OS第三次作业：文件管理

一、项目介绍

- 本项目实现了一个非常简单的文件系统，来加深对文件系统的理解，达到以下要求：
  - **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**
- 具体实现功能如下：
  - **格式化、重命名**
  - **创建子目录、删除子目录**
  - **显示目录、返回上级目录**
  - **创建文件、打开文件、写文件、读文件、删除文件**
  - **文件属性查看**

二、运行要求

- **语言要求：**`python 3.11`
- **开发环境：**`pycharm 2023.3.3`
- **软件包要求：**

```
pyqt5          # 前端UI
datetime       # 计算时间
qdarkstyle     # QSS主题
pickle         #存档读写
bitarry        #位图
```

- **文件结构要求：**

```
FileManagement
|  Report.pdf
|
├─Code
|  |  file_system_components.py
|  |  file_system_save.save
|  |  main.py
|  |  MainWindow.py
|  |
|  └─picture
|        beforedir.png
|        File.png
|        HasContentDir.png
|        NoContentFileDir.png
|
```

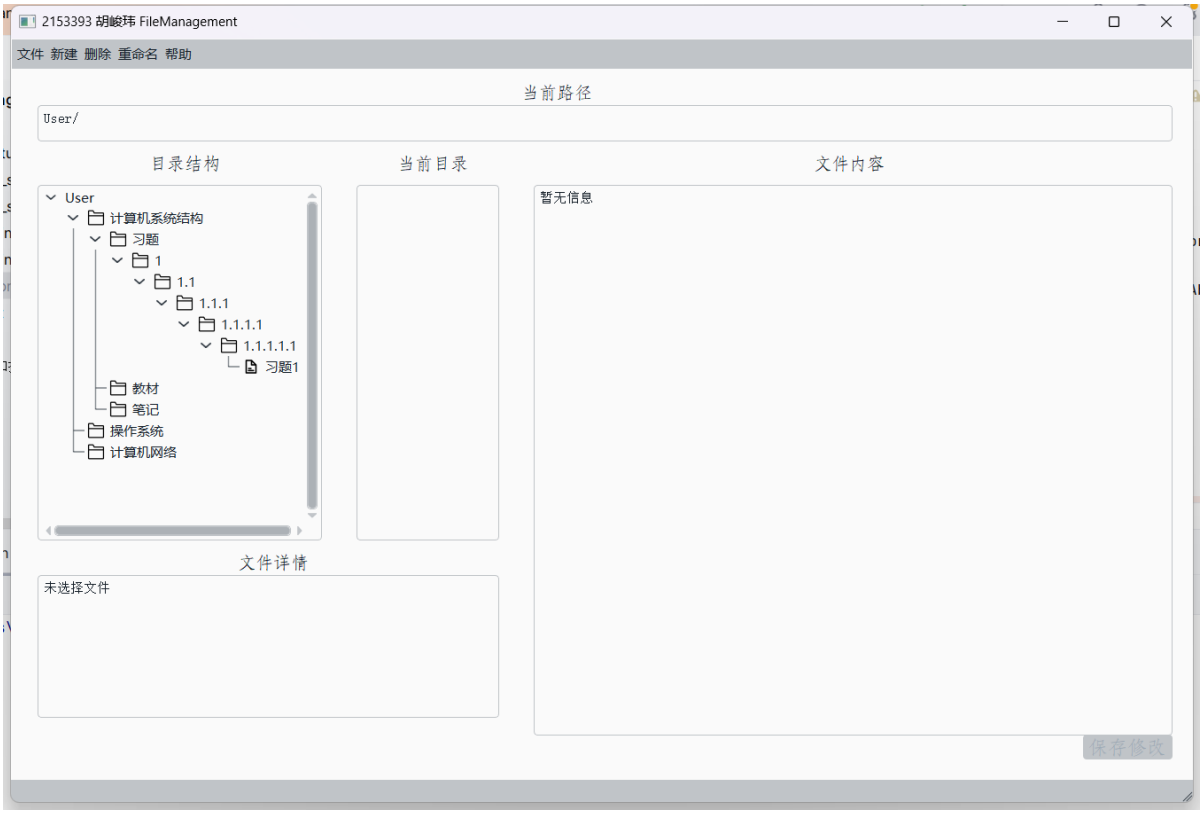
```
└─exe
    file_system_save.save
    main.exe
```

