

# fastdfs存储原理、集群部署和同步机制分析

---

重点内容：

1. 小文件存储的应用场景、难点和解决方法
2. fastdfs存储原理和负载均衡方法
3. fastdfs集群部署
4. fastdfs同步机制分析

## 1 海量小文件存储

重点内容

- 什么是海量小文件存储，多小为小，多少文件为海量
- 为什么要针对海量小文件存储做优化
- 小文件如何存储-存储结构
- 小文件如何查找-查找的条件
- 小文件如何删除-删除后文件是否需要移动，如果文件不移动，会不会造成碎片空间
- 小文件如何更新-能否更新

### 1.1 为什么需要小文件存储

---

#### 1.1.1 小文件应用场景

通常我们认为大小在**1MB以内的文件称为小文件**，百万级数量及以上称为海量，由此量化定义海量小文件问题。如社交网站、电子商务、广电、网络视频、高性能计算，这里举几个典型应用场景。

- 著名的社交网站Facebook存储了600亿张以上的图片，推出了专门针对海量小图片定制优化的Haystack进行存储。
- 淘宝目前应该是最大C2C电子商务网站，存储超过200亿张图片，平均大小仅为15KB，也推出了针对小文件优化的TFS文件系统存储这些图片，并且进行了开源。
- 歌华有线可以进行图书和视频的在线点播，图书每页会扫描成一个几十KB大小的图片，总图片数量能够超过20亿；视频会由切片服务器根据视频码流切割成1MB左右的分片文件，100个频道一个星期的点播量，分片文件数量可达到1000万量级。
- 动漫渲染和影视后期制作应用，会使用大量的视频、音频、图像、纹理等原理素材，一部普通的动画电影可能包含超过500万的小文件，平均大小在10-20KB之间。
- 金融票据影像，需要对大量原始票据进行扫描形成图片和描述信息文件，单个文件大小为几KB至几百KB的不等，文件数量达到数千万乃至数亿，并且逐年增长。

应用范例： vivo FastDFS 海量小文件存储解决之道[https://mp.weixin.qq.com/s/Tk\\_H-ofrS5\\_kLwT1DZsxyA](https://mp.weixin.qq.com/s/Tk_H-ofrS5_kLwT1DZsxyA)

### 1.1.2 小文件存储带来的问题

Linux通过node存储文件信息，但inode也会消耗硬盘空间，所以硬盘格式化的时候，操作系统自动将硬盘分成两个区域。

1. 一个是数据区，存放文件数据；
2. 另一个是inode区（inode table），存放inode所包含的信息。

每个inode节点的大小，一般是128字节或256字节。inode节点的总数，在格式化时就给定，一般是每1KB或每2KB就设置一个inode。假定在一块1GB的硬盘中，每个inode节点的大小为128字节，每1KB就设置一个inode，那么inode table的大小就会达到128MB，占整块硬盘的12.8%。

小文件主要有2个问题：

1. 如果小文件都小于<1KB，而系统初始化的时候每个2K设置一个node，则此时一个文件还是至少占用2K的空间，最后导致磁盘空间利用率不高，< 50%。
2. 大量的小文件，导致在增加、查找、删除文件的时候需要遍历过多的node节点，影响效率。

## 1.2 小文件机制配置

合并文件存储相关的配置都在tracker.conf中。配置完成后，重启tracker和storage server。

支持小文件存储，只需要设置tracker的use\_trunk\_file=true，store\_server=1，其他保持默认即可，但也要注意slot\_max\_size 的大小，这样才知道多大的文件触发小文件存储机制。

#是否启用trunk存储，缺省false，需要打开配置

use\_trunk\_file = true

#trunk文件最小分配单元 字节，缺省256，即使上传的文件只有10字节，也会分配这么多空间。

slot\_min\_size = 256

#trunk内部存储的最大文件，超过该值会被独立存储，缺省16M，超过这个size的文件，不会存储到trunk file中，而是作为一个单独的文件直接存储到文件系统中。

