**FAT文件管理系统**

2152814 周成杰

1. **引言**

**1.1 实验目的**

本次实验旨在通过实现一个简单的FAT文件系统，加深对文件系统的理解，掌握文件系统的基本操作，如格式化、创建子目录、删除子目录、显示目录、更改当前目录、创建文件、打开文件、关闭文件、写文件、读文件、删除文件等。

**1.2 实验环境和工具**

本次实验选用python语言，版本为Python 3.10.6，选择的python图形库为Pyqt5，IDE选用Pycharm，配合外部工具Qt designer设计UI界面，使用外部工具PyUIC将设计好的UI界面框架由ui文件转为py文件，再开始在这些的基础上编程。 实验在Windows操作系统上完成。

**1.3 简述问题**

1.3.1基本要求

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，以便下次可以将其恢复到内存中来。

1.3.2技术细节

* 文件存储空间管理可采取显式链接（如FAT）或者其他方法。（即自选一种方法）
* 空闲空间管理可采用位图或者其他方法。如果采用了位图，可将位图和FAT表合二为一。
* 文件目录采用多级目录结构。至于是否采用索引节点结构，自选。目录项目中应包含：文件名、物理地址、长度等信息。同学可在这里增加一些其他信息

1.3.3文件系统提供的操作：

* 格式化
* 创建子目录
* 删除子目录
* 显示目录
* 更改当前目录
* 创建文件
* 打开文件
* 关闭文件
* 写文件
* 读文件
* 删除文件
* ……

**2. 实验步骤：**

**2.1. 确定文件存储空间管理方式**

本次实验采用FAT表进行文件存储空间管理。FAT表是一种显式链接的文件系统，通过将文件存储空间分为若干个簇(cluster)，并在FAT表中记录每个簇之间的链接关系，来实现文件的存储和访问。

除了FAT表，常见的文件管理模式还包括NTFS、ext4、HFS+等。

* NTFS是Windows操作系统中常用的文件系统，它采用了类似于FAT表的簇的概念，但是它的簇的大小可以根据存储设备的大小进行动态调整。此外，NTFS还支持文件和目录的访问控制、安全性、日志记录等功能。
* ext4是Linux操作系统中常用的文件系统，它支持更大的存储设备和更大的文件，同时还支持日志记录、快照、扩展属性等功能。
* HFS+是苹果公司的文件系统，它支持日志记录、元数据加密、文件压缩、快照等功能，同时还能够在Mac OS和Windows操作系统之间共享文件。

选择FAT表作为文件存储空间管理方式的原因主要有以下几点：

1. 兼容性好：FAT表是一种通用的文件系统，能够在不同的操作系统和设备上使用，这使得它成为了许多嵌入式系统、移动设备和存储设备的首选文件系统。
2. 简单易用：FAT表的设计简单，易于实现和维护，同时它也非常容易被用户理解和使用。这使得它成为了许多小型设备和移动设备的理想选择。
3. 空间利用率高：FAT表采用了簇的概念，将文件存储空间划分为若干个固定大小的块，这可以减少文件碎片，提高存储设备的空间利用率。
4. 适用范围广：FAT表不仅适用于小型设备和移动设备，还可以用于桌面计算机和服务器等大型系统。同时，FAT32文件系统支持存储单个文件的最大大小达到4GB，也能够满足大多数用户的需求。

**2. 2确定空闲空间管理方式**

本次实验采用位图进行空闲空间管理。位图是一种将每个簇的空闲状态记录在一个二进制位上的方法。如果该簇空闲，则对应的属性为free；如果该簇已经被占用，则对应的属性为busy。位图和FAT表可以合并成一个簇链表，通过遍历簇链表，即可实现空闲簇的查找和使用。

我最终将空闲信息合并到簇的属性中，用一个list列表模拟磁盘，模拟的磁盘中有512个模拟的簇，这些簇是由字典模拟的，下面给出具体的结构：

# 磁盘——管理文件:创建一个模拟磁盘，包含512个簇(0-511)，簇大小为1024B(4KB)，簇中记录数据  
self.disk = [{"cluster\_id": id, "size": 1024, "data": None, "status": "free", "next": None} for id in  
 range(512)]

在此结构中，一个簇有上面的五个属性，分别记录了簇的编号、剩余空间、数据、是否空闲以及下一个相连簇的编号，通过这些属性和目录结构将文件内容和模拟磁盘联系起来。

3. 确定文件目录结构

本次实验采用多级目录结构，目录结构可以通过一个树形结构来表示。每个目录节点包含文件名、物理地址、长度等信息，同时也可以添加其他信息。

我才用类似于递归的结构来设计这个目录，也就是文件夹中嵌套文件夹，每个类中有基本的属性和地址信息，地址储存父文件夹的带名路径，方便构建树形结构，下面给出我的文件夹结构，也就是我的“目录结构“：

def \_\_init\_\_(self, path, name):  
 try:  
 os.mkdir(path + name + '/')  
 except:  
 print("文件夹创建失败")  
 self.folderPath = path  
 self.folderName = name  
 self.fileList = [] # 文件夹中包含的文件  
 self.folderList = [] # 文件夹中包含的文件夹  
  
 # 簇号记录文件磁盘位置  
 self.start\_cluster = None  
 self.end\_cluster = None

当我写入磁盘时，会构建一个字典将这些信息存入一个文件中，程序再次启动时会自动读取该文件将这个目录中的所有文件都在本地构建出来，同时显示在我的UI上。

4. 程序实现

根据上述设计思路，编写程序实现FAT文件系统的基本操作，包括格式化、创建子目录、删除子目录、显示目录、更改当前目录、创建文件、打开文件、关闭文件、写文件、读文件、删除文件等。

首先使用qt designer设计一个整体的框架UI，导出ui文件，使用pyuic转为py文件，将其作为程序的UI的主类，单独一个文件不对其另外的编辑，方便随时更改ui文件重新转换代替，利用python多文件编程，编写其他的逻辑类文件

根据上述的结构，设计编写函数实现要求的几个功能，比如下面folder类中的：



主要的逻辑：

模拟disk是一个list，list中的元素是512个字典模拟的簇；模拟目录也是一个储存了文件树转换而来的字典集合的文件，两个文件都是pikcle文件，并提供txt文件方便查看内容。

