

# 分布式lab1

---

## 注册代码

---

```
cd src
idlj -fall api.idl
javac *.java api/*.java impl/*.java utils/*.java -d bin/
orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
cd src
java -cp bin/ NameNodeLauncher -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
cd src
java -cp bin/ DataNodeLauncher -ORBInitialPort 1050 -ORBInitialHost localhost
[id]
```

```
cd src
java -cp bin/ ClientLauncher -ORBInitialPort 1050 -ORBInitialHost localhost
```

## 启动说明

---

1. DataNode文件夹和FsImage.txt需要保持原状
2. DataNode暂时设为2个(id分别为0和1)，也可根据需要修改代码中的MAX\_DATA\_NODE新增

## 结构

---

### 文件元数据FileDesc

- 时间
  - 创建时间createTime，在第一次创建文件时初始化
  - 最后修改时间lastModifyTime
  - 最后访问时间lastVisitTime
- BlockInfo列表

- 储存该文件储存在哪个DataNode的哪块block上
- 字符串转换
  - 提供toString和fromString两个方法进行fileDesc和String之间的转换
  - filepath/三个时间/blockInfoList分别以逗号为分隔符；blockInfo之间以分号为分隔符

## DataNode

- 每个DataNode对应一个static的id
- 每个DataNode的数据储存在自己的文件夹下
  - 比如DataNode1，其中的block全部储存在DataNodes/DataNode1/...中；DataNodeLauncher中检查是否存在该DataNode对应的文件夹，若不存在需要新建
  - block块的维护
    - 数据块.txt的形式储存，一个数据块对应一个文件(block\_id.txt)
    - 每次启动一个DataNode时，先查询对应文件夹，维护一个已存在的blockid数组blockIdList
    - 在需要分配新数组时，先random一个不在blockIdList中的blockid

## NameNode

- NameNode维护文件系统，表示成filePath到FileDesc的映射关系fileSystemMap
  - 文件元数据储存在FsImage中，每个文件对应一行，调用toString()方法得到对应的字符串表示
  - 创建NameNode时从磁盘加载FsImage，并根据filepath初始化fileSystemMap
  - 修改文件元数据时同步到FsImage
- NameNode维护列表writeList，储存被以w形式访问的文件filePath以实现写的原子性

## FsImage

- 每个文件的元数据对应一行
- 对每个文件而言
  - 各个数据以逗号分隔，储存顺序如下
  - filepath:test.txt
  - 创建时间:2023-11-12T14:17:39.309
  - 最后访问时间:2023-11-12T14:24:52.468
  - 最后修改时间:2023-11-12T14:24:43.652

- blockInfoList分别以逗号为分隔符
    - 每个blockInfo表示“datanode编号:blockid”，比如第一块储存在0号dataNode的第743块
- ```
test.txt,2023-11-12T14:17:39.309,2023-11-12T14:24:52.468,2023-11-12T14:24:43.652,0:743;
test,2023-11-11T21:21:30.694,2023-11-11T21:21:30.694,2023-11-12T13:59:06.619,0:2;
```

## 功能实现

---

### open

1. client调用open(filepath, mode)
2. 在ClientImpl中调用nameNode.open(filepath, mode)，返回fd，之后ClientImpl面向用户的接口中文件以fd标识。
  - ClientImpl需要有一个fd与filepath的映射关系，同时需要储存open的mode，还需要有元数据，因此维护一个Map<Integer, Pair<String, FileDesc, Integer>> fdToFileMap，其中Pair为自定义类
  - 如果nameNode.open(filepath, mode)返回空字符串，说明该文件正在被其他客户端以w的形式访问，本客户访问失败，返回-1
  - 否则新指定一个fd，并根据返回值新建fileDesc，将映射关系添加在fdToFilePathModeMap中，然后将fd返回
3. 在NameNodeImpl中实现open()方法
  - 由于同一时刻只能有一个client以w的形式打开文件，因此NameNodeImpl需要维护一个已被以w形式打开的文件列表writeFileList
  - 首先检查mode
    - 如果mode为w，首先检查filepath是否在writeFileList中，如果在，直接返回空字符串
  - 然后用get检查文件系统中的filepath是否已经存在文件
    - 如果不存在，新建一个FileDesc;并且新建一个BlockInfo，随机选择一个DataNode并初始化blockId为-1，表示尚未分配具体block；在实际写入数据时，由DataNode为该文件分配具体的block，并通过setBlockId(filepath, index, block\_id)方法表示设置filepath文件第index个块的block\_id更新元数据
  - 再次检查mode
    - 如果为w/rw，则将filepath写入writeFileList
  - 最后返回fileDesc.toString()给client

### read

1. client调用read(fd)
2. ClientImpl类的read()
  - 首先判断文件有没有被close(), 在fdToFilePathModeMap查找fd是否存在, 若不存在直接抛出异常
  - 再检查打开方式: fdToFileMap中查找对应fd的mode, 如果为w则读文件失败, 返回null
  - 在fdToFileMap中查找对应fileDesc中的blockList, 依次调用DataNode的read(), 拼合成一个新的bytes并利用copyOf生成一个和文本内容相同的块返回
  - 然后修改fileDesc的lastVisitTime
3. DataNode的read(block\_id)
  - 如果block\_id是-1, 直接返回null
  - 在对应文件夹下读取block\_id.txt, 将结果返回即可

