**文件系统管理设计文档**

2253893 苗君文

**1. 项目简介**

**1.1. 基本任务**

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。

实现过程中，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。

**1.2. 功能描述**

本项目的实现中使用FAT的显式链接管理文件存储空间，并通过遍历FAT表查找状态为-2的物理块以管理磁盘空闲空间，文件目录采用多级目录结构，退出文件系统时可以选择将文件系统内容写入磁盘中以永久保存。

此文件系统可提供的操作包括：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。

**2. 开发环境**

系统：Windows 11

IDE：PyCharm 2023.3.3

语言：Python 3.11.5

环境：PyQt5 5.15.7

**3. 系统架构**

**3.1. 项目结构**

该文件系统管理项目使用python开发，基于PyQt5进行构建。项目的结构如下：

.

├── main.py

├── src.py

└── README.md

main.py: 负责文件系统的界面操作，包括启动、格式化、文件和目录的创建、删除等功能。它主要是用户界面和控制逻辑的结合。

src.py: 实现了文件系统的核心逻辑，包括文件分配表（FAT）的管理、物理块（Block）的读写、文件目录（Catalog）和文件控制块（FCB）的定义和操作等。

**3.2. 类设计**

首先介绍main.py中的类，均与用户界面的交互有关。

**3.2.1. EditingInterface 类**

 初始化界面：设置编辑界面，包括文本编辑区域和保存按钮。

 读取文件内容：从文件系统中读取文件内容并显示在文本编辑区域。

 保存文件内容：将编辑后的文件内容保存回文件系统。

 关闭编辑界面：关闭编辑界面，返回到主界面。

EditingInterface 类用于文件内容的编辑操作。当用户选择编辑文件时，会弹出一个编辑界面，用户可以在其中修改文件内容，并将修改后的内容保存到文件系统中。主要功能包括从文件系统读取文件内容、显示在编辑区域、保存修改和关闭编辑界面。

**3.2.2. AttributeInterface 类**

显示文件或目录属性：显示选中文件或目录的详细属性，包括名称、类型、创建时间、修改时间等。

编辑文件或目录属性：允许用户修改文件或目录的部分属性，如名称。

保存属性修改：将修改后的属性保存回文件系统。

关闭属性界面：关闭属性界面，返回到主界面。

AttributeInterface 类用于显示和编辑文件或目录的属性。用户可以查看选中文件或目录的详细信息，并进行必要的修改。修改后的属性会被保存到文件系统中，确保属性信息的更新。

**3.2.3. ListWidget 类**

显示当前目录内容：显示当前目录下的所有文件和子目录。

更新显示：在目录内容发生变化时，更新显示内容。

处理用户选择：处理用户对列表中项目的选择操作，如打开文件、进入子目录等。

支持拖放操作：支持文件和目录的拖放操作，实现文件和目录的移动或复制。

ListWidget 类用于显示当前目录的内容，并处理用户的选择和操作。它提供了一个可视化的文件和目录列表，用户可以在其中浏览、选择和操作文件和目录。ListWidget 类还支持拖放操作，允许用户通过拖放实现文件和目录的移动或复制。

**3.2.4. FileSystemUI 类**

（1）初始化和界面设置：

设置主窗口和用户界面，包括菜单、按钮、列表视图等。

初始化文件系统数据结构，如FAT、磁盘块和目录。

（2）文件系统操作：

格式化：清空FAT表、磁盘块和目录结构，重新初始化文件系统。

创建文件和目录：在当前目录下创建新的文件或子目录，并更新界面显示。

删除文件和目录：从当前目录下删除指定的文件或子目录，并更新界面显示。

显示目录：显示当前目录下的文件和子目录，更新列表视图。

更改当前目录：切换当前目录到指定的子目录或父目录，并更新路径显示。

（3）文件操作：

打开文件：打开指定的文件，读取并显示文件内容。

关闭文件：关闭当前打开的文件，清空显示区域。

写文件：将数据写入当前打开的文件，保存修改并更新FAT表和磁盘块。

读文件：从当前打开的文件中读取数据，并显示在界面上。

（4）界面交互：

点击事件处理：处理用户在目录树或文件列表中的点击事件，执行相应的文件系统操作。

更新显示：在文件系统操作后，更新目录树、文件列表和路径显示。

FileSystemUI 类是文件系统的用户界面类，负责用户与文件系统的交互操作。通过图形界面提供文件和目录的创建、删除、读取和写入操作，同时管理文件系统的数据结构（如FAT表、磁盘块和目录）。具体操作通过调用底层逻辑（如FAT、Block和Catalog类）实现，并在界面上实时更新操作结果。

文件系统的核心部分的类设计放在src.py中，涉及物理块、FAT文件分配表、FCB文件控制块、文件目录。具体介绍如下：

**3.2.5. Block 类**

blockID: 物理块的唯一标识符，用于区分不同的块。

data: 存储在块中的数据，初始为空字符串。

write(data): 将数据写入块中，替换当前存储的数据。

read(): 从块中读取并返回数据。

clear(): 清空块中的数据，将其重置为空字符串。

Block 类表示磁盘中的一个块，主要用于数据的基本存储单元操作。通过 write 方法将数据写入块，通过 read 方法读取块中的数据，通过 clear 方法清空块。

**3.2.6. FAT类**

fat: 一个数组，记录每个块的状态和链表结构。

allocate(size): 分配指定数量的连续块。如果空间不足，返回 -1；否则返回起始块号。

delete(start, disk): 从指定的起始块开始，删除链表中的所有块，释放它们，并清空相应块的数据。

update(start, data, disk): 更新文件数据，从起始块开始删除原有数据，并写入新的数据。

read(start, disk): 从起始块开始，读取文件的所有数据。

get\_usage\_percentage(): 计算并返回磁盘已使用块的百分比。

FAT 类实现了使用显式链接管理文件存储空间的管理。通过 allocate 方法分配新的块，通过 delete 方法删除文件占用的块，通过 update 方法更新文件内容，通过 read 方法读取文件内容。get\_usage\_percentage 方法用于监控磁盘空间的使用情况。并且通过 FAT 实现对空闲空间的管理。

**3.2.7. FCB类**

name: 文件名，标识文件。

createTime: 文件的创建时间。

updateTime: 文件的最后修改时间，初始值为创建时间。

start: 文件在 FAT 表中的起始块号。

update(newData, fat, disk): 更新文件内容，重新分配并写入数据，同时更新修改时间。

delete(fat, disk): 删除文件，从 FAT 表中释放所有相关块。

read(fat, disk): 读取文件内容，返回文件的完整数据。

FCB 类表示文件控制块，管理文件的基本信息和数据存储。通过 update 方法更新文件内容，通过 delete 方法删除文件，通过 read 方法读取文件内容。

**3.2.8. Catalog 类**

name: 目录或文件的名称。

isFile: 表示该节点是否为文件。

parent: 父目录节点，如果是根目录则为 None。

createTime: 节点的创建时间。

updateTime: 节点的最后修改时间，初始值为创建时间。

children: 子节点列表，仅在节点为目录时使用。

data: 文件控制块，仅在节点为文件时使用。

Catalog 类实现了多级目录结构。目录节点包含子节点列表，文件节点包含文件控制块（FCB）实例。Catalog 类用于管理文件和目录的层次结构，包括创建、删除、读取和更新操作。

**3.3. 类之间的交互**

用户通过 FileSystemUI 类与文件系统交互，执行文件和目录的操作。

FileSystemUI 类调用 ListWidget 类显示当前目录的内容，并处理用户的选择和操作。

当用户选择编辑文件时，FileSystemUI 类调用 EditingInterface 类显示编辑界面，处理文件内容的读写操作。

当用户选择查看或编辑文件或目录属性时，FileSystemUI 类调用 AttributeInterface 类显示属性界面，处理属性的显示和修改。

所有的文件和目录操作都通过 FAT、Block、FCB 和 Catalog 类实现，确保文件系统的数据管理和存储。

**4. 核心算法**

**4.1. 格式化**

格式化功能通过format函数实现。文件系统的格式化功能用于清空整个文件系统的数据，包括文件分配表（FAT）、磁盘块和目录结构。在进行格式化操作前，确保没有正在进行的文件编辑操作，关闭所有编辑界面。而后，弹出一个消息框，提示用户确认是否要格式化磁盘。若用户选择“取消”，格式化操作终止，返回主界面。接着，清空FAT表，创建一个新的 FAT 实例，将所有块的状态设置为未使用（用 -2 表示）。将清空后的 FAT 表保存到磁盘文件中（如 fat 文件），以便持久化存储。清空物理块，创建一个新的物理块列表，每个块都是一个新的 Block 实例，表示磁盘中的一个存储单元。初始化每个块，确保其数据为空。将清空后的磁盘块列表保存到磁盘文件中（如 disk 文件）。清空文件目录，创建一个新的目录列表，初始化根目录，将清空后的目录结构保存到磁盘文件中（如 catalog 文件）。最后重新加载用户界面，更新目录树的显示。

**4.2. 创建子目录**

创建子目录功能通过create\_folder函数实现。该功能用于在当前目录下创建一个新的子目录，并更新文件系统的目录结构和用户界面。先创建一个新的 QListWidgetItem 实例，使用文件夹图标和默认名称“新建文件夹”，将该目录项添加到列表视图中。而后，更新目录结构，将新建的目录项添加到当前目录的子目录列表中，添加到文件系统的目录表中，并更新目录树的显示。

**4.3. 删除子目录/删除文件**

删除子目录/删除文件功能通过delete函数实现。首先检查首先检查是否有选中的文件或文件夹。如果没有选中任何项目，直接返回。获取最后一个选中的项目以及其在列表中的索引。而后，弹出提示框，询问用户是否确认删除选中的文件或文件夹。接着，执行删除操作，从列表视图中移除选中的项目。通过递归函数deleteFileRecursive递归删除文件夹及其内部所有文件；或者直接删除文件。并更新FAT表和目录表，更新目录树视图。

**4.4. 显示目录**

目录显示功能包括两个主要部分：目录树和路径名。目录树显示在窗口的左侧，路径名显示在窗口的上侧。目录树的实现分为以下步骤：首先初始化根节点和跟项目，根节点rootNode表示整个目录结构的根，rootItem表示树形控件的根项目。update\_tree函数用于更新目录树，确保当前目录结构的所有变化在界面上及时显现。使用函数updateTreeRecursive递归遍历目录节点及其子节点，并更新树形控件中的对应项目。根据节点类型（文件或文件夹），设置不同的图标。根据子节点的数量调整树形控件中的子项目，确保树形控件与目录结构保持一致。对于地址栏，使用update\_address\_bar函数更新窗口上方的路径名，显示当前所在的目录路径，并在状态栏显示当前目录下的项目数量。

**4.5. 更改当前目录**

用户可以通过点击左侧目录树的文件夹或点击右侧展示栏的文件夹或文件或点击左上角的“返回”或“前进”键进行更改当前目录。

（1）点击目录树节点跳转：获取点击节点的路径，存储在ways列表中。从当前目录回退到根节点。清空路径和路径树，只保留根节点。根据路径中的每个节点，依次查找对应的目录并跳转。更新地址栏内容。设置回退和前进按钮的可用状态。

（2）双击或右键文件夹菜单打开文件夹：获取用户点击的项目，如果出错，则尝试获取最后一个选中的项目。如果可以前进，并且nextStep为真，则获取上次位置对应的项目，并重置lastLoc和nextStep。在当前节点的子节点中查找与项目名称相同的节点。如果找到的是文件，读取文件数据并打开编辑窗口显示文件内容。如果找到的是目录，更新当前节点并加载新节点的文件。更新路径和地址栏，设置回退按钮的可用状态。

（3）通过“返回”或“前进”按钮进行跳转：点击“返回”按钮，返回到上一级目录。关闭可能正在进行的文件编辑。如果当前节点是根节点，则无法返回。记录当前节点在父节点中的位置。更新当前节点为父节点。更新路径和路径树，设置回退和前进按钮的可用状态。点击“前进”按钮，跳转至下一级目录，设置nextStep为真，而后调用open\_file函数实现跳转。

**4.6. 创建文件**

创建文件功能通过 create\_file 函数实现。创建一个新文件项，并设置其图标为文件图标，初始名称为“新建文件”，将新文件项添加到listView中，调用editLast函数云讯用户立即编辑文件名，然后创建一个新的Catalog对象，表示新文件的目录项。将新文件的名称、类型（文件）、文件分配表（FAT）、磁盘、创建时间和父节点（当前节点）等信息传递给Catalog对象。将新创建的Catalog对象添加到当前目录的子节点和全局目录表中。最后更新目录树。

**4.7. 打开文件**

打开文件功能通过 open\_file 函数实现。首先，关闭任何可能正在进行的文件编辑，确保没有未保存的更改。然后尝试获取用户点击的项目，如果获取失败（例如用户右键点击菜单时），检查是否有选中的项目。如果没有选中任何项目，则直接返回；否则，获取最后一个选中的项目。如果当前处于前进状态（由lastLoc和nextStep标识），则获取前进位置的项目，并重置前进标识。在当前目录节点的子节点中查找与项目名称相同的节点，找到后存储到newNode中。如果找到的节点是文件，则读取文件数据；打开一个编辑窗口，显示文件内容，并允许用户进行编辑；设置信号连接，确保用户保存编辑内容时，更新文件数据和更新时间。如果节点是文件夹，则更新当前节点为新节点，并加载新节点的文件内容；更新当前路径，添加新节点名称到路径列表；在树状视图中找到对应的新节点项目，并设置为当前项目，允许用户在新目录中浏览。此外，更新地址栏，显示当前路径；更新目录树，反映新目录或文件的状态。根据情况启用或禁用回退和前进按钮，确保导航功能正常。

**4.8. 关闭文件**

关闭文件会触发closeEvent函数。首先系统会检查文件是否被修改，比较当前文本编辑框中的内容与初始内容。如果两者相同，表示文件未被修改，可以直接关闭窗口。弹出保存提示框，若用户选择“保存”，则将当前文本编辑框中的内容保存到文件中，接受关闭事件，关闭窗口。若用户选择“不保存”，则直接接受关闭时间，关闭窗口，不保存修改；若用户选择“取消”，则忽略关闭时间，保持窗口打开状态，让用户继续编辑事件。

**4.9. 写文件**

写文件的功能，即在用户编辑文件并选择保存时，将编辑后的内容写入磁盘中。在打开文件时，首先关闭可能正在进行的其他文件编辑窗口，确保只有一个编辑窗口处于活动状态。通过列表视图中的索引，尝试获取用户点击的项目。如果出现异常（例如用户通过右键菜单操作），则获取最后一个选中的项目。通过getData函数获取数据并将新数据写入文件系统，并更新文件的修改时间。此外，还需更新地址栏和树视图。

