**同济大学计算机系**

**网盘大作业**

**技术手册**

****

**院 系 电子与信息工程学院**

**专 业 计算机科学与技术**

**组 员 1 1952893邬嘉晟**

**组 员 2 1953461高志成**

**组 员 3 1953492李强**

**授课老师 沈坚**

目录

[一、数据库设计 3](#_Toc107677155)

[1.1. 逻辑文件表 3](#_Toc107677156)

[1.2. 物理文件表 3](#_Toc107677157)

[1.3. 用户表 4](#_Toc107677158)

[1.4. 待上传文件表 4](#_Toc107677159)

[1.5. 待上传文件夹表 5](#_Toc107677160)

[1.6. 待下载文件表 6](#_Toc107677161)

[1.7. 待下载文件表 6](#_Toc107677162)

[1.8. 用户登录日志表 7](#_Toc107677163)

[二、文件在server端的存放 7](#_Toc107677164)

[2.1. 文件上传 8](#_Toc107677165)

[2.2. 断点续传 9](#_Toc107677166)

[2.3. 秒传 9](#_Toc107677167)

[三、用户目录结构 10](#_Toc107677168)

[四、文件下载 10](#_Toc107677169)

[4.1. 单文件下载 10](#_Toc107677170)

[4.2. 文件夹下载 11](#_Toc107677171)

[4.3. 秒下 11](#_Toc107677172)

[4.4. 单文件断点续下 11](#_Toc107677173)

[4.5. 文件夹断点续下 11](#_Toc107677174)

[五、项目总结 12](#_Toc107677175)

# 一、数据库设计

本次网盘的作业对数据库进行了充分的设计和研究，使其能够符合我们网盘的使用需要。首先来对数据库的设计进行说明：

## 1.1. 逻辑文件表

逻辑文件表中存储了用户文件和文件夹在网盘上的逻辑结构，其中包含了该文件的id、文件名、文件的md5码、其父目录节点的逻辑id、以及是否为文件的标志位。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| id | int | 文件（夹）的唯一标识符（主键） |
| file\_name | str | 文件名 |
| md5 | str | 文件的md5码，关联到真实文件 |
| parent\_id | int | 指向父目录的id |
| is\_dir | bool | 是否为文件 |

## 1.2. 物理文件表

物理文件表中存储了文件实际在服务器上的存储信息，其中以每个文件自己独特的md5码为主键，还记录了该文件的文件大小和其与用户的目录结构中的逻辑文件的链接数。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| md5 | str | 该文件的md5码，主键 |
| real\_path（后被废弃） | str | 指向文件在服务器上的存储路径 |
| file\_size | int | 文件大小 |
| file\_link\_cnt | int | 逻辑文件链接数 |

## 1.3. 用户表

在用户表中，我们记载了网盘用户的一些信息，来使其能够正常登录使用并正确地访问到其文件目录结构。首先我们给每一个用户分配了一个唯一id，然后我们记载了用户的名字、加密后的密码以及其根目录的逻辑文件id。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| id | int | 用户id，主键 |
| name | str | 用户名 |
| passwd | str | 用户密码md5 |
| root\_id | int | 用户的根目录 |

## 1.4. 待上传文件表

待上传文件表主要服务于用户上传的过程，其中详细地记录了很多信息。每一个待上传（上传中）的文件都有一个上传id，并记录了是哪位用户上传了这个文件。此外表中还登记了该文件在用户主机上的真实文件路径、该文件的md5码，文件的总长度、当前读写指针的位置、其网盘中父节点的逻辑id以及其对应的带上传文件夹表中的文件夹id。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| id | int | 待上传文件的id，主键 |
| u\_id | int | 用户的id |
| file\_real\_path | string | 用户主机上的真实文件路径 |
| md5 | string | 待上传文件的md5验证码 |
| file\_len | int | 文件总长度 |
| file\_pointer | int | 当前读写指针的位置 |
| parent\_id | int | 上传到服务器的哪个地方？ |
| uploading\_dir\_id | int | 该待上传文件对应的待上传文件夹表中的文件夹序号 |

