# FastDFS

## (讲师：程道)

## 主要课程内容

FastDFS是一个开源的 轻量级 分布式文件系统。它解决了大数据量存储和负载均衡问题。特别适合以中小[4k-500M]文件为载体的在线服务。

- **第一部分：FastDFS 基础回顾**

  为什么要有分布式文件系统 分布式文件系统对比 FastDFS特性 构成 Linux下的安装 java访问FastDFS

- **第二部分：FastDFS 系统架构和功能原理**

  架构详解 架构涉及的概念 设计理念 功能原理(上传 下载 文件同步 删除 断点续传 http访问)

- **第三部分：FastDFS 集群 和 配置优化**

  如何搭建FastDFS 集群 配置优化对应的参数

- **第四部分：FastDFS 项目实战**

  SpringBoot访问FastDFS集群 拉勾业务场景中如何使用FastDFS

## 第一部分 FastDFS 基础回顾

### 1.1 为什么要有分布式文件系统

**单机时代**

**优点**：文件访问比较便利，项目直接引用，实现起来简单，无需任何复杂技术，保存文件和访问文件都很方便。

**缺点**：一方面，文件和代码耦合在一起，文件越多存放越混乱。另一方面，如果流量比较大，静态文件访问会占据一定的资源，影响正常业务进行，不利于网站快速发展。

**独立文件服务器**

**优点**：Web/App服务器可以更专注发挥动态处理的能力。独立存储，更方便做扩容、容灾和数据迁移；方便做图片等资源请求的负载均衡，方便应用各种缓存策略（HTTP Header、Proxy Cache等），也更加方便迁移到CDN。

**缺点**：单机存在性能瓶颈，容灾、垂直扩展性稍差

**分布式文件系统**

**优点**：扩展能力: 毫无疑问，扩展能力是一个分布式文件系统最重要的特点；高可用性: 在分布式文件系统中，高可用性包含两层，一是整个文件系统的可用性，二是数据的完整和一致性；弹性存储: 可以根据业务需要灵活地增加或缩减数据存储以及增删存储池中的资源，而不需要中断系统运行。

**缺点**：系统复杂度稍高，需要更多服务器。

### 1.2 分布式存储解决方案对比

| 指标 | 适合类型 | 文件分布 | 复杂度 | FUSE | POSIX | 备份机制 | 通讯协议接口 | 社区支持 | 开发语言 |
|---|---|---|---|---|---|---|---|---|---|
| FastDFS | 4KB~500MB | 小文件合并存储不分片处理 | 简单 | 不支持 | 不支持 | 组内冗余备份 | Api http | 国内用户群 | C |
| TFS | 所有文件 | 小文件合并，以block组织分片 | 复杂 | 不支持 | 不支持 | Block存储多份,主辅灾备 | API http | 少 | C++ |
| MFS | 大于64K | 分片存储 | 复杂 | 支持 | 支持 | 多点备份动态冗余 | 使用fuse | 较多 | Perl |
| HDFS | 大文件 | 大文件分片分块存储 | 简单 | 支持 | 支持 | 多副本 | 原生api | 较多 | Java |
| Ceph | 对象文件块 | OSD一主多从 | 复杂 | 支持 | 支持 | 多副本 | 原生api | 较少 | C++ |
| MogileFS | 海量小图片 | 不分片存储 | 复杂 | 可以支持 | 不支持 | 动态冗余 | http原生api | 文档少 | Perl |

## 1.3 什么是FastDFS

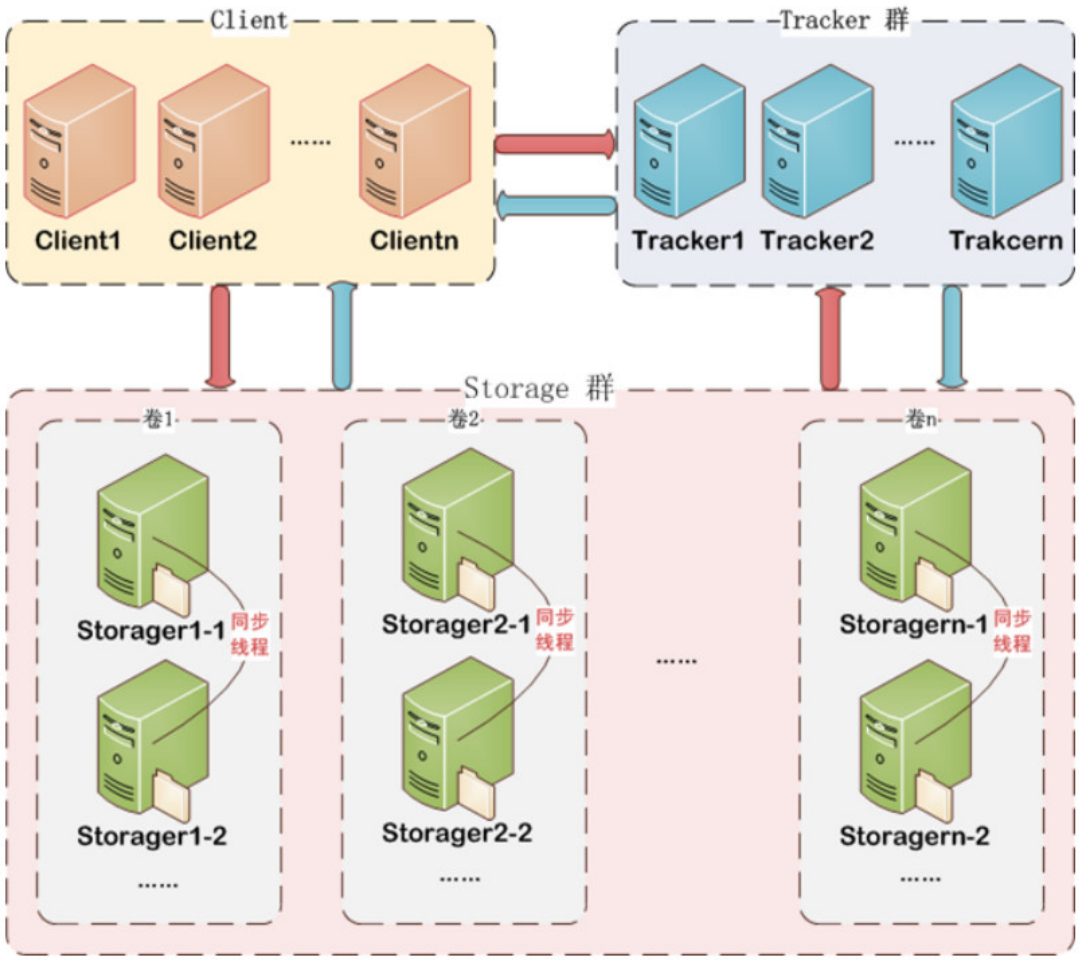
FastDFS是用C语言编写的一款开源的轻量级分布式文件系统。它对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。
FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

