优极限

“极限教育，挑战极限”

www.yjxxt.com

极限教育，挑战极限。优极限是一个让 95%的学生年薪过 18 万的岗前培训公司，让我们的学员具备优秀的互联网技术和职业素养，勇攀高薪，挑战极限。公司位于上海浦东，拥有两大校区，共万余平。累计培训学员超 3 万名。我们的训练营就业平均月薪 19000，最高年薪 50 万。

核心理念：让学员学会学习，拥有解决问题的能力，拿到高薪职场的钥匙。

项目驱动式团队协作、一对一服务、前瞻性思维、教练式培养模型-培养你成为就业明星。首创的老学员项目联盟给学员充分的项目、技术支撑，利用优极限平台这根杠杆，不断挑战极限，勇攀高薪，开挂人生。

扫码关注优极限微信公众号：

（获取最新技术相关资讯及更多源码笔记）

什么是文件系统

文件系统是操作系统用于在磁盘或分区上组织文件的方法和数据结构。磁盘空间是什么样的我们并不清楚，但文件系统可以给我们呈现一个非常清晰的表象，我们可以创建、删除、修改和复制这些文件，而实现这些功能的软件就是文件系统。操作系统中负责**管理和存储文件信息的软件被称为文件管理系统**，简称文件系统。

文件系统是操作系统的一个重要组成部分，通过对操作系统所管理的存储空间的抽象，向用户提供统一的、对象化的访问接口，屏蔽对物理设备的直接操作和资源管理。也就是说，**文件系统解决了普通用户使用磁盘存储数据的问题。**

