

Lab1设计文档

idl接口文档设计

[DataBlock类](#)

[FileMetaData类](#)

[DataNode接口](#)

[NameNode接口](#)

文件目录解释

文件系统相关类实现与FSImage记录

[FileSystem类实现](#)

[FileDesc类实现](#)

[FSImage的记录与读取](#)

重要函数实现思路

[NameNodeImpl](#)

[open函数](#)

[allocateDN函数](#)

[close函数](#)

[DataNodeImpl](#)

[append函数](#)

[read函数](#)

[ClientImpl](#)

[open函数](#)

[append函数](#)

[read函数](#)

一些细节

[分配DataNode方法](#)

[开启服务参数的设置](#)

[文件存储位置](#)

测试结果

[手动测试：](#)

单元测试

client测试

NameNode测试

DataNode测试

idl接口文档设计

```
api.idl                                     Java 复制代码

1  module api {
2      typedef octet byteArray[4*1024];
3      struct DataBlock{
4          string dataNodeIP;
5          long long dtBlockId;
6          long long offsize;
7      };
8      typedef sequence<string> dataBlockInfo;
9      struct FileMetaData{
10         long fileType;
11         string fileName;
12         long long fileSize;
13         sequence<DataBlock> dataMap;
14         long long accessTime;
15         long long createTime;
16         long long modifyTime;
17         sequence<FileMetaData> childFileMetaData;
18     };
19     interface DataNode {
20         byteArray read(in long block_id);
21         void append(in long block_id, in byteArray bytes);
22         long randomBlockId();
23     };
24     interface NameNode{
25         //TODO: complete the interface design
26
27         FileMetaData open(in string filepath, in long mode);
28         //allocateBlock(string filepath,int filesize);
29         void close(in string fileInfo);
30         dataBlockInfo allocatedDN(in string filepath,in long filesize);
31     };
32 };
```

DataBlock类

作用：记录文件中的某数据块位于哪一个DataNode中的第几个block上，以及在该block上的偏移量（是否写完该block）

注：java-corba由两个端口号来确定服务

▼ DataBlock类

Java

📄 复制代码

```
1 struct DataBlock{
2     string dataNodeIP; //记录DataNode位置，由端口号和ip组成，形如"<port1>/<port
  2>/<ip>"
3     long long dtBlockId; //记录在该DataNode的第几个block，形如1,2,3
4     long long offsize; //记录在block中的偏移量，最大为blocksize，即4*1024
5 };
```

FileMetaData类

作用：文件元数据类，存储文件的元数据

▼ FileMetaData类

Java

📄 复制代码

```
1 struct FileMetaData{
2     long fileType; //记录文件类型，0代表目录，1代表文件
3     string fileName; //记录文件名
4     long long fileSize; //记录文件大小
5     sequence<DataBlock> dataMap; //记录数据块与datablock的映射关系
6     long long accessTime; //记录访问时间
7     long long createTime; //记录创建时间，在open时创建新文件的情况下更新
8     long long modifyTime; //记录修改时间，在append时更新
9     sequence<FileMetaData> childFileMetaData; //存储目录下的子文件（目录）元
  数据
10 };
```

DataNode接口

和助教初始给的没有变化

NameNode接口

作用：作为管理DataNode服务端，实现open, close, allocateDN功能

open: 接收来自client的请求，返回文件元数据，若文件不存在则新建

close: 关闭某文件，清除NameNode记录的文件状态信息（例如，清除该文件正在被写的状态）

allocateDN: 在client发送append请求前，client需要首先向NameNode获得分配的DataNode，NameNode通过该函数传递来的filesize为该文件**合适地**分配DataNode（实现时采取的方法是从容量最小的DataNode开始轮巡），返回DataNode的信息，形如
["1051/1052/localhost","1053/1054/localhost"]

▼ NameNode接口

Java

📄 复制代码

```
1 interface NameNode{
2     //TODO: complete the interface design
3     FileMetaData open(in string filepath, in long mode);
4     //allocateBlock(string filepath,int filesize);
5     void close(in string fileInfo);
6     dataBlockInfo allocateDN(in string filepath,in long filesize);
7 };
```

文件目录解释

```
1  |   api.idl
2  |   Main.class
3  |   Main.java
4  |
5  |   -api   #记录idl自动生成的一些结构、接口等
6  |
7  |   -config
8  |       Config.java   #配置文件, 只记录BLOCKSIZE大小, 为4*1024
9  |
10 |   -impl
11 |       ClientImpl.java   #client端的实现函数
12 |       DataNodeImpl.java #DataNode的实现
13 |       NameNodeImpl.java #NameNode的实现
14 |
15 |       -orb.db
16 |           counter
17 |           NC0
18 |           servers.db
19 |
20 |       -logs
21 |
22 |   -launcher
23 |       ClientLauncher.java   #开启client服务入口
24 |       DataNodeLauncher.java #开启DataNode服务入口
25 |       NameNodeLauncher.java #开启NameNode服务入口
26 |
27 |   -orb.db
28 |       counter
29 |       NC0
30 |       servers.db
31 |
32 |   -logs
33 |
34 |   -test
35 |       ClientTest.java   #client测试
36 |       DataNodeTest.java   #DataNode测试
37 |       NameNodeTest.java   #NameNode测试
38 |
39 |   -utils
40 |       FileDesc.java   #文件描述符类, 根据返回的元数据产生文件描述符
41 |       FileSystem.java #文件系统类, 管理整个文件系统
```

