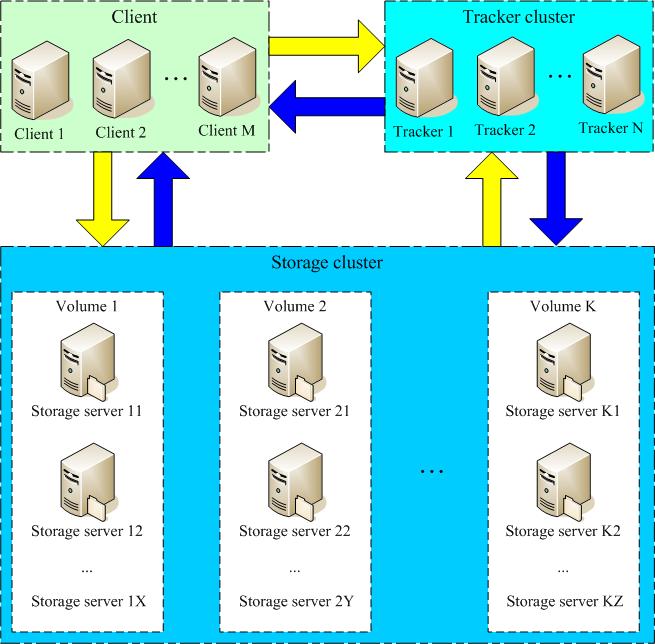
# FastDFS介绍

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，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。

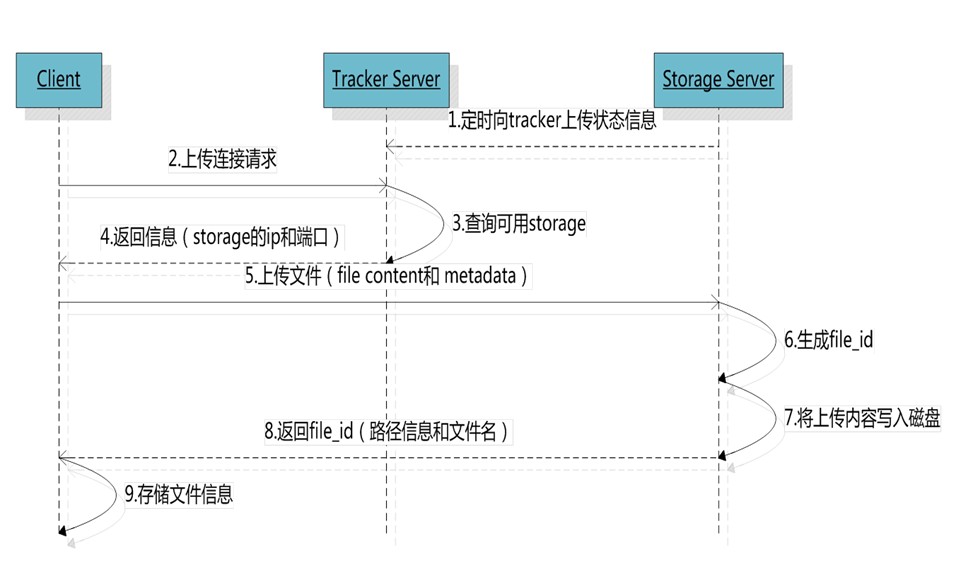
**Tracker server**

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stroage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trakcer。

