零声教育 Darren QQ326873713

[零声教育 C/C++Linux服务器开发/高级架构师]

代码：2404_vip/tuchuang/tc-mini4

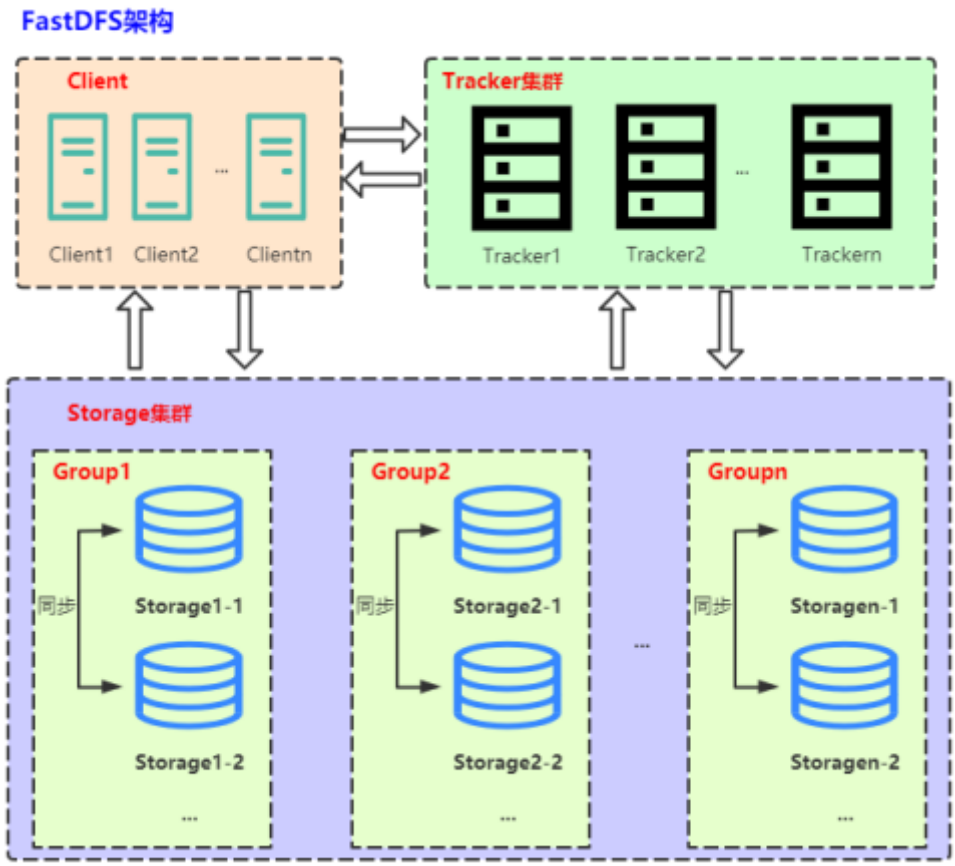
分布式FastDFS存储集群部署

FastDFS集群部署，课上不讲怎么部署，只讲同步的原理，主要是binlog原理

同步机制和线性扩容分析

整体功能查漏补缺

1 集群部署-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。

规划如下所示，**如果是云服务器一定要记得开放对应的端口。**

服务器地址	服务程序	对应配置文件(端口区分)	

服务器地址	服务程序	对应配置文件(端口区分)	
192.168.1.22	fdfs_trackerd	tracker_1.conf，之前tracker.conf 路径 tracker_1	
192.168.1.18	fdfs_trackerd	tracker_1.conf	
192.168.1.22	**fdfs_storaged**	storage_1.conf	
192.168.1.18	**fdfs_storaged**	storage_1.conf	

storage是要部署在不同的服务器上，改端口是没有用。

1.1 192.168.1.22服务器

进入/etc/fdfs，创建配置文件和创建目录

```
cd /etc/fdfs

sudo cp tracker.conf.sample  tracker_1.conf

sudo mkdir /home/fastdfs/tracker_1    同一个服务器创建多个tracker存储路径

sudo cp storage.conf.sample  storage_1.conf

sudo mkdir /home/fastdfs/storage_1
```

把当前服务器现有的tracker、storage全部停止，之前启动用哪个配置文件，现在就用哪个配置文件停止

```
sudo /usr/bin/fdfs_trackerd /etc/fdfs/tracker_1.conf stop
sudo /usr/bin/fdfs_storaged /etc/fdfs/storage_1.conf stop
```

然后我们要修改对应的配置文件

1.1.1 tracker_1.conf

在这里，tracker_1.conf 只是修改一下 Tracker 存储日志和数据的路径

```
# 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_1
```

主要修改**port、base_path**路径。

启动tracker_1

```
sudo /usr/bin/fdfs_trackerd /etc/fdfs/tracker_1.conf
```

此时查看启动的tracker

lqf@ubuntu:/etc/fdfs$ sudo ps -ef | grep tracker

root 18100 1 0 22:12 ? 00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker_1.conf

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

1.1.2 storage_1.conf

在这里，storage_1.conf 只是修改一下 storage 存储日志和数据的路径

```
# 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_1
# 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_1
# 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=192.168.1.22:22122
tracker_server=192.168.1.18:22122
```

