# 分布式文件系统 FastDFS

## 1. 分布式文件系统简介

### 1.1 什么是分布式文件系统?

分布式文件系统（Distributed File System，DFS）是指 文件系统 管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连；或是若干不同的逻辑 磁盘 分区或卷标组合在一起而形成的完整的有层次的文件系统。DFS为分布在网络上任意位置的资源提供一个逻辑上的树形文件系统结构，从而使用户访问分布在网络上的共享文件更加简便。

**文件存储变迁史：**

- **直连存储：**存储和数据直连，拓展性、灵活性差，如：Tomcat、Nginx等等
- **中心化存储：**设备类型丰富，通过网络互连，具有一定的拓展性，但是受到控制器能力限制，拓展能力有限，如：**NAS**、SAN等等
- **分布式存储**：通过网络使用多态服务器的磁盘空间，并将这些分散的存储资源构成一个虚拟的存储设备，如：**OSS、FastDFS**、七牛云存储等等

**优势：**

- **可扩展**：分布式存储系统可以扩展到数百甚至数千个这样的集群大小，并且系统的整体性能可以线性增长。
- **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
- **低成本**：分布式存储系统的自动容错和自动负载平衡允许在成本较低服务器上构建分布式存储系统。此外，线性可扩展性还能够增加和降低服务器的成本。
- **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

### 1.2 常见的DFS

目前常见的分布式文件系统有：FastDFS、HDFS、TFS、GridFS、GFS、MogileFS等等。

- **1）FastDFS**：**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。纯C语言开发，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
- **2）HDFS**：全称 Hadoop Distributed File System是 Hadoop 项目的一个子项目，也是 Hadoop 核心组件之一。Hadoop 非常适于存储大型数据 (比如 TB 和 PB)，Hadoop 的存储系统使用 HDFS
- **3）TFS**：Taobao FileSystem是一个高可扩展、高可用、高性能、面向互联网服务的分布式文件系统，主要针对海量的非结构化数据，它构筑在普通的Linux机器 集群上，可为外部提供高可靠和高并发的存储访问

    - TFS为淘宝提供海量小文件存储，通常文件大小不超过1M，满足了淘宝对小文件存储的需求，被广泛地应用在淘宝各项应用中
    - TFS采用了HA架构和平滑扩容，保证了整个文件系统的可用性和扩展性。同时扁平化的数据组织结构，可将文件名映射到文件的物理地址，简化 了文件的访问流程，一定程度上为TFS提供了良好的读写性能
- **4）GridFS**：是MongoDB的一个内置功能，它提供一组文件操作的API以利用MongoDB存储文件
    - GridFS的基本原理是将文件保存在两个Collection中，一个保存文件索引，一个保存文件内容，文件内容按一定大小分成若干块，每一块存在一个Document中，这种方法不仅提供了文件存储，还提供了对文件相关的一些附加属性（比如MD5值，文件名等等）的存储。文件在GridFS中会按4MB为单位进行分块存储
- **5）GFS**：Google File System公司为了满足本公司需求而开发的基于Linux的专有分布式文件系统。尽管Google公布了该系统的一些技术细节，但Google并没有将该系统的软件部分作为开源软件发布
- **6）MogileFS**：由memcahed的开发公司danga一款perl开发的产品，目前国内使用mogielFS的有图片托管网站yupoo等。MogileFS是一套高效的文件自动备份组件，由Six Apart开发，广泛应用在包括LiveJournal等web2.0站点上

**特性对比：**

| 文件系统 | FastDFS | HDFS | TFS | MogileFS |
| --- | --- | --- | --- | --- |
| 数据存储方式 | 文件/块 | 文件 | 文件 | 文件 |
| 集群通讯协议 | 私有协议 | 私有协议 | 私有协议 | Http |
| 扩容 | 支持 | 支持 | 支持 | 支持 |
| 冗余备份 | 支持 | 支持 | 支持 | 不支持 |
| 单点故障 | 不存在 | 存在 | 存在 | 存在 |
| 跨集群同步 | 部分支持 | 不支持 | 支持 | 不支持 |
| 开发语言 | C | Java | C++ | Perl |
| 适合类型 | 4KB - 500MB | 大文件 | 所有文件 | 海量小图片 |
| 复杂度 | 简单 | 简单 | 复杂 | 复杂 |

| 文件系统 | FastDFS | HDFS | TFS | MogileFS |
| --- | --- | --- | --- | --- |
| 易用性 | 安装简单，社区活跃 | 安装简单，文档专业 | 安装复杂，文档较少 | 安装复杂，文档较少 |
| 研发团队 | 国内开发者-余庆 | Apache | Alibaba | Danga Interactive |
| FUSE | 不支持 | 支持 | 不支持 | 支持 |
| POSIX | 不支持 | 支持 | 无资料 | 不支持 |