slot\_max\_size = 1MB

#trunk文件大小，默认 64MB，不要配置得过大或者过小，最好不要超过256MB。

trunk\_file\_size = 64MB

## 1.3 合并存储文件命名与文件

向FastDFS上传文件成功时，服务器返回该文件的存取ID叫做fileid：

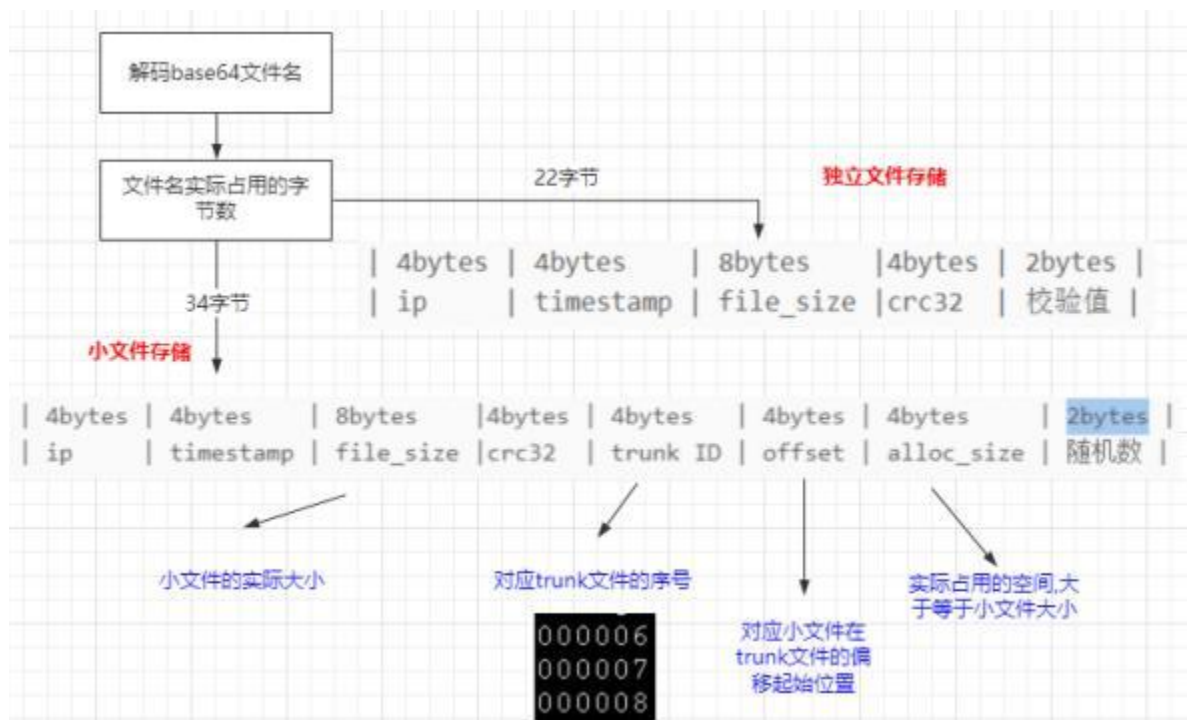
- 当没有启动合并存储时该fileid和磁盘上实际存储的文件一一对应
- 当采用合并存储时就不再一一对应而是多个fileid对应的文件被存储成一个大文件。

注：下面将采用合并存储后的大文件统称为**Trunk文件**，没有合并存储的文件统称为**源文件**；

请大家注意区分三个概念：

- 1) Trunk文件： storage服务器磁盘上存储的实际文件，默认大小为64MB
  - 2) 合并存储文件的FileId：表示服务器启用合并存储后，每次上传返回给客户端的FileId，注意此时该FileId与磁盘上的文件没有一一对应关系；
  - 3) 没有合并存储的FileId：表示服务器未启用合并存储时， Upload时返回的FileID
- Trunk文件文件名格式： fdfs\_storage1/data/00/00/**000001** 文件名从1开始递增，类型为int。

### 1.3.1 启动小文件存储时服务返回给客户端的fileid有变化



#### 1. 独立文件存储的file id

文件名（不含后缀名）采用Base64编码，包含如下5个字段（每个字段均为4字节整数）：

group1/M00/00/00/wKgqHV4OQQyAbo9YAAAA\_fdSpmg855.txt

这个文件名中，除了.txt为文件后缀， wKgqHV4OQQyAbo9YAAAA\_fdSpmg855 这部分是一个base64编码缓冲区，组成如下：

- 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)

wKgqHV4OQQyAbo9YAAAA\_fdSpmg855

4bytes	4bytes	8bytes	4bytes	2bytes
ip	timestamp	file_size	crc32	随机数

2 小文件存储的file id

如果采用了合并存储，生成的文件ID将变长，文件名后面多了base64文本长度16字符（12个字节）。这部分同样采用Base64编码，包含如下几个字段：

group1/M00/00/00/eBuDxWCwrDqITi98AAAA-3Qtcs8AAAAAQAAAgAAAAIA833.txt

采用合并的文件ID更长，因为其中需要加入保存的大文件id以及偏移量，具体包括了如下信息：

- 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: 实际的文件大小
- crc32: 文件内容的crc32码
- trunk file ID: 大文件ID如000001
- offset: 文件内容在trunk文件中的偏移量
- alloc\_size: 分配空间，大于或等于文件大小
- 随机数 （引入随机数的目的是防止生成重名文件

