同济大学

TONGJI UNIVERSITY

《WEB 技术》

实验报告（大作业）

实验名称	同济云盘
小组成员	马家昱（1950509）　陈冠忠（1950638）
学院（系）	电子与信息工程学院
专　　业	计算机科学与技术
任课教师	郭玉臣
日　　期	2022 年 6 月 1 日

目录

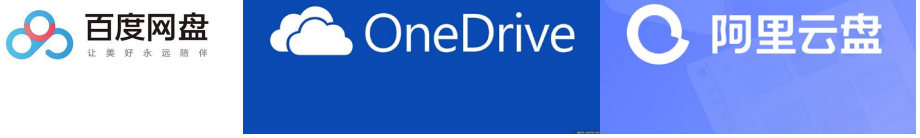
装 订 线

装 订 线

一、项目背景

1.1 背景分析

网盘（网络云盘、网络硬盘）因为其便捷性被很多人青睐，一个便捷的云盘服务，可以跨平台、跨地域，实现文件的存储、共享等功能；不需要随身携带存储介质，只要能上网，就能访问文件。随着网络带宽的增长，以及便携式设备越来越普及与人均设备数越来越多的今天，人们更趋向于将需要随时取用的文件上传至云端，既实现了数据备份，也便于随时存取。目前市面上主要的云盘服务有百度网盘、腾讯微云等。其主要功能均为给每个用户以目录形式提供文件的**访问、存储与共享**功能。



实现一个基本的，将所有用户的文件分别按目录保存的，具有上传下载功能的网盘看似容易，但在可扩展性与性能方面较差。网盘的核心特性是**灵活**，另一个核心吸引力是**分享与转存**，用户可以自由操纵自己的网络空间，轻松实现不同用户空间的分享。

用户的大部分云盘空间保存的是别人分享的内容。这些内容是重复的。如果按照最原始的方法，将所有文件都保存在服务端，空间占用是一方面，另一方面那么，每一次移动和分享操作，都意味着大量文件的复制和移动，这对服务器的性能消耗是巨大的。因此，需要有一种更灵活的云盘设计。

1.2 项目简介

本次大作业的项目名称为**同济云盘**。我们从零开始设计并实现了一个 Web 端云盘应用。仿照市面上流行云盘应用的流程，实现了用户对文件和目录的**上传、下载、删除、复制、移动、重命名、分享与转存**。我们同样实现了管理员对数据库的管理。

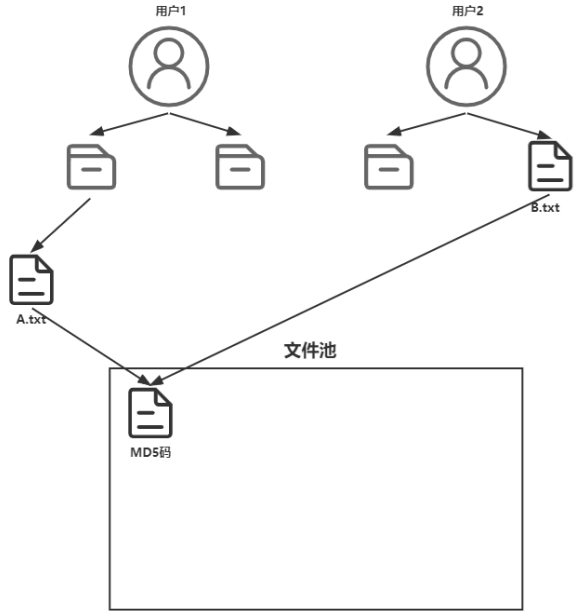
该项目采用前后端分离的开发方法，进行了数据库模型与 API 接口的详细设计。主要采用 Web 技术有：

- 前端：Vue + element UI
- 后端：Django + Django Rest Framework
- 数据库：MySQL + Redis

其中，MySQL 作为持久化的数据库，Redis 作为网站的全局缓存，以及保存 session

与分享链接等有过期时间的数据。

该项目最大的特色是数据库设计。我们模仿 **Linux 系统硬连接**与 **C++11 智能指针 share_ptr** 的实现思路，采用文件实体和文件记录分离的设计方式，将具体的文件数据以 MD5 码为标识，统一存放在文件池中，并且记录其链接数。使用 3 张表，分别是**文件实体表、目录实体表与文件目录映射表**，记录了所有用户的文件和目录。如下图所示，内容相同的文件指向的是文件池中的同一个文件，尽管这两个文件属于不同的用户，所在的目录不相同，甚至文件名及其后缀也不同。



通过这样的设计，文件和目录的大部分操作可以在 O(1)的时间复杂度下完成。少部分涉及目录的操作，例如复制整个目录，也只需要在 O(n)时间内完成，而且只在数据库层面进行增删查改，不涉及实体文件的移动和复制。

除此之外，在安全与交互性方面，我们也做了大量工作。对前端用户输入的数据与后端接口收到的数据进行了合法性的双重判断，保证了接口的幂等性。在重复的目录名或文件名上，自动生成“副本”，“副本(1)”等后缀。与此同时，前端实现了大量的用户提示信息，保证了用户体验的流畅。

二、需求分析

2.1 功能需求抽取

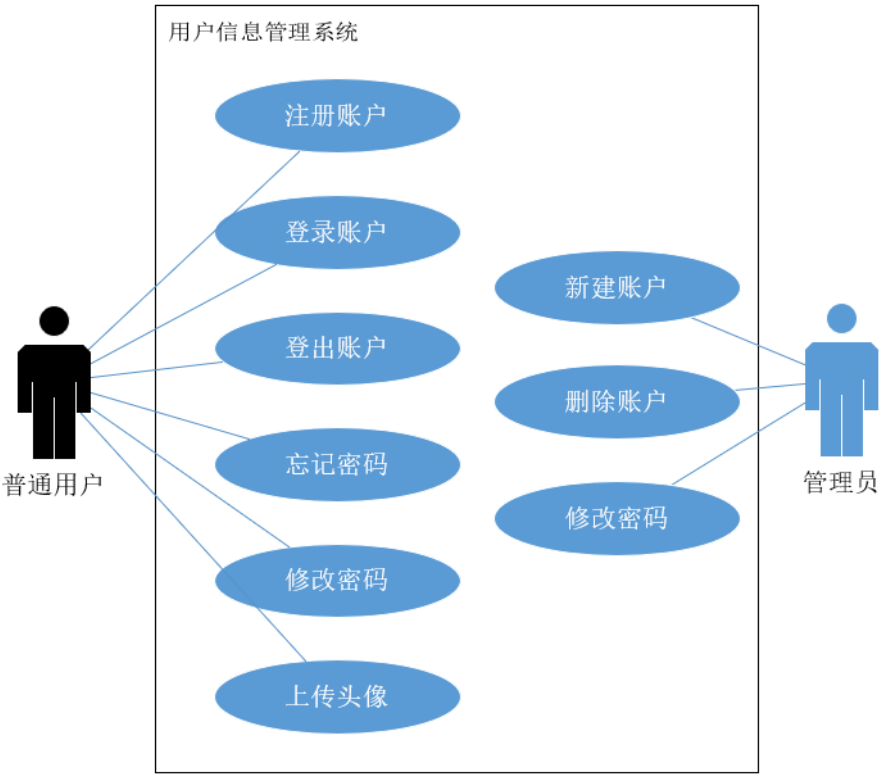
功能需求	普通用户	管理员
注册(新建账号)	√	√
登录账号	√	√
登出账号	√	√
忘记密码	√	

修改密码	√	
上传头像	√	√
上传文件	√	
新建目录	√	√
下载文件	√	
下载目录	√	
复制文件	√	√
复制目录	√	√
移动文件	√	
移动目录	√	
删除文件	√	√
删除目录	√	√
重命名文件	√	√
重命名目录	√	√
分享文件	√	
分享目录	√	
转存文件	√	
转存目录	√	
删除用户		√

