网盘项目技术手册

目录

[网盘项目技术手册 1](#_Toc107611181)

[概述 2](#_Toc107611182)

[数据库设计 3](#_Toc107611183)

[文件上传 5](#_Toc107611184)

[服务端 5](#_Toc107611185)

[API 5](#_Toc107611186)

[客户端 6](#_Toc107611187)

[client完整上传流程 6](#_Toc107611188)

[细节 6](#_Toc107611189)

[文件下载 7](#_Toc107611190)

[服务端 7](#_Toc107611191)

[API 7](#_Toc107611192)

[客户端 7](#_Toc107611193)

[client完整下载流程 7](#_Toc107611194)

[细节 7](#_Toc107611195)

[文件复制、移动、改名、删除等操作 8](#_Toc107611196)

[link 8](#_Toc107611197)

[每个用户的目录结构设计 8](#_Toc107611198)

[待改进之处 9](#_Toc107611199)

# 概述

所有用户上传的文件，服务端不再存储其完整文件数据，而是存储元数据(文件碎块，大小不超过MAX\_FRAGMENT\_SIZE)。将元数据块全部存储在pool文件夹中，文件名为其MD5码。暂定MAX\_FRAGMENT\_SIZE = 4M.pool文件夹结构如图：

文本

中度可信度描述已自动生成

上传文件时，将文件分为若干个大小不超过MAX\_FRAGMENT\_SIZE的碎片，每次先计算碎片的MD5码，向服务器发出get请求，看看服务器里是否已存在这个碎片。若存在则不发送文件碎片，否则发送。服务器收到碎片后，维护数据库表项，并将碎片存储在pool文件夹。

秒传的实现：上传时先计算整个文件的MD5码，查询服务端该MD5码对应的文件是否存在，若存在直接秒传，否则进入正常上传流程。

下载时，client先查看下载目录是否有同名文件，若有则计算同名文件MD5码，并与待下载文件的MD5码比对，若相同，则秒下。否则向服务端申请文件碎块内容，然后在指定临时目录新建一个名字为fdid的文件夹。下载时，每次只下载相应碎块，客户端post对应的文件fdid，要下载的块序号，服务端将文件对应的碎片从数据库表项中找到，然后逐块发送给client，client端拿到碎块后更新文件夹内容。全部传输完毕后组合形成最终文件，校验最终文件的MD5码即可。

文件复制、移动、改名等操作，均只添加、修改相应数据库表项。删除操作同时删除pool里的文件碎块

如此，便可以实现所有基本要求。

# 数据库设计

文件碎片实体表 FileFragmentEntity

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| MD5码 | MD5 | VARCHAR | NOT NULL |
| 碎片大小 | fgsize | INT | NOT NULL |
| 被链接的个数 | link\_num | INT | NOT NULL |

备注：

文件上传、复制、修改、删除都可能会改变link\_num !!!

link\_num减少至0时可以从pool中删除

目录实体表DirectoryEntity

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| 目录名 | dname | VARCHAR | NOT NULL |
| 父目录id | parent\_id | INT | NOT NULL |
| 修改时间 | last\_change\_time | TIMESTAMP | NOT NULL |

额外的约束：

目录名dname不能含有“/”等字符，参考：

[Linux文件（目录）命名规则 (biancheng.net)](http://c.biancheng.net/view/3608.html)

备注：

用户根目录的did可以是这个表项的id，也可以是NULL，待商议

文件实体表FileEntity

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| MD5码 | MD5 | VARCHAR | NOT NULL |
| 文件大小 | fsize | INT | NOT NULL |
| 被链接的个数 | link\_num | INT | NOT NULL |
| 修改时间 | last\_change\_time | TIMESTAMP | NOT NULL |
| 下个分配的块号 | next\_index | INT | NOT NULL |
| 文件实体是否完整 | is\_complete | BOOLEAN | NOT NULL |

备注：

文件上传、复制、修改、删除都会改变link\_num !!! 同时也会改变元数据的link\_num.

link\_num减少至0时从数据库表项中删除

文件碎片映射表FileFragmentMap

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| 文件编号 | fid | INT | foreign key |
| 块索引(文件的第几块) | index | INT | UNIQUE, NOT NULL |
| 文件碎片编号 | fgid | INT | foreign key |