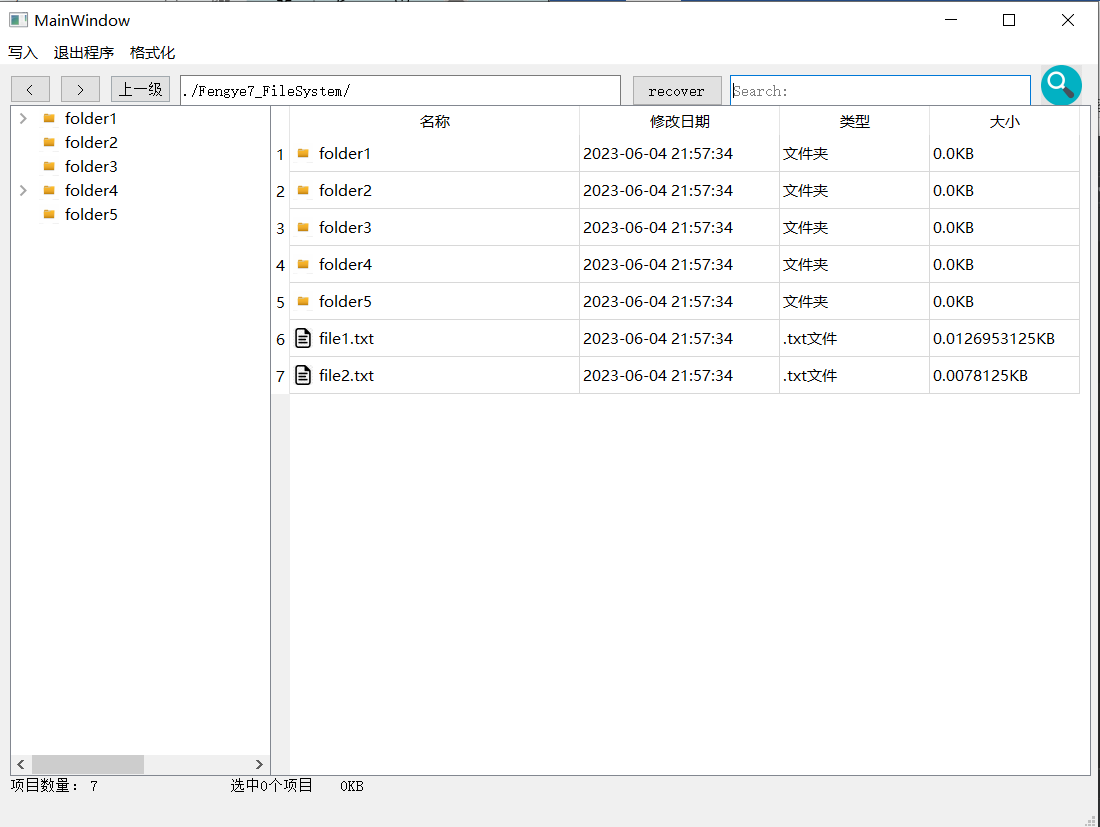
格式化逻辑：提前准备上述两个文件的标准文件，当用户选择格式化的时候，将上述两个文件的内容赋值为标准文件，自动退出程序，等待重新启动时重新读取。

5. 测试，

对实现的FAT文件系统进行测试，包括对各种操作的正确性和效率进行测试。

3. UI界面

3.1 主窗口：

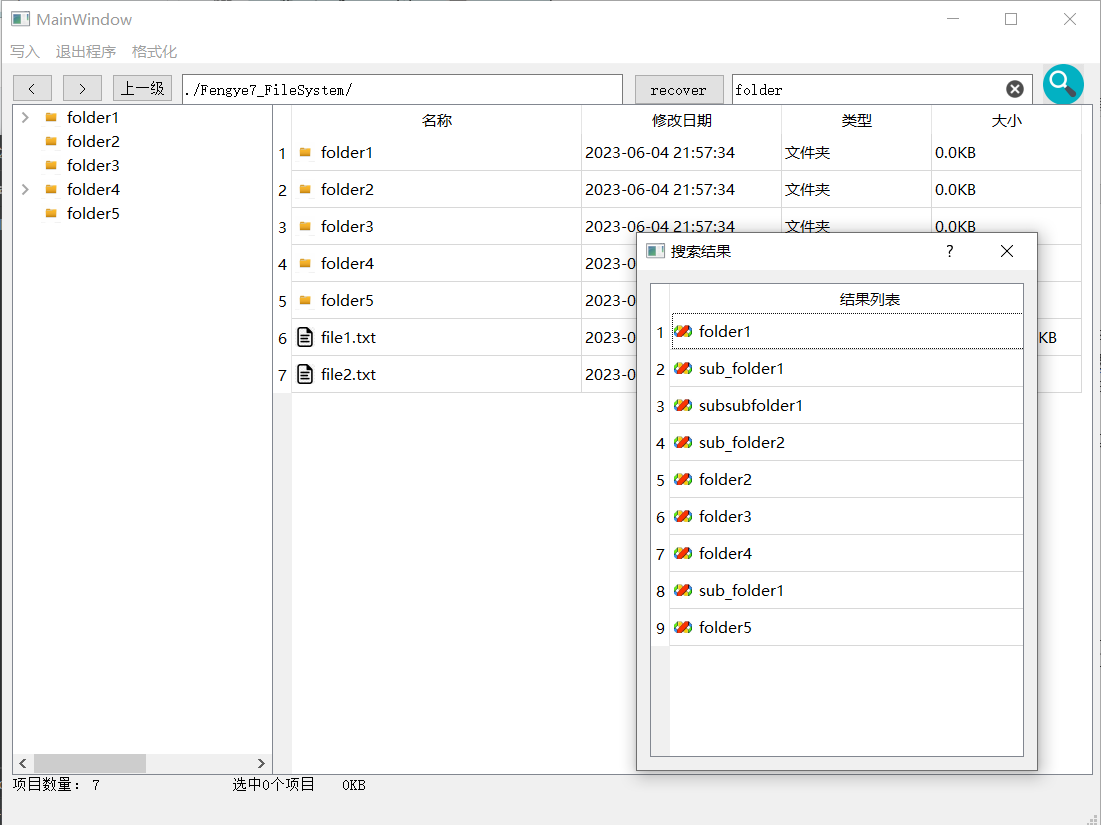


该窗口模拟windows的文件资源管理器，主要包括地址显示栏，搜索栏，文件列表和文件目录树，实现了跳转、搜索、使用本地关联打开文件、读写文件、格式化等功能，提供磁盘写入和退出程序按钮、格式化几个按钮。