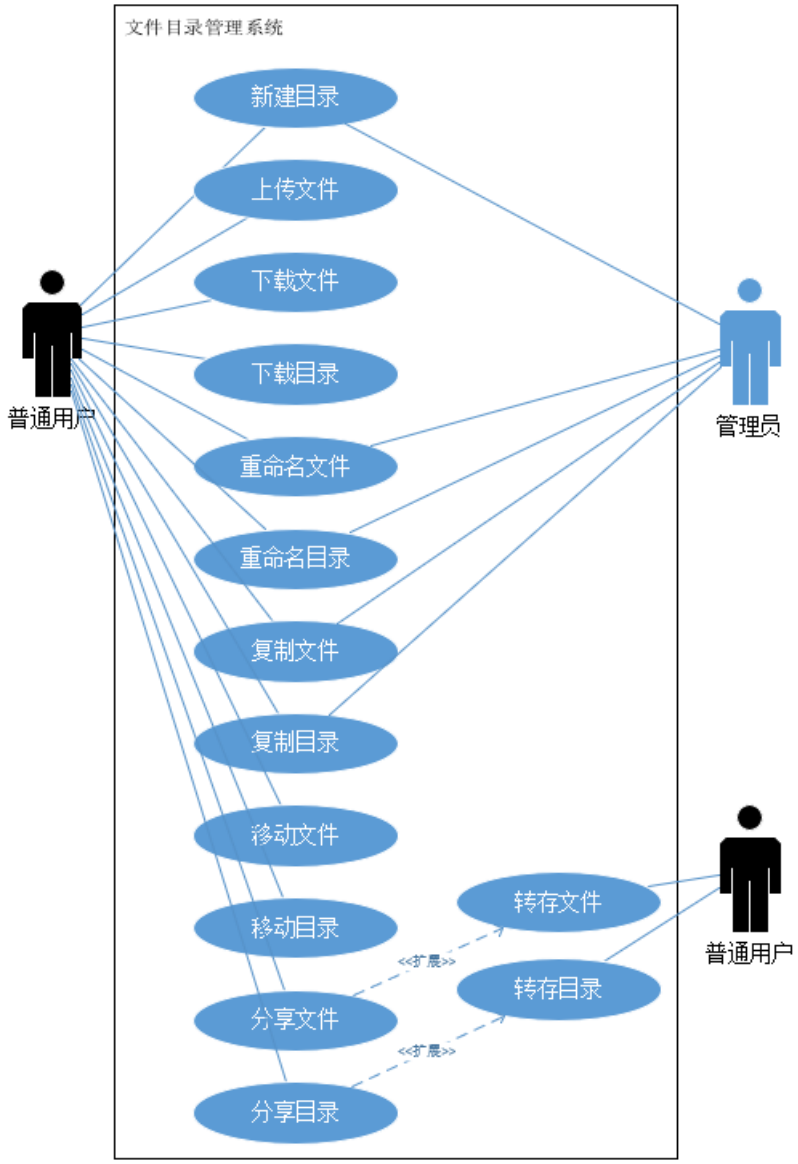
其中，管理员可以对任意用户的文件和目录进行修改和删除。

2.2 用例图

用户信息管理系统：


用户信息管理系统
注册账户
登录账户
登出账户
忘记密码
修改密码
上传头像
新建账户
删除账户
修改密码
普通用户
管理员

文件目录管理系统：



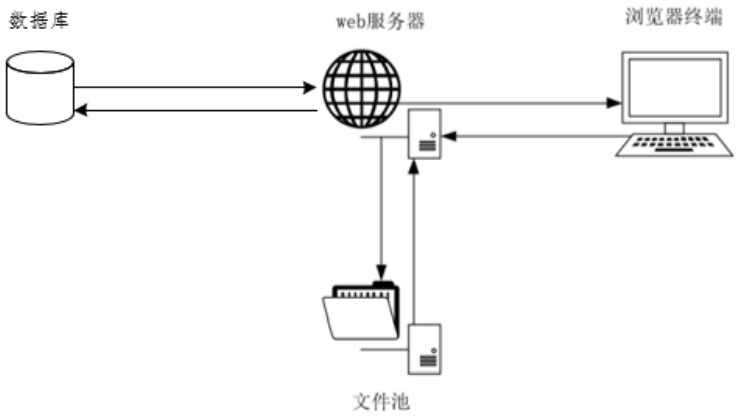
2.3 非功能性需求

- 安全性：
  使用 session 记录用户登录信息，数据库使用密文保存密码，前端 URL 中不包含用户数据。前端与后端双重判断用户输入数据合法性。API 结构的幂等性。
- 健壮性：
  对重复文件名和目录名自动生成副本后缀。
- 性能要求：
  除上传下载外、对用户操作的响应在可接受范围内。
- 可交互性：
  UI 界面完整、交互过程流畅、提示信息完备。

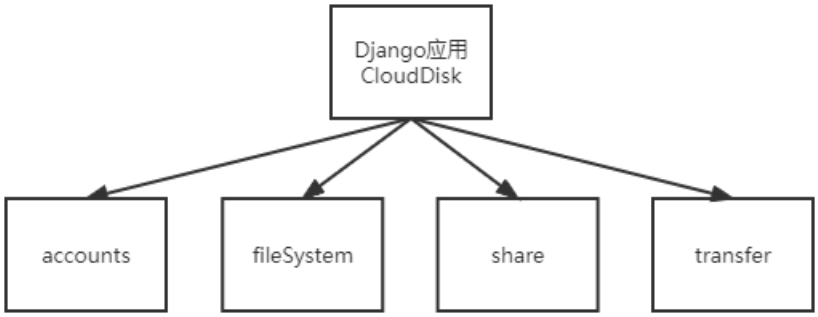
三、整体设计

3.1 模块结构

项目的整体结构如图所示，基本符合 MVC 设计模式



后端采用模块化设计，共有 4 个 APP：



- Accounts: 用户信息管理系统，与账户相关的业务逻辑
- Filesystem: 文件系统，负责文件与目录的操作
- Share: 分享系统，与文件和目录的分享和移动有关的操作
- Transfer:： 传输系统，负责文件上传、打包与下载

3.3 数据库结构

User 用户模型：

使用 Django 自带的用户模型，使用到的字段有 username、password、email 等

文件实体表：FileEntity:

属性名	字段	类型	约束
编号	id	INT	PK,AUTO_INCREMENT, UNIQUE,NOT NULL
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

MD5 码为文件经过 MD5 哈希后得到的字符串，被链接的个数初始为 1，复制、上传与分享等操作会使其增加，删除会使其减少。当其减少至 0 时，将文件从文件池中删除。

目录实体表：DirectoryEntity

属性名	字段	类型	约束
编号	id	INT	PK,AUTO_INCREMENT, UNIQUE,NOT NULL
名称	dname	VARCHAR	NOT NULL
所属用户编号	u_id	INT	NOT NULL
父目录编号	parent_id	INT	NOT NULL
修改时间	lastchange	TIMESTAMP	NOT NULL

其中父目录编号为外键，对象为自己，即自相连。

文件目录映射表：FileDirectory

属性名	字段	类型	约束
文件编号	fid	INT	NOT NULL
目录编号	did	INT	NOT NULL
文件名称	fname	VARCHAR	NOT NULL
修改时间	lastchange	TIMESTAMP	NOT NULL

文件名称即为用户看到的文件名，其文件编号与目录编号均为外键，指向文件实体表与目录实体表。该表的主键为多重主键(fid, did, fname)

数据库使用 Django ORM 与 MySQL 连接，模型定义代码如下：

```
详 class FileEntity(models.Model):
    md5 = models.CharField(max_length=128)
    fsize = models.IntegerField()
    link_num = models.IntegerField()


class DirectoryEntity(models.Model):
    dname = models.CharField(max_length=128)
    uid = models.ForeignKey(User, on_delete=models.CASCADE)
```

```
    parentid= models.ForeignKey('self',on_delete=models.CASCADE,null=True)
    lastchange = models.DateTimeField(default=timezone.now)

class FileDirectory(models.Model):
    lastchange = models.DateTimeField(default=timezone.now)
    fid=models.ForeignKey(FileEntity,on_delete=models.CASCADE)
    did=models.ForeignKey(DirectoryEntity,on_delete=models.CASCADE)
    fname=models.CharField(max_length=128)
    class Meta:
        unique_together = ("fid", "did","fname")
```

四、详细设计

4.1 API 接口列表

参数与返回数据及状态码太过繁杂，在此处不赘述

类型	API	描述
GET	/api/accounts/myinfo	获取当前用户信息
GET	/api/accounts/logout	登出当前账号
GET	/api/accounts/getIdentity	获取登录状态
GET	/api/filesystem/getroot	获取根目录详细信息
GET	/api/filesystem/getdir	获取目录详细信息
GET	/api/filesystem/removefile	删除文件
GET	/api/filesystem/removedir	删除目录
GET	/api/share/getdirtree	获取用户目录树
GET	/api/share/getsharetoken	获取分享 token
POST	/api/accounts/verify	像邮箱发送验证码
POST	/api/accounts/login	登录
POST	/api/accounts/register	注册
POST	/api/accounts/forgetpassword1	忘记密码第一步
POST	/api/accounts/forgetpassword2	忘记密码第二步
POST	/api/accounts/uploadavatar	上传头像
POST	/api/filesystem/createdir	新建目录
POST	/api/filesystem/renamedir	重命名目录
POST	/api/filesystem/renamefile	重命名文件
POST	/api/share/movefile	移动文件
POST	/api/share/movedir	移动目录
POST	/api/share/copyfile	复制文件
POST	/api/share/copydir	复制目录
POST	/api/share/verifysharetoken	验证分享 token
POST	/api/transfer/uploadfile1	上传文件第一步
POST	/api/transfer/uploadfile2	上传文件第二步
POST	/api/transfer/downloadfile	下载文件
POST	/api/transfer/downloaddir	下载目录

4.2 后端主要功能算法与流程

以具有代表性的算法为例，描述业务逻辑

后端：

注册

1. 向发送邮箱验证码
2. 检查用户名或邮箱是否存在
3. 检查验证码是否正确
4. 为新用户创建根目录和默认头像
5. 注册完成

```
class Register(APIView):
    authentication_classes = (CsrfExemptSessionAuthentication, BasicAuthentication)
    def post(self, request, format=None):
        data = json.loads(request.body)
        username = data.get("username").strip()
        password = data.get("password").strip()
        email = data.get("email").strip()
        verify_code = data.get("verify_code").strip()
        # empty field
        if not username or not password or not email or username == "" or password==""  or email == "":
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户名、密码、邮箱不能为空"
                }
            })
        if email not in verify_list or verify_list[email]!=verify_code:
            return Response({
                "message":"failure",
                "data":{
                    "detail":"验证码错误"
                }
            },status=400)
        # user already exists
        elif User.objects.filter(username=username).exists():
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户名已存在"
                }
            })
        # email already exists
        elif User.objects.filter(email=email).exists():
            return Response({
                "message":"failure",
                "data":{
                    "detail":"邮箱已存在"
                }
            })
        # success
        else:

            user=User.objects.create_user(username=username, password=password,email=email)
            DirectoryEntity.objects.create(dname="root",uid=user)
            verify_list.pop(email)

```

```
            os.system('cp '+os.path.join(CloudDisk.settings.STATICFILES_DIRS[0],'avatar','default.jpg')+' '+os.path.join(CloudDisk.settings.STATICFILES_DIRS[0],'avatar',str(user.id)+'.jpg'))


            return Response({
                "message":"success"
            })
```