## 1.3 常见DFS提供商

- 阿里OSS
- 七牛云存储
- 百度云存储

# 2. FastDFS简介

## 2.1 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**，它是由淘宝资深架构师余庆编写并开源。FastDFS专为互联网量身定制，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。

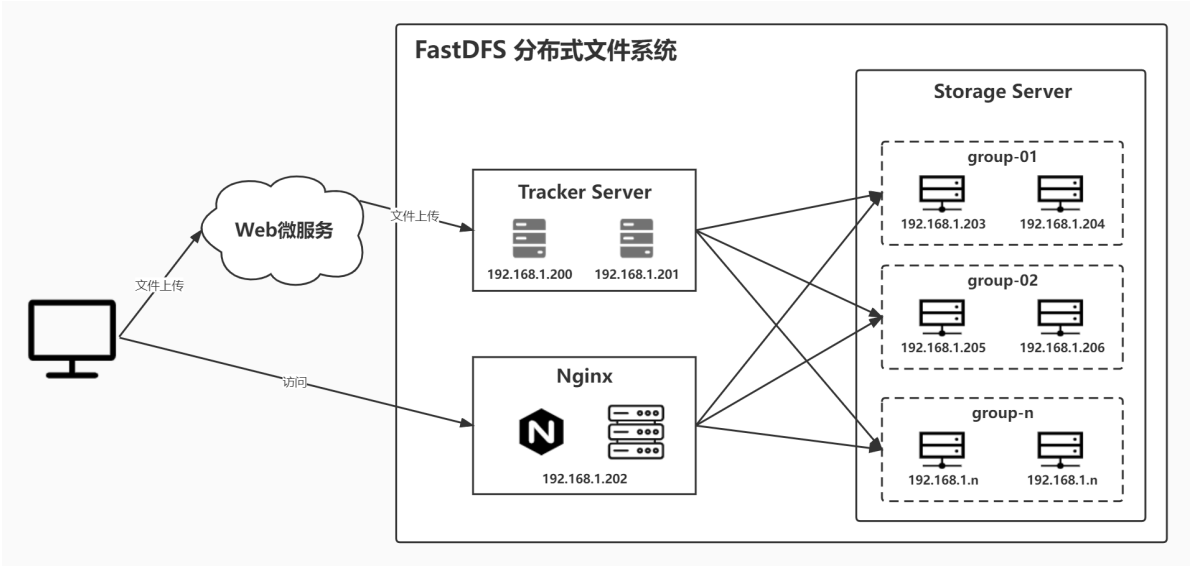
功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以中小文件（ 建议范围： 4KB 到 500MB ） 为载体的在线服务，如：相册网站、视频网站等等

## 2.2 工作原理

### 2.2.1 FastDFS架构

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。

如下图:




**1）Tracker Server**

Tracker 作用是负载均衡和调度，通过Tracker 在文件上传时可以根据一些策略找到Storage 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。

FastDFS集群中的Tracker 可以有多台，Tracker 之间是相互平等关系同时提供服务，Tracker 不存在单点故障。客户端请求Tracker 采用轮询方式，如果请求的Tracker 无法提供服务则换另一个Tracker 。

**2）Storage Server**

Storage 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 没有实现自己的文件系统而是使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。