文件系统相关类实现与FSImage记录

FileSystem类实现

▼ FileSystem类

Java

📄 复制代码

```
1  /**
2   * 文件系统类
3   */
4  public class FileSystem implements Serializable {
5      api.FileMetaData rootFileData; //存储根目录的文件元数据
6      String fileName; //文件系统的名字
7      public ArrayList<Integer> dNBlockNum = new ArrayList<>(); //存储每个DataNode下的数据块数目
8      public ArrayList<String> dataNodeData = new ArrayList<>(); //存储DataNode的ip, 写死
9
10     public FileSystem(String rootpath, String fileName) {
11         this.fileName = fileName;
12         rootFileData = new FileMetaData(rootpath, 0);
13         //写死DataNode的ip和端口号 (有两个), 形式类似于"<port1>/<port2>/<ip>"
14         //这里初始化了两个DataNode
15         dataNodeData.add("1051/1052/localhost");
16         dataNodeData.add("1053/1054/localhost");
17         dNBlockNum.add(0);
18         dNBlockNum.add(0);
19     }
20
21
22     public static String newFilename(){
23         /* Wish the name has not appeared before in your NameNode, Good Luck _< */
24         return String.valueOf((int) (Math.random() * Integer.MAX_VALUE)) +
25             "TEST";
26     }
27
28     //实现根据文件路径找到对应文件的元数据, 该方法在filesystem中实现
29     public api.FileMetaData searchFile(String filepath){ ... }
30
31
32     //实现根据文件路径创建新的文件
33     public api.FileMetaData addNewFile(String filepath){ ... }
34
35
36 }
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
```

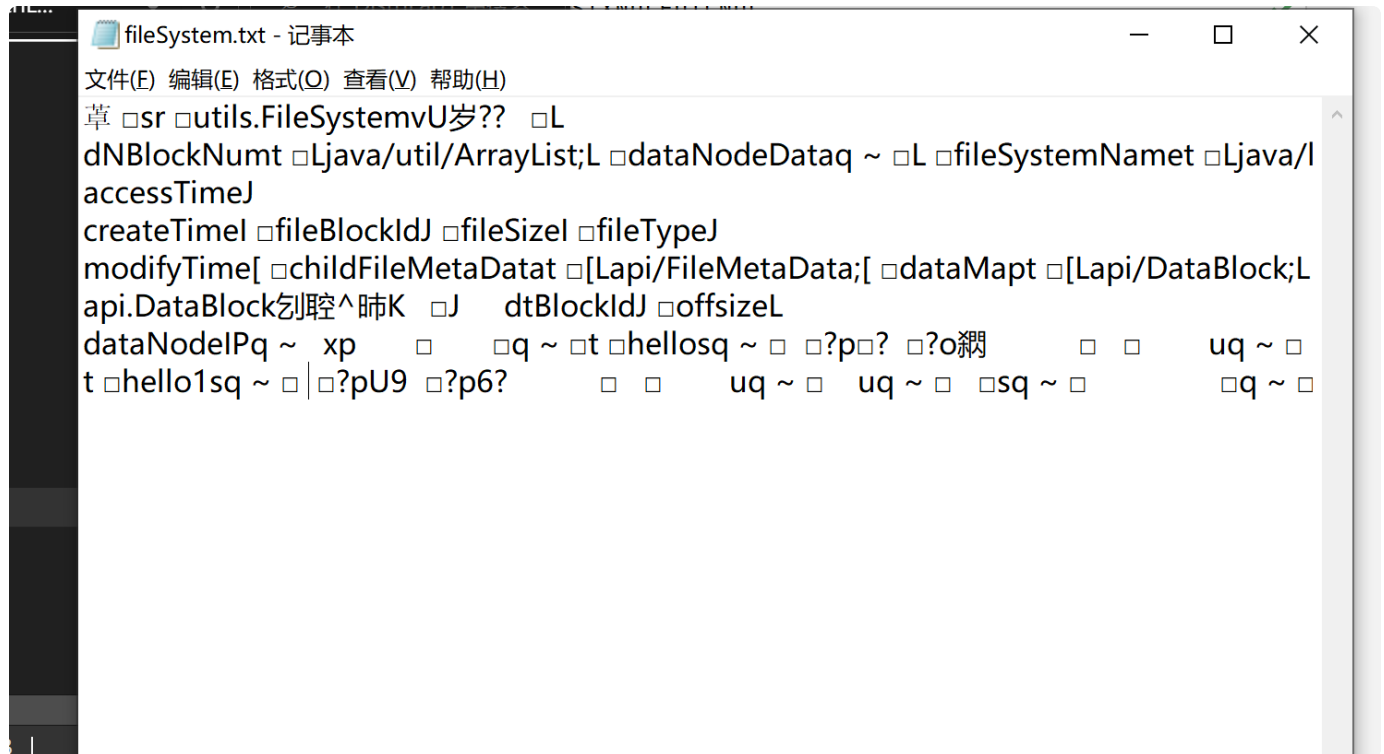
FileDesc类实现

```
1 public class FileDesc {
2     final long id; //返回的整数id值
3     FileMetaData fileMetaData;
4     int mode; //文件的打开方式
5     String filepath; //文件路径
6
7     public FileDesc(long id,int mode,FileMetaData fileMetaData,String file
      path) {
8         this.id = id;
9         this.mode = mode;
10        this.fileMetaData = fileMetaData;
11        this.filepath = filepath;
12    }
13
14    public FileMetaData getFileMetaData() {
15        return fileMetaData;
16    }
17
18    /* The following method is for conversion, so we can have interface th
      at return string, which is easy to write in idl */
19    @Override
20    public String toString() {
21        return null;
22    }
23
24    public long getId(){return id;}
25
26    public int getMode(){return mode;}
27
28    public static FileDesc fromString(FileMetaData str){
29        return null;
30    }
31
32    public String getFilePath() {
33        return filepath;
34    }
35
36    public void setFileMetaData(FileMetaData fileMetaData) {
37        this.fileMetaData = fileMetaData;
38    }
39 }
```