## 1.4 FastDFS特性

- 分组存储，灵活简洁、对等结构，不存在单点
- 文件不分块存储，上传的文件和OS文件系统中的文件一一对应
- 文件ID由FastDFS生成，作为文件访问凭证，FastDFS不需要传统的name server
- 和流行的web server无缝衔接，FastDFS已提供apache和nginx扩展模块
- 中、小文件均可以很好支持，支持海量小文件存储
- 支持多块磁盘，支持单盘数据恢复
- 支持相同内容的文件只保存一份，节约磁盘空间
- 支持在线扩容 支持主从文件
- 存储服务器上可以保存文件属性（meta-data）V2.0网络通信采用libevent，支持大并发访问，整体性能更好

- 下载文件支持多线程方式，支持断点续传

## 1.5 FastDFS的构成




FastDFS由客户端(Client)、 跟踪服务器(Tracker Server)和存储服务器(Storage Server)构成。

### 客户端（client）

客户端（client），作为业务请求的发起方，通过专有接口，使用TCP/IP协议与跟踪器服务器或存储节点进行数据交互。

### 跟踪器（tracker）

Trackerserver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。

跟踪器在访问上起负载均衡的作用。可以随时增加或下线而不会影响线上服务。

### 存储节点（storage）

Storageserver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。

存储节点中的服务器均可以随时增加或下线而不会影响线上服务。

## 1.6 Linux下FastDFS 安装

## 1. 安装编译环境

yum install git gcc gcc-c++ make automake  vim   wget  libevent -y

## 2.安装libfastcommon 基础库

```
mkdir  /root/fastdfs
cd     /root/fastdfs
git clone https://github.com/happyfish100/libfastcommon.git --depth 1
cd libfastcommon/
./make.sh && ./make.sh install
```

## 3.安装FastDFS

```
cd  /root/fastdfs
wget https://github.com/happyfish100/fastdfs/archive/V5.11.tar.gz
tar -zxvf V5.11.tar.gz
cd fastdfs-5.11
./make.sh && ./make.sh install

#配置文件准备
cp /etc/fdfs/tracker.conf.sample /etc/fdfs/tracker.conf
cp /etc/fdfs/storage.conf.sample /etc/fdfs/storage.conf
cp /etc/fdfs/client.conf.sample  /etc/fdfs/client.conf
cp /root/fastdfs/fastdfs-5.11/conf/http.conf  /etc/fdfs
cp /root/fastdfs/fastdfs-5.11/conf/mime.types  /etc/fdfs
```

```
vim /etc/fdfs/tracker.conf
#需要修改的内容如下
port=22122
base_path=/home/fastdfs
```

```
vim /etc/fdfs/storage.conf
#需要修改的内容如下
port=23000
base_path=/home/fastdfs  # 数据和日志文件存储根目录
store_path0=/home/fastdfs  # 第一个存储目录
tracker_server=192.168.211.136:22122
 # http访问文件的端口(默认8888,看情况修改,和nginx中保持一致)
http.server_port=8888
```

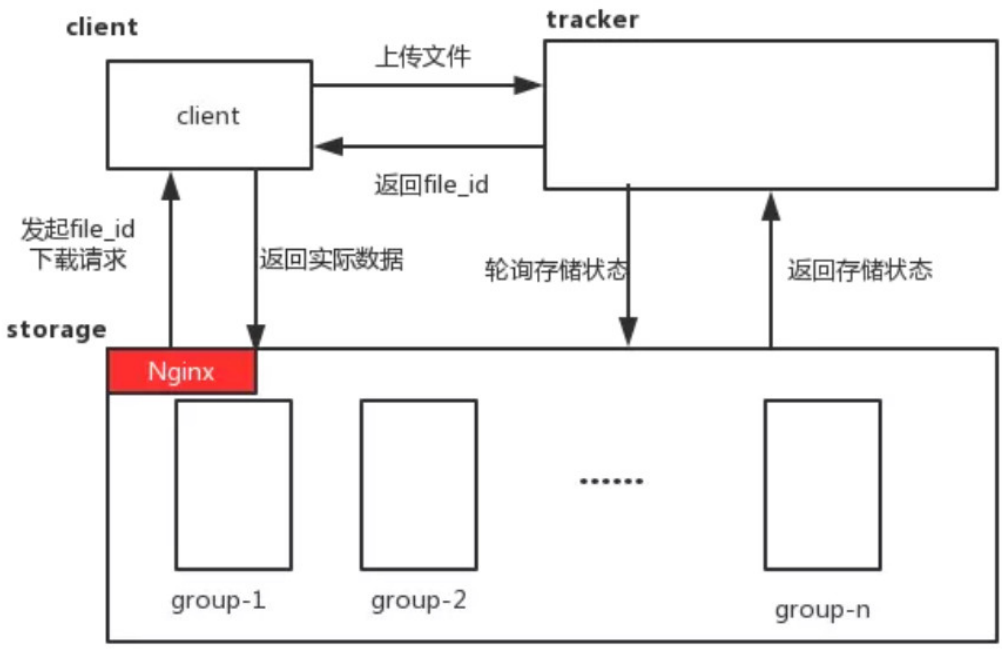
## 4.启动

```
mkdir  /home/fastdfs  -p
/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf restart
/usr/bin/fdfs_storaged /etc/fdfs/storage.conf restart
查看所有运行的端口
netstat -ntlp
```

## 5.测试上传

```
vim /etc/fdfs/client.conf
#需要修改的内容如下
base_path=/home/fastdfs
 #tracker服务器IP和端口
tracker_server=192.168.211.136:22122
#保存后测试,返回ID表示成功 如：group1/M00/00/00/xxx.png
/usr/bin/fdfs_upload_file /etc/fdfs/client.conf /root/fastdfs/1.png
group1/M00/00/00/wKjTiF7h5EWASb5aAACGZa9JdFo611.png
```