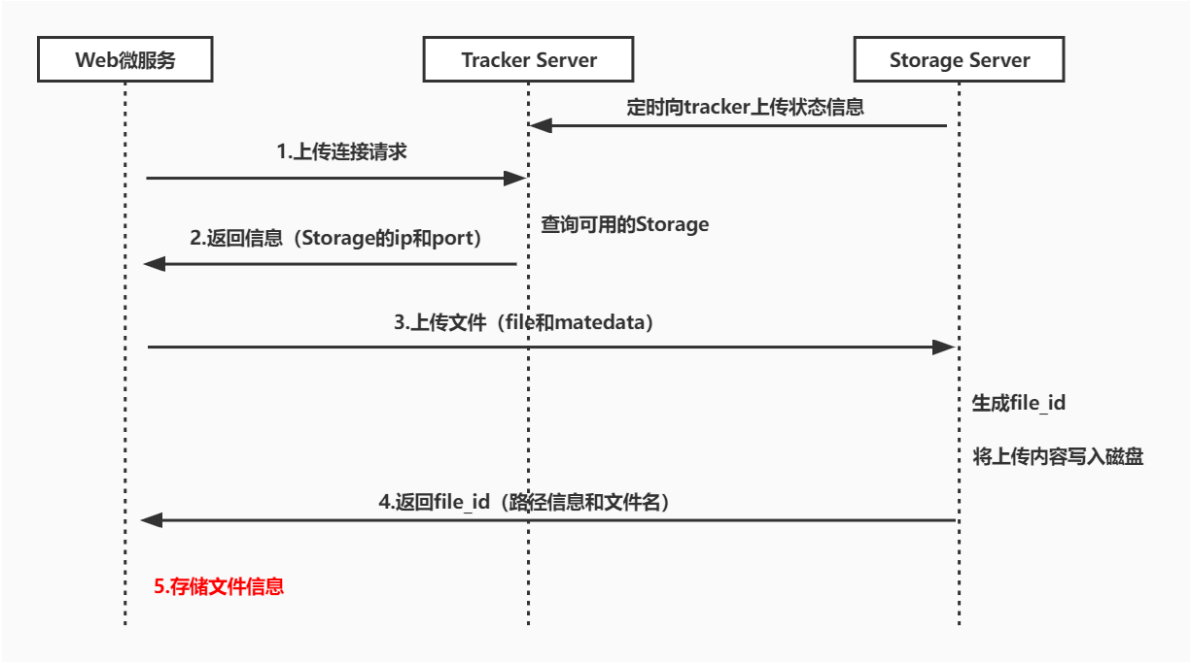
Storage集群采用了分组存储方式。Storage 集群由一个或多个组构成，集群存储总容量为集群中所有组的存储容量之和。一个组由一台或多台存储服务器组成，组内的Storage 之间是平等关系，不同组的Storage 之间不会相互通信，同组内的Storage 之间会相互连接进行文件同步，从而保证同组内每个Storage 上的文件完全一致的。一个组的存储容量为该组内的存储服务器容量最小的那个，由此可见组内存储服务器的软硬件配置最好是一致的。

采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也可以由Tracker 进行调度选择。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）。

