**题目名称**

**分布式文件系统**

姓名：陈欣宇

班级：人工智能与大数据

学号：21307347

日期：12.24

正文：

1. 题目要求

设计一个分布式文件系统。该文件系统可以是 client-server 架构，也可以是 P2P 非集中式架构。

**基本要求：**

（1）编程语言不限，选择自己熟悉的语言，但是推荐用Python或者Java语言实现；

（2）文件系统中不同节点之间的通信方式采用 RPC 模式，可选择 Python 版本的 RPC、gRPC 等；

（3）文件系统具备基本的文件操作模型包括：创建、删除、访问等功能；

（4）作为文件系统的客户端要求具有缓存功能即文件信息首先在本地存储搜索，作为缓存的介质可以是内存也可以是磁盘文件；

（5）为了保证数据的可用性和文件系统性能，数据需要创建多个副本，且在正常情况下，多个副本不在同一物理机器，多个副本之间能够保持一致性（可选择最终一致性即延迟一致性也可以选择瞬时一致性即同时写）；

（6）支持多用户即多个客户端，文件可以并行读写（即包含文件锁）；

（7）对于上述基本功能，可以在本地测试，利用多个进程模拟不同的节点， 需要有相应的测试命令或者测试用例，并有截屏或者video支持；

**加分项：**

（1）加入其它高级功能如缓存更新算法；

（2）Paxos 共识方法或者主副本选择算法等；

（3）访问权限控制；

（4）其他高级功能；

1. 解决思路
2. 本次课程设计的实现思路主要按照client-server架构，编程语言为python代码，调用python的xmlrpc实现不同节点之间的RPC通信。
3. 实现文件系统的功能有创建目录，删除目录，改变当前目录路径，写文件，读文件，删除文件6个功能。
4. 为了比对文件一致性，为文件计算哈希值并储存起来，当再次需要读或写文件时会将该文件计算哈希值与存储哈希值比较，确定文件是否被篡改。
5. 调用python的sqlite3用于存储和查询信息，因此在client-server架构基础上添加了database文件，用于调用SQL语句，并且起着维持服务器和客户端正常运行的功能。数据库主要设置了以下三个表：

服务器表(SERVERS)

|  |  |
| --- | --- |
| server\_id | address |
| 服务器ID | 服务器url，可被客户端查询调用 |

文件表(FILES)

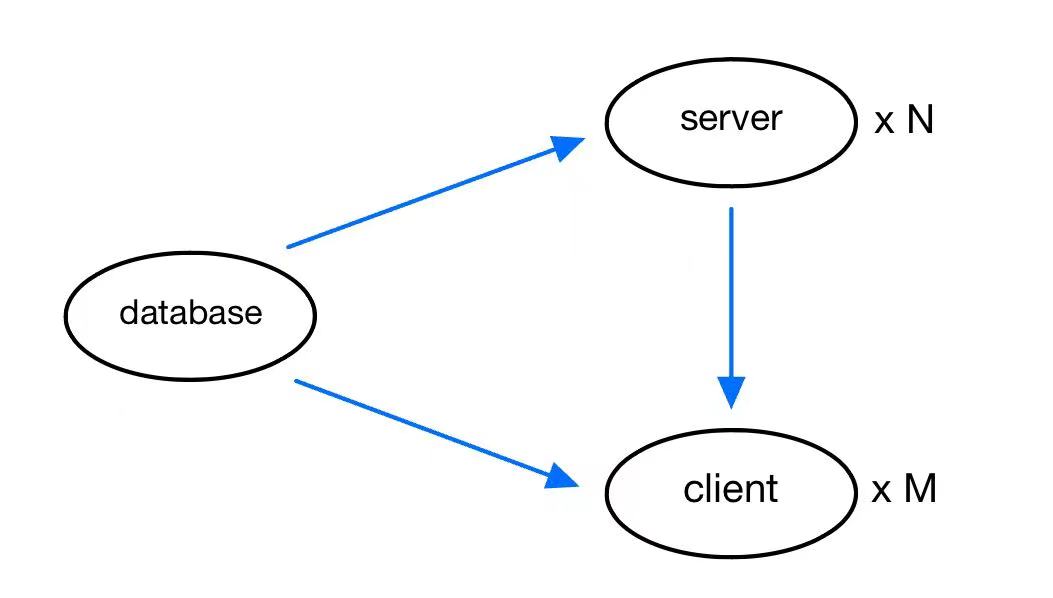
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Server\_id | Path | Name | Is\_backup | File\_hash | lastmodified |
| 服务器ID | 文件相对路径 | 文件名 | 是否为备份文件 | 哈希值 | 更新时间 |