文件系统的发展史

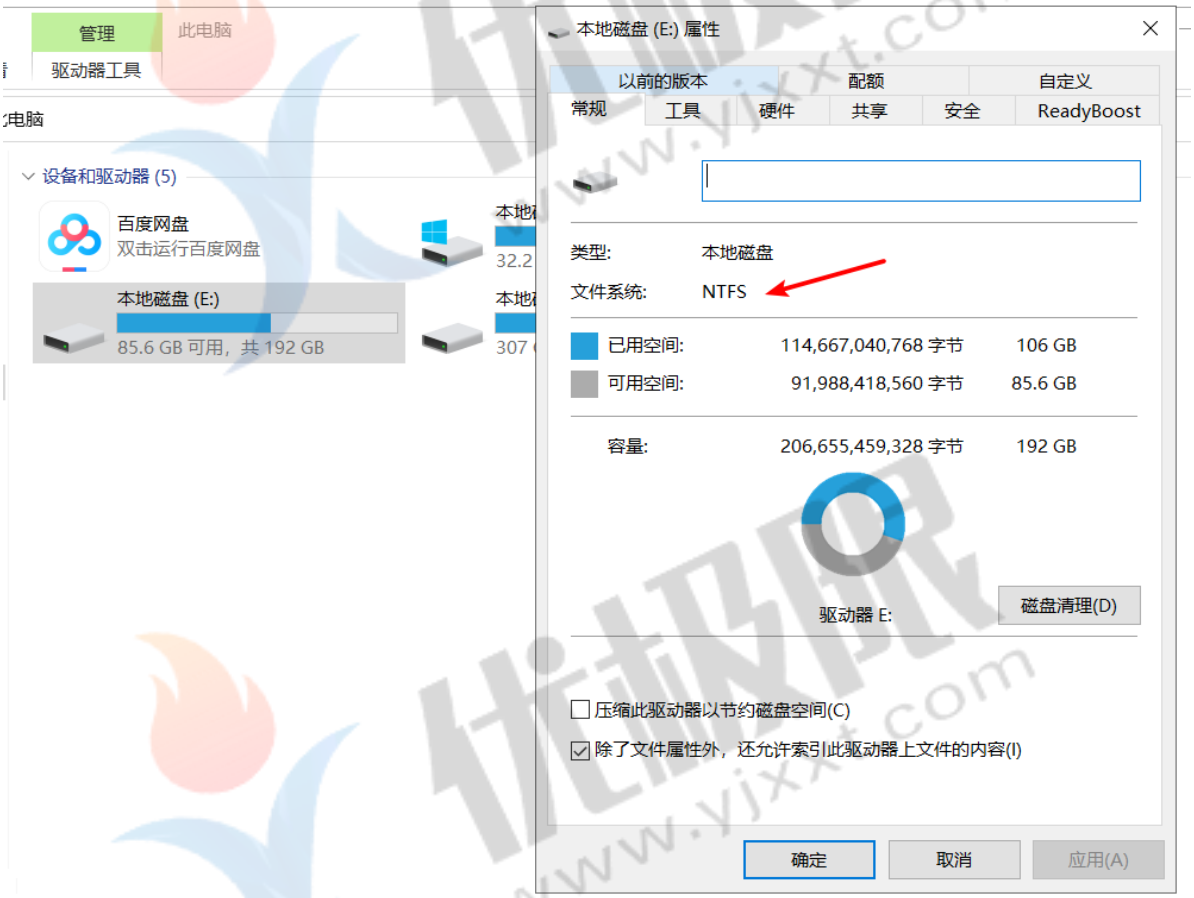
根据计算环境和所提供功能的不同，文件系统可划分为以下几种。

单机文件系统

特点：用于操作系统和应用程序的本地存储。

缺点：数据无法在多台机器之间共享。

代表：EXT2、EXT3、EXT4、NTFS、FAT、FAT32、XFS、JFS 等等。

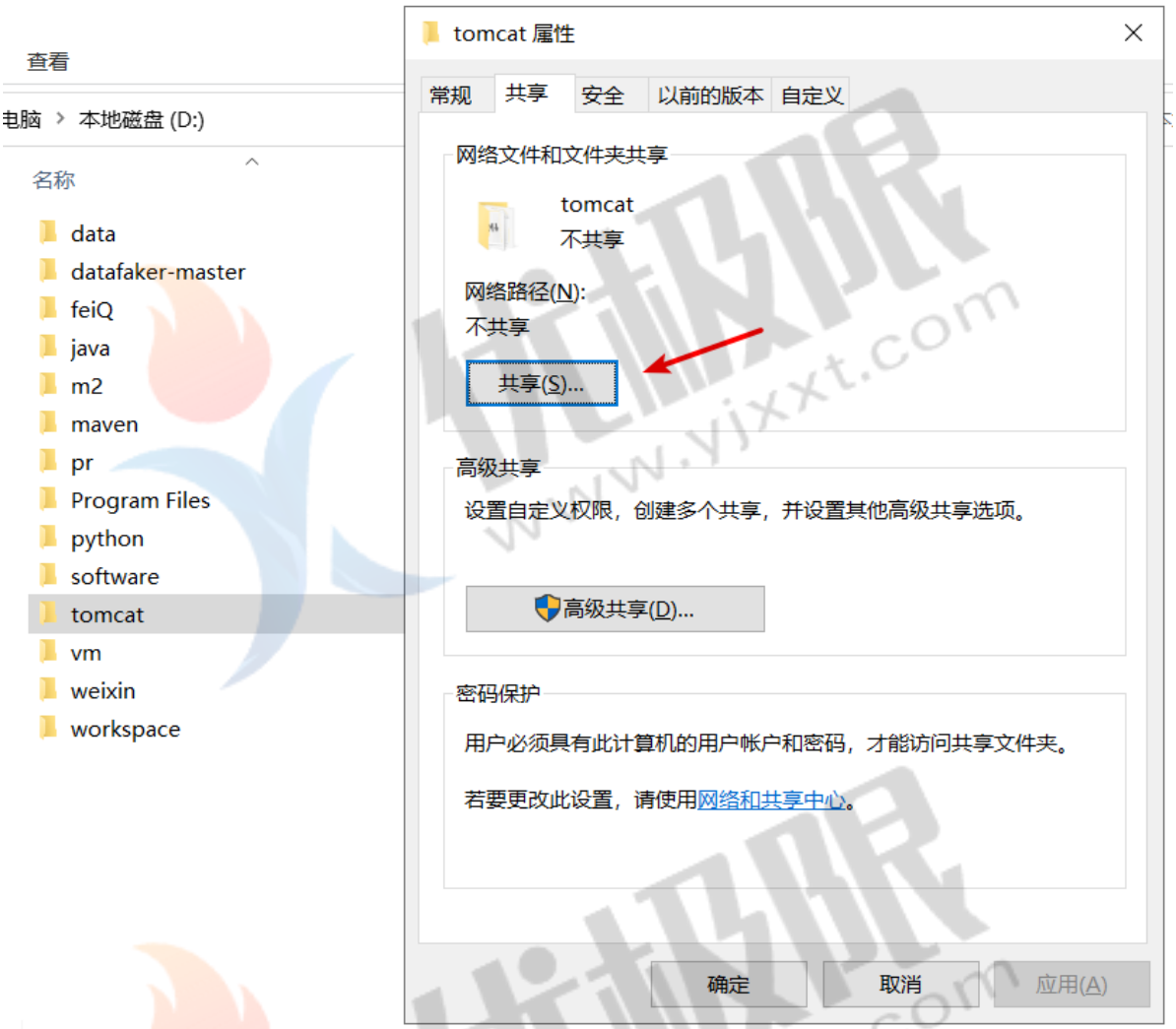


网络文件系统

特点：基于现有以太网架构，实现不同服务器之间传统文件系统的数据共享。

缺点：两台服务器不能同时访问修改，性能有限。

代表：NFS、CIFS 等等，比如下图 Windows 主机之间进行网络文件共享就是通过微软公司自己的 CIFS 服务实现的。



分布式文件系统

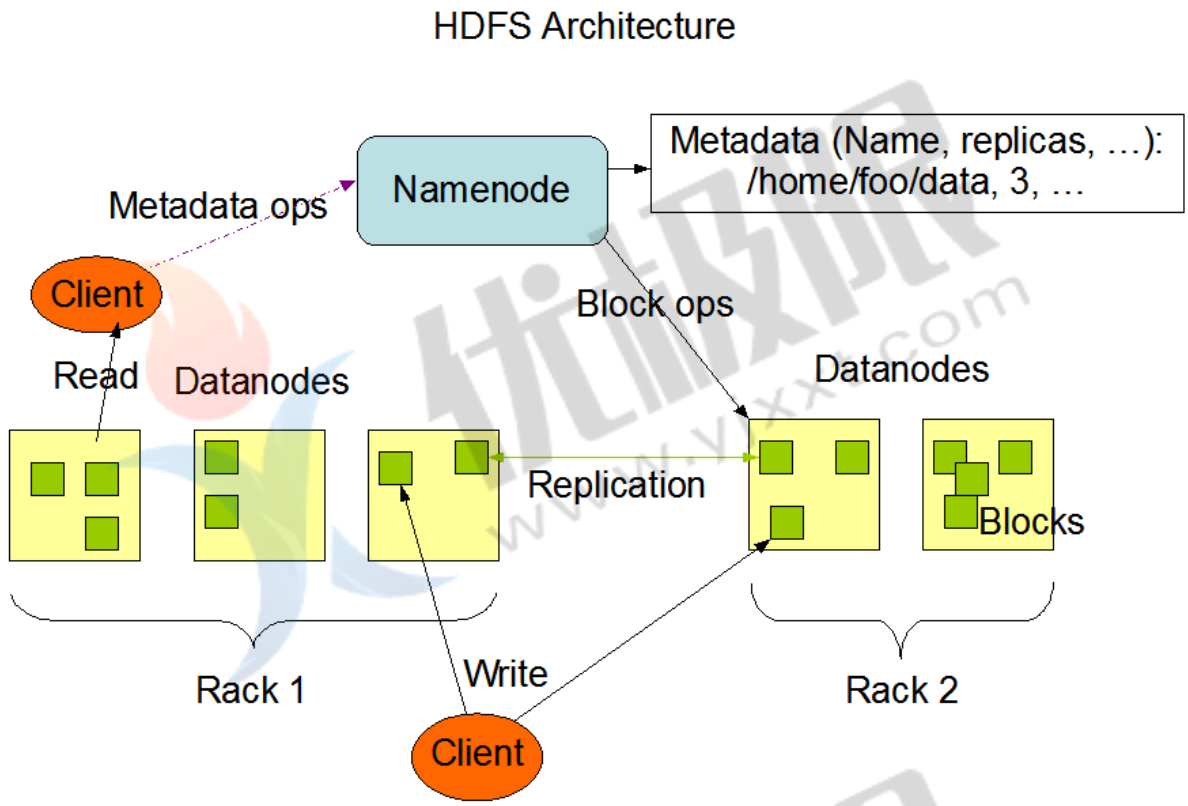
数据量越来越多，在一个操作系统管辖的范围存不下了，那么就分配到更多的操作系统管理的磁盘中，但是不方便管理和维护，因此迫切需要一种系统来管理多台机器上的文件，**这就是分布式文件管理系统。**

分布式文件系统（Distributed File System）是一种允许文件通过网络在多台主机上共享的文件系统，可以让多机器上的多用户进行文件分享和存储。在这样的文件系统中，客户端并非直接访问底层的数据存储区块，而是通过网络，以特定的通信协议和服务器沟通。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，让用户访问分布在网络上的共享文件更加简便。所有高层次的文件系统都是以低层次的传统文件系统为基础，实现了更高级的功能。

特点：在传统文件系统上，通过额外模块实现数据跨服务器分布，并且自身集成 RAID 保护功能，可以保证多台服务器同时访问、修改同一个文件系统。性能优越，扩展性强，可靠性高。

缺点：部分类型存在单点故障风险。

代表：HDFS（ASF）、MogileFS（LiveJournal）、FastDFS（余庆）、Lustre（Oracle）、GlusterFS（RedHat）等等。



通用型

通用分布式文件系统和传统的本地文件系统（如 EXT4、NTFS 等）相对应。典型代表：Lustre、MooseFS。

优点：传统文件系统的操作方式，对开发者门槛较低。

缺点：系统复杂性较高，需要支持若干标准的文件操作，如：目录结构、文件读写权限、文件锁等。系统整体性能有所降低，因为要支持 POSIX 标准（可移植操作系统接口 Portable Operating System Interface of UNIX）。

