

FastDFS 使用文档

目录

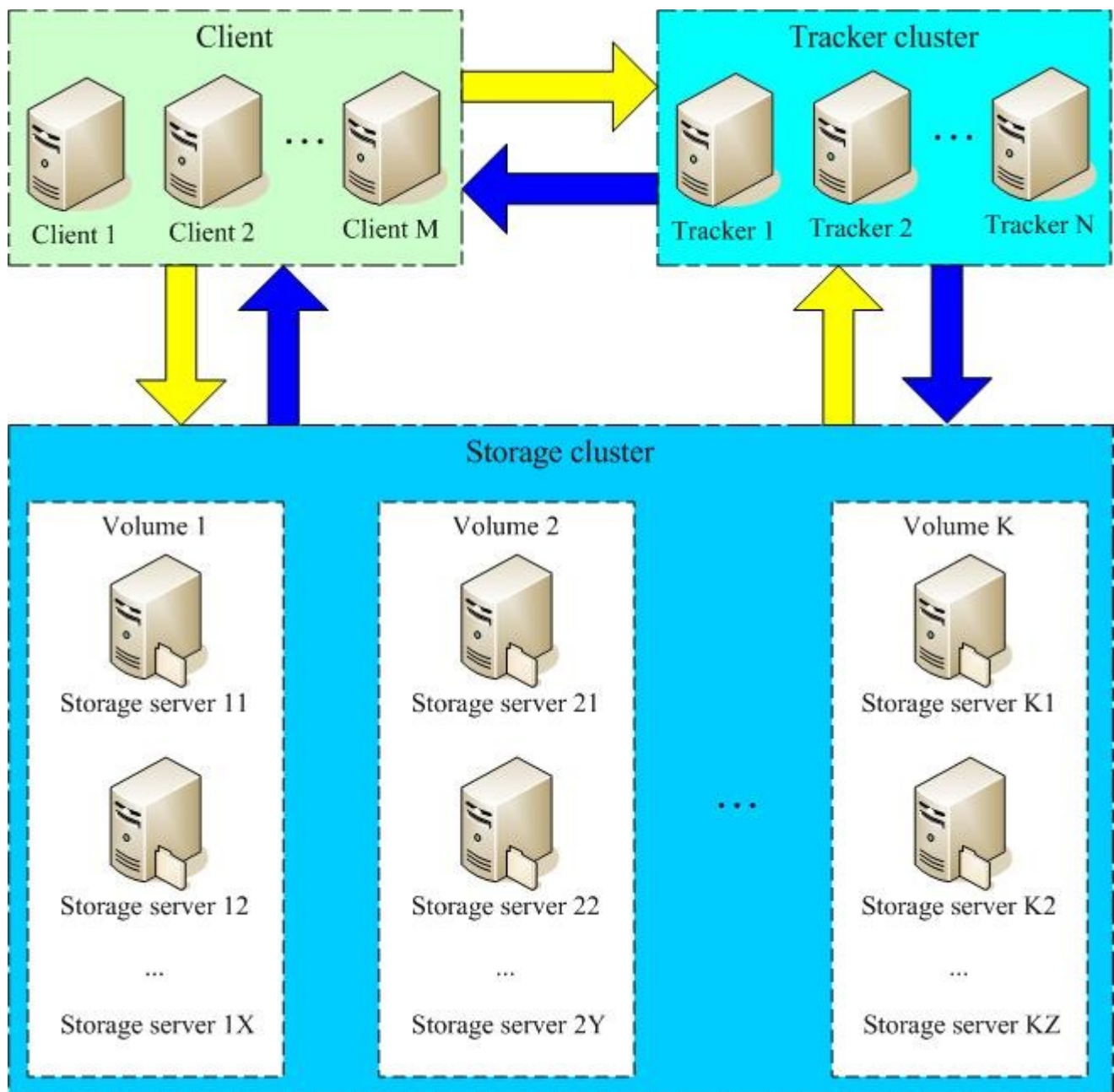
一、FastDFS 原理.....	3
二、FastDFS 分布 式 文件系统的安装与使用.....	9
1、 所 有 跟 踪 服 务 器 和 存 储 服 务 器 均 执 行 如 下 操 作.....	10
1.1、 编译和安装所需的依赖包:.....	10
1.2、 安装 libfastcommon:.....	10
1.3、 安装 FastDFS.....	11
2、 配 置 置 S FastDFS 跟 踪 器 ((192.168.4.121)).....	14
2.1、 复制 FastDFS 跟踪器样例配置文件,并重命名:.....	14
2.2、 编辑跟踪器配置文件 :	14
2.3、 创建基础数据目录 (参考基础目录 base_path 配置) :.....	15
2.4、 防火墙中打开跟踪器端口 (默认为 22122) :.....	15
2.5、 启动 Tracker :	15
2.6、 关闭 Tracker :	15
2.7、 设置 FastDFS 跟踪器开机启动 :	15
3 、 配 置 置 S FastDFS 存 储 ((192.168.4.125)).....	16
3.1、 复制 FastDFS 存储器样例配置文件,并重命名:.....	16
3.2、 编辑存储器样例配置文件 :	16
3.3、 创建基础数据目录 (参考基础目录 base_path 配置) :.....	17
3.4、 防火墙中打开存储器端口 (默认为 23000) :.....	17