缓存表(CACHES)

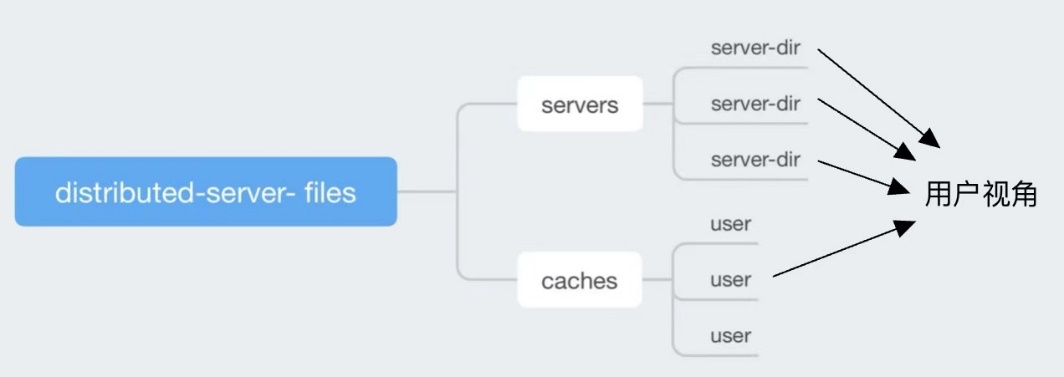
|  |  |  |
| --- | --- | --- |
| Server\_id | Path | Cache\_hash |
| 服务器ID | 文件相对路径 | 哈希值 |

1. 对于客户端缓存功能，我为每个用户创建了一个cache目录用于将存储最近文件，当用户读取文件时，都会将文件放到缓存中，并更新缓存表信息，在下次读取文件操作时，优先比对是否存在于缓存目录中（使用表查询即可比对），若缓存信息与文件原路径信息不符，则仍读取原路径文件，并更新缓存。对文件的写操作使用write through策略，即对写入本地缓存的同时也将数据写入分布式持久化存储中，这样也能够保持了不同用户之间多个缓存之间的一致性。
2. 对于数据的可用性和文件系统性能，我为文件创建了副本，副本的更新主要在写文件后，以及在读文件时若检测到原文件受损，通过筛选可用副本，查找到第一个可用副本后将其作为原文件，并更新其他副本，保持副本之间的一致性。
3. 对于文件并行读写功能，我使用了两个列表表示共享锁和排他锁，存储着文件的相对路径，根据共享锁和排他锁的特性设置冲突条件，请求锁则判断是否冲突并添加文件路径，释放锁则删除相应的文件路径，读操作为路径申请共享锁，写操作申请排他锁。
4. 实现细节