FSImage的记录与读取

将FSImage写入磁盘中采取的是java中的序列化技术，继承Serializable技术，写入根目录下的fileSystem.txt文件中，重启NameNode后，从该文件中读取并还原出filesystem。

某一状态下的Fsimage如下：



```
fileSystem.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
草 □sr □utils.FileSystemvU岁?? □L
dNBlockNumt □Ljava/util/ArrayList;L □dataNodeDataq ~ □L □fileSystemName □Ljava/I
accessTimeJ
createTime □fileBlockIdJ □fileSize □fileTypeJ
modifyTime[ □childFileMetaDatat □[Lapi/FileMetaData;[ □dataMapt □[Lapi/DataBlock;L
api.DataBlock刳腔^肺K □J dtBlockIdJ □offsetL
dataNodeIPq ~ xp □ □q ~ □t □hello □sq ~ □ □?p□? □?o網 □ □ uq ~ □
t □hello1sq ~ □ □?pU9 □?p6? □ □ uq ~ □ uq ~ □ □sq ~ □ □q ~ □
```

重要函数实现思路

NameNodeImpl

open函数


```
1 public api.FileMetaData open(String filepath, int mode) {
2     /*
3     mode: 0只读; 1只写; 2可读可写
4     */
5     FileMetaData fileMetaData = fileSystem.searchFile(filepath);
6     if(fileMetaData == null){ //如果找不到的话则新建文件
7         fileMetaData = fileSystem.addNewFile(filepath);
8         writeFS();
9     }
10    if(mode != 0){ //如果是w或者rw的话, 则加入usingFile数组中, 从而防止多个进程写
        操作
11        if(usingFile.contains(filepath)){
12            return new FileMetaData("", 0);
13        }else{
14            usingFile.add(filepath);
15        }
16    }else {
17        fileMetaData.accessTime = System.currentTimeMillis(); //修改访问
        时间
18    }
19    }
20    return fileMetaData; //返回文件元数据
21 }
```

