# OS_文件管理

2253206 韩明洋

# 1 项目分析

## 1.1 项目背景

文件管理是指对计算机系统中存储的文件进行管理、维护和组织的一种操作。随着计算机技术的不断发展，人们在日常生活和工作中所需处理的文件数量和种类也越来越多，因此对文件进行有效的管理和分类显得尤为重要。文件管理软件通常提供了许多功能，包括浏览文件、搜索文件、复制、剪切、粘贴、重命名、删除、创建新文件夹等。这些功能使得用户能够更加方便地管理和组织文件，并提高工作效率。同时，文件管理也有助于保护计算机系统中的数据安全，避免数据丢失或被损坏。

本项目的目的为：

- 理解文件存储空间的管理；
- 掌握文件的物理结构、目录结构和文件操作；
- 实现简单文件系统管理；
- 加深文件系统实现过程的理解；

## 1.2 项目需求

在内存中开辟一个空间作为文件存储器，在其上实现一个简单的文件系统;退出这个文件系统时，需要该文件系统的内容保存到磁盘上，以便下次可以将其回复到内存中来。

- 文件存储空间管理可采取链接结构（如 FAT 文件系统中的显式链接等）或者其他学过的方法；
- 空闲空间管理可采用位图或者其他方法；
- 文件目录采用多级目录结构，目录项目中应包含：文件名、物理地址、长度等信息。

# 2 开发环境

开发语言：Javascript + html + css

开发框架：Vue.js 3.0 + Element-plus

开发工具：Vue-cli、Vue-devtools、VScode、Edge

# 3 实现方法

引入 Element-plus 组件作为 UI，采用 Vue3 框架进行组件化开发

采用 JS 在内存中开辟的空间作为文件资源管理器所需的内存部分，使用浏览器缓存（localStorage）作为外部磁盘，将文件写入的数据存至其中。

在退出资源管理器时(即关闭浏览器页面)将必要的目录文件结构(如位图、文件目录等）也一并存入浏览器缓存中，模拟关闭系统时文件存入磁盘。在下次访问时从浏览器缓存中读取目录文件结构数据，模拟进入系统后从磁盘中取出文件目录等操作。

