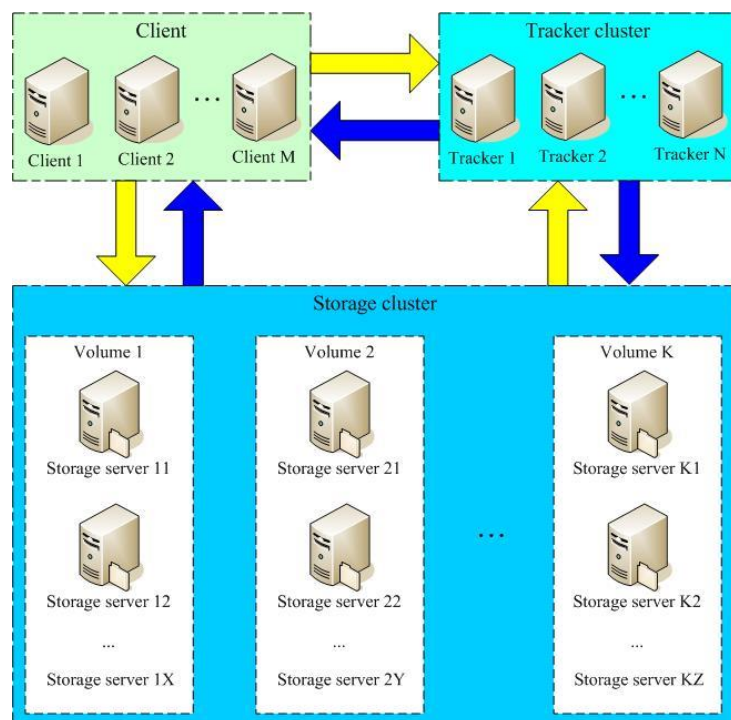


1. FastDFS 介绍

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，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。

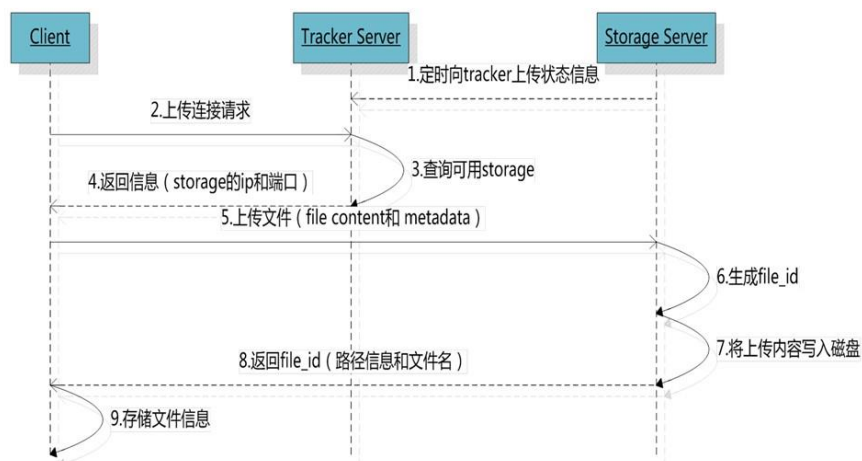
Tracker server

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. Specified group, 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。