**4.10. 读文件**

如果当前节点是文件，则可以读取文件数据，打开一个编辑窗口显示文件内容。并且可以通过update\_status\_bar函数显示当前文件所含的总字数。

**4.11. 属性界面**

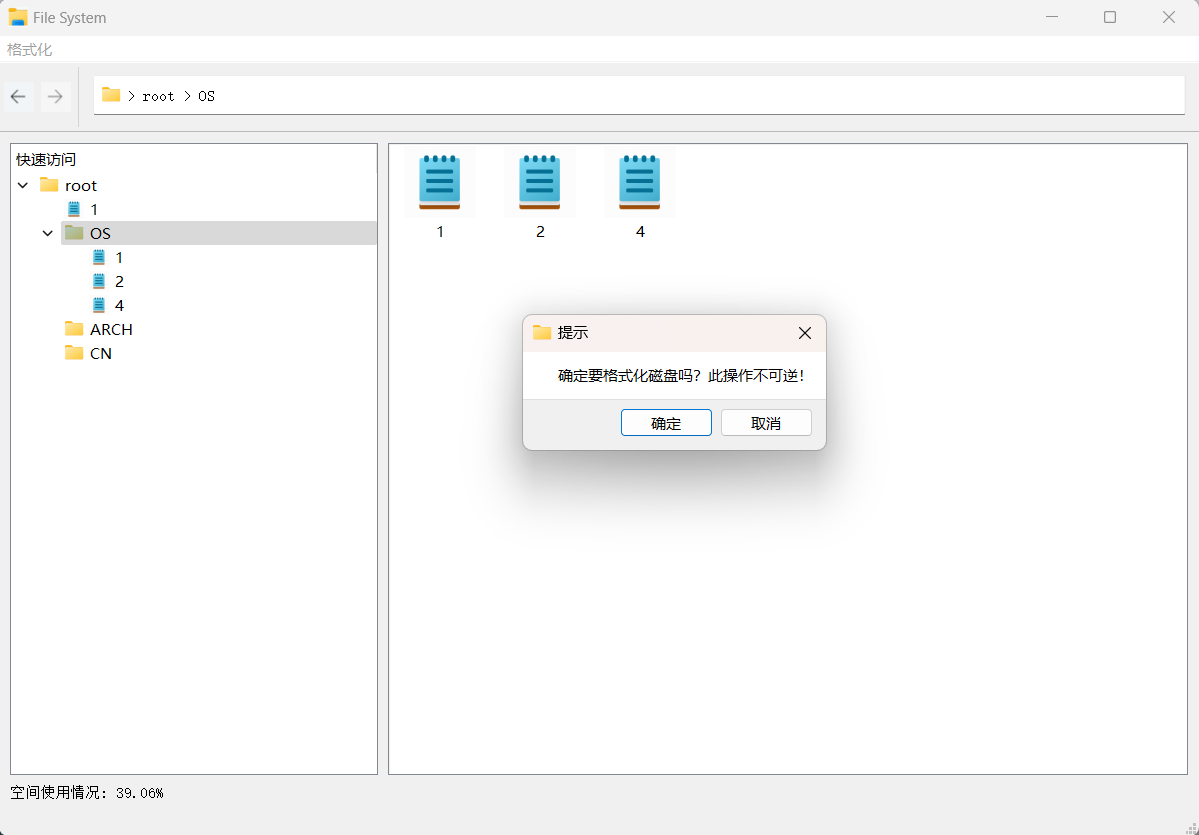
属性界面的展示通过AttributeInterface得以实现。其构造函数接受文件或文件夹的名称、类型（是文件还是文件夹）、创建时间、更新时间（对于文件）、子项目数（对于文件夹）。然后，设置了窗口的标题和图标，并创建了垂直布局。接下来，根据文件类型加载对应的图标，并将图标和名称添加到水平布局中，随后将水平布局添加到垂直布局中。然后，根据文件类型添加创建时间或者更新时间（对于文件）、子项目数（对于文件夹）到垂直布局中。最后，将垂直布局设置为窗口的主布局。格式化时间的函数根据给定的时间返回格式化后的字符串，包括年、月、日、时、分、秒。

