3-FastDFS存储原理-课件v1.1

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

本文为 图床项目 第三节课程的课件。

主要内容：

- fastdfs协议解析
- 文件上传原理
- 文件下载原理
- fastdfs网络模型
- 多个tracker和server的搭建

1 源码和客户端程序

fdfs_append_file
fdfs_download_file
fdfs_monitor
fdfs_upload_file
fdfs_upload_appender

├── client │ ├── client_func.c 客户端程序
│ ├── client_func.h
│ ├── client_global.c
│ ├── client_global.h
│ ├── fdfs_appender_test1.c

```
│   ├── fdfs_appender_test.c
│   ├── fdfs_append_file.c
│   ├── fdfs_client.h
│   ├── fdfs_crc32.c
│   ├── fdfs_delete_file.c              // 删除文件
│   ├── fdfs_download_file.c            // 下载文件
│   ├── fdfs_file_info.c                // 查看文件信息
│   ├── fdfs_link_library.sh
│   ├── fdfs_link_library.sh.in
│   ├── fdfs_monitor.c                  // 监控storage
│   ├── fdfs_regenerate_filename.c
│   ├── fdfs_test1.c
│   ├── fdfs_test.c
│   ├── fdfs_upload_appender.c // 断点上传文件
│   ├── fdfs_upload_file.c              // 上传文件
│   ├── Makefile
│   ├── Makefile.in
│   ├── storage_client1.h
│   ├── storage_client.c        // 和storage交互的接口和实现
│   ├── storage_client.h
│   ├── test
│   ├── tracker_client.c   // 和tracker交互的接口和实现
│   └── tracker_client.h
├── common
│   ├── fdfs_define.h
│   ├── fdfs_global.c
│   ├── fdfs_global.h
│   ├── fdfs_http_shared.c
│   ├── fdfs_http_shared.h
│   ├── Makefile
│   ├── mime_file_parser.c
│   └── mime_file_parser.h
├── conf                                配置文件
│   ├── anti-steal.jpg
│   ├── client.conf
│   ├── http.conf
│   ├── mime.types
│   ├── storage.conf
│   ├── storage_ids.conf
│   └── tracker.conf
```

```
├── init.d                    启动脚本
│   ├── fdfs_storaged         脚本，默认执行配置文件，以及执行程序路径
│   └── fdfs_trackerd 单台机器多个tracker不要用这个脚本了
├── make.sh
├── README_zh.md
├── restart.sh
├── setup.sh
├── stop.sh
├── storage   服务器
│   ├── fdfs_storaged.c  main函数入口
│   ├── fdht_client
│   ├── Makefile
│   ├── Makefile.in
│   ├── storage_dio.c       文件data io操作
│   ├── storage_dio.h
│   ├── storage_disk_recovery.c 磁盘恢复
│   ├── storage_disk_recovery.h
│   ├── storage_dump.c
│   ├── storage_dump.h
│   ├── storage_func.c
│   ├── storage_func.h
│   ├── storage_global.c
│   ├── storage_global.h
│   ├── storage_ip_changed_dealer.c
│   ├── storage_ip_changed_dealer.h
│   ├── storage_nio.c         网络io相关 network io
│   ├── storage_nio.h
│   ├── storage_param_getter.c
│   ├── storage_param_getter.h
│   ├── storage_service.c       storage服务核心，各种信令的处理响应， 核心：
storage_deal_task
│   ├── storage_service.h
│   ├── storage_sync.c    文件同步
│   ├── storage_sync_func.c
│   ├── storage_sync_func.h
│   ├── storage_sync.h
│   ├── tracker_client_thread.c
│   ├── tracker_client_thread.h
│   └── trunk_mgr
├── test                  测试文件
```

```
│   ├── combine_result.c
│   ├── common_func.c
│   ├── common_func.h
│   ├── dfs_func.c
│   ├── dfs_func.h
│   ├── dfs_func_pc.c
│   ├── gen_files.c
│   ├── Makefile
│   ├── test_delete.c      性能测试删除文件
│   ├── test_delete.sh
│   ├── test_download.c 性能测试下载文件
│   ├── test_download.sh
│   ├── test_types.h
│   ├── test_upload.c          性能测试上传文件
│   └── test_upload.sh
└── tracker 访问
    ├── fdfs_server_id_func.c
    ├── fdfs_server_id_func.h
    ├── fdfs_shared_func.c
    ├── fdfs_shared_func.h
    ├── fdfs_trackerd.c   main函数入口
    ├── Makefile
    ├── Makefile.in
    ├── tracker_dump.c
    ├── tracker_dump.h
    ├── tracker_func.c
    ├── tracker_func.h
    ├── tracker_global.c
    ├── tracker_global.h
    ├── tracker_http_check.c
    ├── tracker_http_check.h
    ├── tracker_mem.c
    ├── tracker_mem.h   storage上报的信息在内存也存储
    ├── tracker_nio.c
    ├── tracker_nio.h       tracker网络io相关
    ├── tracker_proto.c
    ├── tracker_proto.h      fastdfs协议相关，包括storage的交互信令
    ├── tracker_relationship.c
    ├── tracker_relationship.h
    ├── tracker_service.c   tracker 服务核心  tracker_deal_task
```

```
├── tracker_service.h
├── tracker_status.c
├── tracker_status.h
└── tracker_types.h
```

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。

debug举例

2.1 debug范例-文件上传

```
ot@iZbp1h2l856zgoegc8rvnhZ:~/tuchuang/0voice_tuchuang# ll 0voice_tuchuang.sql
w-rw-r-- 1 1002 1002 4034 Feb 24 13:50 0voice_tuchuang.sql
```

1. 进入0voice_tuchuang目录： cd tuchuang/0voice_tuchuang
2. gdb启动上传文件程序：gdb /usr/bin/fdfs_upload_file
3. 设置参数：set args /etc/fdfs/client.conf ./0voice_tuchuang.sql
4. 在main函数打断点：b main
5. 在上传函数断点：b storage_do_upload_file

3 协议格式

3.0 协议

FastDFS采用二进制TCP通信协议。一个数据包由 包头（header）和包体（body）组成。包头只有10个字节，格式如下：
@ pkg_len：8字节整数，body长度，不包含header，只是body的长度
@ cmd：1字节整数，命令码
@ status：1字节整数，状态码，0表示成功，非0失败（UNIX错误码）

Bash 复制代码

```
tracker\tracker_proto.h TrackerHeader

typede fstruct
{
    char pkg_len[FDFS_PROTO_PKG_LEN_SIZE];  // body长度，不包括header
    char cmd;    //command 命令
    char status; //status code for response 响应的状态码
} TrackerHeader;

即是头部固定10字节，body长度通过pkg_len给出。

即是一帧完整的协议为 TrackerHeader + body数据（可以为0）.
```

数据包中的类型说明：
1） 整数类型采用网络字节序（Big-Endian），包括4字节整数和8字节整数；
2） 1字节整数不存在字节序问题，在Java中直接映射为byte类型，C/C++中为char类型；
3） 固定长度的字符串类型以 ASCII码0结尾，对于Java等语言需要调用trim处理返回的字符串。变长字符串的长度可以直接拿到或者根据包长度计算出来，不以ASCII 0结尾。
下面将列举client发送给FastDFS server的命令码及其body（包体）结构。

公共命令码

* FDFS_PROTO_CMD_ACTIVE_TEST：激活测试，通常用于检测连接是否有效。客户端使用连接池的情况下，建立连接后发送一次active test即可和server端保持长连接。

请求body：无
响应body：无

发送给tracker server的命令码

*** TRACKER_PROTO_CMD_SERVER_LIST_ONE_GROUP：查看一个group状态**

请求body：
@group_name：16字节字符串，组名
响应body：
@group_name：17字节字符串
@total_mb：8字节整数，磁盘空间总量，单位MB
@free_mb：8字节整数，磁盘剩余空间，单位MB
@trunk_free_mb：8字节整数，trunk文件剩余空间，单位MB（合并存储开启时有效）
@server_count：8字节整数，storage server数量
@storage_port：8字节整数，storage server端口号
@storage_http_port：8字节整数，storage server上的HTTP端口号
@active_count：8字节整数，当前活着的storage server数量
@current_write_server：8字节整数，当前写入的storage server顺序号
@store_path_count：8字节整数，storage server存储路径数
@subdir_count_per_path：8字节整数，存储路径下的子目录数（FastDFS采用两级子目录），如 256
@current_trunk_file_id：8字节整数，当前使用的trunk文件ID（合并存储开启时有效）

*** TRACKER_PROTO_CMD_SERVER_LIST_ALL_GROUPS：列举所有group**

#请求body：无
#响应body：n 个group实体信息，n >= 0。每个group的数据结构参见上面的
TRACKER_PROTO_CMD_SERVER_LIST_ONE_GROUP。

*** TRACKER_PROTO_CMD_SERVER_LIST_STORAGE：列举一个group下的storage server**

请求body：@group_name：16字节字符串，组名
@server_id：不定长，最大长度为15字节，storage server id，可选参数
响应body： n 个storage server实体信息，n >= 0。 每个storage实体结构如下：
[//列表开始
@status：1字节整数，storage server状态
@id：16字节字符串，server ID
@ip_addr：16字节字符串，IP地址
@domain_name：128字节字符串，域名
@src_storage_id：16字节字符串，同步源storage的server ID
@version：6字节字符串，运行的FastDFS版本号，例如6.04
@join_time: 8字节整数，加入集群时间
@up_time: 8字节整数，fdfs_storaged启动时间

