4-FastDFS集群部署和同步机制分析-课件

零声教育 Darren QQ: 326873713
https://ke.qq.com/course/420945?tuin=137bb271

主要内容：

- 多个tracker和server的搭建
- storage的轮询机制算法
- storage同步方式
- 一个大文件上传到storage1，然后同步到storage2，在还没有同步完数据的情况下客户端是否会从storage2访问该文件
- 文件同步机制

storage–>tracker

目的：掌握storage如何上报信息给tracker，加深对fastdfs架构的理解

打印级别设置，把tracker.conf和storage.conf的 log_level设置为debug
/etc/init.d/fdfs_trackerd restart
/etc/init.d/fdfs_storaged stop

监测日志：
tail –f /home/fastdfs/tracker/logs/trackerd.log

课程内容顺序

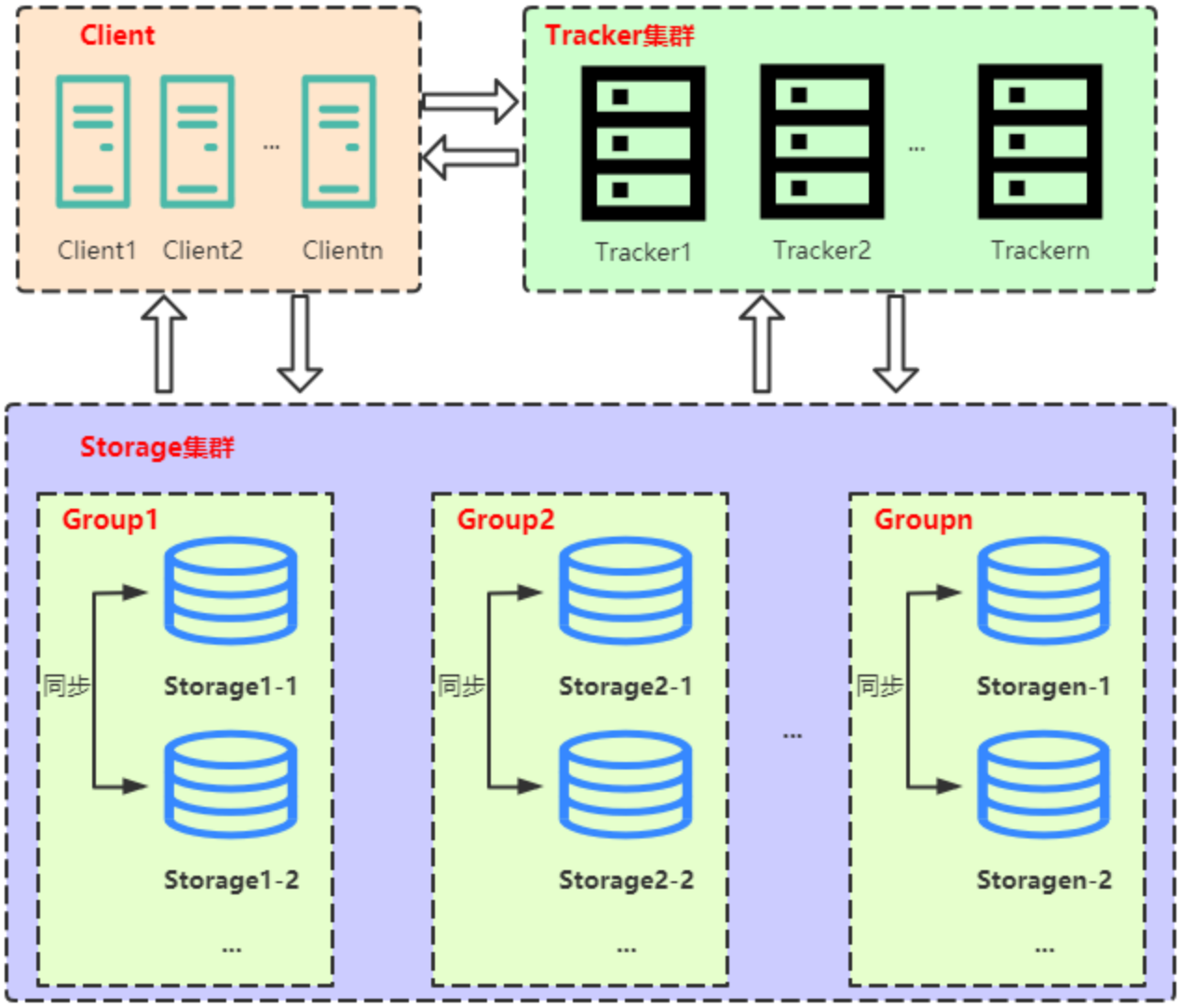
1. 复习文件上传的整个逻辑
2. 在掌握FastDFS整个逻辑的基础上搭建2个tracker、2个storage的服务
3. 掌握tracker、storage的存储结构，目的是掌握tracker、storage的目录结构以及存储了哪些数据信息，比如storage要存储同步状态
4. 掌握同步机制，主要是要理解原理，理解为什么是这么做同步；不要急着去一行行分析源码，除非你刚好要做这样的项目
5.

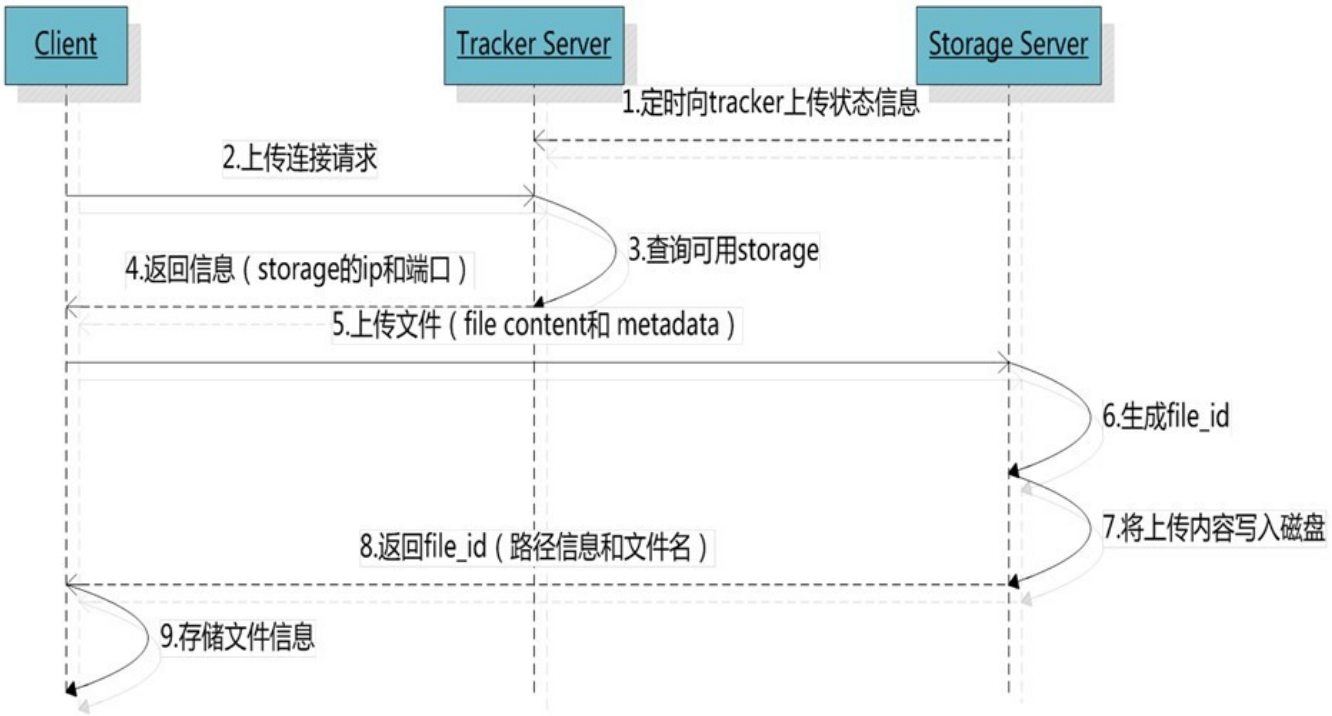
0 基于文件上传复习FastDFS架构

回顾文件上传机制，思考以下几个点：

1. 如何选择tracker
2. 如何选择group
3. 如何选择storage
4. 如何选择path
5. 下载文件的时候如何选择storage

FastDFS架构


Client
Client1
Client2
...
Clientn
Tracker集群
Tracker1
Tracker2
...
Trackern
Storage集群
Group1
同步
Storage1-1
Storage1-2
...
Group2
同步
Storage2-1
Storage2-2
...
...
Groupn
同步
Storagen-1
Storagen-2
...



