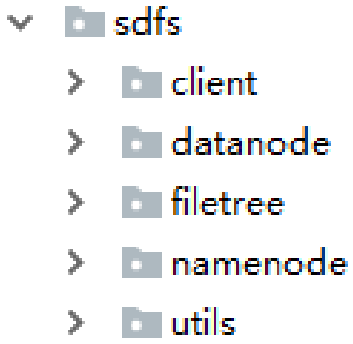
计算机系统工程 Lab1 文档

分布式文件系统—SDFS

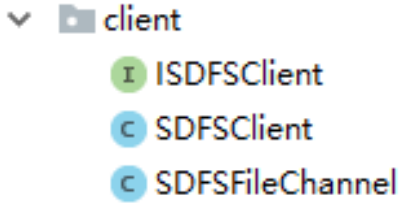
结构

代码结构

本次实现的文件系统与书本所介绍的文件系统类似，采用一系列 node 与 block 构成文件系统。Node 存储文件信息，block 存储文件数据。具体实现如下：

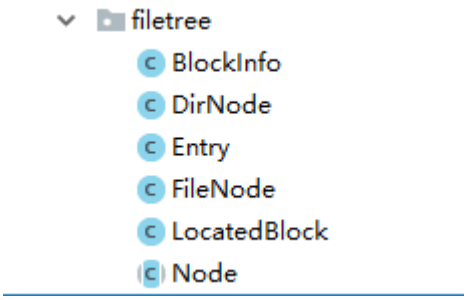


client 包：用于客户端操作，即文件系统的使用者操作类，其中 SDFSFileChannel 类是客户端与文件系统的读写操作接口，用于进行对文件的操作，类似于 unix 文件系统中文件描述符 fd 或者 open file table 中的表项，通过其可以实现对文件的读写操作。



DataNode 包：文件系统与磁盘的文件存储的接口，实现了 write 与 read 函数，用于实现对 block 即文件内容的读写操作

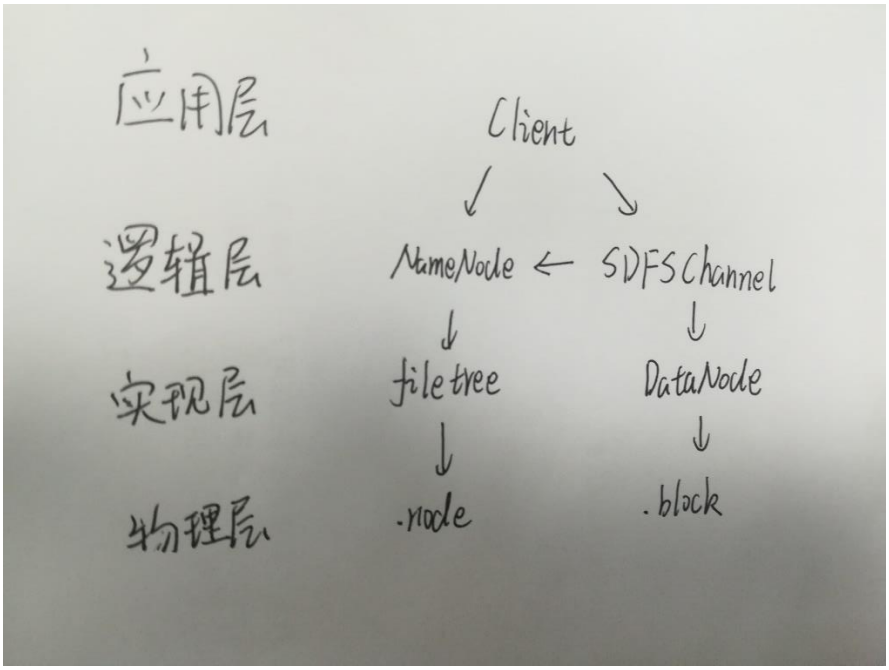
Filetree 包：该文件系统的主体架构，DirNode 与 FileNode 分别用于表示文件夹与文件节点；Entry 为文件夹的条目，存储在 DirNode 下，保存了该文件夹目录下条目；BlockInfo 存储在 FileNode 底下，用于存储该文件所具有的 block(即包含文件内容的块)