主要修改：port、base_path、store_path0、tracker_server

启动storage_1

```
sudo /usr/bin/fdfs_storaged /etc/fdfs/storage_1.conf
```

1.2 192.168.1.18服务器

进入/etc/fdfs，创建配置文件和创建目录

```
cd /etc/fdfs
sudo cp tracker.conf.sample  tracker_1.conf
sudo mkdir -p /home/fastdfs/tracker_1


sudo cp storage.conf.sample  storage_1.conf
sudo mkdir -p /home/fastdfs/storage_1
```

把当前服务器现有的tracker、storage全部停止（如果有），之前启动用哪个配置文件，现在就用哪个配置文件停止

```
sudo /usr/bin/fdfs_trackerd /etc/fdfs/tracker_1.conf stop
sudo /usr/bin/fdfs_storaged /etc/fdfs/storage_1.conf stop
```

然后我们要修改对应的配置文件

1.2.1 tracker_1.conf

在这里，tracker_1.conf 只是修改一下 Tracker 存储日志和数据的路径

```
# 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_1
```

主要修改**port、base_path**路径。

启动tracker_1

```
sudo /usr/bin/fdfs_trackerd /etc/fdfs/tracker_1.conf
```

此时查看启动的tracker

lqf@ubuntu:/etc/fdfs$ sudo ps -ef | grep tracker
root 23369 1 0 05:12 ? 00:00:00 /usr/bin/fdfs_trackerd /etc/fdfs/tracker_1.conf

1.2.2 storage_1.conf

在这里，storage_1.conf 只是修改一下 storage 存储日志和数据的路径

```
# 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_1
# 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_1
# 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=192.168.1.22:22122
tracker_server=192.168.1.18:22122
```

主要修改：port、base_path、store_path0、tracker_server

启动storage_1

```
sudo /usr/bin/fdfs_storaged /etc/fdfs/storage_1.conf
```

1.3 查看集群情况

可以通过 fdfs_monitor 查看集群的情况

```
# 查看 Storage 是否已经注册到 Tracker 服务器中
# 当查看到 ip_addr = 192.168.1.27: (localhost.localdomain)  ACTIVE
# ACTIVE 表示成功
sudo /usr/bin/fdfs_monitor /etc/fdfs/storage_1.conf
```

1.4 测试集群

1.4.1 配置client_1.conf

```
sudo mkdir /home/fastdfs/client_1

sudo cp client_1.conf.sample client_1.conf
```

修改client_1.conf

```
# 修改client的base path路径
base_path = /home/fastdfs/client_1
# 配置tracker server地址
tracker_server=192.168.1.22:22122
tracker_server=192.168.1.18:22122
```

1.4.2 配置mod_fastdfs.conf

在两台服务器都需要部署fastdfs-nginx-module

修改sudo vim /etc/fdfs/mod_fastdfs.conf

store_path0=/home/fastdfs/storage_1#保存日志目录, 跟storage 一致即可

tracker_server = 192.168.1.22**:22122**

tracker_server=192.168.1.18:22122 #tracker服务器的IP地址以及端口号, 确保跟storage 一致即可

```
# Tracker 服务器IP和端口修改
tracker_server=192.168.1.22:22122
tracker_server=192.168.1.18:22122
# 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_1
# 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，
store_path_count=1
```

主要修改tracker_server、**url_have_group_name**、store_path0。

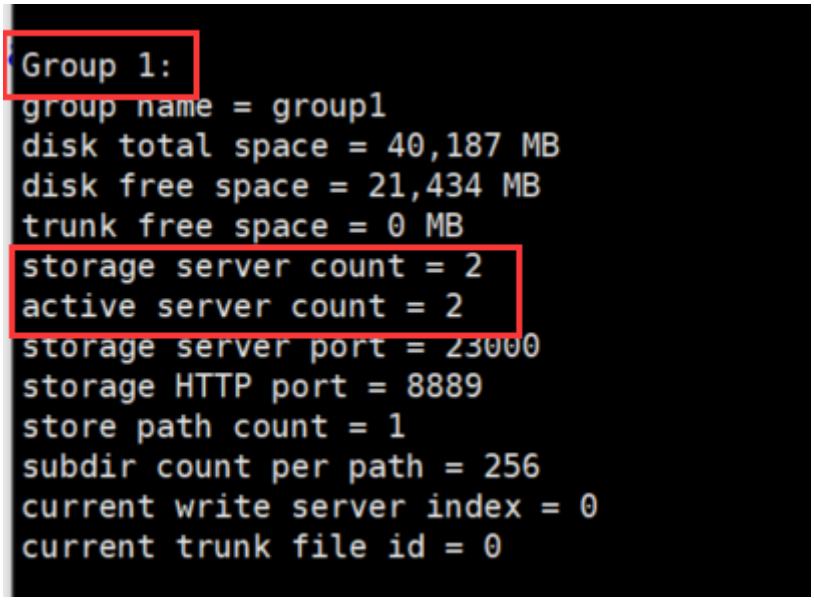
注意：两台服务器都需要配置nginx，参考第一节课的1-3.1-FastDFS_单机版环境搭建.pdf，否则在启用集群后，有些没有正确搭建nginx的服务器对应的ip地址不能正常访问。