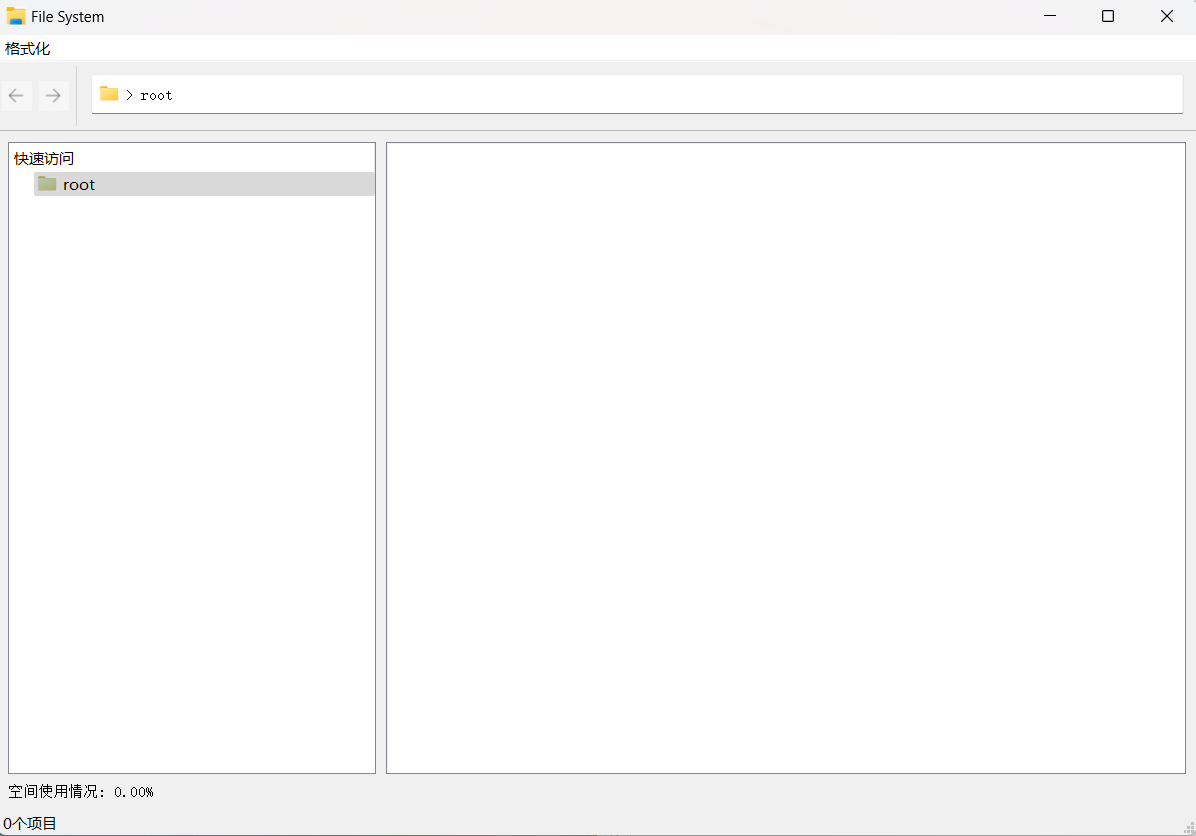
**4.12. 磁盘空间使用率**

通过update\_usage\_label()函数计算出磁盘的空间使用率，并精确到小数点后两位，作为label显示在左下角。

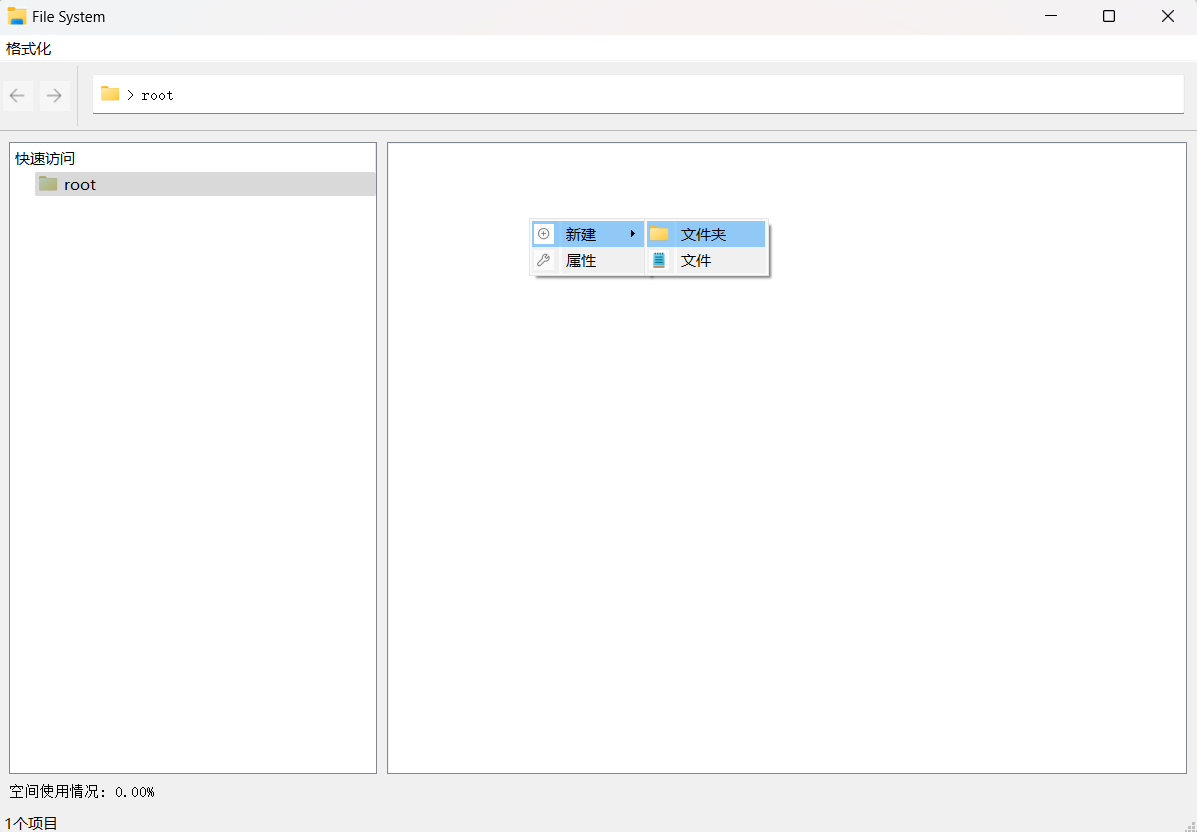
**5. 用户界面设计**

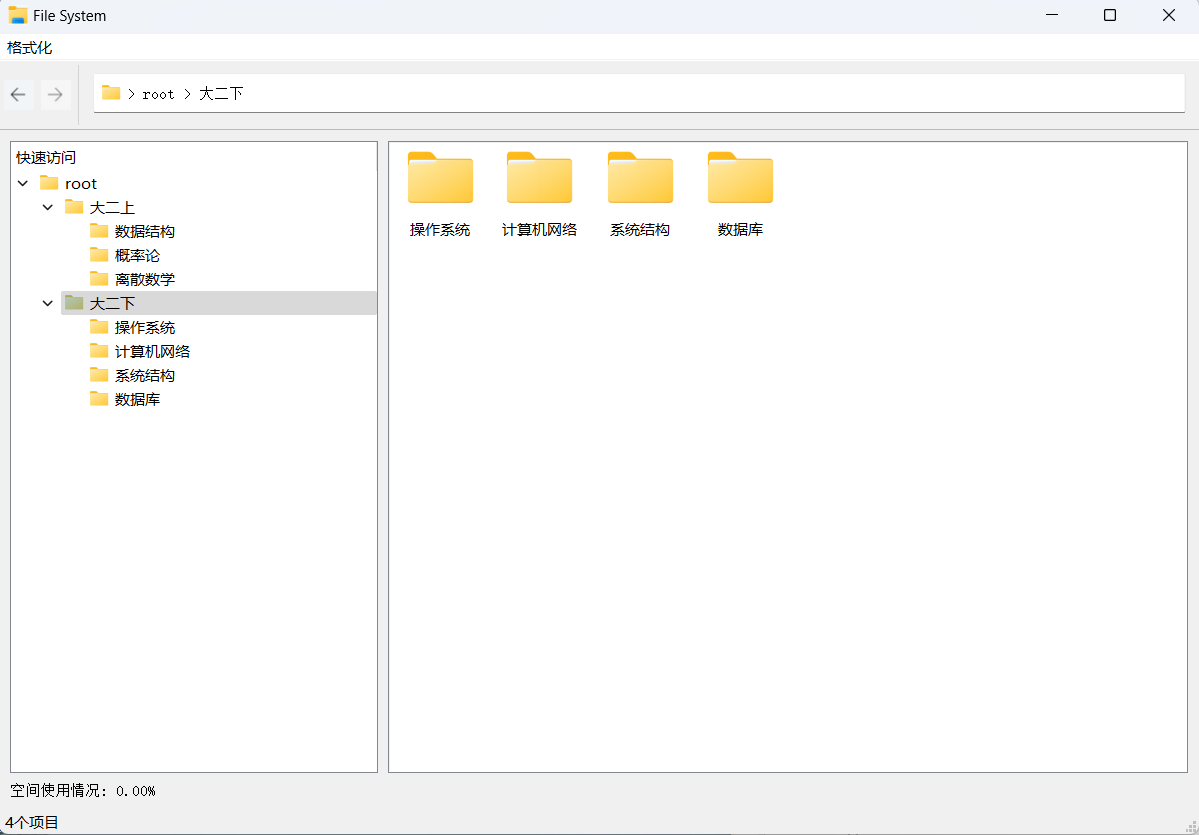
**5.1. 格式化**



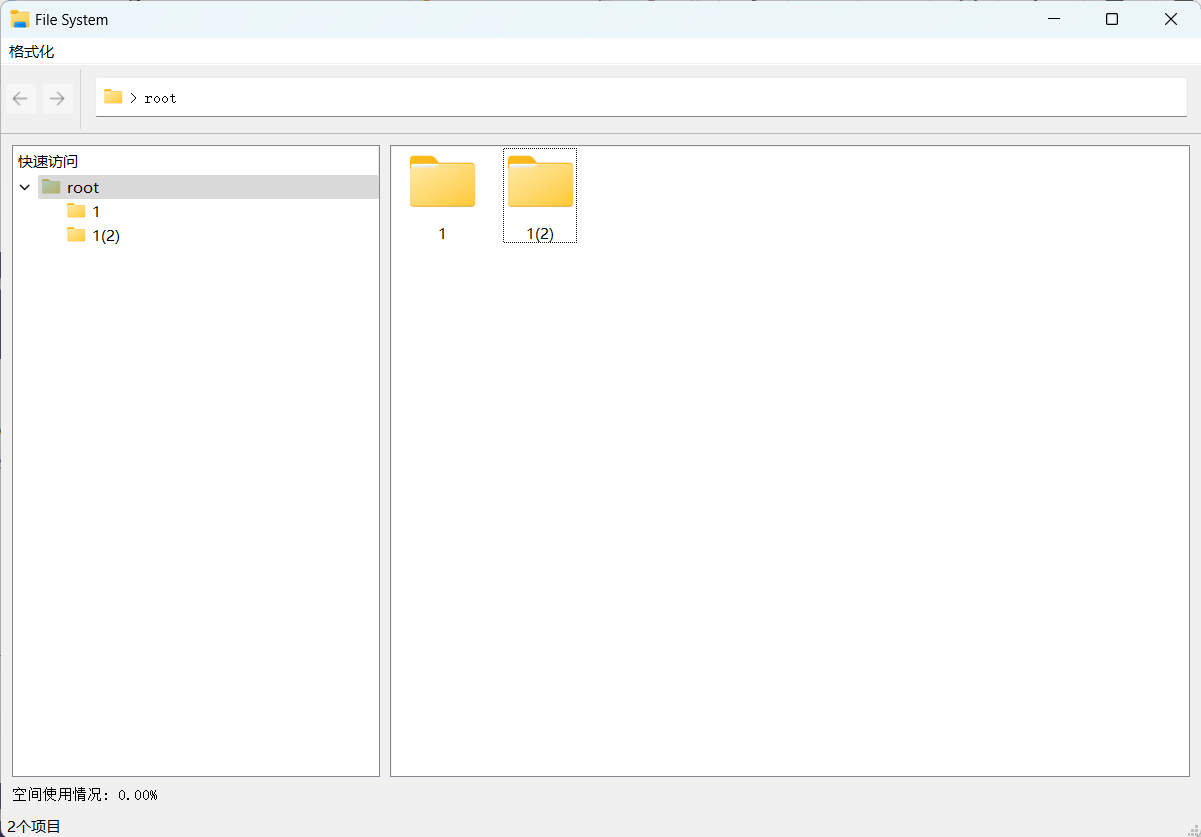


**5.2. 创建子目录**

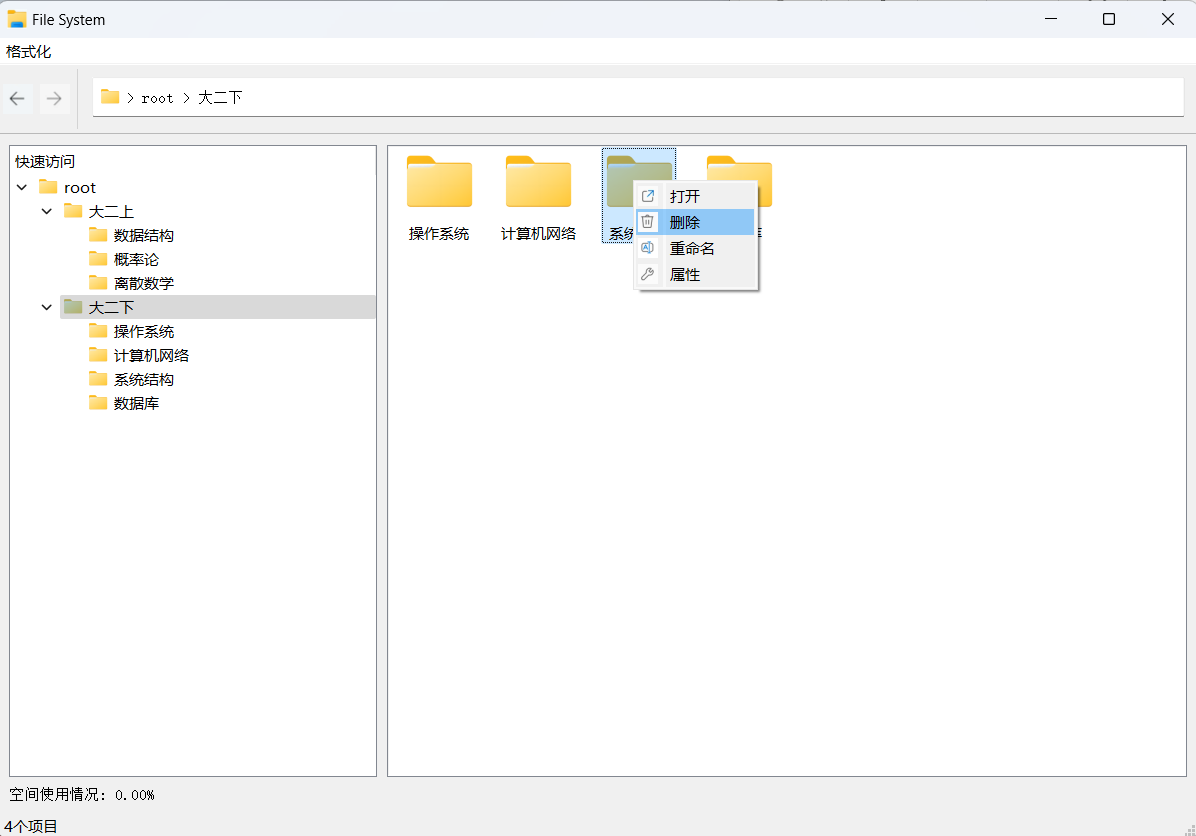
****

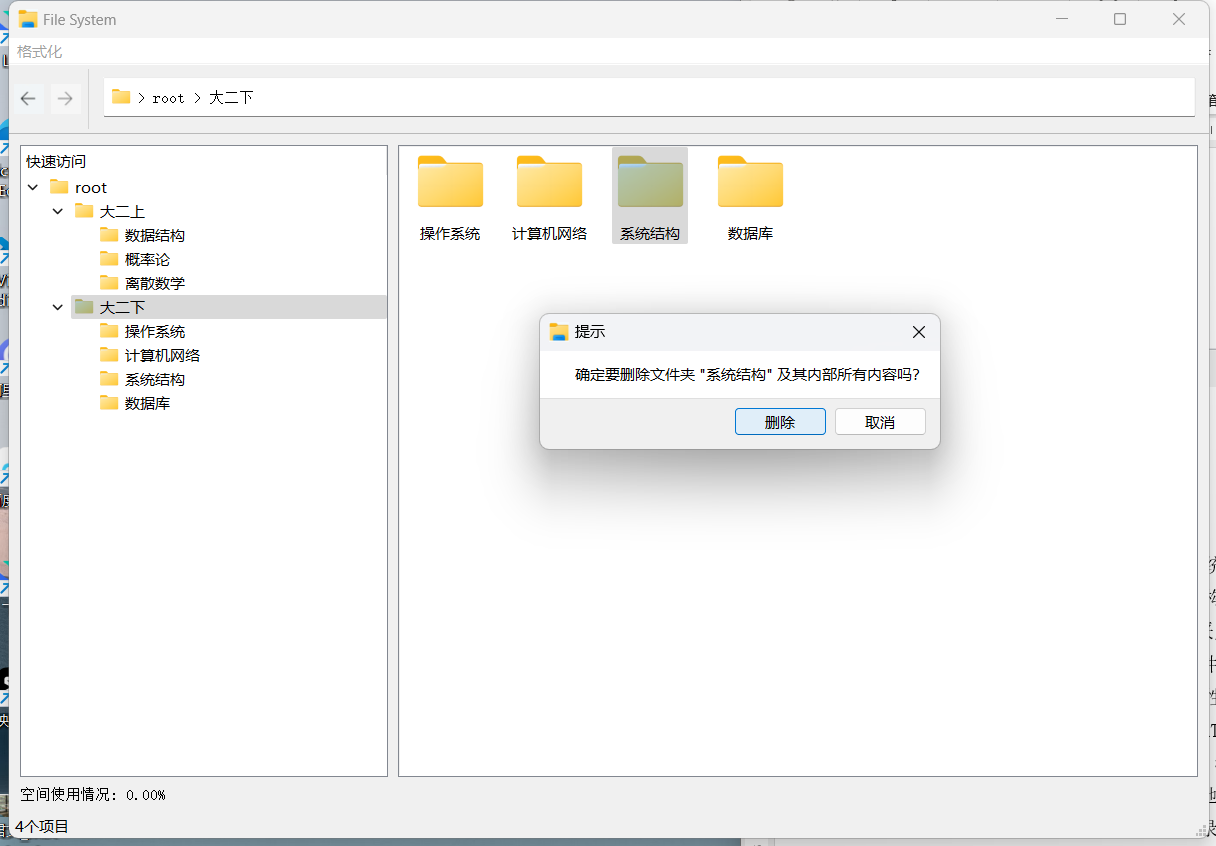