group1/M00/00/0C/wKjRbExx2K0AAAAAANiSQUgyg37275.h

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。

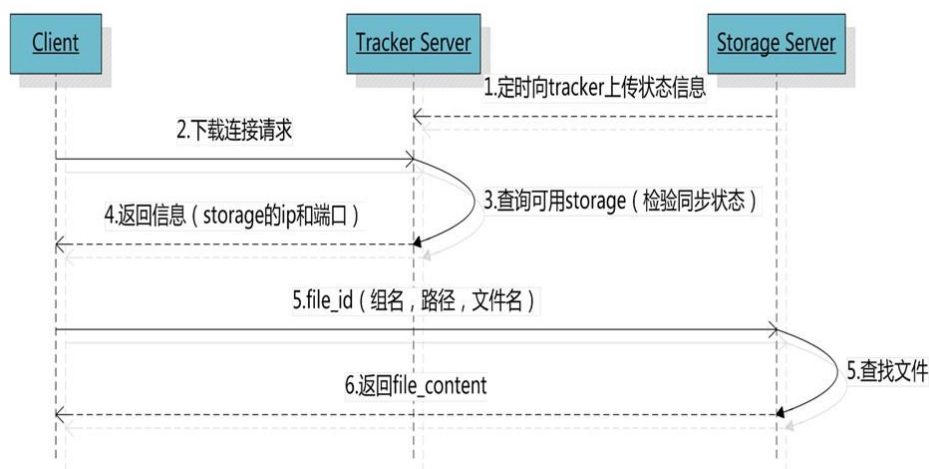
每个 storage 写文件后，同时会写一份 binlog，binlog 里不包含文件数据，只包含文件名等元信息，这份 binlog 用于后台同步，storage 会记录向 group 内其他 storage 同步的进度，以便重启后能接上次的进度继续同步；进度以时间戳的方式进行记录，所以最好能保证集群内所有 server 的时钟保持同步。

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 以前写的文件都已经同步到 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，tracker 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 $(\text{当前时间} - \text{文件创建时间戳}) > \text{文件同步最大时间 (如 5 分钟)} - \text{文件创建后}$ ，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. $(\text{当前时间} - \text{文件创建时间戳}) > \text{同步延迟阈值 (如一天)}$ 。 - 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

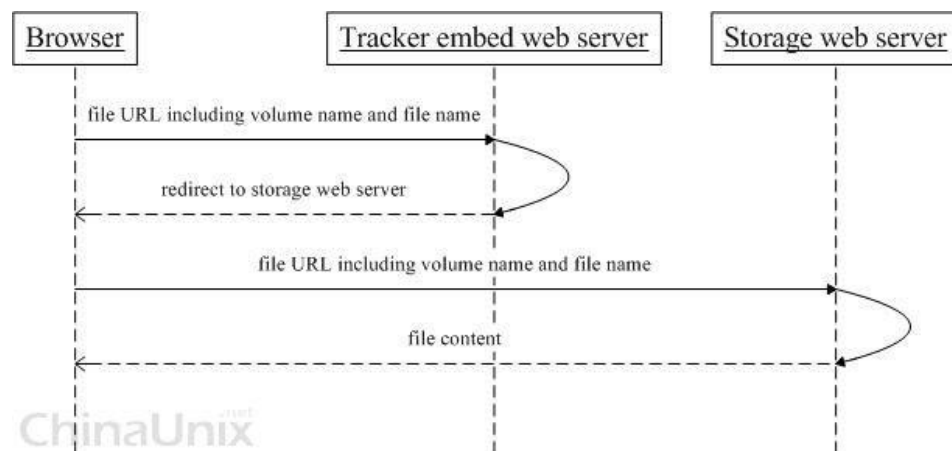
1. trunk file id
2. 文件在 trunk file 内部的 offset
3. 文件占用的存储空间大小, 字节对齐及删除空间复用, 文件占用存储空间 \geq 文件大小

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

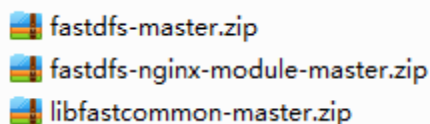
FastDFS 提供了设置/获取文件扩展属性的接口 (setmeta/getmeta)，扩展属性以 key-value 对的方式存储在 storage 上的同名文件（拥有特殊的前缀或后缀），比如 /group/M00/00/01/some_file 为原始文件，则该文件的扩展属性存储在 /group/M00/00/01/.some_file.meta 文件（真实情况不一定是这样，但机制类似），这样根据文件名就能定位到存储扩展属性的文件。

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。

2. 安装单机版 FastDFS

将安装包上传到服务器的 /usr/local/software 目录下



fastdfs-master.zip
fastdfs-nginx-module-master.zip
libfastcommon-master.zip

2.1 安装所需的依赖包

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

这个下载的过程很慢。中间会遇到两次确认，[Y/N]，选择 Y 确认

2.2 安装 libfastcommon

```
[root@bogon ~]# cd /usr/local/software/  
[root@bogon software]# unzip libfastcommon-master.zip
```

unzip libfastcommon-master.zip

进入解压后的路径，执行 ./make.sh

```
[root@bogon software]# cd libfastcommon-master  
[root@bogon libfastcommon-master]# ./make.sh
```