3.5、启动 Storage :	17
3.6、关闭 Storage :	17
3.7、设置 FastDFS 存储器开机启动 :	18
4、文件上传测试((192.168.4.121)).....	18
4.1、修改 Tracker 服务器中的客户端配置文件 :	18
4.2、执行如下文件上传命令 :	18
5、在每个存储节点上安装 nginx.....	19
5.1、fastdfs-nginx-module 作用说明.....	19
5.2、上传 fastdfs-nginx-module_v1.16.tar.gz.....	19
位置 : /usr/local/src.....	19
5.3、解压.....	19
5.4、修改 fastdfs-nginx-module 的 config 配置文件.....	20
5.5、上传当前的稳定版本 Nginx(nginx-1.6.2.tar.gz).....	20
位置 : /usr/local/src 目录.....	20
5.6、安装编译 Nginx 所需的依赖包.....	20
5.7、编译安装 Nginx (添加 fastdfs-nginx-module 模块)	20
5.8、复制 fastdfs-nginx-module 源码中的配置文件.....	21
复制 fastdfs-nginx-module 到 /etc/fdfs 目录 , 并修改.....	21
5.9、复制 FastDFS 的部分配置文件到/etc/fdfs 目录.....	21
5.10、在/fastdfs/storage 文件存储目录下创建软连接.....	21
创建软连接将其链接到实际存放数据的目录.....	21
5.11、配置 Nginx.....	21
5.12、防火墙中打开 Nginx 的 8888 端口.....	23
5.13、启动 Nginx.....	23

5.14、通过浏览器访问测试时上传的文件.....	23
三、java 客户端测试代码.....	24

一、FastDFS 原理

[FastDFS](#) 是一个开源的轻量级分布式文件系统，由跟踪服务器（tracker server）、存储服务器（storage server）和客户端（client）三个部分组成，主要解决了海量数据存储问题，特别适合以中小文件（建议范围：4KB < file_size < 500MB）为载体的在线服务。



Storage server

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** 内地其他机器, 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（后面会介绍），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，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。

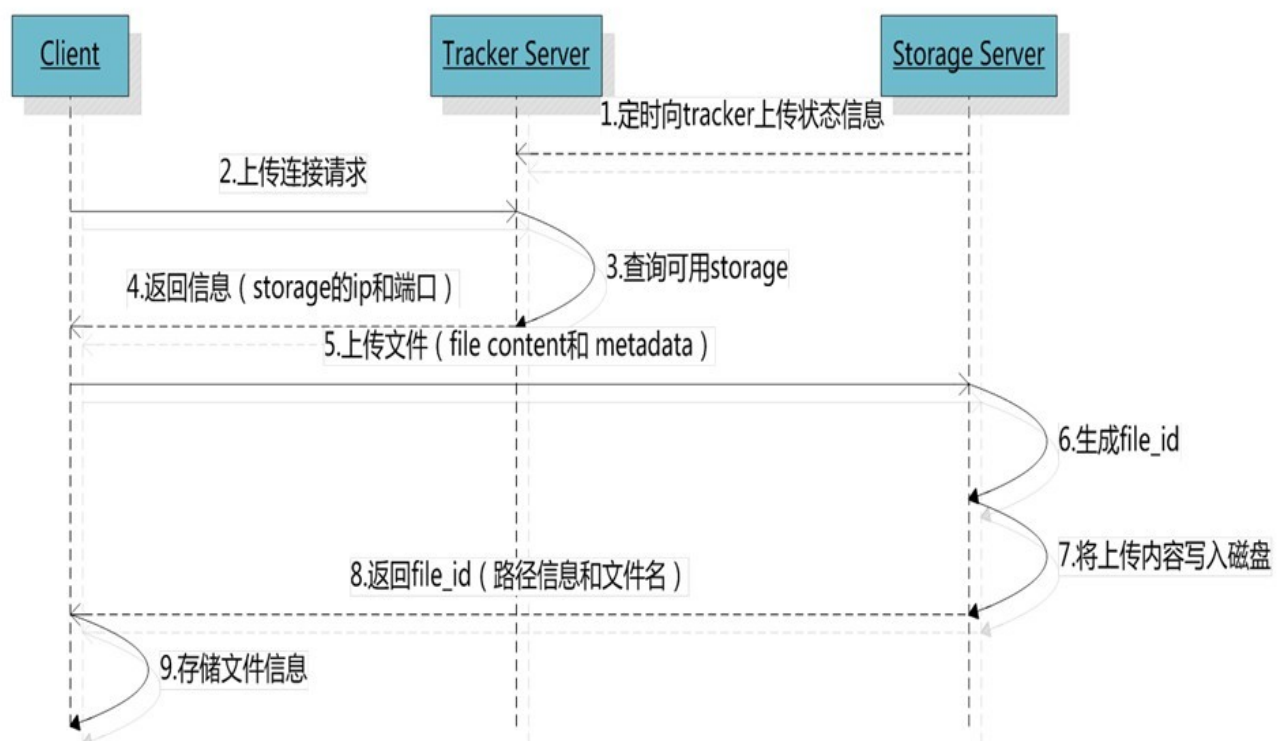
Tracker server

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 storage 的心跳信息，生成元数据信息来提供读写服务。

Upload file

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。



选择 tracker server

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 tracker。

选择存储的 group

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**，支持如下选择 **group** 的规则：1.Round robin，所有的 **group** 间轮询 2.Specifiedgroup，指定某一个确定的 **group** 3.Load balance，剩余存储空间多多 **group** 优先

选择 storage server

当选定 **group** 后，tracker 会在 **group** 内选择一个 storage server 给客户端，支持如下选择 storage 的规则：1.Round robin，在 **group** 内的所有 storage 间轮询 2.First server ordered by ip，按 ip 排序 3.First server ordered by priority，按优先级排序（优先级在 storage 上配置）

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，支持如下规则：1.Round robin，多个存储目录间轮询 2.剩余存储空间最多的优先