- **代码运行：**

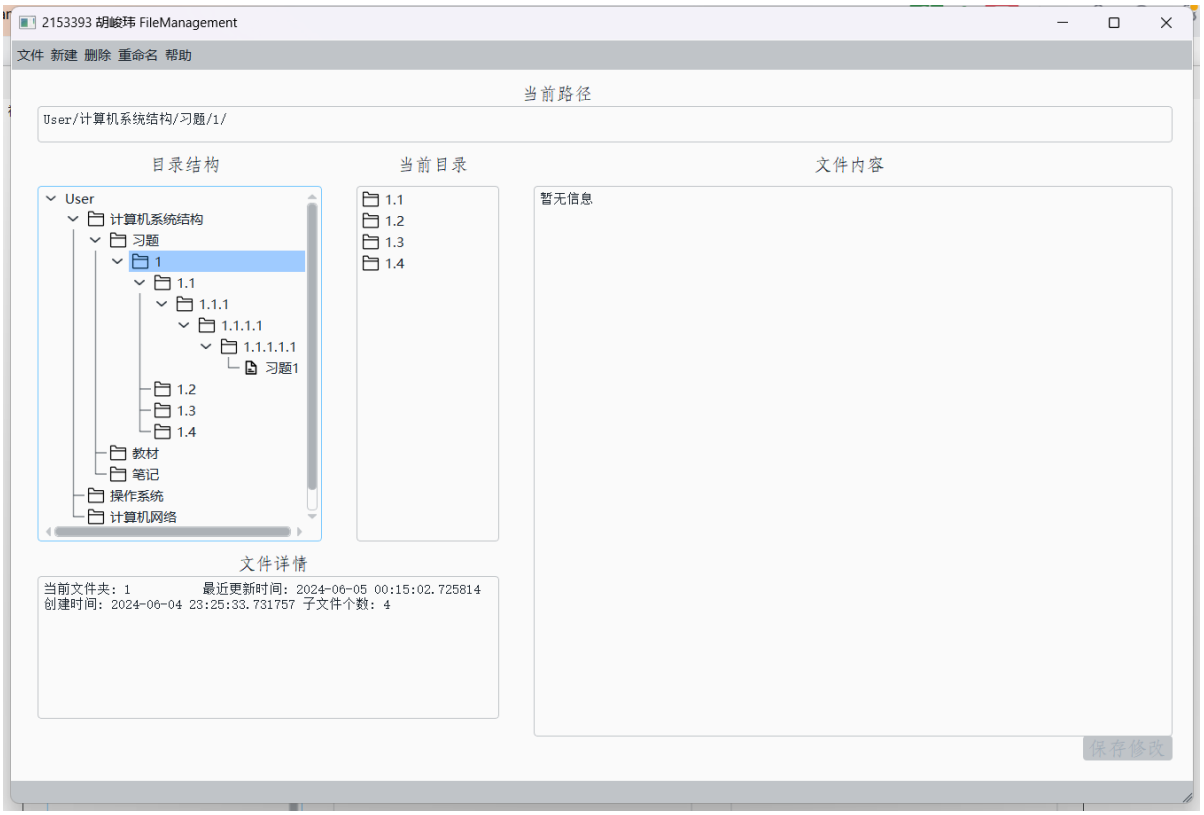
```
python main.py
```

三、程序介绍

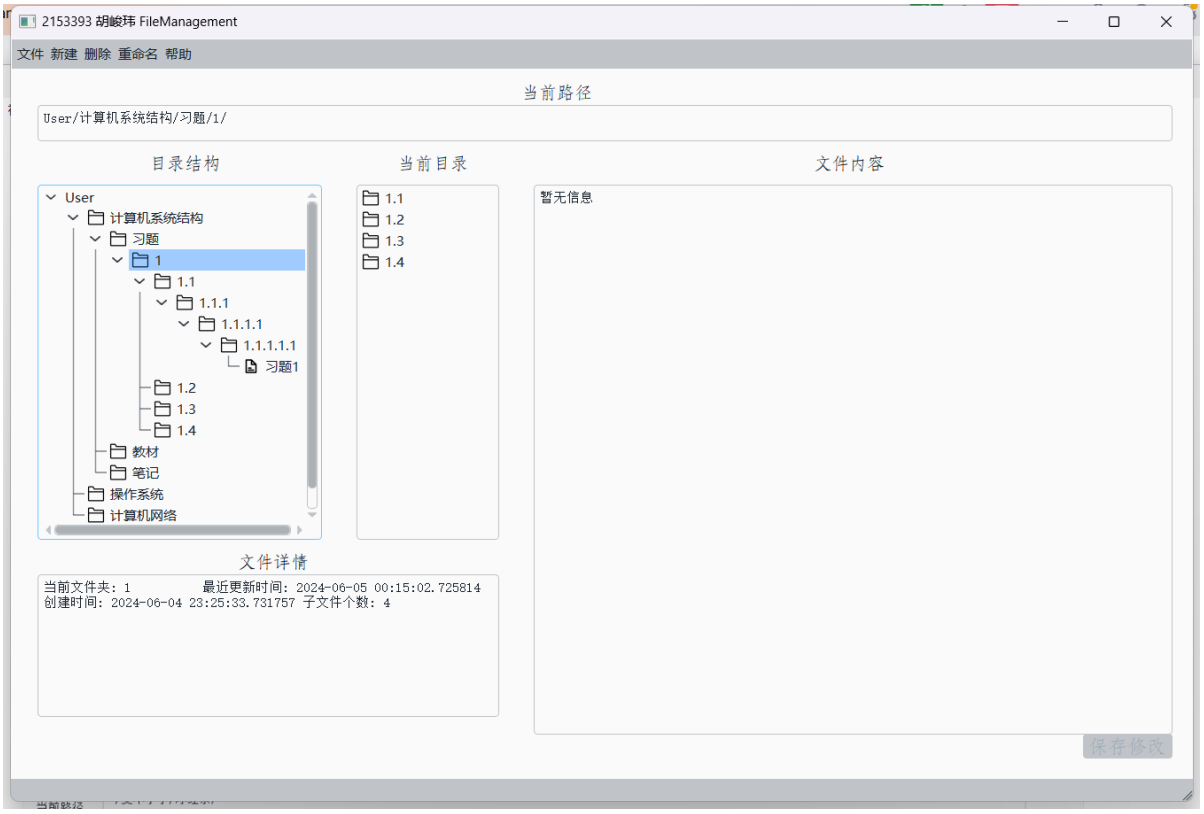
- 程序初始界面如下：



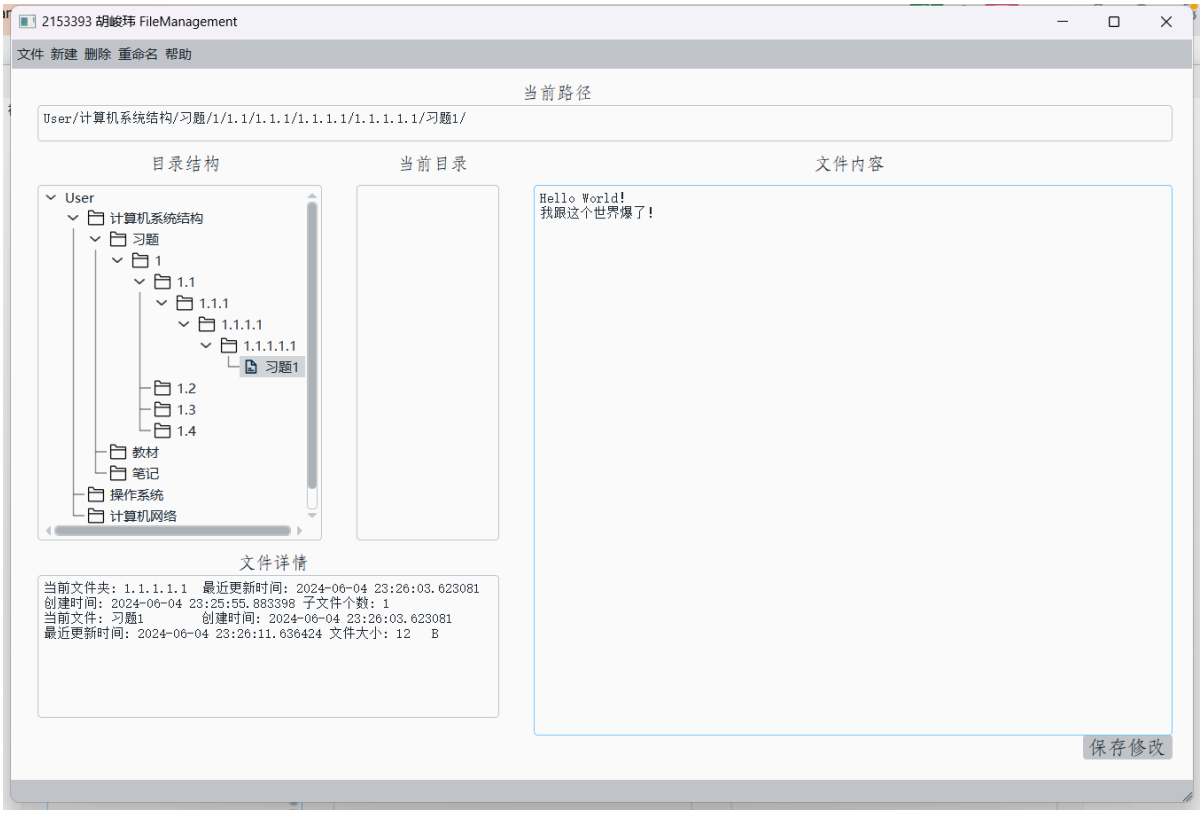