1.4.3 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_1.conf

正常两边都提示：



存在2个Active的storage。

1.4.4 测试上传文件

```
/usr/bin/fdfs_upload_file   /etc/fdfs/client_1.conf  /etc/fdfs/storage_1.conf
```

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

小文件存储的：group1/M00/00/00/MetMcGNqQLmAHRSMAAAoZ6CgKMM56.conf（这里只是为了对比不是小文件存储的fileid）

查看两台服务器下的00/00目录是否存在相同的文件。

1.4.5 下载测试

（1）正常下载

```
fdfs_download_file /etc/fdfs/client_1.conf
group1/M00/00/00/wKgBEmaqMsiAPwKAAAAoBuJE_mE67.conf
```

（2）停止192.168.1.22的storage

```
sudo /usr/bin/fdfs_storaged /etc/fdfs/storage_1.conf  stop
```

然后再下载数据

```
fdfs_download_file /etc/fdfs/client_1.conf
group1/M00/00/00/wKgBEmaqMsiAPwKAAAAoBuJE_mE67.conf
```

此时还可以正常下载数据

（3）继续停止另一个storage server(192.168.1.18)

```
sudo /usr/bin/fdfs_storaged /etc/fdfs/storage_1.conf  stop
```

然后再继续下载数据

```
fdfs_download_file /etc/fdfs/client_1.conf
group1/M00/00/00/wKgBEmaqMsiAPwKAAAAoBuJE_mE67.conf
```

此时就报错了，因为storage都已经停止了。

root@iZbp1d83xkvoja33dm7ki2Z:~# fdfs_download_file /etc/fdfs/client_1.conf group1/M00/00/00/eBuDxWIgcYqACM8LAAAoaTcAqLc40.conf[2021-05-22 15:49:51] ERROR - file: tracker_proto.c, line: 50, server: 192.168.1.22:22122, 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://192.168.1.22/group1/M00/00/00/wKgBEmaqMsiAPwKAAAAoBuJE_mE67.conf

1.4.6 恢复storage的运行

两台服务器都执行：/usr/bin/fdfs_storaged /etc/fdfs/storage_1.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。

1.4.7 报错处理

tracker_query_storage fail, error no: 28, error info**: No space left on device**

在**tracker xx.conf**中，将reserved_storage_space的值修改为5%，预留5%的磁盘空间。

```
# reserved storage space for system or other applications.
# if the free(available) space of any stoarge server in
# a group <= reserved_storage_space, no file can be uploaded to this group.
# bytes unit can be one of follows:
### G or g for gigabyte(GB)
### M or m for megabyte(MB)
### K or k for kilobyte(KB)
### no unit for byte(B)
### XX.XX% as ratio such as: reserved_storage_space = 10%
reserved_storage_space = 5%
```

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

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

2 tracker和storage目录结构

2.1 tracker server目录及文件结构

```
/home/fastdfs/tracker_1# 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  重要
|   └── 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。

unix时间转换网站：[时间戳(Unix timestamp)转换工具 - 在线工具 (tool.lu)]

storage_groups_new.dat

各个参数如下

group_name：组名

storage_port：storage server端口号

比如：

```
# 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_new.dat 重要

比如

```
# storage 192.168.1.18:23000
[Storage001]
    group_name=group1
    ip_addr=192.168.1.18
    status=1
    version=6.07
    join_time=1722429095
    storage_port=23000
    storage_http_port=8888
    domain_name=
    sync_src_server=192.168.1.22

....
# storage 192.168.1.22:23000
[Storage002]
    group_name=group1
    ip_addr=192.168.1.22
    status=7
    version=6.07
```

```
    join_time=1722427336
    storage_port=23000
    storage_http_port=8888
    domain_name=
    sync_src_server=192.168.1.18
```

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）

2.2 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
|     |                                  192.168.1.22_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 （推送方标记）
  - **until_timestamp：同步已有数据截至时间点**（UNIX时间戳） （推送方） 上次同步时间结点
  - scan_row_count：总记录数
  - sync_row_count：已同步记录数

  如果还有其他storage，则有更多的${ip_addr}_${port}.mark，比如10.0.0.2_23000.mark 10.0.0.3_23000.mark。

3 FastDFS文件同步

文件上传成功后，其它的storage server才开始同步，其它的storage server怎么去感知，tracker server是怎么通知storage server?

storage定时发送心跳包到tracker，并附带同步的时间节点 --- tracker 返回我们其他storage的状态。

正常文件上传完成后，就记录到binlog缓存中，系统定时刷入binlog文件。
系统有线程定时读取binlog文件，当有新增行时，判断该记录是源文件记录还是副本文件记录。
系统只主动发送源文件，副本文件不做处理(非启动时流程)。