## 6.安装fastdfs-nginx-module




```
cd  /root/fastdfs
wget https://github.com/happyfish100/fastdfs-nginx-module/archive/V1.20.tar.gz
解压
tar -xvf V1.20.tar.gz
cd fastdfs-nginx-module-1.20/src
vim config
修改第5 行 和 15 行 修改成
ngx_module_incs="/usr/include/fastdfs /usr/include/fastcommon/"
CORE_INCS="$CORE_INCS /usr/include/fastdfs /usr/include/fastcommon/"
```

```
cp mod_fastdfs.conf /etc/fdfs/
```

```
vim /etc/fdfs/mod_fastdfs.conf
#需要修改的内容如下
tracker_server=192.168.211.136:22122
url_have_group_name=true
store_path0=/home/fastdfs
```

```
mkdir -p /var/temp/nginx/client
```

### 7.安装nginx

```
cd  /root/fastdfs
wget http://nginx.org/download/nginx-1.15.6.tar.gz
tar -zxvf nginx-1.15.6.tar.gz
cd nginx-1.15.6/
```

```
yum -y install pcre-devel openssl openssl-devel
# 添加fastdfs-nginx-module模块
./configure  --add-module=/root/fastdfs/fastdfs-nginx-module-1.20/src
```

```
编译安装
make && make install
查看模块是否安装上
/usr/local/nginx/sbin/nginx -V
```

vim /usr/local/nginx/conf/nginx.conf

```
#添加如下配置
server {
        listen       8888;
        server_name  localhost;
        location ~/group[0-9]/ {
            ngx_fastdfs_module;
        }
}
```

```
/usr/local/nginx/sbin/nginx
```

### 8.测试下载

```
关闭防火墙
systemctl stop firewalld
http://192.168.211.136:8888/group1/M00/00/00/xxx.png
http://192.168.211.136:8888/group1/M00/00/00/wKjTiF7h5EWASb5aAACGZa9JdFo611.png
```

## 1.7 java 访问FastDFS

可以参考官方案例 https://github.com/happyfish100/fastdfs-client-java

1.建立maven 工程 引入jar包

```
<!--fastdfs的java客户端-->
<dependency>
    <groupId>cn.bestwu</groupId>
    <artifactId>fastdfs-client-java</artifactId>
    <version>1.27</version>
</dependency>
```

2.引入配置文件

```
#fastdfs-client.properties
fastdfs.connect_timeout_in_seconds = 5
fastdfs.network_timeout_in_seconds = 30
fastdfs.charset = UTF-8
fastdfs.tracker_servers = 192.168.211.136:22122
```

3.编写测试类

```
@Test
public void testUpload() {
    try {
        //加载配置文件
        ClientGlobal.initByProperties("fastdfs-client.properties");
        //创建tracker客户端
        TrackerClient tc = new TrackerClient();
        //根据tracker客户端创建连接  获取到跟踪服务器对象
        TrackerServer ts = tc.getConnection();
        StorageServer ss = null;
        //定义storage客户端
        StorageClient1 client = new StorageClient1(ts, ss);
        //文件元信息
        NameValuePair[] list = new NameValuePair[1];
        list[0] = new NameValuePair("fileName", "1.png");
        // 上传，返回fileId
        String fileId = client.upload_file1("****.png", "png", list);
        System.out.println(fileId);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

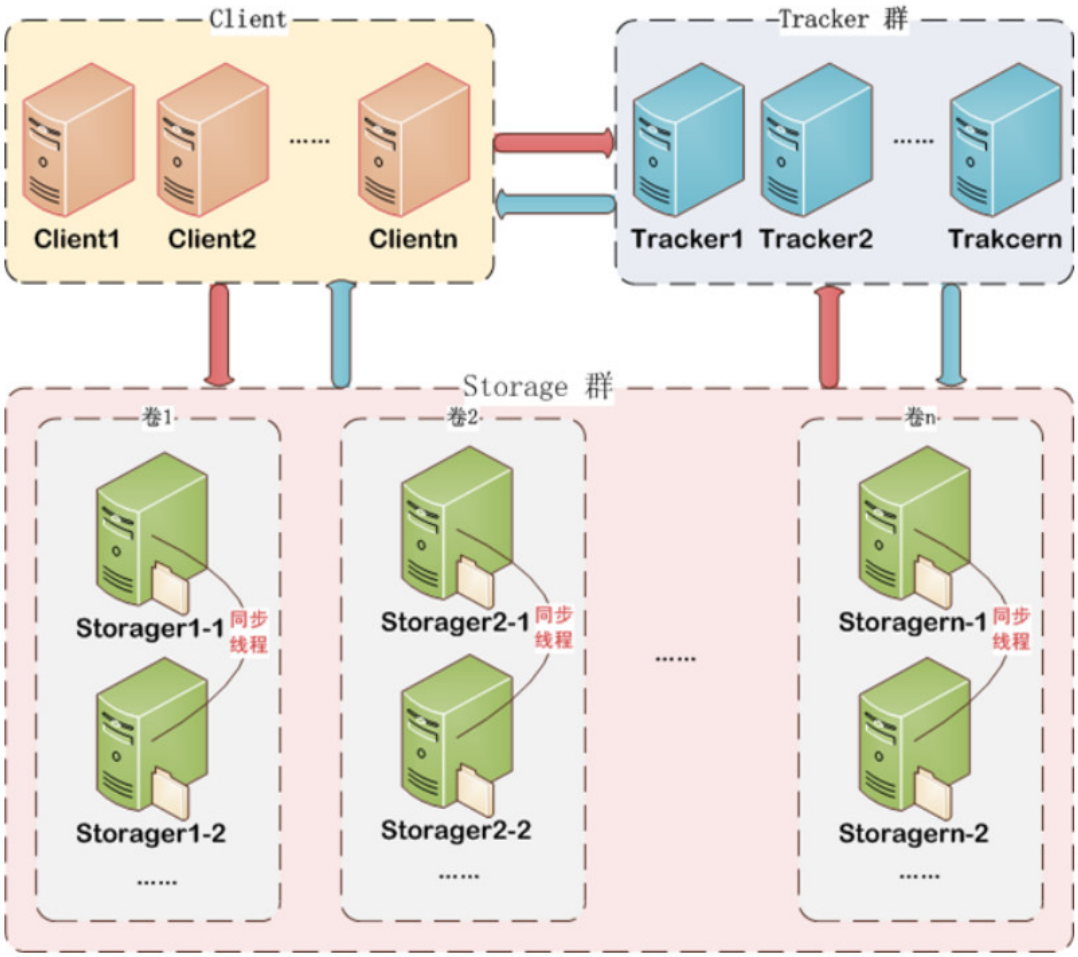
```
@Test
public void testQuery() {
    try {
        //加载配置文件
        ClientGlobal.initByProperties("fastdfs-client.properties");
        // 创建tracker客户端
        TrackerClient tc = new TrackerClient();
        // 根据tracker客户端创建连接 获取到跟踪服务器对象
        TrackerServer ts = tc.getConnection();
        StorageServer ss = null;
        //定义storage客户端
        StorageClient1 client = new StorageClient1(ts, ss);
        // 查询文件信息
        FileInfo fileInfo = client.query_file_info1
        ("group1/M00/00/00/****.png");
        System.out.println(fileInfo);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
@Test
```