NameNode 包：文件系统的主要功能承担者之一，实现了文件系统的基本操作：创建文件与文件夹，打开文件通道，关闭文件通道等。是用于客户端与文件系统的操作接口。

SdfsUtils 包：个人添加的工具类包，实现了几个小函数，用于系统实现过程中的一些功能。

分层结构



个人理解中，该文件系统初步分为四层：应用层，逻辑层，实现层，物理层

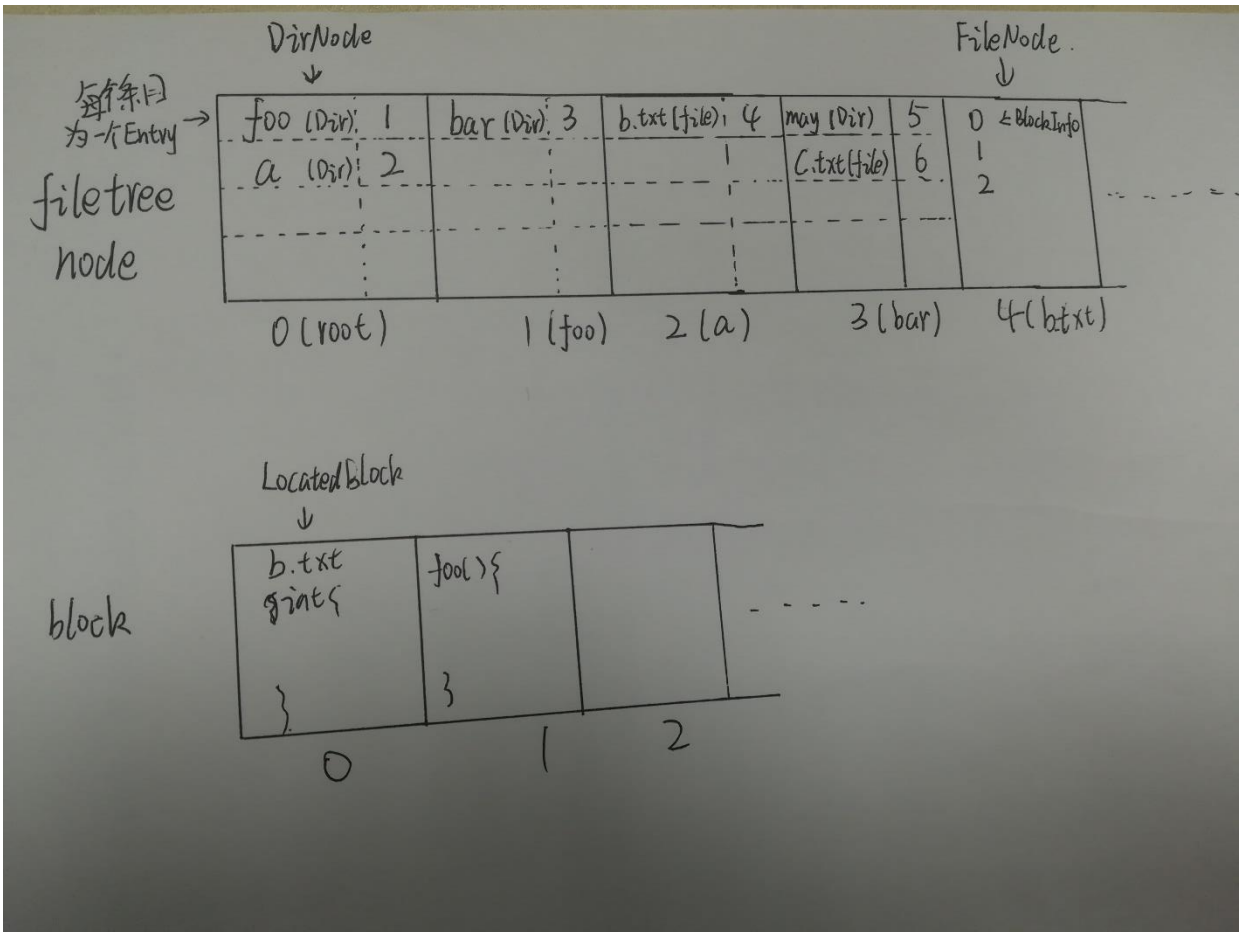
应用层：Client 即为客户端操作，客户端在创建文件或者文件夹时通过与逻辑层的 NameNode 交互，通过调用其函数实现文件与文件夹的创建，当需要对文件进行操作时，通过 NameNode 获取 SDFSChannel，即通道，而后通过对通道的操作实现对文件的读写

逻辑层：连接实现层与应用层，NameNode 主要接收应用层信息，对实现层的 DirNode 与 FileNode 进行创建操作，SDFSChannel 作为文件通道，在需要读写文件时与下层 DataNode 接口进行交互完成读写

实现层：filetree 作为文件系统的架构，维持了文件树的形态，并将文件树存储在物理层的.node 文件中，DataNode 通过对物理层的.block 文件的操作，实现对文件内容的读写

物理层：即硬盘，存储了文件树与文件内容

文件系统结构示意：



类似于 Unix 文件系统的底层实现，node 存储了文件与文件夹信息。文件夹 node 中的条目包含了子文件的名称与对应的 node；文件 node 中包含了该文件所占有的 block 的编号。