# 4 How To Run

解压作业压缩包，用 VScode 打开，并开启终端

环境配置：

若未安装 Node.js，需要先安装 Node.js（https://nodejs.cn/download/）

若未安装 Vue cli，在终端输入 npm install -g @vue/cli

运行 npm install 安装项目依赖包

运行 npm run build 构建发布版本

在根目录下会出现**dist**文件夹，使用浏览器打开其中的 **index.html** 即可浏览

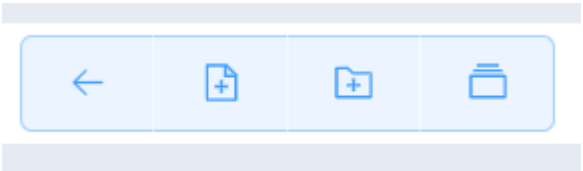
# 5 界面设计

1.打开界面：
显示磁盘 D
右侧会有操作提示


2. 双击进入磁盘

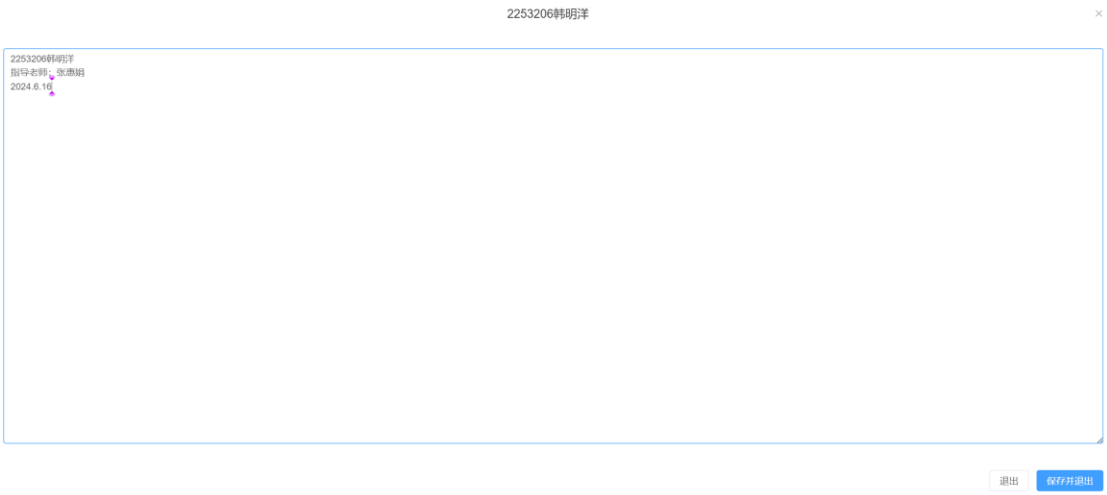

里面已经有作者创建的两个文件夹

3. 主要操作按钮



分别为：退回上一级　　新建记事本文本文件　　新建文件夹　　显示磁盘块使用情况

4. 双击进入文件夹再双击进入文本文件




可以在里面编辑文本，右下角有退出和保存并退出两个选项

5. 点击查看磁盘块使用情况

磁盘块使用情况

| 块号 | 存放内容 | 下一块号 |
| --- | --- | --- |
| 0 | 2253206韩明洋 | -1 |
| 1 | | -1 |
| 2 | | -1 |
| 3 | | -1 |
| 4 | | -1 |
| 5 | | -1 |
| 6 | | -1 |

# 6　主要功能设计及算法实现

## 6.1 管理方式

1. 文件目录——多级目录
2. 文件存储空间管理——链接结构
3. 空闲空间管理——位图

## 6.2 新建文件

1. 由于文件目录以多级目录方式组织，因此直接遍历树找到当前目录下新建的

文件所属的直接父文件夹（父结点）

```javascript
  let dir = this.totol_dir; //保存当前层的情况
      //这里的 i 相当于深度
      for (let i = 0; i < this.cur_path.length - 1; ++i) {
        //这里的 j 是为了遍历每一层的子结点
        for (let j = 0; j < dir.length; ++j) {
          if (dir[j].name == this.cur_path[i] && dir[j].type != 1) {
            //一定可以找到一个结点
            dir = dir[j].children; //dir 定位到前一层
          }
        }
      }
      //查询当前层哪个结点为要添加结点的父节点
      let index = -1;
      console.log(222222);
      console.log(dir);
      for (let i = 0; i < dir.length; ++i) {
        if (
          dir[i].name == this.cur_path[this.cur_path.length - 1] &&
          (dir[i].type == 2 || dir[i].type == 0)
        ) {
          index = i;
        }
      }
```

2. 检查文件名是否冲突及合法性（同一目录下已存在同类型同名文件）
    1. 若非法则输出提示，等待新的文件名输入
    2. 若合法则进行下一步

```javascript
    //检测名字合法性
        if (this.new_doc_name == "") {
          if (this.new_doc_type == 0) {
            ElMessage.error("文件夹名不能为空！");
          } else {
            ElMessage.error("文件名不能为空！");
          }
          return;
        }
        //检测是否重名
        for (let i = 0; i < dir[index].children.length; ++i) {
          if (
            dir[index].children[i].name == this.new_doc_name &&
```

```
            dir[index].children[i].type == this.new_doc_type
          ) {
            if (this.new_doc_type == 0) {
              ElMessage.error("已存在同名文件夹！");
            } else {
              ElMessage.error("已存在同名文件！");
            }
            this.new_doc_name = "";
            this.new_doc_type = -1;
            return;
          }
        }
```