```
public void testDownload() {
    try {
        //加载配置文件
        ClientGlobal.initByProperties("config/fastdfs-client.properties");
        // 创建tracker客户端
        TrackerClient tc = new TrackerClient();
        //  根据tracker客户端创建连接
        TrackerServer ts = tc.getConnection();
        StorageServer ss = null;
        //  定义storage客户端
        StorageClient1 client = new StorageClient1(ts, ss);
        // 下载
        byte[] bs = client.download_file1
            ("group1/M00/00/00/****.png");
        FileOutputStream fos = new FileOutputStream(new File("xxxx.png"));
        fos.write(bs);
        fos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

# 第二部分：FastDFS 系统架构和功能原理

## 2.1 架构详解



**storage server**：存储服务器（又称存储节点或数据服务器），文件和文件属性（meta data）都保存到存储服务器上。Storage server直接利用OS的文件系统调用管理文件。

Storage server（后简称storage）以组（卷，group或volume）为单位组织，一个group内包含多台storage机器，数据互为备份，存储空间以group内容量最小的storage为准，所以建议group内的多个storage尽量配置相同，以免造成存储空间的浪费。

以group为单位组织存储能方便的进行应用隔离、负载均衡、副本数定制（group内storage server数量即为该group的副本数），比如将不同应用数据存到不同的group就能隔离应用数据，同时还可根据应用的访问特性来将应用分配到不同的group来做负载均衡；缺点是group的容量受单机存储容量的限制，同时当group内有机器坏掉时，数据恢复只能依赖group内的其它机器，使得恢复时间会很长。

group内每个storage的存储依赖于本地文件系统，storage可配置多个数据存储目录，比如有10块磁盘，分别挂载在`/data/disk1-/data/disk10`，则可将这10个目录都配置为storage的数据存储目录。

storage接受到写文件请求时，会根据配置好的规则，选择其中一个存储目录来存储文件。为了避免单个目录下的文件数太多，在storage第一次启动时，会在每个数据存储目录里创建2级子目录，默认每级256个，总共65536个文件，新写的文件会以hash的方式被路由到其中某个子目录下，然后将文件数据直接作为一个本地文件存储到该目录中。

**group** ：组， 也可称为卷。 同组内服务器上的文件是完全相同的 ，同一组内的storage server之间是对等的， 文件上传、 删除等操作可以在任意一台storage server上进行 。

**meta data** ：文件相关属性，键值对（ Key Value Pair ） 方式，如：width: 1118。

**tracker server**：跟踪服务器，主要做调度工作，起负载均衡的作用。在内存中记录集群中所有存储组和存储服务器的状态信息，是客户端和数据服务器交互的枢纽。因为不记录文件索引信息，所以占用的内存量很少。

Tracker是FastDFS的协调者，负责管理所有的storage server和group，每个storage在启动后会连接Tracker，告知自己所属的group等信息，并保持周期性的心跳，tracker根据storage的心跳信息，建立group==>[storage server list]的映射表。

Tracker需要管理的元信息很少，会全部存储在内存中；另外tracker上的元信息都是由storage汇报的信息生成的，本身不需要持久化任何数据，这样使得tracker非常容易扩展，直接增加tracker机器即可扩展为tracker cluster来服务，cluster里每个tracker之间是完全对等的，所有的tracker都接受stroage的心跳信息，生成元数据信息来提供读写服务。

**client**：客户端，作为业务请求的发起方，通过专有接口，使用TCP/IP协议与跟踪器服务器或存储节点进行数据交互。FastDFS向使用者提供基本文件访问接口，比如upload、download、append、delete等，以客户端库的方式提供给用户使用。

## 2.2 设计理念

**轻量级**

`FastDFS`服务端只有两个角色：`Tracker server`和`Storage server`。

`Tracker server`在内存中记录`分组`和`Storage server`的状态等信息，不记录文件索引信息，占用的内存量很少。另外，客户端（应用）和`Storage server`访问`Tracker server`时，`Tracker server`扫描内存中的分组和`Storage server`状态信息，然后给出应答。由此可以看出`Tracker server`非常轻量化，不会成为系统瓶颈。

FastDFS 中的 Storage server 直接利用 OS 的文件系统存储文件。 FastDFS 不会对文件进行分块存储，客户端上传的文件和 Storage server 上的文件一一对应。对于互联网应用，文件分块存储没有多大的必要。它既没有带来多大的好处，又增加了系统的复杂性。 FastDFS 不对文件进行分块存储，与支持文件分块存储的 DFS 相比，更加简洁高效，并且完全能满足绝大多数互联网应用的实际需要。

在 FastDFS 中，客户端上传文件时，文件 ID 不是由客户端指定，而是由 Storage server 生成后返回给客户端的。文件 ID 中包含了 组名 、 文件相对路径 和 文件名 ， Storage server 可以根据文件 ID 直接定位到文件。因此 FastDFS 集群中根本不需要存储文件索引信息，这是 FastDFS 比较轻量级的一个例证。而其他文件系统则需要存储文件索引信息，这样的角色通常称作 NameServer 。其中 mogileFS 采用 MySQL 数据库来存储文件索引以及系统相关的信息，其局限性显而易见， MySQL 将成为整个系统的瓶颈。