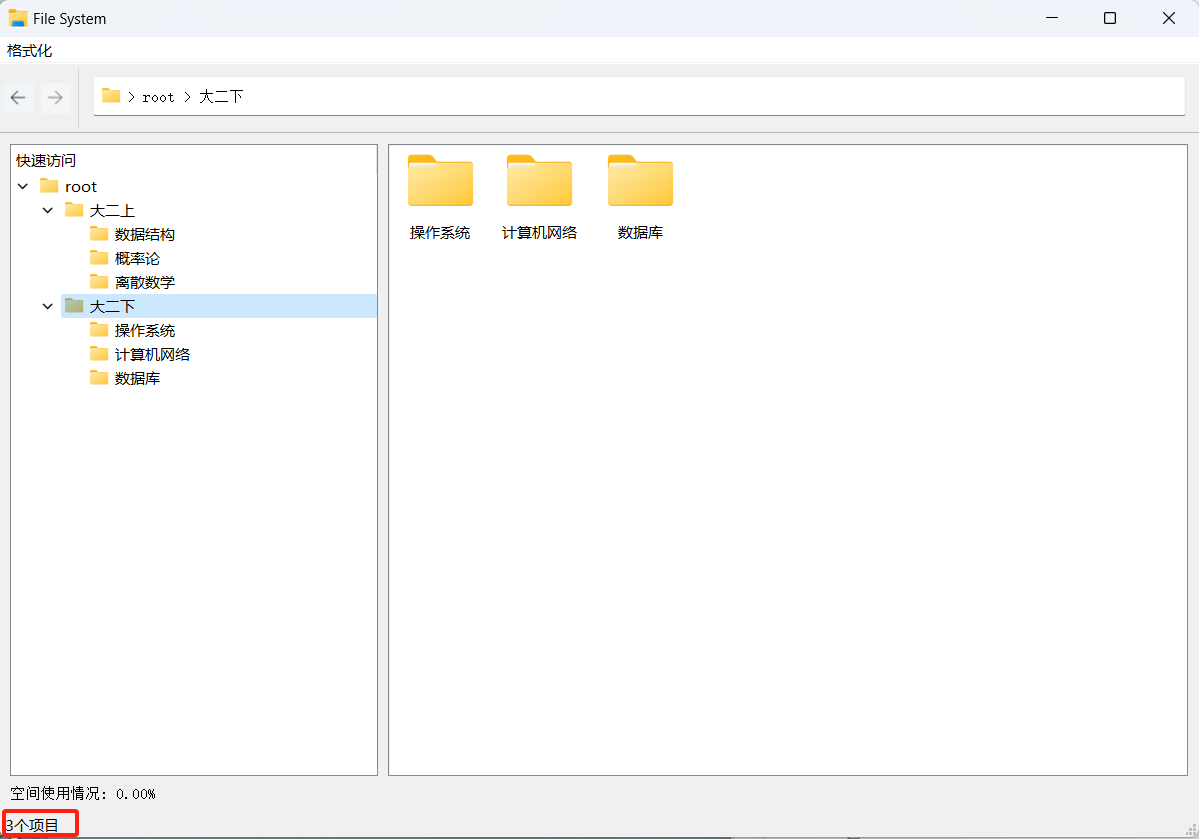
创建时若有重名，则会再文件名后增加数字以区分。



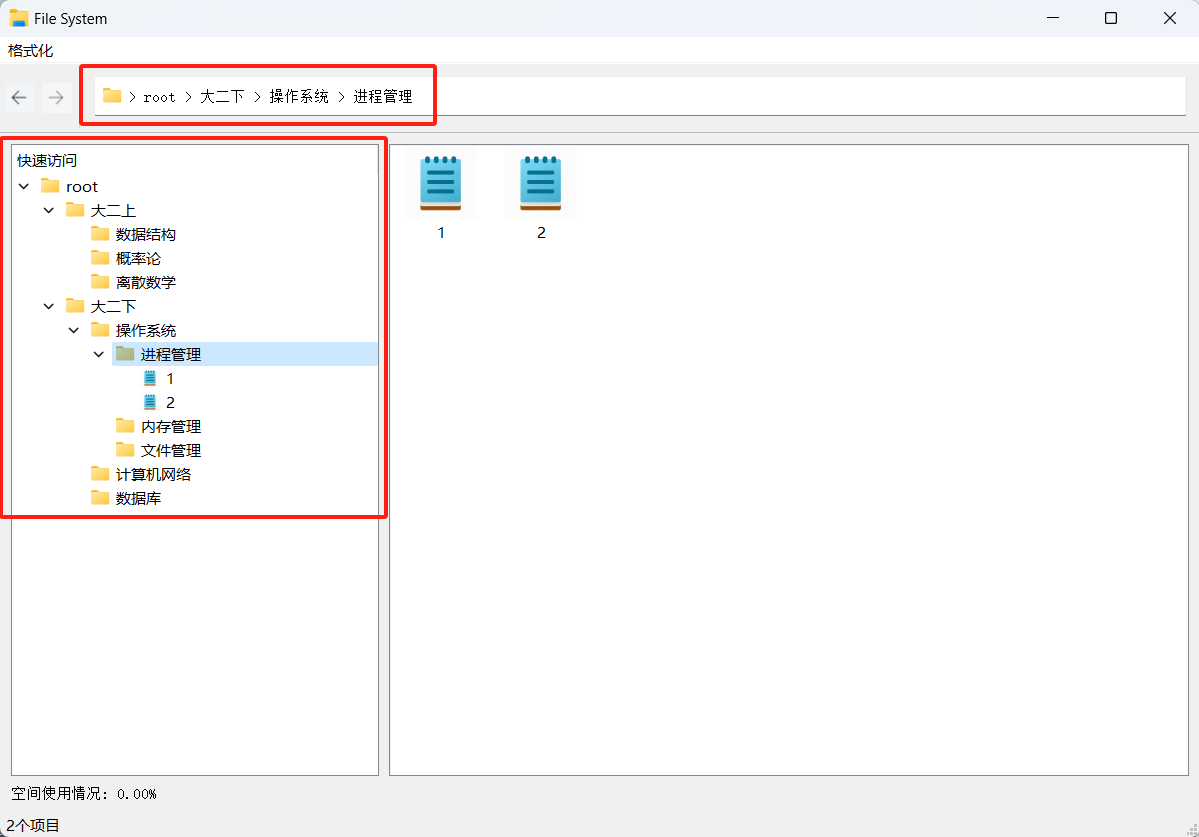
**5.3. 删除子目录**

****



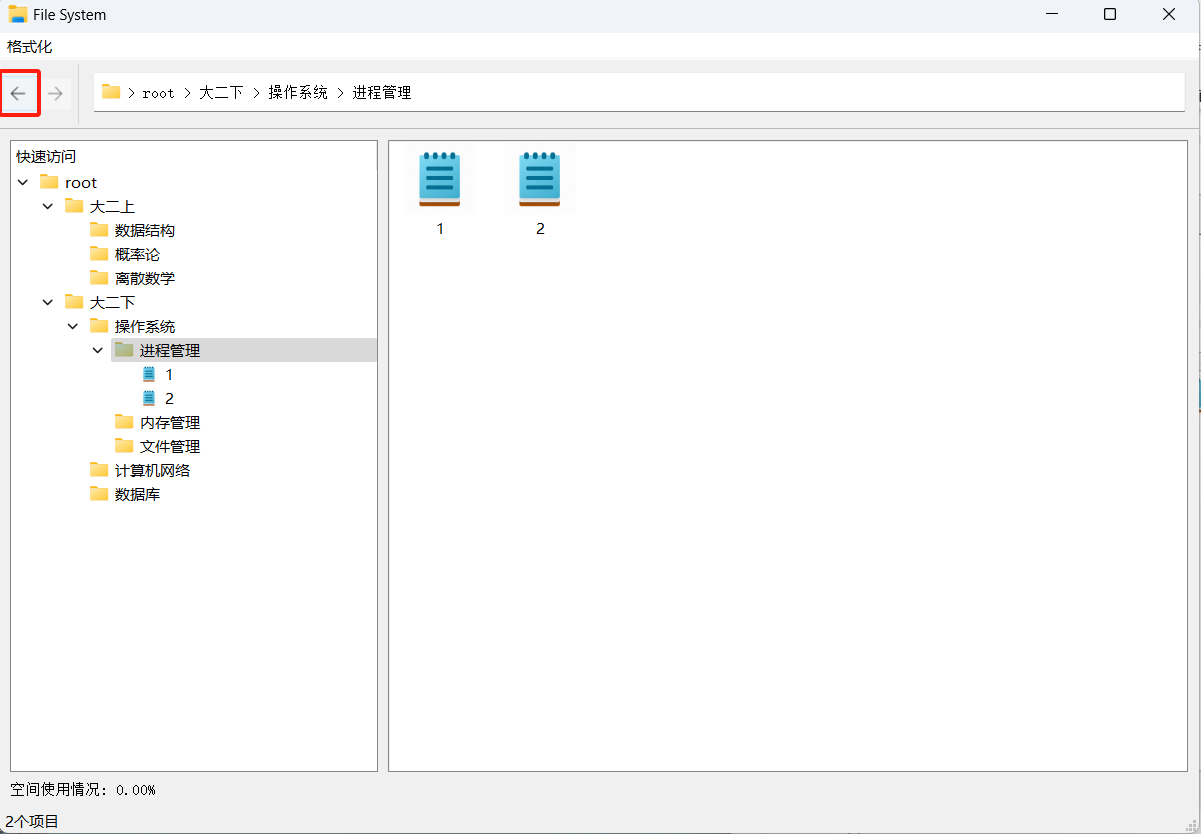


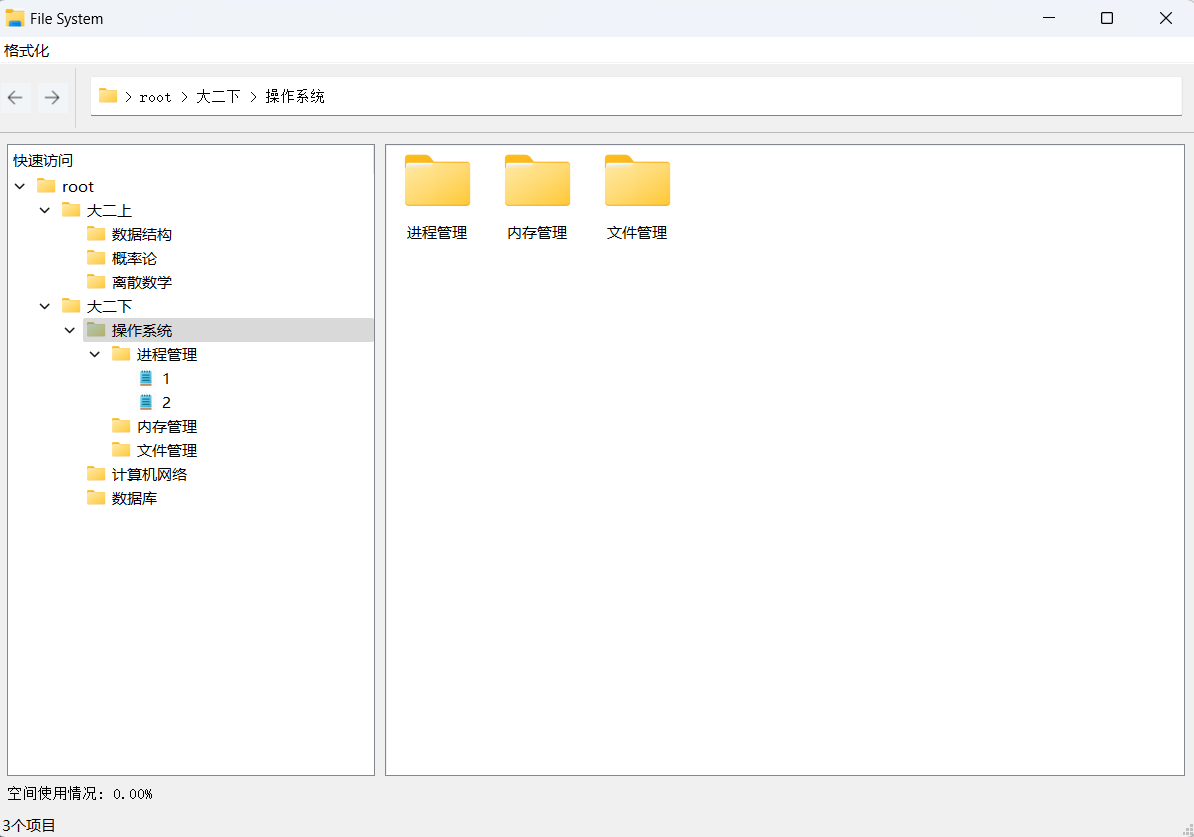
**5.4. 显示目录**



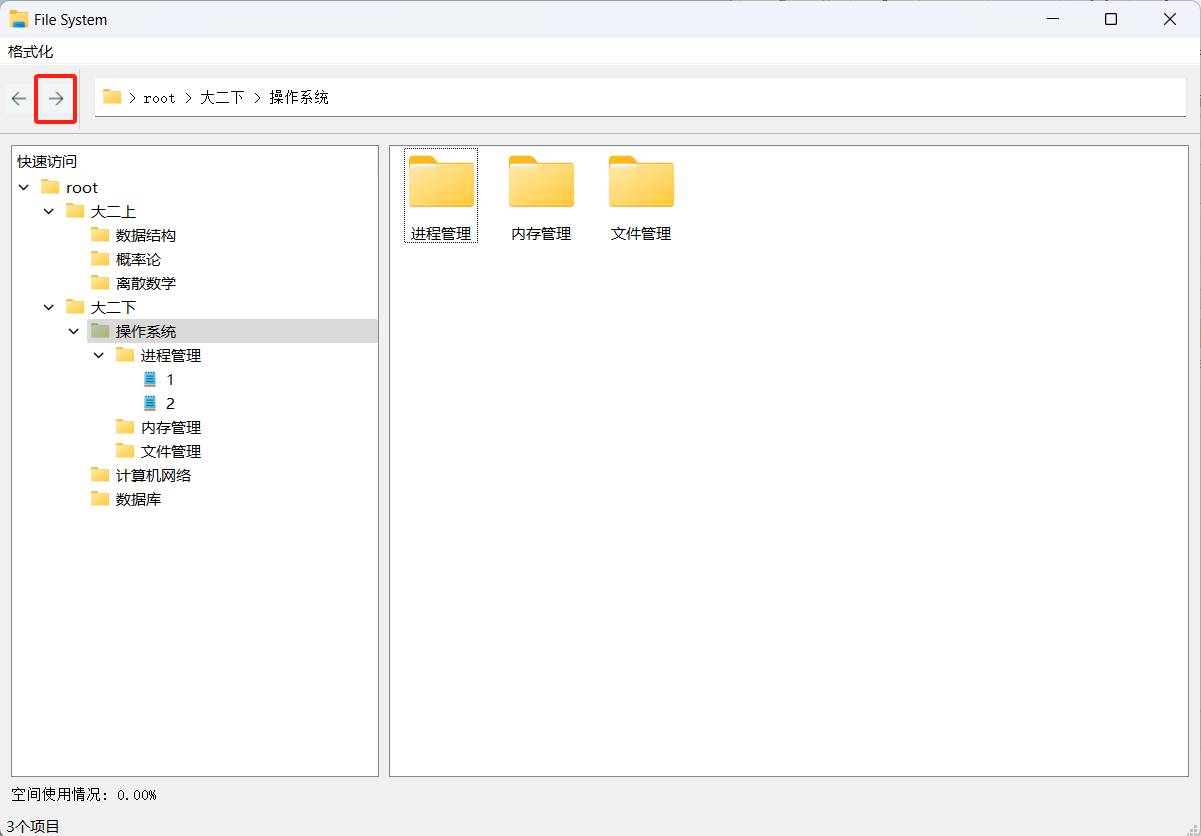
**5.5. 更改当前目录**

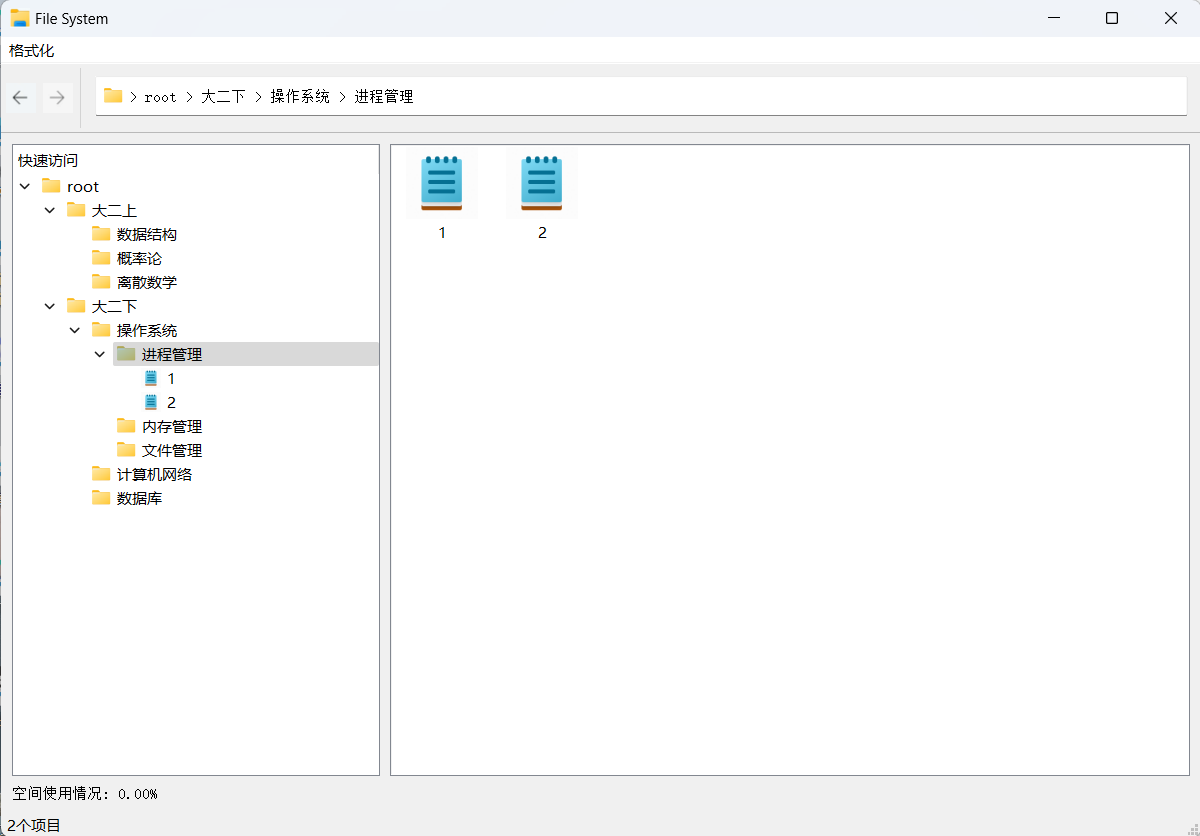
（1）点击“返回”键



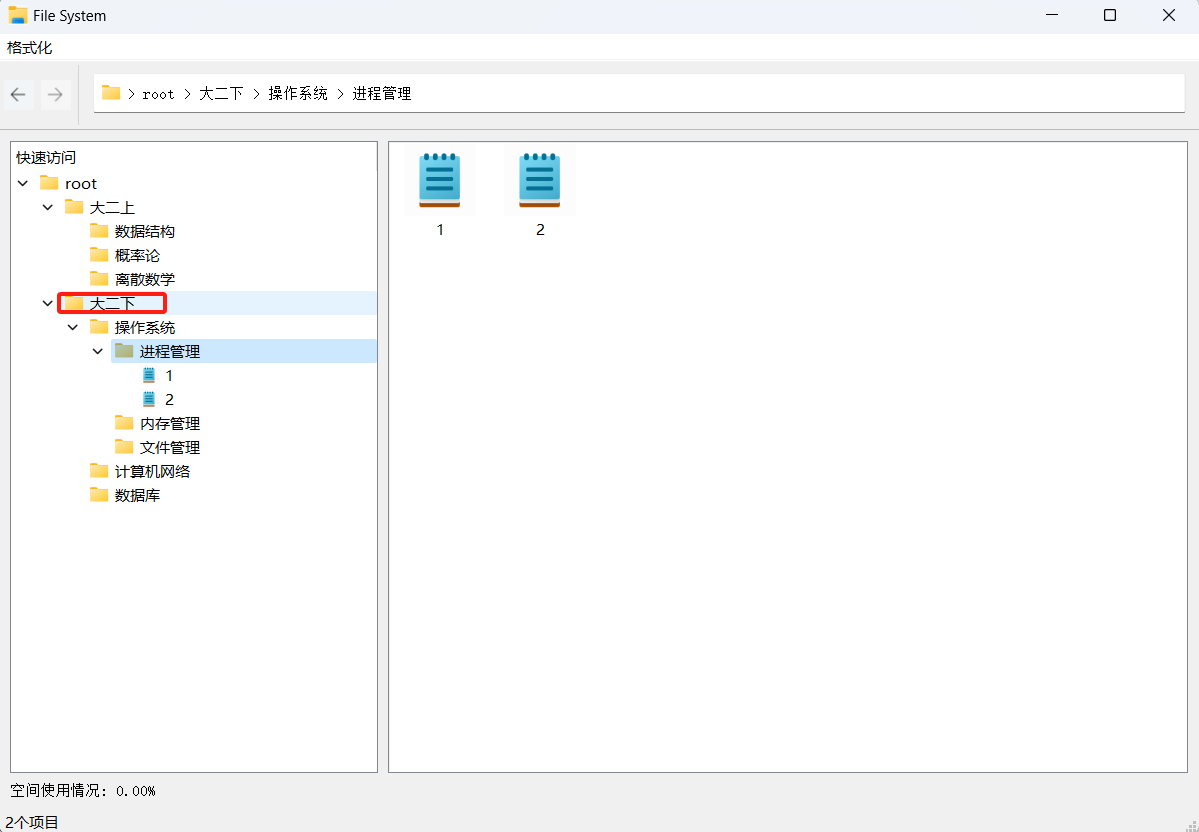


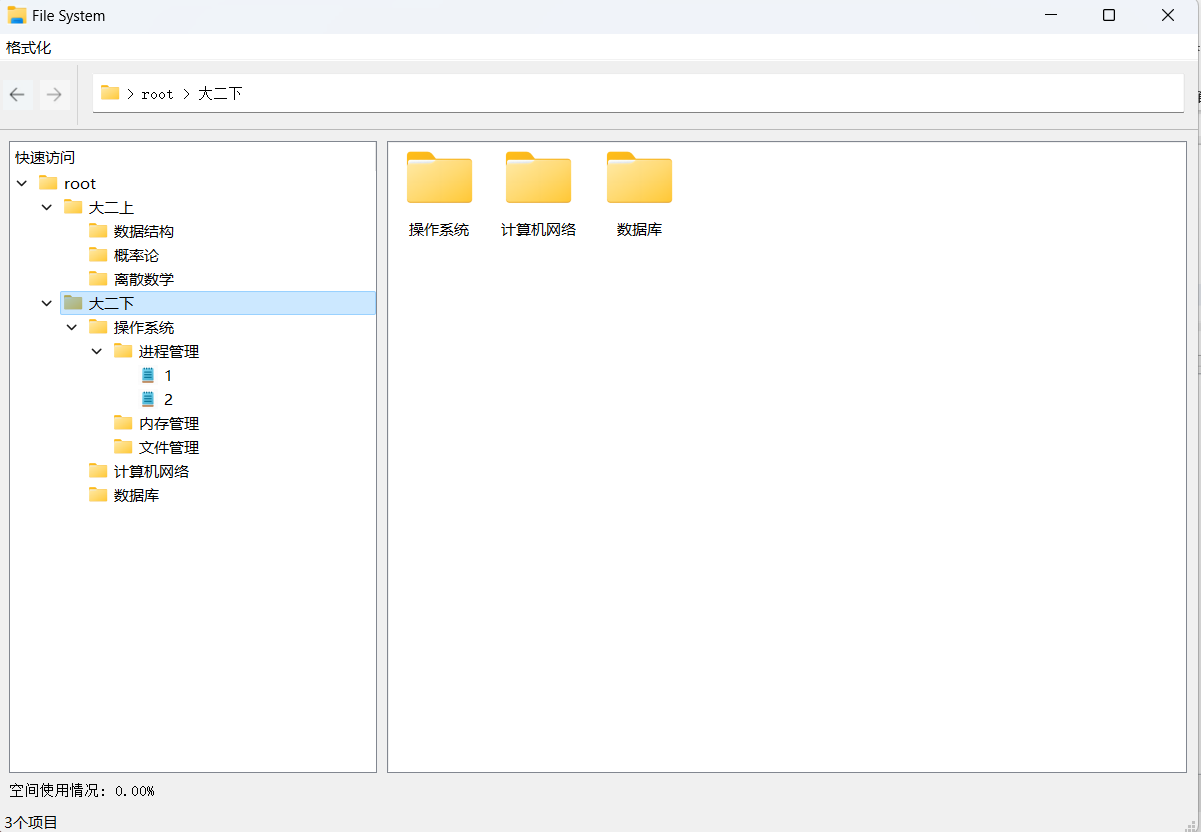