3. 生成文件其他信息，在文件目录下创建相应 FCB

```javascript
    //生成文件路径
        let new_doc_path = "";
        for (let i = 0; i < this.cur_path.length; ++i) {
          new_doc_path += this.cur_path[i] + "\\";
        }
        let doc_size = 0;
        if (this.new_doc_type == 0) {
          doc_size = "-";
        }
        dir[index].children.push({
          name: this.new_doc_name,
          last_edit_timestr: this.new_doc_timestr,
          last_edit_time: this.new_doc_time,
          type: this.new_doc_type,
          size: doc_size,
          path: new_doc_path,
          children: [],
          p_begin: -1, //非文件类型不需要设置此项
          p_end: -1, //非文件类型不需要设置此项
        });
        dir[index].last_edit_timestr = this.new_doc_timestr;
        dir[index].last_edit_time = this.new_doc_time;
        this.cur_dir = [].concat(dir[index].children); //更新当前目录
      } else {
        alert("下标查找有问题");
        return;
      }
```

```
      this.new_doc_name = "";
      this.new_doc_type = -1;
      this.show_dialog = false;
      ElMessage({
        message: "创建成功！",
        type: "success",
      });
      return;
    },
```

4. 创建成功（P.S windows 11 文件管理系统不会在创建文件时要求输入文本内容，因此此处也不要求，初始化文件大小均为 0 KB）

## 6.3 打开文件

根据前端传入的打开文件名称，查找文件目录表，根据其对应的 p_begin，p_end 指针从”外部磁盘”中读取数据

```javascript
  openFile(index) {
      this.open_doc_name = this.cur_dir[index].name;
      this.open_doc_index = index;
      this.show_content = true;
      this.open_doc_content = this.readDisk(
        this.cur_dir[index].p_begin,
        this.cur_dir[index].p_end
      );
    },
```

由于打开文件事件由鼠标双击触发，因此此处传入的文件名称必然合法，且一定在 FCB 中存在（若不存在则不会显示在页面上，自然 无法被点击），因此无需检测文件名

下面介绍 readDisk() 函数，其主要功能为根据文件块指针从“磁盘”读取数据

判断 p_begin 是否为 -1 ，若为 -1 则说明是一个空文件，直接返回即可；若不为 -1，则从起始块读数据，依据起始块中的指向下一块的指针找到下一块……直至读取完毕，即 p_begin 与 p_end 相等

```javascript
```

```
  //从磁盘中读出数据
    readDisk(p_begin, p_end) {
      if (p_begin == -1) {
        //说明为新文件
        return;
      }
      let content = "";

      let p_cur = p_begin;
      while (1) {
        content += this.physical_disk[p_cur].content;
        if (p_cur == p_end) {
          break;
        }
        p_cur = this.physical_disk[p_cur].disk_next;
      }
      return content;
    },
```

## 6.4 保存文件

将文件数据存入磁盘，若磁盘有剩余空间则返回 true，记录相关信息并输出提示；若无剩余空间则输出提示，等待文本内容修改

```javascript
  saveDoc() {
      if (this.writeOutDisk()) {
        //存储至“磁盘”—若成功则返回 true，空间不足则返回 false
        this.cur_dir[this.open_doc_index].size =
          this.open_doc_content.length / 1024;
        let a = new Date().toLocaleDateString();
        this.cur_dir[this.open_doc_index].last_edit_time = new Date(a) /
1000;
        this.cur_dir[this.open_doc_index].last_edit_timestr = a;
        ElMessage({
          message: "更改已保存！",
          type: "success",
        });
        this.show_content = false;
      } else {
        ElMessage.error("磁盘空间不足！");
```

```
      }
    },
```

下面主要介绍下 writeOutDisk()函数，其主要功能为根据文件块指针从“磁盘”读取数据

1. 查看 p_begin 是否为-1，若为-1 则说明该文件之前在磁盘上没有存储的数据，则直接进行第 3 步
2. 若 p_begin 不为-1，则说明该文件之前在磁盘上存储有数据。此处设计为将磁盘上该文件的旧数据擦除，再存储新数据

```` javascript
  if (this.cur_dir[this.open_doc_index].p_begin != -1) {
        //说明不是第一次写 则删除
        this.deleteFromDisk(
          this.cur_dir[this.open_doc_index].p_begin,
          this.cur_dir[this.open_doc_index].p_end
        );
        //修改目录项
        this.cur_dir[this.open_doc_index].p_begin = -1;
        this.cur_dir[this.open_doc_index].p_end = -1;
    }
```
````

3. 计算保存的数据所需要的磁盘块数，并将数据按块大小分隔成相应份