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，选择最大剩余空 间的组上传文件

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。

文件名规则：

- storage_id（ip的数值型）源storage server ID或IP地址
- timestamp（文件创建时间戳）
- file_size（若原始值为32位则前面加入一个随机值填充，最终为64位）
- crc32（文件内容的检验码）

随机数 （引入随机数的目的是防止生成重名文件）

SQL 复制代码

```
eBuDxWCb2qmAQ89yAAAAKeR1iIo162
| 4bytes | 4bytes    | 8bytes    |4bytes | 2bytes |
| ip     | timestamp | file_size |crc32  | 校验值 |
```

1 部署2个tracker server，两个storage server

部署2个tracker server，两个storage server。

ps：模拟测试时多个tracker可以部署在同一台机器上，但是storage不能部署在同一台机器上。

规划

服务器地址	服务程序	对应配置文件(端口区分)	
120.27.131.197	fdfs_trackerd	tracker_22122.conf **22124**	
120.27.131.197	fdfs_trackerd	tracker_22123.conf **22124**	
120.27.131.197	fdfs_storaged	storage_group1_23000.conf	
114.215.169.66	fdfs_storaged	storage_group1_23000.conf	

1.1 120.27.131.197服务器

进入

cd /etc/fdfs

cp tracker.conf.sample tracker_22122.conf

cp tracker.conf.sample tracker_22123.conf

mkdir /home/fastdfs/tracker_22122 同一个服务器创建多个tracker存储路径

mkdir /home/fastdfs/tracker_22123

cp storage.conf.sample storage_group1_23000.conf

mkdir /home/fastdfs/storage_group1_23000

把现有的tracker、storage全部停止

```bash
root@iZbp1d83xkvoja33dm7ki2Z:/etc/fdfs# ps -ef | grep tracker
root     17405     1  0 17:40 ?        00:00:01 /usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
root     18074 17189  0 22:01 pts/3    00:00:00 grep --color=auto tracker
root@iZbp1d83xkvoja33dm7ki2Z:/etc/fdfs# kill -9 17405

root@iZbp1d83xkvoja33dm7ki2Z:/etc/fdfs# ps -ef | grep storage
root     16219     1  0 11:33 ?        00:00:06 fdfs_storaged /etc/fdfs/storage.conf
root     18085 17189  0 22:11 pts/3    00:00:00 grep --color=auto storage
root@iZbp1d83xkvoja33dm7ki2Z:/etc/fdfs# kill -9 16219


```

然后我们要修改对应的配置文件

tracker_22122.conf

在这里，tracker_22122.conf 只是修改一下 Tracker 存储日志和数据的路径

```bash
# 启用配置文件（默认为 false，表示启用配置文件）
disabled=false
# Tracker 服务端口（默认为 22122）
port=22122
# 存储日志和数据的根目录
base_path=/home/fastdfs/tracker_22122
```

主要修改port、base_path路径。

启动tracker_22122

```bash
/usr/bin/fdfs_trackerd /etc/fdfs/tracker_22122.conf
```

tracker_22123.conf

在这里，tracker.conf 只是修改一下 Tracker 存储日志和数据的路径

```bash
# 启用配置文件（默认为 false，表示启用配置文件）
disabled=false
# Tracker 服务端口（默认为 22122）
port=22123
# 存储日志和数据的根目录
base_path=/home/fastdfs/tracker_22123
```

主要修改port、base_path路径。

启动tracker_22123

```bash
/usr/bin/fdfs_trackerd /etc/fdfs/tracker_22123.conf
```

此时查看启动的tracker

root@iZbp1d83xkvoja33dm7ki2Z:/etc/fdfs# ps –ef | grep tracker

root 18100 1 0 22:12 ? 00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker_22122.conf

root 18138 1 0 22:13 ? 00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker_22123.conf

root 18146 17189 0 22:13 pts/3 00:00:00 grep ––color=auto tracker

storage_group1_23000.conf

在这里，storage_group1_23000.conf 只是修改一下 storage 存储日志和数据的路径

```bash
# 启用配置文件（默认为 false，表示启用配置文件）
disabled=false
# Storage 服务端口（默认为 23000）
port=23000
# 数据和日志文件存储根目录
base_path=/home/fastdfs/storage_group1_23000
# 存储路径，访问时路径为 M00
# store_path1 则为 M01，以此递增到 M99（如果配置了多个存储目录的话，这里只指定 1 个）
store_path0=/home/fastdfs/storage_group1_23000
# Tracker 服务器 IP 地址和端口，单机搭建时也不要写 127.0.0.1
# tracker_server 可以多次出现，如果有多个，则配置多个
tracker_server=120.27.131.197:22122
tracker_server=120.27.131.197:22123
```

主要修改：port、base_path、store_path0、tracker_server

启动storage_group1_23000

```bash
/usr/bin/fdfs_storaged /etc/fdfs/storage_group1_23000.conf
```

1.2 114.215.169.66服务器

storage_group1_23000.conf

在这里，storage_group1_23000.conf 只是修改一下 storage 存储日志和数据的路径

```bash
# 启用配置文件（默认为 false，表示启用配置文件）
disabled=false
# Storage 服务端口（默认为 23000）
port=23000
# 数据和日志文件存储根目录
base_path=/home/fastdfs/storage_group1_23000
# 存储路径，访问时路径为 M00
# store_path1 则为 M01，以此递增到 M99（如果配置了多个存储目录的话，这里只指定 1 个）
store_path0=/home/fastdfs/storage_group1_23000
# Tracker 服务器 IP 地址和端口，单机搭建时也不要写 127.0.0.1
# tracker_server 可以多次出现，如果有多个，则配置多个
tracker_server=120.27.131.197:22122
tracker_server=120.27.131.197:22123
```

主要修改：port、base_path、store_path0、tracker_server

启动storage_group1_23000

```bash
/usr/bin/fdfs_storaged /etc/fdfs/storage_group1_23000.conf
```

1.3 测试

配置client.conf

创建client目录：mkdir /home/fastdfs/client

修改client.conf

```bash
# 修改client的base path路径
base_path = /home/fastdfs/client
# 配置tracker server地址
tracker_server=120.27.131.197:22122
tracker_server=120.27.131.197:22123
```

配置mod_fastdfs.conf

修改vim /etc/fdfs/mod_fastdfs.conf

store_path0=/home/fastdfs/storage_group1_23000#保存日志目录，跟storage 一致即可
tracker_server = 120.27.131.197:22122
tracker_server=120.27.131.197:22123 #tracker服务器的IP地址以及端口号，确保跟storage 一致即可

```bash
# Tracker 服务器IP和端口修改
tracker_server=120.27.131.197:22122
tracker_server=120.27.131.197:22123
# url 中是否包含 group 名称，改为 true，包含 group
url_have_group_name = true
# 配置 Storage 信息，修改 store_path0 的信息
store_path0=/home/fastdfs/storage_group1_23000
# 其它的一般默认即可，例如
base_path=/tmp
group_name=group1
# storage服务器端口号
storage_server_port=23000
#存储路径数量，必须和storage.conf文件一致
store_path_count=1
```

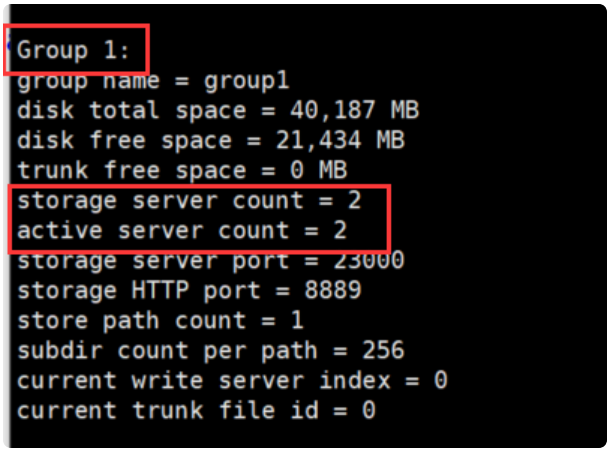
主要修改tracker_server、url_have_group_name、store_path0。

检测是否正常启动

分别在两台服务器执行：

/usr/bin/fdfs_monitor /etc/fdfs/storage_group1_23000.conf

正常两边都提示：



存在2个Active的storage。

测试上传文件

Bash 复制代码

```
/usr/bin/fdfs_upload_file   /etc/fdfs/client.conf
/etc/fdfs/storage_group1_23000.conf
```

返回 group1/M00/00/00/eBuDxWIgcYqACM8LAAAoaTcAqLc40.conf

查看两台服务器下的00/00目录是否存在相同的文件。

下载测试

（1）正常下载

```
fdfs_download_file /etc/fdfs/client.conf
group1/M00/00/00/ctepQmCotjqAIRNPAAAjuZXPuAg28.conf
```

（2）停止120.27.131.197的storage

```
/usr/bin/fdfs_storaged /etc/fdfs/storage_group1_23000.conf  stop
```

然后再下载数据

```
fdfs_download_file /etc/fdfs/client.conf
group1/M00/00/00/ctepQmCotjqAIRNPAAAjuZXPuAg28.conf
```

此时还可以正常下载数据

（3）继续停止另一个storage server(114.215.169.66)

```
/usr/bin/fdfs_storaged /etc/fdfs/storage_group1_23000.conf  stop
```

然后再继续下载数据

```
fdfs_download_file /etc/fdfs/client.conf
group1/M00/00/00/ctepQmCotjqAIRNPAAAjuZXPuAg28.conf
```

此时就报错了，因为storage都已经停止了。

root@iZbp1d83xkvoja33dm7ki2Z:~# fdfs_download_file /etc/fdfs/client.conf group1/M00/00/00/eBuDxWIgcYqACM8LAAAoaTcAqLc40.conf[2021-05-22 15:49:51] ERROR - file: tracker_proto.c, line: 50, server: 120.27.131.197:22122, response status 2 != 0

[2021-05-22 15:49:51] ERROR - file: ../client/tracker_client.c, line: 716, fdfs_recv_response fail, result: 2
download file fail, error no: 2, error info: No such file or directory