> POSIX 全称：可移植操作系统接口。当 Unix 诞生之后，各个厂商都实现了自己的 Unix 系统，导致接口不统一，基于不同的操作系统开发变得极其混乱，为了解决这一问题，便有了 POSIX 标准。
>
> 总结：POSIX 标准的诞生就是为了统一操作系统的接口，方便开发者开发应用程序，写出可移植的代码程序。基于 POSIX 标准的库函数都是可以在此标准的操作系统平台上移植。

专用型

专用分布式文件系统基于谷歌文件系统论文（Google File System）的设计思想而来，文件上传后不能修改。使用专有 API 对文件进行访问，也可称为分布式文件存储服务。典型代表：HDFS、MogileFS、FastDFS。

优点：系统复杂性较低，不需要支持若干标准的文件操作，如：目录结构、文件读写权限、文件锁等。系统整体性能较高，因为无需支持 POSIX 标准，系统更加高效。

缺点：采用专有 API 对文件进行访问，对开发者门槛较高，一般都是**直接封装成工具类**进行使用。

文件服务器的发展史

随着互联网图片、视频时代的到来，对文件的处理成为各个业务系统面临的巨大挑战，亟需搭建特有的文件服务器解决文件共享的问题。

本地文件服务器

特点：本地文件服务器是指**文件数据直接存储在本地节点**中。比如直接在项目目录下建立文件夹存放项目文件资源，如果按不同类型再细分，可以在项目目录下继续创建不同的子目录用于区分。

优点：简单便捷，项目可以直接引用，访问方便。

缺点：文件与代码混合存储不便于管理，随着文件的增多影响项目发布上线周期。

独立文件服务器

特点：搭建一台独立的服务器用于文件存储使用，项目上传文件时，先通过 ftp 或者 ssh 将文件上传至服务器某个目录下，再通过 Ngnix 或者 Apache Http Server 反向代理此目录，返回一个独立域名的文件 URL 地址，前端通过这个 URL 地址即可直接访问文件。

优点：独立存储，可以方便扩容、容灾和数据迁移。方便做图片访问请求的负载均衡，方便应用各种缓存策略（HTTP Header、Proxy Cache 等），也更加方便迁移到 CDN。而且图片访问是很消耗服务器资源的（因为会涉及到操作系统的上下文切换和磁盘 I/O 操作），分离出来以后，Web/App 服务器可以更专注发挥动态处理的能力。

缺点：单机存在性能瓶颈，容灾、垂直扩展性差。

分布式文件服务器

特点：分布式文件系统一般包括**访问的仲裁**，**文件的存储**，**文件的容灾**三大块。仲裁模块相当于文件服务器的大脑，根据一定的算法来决定文件存储的位置。文件存储模块负责保存文件。容灾模块负责文件数据的相互备份。

优点：弹性伸缩，性能优越，扩展性强，可靠性高。

缺点：系统复杂度稍高，需要更多服务器。

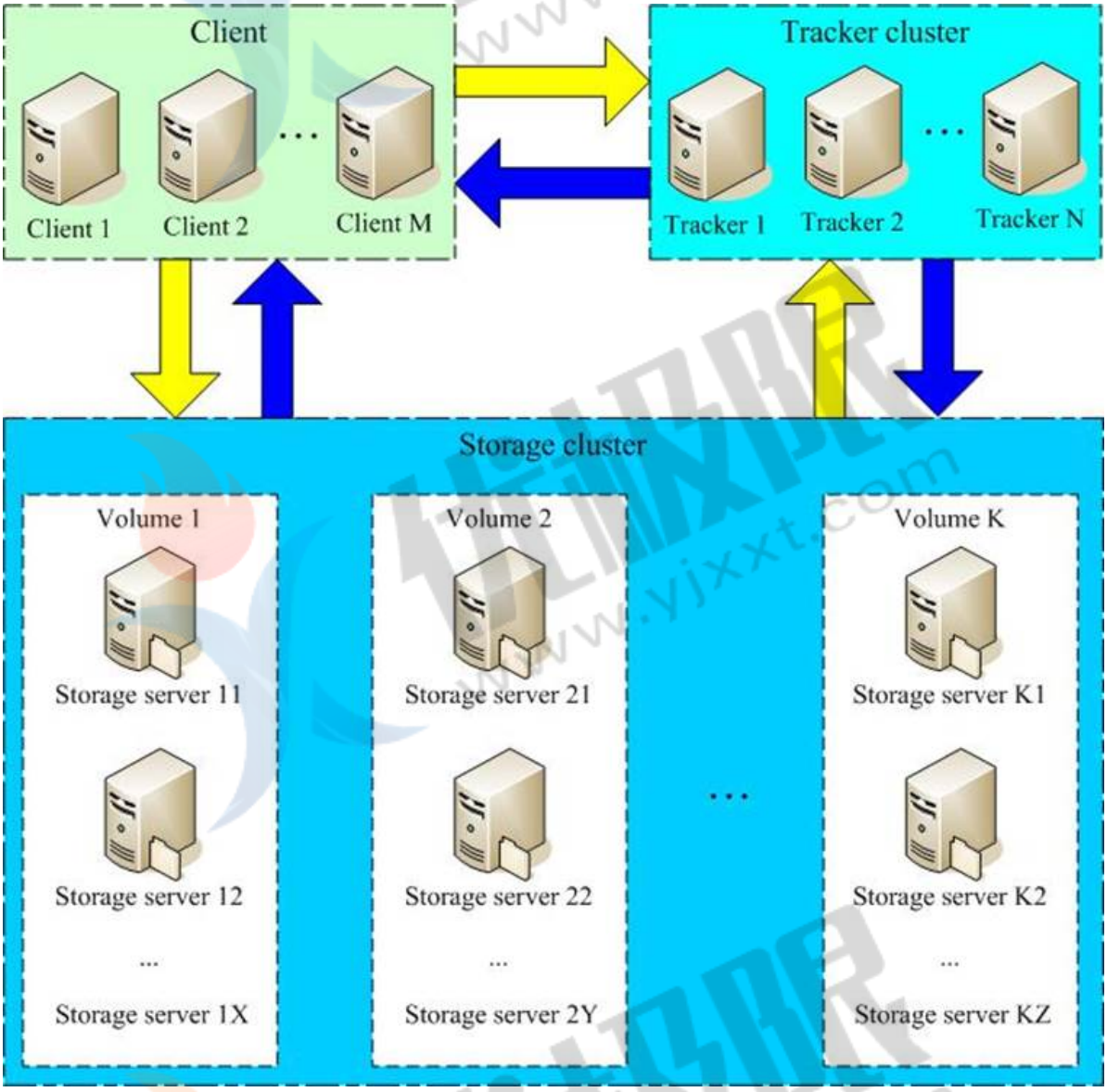
FastDFS 简介

FastDFS 就是我们上述所说的**专用型分布式文件系统**，接下来我们就详细了解它的核心概念，架构体系及环境的搭建与使用。

FastDFS 是基于 C 语言开发的，是一个轻量级开源的高性能分布式文件系统。主要功能有：文件存储、文件同步、文件访问(文件上传/下载)，解决了大容量的文件存储和高并发访问的问题，文件存取时实现了负载均衡。FastDFS 特别适合中大型网站以文件为载体的在线服务，适合存储 `4KB ~ 500MB` 之间的小文件，如照片共享网站、视频共享网站（图片、文档、音频、视频等等）。