@total_mb: 8字节整数，磁盘空间总量，单位MB
@free_mb: 8字节整数，磁盘剩余空间，单位MB
@upload_priority: 8字节整数，上传文件优先级
@store_path_count: 8字节整数，存储路径数
@subdir_count_per_path: 8字节整数，存储路径下的子目录数（FastDFS采用两级子目录），如 256
@current_write_path: 8字节整数，当前写入的存储路径（顺序号）
@storage_port: 8字节整数，storage server服务端口号
@storage_http_port: 8字节整数，HTTP服务端口号
@alloc_count：4字节整数，已分配的连接buffer数目
@current_count：4字节整数，当前连接数
@max_count：4字节整数，曾经达到过的最大连接数
@total_upload_count：8字节整数，上传文件总数
@success_upload_count：8字节整数，成功上传文件数
@total_append_count：8字节整数，调用append总次数
@success_append_count：8字节整数，成功调用append次数
@total_modify_count：8字节整数，调用modify总次数
@success_modify_count：8字节整数，成功调用modify次数
@total_truncate_count：8字节整数，调用truncate总次数
@success_truncate_count：8字节整数，成功调用truncate次数
@total_set_meta_count：8字节整数，设置文件附加属性（meta data）总次数
@success_set_meta_count：8字节整数，成功设置文件附加属性（meta data）次数
@total_delete_count：8字节整数，删除文件总数
@success_delete_count：8字节整数，成功删除文件数
@total_download_count：8字节整数，下载文件总数
@success_download_count：8字节整数，成功下载文件数
@total_get_meta_count：8字节整数，获取文件附加属性（meta data）总次数
@success_get_meta_count：8字节整数，成功获取文件附加属性（meta data）次数
@total_create_link_count：8字节整数，创建文件符号链接总数
@success_create_link_count：8字节整数，成功创建文件符号链接数
@total_delete_link_count：8字节整数，删除文件符号链接总数
@success_delete_link_count：8字节整数，成功删除文件符号链接数
@total_upload_bytes：8字节整数，上传文件总字节数
@success_upload_bytes：8字节整数，成功上传文件字节数
@total_append_bytes：8字节整数，append总字节数
@success_append_bytes：8字节整数，成功append字节数
@total_modify_bytes：8字节整数，modify总字节数
@success_modify_bytes：8字节整数，成功modify字节数
@total_download_bytes：8字节整数，下载总字节数
@success_download_bytes：8字节整数，成功下载字节数
@total_sync_in_bytes：8字节整数，文件同步流入总字节数

@success_sync_in_bytes：8字节整数，文件同步成功流入字节数
@total_sync_out_bytes：8字节整数，文件同步流出总字节数
@success_sync_out_bytes：8字节整数，文件同步成功流出字节数
@total_file_open_count：8字节整数，文件打开总次数
@success_file_open_count：8字节整数，文件成功打开次数
@total_file_read_count：8字节整数，文件读总次数
@success_file_read_count：8字节整数，文件成功读次数
@total_file_write_count：8字节整数，文件写总次数
@success_file_write_count：8字节整数，文件成功写次数
@last_source_update：8字节整数，最近一次源头更新时间
@last_sync_update：8字节整数，最近一次同步更新时间
@last_synced_timestamp：8字节整数，最近一次被同步到的时间戳
@last_heart_beat_time：8字节整数，最近一次心跳时间
] //列表结束

*** TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ONE：获取一个storage server用来存储文件（不指定group name）**
# 请求body：无
# 响应body：
@group_name：16字节字符串，组名@ip_addr：15字节字符串， storage server IP地址
@port：8字节整数，storage server端口号
@store_path_index：1字节整数，基于0的存储路径顺序号

*** TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITH_GROUP_ONE：获取一个storage server用来存储文件（指定组名）**
# 请求body：
@group_name：16字节字符串，组名
响应body：@ip_addr：15字节字符串， storage server IP地址
@port：8字节整数，storage server端口号
@store_path_index：1字节整数，基于0的存储路径顺序号

***TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ALL ： 获 取 storage server列表用来存储文件（不指定组名）**
请求body：无
响应body：
@group_name：16字节字符串，组名
[//列表开始
@ip_addr：15字节字符串， storage server IP地址
@port：8字节整数，storage server端口号
] //列表结束

@store_path_index：1字节整数，基于0的存储路径顺序号

***TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITH_GROUP_ALL：获取storage server列表用来存储文件（指定组名）**

请求body：
@group_name：16字节字符串，组名
响应body：
@group_name：16字节字符串，组名
[//列表开始
@ip_addr：15字节字符串， storage server IP地址
@port：8字节整数，storage server端口号
] //列表结束
@store_path_index：1字节整数，基于0的存储路径顺序号

*** TRACKER_PROTO_CMD_SERVICE_QUERY_FETCH_ONE：获取一个storage server用来下载文件**
*** TRACKER_PROTO_CMD_SERVICE_QUERY_UPDATE：获取storage server列表用来修改文件或文件附加信息**

请求body：
@group_name：16字节字符串，组名
@filename：不定长字符串，文件名
响应body：
@group_name：16字节字符串，组名
@ip_addr：15字节字符串， storage server IP地址
@port：8字节整数，storage server端口号

*** TRACKER_PROTO_CMD_SERVICE_QUERY_FETCH_ALL：获取storage server列表用来下载文件**

请求body：
@group_name：16字节字符串，组名
@filename：不定长字符串，文件名
响应body：
@group_name：16字节字符串，组名
@ip_addr：15字节字符串， 第一个storage server IP地址
@port：8字节整数，storage server端口号
[//列表开始
@ip_addr：15字节字符串，其他storage server IP地址
] //列表结束

发送给storage server的命令码

* STORAGE_PROTO_CMD_UPLOAD_FILE：上传普通文件
* STORAGE_PROTO_CMD_UPLOAD_APPENDER_FILE：上传appender类型文件
# 请求body：
@store_path_index：1字节整数，基于0的存储路径顺序号
@meta_data_length：8字节整数，meta data（文件附加属性）长度，可以为0@file_size：8字节整数，文件大小
@file_ext_name：6字节字符串，不包括小数点的文件扩展名，例如 jpeg、tar.gz
@meta_data： meta_data_length字节字符串，文件附加属性，每个属性用字符 \x01分隔，名称key和取值value之间用字符 \x02分隔@file content：file_size字节二进制内容，文件内容 # 响应body：
@group_name：16字节字符串，组名 @ filename：不定长字符串，文件名

* STORAGE_PROTO_CMD_UPLOAD_SLAVE_FILE：上传slave文件
# 请求body：
@master_filename_length：8字节整数，主文件名长度
@meta_data_length：8字节整数，meta data（文件附加属性）长度，可以为0@file_size：8字节整数，文件大小
@filename_prefix：16字节字符串， 从文件前缀名
@file_ext_name：6字节字符串，不包括小数点的文件扩展名，例如 jpeg、tar.gz
@master_filename：master_filename_length字节字符串，主文件名
@meta_data：meta_data_length字节字符串，文件附加属性，每个属性记录用字符 \x01分隔，名称key和取值value之间用字符 \x02分隔@file content：file_size字节二进制内容，文件内容 # 响应body：@group_name：16字节字符串，组名@ filename：不定长字符串，文件名

* STORAGE_PROTO_CMD_DELETE_FILE：删除文件
# 请求body：
@group_name：16字节字符串，组名
@filename：不定长字符串，文件名# 响应body：无

* STORAGE_PROTO_CMD_SET_METADATA：设置meta data（文件附加属性）
# 请求body：
@filename_length：8字节整数，文件名长度
@meta_data_length：8字节整数，meta data（文件附加属性）长度，可以为0
@op_flag：1字节字符，操作标记，取值说明如下：
'O' – 覆盖方式，覆盖原有meta data'M' – merge方式，和原有meta data合并到一起，已存在的属性将被覆盖@group_name：16字节字符串，组名
@filename：filename_length字节的字符串，文件名@meta_data：meta_data_length字节字符串，文件附加属性，每个属性记录用字符 \x01分隔，名称key和取值value之间用字符 \x02分隔
响应body：无

* STORAGE_PROTO_CMD_DOWNLOAD_FILE：下载文件# 请求body：

@file_offset：8字节整数，文件偏移量
@download_bytes：8字节整数，下载字节数
@group name：16字节字符串，组名 @filename：不定长字符串，文件名
响应body：@file_content：不定长二进制内容，文件内容

*** STORAGE_PROTO_CMD_GET_METADATA：获取meta data（文件附加属性）**
# 请求body：
@group name：16字节字符串，组名
@filename：不定长字符串，文件名
响应body：
@meta_data：不定长字符串，文件附加属性，每个属性记录用字符 \x01分隔，名称key和取值value之间用字符 \x02分隔

*** STORAGE_PROTO_CMD_QUERY_FILE_INFO：获取文件信息**
# 请求body：@group name：16字节字符串，组名@filename：不定长字符串，文件名# 响应body：
@file_size：8字节整数，文件大小
@create_timestamp：8字节整数，文件创建时间（Unix时间戳）
@crc32：8字节整数，文件内容CRC32校验码
@source_ip_addr：16字节字符串，源storage server IP地址

*** STORAGE_PROTO_CMD_APPEND_FILE：追加文件内容**
# 请求body：
@appender_filename_length：8字节整数，appender文件名长度
@file_size：8字节整数，文件大小
@appender_filename：appender_filename_length字节数字符串，appender文件名
@file_content：file_size字节数的二进制内容，追加的文件内容
响应body：无

*** STORAGE_PROTO_CMD_MODIFY_FILE：修改文件内容**
# 请求body：@appender_filename_length：8字节整数，appender文件名长度
@file_offset：8字节整数，文件偏移量
@file_size：8字节整数，文件大小
@appender_filename：appender_filename_length字节数字符串，appender文件名
@file_content：file_size字节数的二进制内容，新（目标）文件内容
响应body：无

*** STORAGE_PROTO_CMD_TRUNCATE_FILE：改变文件大小**
# 请求body：@appender_filename_length：8字节整数，appender文件名长度
@truncated_file_size：8字节整数，truncate后的文件大小
@appender_filename：appender_filename_length字节数字符串，appender文件名

响应body：无

*** STORAGE_PROTO_CMD_REGENERATE_APPENDER_FILENAME：appender类型文件改名为普通文件**

请求body：
@appender_filename：不定长字符串，appender文件名
响应body：
@group name：16字节字符串，组名 @filename：不定长字符串，重新生成的文件名（普通类型）