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。



**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。

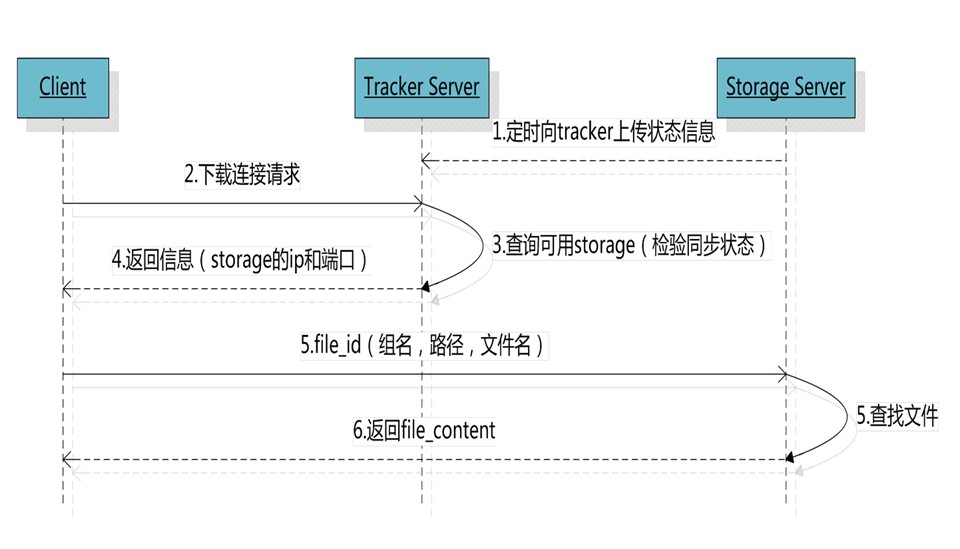
每个storage写文件后，同时会写一份binlog，binlog里不包含文件数据，只包含文件名等元信息，这份binlog用于后台同步，storage会记录向group内其他storage同步的进度，以便重启后能接上次的进度继续同步；进度以时间戳的方式进行记录，所以最好能保证集群内所有server的时钟保持同步。

storage的同步进度会作为元数据的一部分汇报到tracker上，tracke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（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

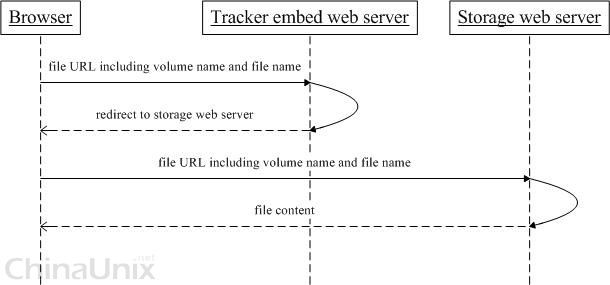
3. 文件占用的存储空间大小,字节对齐及删除空间复用，文件占用存储空间>=文件大小

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**

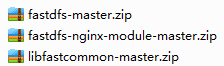
FastDFS提供了设置/获取文件扩展属性的接口（setmeta/getmeta)，扩展属性以key-value对的方式存储在storage上的同名文件（拥有特殊的前缀或后缀），比如/group/M00/00/01/some\_file为原始文件，则该文件的扩展属性存储在/group/M00/00/01/.some\_file.meta文件（真实情况不一定是这样，但机制类似），这样根据文件名就能定位到存储扩展属性的文件。

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。

# 安装单机版FastDFS

将安装包上传到服务器的/usr/local/software目录下



## 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

## 安装libfatscommon

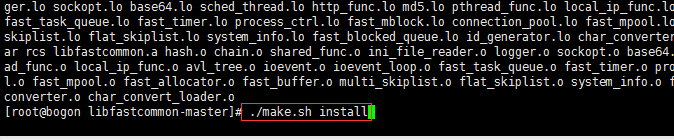


unzip libfastcommon-master.zip

进入解压后的路径，执行./make.sh



等编译完成后，执行./make.sh install



## 安装FastDFS

返回/usr/local/software目录，解压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



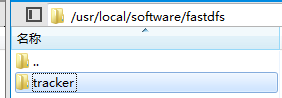
等编译成功后，进行安装



cd ..

# 配置Tracker服务器

## 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：

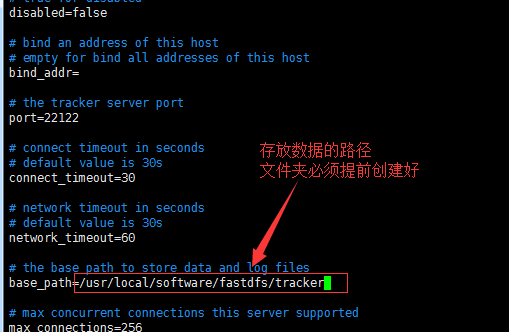
你也可以把tracker创建在别的地方，只要在配置文件中填写好你存放数据的位置就行。