FastDFS 是一款国产开源软件，作者余庆，项目开源地址 Github：https://github.com/happyfish100/fastdfs 官方论坛：http://bbs.chinaunix.net/forum-240-1.html

FastDFS 架构



Client

客户端，实现文件上传下载的服务器，就是我们自己的项目所部署在的服务器。通过专有接口，使用 TCP/IP 协议与跟踪服务器或存储服务器进行数据交互。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。

Tracker Server

跟踪服务器，负责文件访问的调度和负载均衡，负责管理所有的 Storage Server 和 group 组/卷。

Storage Server

存储服务器，负责文件存储，文件同步/备份，提供文件访问接口，文件元数据管理。以 group 为单位，每个 group 内可以有多台 Storage Server，数据互为备份，达到容灾的目的。每个 Storage 在启动以后会主动连接 Tracker，告知自己所属 group 等存储相关信息，并保持周期性心跳。

Group

组， 也可称为 Volume 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。

Metadata

文件系统中存储的数据分为**数据**和**元数据**两部分，数据是指文件中的实际数据，即文件的实际内容；而元数据是用来描述一个文件特征的系统数据，诸如访问权限、文件拥有者以及文件数据块的分布信息等等。如果文件是一张图片，元数据就是图片的宽，高等等。

FastDFS 存储策略

为了支持大容量存储，Storage 存储服务器采用了分组（或分卷）的方式。存储系统由一个或多个组组成，组与组之间的文件是相互独立的，所有组的文件容量累加就是整个存储系统中的文件容量。一个组可以由一台或多台存储服务器组成，一个组下的存储服务器中的文件都是相同的，组中的多台存储服务器起到了冗余备份和负载均衡的作用。

当组中增加了新的服务器时，系统会自动同步已有的文件，同步完成后，系统自动将新增的服务器切换至线上提供服务。

当存储空间不足时，可以动态添加组，只需要增加一台或多台服务器，并将它们配置为一个新的组，即可扩大存储系统的容量。当你的某个应用或者模块（对应的 group）的并发过高的时候，可以直接在 group 中增加若干个 Storage 来实现负载均衡。

为了避免单个目录下的文件数太多，当 Storage 第一次启动时，会在每个数据存储目录中创建 2 级子目录，每级 256 个，总共 65536 个目录，上传的文件会以 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。

FastDFS 安装

机器分布

首先要说明一下：`tracker` 和 `storage` 其实都是 `fastdfs`，只不过启动时通过不同的配置文件启动，所扮演的角色不同而已。也就是说，安装 `tracker` 和 `storage` 就是在安装 `fastdfs`，然后通过每个角色具体的配置文件启动即可。

IP	角色
192.168.10.101	Tracker
192.168.10.102	Storage

下载资源

直接通过 Github：https://github.com/happyfish100 下载 `libfastcommon`，`fastdfs`，`fastdfs-nginx-module` 三个项目对应的压缩包或者使用 git 命令下载，或者通过资源地址：https://sourceforge.net/projects/fastdfs/files/ 下载。

- `libfastcommon`：从 `fastdfs` 项目和 `fastdht` 项目中提取出来的公共 C 函数库。
- `fastdfs`：FastDFS 核心项目。
- `fastdfs-nginx-module`：Nginx 整合 FastDFS 时 Nginx 需要添加的模块资源。

安装依赖

FastDFS 是基于 C 语言开发的，安装它之前必须先安装它所依赖的环境。

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

安装公共函数库

上传资源 `libfastcommon-master.zip` 至服务器 `/usr/local/src` 目录后并解压。

```
# 安装 unzip 用于解压
yum install -y unzip
# 解压 libfastcommon 至当前所在目录
unzip libfastcommon-master.zip
```

编译并安装。

```
# 进入解压后的 libfastcommon-master 目录
cd libfastcommon-master
# 编译并安装
./make.sh && ./make.sh install
```

`libfastcommon` 默认安装在 `/usr/lib64` 和 `/usr/include/fastcommon` 两个目录中，并且会在 `/usr/lib` 目录中创建软链接。

```
[root@localhost libfastcommon-master]# ./make.sh install
mkdir -p /usr/lib64
mkdir -p /usr/lib
mkdir -p /usr/include/fastcommon
install -m 755 libfastcommon.so /usr/lib64
install -m 644 common_define.h hash.h chain.h logger.h base64.h shared_func.h pthread_func.h ini_file_reader.h _os_define.h sockopt.h sched_thread.h
http_func.h md5.h local_ip_func.h avl_tree.h ioevent.h ioevent_loop.h fast_task_queue.h fast_timer.h process_ctrl.h fast_mblock.h connection_pool.h f
ast_mpool.h fast_allocator.h fast_buffer.h skiplist.h multi_skiplist.h flat_skiplist.h skiplist_common.h system_info.h fast_blocked_queue.h php7_ext_
wrapper.h id_generator.h char_converter.h char_convert_loader.h common_blocked_queue.h multi_socket_client.h skiplist_set.h uniq_skiplist.h fc_list.h
 json_parser.h buffered_file_writer.h server_id_func.h fc_queue.h fc_memory.h shared_buffer.h thread_pool.h fc_atomic.h /usr/include/fastcommon
if [ ! -e /usr/lib/libfastcommon.so ]; then ln -s /usr/lib64/libfastcommon.so /usr/lib/libfastcommon.so; fi
```

安装 FastDFS

上传资源 `fastdfs-master.zip` 至服务器 `/usr/local/src` 目录后并解压。

```
# 解压 fastdfs 至当前所在目录
unzip fastdfs-master.zip
```

编译并安装。

```
# 进入解压后的 libfastcommon-master 目录
cd fastdfs-master
# 编译并安装
./make.sh && ./make.sh install
```

`fastdfs` 默认安装在以下位置：

- `/usr/bin`：可执行文件
- `/etc/fdfs`：配置文件
- `/etc/init.d`：主程序代码
- `/usr/include/fastdfs`：插件组

```
[root@localhost fastdfs-master]# ./make.sh install
mkdir -p /usr/bin
mkdir -p /etc/fdfs
cp -f fdfs_trackerd /usr/bin
if [ ! -f /etc/fdfs/tracker.conf.sample ]; then cp -f ../conf/tracker.conf /etc/fdfs/tracker.conf.sample; fi
if [ ! -f /etc/fdfs/storage_ids.conf.sample ]; then cp -f ../conf/storage_ids.conf /etc/fdfs/storage_ids.conf.sample; fi
mkdir -p /usr/bin
mkdir -p /etc/fdfs
cp -f fdfs_storaged  /usr/bin
if [ ! -f /etc/fdfs/storage.conf.sample ]; then cp -f ../conf/storage.conf /etc/fdfs/storage.conf.sample; fi
mkdir -p /usr/bin
mkdir -p /etc/fdfs
mkdir -p /usr/lib64
mkdir -p /usr/lib
cp -f fdfs_monitor fdfs_test fdfs_test1 fdfs_crc32 fdfs_upload_file fdfs_download_file fdfs_delete_file fdfs_file_info fdfs_appender_test fdfs_append
er_test1 fdfs_append_file fdfs_upload_appender fdfs_regenerate_filename /usr/bin
if [ 0 -eq 1 ]; then cp -f libfdfsclient.a /usr/lib64; cp -f libfdfsclient.a /usr/lib/;fi
if [ 1 -eq 1 ]; then cp -f libfdfsclient.so /usr/lib64; cp -f libfdfsclient.so /usr/lib/;fi
mkdir -p /usr/include/fastdfs
cp -f ../common/fdfs_define.h ../common/fdfs_global.h ../common/mime_file_parser.h ../common/fdfs_http_shared.h ../tracker/tracker_types.h ../tracker
/tracker_proto.h ../tracker/fdfs_shared_func.h ../tracker/fdfs_server_id_func.h ../storage/trunk_mgr/trunk_shared.h tracker_client.h storage_client.h
 storage_client1.h client_func.h client_global.h fdfs_client.h /usr/include/fastdfs
if [ ! -f /etc/fdfs/client.conf.sample ]; then cp -f ../conf/client.conf /etc/fdfs/client.conf.sample; fi
```