注意：

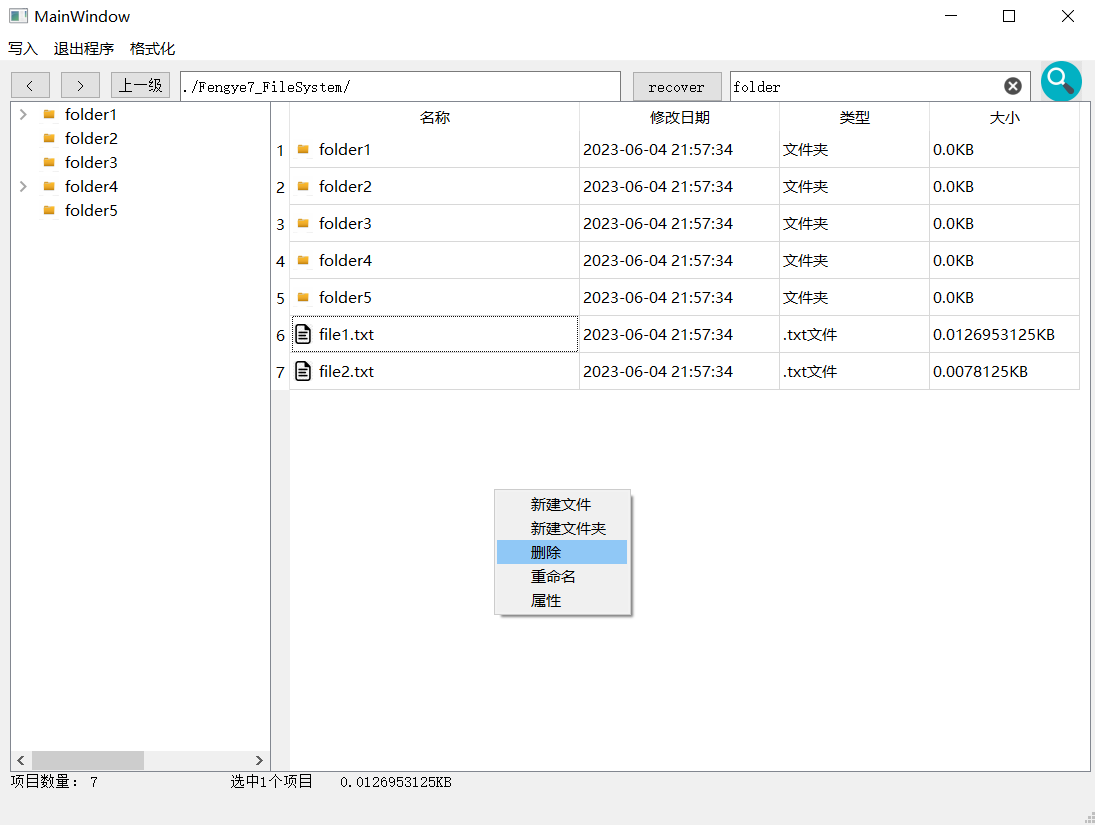
* 左上角的两个箭头的功能未实现
* 上一级按钮可跳转到父级目录
* 下方显示当前列表文件数量，选中文件的大小（仅支持同时选中一个文件）
* Recover按钮刷新文件列表和文件目录树
* 退出程序按钮才会退出时自动写入模拟磁盘本次的更改，通过右上角自带的X号退出不会自动保存
* 地址栏的跳转功能需要你输入合法路径后点击回车
* 搜索栏的当你输入在查找范围内的文本时，弹窗显示查找的结果，你可以像操作文件列表那样操作它，当你不使用时请你及时通过X号关闭这个“搜索结果“弹窗
* 当你在文件列表中右键鼠标会弹出一个窗口选择其他拓展操作：新建、删除、属性等

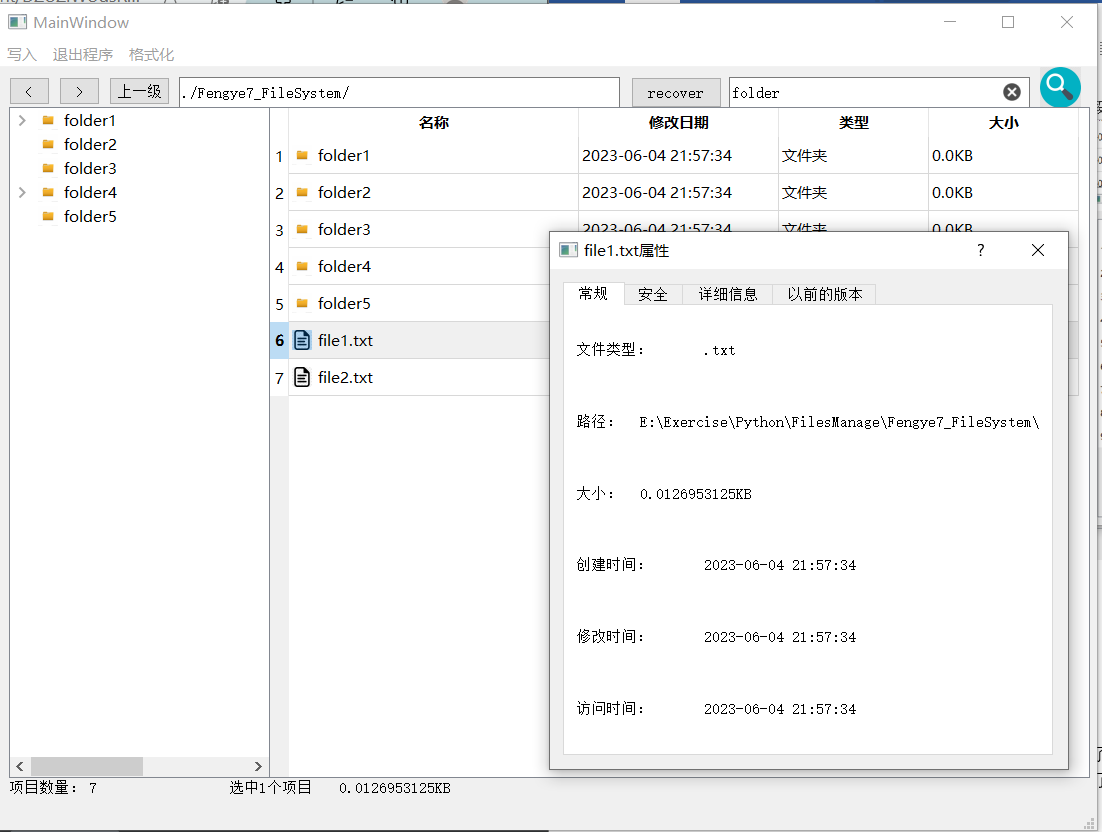
3.2 搜索功能展示：



这里为求代码不重复，选用相同图标表示结果

3.3 右键属性功能展示：





属性页仅实现常规页面的显示，其余几个当前功能不支持。

4. 实验结论：

通过本次实验，不仅掌握了文件系统的基本概念和操作方法，还学习了如何使用 Python 图形框架来构建用户友好的界面，提高了我们的编程能力和代码实现能力。具体来说，学习了以下内容：

* Python 图形框架：了解了常见的 Python 图形框架——PyQt，学习了它的基本使用方法，包括如何创建窗口、添加组件、设置布局等。
* 用户界面设计：如何设计用户友好的界面，包括选择合适的颜色、字体、图标等，以及如何布局和组织界面元素，使其易于使用和美观。
* 事件处理：如何处理用户界面上的事件，如鼠标点击、键盘输入等，以及如何编写相应的处理函数，实现交互式的功能。
* 文件系统的可视化实现：我们将学习到的 Python 图形框架和用户界面设计技巧应用到文件系统的可视化实现中，实现了一个功能完善的文件系统界面。我们可以通过该界面进行文件的创建、删除、修改、查看等操作，同时也可以对文件系统的存储空间进行管理，包括显示空闲空间和已使用空间、支持多级目录结构等。