生成 Fileid

选定存储目录之后，storage 会为文件生一个 Fileid，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 fileid，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（猜测），路由到其中一个子目录，然后将文件以 fileid 为文件名存储到该子目录下。

生成文件名

当文件存储到某个子目录后，即认为该文件存储成功，接下来会为该文件生成一个文件名，文件名由 **group**、存储目录、两级子目录、fileid、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。



文件同步

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。

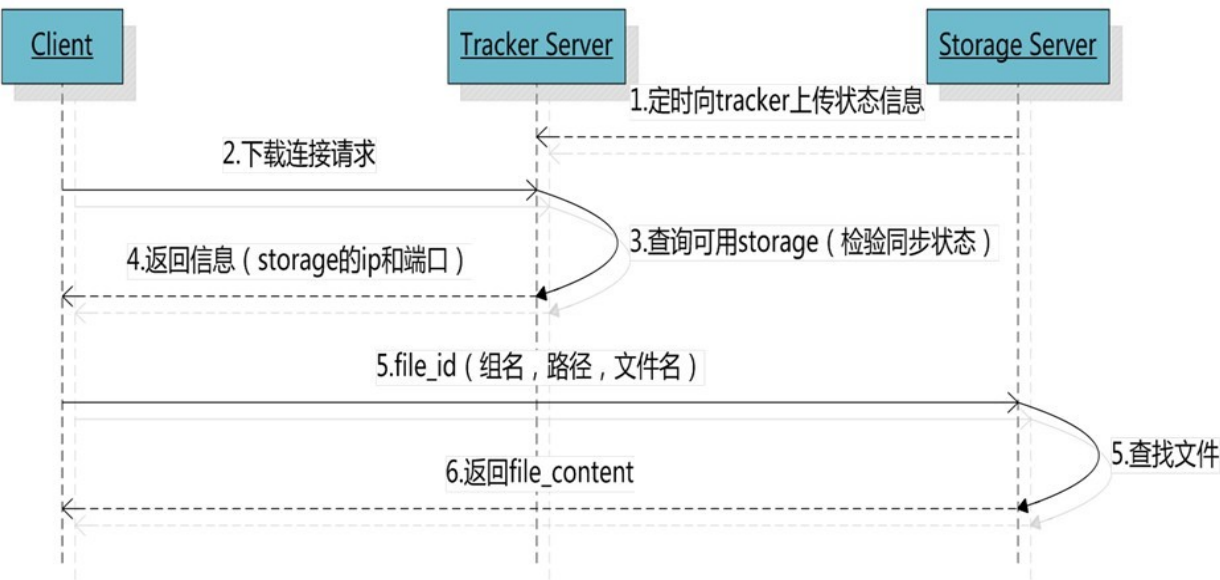
storage 的同步进度会作为元数据的一部分汇报到 tracker 上，tracker 在选择读 storage 的时候会以同步进度作为参考。

比如一个 **group** 内有 A、B、C 三个 storage server，A 向 C 同步到进度为 T1（T1 以前写的文件都已经同步到 B 上了），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

同步时间戳为 **T1**（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。

Download file

客户端 **upload file** 成功后，会拿到一个 **storage** 生成的文件名，接下来客户端根据这个文件名即可访问到该文件。



跟 **upload file** 一样，在 **download file** 时客户端可以选择任意 **tracker server**。

tracker 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。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** 上，为了尽量避免访问到这样的 **storage**，**tracker** 按照如下规则选择 **group** 内可读的 **storage**。

1. 该文件上传的源头 **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) > 同步延迟阈值 (如一天)。- 经过同步延迟阈值时间，认为文件肯定已经同步了。

小文件合并存储

将[小文件合并存储](#)主要解决如下几个问题：

1. 本地文件系统 **inode** 数量有限，从而存储的小文件数量也就受到限制。2. 多级目录+目录里很多文件，导致访问文件的开销很大 (可能导致很多次 IO) 3. 按小文件存储，备份与恢复的效率低

FastDFS 在 V3.0 版本里[引入小文件合并存储](#)的机制，可将多个小文件存储到一个大的文件 (**trunk file**)，为了支持这个机制，**FastDFS** 生成的文件 **fileid** 需要额外增加 16 个字节

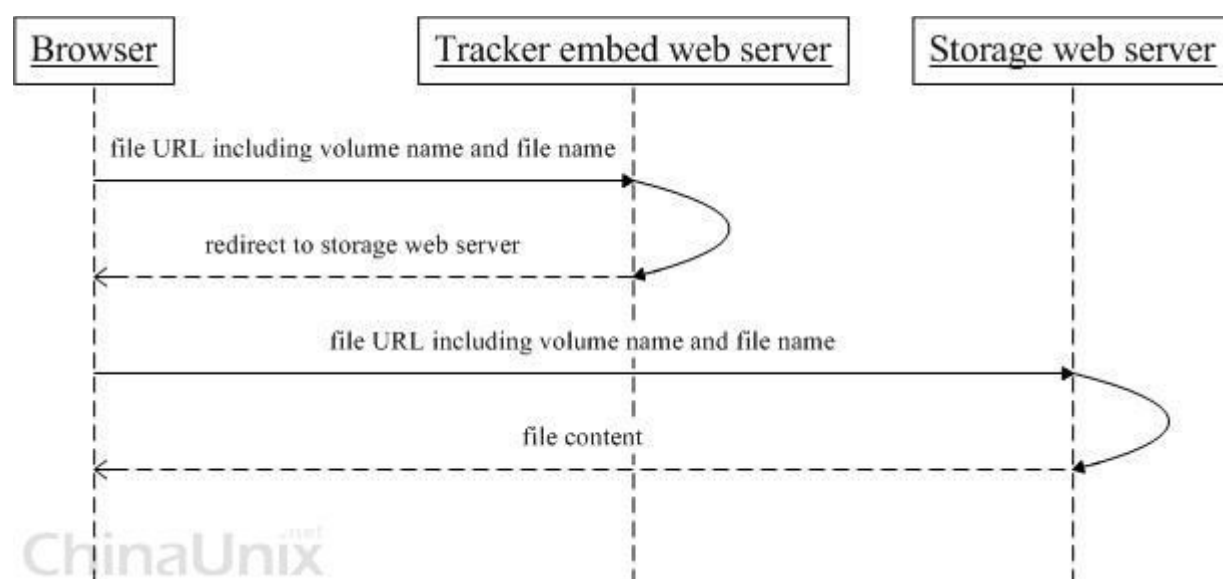
1. **trunk file id** 2. 文件在 **trunk file** 内部的 **offset** 3. 文件占用的存储空间大小 (字节对齐及删除空间复用，文件占用存储空间 >= 文件大小)