额外的约束：

1. index从0开始编号，一直编号到文件的最后一块，即
2. (fid,index,fgid)三元组保证UNIQUE（或者把它们变为多重主键）

文件目录映射表FileDirectoryMap

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| 文件编号 | fid | INT | foreign key |
| 目录编号 | did | INT | foreign key |
| 文件名称 | fname | VARCHAR | NOT NULL |
| 修改时间 | last\_change\_time | TIMESTAMP | NOT NULL |

额外的约束：

1. 文件名fname不能含有“/”等字符，参考：

[Linux文件（目录）命名规则 (biancheng.net)](http://c.biancheng.net/view/3608.html)

1. (fid,did,fname)三元组保证UNIQUE（或者把它们变为多重主键）
2. did是DirectoryEntity的外键，自相连

用户实体表UserEntity

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| 用户名 | user\_name | VARCHAR | UNIQUE, NOT NULL |
| 密码的MD5 | password\_hash | VARCHAR | NOT NULL |
| 根目录ID | root\_dir\_id | INT | foreign key |

备注：

根目录ID是DirectoryEntity的外键

用户登录日志表UserLogin

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 字段 | 类型 | 约束 |
| 编号 | id | INT | primary key |
| 用户id | user\_id | VARCHAR | foreign key |
| 登录ip | login\_ip | VARCHAR | NOT NULL |
| 登录时间 | login\_time | TIMESTAMP | NOT NULL |

# 文件上传

client先计算文件的MD5码，然后GET / api/file\_exist?md5=文件的MD5码，服务器返回true则不用上传此文件，server端修改相应数据库表项即可。

否则client将文件分为若干个大小不超过MAX\_FRAGMENT\_SIZE的碎片，每次先向server申请要上传的块号index，然后计算index对应块的MD5，查看文件碎片是否存在是否(fragment\_exist)，服务器返回true则不发此碎片，否则发送此碎片。无论服务器是否收到碎片，均维护数据库表项，并将碎片存储在pool文件夹。

## 服务端

### API

服务端主要维护这几个API:

file\_exist API：

用完整文件的MD5码，查文件实体表，看对应表项是否存在，返回结果

upload\_allocation API：

1. 传入完整文件的MD5码，用MD5码查文件实体表
   1. 若不存在，创建对应表项，初始next\_index为0，is\_complete为false
   2. 若存在且is\_complete为true，返回next\_index=-1，代表所有碎块已上传
2. 将next\_index返回给客户端，继续下面的流程
3. next\_index变为下一个未上传的碎片块号，若next\_index达到文件index上限，即，用完整文件MD5码查文件碎片映射表，看0~max\_index-1 号碎片是否都已上传
   1. 已全部上传，is\_complete变为true，next\_index变为-1
   2. 未全部上传，next\_index变为未上传的块号中最小者

fragment\_exist API：

用文件碎片的MD5码，查文件碎片实体表，看对应表项是否存在，返回结果

upload\_fragment API：

1. 用文件碎片的MD5码，查文件碎片实体表，看对应表项是否存在
   1. 当client传入file\_fragment参数时，

1) 若文件碎片实体表中不存在相应碎片，增加文件碎片实体，初始link\_num为0，文件存储到fragment\_pool；

2) 若文件碎片实体表中已存在相应碎片(两个用户竞争上传的情形)，跳到第2步

* 1. 若client不传入file\_fragment参数（这个碎片已存在），跳到第2步

1. 若映射表中无相应表项，修改文件碎片映射表，建立文件实体和文件碎片实体的链接；修改文件碎片实体表，link\_num++

upload\_file API：

1. 用完整文件的MD5码查文件实体表，判断is\_complete值
   1. true，修改文件目录映射表，增加表项
   2. false，返回错误

## 客户端

### client完整上传流程

1. 上传完整文件MD5码(file\_exist)，查看是否存在，若存在，跳至第5步
2. upload\_allocation，获取要上传的块号index
   1. 若返回所有碎块上传完毕(index为-1)，跳至第5步
3. 计算index对应块的MD5，查看文件碎片是否存在是否(fragment\_exist)
   1. 若文件碎片存在，调用(upload\_fragment)且不需上传具体文件碎块，只需上传文件碎片MD5码
   2. 若文件碎片不存在，调用(upload\_fragment)但需上传具体文件碎块
4. 返回第2步
5. 调用upload\_file，更新文件映射表

### 细节

#### 文件名

记得处理带有“/”等特殊字符的文件名，不允许其上传，参考：

[Linux文件（目录）命名规则 (biancheng.net)](http://c.biancheng.net/view/3608.html)