allocateDN函数

使用轮巡的方法分配DataNode

```
1 public String[] allocateDN(String filepath, int filesize) {
2     FileMetaData fileMetaData = fileSystem.searchFile(filepath);
3     if(fileMetaData == null){
4         return null;
5     }
6     fileMetaData.modifyTime = System.currentTimeMillis();
7     DataBlock dataBlock = fileMetaData.getLastDataBlock();
8     fileMetaData.fileSize+=filesize;
9
10    //filesize+offsize没有超过一个块，使用的是最后一个datablock的DataNode的block
11    if(dataBlock!=null && filesize+dataBlock.offsize<Config.BLOCKSIZE){
12        fileMetaData.dataMap[fileMetaData.dataMap.length-1].offsize = file
size+dataBlock.offsize;
13        String[] r = new String[1];
14        r[0] = dataBlock.dataNodeIP;
15        writeFS();
16        return r;
17    }
18
19    //计算新的filesize, 除去前面的offsize
20    if(dataBlock!=null){
21        filesize = filesize + (int) dataBlock.offsize - Config.BLOCKSIZE;
22    }
23    //计算需要的block数目
24    int blocks = filesize / Config.BLOCKSIZE + (filesize % Config.BLOCKSI
E != 0 ? 1 : 0);
25    //加上第一个未满足的block
26    if(dataBlock!=null && dataBlock.offsize>0 && dataBlock.offsize<Config.
BLOCKSIZE){
27        blocks=blocks+1;
28    }
29    String[] results = new String[blocks];
30
31    int startIndex=0;
32    if(dataBlock!=null&&dataBlock.offsize>0&&dataBlock.offsize<Config.BLOC
KSIZE){
33        results[startIndex] = dataBlock.dataNodeIP;
34        startIndex++;
35    }
36
37    //TODO:使用轮巡的方法分配DataNode,
38    // startIndex为DataNode列表中block数目最少的DataNode下标, 是轮巡的开始位置
39    int shortestId=0;
40    for(int j = 0;j<fileSystem.dNBlockNum.size()-1;j++){
41        if(fileSystem.dNBlockNum.get(j)>fileSystem.dNBlockNum.get(j+1)){
```

```

42         shortestId = j+1;
43     }
44 }
45 }
46 for(int i=0;i<blocks;i++){
47     int index = (i+shortestId)%fileSystem.dataNodeData.size();
48     results[startIndex+i] = fileSystem.dataNodeData.get(index);
49     fileSystem.dNBlockNum.set(index,fileSystem.dNBlockNum.get(index)+1
);
50     DataBlock newdataBlock = new DataBlock(fileSystem.dataNodeData.get
(index),fileSystem.dNBlockNum.get(index),0);
51     fileMetaData.addOneDataBlock(newdataBlock,fileMetaData.dataMap.len
gth);
52     //修改最后一个block的offsize
53     if(i==blocks-1){
54         fileMetaData.dataMap[fileMetaData.dataMap.length-1].offsize =
filesize - i*Config.BLOCKSIZE;
55     }
56 }
57 writeFS();
58 return results;
59 }

```

close函数

close函数只需要将该文件从usingFile数组中移出即可

DataNodeImpl

append函数

只需要根据传来的block_id与bytes数组写入对应文件中（后缀名为txt）即可，若block_id为3，则写入该DataNode中的第三个block

```

FileOutputStream fos = new FileOutputStream(dir+ String.valueOf(block_id)
+".txt",true);

```

read函数

只需要根据传来的block_id来读取对应的文件即可

```

FileInputStream fis = new FileInputStream(dir + String.valueOf(block_id)+".
txt");

```

ClientImpl

open函数

作用：根据NameNode传回来的FileMetaData产生对应的文件描述符

▼ open函数

Java

📄 复制代码

```
1 public int open(String filepath, int mode) {
2     //根据接收到的文件元数据信息和读写模式生成文件描述符
3     this.filepath = filepath;
4     FileMetaData fileMetaData = nameNode.open(filepath,mode);
5     if(fileMetaData.fileName.equals("")&&fileMetaData.fileType==0){
6         return 0;
7     }else {
8         FileDesc fileDesc = new FileDesc(fdId,mode,fileMetaData,filepath);
9         fileDescArrayList.add(fdId-4096,fileDesc);
10        return fdId++;
11    }
12 }
```

为了和现实中的文件系统保持一致，将fd的起始值设置成了4096，也就是说第一个文件描述符为4096

append函数

和NameNode中的allocateDN函数相似，也需要处理未写满一个block的情况

```
1 public void append(int fd, byte[] bytes) {
2     FileDesc fileDesc = fileDescArrayList.get(fd-4096);
3     //检查是否有权限
4     if(fileDesc==null||fileDesc.getMode() == 0){
5         System.out.println("INFO: APPEND NOT ALLOWED");
6         return;
7     }
8     fileDesc.setFileMetaData(nameNode.open(fileDesc.getFilepath(),0));
9     fileMetaData = fileDesc.getFileMetaData();
10    DataBlock dataBlock = fileMetaData.getLastDataBlock();
11    int initLength = fileMetaData.dataMap.length;
12    //获得DataNode的地址
13    String[] dnIPs = nameNode.allocateDN(fileDesc.getFilepath(),bytes.length);
14    fileMetaData = nameNode.open(fileDesc.getFilepath(),0);
15    if(dnIPs == null){
16        System.out.println("the file doesn't exists!");
17        return;
18    }
19    int headLength=0;
20    for(int i=0; i < dnIPs.length;i++){
21        String[] ports = dnIPs[i].split("/");
22        connectDataNode(ports);
23        System.out.println("client append for datanode start...");
24        int length;
25        if(bytes.length + (dataBlock==null?0:dataBlock.offsize)<Config.BLOCKSIZE){
26            length = bytes.length;
27        }else{
28            if(i == 0){
29                length = (int) (Config.BLOCKSIZE - dataBlock.offsize);
30                headLength = length;
31            }else{
32                length = Math.min(Config.BLOCKSIZE,bytes.length-Config.BLOCKSIZE*i-headLength);
33            }
34        }
35        byte[] b1 = new byte[Config.BLOCKSIZE];
36        System.arraycopy(bytes,Config.BLOCKSIZE*i,b1,0,length);
37        int fileBlockId;
38        if(initLength+i-1<0){
39            fileBlockId = 0;
40        }else {
41            fileBlockId = initLength+i-1;
42        }
```

```
43         dataNode.append((int)fileMetaData.dataMap[fileBlockId].dtBlockId,b  
44         1);  
45     }
```