上传文件

1. 前端计算文件 md5 码，后端收到后判断是否已存在对应文件。若存在，则直接在数据库中增加映射表项，返回成功
2. 若是新文件，则接收文件数据，以 md5 码为文件名保存到服务端，同时新建数据库表
3. 递归修改父目录最后修改时间

```
class UploadFile1(APIView):
    authentication_classes = (CsrfExemptSessionAuthentication, BasicAuthentication)
    def post(self, request):
        # 判断用户登录
        user=request.user
        if not user.is_authenticated:
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户未登录"
                }
                },status=400)
        else:
            md5=request.data.get("md5")
            pid=request.data.get("pid")
            fname=request.data.get("fname")

            if not DirectoryEntity.objects.filter(uid=user,pk=int(pid)).exists():

                return Response({
                    "message":"failure",
                    "data":{
                        "detail":"错误访问"
                    }
                    },status=400)

            parentdir=DirectoryEntity.objects.get(pk=pid)
            if FileEntity.objects.filter(md5=md5).exists():
                file=FileEntity.objects.filter(md5=md5)[0]
                file.link_num+=1
                file.save()
                fname1,fname2=os.path.splitext(fname)
                if FileDirectory.objects.filter(did=parentdir,fname=fname):
                    if FileDirectory.objects.filter(did=parentdir,fname=fname1+'-副本'+fname2):
                        i=1
                        fname=fname1+'-副本({0})'.format(i)+fname2
                        while FileDirectory.objects.filter(did=parentdir,fname=fname).exists():
                            i+=1
                            fname=fname1+'-副本({0})'.format(i)+fname2
                        FileDirectory.objects.create(fid=file,did=parentdir,fname=fname)
                    else:
```

```
                        FileDirectory.objects.create(fid=file,did=parentdir,fname=fname1+'-副本'+fname2)
                else:
                    FileDirectory.objects.create(fid=file,did=parentdir,fname=fname)
                updateLastChange(pid)
                return Response({
                    "message":"success"
                    })
            else:
                return Response({
                    "message":"failure"
                    },status=201)

class UploadFile2(APIView):
    authentication_classes = (CsrfExemptSessionAuthentication, BasicAuthentication)
    def post(self, request):
        # 判断用户登录
        user=request.user
        if not user.is_authenticated:
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户未登录"
                }
                },status=400)
        else:
            pid=request.data.get("pid")
            md5=request.data.get("md5")
            files = request.FILES.getlist('file')

            if not DirectoryEntity.objects.filter(uid=user,pk=int(pid)).exists():

                return Response({
                    "message":"failure",
                    "data":{
                        "detail":"错误访问"
                    }
                    },status=400)

            file_path=None
            fname=None
            for f in files:
                file_path = os.path.join(CloudDisk.settings.FILE_POOL,md5)
                fname=f.name
                with open(file_path,'wb') as fp :
                    for chunk in f.chunks():
                        fp.write(chunk)

            fsize=os.path.getsize(file_path)
            file=FileEntity.objects.create(md5=md5,fsize=fsize,link_num=1)

            parentdir=DirectoryEntity.objects.get(pk=pid)
            files=FileDirectory.objects.filter(did=parentdir)


            fname1,fname2=os.path.splitext(fname)
            if FileDirectory.objects.filter(did=parentdir,fname=fname):
                if FileDirectory.objects.filter(did=parentdir,fname=fname1+'-副本'+fname2):
                    i=1
```

```
                    fname=fname1+'-副本({0})'.format(i)+fname2
                    while FileDirectory.objects.filter(did=parentdir,fname=fname)
.exists():
                        i+=1
                        fname=fname1+'-副本({0})'.format(i)+fname2
                     FileDirectory.objects.create(fid=file,did=parentdir,fname=fn
ame)
                else:
                     FileDirectory.objects.create(fid=file,did=parentdir,fname=fn
ame1+'-副本'+fname2)
            else:
                FileDirectory.objects.create(fid=file,did=parentdir,fname=fname)

            updateLastChange(pid)
            return Response({
                    "message":"success"
                    })
```

移动目录

1. 检查目标目录是否有相同目录名的目录，若存在则建立副本
2. 以递归的方法新建目录表与目录文件映射表
3. 修改源目录和目标目录的所有父目录的最后修改时间
4. 删除源目录表项，自动删除所有以其为外键的映射表

```
class MoveDir(APIView):
    authentication_classes = (CsrfExemptSessionAuthentication, BasicAuthenticatio
n)
    def post(self,request):
        user=request.user
        if not user.is_authenticated:
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户未登录"
                }
                },status=400)
        else:
            did=int(request.data.get("did"))
            pid=int(request.data.get("pid"))

            parentdir=DirectoryEntity.objects.get(pk=pid)
            dir=DirectoryEntity.objects.get(pk=did)

            if dir.parentid.id==pid:
                return Response({
                    "message":"success"
                    })

            temp=parentdir
            while temp:
                if temp.id==did:
                    return Response({
                    "message":"failure",
                    "data":{
                        "detail":"无法将文件移动至其自身下"
                    }
                    },status=400)
                temp=temp.parentid

```

```
            dname=dir.dname
            newdir=None

            if DirectoryEntity.objects.filter(parentid=parentdir,dname=dname):
                if DirectoryEntity.objects.filter(parentid=parentdir,dname=dname+'-副本'):
                    i=1
                    dname=dname+'-副本({0})'.format(i)
                    while DirectoryEntity.objects.filter(parentid=parentdir,dname=dname).exists():
                        i+=1
                        dname=dir.dname+'-副本({0})'.format(i)
                    newdir=DirectoryEntity.objects.create(uid=user,parentid=parentdir,dname=dname)
                else:
                    newdir=DirectoryEntity.objects.create(uid=user,parentid=parentdir,dname=dname+'-副本')
            else:
                newdir=DirectoryEntity.objects.create(uid=user,parentid=parentdir,dname=dname)

            copydir(user,newdir.id,did)
            updateLastChange(pid)



            updateLastChange(dir.parentid.id)
            dir.delete()

            return Response({
            "message":"success"
            })


def copydir(user,newid,oldid):
    olddir=DirectoryEntity.objects.get(pk=oldid)
    newdir=DirectoryEntity.objects.get(pk=newid)

    files=FileDirectory.objects.filter(did=olddir)
    for file in files:
        FileDirectory.objects.create(fid=file.fid,did=newdir,fname=file.fname)
        file.fid.link_num+=1
        file.fid.save()

    dirs=DirectoryEntity.objects.filter(parentid=olddir)
    for dir in dirs:
        t=DirectoryEntity.objects.create(uid=user,dname=dir.dname,parentid=newdir)
        copydir(user,t.id,dir.id)
```

分享文件或目录

1. 根据前端数据，在 redis 中建立以随机 token 为键，所分享文件 id 为值的哈希数据结构，设置过期时间，返回 token
2. 前端生成分享链接
3. 点击链接后验证 token，在 redis 中查询链接是否不存在或过期，文件或目录是否还存在，根据键值对跳转至分享界面
4. 用户浏览目录或文件，选择下载或转存

```
class GetShareToken(APIView):
```

```
    def get(self, request):
        user=request.user
        if not user.is_authenticated:
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户未登录"
                }
                },status=400)

        type=request.GET['type']
        id=int(request.GET['id'])

        if type=="file" and not FileDirectory.objects.filter(pk=id).exists \
             or type=="dir" and not DirectoryEntity.objects.filter(pk=id).exists:

            return Response({
                "message":"failure",
                "data":{
                    "detail":"错误访问"
                }
                },status=400)

        alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
        characters = "".join(random.sample(alphabet, 32))



        r = redis.Redis(host='localhost', port=6379, db=0)

        r.hset(characters,"type",type)
        r.hset(characters,"id",id)
        r.expire(characters,3600)

        return Response({
            "message":"success",
            "token": characters
            })


class VerifyShareToken(APIView):
    def get(self, request):
        user=request.user
        if not user.is_authenticated:
            return Response({
                "message":"failure",
                "data":{
                    "detail":"用户未登录"
                }
                },status=400)

        else:

            token=request.GET['token']

            r = redis.Redis(host='localhost', port=6379, db=0)
            if not r.exists(token):
                return Response ({
                "message":"failure",
                "data":{
                    "detail":"分享链接不存在或已失效"
```

```
                }
                },status=400)
            else:
                type=r.hget(token,"type").decode()
                id=int(r.hget(token,"id"))

                if type=="dir" and not DirectoryEntity.objects.filter(pk=id).exis
ts() or\
                    type=="file" and not FileDirectory.objects.filter(pk=id).exis
ts():
                    return Response({
                    "message":"failure",
                    "data":{
                        "detail":"文件已不存在"
                    }
                    },status=401)

                if type=="file":
                    file=FileDirectory.objects.get(pk=id)
                    return Response({
                        "message":"success",
                        "data":[{"type":type,"id":id,"name":file.fname,"fsize":fi
le.fid.fsize,"lastchange":file.lastchange.strftime('%Y-%m-%d %H:%M:%S')}]
                    })
                else:
                    dir=DirectoryEntity.objects.get(pk=id)
                    return Response({
                        "message":"success",
                        "data":[{"type":type,"id":id,"name":dir.dname,"lastchange
":dir.lastchange.strftime('%Y-%m-%d %H:%M:%S')}]
                    })
```

4.3 前端设计

以具有代表性的前端设计为例，描述前端实现

整体布局

整体布局使用 Element Plus 提供的 El-Container 容器，将网盘页面分为三部分

```
<el-container>
    <el-header>
        <!-- 导航栏内容 -->
    </el-header>
    <el-container>
        <el-header>
            <!-- 功能按钮和文件夹路径 -->
        </el-header>
        <el-main>
            <!-- 文件展示内容 -->
        </el-main>
    </el-container>
</el-container>
```

右键菜单

通过 DirMenuVisible 变量控制右键菜单是否展示

HTML 代码：

```
<div id="dirmenu" v-show="DirMenuVisible" class="menu">
<div class="contextmenu_item" @click="removeMenu();renameForm.name=selectRow.name
;renameForm.visible=true">重命名</div>
<div class="contextmenu_item" @click="removeMenu();deleteFunc()">删除文件夹
</div>
<div class="contextmenu_item" @click="removeMenu();downloadFunc()">下载文件夹
</div>
```

```
<div class="contextmenu_item" @click="removeMenu();openCopyForm()">复制</div>
<div class="contextmenu_item" @click="removeMenu();openMoveForm()">移动</div>
<div class="contextmenu_item" @click="removeMenu();openShareForm()">共享</div>
</div>
```

JS 代码：

```
rightClick: function (row, column, event) {
    // console.log(row, column, event)
    this.selectRow=row;
    event.preventDefault();
    this.removeMenu();
    let menu;
    if(row.type=='dir'){
        menu=document.querySelector("#dirmenu");
        this.DirMenuVisible=true;
    }
    else if(row.type=='file'){
        menu=document.querySelector("#filemenu");
        this.FileMenuVisible=true;
    }
    this.styleMenu(event,menu);
},
removeMenu: function () {
    // 取消鼠标监听事件 菜单栏
    // console.log('removeTreeMenu')
    this.DirMenuVisible = false;
    this.FileMenuVisible = false;
    document.removeEventListener("click", this.removeMenu); // 关掉监听，
},
styleMenu: function (key, menu) {
    // 给整个 document 新增监听鼠标事件，点击任何位置执行 foo 方法
    document.addEventListener("click", this.removeMenu);
    menu.style.left = key.pageX  + "px";
    menu.style.top = key.pageY  + "px";
    //console.log('for menu style',menu.style.left,menu.style.top);
},
```