每个 **trunk file** 由一个 **id** 唯一标识，**trunk file** 由 **group** 内的 **trunk server** 负责创建 (**trunk server** 是 **tracker** 选出来的)，并同步到 **group** 内其他的 **storage**，文件存储合并存储到 **trunk file** 后，根据其 **offset** 就能从 **trunk file** 读取到文件。

文件在 **trunk file** 内的 **offset** 编码到文件名，决定了其在 **trunk file** 内的位置是不能更改的，也就不能通过 **compact** 的方式回收 **trunk file** 内删除文件的空间。但当 **trunk file** 内有文件删除时，其删除的空间是可以被复用的，比如一个 100KB 的文件被删除，接下来存储一个 99KB 的文件就可以直接复用这片删除的存储空间。

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。



其他特性

FastDFS 提供了设置/获取文件扩展属性的接口（setmeta/getmeta），扩展属性以 key-value 对的方式存储在 storage 上的同名文件（拥有特殊的前缀或后缀），比如/group/M00/00/01/some_file 为原始文件，则该文件的扩展属性存储在/group/M00/00/01/.some_file.meta 文件（真实情况不一定是这样，但机制类似），这样根据文件名就能定位到存储扩展属性的文件。

以上两个接口作者不建议使用，额外的 meta 文件会进一步“放大”海量小文件存储问题，同时由于 meta 非常小，其存储空间利用率也不高，比如 100bytes 的 meta 文件也需要占用 4K（block_size）的存储空间。

FastDFS 还提供 appender file 的支持，通过 upload_appender_file 接口存储，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。

二、FastDFS 分布式文件系统的安装与使用

跟踪 服务器：192.168.4.121

存储 服务器 : 192.168.4.125

环 境 : CentOS 6.6

用 户 : root

数 据 目录 : / / fastdfs (注 : 数据目录按你的数据盘挂载路径而定)

安 装 包 :

FastDFS v5.05

libfastcommon- - master.zip (是从 FastDFS 和 FastDHT 中提取出来的公共 C 函数库)

fastdfs- - nginx- - module_v1.16.tar.gz

nginx- - 1.6.2.tar.gz

fastdfs_client_java._v1.25.tar.gz

源码地址 : <https://github.com/happyfish100/>

下载地址 : <http://sourceforge.net/projects/fastdfs/files/>

官方论坛 : <http://bbs.chinaunix.net/forum-240-1.html>

1、 所 有 跟 踪 服务器 和 存储服务器均执行 如下操作

1.1、 编译和安装所需的依赖包:

```
# yum install make cmake gcc gcc-c++
```

1.2、 安装 libfastcommon:

(1)上传或下载 libfastcommon-master.zip 到/usr/local/src 目录

(2)解压

```
# cd /usr/local/src/
```

```
# unzip libfastcommon-master.zip
```

```
# cd libfastcommon-master
```

```
[root@edu-dfs-tracker-01 libfastcommon-master]# ll
total 28
-rw-r--r--. 1 root root 2913 Feb 27 17:27 HISTORY
-rw-r--r--. 1 root root 582 Feb 27 17:27 INSTALL
-rw-r--r--. 1 root root 1342 Feb 27 17:27 libfastcommon.spec
-rwxr-xr-x. 1 root root 2151 Feb 27 17:27 make.sh
drwxr-xr-x. 2 root root 4096 Feb 27 17:27 php-fastcommon
-rw-r--r--. 1 root root 617 Feb 27 17:27 README
drwxr-xr-x. 2 root root 4096 Feb 27 17:27 src
```

(3) 编译、安装

```
# ./make.sh
```

```
# ./make.sh install
```

libfastcommon 默认安装到了

/usr/lib64/libfastcommon.so

/usr/lib64/libbdfsclient.so

(4)因为 FastDFS 主程序设置的 lib 目录是/usr/local/lib，所以需要创建软链接。

```
# ln -s /usr/lib64/libfastcommon.so /usr/local/lib/libfastcommon.so
```

```
# ln -s /usr/lib64/libfastcommon.so /usr/lib/libfastcommon.so
```

```
# ln -s /usr/lib64/libbdfsclient.so /usr/local/lib/libbdfsclient.so
```

```
# ln -s /usr/lib64/libbdfsclient.so /usr/lib/libbdfsclient.so
```

