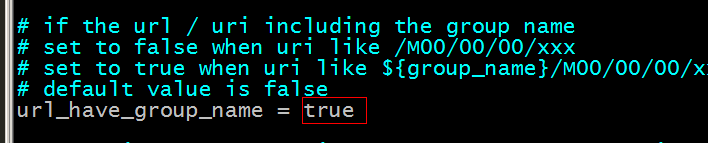
--http-scgi-temp-path=/var/temp/nginx/scgi \

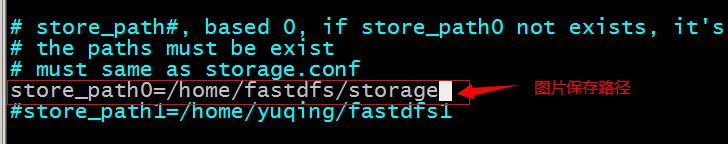
--add-module=/root/fastdfs-nginx-module/src

1. make
2. make install
3. 把/root/fastdfs-nginx-module/src/mod\_fastdfs.conf文件复制到/etc/fdfs目录下。编辑：









1. nginx的配置

在nginx的配置文件中添加一个Server：

server {

listen 80;

server\_name 192.168.101.3;

location /group1/M00/{

#root /home/FastDFS/fdfs\_storage/data;

ngx\_fastdfs\_module;

}

}

1. 将libfdfsclient.so拷贝至/usr/lib下

cp /usr/lib64/libfdfsclient.so /usr/lib/

1. 启动nginx

## 图片服务的使用

官方提供一个jar包

使用方法：

1. 把FastDFS提供的jar包添加到工程中
2. 初始化全局配置。加载一个配置文件。
3. 创建一个TrackerClient对象。
4. 创建一个TrackerServer对象。
5. 声明一个StorageServer对象，null。
6. 获得StorageClient对象。
7. 直接调用StorageClient对象方法上传文件即可。

|  |
| --- |
| @Test  **public** **void** testUpload() **throws** Exception {  // 1、把FastDFS提供的jar包添加到工程中  // 2、初始化全局配置。加载一个配置文件。  ClientGlobal.*init*("D:\\workspaces-itcast\\JaveEE18\\taotao-manager\\taotao-manager-web\\src\\main\\resources\\properties\\client.conf");  // 3、创建一个TrackerClient对象。  TrackerClient trackerClient = **new** TrackerClient();  // 4、创建一个TrackerServer对象。  TrackerServer trackerServer = trackerClient.getConnection();  // 5、声明一个StorageServer对象，null。  StorageServer storageServer = **null**;  // 6、获得StorageClient对象。  StorageClient storageClient = **new** StorageClient(trackerServer, storageServer);  // 7、直接调用StorageClient对象方法上传文件即可。  String[] strings = storageClient.upload\_file("D:\\Documents\\Pictures\\images\\2f2eb938943d.jpg", "jpg", **null**);  **for** (String string : strings) {  System.***out***.println(string);  }  } |

Client.conf

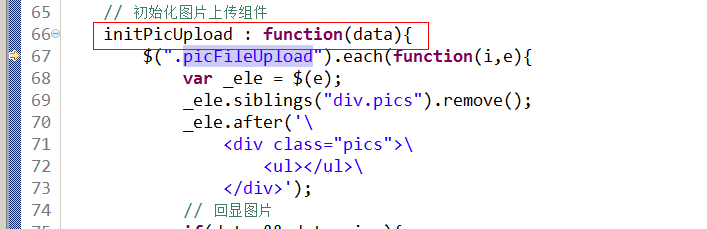
|  |
| --- |
| tracker\_server=192.168.25.133:22122 |

## 使用工具类上传图片

|  |
| --- |
| @Test  **public** **void** testFastDfsClient() **throws** Exception {  FastDFSClient client = **new** FastDFSClient("D:\\workspaces-itcast\\JaveEE18\\taotao-manager\\taotao-manager-web\\src\\main\\resources\\properties\\client.conf");  String uploadFile = client.uploadFile("D:\\Documents\\Pictures\\images\\200811281555127886.jpg", "jpg");  System.***out***.println(uploadFile);  } |

# 图片上传功能

## 功能分析





### 请求的参数：



### 请求的url：

/pic/upload

参考文档：

<http://kindeditor.net/docs/upload.html>

### 添加jar包：

Commons-io、fileupload，两个jar包。

### 在springmvc.xml中配置多媒体解析器

|  |
| --- |
| <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设定默认编码 -->  <property name=*"defaultEncoding"* value=*"UTF-8"*></property>  <!-- 设定文件上传的最大值5MB，5\*1024\*1024 -->  <property name=*"maxUploadSize"* value=*"5242880"*></property>  </bean> |