- 点击左侧目录结构可以选择任一文件

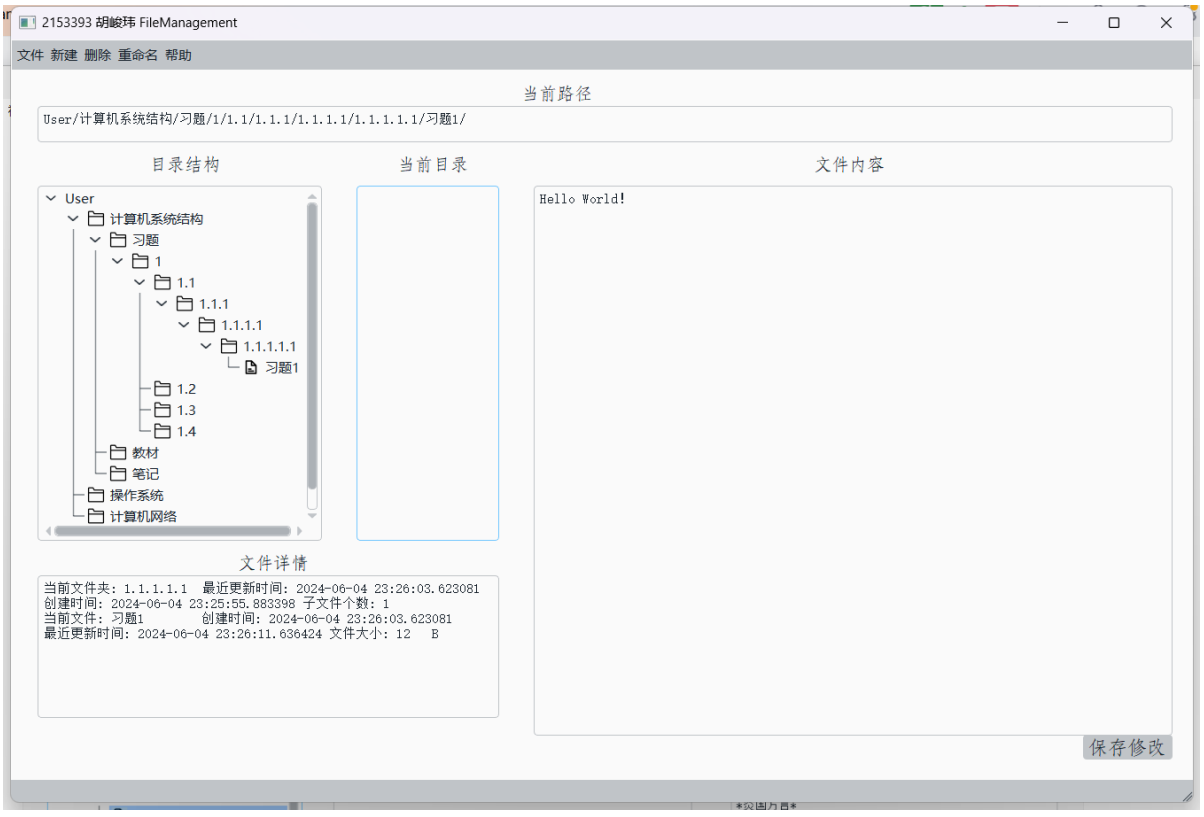


- 点击中间“当前目录下文件"可以选择文件/打开文件夹

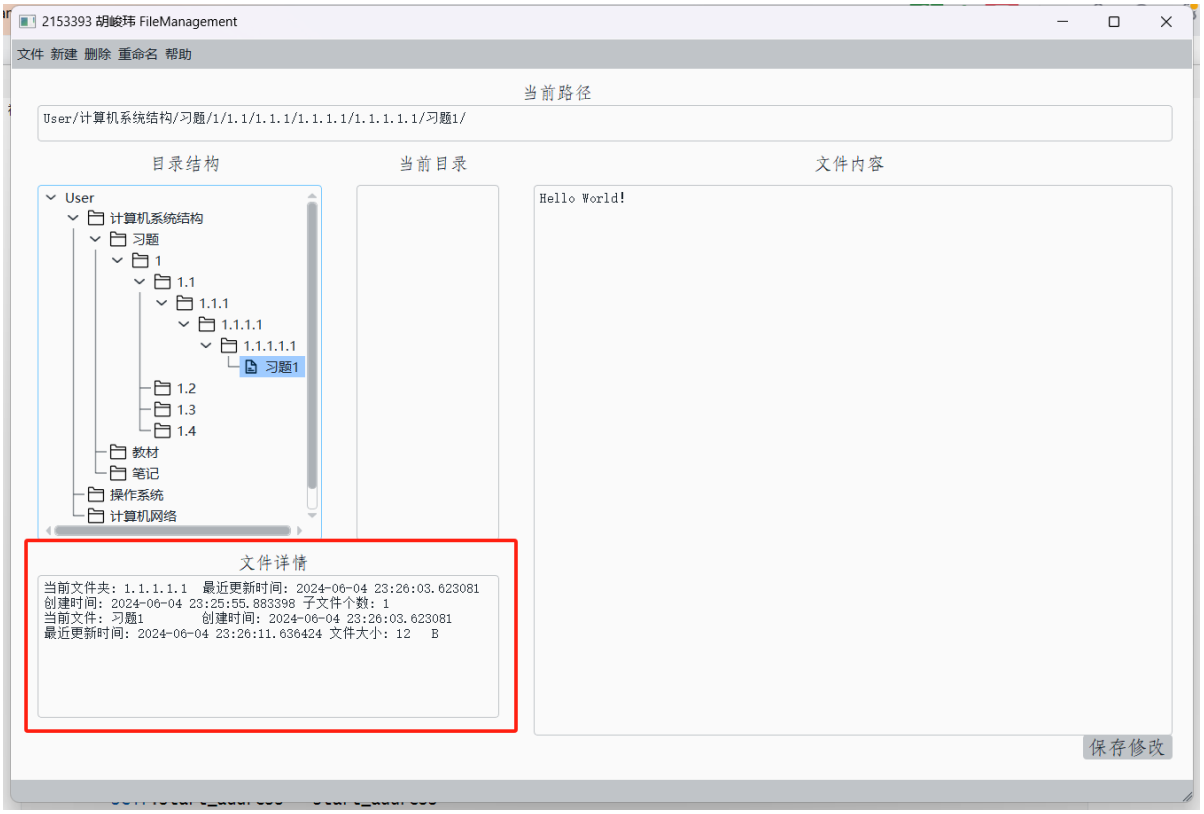


- 在左侧/中间点击文件都可以在右侧进行读取和修改