5． 附录：Main.py------主要的逻辑文件

import os.path  
  
from Frame import \*  
from PyQt5 import QtGui, QtWidgets, QtCore  
from PyQt5.QtCore import Qt  
from datetime import datetime  
from enum import Enum  
import atexit  
import pickle # 处理序列化  
import watchdog  
from watchdog.events import PatternMatchingEventHandler  
from watchdog.observers import Observer  
  
from FileManage import \*  
  
  
# 双击的对象类型  
class Type(Enum):  
 Folder = 1  
 File = 2  
  
  
class Ui\_Fengye7\_FileSystem(Ui\_FileSysFrame):  
 # 构造函数  
 def \_\_init\_\_(self, FileSystem):  
 self.FileSys = FileManage("./", "Fengye7\_FileSystem")  
 self.setupUi(FileSystem) # 使用的.py由.ui文件转化而来，方便实时更改.ui  
 self.finishUI() # 完成界面的初始化  
  
 # 声明在groupBox创建右键菜单  
 self.groupBox.setContextMenuPolicy(Qt.Qt.CustomContextMenu)  
 self.groupBox.customContextMenuRequested.connect(self.create\_rightmenu) # 连接到菜单显示函数  
  
 # 链接写入磁盘和退出  
 actionA = QtWidgets.QAction(u'写入磁盘', self.menu\_write)  
 self.menu\_write.addAction(actionA) # 把动作A选项添加到菜单  
 actionA.triggered.connect(self.write\_disk)  
  
 actionB = QtWidgets.QAction(u'写入目录', self.menu\_write)  
 self.menu\_write.addAction(actionB) # 把动作A选项添加到菜单  
 actionB.triggered.connect(self.write\_directory)  
  
 actionC = QtWidgets.QAction(u'退出程序', self.menu\_exit)  
 self.menu\_exit.addAction(actionC) # 把动作A选项添加到菜单  
 actionC.triggered.connect(self.exit\_app)  
  
 actionD = QtWidgets.QAction(u'格式化', self.menu\_format)  
 self.menu\_format.addAction(actionD) # 把动作A选项添加到菜单  
 actionD.triggered.connect(self.format)  
  
 # atexit.register(self.write\_disk) # 程序退出时自动写入，防止忘记\_\_\_这里和格式化操作冲突，格式化操作后退出又将格式化的磁盘文件给改变了  
 # atexit.register(self.write\_directory)  
  
 # 链接recover刷新显示文件列表  
 self.RecoverButton.clicked.connect(self.recover\_table)  
  
 # 地址栏添加跳转操作  
 self.AddressBar.setReadOnly(False) # 设置为可编辑  
 self.AddressBar.editingFinished.connect(self.jump\_path) # 连接函数  
  
 # 搜索栏  
 self.SearchBar.setReadOnly(False)  
 self.SearchBar.returnPressed.connect(self.search\_file)  
  
 # 链接上一级按钮  
 self.BackUp.clicked.connect(self.back\_to\_father)  
  
 # 按钮返回上一级目录  
 def back\_to\_father(self):  
 mid\_folder = self.search\_ob(self.FileSys,  
 self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if mid\_folder is not None:  
 targetfolder = self.search\_ob(self.FileSys,  
 mid\_folder.folderPath) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if targetfolder is not None:  
 self.tableWidget.setRowCount(0) # 清空列表行  
 self.tableWidget.clearContents() # 清空内容  
 self.show\_table(targetfolder) # 展示新位置  
 else:  
 QtWidgets.QMessageBox.warning(None, 'error', f'未知问题返回失败！')  
 else:  
 QtWidgets.QMessageBox.warning(None, 'error', f'未知问题返回失败！')  
  
 # 搜索栏函数  
 def search\_file(self):  
 target\_list = []  
 bar\_text = self.SearchBar.text()  
  
 def sub\_search(sourse):  
 for i in sourse.folderList:  
 if bar\_text in i.folderName:  
 target\_list.append(i)  
 sub\_search(i)  
 for j in sourse.fileList:  
 if bar\_text in j.fileName:  
 target\_list.append(j)  
  
 sub\_search(self.FileSys) # 获得目标的list  
  
 new\_widget = QtWidgets.QDialog()  
 new\_widget.setWindowTitle("搜索结果")  
 new\_widget.move(self.SearchBar.pos().x(), self.SearchBar.pos().y() + 20)  
 new\_widget.resize(400, 500)  
  
 layout = QtWidgets.QVBoxLayout()  
 new\_widget.setLayout(layout)  
  
 v\_table = QtWidgets.QTableWidget()  
 v\_table.setColumnCount(1)  
 v\_table.setColumnWidth(0, 400)  
 v\_table.setEditTriggers(QtWidgets.QTableWidget.EditTrigger.NoEditTriggers)  
 v\_table.setHorizontalHeaderLabels(["结果列表"])  
 layout.addWidget(v\_table)  
  
 def double\_click(row, column):  
 \_, file\_ext = os.path.splitext(target\_list[row].get\_name())  
 # 双击文件夹  
 if file\_ext == "":  
 if target\_list[row] is not None:  
 self.tableWidget.setRowCount(0) # 清空列表行  
 self.tableWidget.clearContents() # 清空内容  
 try:  
 self.show\_table(target\_list[row]) # 展示新位置  
 except:  
 QtWidgets.QMessageBox.information(None, 'error', '该对象是无后缀名的文件，暂不支持打开！')  
 pass  
 # 双击文件  
 else:  
 if target\_list[row] is not None:  
 self.on\_modified(target\_list[row])  
  
 # 链接函数，这个cell链接自动传入row,column  
 v\_table.cellDoubleClicked.connect(double\_click)  
  
 # 添加选项  
 for k in target\_list:  
 v\_table.insertRow(v\_table.rowCount()) # 添加行  
 newItem = QtWidgets.QTableWidgetItem(QtGui.QIcon("./imags/answer.png"), k.get\_name())  
 v\_table.setItem(v\_table.rowCount() - 1, 0, newItem)  
 new\_widget.exec\_()  
  
 # 刷新窗口中的table  
 def recover\_table(self):  
 tagetfolder = self.search\_ob(self.FileSys,  
 self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if tagetfolder is not None:  
 self.tableWidget.setRowCount(0) # 清空列表行  
 self.tableWidget.clearContents() # 清空内容  
 self.show\_table(tagetfolder) # 展示新位置  
 self.treeWidget.clear() # 清空  
 self.show\_list(self.treeWidget, self.FileSys.folderList, Type.Folder) # 重建  
  