**架构**如下：database为server和client提供远程调用，server为client提供远程调用



实现**目录结构**如下：文件散列在不同服务器节点中，用户能够查询到所有服务器上存在文件，以及每个用户都有一个本地存储目录cache



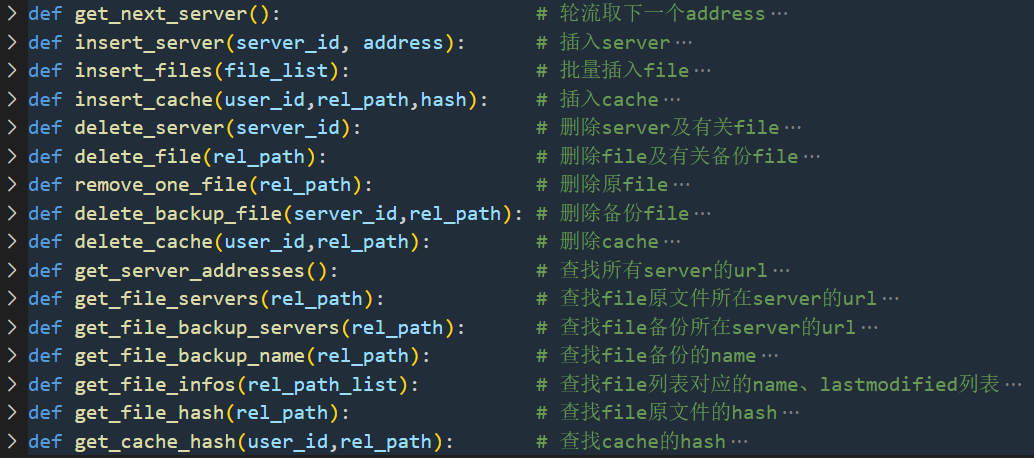
**1、Database.py初始化：**

1. 连接info.db数据库
2. SimpleXMLRPCServer创建RPC服务器，供client和server远程调用以下函数，其中data\_info选择在文件config.py中定义，通过data\_url创建服务器代理对象即可调用database.py的函数