PS：可以使用浏览器去测试：
http://120.27.131.197:80/group1/M00/00/00/eBuDxWIgcYqACM8LAAAoaTcAqLc40.conf

恢复storage的运行

两台服务器都执行：/usr/bin/fdfs_storaged /etc/fdfs/storage_group1_23000.conf

PS：可以先恢复一台storage，然后上传文件，再恢复另一台storage，然后在新启动的storage观察文件是否被同步。

1.4 拓展阅读

FastDFS tracker leader机制介绍

https://www.yuque.com/docs/share/130e0460-fed5-41d7-bd32-c3b4bb2e4a1d?# 《FastDFS tracker leader机制介绍》

FastDFS配置详解之Tracker配置

https://www.yuque.com/docs/share/0294fba8-a1d4-4e86-a43f-cb289ec636be?# 《FastDFS配置详解之Tracker配置》

FastDFS配置详解之Storage配置

https://www.yuque.com/docs/share/21dda82f-5d44-4e71-87e4-0bac39731b20?# 《FastDFS配置详解之Storage配置》

FastDFS集群部署指南

https://www.yuque.com/docs/share/c903aba6-720c-4a36-8779-f78e3a0f6827?# 《FastDFS集群部署指南》

2 默认编译支持debug

debug子进程

follow-fork-mode的用法为：
set follow-fork-mode [parent|child]

- parent: fork之后继续调试父进程，子进程不受影响。
- child: fork之后调试子进程，父进程不受影响。

因此如果需要调试子进程，在启动gdb后：

```
(gdb) set follow-fork-mode child
```

因为我们的程序最终是以demon的方式运行，可以就涉及到了子进程运行的问题。

另一种方式 gdb attach pid进行跟踪调试。

3 storage->tracker

注意：该章节不是必要的，只是提供一个研究源码的方法而已，不是特别想深入研究源码的朋友不用理会。

1. 连接tracker附带的具体内容
2. tracker返回什么内容
3. 文件同步信息

准备工作

修改fastdfs源码和重新编译

1. 修改源码

fastdfs# vim tracker/tracker_service.c

在3912行增加红色部分的代码，不要另起一行，另一起一行会改变原有代码行的位置不方面课程讲解。
pHeader = (TrackerHeader *)pTask->data;**logNotice("cmd:%d, len:%ld\n", pHeader->cmd, *((long *)pHeader->pkg_len));**

2. 编译和安装

./make.sh
./make.sh install

3. 重启tracker

/etc/init.d/fdfs_trackerd restart

4. 监听log

tail –f /home/fastdfs/tracker/logs/trackerd.log

4 tracker和storage目录结构

4.1 tracker server目录及文件结构

Bash | 复制代码

```
/home/fastdfs/tracker_22122# tracker server目录及文件结构
├── data
│   ├── fdfs_trackerd.pid
│   ├── storage_changelog.dat   storage有修改过ip
│   ├── storage_groups_new.dat  存储分组信息
│   ├── storage_servers_new.dat 存储服务器列表
│   └── storage_sync_timestamp.dat 同步时间戳
└── logs
    └── trackerd.log Server日志文件

```

数据文件storage_groups.dat和storage_servers.dat中的记录之间以换行符（n）分隔，字段之间以西文逗号（,）分隔。

storage_changelog.dat

比如

1645866390 group1 114.215.169.66 3 172.19.24.119

storage_groups_new.dat

各个参数如下

group_name：组名

storage_port：storage server端口号

比如：

Bash 复制代码

```
# global section
[Global]
  group_count=1

# group: group1
[Group001]
  group_name=group1
  storage_port=23000
  storage_http_port=8888
  store_path_count=1
  subdir_count_per_path=256
  current_trunk_file_id=0
  trunk_server=
  last_trunk_server=

```

storage_servers.dat

比如

```bash
# storage 120.27.131.197:23000
[Storage001]
  group_name=group1
  ip_addr=120.27.131.197
  status=7
  version=6.07
  join_time=1646292828
....
# storage 114.215.169.66:23000
[Storage002]
  group_name=group1
  ip_addr=114.215.169.66
  status=7
  version=6.07
  join_time=1646292925
  storage_port=23000
```

主要参数如下
group_name：所属组名
ip_addr：ip地址
status：状态
sync_src_ip_addr：向该storage server同步已有数据文件的源服务器
sync_until_timestamp：同步已有数据文件的截至时间（UNIX时间戳）
stat.total_upload_count：上传文件次数
stat.success_upload_count：成功上传文件次数
stat.total_set_meta_count：更改meta data次数
stat.success_set_meta_count：成功更改meta data次数
stat.total_delete_count：删除文件次数
stat.success_delete_count：成功删除文件次数
stat.total_download_count：下载文件次数
stat.success_download_count：成功下载文件次数
stat.total_get_meta_count：获取meta data次数
stat.success_get_meta_count：成功获取meta data次数
stat.last_source_update：最近一次源头更新时间（更新操作来自客户端）
stat.last_sync_update：最近一次同步更新时间（更新操作来自其他storage server的同步）

4.2 storage server目录及文件结构

```
|__data
|     |__.data_init_flag：当前storage server初始化信息
|     |__storage_stat.dat：当前storage server统计信息
|     |__sync：存放数据同步相关文件
|     |     |__binlog.index：当前的binlog（更新操作日志）文件索引号
|     |     |__binlog.###：存放更新操作记录（日志）
|     |     |__${ip_addr}_${port}.mark：存放向目标服务器同步的完成情况，比如
|     |                                  114.215.169.66_23000.mark
|     |__一级目录：256个存放数据文件的目录，目录名为十六进制字符，如：00, 1F
|           |__二级目录：256个存放数据文件的目录，目录名为十六进制字符，如：0A, CF
|__logs
      |__storaged.log：storage server日志文件
```

.data_init_flag文件格式为ini配置文件方式

各个参数如下

storage_join_time：本storage server创建时间

sync_old_done：本storage server是否已完成同步的标志（源服务器向本服务器同步已有数据）

sync_src_server：向本服务器同步已有数据的源服务器IP地址，没有则为空

sync_until_timestamp：同步已有数据文件截至时间（UNIX时间戳）

storage_stat.dat文件格式为ini配置文件方式

各个参数如下：

total_upload_count：上传文件次数

success_upload_count：成功上传文件次数

total_set_meta_count：更改meta data次数

success_set_meta_count：成功更改meta data次数

total_delete_count：删除文件次数

success_delete_count：成功删除文件次数

total_download_count：下载文件次数

success_download_count：成功下载文件次数

total_get_meta_count：获取meta data次数

success_get_meta_count：成功获取meta data次数

last_source_update：最近一次源头更新时间（更新操作来自客户端）

last_sync_update：最近一次同步更新时间（更新操作来自其他storage server）

sync 目录及文件结构

- **binlog.index**中只有一个数据项：当前binlog的文件索引号 binlog.###，
- **binlog.###**为索引号对应的3位十进制字符，不足三位，前面补0。索引号基于0，最大为999。
  一个binlog文件最大为1GB。记录之间以换行符（n）分隔，字段之间以西文空格分隔。

字段依次为：

1. **timestamp**：更新发生时间（Unix时间戳）
2. **op_type**：操作类型，一个字符
3. **filename**：操作（更新）的文件名，包括相对路径，如：5A/3D/FE_93_SJZ7pAAAO_BXYD.S

- **${ip_addr}_${port}.mark**：
  ip_addr为同步的目标服务器IP地址，port为本组storage server端口。例如：10.0.0.1_23000.mark。
  各个参数如下：
  - binlog_index：已处理（同步）到的binlog索引号
  - binlog_offset：已处理（同步）到的binlog文件偏移量（字节数）
  - need_sync_old：同步已有数据文件标记，0表示没有数据文件需要同步
  - sync_old_done：同步已有数据文件是否完成标记，0表示未完成，1表示已完成
  - until_timestamp：同步已有数据截至时间点（UNIX时间戳）
  - scan_row_count：已扫描的binlog记录数
  - sync_row_count：已同步的binlog记录数

5 tracker主要线程处理核心

int tracker_deal_task(struct fast_task_info *pTask)

1 storage心跳协议

C 复制代码

```c
case TRACKER_PROTO_CMD_STORAGE_BEAT:  //  83 storage heart beat
            TRACKER_CHECK_LOGINED(pTask)
            result = tracker_deal_storage_beat(pTask);
            break;
```

storage->tracker的心跳包，storage在启动的时候，会开启tracker_report_thread_entrance线程（**注意：每个storage->tracker都有唯一的线程，连接2个tracker就有2个线程**）：

```c
static void *tracker_report_thread_entrance(void *arg)
```

该函数主要是根据配置连接相应的它的组的tacker，并维持和tracker之间的联系，代码如下

```c
current_time = g_current_time;
if (current_time - last_beat_time >=
    g_heart_beat_interval)  // 默认30秒报告一次
{
    if (tracker_heart_beat(conn, tracker_index,
                           &stat_chg_sync_count,
                           &bServerPortChanged) != 0)
    {
        break;
    }

    if (g_storage_ip_changed_auto_adjust &&
        tracker_storage_changelog_req(conn) != 0)
    {
        break;
    }

    last_beat_time = current_time;
}

```

默认至少30秒钟发1次心跳，心跳包的主要数据是包头和当前storage的状态信息，在storage.conf可以配置heart_beat_interval心跳间隔。
上报的内容结构体如下所示，主要是上传下载修改、同步时间等的统计信息，我们这里主要关注的是同步时间相关的信息。

```c
/* struct for network transfering */
typedef struct
{
    struct {
        char sz_alloc_count[4];
        char sz_current_count[4];
        char sz_max_count[4];
    } connection;

    char sz_total_upload_count[8];      // 总的上传次数
    char sz_success_upload_count[8];    // 成功上传次数
    char sz_total_append_count[8];
    char sz_success_append_count[8];
    char sz_total_modify_count[8];
    ....省略
    char sz_success_sync_out_bytes[8];
    char sz_total_file_open_count[8];
    char sz_success_file_open_count[8];
    char sz_total_file_read_count[8];
    char sz_success_file_read_count[8];
    char sz_total_file_write_count[8];
    char sz_success_file_write_count[8];
    char sz_last_source_update[8];  // 最新的源更新
    char sz_last_sync_update[8];    // 最新的同步
    char sz_last_synced_timestamp[8];   // 最新的同步时间
    char sz_last_heart_beat_time[8];    // 最新的心跳时间
} FDFSStorageStatBuff;
```

char out_buff[sizeof(TrackerHeader) + sizeof(FDFSStorageStatBuff)];
TrackerHeader里面有cmd字段指明为：TRACKER_PROTO_CMD_STORAGE_BEAT

tracker主要是做了什么呢?