 # 将模拟的磁盘的内容写入本地  
 def write\_disk(self):  
 # 给一个字符串副本  
 with open("./Fengye7\_FileSystem/disk.txt", 'w') as f:  
 for block in self.FileSys.disk:  
 f.write(str(block) + '\n')  
 # 实际的模拟磁盘  
 # 如果您想将一个字典对象写入文件，您需要将其序列化为一个字节串对象。在Python中，可以使用pickle或json模块来序列化和反序列化Python对象。  
 with open('./Fengye7\_FileSystem/disk.pickle', 'wb') as f:  
 pickle.dump(self.FileSys.disk, f)  
  
 # 将文件结构转为目录储存到本地磁盘  
 def write\_directory(self):  
 # Serialize the file structure to a dictionary  
 def serialize\_file(file):  
 file\_dict = {  
 'path': file.filePath,  
 'name': file.fileName,  
 'start\_cluster': file.start\_cluster,  
 'end\_cluster': file.end\_cluster  
 }  
 return file\_dict  
  
 # Serialize the folder structure to a dictionary  
 def serialize\_folder(folder):  
 folder\_dict = {  
 'path': folder.folderPath,  
 'name': folder.folderName,  
 'file\_list': [serialize\_file(f) for f in folder.fileList],  
 'folder\_list': [serialize\_folder(f) for f in folder.folderList],  
 'start\_cluster': folder.start\_cluster,  
 'end\_cluster': folder.end\_cluster  
 }  
 return folder\_dict  
  
 root\_folder\_dict = serialize\_folder(self.FileSys) # 将整个管理系统文件树转为字典  
 # Save the folder structure to a file  
 with open('./Fengye7\_FileSystem/directory.pickle', 'wb') as f:  
 pickle.dump(root\_folder\_dict, f)  
 # 给一个字符串副本  
 with open("./Fengye7\_FileSystem/directory.txt", 'w') as f:  
 f.write(str(root\_folder\_dict))  
  
 # 退出程序  
 def exit\_app(self):  
 self.write\_disk() # 通过退出程序按钮退出的提供自动保存的功能，防止忘记  
 self.write\_directory()  
 QtCore.QCoreApplication.exit()  
  
 # 完善UI界面  
 def finishUI(self):  
 # UI界面中添加目录  
 self.treeWidget.setHeaderHidden(True)  
 self.treeWidget.header().setMinimumSectionSize(500) # 添加水平滚动条，尽量把这一列的尺寸设置大一点  
 self.treeWidget.itemClicked.connect(self.enter\_list) # 处理列表上的点击事件  
 self.show\_list(self.treeWidget, self.FileSys.folderList, Type.Folder) # 底层文件夹的显示  
 # self.show\_list(self.treeWidget, self.FileSys.fileList, Type.File) # 底层文件的显示  
  
 # 初始化展示第一个文件夹的信息,同时处理地址栏  
 self.tableWidget.setColumnCount(4) # 设置表格列数  
 self.tableWidget.setColumnWidth(0, 290) # 设置表格列宽  
 self.tableWidget.setColumnWidth(1, 200)  
 self.tableWidget.setColumnWidth(2, 150)  
 self.tableWidget.setColumnWidth(3, 150)  
 self.tableWidget.setEditTriggers(QtWidgets.QTableWidget.EditTrigger.NoEditTriggers)  
 self.tableWidget.itemDoubleClicked.connect(self.enter\_item) # 链接函数处理点击  
 self.tableWidget.itemClicked.connect(self.show\_below) # 处理下框  
 self.tableWidget.setHorizontalHeaderLabels(["名称", "修改日期", "类型", "大小"]) # 设置表头  
 self.tableWidget.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectionBehavior.SelectRows) # 设置整行同时选中  
 self.show\_table(self.FileSys) # 调用函数展示表  
  
 # 设置图标  
 self.SearchLabel.setPixmap(QtGui.QPixmap("./imags/search.png"))  
 self.SearchLabel.setScaledContents(True)  
  
 # 处理下框显示  
 def show\_below(self):  
 selecteditem = self.search\_ob(self.FileSys, self.AddressBar.text() + self.tableWidget.selectedItems()[0].text())  
 if selecteditem is not None:  
 self.selected\_item\_num.setText("选中1个项目")  
 self.selected\_item\_size.setText(str(selecteditem.get\_space() / 1024) + "KB")  
  
 # 创建右键菜单函数  
 def create\_rightmenu(self):  
 # 菜单对象  
 groupBox\_menu = QtWidgets.QMenu(self.groupBox)  
  
 actionA = QtWidgets.QAction(u'新建文件', groupBox\_menu)  
 groupBox\_menu.addAction(actionA) # 把动作A选项添加到菜单  
  
 actionB = QtWidgets.QAction(u'新建文件夹', groupBox\_menu)  
 groupBox\_menu.addAction(actionB)  
  
 actionC = QtWidgets.QAction(u'删除', groupBox\_menu)  
 groupBox\_menu.addAction(actionC)  
  
 actionD = QtWidgets.QAction(u'重命名', groupBox\_menu)  
 groupBox\_menu.addAction(actionD)  
  
 actionE = QtWidgets.QAction(u'属性', groupBox\_menu)  
 groupBox\_menu.addAction(actionE)  
  
 actionA.triggered.connect(self.create\_file)  
 actionB.triggered.connect(self.create\_folder)  
 actionC.triggered.connect(self.delete\_ob)  
 actionD.triggered.connect(self.rename\_ob)  
 actionE.triggered.connect(self.show\_attributes)  
  
 groupBox\_menu.popup(QtGui.QCursor.pos()) # 声明当鼠标在groupBox控件上右击时，在鼠标位置显示右键菜单,exec\_,popup两个都可以，  
  
 # 新建文件  
 def create\_file(self):  
 # 弹出一个输入对话框，让用户输入新文件名称  
 file\_name, ok = QtWidgets.QInputDialog.getText(None, '创建文件', '请输入新文件名称：')  
 if ok:  
 targetfolder = self.search\_ob(self.FileSys, self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 try:  
 targetfolder.create\_file(file\_name, self.FileSys.disk)  
 QtWidgets.QMessageBox.information(None, '创建文件', '文件创建成功！')  
 self.recover\_table(targetfolder)  
 except Exception as e:  
 QtWidgets.QMessageBox.warning(None, '创建文件', f'创建文件失败：{str(e)}')  
  
 # 新建文件夹  
 def create\_folder(self):  
 # 弹出一个输入对话框，让用户输入新文件夹名称  
 file\_name, ok = QtWidgets.QInputDialog.getText(None, '创建文件夹', '请输入新文件夹名称：')  
 if ok:  
 targetfolder = self.search\_ob(self.FileSys, self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 try:  
 targetfolder.create\_folder(file\_name, self.FileSys.disk)  
 QtWidgets.QMessageBox.information(None, '创建文件夹', '文件夹创建成功！')  
 self.recover\_table(targetfolder)  
 except Exception as e:  
 QtWidgets.QMessageBox.warning(None, '创建文件夹', f'创建文件夹失败：{str(e)}')  
  