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 初始化，尚未得到同步已有数据的源服务器
- #define FDFS_STORAGE_STATUS_WAIT_SYNC 1 等待同步，已得到同步已有数据的源服务器
- #define FDFS_STORAGE_STATUS_SYNCING 2 同步中
- #define FDFS_STORAGE_STATUS_IP_CHANGED 3
- #define FDFS_STORAGE_STATUS_DELETED 4 已删除，该服务器从本组中摘除
- #define FDFS_STORAGE_STATUS_OFFLINE 5 离线
- #define FDFS_STORAGE_STATUS_ONLINE 6 在线，尚不能提供服务
- #define FDFS_STORAGE_STATUS_ACTIVE 7 在线，可以提供服务
- #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

3.1 同步日志所在目录

比如在**192.168.1.18**服务器看到的sync log：

root@xx:/home/fastdfs/storage_group1_23000/data/sync#

192.168.1.22_23000.mark **同步状态文件，记录本机到**192.168.1.22 **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**192.168.1.22**，**则该目录还存在**192.168.1.22_23000.mark

比如在**192.168.1.22**服务器看到的sync log：

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

192.168.1.18_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

3.2 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/oYYBAF285cOIHiVCAACI-7zX1qUAAAAVgAACC8AAIkT490.txt

1646123047 c M00/00/00/oYYBAF285luIK8jCAAAJeheau6AAAAAVgABI-cAAAmS021.xml

1646123193 A M00/00/00/rBMYd2IaLXqASSVXAAAHuj79dAY65.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）

源	副本
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：

192.168.1.22_23000.mark

binlog_index=0 **//binlog索引id** 表示上次同步给192.168.1.22机器的最后一条binlog文件索引

binlog_offset=3944 **//当前时间binlog 大小 (单位是字节)**表示上次同步给192.168.1.22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

192.168.1.18_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

3.3 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。

3.4 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

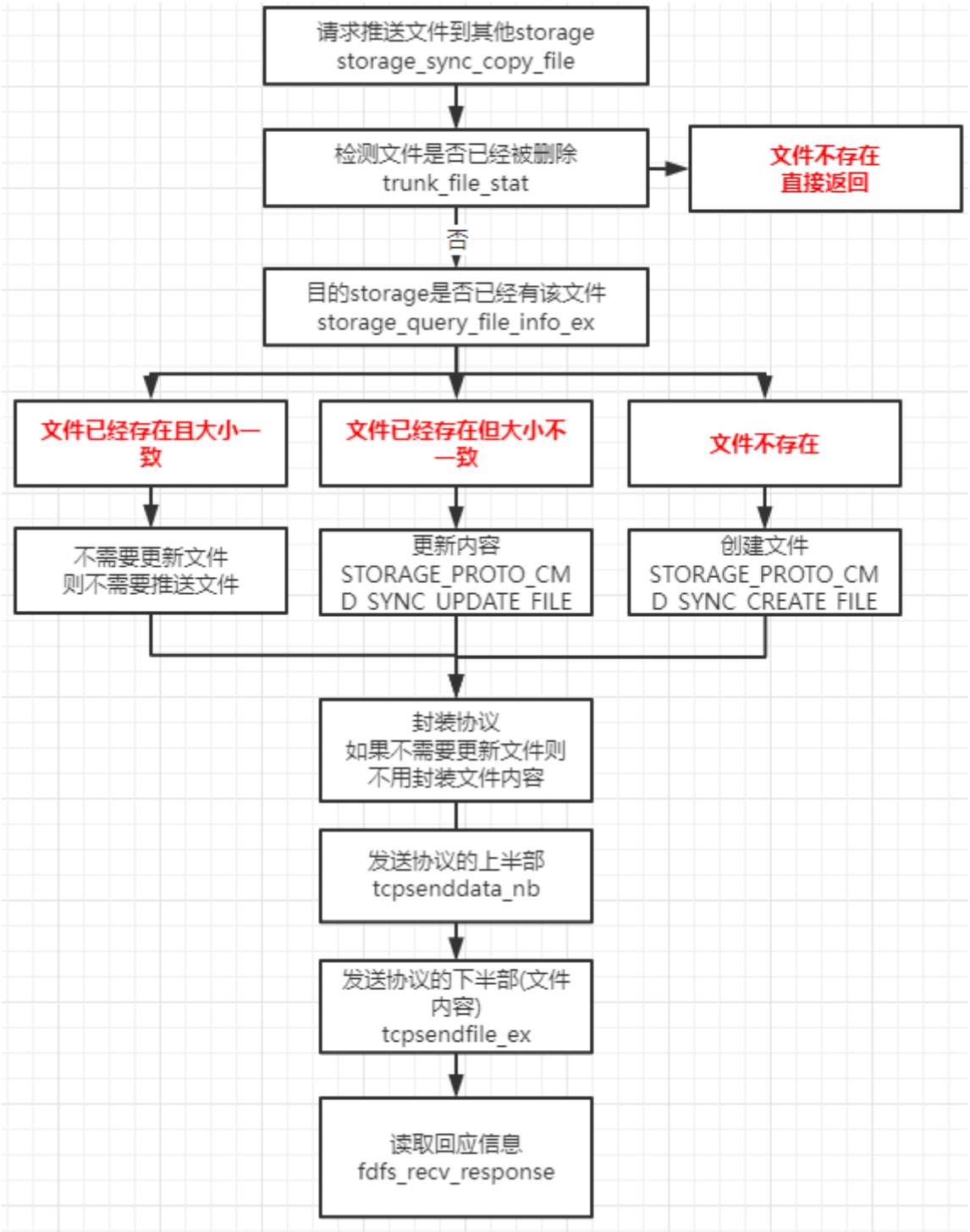
storage_sync_thread_entrance 同步线程

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192.168.1.22的同步则打开192.168.1.22_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192.168.1.22_23000.mark文件之中。