对其进行解包，然后对这个保存在本地的storage的信息进行保存到文件中，调用

```c
status = tracker_save_storages();
```

调用

```c
tracker_mem_active_store_server(pClientInfo->pGroup, \
            pClientInfo->pStorage);
```

将这个存储服务器如果没有，就插入到group中。因为storage是分组的。

最后调用

```c
static int tracker_check_and_sync(struct fast_task_info *pTask, \
            const int status)
```

检查相应的改变状态，并将其同步等。（需要再详细看看）

具体的统计在：storage_service.c

storage_upload_file_done_callback

xxx done_callback

2 报告相应同步时间

```c
#define TRACKER_PROTO_CMD_STORAGE_SYNC_REPORT  89  //report src last
synced time as dest server
```

同样在storage的tracker_report_thread_entrance线程执行

```c
if (sync_time_chg_count != g_sync_change_count &&       // 报告相应同步时间
                current_time - last_sync_report_time >=
                    g_heart_beat_interval)
{
    if (tracker_report_sync_timestamp(conn,
                                      tracker_index,
                                      &bServerPortChanged) != 0)
    {
        break;
    }

    sync_time_chg_count = g_sync_change_count;
    last_sync_report_time = current_time;
}
```

具体的数据包为

```c
pEnd = g_storage_servers + g_storage_count;
for (pServer=g_storage_servers; pServer<pEnd; pServer++)
{
    memcpy(p, pServer->server.id, FDFS_STORAGE_ID_MAX_SIZE);
    p += FDFS_STORAGE_ID_MAX_SIZE;
    int2buff(pServer->last_sync_src_timestamp, p);
    p += 4;
}
```

也就是遍历当前进程的本组所有storage服务器（数据内容相同），和上次同步的时间戳，给tracker服务器。

然后tracker的服务器存储结构为

```c
pClientInfo->pGroup->last_sync_timestamps \
                [src_index][dest_index] = sync_timestamp;
```

dest_index 值为当前连接所在组的索引值

```c
dest_index = tracker_mem_get_storage_index(pClientInfo->pGroup,
            pClientInfo->pStorage);
if (dest_index < 0 || dest_index >= pClientInfo->pGroup->count)
{
    status = 0;
    break;
}
```

因为 本链接的storage是固定不变的，而src_index就是为本组的其他storage的id索引，
首相通过id，（ip地址）找到具体的storage，然后在通过指针找到索引位置，最后，调用

```c
    if (++g_storage_sync_time_chg_count % \
                    TRACKER_SYNC_TO_FILE_FREQ == 0)
    {
        status = tracker_save_sync_timestamps();
    }
    else
    {
        status = 0;
    }
} while (0);

return tracker_check_and_sync(pTask, status);
```

定时保存文件和检查等

3 上报磁盘情况

```c
#define TRACKER_PROTO_CMD_STORAGE_REPORT_DISK_USAGE 84  //report disk usage
```

同样线程定时调用，

```c
if (current_time - last_df_report_time >=
                    g_stat_report_interval)  // 默认是300秒间隔
{
    if (tracker_report_df_stat(conn,
                               tracker_index,
                               &bServerPortChanged) != 0)
    {
        break;
    }

    last_df_report_time = current_time;
}
```

同样上报这些数据

```c
for (i=0; i<g_fdfs_store_paths.count; i++)
{
    if (statvfs(g_fdfs_store_paths.paths[i].path, &sbuf) != 0)
    {
        logError("file: "__FILE__", line: %d, " \
                 "call statfs fail, errno: %d, error info: %s.",\
                 __LINE__, errno, STRERROR(errno));

        if (pBuff != out_buff)
        {
            free(pBuff);
        }
        return errno != 0 ? errno : EACCES;
    }

    g_fdfs_store_paths.paths[i].total_mb = ((int64_t)(sbuf.f_blocks) * \
                                            sbuf.f_frsize) / FDFS_ONE_MB;
    g_fdfs_store_paths.paths[i].free_mb = ((int64_t)(sbuf.f_bavail) * \
                                           sbuf.f_frsize) / FDFS_ONE_MB;
    long2buff(g_fdfs_store_paths.paths[i].total_mb, pStatBuff->sz_total_mb);
    long2buff(g_fdfs_store_paths.paths[i].free_mb, pStatBuff->sz_free_mb);

    pStatBuff++;
}
```

tracker这边在tracker_deal_storage_df_report函数响应，**存储在**

```c
path_total_mbs = pClientInfo->pStorage->path_total_mbs;
path_free_mbs = pClientInfo->pStorage->path_free_mbs;
```

这里

```c
path_total_mbs[i] = buff2long(pStatBuff->sz_total_mb);
path_free_mbs[i] = buff2long(pStatBuff->sz_free_mb);

pClientInfo->pStorage->total_mb += path_total_mbs[i];
pClientInfo->pStorage->free_mb += path_free_mbs[i];
```

4 storage服加入到tracker

```c
#define TRACKER_PROTO_CMD_STORAGE_JOIN             81
```

storage线程同样在该处调用

```c
if (tracker_report_join(conn, tracker_index,
                    sync_old_done) != 0)
{
    sleep(g_heart_beat_interval);
    continue;
}
```

发送的包体数据包为：

```c
typedef struct
{
    char group_name[FDFS_GROUP_NAME_MAX_LEN+1];
    char storage_port[FDFS_PROTO_PKG_LEN_SIZE];
    char storage_http_port[FDFS_PROTO_PKG_LEN_SIZE];
    char store_path_count[FDFS_PROTO_PKG_LEN_SIZE];
    char subdir_count_per_path[FDFS_PROTO_PKG_LEN_SIZE];
    char upload_priority[FDFS_PROTO_PKG_LEN_SIZE];
    char join_time[FDFS_PROTO_PKG_LEN_SIZE]; //storage join timestamp
    char up_time[FDFS_PROTO_PKG_LEN_SIZE];   //storage service started timestamp
    char version[FDFS_VERSION_SIZE];   //storage version
    char domain_name[FDFS_DOMAIN_NAME_MAX_SIZE];
    char init_flag;
    signed char status;
    char tracker_count[FDFS_PROTO_PKG_LEN_SIZE];  //all tracker server count
} TrackerStorageJoinBody;
```

当赋值完成后，在其后变加入

```c
p = out_buff + sizeof(TrackerHeader) + sizeof(TrackerStorageJoinBody);
pServerEnd = g_tracker_group.servers + g_tracker_group.server_count;
for (pServer=g_tracker_group.servers; pServer<pServerEnd; pServer++)
{
    fdfs_server_info_to_string(pServer, p,
                               FDFS_PROTO_MULTI_IP_PORT_SIZE);
    p += FDFS_PROTO_MULTI_IP_PORT_SIZE;
}
```

加入所有tracker的服务器信息格式为ip:port

tracker 服务器接收

```c
case TRACKER_PROTO_CMD_STORAGE_JOIN:
    result = tracker_deal_storage_join(pTask);
    break;
```

获取到的相关信息存储到

```c
typedef struct
{
    int storage_port;
    int storage_http_port;
    int store_path_count;
    int subdir_count_per_path;
    int upload_priority;
    int join_time; //storage join timestamp (create timestamp)
    int up_time;   //storage service started timestamp
    char version[FDFS_VERSION_SIZE];   //storage version
    char group_name[FDFS_GROUP_NAME_MAX_LEN + 1];
    char domain_name[FDFS_DOMAIN_NAME_MAX_SIZE];
    char init_flag;
    signed char status;
    int tracker_count;
    ConnectionInfo tracker_servers[FDFS_MAX_TRACKERS];
} FDFSStorageJoinBody;
```

这些结构体内
同时插入本地内存

```c
result = tracker_mem_add_group_and_storage(pClientInfo, \
                pTask->client_ip, &joinBody, true);
```