命令码列表 （代号及取值）

TRACKER_PROTO_CMD_RESP100 //tracker 响应码
STORAGE_PROTO_CMD_RESP 100 //storage 响应码

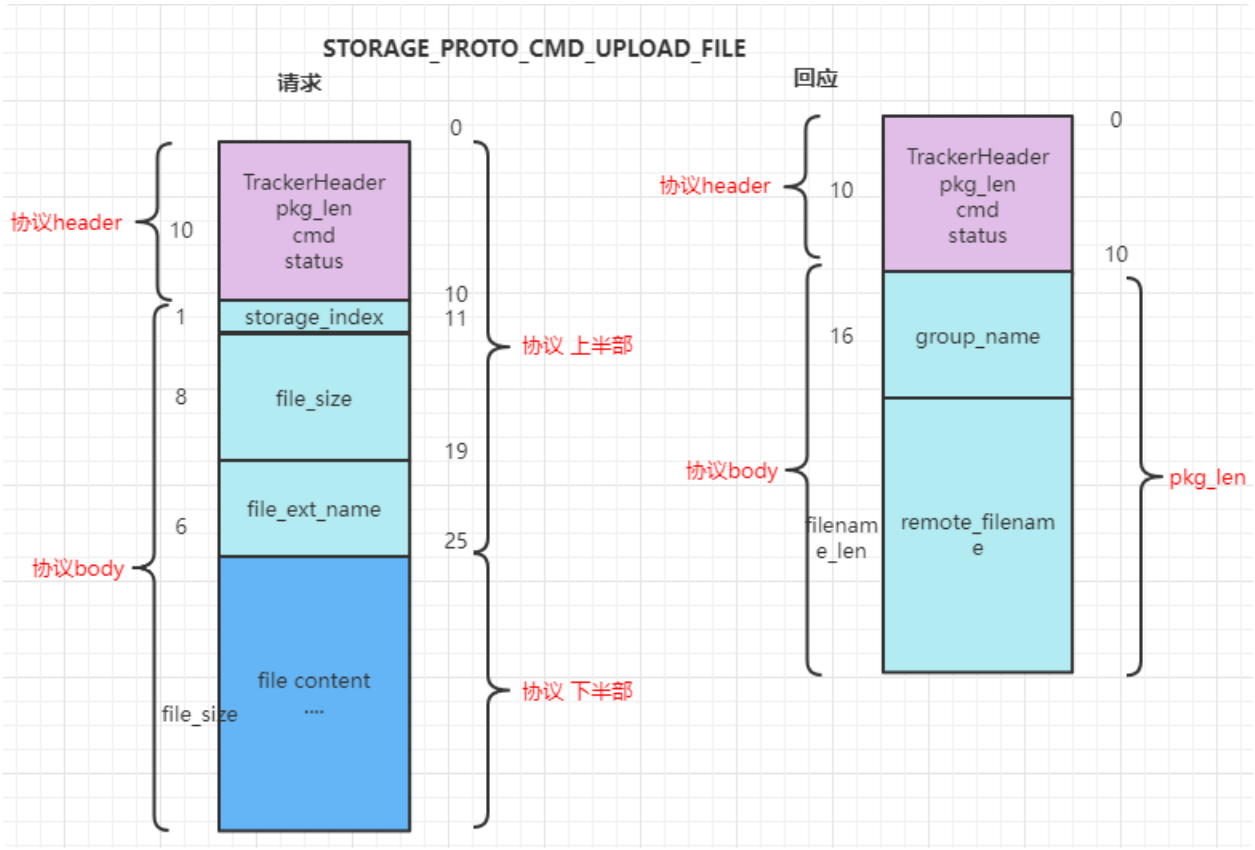
FDFS_PROTO_CMD_ACTIVE_TEST 111
TRACKER_PROTO_CMD_SERVER_LIST_ONE_GROUP90
TRACKER_PROTO_CMD_SERVER_LIST_ALL_GROUPS91
TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ONE 101
TRACKER_PROTO_CMD_SERVICE_QUERY_FETCH_ONE102
TRACKER_PROTO_CMD_SERVICE_QUERY_UPDATE103
TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITH_GROUP_ONE104
TRACKER_PROTO_CMD_SERVICE_QUERY_FETCH_ALL105
TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ALL 106
TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITH_GROUP_ALL107
STORAGE_PROTO_CMD_UPLOAD_FILE11
STORAGE_PROTO_CMD_DELETE_FILE12
STORAGE_PROTO_CMD_SET_METADATA13
STORAGE_PROTO_CMD_DOWNLOAD_FILE14
STORAGE_PROTO_CMD_GET_METADATA15
STORAGE_PROTO_CMD_UPLOAD_SLAVE_FILE 21
STORAGE_PROTO_CMD_QUERY_FILE_INFO22
STORAGE_PROTO_CMD_UPLOAD_APPENDER_FILE23
STORAGE_PROTO_CMD_APPEND_FILE24
STORAGE_PROTO_CMD_MODIFY_FILE34
STORAGE_PROTO_CMD_TRUNCATE_FILE36
STORAGE_PROTO_CMD_REGENERATE_APPENDER_FILENAME 38

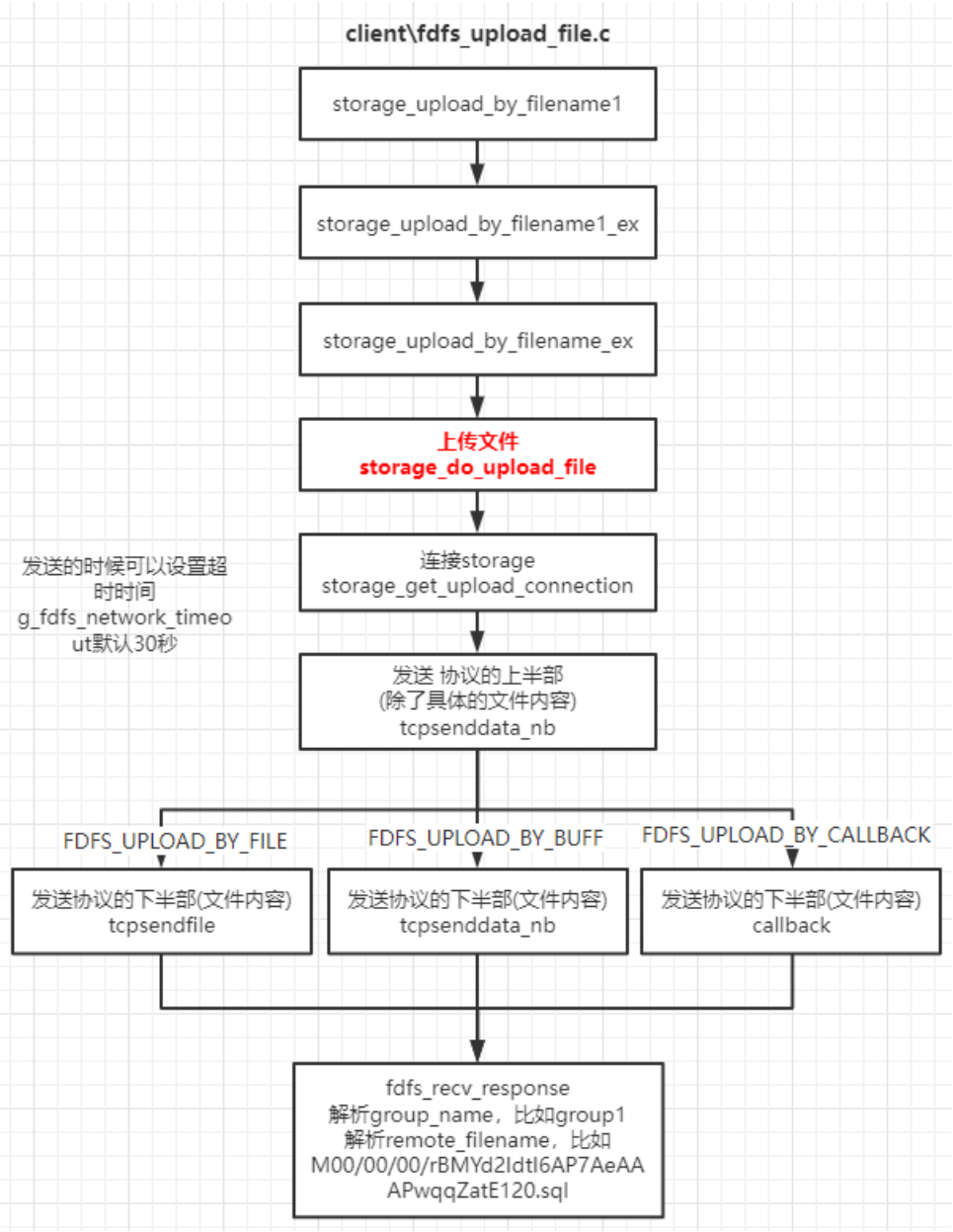
以上传和下载文件为例。

3.1 文件上传

client\fdfs_test1.c

1. FDFS_UPLOAD_BY_FILE： storage_upload_by_filename1
2. FDFS_UPLOAD_BY_BUFF：storage_upload_by_filebuff1
3. FDFS_UPLOAD_BY_CALLBACK： storage_upload_by_callback1




client\fdfs_upload_file.c
storage_upload_by_filename1
storage_upload_by_filename1_ex
storage_upload_by_filename_ex
上传文件
storage_do_upload_file
连接storage
storage_get_upload_connection
发送的时候可以设置超时时间
g_fdfs_network_timeout默认30秒
发送 协议的上半部
(除了具体的文件内容)
tcpsenddata_nb
FDFS_UPLOAD_BY_FILE
FDFS_UPLOAD_BY_BUFF
FDFS_UPLOAD_BY_CALLBACK
发送协议的下半部(文件内容)
tcpsendfile
发送协议的下半部(文件内容)
tcpsenddata_nb
发送协议的下半部(文件内容)
callback
fdfs_recv_response
解析group_name，比如group1
解析remote_filename，比如
M00/00/00/rBMYd2IdtI6AP7AeAA
APwqqZatE120.sql