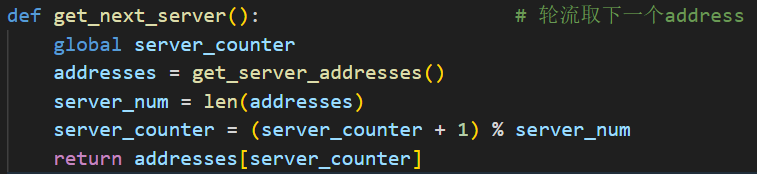


其中函数的主要功能介绍如下图展示

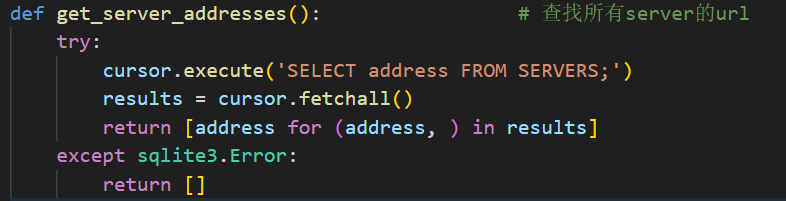


这些函数都是调用SQL语句，实现逻辑较简单，不一一展示，取有代表性进行介绍

其中get\_next\_server()主要用于轮流取server为用户提供服务



下面的get\_server\_address()获取所有server的url供用户，主要在查询所有文件时调用

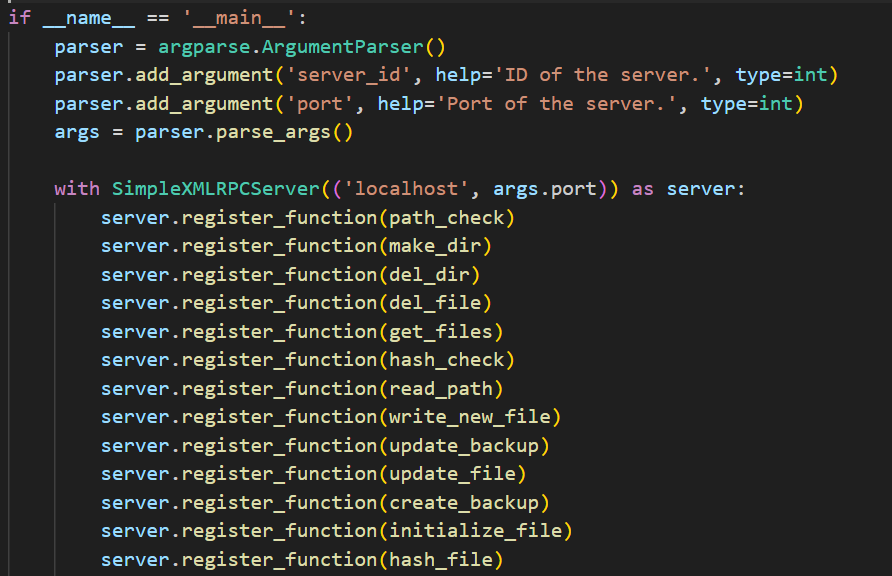


**2、server.py初始化**

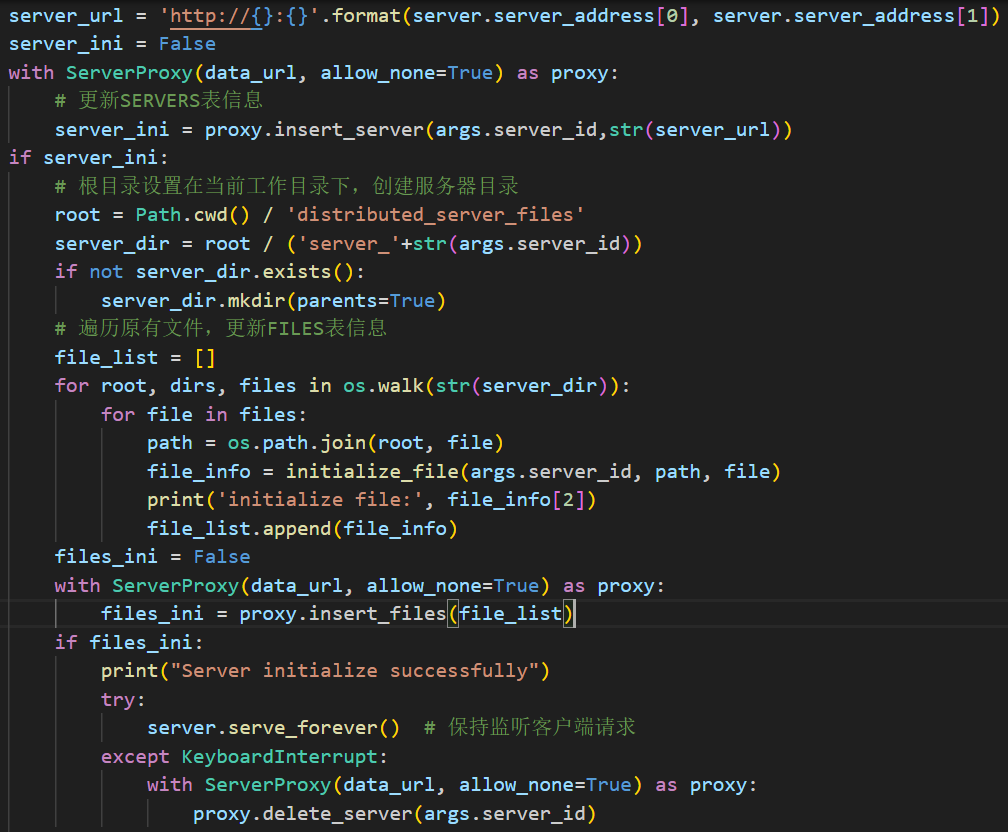
(1) 输入：调用argparse完善输入设置，要求在命令行键入运行

python server.py [server\_id] [port]

(2) SimpleXMLRPCServer创建RPC服务器，供client远程调用以下函数



1. 更新SERVERS表信息（插入该server记录）
2. 创建该服务器目录
3. 遍历该服务器目录下文件，更新FILES表信息（插入该路径下所有文件记录）



**3、client.py初始化**

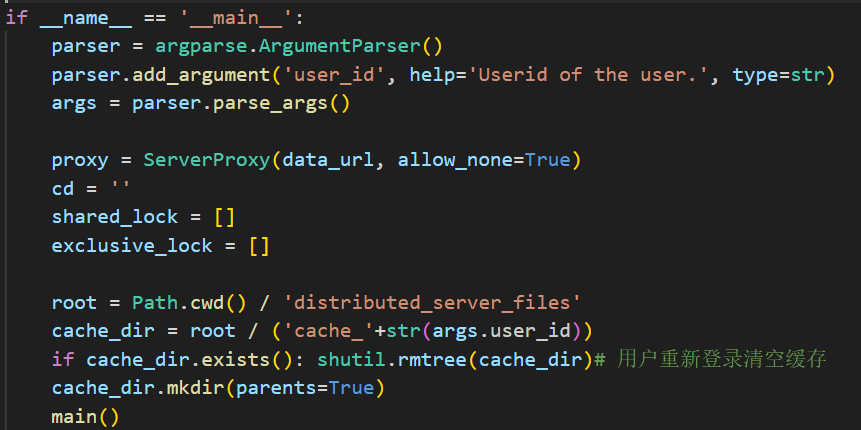
(1) 同样设置输入格式为：

python client.py [user\_id]