#### 文件目录

如果用户上传的是一个目录，则需要对此目录递归地上传文件，同时要先更新数据库表项中的目录（相当于用户省去了新建目录的一个操作）

# 文件下载

## 服务端

### API

服务端主要维护这个API:

/api/download\_fragment API:

client传入FileDirectoryMap的id和文件索引块序号index，服务端返回对应的二进制流文件碎片内容。

## 客户端

### client完整下载流程

client先计算文件MD5码，查看对应目录是否有相同文件，有则秒下。否则向服务端申请文件碎块内容，然后在指定目录新建一个fdid为名字的文件夹。下载时，每次只下载相应碎块，客户端post对应的文件fdid，要下载的块id，服务端将文件对应的碎片从数据库表项中找到，然后逐块发送给client，client端拿到碎块后更新文件夹内容。全部传输完毕后组合形成最终文件，检查文件的MD5码即可。

### 细节

#### 文件目录

如果用户下载的是一个目录，则需要对此目录递归地创建目录并下载文件（相当于用户省去了新建目录的操作）。若有相同目录，则依次检验目录下方是否有文件无法秒下，然后下载。

# 文件复制、移动、改名、删除等操作

文件复制、移动、改名等操作，均只添加、修改相应数据库表项。删除操作同时删除pool里的文件碎块，具体逻辑不再赘述，可以参考Api\_file.cpp文件和我们相应的API。

本项目所有API可参考:

<https://www.apifox.cn/apidoc/shared-3b6f3060-364e-4cf7-9570-0c23e698c12e>

（链接有效期至2022.8）

## link

需要注意的是，文件实体表的link只与用户实体表有关，所有用户加起来有几个该文件实体，文件实体表的link就是几。而文件碎片实体表的link只与文件实体表有关（不需要乘以文件实体表的link使其与用户实体表相关），所有文件按照我们的规则分块后一共有多少个同md5的碎片，文件碎片实体表的link就是多少。（暂不考虑内容不同，md5相同的情形）显然，如果一个文件实体分块后有两个相同的文件碎片，那么这个文件实体对该文件碎片的link贡献是2.

# 每个用户的目录结构设计

在文件目录映射表中记录了所有目录与文件间的映射关系，在文件目录实体表中记录了所有目录与其根目录间的映射关系，在用户实体表中记录了用户对应的根目录的目录实体表id。

因此，要找每个用户的目录，只需从用户实体表的根目录id出发，查找文件目录映射表得到其所有子文件，查找文件目录实体表得到所有子目录。

# 待改进之处

## Server

Server端尝试模仿Reactor网络编程模型：一个线程负责监听TCP链接，当收到用户的TCP链接后，创建client-fd ，然后从线程池中挑选一个线程，将client -fd添加到工作线程的epoll中，当client-fd收到HTTP请求，由工作线程处理之。

不过最后由于时间原因，只是为每个文件描述符对应的http报文解析和api调用单独开一个工作线程，所有的收发依旧由主线程维护。在这种模式下，每个线程都处理自己的fd，从而完全避免了多线程安全问题。

因此我们也算是部分实现了高并发、高效率的要求。最终单用户下我们的网盘上传速度达到下载150M 文件44s，上传500M 文件1min56s（下载文件速度4.31M/s，上传文件速度3.4M/s）。真正实现了一个不限速网盘。

## Client

依照我们原先的构想，下载文件时，client向服务端get文件信息，包括该文件大小fsize，文件MD5，文件id等信息，然后在指定目录新建一个空文件，大小为 ⌊fsize /MAX\_FRAGMENT\_SIZE⌋ +552字节，命名为“文件名.tmp”其结构如图：



下载时，每次只下载相应碎块，客户端post对应的文件id，要下载的块id，服务端将文件对应的碎片从数据库表项中找到，然后逐块发送给client，client端拿到碎块后更新bitmap（这里我让每个字节对应该文件的一个index）。全部传输完毕后检查文件体的MD5码，然后将文件头删掉，保留文件体并将文件重命名(把.tmp删掉)。

然而，在实际的client端书写时，由于我们使用了python作为开发语言，导致很难灵活地使用文件指针对文件进行修改。例如：通过”ab+”即追加读写二进制文件的打开方式打开文件后，write函数无法随着文件指针的移动而改变写入地址，必须先以只读方式打开一份文件，将文件内容导入内存再以写入方式打开文件修改相应内容。

因此我们放弃了这一技术构想，转而为每个下载队列的文件新开一个文件夹，将碎片存放在这个文件夹中，全部下载完成后再合并为最终文件。