FastDFS 轻量级的另外一个体现是代码量较小。最新的 V2.0 包括了 C 客户端 API 、 FastDHT 客户端 API 和 PHP extension 等，代码行数不到 5.2 万行。

**分组存储**

FastDFS 采用了 分组存储 方式。集群由一个或多个组构成，集群存储总容量为集群中所有组的存储容量之和。一个组由一台或多台存储服务器组成，同组内的多台 Storage server 之间是对等的互备关系。文件上传、下载、删除等操作可以在组内任意一台 Storage server 上进行。类似木桶短板效应，一个组的存储容量为该组内存储服务器容量最小的那个，由此可见组内存储服务器的软硬件配置最好是一致的。用分组存储方式的好处是灵活、可控性较强。比如上传文件时，可以由客户端直接指定上传到的组。一个分组的存储服务器访问压力较大时，可以在该组增加存储服务器来扩充服务能力（纵向扩容）。当系统容量不足时，可以增加组来扩充存储容量（横向扩容）。采用这样的分组存储方式，可以使用 FastDFS 对文件进行管理，使用主流的 Web server 如 Apache、nginx 等进行文件下载。

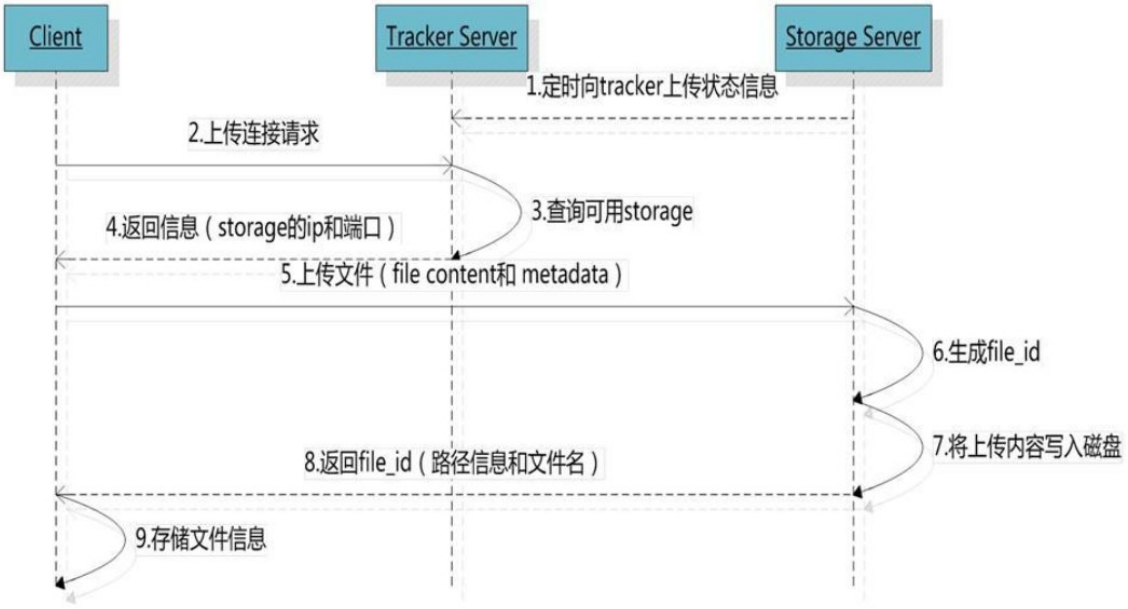
**对等结构**

FastDFS 集群中的 Tracker server 也可以有多台， Tracker server 和 Storage server 均不存在单点问题。 Tracker server 之间是对等关系，组内的 Storage server 之间也是对等关系。传统的 Master-Slave 结构中的 Master 是单点，写操作仅针对 Master 。如果 Master 失效，需要将 Slave 提升为 Master ，实现逻辑会比较复杂。和 Master-Slave 结构相比，对等结构中所有结点的地位是相同，每个结点都是Master，不存在单点问题。

# 2.3 FastDFS 功能原理

## 2.3.1 文件上传

**文件上传流程**




**文件上传内部原理**

**1、选择tracker server和group**

当集群中不止一个tracker server时，由于tracker之间是完全对等的关系，客户端在upload文件时可以任意选择一个trakcer。 当tracker接收到upload_file的请求时，会为该文件分配一个可以存储该文件的group，使用store_lookup选择group的规则：

- 0、Round robin，所有的group间轮询
- 1、Specified group，指定某一个确定的group
- 2、Load balance，剩余存储空间多的group优先

**2、选择storage server**

当选定group后，tracker会在group内选择一个storage server给客户端，使用store_server选择storage的规则：

- 0、Round robin，在group内的所有storage间轮询
- 1、First server ordered by ip，按ip排序
- 2、First server ordered by priority，按优先级排序（优先级在storage上配置）

**3、选择storage path**

当分配好storage server后，客户端将向storage发送写文件请求，storage将会为文件分配一个数据存储目录 storage server可以有多个存放文件的存储路径（可以理解为多个磁盘），store_path支持如下规则：

- 0、Round robin，多个存储目录间轮询
- 2、剩余存储空间最多的优先

**4、生成文件名**

选定存储目录之后，storage会为文件生一个文件名，由storage server ip、文件创建时间、文件大小、文件crc32和一个随机数拼接而成，然后将这个二进制串进行base64编码，转换为可打印的字符串。 选择两级目录 当选定存储目录之后，storage会为文件分配一个文件名，每个存储目录下有两级256*256的子目录，storage会按文件fileid进行两次hash，路由到其中一个子目录，然后将文件以这个文件标示为文件名存储到该子目录下。