**3）Storage状态收集**

Storage 会连接集群中所有的Tracker，定时向他们报告自己的状态，包括磁盘剩余空间、文件同步状况、文件上传下载次数等统计信息。

## 2.2.2 文件上传流程



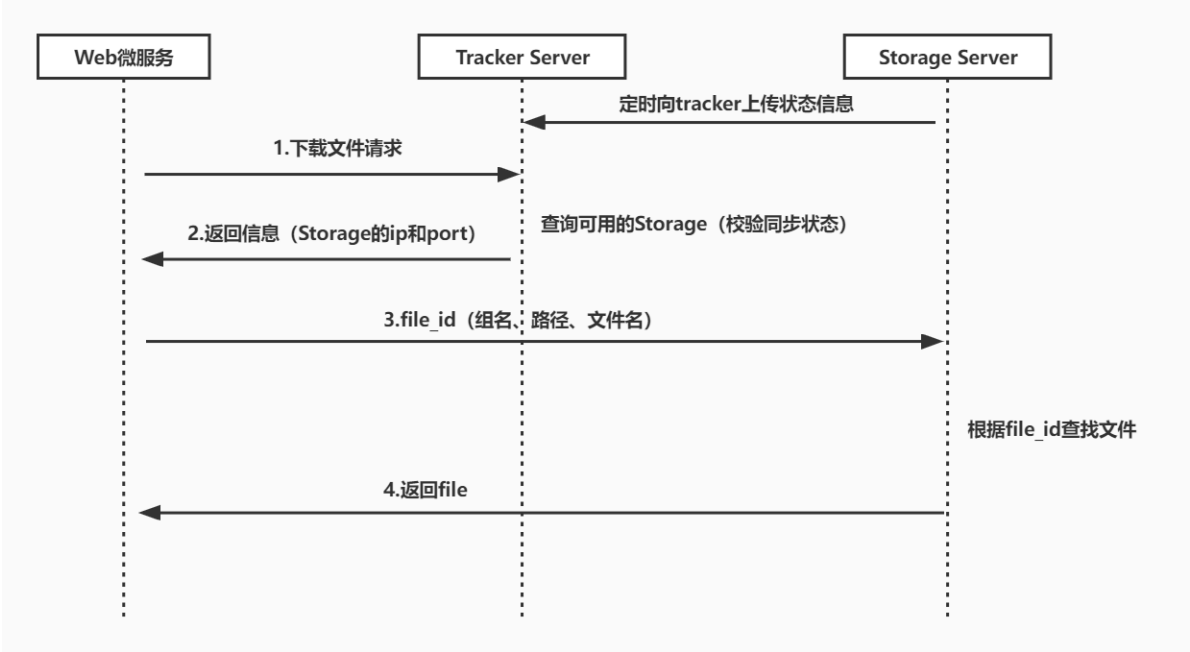

客户端上传文件后存储服务器将文件ID返回给客户端，此文件ID用于以后访问该文件的索引信息。

文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

```
group1/M00/00/00/wKjIgGNslmOAf5VSAACQjdb7ANw5904822
```

- 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。

### 2.2.3 文件下载流程




tracker根据请求的文件路径即文件ID 来快速定义文件。

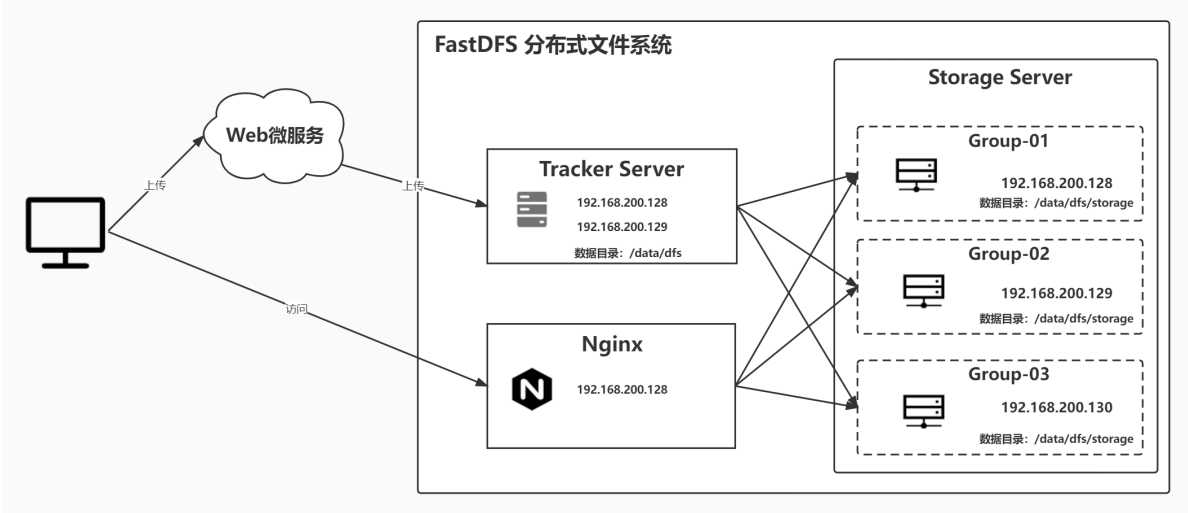
比如请求下边的文件：

```
group1/M00/00/00/wKjIgGNslmOAf5VSAACQjdb7ANw5904822
```

1. 通过组名tracker能够很快的定位到客户端需要访问的存储服务器组是group1，并选择合适的存储服务器提供客户端访问。
2. 存储服务器根据“文件存储虚拟磁盘路径”和“数据文件两级目录”可以很快定位到文件所在目录，并根据文件名找到客户端需要访问的文件。

# 3. FastDFS部署与使用

## 3.1 分布式环境部署




### 3.1.1 搭建步骤

- 第一步：环境准备
- 第二步：配置2台Tracker，并启动
- 第三步：配置3台Storage，并启动
- 第四步：测试分布式部署是否成功
- 第五步：配置1台Nginx，并启动
- 第六步：测试Nginx整合FastDFS是否成功

### 3.1.2 环境准备

FastDFS 安装环境：

| 名称 | 说明 |
| --- | --- |
| centos | 7.x |
| libfastcommon | FastDFS分离出的公用函数库 |
| libserverframe | FastDFS分离出的网络框架 |
| FastDFS | FastDFS本体 |
| fastdfs-nginx-module | FastDFS和Nginx的关联模块 |
| nginx | nginx1.15.4 |

