|  |
| --- |
| Nullptr小组 |
| Minifs系统概要设计文档 |
|  |

|  |
| --- |
| Nullptr小组成员：刘硕，何祎君， 张嘉熙，张歆，彭青峰  2018-8-30 |

目录

[前言 2](#_Toc523424729)

[一、 整体设计思路 2](#_Toc523424730)

[1.1计划实现的功能 2](#_Toc523424731)

[1.2数据结构设计 3](#_Toc523424732)

[1.2.1数据内容 3](#_Toc523424733)

[1.2.2索引 3](#_Toc523424734)

[1.2.3目录 4](#_Toc523424735)

[1.3系统流程描述 5](#_Toc523424736)

[1.3.1创建空间 5](#_Toc523424737)

[1.3.2开启空间 5](#_Toc523424738)

[1.3.3读入文件 5](#_Toc523424739)

[1.3.4修改文件和文件夹 5](#_Toc523424740)

[1.3.5关闭空间 6](#_Toc523424741)

[二、 UML建模 7](#_Toc523424742)

[2.1类设计 7](#_Toc523424743)

[2.1.1类图 7](#_Toc523424744)

[2.1.2类图描述 8](#_Toc523424745)

[2.2用例设计 9](#_Toc523424746)

[2.2.1用例图 9](#_Toc523424747)

# 前言

我们组在需求分析过程中，发现项目的最低要求中整体目录结构是单层的，没有要求对文件夹的创建和管理，而现实中windows，linux中的文件系统一定是多层的，这样单层的系统在实际使用时并不实用，所以我们决定将整体目录结构变为多层的结构，并且加入对文件夹创建，管理的一系列功能，以下所有的设计都建立在多层目录结构的基础上。

我们将从整体设计思路和uml建模两大方面进行思路描述，其中整体设计思路分为计划实现的功能，数据结构设计，以及系统流程描述三部分；uml建模分为类图及类的设计以及用例图。

# 整体设计思路

## 1.1计划实现的功能

**操作空间：**

create：建立一个新mini-FS空间

mount：命令安装要操作的空间，在硬盘中开辟一块外存空间进行操作利用

fmt：初始化当前的空间

close：退出mini-FS

help：显示帮助信息

**操作目录及文件：**

dr：显示空间的文件目录

mkdir：创建子目录

cp：复制文件或者子目录

dl：删除空间中的文件或者子目录

cd：打开目标目录

att：显示文件或者目录属性

**map <文件名>**：显示指定文件在空间所占用的全部块的“块号”。  
**opt**：优化整个空间，将每个文件尽可能连续存放（使用连续的块号存储文件内容）

Import 导入硬盘上的文件到本系统中。(与cp –i作用相同)

Export 将本系统中的文件导出到外部硬盘中。(与cp –e 作用相同)

tree：以树型方式显示文件目录和文件（模仿DOS的 tree 命令）。

move <指定空间中的文件名> <文件要移至空间的路径>：将指定路径下的文件移动到新的路径中。

## 1.2数据结构设计

我们组将整个minifs的空间分成**三部分：索引，目录，数据内容**。下面会详细介绍三部分的具体设计以及数据在外存中的组织，并介绍系统执行的流程

### 1.2.1数据内容

数据内容块即为存放文件的部分。

在外存中，我们决定以**离散**的方式存放文件，其中文件夹不属于文件。为实现离散地存放文件，我们将1G空间中分割为索引、目录和文件三部分，其中索引是一个固定大小的数组，占用1MB空间，放在整个空间的最前面；然后再存放目录信息，占用8MB的空间；后续部分外存进行分块，每块大小暂定为4KB，每个文件以链式的方式分散地存储在多个块中，具体存放方式后续会讲到。

### 1.2.2索引

索引是文件的索引编号与文件在外存中存储位置（即分块编号）的一张对应表。

索引的本身结构是一个静态链表，存储上是使用的int型的数组，大小为1024\*1024B，数组下标与数据内容分块中块的编号一一对应，由于文件不可能都小于4KB，所以一个文件往往需要多个块才能存完，这就导致每个文件的索引也理应是一个链式的结构，在minifs每次mount完毕后，将索引整个加载到内存中，直到minifs关闭前索引都在内存中保存，方便调用。