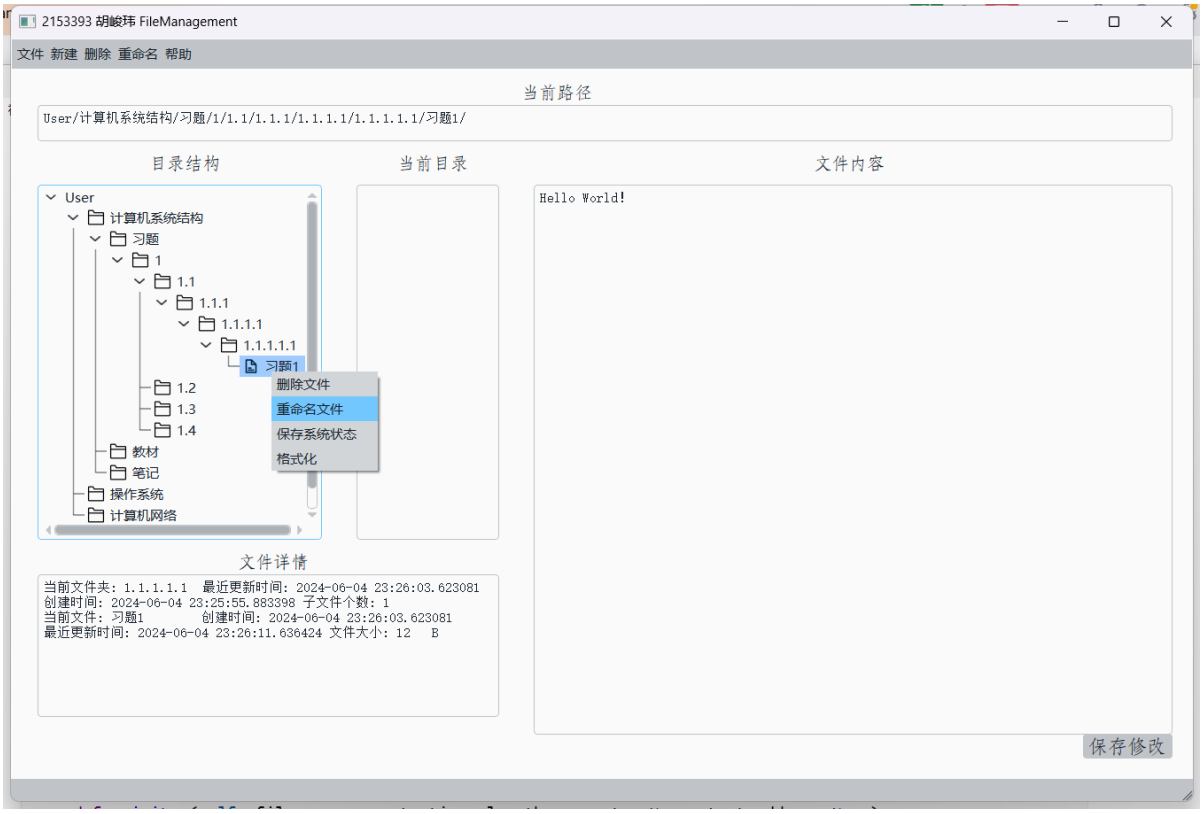


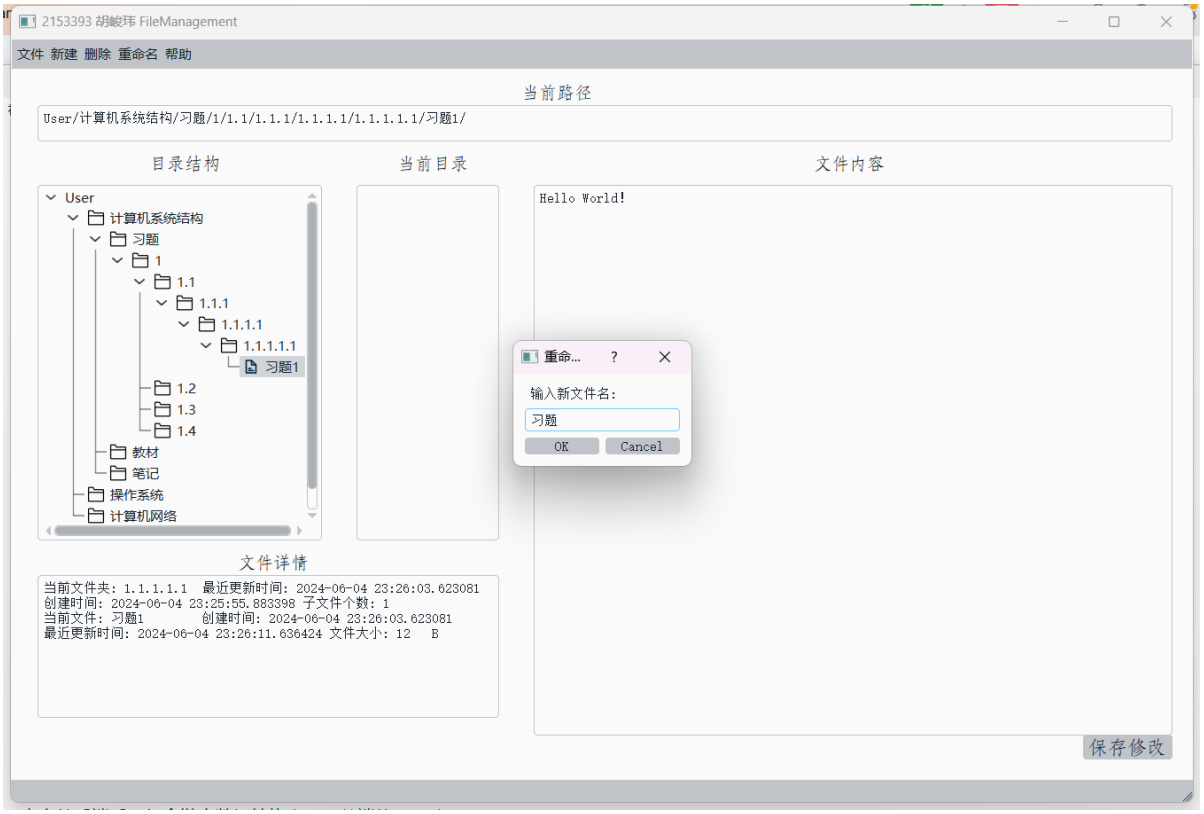


- 下方可以读取相关的信息：



- 上方文件栏/在左侧目录结构右键点击文件或文件夹可以进行文件修改





四、逻辑实现

本文件后端采用复合链表数据结构实现，前端使用`PyQt5`实现`UI`

后端

- 相关数据结构如下：
  - `FCB`：记录存储的文件信息

```
class FCB:
    def __init__(self, file_name, create_time, length, parent =
None,start_address=None):
        self.file_name = file_name
        self.create_time = create_time
        self.modify_time = create_time
        self.length = length
        self.start_address = start_address
        self.parent=parent
```