磁盘目录

| 说明 | 位置 |
| --- | --- |
| 所有安装包 | /usr/local/src |
| 数据/日志存储位置 | /data/dfs/ |

```
mkdir -p /data/dfs # 创建数据存储目录
cd /usr/local/src # 切换到安装目录准备下载安装包
```

### 构建编译环境

> 注意：此安装环境命令仅适用于CentOS

```
yum install git gcc gcc-c++ make automake autoconf libtool pcre pcre-devel
zlib zlib-devel openssl-devel wget vim -y
```

### 安装libfastcommon

```
cd /usr/local/src
git clone https://github.com/happyfish100/libfastcommon.git --depth 1
cd libfastcommon/
./make.sh && ./make.sh install #编译安装
```

### 安装libserverframe

```
cd /usr/local/src
git clone https://github.com/happyfish100/libserverframe.git --depth 1
cd libserverframe/
./make.sh && ./make.sh install #编译安装
```

### 安装FastDFS

```
cd /usr/local/src
git clone https://github.com/happyfish100/fastdfs.git --depth 1
cd fastdfs/
./make.sh && ./make.sh install #编译安装
#配置文件准备
cp /usr/local/src/fastdfs/conf/http.conf /etc/fdfs/ #供nginx访问使用
cp /usr/local/src/fastdfs/conf/mime.types /etc/fdfs/ #供nginx访问使用
```

### 安装fastdfs-nginx-module

```
cd /usr/local/src
git clone https://github.com/happyfish100/fastdfs-nginx-module.git --depth 1
cp /usr/local/src/fastdfs-nginx-module/src/mod_fastdfs.conf /etc/fdfs
```

**安装nginx**

```
cd /usr/local/src
wget http://nginx.org/download/nginx-1.15.4.tar.gz #下载nginx压缩包
tar -zxvf nginx-1.15.4.tar.gz #解压
cd nginx-1.15.4/
#添加fastdfs-nginx-module模块
./configure --add-module=/usr/local/src/fastdfs-nginx-module/src/
make && make install #编译安装
```

### 3.1.3 Tracker配置

```
# 服务器ip为 192.168.200.128,192.168.200.129
vim /etc/fdfs/tracker.conf
#需要修改的内容如下
port=22122  # tracker服务器端口（默认22122,一般不修改）
base_path=/data/dfs  # 存储日志和数据的根目录，tracker在运行时会向此目录存储storage的管理数据。
store_lookup=2 # 存储策略默认是2
# 0=轮询向storage存储文件
# 1=指定具体的group
# 2=负载均衡，选择空闲的storage存储
# 注意：store_lookup一般选择2，如果选择1则需要指定具体的group
```

启动

```
# 修改 /usr/lib/systemd/system/fdfs_trackerd.service 中的 PIDFile，格式为：PIDFile=$base_path/data/dfs/fdfs_trackerd.pid
vim /usr/lib/systemd/system/fdfs_trackerd.service
# 比如：
PIDFile=/data/dfs/data/fdfs_trackerd.pid

systemctl start fdfs_trackerd #启动tracker服务
systemctl status fdfs_trackerd
systemctl restart fdfs_trackerd #重启动tracker服务
systemctl stop fdfs_trackerd #停止tracker服务
systemctl enable fdfs_trackerd  #开机自启动
```

关闭防火墙：

```
systemctl stop firewalld
systemctl status firewalld
```

### 3.1.4 Storage配置

```
vim /etc/fdfs/storage.conf
# 需要修改的内容如下
port=23000  # storage服务端口（默认23000,一般不修改）
group_name=group1 # 分组
base_path=/data/dfs/storage  # 数据和日志文件存储根目录
store_path0=/data/dfs/storage  # 第一个存储目录
# 磁盘存储目录，可定义多个store_path：
# store_path1=...
tracker_server=192.168.200.128:22122  # 上报tracker的地址01
tracker_server=192.168.200.129:22122  # 上报tracker的地址02
http.server_port=8888  # http访问文件的端口(默认8888,看情况修改,和nginx中保持一致)
```