上传文件

1. 根据文件计算 md5 码
2. 传至后端检测文件是否存在
   a) 存在，后端建立映射，前端完成秒传（不发送文件）
   b) 不存在，前端发送文件

HTML 代码：

```
<el-dialog
    title="上传文件"
    v-model="uploadForm.visible"
    width="30%"
>
    <el-upload
      class="upload-demo"
            ref="upload"
            :show-file-list=false
            :action="uploadURL"
      :on-success="handleUploadSucess"
      multiple
      :limit = "10"
      :before-upload="beforeUpload"
      :on-exceed="handleExceed"
      :on-error="handleFail"
      :file-list="uploadForm.fileList"
```

```
            :http-request="uploadFile">
      <el-button  type="primary">点击上传</el-button>
    </el-upload>
    <template #footer>
    <span class="dialog-footer">
      <el-button @click="uploadForm.visible = false">取消</el-button>
      <el-button type="primary" @click="uploadForm.visible = false">确认
  </el-button>
    </span>
    </template>
  </el-dialog>
```

JS 代码：

```
beforeUpload: function (file) {
    const chunkSize = 1024*1024;
    const fileReader = new FileReader();
    const md5 = new SparkMD5();
    let index = 0;
    file.md5=-1;
    const loadFile = () => {
        const slice = file.slice(index, index + chunkSize);
        fileReader.readAsBinaryString(slice);
        }
    loadFile();
    fileReader.onload = e => {
        md5.appendBinary(e.target.result);
        if (index < file.size) {
            index += chunkSize;
            loadFile();
        } else {
            file.md5=md5.end();
        }
    };
},
uploadFile: async function(request){
    function sleep(time) {
        return new Promise(resolve =>
        setTimeout(resolve,time)
    )}
    while(request.file.md5==-1){
        console.log("正在计算 md5");
        await sleep(1);
    }
    console.log(request.file.md5);
    this.axios.post(`/api/transfer/uploadfile1/`,{pid:this.getPid(),fname:request
.file.name,md5:request.file.md5})
        .then((response)=>{
            console.log(response);
            if(response.status==201){
                ElMessage.success(`正在上传${request.file.name}`);
                let fd=new FormData();
                fd.append('pid',this.getPid());
                fd.append('md5',request.file.md5);
                fd.append('file',request.file);
                this.axios.post(`/api/transfer/uploadfile2/`,fd)then((response)=>
{
                    console.log(response);
                })
                .then(()=>{
                    ElMessage.success(`上传${request.file.name}成功！`);
                    this.refreshPath();
                })
```

```
            }
            else if(response.status==200){
                ElMessage.success(`上传${request.file.name}成功！`);
                this.refreshPath();
            }
            else{
                ElMessage.error("上传失败")
            }
        })
},
```

下载文件

向后端发送 get 请求，以 blob 形式获取返回数据，生成下载链接，完成下载

JS 代码：

```
downloadFunc: function () {
    let row=this.selectRow;
    let URL=row.type=='file'?'/api/transfer/downloadfile?fid='+row.id:'/api/transfer/downloaddir?did='+row.id;
    this.axios.get(URL, {responseType: "blob"}).then((response) => {
        let blob = new Blob([response.data])
        const utf8FilenameRegex = /filename\*=utf-8''([\w%\-.]+)(?:; ?|$)/i;
        const asciiFilenameRegex = /filename=(["']?)(.*?[^\\])\1(?:; ?|$)/i;
        let filename = 'download'
        const fileName = response.headers['content-disposition'];
        const utf8filename = utf8FilenameRegex.exec(fileName)
        const asciifilename = asciiFilenameRegex.exec(fileName)
        if (utf8filename)
          filename = utf8filename[1]
        else
          filename = asciifilename[2]
        console.log(fileName)
                console.log(filename)
        let dom = document.createElement('a')
        let url = window.URL.createObjectURL(blob)
        dom.href = url
        dom.download = decodeURI(filename)

        dom.style.display = 'none'
        document.body.appendChild(dom)
        dom.click()
        dom.parentNode.removeChild(dom)
        window.URL.revokeObjectURL(url)
        ElMessage.success(row.name+'下载成功!');
      });
},
```

删除功能

向后端发送 get 请求即可

JS 代码：

```
deleteFunc: function () {
    console.log(this.selectRow);
    let row=this.selectRow;
    let URL=row.type=='file'?'/api/filesystem/removefile?fid='+row.id:'/api/filesystem/removedir?did='+row.id;
    this.axios.get(URL, {}).then((response) => {
        console.log(response);
        ElMessage.success(row.name+'删除成功!');
    this.refreshPath();
    });
},
```

重命名功能

向后端发送 post 请求，需要处理返回的信息

JS 代码：

```
renameFunc: function () {
    console.log(this.selectRow);
    let row=this.selectRow;
    let URL=row.type=='file'?'/api/filesystem/renamefile/':'/api/filesystem/renamedir/';
    let newName=this.renameForm.name;
    let content=row.type=='file'?{fid:row.id,fname:newName}:{did:row.id,dname:newName}
    if(newName==""){
        ElMessage.error("不能重命名为空！");
        return;
    }
    this.axios.post(URL, content).then((response) => {
        console.log(response);
        ElMessage.success(row.name+'重命名为'+newName+'成功!');
        this.renameForm.visible=false;
        this.renameForm.name='';
        this.refreshPath();
      })
    .catch((e)=>{
        console.log(e);
        ElMessage.error(newName+'已存在');
    });
},
```

复制功能

1. 右键选中待复制的文件/文件夹
2. 打开目标目录选择界面，发送 get 请求获取用户目录树
3. 界面中选择目标目录
4. 向后端发送 post 请求，需要处理返回的信息

HTML 代码：

```
<el-dialog width="40%" v-model="this.copyForm.visible" title="复制到">
    <div class="el-dialog-div">
    <el-container>
    <el-main>
    <el-tree
    ref="copyFileTree"
    class="file-tree"
    :data="fileTree"
    node-key="id"
    empty-text=""
    :expand-on-click-node="false"
    @node-expand="(data,node,t)=>{data.isopen=true;}"
    @node-collapse="(data,node,t)=>{data.isopen=false;}">
    <template #default="{ node, data }">
    <span  class="custom-tree-node">
        <el-icon v-if="data.children && data.isopen"><FolderOpened /></el-icon>
        <el-icon v-else-if="data.children && !data.isopen"><Folder/></el-icon>
        <el-icon v-else><Document /></el-icon>
        <i v-if="data.children && !data.isopen" class="el-icon-folder"></i>
        <i v-else-if="data.children && data.isopen" class="el-icon-folder-opened"></i>
        <i v-else class="el-icon-document"></i>
        <span>{{ node.label }}</span>
    </span>
    </template>
    </el-tree>
```

```
        </el-main>
        <el-footer>
        <span class="dialog-footer">
        <el-button @click="closeCopyForm()">取消</el-button>
        <el-button type="primary" @click="copyFunc()">确认</el-button>
        </span>
        </el-footer>
    </el-container>
    </div>
</el-dialog>
```

JS 代码：

```
openCopyForm: function(){
    this.axios.get("/api/share/getdirtree/",{}).then((response)=>{
        console.log(response);
        this.fileTree=response.data.data;
        this.copyForm.visible=true;
    })
},
closeCopyForm: function(){
    this.copyForm.visible=false;
    this.fileTree=[];
},
copyFunc: function (){
    let row=this.selectRow;
    let node=this.$refs.copyFileTree.getCurrentNode();
    let URL=row.type=='file'?'/api/share/copyfile/':'/api/share/copydir/';
    let content=row.type=='file'?{fid:row.id,pid:node.did}:{did:row.id,pid:node.d
id};
    console.log(node);
    this.axios.post(URL, content).then((response) => {
    console.log(response);
    ElMessage.success(row.name+'复制成功!');
    this.refreshPath();
    this.closeCopyForm();
    })
    .catch((e)=>{
        ElMessage.error(e.response.data.data.detail)
    });
},
```

移动功能

前端逻辑和前端代码基本与复制功能一致，只在 api 接口和后端实现上有所区分，故不再重复叙述

分享功能

选择某一文件/文件夹进行分享，像后端发送 get 请求，获取分享 token，生成共享连接

JS 代码：

```
openShareForm: function(){
    let row=this.selectRow;
    this.axios.get("/api/share/getsharetoken/?type="+row.type+"&id="+row.id,{}).t
hen((response)=>{
        this.shareForm.shareURL=window.location.origin+"/#/Share/"+response.data.
token;
        this.shareForm.visible=true;
        console.log(this.shareForm.shareURL)
    })
},
```

打开分享页面

根据路由中的参数获取 token，向后端发送 get 请求，获取分享的文件夹信息，展示的页面逻辑与个人空间基本一致，但是禁用了大部分功能，只能进行浏览、下载和转存（逻辑类似复制）三种功能。

装 订 线

五、部署与运行方法

5.1 前端

目录结构：

```
├─── node_modules
│      └─── ..... //安装的包依赖
├─── package.json
├─── public
│      ├─── favicon.ico
│      └─── logo_v.png
├─── README.md          //使用说明
├─── src
│      ├─── App.vue
│      ├─── assets        //静态资源
│      │      ├─── dog.png
│      │      ├─── logo_h.png
│      │      └─── logo_v.png
│      ├─── components    //组件
│      │      ├─── 404.vue
│      │      ├─── ForgetPassword.vue  //忘记密码
│      │      ├─── Header.vue          //顶部导航栏
│      │      ├─── Login.vue           //登录
│      │      ├─── Register.vue        //注册
│      │      ├─── Revise.vue          //修改信息
│      │      ├─── ShareSpace.vue      //分享页面
│      │      └─── UserSpace.vue       //个人空间
│      ├─── main.js                  //创建 app
│      ├─── package.json
│      ├─── router
│      │      └─── index.js           //路由
│      └─── views
├─── jsconfig.json
└─── vue.config.js                 //全局设置
```