## 1.5. 待上传文件夹表

待上传文件夹表主要用于记录用户上传文件夹时的一些信息。每一项也都会分配一个唯一id，并记录上传这个文件夹的用户id，此外还会记录这个文件夹在用户主机上的真实路径、其在用户的网盘存储结构中的逻辑id、其一共包含多少个文件和当前上传到了第几个文件。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| id | int | 上传文件夹的唯一id |
| u\_id | int | 用户的id |
| file\_real\_path | string | 用户主机上的真实文件路径 |
| logic\_id | int | 本目录的逻辑id |
| file\_now | int | 传到第几个文件 |
| file\_count | int | 一共多少个文件 |

## 1.6. 待下载文件表

待下载文件表和待上传文件表有些许相似，也是记录用户在传输文件中的一些信息。首先我们会为所有的待下载文件分配一个id，也记录了下载这个文件的用户id，随后记录了下载的文件在用户网盘中的逻辑id、文件的总长度、用户机器上存放的真实位置以及这个文件对应的待下载文件夹表中的文件夹的id。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| id | int | 下载文件的id，主键 |
| u\_id | int | 用户的id |
| logic\_id | int | 下载的文件的id |
| file\_len | int | 文件总长度 |
| file\_real\_path | string | 用户主机上的真实文件路径 |
| downloading\_dir\_id | int | 该待下载文件对应的待下载文件夹表中的文件夹序号 |

## 1.7. 待下载文件表

待下载文件夹表也是至关重要的一个表，其记录了用户下载文件夹时的若干信息。首先依然为每一个下载的文件夹分配一个下载id并记录了其下载用户的id，此外还记录了这个文件夹在用户主机上的存放位置、这个文件夹在云盘上的逻辑id、这个文件夹里一共有多少文件以及当前下载到第几个文件。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| id | int | 待下载文件夹的唯一id，主键 |
| u\_id | int | 用户的id |
| file\_real\_path | string | 用户主机上的真实文件路径 |
| logic\_id | int | 本目录的逻辑id |
| file\_now | int | 传到第几个文件 |
| file\_count | int | 一共多少个文件 |

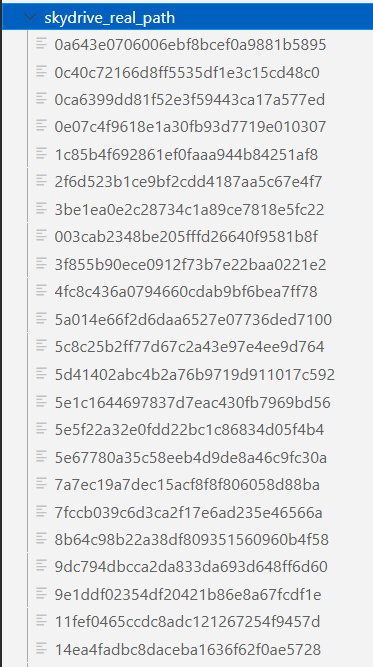
## 1.8. 用户登录日志表

我们用数据库的形式来记录用户登录的情况，在这个表中，我们会记录用户登录的id，其登录使用的ip地址和对应的端口，以及一个登录时的时间戳信息。

| **表项名** | **类型** | **注释** |
| --- | --- | --- |
| user\_id | int | 用户的id，主键 |
| ipaddr | string | 用户的ip地址（加端口） |
| login\_time | bigint | 用户登录的时间戳 |

# 二、文件在server端的存放

在我们设计的网盘中，所有的文件只会在server端保存一次，我们称之为物理文件，这些文件被存放在一个统一的目录下，并以其md5码来命名。无论其上传时是什么格式、什么名称，我们只关心其文件的内容，如下图所示：



## 2.1. 文件上传

文件上传主要分为单文件上传和文件夹上传。

* 单文件上传

首先取得一些必要的文件信息，如md5码等，然后去比对物理文件表检查是否可以秒传。如果不能秒传就需要进入正式的文件上传流程。开始上传之前，我们设计要让用户端和服务器进行一次“交流”，客户端以此得知关于上传的一些信息。我们的文件上传是采用分包上传的方式，每一次只上传一个固定的大小，循环发送直到文件发送结束。我们每次上传的方法是发送post请求并把数据携带在报文里面。服务端每一次收到后都会根据写入文件的长度去修改数据库中待上传文件表中文件当前的指针，然后返回给客户端。客户端由此服务端是否已经成功写入，随后按照返回的指针位置开始下一次发送。最后当服务器已经接收了文件的全部内容之后，其就会删除待上传文件表中的这个表表项，随后在物理文件表和逻辑文件表中加入这个文件的相关内容。