文件上传函数层次接口

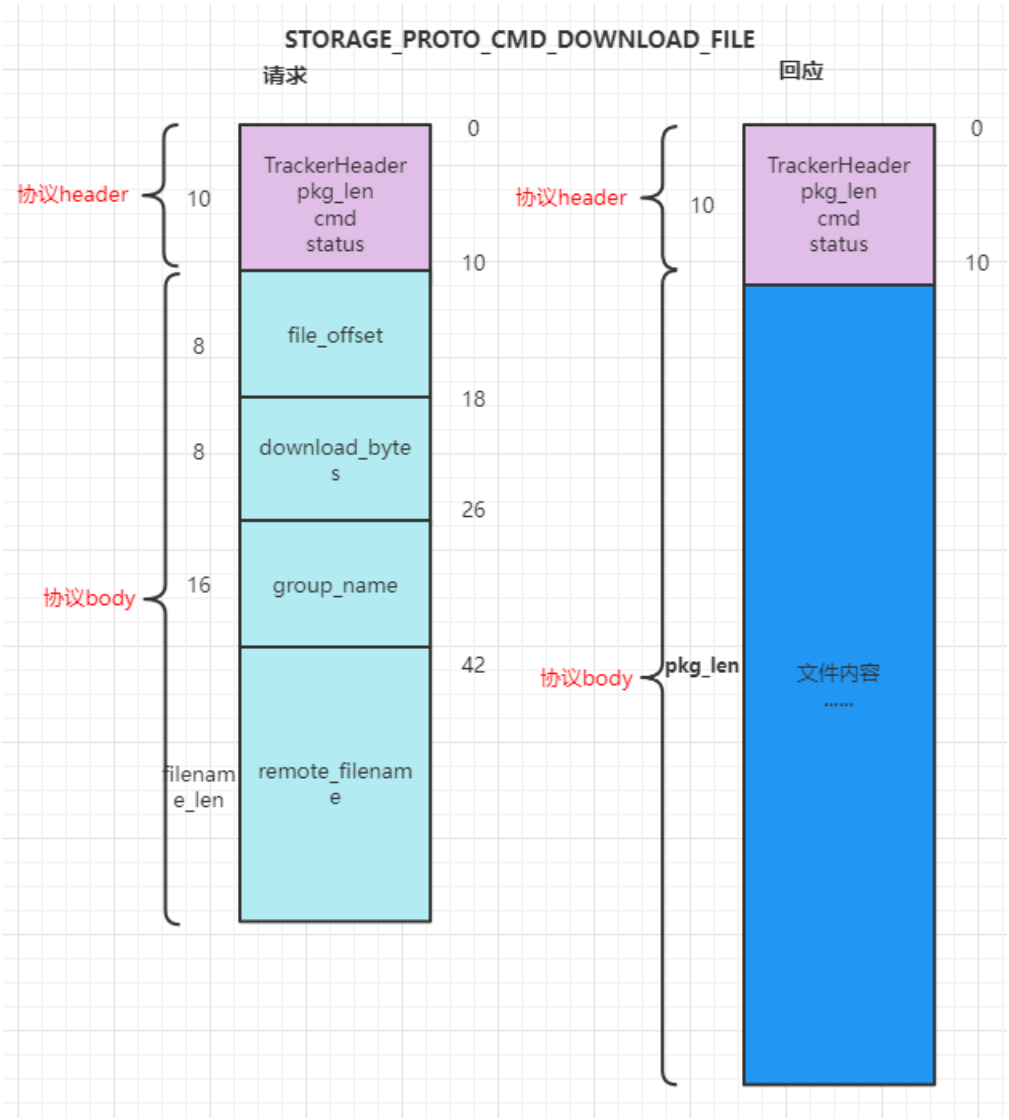
Bash | 复制代码

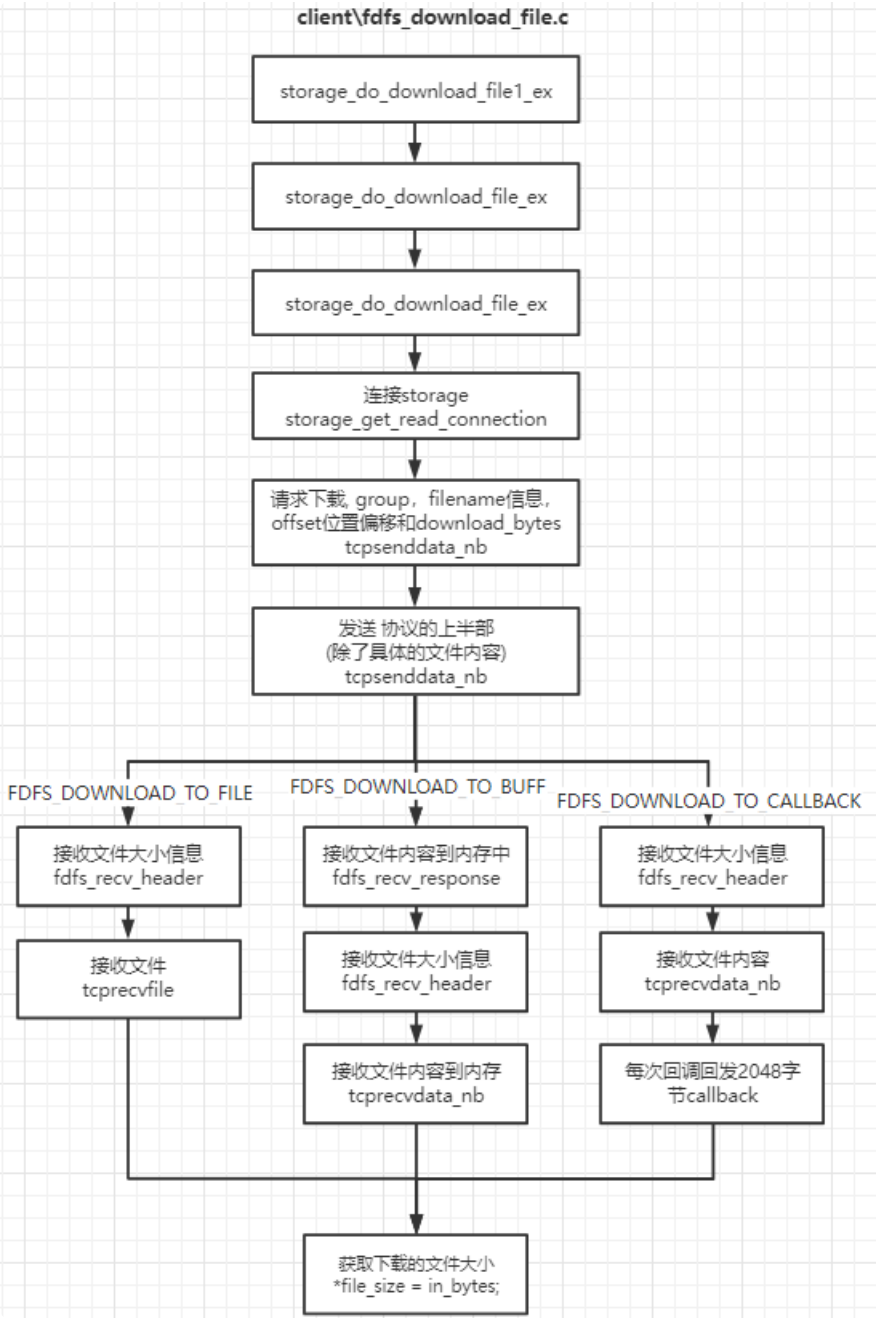
```
#define storage_upload_by_filename1(pTrackerServer,  // tracker server的地址信息
    pStorageServer,  // storage server的地址信息
    store_path_index, // 具体path,比如0、1 对应配置文件的store_path0、store_path1
    local_filename,   // 要上传的文件名
    file_ext_name,  // 文件类型，比如txt、jpg
    meta_list,  //比如 "width"  "160" 参考：client\fdfs_test1.c 201行
    meta_count,   // meta_list列表的元素个数
    group_name, // 所属的group名
    file_id)  // 上传成功后返回file_id
实际上传调用的函数
storage_upload_by_filename1_ex(pTrackerServer, pStorageServer, \
    store_path_index,
    STORAGE_PROTO_CMD_UPLOAD_FILE, \ // 服务器响应的命令
    local_filename, file_ext_name, meta_list, meta_count, \
    group_name, file_id)

继续往下调用

int storage_upload_by_filename_ex(ConnectionInfo *pTrackerServer, \
    ConnectionInfo *pStorageServer, const int store_path_index, \
    const char cmd, const char *local_filename, \
    const char *file_ext_name, const FDFSMetaData *meta_list, \
    const int meta_count, char *group_name, char *remote_filename)


 在继续
storage_do_upload_file(pTrackerServer, pStorageServer, \
  store_path_index, cmd, \
  FDFS_UPLOAD_BY_FILE, // 以文件的方式上传内容，还有FDFS_UPLOAD_BY_BUFF以内存的载体方式上传
  local_filename, \ // 作为文件名或者 内存数据的起始地址
  NULL,   // arg
  stat_buf.st_size, // file_size
  NULL, // master_filename
  NULL, // prefix_name
  file_ext_name, \ // file_ext_name
  meta_list,
  meta_count, group_name,
  remote_filename   // file_id
  )

```

3.2 文件下载

client\fdfs_test1.c client\fdfs_download_file.c

- FDFS_DOWNLOAD_TO_FILE：storage_do_download_file1_ex
- FDFS_DOWNLOAD_TO_BUFF：storage_download_file1
- FDFS_DOWNLOAD_TO_CALLBACK：storage_download_file_ex





3.3 断点续传

先命令操作

```bash
echo hello > test1.txt
echo world > test2.txt

fdfs_upload_appender /etc/fdfs/client.conf test1.txt
得到：group1/M00/00/00/rBMYd2Id2FmEay2-AAAAADY6MCA286.txt ，在fdfs_append_file的时候需要

fdfs_append_file /etc/fdfs/client.conf group1/M00/00/00/rBMYd2Id2FmEay2-AAAAADY6MCA286.txt test2.txt

在服务器相应的目录下查找对应的文件，用cat读取文件内容。
root@iZbp1h2l856zgoegc8rvnhZ:/home/fastdfs/storage/data/00/00# cat rBMYd2Id2FmEay2-AAAAADY6MCA286.txt
hello
world
```

断点续传文件分为两个阶段：

1. fdfs_upload_appender 上传第一部分文件，以STORAGE_PROTO_CMD_UPLOAD_APPENDER_FILE命令
2. **fdfs_append_file 上传其他部分的文件，以STORAGE_PROTO_CMD_APPEND_FILE命令。**

需要注意：

- 注意断点续传的顺序性
- 支持断点续传，但fastdfs并不支持多线程分片上传同一个文件。

4 网络IO模型

对应配置文件

Bash 复制代码

```
# accept thread count
# default value is 1 which is recommended
# since V4.07
accept_threads = 1

# work thread count
# work threads to deal network io
# default value is 4
# since V2.00
work_threads = 4
# if disk read / write separated
##  false for mixed read and write
##  true for separated read and write
# default value is true
# since V2.00
disk_rw_separated = true

# disk reader thread count per store path
# for mixed read / write, this parameter can be 0
# default value is 1
# since V2.00
disk_reader_threads = 1

# disk writer thread count per store path
# for mixed read / write, this parameter can be 0
# default value is 1
# since V2.00
disk_writer_threads = 1

```

线程初始化

默认：g_accept_threads 1
g_work_threads 4
g_disk_reader_threads默认1
g_disk_writer_threads 1

文件操作线程： 根据读写线程数量，path数量确定 文件操作线程数量

```
threads_count_per_path = g_disk_reader_threads + g_disk_writer_threads;
context_count = threads_count_per_path * g_fdfs_store_paths.count;
```

文件名	作用	初始化函数	工作线程
storage\storage_service.c	网络IO任务处理	storage_service_init	work_thread_entrance
storage\storage_service.c	accept线程	storage_accept_loop	accept_thread_entrance
storage\storage_dio.c	文件操作相关初始化	storage_dio_init	dio_thread_entrance