- `FAT`：数组模拟链表，内部存储的都是下一个内存块的地址

```
class FAT:
    def __init__(self):
        self.block_num = BLOCK_NUM
        self.table = []
        for i in range(BLOCK_NUM):
            self.table.append(FAT_FREE)
```

- `Disk`：模拟磁盘

```
class Disk():
    def __init__(self):
        # 用list代替链表指针
        self.list = []
        for i in range(BLOCK_NUM):
            self.list.append("")
```

- `Freespace`：记录每个块的使用情况

```
class FreeSpace:
    def __init__(self):
        self.bitmap = bitarray(BLOCK_NUM)
        self.bitmap.setall(0)
```

- `FileTreeNode`：文件夹结点，子文件夹存储在`DirNode`，子文件存储在`FileNode`里，`parent`记录上一级文件夹的信息。同时存储了一些文件夹的信息

```
class FileTreeNode:  # dir
    def __init__(self, name: str,create_time, parent=None):
        self.DirNode = []
        self.FileNode = []
        self.parent = parent
        self.dir_name = name
        self.create_time = create_time
        self.modify_time = create_time
```

- `FileSystem`：实现了对文件系统的增删改查

```
class FileSystem:
    def __init__(self):
```

```
        # 存在存档文件
        if os.path.exists(SAVEFILE):
            # 按序读出文件信息
            with open(SAVEFILE, 'rb') as f:
                self.file_tree = pickle.load(f)
                self.free_space = pickle.load(f)
                self.disk = pickle.load(f)
                self.fat = pickle.load(f)
        # 不存在文件，自己创建一个
        else:
            self.file_tree = FileTreeNode("User",datetime.now())
            self.free_space = FreeSpace()
            self.disk = Disk()
            self.fat = FAT()

    def find_free_index(self):
        # 0 -> free
        return self.free_space.bitmap.find(0)

    def SaveSystemState(self):
        with open(SAVEFILE, 'wb') as f:
            pickle.dump(self.file_tree, f)
            pickle.dump(self.free_space, f)
            pickle.dump(self.disk, f)
            pickle.dump(self.fat, f)

    def FormatSystem(self):
        self.file_tree = FileTreeNode("User",datetime.now())
        self.free_space = FreeSpace()
        self.disk = Disk()
        self.fat = FAT()
        print("finish format")

    def createDir(self, curDir: FileTreeNode, Dirname, Curtime):
        for file in curDir.DirNode:
            if file.dir_name == Dirname:
                print("name exist")
                return False
        curDir.DirNode.append(FileTreeNode(Dirname,Curtime,curDir))
        curDir.modify_time = Curtime
        return True

    def createFile(self, curDir: FileTreeNode, Filename, Curtime):
        for file in curDir.FileNode:
            if file.file_name == Filename:
                print("File exist")
                return False

        curDir.modify_time = Curtime
        curDir.FileNode.append(FCB(Filename, Curtime, 0,curDir))

    def WriteFile(self, File: FCB, data):
        File.modify_time = datetime.now()
        File_Pointer = -1

        # 将data逐元素读入
```

```
        while data !="":
            # 找到第一个空的块
            next_point = self.find_free_index()
            if next_point == -1:
                # 满了
                print("no more free space")
                raise AssertionError("no more space")
            if File_Pointer == -1:
                # 第一个块的位置
                File.start_address = next_point
            else:
                self.fat.table[File_Pointer] = next_point

            self.disk.list[next_point] = data[:BLOCK_SIZE]
            data = data[BLOCK_SIZE:]
            self.free_space.bitmap[next_point] = SPACE_OCCUPY

            File_Pointer = next_point
            self.fat.table[File_Pointer] = FAT_END
            File.length += BLOCK_SIZE


    def DeleteFile(self,CurDir: FileTreeNode,File:FCB):
        # 删去记录
        CurDir.modify_time = datetime.now()
        CurDir.FileNode.remove(File)
        pointer = File.start_address
        if pointer is None:
            return False
        # 在位图中将相关的记录都删掉
        while self.fat.table[pointer] != FAT_END:
            self.free_space.bitmap[pointer]=SPACE_FREE
            pointer = self.fat.table[pointer]
        self.free_space.bitmap[pointer] = SPACE_FREE
        return True

    # 为递归清空文件夹提供函数
    def ClearDir(self,CurDir:FileTreeNode, DeleteDir:FileTreeNode):
        # 清空当前目录下的文件
        for file in DeleteDir.FileNode:
            self.DeleteFile(DeleteDir,file)
        # 递归清空当前目录下的子目录
        while len(DeleteDir.DirNode) >0:
            ChildDir = DeleteDir.DirNode[0]
            self.ClearDir(DeleteDir,ChildDir)
        CurDir.DirNode.remove(DeleteDir)
    def deleteDir(self, DeleteDir:FileTreeNode):
        pointer = DeleteDir.parent
        self.ClearDir(pointer,DeleteDir)

    def ReadFile(self,File:FCB):
        pointer = File.start_address
        if pointer is None:
            AssertionError("Open File which start addr is none")
        data = ""
        while pointer != FAT_END:
```

```
            data += self.disk.list[pointer]
            pointer=self.fat.table[pointer]
        return data

    def RenameFile(self,File:FCB, NewName:str, CurDir:FileTreeNode):
        File.file_name = NewName
        File.modify_time=datetime.now()
        CurDir.modify_time=datetime.now()

    def RenameDir(self, NewName:str, CurDir:FileTreeNode):
        CurDir.modify_time = datetime.now()
        CurDir.dir_name = NewName
```