eBuDxWCwrDqITi98AAAA-3Qtcs8AAAAAQAAAgAAAAIA833

| 4bytes | 4bytes | 8bytes | 4bytes | 4bytes | 4bytes | 4bytes

2bytes |

| ip | timestamp | file\_size | crc32 | trunk ID | offset | alloc\_size | 随机数

|

1.3.2 Trunk文件存储结构

1.3.2.1 磁盘数据内部结构

trunk内部是由多个小文件组成，每个小文件都会有一个trunkHeader，以及紧跟在其后的真实数据，结构如下：

|||-----24bytes-----|||

-1byte -|- 4bytes -|- 4bytes -|-4bytes-|-4bytes-|-----7bytes-----|

-filetype-|-alloc\_size-|-filesize-|-crc32 -|-mtime -|-formatted\_ext\_name-|

|||-----file\_data filesize bytes-----|||

-----file\_data-----|

Trunk文件为64MB（默认），因此每次创建一次Trunk文件总是会产生空余空间，比如为存储一个10MB文件，创建一个Trunk文件，那么就会剩下接近54MB的空间（TrunkHeader 会24字节，后面为了方便叙述暂时忽略其所占空间），下次要想再次存储10MB文件时就不需要创建新的文件，存储在已经创建的Trunk文件中即可。另外当删除一个存储的文件时，也会产生空余空间。

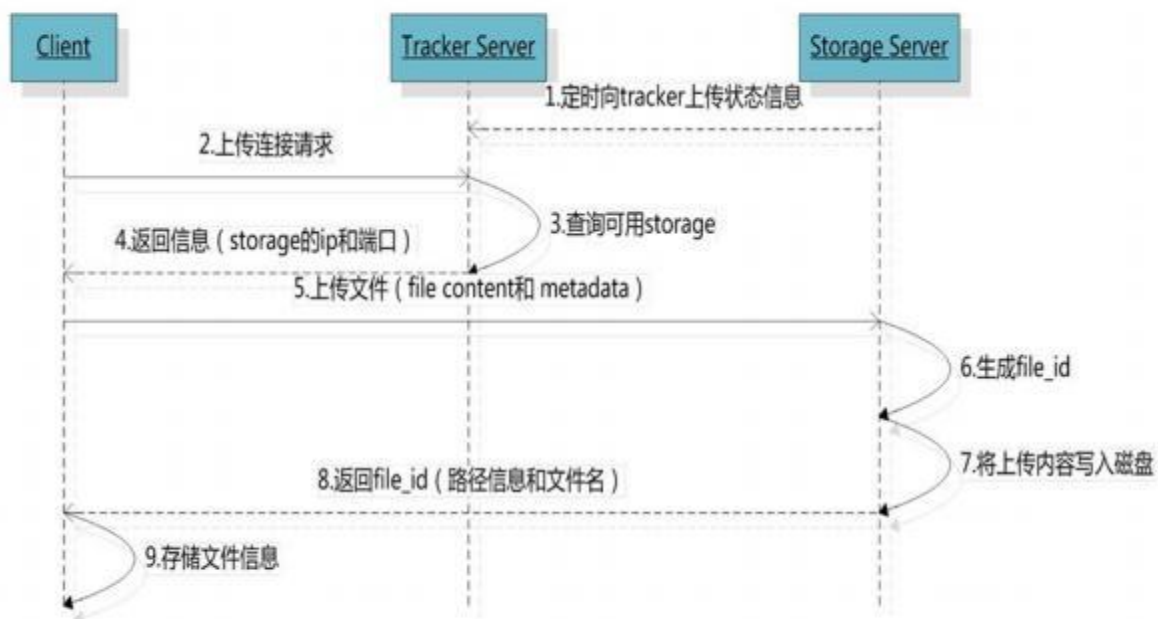
即是：每次创建一个trunk文件时，是安装既定的文件大小去创建该文件trunk\_file\_size，当要删除trunk文件时，需要该trunk文件没有存储小文件才能删除。

### 1.3.2.2 小文件存储平衡树

在Storage内部会为每个store\_path构造一颗以空闲块大小作为关键字的空闲平衡树，相同大小的空闲块保存在链表之中。每当需要存储一个文件时会首先到空闲平衡树中查找大于并且最接近的空闲块，然后试着从该空闲块中分割出多余的部分作为一个新的空闲块，加入到空闲平衡树中。例如：

