文件创建和上传：

1. 登录成功后，前端跳转到主页面，浏览器主页面上展示用户信息，文件目录、文件大小修改时间等信息。用户在前端页面上可以浏览已上传的文件信息，也可以在搜索框里输入关键字搜索文件。
2. 若要新建文件夹，用户需点击“新建文件夹”按钮并输入文件名即可新建文件夹；若要上传文件，用户点击“上传文件”按钮并选择需要上传的文件即可上传文件。
3. 若要上传文件，用户需点击“上传文件”的按钮，就会弹出一个弹窗，用户点击“选择文件”按钮可以选择要上传的文件，选择好后再弹窗内可以看到已经选择的文档。点击“确认上传”就可以将选择好的文件上传。在上传时用户可以看到上传进度。
4. 创建文件夹或上传文件后用户可在浏览器页面上看到新创建的文件夹或新上传的文件信息。
5. 上传文件时文件分为两种类型，一种是目录文件，另一种是普通文件。
6. 前端浏览器获取文件基本信息：文件名、文件大小、文件类型、ctime（最后一次访问时间）、mtime（最后一次修改时间）和atime（最后一次对文件属性改变）。
7. 上传目录文件时，先生成128位密钥对文件名、ctime、mtime和atime进行AES加密，使用主密钥对生成的128位密钥进行AES加密，将加密的密钥、文件名、ctime、mtime和atime等信息打包发送到后端。
8. 上传普通文件时，前端浏览器读取文件前128 KB数据利用SHA-256生成256位哈希值，截取前128位作为文件密钥，对文件名、ctime、mtime和atime进行AES加密，使用主密码对128位密钥进行AES加密。
9. 将文件按每块1MB的大小分块，每1MB数据块使用SHA-1生成fingerprint用于表示文件块名称，再利用文件密钥和AES算法对文件块内容加密进行加密。
10. 将文件加密的密钥、文件名、ctime、mtime和atime、fingerprint和加密的文件块等信息打包发送到后端。
11. 后端接收前端发送的信息，并将文件名、文件类型、文件大小、atime、ctime、mtime和主密钥加密的密钥，储存到文件系统表中，在储存的同时数据库生成uint64位的inode值用于文件唯一标识。将第几块、fingerprint、块大小等信息存入文件块信息表，数据库生成blockId并将对应的inode存入。将fingerprint、加密的文件块存入Ceph系统中。

更新文件大小和对应数据块fingerprint到数据块中。

文件上传流程的实现：

前端页面：

电脑软件截图

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

浏览器页面实现：

最上层（顶部），在左侧显示有“首页”的标题，右侧显示用户的头像和用户名。点击用户的头像就显示隐藏按钮，“设置”和“退出登陆”。

<template>

<el-card class="box-card">

<a href="javascript:void(0)" class="nav first">首页</a>

<a href="javascript:void(0)" class="nav"></a>

<a href="javascript:void(0)" class="nav"></a>

<span style="float: right;" class="name">{{ name }}</span>

<el-dropdown class="avatar">

<span class="el-dropdown-link">

<img src="../assets/avatar.svg" alt="" slot="reference">

</span>

<el-dropdown-menu slot="dropdown">

<el-dropdown-item>设置</el-dropdown-item>

<el-dropdown-item divided><a @click="loginout">退出登陆</a></el-dropdown-item>

</el-dropdown-menu>

</el-dropdown>

</el-card>

</template>

左侧的功能栏，有全部文件、图片、文档、视频、音乐和其他六个选项。目前除了全部文件其他功能尚未完善。

<template>

<el-container>

<el-header>

<el-menu

default-active="2"

class="el-menu-vertical-demo"