 # 删除文件或文件夹  
 def delete\_ob(self):  
 targetfolder = self.search\_ob(self.FileSys, self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if targetfolder is not None:  
 selecteditem = self.tableWidget.selectedItems()[0].text() # 获取文件名  
 \_, file\_ext = os.path.splitext(selecteditem)  
 if file\_ext == '':  
 try:  
 targetfolder.delete\_folder(selecteditem, self.FileSys.disk)  
 self.recover\_table(targetfolder)  
 except:  
 QtWidgets.QMessageBox.information(None, '删除', '未选中对象无法删除！')  
 else:  
 try:  
 targetfolder.delete\_file(selecteditem, self.FileSys.disk)  
 self.recover\_table(targetfolder)  
 except:  
 QtWidgets.QMessageBox.information(None, '删除', '未选中对象无法删除！')  
 else:  
 QtWidgets.QMessageBox.information(None, '删除', '未知问题！未找到目标对象的父文件夹')  
  
 # 显示属性  
 def show\_attributes(self):  
 if self.tableWidget.selectedItems() != []:  
 selecteditem = self.search\_ob(self.FileSys, self.AddressBar.text()  
 + self.tableWidget.selectedItems()[0].text()) # 获取文件  
 else:  
 selecteditem = self.search\_ob(self.FileSys, self.AddressBar.text()) # 获取文件  
 if selecteditem is None:  
 return  
 mid\_name = selecteditem.get\_name()  
 mid\_path = selecteditem.get\_path()  
 mid\_size = str(selecteditem.get\_space() / 1024) + "KB"  
 mid\_ctime = None  
 mid\_mtime = None  
 mid\_atime = None  
 \_, mid\_type = os.path.splitext(selecteditem.get\_name())  
 if mid\_type == "":  
 mid\_type = "文件夹"  
 else:  
 mid\_ctime = datetime.fromtimestamp(int(os.path.getctime(mid\_path + mid\_name))).strftime("%Y-%m-%d %H:%M:%S")  
 mid\_mtime = datetime.fromtimestamp(int(os.path.getmtime(mid\_path + mid\_name))).strftime("%Y-%m-%d %H:%M:%S")  
 mid\_atime = datetime.fromtimestamp(int(os.path.getatime(mid\_path + mid\_name))).strftime("%Y-%m-%d %H:%M:%S")  
  
 qdialog = QtWidgets.QDialog()  
 qdialog.move(QtGui.QCursor.pos().x() + 30, QtGui.QCursor.pos().y() + 30)  
 qdialog.resize(400, 500)  
 qdialog.setWindowTitle(mid\_name + u"属性")  
 # 创建一个 QTabWidget 控件  
 tab\_widget = QtWidgets.QTabWidget()  
 tab\_widget.setMovable(True)  
  
 # 创建并添加属性页  
 tab = QtWidgets.QWidget()  
 tab1 = QtWidgets.QWidget()  
 tab2 = QtWidgets.QWidget()  
 tab3 = QtWidgets.QWidget()  
 tab\_widget.addTab(tab, "常规")  
 tab\_widget.addTab(tab1, "安全")  
 tab\_widget.addTab(tab2, "详细信息")  
 tab\_widget.addTab(tab3, "以前的版本")  
 # 属性页的信息(这里只用到常规页)  
 vlayout = QtWidgets.QVBoxLayout(tab)  
 label1 = QtWidgets.QLabel("文件类型：\t" + mid\_type)  
 label2 = QtWidgets.QLabel("路径：\t" + mid\_path)  
 label3 = QtWidgets.QLabel("大小：\t" + mid\_size)  
 vlayout.addWidget(label1)  
 vlayout.addWidget(label2)  
 vlayout.addWidget(label3)  
 if mid\_type != "文件夹":  
 label4 = QtWidgets.QLabel("创建时间：\t" + mid\_ctime)  
 label5 = QtWidgets.QLabel("修改时间：\t" + mid\_mtime)  
 label6 = QtWidgets.QLabel("访问时间：\t" + mid\_atime)  
 vlayout.addWidget(label4)  
 vlayout.addWidget(label5)  
 vlayout.addWidget(label6)  
 else:  
 label7 = QtWidgets.QLabel("文件总数（含文件夹）：\t" + str(selecteditem.get\_filenum()))  
 vlayout.addWidget(label7)  
  
 # 将属性页添加到弹窗中  
 layout = QtWidgets.QVBoxLayout()  
 layout.addWidget(tab\_widget)  
 qdialog.setLayout(layout)  
  
 # 显示弹窗  
 qdialog.exec\_()  
  
 # 格式化  
 def format(self):  
 # 删除本地的文件多余文件  
 for i in self.FileSys.folderList:  
 shutil.rmtree(i.folderPath + i.folderName + '/') # 删除本地的内容  
 for j in self.FileSys.fileList:  
 os.remove(j.filePath + j.fileName) # 删除本地的文件  
  
 # 读取pickle文件  
 with open("./Fengye7\_FileSystem/format\_disk.pickle", 'rb') as f1:  
 disk\_data\_pickle = pickle.load(f1)  
 # 写入到另一个pickle文件  
 with open("./Fengye7\_FileSystem/disk.pickle", 'wb') as f2:  
 pickle.dump(disk\_data\_pickle, f2)  
 # 同时修改副本方便查看  
 with open("./Fengye7\_FileSystem/format\_disk.txt", 'r') as f3:  
 disk\_data\_txt = f3.read()  
 with open("./Fengye7\_FileSystem/disk.txt", 'w') as f4:  
 f4.write(disk\_data\_txt)  
 # \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*同样的下面格式化目录\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 with open("./Fengye7\_FileSystem/format\_directory.pickle", 'rb') as f5:  
 directory\_data\_pickle = pickle.load(f5)  
 # 写入到另一个pickle文件  
 with open("./Fengye7\_FileSystem/directory.pickle", 'wb') as f6:  
 pickle.dump(directory\_data\_pickle, f6)  
 # 同时修改副本方便查看  
 with open("./Fengye7\_FileSystem/format\_directory.txt", 'r') as f7:  
 directory\_data\_txt = f7.read()  
 with open("./Fengye7\_FileSystem/directory.txt", 'w') as f8:  
 f8.write(directory\_data\_txt)  
 QtWidgets.QMessageBox.information(None, '格式化', '格式化完毕！下面自动关闭，请重启！')  
 QtCore.QCoreApplication.exit()  
  