read函数

只需要根据文件描述符中的文件元数据中存储的datablock的数据来依次向对应的DataNode读取数据即可

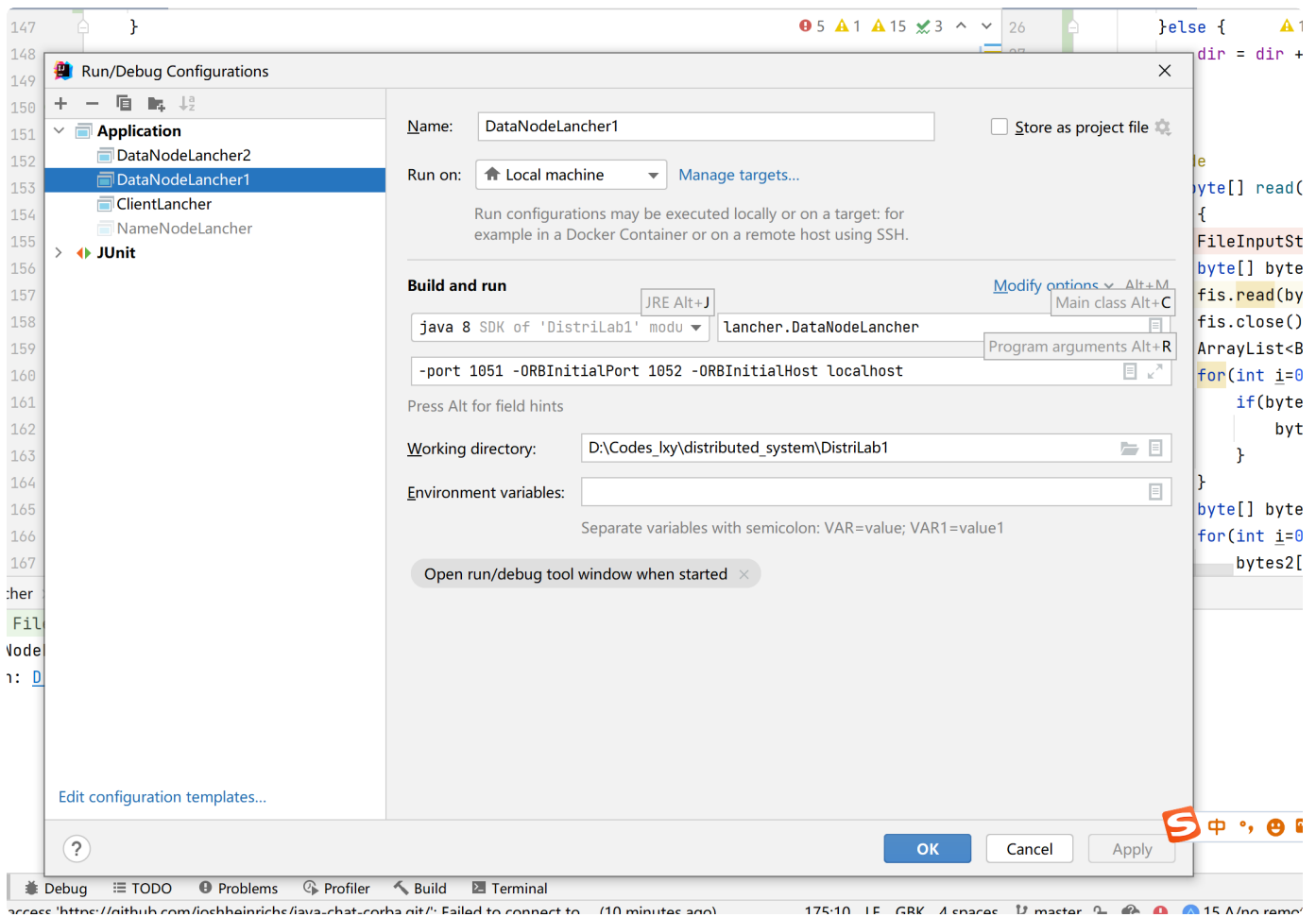
一些细节

分配DataNode方法

按顺序循环，从有最少block的DataNode开始，按照顺序分配

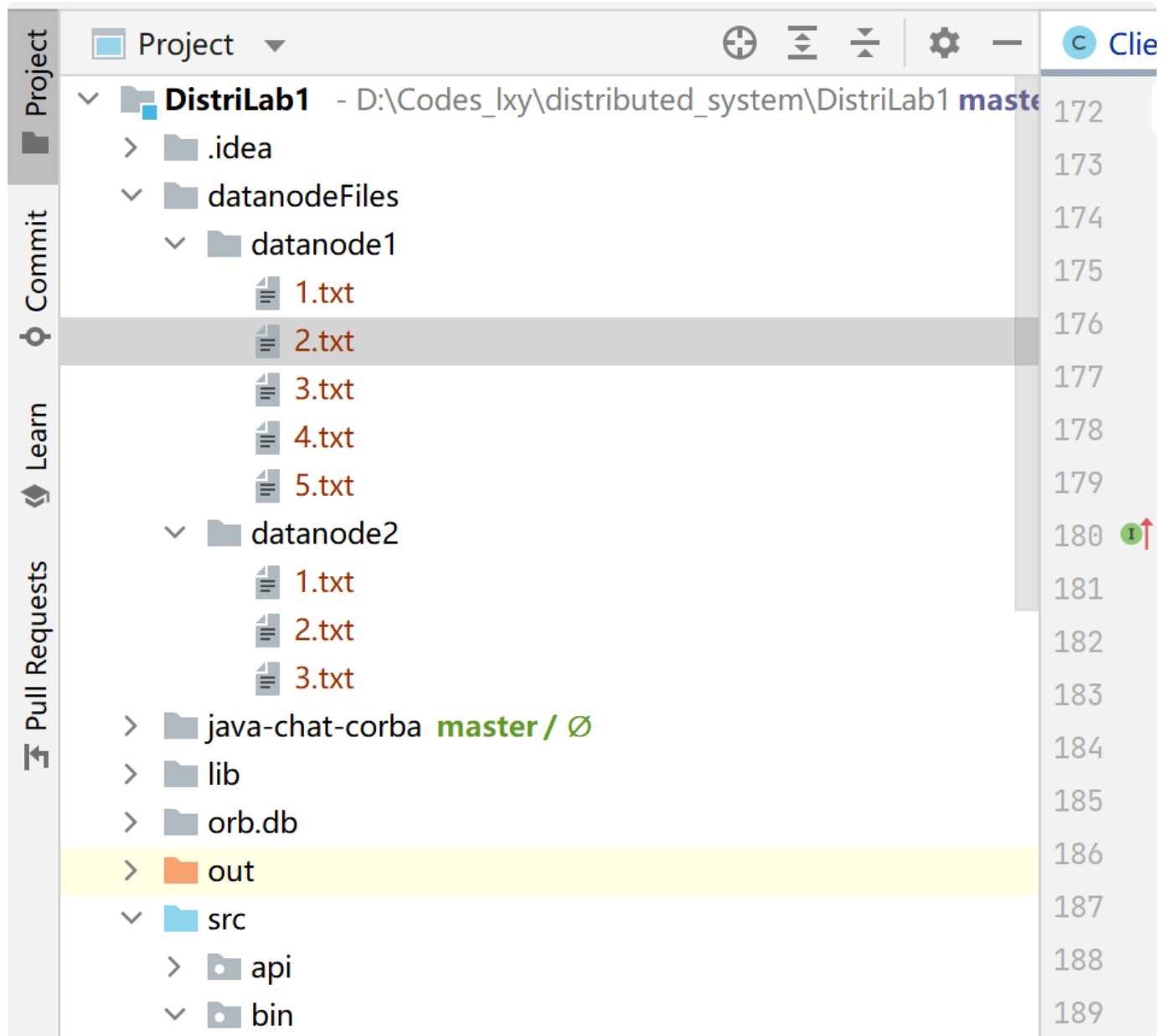
开启服务参数的设置

在idea中设置，截图如下



文件存储位置


在datanodeFiles文件夹下，如下



测试结果

手动测试：

为了方便直接在idea中运行测试

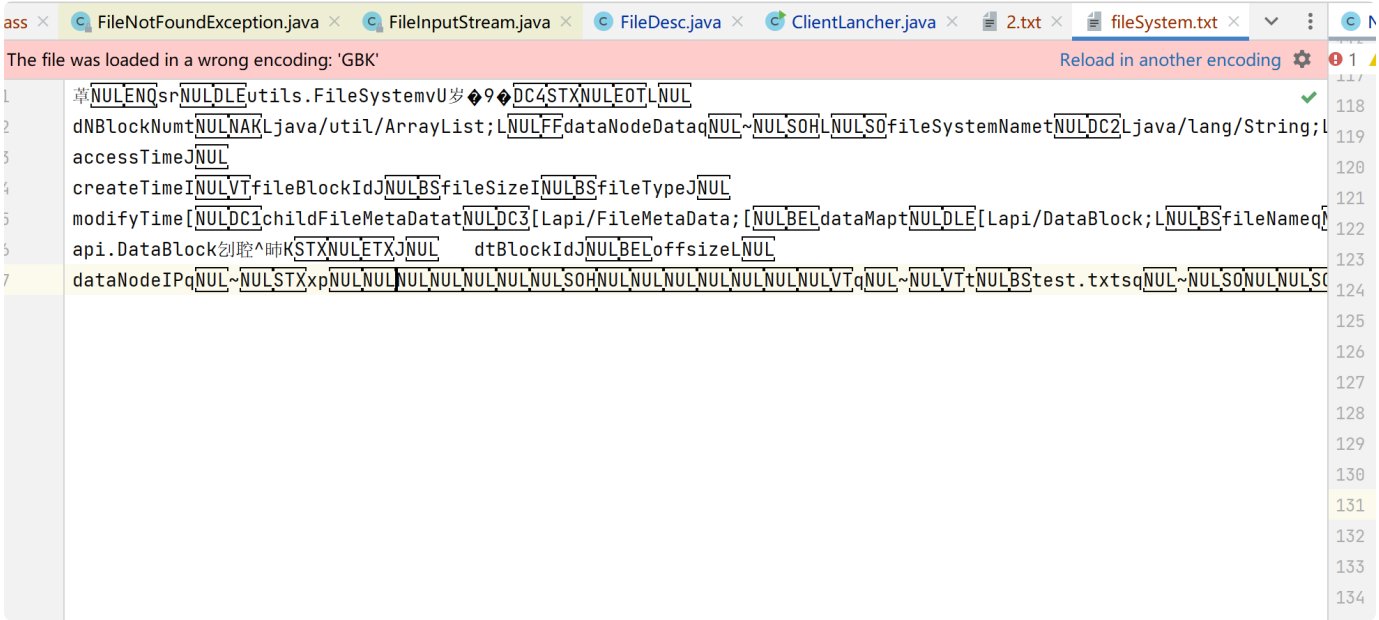


```
help
exit
open test4.txt w
INFO: test4.txt fd: 4096
read 4096
INFO: READ NOT ALLOWED
Commands:
open
append
read
close
help
exit
append 4096 hello world
run append
client append for datanode start...
INFO: write done
close 4096
INFO: fd 4096closed
open test4.txt rw
INFO: test4.txt fd: 4097
read 4097
hello world
exit
INFO: bye
```

```
Process finished with exit code 0
```

```
|
```