storage_open_readable_binlog



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**

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。

3.5 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**（Binlog中的第一个字段timpstamp**，这个时间也会更新到storage_stat.dat的last_sync_update）。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。**

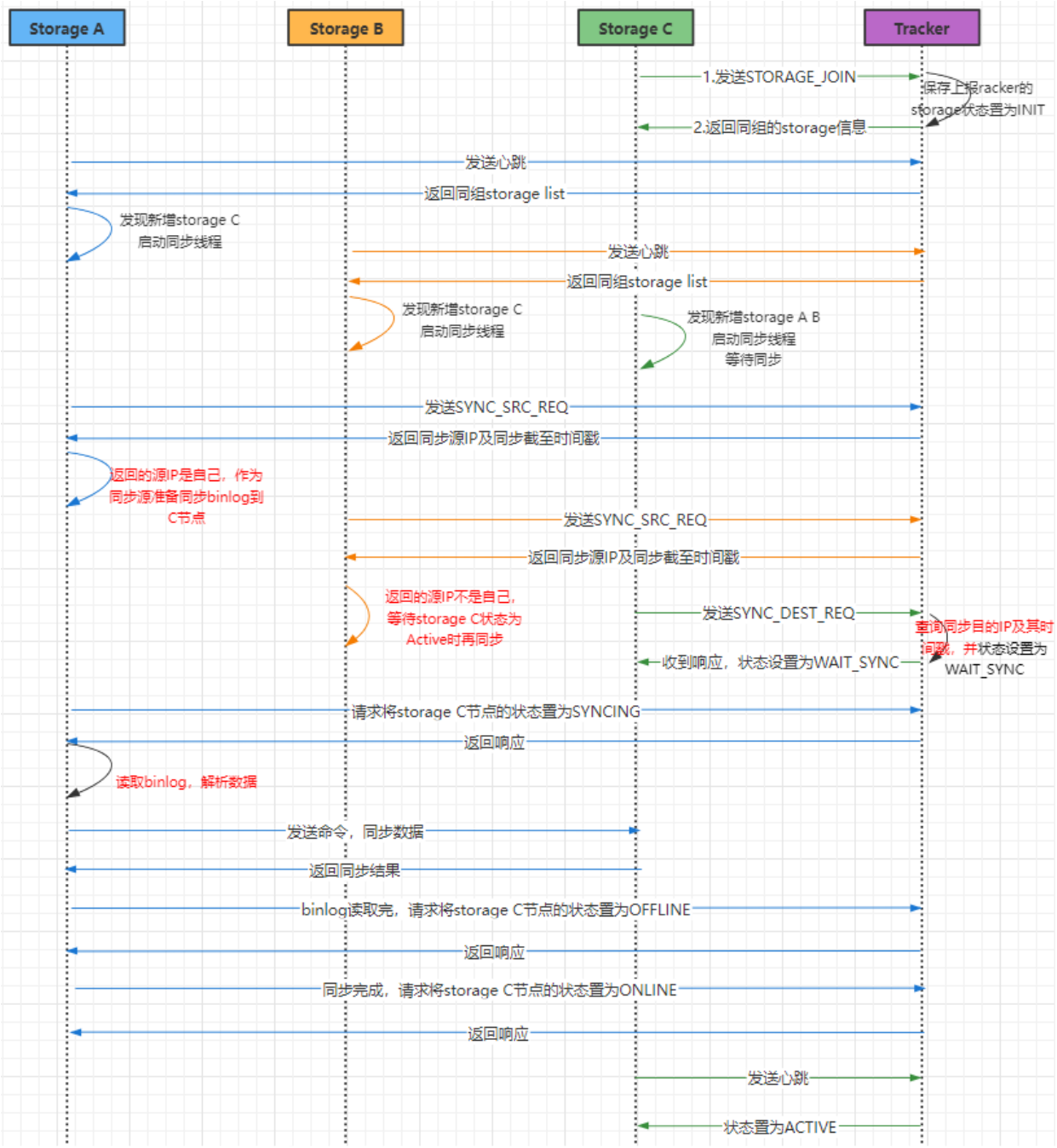
Tracker也会定期将该值写入到一个文件之中，Storage_sync_timestamp.dat，内容如下：

```
group1,10.0.0.1,    0,          1408524351, 1408524352
group1,10.0.0.2,    1408524353, 0,          1408524354
group1,10.0.0.3,    1408524355, 1408524356, 0
```