**5、返回文件id**

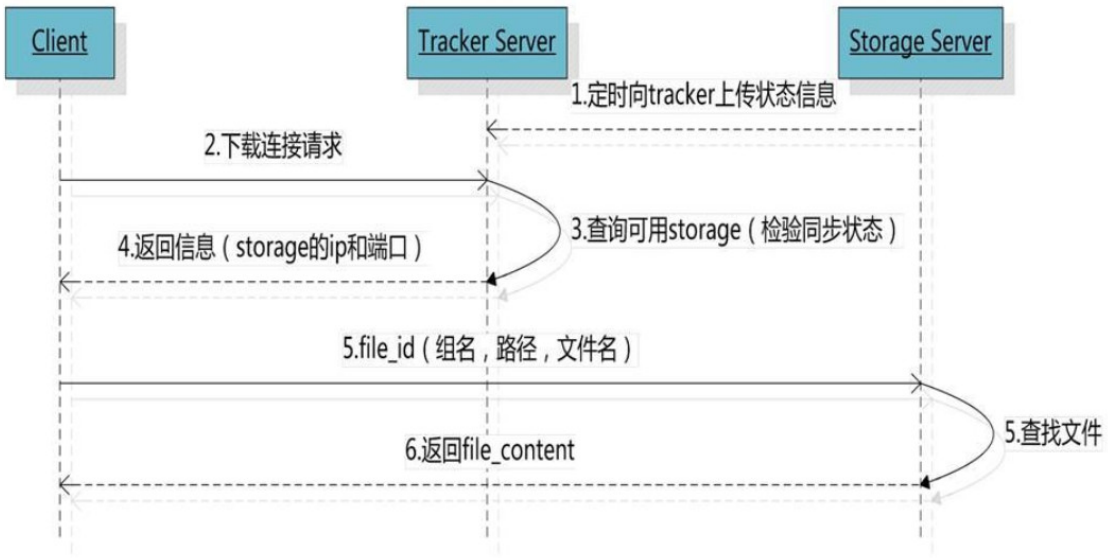
当文件存储到某个子目录后，即认为该文件存储成功，接下来会为该文件返回一个文件id，由group、存储目录、两级子目录、内部文件名、文件后缀名（由客户端指定，主要用于区分文件类型）拼接而成。

```
group1/M00/00/00/wKjTiF7iGy6AMefcAACGZa9JdFo097.png
```

- 组名：文件上传后所在的存储组名称，在文件上传成功后有存储服务器返回，需要客户端自行保存。
- 虚拟磁盘路径：存储服务器配置的虚拟路径，与磁盘选项store_path*对应。
- 数据两级目录：存储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。

### 2.3.2 文件下载

客户端带上文件名信息请求Tracker服务获取到存储服务器的ip地址和端口，然后客户端根据返回的IP地址和端口号请求下载文件，存储服务器接收到请求后返回文件给客户端。




跟upload_file一样，在download_file时客户端可以选择任意tracker server。

客户端发送download请求给某个tracker，必须带上文件名信息，tracke从文件名中解析出文件的group、大小、创建时间等信息，然后为该请求选择一个storage用来服务读请求。

选择哪个 storage server 作为下载服务器 使用download_server 规则如下:

0: 轮询方式，可以下载当前文件的任意一个 storage server进行轮询

1: 哪个为源storage server 就用哪个

由于group内的文件同步时在后台异步进行的，所以有可能出现在读到时候，文件还没有同步到某些storage  server ,为了尽量避免访问到这样的storage，会有相应的文件同步规则。

### 2.3.3 文件同步

**文件同步原理**

写文件时，客户端将文件写至group内一个storage server即认为写文件成功，storage server写完文件后，会由后台线程将文件同步至同group内其他的storage server。

每个storage写文件后，同时会写一份binlog，binlog里不包含文件数据，只包含文件名等元信息，这份binlog用于后台同步，storage会记录向group内其他storage同步的进度，以便重启后能接上次的进度继续同步。进度以时间戳的方式进行记录，所以最好能保证集群内所有server的时钟保持同步。

storage的同步进度会作为元数据的一部分汇报到tracker上，tracke在选择storage的时候会以同步进度作为参考。比如一个group内有A、B、C 三个storage server，A向C同步到进度为T1 ，B向C同步到时间戳为T2（T2 > T1)，tracker接收到这些同步进度信息时，就会进行整理，将最小的那个做为C的同步时间戳，本例中T1即为C的同步时间戳（即所有T1以前写的数据都已经同步到C上了）。同理，根据上述规则，tracker会为A、B生成一个同步时间戳。

**tracker选择group内可用的storage的规则**

1. 该文件上传到的源头storage

- 源头storage只要存活着，肯定包含这个文件，源头的地址被编码在文件名中。

2. 文件创建时间戳==storage被同步到的时间戳 且(当前时间-文件创建时间戳) > 文件同步最大时间（如5分钟)

- 文件创建后，认为经过最大同步时间后，肯定已经同步到其他storage了。

3. 文件创建时间戳 < storage被同步到的时间戳。

- 同步时间戳之前的文件确定已经同步了

4. (当前时间 - 文件创建时间戳) > 同步延迟阀值。

- 经过同步延迟阈值时间，认为文件肯定已经同步了。

### 2.3.4 文件删除

删除处理流程与文件下载类似：

1. Client询问Tracker server可以删除指定文件的Storage server，参数为文件ID（包含组名和文件名）。
2. Tracker server返回一台可用的Storage server。
3. Client直接和该Storage server建立连接，完成文件删除。