>

<el-menu-item index="2">

<i class="el-icon-tickets"></i>

<span slot="title">全部文件</span>

</el-menu-item>

<el-menu-item index="3" class="item">

<i class="el-icon-picture"></i>

<span slot="title">图片</span>

</el-menu-item>

<el-menu-item index="3" class="item">

<i class="el-icon-document"></i>

<span slot="title" class="item">文档</span>

</el-menu-item>

<el-menu-item index="3">

<i class="el-icon-video-camera"></i>

<span slot="title" class="item">视频</span>

</el-menu-item>

<el-menu-item index="3">

<i class="el-icon-service"></i>

<span slot="title" class="item">音乐</span>

</el-menu-item>

<el-menu-item index="3">

<i class="el-icon-setting"></i>

<span slot="title" class="item">其他</span>

</el-menu-item>

</el-menu>

</el-header>

<el-main></el-main>

<el-footer>

<el-progress :text-inside="false" :stroke-width="10" :percentage="used" :show-text="false"></el-progress>

<el-breadcrumb separator="/" style="margin-top: 5px;font-size: 12px;">

<el-breadcrumb-item>{{ used }}MB</el-breadcrumb-item>

<el-breadcrumb-item>{{ total }}MB</el-breadcrumb-item>

</el-breadcrumb>

</el-footer>

</el-container>

</template>

右部文件展示区，顶端有“文件上传”、“新建文件夹”两个按钮和搜索栏。下面会显示文件位置。当位于根目录时隐藏按钮“上级目录”、“返回根目录”。再下面有三列信息分别是：文件名、大小和修改时间。

<template>

<div>

<el-header>

<UploadFile id="uploadfile" :parentInode="curInode"></UploadFile>

<el-button @click="addfolder = true" class="el-icon-folder-add">新建文件夹</el-button>

<div id="search"><input placeholder="请输入内容" class="search" v-model="keywords"/><i class="el-icon-search"></i>

</div>

<div style="height: 15px"></div>

</el-header>

<div class="middle-wrapper" style="padding: 10px">

<!-- 面包屑导航栏 -->

<el-breadcrumb separator-class="el-icon-arrow-right">

<el-breadcrumb-item :to="{ path: '/Home' }">首页</el-breadcrumb-item>

<el-breadcrumb-item><a href="/">文件夹一</a></el-breadcrumb-item>

</el-breadcrumb>

</div>

<el-dialog

title="输入文件夹名字"

:visible.sync="addfolder"

width="20%"

<el-table-column prop="dir" width="60" align="center">

</el-table-column>

<el-table-column prop="dir" width="60" align="center">

<template slot-scope="scope">

<img class='image' v-if="scope.row.type==='FILE'" src="../assets/afile.png"

style="height: 30px;max-height: 100%;max-width: 100%">

<img class='image' v-else src="../assets/folder.png" style="height: 30px;max-height: 100%;max-width: 100%">

</template>

</el-table-column>

<el-table-column

prop="filename"

label="文件名"

width="300">

</el-table-column>

<el-table-column

prop="size"

label="大小"

width="270">

</el-table-column>

<el-table-column

prop="mtime"

label="修改时间"

width="220">

</el-table-column>

当新建文件夹时，用户点击“新建文件夹”按钮后就会弹出一个弹窗，用户需要填写文件夹的名称点击确定即可新建文件夹，点击取消即可取消新建文件夹。

>

<el-input v-model="input" placeholder="请输入内容"></el-input>

<span slot="footer" class="dialog-footer">

<el-button @click="addfolder = false">取 消</el-button>

<el-button type="primary" @click="newFolder">确 定</el-button>

</span>

</el-dialog>

<el-table

:data="tableData"

style="width: 100%;margin-bottom: 20px;"

row-key="inode"

>

文件上传弹窗实现代码：

当点击文件上传按钮时，就会执行UploadFile.vue文件。浏览器页面就会显示弹窗。弹窗的标题是“上传”，有两个按钮“上传文件”和“确认上传”。点击上传文件按钮就会弹出本地的文件目录选择要上传的文件。选好的文件就会再弹窗内显示出来，用户可选择上传一个或同时上传多个文件。点击确认上传就会上传文件，并且会显示上传进度。

<div class="file-up">

<el-button type="primary" size="medium" icon="el-icon-upload" @click="dialogVisible = true">上传文件</el-button>

<el-dialog

title="上传"

:visible.sync="dialogVisible"

width="40%"

v-on:open="openSocket"

v-on:close="closeSocket"