启动 Tracker

`tracker` 和 `storage` 其实都是 `fastdfs`，只不过启动时通过不同的配置文件启动，所扮演的角色不同而已。也就是说，安装 `tracker` 和 `storage` 就是在安装 `fastdfs`，然后通过每个角色具体的配置文件启动即可。

查看 `/etc/fdfs` 目录下所有配置文件。

```
[root@localhost ~]# ls /etc/fdfs/
client.conf.sample  storage.conf.sample  storage_ids.conf.sample
tracker.conf.sample
```

- `client.conf.sample`：客户端的配置文件，测试用
- `storage.conf.sample`：存储器的配置文件
- `tracker.conf.sample`：跟踪器的配置文件

编辑 `tracker.conf` 配置文件。

```
# 拷贝文件 tracker.conf.sample 并重命名为 tracker.conf
cp /etc/fdfs/tracker.conf.sample /etc/fdfs/tracker.conf
# 编辑 tracker.conf 配置文件
vi /etc/fdfs/tracker.conf
```

配置文件中的配置项还是蛮多的，这里暂且关注以下几个即可，后期根据实际情况再对其他配置项作出调整。

```
# 允许访问 tracker 服务器的 IP 地址，为空则表示不受限制
bind_addr =

# tracker 服务监听端口
port = 22122

# tracker 服务器的运行数据和日志的存储父路径（需要提前创建好）
base_path = /fastdfs/tracker

# tracker 服务器 HTTP 协议下暴露的端口
http.server_port = 8080
```

启动 `tracker` 服务。

```
# 创建 tracker 服务器的运行数据和日志的存储父路径
mkdir -p /fastdfs/tracker
# 启动 tracker 服务
service fdfs_trackerd start
# 查看 tracker 服务状态
service fdfs_trackerd status
# 重启 tracker 服务
service fdfs_trackerd restart
# 停止 tracker 服务
service fdfs_trackerd stop
```

```
[root@localhost fdfs]# service fdfs_trackerd start
Reloading systemd:                                         [  确定  ]
Starting fdfs_trackerd (via systemctl):                    [  确定  ]
[root@localhost fdfs]# service fdfs_trackerd status
● fdfs_trackerd.service - LSB: FastDFS tracker server
   Loaded: loaded (/etc/rc.d/init.d/fdfs_trackerd; bad; vendor preset: disabled)
   Active: active (exited) since 四 2020-09-24 18:09:13 CST; 12s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 2018 ExecStart=/etc/rc.d/init.d/fdfs_trackerd start (code=exited, status=0/SUCCESS)

9月 24 18:09:13 localhost.localdomain systemd[1]: Starting LSB: FastDFS tracker server...
9月 24 18:09:13 localhost.localdomain fdfs_trackerd[2018]: Starting FastDFS tracker server:
9月 24 18:09:13 localhost.localdomain systemd[1]: Started LSB: FastDFS tracker server.
9月 24 18:09:13 localhost.localdomain fdfs_trackerd[2018]: [2020-09-24 18:09:13] ERROR - file: process_ctrl.c, line: 288, "/fastdfs/tracker...ectory
Hint: Some lines were ellipsized, use -l to show in full.
```

启动 Storage

编辑 `storage.conf` 配置文件。

```
# 拷贝文件 storage.conf.sample 并重命名为 storage.conf
cp /etc/fdfs/storage.conf.sample /etc/fdfs/storage.conf
# 编辑 storage.conf 配置文件
vi /etc/fdfs/storage.conf
```

配置文件中的配置项还是蛮多的，这里暂且关注以下几个即可，后期根据实际情况再对其他配置项作出调整。

```
# storage 组名/卷名，默认为 group1
group_name = group1

# 允许访问 storage 服务器的 IP 地址，为空则表示不受限制
bind_addr =

# storage 服务器的运行数据和日志的存储父路径（需要提前创建好）
base_path = /fastdfs/storage/base

# storage 服务器中客户端上传的文件的存储父路径（需要提前创建好）
store_path0 = /fastdfs/storage/store

# storage 服务器 HTTP 协议下暴露的端口
http.server_port = 8888

# tracker 服务器的 IP 和端口
tracker_server = 192.168.10.101:22122
```

启动 `storage` 服务。

```
# 创建 storage 服务器的运行数据和日志的存储父路径
mkdir -p /fastdfs/storage/base
# 创建 storage 服务器中客户端上传的文件的存储父路径
mkdir -p /fastdfs/storage/store
# 启动 storage 服务
service fdfs_storaged start
# 查看 storage 服务状态
service fdfs_storaged status
# 重启 storage 服务
service fdfs_storaged restart
# 停止 storage 服务
service fdfs_storaged stop
```

```
[root@localhost fdfs]# service fdfs_storaged start
Reloading systemd:                                         [  确定  ]
Starting fdfs_storaged (via systemctl):                    [  确定  ]
[root@localhost fdfs]# service fdfs_storaged status
● fdfs_storaged.service - LSB: FastDFS storage server
   Loaded: loaded (/etc/rc.d/init.d/fdfs_storaged; bad; vendor preset: disabled)
   Active: active (running) since 四 2020-09-24 18:53:14 CST; 7s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 2026 ExecStart=/etc/rc.d/init.d/fdfs_storaged start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/fdfs_storaged.service
           └─2031 /usr/bin/fdfs_storaged /etc/fdfs/storage.conf

9月 24 18:53:14 localhost.localdomain systemd[1]: Starting LSB: FastDFS storage server...
9月 24 18:53:14 localhost.localdomain fdfs_storaged[2026]: Starting FastDFS storage server:
9月 24 18:53:14 localhost.localdomain systemd[1]: Started LSB: FastDFS storage server.
```