* 上传文件夹

上传文件夹首先也需要客户端和服务器进行一次通讯，得到关于文件夹上传的一些细节。然后其以DFS的算法去遍历用户本地的文件夹，对于其中每一个文件进行文件上传的流程。其中，特别的，如果客户端在递归过程中上传的文件是一个文件夹，服务器其实是不建立物理文件的，而是直接去创建一个逻辑文件即可。

## 2.2. 断点续传

* 单文件续传

由于我们出色的设计，续传的逻辑可以和上传相统一。根据刚才所述的设计，服务器真正写入文件之后会去更改待上传文件表中记录的文件指针，这就使得接收端服务器实际的文件和数据库的表中记录拥有相当的一致性。而我们每一次上传文件之前都会让客户端和服务端通信一次，在这一次的沟通中服务器就会给前端返回这个文件当前的指针，如果是第一次上传指针自然是0，而如果是续传这个指针就会准确的指向断点的位置，然后客户端再从这里继续文件上传的操作即可。

* 文件夹续传

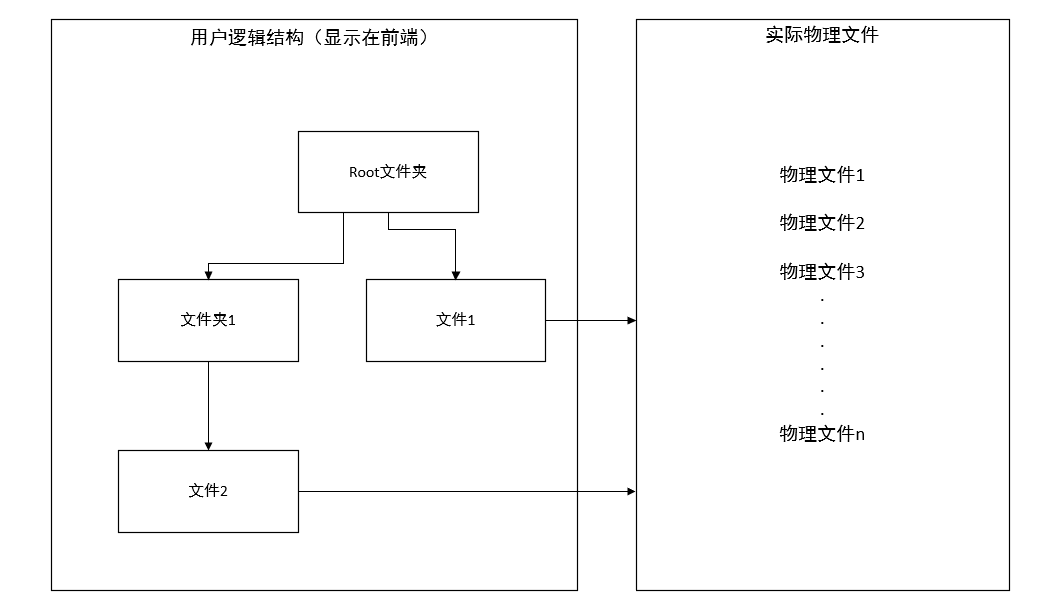
文件夹续传考虑的问题和单文件上传不同，文件夹续传主要需要考虑传输到了下面哪个文件，而怎么续传文件已经考虑好了。由于用DFS遍历文件的顺序是固定的不具有随机性，所以我们直接保存了文件夹的总文件数和当前上传到的文件位置，这与文件类比就相当于是一个指针，由服务器维护指向文件夹的断点。而每一次上传文件夹时客户端需要与服务器进行一次通信，取得这个断点的位置，然后在遍历时跳过所有小于这个断点的文件，之后再执行上传流程。

## 2.3. 秒传

由于我们的设计严格把逻辑结构和实际的文件分离开来，所以秒传并非什么难事。秒传的条件是上传的文件已经存在于服务器上，而我们服务器上所有文件都只上传一份并以md5码唯一标识其特征。因此，上传之前只要判断一下物理文件表里是否存在于上传文件md5相同的文件，如果存在就证明这个文件已经存放在服务器上了，只要取建立其逻辑结构并把这个物理文件的引用数增加即可完成秒传。