每一行记录了，对应Storage同步给其他Storage的最后时间，如第一行表示10.0.0.1同步给10.0.0.2的最后时间为1408524351，同步给10.0.0.3的最后时间为1408524352；同理可以知道后面两行的含义了。每行中间都有一个0，这个零表示自己同步给自己，没有记录。按照纵列看，取非零最小值就是最后最早同步时间了，如第三纵列为10.0.0.1被同步的时间，其中1408524353就是其最后最早被同步时间。

3.6 新增节点同步流程（选学）

在已有A、B节点上，新增节点storage C。



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/CtAqWVjTbm2AIqTkAAACd_nIZ7M797.jpg

1490251373 表示时间戳

C 表示操作类型

M02/52/CB/CtAqWVjTbm2AIqTkAAACd_nIZ7M797.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。

3.7 Tracker选择客户端下载文件的storage的原则

1. 在同group下，获取最小的一个同步时间点(各个storage在同一时间，同步完成的时间点不一样)
2. 在最小同步时间点之前的文件，按照用户的规则随意选择一个storage。
3. 在最小同步时间点之后的文件，选择源storage提供给客户端。

4 改进图片分享功能

4.1 数据库设计

添加了新的数据库表，以便和原理的设计有所区别，主要是把文件下载路径**file_url**直接放在了共享图片列表，这样不用每次浏览分享的图片时还要去file_info查询。

共享图片列表 (share_picture_list2)

字段名	中文含义	类型	是否允许空	默认值	
id	序号	int	否		自动递增，主键
user	文件所属用户	varchar(32)	否		
file_url	**文件的下载地址**	**varchar(256)**	**否**		
file_name	文件名字	varchar(128)	否		
urlmd5	图床urlmd5	varchar(256)	否		
key	提取码	varchar(8)	否		
pv	访问量 page view	int	否		默认值为1，访问一次加1
size	文件大小	bigint	否	0	以字节为单位
create_time	文件共享时间	timestamp	否	CURRENT_TIMESTAMP	

表创建

```
CREATE TABLE `share_picture_list2` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '编号',
  `user` varchar(32) NOT NULL COMMENT '文件所属用户',
  `file_url` varchar(256) NOT NULL COMMENT '文件md5',
  `file_name` varchar(128) DEFAULT NULL COMMENT '文件名字',
  `urlmd5` varchar(256) NOT NULL COMMENT '图床urlmd5',
  `key` varchar(8) NOT NULL COMMENT '提取码',
  `pv` int(11) DEFAULT '1' COMMENT '文件下载量，默认值为1，下载一次加1',
  `size` bigint(20) DEFAULT '0' COMMENT '文件大小',
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '文件创建时间',
  PRIMARY KEY (`id`),
  KEY   `idx_urlmd5_user` (`urlmd5`, `user`)
) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8 COMMENT='图床文件列表';
```

4.2 redis的使用

代码在api_sharepicture2.cc，

1. 分享图片后 将分享的图片信息缓存到redis，以urlmd5为key， 以图片信息作为hash结构存储；
2. 浏览分享图片时先根据urlmd5从redis读取图片信息，如果没有再从MySQL读取，并缓存到redis。

5 图床功能的完善

1. 图片分享功能使用redis缓存浏览时要通过urlmd5查询的数据，这样能减少mysql的操作压力
2. 增加共享文件列表的获取
3. 增加共享文件下载帮的排行，**使用redis做排行**
4. 增加文件的删除处理

具体参考第一节课的课件：课程资料/1/1-2-零声图床架构和功能分析v1.1.pdf · master · 2404_vip / 9.2-tuchuang · GitLab (0voice.com)