>

<div>

<template>

<el-button type="primary" class="file">选择文件

<input type="file" name="" @change="fileOnchange($event)"> <span id="filename"

style="color: red"></span>

</el-button>

</template>

<el-button class="ml-3" type="success" @click="uploadFileFun">

确认上传

</el-button>

</div><

<br>

<el-progress :text-inside="true" :stroke-width="20" :percentage="SpeedOfProgress" :status="status"></el-progress>

<div style="margin: 20px 0;"></div>

<el-input

id="message\_content"

type="textarea"

:autosize="{ minRows: 6, maxRows: 15}"

placeholder="上传文件块统计框"

v-model="textarea"

ref="textarea"

>

上传文件时，上传目录文件和上传普通文件过程相同，在这以上传普通文件为例子进行流程讲解。首先在打开文件上传的弹窗时，前端先建立一个websocket链接，以便进行文件上传。当上传成功或关闭弹窗时就会断开链接。

// 打开上传框的时做的事件

openSocket() {

// 这里面存有 用户的 id

this.SpeedOfProgress = 0

let userid = JSON.parse(localStorage.getItem("userid"));

if (userid == null) {

userid = "1"

}

var webSocketUrl = 'ws://127.0.0.1:8081/cloud/upload/';//websocketi连接地址 //ws://127.0.0.1:8080/upload/block

if (!this.socket || this.socket) {//避免重复连接

this.socket = new WebSocket(webSocketUrl + userid);

console.log(this.socket)

this.socket.onopen = function () {

console.log("websocket已连接");

};

this.socket.onmessage = function (e) {

if (this.uploadFlag) {

//服务端发送的消息

this.textarea += (e.data + '\n');

}

};

this.socket.onclose = function () {

console.log("websocket已断开");

this.dialogVisible = false;

}

}

},

// 关闭上传框的时做的事件

closeSocket() {

if (this.socket != null) {

console.log("端口连接")

this.socket.close();

}

this.textarea = "";

this.socket = null;

}

}

}

上传文件时，首先要获取文件的基本信息，建立一个对象获取文件信息：

fileOnchange(e) {

// 获取文件对象

this.fileObject = e.target.files[0];

console.log(this.fileObject)

this.textarea += (this.fileObject.name + "\n")

this.textarea += ("共计" + Math.ceil(this.fileObject.size / this.paragraph) + "部分\n")

}

读取文件的前128KB数据利用SHA-256生成256位密钥，截取前128位作为文件的文件密钥：

//读取文件前128 KB数据利用SHA-256生成256位密钥

var v = await this.readAsBinaryString(filedata, 0, 1024 \* 128);

const sha256Key = await crypto.subtle.digest('SHA-256', v)

var fileKey = new Uint8Array(sha256Key).subarray(0, 16);

将在登陆是储存在浏览器的用户随机数取出来，用于接下来的文件加密：

// 获取用户随机数

var fileSize = filedata.size;

console.log("file size:" + fileSize);

let clientRandomValue = this.stringtoUint8Array(localStorage.getItem('clientRandomValue'));

将文件名、上次修改时间和文件密钥加密，加密方法为AES加密，与注册时的加密方法相同：

// 文件名加密

var fileName = filedata.name;

let data1 = this.stringtoUint8Array(fileName)

var encryptedMasterKeyHashValue1 = await this.encryptKey(fileKey, clientRandomValue, data1)

var encryptedData1 = new Uint8Array(encryptedMasterKeyHashValue1);

// 上次修改时间

var Mtime = filedata.lastModifiedDate;

var data2 = this.stringtoUint8Array(Mtime.toString())

console.log("Mtime:" + data2);

var encryptedMasterKeyHashValue2 = await this.encryptKey(fileKey, clientRandomValue, data2)

var encryptedData2 = new Uint8Array(encryptedMasterKeyHashValue2)

//对密钥加密

let masterKey = this.stringtoUint8Array(localStorage.getItem('masterKey'));

var encryptedMasterKeyHashValue = await this.encryptKey(masterKey, clientRandomValue, fileKey)

var encryptedkey = new Uint8Array(encryptedMasterKeyHashValue)

由于后端只接受字符串类型的数据，在前端要先定义一个字符串的json对象方便后端解析，将加密的文件名、文件大小、文件有几块、修改时间、加密的文件密钥、用户名和文件父目录的inode打包发送到后端：