部署系统：

Ubuntu 20.04 server 64bit

依赖：

NodeJS v16.14.2

apt install nodejs

NodeJS 依赖：

npm install

修改 vue.config.js
修改 devServer 中的 proxy，代理至后端所在的 ip 和端口

```
devServer: {
      proxy: {
        '/api': {
          target: 'http://<server-ip>:<port>/',//接口域名
          changeOrigin: true,              //是否跨域
          ws: false,
        },
        '/static': {
          target: 'http://<server-ip>:<port>/',//接口域名
          changeOrigin: true,              //是否跨域
          ws: false,
        },
      }
  }
```

启动
在根目录输入以下命令以启动前端

npm run serve

5.2 后端

目录结构：

```
├─── accounts          //用户信息管理模块
│      ├─── admin.py
│      ├─── apps.py
│      ├─── __init__.py
│      ├─── models.py
│      ├─── tests.py
│      ├─── urls.py      //用户信息路由
│      └─── views.py     //用户信息业务逻辑
├─── CloudDisk
│      ├─── asgi.py
│      ├─── __init__.py
│      ├─── settings.py //全局设置
│      ├─── urls.py      //全局路由
│      └─── wsgi.py
├─── filepool           //文件池
├─── filesystem         //文件系统模块
│      ├─── admin.py
│      ├─── apps.py
```

```
│       ├── __init__.py
│       ├── models.py  //数据模型
│       ├── tests.py
│       ├── urls.py      //文件系统路由
│       └── views.py     //文件系统业务逻辑
├── manage.py
├── share               //分享功能模块
│       ├── admin.py
│       ├── apps.py
│       ├── __init__.py
│       ├── tests.py
│       ├── urls.py      //分享功能路由
│       └── views.py     //分享功能业务逻辑
├── static              //静态文件目录
│       └── avatar       //头像目录
├── tempdir             //下载目录时的临时文件夹
└── transfer            //传输模块
        ├── admin.py
        ├── apps.py
        ├── __init__.py
        ├── tests.py
        ├── urls.py      //传输模块路由
        └── views.py     //传输模块业务逻辑
```

部署系统：

Ubuntu 20.04 server 64bit

依赖：

MySQL　Ver 8.0.28-0ubuntu0.20.04.3 for Linux on x86_64 ((Ubuntu))

```
1.  sudo apt update
2.  sudo apt install mysql-server
3.  sudo systemctl start mysql.service
```

Redis 5.0.7

```
1.  apt install redis
```

Python 包：

依赖列表已经放在 requirements.txt 中

```
1.  pip install -r requirements.txt
```

修改 coconow/settings.py

ALLOWED_HOSTS 为允许访问的主机 IP

```
1.  ALLOWED_HOSTS = ['124.70.221.116','127.0.0.1']
```


装 订 线

修改数据库设置

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': ,
        'HOST': '127.0.0.1',
        'PROT': 3306,
        'USER': ,
        'PASSWORD':
    }
}
```

邮箱设置，给用户发送验证码时用到的邮箱

```
EMAIL_FROM=
EMAIL_BACKEND ='django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.qq.com' # 腾讯 QQ 邮箱 SMTP 服务器地址
EMAIL_PORT = 25 # SMTP 服务的端口号
EMAIL_HOST_USER =#你的 qq 邮箱，邮件发送者的邮箱
EMAIL_HOST_PASSWORD = #你申请的授权码（略）
EMAIL_USE_TLS = False #与 SMTP 服务器通信时,是否启用安全模式
```

在根目录下输入以下命令以启动后端

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver <ip>:<port>
```