等编译完成后，执行 ./make.sh install


```

ger.lo sockopt.lo base64.lo sched_thread.lo http_func.lo md5.lo pthread_func.lo local_ip_func.lo
fast_task_queue.lo fast_timer.lo process_ctrl.lo fast_mblock.lo connection_pool.lo fast_mpool.lo
skiplist.lo flat_skiplist.lo system_info.lo fast_blocked_queue.lo id_generator.lo char_converter
ar rcs libfastcommon.a hash.o chain.o shared_func.o ini_file_reader.o logger.o sockopt.o base64.
ad_func.o local_ip_func.o avl_tree.o ioevent.o ioevent_loop.o fast_task_queue.o fast_timer.o pro
l.o fast_mpool.o fast_allocator.o fast_buffer.o multi_skiplist.o flat_skiplist.o system_info.o f
converter.o char_convert_loader.o
[root@bogon libfastcommon-master]# ./make.sh install

```

2.3 安装 FastDFS

返回/usr/local/software 目录，解压 fastdfs-master.zip

```
[root@bogon software]# unzip fastdfs-master.zip
```

unzip fastdfs-master.zip

进入解压后的目录，执行编译命令

```
cd fastdfs-master
```

```
./make.sh
```

```
[root@bogon software]# cd fastdfs-master
[root@bogon fastdfs-master]# ./make.sh

```

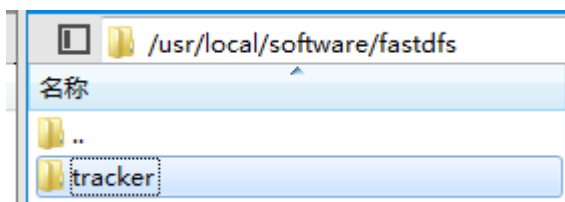
等编译成功后，进行安装

```
[root@bogon fastdfs-master]# ./make.sh install
```

```
cd ..
```

3.配置 Tracker 服务器

3.1 创建 base_path 目录



在 software 下创建 fastdfs 文件文件夹，在里面再写一个 tracker 文件夹。

创建文件夹的命令是：

你也可以把 tracker 创建在别的地方，只要在配置文件中填写好你存放数据的位置就行。

我存放数据的路径是/usr/local/software/fastdfs/tracker

3.2 修改 tracker 配置文件

复制 tracker 的配置文件并重命名，编辑配置文件

```
[root@bogon fastdfs-master]# cp /etc/fdfs/tracker.conf.sample /etc/fdfs/tracker.conf
[root@bogon fastdfs-master]# vim /etc/fdfs/tracker.conf
```

```
# the tracker server port
port=22122

# connect timeout in seconds
# default value is 30s
connect_timeout=30

# network timeout in seconds
# default value is 30s
network_timeout=60

# the base path to store data and log files
base_path=/usr/local/software/fastdfs/tracker

# max concurrent connections this server supported
```

存放数据的路径
文件夹必须提前创建好

3.3 开放端口

如果你直接关闭了防火墙，可以省略这一步。

使用 iptables 开放如下端口

```
/sbin/iptables -I INPUT -p tcp --dport 22122 -j ACCEPT
```

保存

```
/etc/rc.d/init.d/iptables save
```

重启服务

```
service iptables restart
```

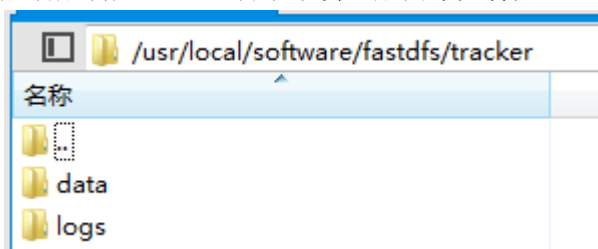
3.4 启动 tracker

```
[root@bogon fastdfs-master]# /etc/init.d/fdfs_trackerd start
Starting FastDFS tracker server:
[root@bogon fastdfs-master]#
```