队列：
storage_dio_queue 文件处理队列
task_queue 任务对象池队列

管道
thread_data.pipe_fds[2]：IO工作线程的触发
storage_nio_notify 触发取读取网络io数据

accept_thread_entrance：

- 本身是一个线程
- 调用accept获取新的连接 incomesock = accept(server_sock, (struct sockaddr*)&inaddr, \
  &sockaddr_len);
- 获取对方ip地址：client_addr = getPeerIpaddr(incomesock, \ szClientIp, IP_ADDRESS_SIZE);
- 从对象池取一个task对象：pTask = free_queue_pop();
- task对象里面有client的封装信息，需要设置：pClientInfo = (StorageClientInfo *)pTask->arg;
  - pTask->event.fd = incomesock;
  - pClientInfo->stage = FDFS_STORAGE_STAGE_NIO_INIT;
  - 轮询线程：pClientInfo->nio_thread_index = pTask->event.fd % g_work_threads;
  - 通知io线程有新的连接：write(pThreadData->thread_data.pipe_fds[1], &task_addr, \
    sizeof(task_addr)) != sizeof(task_addr)

work_thread_entrance：

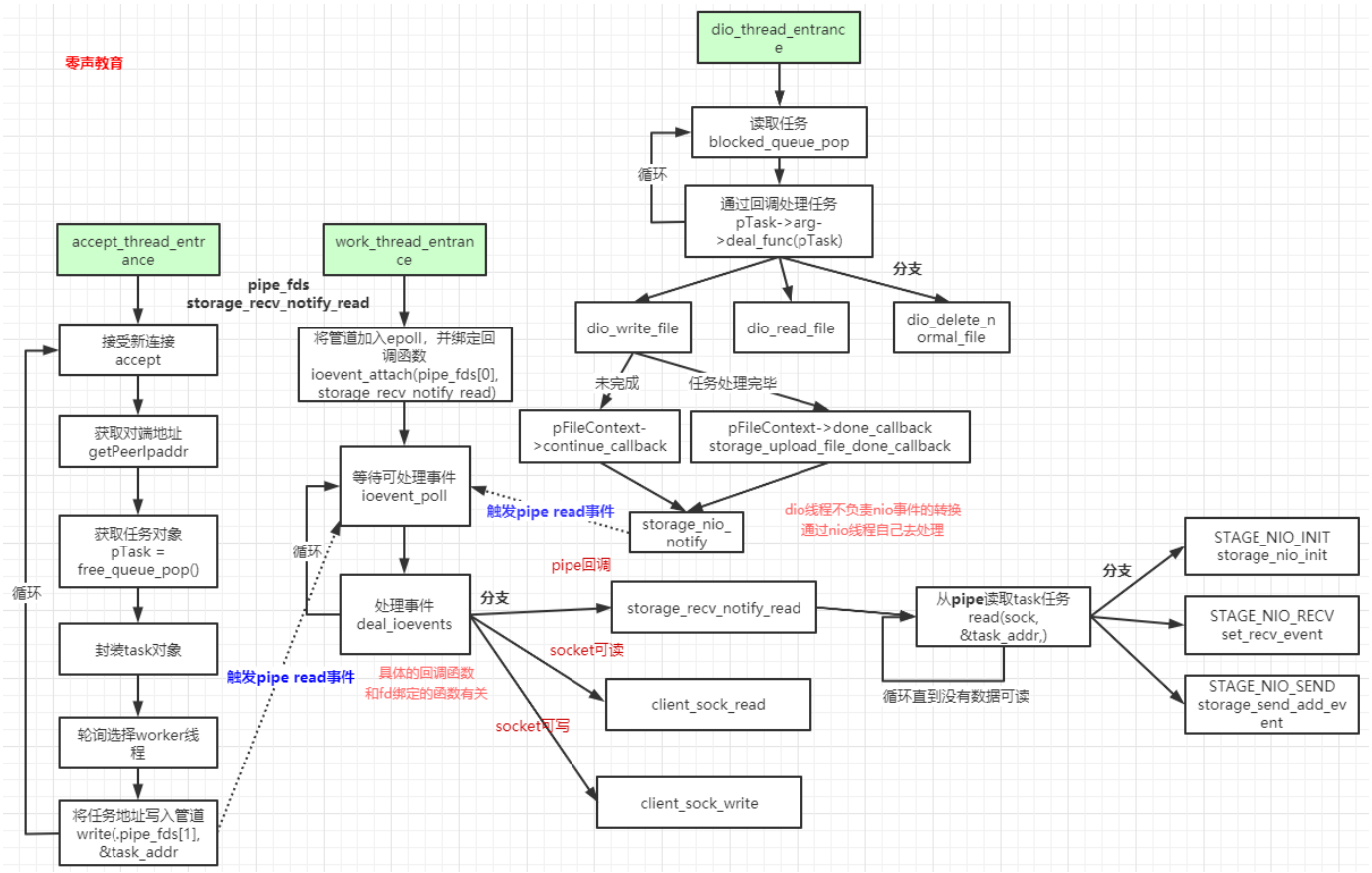
- 本身是一个线程
- 核心调用 ioevent_loop(&pThreadData->thread_data, storage_recv_notify_read, task_finish_clean_up, &g_continue_flag);
  - storage_recv_notify_read 当数据可读时触发
  -
- 进入到ioevent_loop核心：
  - 实际是调用epoll_wait：pThreadData->ev_puller.iterator.count = ioevent_poll(&pThreadData->ev_puller)；返回可处理事件
  - 循环处理事件：deal_ioevents
    - 调用storage_recv_notify_read

文件IO线程dio_thread_entrance：

- 本身是一个线程
- 任务由storage_dio_queue_push 投递
- 从blocked_queue_pop读取任务
- 通过回调写入文件dio_write_file 或者 读取文件dio_read_file

(gdb) b storage_dio.c:748Breakpoint 1 at 0x55c62441f084: file storage_dio.c, line 748.
(gdb) b client_sock_read
Breakpoint 2 at 0x55c62441e852: file storage_nio.c, line 245.
(gdb) b client_sock_write
Breakpoint 3 at 0x55c62441eb80: file storage_nio.c, line 434.
(gdb) b storage_recv_notify_read
Breakpoint 4 at 0x55c62441ee7f: file storage_nio.c, line 121.
(gdb) b dio_write_file
Breakpoint 5 at 0x55c62441fa46: file storage_dio.c, line 405.
(gdb) b storage_nio_notify
Breakpoint 6 at 0x55c62440adf1: file storage_service.c, line 1918.
(gdb) b storage_upload_file_done_callback
Breakpoint 7 at 0x55c624416da4: file storage_service.c, line 1131.

框架

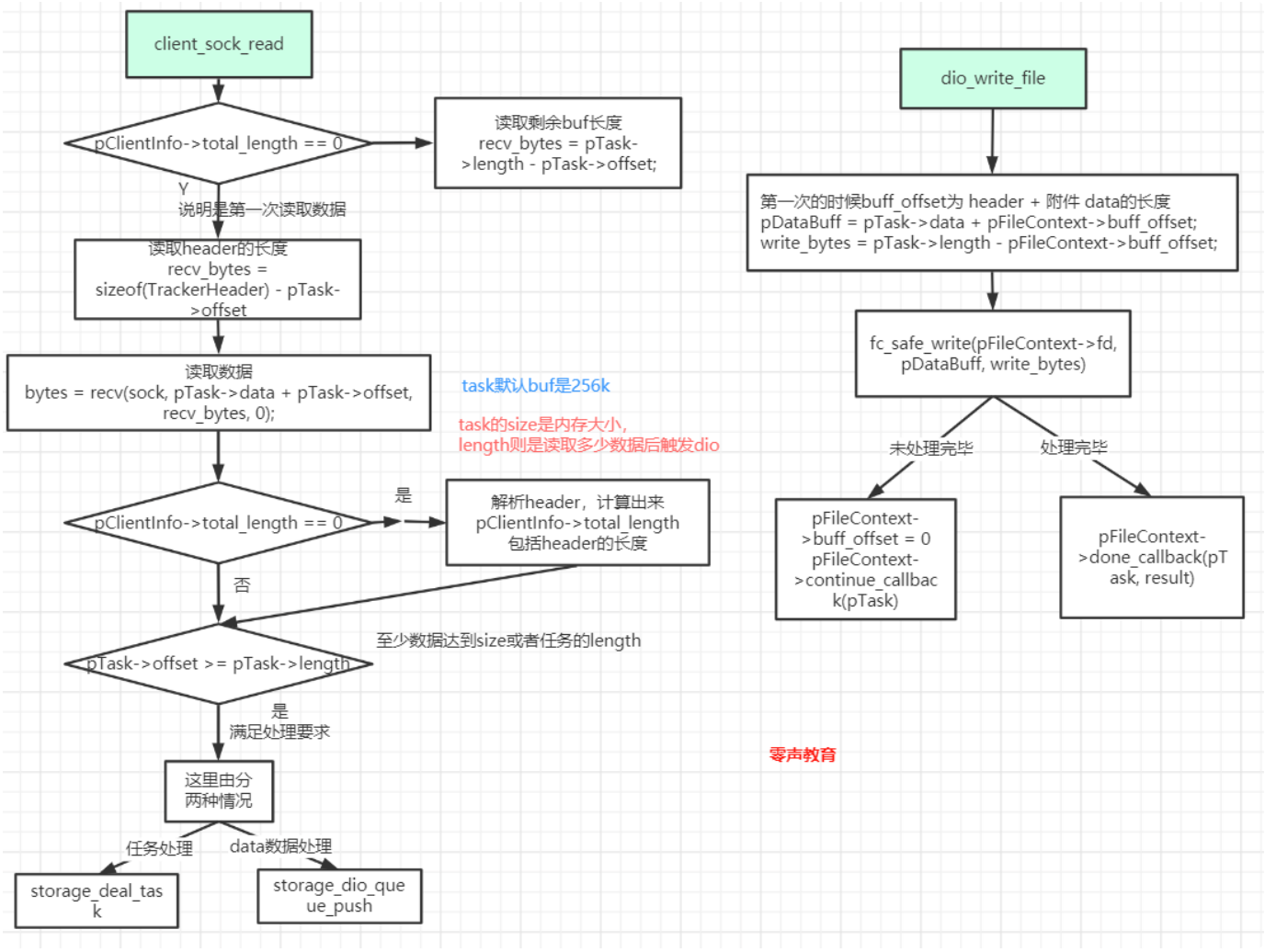


dio线程不会直接给nio线程设置各种读写事件，而是通过FDFS_STORAGE_STAGE_NIO_INIT、FDFS_STORAGE_STAGE_NIO_RECV、FDFS_STORAGE_STAGE_NIO_SEND、FDFS_STORAGE_STAGE_NIO_CLOSE、FDFS_STORAGE_STAGE_DIO_THREAD等状态 + 通过pipe通知nio线程响应**storage_recv_notify_read进行io事件的处理。**

