

分布式文件系统报告

姓名	班级	学号	日期	联系方式
张家豪	16计科8班	16337303	2019.1.4	994328597@qq.com

实验完成情况

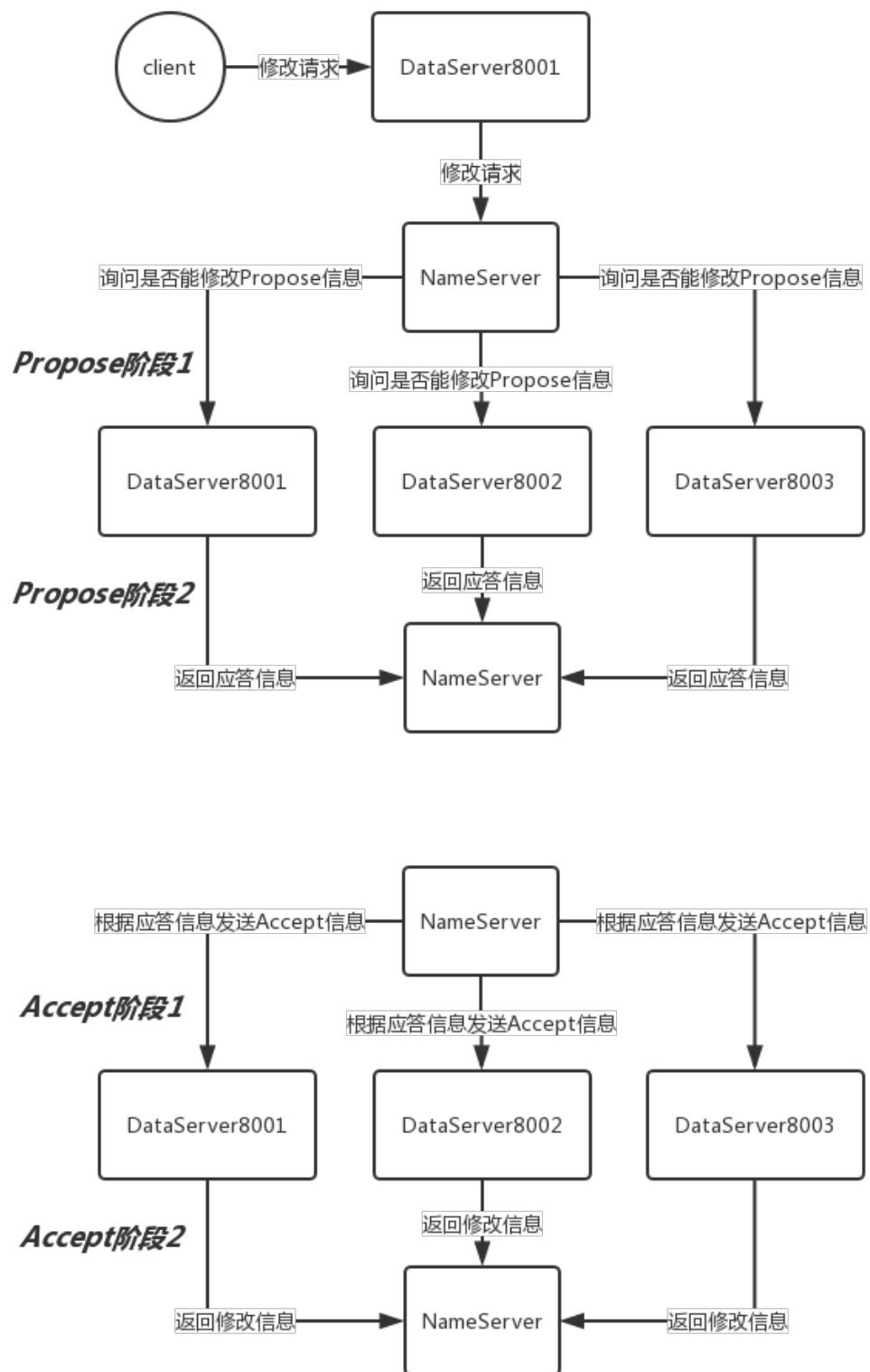
1. 用Python语言实现的简单的DFS
2. 用rpyc模块实现DFS中不同节点之间的通信
3. DFS具备创建create、删除del、访问list和下载get功能
4. DFS上的文件可以通过get下载下来作为磁盘文件缓存在本地
5. DFS采用强一致性，达成共识后立刻修改
6. DFS支持多用户多个客户端，文件可以并行读写
7. 使用Paxos共识方法达成副本服务器文件的一致性