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

## 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





## 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

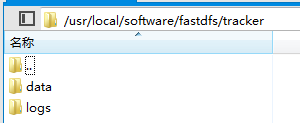
## 启动tracker



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

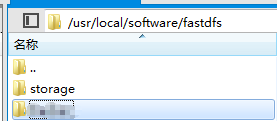


# 配置Storage服务器

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。

## 创建base\_path目录

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

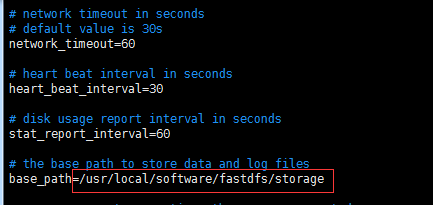


## 修改storage配置文件

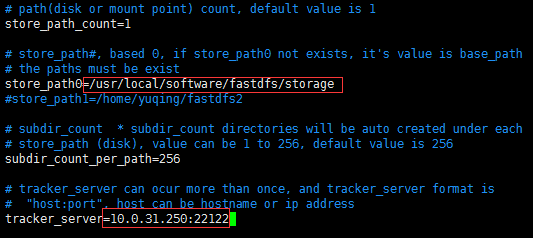


cp /etc/fdfs/storage.conf.sample /etc/fdfs/storage.conf

vim /etc/fdfs/storage.conf



往下找，还有个store\_path0和tracker\_server, tracker\_server的ip就是tracker服务器的ip



## 开放端口

如果你直接关闭了防火墙，可以省略这一步。

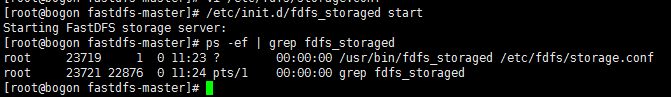
使用iptables开放如下端口  
/sbin/iptables -I INPUT -p tcp --dport 23000 -j ACCEPT  
保存  
/etc/rc.d/init.d/iptables save  
重启服务  
service iptables restart

## 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

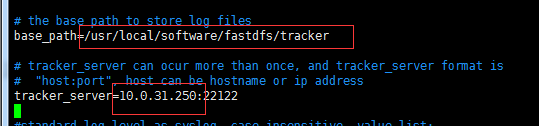


# 测试

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



保存并退出

执行

/usr/bin/fdfs\_upload\_file /etc/fdfs/client.conf /usr/1712.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。

# 与nginx整合

把压缩包上传到/usr/local/software目录下



解压

unzip fastdfs-nginx-module-master.zip

进入解压后的路径

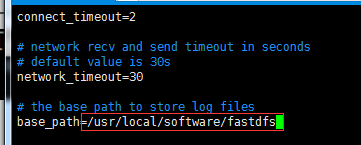
cd /usr/local/software/fastdfs-nginx-module-master/src

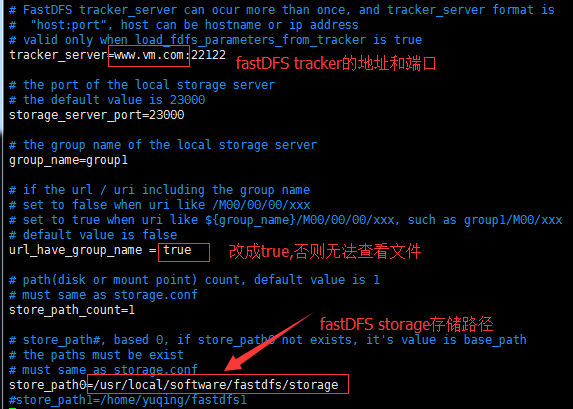


将mod\_fastdfs.conf复制到/etc/fdfs下

cp mod\_fastdfs.conf /etc/fdfs

修改下面的配置：





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



创建nginx/client目录

mkdir -p /var/temp/nginx/client

nginx依赖环境参照nginx安装文档

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



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

# java操作fastDFS文件上传

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/CscAbloEvACATPsCAALl54RJz6c951.jpg