（2）点击“前进”键



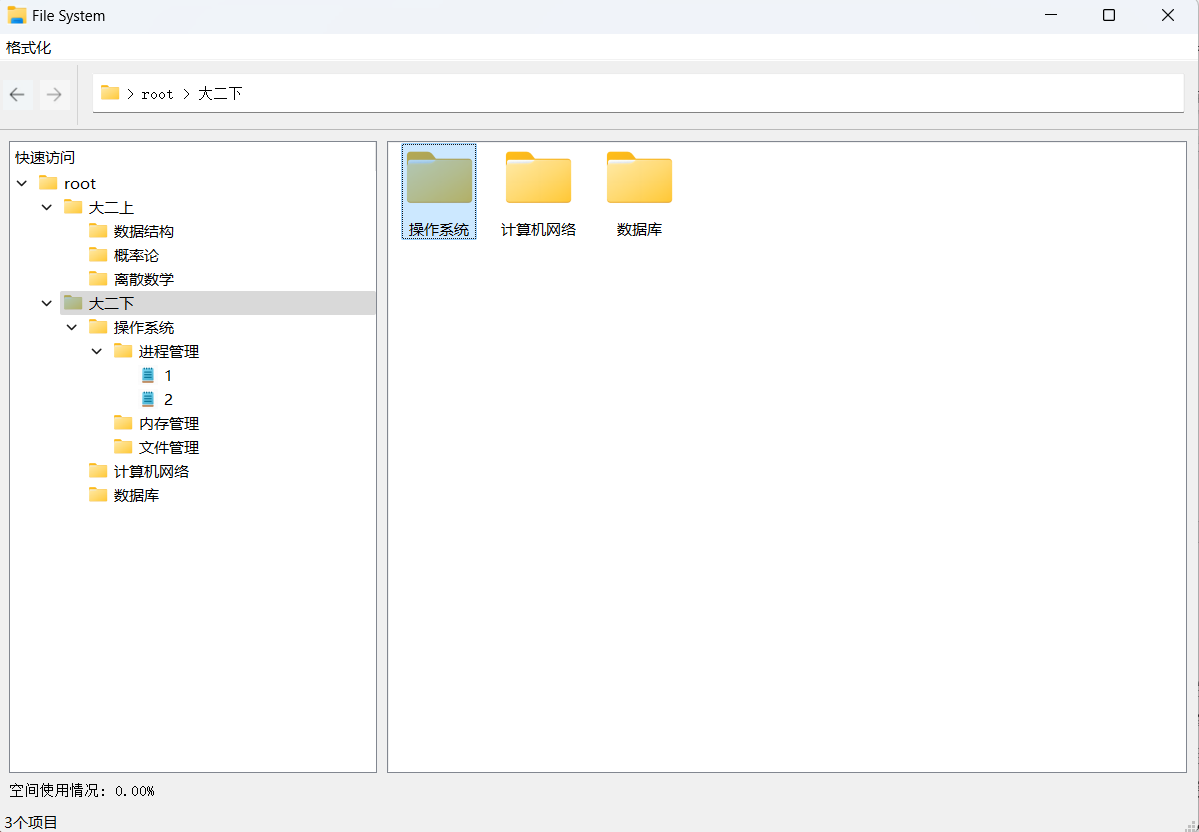


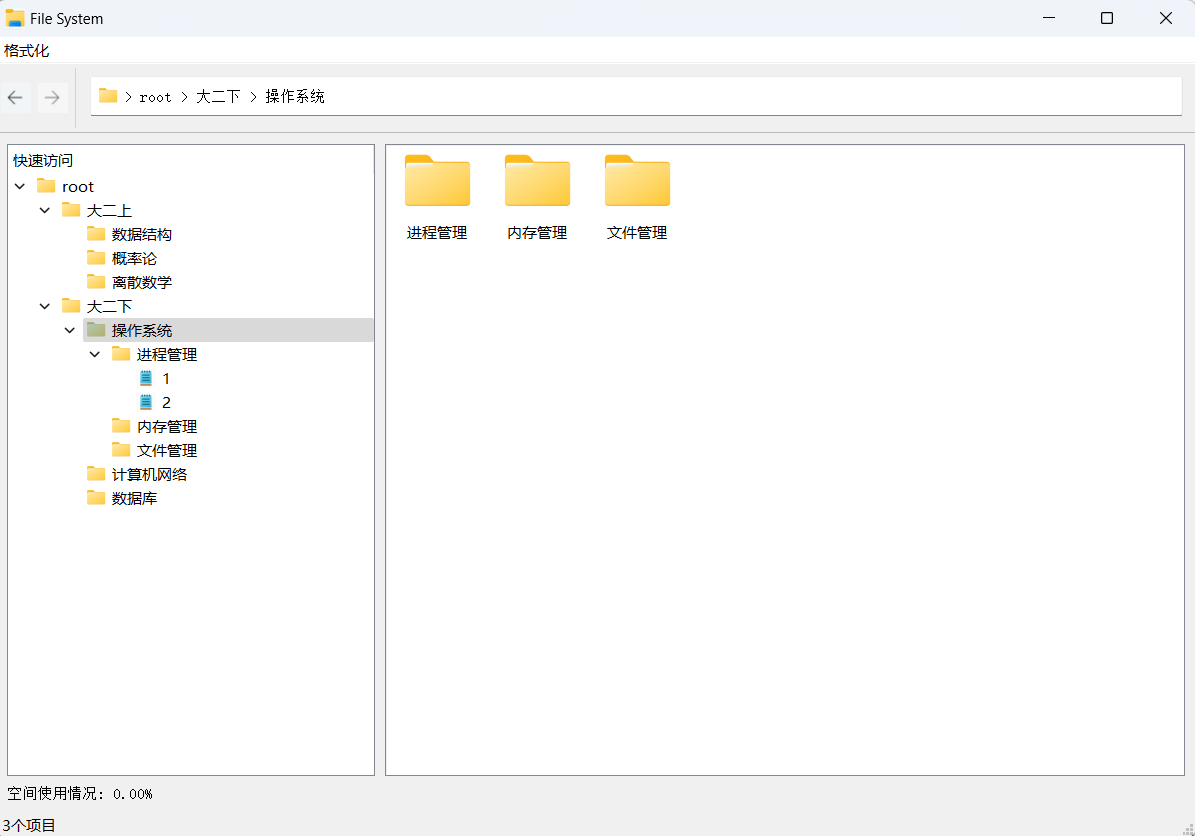
（3）点击“快速访问”栏的目录



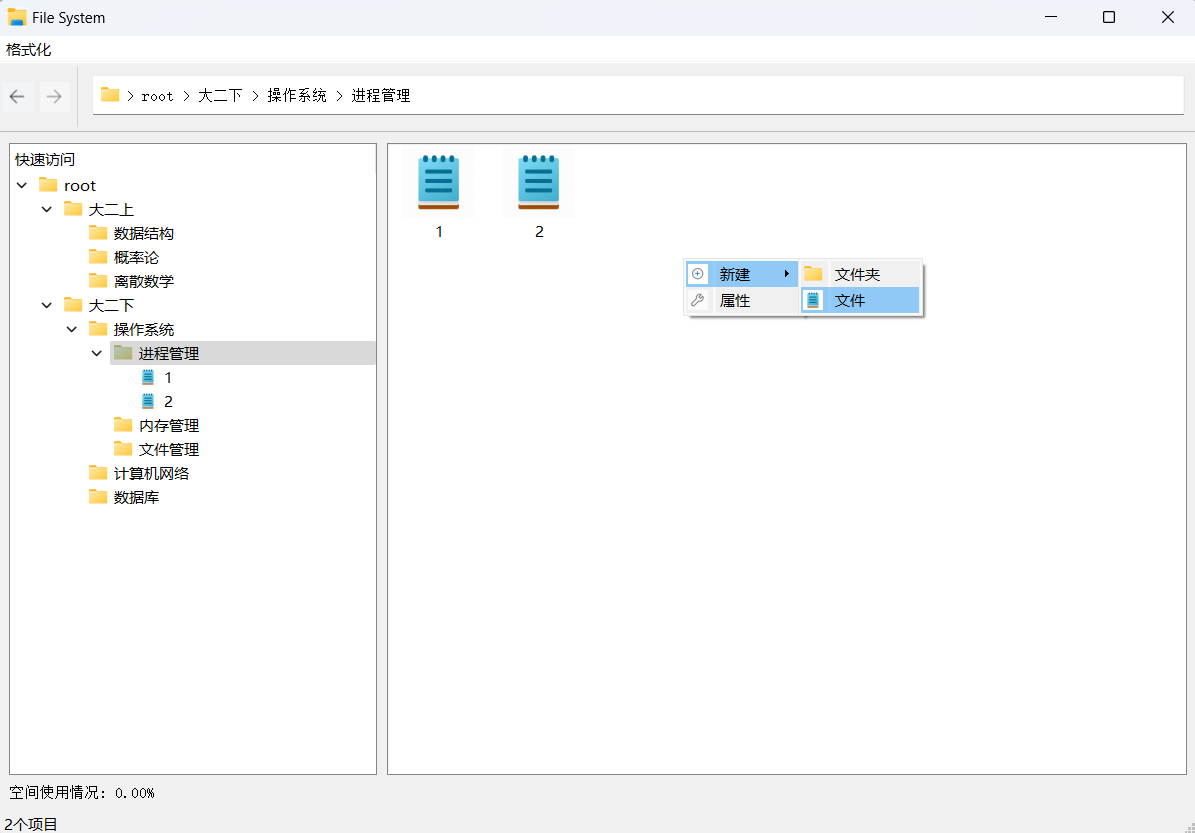


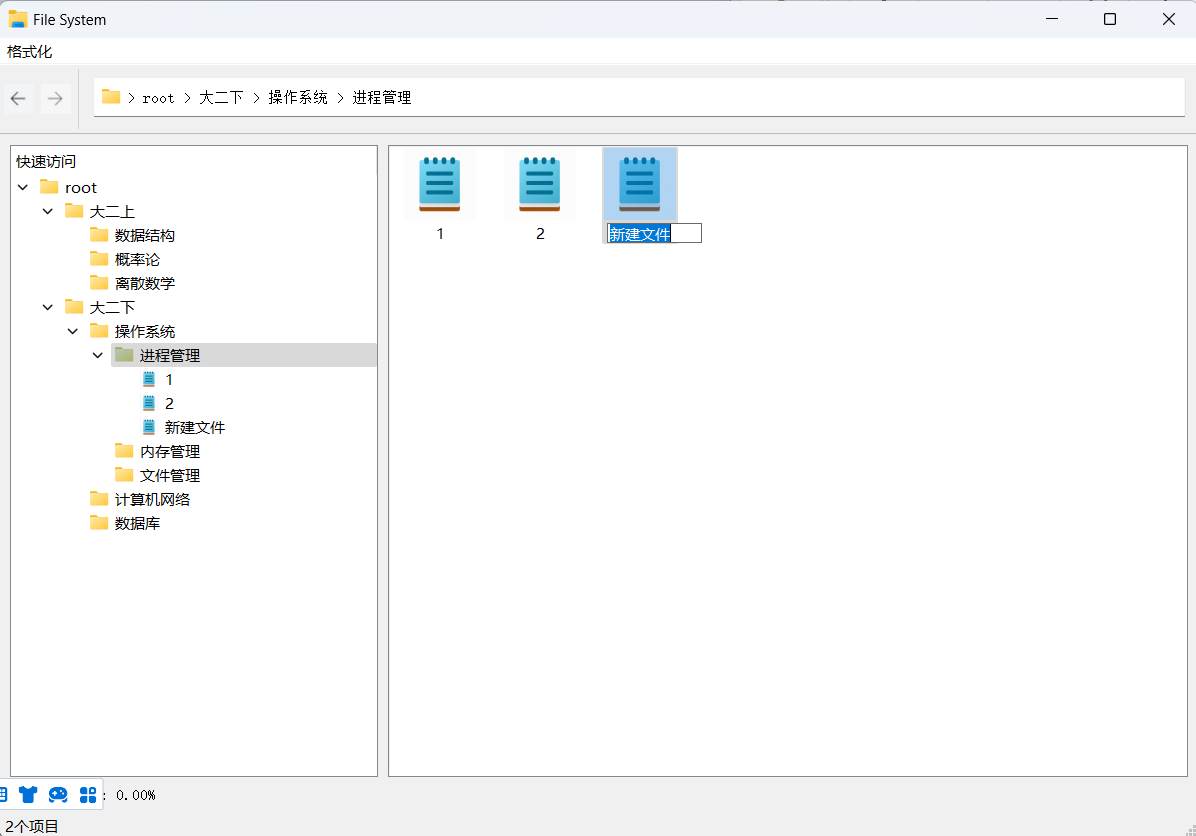
（4）点击右侧部分的文件

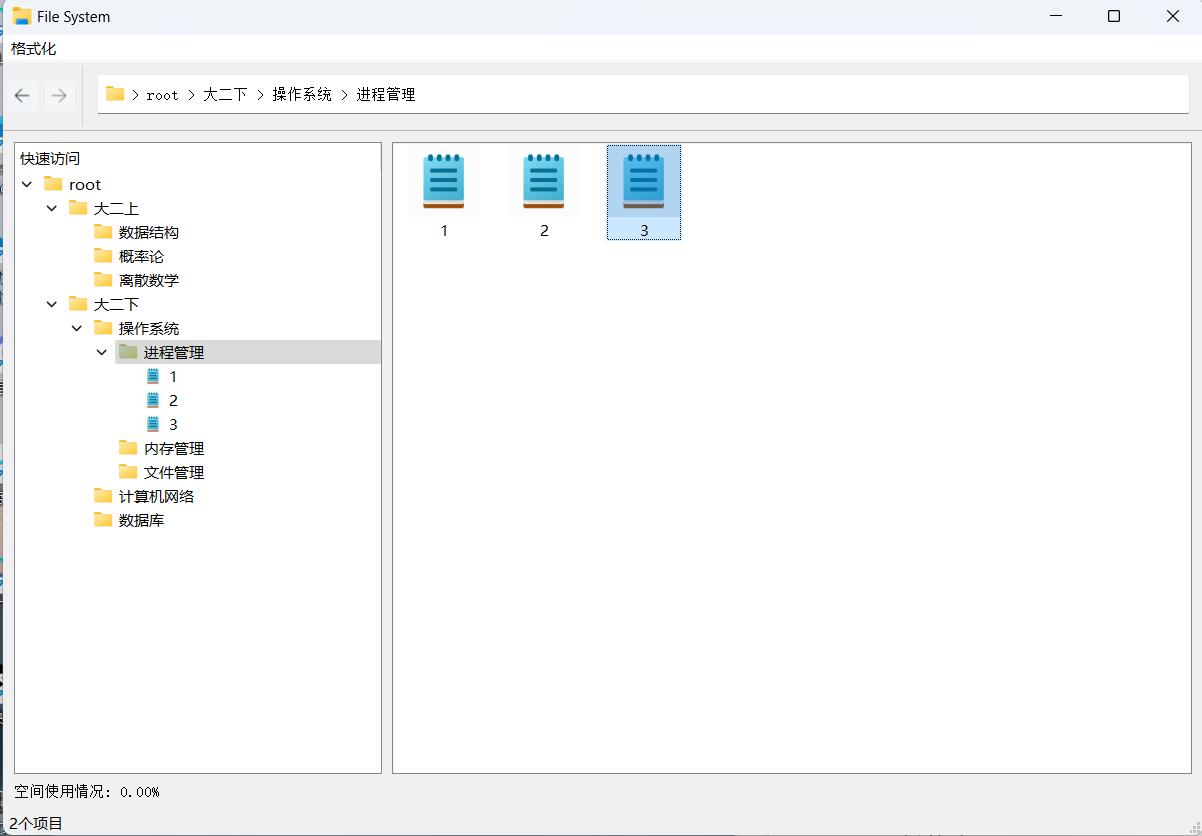




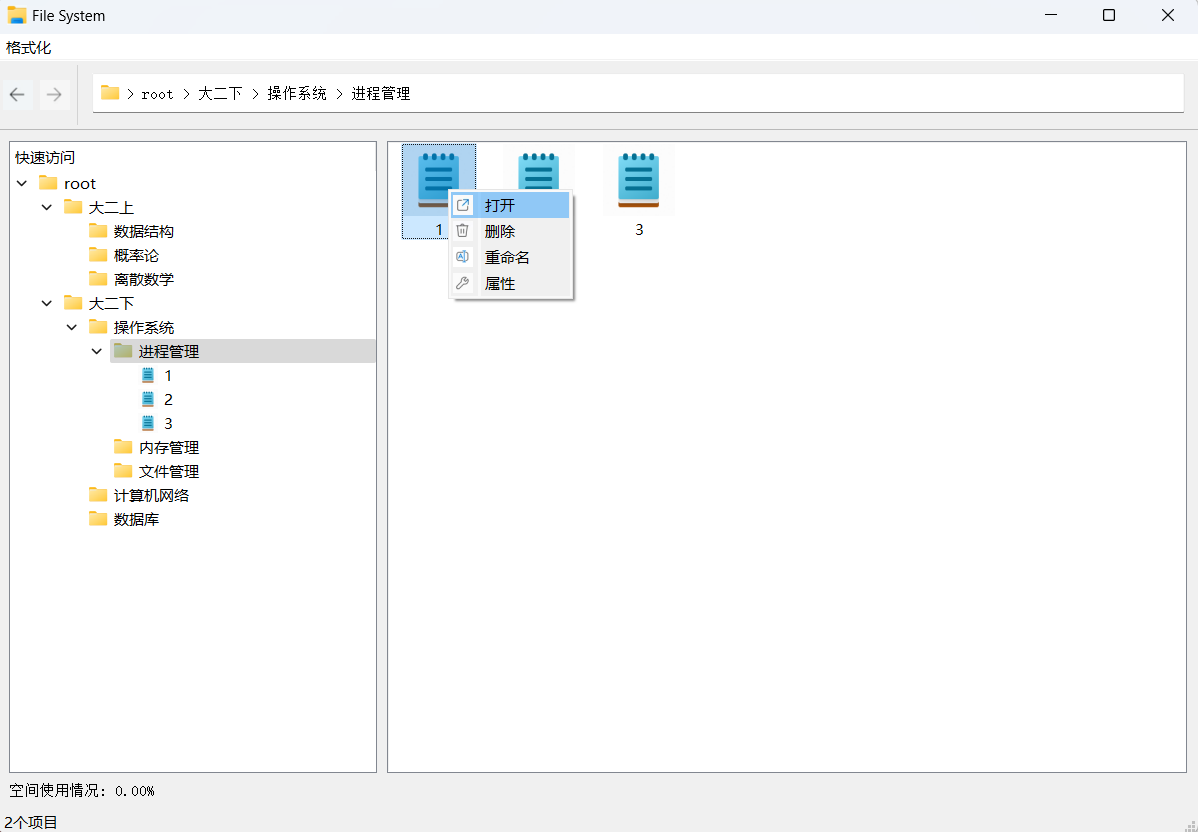
**5.6. 创建文件**



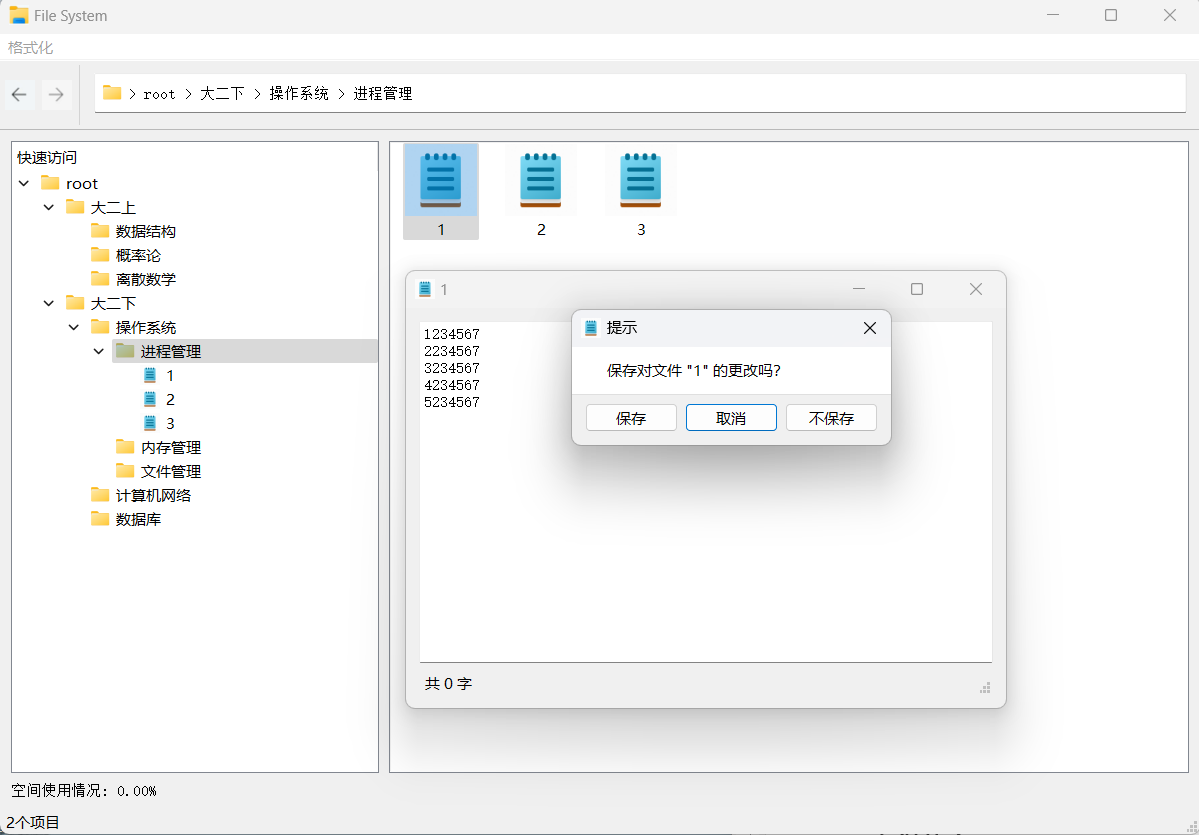




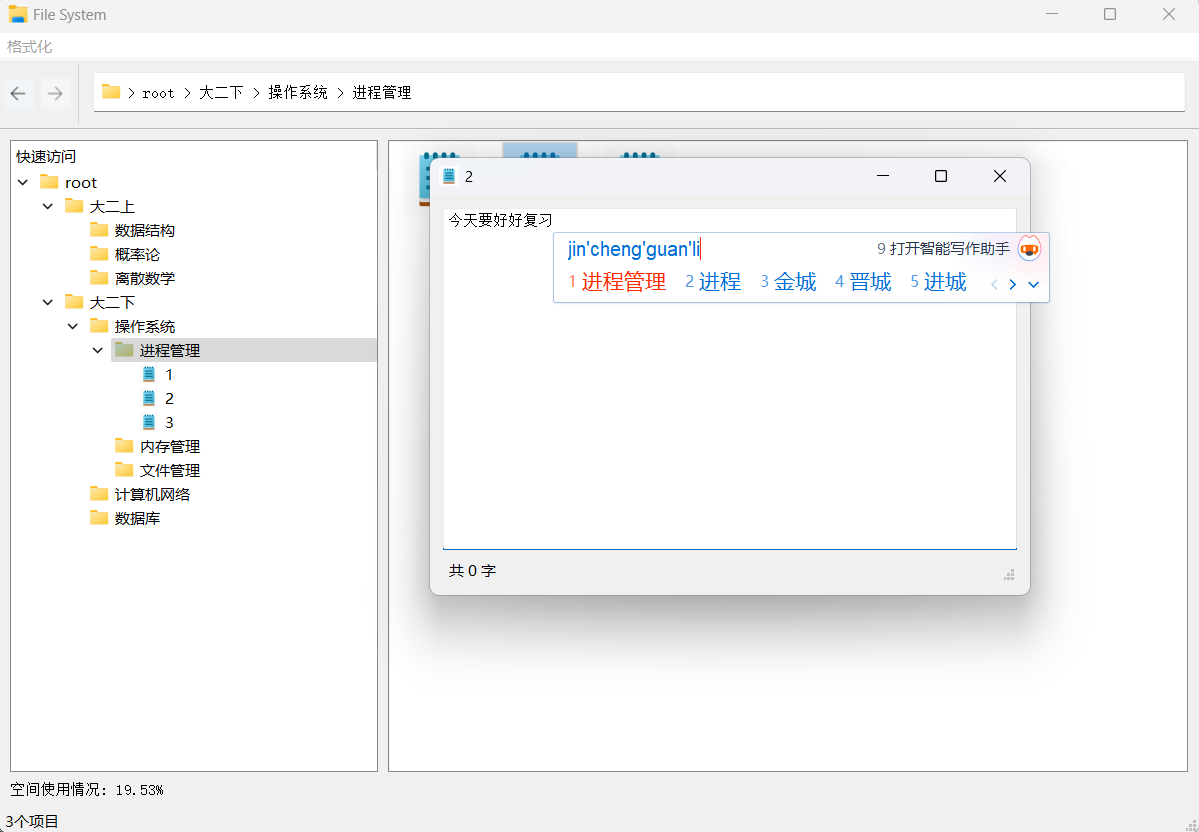