 # 重命名  
 def rename\_ob(self):  
 targetfolder = self.search\_ob(self.FileSys, self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if targetfolder is not None:  
 selecteditem = self.tableWidget.selectedItems()[0].text() # 获取文件名  
 \_, file\_ext = os.path.splitext(selecteditem)  
 # 弹出一个输入对话框，让用户输入新文件夹名称  
 new\_name, ok = QtWidgets.QInputDialog.getText(None, '重命名', '请输入新名称：')  
 if ok:  
 if file\_ext == "":  
 try:  
 targetfolder.rename\_folder(selecteditem, new\_name)  
 except:  
 QtWidgets.QMessageBox.information(None, '重命名', '重命名文件夹失败！')  
 else:  
 try:  
 targetfolder.rename\_file(selecteditem, new\_name)  
 except:  
 QtWidgets.QMessageBox.information(None, '重命名', '重命名文件失败！')  
 self.recover\_table()  
 else:  
 QtWidgets.QMessageBox.information(None, '删除', '未知问题！未找到目标对象的父文件夹')  
  
 # 展示文件列表  
 def show\_table(self, folder):  
 # 处理地址栏  
 self.AddressBar.setText(folder.folderPath + folder.folderName + '/')  
 # 优先展示文件夹  
 if len(folder.folderList) != 0:  
 for i in folder.folderList:  
 self.tableWidget.insertRow(self.tableWidget.rowCount()) # 添加行  
 newItem = QtWidgets.QTableWidgetItem(QtGui.QIcon("./imags/folder.jpg"), i.folderName)  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 0, newItem)  
  
 midtime = datetime.fromtimestamp(int(os.path.getmtime(i.folderPath)))  
 strtime = midtime.strftime("%Y-%m-%d %H:%M:%S")  
 newtime = QtWidgets.QTableWidgetItem(strtime)  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 1, newtime)  
  
 newtype = QtWidgets.QTableWidgetItem("文件夹")  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 2, newtype)  
  
 newsize = QtWidgets.QTableWidgetItem(str(i.get\_space() / 1024) + "KB")  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 3, newsize)  
 # 列出文件  
 if len(folder.fileList) != 0:  
 for i in folder.fileList:  
 self.tableWidget.insertRow(self.tableWidget.rowCount()) # 添加行  
 \_, file\_ext = os.path.splitext(i.fileName)  
 try:  
 newItem = QtWidgets.QTableWidgetItem(QtGui.QIcon("./imags/" + file\_ext + ".png"), i.fileName)  
 except:  
 newItem = QtWidgets.QTableWidgetItem(QtGui.QIcon("./imags/.txt.png"), i.fileName) # 其他没有提供图标的文件类型  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 0, newItem)  
  
 midtime = datetime.fromtimestamp(int(os.path.getmtime(i.filePath)))  
 strtime = midtime.strftime("%Y-%m-%d %H:%M:%S")  
 newtime = QtWidgets.QTableWidgetItem(strtime)  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 1, newtime)  
  
 \_, file\_ext = os.path.splitext(i.fileName)  
 newtype = QtWidgets.QTableWidgetItem(file\_ext + "文件")  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 2, newtype)  
  
 newsize = QtWidgets.QTableWidgetItem(str(i.get\_space() / 1024) + "KB")  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 3, newsize)  
 # 若文件夹为空  
 if len(folder.folderList) == 0 and len(folder.fileList) == 0:  
 self.tableWidget.insertRow(self.tableWidget.rowCount()) # 添加行  
 newItem = QtWidgets.QTableWidgetItem("此文件夹为空。")  
 self.tableWidget.setItem(self.tableWidget.rowCount() - 1, 0, newItem)  
 # 处理下框显示  
 self.item\_num\_label.setText(str(folder.get\_filenum()))  
 self.selected\_item\_num.setText("选中0个项目")  
 self.selected\_item\_size.setText("0KB")  
  
 # 展示目录  
 def show\_list(self, tree, mid\_list, mid\_type): # 这里原来是想文件数上展示文件夹和文件的，用mid\_type区分操作,现摒弃  
 if mid\_type == Type.Folder:  
 for i in mid\_list:  
 newItem = QtWidgets.QTreeWidgetItem()  
 newItem.setText(0, i.folderName)  
 newItem.setIcon(0, QtGui.QIcon("./imags/folder.jpg"))  
 for j in i.folderList:  
 childItem = QtWidgets.QTreeWidgetItem()  
 childItem.setText(0, j.folderName)  
 childItem.setIcon(0, QtGui.QIcon("./imags/folder.jpg"))  
 newItem.addChild(childItem)  
 if len(j.folderList) != 0:  
 self.show\_list(childItem, j.folderList, Type.Folder)  
 # if len(j.fileList) != 0:  
 # self.show\_list(childItem, j.fileList, Type.File)  
 tree.addTopLevelItem(newItem)  
 # for j in i.fileList:  
 # childItem = QtWidgets.QTreeWidgetItem()  
 # childItem.setText(0, j.fileName)  
 # childItem.setIcon(0, QtGui.QIcon("./imags/file.jpg"))  
 # newItem.addChild(childItem)  
 # if mid\_type == Type.File:  
 # for i in mid\_list:  
 # newItem = QtWidgets.QTreeWidgetItem()  
 # newItem.setText(0, i.fileName)  
 # newItem.setIcon(0, QtGui.QIcon("./imags/file.jpg"))  
 # tree.addTopLevelItem(newItem)  
  
 # 处理列表双击事件  
 def enter\_item(self):  
 selecteditem = self.tableWidget.selectedItems()[0].text() # 获取文件名  
 \_, file\_ext = os.path.splitext(selecteditem)  
 # 双击文件夹  
 if file\_ext == "":  
 targetfolder = self.search\_ob(self.FileSys,  
 self.AddressBar.text() + selecteditem) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if targetfolder is not None:  
 self.tableWidget.setRowCount(0) # 清空列表行  
 self.tableWidget.clearContents() # 清空内容  
 self.show\_table(targetfolder) # 展示新位置  
 else:  
 QtWidgets.QMessageBox.information(None, 'error', '可能的情况：\n' +  
 '\*该对象是无后缀名的文件，暂不支持打开！\n' +  
 '\* 未查找到相关的文件夹')  
 # 双击文件  
 else:  
 targetfile = self.search\_ob(self.FileSys, self.AddressBar.text() + selecteditem)  
 if targetfile is not None:  
 self.on\_modified(targetfile)  
  