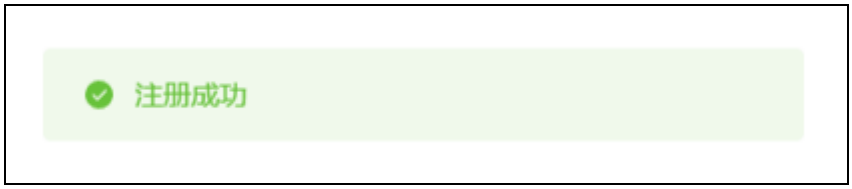
六、测试数据与实验结果

注册

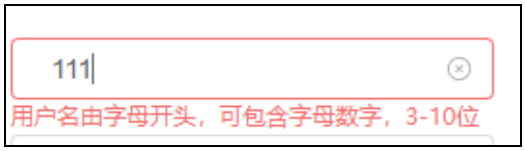
流程：

收到验证码

注册成功，将跳转至登陆界面



错误检查：
用户名位数错误：



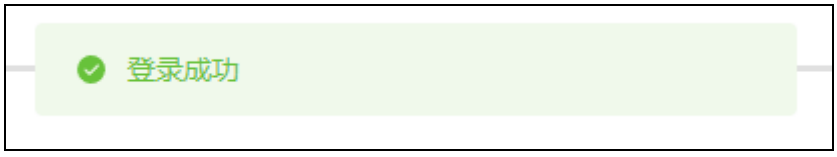
密码位数错误：

密码不一致：

邮箱格式错误：

登录
流程：

登陆成功，跳转至个人主页



错误检查：

用户名或密码不正确，有相应提示

装 订 线

个人主页

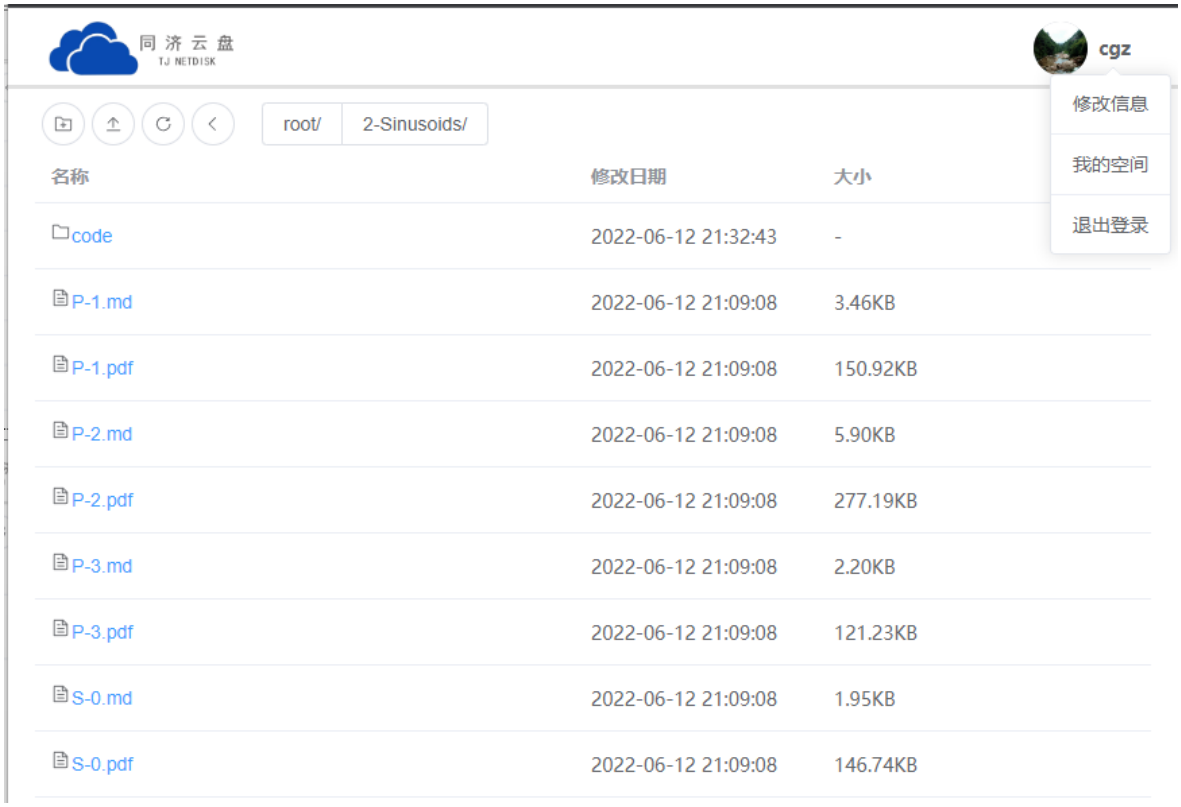
界面展示：

从上到下依次为导航栏、功能按钮和路径、文件列表三部分。

第一部分为导航栏，点击头像会出现下拉框，能修改个人信息、返回个人空间、退出登录。

第二部分为功能按钮和路径，功能按钮包括新建文件夹、上传文件、刷新、回退，路径为一组按钮，按下按钮能跳转到相应路径

第三部分为文件列表，展示了文件的名称、修改时间、大小（文件夹不显示大小）。



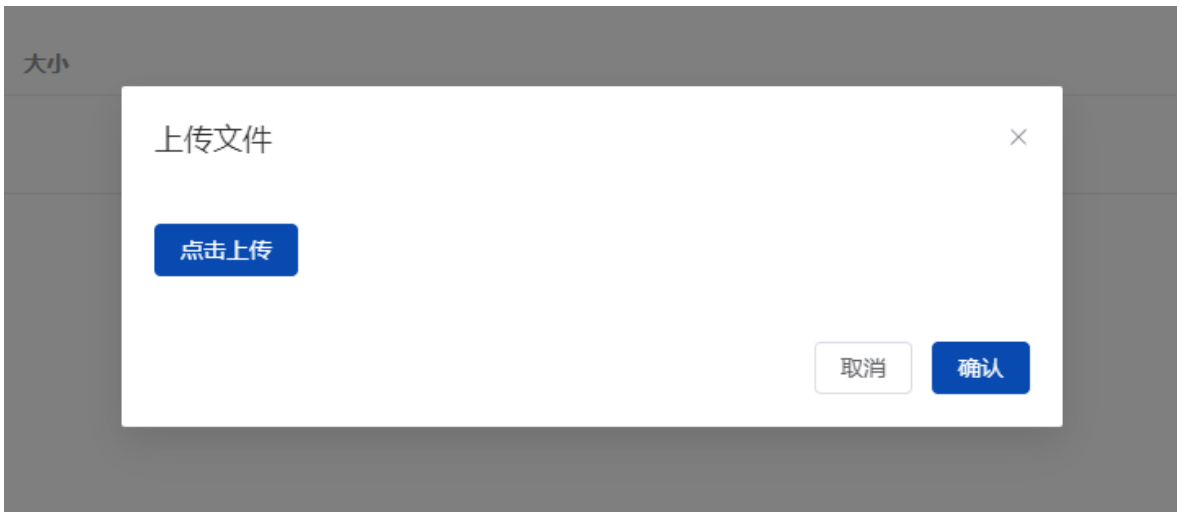
空文件夹的文件列表会显示暂无数据

装 订 线

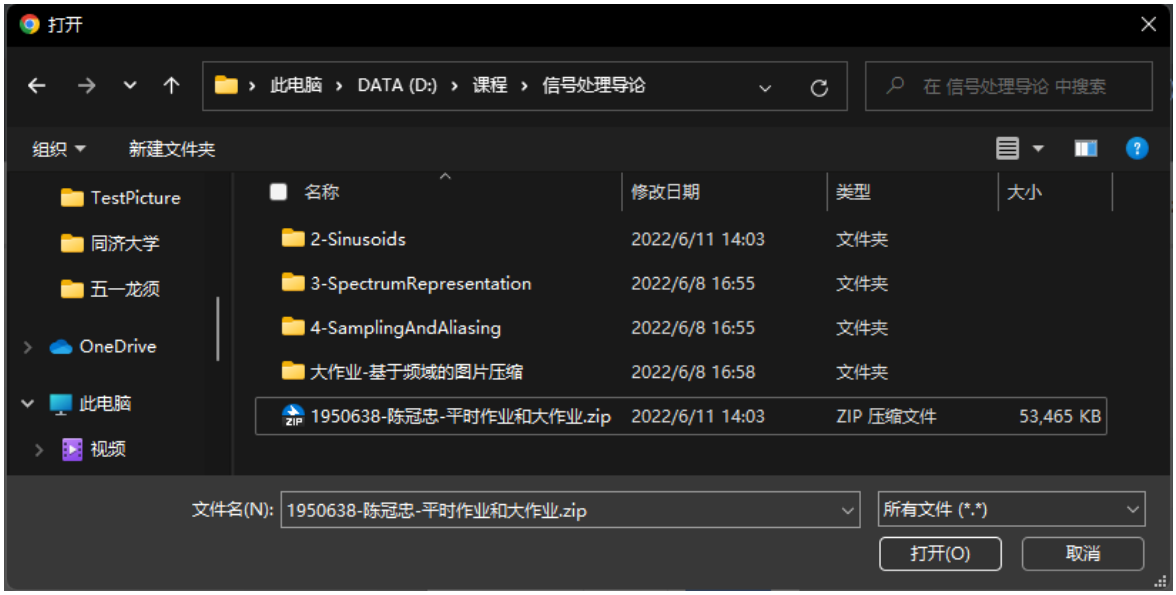
在文件列表右击鼠标，会出现菜单，能完成重命名、删除、下载、复制、移动、共享等操作，对文件夹和文件右击鼠标，会有不同的菜单。

上传文件：

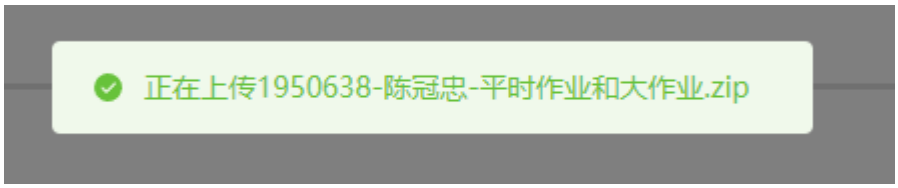
点击“上传文件”功能按钮，打开上传文件窗口



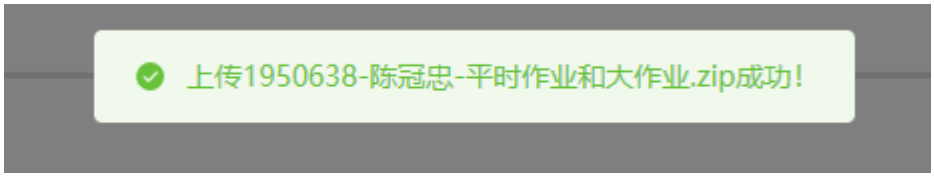
点击“上传”按钮，从系统的文件选择窗口中选择要上传的文件（支持多选）



开始上传



一段时间后，上传完成



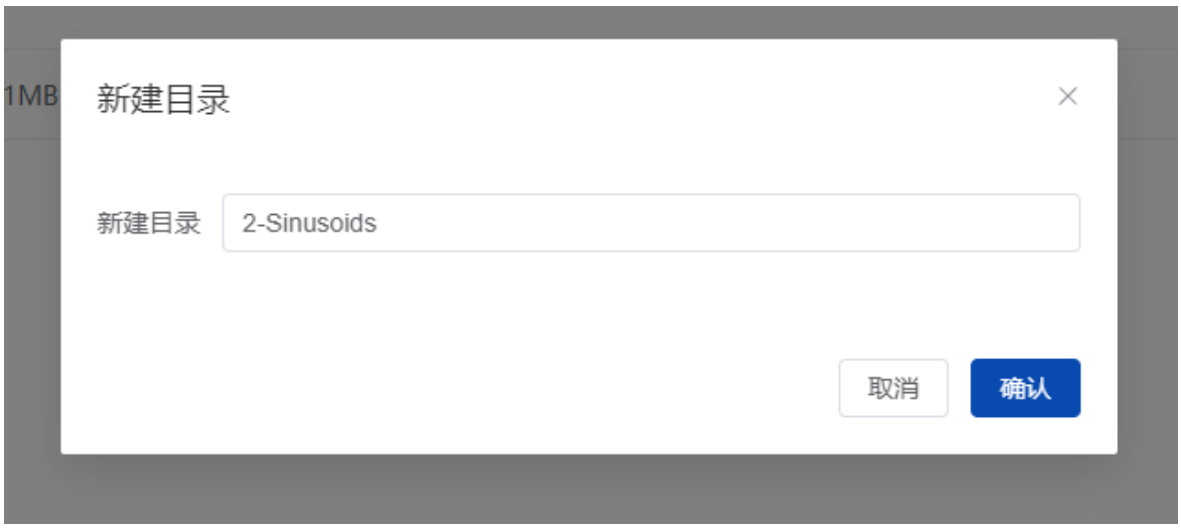
文件出现在相应文件夹下

装 订 线

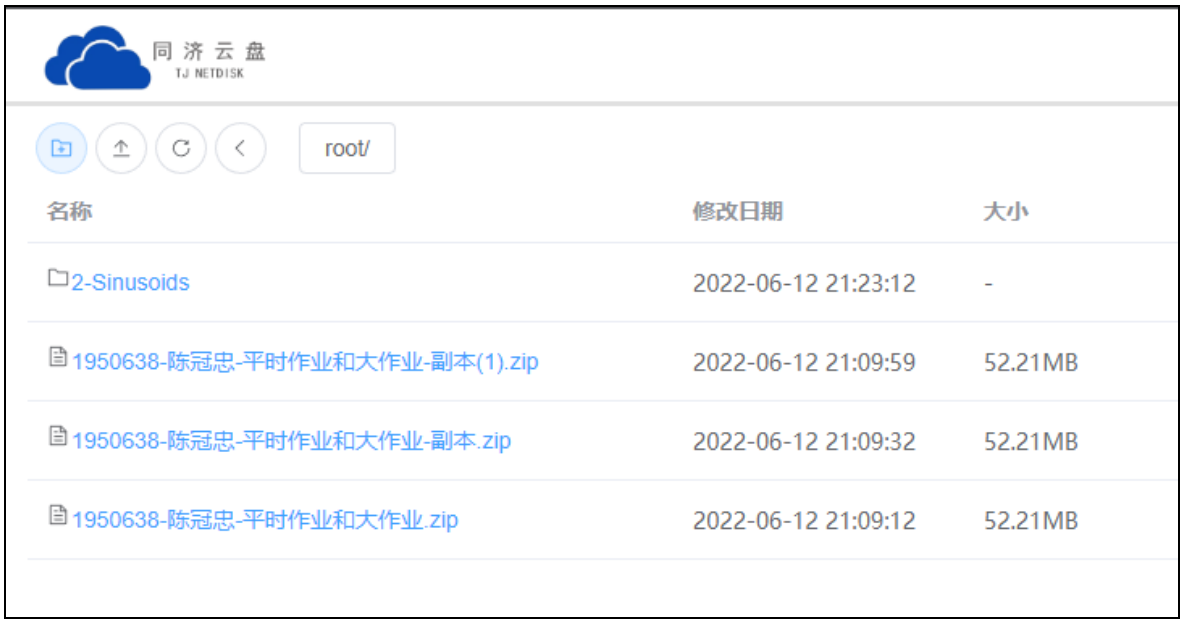
具有秒传功能，若多次上传同一文件，后段识别到相同 md5 码的文件已存在，会直接建立映射关系，不需要再次上传，若相同文件名的文件已存在，新文件会出现“副本”后缀。