文件删除API：delete_file

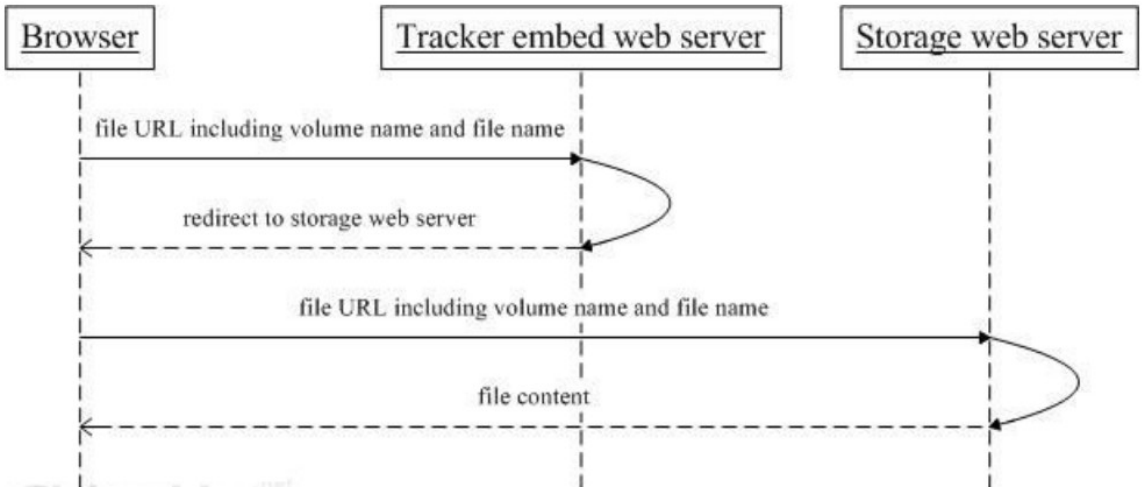
### 2.3.5 断点续传

提供appender file的支持，通过upload_appender_file接口完成，appender file允许在创建后，对该文件进行append操作。实际上，appender file与普通文件的存储方式是相同的，不同的是，appender file不能被合并存储到trunk file。续传涉及到的文件大小MD5不会改变。续传流程与文件上传类似，先定位到源storage，完成完整或部分上传，再通过binlog进行同group内server文件同步。

断点续传的API:upload_appender_file

### 2.3.6 文件HTTP访问支持

FastDFS的tracker和storage都内置了http协议的支持，客户端可以通过http协议来下载文件，tracker在接收到请求时，通过http的redirect机制将请求重定向至文件所在的storage上。除了内置的http协议外，FastDFS还提供了通过apache或nginx扩展模块下载文件的支持。




# 第三部分 FastDFS 集群和配置优化

## 3.1 FastDFS集群架构设置

3台CentOS 7 虚拟机

3个tracker 3个storage

| 192.168.211.130 | 192.168.211.136 | 192.168.211.135 |
| --- | --- | --- |
| tracker | tracker | tracker |
| storage ( group1 ) | storage ( group1 ) | storage ( group2 ) |
| nginx | nginx | nginx |

按照之前的单机方式先进行安装

## 3.2 FastDFS集群架构详细配置

### 1.配置tracker集群 三台相同

vi /etc/fdfs/tracker.conf

```
store_lookup=0 # 0是轮询，1是指定组,2是剩余存储空间多的group优先
```

### 2.配置storage集群

vi /etc/fdfs/storage.conf

```
tracker_server=192.168.211.130:22122
tracker_server=192.168.211.136:22122
tracker_server=192.168.211.135:22122
group_name=group1 #注意组名 192.168.211.135 配置是 group2
port=23000 #storage 的端口号,同一个组的 storage 端口号必须相同
```

```
/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf restart
/usr/bin/fdfs_storaged /etc/fdfs/storage.conf restart
```

### 3.查看storage的日志 查看tracker集群信息

cat /home/fastdfs/logs/storaged.log

访问FastDFS时 可以把Tracker 理解成对等的,但实际底层多个Tracker时在运行过程中会选择其中一个作为Leader，由该Leader执行一些唯一性的操作。在早期版本中Tracker-Leader有两个作用，分别是：为新加入的Storage分配一个源Storage；为开启合并存储的Group选择Trunk-Server。但是在最新的版本中实际上只有第二个作用，也就是选择Trunk-Server。

查看之后 如果连接不上 注意防火墙

systemctl stop firewalld

### 4.查看存储集群信息

/usr/bin/fdfs_monitor /etc/fdfs/storage.conf

### 5.测试上传

```
vim /etc/fdfs/client.conf
tracker_server=192.168.211.130:22122
tracker_server=192.168.211.136:22122
tracker_server=192.168.211.135:22122
```

```
/usr/bin/fdfs_upload_file /etc/fdfs/client.conf /root/fastdfs/1.png
```

```
find / -name   *.png
```

## 3.3 使用Nginx 和 FastDFS集群结合

vi /etc/fdfs/mod_fastdfs.conf

```
tracker_server=192.168.211.130:22122
tracker_server=192.168.211.136:22122
tracker_server=192.168.211.135:22122
group_name=group1 #注意组名 如果是group2 则一定要改
```

vim /usr/local/nginx/conf/nginx.conf

```
#添加如下配置
server {
        listen       8888;
        server_name  localhost;
        location ~/group[0-9]/ {
            ngx_fastdfs_module;
        }
}
```

/usr/local/nginx/sbin/nginx

## 3.4 测试文件服务器

通过任意一台nginx 访问

http://192.168.211.130/group1/M00/00/00/wKjIZVyLMi6AH08jAADtXa53YW0605.png

http://192.168.211.135/group1/M00/00/00/wKjIZVyLMi6AH08jAADtXa53YW0605.png

## 3.5 FastDFS配置优化

### 3.5.1 最大连接数设置

```
配置文件：tracker.conf 和 storage.conf
参数名：max_connections
缺省值：256
说明：FastDFS为一个连接分配一个task buffer，为了提升分配效率，FastDFS采用内存池的做法。
FastDFS老版本直接事先分配 max_connections 个buffer，这个做法显然不是太合理，在
max_connections 设置过大的情况下太浪费内存。v5.04对预分配采用增量方式，tracker一次预分配
1024个，storage一次预分配256个。
#define ALLOC_CONNECTIONS_ONCE 1024
总的task buffer初始内存占用情况测算如下
    改进前：max_connections * buffer_size
    改进后：max_connections和预分配的连接中那个小 * buffer_size
使用v5.04及后续版本，可以根据实际需要将 max_connections 设置为一个较大的数值，比如 10240
甚至更大。
注意此时需要将一个进程允许打开的最大文件数调大到超过max_connections否则FastDFS server启动
会报错。
vi /etc/security/limits.conf  重启系统生效
*               soft     nofile          65535
*               hard     nofile          65535
另外，对于32位系统，请注意使用到的内存不要超过3GB
```

### 3.5.2 工作线程数设置

```
配置文件：tracker.conf 和 storage.conf
参数名： work_threads
缺省值：4
说明：为了避免CPU上下文切换的开销，以及不必要的资源消耗，不建议将本参数设置得过大。为了发挥出多个CPU的效能，系统中的线程数总和，应等于CPU总数。
对于tracker server，公式为：
     work_threads + 1 = CPU数
对于storage，公式为：
    work_threads + 1 + (disk_reader_threads  + disk_writer_threads) *
store_path_count  = CPU数
```