核心函数：

- accept_thread_entrance accept线程入口
- work_thread_entrance nio线程入口
- dio_thread_entrance 数据处理线程入口

数据流程



fast_task_info：

- client_sock_read 读取数据
- 从socket接口读取buffer

StorageClientInfo

- storage_deal_task 处理任务

StorageFileContext

- dio_write_file负责数据写入
- 根据task buf确定要写入文件的起始地址和长度
- 更新写入文件的长度
- 判断是否整个已经写入完毕

storage 客户端封装 StorageClientInfo

pTask = (struct fast_task_info *)arg;
pClientInfo = (StorageClientInfo *)pTask->arg;

fast_task_info 从对象池种取出来

StorageClientInfo
StorageFileContext之间的关系

accept_thread_entrance等待客户端的连接

多线程负责

Thread 5 "fdfs_storaged" hit Breakpoint 1, storage_recv_notify_read (sock=17, event=<optimized out>, arg=<optimized out>) at storage_nio.c:169169 result = storage_nio_init(pTask);
(gdb) bt
#0 storage_recv_notify_read (sock=17, event=<optimized out>, arg=<optimized out>) at storage_nio.c:169
#1 0x00007faabc525d34 in deal_ioevents (ioevent=0x55c6266240a8) at ioevent_loop.c:32
#2 ioevent_loop (pThreadData=pThreadData@entry=0x55c6266240a8, recv_notify_callback=<optimized out>, clean_up_callback=0x55c62441e7ad <task_finish_clean_up>,
continue_flag=0x55c62465a368 <g_continue_flag>) at ioevent_loop.c:129
#3 0x000055c62440a98a in work_thread_entrance (arg=0x55c6266240a8) at storage_service.c:1960
#4 0x00007faabc7566db in start_thread (arg=0x7faab7b1e700) at pthread_create.c:463
#5 0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 5 "fdfs_storaged" hit Breakpoint 3, storage_send_add_event (pTask=pTask@entry=0x7faab7d61968) at storage_nio.c:227227 {
(gdb) bt
#0 storage_send_add_event (pTask=pTask@entry=0x7faab7d61968) at storage_nio.c:227
#1 0x000055c62441efa7 in storage_recv_notify_read (sock=17, event=<optimized out>, arg=<optimized out>) at storage_nio.c:192
#2 0x00007faabc525d34 in deal_ioevents (ioevent=0x55c6266240a8) at ioevent_loop.c:32

```
#3  ioevent_loop (pThreadData=pThreadData@entry=0x55c6266240a8, recv_notify_callback=
<optimized out>, clean_up_callback=0x55c62441e7ad <task_finish_clean_up>,
    continue_flag=0x55c62465a368 <g_continue_flag>) at ioevent_loop.c:129
#4  0x000055c62440a98a in work_thread_entrance (arg=0x55c6266240a8) at
storage_service.c:1960
#5  0x00007faabc7566db in start_thread (arg=0x7faab7b1e700) at pthread_create.c:463
#6  0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 5 "fdfs_storaged" hit Breakpoint 2, set_recv_event
(pTask=pTask@entry=0x7faab7d61968) at storage_nio.c:7171 {
(gdb) bt
#0  set_recv_event (pTask=pTask@entry=0x7faab7d61968) at storage_nio.c:71
#1  0x000055c62441ec17 in client_sock_write (sock=19, event=event@entry=4,
arg=arg@entry=0x7faab7d61968) at storage_nio.c:504
#2  0x000055c62441ee75 in storage_send_add_event (pTask=pTask@entry=0x7faab7d61968) at
storage_nio.c:231
#3  0x000055c62441efa7 in storage_recv_notify_read (sock=17, event=<optimized out>, arg=
<optimized out>) at storage_nio.c:192
#4  0x00007faabc525d34 in deal_ioevents (ioevent=0x55c6266240a8) at ioevent_loop.c:32
#5  ioevent_loop (pThreadData=pThreadData@entry=0x55c6266240a8, recv_notify_callback=
<optimized out>, clean_up_callback=0x55c62441e7ad <task_finish_clean_up>,
    continue_flag=0x55c62465a368 <g_continue_flag>) at ioevent_loop.c:129
#6  0x000055c62440a98a in work_thread_entrance (arg=0x55c6266240a8) at
storage_service.c:1960
#7  0x00007faabc7566db in start_thread (arg=0x7faab7b1e700) at pthread_create.c:463
#8  0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95


Thread 5 "fdfs_storaged" hit Breakpoint 4, client_sock_read (sock=19, event=1,
arg=0x7faab7d61968) at storage_nio.c:245245      if (pTask->canceled)
(gdb) bt
#0  client_sock_read (sock=19, event=1, arg=0x7faab7d61968) at storage_nio.c:245
#1  0x00007faabc525d34 in deal_ioevents (ioevent=0x55c6266240a8) at ioevent_loop.c:32
#2  ioevent_loop (pThreadData=pThreadData@entry=0x55c6266240a8, recv_notify_callback=
<optimized out>, clean_up_callback=0x55c62441e7ad <task_finish_clean_up>,
    continue_flag=0x55c62465a368 <g_continue_flag>) at ioevent_loop.c:129
#3  0x000055c62440a98a in work_thread_entrance (arg=0x55c6266240a8) at
storage_service.c:1960
#4  0x00007faabc7566db in start_thread (arg=0x7faab7b1e700) at pthread_create.c:463
#5  0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
```

0x7faab7d61968 storage_nio_init

#0 __libc_write (fd=18, buf=buf@entry=0x7faab7919e60, nbytes=nbytes@entry=8) at ../sysdeps/unix/sysv/linux/write.c:27#1 0x000055c62440ae2f in storage_nio_notify (pTask=pTask@entry=0x7faab7d61968) at storage_service.c:1927
#2 0x000055c624416f05 in storage_upload_file_done_callback (pTask=0x7faab7d61968, err_no=<optimized out>) at storage_service.c:1219
#3 0x000055c62441fc30 in dio_write_file (pTask=0x7faab7d61968) at storage_dio.c:525
#4 0x000055c62441f091 in dio_thread_entrance (arg=0x55c626634588) at storage_dio.c:748
#5 0x00007faabc7566db in start_thread (arg=0x7faab791a700) at pthread_create.c:463
#6 0x00007faabc22c71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

5 文件上传

需要debug的点

client：

- tracker_query_storage_store_without_group 向tracker请求storage
- storage_do_upload_file 向storage请求上传文件

tracker

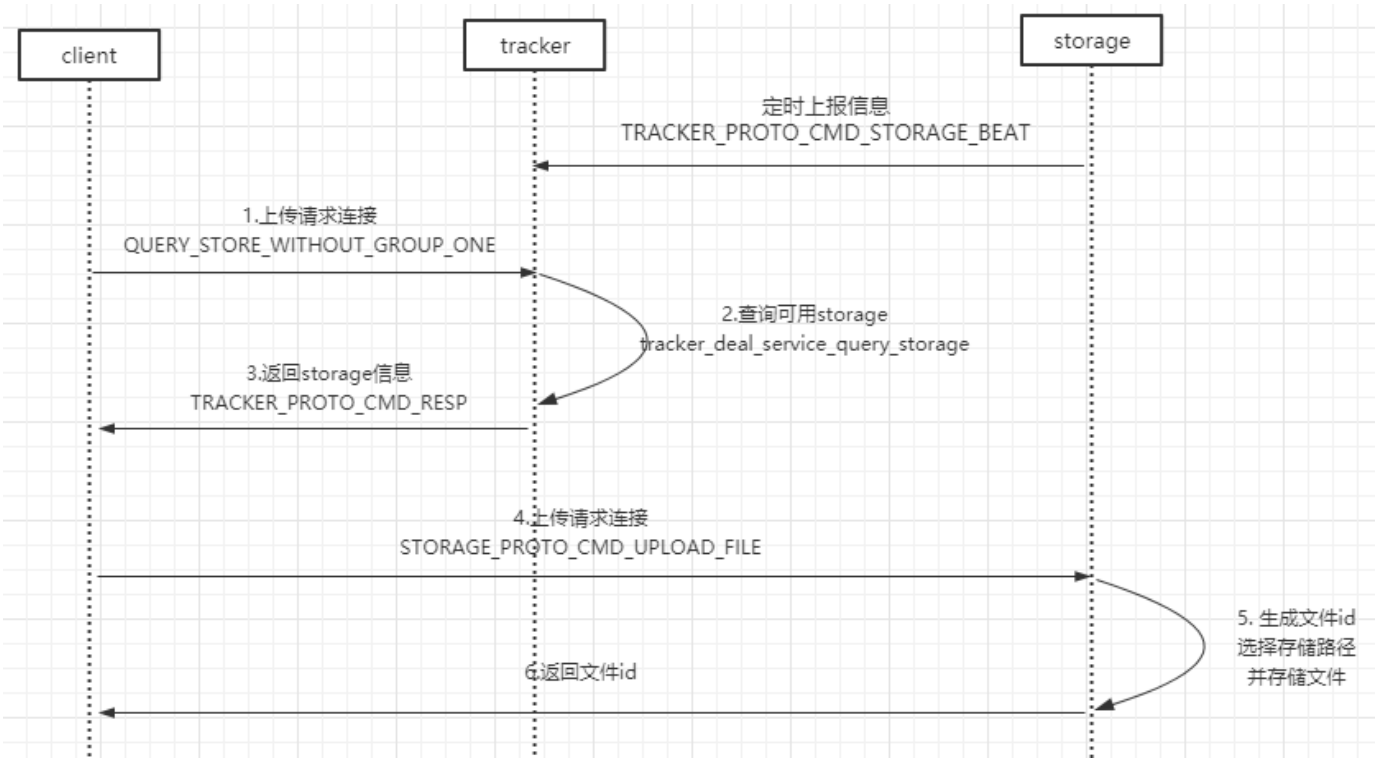
- tracker_deal_task
- tracker_deal_service_query_storage

storage

- storage_upload_file 响应上传
- dio_write_file 从io读取文件进行写入，更底层的写入函数 fc_safe_write
- storage_upload_file_done_callback 文件上传完成要回应客户端

FastDFS上传步骤解析

上传文件通信图如下：



以上传一个文件（upload.txt）为例，文件内容为：（文件大小为251字节）

Bash 复制代码

```
https://github.com/happyfish100/fastdfs/archive/V6.07.tar.gz
https://github.com/happyfish100/fastdfs-nginx-module/archive/V1.22.tar.gz
https://github.com/happyfish100/libfastcommon/archive/V1.0.50.tar.gz
https://nginx.org/download/nginx-1.16.1.tar.gz
```