每个文件的第一个索引编号通过该文件对应目录信息获得，而后索引数组中每个数组元素的内容都对应存储当前文件的下一个分块的索引编号（即分块编号），若文件在当前分块已存储完毕，那么该在该处存入-1代表文件结束。例如一个文件在外存中存放在第1,4,7分块中，其中1,4分块都存满4KB，第10分块存了100B，那么索引数组index[1]=4,index[4]=7,index[7]=-1。

由于索引目录还占用了一共9MB的空间，所以实际分块会少于原有的计算，但为了方便，这部分空间不再进行拆分。

使用数组存储索引相比于链表的方式优点在于外存开销小，查询时间快很多，查询算法简单，缺点在于内存开销会相对较大。

### 1.2.3目录

目录是存放文件的属性信息以及记录文件之间的层级结构关系的**文件。**目录在外存中集中存放，与文件内容分开，目录项之间是链式关系。

目录本身在外存的结构也是一个静态链表，实现也是使用的自定义类型的数组，每个数组元素大小大约为32B左右，该数组存放两种类型的节点：一种是文件节点，其中要存储文件的首个索引编号，文件属性信息（如文件名，扩展名，创建时间等），还要保存每个文件或文件夹的parent信息；另一种是文件夹节点，其中保存文件夹的创建时间信息以及它的parent信息。parent信息就是一个int类型的数，表示该文件的父亲信息在目录数组中存放的位置编号，只有根目录上没有parent。

由于目录要经常进行修改，所以最终我们决定将其固定在索引后的8MB的储存空间中，以方便于查找和修改，增加读取速度。

在实际使用时，由于我们的结构时多层的，目录应以一颗多路树的结构加载到内存中，建树的根据是目录在外存中的数组中存放的parent信息。建好树后，大部分节点应该都有指向父亲和儿子的指针（根目录没有父亲指针，叶子节点没有儿子指针），树的所有叶子结点即为文件节点，非叶节点都是文件夹节点。输入指令时输入的都是文件名，通过文件名在目录里进行遍历查找得到文件的首个索引编号，再从外存中真正读取文件到内存。

## 1.3系统流程描述

### 1.3.1创建空间

首次执行时，通过create指令创建minifs空间，在软件根目录下生成一个.fs文件即为minifs空间，其中文件中只带有一条根目录信息，里面存储着系统的创建时间。

### 1.3.2开启空间

创建空间后，需要用mount指令开启空间，开启空间时会将索引与目录都先加载到内存中，具体文件不加载。其中目录会根据外存中的目录信息在内存中通过算法生成一棵目录的树结构，**目录文件和索引会常驻在内存中，每次对文件或文件夹操作时方便搜索和遍历，直到关闭minifs空间时将两者从内存中删除。**

### 1.3.3读入文件

用户输入指令后，会进行指令解析，并传入文件操作类中，若指令中需要查看或修改某文件，则会调用类中与文件操作有关的函数，在内存的目录树中根据文件名和文件路径搜索文件节点，搜索到该文件的首个索引编号，然后从索引静态链表中一个个获取到存储该文件的所有外存分块的编号，每获取到一个编号则读入该分块的内容，一个块一个块地将文件读入内存。

### 1.3.4修改文件和文件夹

**1）复制文件：**先将需要复制的文件一块一块地复制进外存中标记为空的分块中，若复制成功（即剩余空间足够），则接着修改该文件在内存中对应的索引编号以及目录树中的节点，最后再将修改好的索引和目录信息写回外存，以实现内存外存中的数据同步。若复制不成功，则不修改目录和索引，已在外存复制的分块也不用删除，因为这些分块未被标记已使用，下次再修改外存时还会使用这些分块。