同时把发消息报的id传过来

```c
pJoinBodyResp = (TrackerStorageJoinBodyResp *)(pTask->data + \
            sizeof(TrackerHeader));
memset(pJoinBodyResp, 0, sizeof(TrackerStorageJoinBodyResp));

if (pClientInfo->pStorage->psync_src_server != NULL)
{
    strcpy(pJoinBodyResp->src_id, \
        pClientInfo->pStorage->psync_src_server->id);
}
```

5 报告存储状态

和其他报告不在同一线程，而是在storage_sync_thread_entrance

```c
#define TRACKER_PROTO_CMD_STORAGE_REPORT_STATUS     76  //report
specified storage server status
```

storage服务器调用

```c
int tracker_report_storage_status(ConnectionInfo *pTrackerServer, \
        FDFSStorageBrief *briefServer)
```

内容主要是组名字

```c
strcpy(out_buff + sizeof(TrackerHeader), g_group_name);
```

和简要信息

```c
memcpy(out_buff + sizeof(TrackerHeader) + FDFS_GROUP_NAME_MAX_LEN, \
        briefServer, sizeof(FDFSStorageBrief));
```

其结构体如下

```c
typedef struct
{
    char status;
    char port[4];
    char id[FDFS_STORAGE_ID_MAX_SIZE];
    char ip_addr[IP_ADDRESS_SIZE];
} FDFSStorageBrief;
```

6 client->tracker:从tracker获取storage状态。

#define TRACKER_PROTO_CMD_STORAGE_GET_STATUS 71 //get storage status from tracker

该协议是由client发起
调用流程如下：

```c
int tracker_get_storage_status(ConnectionInfo *pTrackerServer, \
        const char *group_name, const char *ip_addr, \
        FDFSStorageBrief *pDestBuff)
int tracker_get_storage_max_status(TrackerServerGroup *pTrackerGroup, \
        const char *group_name, const char *ip_addr, \
        char *storage_id, int *status)
int tracker_get_storage_status(ConnectionInfo *pTrackerServer, \
        const char *group_name, const char *ip_addr, \
        FDFSStorageBrief *pDestBuff)
```

获取自己的状态，包体格式　组名 ip的字符串

tracker通过获取了相应的数据，查找到storage的信息，结构体为：

```c
typedef struct
{
    char status;
    char port[4];
    char id[FDFS_STORAGE_ID_MAX_SIZE];
    char ip_addr[IP_ADDRESS_SIZE];
} FDFSStorageBrief;
```

赋值后，返回

7 回复给新的storage

```c
#define TRACKER_PROTO_CMD_STORAGE_REPLICA_CHG       85  //repl new storage servers
```

storage服务器调用流程，通过tracker_sync_diff_servers函数请求

```c
static int tracker_merge_servers(ConnectionInfo *pTrackerServer, \
        FDFSStorageBrief *briefServers, const int server_count)
```

8 剩下的协议

C 复制代码

```c
case TRACKER_PROTO_CMD_SERVICE_QUERY_FETCH_ONE:
            result = tracker_deal_service_query_fetch_update( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVICE_QUERY_UPDATE:
            result = tracker_deal_service_query_fetch_update( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVICE_QUERY_FETCH_ALL:
            result = tracker_deal_service_query_fetch_update( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ONE:
            result = tracker_deal_service_query_storage( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITH_GROUP_ONE:
            result = tracker_deal_service_query_storage( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ALL:
            result = tracker_deal_service_query_storage( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITH_GROUP_ALL:
            result = tracker_deal_service_query_storage( \
                    pTask, pHeader->cmd);
            break;
        case TRACKER_PROTO_CMD_SERVER_LIST_ONE_GROUP:
            result = tracker_deal_server_list_one_group(pTask);
            break;
        case TRACKER_PROTO_CMD_SERVER_LIST_ALL_GROUPS:
            result = tracker_deal_server_list_all_groups(pTask);
            break;
        case TRACKER_PROTO_CMD_SERVER_LIST_STORAGE:
            result = tracker_deal_server_list_group_storages(pTask);
            break;
        case TRACKER_PROTO_CMD_STORAGE_SYNC_SRC_REQ:
            result = tracker_deal_storage_sync_src_req(pTask);
            break;
        case TRACKER_PROTO_CMD_STORAGE_SYNC_DEST_REQ:
            TRACKER_CHECK_LOGINED(pTask)
            result = tracker_deal_storage_sync_dest_req(pTask);
            break;
        case TRACKER_PROTO_CMD_STORAGE_SYNC_NOTIFY:
```

```
46              result = tracker_deal_storage_sync_notify(pTask);
47              break;
48          case TRACKER_PROTO_CMD_STORAGE_SYNC_DEST_QUERY:
49              result = tracker_deal_storage_sync_dest_query(pTask);
50              break;
51          case TRACKER_PROTO_CMD_SERVER_DELETE_STORAGE:
52              result = tracker_deal_server_delete_storage(pTask);
53              break;
54          case TRACKER_PROTO_CMD_SERVER_SET_TRUNK_SERVER:
55              result = tracker_deal_server_set_trunk_server(pTask);
56              break;
57          case TRACKER_PROTO_CMD_STORAGE_REPORT_IP_CHANGED:
58              result = tracker_deal_storage_report_ip_changed(pTask);
59              break;
60          case TRACKER_PROTO_CMD_STORAGE_CHANGELOG_REQ:
61              result = tracker_deal_changelog_req(pTask);
62              break;
63          case TRACKER_PROTO_CMD_STORAGE_PARAMETER_REQ:
64              result = tracker_deal_parameter_req(pTask);
65              break;
66          case FDFS_PROTO_CMD_QUIT:
67              close(pTask->ev_read.ev_fd);
68              task_finish_clean_up(pTask);
69              return 0;
70          case FDFS_PROTO_CMD_ACTIVE_TEST:
71              result = tracker_deal_active_test(pTask);
72              break;
73          case TRACKER_PROTO_CMD_TRACKER_GET_STATUS:
74              result = tracker_deal_get_tracker_status(pTask);
75              break;
76          case TRACKER_PROTO_CMD_TRACKER_GET_SYS_FILES_START:
77              result = tracker_deal_get_sys_files_start(pTask);
78              break;
79          case TRACKER_PROTO_CMD_TRACKER_GET_ONE_SYS_FILE:
80              result = tracker_deal_get_one_sys_file(pTask);
81              break;
82          case TRACKER_PROTO_CMD_TRACKER_GET_SYS_FILES_END:
83              result = tracker_deal_get_sys_files_end(pTask);
84              break;
85          case TRACKER_PROTO_CMD_STORAGE_REPORT_TRUNK_FID:
86              TRACKER_CHECK_LOGINED(pTask)
87              result = tracker_deal_report_trunk_fid(pTask);
88              break;
89          case TRACKER_PROTO_CMD_STORAGE_FETCH_TRUNK_FID:
90              TRACKER_CHECK_LOGINED(pTask)
91              result = tracker_deal_get_trunk_fid(pTask);
92              break;
93          case TRACKER_PROTO_CMD_STORAGE_REPORT_TRUNK_FREE:
```

```
94                TRACKER_CHECK_LOGINED(pTask)
95                result = tracker_deal_report_trunk_free_space(pTask);
96                break;
97            case TRACKER_PROTO_CMD_TRACKER_PING_LEADER:
98                result = tracker_deal_ping_leader(pTask);
99                break;
100           case TRACKER_PROTO_CMD_TRACKER_NOTIFY_NEXT_LEADER:
101               result = tracker_deal_notify_next_leader(pTask);
102               break;
103           case TRACKER_PROTO_CMD_TRACKER_COMMIT_NEXT_LEADER:
104               result = tracker_deal_commit_next_leader(pTask);
105               break;
```

6 FastDFS文件同步

提问：

1. 两个storage如何相互备份，会不会出现备份死循环的问题
2. 已经存在两个storage了，然后加入第三个storage，那谁同步给第三个storage
3. binlog的格式是怎么设计的，如果binlog文件太大该怎么处理
4. 已有A、B、C三个storage，我现在上传一个文件到A，然后发起请求下载，会不会出现从B请求下载，但此时B没有同步文件，导致下载失败。

线程：

- tracker_report_thread_entrance ， 连接tracker有独立的线程，连接n个tracker就有n个线程
- storage_sync_thread_entrance ， 给同group的storage做同步，同组有n个storage，就有n-1个线程

storage的状态

- #define FDFS_STORAGE_STATUS_INIT 0
- #define FDFS_STORAGE_STATUS_WAIT_SYNC 1
- #define FDFS_STORAGE_STATUS_SYNCING 2
- #define FDFS_STORAGE_STATUS_IP_CHANGED 3
- #define FDFS_STORAGE_STATUS_DELETED 4
- #define FDFS_STORAGE_STATUS_OFFLINE 5

- #define FDFS_STORAGE_STATUS_ONLINE 6
- #define FDFS_STORAGE_STATUS_ACTIVE 7
- #define FDFS_STORAGE_STATUS_RECOVERY 9

同步命令

#define STORAGE_PROTO_CMD_SYNC_CREATE_FILE 16 //新增文件
#define STORAGE_PROTO_CMD_SYNC_DELETE_FILE 17 // 删除文件
#define STORAGE_PROTO_CMD_SYNC_UPDATE_FILE 18 // 更新文件
#define STORAGE_PROTO_CMD_SYNC_CREATE_LINK 19 // 创建链接

binlog_index.dat 中只有一个数据项：当前binlog的文件索引号
binlog.###，###为索引号对应的3位十进制字符，不足三位，前面补0。索引号基于0，最大为999。一个binlog文件最大为1GB。记录之间以换行符（/n）分隔，字段之间以西文空格分隔。字段依次为：