**5.7. 打开文件**



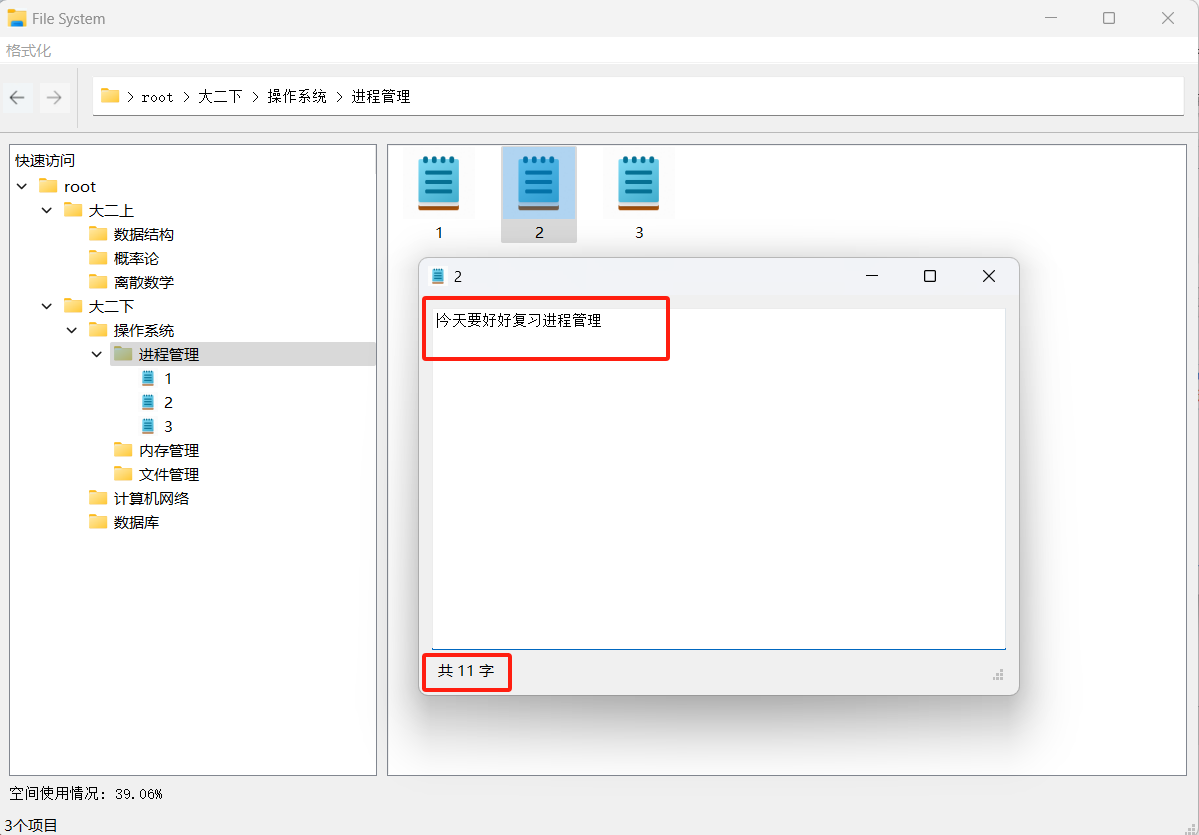
**5.8. 关闭文件**



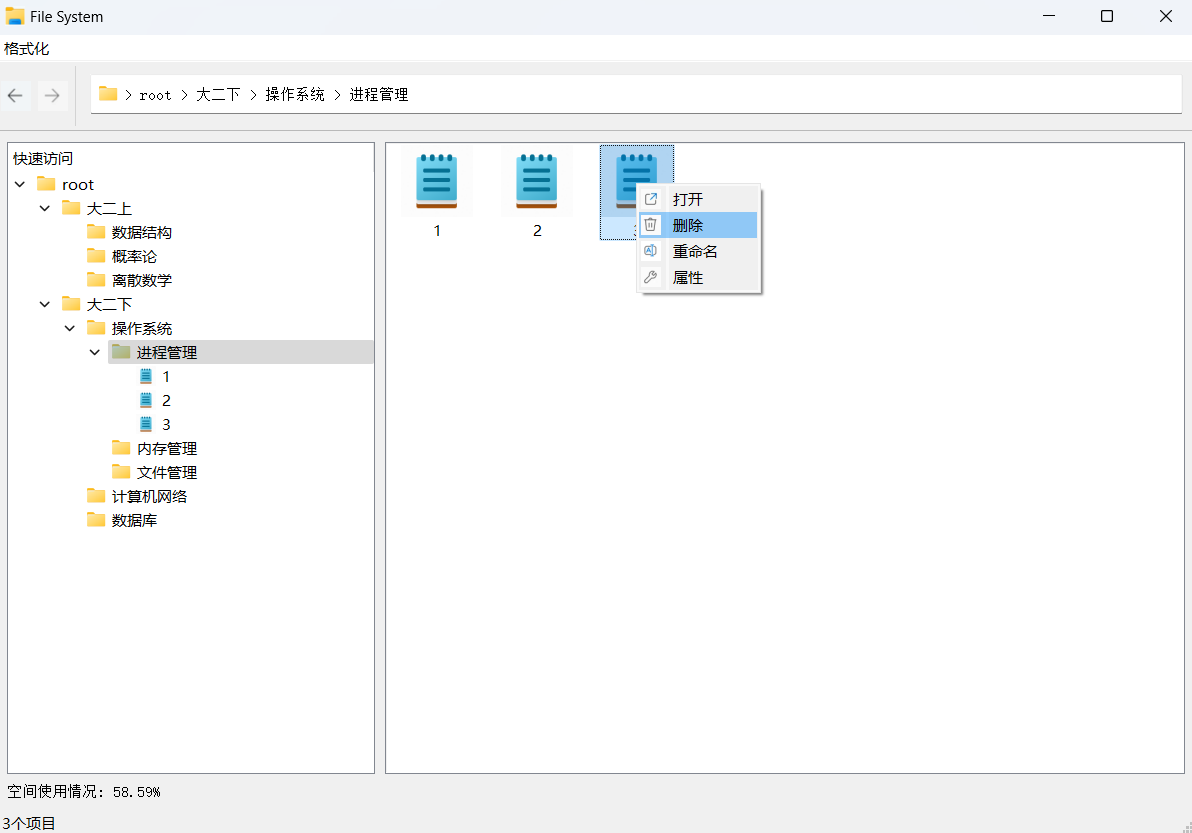
**5.9. 写文件**

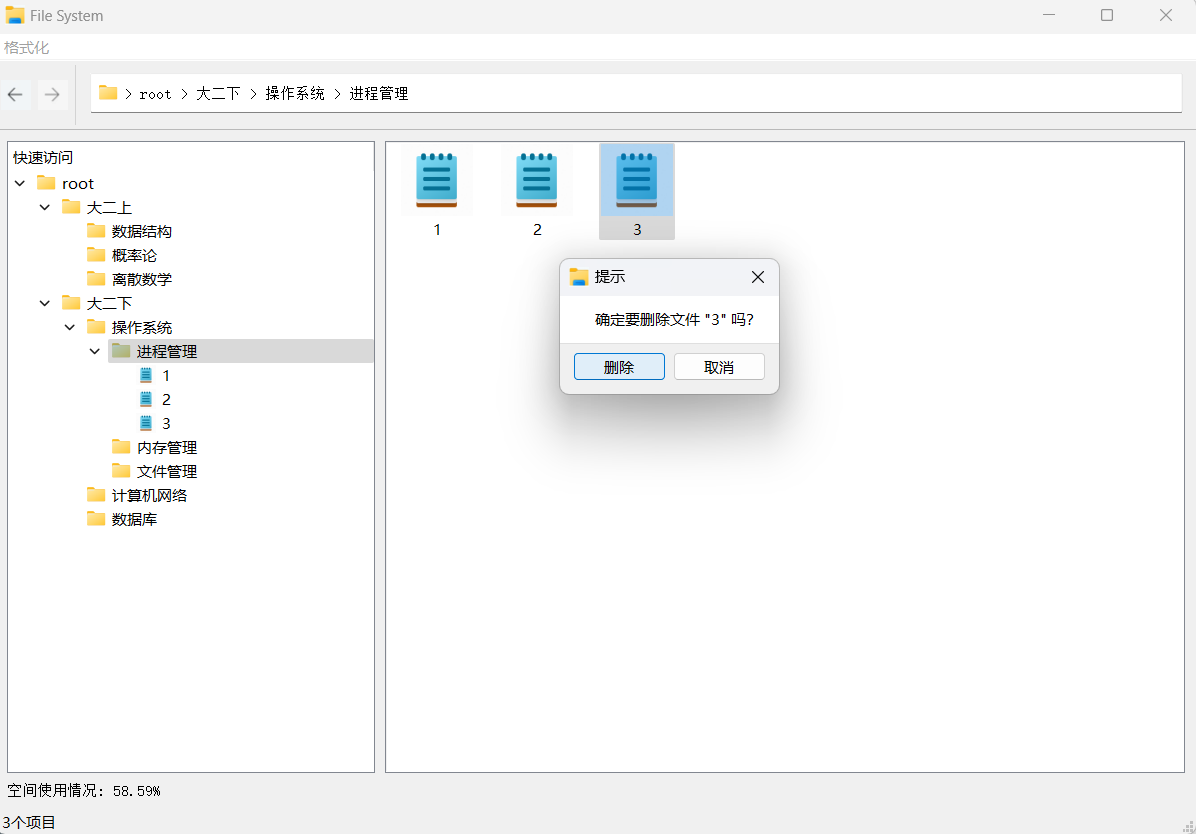


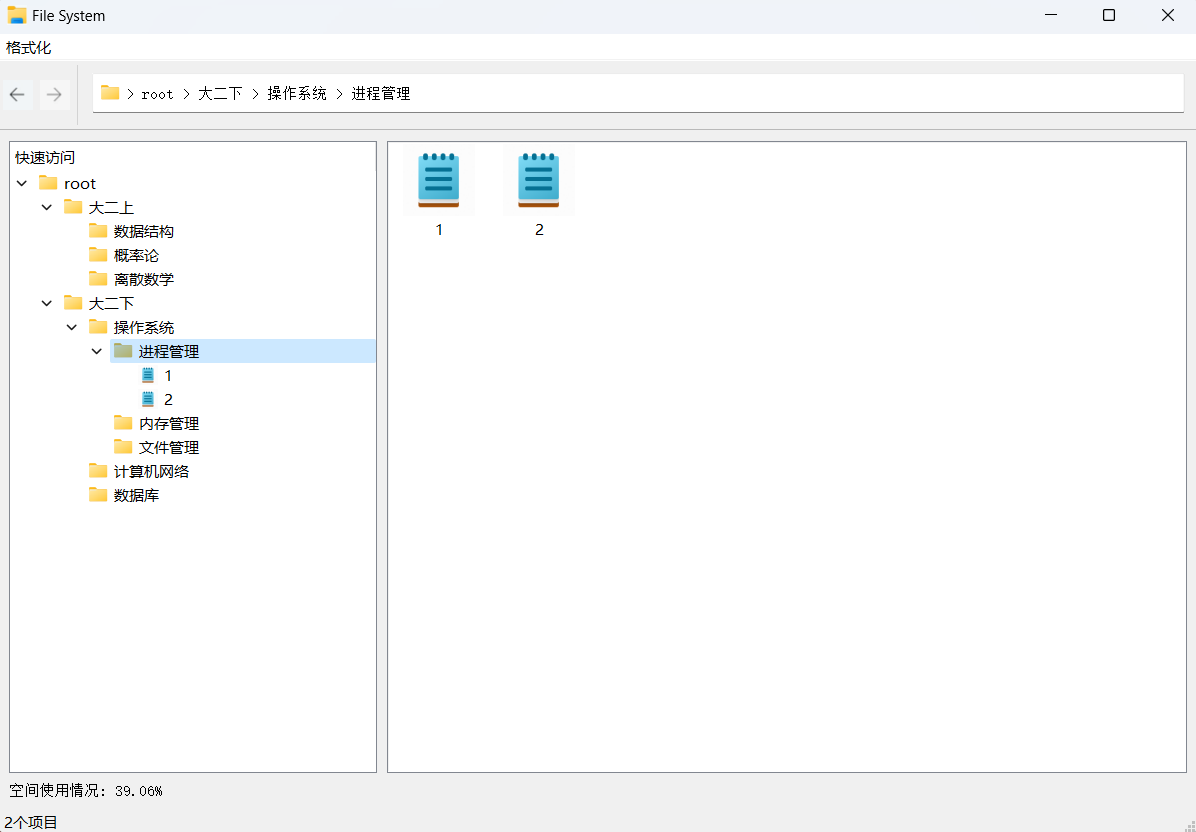
**5.10. 读文件**



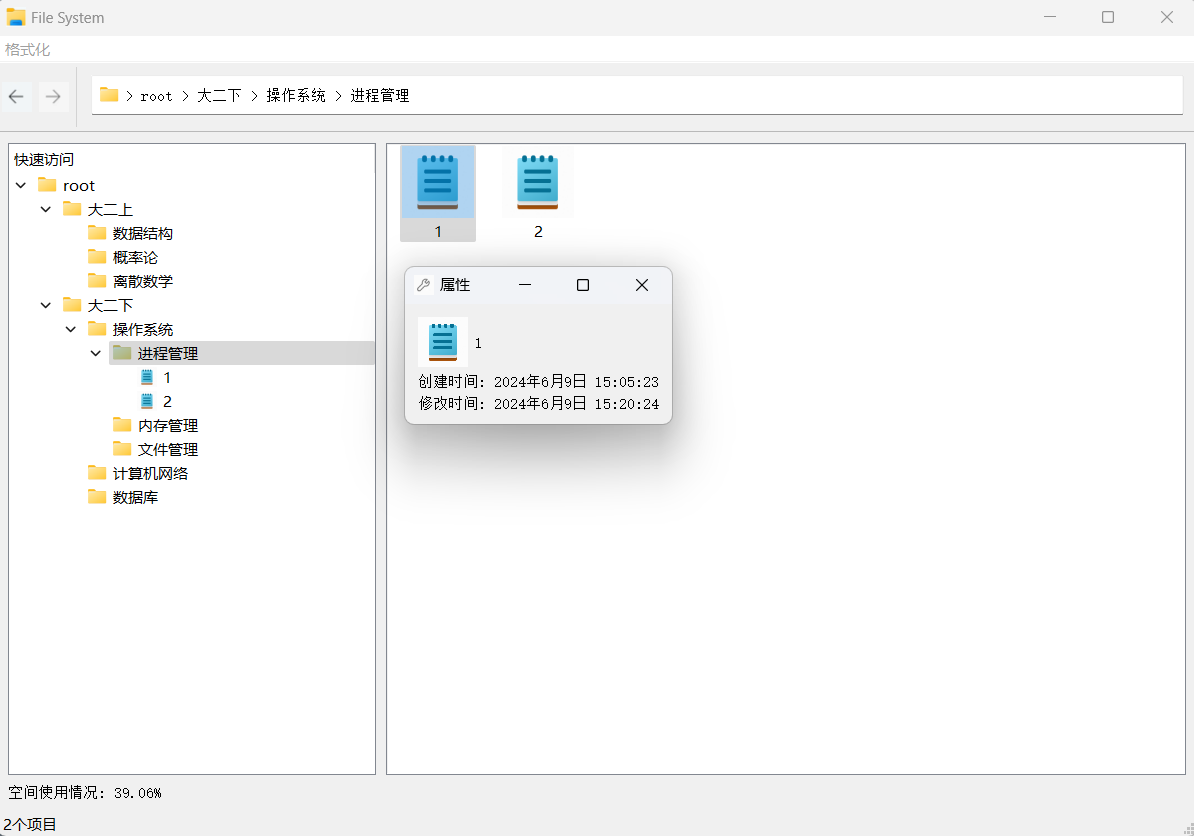
**5.11. 删除文件**

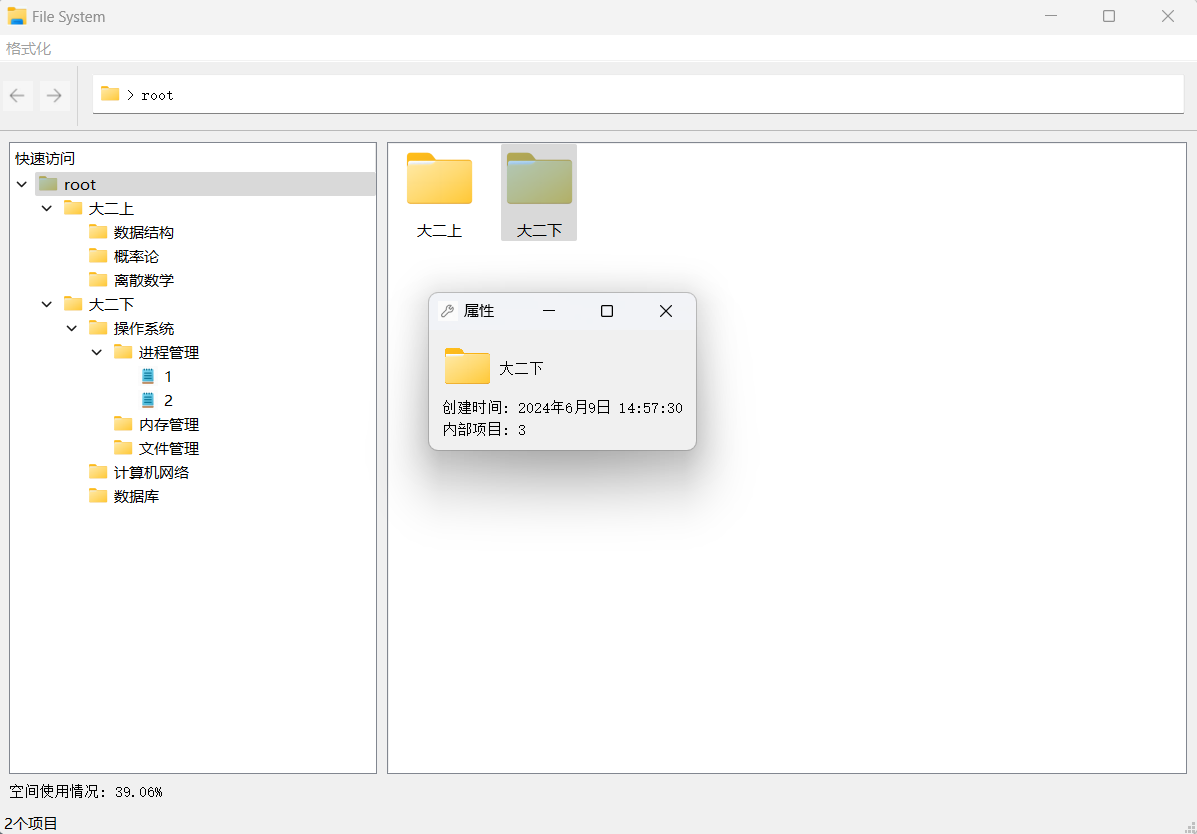




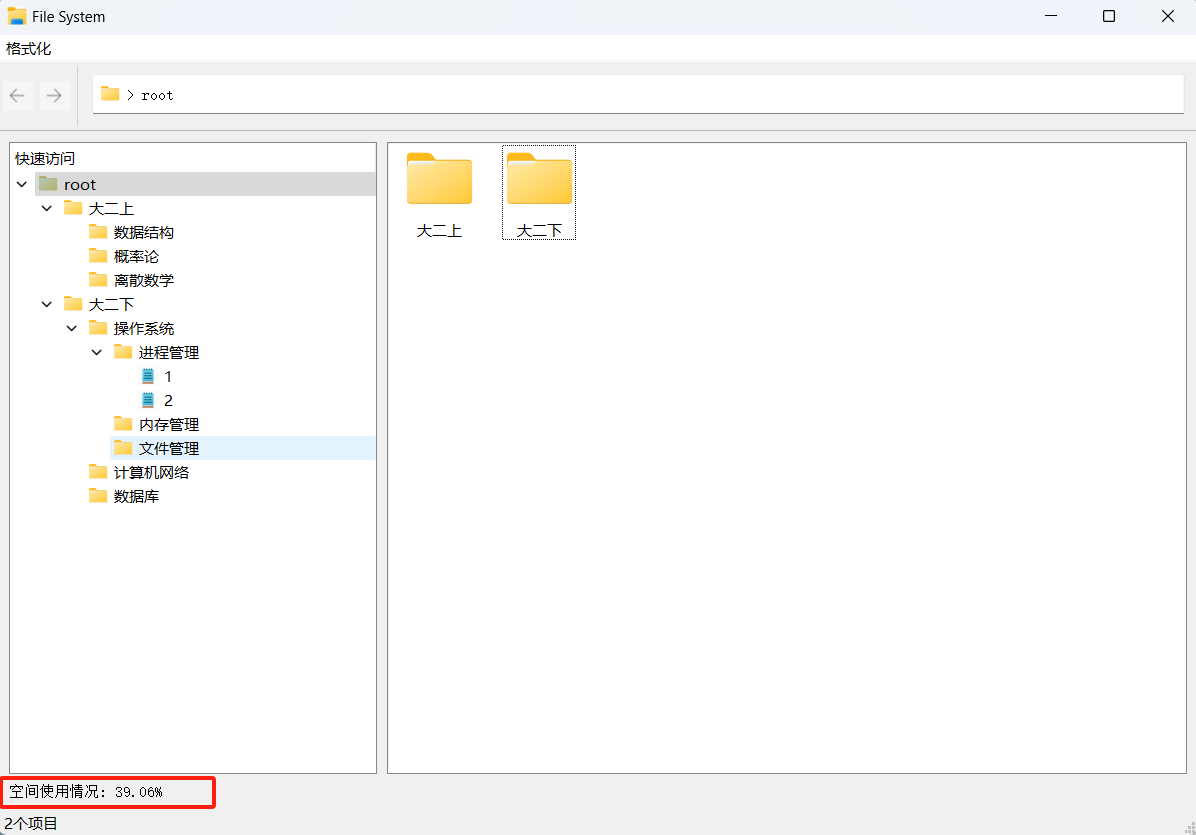