解决思路

1. 一致性：如果在副本服务器上发生了删改、创建、或修改请求时，先不执行操作，通过目录服务器作为Proposer使用Paxos算法使得所有副本服务器达成共识。
2. 文件并行读写：

实现细节

- 流程图（仅修改）：



1. Paxos共识方法的实现：

Paxos达成共识的过程如上所示，其中中心思想是抢占式访问和后者认同前者，以下是几点需要注意的地方：

- NameServer发送给DataSet的值都带有时间戳，如果DataSet还没有完成修改，则会返回空值和空时间戳，若完成了修改，只会对那些时间戳大于自己的Propose信息作应答，返回当前的修改值和该值对应的时间戳。
- 如果NameServer收到的是空值和空时间戳，它会发送包含有自己要修改的值和当

前时间戳的Accept信息给DataServer；如果收到了多个已经修改的值 v_1, v_2, \dots, v_n 和对应的时间戳，（根据递归的证明，这实际上不会发生）它会选择时间戳最大的予以认同，提交最大时间戳对应的值 v_i 和自己的时间戳，如果它没有收到任何信息，则不会进入下一阶段，因为它知道可能是因为网络延迟，自己的修改已经是无效的了，时间戳在它之后的修改已经生效

```
1  # NameServer.py
2  agreeProcess = {} #以文件为下标，记录处于共识过程的文件，如果达成共识之
   后就删掉下标
3
4  def exposed_prepare(self, file, value):
5      if file not in agreeProcess:
6          agreeProcess[file] = 0
7      agreeProcess[file] += 1 #说明多进来了一个冲突操作需要达成共识
8      cur_timestamp = agreeProcess[file]
9      max_timestamp = cur_timestamp
10     accepted_value = value
11     for DS in dataServers:
12         conn = rpyc.connect(DS[0], DS[1])
13         replica = conn.root.Replica()
14         sign, value, timestamp = replica.confirm(file, value,
   cur_timestamp) #向每个DataServer发送想要提交的操作
15         print(sign, value, timestamp)
16         if timestamp != None and timestamp > max_timestamp:
17             max_timestamp = timestamp
18         accepted_value = value #accepted_value记录最大时间戳的
   返回值
19
20     total_ = True
21     # print(max_timestamp, accepted_value) #输出调试语句
22     for DS in dataServers:
23         conn = rpyc.connect(DS[0], DS[1])
24         replica = conn.root.Replica()
25         sign, path, value, timestamp = replica.accept(file,
   accepted_value, cur_timestamp) #发送accept信息
26         print(sign, path, value, timestamp)
27         if sign == True:
28             print(path + "is modified to " + value)
29         else:
30             total_ = False
31
32     if total_ and file not in fileList: #total_是true说明操作成功
33         fileList.append(file)
34
35     if cur_timestamp == agreeProcess[file]: #当前的操作就是最后一个
   提议则完成共识
36         del agreeProcess[file] #达成共识后，删除过程中产生的信息
```

```

37         for DS in dataServers: #在所有DataServer也删除所有达成共识的
            信息来使得后面的修改能够实现，避免因为后者认同前者的机制是的后面的操作无法成功执行
38             conn = rpyc.connect(DS[0], DS[1])
39             replica = conn.root.Replica()
40             replica.done(file)
41
42         return total_