(2) 启动datebase.py的服务器代理

(3) 初始化一些待用参数

(4) 创建缓存目录



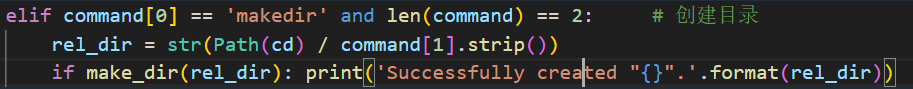
(5) 进入主函数，循环执行用户输入：



**4、文件系统功能的实现：**

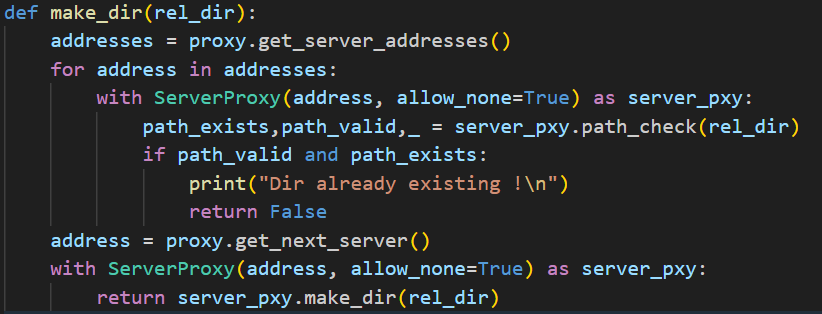
**(1) 创建目录**

用户输入makedir <dir>执行操作：执行make\_dir函数



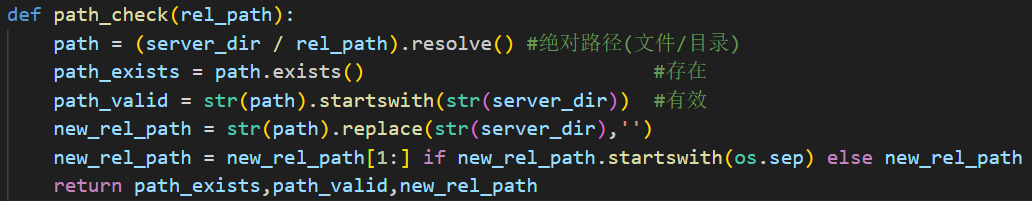
make\_dir函数实现：

* 1. 首先查询所有服务器是否存在该目录
  2. 若存在则无需创建，都不存在该目录则调用一台服务器执行mkdir操作

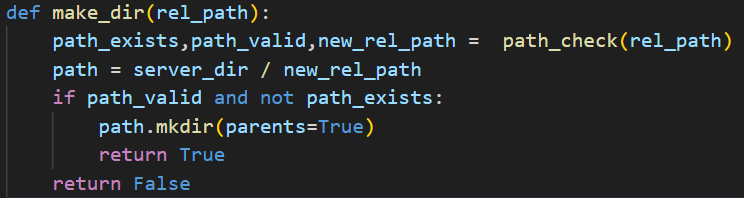


上面涉及到服务器path\_check和make\_dir函数

path\_check函数如下：返回路径”存在性”、”有效性”以及规范后的”相对路径”



server端make\_dir函数如下：检查路径是否有效和存在，执行mkdir



**(2) 列出当前目录内容**

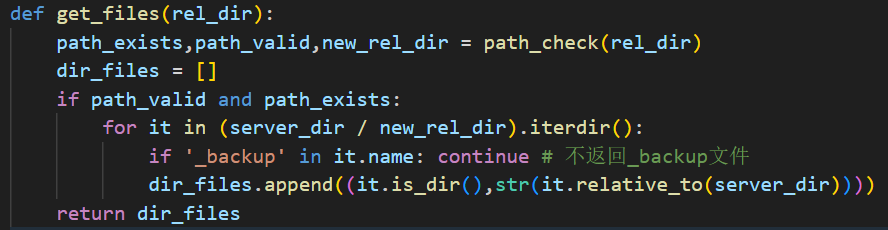
用户键入list，调用list\_file函数：

遍历每个服务器，调用get\_file函数，返回文件路径以及文件是否为目录项，根据文件路径在FILES表中搜索文件信息包括文件名和最近更新时间，打印出来



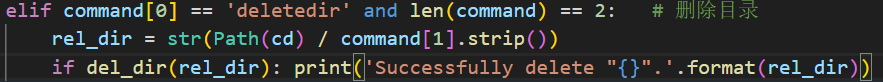
其中调用到服务端的get\_file函数，用于返回该目录下所有文件及目录的路径，但不返回备份文件(隐藏显示)：