1. 上传连接请求

fdfs_upload_file.c

客户端向集群中任意一台Tracker server发起TCP连接，建立连接后，客户端发送上传请求报文。报文data内容（十六进制）如下：

Bash 复制代码

```
0000000000000006500    //10个字节 十六进制
```

解释：data共10个字节，结构就是上面协议中谈到的FastDFS协议头部TrackerHeader。前8个字节为0，因为这个报文没有报文体，pkg_len=0，cmd=0x65（十六进制）=101（十进制），status=0。

101命令对应源码中的定义：

```c
#define TRACKER_PROTO_CMD_SERVICE_QUERY_STORE_WITHOUT_GROUP_ONE 101
```

用来请求group，以及对应的storage地址。

tracker_query_storage_store_without_group函数发起请求并读取响应。

等到响应的调用栈。

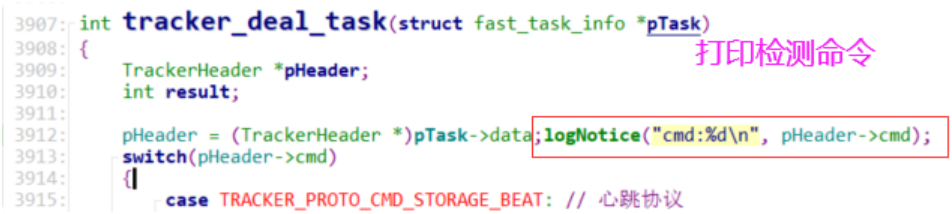
#0 0x00007ffff7684c84 in __GI___poll (fds=fds@entry=0x7fffffffe190, nfds=nfds@entry=1, timeout=timeout@entry=60000) at ../sysdeps/unix/sysv/linux/poll.c:29
#1 0x00007ffff798108d in poll (__timeout=60000, __nfds=1, __fds=0x7fffffffe190)
at /usr/include/x86_64-linux-gnu/bits/poll2.h:46
#2 **tcprecvdata_nb_ms** (sock=3, data=data@entry=0x7fffffffe1ee, size=size@entry=10, timeout_ms=60000, count=count@entry=0x0)
at sockopt.c:383
#3 0x00007ffff79811fb in tcprecvdata_nb_ex (sock=<optimized out>, data=data@entry=0x7fffffffe1ee, size=size@entry=10,
timeout=<optimized out>, count=count@entry=0x0) at sockopt.c:316
#4 0x0000555555557520 in fdfs_recv_header_ex (pTrackerServer=pTrackerServer@entry=0x55555577c7c8,
network_timeout=<optimized out>, in_bytes=in_bytes@entry=0x7fffffffe278) at tracker_proto.c:33
#5 0x000055555555763c in fdfs_recv_header (in_bytes=0x7fffffffe278, pTrackerServer=0x55555577c7c8) at tracker_proto.h:297
#6 fdfs_recv_response (pTrackerServer=pTrackerServer@entry=0x55555577c7c8, buff=buff@entry=0x7fffffffe270, // 读取响应信息
buff_size=buff_size@entry=50, in_bytes=in_bytes@entry=0x7fffffffe278) at tracker_proto.c:79
#7 0x000055555555de9c in tracker_query_storage_store_without_group (pTrackerServer=0x55555577c7c8,

pStorageServer=0x7fffffffe320, group_name=0x7fffffffe360 "",
store_path_index=0x7fffffffe31c)
at ../client/tracker_client.c:893
#8 0x000055555555659b in main (argc=3, argv=0x7fffffffe528) at fdfs_upload_file.c:90

2. 查询可用的的storage

处理函数：tracker_service.c: tracker_deal_service_query_storage，

我们在调用该函数的地方（tracker_deal_task）加打印，方便查看client、storage上报的信令。




```c
int tracker_deal_task(struct fast_task_info *pTask)
{
    TrackerHeader *pHeader;
    int result;

    pHeader = (TrackerHeader *)pTask->data;logNotice("cmd:%d\n", pHeader->cmd);
    switch(pHeader->cmd)
    {
        case TRACKER_PROTO_CMD_STORAGE_BEAT: // 心跳协议
```

Tracker选择group和storage
Tracker收到客户端的上传请求报文后，需要选择文件存储的group，选择方式可在tracker.conf配置文件配置，具体配置项如下：

C 复制代码

```
# 如何选择上传文件的存储group
# 0: 轮询
# 1: 指定group名称
# 2: 负载均衡，选择空闲空间最大的存储group
store_lookup=2
# 选择哪一个存储group，当store_lookup=1时，须指定存储group的名称
store_group=group2
```

选择完group之后，需要选择文件存储的storage server，选择方式可在tracker.conf配置文件配置，具体配置项如下：

```
# 如何选择上传文件的存储server
# 0: 轮询
# 1: 以ip地址排序的第一个地址
# 2: 以优先级排序(优先级在storage上配置)
store_server=0
```

返回的信息是怎么样的?

FDFSStorageDetail存储storage的详细信息
FDFSGroupInfo 存储组信息

FDFSGroups所有的group信息。

返回信息

```
int tracker_deal_task(struct fast_task_info *pTask)
{
    .....

    pHeader = (TrackerHeader *)pTask->data;
    pHeader->status = result;
    pHeader->cmd = TRACKER_PROTO_CMD_RESP; // 不管是什么命令的请求，都是以TRACKER_PROTO_CMD_RESP命令返回
    long2buff(pTask->length - sizeof(TrackerHeader), pHeader->pkg_len);

    send_add_event(pTask);

    return 0;
}
```

全角变量 FDFSGroups g_groups; 记录group以及对应storage的信息

3. 返回Storage信息

Tracker选择完可用的Storage服务器后，向客户端返回信息。返回的报文data内容（十六进制）如下（为了便于观察，做了内容换行）：

```
00000000000000286400                    //10个字节67726f7570320000000000000000000000
0x67, 0x72, 0x6f, 0x75, 0x70, 0x31, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
//16个字节
0x31, 0x32, 0x30, 0x2e, 0x32, 0x37, 0x2e, 0x31, 0x33, 0x31, 0x2e, 0x31, 0x39, 0x37, 0x0, 0x0,
//16个字节
000000000059d8                          //7个字节， 0x59d8 -> 即是23000
00                                      //1个字节
```

解释：data共50个字节

- 第一行10个字节为TrackerHeader，其中前8个字节为报文体长度，0x28（十六进制）=40（十进制），对应的接下来的报文体40个字节；0x64（十六进制）=100（十进制），100命令对应源码中的定义：
  #define TRACKER_PROTO_CMD_RESP 100
- 第二行16个字节为Storage的组名（group name），翻译为ASCII为group1，正好是我Storage中的一个组名。
- 第三行16个字节为Storage的ip，翻译为ASCII为120.27.131.197
- 第四行7个字节为Storage的port，翻译为ASCII为23000
- 第五行1个字节为storage_index 翻译为ASCII为 0

打印方式：
print in_buff
p /x（char[40])*in_buff

4. 上传文件

fdfs_upload_file.c:100行调用storage_upload_by_filename1函数，最终调用storage_do_upload_file

```c
int storage_do_upload_file(ConnectionInfo *pTrackerServer, \
    ConnectionInfo *pStorageServer, const int store_path_index, \
    const char cmd, const int upload_type, const char *file_buff, \
    void *arg, const int64_t file_size, const char *master_filename, \
    const char *prefix_name, const char *file_ext_name, \
    const FDFSMetaData *meta_list, const int meta_count, \
    char *group_name, char *remote_filename)
```

拿到了Tracker返回的Storage服务器的group name，ip和port后，向这台Storage服务器发起TCP连接请求，建立连接后，开始上传文件，先发送一个告知文件大小的报文，header内容（十六机制）如下：

```
00000000000010a0b00        //10个字节
```

解释：header 共10个字节
前8个字节为之后所有报文报文体的长度，0x10c（十六进制）=266（十进制），266字节比真实的文件251字节多了15个字节，这15个字节用于告知storage_index，文件长度和文件名。

cmd=0x0b（十六进制）=11（十进制），11命令对应的源码定义如下：
#define STORAGE_PROTO_CMD_UPLOAD_FILE   11

发送的时候header + data，实际是发送了25字节
(gdb) p /x *out_buff@25 $28 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0xa, 0xb, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xfb, 0x74, 0x78, 0x74, 0x0,
 0x0, 0x0}

```
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0xa, 0xb, 0x0,   10字节 header
0x0,                                          storage_index 1字节
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xfb,          文件大小，0xfb即是251
0x74, 0x78, 0x74, 0x0, 0x0, 0x0                   扩展名6字节，这里是txt
```

b storage_client.c:879

b storage_client.c:940

storage_do_upload_file函数源码分析：

- 封装header
- 封装data （包括 storage index、文件大小、扩展名）
- 然后调用tcpsenddata_nb发送

5.选择存储路径、生成文件id并存储文件

storage响应函数storage_service.c: storage_deal_task，然后调用
int storage_upload_file(struct fast_task_info *pTask, bool bAppenderFile)。

在storage_deal_task函数加上打印

```
8318: int storage_deal_task(struct fast_task_info *pTask)
8319: {
8320:     TrackerHeader *pHeader;
8321:     StorageClientInfo *pClientInfo;
8322:     int result;
8323:
8324:     pClientInfo = (StorageClientInfo *)pTask->arg;
8325:     pHeader = (TrackerHeader *)pTask->data;
8326:     logNotice("cmd:%d\n", pHeader->cmd);
8327:     switch(pHeader->cmd)
8328:     {
8329:         case STORAGE PROTO CMD DOWNLOAD FILE:
```

信令：STORAGE_PROTO_CMD_UPLOAD_FILE 11

分析：storage_upload_file函数

- pClientInfo->total_length 此时为276，为什么我们client里面显示的是266，这里是276呢，是因为这里包括了header的10字节。
- (gdb) p /x *(char *)pTask->data@25$10 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0xa, 0xb, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xfb, 0x74, 0x78, 0x74, 0x0, 0x0, 0x0}
- 调用storage_get_filename生成文件名，保存在pFileContext->filename，比如"/home/fastdfs/storage_group1_23000/data/00/00/eBuDxWCt-P2AdZSMAAAA-zmlXKl556.txt"

b storage_nio_notify
b storage_upload_file_done_callback
b dio_write_file

封装好StorageFileContext，最终扔到线程池里进行处理。

（1）选择storage path

storage server收到客户端上传文件的请求后，为文件分配一个数据存储目录，分配的规则可在tracker.conf配置文件中配置，具体配置项如下：

```
# 选择存储server的哪一个路径(磁盘或挂载点)上传文件
# 0: 轮询
# 2: 负载均衡，选择空闲空间最大的路径来存储文件
store_path=0
```