- 要求存储文件为300KB，通过空闲平衡树找到一个350KB的空闲块，那么就会将350KB的空闲块分裂成两块，前面300KB返回用于存储，后面50KB则继续放置到空闲平衡树之中。
- 假若此时找不到可满足的空闲块，那么就会创建一个新的trunk文件64MB，将其加入到空闲平衡树之中，再次执行上面的查找操作（此时总是能够满足了）。
- 进一步阅读： `storage\trunk_mgr\trunk_mem.c` 搜索 `avl_tree_insert`、`avl_tree_find`等函数。

## 2 文件上传原理和负载均衡方法



### 2.1 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。 --》高可用，通过冗余的方式提供服务。

### 2.2 选择存储的group

(注意不同的负载均衡算法)

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

## 2.3 选择storage server

---

(注意不同的负载均衡算法)

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）

## 2.4 选择storage path

---

(注意不同的负载均衡算法)

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

## 2.5 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
- 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。

### 2.5.1 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。

一个目录存储100百万文件

$65,536 = 15.2 \text{ 文件}$

### 2.5.2 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。


  
 group1/M00/00/00/eBuDxWCeIFCAEFUrAAAAKTIQHvk462.txt

文件名规则：

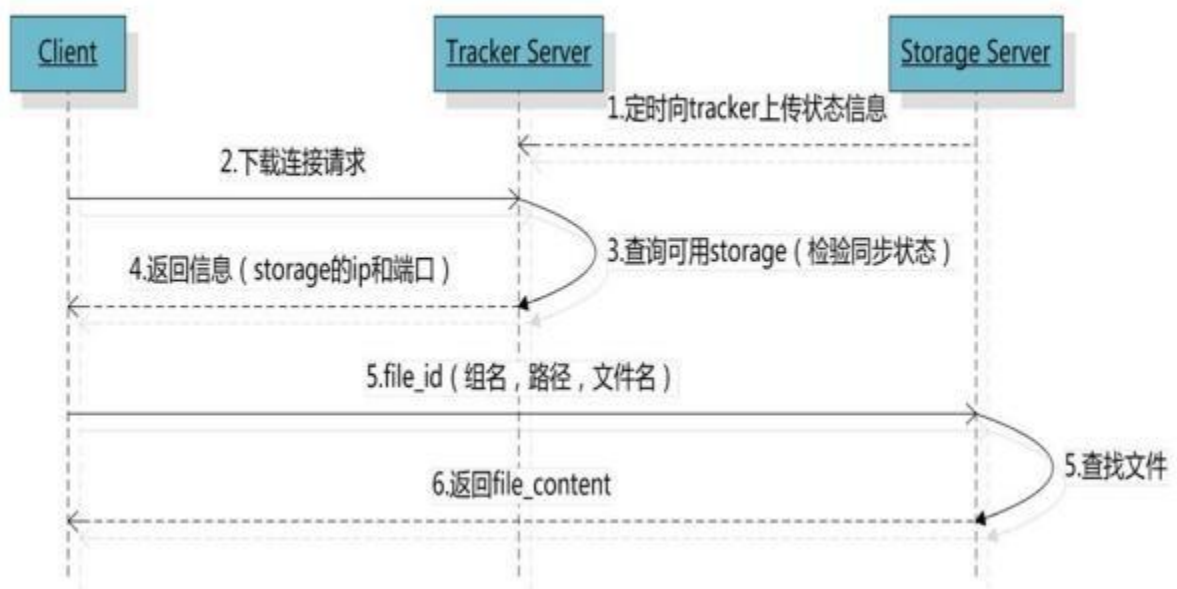
- storage\_id (ip的数值型) 源storage server ID或IP地址
- timestamp (文件创建时间戳)
- file\_size (若原始值为32位则前面加入一个随机值填充，最终为64位)
- crc32 (文件内容的检验码)

随机数 (引入随机数的目的是防止生成重名文件)

```
eBuDxWcb2qmAQ89yAAAAKeR1iIo162
| 4bytes | 4bytes | 8bytes | 4bytes | 2bytes |
| ip     | timestamp | file_size | crc32  | 校验值  |
```