启动命令：

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

启动成功后 tracker 目录下会生成两个文件夹



可以使用命令查看 tracker 的运行状态：

```
ps -ef | grep fdfs_trackerd
```



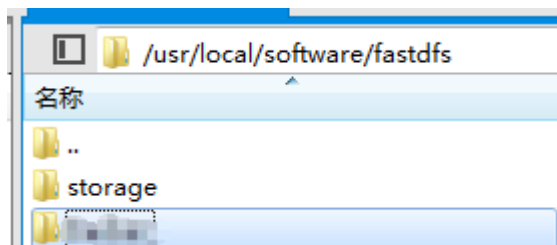
```
[root@bogon fastdfs-master]# ps -ef | grep fdfs_trackerd
root      23673      1  0 11:10 ?        00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
root      23681 22876  0 11:10 pts/1    00:00:00 grep fdfs_trackerd
```

4. 配置 Storage 服务器

注：实际开发中，tracker 和 storage 是在不同的服务器上的，所以安装 tracker 和 storage 都要执行第 1.1-1.3 节中的步骤。在本课程中，只使用一台服务器，所以配置 storage 的时候省略了 1.1-1.3 节中的操作。

4.1 创建 base_path 目录

```
mkdir -p /usr/local/software/fastdfs/storage
```



4.2 修改 storage 配置文件

```
[root@bogon fastdfs-master]# cp /etc/fdfs/storage.conf.sample /etc/fdfs/storage.conf
[root@bogon fastdfs-master]# vi /etc/fdfs/storage.conf
```

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

```
vim /etc/fdfs/storage.conf
```

```
# network timeout in seconds
# default value is 30s
network_timeout=60

# heart beat interval in seconds
heart_beat_interval=30

# disk usage report interval in seconds
stat_report_interval=60

# the base path to store data and log files
base_path=/usr/local/software/fastdfs/storage
```

往下找，还有个 store_path0 和 tracker_server, tracker_server 的 ip 就是 tracker 服务器的 ip

```
# path(disk or mount point) count, default value is 1
store_path_count=1

# store_path#, based 0, if store_path0 not exists, it's value is base_path
# the paths must be exist
store_path0=/usr/local/software/fastdfs/storage
#store_path1=/home/yuqing/fastdfs2

# sub_dir_count * sub_dir_count directories will be auto created under each
# store_path (disk), value can be 1 to 256, default value is 256
sub_dir_count_per_path=256

# tracker_server can occur more than once, and tracker_server format is
# "host:port", host can be hostname or ip address
tracker_server=10.0.31.250:22122
```

4.3 开放端口

如果你直接关闭了防火墙，可以省略这一步。

使用 iptables 开放如下端口

```
/sbin/iptables -I INPUT -p tcp --dport 23000 -j ACCEPT
```

保存

```
/etc/rc.d/init.d/iptables save
```

重启服务

```
service iptables restart
```

4.4 启动 storage

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

pwd 使用命令查看 storage 的运行状态：

```
ps -ef | grep fdfs_storaged
```

```
[root@bogon fastdfs-master]# /etc/init.d/fdfs_storaged start
Starting FastDFS storage server:
[root@bogon fastdfs-master]# ps -ef | grep fdfs_storaged
root      23719      1   0 11:23 ?        00:00:00 /usr/bin/fdfs_storaged /etc/fdfs/storage.conf
root      23721 22876   0 11:24 pts/1    00:00:00 grep fdfs_storaged
[root@bogon fastdfs-master]#
```

5. 测试

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

```
vim /etc/fdfs/client.conf
```

修改下面这两行为 tracker 的配置

```
# the base path to store log files
base_path=/usr/local/software/fastdfs/tracker

# tracker_server can occur more than once, and tracker_server format is
# "host:port" host can be hostname or ip address
tracker_server=10.0.31.250:22122

# standard log level as syslog, case insensitive, value list:
```

保存并退出

执行

`/usr/bin/fdfs_upload_file /etc/fdfs/client.conf /usr/1712.txt`


```
[root@bogon fastdfs-master]# /usr/bin/fdfs_upload_file /etc/fdfs/client.conf /usr/1712.txt
group1/M00/00/00/CgAf-LoD2HiAAhtCAAABS0DuerI148.txt
```