- timestamp：更新发生时间（Unix时间戳）
- op_type：操作类型，一个字符
- filename：操作（更新）的文件名，包括相对路径，如：5A/3D/FE_93_SJZ7pAAAO_BXYD.S

${ip_addr}_${port}.mark
ip_addr为同步的目标服务器IP地址，port为本组storage server端口。例如：10.0.0.1_23000.mark。文件格式为ini配置文件方式，各个参数如下：

- binlog_index：已处理（同步）到的binlog索引号
- binlog_offset：已处理（同步）到的binlog文件偏移量（字节数）
- need_sync_old：同步已有数据文件标记，0表示没有数据文件需要同步
- sync_old_done：同步已有数据文件是否完成标记，0表示未完成，1表示已完成
- until_timestamp：同步已有数据截至时间点（UNIX时间戳）
- scan_row_count：已扫描的binlog记录数
- sync_row_count：已同步的binlog记录数

数据文件名由系统自动生成，包括三部分：当前时间（Unix时间戳）、文件大小（字节数）和随机数。文件名长度为16字节。文件按照PJW Hash算法hash到65536（256*256）个目录中分散存储。

6.1 同步日志所在目录

比如在**120.27.131.197**服务器看到的sync log：

> root@xx:/home/fastdfs/storage_group1_23000/data/sync#
>
> **114.215.169.66_23000**.mark **同步状态文件，记录本机到114.215.169.66的同步状态**
> **文件名由同步源IP_端口组成。**
> binlog.000 **binglog文件，文件大小最大1G，超过1G，会重新写下个文件，同时更新**

```
                              binlog.index 文件中索引值
binlog_index.dat                记录了当前写binlog的索引id。
```

如果有不只2个storage的时候，比如还有**114.215.169.67，**则该目录还存在**114.215.169.67_23000**.mark

比如在**114.215.169.66**服务器看到的sync log：

```
root@iZbp1h2l856zgoegc8rvnhZ:/home/fastdfs/storage_group1_23000/data/sync#

120.27.131.197_23000.mark   // 对应发送同步的storage
binlog.000                  // 本地的binglog日志，可以binlog.000， binlog.001，binlog.002
binlog_index.dat            // 记录当前在操作的binglog.xxx ，比如current_write=0 代表
binlog.000
```

6.2 binlog格式

FastDFS文件同步采用binlog异步复制方式。storage server使用binlog文件记录文件上传、删除等操作，根据binlog进行文件同步。binlog中只记录文件ID和操作，不记录文件内容。下面给出几行binlog文件内容示例：

1646123002 C M00/00/00/oYYBAF285cOlHiVCAACI-7zX1qUAAAAVgAACC8AAlkT490.txt
1646123047 c M00/00/00/oYYBAF285luIK8jCAAAJeheau6AAAAAVgABI-cAAAmS021.xml
1646123193 A M00/00/00/rBMYd2laLXqASSVXAAAHuj79dAY65.txt 6 6
1646123561 d M00/00/00/oYYBAF285luIK8jCAAAJeheau6AAAAAVgABI-cAAAmS021.xml

从上面可以看到，binlog文件有三列，依次为：

- **时间戳**
- **操作类型**
- **文件ID（不带group名称）**
  - Storage_id（ip的数值型）
  - timestamp（创建时间）
  - file_size（若原始值为32位则前面加入一个随机值填充，最终为64位）
  - crc32（文件内容的检验码）

文件操作类型采用单个字母编码，其中源头操作用大写字母表示，被同步的操作为对应的小写字母。文件操作字母含义如下：

源	副本
C ： 上传文件（upload）	c：副本创建
D：删除文件（delete）	d：副本删除
A：追加文件（append）	a：副本追加
M：部分文件更新（modify）	m：副本部分文件更新（modify）
U：整个文件更新（set metadata）	u：副本整个文件更新（set metadata）
T：截断文件（truncate）	t：副本截断文件（truncate）
L：创建符号链接（文件去重功能，相同内容只保存一份）	l：副本创建符号链接（文件去重功能，相同内容只保存一份）

同组内的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。文件同步只在同组内的storage server之间进行，采用push方式，**即源头服务器同步给本组的其他存储服务器**。对于同组的其他storage server，一台storage server分别启动一个线程进行文件同步。

注：源表示客户端直接操作的那个Storage即为源，，其他的Storage都为副本。

文件同步采用增量方式，**记录已同步的位置到mark文件中**。mark文件存放路径为$base_path/data/sync/。mark文件内容示例：

114.215.169.66_23000.mark

```
binlog_index=0          //binlog索引id 表示上次同步给114.215.169.66机器的最后一条binlog文件索引
binlog_offset=3944          //当前时间binlog 大小 表示上次同步给114.215.169.66机器的最后一条binlog偏移
                                    // 量，若程序重启了，也只要从这个位置开始向后同步即可。
need_sync_old=1             //是否需要同步老数据
sync_old_done=1              //是否同步完成
until_timestamp=1621667115 //同步已有数据文件的截至时间
scan_row_count=68           //扫描记录数
sync_row_count=53          //同步记录数
```

120.27.131.197_23000.mark

```
binlog_index=0
```

```
binlog_offset=4350
need_sync_old=0
sync_old_done=0
until_timestamp=0
scan_row_count=75
sync_row_count=15
```

6.3 同步规则

1. 只在本组内的storage server之间进行同步；
2. 源头数据才需要同步，备份数据不需要再次同步，否则就构成环路了，源数据和备份数据区 分是用binlog的操作类型来区分，操作类型是大写字母，表示源数据，**小写字母表示备份数据**；
3. 当先新增加一台storage server时，由已有的一台storage server将已有的所有数据（包括源头数据和备份数据）同步给该新增服务器。

6.4 Binlog同步过程

在FastDFS之中，每个Storaged之间的同步都是由一个独立线程负责的，该线程中的所有操作都是以同步方式执行的。比如一组服务器有A、B、C三台机器，那么在每台机器上都有两个线程负责同步，如A机器，线程1负责同步数据到B，线程2负责同步数据到C。

1 获取组内的其他Storage信息tracker_report_thread_entrance ，并启动同步线程

tracker_report_thread_entrance 线程负责向tracker上报信息。

在Storage.conf配置文件中，只配置了Tracker的IP地址，并没有配置组内其他的Storage。因此同组的其他Storage必须从Tracker获取。具体过程如下：

1）Storage启动时为每一个配置的Tracker启动一个线程负责与该Tracker的通讯。

2）默认每间隔30秒，与Tracker发送一次心跳包，**在心跳包的回复中，将会有该组内的其他Storage信息**。

3）Storage获取到同组的其他Storage信息之后，为组内的每个其他Storage开启一个线程负责同步。

2 同步线程执行过程storage_sync_thread_entrance

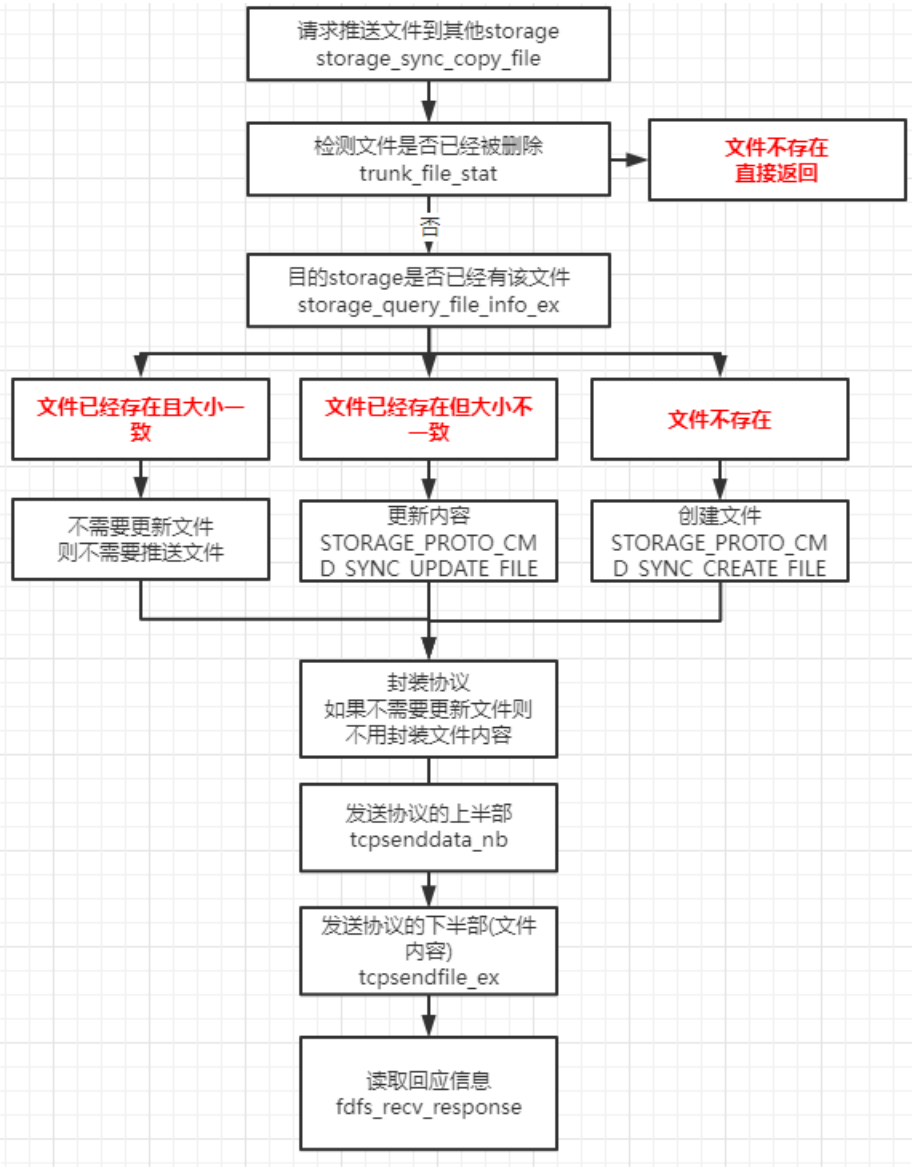
storage_sync_thread_entrance 同步线程

每个同步线程负责到一台Storage的同步，以阻塞方式进行。

1）打开对应Storage的mark文件，如负责到114.215.169.66的同步则打开114.215.169.66_23000.mark文件，从中读取binlog_index、binlog_offset两个字段值，如取到值为：0、100，那么就打开binlog.000文件，seek到100这个位置。