### 3 下载文件逻辑

客户端upload file成功后，会拿到一个storage生成的文件名，接下来客户端根据这个文件名即可访问到该文件。



跟upload file一样，在download file时客户端可以选择任意tracker server。

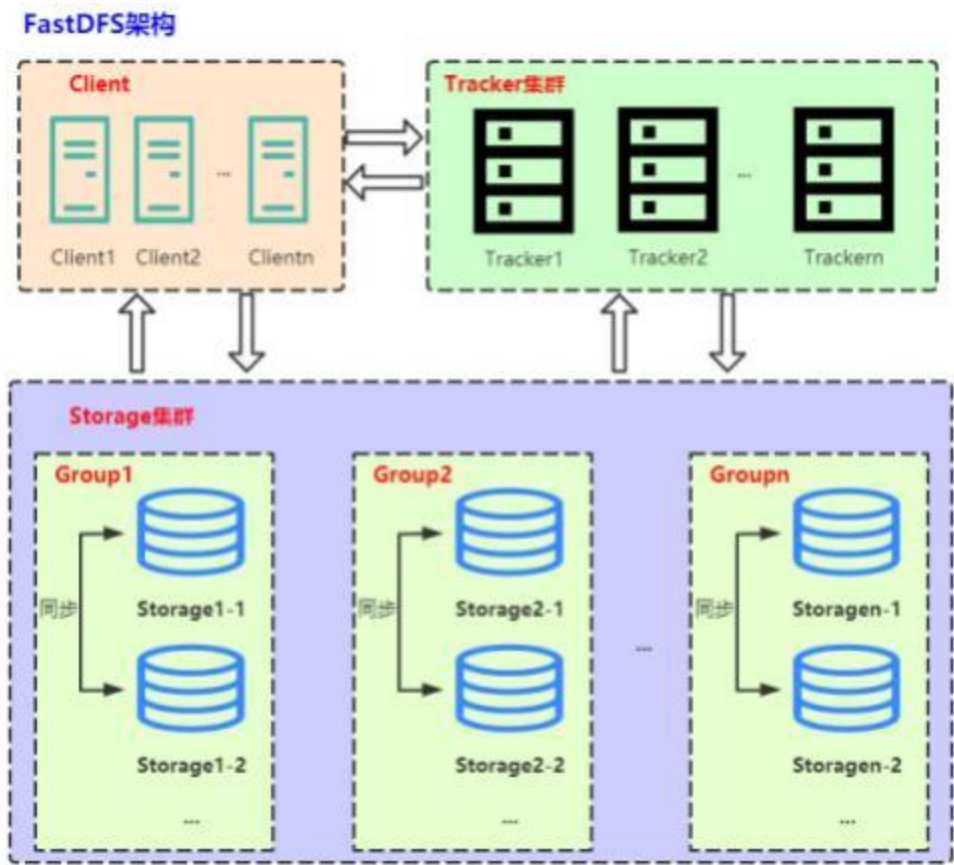
tracker发送download请求给某个tracker，必须带上文件名信息，tracker从文件名中解析出文件的group、大小、创建时间等信息，然后为该请求选择一个storage用来服务读请求。



由于group内的文件同步时在后台异步进行的，所以有可能出现在读到时候，文件还没有同步到某些storage server上，为了尽量避免访问到这样的storage，tracker按照如下规则选择group内可读的storage。

- 1. 该文件上传到的源头storage，源头storage只要存活着，肯定包含这个文件，源头的地址（IP）被编码在文件名中。
- 2. 文件创建时间戳==storage被同步到的时间戳 且(当前时间-文件创建时间戳) > 文件同步最大时间（如5分钟）。文件创建后，认为经过最大同步时间后，肯定已经同步到其他storage了。
- 3. 文件创建时间戳 < storage被同步到的时间戳。 - 同步时间戳之前的文件确定已经同步了
- 4. (当前时间-文件创建时间戳) > 同步延迟阈值（如一天）。 经过同步延迟阈值时间，认为文件肯定已经同步了。

## 4 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)	
120.27.131.198	fdfs_trackerd	tracker_22122.conf <b>22122</b>	
120.27.131.198	fdfs_trackerd	tracker_22123.conf <b>22123</b>	



服务器地址	服务程序	对应配置文件(端口区分)	
120.27.131.198	<b>fdfs_storaged</b>	storage_group1_23000.conf	
114.215.169.67	<b>fdfs_storaged</b>	storage_group1_23000.conf	

**storage**是要部署在不同的服务器上，改端口是没有用。

## 4.1 120.27.131.198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**

```
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

### 4.1.1 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

```
# 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

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

### 4.1.2 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

```
# 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

```
/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
```

## 4.1.3 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

```
# 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=120.27.131.198:22122
tracker_server=120.27.131.198:22123
```

主要修改： port、base\_path、store\_path0、tracker\_server

启动storage\_group1\_23000

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

## 4.2 114.215.169.67服务器

进入

```
cd /etc/fdfs
cp storage.conf.sample storage_group1_23000.conf
mkdir /home/fastdfs/storage_group1_23000
```

把现有的tracker、storage全部停止

```
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

## 4.2.1 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

```
# 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=120.27.131.198:22122
tracker_server=120.27.131.198:22123
```

主要修改： port、base\_path、store\_path0、tracker\_server

把现有的storage先停止服务。

启动storage\_group1\_23000

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

## 4.3 测试

### 4.3.1 配置client.conf

创建client目录： mkdir /home/fastdfs/client

修改client.conf

```
# 修改client的base path路径
base_path = /home/fastdfs/client
# 配置tracker server地址
tracker_server=120.27.131.198:22122
tracker_server=120.27.131.198:22123
```