## append

1. client调用append(fd, bytes)
  - fd指定文件, bytes表示需要写入的数据流
2. clientImpl中实现append方法
  - 首先检查fd是否合法
    - 在fdToFilePathModeMap查找fd对应的文件
    - 如果fd不在映射中, 抛出异常
  - 然后检查是否以w/rw的形式打开
    - 如果不是提示无法写入
  - 调用DataNode的append方法
    - 先处理最后一个块
      - 对于block\_id为-1的块, 需要先调用randomBlockId为其分配一个具体的blockid并更新DataNode的blockIdList/NameNode和当前client的文件元数据
      - 将最后一个文件块写满: 调用DataNodeImpl.getBlockSize()方法得到已写入的文件的大小blockSize, 再将前chunkSize - blockSize个字符写入该块中
    - 如果仍有剩余字符未写入
      - 将剩余数据切割成4\*1024大小的若干块
      - 循环以下过程写入各个块
        - 调用和最后一个块同样的DataNode的randomBlockId分配一个新的块, 更新DataNode/NameNode/client相关信息
        - 将数据写入新块
  - 最后更新client的fileDesc的blockIdList, 以及文件的lastModifyTime

### 3. DataNode中实现append方法

- 在指定block\_id.dat后添加新的数据即可

## close

#### 1. client调用close(fd)

#### 2. clientImpl的close()

- 先检查fd是否合法，不合法直接抛出异常
- 调用NameNode的close
- 删除对应的fd

#### 3. NameNode的close()

- 如果以w/rw的方式打开
  - 从writeFileList中删除对应filepath
  - 如果lastModifyTime晚于NameNode中fileDesc的lastModifyTime，则更新lastModifyTime和blockId
- 如果以r/rw的方式打开
  - 如果lastVisitTime晚于NameNode中fileDesc的lastVisitTime，则更新
- 将更新后的文件元数据储存到FsImage

## exit

#### 1. client调用clientImpl()提供的exit方法

#### 2. clientImpl的exit()

- 遍历已打开的文件，依次调用close()

## 测试

### 自动测试

| DataNodeTest          |      |
|-----------------------|------|
| ✓                     | ✗    |
| ↓ <sup>a</sup>        | ↓    |
| ⌵                     | ⌶    |
| ↑                     | ↓    |
| 🔍                     | »    |
| ✓ DataNodeTest (test) | 4 ms |
| ✓ testRead            | 3 ms |
| ✓ testAppend          | 1 ms |

|                |       |
|----------------|-------|
| NameNodeTest x |       |
| ✓              | 93 ms |
| testOpenWrite  | 78 ms |
| testOpen       | 5 ms  |
| testOpenRead   | 7 ms  |
| testCreate     | 3 ms  |

|               |        |
|---------------|--------|
| ClientTest x  |        |
| ✓             | 252 ms |
| testWriteFail | 192 ms |
| testWriteRead | 40 ms  |
| testReadFail  | 20 ms  |

## 手动测试

- 初始FsImage

```
test,2023-11-11T21:21:30.694,2023-11-11T21:21:30.694,2023-11-12T13:59:06.619,0:2;
```

- 新建文件

- open打开一个新文件

```
>>> open test.txt w
```

```
INFO: fd=1
```

- 此时FsImage为：

```
test.txt,2023-11-12T14:56:38.233,2023-11-12T14:56:38.233,2023-11-12T14:56:38.233,0:-1;
test,2023-11-11T21:21:30.694,2023-11-11T21:21:30.694,2023-11-12T13:59:06.619,0:2;
```

- 创建/最后访问/最后修改时间都一样
- -1表示尚未分配具体的blockid

- 普通读写操作

- 进行如下操作

```
>>> read 1
INFO: READ not allowed
>>> append 1 hello world
INFO: write done
>>> close 1
INFO: fd 1 closed
>>> open test.txt rw
INFO: fd=2
>>> read 2
hello world
>>> exit
INFO: bye
```

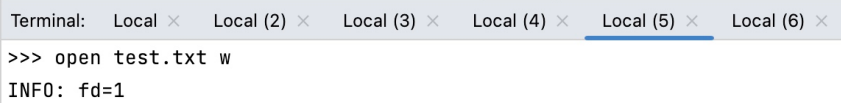
- 此时FsImage更新为：

```
test.txt,2023-11-12T14:56:38.233,2023-11-12T14:56:43.813,2023-11-12T14:58:57.266,0:192;
test,2023-11-11T21:21:30.694,2023-11-11T21:21:30.694,2023-11-12T13:59:06.619,0:2;
```

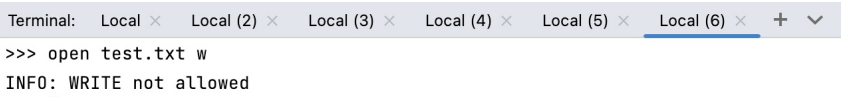
- “最后修改时间”和“最后访问时间”都已更新
- blockid也分配为192，即数据储存在192.dat上

- 写的原子性测试

- 在Terminal 5中以写的方式打开test.txt

A screenshot of a terminal window titled 'Terminal:'. It shows a tab bar with 'Local', 'Local (2)', 'Local (3)', 'Local (4)', 'Local (5)', and 'Local (6)'. The 'Local (5)' tab is selected and highlighted with a blue underline. The terminal content shows the command '>>> open test.txt w' and the response 'INFO: fd=1'.

- 在Terminal 6中同样以写的方式打开test.txt

A screenshot of a terminal window titled 'Terminal:'. It shows a tab bar with 'Local', 'Local (2)', 'Local (3)', 'Local (4)', 'Local (5)', and 'Local (6)'. The 'Local (6)' tab is selected and highlighted with a blue underline. The terminal content shows the command '>>> open test.txt w' and the response 'INFO: WRITE not allowed'.