1.3、安装 FastDFS

(1)上传或下载 FastDFS 源码包（FastDFS_v5.05.tar.gz）到
/usr/local/src 目录

(2)解压

```
# cd /usr/local/src/
```

```
# tar -zxvf FastDFS_v5.05.tar.gz
```

```
# cd FastDFS
```

```
[root@edu-dfs-tracker-01 FastDFS]# ll
total 132
drwxr-xr-x. 3 8980 users 4096 Dec  2 11:26 client
drwxr-xr-x. 2 8980 users 4096 Dec  2 11:27 common
drwxr-xr-x. 2 8980 users 4096 Dec  2 11:26 conf
-rw-r--r--. 1 8980 users 35067 Dec  2 11:26 COPYING-3_0.txt
-rw-r--r--. 1 8980 users 2802 Dec  2 11:26 fastdfs.spec
-rw-r--r--. 1 8980 users 31386 Dec  2 11:27 HISTORY
drwxr-xr-x. 2 8980 users 4096 Dec  2 11:26 init.d
-rw-r--r--. 1 8980 users 7755 Dec  2 11:26 INSTALL
-rwxr-xr-x. 1 8980 users 5813 Dec  2 11:27 make.sh
drwxr-xr-x. 2 8980 users 4096 Dec  2 11:26 php_client
-rw-r--r--. 1 8980 users 2380 Dec  2 11:26 README.md
-rwxr-xr-x. 1 8980 users 1768 Dec  2 11:26 restart.sh
-rwxr-xr-x. 1 8980 users 1680 Dec  2 11:26 stop.sh
drwxr-xr-x. 4 8980 users 4096 Dec  2 11:27 storage
drwxr-xr-x. 2 8980 users 4096 Dec  2 11:26 test
drwxr-xr-x. 2 8980 users 4096 Dec  2 11:27 tracker
```

(3)编译、安装（编译前要确保已经成功安装了

libfastcommon）

```
# ./make.sh
```

```
# ./make.sh install
```

采用默认安装的方式安装,安装后的相应文件与目录：

A、服务脚本在：

```
/etc/init.d/fdfs_storaged
```

```
/etc/init.d/fdfs_tracker
```

B、配置文件在（ 样例配置文件 ）：

```
/etc/fdfs/client.conf.sample
```

```
/etc/fdfs/storage.conf.sample
```

```
/etc/fdfs/tracker.conf.sample
```

C、命令工具在/usr/bin/目录下的：

```
fdfs_appender_test
```

```
fdfs_appender_test1
```

```
fdfs_append_file
```

```
fdfs_crc32
```

```
fdfs_delete_file
```

fdfs_download_file
fdfs_file_info
fdfs_monitor
fdfs_storaged
fdfs_test
fdfs_test1
fdfs_trackerd
fdfs_upload_appender
fdfs_upload_file
stop.sh
restart.sh

(4)因为 FastDFS 服务脚本设置的 bin 目录是/usr/local/bin，但实际命令安装在/usr/bin

可以进入/usr/bin 目录使用以下命令查看 fdfs 的相关命令：

```
# cd /usr/bin/  
# ls | grep fdfs
```

```
[root@edu-dfs-tracker-01 ~]# cd /usr/bin/  
[root@edu-dfs-tracker-01 bin]# ls | grep fdfs  
fdfs_appender_test  
fdfs_appender_test1  
fdfs_append_file  
fdfs_crc32  
fdfs_delete_file  
fdfs_download_file  
fdfs_file_info  
fdfs_monitor  
fdfs_storaged  
fdfs_test  
fdfs_test1  
fdfs_trackerd  
fdfs_upload_appender  
fdfs_upload_file  
[root@edu-dfs-tracker-01 bin]#
```

因此需要修改 FastDFS 服务脚本中相应的命令路径，也就是把/etc/init.d/fdfs_storaged

和/etc/init.d/fdfs_tracker 两个脚本中的/usr/local/bin 修改成/usr/bin :

```
# vi fdfs_trackerd
```

使用查找替换命令进统一修改:%s+/usr/local/bin+/usr/bin

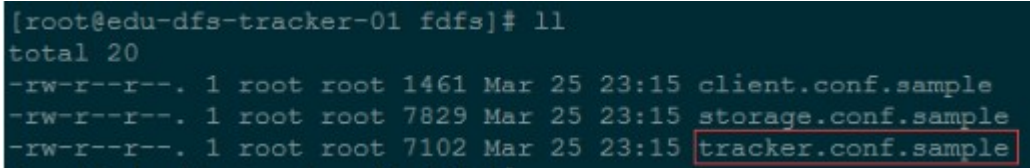
```
# vi fdfs_storaged
```

使用查找替换命令进统一修改:%s+/usr/local/bin+/usr/bin

2、 配置 置 S FastDFS 跟踪 器((192.168.4.121))

2.1、 复制 FastDFS 跟踪器样例配置文件,并重命名:

```
# cd /etc/fdfs/
```



```
[root@edu-dfs-tracker-01 fdfs]# ll
total 20
-rw-r--r--. 1 root root 1461 Mar 25 23:15 client.conf.sample
-rw-r--r--. 1 root root 7829 Mar 25 23:15 storage.conf.sample
-rw-r--r--. 1 root root 7102 Mar 25 23:15 tracker.conf.sample
```

```
# cp tracker.conf.sample tracker.conf
```

2.2、 编辑跟踪器配置文件 :

```
# vi /etc/fdfs/tracker.conf
```

修改的内容如下 :

```
disabled=false
```

```
port=22122
```

```
base_path=/fastdfs/tracker
```

(其它参数保留默认配置 , 具体配置解释请参考官方文档说明 :

<http://bbs.chinaunix.net/thread-1941456-1-1.html>)