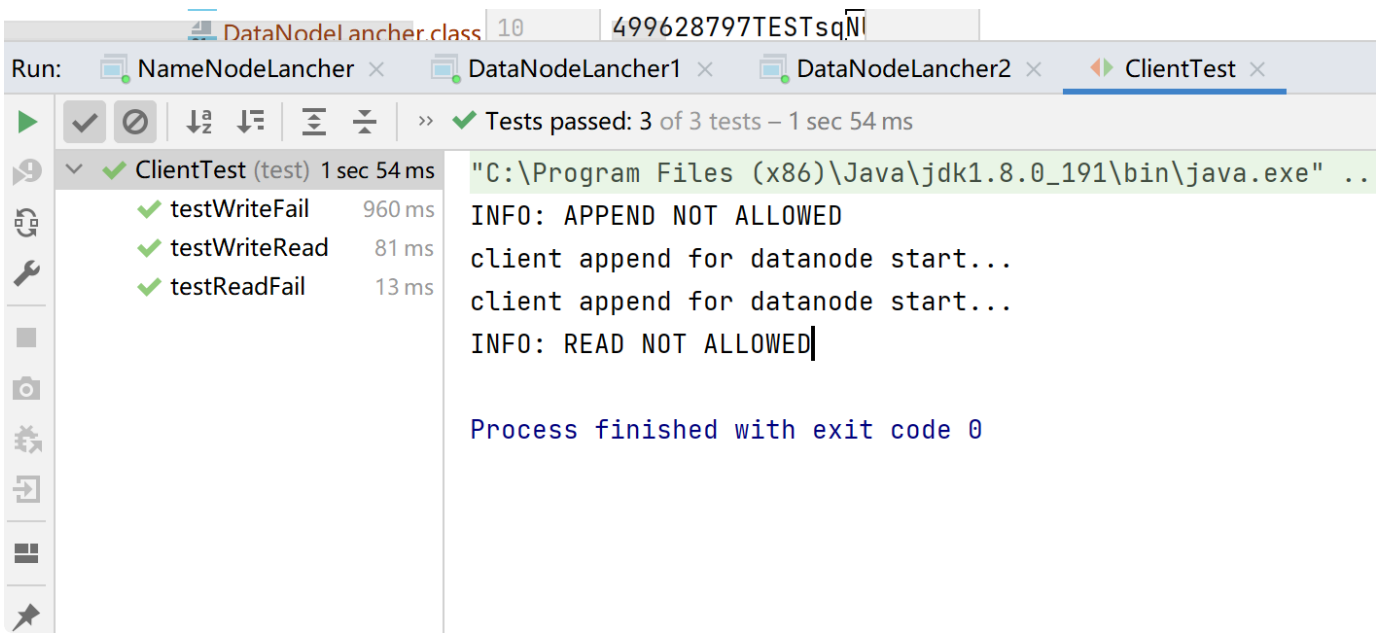
fileSystem.txt文件改变如下：



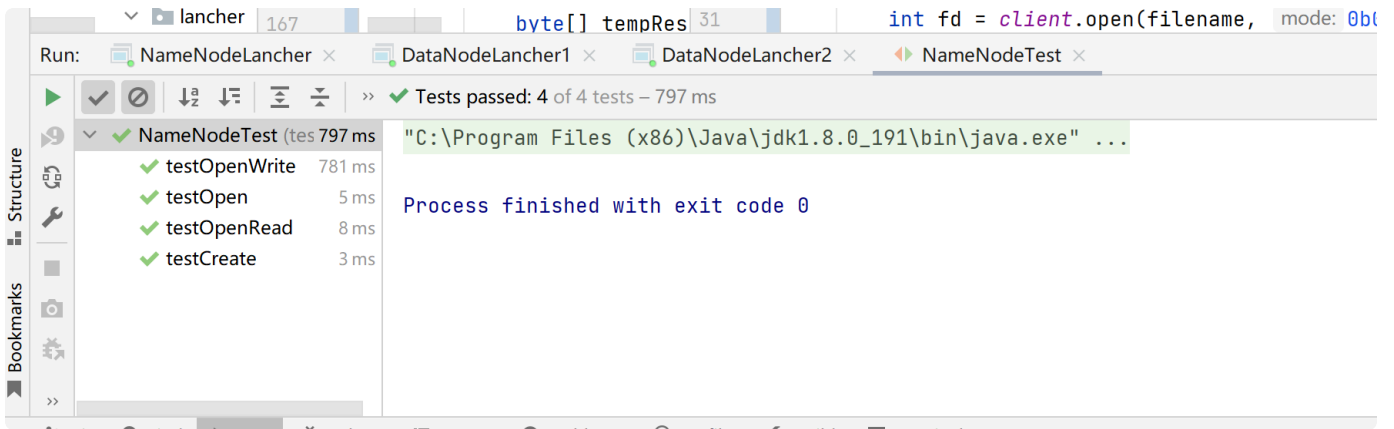
单元测试