2）进入一个while循环，尝试着读取一行，若读取不到则睡眠等待。若读取到一行，并且该行的操作方式为源操作，如C、A、D、T（大写的都是），则将该行指定的操作同步给对方（非源操作不需要同步），同步成功后更新binlog_offset标志，该值会定期写入到114.215.169.66_23000.mark文件之中。

storage_open_readable_binlog



#0 storage_sync_copy_file (pStorageServer=pStorageServer@entry=0x7faab7898a00, pReader=pReader@entry=0x7faaac000b20, pRecord=pRecord@entry=0x7faab7898ad0, proto_cmd=proto_cmd@entry=16 '\020') at storage_sync.c:92

#1 0x000055c62441dbf9 in storage_sync_data (pRecord=0x7faab7898ad0, pStorageServer=0x7faab7898a00, pReader=0x7faaac000b20)
    at storage_sync.c:1103
#2 storage_sync_thread_entrance (arg=0x55c62465c440 <g_storage_servers>) at storage_sync.c:3177
#3 0x00007faabc7566db in start_thread (arg=0x7faab7899700) at pthread_create.c:463
#4 0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

#0 storage_write_to_mark_file (pReader=pReader@entry=0x7faaac000b20) at storage_sync.c:2417#1 0x000055c62441da9e in storage_sync_thread_entrance (arg=0x55c62465c440 <g_storage_servers>) at storage_sync.c:3129
#2 0x00007faabc7566db in start_thread (arg=0x7faab7899700) at pthread_create.c:463
#3 0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

storage_sync.c　　发送同步文件 **storage_sync_copy_file**
storage_service.c 接收同步文件 **storage_sync_copy_file**

storage_sync_copy_file

- tcpsenddata_nb
- tcpsendfile_ex
- fdfs_recv_response

storage_write_to_mark_file

STORAGE_PROTO_CMD_SYNC_CREATE_FILE 同步新增文件

storage_binlog_write

3 同步前删除

假如同步较为缓慢，那么有可能在开始同步一个文件之前，该文件已经被客户端删除，此时同步线程将打印一条日志，然后直接接着处理后面的Binlog。

6.5 Storage的最后最早被同步时间

这个标题有点拗口，先举个例子：一组内有Storage-A、Storage-B、Storage-C三台机器。对于A这台机器来说，B与C机器都会同步Binlog（包括文件）给他，A在接收同步时会记录每台机器同步给他的最后时间（也就是Binlog中的第一个字段timpstamp）。比如B最后同步给A的Binlog-timestamp为100，C最后同步给A的Binlog-timestamp为200，那么A机器的最后最早被同步时间就为100.

这个值的意义在于，判断一个文件是否存在某个Storage上。比如这里A机器的最后最早被同步时间为100，那么如果一个文件的创建时间为99，就可以肯定这个文件在A上肯定有。为什呢？一个文件会Upload到组内三台机器的任何一台上：

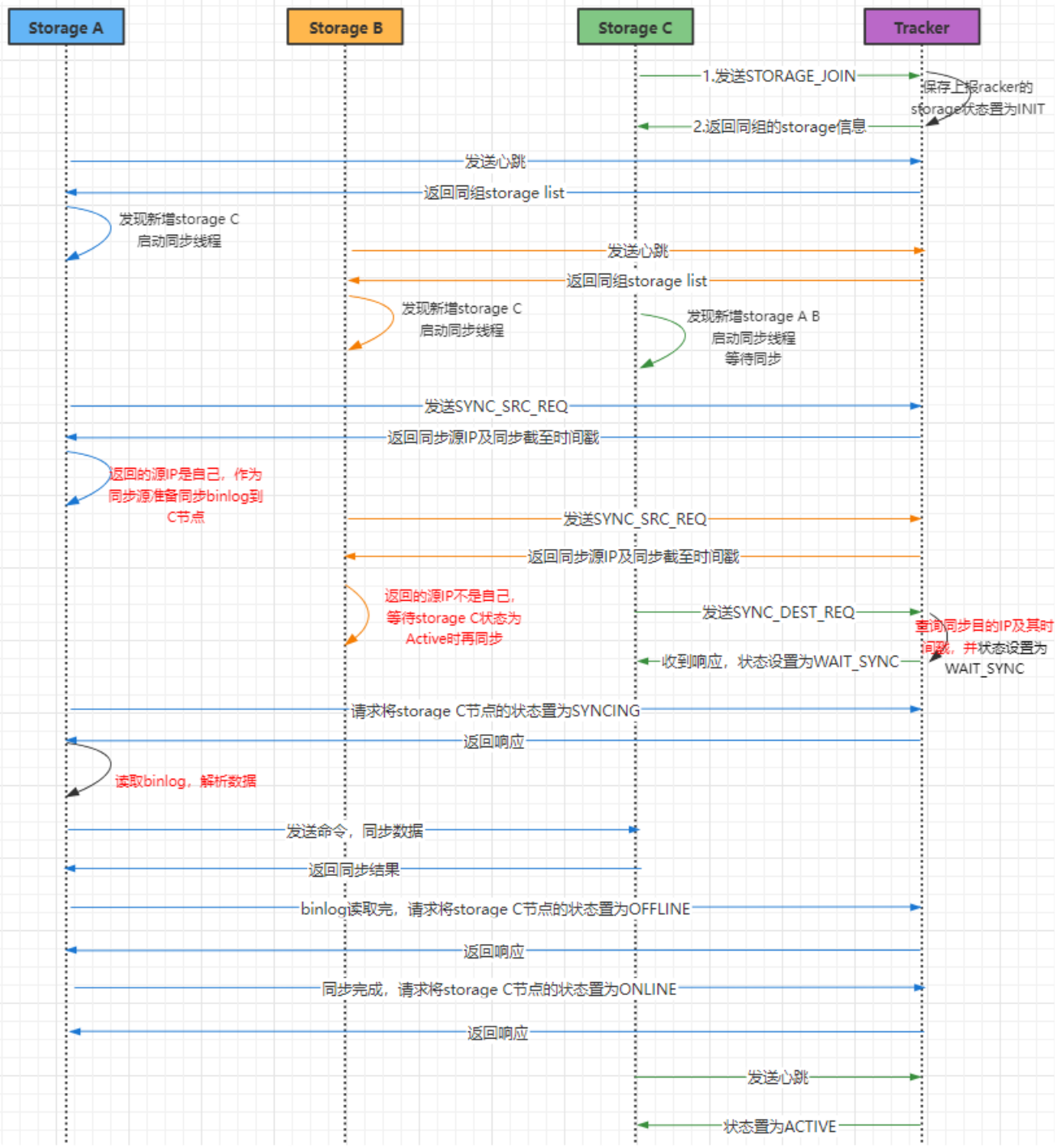
1）若这个文件是直接Upload到A上，那么A肯定有。

2）若这个文件是Upload到B上，由于B同步给A的最后时间为100，也就是说在100之前的文件都已经同步A了，那么A肯定有。

3）同理C也一样。

Storage会定期将每台机器同步给他的最后时间告诉给Tracker，Tracker在客户端要下载一个文件时，需要判断一个Storage是否有该文件，只要解析文件的创建时间，然后与该值作比较，若该值大于创建创建时间，说明该Storage存在这个文件，可以从其下载。

6.6 新增节点同步流程



1、新节点storage C 启动的时候会创建线程tracker_report_thread_entrance，调用tracker_report_join向tracker 发送命令TRACKER_PROTO_CMD_STORAGE_JOIN（81）报告，自己的group名称，ip,端口，版本号，存储目录数，子目录数，启动时间，老数据是否同步完成，当前连接的tracker信息，当前状态信息（FDFS_STORAGE_STATUS_INIT）等信息

2、tracker收到TRACKER_PROTO_CMD_STORAGE_JOIN命令后，将上报的信息和已有（tracker数据文件中保存的信息）的信息进行比较，如果有则更新，没有的话，将节点及状态信息写入缓存和数据文件中，**并查找同group的其他节点做为同步源，如果有返回给stroage C**

3、新节点stroage C 收到tracker响应继续流程。发送TRACKER_PROTO_CMD_STORAGE_SYNC_DEST_REQ （87）查询同步目的