新建目录：

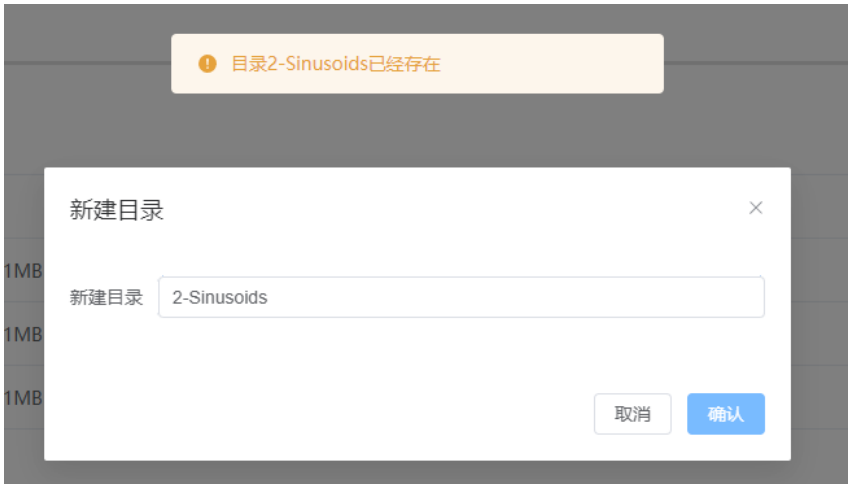
点击“新建目录”功能按钮，打开新建目录窗口，输入目录名称，点击确定



创建成功



再次创建相同目录，给出报错

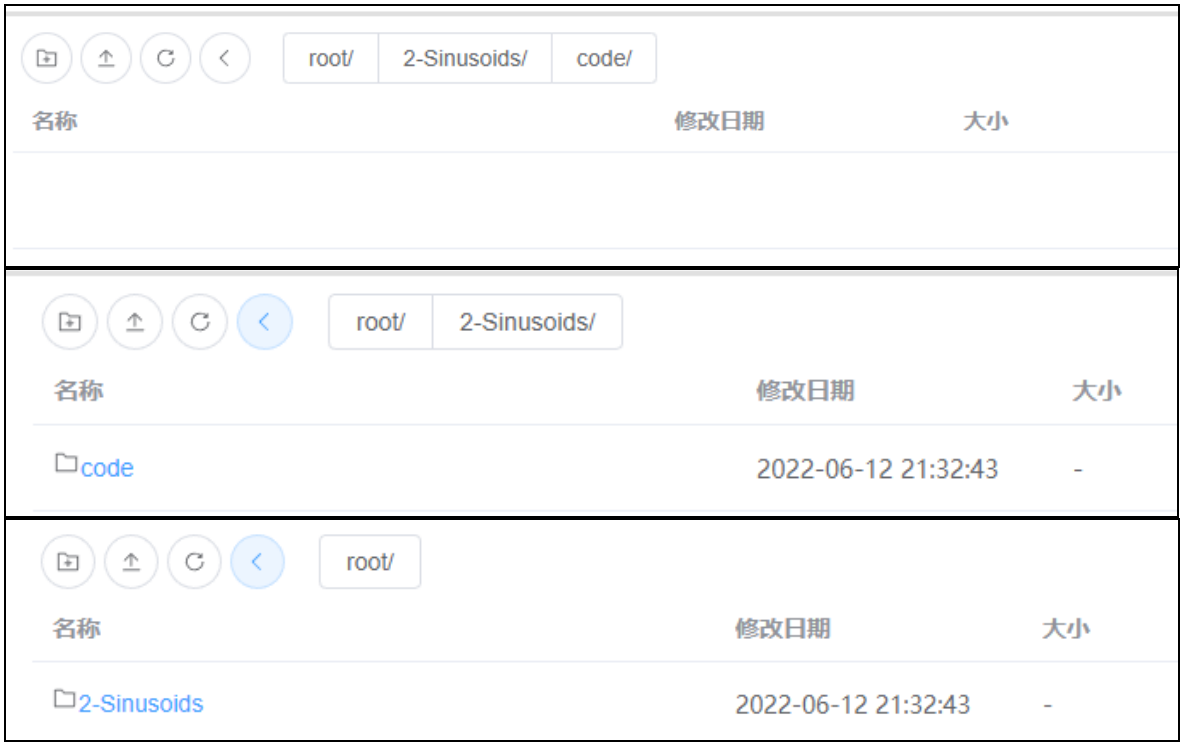


刷新：

“刷新”功能按钮一般使用不到，主要为了应对服务器卡顿的情况

回退：

点击回退按钮，会返回上级目录



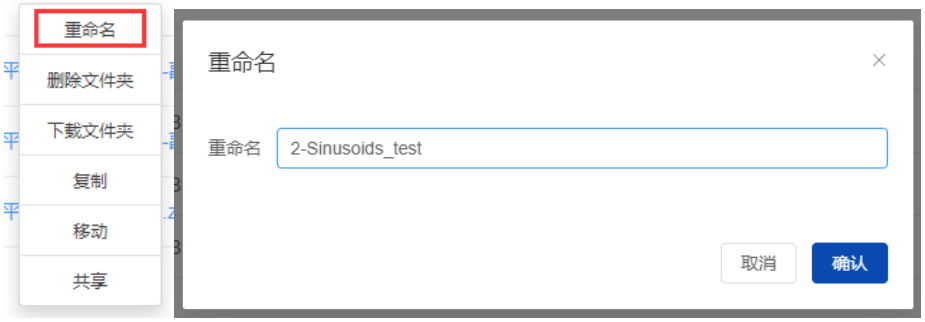
地址栏：

点击地址栏的按钮，能跳转到对应的文件夹下

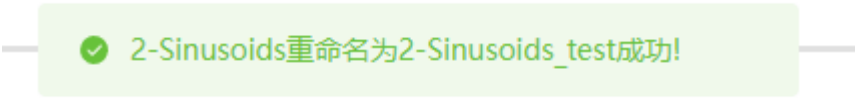
装 订 线

重命名：

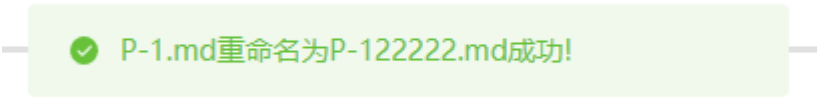
右击文件或文件夹，出现菜单，选择“重命名”，打开重命名窗口，输入新名称



点击“确认”按钮，重命名成功

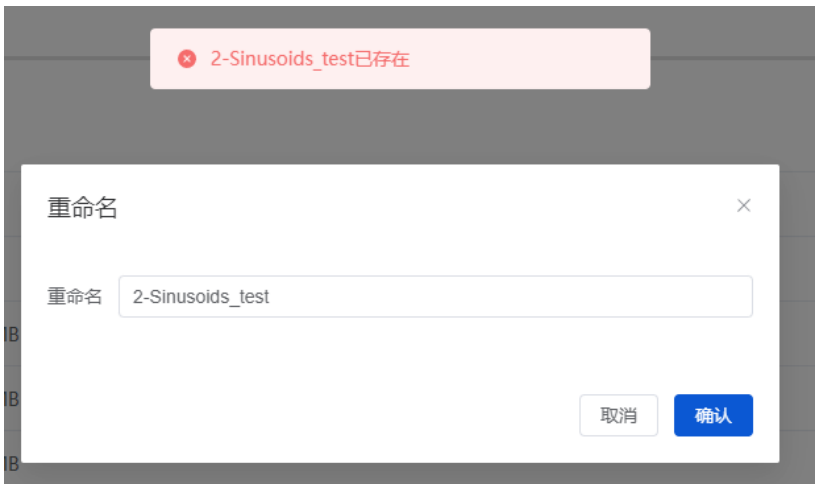


文件重命名也成功



若新名字在当前文件夹下已存在，有错误提示。

装 订 线



删除：

右击文件或文件夹，点击“删除”，删除成功

文件也类似

下载：

右击文件或文件夹，点击“下载”，下载成功

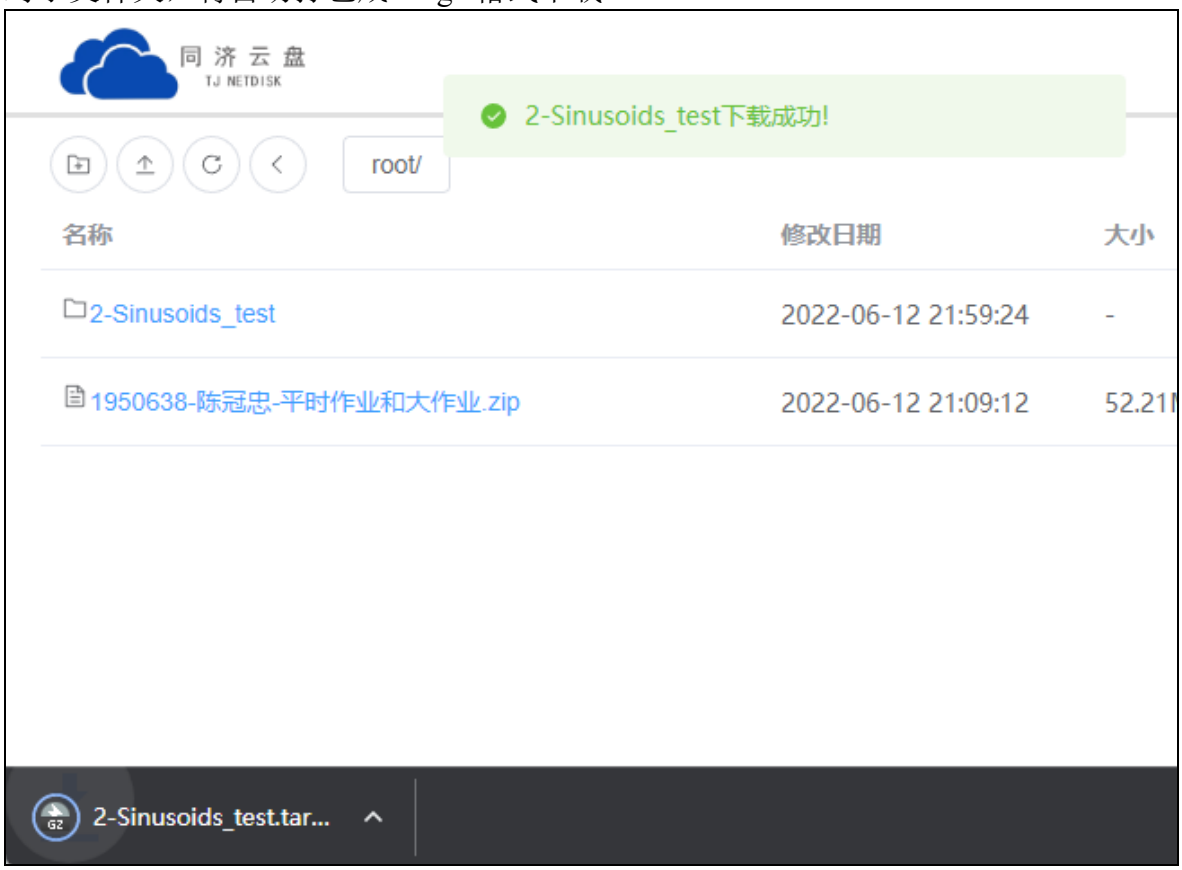

code
P-1.md
P-1.pdf
P-2.md
P-2.pdf
P-3.md
重命名
删除文件
下载文件
复制
移动
共享
同济云盘
TJ NETDISK
P-1.md下载成功!
root/
2-Sinusoids_test/
名称
修改日期
大小
code
2022-06-12 21:32:43
P-1.md
2022-06-12 21:09:08
3.46KB
P-1.pdf
2022-06-12 21:09:08
150.92KB
P-2.md
2022-06-12 21:09:08
5.90KB
P-2.pdf
2022-06-12 21:09:08
277.19KB
P-3.md
2022-06-12 21:09:08
2.20KB
P-1.md