**5.12. 文件/文件夹属性**

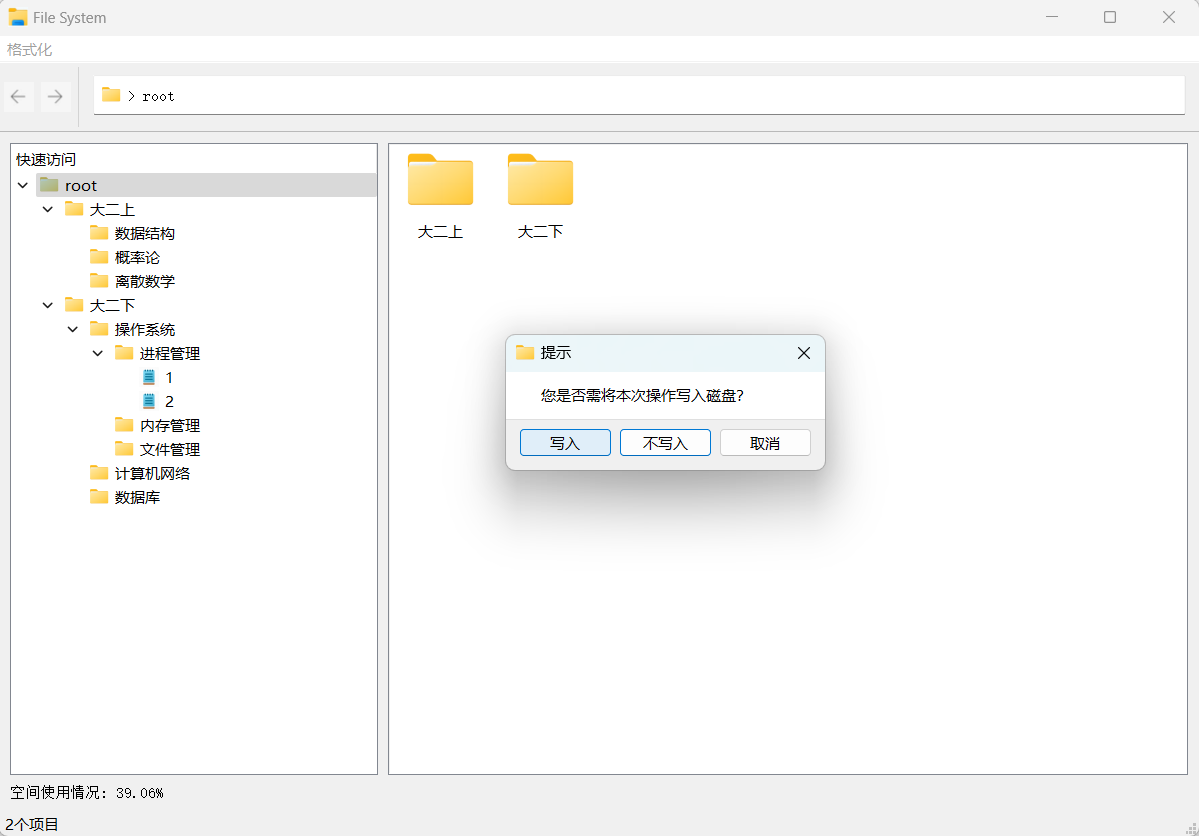




**5.13. 磁盘空间使用情况**



**5.14. 关闭系统可写入/不写入磁盘**



**6. 心得体会**

**6.1. 感想**

开发文件系统让我深入探索了文件系统的内部运作机制，并在实践中加深了对文件存储空间管理、空闲空间管理以及多级目录结构等核心技术的理解。

首先，在文件存储空间管理方面，我采用了链接结构来模拟FAT文件系统的显式链接方式。通过编程实现这一结构，我体会到文件系统如何高效、有序地管理文件存储空间。这种管理方式确保了文件数据的连续性和完整性，同时也为文件访问提供了便捷的途径。其次，在空闲空间管理方面，我采用的是查找FAT表为-2的方式，后续可以尝试使用位示图进行对空闲空间的管理。在文件目录管理方面，我采用了多级目录结构来组织文件系统中的文件和目录。目录项目中包含了文件名、物理地址、长度等关键信息，这些信息为文件的访问和管理提供了重要的支持。通过实现多级目录结构，我不仅学会了如何组织和管理文件系统中的文件和目录，也提高了系统的可扩展性和易用性。此外，我也深刻认识到了良好的编程习惯的重要性。在编写代码的过程中，我始终遵循着代码整洁、规范的原则，这不仅提高了代码的可读性和可维护性，也使我能够更快地完成项目。

**6.2. 改进方向**

项目目前仅实现了一些文件系统的最基本的功能，后续可以进一步改进，学习Windows系统的文件资源管理器，进行更进一步的复现。例如，可以增强文件操作功能，增加批量删除、批量复制和批量移动文件和文件夹的功能。可以进一步改进性能，对于频繁访问的文件和目录，采用缓存机制减少磁盘I/O。可以增加文件搜索功能，支持文件类型、修改时间等条件进行搜索。对于文件的展现形式，可以支持按文件大小、修改时间排列。代码方面，可以对现有代码进行重构，减少冗余代码，提升可读性和可维护性。