```

```

1  # DataServer8001.py
2  def exposed_done(self, file): #删除达成共识期间的信息
3      if file in laststamp:
4          del laststamp[file]
5      if file in value4file:
6          del value4file[file]
7      if file in timestamp4file:
8          del timestamp4file[file]
9
10 def exposed_accept(self, file, value, timestamp): #响应accept信息
11     if file not in value4file and file not in timestamp4file:
12         sign = True
13         value4file[file] = value
14         timestamp4file[file] = timestamp
15
16         if file not in locks4file: #用于创建时加锁
17             ocks4file[file] = 1
18         else:
19             while locks4file[file] != 0: #修改时加锁
20                 time.sleep(0.1)
21
22         locks4file[file] = 1
23         with open(path+file, 'w') as f:
24             print('要写入的值是:', value)
25             f.write(value)
26         locks4file[file] = 0
27
28     elif timestamp4file[file] < timestamp:
29         sign = True
30         timestamp4file[file] = timestamp
31     else:
32         sign = False
33         pass
34     return sign, path + file, value4file[file],
    timestamp4file[file]
35
36

```

```

37 def exposed_confirm(self, file, value, timestamp): #这里为了简化
    propose的第二阶段，即使propose的timestamp小于本地已经修改的值对应的
    timestamp也会发送应答信息，处理交由NameServer负责
38     if file not in value4file and file not in timestamp4file:
39         return True, None, None
40     else:
41         return True, value4file[file], timestamp4file[file]

```

2. 排他锁和共享锁的实现：

- 在本次实验中为了实现并行读写，加入了排他锁和共享锁，当文件创建、删除和修改时，都会拿到排他锁，在文件读时会拿到共享锁。
- 在每个副本服务器行，共享锁由一个全局变量实现，当读操作拿不到锁时会返回失败，但是为了保证一致性，在创建、删除和修改时，会堵塞直到拿到锁完成对应操作再释放锁。

```

1  locks4file = {} #0时无锁，1代表互斥锁，2代表共享锁
2
3  #修改和创建过程
4  if file not in locks4file: #用于创建时加锁
5      locks4file[file] = 1
6  else: #用于修改时加锁
7      while locks4file[file] != 0: #堵塞至直到获取锁
8          time.sleep(0.1)
9      locks4file[file] = 1 #
10 with open(path+file, 'w') as f:
11     f.write(value)
12 locks4file[file] = 0 #完成写后释放锁
13
14 #删除过程
15 def exposed_do_delete(self, file):
16     if(self.isExist(file)):
17         while locks4file[file] != 0:
18             time.sleep(0.1)
19         locks4file[file] = 1
20         os.remove(path+file)
21         del locks4file[file] #删除文件也删除文件的锁
22         return True, "delete successfully on " + path
23     else:
24         return False, "file not found on " + path
25
26 #读过程
27 def exposed_get(self, file):
28     if not self.isExist(file):
29         return False, "file not found!"
30     if locks4file[file] == 1:
31         return False, "Fail to read " + file
32

```

```
33     locks4file[file] = 2 #获得共享锁
34     file = path + file
35     print("file location", file)
36     content = open(file).read()
37     locks4file[file] = 0 #释放锁
38     return True, content
```

遇到的问题

- Paxos达成共识后，后面的再次修改无法生效

问题成因：

因为Paxos使用后者认同前者机制，前者留下的修改信息（value和timestamp）没有被删除，后者会用同样的value但是更大的timestamp让DataServer进行更新，这样只会修改timestamp的值并不会修改value。