```javascript
    let size = this.open_doc_content.length; //假设 1 个字符占 1 个字节
      let block_size = this.disk_bitmap[0].bolck_size;
      let block_need_num = Math.ceil(size / block_size);
      var block_content_ary = []; //将字符串按块能存储的最大长度切割，子串存至数组中
      for (let i = 0; i < this.open_doc_content.length; i += block_size) {
        block_content_ary.push(this.open_doc_content.slice(i, i + block_size));
      }
```

4. 通过位图磁盘是否有剩余空间，若不足，则返回 false 输出提示并等待文本内容修改；若充足则进行磁盘块空间分配。分配规则为通过位图寻找前 N 个（假设需要 N 块）空闲块，依次将数据存入，并建立块指针链接。完成后返回 true

```javascript
    //判断空间是否充足
```
```

```
      if (this.disk_bitmap[0].block_free_num >= block_need_num) {
        this.disk_bitmap[0].block_free_num -= block_need_num; //修改空闲
块数
        //分配空间
        let count = 0; //计数
        let last_block_index = -1; //记录上一个
        let bitmap_change_index = [];
        for (let i in this.disk_bitmap[0].bitmap) {
          if (count == block_need_num) {
            this.cur_dir[this.open_doc_index].p_end = last_block_index;
//记录终止指针
            break;
          }
          if (this.disk_bitmap[0].bitmap[i] == "0") {
            if (count == 0) {
              this.cur_dir[this.open_doc_index].p_begin = i; //记录起始指
针
              bitmap_change_index.push(i); //记录位图
              this.physical_disk[i].content = block_content_ary[count];
//存入外部磁盘
              this.physical_disk[i].des_content = block_content_ary[
                count
              ].slice(0, 10);
              count++;
              last_block_index = i; //记录
            } else {
              this.physical_disk[last_block_index].disk_next = i; //上一
个磁盘块指向此块
              bitmap_change_index.push(i); //记录位图
              this.physical_disk[i].content = block_content_ary[count];
//存入外部磁盘
              this.physical_disk[i].des_content = block_content_ary[
                count
              ].slice(0, 10);
              count++;
              last_block_index = i; //记录
            }
          }
        }
        ...
```

## 6.5 删除文件

1. 由于文件目录以多级目录方式组织，因此直接遍历树找到当前目录下要删除的文件所属的直接父文件夹及删除过程中所需参数

```javascript
  /*确定 delete_index，delete_cur_path,delete_cur_dir*/
  let delete_cur_path = delete_doc_path.split("\\"); //转为数组
  delete_cur_path.pop();
  let dir = this.totol_dir; //保存当前层的情况
  //这里的 i 相当于深度
  for (let i = 0; i < delete_cur_path.length - 1; ++i) {
    //这里的 j 是为了遍历每一层的子结点
    for (let j = 0; j < dir.length; ++j) {
      if (dir[j].name == delete_cur_path[i] && dir[j].type != 1) {
        //一定可以找到一个结点
        dir = dir[j].children; //dir 定位到前一层
      }
    }
  }
  //查询当前层哪个结点为要删除结点的父节点
  let index = -1;
  for (let i = 0; i < dir.length; ++i) {
    if (
      dir[i].name == delete_cur_path[delete_cur_path.length - 1] &&
      dir[i].type != 1
    ) {
      index = i;
    }
  }
  let delete_cur_dir = [].concat(dir[index].children); //确定 cur_dir
  if (index == -1) {
    alert(-1);
    return;
  }
  let delete_index = -1;
  //确定 selected_index
  for (let i = 0; i < delete_cur_dir.length; ++i) {
    if (
      delete_cur_dir[i].name == delete_doc_name &&
      delete_cur_dir[i].type == type
    ) {
      delete_index = i;
      break;
```

```
        }
      }
```