//后台只接收字符串类型，我们定义一个字符串的json对象给后台解析

let fileJson = {

opt: "fileUpload",

data: {

filename: \_this.uint8ArrayToString(encryptedData1),

size: filedata.size,

blockSize: blockSize,

mtime:\_this.uint8ArrayToString(encryptedData2),

fileKey: \_this.uint8ArrayToString(encryptedkey),

userId: localStorage.getItem("uid"),

parentInode: this.parentInode

}

};

在正式上传文件前先向后端发送文件名，后端接受到文件名后才会正式开始传输文件：

//后台接收到文件名以后会正式开始传输文件

console.log("向后台发送消息，请求上传数据")

this.socket.send(JSON.stringify(fileJson));

正式上传文件时，若小于1MB的文件，直接加密并上传。若文件大于1MB，就需要用readAsArrayBuffer();函数对文件进行分块，同时计算这是文件的第几块，将分块的文件利用SHA-1生成fingerprint，作为每一个小分块文件块名。将fingerprint、idx（第几块）和分块大小打包成json对象发送到后端。

// 上传块元数据

if (tip.opt === 'blockMetadata') {

console.log("上传块元数据")

if (tip.next >= blockSize) {

\_this.socket.send(JSON.stringify({opt: "over"}))

\_this.SpeedOfProgress = 100

} else {

\_this.SpeedOfProgress = Math.floor(tip.next / blockSize \* 100)

// 代表第几个块

var start = tip.next \* paragraph;

var end = start + paragraph > filedata.size ? filedata.size : start + paragraph;

var block = filedata.slice(start, end);

var reader = new FileReader();

reader.readAsArrayBuffer(block);

reader.onload = async function loaded(evt) {

var arrayBuffer = evt.target.result;

console.log(arrayBuffer);

\_this.textarea += ("发送文件第" + tip.next + "部分" + '\n');

// 加密块，生成块的指纹

console.log(new Uint8Array(arrayBuffer))

var fingerprint = await crypto.subtle.digest('SHA-1', new Uint8Array(arrayBuffer))

fingerprint = new Uint8Array(fingerprint).subarray(0, 20)

console.log("--------------" + fingerprint)

fingerprint = \_this.uint8ArrayToString(fingerprint)

let fileObjJson = {

opt: "block",

data: {

fingerprint: fingerprint,

idx: tip.next,

size: end - start

}

};

\_this.socket.send(JSON.stringify(fileObjJson));

}

}

接下来就是上传文件块数据本身，同样将文件进行分块，并将文件进行加密并将文件打包成json对象，发送到后端。

// 上传块数据本身

} else if (tip.opt === 'blockData') {

console.log("上传块数据")

var start = tip.next \* paragraph;

var end = start + paragraph > filedata.size ? filedata.size : start + paragraph;

var block = filedata.slice(start, end);

var reader = new FileReader();

reader.readAsArrayBuffer(block);

reader.onload = function loaded(evt) {

var arrayBuffer = evt.target.result;

\_this.socket.send(arrayBuffer)

}

} else if (tip.opt === 'over') {

console.log("上传成功")

\_this.socket.send(JSON.stringify({opt: "over"}))

}

文件分块读取的函数readAsArrayBuffer(block)如下:

//分块读取文件

async readAsBinaryString(file, start, end) {

return new Promise((resolve, reject) => {

var reader = new FileReader();

var blob = file.slice(start, end);

reader.readAsArrayBuffer(blob);

reader.onload = (e) => {

let num = e.target.result

let v = new Uint8Array(num);

console.log(v)

resolve(v);

};

将所有文件上传后前端页面的弹窗上就会显示上传成功，同时前端将之前建立的socket链接断开。

else if (tip.opt === 'over') {

console.log("上传成功")

\_this.socket.send(JSON.stringify({opt: "over"}))

}

}

this.socket.onclose = function () {

alert("后端已经断开连接")

this.socket = null

this.dialogVisible = false

}

在前端创立连接时后端触发openSession()函数，与前端建立连接：

    @OnOpen

    public void openSession(Session session, @PathParam("userId") Long userId) {

        this.session = session;

        blockWebsockets.put(userId, this);

        addOnlineCount();

        log.info("创建连接:"+"当前连接数"+onlineCount);