前端

为后端增删改查提供接口，并且对某些非法行为进行提示，提高系统的可用性

- 左侧树状结构生成：

```
def
dfsBuildTreeModel(self,model:QStandardItemModel,file_tree_node:file_system_compon
ents.FileTreeNode):
    for file in file_tree_node.FileNode:
        Item = QStandardItem(file.file_name)
        Item.setIcon(QIcon('picture/File.png'))
        model.appendRow(Item)

    for dirnode in file_tree_node.DirNode:
        child_model = QStandardItem(dirnode.dir_name)
        child_model.setIcon(QIcon('picture/NoContentFileDir.png'))
        self.dfsBuildTreeModel(child_model,dirnode)
        model.appendRow(child_model)
```

- 中间文件结构生成：

```
def bulidListView(self) ->QStandardItemModel:
    model = QStandardItemModel()
    if self.cur_selected_dir is not None and self.cur_selected_file is None:
        for file in self.cur_selected_dir.FileNode:
            Item =QStandardItem(file.file_name)
            Item.setIcon(QIcon('picture/File.png'))
            model.appendRow(Item)
        for childdir in self.cur_selected_dir.DirNode:
            Item = QStandardItem(childdir.dir_name)
            Item.setIcon(QIcon('picture/NoContentFileDir.png'))
            model.appendRow(Item)
    else:
        model.clear()
    return model
```

- UI信号与函数连接（前端的组件连接的处理函数）：

```
def setui(self):
    self.UpdateUI()
```

```
        self.ui.filecontent.setWordWrapMode(QTextOption.WrapAnywhere)
        # 记录connect
        self.ui.SaveFile.clicked.connect(self.SaveFile)
        self.ui.FormatFileSystem.triggered.connect(self.sys_format)
        self.ui.Save_System_Status.triggered.connect(self.sys_SaveSys)
        self.ui.CreateDir.triggered.connect(self.sys_create_dir)
        self.ui.CreateFile.triggered.connect(self.sys_create_file)
        self.ui.DeleteDir.triggered.connect(self.sys_delete_dir)
        self.ui.DeleteFile.triggered.connect(self.sys_delete_file)
        self.ui.RenameDir.triggered.connect(self.sys_rename_dir)
        self.ui.RenameFile.triggered.connect(self.sys_rename_file)
        self.ui.actionHelp.triggered.connect(self.sys_Help)
        self.ui.actionAbout.triggered.connect(self.sys_About)
        self.ui.actionaddition.triggered.connect(self.sys_Addition)
```

五、项目反思

- **项目亮点**
  - 实现基本的文件管理功能
  - 界面简洁美观
  - 运用了面向对象的方法，代码复用率高
  - 支持对文件重名的检测，对重名文件会无法创建
  - 不限制目录树的高度和宽度，自由度高
  - 实时更新文件修改时间
- **项目改进**
  - 可以增加更多功能，比如复制、粘贴等
  - 可以让文件树形图界面显示除文件夹外的文件
  - 可以让文件列表界面支持排序
  - 支持更多的文件类型
  - 由于框架的使用，仅限于在Windows平台运行。在未来项目开发过程中，可以考虑采取其他开发工具，使得项目可移植性更好
- 通过本次项目我感触颇多，也收获很多
  - 这个项目让我更深入地理解了文件系统，通过实际运用`Python`编程语言来模拟一个`windows`文件管理系统，我更好地理解了文件的存储、操作以及目录结构的管理方式。通过项目的实践，我现在可以更加熟练地应用相关知识，并且，我也加深我的对文件系统实现过程的理解。
  - 在实现具体功能过程中，比如格式化、创建子目录、显示目录、操作文件等等，我逐渐理解和掌握了如何应用`Python`的各种库和模块来实现这些功能。这个过程对我在之后的编程学习和实践过程中带来了很大的帮助。