解决方法：

在达成共识后删除相关信息（上面的代码注释中有，这里就不赘述了）

演示视频说明

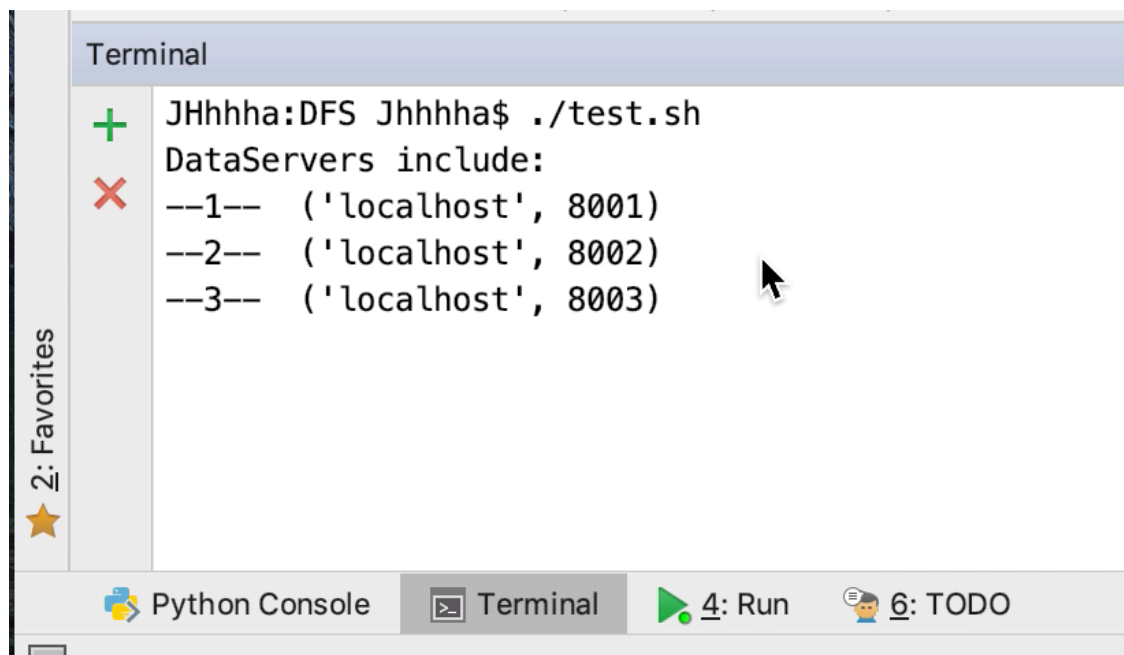
（以下截图取自录制的"演示视频.mp4"）

演示使用的脚本是内容如下

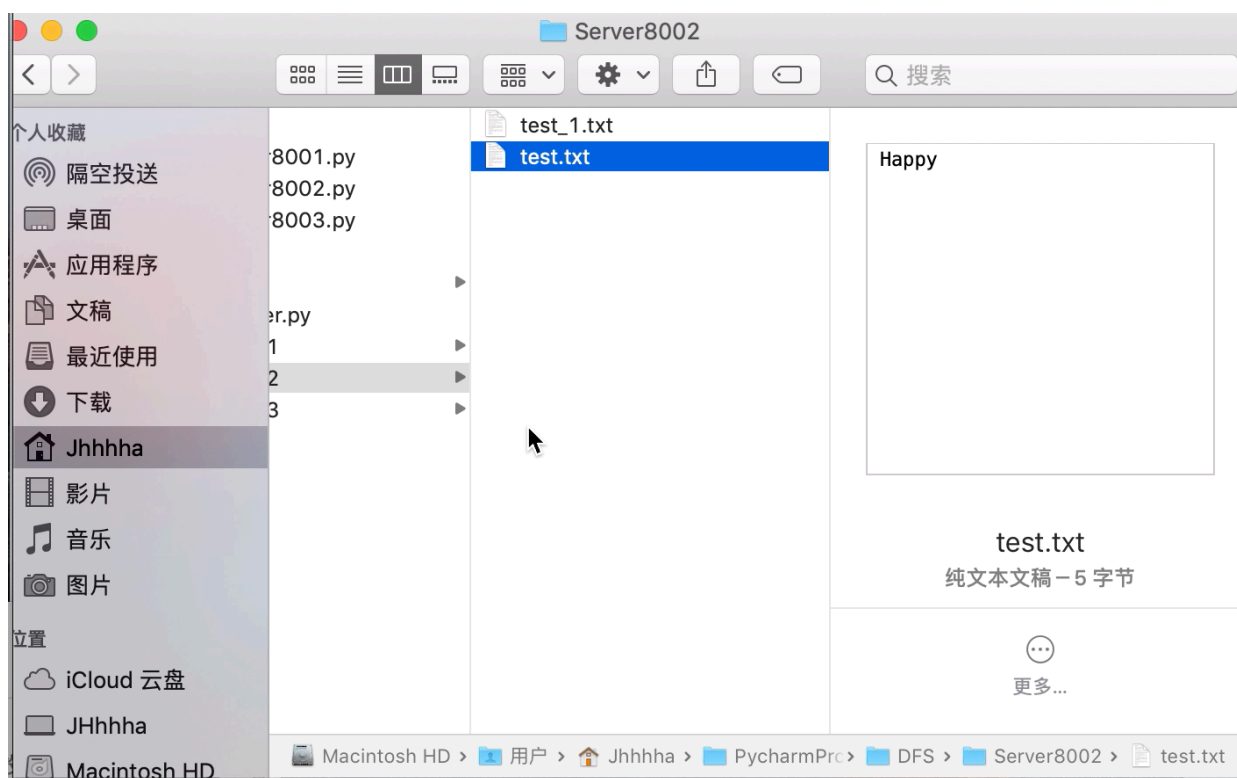
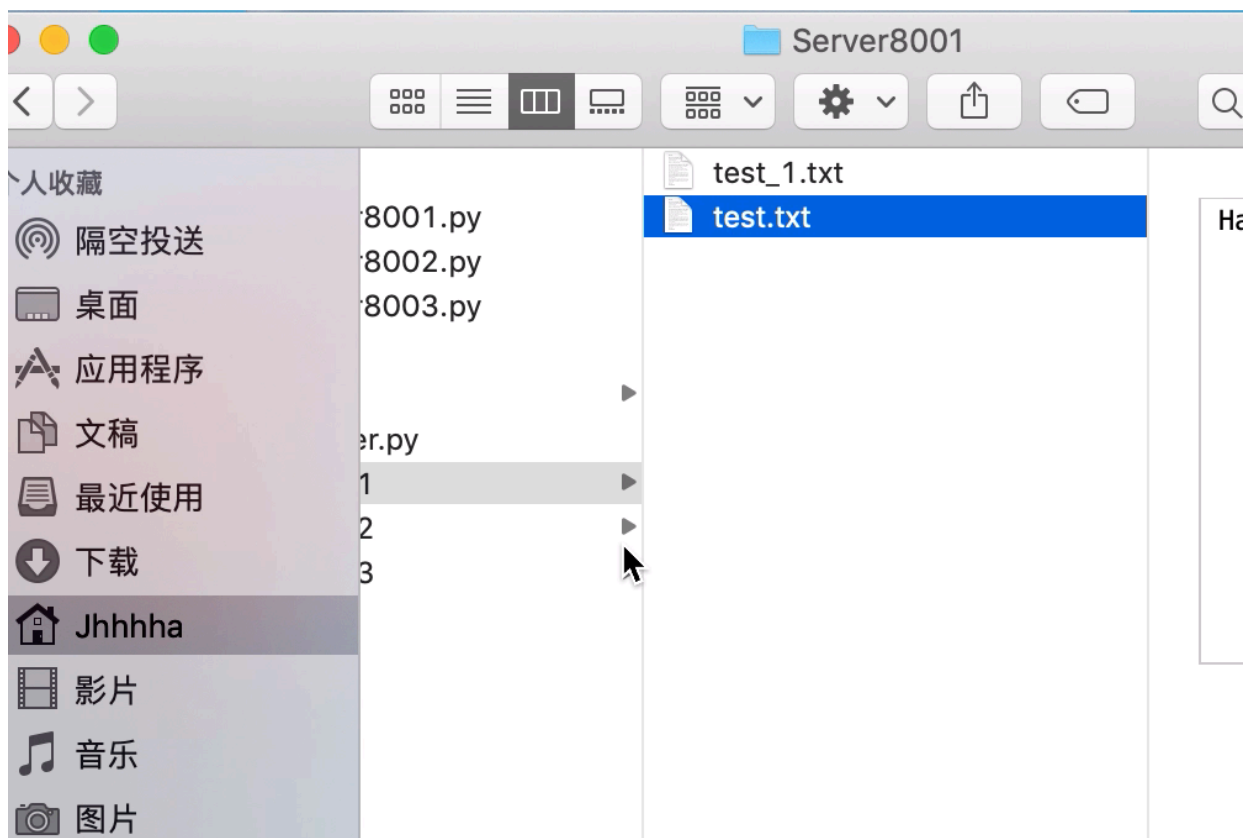
```
1  #!/bin/bash
2
3  python client.py show & #显示DFS中包含哪些数据节点
4  #显示内容如下
5  #DataServers include:
6  #--1-- ('localhost', 8001)
7  #--2-- ('localhost', 8002)
8  #--3-- ('localhost', 8003)
9  sleep 1
10
11 python client.py new localhost:8001 test.txt &#在8001端口上的服务器上创建文件
12 python client.py new localhost:8002 test_1.txt &#在8002端口上的服务器上创建文件
13 sleep 1
14
15 python client.py list localhost:8003 &#在8003端口上的服务器要求显示文件列表
16 sleep 1
17
18 python client.py write localhost:8001 test.txt 'sad' &
```