重点讲解涉及到redis的功能

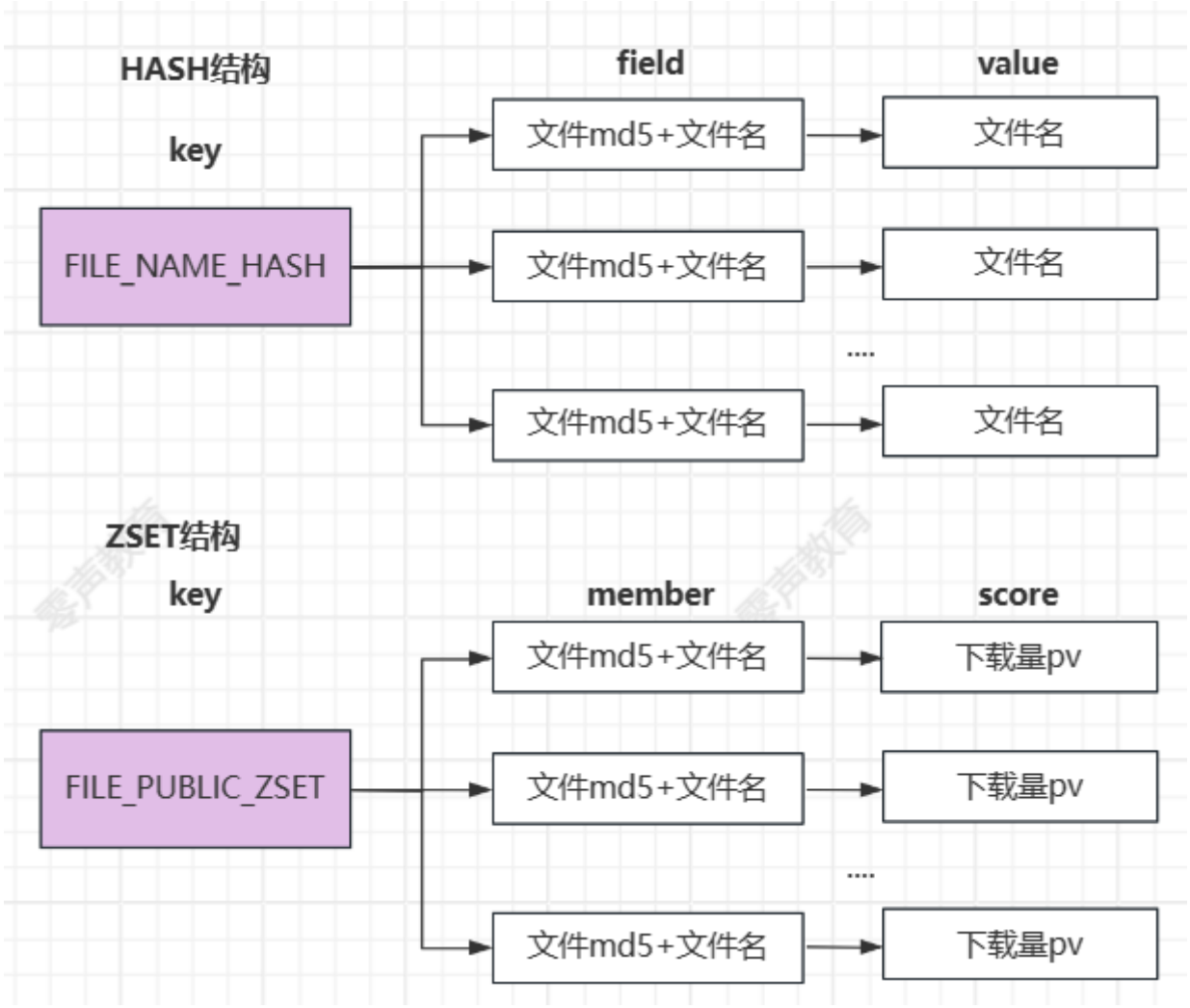
redis-cli显示中文乱码问题，加上--raw方式启动

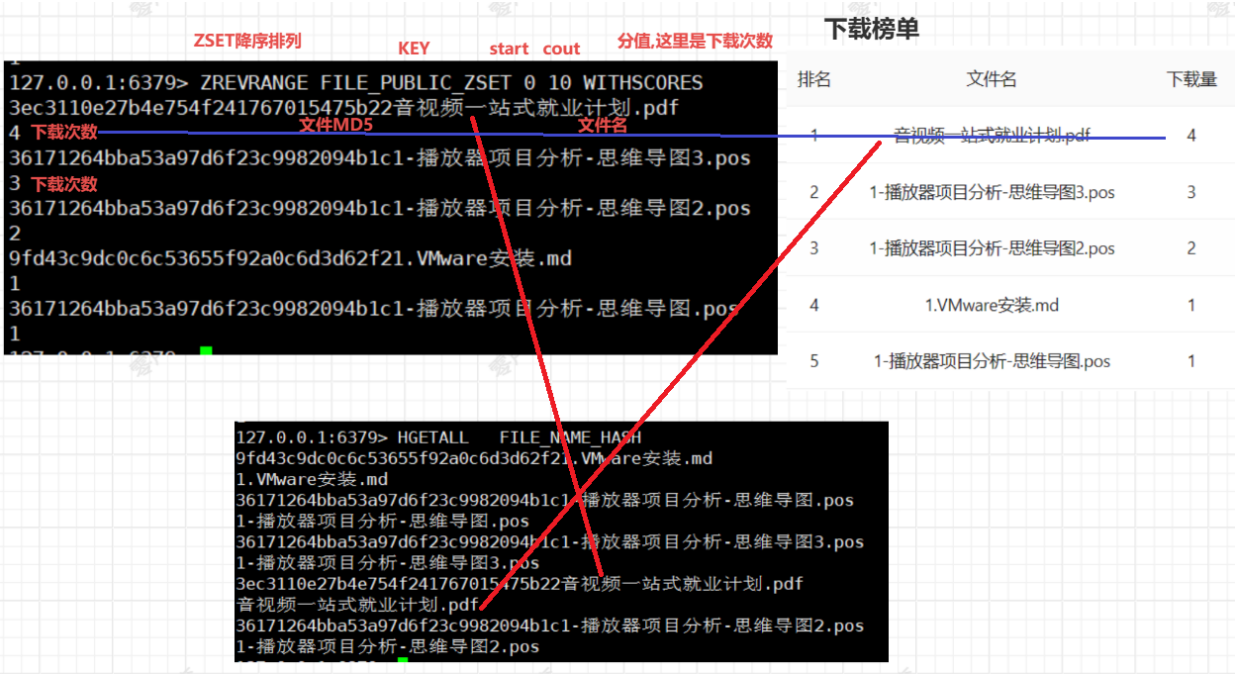
redis-cli --raw

5.1 下载排行榜实现

api_sharefiles.cc 具体处理函数handleGetRankingFilelist

对应api：/api/sharefiles?cmd=pvdesc





ZREVRANGE FILE_PUBLIC_ZSET 0 10 WITHSCORES

HGETALL FILE_NAME_HASH

5.2 图片分享功能

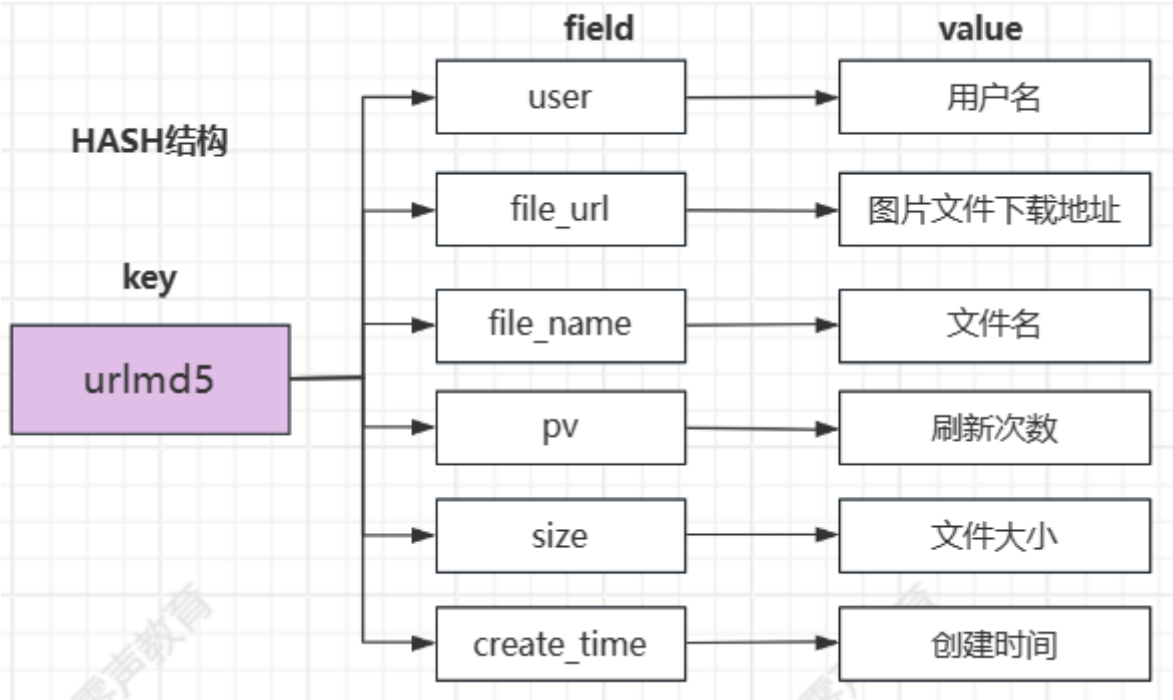
api_sharepicture2.cc 具体处理函数

- 分享图片：handleSharePicture
- 浏览图片：handleBrowsePicture

分享图片

api_sharepicture2.cc：handleSharePicture

1. 根据文件内容的MD5从file_info表获取图片文件的完整下载路径；
2. 创建一条分享图片的记录，操作share_picture_list2表（和第四节课有区别，share_picture_list2直接存储图片文件的下载路径）
3. 将图片分享信息插入到redis， key为urlmd5，value使用hash结构，存储user、file_url、file_name、pv、size、create_time等。