启动

```
# 修改 /usr/lib/systemd/system/fdfs_storaged.service 中的 PIDFile，格式为：
PIDFile=$base_path/data/fdfs_storaged.pid
vim /usr/lib/systemd/system/fdfs_storaged.service
# 比如：
PIDFile=/data/dfs/data/fdfs_storaged.pid

systemctl start fdfs_storaged # 启动storage服务
systemctl status fdfs_storaged # 查看状态
systemctl restart fdfs_storaged  #重动storage服务
systemctl stop fdfs_storaged  #停止动storage服务
systemctl enable fdfs_storaged   #开机自启动
```

检测集群

```
/usr/bin/fdfs_monitor /etc/fdfs/storage.conf
# 会显示会有几台服务器 有3台就会 显示 Storage 1-Storage 3的详细信息
```

配置项说明

- tracker_server：有几台服务器写几个
- group_name：地址的名称的命名
- bind_addr：服务器ip绑定
- store_path_count #store_path(数字)有几个写几个
    - store_path(数字) #设置几个储存地址写几个 从0开始

### 3.1.5 客户端测试上传

```
vim /etc/fdfs/client.conf
#需要修改的内容如下
base_path=/data/moe/dfs
tracker_server=192.168.200.128:22122  # 服务器1
tracker_server=192.168.200.129:22122  # 服务器2
#保存后测试,返回ID表示成功 如：group1/M00/00/00/xx.tar.gz
fdfs_upload_file /etc/fdfs/client.conf /usr/local/src/nginx-1.15.4.tar.gz
```

### 3.1.6 配置nginx访问

```
vim /etc/fdfs/mod_fastdfs.conf
# 需要修改的内容如下
tracker_server=192.168.200.128:22122  # 服务器1
tracker_server=192.168.200.129:22122  # 服务器2
url_have_group_name=true
store_path0=/home/dfs
# 配置nginx.config
vim /usr/local/nginx/conf/nginx.conf
# 添加如下配置
server {
    listen       8888;    ## 该端口为storage.conf中的http.server_port相同
    server_name  localhost;
    location ~/group[0-9]/ {
        ngx_fastdfs_module;
    }
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
    root   html;
    }
}
```

启动

```
/usr/local/nginx/sbin/nginx #启动nginx
/usr/local/nginx/sbin/nginx -s reload #重启nginx
/usr/local/nginx/sbin/nginx -s stop #停止nginx
```

## 3.2 Docker部署FastDFS

### 3.2.1 搜索FastDFS镜像

```
docker search fastdfs
```

### 3.2.2 拉取最新镜像

```
docker pull morunchang/fastdfs
```

### 3.2.3 运行Tracker

```
docker run -d --name tracker --net=host morunchang/fastdfs sh tracker.sh
# -net=host：表示的是使用的网络模式为net模式
```

### 3.2.4 运行Storage

```
docker run -d --name storage --net=host -e TRACKER_IP=192.168.200.128:22122 -
e GROUP_NAME=group1 morunchang/fastdfs sh storage.sh
```

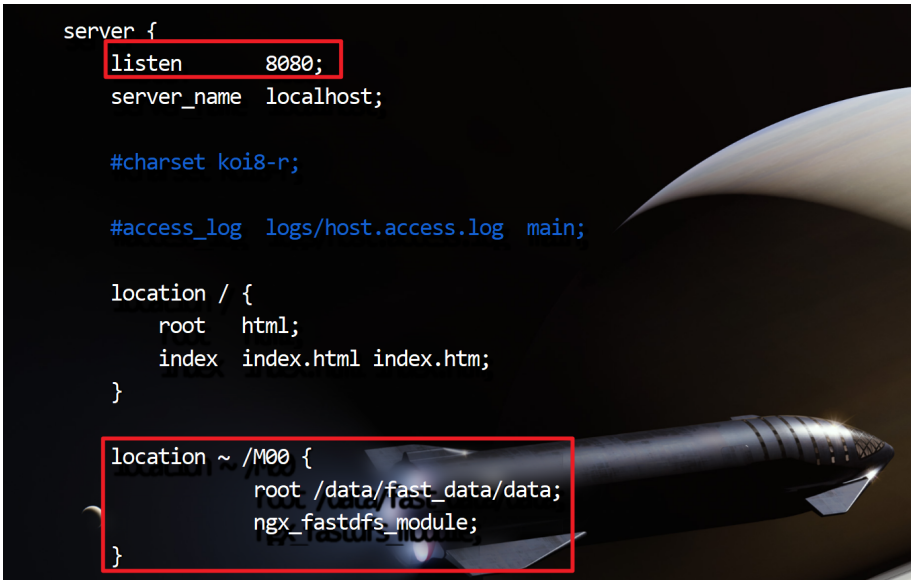
- group1是组名，即storage的组，如果想要增加新的storage服务器，再次运行该命令，更换新组名