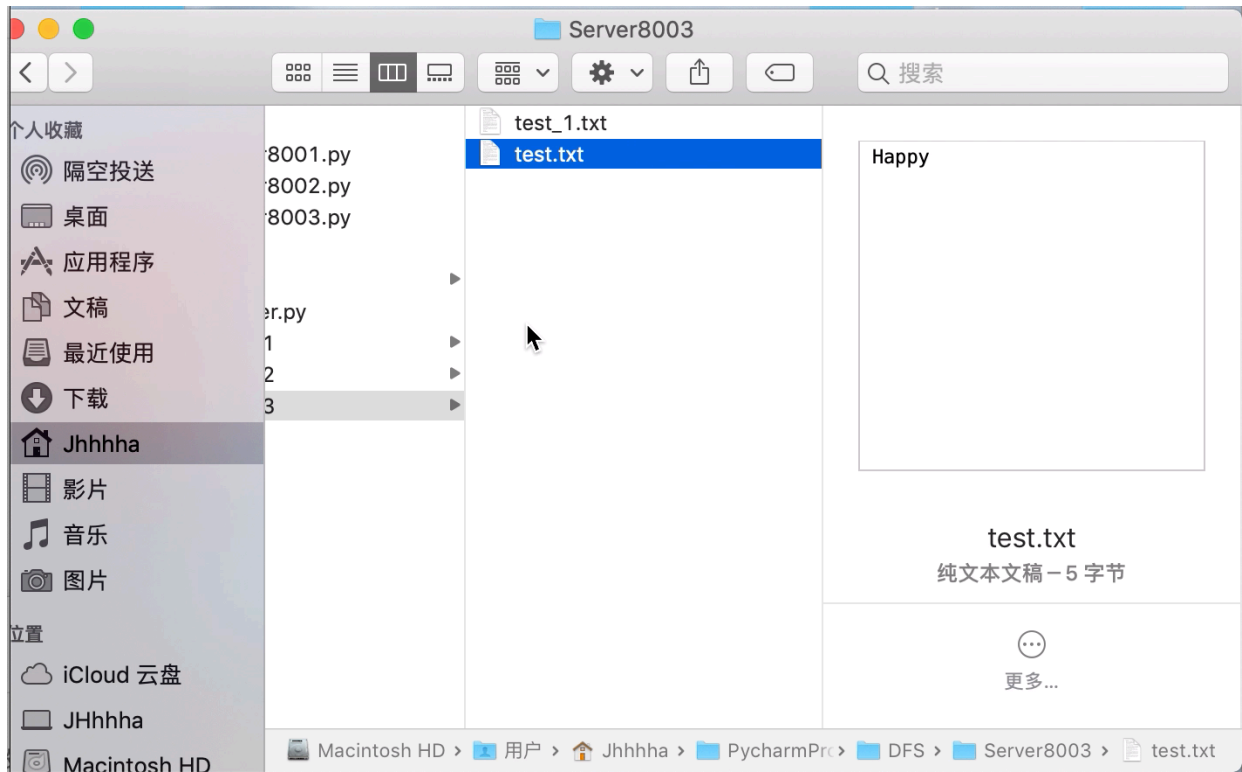
```
19 python client.py write localhost:8001 test.txt 'Happy' &#尽可能保证同时写入
20
21 sleep 1
22 python client.py get localhost:8003 test.txt &#从8003端口上的服务器拉取文件到本地，并保存在磁盘中
```