mode: 0b10可读可写 0b01写 0b00读，与助教给的不同，除此之外都一样

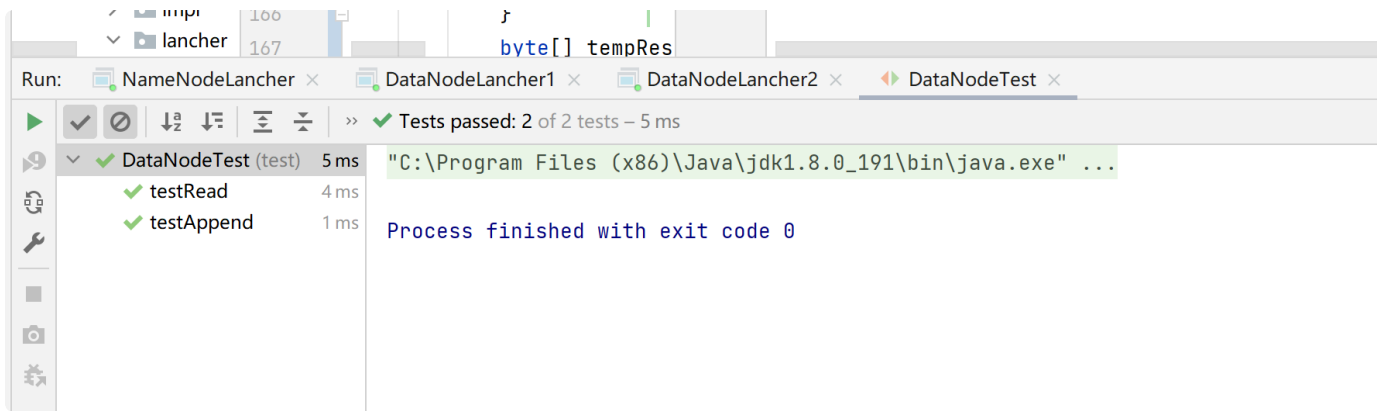
client测试



NameNode测试



DataNode测试



如果出错，建议重启NameNode