### 3.5.3 storage目录数设置

```
配置文件： storage.conf
参数名：subdir_count_per_path
缺省值：256
说明：FastDFS采用二级目录的做法，目录会在FastDFS初始化时自动创建。存储海量小文件，打开了trunk存储方式的情况下，建议将本参数适当改小，比如设置为32，此时存放文件的目录数为 32 * 32 = 1024。假如trunk文件大小采用缺省值64MB，磁盘空间为2TB，那么每个目录下存放的trunk文件数均值为：2TB/(1024 * 64MB) = 32个
```

### 3.5.4 storage磁盘读写线程设置

```
配置文件： storage.conf
参数名:disk_rw_separated：磁盘读写是否分离
参数名:disk_reader_threads：单个磁盘读线程数
参数名:disk_writer_threads：单个磁盘写线程数
如果磁盘读写混合，单个磁盘读写线程数为读线程数和写线程数之和，对于单盘挂载方式，磁盘读写线程分别设置为    1即可
如果磁盘做了RAID，那么需要酌情加大读写线程数，这样才能最大程度地发挥磁盘性能
```

### 3.5.5 storage同步延迟相关设置

```
配置文件： storage.conf
参数名:sync_binlog_buff_interval：将binlog buffer写入磁盘的时间间隔，取值大于0，缺省值为60s
参数名:sync_wait_msec：如果没有需要同步的文件，对binlog进行轮询的时间间隔，取值大于0，缺省值为200ms
参数名: sync_interval：同步完一个文件后，休眠的毫秒数，缺省值为0

为了缩短文件同步时间，可以将上述3个参数适当调小即可
```

# 第四部分 FastDFS 应用实战

## 4.1 SpringBoot访问FastDFS

### 1.引入jar依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.5.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.github.tobato</groupId>
        <artifactId>fastdfs-client</artifactId>
        <version>1.26.1-RELEASE</version>
    </dependency>
</dependencies>
```

### 2.配置文件 application.yml

```
fdfs:
  connectTimeout: 600
  trackerList:
    - 192.168.211.130:22122
    - 192.168.211.135:22122
    - 192.168.211.136:22122

server:
  port: 8899
```

### 3.编写上传下载文件的服务类

```
package com.lagou.service;

import com.github.tobato.fastdfs.domain.StorePath;
import com.github.tobato.fastdfs.proto.storage.DownloadByteArray;
import com.github.tobato.fastdfs.service.FastFileStorageClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.io.ByteArrayInputStream;
import java.io.IOException;

@Component
public class FastDFSClientService {
    @Autowired
    private FastFileStorageClient fastFileStorageClient;

    /**
     * 文件上传
     *
```

```
     * @param bytes      文件字节
     * @param fileSize   文件大小
     * @param extension 文件扩展名
     * @return fastDfs路径
     */
    public String uploadFile(byte[] bytes, long fileSize, String extension) {
        ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(bytes);
        StorePath storePath =
fastFileStorageClient.uploadFile(byteArrayInputStream,
          fileSize, extension, null);
        System.out.println(storePath.getGroup() + ":" + storePath.getPath() +
":"
          + storePath.getFullPath());
        return storePath.getFullPath();
    }

    /**
     * 下载文件
     *
     * @param fileUrl 文件URL
     * @return 文件字节
     * @throws IOException
     */
    public byte[] downloadFile(String fileUrl) throws IOException {
        String group = fileUrl.substring(0, fileUrl.indexOf("/"));
        String path = fileUrl.substring(fileUrl.indexOf("/") + 1);
        DownloadByteArray downloadByteArray = new DownloadByteArray();
        byte[] bytes = fastFileStorageClient.downloadFile(group, path,
          downloadByteArray);
        return bytes;
    }
}
```

**4.控制器类**

```
package com.lagou.controller;

import com.lagou.service.FastDFSClientService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.net.URLEncoder;

@RestController
@RequestMapping("/fastdfs")
public class FastDFSController {
    @Autowired
    private FastDFSClientService fastDFSClientService;

    @RequestMapping("/upload")
    public String uploadFile(MultipartFile file) throws IOException {
```

```
        byte[] bytes = file.getBytes();
        String originalFileName = file.getOriginalFilename();
        String extension =
originalFileName.substring(originalFileName.lastIndexOf(".") + 1);
        String fileName = file.getName();
        long fileSize = file.getSize();
        System.out.println(originalFileName + ":" + fileName + ":" + fileSize +
":" + extension + ":" + bytes.length);
        return fastDFSClientService.uploadFile(bytes, fileSize, extension);
    }

    @RequestMapping("/download")
    public void downloadFile(String fileUrl, HttpServletResponse response)
throws IOException {
        byte[] bytes = fastDFSClientService.downloadFile(fileUrl);
        response.setHeader("Content-disposition", "attachment;filename=" +
URLEncoder.encode(fileUrl, "UTF-8"));
        response.setCharacterEncoding("UTF-8");
        ServletOutputStream outputStream = null;
        try {
            outputStream = response.getOutputStream();
            outputStream.write(bytes);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                outputStream.flush();
                outputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

**5.编写启动类**

```
package com.lagou;

import com.github.tobato.fastdfs.FdfsClientConfig;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.EnableMBeanExport;
import org.springframework.context.annotation.Import;
import org.springframework.jmx.support.RegistrationPolicy;

//解决jmx重复注册bean的问题
@EnableMBeanExport(registration = RegistrationPolicy.IGNORE_EXISTING)
//只需要一行注解就可以拥有带有连接池的FastDFS Java客户端了
@Import(FdfsClientConfig.class)
@SpringBootApplication
public class FastDfsApplicationMain {

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

**6.使用postman 进行测试**

# 4.2 拉勾业务场景

## 4.2.1 头像上传

## 4.2.2 nginx 头像缩略图

处理思路:

上传图片时 直接处理成缩略图 不是很灵活 业务很确定可以使用

先存储图片 使用时 实时计算

## 4.2.3 简历附件

**上传简历附件**

**下载简历附件**

思考是使用 直接的url 还是需要写一个处理请求

## 4.2.4 避免重复

如何避免文件重复上传 解决方案 上传成功后计算文件对应的MD5然后存入MySQL,添加文件时把文件MD5和之前存入MYSQL中的存储的信息对比 。参考:DigestUtils.md5DigestAsHex(bytes)。