（2）生成fileId

选定了存储目录后，storage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。
调用storage_get_filename函数

（3）选择两级目录

默认每个存储目录下有两级子目录，分别是00~FF，共256*256个子目录，storage选择两级目录的规则可以在storage.conf配置文件中配置，具体配置项如下：

```
# 选择文件存储的路径
# 0: 轮询（默认）
# 1: 随机，通过hash分布
file_distribute_path_mode=0

# 当选择了轮询方式，存储路径从00/00开始，目录下的文件数到达这个值（默认100），
# 就换下一个目录00/01，依次直到FF/FF
file_distribute_rotate_count=100
```

选择了目录后，将文件以fileId为文件名存储到该子目录下。

（4）存储文件

client客户端：storage_client.c:951 ，然后调用 tcpsendfile_ex 上传文件（走1488行逻辑），本质是调用 sendfile 发送（sendfile函数在两个文件描述符之间传递数据（完全在内核中操作），从而避免了内核缓冲区和用户缓冲区之间的数据拷贝，效率很高，**被称为零拷贝**。）

storage服务器：dio_write_file函数负责写入数据

开始发送真正的文件内容，报文data内容（十六进制）如下：

```
00                      //1个字节
00000000000000fd        //8个字节
747874000000            //6个字节
68747470733a2f2f6769746875622e636f6d2f686170707966697368313030302f666173746466732f
617263686976652f56362e30342e7461722e677a0d0a68747470733a2f2f6769746875622e636f6d
2f686170707966697368313030302f666173746466732d6e67696e782d6d6f64756c652f6172636869
76652f56312e32322e7461722e677a0d0a68747470733a2f2f6769746875622e636f6d2f68617070
796669736831303032f6c696266617374636f6d6d6f6e2f617263686976652f56312e302e34322e74
61722e677a0d0a68747470733a2f2f6e67696e782e6f72672f646f776e6c6f61642f6e67696e782d3
12e31362e312e7461722e677a//253个字节
```

解释：data共266个字节
第一行1个字节为storage_index

第二行8个字节为文件长度，fd（十六进制）=251（十进制），表示真正的文件大小
第三行7个字节为文件后缀名，翻译为ASCII为txt
第四行253个字节为上传文件的内容，翻译为ASCII即为开头出的“upload.txt”文件中的内容

如果是大文件上传的时候

dio_write_file的处理，比如上传redis压缩包，是怎么处理?
fdfs_upload_file /etc/fdfs/client.conf ~/0voice/cloud-drive/redis-6.2.3.tar.gz
group1/M00/00/00/eBuDxWCuAUaAYLNfACV58jYX3JA.tar.gz

6. 上传成功，返回访问路径

最后通过storage_upload_file_done_callback 返回。

Storage将文件写入磁盘后，返回客户端文件的路径和文件名，报文data内容（十六进制）如下：

```
000000000000003c6400                    //10个字节
67726f7570310000000000000000000000      //16个字节
4d30302f30302f30302f774b67714856344f51517941626f3959414141415f666453706d67383535
2e747874                                 //44个字节
```

解释：data共70个字节

- 第一行10个字节为TrackerHeader，前8个字节为报文体长度，3c（十六进制）=60（十进制）；cmd=64（十六进制）=100（十进制），100命令对应的源码定义如下：
- #define STORAGE_PROTO_CMD_RESP 100
- 第二行16个字节为组名（group name），翻译为ASCII为group1
- 第三行44个字节为文件路径和文件名，翻译为ASCII为M00/00/00/wKgqHV4OQQyAbo9YAAAA_fdSpmg855.txt

6 文件下载

fdfs_download_file /etc/fdfs/client.conf
group1/M00/00/00/eBuDxWCuAUaAYLNfACV58jYX3JA.tar.gz

或者fdfs_download_file /etc/fdfs/client.conf group1/M00/00/00/eBuDxWCuAbOAJAJnAAAA-3Qtcs8577.txt

需要debug的点

client：

- storage_download_file_to_file，最终调用为storage_do_download_file_ex
- 请求入口 STORAGE_PROTO_CMD_DOWNLOAD_FILE

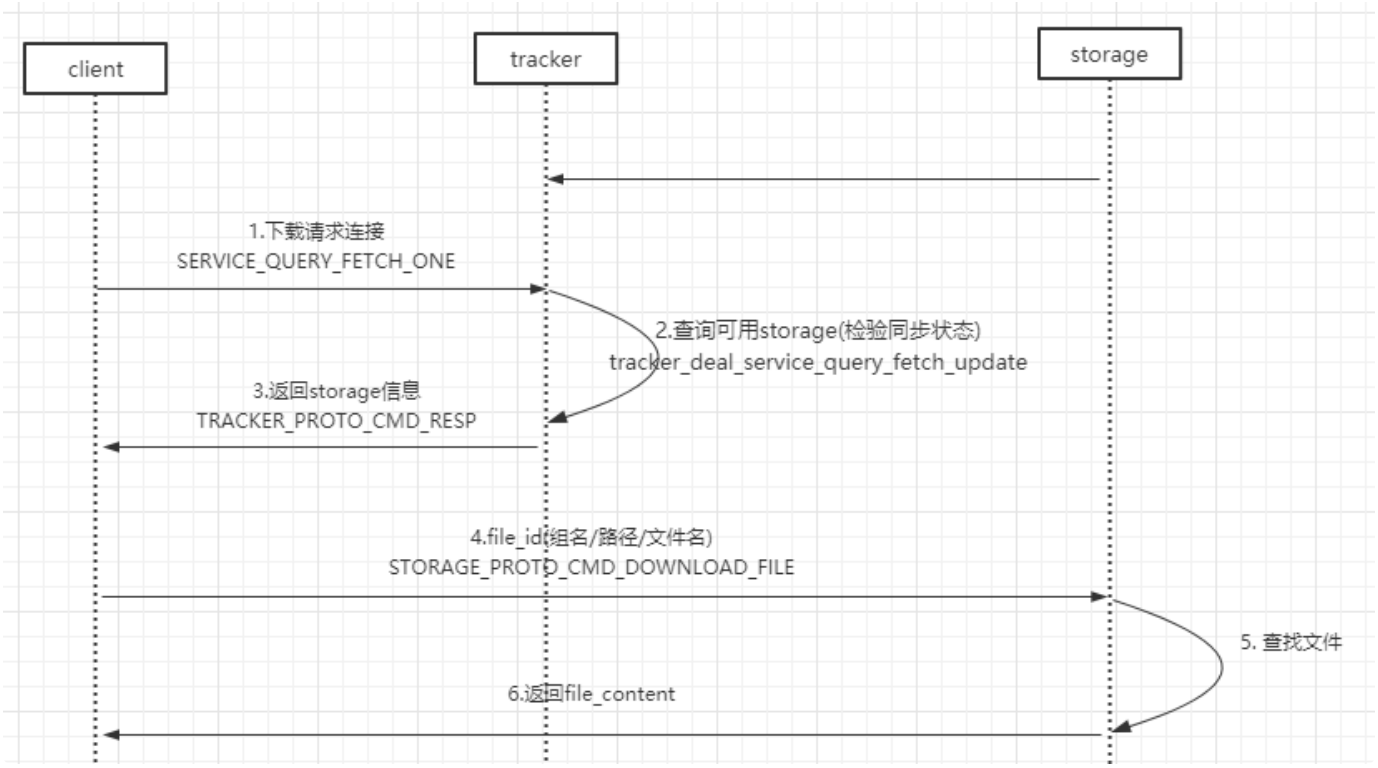
tracker

- SERVICE_QUERY_FETCH_ONE
- tracker_deal_service_query_fetch_update

storage

- storage_server_download_file 处理入口
- dio_read_file
- fc_safe_read

文件下载步骤解析



1 上传请求连接

发起 storage_do_download_file_ex

2 查询可用的storage

服务器响应：tracker_deal_service_query_fetch_update

3 返回storage信息

和上传文件是类似的

4 下载文件

client客户端请求：storage_do_download_file_ex， 然后
STORAGE_PROTO_CMD_DOWNLOAD_FILE请求下载。
storage服务响应：storage_server_download_file
storage_read_from_file
storage_download_file_done_callback
真正读取文件的函数：dio_read_file

io模型

数据如何来

如何发送

7 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	
120.27.131.197	fdfs_trackerd	tracker_22123.conf	
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


```

fdfs_trackerd /etc/fdfs/tracker_22124.conf stop

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
```

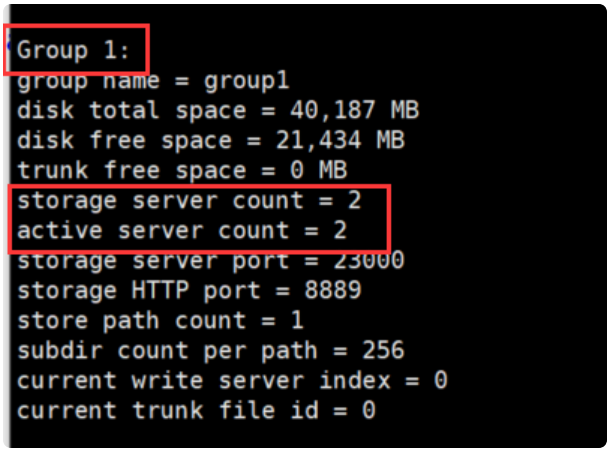
主要修改tracker_server、url_have_group_name、store_path0。

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/ctepQmCotjqAIRNPAAAjuZXPuAg28.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/ctepQmCotjqAIRNPAAAjuZXPuAg28.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/ctepQmCotjqAIRNPAAAjuZXPuAg28.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》

8 部分调试记录

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

服务器常见报错处理

FastDFS 磁盘空间不足（tracker_query_storage fail,error no : 28,error info : No space left on device）

1、在文件服务器上中用fastDFS自带命令测试，返回空间不足。

2、查看了data所在路径磁盘大小，发现当前挂载磁盘空间剩余10%（fastDFS默认预留10%的磁盘空间）
3、查询了该服务器所有挂载磁盘大小，都不足支撑当前服务，服务器管理人员已下班，但系统第二天还要继续使用，所以这剩余的10%磁盘空间要用起来。
4、在tracker.conf中，将reserved_storage_space的值修改为5%，预留5%的磁盘空间。

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