浏览图片

api_sharepicture2.cc：handleBrowsePicture

1. 根据urlmd5从redis读取hash结构，获取user、file_url、file_name、pv、size、create_time等字段；
2. 如果redis没有则从msyql读取
    1. 如果mysql的share_picture_list2表也查询不到urlmd5对应的记录说明该分享链接已经失效了；
    2. 如果msyqll的share_picture_list2表能查到记录，则更新到redis

3. redis存储的pv值实时更新，msyql的share_picture_list2的pv每隔5个计数刷新一次，这么做的目的是减少mysql的操作，即使从mysql重新加载查询到的pv值和实际的pv有个最大为5的误差也没有关系，因为这里只是一个阅读量，有点误差没有关系。
4. 把查询到的结果返回给客户端

取消分享时和我的分享图片

取消分享时：记得删除urlmd5对应的redis缓存，目前代码里没有实现，建议大家自己尝试添加

我的分享图片：拉取我的分享图片是否一定要从mysql加载，可以考虑增加一个zset结构或者hash结构存储用户名对应的urlmd5，具体大家思考下，增加对zset、hash结构的理解。

5.3 其他功能讲解

具体参考第一节课的课件：课程资料/1/1-2-零声图床架构和功能分析v1.1.pdf · master · 2404_vip / 9.2-tuchuang · GitLab (0voice.com)