### 4.3.2 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 = 120.27.131.198:22122

tracker\_server=120.27.131.198:22123 #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=120.27.131.198:22122
tracker_server=120.27.131.198:22123
# 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
```

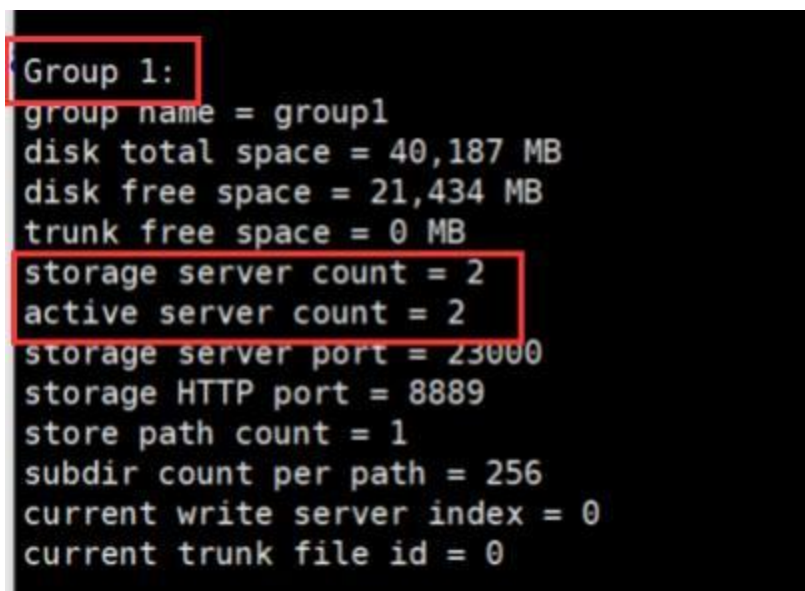
主要修改tracker\_server、url\_have\_group\_name、store\_path0。

### 4.3.4 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：



```
Group 1:
group name = group1
disk total space = 40,187 MB
disk free space = 21,434 MB
trunk free space = 0 MB
storage server count = 2
active server count = 2
storage server port = 23000
storage HTTP port = 8889
store path count = 1
subdir count per path = 256
current write server index = 0
current trunk file id = 0
```

存在2个Active的storage。

### 4.3.4 测试上传文件

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

返回 group1/M00/00/00/eBuDxWlGcYqACM8LAAAOaTcAqLc40.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。

### 4.3.5 下载测试

(1) 正常下载

```
fdfs_download_file /etc/fdfs/client.conf  
group1/M00/00/00/ctepQmCotjqAIRNPAAjuZXPuAg28.conf
```

(2) 停止120.27.131.198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/ctepQmCotjqAIRNPAAjuZXPuAg28.conf
```

此时还可以正常下载数据

(3) 继续停止另一个storage server(114.215.169.67)

```
/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/ctepQmCotjqAIRNPAAjuZXPuAg28.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/eBuDxWlGcYqACM8LAAAOaTcAqLc40.conf[2021-05-22 15:49:51] ERROR - file:  
tracker_proto.c, line: 50, server: 120.27.131.198:22122, response status 2 != 0  
  
[2021-05-22 15:49:51] ERROR - file: ../client/tracker_client.c, line: 716, fdfs_rcv_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://120.27.131.198:80/group1/M00/00/00/eBuDxWlGcYqACM8LAA AoaTcAqLc40.conf>

### 4.3.6 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。

### 4.3.7 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%
```

## 4.4 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?#>

## 5 tracker和storage目录结构

### 5.1 tracker server目录及文件结构



```
/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 114.215.169.67 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:

```
# 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

---

```
# storage 120.27.131.198:23000
[Storage001]
  group_name=group1
  ip_addr=120.27.131.198
  status=7
  version=6.07
  join_time=1646292828
....
# storage 114.215.169.67:23000
[Storage002]
  group_name=group1
  ip_addr=114.215.169.67
  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）

## 5.2 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
|    |    |    114.215.169.67_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
|    |___.storage.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。

## 6 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，就记录近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
```

## 6.1 同步日志所在目录

---

比如在**120.27.131.198**服务器看到的sync log:

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

**114.215.169.67\_23000.mark** 同步状态文件，记录本机到114.215.169.67 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**114.215.169.68**，则该目录还存在  
114.215.169.68\_23000.mark

比如在**114.215.169.67**服务器看到的sync log:

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

**120.27.131.198\_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/oYYBAF285cOIHiVCAACI-7zX1qUAAAAVgAACCC8AAIkT490.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：

#### **114.215.169.67\_23000.mark**

```
binlog_index=0      //binlog索引id 表示上次同步给114.215.169.67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114.215.169.67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
```

#### **120.27.131.198\_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Storage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。

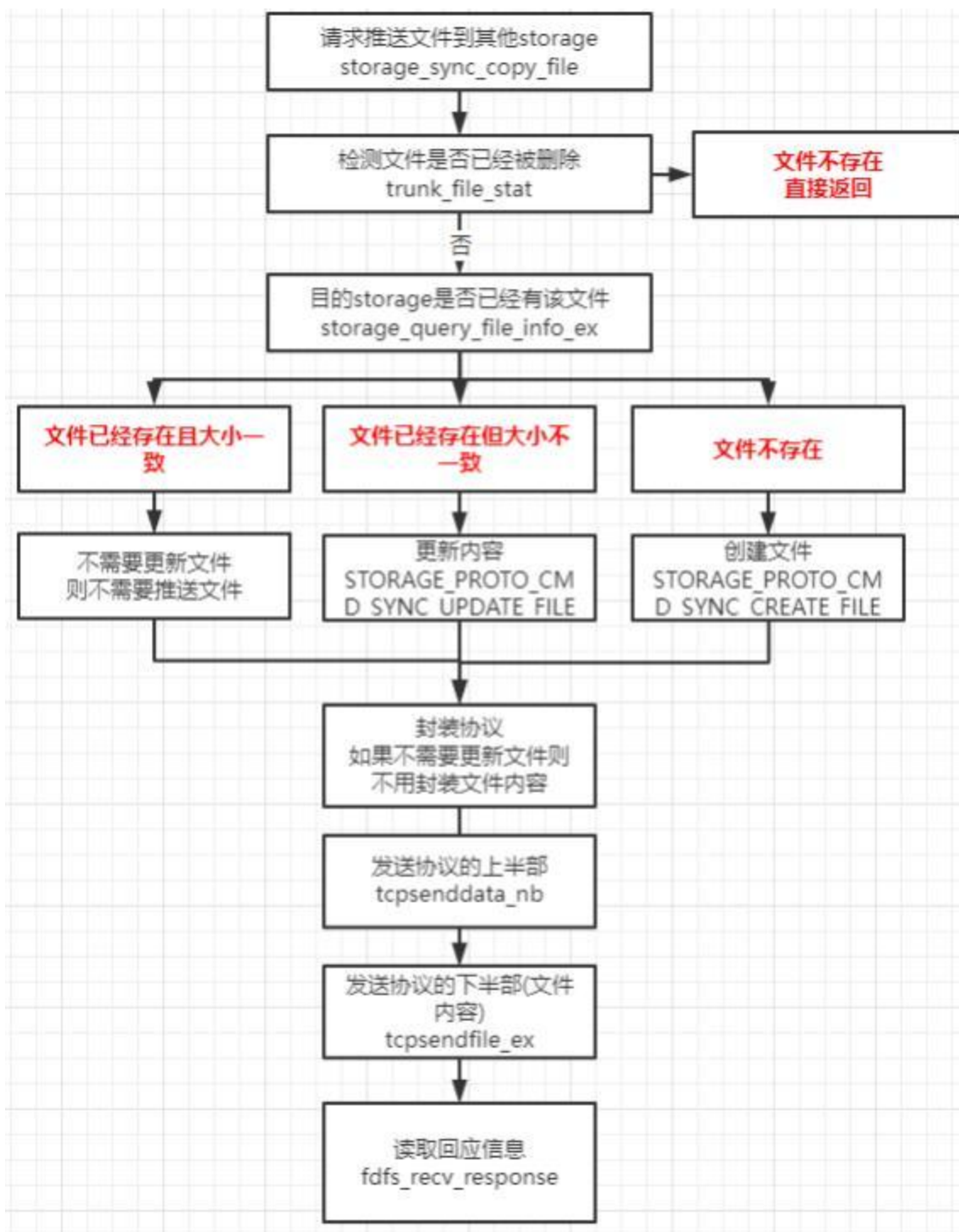
## 2 同步线程执行过程storage\_sync\_thread\_entrance

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114.215.169.67的同步则打开114.215.169.67\_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114.215.169.67\_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。

## 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（Binlog中的第一个字段timestamp，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。为什么呢？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：