**2）删除文件**：删除文件时不需要真的去外存中将已经存入的内容删除，而是直接将内存索引数组中对应存储该文件的分块编号的数组元素中的内容修改为0即可。然后再删去内存目录树中对应该文件的节点，最后将内存和目录更新后的信息写回外存。

**3）创建文件夹：**文件夹节点只与目录树有关，创建文件夹节点只需先在内存目录树中添加文件夹节点，再写回外存目录数组即可，不需修改索引和文件内容。

**4）复制文件夹，删除文件夹：**这两种操作实际上是对内存目录树中该文件夹节点对应的子树中的所有文件进行上述复制，删除操作。

### 1.3.5 导入导出

**1）导入**

使用import命令从外部硬盘中将文件导入到本系统中。首先检测目标文件大小，然后后在本系统的硬盘空间中寻找空闲块计算本系统中的空闲空间大小，若不够容纳目标文件，则返回报错提醒。能够容纳的情况下，使用大小为64M的缓冲区。若目标文件大小小于64M，则一次性读入整个文件到内存中，然后向本系统查询空闲块并更新目录。写入系统；否则每次最多读入64M到内存中，写入本系统之后再继续读入其余部分，每次最多读入64M。

**2）导出**

使用export命令，将本系统内部文件导出到外部硬盘中，不需要考虑外部硬盘的容量，使用64M作为内存缓冲区， 每次最多从本系统中读入64M，然后写入外部硬盘。

**3）数据写入写出优化**

为了加快数据的写入写出速度，以及减少硬盘的操作次数，我们决定将相邻的数据操作进行合并；当两条写入写出操作之间距离不超过4096B时，就将两条操作合并，比如写入目录信息时，记录所有的目录修改项，然后排序，然后分析两条修改之间的距离是否超过1024（即4096B），如果满足条件，则增大写入长度，继续向后分析；如果不满足条件，则立即写入当前长度的数据，再以新的一条指令为起点继续进行合并。这样可以大大减少数据操作的次数，尤其是在写入写出连续空间的时候，所以我们采取这样的策略来加快数据的操作速度。