 # 监听文件保存  
 class Monitor:  
 def \_\_init\_\_(self, father, targetfile):  
 self.father = father  
 self.targetfile = targetfile  
  
 class OnCloseHandler(PatternMatchingEventHandler):  
 def \_\_init\_\_(self, father):  
 *"""  
 内部类初始化函数，构造时自动调用* ***:param*** *father: 外部类的实例对象  
 """* self.father = father  
  
 def dispatch(self, event):  
 print(event.event\_type)  
 if event.event\_type == 'modified':  
 self.on\_modified(event)  
  
 def on\_modified(self, event):  
 if event.src\_path == self.father.targetfile.filePath + self.father.targetfile.fileName:  
 new\_data = self.father.targetfile.get\_data()  
 # 每次文件更改需要调整文件占用的整个簇链，因为可能大小不变，内容变了  
 start\_cluster = self.father.father.disk[self.father.targetfile.start\_cluster]  
 end\_cluster = self.father.father.disk[self.father.targetfile.end\_cluster]  
 mid\_cluster = start\_cluster  
 mid\_len = len(new\_data)  
 while True:  
 if mid\_len > 1024:  
 mid\_cluster["size"] = 0  
 mid\_cluster["data"] = new\_data[:1024] # 字符串的切片  
 new\_data = new\_data[1024:]  
 mid\_len = len(new\_data)  
 if mid\_cluster["next"] is not None and mid\_cluster != end\_cluster:  
 mid\_cluster = self.father.father.disk[mid\_cluster["next"]]  
 else:  
 break  
 else:  
 mid\_cluster["size"] = 1024 - mid\_len  
 mid\_cluster["data"] = new\_data  
 new\_data = None  
 mid\_len = 0  
 break  
 if mid\_cluster != end\_cluster: # 这种情况就是文件变小了，将后面的释放  
 mid\_cluster["next"] = end\_cluster["next"] # 重连簇链，下面释放空间  
 while mid\_cluster != self.father.father.disk[end\_cluster["next"]]:  
 mid\_cluster["data"] = None  
 mid\_cluster["status"] = "free"  
 mid\_cluster["size"] = 1024  
 mid\_id = mid\_cluster["next"]  
 mid\_cluster["next"] = None  
 mid\_cluster = self.father.father.disk[mid\_id]  
 else: # 这种情况就是文件扩大或不变  
 while mid\_len > 0:  
 # 计算空间，只要还有空闲簇都能创建，空文件占用一个簇，当该文件文件输入内容超过该簇后，申请新簇，直到没有空闲簇  
 cluster\_id = None  
 for i in self.father.father.disk:  
 if i["status"] == "free":  
 cluster\_id = i["cluster\_id"]  
 break  
 mid\_id = end\_cluster["next"]  
 end\_cluster["next"] = cluster\_id  
 self.father.father.disk[cluster\_id]["next"] = mid\_id # 这三行在原来的簇链中插入新簇  
 self.father.father.disk[cluster\_id]["status"] = "busy" # 表示次簇已被使用  
 if mid\_len > 1024:  
 self.father.father.disk[cluster\_id]["size"] = 0  
 self.father.father.disk[cluster\_id]["data"] = new\_data[:1024] # 字符串的切片  
 new\_data = new\_data[1024:]  
 mid\_len = len(new\_data)  
 else:  
 self.father.father.disk[cluster\_id]["size"] = 1024 - mid\_len  
 self.father.father.disk[cluster\_id]["data"] = new\_data  
 new\_data = None  
 mid\_len = 0  
 break  
  
 # 处理文件外部打开用于读写，关闭后写入磁盘  
 def on\_modified(self, targetfile):  
 # 监听文件关闭  
 handler = self.Monitor(self.FileSys, targetfile)  
 observer = Observer()  
 observer.schedule(handler.OnCloseHandler(handler), path=targetfile.filePath, recursive=False)  
 observer.start()  
 targetfile.open\_file()  
  
 # 处理目录树点击事件  
 def enter\_list(self):  
 selecteditem = self.treeWidget.selectedItems()[0].text(0) # 获取文件对应的节点  
 \_, file\_ext = os.path.splitext(selecteditem)  
 # 点击文件夹  
 if file\_ext == "":  
 targetpath = self.getpath(self.treeWidget.currentItem())  
 targetfolder = self.search\_ob(self.FileSys, targetpath)  
 if targetfolder is not None:  
 self.tableWidget.setRowCount(0) # 清空列表行  
 self.tableWidget.clearContents() # 清空内容  
 self.show\_table(targetfolder) # 展示新位置  
  
 # 目录树从一个节点获取路径  
 def getpath(self, item):  
 if item.parent():  
 temp = item.text(0)  
 parent = self.getpath(item.parent()) # 递归获取完整路径  
 if parent is not None:  
 res = os.path.join(parent, temp)  
 return res  
 else:  
 return temp  
 else:  
 return "./Fengye7\_FileSystem/" + item.text(0) + '/'  
  
 # 处理页面跳转  
 def jump\_path(self):  
 tagetfolder = self.search\_ob(self.FileSys,  
 self.AddressBar.text()) # 获取文件对象,从系统开始文件夹开始递归搜索  
 if tagetfolder is not None:  
 self.tableWidget.setRowCount(0) # 清空列表行  
 self.tableWidget.clearContents() # 清空内容  
 self.show\_table(tagetfolder) # 展示新位置  
 else:  
 QtWidgets.QMessageBox.warning(None, '跳转',  
 f'跳转失败！目标目录不存在。\n请点击目录刷新地址栏信息，以便其他操作！！！')  
  
 # 搜索路径对应的文件对象  
 def search\_ob(self, folder, path):  
 if path == './Fengye7\_FileSystem/':  
 return self.FileSys  
 \_, file\_ext = os.path.splitext(path)  
 if file\_ext == "":  
 if len(folder.folderList) != 0:  
 answer = None  
 for i in folder.folderList:  
 if i.folderPath + i.folderName == path or i.folderPath + i.folderName + '/' == path:  
 # print(i.folderName)测试是否找到  
 answer = i  
 if len(i.folderList) != 0 and answer is None: # 没有搜索到则继续进入子文件夹搜索  
 answer = self.search\_ob(i, path)  
 if answer is not None:  
 break  
 return answer  
 else: # 查询到底  
 return None  
 else: # 此种情况应该是查找文件  
 if len(folder.fileList) != 0:  
 answer = None  
 for i in folder.fileList:  
 if i.filePath + i.fileName == path:  
 answer = i  
 if len(folder.folderList) != 0 and answer is None:  
 for j in folder.folderList:  
 answer = self.search\_ob(j, path)  
 if answer is not None:  
 break  
 return answer  
 else:  
 return None  
  
  
import sys  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 适配2k高分辨率屏幕  
 QtCore.QCoreApplication.setAttribute(QtCore.Qt.AA\_EnableHighDpiScaling)  
  
 app = QtWidgets.QApplication(sys.argv)  
 MainWindow = QtWidgets.QMainWindow() # 创建窗体对象  
 ui = Ui\_Fengye7\_FileSystem(MainWindow) # 创建PyQt设计的窗体对象  
 MainWindow.show() # 显示窗体  
 sys.exit(app.exec\_()) # 程序关闭时退出进程