最后一个/usr/1712.txt 是提前放到服务器上的文件。

返回的 group1/xxxxx 就是上传成功后的访问地址。暂时还不能通过 http 查看。

6. 与 nginx 整合

把压缩包上传到/usr/local/software 目录下

 fastdfs-nginx-module-master.zip

解压

`unzip fastdfs-nginx-module-master.zip`

进入解压后的路径

`cd /usr/local/software/fastdfs-nginx-module-master/src`

```
[root@localhost software]# cd fastdfs-nginx-module-master
[root@localhost fastdfs-nginx-module-master]# cd src
[root@localhost src]#
```

将 mod_fastdfs.conf 复制到/etc/fdfs 下

`cp mod_fastdfs.conf /etc/fdfs`

修改下面的配置：

```
connect_timeout=2

# network recv and send timeout in seconds
# default value is 30s
network_timeout=30

# the base path to store log files
base_path=/usr/local/software/fastdfs
```

```

# FastDFS tracker_server can occur more than once, and tracker_server format is
# "host:port", host can be hostname or ip address
# valid only when load_fdfs_parameters_from_tracker is true
tracker_server=www.vm.com:22122 fastDFS tracker的地址和端口

# the port of the local storage server
# the default value is 23000
storage_server_port=23000

# the group name of the local storage server
group_name=group1

# if the url / uri including the group name
# set to false when uri like /M00/00/00/xxx
# set to true when uri like ${group_name}/M00/00/00/xxx, such as group1/M00/xxx
# default value is false
url_have_group_name = true 改成true,否则无法查看文件

# path(disk or mount point) count, default value is 1
# must same as storage.conf
store_path_count=1

# store_path#, based 0, if store_path# not exists, it's value is base_path
# the paths must be exist
# must same as storage.conf
store_path0=/usr/local/software/fastdfs/storage fastDFS storage存储路径
store_path1=/home/yingqing/fastdfs1

```

将 libfdfsclient.so 拷贝至 /usr/lib 下

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

将 /usr/local/software/fastdfs-master/conf 下这两个文件复制过去

```

[root@localhost conf]# cp http.conf /etc/fdfs/
[root@localhost conf]# cp mime.types /etc/fdfs/

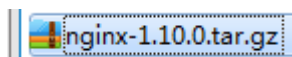
```

创建 nginx/client 目录

mkdir -p /var/temp/nginx/client

安装 nginx

将 nginx 压缩包上传到 /usr/local/software 下



安装 nginx

进入 /usr/local/software/nginx-1.10.0

安装依赖包

```

[root@localhost nginx-1.10.0]# yum -y install gcc pcre pcre-devel zlib zlib-devel openssl openssl-devel

```

yum -y install gcc pcre pcre-devel zlib zlib-devel openssl openssl-devel

这个下载的过程有点慢。

执行

./configure --prefix=/usr/local/software/nginx --sbin-path=/usr/bin/nginx

```
--add-module=/usr/local/software/fastdfs-nginx-module-master/src
```

编译后执行

```
make
```

再执行

```
make install
```

修改/usr/local/software/nginx/conf/nginx.conf

添加 server:

```
server {  
    listen      8888;  
    server_name localhost;  
  
    location /group1/M00/{  
        ngx_fastdfs_module;  
    }  
}
```

说明:

server_name 指定本机 ip

location /group1/M00/: group1 为 nginx 服务 FastDFS 的分组名称, M00 是 FastDFS 自动生成编号, 对应 store_path0, 如果 FastDFS 定义 store_path1, 这里就是 M01

8888 端口号与/etc/fdfs/storage.conf 中的 http.server_port=8888 相对应

启动 nginx

```
nginx
```

停止 nginx

```
nginx -s stop
```

重新启动

```
nginx -s reload
```

7.java 操作 fastDFS 文件上传

这个 maven 中央仓库里没有 jar 包, 需要下载工程:

<https://github.com/happyfish100/fastdfs-client-java>

安装到自己的 maven 仓库

导入 jar 包:

```
<!-- fastdfs -->  
<dependency>  
    <groupId>org.csource</groupId>  
    <artifactId>fastdfs-client-java</artifactId>
```

```
<version>1.27-SNAPSHOT</version>
</dependency>
```

创建配置文件 client.conf:

```
tracker_server=服务器的 ip 或域名:22122
```

测试方法:

```
import org.csource.fastdfs.*;

public class FastDFSTest {
    public static void main(String[] args) throws Exception {
        // 1、向工程中添加 jar 包
        // 2、创建一个配置文件。配置 tracker 服务器地址
        // 3、加载配置文件(绝对路径, 工程目录不要有中文)
        ClientGlobal.init(FastDFSTest.class.getResource("/").getPath() +
            "client.conf");
        // 4、创建一个 TrackerClient 对象。
        TrackerClient trackerClient = new TrackerClient();
        // 5、使用 TrackerClient 对象获得 trackerserver 对象。
        TrackerServer trackerServer = trackerClient.getConnection();
        // 6、创建一个 StorageServer 的引用 null 就可以。
        StorageServer storageServer = null;
        // 7、创建一个 StorageClient 对象。trackerserver、StorageServer 两个参数。
        StorageClient storageClient = new StorageClient(trackerServer, storageServer)
        // 8、使用 StorageClient 对象上传文件。
        String[] strings = storageClient.upload_file("f:/qf_logo.jpg", "jpg", null);
        for (String string : strings) {
            System.out.println(string);
        }
    }
}
```

封装工具类:

```
import org.csource.common.NameValuePair;
import org.csource.fastdfs.ClientGlobal;
import org.csource.fastdfs.StorageClient1;
import org.csource.fastdfs.StorageServer;
import org.csource.fastdfs.TrackerClient;
import org.csource.fastdfs.TrackerServer;

public class FastDFSClient {

    private TrackerClient trackerClient = null;
    private TrackerServer trackerServer = null;
    private StorageServer storageServer = null;
```

```

private StorageClient1 storageClient = null;

public FastDFSClient(String conf) throws Exception {
    if (conf.contains("classpath:")) {
        conf = conf.replace("classpath:",
this.getClass().getResource("/").getPath());
    }
    ClientGlobal.init(conf);
    trackerClient = new TrackerClient();
    trackerServer = trackerClient.getConnection();
    storageServer = null;
    storageClient = new StorageClient1(trackerServer, storageServer);
}

/**
 * 上传文件方法
 * <p>Title: uploadFile</p>
 * <p>Description: </p>
 * @param fileName 文件全路径
 * @param extName 文件扩展名, 不包含 (.)
 * @param metas 文件扩展信息
 * @return
 * @throws Exception
 */
public String uploadFile(String fileName, String extName, NameValuePair[] metas)
throws Exception {
    String result = storageClient.upload_file1(fileName, extName, metas);
    return result;
}

public String uploadFile(String fileName) throws Exception {
    return uploadFile(fileName, null, null);
}

public String uploadFile(String fileName, String extName) throws Exception {
    return uploadFile(fileName, extName, null);
}

/**
 * 上传文件方法
 * <p>Title: uploadFile</p>
 * <p>Description: </p>
 * @param fileContent 文件的内容, 字节数组
 * @param extName 文件扩展名

```



```
    * @param metas 文件扩展信息
    * @return
    * @throws Exception
    */
    public String uploadFile(byte[] fileContent, String extName, NameValuePair[] metas)
    throws Exception {

        String result = storageClient.upload_file1(fileContent, extName, metas);
        return result;
    }

    public String uploadFile(byte[] fileContent) throws Exception {
        return uploadFile(fileContent, null, null);
    }

    public String uploadFile(byte[] fileContent, String extName) throws Exception {
        return uploadFile(fileContent, extName, null);
    }
}
```

测试方法：

```
FastDFSClient fastDFSClient = new FastDFSClient(
    "classpath:client.conf");
String string = fastDFSClient.uploadFile("f:/logo.jpg");
System.out.println(string);
```

返回结果：

group1/M00/00/00/CscAbloEvACATPsCAALI54RJz6c951.jpg