查看 `/fastdfs/storage/store` 目录可以看到 Storage 服务器创建了 `65536` 个文件夹用于存储客户端上传的文件。

```
[root@localhost store]# pwd
/fastdfs/storage/store
[root@localhost store]# ls
data
[root@localhost store]# ls data/
00  07  0E  15  1C  23  2A  31  38  3F  46  4D  54  5B  62  69  70  77  7E  85  8C  93  9A  A1  A8  AF  B6  BD  C4  CB  D2  D9  E0  E7  EE  F5  FC
01  08  0F  16  1D  24  2B  32  39  40  47  4E  55  5C  63  6A  71  78  7F  86  8D  94  9B  A2  A9  B0  B7  BE  C5  CC  D3  DA  E1  E8  EF  F6  FD
02  09  10  17  1E  25  2C  33  3A  41  48  4F  56  5D  64  6B  72  79  80  87  8E  95  9C  A3  AA  B1  B8  BF  C6  CD  D4  DB  E2  E9  F0  F7  FE
03  0A  11  18  1F  26  2D  34  3B  42  49  50  57  5E  65  6C  73  7A  81  88  8F  96  9D  A4  AB  B2  B9  C0  C7  CE  D5  DC  E3  EA  F1  F8  FF
04  0B  12  19  20  27  2E  35  3C  43  4A  51  58  5F  66  6D  74  7B  82  89  90  97  9E  A5  AC  B3  BA  C1  C8  CF  D6  DD  E4  EB  F2  F9
05  0C  13  1A  21  28  2F  36  3D  44  4B  52  59  60  67  6E  75  7C  83  8A  91  98  9F  A6  AD  B4  BB  C2  C9  D0  D7  DE  E5  EC  F3  FA
06  0D  14  1B  22  29  30  37  3E  45  4C  53  5A  61  68  6F  76  7D  84  8B  92  99  A0  A7  AE  B5  BC  C3  CA  D1  D8  DF  E6  ED  F4  FB
[root@localhost store]# ls data/00
00  07  0E  15  1C  23  2A  31  38  3F  46  4D  54  5B  62  69  70  77  7E  85  8C  93  9A  A1  A8  AF  B6  BD  C4  CB  D2  D9  E0  E7  EE  F5  FC
01  08  0F  16  1D  24  2B  32  39  40  47  4E  55  5C  63  6A  71  78  7F  86  8D  94  9B  A2  A9  B0  B7  BE  C5  CC  D3  DA  E1  E8  EF  F6  FD
02  09  10  17  1E  25  2C  33  3A  41  48  4F  56  5D  64  6B  72  79  80  87  8E  95  9C  A3  AA  B1  B8  BF  C6  CD  D4  DB  E2  E9  F0  F7  FE
03  0A  11  18  1F  26  2D  34  3B  42  49  50  57  5E  65  6C  73  7A  81  88  8F  96  9D  A4  AB  B2  B9  C0  C7  CE  D5  DC  E3  EA  F1  F8  FF
04  0B  12  19  20  27  2E  35  3C  43  4A  51  58  5F  66  6D  74  7B  82  89  90  97  9E  A5  AC  B3  BA  C1  C8  CF  D6  DD  E4  EB  F2  F9
05  0C  13  1A  21  28  2F  36  3D  44  4B  52  59  60  67  6E  75  7C  83  8A  91  98  9F  A6  AD  B4  BB  C2  C9  D0  D7  DE  E5  EC  F3  FA
06  0D  14  1B  22  29  30  37  3E  45  4C  53  5A  61  68  6F  76  7D  84  8B  92  99  A0  A7  AE  B5  BC  C3  CA  D1  D8  DF  E6  ED  F4  FB
```

Client 操作

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。

在 Tracker 服务器的机器上编辑 `tracker.conf` 配置文件。

```
# 拷贝文件 client.conf.sample 并重命名为 client.conf
cp /etc/fdfs/client.conf.sample /etc/fdfs/client.conf
# 编辑 client.conf 配置文件
vi /etc/fdfs/client.conf
```

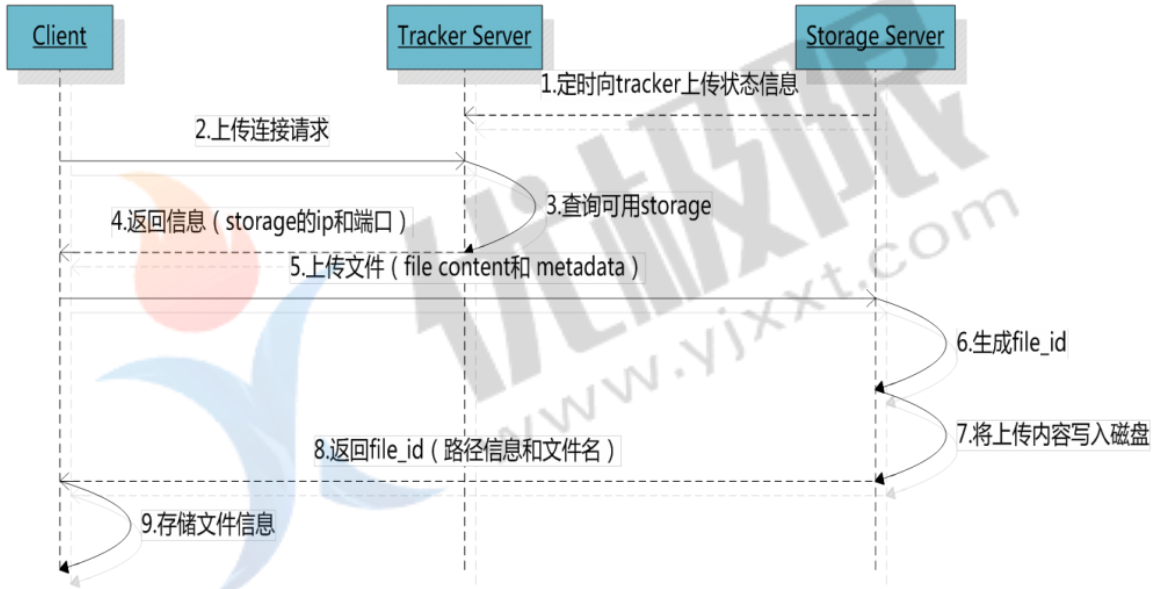
修改配置文件中以下两处内容即可。

```
# client 客户端的运行数据和日志的存储父路径（需要提前创建好）
base_path = /fastdfs/client

# tracker 服务器的 IP 和端口
tracker_server = 192.168.10.101:22122
```

记得 `mkdir -p /fastdfs/client` 创建 Client 目录。

上传



选择 Tracker Server

如上图所示，Storage Server 会定期向 Tracker Server 发送自己的所属 group 等存储相关信息。如果 Tracker Server 是集群环境，因为各个 Tracker 之间的关系是对等的，所以客户端上传时可以选择任意一个 Tracker。

选择 group

当 Tracker 收到客户端上传文件的请求时，会为该文件分配一个可用的 group 用于存储，当选定了 group 以后就要决定给客户端分配 group 中的哪个 Storage Server。

```
# the method for selecting group to upload files
# 0: round robin
# 1: specify group
# 2: load balance, select the max free space group to upload file
store_lookup = 2

# which group to upload file
# when store_lookup set to 1, must set store_group to the group name
store_group = group2
```

如上图所示，`tracker.conf` 配置文件中 group 的可选规则有：

- `round robin`：所有的 group 间轮询
- `specify group`：指定一个具体的 group
- `load balance`：优先选择剩余存储空间多的 group

选择 Storage Server

当分配好 Storage Server 以后，客户端会向 Storage Server 发送上传文件请求，Storage Server 会为文件分配一个具体的数据存储目录。

```
# the mode of the files distributed to the data path
# 0: round robin(default)
# 1: random, distributted by hash code
file_distribute_path_mode = 0

# valid when file_distribute_to_path is set to 0 (round robin).
# when the written file count reaches this number, then rotate to next path.
# rotate to the first path (00/00) after the last path (such as FF/FF).
# default value is 100
file_distribute_rotate_count = 100
```