首先检查路径有效和存在，之后通过path.iterdir()遍历目录中内容，存储返回

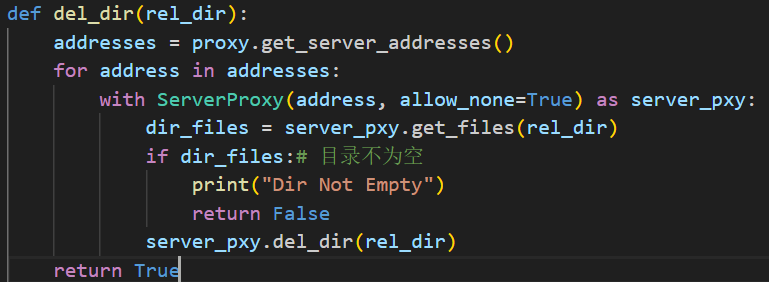


**(3) 删除目录**

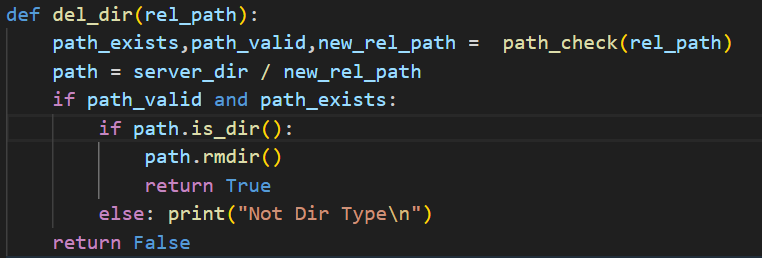
用户键入deletedir <dir>后调用del\_dir函数：



调用get\_files返回目录内容，若该目录不为空则返回不删除，空则调用服务器删除目录

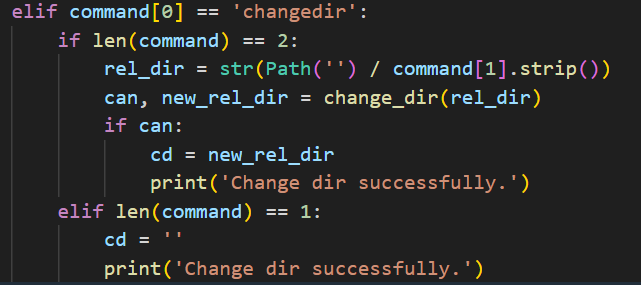


server端del\_dir函数：例行检查并判断是否为目录，执行rmdir()