2.3、创建基础数据目录（参考基础目录 `base_path` 配置）：

```
# mkdir -p /fastdfs/tracker
```

2.4、防火墙中打开跟踪器端口（默认为 22122）：

```
# vi /etc/sysconfig/iptables
```

添加如下端口行：

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22122 -j ACCEPT
```

重启防火墙：

```
# service iptables restart
```

2.5、启动 Tracker：

```
# /etc/init.d/fdfs_trackerd start
```

（初次成功启动，会在 `/fastdfs/tracker` 目录下创建 `data`、`logs` 两个目录）

查看 FastDFS Tracker 是否已成功启动：

```
# ps -ef | grep fdfs
```

```
[root@edu-dfs-tracker-01 init.d]# ps -ef | grep fdfs
root      2415      1  0 01:47 ?        00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
root      2423    2160  0 01:47 pts/1    00:00:00 grep fdfs
[root@edu-dfs-tracker-01 init.d]#
```

2.6、关闭 Tracker：

```
# /etc/init.d/fdfs_trackerd stop
```

2.7、设置 FastDFS 跟踪器开机启动：

```
# vi /etc/rc.d/rc.local
```

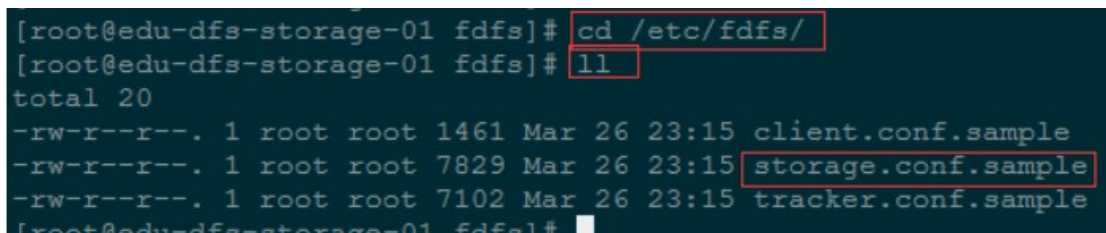
添加以下内容：


```
## FastDFS Tracker  
/etc/init.d/fdfs_trackerd start
```

3、配置 置 S FastDFS 存储((192.168.4.125))

3.1、复制 FastDFS 存储器样例配置文件,并重命名:

```
# cd /etc/fdfs/
```



```
[root@edu-dfs-storage-01 fdfs]# cd /etc/fdfs/  
[root@edu-dfs-storage-01 fdfs]# ll  
total 20  
-rw-r--r--. 1 root root 1461 Mar 26 23:15 client.conf.sample  
-rw-r--r--. 1 root root 7829 Mar 26 23:15 storage.conf.sample  
-rw-r--r--. 1 root root 7102 Mar 26 23:15 tracker.conf.sample  
[root@edu-dfs-storage-01 fdfs]#
```

```
# cp storage.conf.sample storage.conf
```

3.2、编辑存储器样例配置文件：

```
# vi /etc/fdfs/storage.conf
```

修改的内容如下:

```
disabled=false
```

```
port=23000
```

```
base_path=/fastdfs/storage
```

```
store_path0=/fastdfs/storage
```

```
tracker_server=192.168.4.121:22122
```

```
http.server_port=8888
```

(其它参数保留默认配置，具体配置解释请参考官方文档说明：

<http://bbs.chinaunix.net/thread-1941456-1-1.html>)

3.3、创建基础数据目录（参考基础目录 base_path 配置）：

```
# mkdir -p /fastdfs/storage
```

3.4、防火墙中打开存储器端口（默认为 23000）：

```
# vi /etc/sysconfig/iptables
```

添加如下端口行：

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 23000 -j ACCEPT
```

重启防火墙：

```
# service iptables restart
```

3.5、启动 Storage：

```
# /etc/init.d/fdfs_storaged start
```

（初次成功启动，会在 /fastdfs/storage 目录下创建 data、logs 两个目录）

查看 FastDFS Storage 是否已成功启动

```
# ps -ef | grep fdfs
```

```
[root@edu-dfs-storage-01 ~]# ps -ef | grep fdfs
root      2061      1  0 01:45 ?        00:00:00 /usr/bin/fdfs_storaged /etc/fdfs/storage.conf
root      2074    2007  0 01:45 pts/1    00:00:00 grep fdfs
[root@edu-dfs-storage-01 ~]#
```

3.6、关闭 Storage：

```
# /etc/init.d/fdfs_storaged stop
```

3.7、 设置 FastDFS 存储器开机启动：

```
# vi /etc/rc.d/rc.local  
  
添加：  
  
## FastDFS Storage  
  
/etc/init.d/fdfs_storaged start
```

4、 文件 上传 测试((192.168.4.121))

4.1、 修改 Tracker 服务器中的客户端配置文件：

```
# cp /etc/fdfs/client.conf.sample /etc/fdfs/client.conf  
  
# vi /etc/fdfs/client.conf  
  
base_path=/fastdfs/tracker  
  
tracker_server=192.168.4.121:22122
```

4.2、 执行如下文件上传命令：

```
# /usr/bin/fdfs_upload_file /etc/fdfs/client.conf  
/usr/local/src/FastDFS_v5.05.tar.gz
```

返回 ID 号：group1/M00/00/00/wKgEfVUYNYeAb7XFAAVFOL7FJU4.tar.gz

(能返回以上文件 ID，说明文件上传成功)