如上图所示，`storage.conf` 配置文件中文件分发的可选规则有：

- `round robin`：在 group 中的所有 Storage 间轮询
- `random`：随机，按 hash code 分发

生成 file_id

选定存储目录以后，Storage 会为文件生一个 file_id，由 Storage Server IP、文件创建时间、文件大小、文件 crc32 和一个随机数组成，然后将这个二进制串进行 base64 编码，转换为字符串。

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名称/存储目录/两级子目录/file_id.后缀名` 拼接而成。

FastDFS 文件上传返回信息解读

- `group1`：**组名/卷名。**文件上传成功以后所在的 Storage 组名称，由 Storage 服务器返回。
- `M00`：**虚拟磁盘路径。**与 Storage 配置文件中磁盘选项 `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。比如：`store_path0 = /fastdfs/storage/store`，`M00` 则表示：`/fastdfs/storage/store/data`。
- `/02/44`：**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。
- `wKgDrE34E8wAAAAAAAAGkEIYJK42378`：file_id，由 Storage Server IP、文件创建时间、文件大小、文件 crc32 和一个随机数组成，然后将这个二进制串进行 base64 编码，转换为字符串。
- `group1/M00/02/44/wKgDrE34E8wAAAAAAAAGkEIYJK42378.sh`：文件名。

方式一

上传命令格式为：`fdfs_upload_file /etc/fdfs/client.conf 要上传的文件`。

```
[root@localhost ~]# fdfs_upload_file /etc/fdfs/client.conf
/usr/local/src/china.jpg
group1/M00/00/00/wKgKZl9skn6AHZKUAADhaCZ_RF0650.jpg
```

```
[root@localhost ~]# fdfs_upload_file /etc/fdfs/client.conf /usr/local/src/china.jpg
group1/M00/00/00/wKgKZl9skn6AHZKUAADhaCZ_RF0650.jpg
```

文件上传成功以后，会返回该文件在 Storage 服务器中的存储位置及随机生成的文件名。其中 `group1` 表示 Storage 组名/卷名，`M00` 是一个虚拟目录，表示 `/fastdfs/storage/store/data/` 真实路径中的 `data` 目录。

如下图所示，查看 Storage 服务器发现该文件已成功上传。

```
[root@localhost ~]# ls /fastdfs/storage/store/data/00/00/
wKgKZl9skn6AHZKUAADhaCZ_RF0650.jpg
```

方式二

或者使用：`fdfs_test /etc/fdfs/client.conf upload 要上传的文件`。

```
[root@localhost ~]# fdfs_test /etc/fdfs/client.conf upload
/usr/local/src/china.jpg
This is FastDFS client test program v6.07

Copyright (C) 2008, Happy Fish / YuQing

FastDFS may be copied only under the terms of the GNU General
```

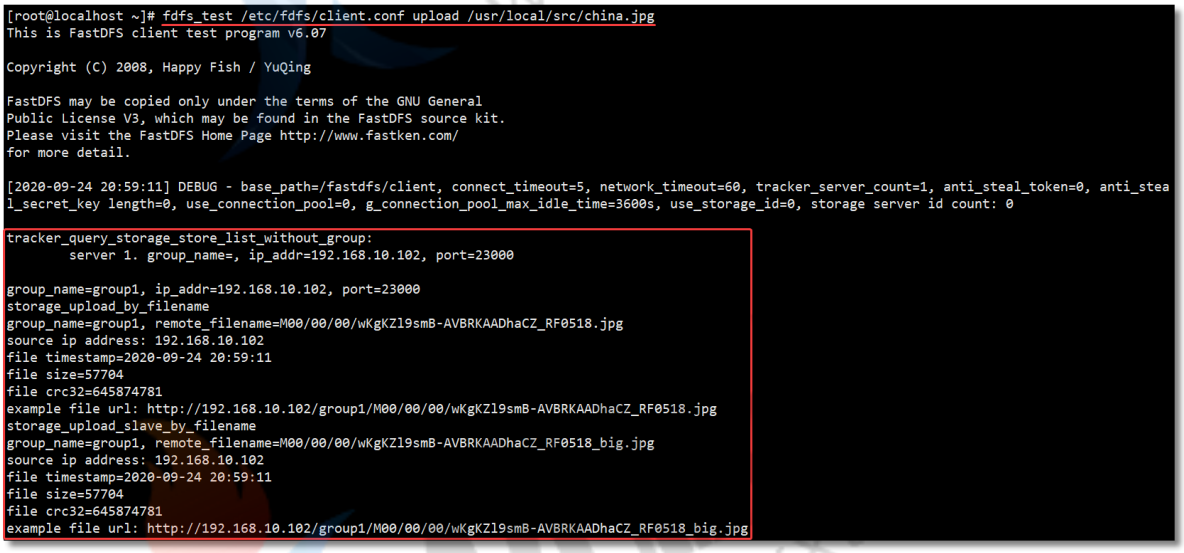
```
Public License V3, which may be found in the FastDFS source kit.
Please visit the FastDFS Home Page http://www.fastken.com/
for more detail.

[2020-09-24 20:59:11] DEBUG - base_path=/fastdfs/client, connect_timeout=5,
network_timeout=60, tracker_server_count=1, anti_steal_token=0,
anti_steal_secret_key length=0, use_connection_pool=0,
g_connection_pool_max_idle_time=3600s, use_storage_id=0, storage server id
count: 0

tracker_query_storage_store_list_without_group:
    server 1. group_name=, ip_addr=192.168.10.102, port=23000

group_name=group1, ip_addr=192.168.10.102, port=23000
storage_upload_by_filename
group_name=group1, remote_filename=M00/00/00/wKgKZl9smB-AVBRKAADhaCZ_RF0518.jpg
source ip address: 192.168.10.102
file timestamp=2020-09-24 20:59:11
file size=57704
file crc32=645874781
example file url: http://192.168.10.102/group1/M00/00/00/wKgKZl9smB-
AVBRKAADhaCZ_RF0518.jpg
storage_upload_slave_by_filename
group_name=group1, remote_filename=M00/00/00/wKgKZl9smB-
AVBRKAADhaCZ_RF0518_big.jpg
source ip address: 192.168.10.102
file timestamp=2020-09-24 20:59:11
file size=57704
file crc32=645874781
example file url: http://192.168.10.102/group1/M00/00/00/wKgKZl9smB-
AVBRKAADhaCZ_RF0518_big.jpg
```

通过 `fdfs_test` 的方式上传文件，会返回该文件上传成功以后详细的相关信息。



- `group_name`：Storage 组名/卷名
- `remote_filename`：上传成功文件的存储路径及文件名
- `source_ip address`：上传成功文件所在的 Storage 服务器的 IP 地址
- `file timestamp`：上传成功文件时的时间戳
- `file size`：上传成功文件的文件大小
- `example file url`：上传成功文件的 url 地址，**配合 Nginx 可以直接访问**
- `storage_upload_slave_by_filename`：FastDFS 的文件主/从特性，由主文件产生从文件