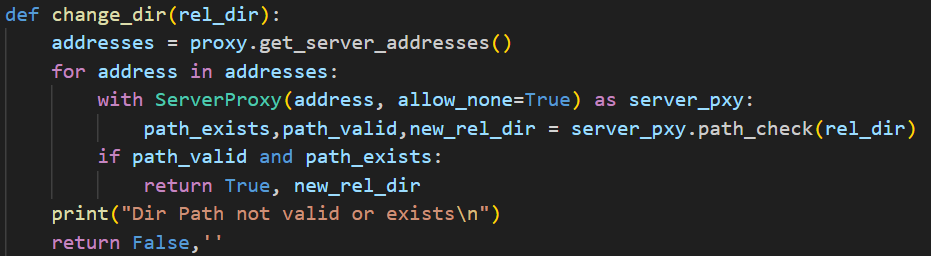


**(4) 改变当前目录**

用户键入changedir <dir-path>，如果没有<dir-path>则返回根路径，有则调用change\_dir函数判断是否可用进入该目录

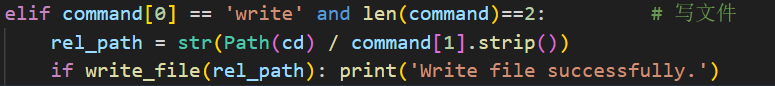


change\_dir函数：判断路径是否有效和存在即可



**(5) 写文件**

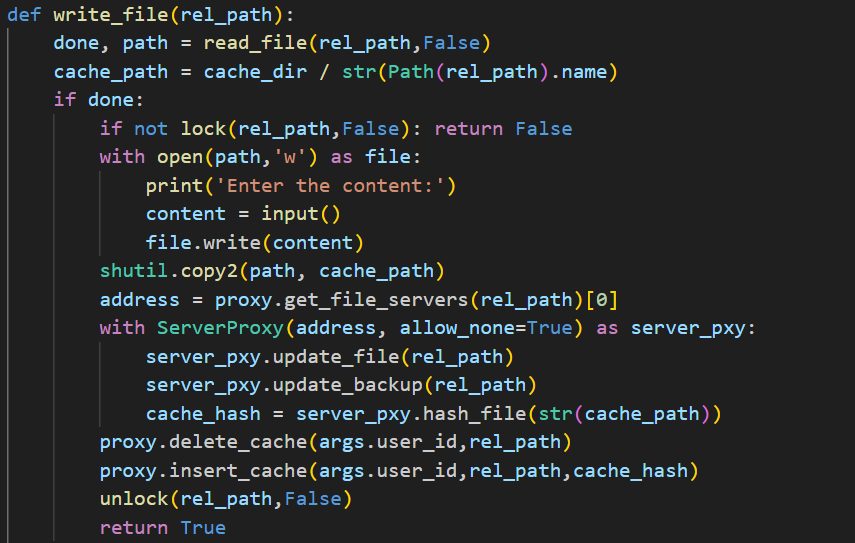
同样根据用户键入write <file>调用write\_file函数



读写操作涉及到缓存、副本一致性和并行读写问题，先看write\_file大致思路：

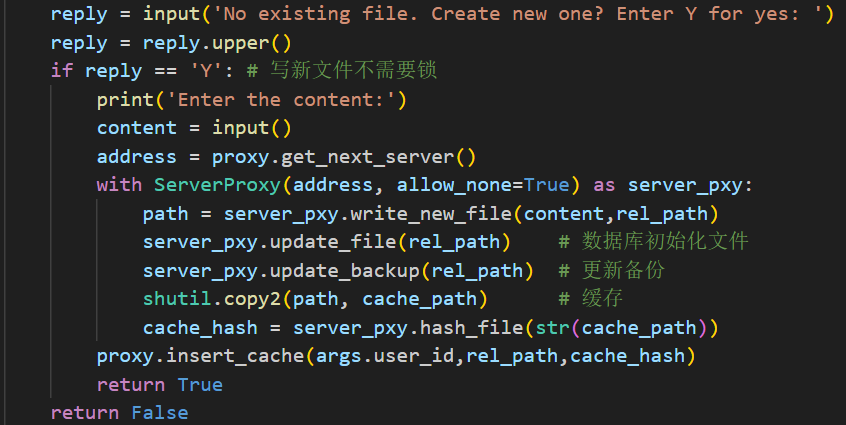
调用读文件read\_file函数(后面介绍)，若文件存在返回done=true和读文件的实际路径：

调用lock**排他锁**，根据实际路径打开文件file写入，同时复制到缓存目录中（**write through**），在原文件所在服务器中update\_file更新原文件以及update\_backup更新备份文件（**瞬时一致性**），更新CACHES表