1. python client.py show指令获得数据服务器节点



2. python client.py new localhost:8001 test.txt 和 python client.py new localhost:8002 test_1.txt 指令分别用两个客户端在两个DataServer上创建文件text.txt 和test_1.txt 然后通过python client.py write localhost:8001 test.txt 'Sad' & python client.py write localhost:8001 test.txt 'Happy'同时写入，从截图可见DFS达成了一致。可能原因是DataServer上第一个提议在还没修改之前，就收到了第二个提议。





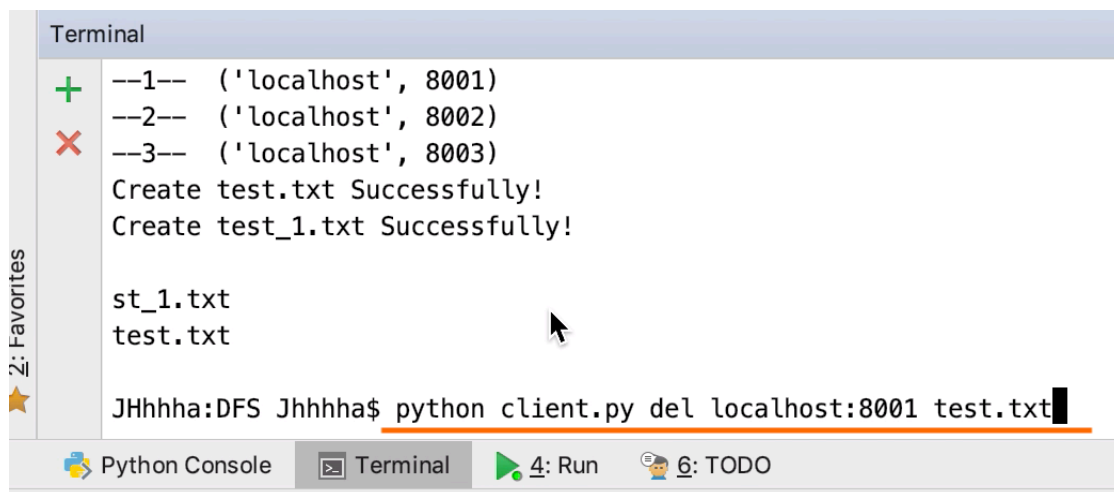
3. 执行python client.py list localhost:8003后显示文件列表

'st_1.txt'骑士就是'test_1.txt', 苹果电脑上显示文件的时候会多显示'.DS_Store', 我不知道怎么删去, 就用了`content = replica.list().strip('.DS_Store').lstrip()`, 结果会影响第一个文件的显示。

```
Terminal
+ --1-- ('localhost', 8001)
- --2-- ('localhost', 8002)
- --3-- ('localhost', 8003)
Create test.txt Successfully!
Create test_1.txt Successfully!

st_1.txt
test.txt
```

4. 执行del操作



```
Terminal
+ --1-- ('localhost', 8001)
- --2-- ('localhost', 8002)
- --3-- ('localhost', 8003)
Create test.txt Successfully!
Create test_1.txt Successfully!

st_1.txt
test.txt

JHhhha:DFS Jhhha$ python client.py del localhost:8001 test.txt
```

执行完后，每个DataServer的'text.txt'文件都被删除了，但是本地缓存的文件没有删除，可以根据用户自己的需要自行删除，具体请看视频演示。

总结

上面实现的过程有一个重大缺陷就是如果在达成共识的阶段有来自客户端的读请求时，读请求（也就是下载请求）会被响应，如果读的和正在达成共识的文件是同一个文件，而且正在达成共识的文件最后发生了修改，就会发生本地缓存和服务端上的文件不一致的情况。

锁机制之前就知道，至于新学到的东西，大概就是自己写一下的话，对Paxos算法有了更深的理解。之前都不知道DFS到底在干什么。实现得不容易啊，又要期末复习，总结就水一水了吧。