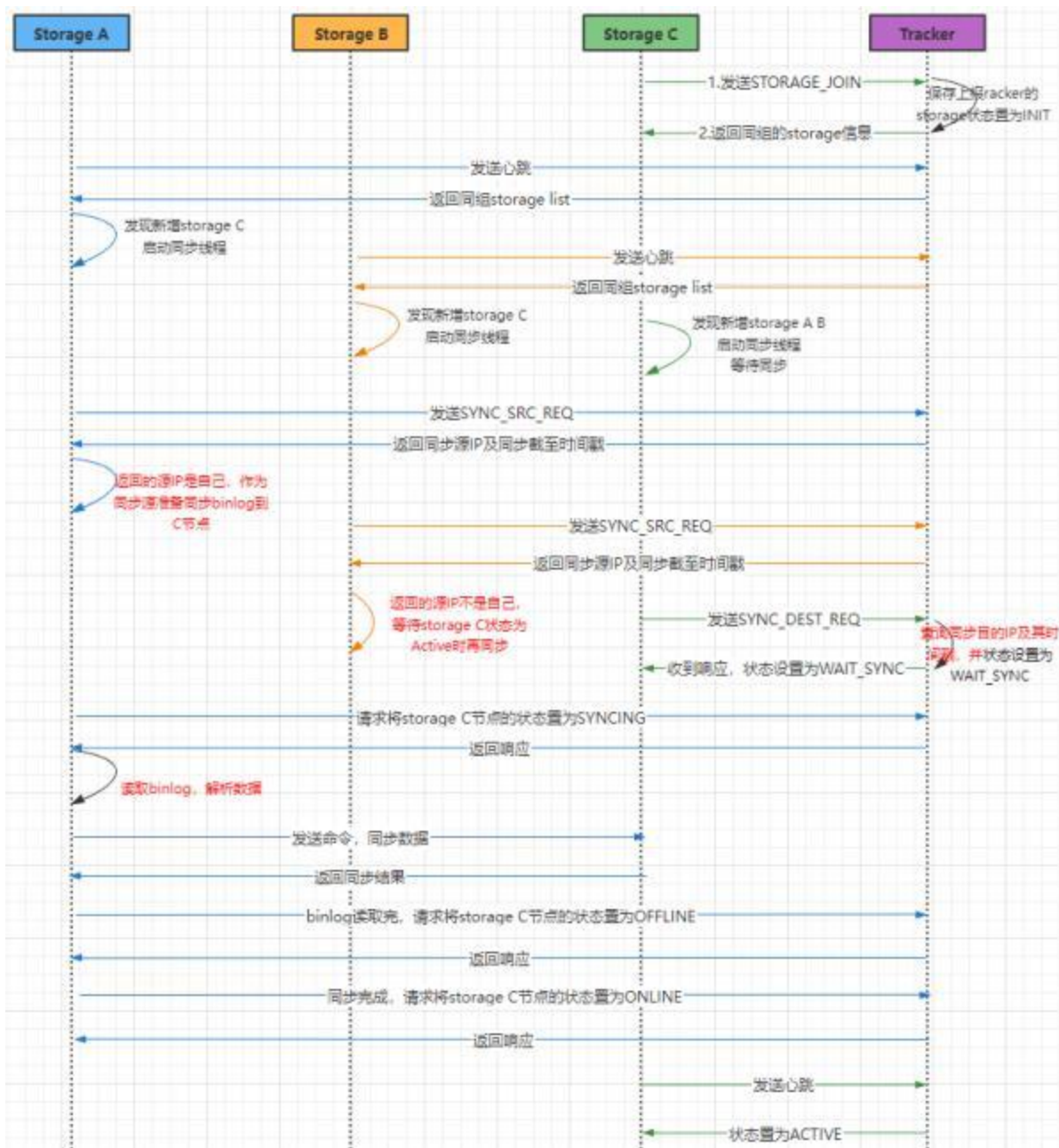
```
group1,10.0.0.1,    0,          1408524351 , 1408524352
group1,10.0.0.2,    1408524353 , 0,          1408524354
group1,10.0.0.3,    1408524355 , 1408524356 , 0
```

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。

## 6.6 新增节点同步流程

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/CtAqWVjTbm2AIqTkaaCd\_nIZ7M797.jpg

1490251373 表示时间戳

C 表示操作类型

M02/52/CB/CtAqWVjTbm2AIqTkaaCd\_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Storage C，Storage C 收数据并保存
4. binlog文件读完之后，会将Storage 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heartbeat时(比如间隔30秒)**，tracker server将其状态更改为FDFS\_STORAGE\_STATUS\_ACTIVE。

## 6.7 Tracker选择客户端下载文件的storage的原则

- a)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)
- b)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。
- c)在最小同步时间点之后的文件，选择源storage提供给客户端。

## 7 断点续传

可以参考fdfs\_append\_file 命令，本质而言，断点续传实际就是指定文件要上传的位置