寻找文件：每当寻找文件时，从 root 开始寻找，根据名称寻找到对应的 node 编号；之后到对应的 node 读取信息，继续寻找；最后到文件时，读取文件 node 中的 block 信息；然后读取对应的 block 中的内容，对 block 进行操作完成读写。当文件不存在的，可以选择创建文件或者目录，然后修改 node，并将 node 存储到物理层。

读写：当需要对文件进行读写操作时，依据上述方法寻找文件，之后获取对应的 fileNode 并打开文件流通道 SDFSChannel，将通道指向对应的 node。并且该通道通过 DataNode 接口，传入文件块号，对该文件的内容进行读写。

SDFS 与 Unix 文件系统的异同

相同点：

都是通过 node 与 block 层来实现文件系统结构：node 用于存储文件与文件夹的信息，block 用于存储文件内容

不同点：

本文件系统只能通过绝对路径寻找文件，不能通过相对路径寻找文件

本文件系统不能为文件创建连接
本文件系统没有实现数据恢复功能

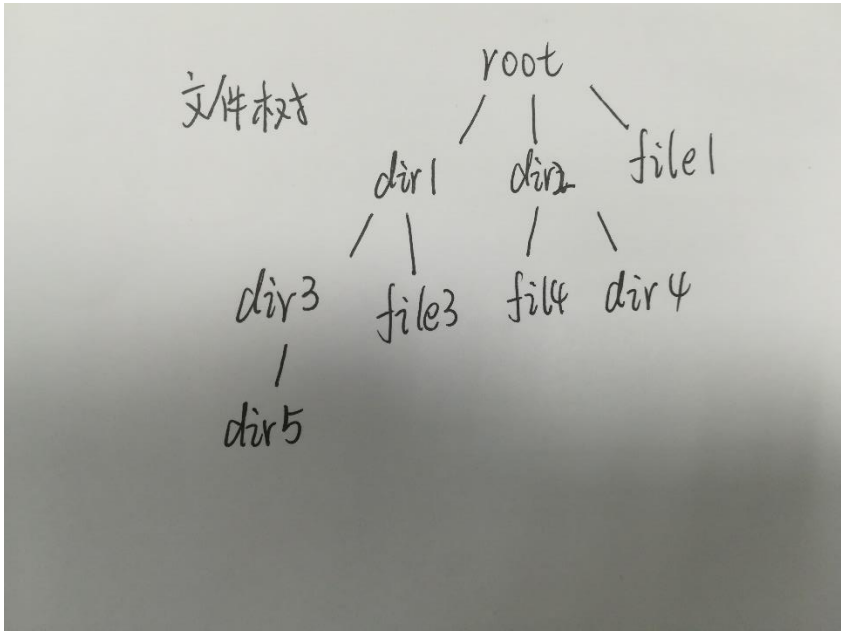
唯一写者问题解决：

所有被打开的通道都会存储在 NameNode 的一个链表中。
每次打开新的读写通道时，会遍历链表，寻找是否有一个文件已经以读写的方式打开：如果有，则抛出异常，如果没有，则为该文件打开读写通道，并保存该通道
关闭通道时，将通道从链表中移除，以便后续可以再次以读写方式打开该文件

关闭时存储文件问题解决：

每次打开读写通道时，采用从 root 开始寻找文件，从 node 文件中读取文件状态，这样每次打开读取的文件都是上一次关闭之后的内容
在操作过程中，不将改变内容存储到磁盘，只有等待关闭该通道时才将改变存储到磁盘，并行更新对应 node 文件，以更新信息

文件树格式：



运行时文件树类似上图，但是每个父级目录只会存储该目录下的条目，不会越级存储。

为了做到持久化，将文件树节点以 node 文件形式存储在磁盘上，每次创建、读写文件时都要更新文件树并存储到磁盘

每次读取时，从 root 文件开始读取，反序列化，获取子目录并继续读取子目录，以寻找目标文件

遇到的困难与解决

序列化与反序列化的问题

解决：最初不知道怎么实现，后面查询了序列化操作，发现只需实现接口，并重写 writeObject 与 readObject 即可

每次打开通道时，两个不同的通道的操作会互相影响

解决：考虑后发现最初的寻找文件与打开通道时通过遍历运行时的对象树实现查找文件的，因此两个通道中的文件节点时同一个对象，改变会互相影响。之后改变方法，寻找文件以及打开通道每次都从底层文件树读取并创建新的对象，这样两个通道就不会影响到同一文件

写入与读取时由于文件内容分布在不同的块，判断条件繁琐，不知如何实现

解决：想到了 ICS 课程中 write 函数中有一个参数用于记录剩余需要操作的字节，便采用相同的变量记录，这样一来简便了写入不同块的操作。