2. 判断删除的文件类型

若不为文件夹，且在磁盘内存有数据（p_begin 不为-1），则先根据文件目录表中 FCB 所记录的块指针，调用` deleteFromDisk(p_begin,p_end)`将磁盘对应块中的数据擦除，并取消块之间的链接

```javascript
    if (delete_cur_dir[delete_index].p_begin != -1) {
          this.deleteFromDisk(
            delete_cur_dir[delete_index].p_begin,
            delete_cur_dir[delete_index].p_end
          );
        }
```

在从对应的文件目录中将对应的 FCB 项去除，即从树中去除相应结点

```javascript
    for (let i = 0; i < dir[index].children.length; ++i) {
          if (
            dir[index].children[i].name ==
delete_cur_dir[delete_index].name &&
            dir[index].children[i].type ==
delete_cur_dir[delete_index].type
          ) {
            dir[index].children.splice(i, 1);
            this.cur_dir = [].concat(dir[index].children); //更新目录
            return;
          }
        }
```

若为文件夹，则递归调用本函数，将所有的子文件删除后，再将其删除

```javascript
    ...
    else if (type == 0) {
        //删除文件夹
        let i = 0;
        while (i < delete_cur_dir[delete_index].children.length) {
```

```
          this.deleteFile(
            delete_cur_dir[delete_index].children[0].type,
            delete_cur_dir[delete_index].children[0].path,
            delete_cur_dir[delete_index].children[0].name
          );
        }
        for (let i = 0; i < dir[index].children.length; ++i) {
          if (
            dir[index].children[i].name == delete_doc_name &&
            dir[index].children[i].type == 0
          ) {
            dir[index].children.splice(i, 1);
            this.cur_dir = [].concat(dir[index].children);

            break;
          }
        }
        return;
      }
```

## 6.6 数据记录/恢复

使用 **localStorage** 对象，其允许在浏览器中存储 key/value 对的数据，可长久保存整个网站的数据，保存的数据没有过期时间，直到手动去删除，所以很适合在本项目使用

## 6.7 保存数据

在页面关闭/刷新/前进/后退时，将文件目录表、位图、用内存空间模拟的磁盘空间数据转换为 JSON 格式，存入缓存中。需要在 mounted 钩子中设置

```javascript
mounted() {
    //浏览器退出时存储
    window.onbeforeunload = (e) => {
      e = e || window.event;
      if (e) {
        e.returnValue = "关闭提示";
      }
      localStorage.clear();
```

```
      let physical_disk_JSON = JSON.stringify(this.physical_disk);
      let totol_dir_JSON = JSON.stringify(this.totol_dir);
      let disk_bitmap_JSON = JSON.stringify(this.disk_bitmap);
      localStorage.setItem("physical_disk", physical_disk_JSON);
      localStorage.setItem("total_dir", totol_dir_JSON);
      localStorage.setItem("disk_bitmap", disk_bitmap_JSON);
      return "关闭提示";
    };
    ...
}
```

## 6.8 读取数据

在页面打开后，变量创建完成但页面未渲染时，检测本地缓存中是否有相应数据，若有则读取数据，将读到的 JSON 格式转换回对象，完成相应状态的恢复。在 created 钩子设置：

```javascript
  created() {
    this.physicalDiskInit(); //初始化硬盘
    if (localStorage.length != 0) {
      this.totol_dir = JSON.parse(localStorage.getItem("total_dir"));
      this.physical_disk =
JSON.parse(localStorage.getItem("physical_disk"));
      this.disk_bitmap =
JSON.parse(localStorage.getItem("disk_bitmap"));
    }
    this.cur_dir = [];
    this.cur_dir.push(this.totol_dir[0]); //初始化当前目录
  },
```