装

订

线

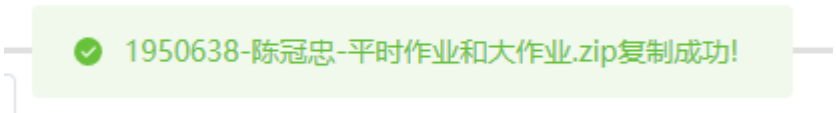
对于文件夹，将自动打包成.tar.gz 格式下载。

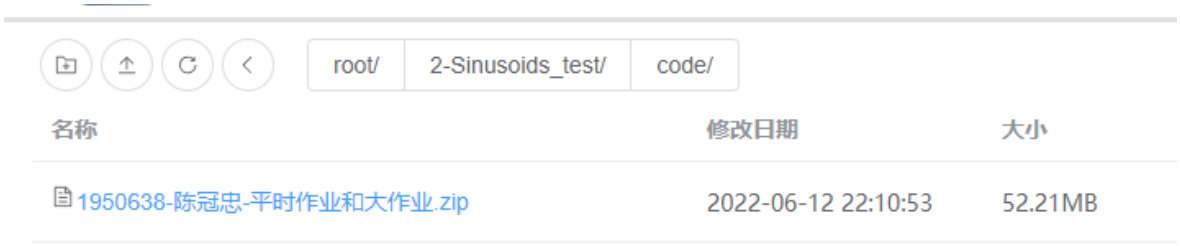


复制：

右击文件或文件夹，点击“复制”，打开复制窗口，选择要复制到的文件夹

以复制到/root/2-Sinusoids_test/code/下为例，复制成功

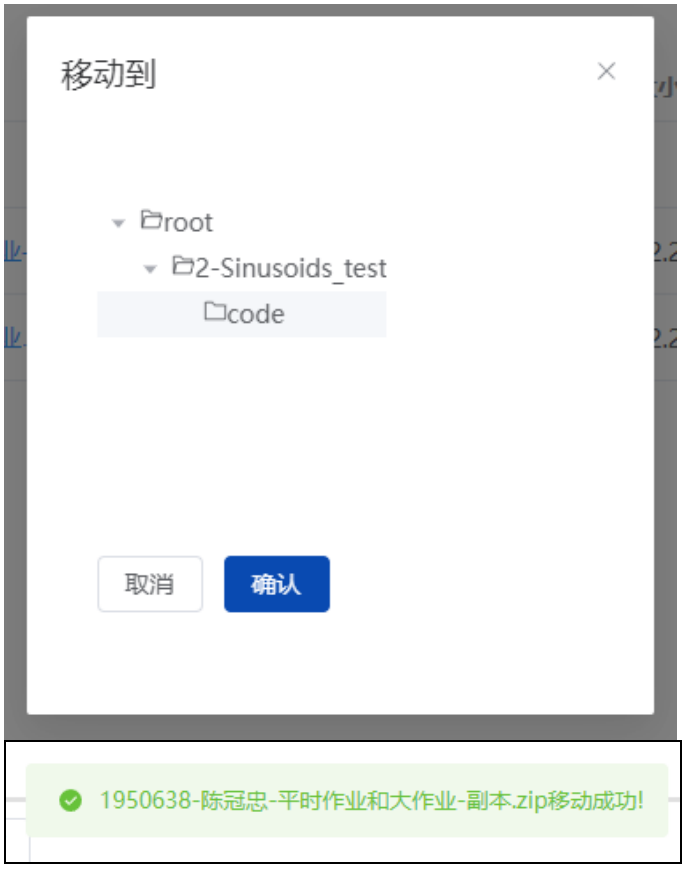




复制到同目录下，与上传功能类似，自动加上“-副本”后缀

移动：

右击文件或文件夹，点击“移动”打开移动窗口，选择要移动到的文件夹，点击确认，移动成功



移动功能与复制类似，会自动加上“-副本”后缀

共享：

右击文件或文件夹，点击“共享”打开分享窗口，可手动复制链接或点击“复制链接”按钮

点击“复制链接”，跳出提示，复制成功。

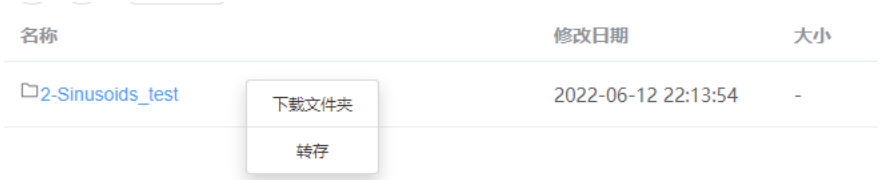
分享页面

在已登录状态下打开分享链接，来到分享页面（已切换至另一账号）

与个人主页相似，能自由浏览分享文件夹下的内容，但根目录不再为”root/”而是”/share”，只能浏览共享文件夹及其子文件夹的内容。

若分享的是文件，则只能浏览一个文件

右击菜单只有下载和转存两个功能，下载功能与个人空间的下载功能完全一致，不再赘述



转存功能与复制功能类似，下面进行演示，选择转存到 mjy 用户的 root/dir/文件夹下

转存成功

回到个人空间，能正常对转存的文件进行各种操作

七、小组分工

- 马家昱：后端所有业务逻辑
- 陈冠忠：前端所有业务逻辑

八、心得体会

本次项目由于只有两人参与，分工明确，沟通成本较低，且采用敏捷开发的方法，开发过程较为高效。该应用最核心的地方是数据库设计，良好的数据库设计使得所有业务逻辑的实现在算法上没有出现重大困难。

我们对所有接口都进行了比较完备的测试，比较全面的考虑了各种情况，修复了许多 bug，使得项目完全可以作为一个轻量级的应用，而不仅仅是一个可以运行的作业。

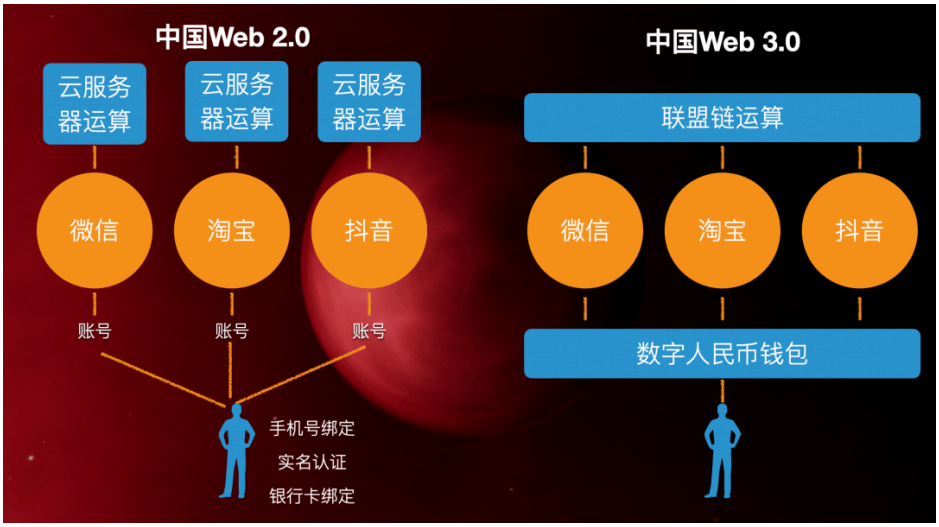
在此次项目中，我们也尝试了许多新的技术。例如，我们尝试将 session 保存在缓存中而不是数据库中，同时尝试了使用非关系型数据库 Redis 作为网站的全局缓存。参考百度网盘分享功能的流程，我们实现了和其几乎一模一样的功能。前端方面，我们自学了 Vue+element UI 的技术组合，并且使用了功能较复杂的组件。

通过此次项目，我们对 Web 应用开发的相关技术，从数据库到前后端框架，都有了更加熟练的掌握水平。同时在掌握原理的基础上，学习和使用新技术的速度越来越快。同时，我们对网站的认识不仅仅停留在业务逻辑上，而是对性能和安全性有了更高的要求，不再像以前一样所有操作都通过查表来实现。

九、Web 技术国内外认知与体会

中国 Web3.0 发展：

目前中国互联网使用云服务器的占比，在 2020 年时已经超过 7 成。大部分互联网公司使用云服务器提供的服务来运营自己的互联网产品，并未各自投入成本建设自己的服务器。将这个云服务器的集体结合成联盟链，即可用较少的成本、时间，完成中国互联网从 Web2.0 一越升级到 Web 3.0 的愿景。



国外 Web 技术发展趋势：

1.Java 发行版

装 订 线

根据 JetBrains 公司的一项调查，我们发现，尽管年代久远，但 Java8 仍然是最受欢迎的版本，而且比例高达受访者的 75%，而 Java11 作为另一个稳定版本，排名第二，比例占受访者的 32%。也就是说，这两个 LTS（长期支持）版本，基本覆盖了所有的 Java 开发者。

2.Spring

到目前为止，Spring 仍然是最流行的 Java 开发框架，并且几乎应用在了所有地方——从流媒体平台到在线购物。它的设计目的是为基于 JavaEE 平台的 Java 应用程序创建后端。Spring 框架基于依赖注入的功能（控件反转 IOC 的一部分），它是在 Java 中构建业务应用程序的理想解决方案：微服务、复杂的数据处理系统、云应用程序或快速、安全且响应迅速的 Web 应用程序。

3.Serverless

另一个持续发展的趋势是由传统巨石后端应用向无服务器架构的迁移，这种趋势将在 2021 年变得更加普遍。Serverles 是一种云服务模型，开发人员只需专注于编码和实现业务，而不必关注基础架构。无服务器架构一词可能会引起误解，意味着不需要服务器。当然，Serverless 解决方案的本身是基于服务器构建的。