4、tracker收到TRACKER_PROTO_CMD_STORAGE_SYNC_DEST_REQ 请求后， 查找同group的其他节点做为同步目标，及时间戳返回给新storage节点

5、新storage节点收到响应后，保存同步源及同步时间戳。继续流程，发送TRACKER_PROTO_CMD_STORAGE_BEAT（83） 给tracker

6、tracker收到心跳报告后，leader trakcer（非leader不返回数据），把最新的group的storagelist返回给新的stroaged

7、stroage C 收到 tracker storage list后，启动2个同步线程，准备将binlog同步到 节点 A和B（此时还不能同步，因为stroage C 还是WAIT_SYNC 状态）

8、这时候，其他的已在线的storage 节点 A 、B会发送心跳给tracker ，tracker 把会收到最新的stroagelist，A、B、C返回给Storage A，B

9、storage A，B 收到tracker响应后，会发现本地缓存中没有stroage C，会启动binlog同步线程，将数据同步给 stroage C

10、storage A 、B分别启动storage_sync_thread_entrance 同步线程，先向 tracker 发送TRACKER_PROTO_CMD_STORAGE_SYNC_SRC_REQ（86）命令，请求同步源，tracker会把同步源IP及同步时间戳返回

11、stroage A 、B节点的同步线程收到TRACKER_PROTO_CMD_STORAGE_SYNC_SRC_REQ 响应后，会检查返回的同步源IP是否和自己本地ip一致，如果一致置need_sync_old=1表示将做为源数据将老的数据，同步给新节点C，如果不一致置need_sync_old=0，则等待节点C状态为Active时，**再同步（增量同步）**。因为，如果A、 B同时作为同步源，同步数据给C的话，C数据会重复。这里假设节点A，判断tracker返回的是同步源和自己的iP一致，A做为同步源，将数据同步给storage C节点。

12、Storage A同步线程继续同步流程，用同步目的的ip和端口，为文件名，.mark为后缀，如192.168.1.3_23000.mark,将同步信息写入此文件。将Storage C的状态置为FDFS_STORAGE_STATUS_SYNCING 上报给tracker，开始同步

1. **从data/sync目录下**，读取binlog.index 中的，binlog文件Id，binlog.000读取逐行读取，进行解析
   具体格式 如下：
   1490251373 C M02/52/CB/CtAqWVjTbm2AlqTkAAACd_nlZ7M797.jpg
   1490251373 表示时间戳
   C 表示操作类型
   M02/52/CB/CtAqWVjTbm2AlqTkAAACd_nlZ7M797.jpg 文件名
   因为storage C是新增节点，这里需要全部同步给storage C服务
2. **根据操作类型，将数据同步给storage C，具体有如下类型**
   #define STORAGE_OP_TYPE_SOURCE_CREATE_FILE 'C' //upload file
   #define STORAGE_OP_TYPE_SOURCE_APPEND_FILE 'A' //append file
   #define STORAGE_OP_TYPE_SOURCE_DELETE_FILE 'D' //delete file
   #define STORAGE_OP_TYPE_SOURCE_UPDATE_FILE 'U' //for whole file update such as metadata file
   #define STORAGE_OP_TYPE_SOURCE_MODIFY_FILE 'M' //for part modify
   #define STORAGE_OP_TYPE_SOURCE_TRUNCATE_FILE 'T' //truncate file
   #define STORAGE_OP_TYPE_SOURCE_CREATE_LINK 'L' //create symbol link
   #define STORAGE_OP_TYPE_REPLICA_CREATE_FILE 'c'

#define STORAGE_OP_TYPE_REPLICA_APPEND_FILE 'a'
#define STORAGE_OP_TYPE_REPLICA_DELETE_FILE 'd'
#define STORAGE_OP_TYPE_REPLICA_UPDATE_FILE 'u'
#define STORAGE_OP_TYPE_REPLICA_MODIFY_FILE 'm'
#define STORAGE_OP_TYPE_REPLICA_TRUNCATE_FILE 't'
#define STORAGE_OP_TYPE_REPLICA_CREATE_LINK 'l'
具体同步函数storage_sync_data

3. 发送数据给Stroage C，StroageC 收数据并保存
4. binlog文件读完之后，会将Stroage C 状态 置为FDFS_STORAGE_STATUS_OFFLINE，向tracker报告，同时更新同步状态到本地文件mark文件
5. 同步完成后调用 tracker_sync_notify 发送TRACKER_PROTO_CMD_STORAGE_SYNC_NOTIFY通知tracker同步完成，将storage C的 状态置为 FDFS_STORAGE_STATUS_ONLINE
6. **当storage server C向tracker server发起heart beat时(比如间隔30秒)**，tracker server将其状态更改为FDFS_STORAGE_STATUS_ACTIVE。

部分调试记录–仅供参考

FDFS_PROTO_CMD_ACTIVE_TEST storage活性测试

storage_upload_file

#0 storage_upload_file (pTask=pTask@entry=0x7fe7d7081010, bAppenderFile=bAppenderFile@entry=false) at storage_service.c:4547#1 0x00005608b83c6d26 in storage_deal_task (pTask=pTask@entry=0x7fe7d7081010) at storage_service.c:8345
#2 0x00005608b83cdb79 in client_sock_read (sock=21, event=<optimized out>, arg=0x7fe7d7081010) at storage_nio.c:409
#3 0x00007fe7dba87d34 in deal_ioevents (ioevent=0x5608b9df4ed8) at ioevent_loop.c:32
#4 ioevent_loop (pThreadData=pThreadData@entry=0x5608b9df4ed8, recv_notify_callback=<optimized out>,
clean_up_callback=0x5608b83cd7ad <task_finish_clean_up>, continue_flag=0x5608b8609368 <g_continue_flag>)
at ioevent_loop.c:129

#5 0x00005608b83b998a in work_thread_entrance (arg=0x5608b9df4ed8) at storage_service.c:1960
#6 0x00007fe7dbcb86db in start_thread (arg=0x7fe7dc025700) at pthread_create.c:463
#7 0x00007fe7db78e71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

dio_write_file 负责文件的写入

storage_upload_file_done_callback

(gdb) bt#0 storage_upload_file_done_callback (pTask=0x7fe7d7081010, err_no=0) at storage_service.c:1131
#1 0x00005608b83cec30 in dio_write_file (pTask=0x7fe7d7081010) at storage_dio.c:525
#2 0x00005608b83ce091 in dio_thread_entrance (arg=0x5608b9e05588) at storage_dio.c:748
#3 0x00007fe7dbcb86db in start_thread (arg=0x7fe7d6e7c700) at pthread_create.c:463
#4 0x00007fe7db78e71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

storage_recv_notify_read

Thread 4 "fdfs_storaged" hit Breakpoint 6, storage_recv_notify_read (sock=14, event=1, arg=0x7fe7dbfa3e70) at storage_nio.c:131131 if ((bytes=read(sock, &task_addr, sizeof(task_addr))) < 0)
(gdb) bt
#0 storage_recv_notify_read (sock=14, event=1, arg=0x7fe7dbfa3e70) at storage_nio.c:131
#1 0x00007fe7dba87d34 in deal_ioevents (ioevent=0x5608b9df4fc0) at ioevent_loop.c:32
#2 ioevent_loop (pThreadData=pThreadData@entry=0x5608b9df4fc0, recv_notify_callback=<optimized out>,
    clean_up_callback=0x5608b83cd7ad <task_finish_clean_up>, continue_flag=0x5608b8609368 <g_continue_flag>)
    at ioevent_loop.c:129
#3 0x00005608b83b998a in work_thread_entrance (arg=0x5608b9df4fc0) at storage_service.c:1960
#4 0x00007fe7dbcb86db in start_thread (arg=0x7fe7dbfa4700) at pthread_create.c:463
#5 0x00007fe7db78e71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

client_sock_read负责文件数据的读取

读取到数据插入队列
Thread 5 "fdfs_storaged" hit Breakpoint 7, client_sock_read (sock=19, event=<optimized out>, arg=0x7fe7d7101940) at storage_nio.c:328

```
328             bytes = recv(sock, pTask->data + pTask->offset, recv_bytes, 0);
(gdb) bt
#0  client_sock_read (sock=19, event=<optimized out>, arg=0x7fe7d7101940) at
storage_nio.c:328
#1  0x00007fe7dba87d34 in deal_ioevents (ioevent=0x5608b9df50a8) at ioevent_loop.c:32
#2  ioevent_loop (pThreadData=pThreadData@entry=0x5608b9df50a8, recv_notify_callback=
<optimized out>,
    clean_up_callback=0x5608b83cd7ad <task_finish_clean_up>, continue_flag=0x5608b8609368
<g_continue_flag>)
    at ioevent_loop.c:129
#3  0x00005608b83b998a in work_thread_entrance (arg=0x5608b9df50a8) at
storage_service.c:1960
#4  0x00007fe7dbcb86db in start_thread (arg=0x7fe7d7080700) at pthread_create.c:463
#5  0x00007fe7db78e71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
```

引申阅读

fastDFS教程 II –文件服务器迁移 https://www.cnblogs.com/wlandwl/p/fastdfsmove.html
FastDFS教程IV–文件服务器集群搭建 https://www.cnblogs.com/wlandwl/p/fastdfsAdd.html
https://www.cnblogs.com/myitnews/p/12271950.html