    }

后端接收前端发送的文件名后，前端正式开始文件上传。后端接收前端发来的字节流，并将接收到的文件信息保存到后端。

@OnMessage

    public void onMessage(String message, @PathParam("userId") String userId) {

        log.info("字符串消息" + message);

        // 前端传进来一个字符串信息

        TransVo transVo = JSONObject.parseObject(message, TransVo.class);

        String opt = transVo.getOpt();

        // 文件上传开始

        if (FILE\_UPLOAD.equals(opt)) {

            FileDto fileDto = new FileDto();

            BeanUtil.copyProperties(transVo.getData(),fileDto,false);

            // 保存文件信息

            FilePo filePo = new FilePo();

            BeanUtil.copyProperties(fileDto,filePo,true);

            filePo.setState(FileState.NEW);

            filePo.setType(FileType.FILE);

            filePo.setParentDir(new FilePo(fileDto.getParentInode()));

            filePo.setUser(new UserPo(fileDto.getUserId()));

            this.filePo = fileService.save(filePo);

            try {

                this.sendMessage(JSONObject.toJSONString(new BlockVo("blockMetadata",0)));

            } catch (IOException e) {

                e.printStackTrace();

            }

        } else if (BLOCK\_UPLOAD.equals(opt)) {

在上传文件过程中要实时检查文件的上传状态，若后端还在接收文件块，就将状态改为正在上传的状态：

          // 检查文件状态是否为新建状态，将其改为上传状态

if (filePo.getState() == FileState.NEW) {

                filePo.setState(FileState.UPLOADING);

                fileService.save(filePo);

            }

将接收到的文件块信息保存到数据库中。检查当前接收到的文件是否在缓存中，若在缓存中，就代表该文件块已经上传过，就无需再上传一次。

            FileBlockPo blockPo = new FileBlockPo();

            BeanUtil.copyProperties(transVo.getData(),blockPo,false);

            blockPo.setParentFilePo(this.filePo);

            blockPo.setFingerprint(Base64.encode(blockPo.getFingerprint()));

            // 将块元数据保存到数据库

            this.blockPo = blockService.save(blockPo);

            // 检查 当前块的 数据是否已经存在与redis中

            String size = redisTemplate.opsForValue().get(blockPo.getFingerprint());

            // 如果当前块存在于缓存中，表示当前数据已经存在，不需要进行接下来的上传操作

            if (size != null) {

                try {

                    redisTemplate.opsForValue().set(blockPo.getFingerprint(), String.valueOf(Integer.parseInt(size) + 1));

                    sendMessage(JSONObject.toJSONString(new BlockVo("blockMetadata",this.blockPo.getIdx()+1)));

                } catch (IOException e) {

                    e.printStackTrace();

                }

                // 不存在与缓存中

            }else {

                redisTemplate.opsForValue().set(blockPo.getFingerprint(), String.valueOf(1));

                try {

                    sendMessage(JSONObject.toJSONString(new BlockVo("blockData",this.blockPo.getIdx())));

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        } else if (END\_UPLOAD.equals(opt)) {

            filePo.setState(FileState.UPLOADED);

            fileService.save(filePo);

            try {

                sendMessage(JSONObject.toJSONString(new BlockVo("uploaded",null)));

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

    }

前端将文件上传成功后断开链接时，后端触发closeSession()函数断开链接：

    @OnClose

    public void closeSession() {

        subOnlineCount();

        log.info("关闭socket通信:"+"当前连接数"+onlineCount);

    }

将文件块信息存到Ceph分布式形同中：

    @Override

    public boolean uploadBlock(FileBlockPo fileBlockPo) throws Exception {

        List<Bucket> buckets = amazonS3.listBuckets();

        InputStream inputStream =new  ByteArrayInputStream(fileBlockPo.getData());

        PutObjectResult putObjectResult = amazonS3.putObject(bucketName, fileBlockPo.getFingerprint(), inputStream, null);

        return putObjectResult.isRequesterCharged();

    }