5、在 每个 存储 节 点上 安装 nginx

5.1、fastdfs-nginx-module 作用说明

FastDFS 通过 Tracker 服务器,将文件放在 Storage 服务器存储,但是同组存储服务器之间需要进入

文件复制,有同步延迟的问题。假设 Tracker 服务器将文件上传到了 192.168.4.125,上传成功后文件 ID

已经返回给客户端。此时 FastDFS 存储集群机制会将这个文件同步到同组存储 192.168.4.126,在文件还

没有复制完成的情况下,客户端如果用这个文件 ID 在 192.168.4.126 上取文件,就会出现文件无法访问的

错误。而 fastdfs-nginx-module 可以重定向文件连接到源服务器取文件,避免客户端由于复制延迟导致的

文件无法访问错误。(解压后的 fastdfs-nginx-module 在 nginx 安装时使用)

5.2、上传 fastdfs-nginx-module_v1.16.tar.gz

位置: /usr/local/src

5.3、解压

```
# cd /usr/local/src/  
# tar -zxvf fastdfs-nginx-module_v1.16.tar.gz
```

5.4、修改 fastdfs-nginx-module 的 config 配置文件

```
# cd fastdfs-nginx-module/src
```

```
# vi config
```

```
CORE_INCS="$CORE_INCS /usr/local/include/fastdfs  
/usr/local/include/fastcommon/"
```

修改为：

```
CORE_INCS="$CORE_INCS /usr/include/fastdfs /usr/include/fastcommon/"
```

（注意：这个路径修改是很重要的，不然在 nginx 编译的时候会报错的）

5.5、上传当前的稳定版本 Nginx(nginx-1.6.2.tar.gz)

位置：/usr/local/src 目录

5.6、安装编译 Nginx 所需的依赖包

```
# yum install gcc gcc-c++ make automake autoconf libtool pcre* zlib openssl  
openssl-devel
```

5.7、编译安装 Nginx (添加 fastdfs-nginx-module 模块)

```
# cd /usr/local/src/
```

```
# tar -zxvf nginx-1.6.2.tar.gz
```

```
# cd nginx-1.6.2
```

```
# ./configure --add-module=/usr/local/src/fastdfs-nginx-module/src
```

```
# make && make install
```

5.8、复制 fastdfs-nginx-module 源码中的配置文件

复制 fastdfs-nginx-module 到 /etc/fdfs 目录，并修改

```
# cp /usr/local/src/fastdfs-nginx-module/src/mod_fastdfs.conf /etc/fdfs/  
# vi /etc/fdfs/mod_fastdfs.conf
```

修改以下配置：

```
connect_timeout=10  
base_path=/tmp  
tracker_server=192.168.4.121:22122  
storage_server_port=23000  
group_name=group1  
url_have_group_name = true  
store_path0=/fastdfs/storage
```

5.9、复制 FastDFS 的部分配置文件到/etc/fdfs 目录

```
# cd /usr/local/src/FastDFS/conf  
# cp http.conf mime.types /etc/fdfs/
```

5.10、在/fastdfs/storage 文件存储目录下创建软连接

创建软连接将其链接到实际存放数据的目录

```
# ln -s /fastdfs/storage/data/ /fastdfs/storage/data/M00
```

5.11、配置 Nginx

简洁版 nginx 配置样例：

```
user root;
```

```

worker_processes 1;

events {
worker_connections 1024;
}

http {
include mime.types;
default_type application/octet-stream;
sendfile on;
keepalive_timeout 65;

server {
listen 8888;

server_name localhost;

location ~/group([0-9])/M00 {
#alias /fastdfs/storage/data;
ngx_fastdfs_module;
}

error_page 500 502 503 504 /50x.html;

location = /50x.html {
root html;
}
}
}

```

注意、说明：

A、8888 端口值是要与/etc/fdfs/storage.conf 中的 http.server_port=8888 相对应，

因为 http.server_port 默认为 8888,如果想改成 80，则要对对应修改过来。

B、Storage 对应有多组 group 的情况下，访问路径带 group 名，
如/group1/M00/00/00/xxx，

对应的 Nginx 配置为：


```
location ~/group([0-9])/M00 {  
    ngx_fastdfs_module;  
}
```

C、如查下载时如发现老报 404，将 nginx.conf 第一行 user nobody 修改为 user root 后重新启动。

5.12、防火墙中打开 Nginx 的 8888 端口

```
# vi /etc/sysconfig/iptables
```

添加：

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8888 -j ACCEPT
```

```
# service iptables restart
```

5.13、启动 Nginx

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

```
ngx_http_fastdfs_set pid=xxx
```

(重启 Nginx 的命令为：/usr/local/nginx/sbin/nginx -s reload)

5.14、通过浏览器访问测试时上传的文件

<http://192.168.4.125:8888/group1/M00/00/00/wKgEfVUYNYeAb7XFAAVFOL7FJU4.tar.gz>

三、java 客户端测试代码

<FastDFSTest.java>

```
package com.test;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

import org.csource.common.NameValuePair;
import org.csource.fastdfs.ClientGlobal;
import org.csource.fastdfs.FileInfo;
import org.csource.fastdfs.StorageClient;
import org.csource.fastdfs.StorageServer;
import org.csource.fastdfs.TrackerClient;
import org.csource.fastdfs.TrackerServer;

public class FastDFSTest {

    public static void main(String[] args) throws Exception {
        //加载配置文件的方式
        String configFileName = "src/main/resources/fdfs_client.conf";
        try {
            ClientGlobal.init(configFileName);
        } catch (Exception e) {
            e.printStackTrace();
        }

        /*
         * 也可以使用代码设置配置
         * // 连接超时的时限
            ClientGlobal.setG_connect_timeout(2);
            // 网络超时的时限，单位为秒
            ClientGlobal.setG_network_timeout(30);
            ClientGlobal.setG_anti_steal_token(false);
            // 字符集
            ClientGlobal.setG_charset("UTF-8");
            ClientGlobal.setG_secret_key(null);
            // HTTP 访问服务的端口号
        */
    }
}
```