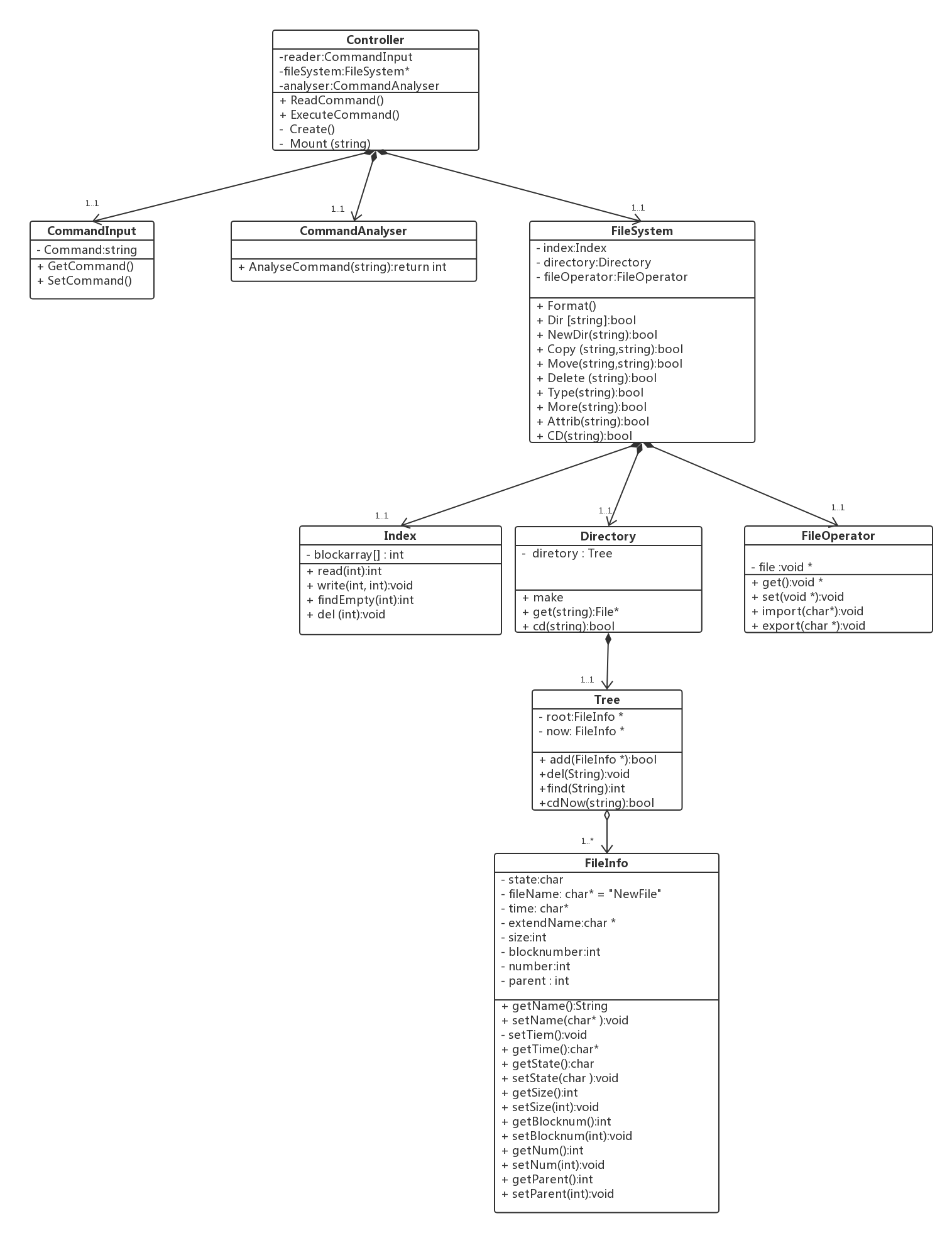
### 1.3.5关闭空间

释放索引和目录所占用的内存空间，释放系统文件指针。

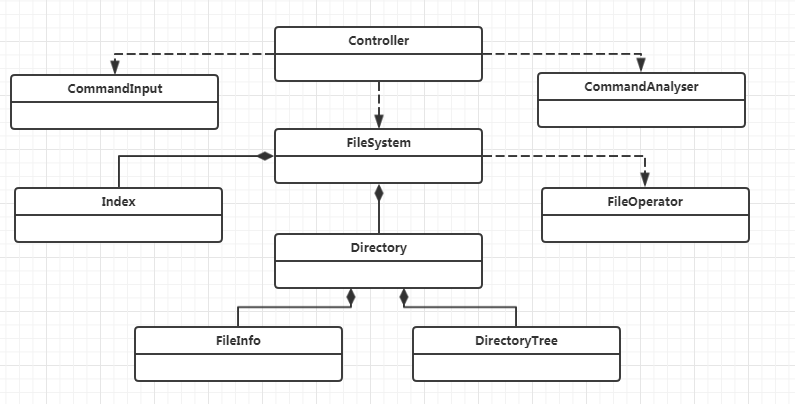
# UML建模

## 2.1类设计

### 2.1.1类图



简化类图



### 2.1.2类图描述

Mini-fs系统包括Controller，CommandInput，CommandAnalyser，FileSystem，Index，Directory，FileOperator，Tree，FileInfo类

Controller类负责控制整个系统的运行，包括CommandInput，CommandAnalyser，FileSystem类的实例，在系统开始时，Controller类使用ReadCommand()控制CommandInput类读入命令，然后将命令交给CommandAnalyser类分析命令，根据分析结果执行对应的命令，其中create与mount指令由Controller执行，其余命令由FileSystem类执行。

FileSystem类代表一个1G空间的文件系统，包括Index，Directory，FileOperator类的实例以及完成除create与mount外所有指令所需要的方法，例如Copy（）是完成copy指令所使用的方法。

Directory类代表这个文件系统的目录，Index是这个文件系统的索引表，FileOperator控制文件系同与外存数据块的读取交互。

Directory使用树的结构来表示整个目录的结构，树的每一个结点代表为一个FileInfo类的实例，其中包括这个文件的基本信息，包括文件名，扩展名，修改时间，节点编号，父节点编号，第一个数据块的索引等。

Index中包含一个大小为1024\*1024的静态链表，储存下一个数据块的索引。

FileOperator执行将数据信息写入到外存以及从外存中读取。

FileSystem指令控制这三个类交互完成不同指令，例如FileSystem需要读取文件时，先从Directory中找到该文件，并返回第一个数据块的索引，由FileOperator到对应位置读取数据，之后查找Index表获取下一个数据块所在的索引，重复操作指导遇见结束标志；FileSystem写入文件时，将文件以及一些信息加入到Directory的树中，并查找索引表找的空白快，更新该文件所代表的节点的初始索引值，并由FileInfo写入数据，之后查找下一个空白快，重复操作直到文件储存结束。

## 2.2用例设计

### 2.2.1用例图