### 响应的内容

返回格式(JSON)

|  |
| --- |
| //成功时  {  "error" : 0,  "url" : "http://www.example.com/path/to/file.ext"  }  //失败时  {  "error" : 1,  "message" : "错误信息"  } |

需要创建一个pojo描述返回值。

Pojo中有三个属性：error、url、message，可以放到taotao-common工程中。

|  |
| --- |
| **public** **class** PictureResult {  **private** **int** error;  **private** String url;  **private** String message;  **public** **int** getError() {  **return** error;  }  **public** **void** setError(**int** error) {  **this**.error = error;  }  **public** String getUrl() {  **return** url;  }  **public** **void** setUrl(String url) {  **this**.url = url;  }  **public** String getMessage() {  **return** message;  }  **public** **void** setMessage(String message) {  **this**.message = message;  }    } |

## Service层

接收图片数据，把图片上传到图片服务器，返回PictureResult 。需要使用FastDFSClient工具类。

参数：MultiPartFile pictureFile

返回值：PictureResult

|  |
| --- |
| @Service  **public** **class** PictureServiceImpl **implements** PictureService {  @Override  **public** PictureResult uploadPic(MultipartFile picFile) {  PictureResult result = **new** PictureResult();  //判断图片是否为空  **if** (picFile.isEmpty()) {  result.setError(1);  result.setMessage("图片为空");  **return** result;  }  //上传到图片服务器  **try** {  //取图片扩展名  String originalFilename = picFile.getOriginalFilename();  //取扩展名不要“.”  String extName = originalFilename.substring(originalFilename.lastIndexOf(".") + 1);  FastDFSClient client = **new** FastDFSClient("classpath:properties/client.conf");  String url = client.uploadFile(picFile.getBytes(), extName);  //把url响应给客户端  result.setError(0);  result.setUrl(url);  } **catch** (Exception e) {  e.printStackTrace();  result.setError(1);  result.setMessage("图片上传失败");  }  **return** result;  }  } |

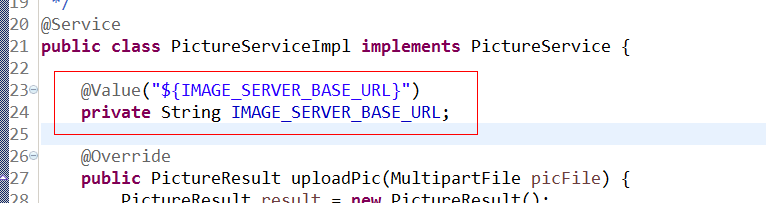
## Controller

接收上传的图片信息，调用Service把图片上传到图片服务器。返回json数据。需要使用@ResponseBody

|  |
| --- |
| @Controller  **public** **class** PictureController {  @Autowired  **private** PictureService pictureService;    @RequestMapping("/pic/upload")  @ResponseBody  **public** PictureResult uploadFile(MultipartFile uploadFile) {  PictureResult result = pictureService.uploadPic(uploadFile);  **return** result;  }  } |

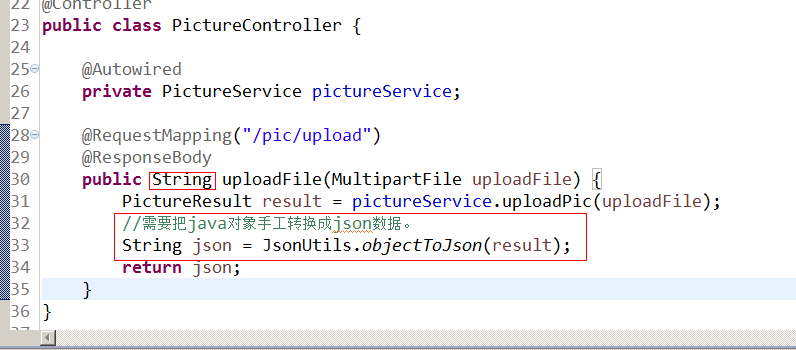
## 加载属性文件

1. 创建一个属性文件
2. 使用spring容器扫描属性文件。
3. @Value注解取属性的值。



## 解决浏览器兼容性问题

要求返回的数据是一个文本类型，要求content-type 为text/plan



# 父子容器

Springmvc（前端控制器）（子容器）

Controller对象

Spring容器（父容器）

1. Mapper代理对象
2. Service对象

子容器可以访问父容器的对象，父容器不能访问子容器中的对象。