# 三、用户目录结构

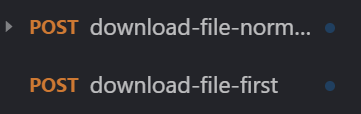
我们对于用户拥有的文件采用逻辑文件的形式来表示，即在逻辑文件表中所有用户都会拥有一个根目录表项，而之后以其为父节点又会产生新的逻辑文件表项，这些逻辑文件表项如果is\_dir是0，则是文件，其会通过md5码指向一个物理文件，而如果is\_dir是1，则其为目录，不会匹配到任何的物理文件，而新的文件节点可以以其为父节点插入表中。也就是说，所有用户都拥有一棵逻辑文件树，而其中的所有文件都是在引用服务器上的整个物理文件存储库中的某一个已经上传的文件，这样既能做到正确显示每一个用户的目录结构，又能确保所有文件都只有一份存储不造成浪费，示意图如下：



# 四、文件下载

## 4.1. 单文件下载

文件下载和上传的操作有些许不同，因为下载是否完毕取决于客户端收了多少数据，而不在于服务器发了多少数据。因此，下载文件方面，服务端提供了三个接口：

这三个接口分别用于提醒服务器开始下载文件（服务器在待下载文件表中创建表项，进行相应的准备操作）；分片传输文件；通知服务器下载已经完成（服务器删除对应表项，完成一些收尾操作）。

## 4.2. 文件夹下载

文件夹下载和文件夹上传也不一样。文件夹上传需要更改用户在数据库中的目录结构，而文件夹下载不需要，故服务器仅提供以下两个接口：

用来通知服务器，客户端即将下载文件夹，以及文件夹下载完毕。真正的下载过程，全部是递归调用文件下载的接口实现的。

## 4.3. 秒下

下载功能，秒下的检查比较严苛，要求文件名和文件内容都不能改变。因此，用户点击下载按钮后，我们首先查看本地目标路径有无文件，如果有则比对md5码，如果相同则直接提示秒下成功，如果不同再提示是否覆盖等等。文件夹秒下就是递归地判断内部的文件是否可以秒下。

## 4.4. 单文件断点续下

单文件断点续下的实现是比较精彩的。一开始，我们在数据库的待下载文件表中建立了file\_pointer表示文件下载到了什么位置，并且每次在后端返回文件片段时修改改值。但是这样做实际上是不正确的。由于下载的最终目标位置是客户的机器，因此客户硬盘中文件的长度才是真正准确的下载到的位置。于是我们删除了file\_pointer项（最后为了显示进度条又加上了，但是仅作为进度条使用，不做实际下载使用），客户端每次请求文件片段时都在post请求体中带上本地文件的长度（也就是所谓的file\_pointer，但这里是绝对准确的），服务器返回其要求的片段。

我们的系统就是因为采用这种“以客户端为准”的思想设计，才能保证不论在什么样的网络环境下，续下功能都不会产生错误。

## 4.5. 文件夹断点续下

文件夹断点续下和断点续传的逻辑又不一样，比其简单许多。我们在数据库中记录了一个file\_now表示目前下载了几个文件，但是仅作为进度条展示使用，不做实际使用。实际情况下，我们每次续下文件夹都会重新把所有的文件都下载一遍，但是由于续下时前面的部分已经下载完毕，所以系统会自动转为秒下。文件夹续下的问题就转为前半段秒下，后半段正常下载的，巧妙的化解了这个问题。

这里另外提一句，文件夹续传不能采用这种思路，因为秒传会对文件的链接数产生影响。

# 五、项目总结

本次项目完成度比较高，除了多用户上传同一文件的功能没有实现（多用户上传不同文件、多用户同时下载同一文件等等功能全部实现了，仅上传同一文件没有实现），其余的功能全部实现了，并且编写了不错的前端界面，小组成员总体来说还是较为满意的。

本次项目我们踩了非常多的坑，同时也学习到了许多的知识，尝试了使用c++手动解析http请求，认识到了网络传输的不可靠性，认识到了使用算法确保传输正确的重要性，同时也对http请求的各项规则和格式（如跨域问题提前发送OPTIONS请求等等）有了更深的了解。不足之处就是部分前端的提示信息还没有太完善，以及整个项目难度最高的多用户上传同一文件没有实现。期待日后的开发完成度可以更高。