```

ClientGlobal.setG_tracker_http_port(8088);
// Tracker 服务器列表
InetSocketAddress[] tracker_servers = new InetSocketAddress[1];
tracker_servers[0] = new InetSocketAddress("192.168.18.43", 22122);
ClientGlobal.setG_tracker_group(new TrackerGroup(tracker_servers));
*/
File file = new File("src/main/resources/test.jpg");
//返回储存路径:group1 M00/00/00/wKhuWlVmJ6KAZ09pAAC9przUxEk788.jpg
String[] files = uploadFile(file, "test.jpg", file.length());
System.out.println(Arrays.asList(files));

//下载文件
downloadFile(files[0], files[1]);
//查看文件信息
getFileInfo(files[0], files[1]);
getFileMate(files[0], files[1]);
//deleteFile(files[0], files[1]);
}

/**
 * 上传文件
 */
public static String[] uploadFile(File file, String uploadFileName,
long fileLength) throws IOException {
    System.out.println("上传文件=====");
    byte[] fileBuff = getFileBuffer(new FileInputStream(file),
fileLength);
    String[] files = null;
    String fileExtName = "";
    if (uploadFileName.contains(".")) {
        fileExtName =
uploadFileName.substring(uploadFileName.lastIndexOf(".") + 1);
    } else {
        System.out.println("Fail to upload file, because the format of
filename is illegal.");
        return null;
    }

    // 建立连接
    TrackerClient tracker = new TrackerClient();
    TrackerServer trackerServer = tracker.getConnection();
    StorageServer storageServer = null;
    StorageClient client = new StorageClient(trackerServer,
storageServer);

    // 设置元信息
    NameValuePair[] metaList = new NameValuePair[3];
    metaList[0] = new NameValuePair("fileName", uploadFileName);
    metaList[1] = new NameValuePair("fileExtName", fileExtName);

```

```

        metaList[2] = new NameValuePair("fileLength",
String.valueOf(fileLength));

        // 上传文件
        try {
            files = client.upload_file(fileBuff, fileExtName, metaList);
        } catch (Exception e) {
            System.out.println("Upload file \"" + uploadFileName +
"\fails");
        }
        trackerServer.close();
        return files;
    }

    private static byte[] getFileBuffer(InputStream inStream, long
fileLength) throws IOException {

        byte[] buffer = new byte[256 * 1024];
        byte[] fileBuffer = new byte[(int) fileLength];

        int count = 0;
        int length = 0;

        while ((length = inStream.read(buffer)) != -1) {
            for (int i = 0; i < length; ++i) {
                fileBuffer[count + i] = buffer[i];
            }
            count += length;
        }
        return fileBuffer;
    }

    //下载文件
    public static void downloadFile(String groupName,String filepath)
throws Exception{
        System.out.println("下载文件=====");
        TrackerClient tracker = new TrackerClient();
        TrackerServer trackerServer = tracker.getConnection();
        StorageServer storageServer = null;

        StorageClient storageClient = new StorageClient(trackerServer,
storageServer);
        byte[] b = storageClient.download_file(groupName, filepath);
        System.out.println("文件大小:"+b.length);
        String fileName = "src/main/resources/test1.jpg";
        FileOutputStream out = new FileOutputStream(fileName);
        out.write(b);
        out.flush();
        out.close();
    }

```

```

//查看文件信息
public static void getFileInfo(String groupName, String filepath) throws
Exception{
    System.out.println("获取文件信息=====");
    TrackerClient tracker = new TrackerClient();
    TrackerServer trackerServer = tracker.getConnection();
    StorageServer storageServer = null;

    StorageClient storageClient = new StorageClient(trackerServer,
storageServer);
    FileInfo fi = storageClient.get_file_info(groupName, filepath);
    System.out.println("所在服务器地址:"+fi.getSourceIpAddr());
    System.out.println("文件大小:"+fi.getFileSize());
    System.out.println("文件创建时间:"+fi.getCreateTimestamp());
    System.out.println("文件CRC32 signature:"+fi.getCrc32());
}

public static void getFileMate(String groupName, String filepath) throws
Exception{
    System.out.println("获取文件 Mate=====");
    TrackerClient tracker = new TrackerClient();
    TrackerServer trackerServer = tracker.getConnection();
    StorageServer storageServer = null;

    StorageClient storageClient = new StorageClient(trackerServer,
storageServer);
    NameValuePair nvps[] =
storageClient.get_metadata(groupName, filepath);
    for (NameValuePair nvp : nvps) {
        System.out.println(nvp.getName() + ":" + nvp.getValue());
    }
}

public static void deleteFile(String groupName, String filepath) throws
Exception{
    System.out.println("删除文件=====");
    TrackerClient tracker = new TrackerClient();
    TrackerServer trackerServer = tracker.getConnection();
    StorageServer storageServer = null;
    StorageClient storageClient = new StorageClient(trackerServer,
storageServer);
    int i = storageClient.delete_file(groupName, filepath);
    System.out.println(i == 0 ? "删除成功" : "删除失败:" + i);
}
}

```

<fdfs_client.conf>

connect_timeout =2

network_timeout = 30

charset = UTF-8

http.tracker_http_port = 8080

http.anti_steal_token = no

http.secret_key = FastDFS1234567890

tracker_server = 192.168.0.128:22122

#tracker_server = 192.168.0.131:22122

#storage_server = 192.168.199.205:23000 #no need here