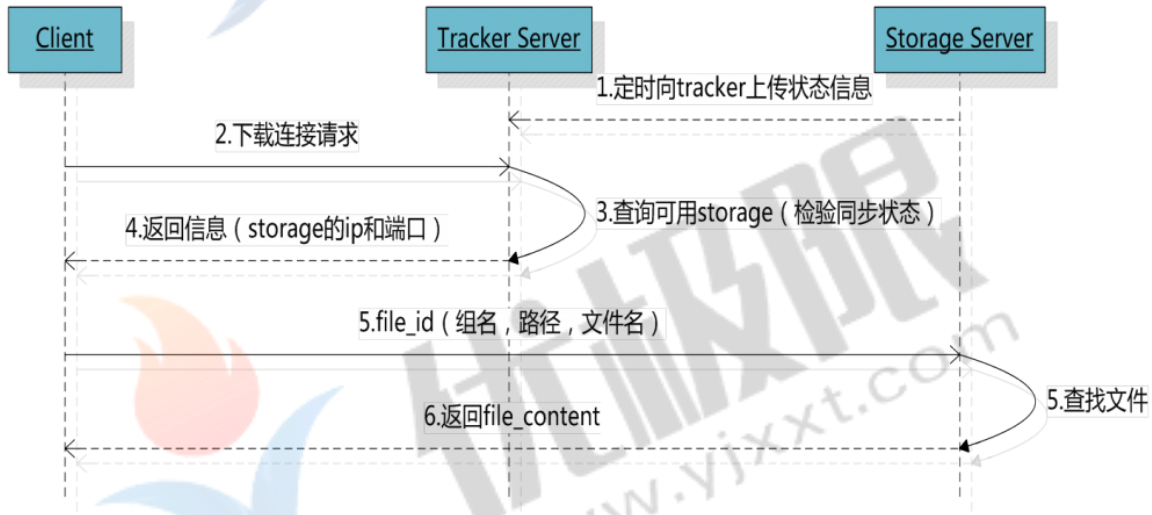
如下图所示，查看 Storage 服务器发现该文件已成功上传。后缀为 `jpg-m` 的文件存放了上传成功文件的元数据信息。

```
[root@localhost ~]# ls /fastdfs/storage/store/data/00/00/
wKgKZl9skn6AHZKUAADhaCZ_RF0650.jpg      wKgKZl9smB-AVBRKAADhaCZ_RF0518_big.jpg-m  wKgKZl9smB-AVBRKAADhaCZ_RF0518.jpg-m
wKgKZl9smB-AVBRKAADhaCZ_RF0518_big.jpg  wKgKZl9smB-AVBRKAADhaCZ_RF0518.jpg
```

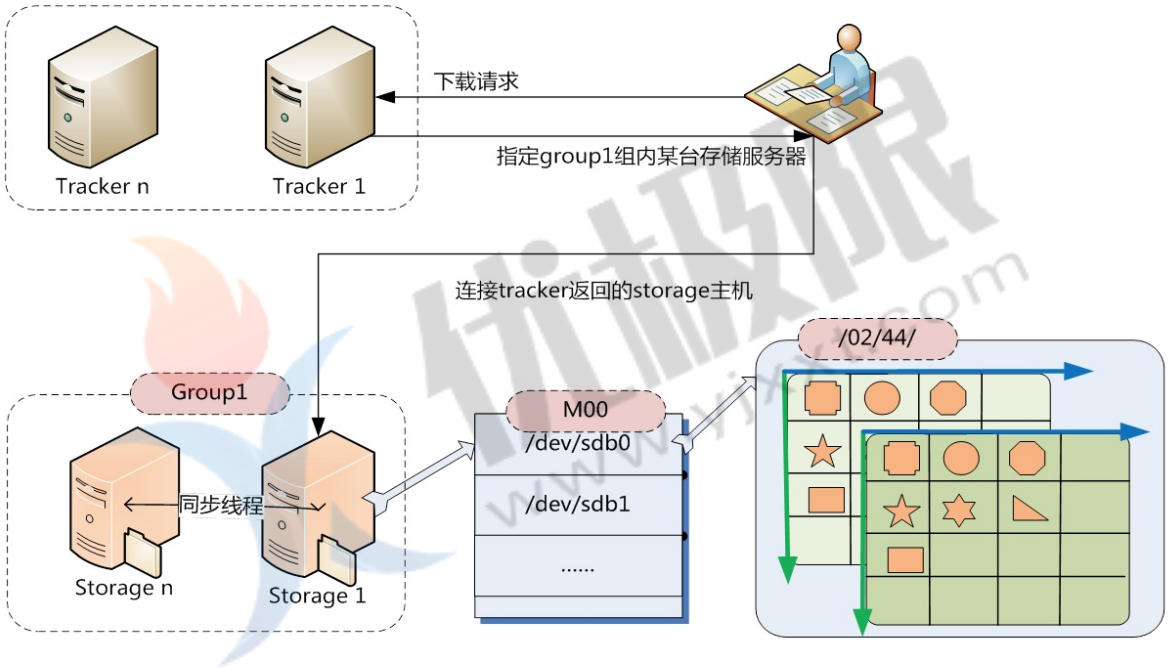
查看元数据信息如下：

```
[root@localhost ~]# more /fastdfs/storage/store/data/00/00/wKgKZl9smB-
AVBRKAADhaCZ_RF0518.jpg-m
ext_namejpgfile_size115120height80width160
```

下载



客户端 upload file 成功以后，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。客户端发送 download 请求给某个 Tracker，必须带上文件名信息，Tracke 从文件名中解析出该文件的 group、大小、创建时间等信息，然后为该请求选择一个 Storage 用于提供读取服务。



方式一

下载命令格式为：`fdfs_download_file /etc/fdfs/client.conf group_name/remote_filename`。

```
fdfs_download_file /etc/fdfs/client.conf group1/M00/00/00/wKgKZl9smB-AVBRKAADhaCZ_RF0518.jpg
```

方式二

或者使用：`fdfs_test /etc/fdfs/client.conf download group_name remote_filename`。

```
fdfs_test /etc/fdfs/client.conf download group1 M00/00/00/wKgKZl9smB-AVBRKAADhaCZ_RF0518.jpg
```

删除

方式一

删除命令格式为：`fdfs_delete_file /etc/fdfs/client.conf 要删除的文件`。

```
fdfs_delete_file /etc/fdfs/client.conf group1/M00/00/00/wKgKZl9smB-AVBRKAADhaCZ_RF0518.jpg
```

> Tips：删除文件操作会将元数据文件一并删除。

方式二

或者使用：`fdfs_test /etc/fdfs/client.conf delete group_name remote_filename`。

```
fdfs_test /etc/fdfs/client.conf delete group1 M00/00/00/wKgKZl9smB-AVBRKAADhaCZ_RF0518_big.jpg
```

至此 FastDFS 的核心概念，架构体系及环境的搭建与使用就到这里。说到文件服务器的使用，我们最终的目的是通过 HTTP 实现对文件的访问，但是此时还无法通过 HTTP 对文件进行访问，这就需要借助其他工具来实现了，Nginx 就是一个不错的选择，它是一个高性能的 HTTP 和反向代理 Web 服务器。下一篇我们就使用 Nginx 整合 FastDFS 实现文件服务器的搭建。