Storage内部已经集成了nginx，主要提供对FastDFS图片访问的支持。

如果你想要修改其中的配置:

- 进入容器内部：docker exec -it storage /bin/bash
- 打开nginx.conf配置文件：vi /etc/nginx/conf/nginx.conf
- 修改对应nginx配置

可以看出当我访问/M00时，它先先交给ngx_fastdfs_module模块进行处理




### 3.2.5 查看安装结果

```
1 docker ps
```

## 3.3 文件上传下载测试

### 3.3.1 搭建环境

这里我们使用Java完成文件上传下载用例

**1）创建maven工程**

pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.csource</groupId>
        <artifactId>fastdfs-client-java</artifactId>
        <version>1.29-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

**2）配置文件**

在classpath:下创建**fdfs_client.conf**文件

```
# http连接超时时间
connect_timeout = 2
# tracker与storage网络通信超时时间
network_timeout = 30
charset = UTF-8
http.tracker_http_port = 80
http.anti_steal_token = no
http.secret_key = FastDFS1234567890
# tracker服务器地址，可以重复配置多个
tracker_server = 192.168.200.128:22122
#tracker_server = 192.168.200.129:22122
#tracker_server = 192.168.200.130:22122

# 连接池配置
connection_pool.enabled = true
connection_pool.max_count_per_entry = 500
connection_pool.max_idle_time = 3600
connection_pool.max_wait_time_in_ms = 1000
```

### 3.3.2 文件上传

```java
public class FastDFSDemo {
    private static final String CONF_NAME = "fdfs_client.conf";

    private StorageClient storageClient;

    private TrackerServer trackerServer;

    @Before
    public void initStorageClient() throws Exception {
        ClientGlobal.init(CONF_NAME);
        System.out.println("network_timeout=" +
ClientGlobal.g_network_timeout + "ms");
        System.out.println("charset=" + ClientGlobal.g_charset);
        TrackerClient tracker = new TrackerClient();
        trackerServer = tracker.getTrackerServer();
```

```
        StorageServer storageServer = null;
        storageClient = new StorageClient(trackerServer, storageServer);
    }
    /**
     * 测试上传文件
     */
    @Test
    public void upload() throws Exception{
        NameValuePair[] metaList = new NameValuePair[1];
        String local_filename = "dog.png";
        metaList[0] = new NameValuePair("fileName", local_filename);
        File file = new File("D:/dog.png");
        InputStream inputStream = new FileInputStream(file);
        int length = inputStream.available();
        byte[] bytes = new byte[length];
        inputStream.read(bytes);
        String[] result = storageClient.upload_file(bytes, null, metaList);
        System.out.println("result {}" + Arrays.asList(result));
    }
    @After
    public void closeClient() {
        System.out.println("close connection");
        if(storageClient != null){
            try {
                storageClient.close();
            }catch (Exception e){
                e.printStackTrace();
            }catch (Throwable e){
                e.printStackTrace();
            }
        }
    }

    public void writeByteToFile(byte[] fbyte, String fileName) throws
IOException {
        BufferedOutputStream bos = null;
        FileOutputStream fos = null;
        File file = new File(fileName);
        try {
            fos = new FileOutputStream(file);
            bos = new BufferedOutputStream(fos);
            bos.write(fbyte);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (bos != null) {
                bos.close();
            }
            if (fos != null) {
                fos.close();
            }
        }
    }
}
```

### 3.3.3 文件查询

```
//查询文件
@Test
public void testQueryFile() throws IOException, MyException {
    FileInfo fileInfo = storageClient.query_file_info("group01","M00/00/00/wKjIgGNslmOAf5VSAACQjdb7ANw5904822");
    System.out.println(fileInfo);
}
```

### 3.3.4 文件下载

```
/**
 * 测试下载
 */
@Test
public void download() throws Exception {
    String[] uploadresult = {"group01", "M00/00/00/wKjIgGNslmOAf5VSAACQjdb7ANw5904822"};
    byte[] result = storageClient.download_file(uploadresult[0], uploadresult[1]);
    String local_filename = "dog_two.png";
    //文件写入磁盘
    writeByteToFile(result, local_filename);
    File file = new File(local_filename);
    System.out.println("file.isFile = " + file.isFile());
}
```