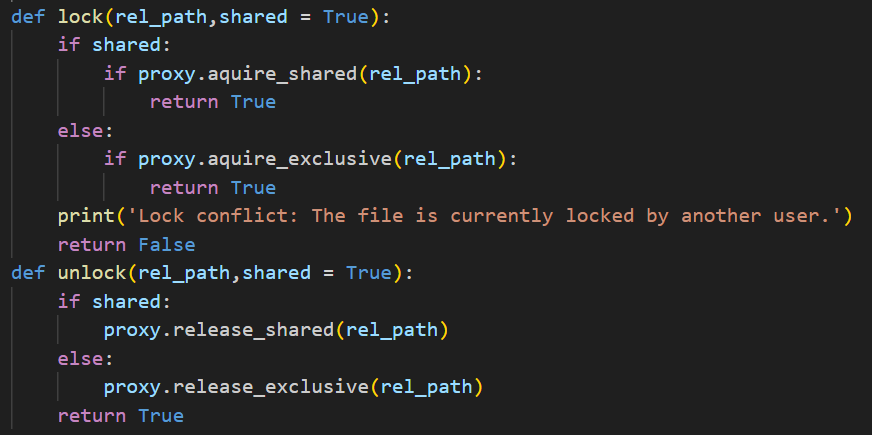
若文件不存在则直接创建新文件：

创建新文件这里不需要加锁，直接使用一台服务器执行write\_new\_file，最后同样执行update\_file、update\_backup以及缓存更新

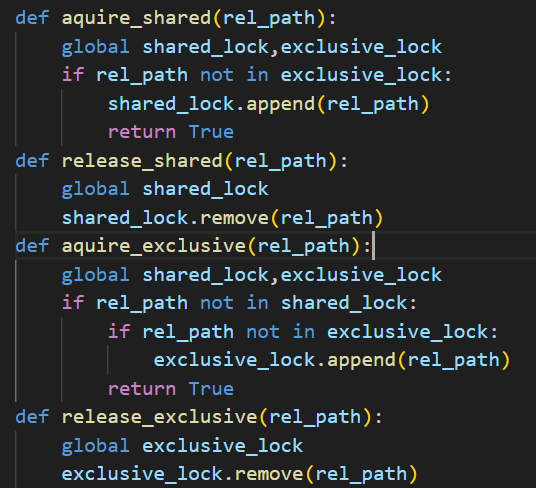
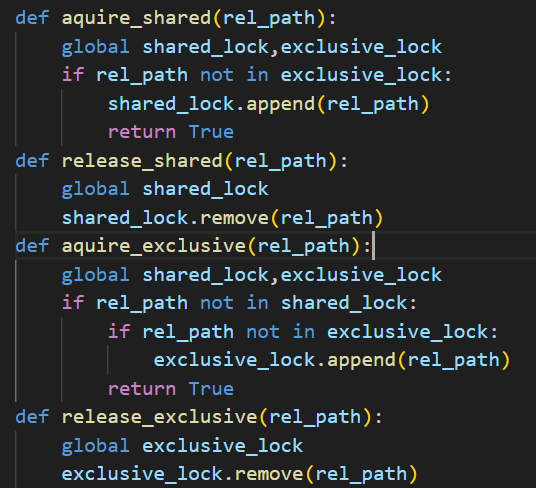


以上是服务端write\_file的思路，下面介绍其中涉及函数：lock、unlock，update\_file、update\_backup函数

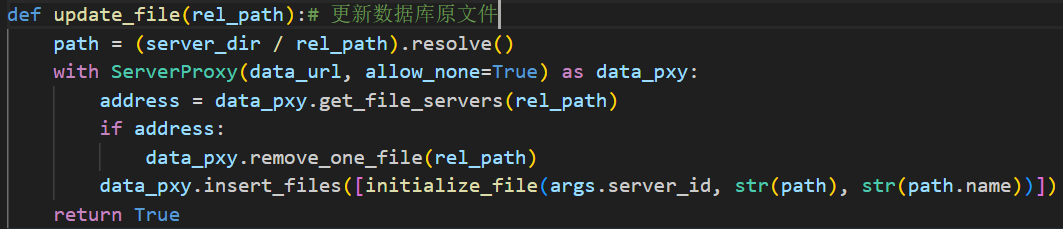
**Lock、unlock**：在database.py中通过share\_lock、exclusive\_lock表示共享锁和排他锁，分别用于读写操作，申请共享锁只需不存在申请rel\_path不存在于排他锁，申请排他锁需要该rel\_path不在共享锁和排他锁列表中，若不成立则返回冲突提醒



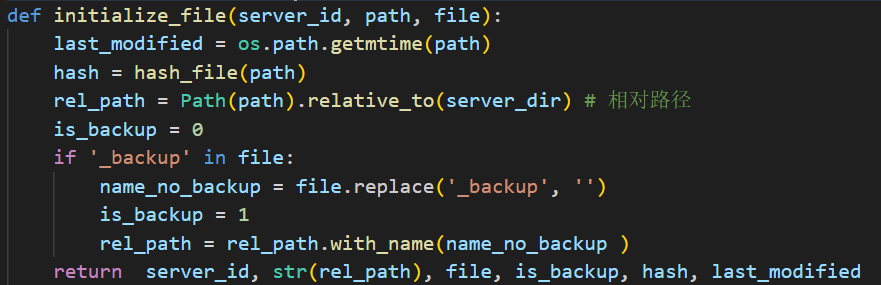
database.py中申请锁和释放锁流程如下：



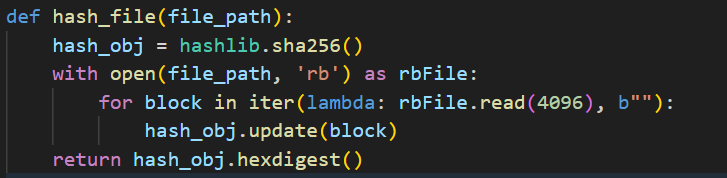
server端的**update\_file**：更新数据库中原文件的记录，存在记录则将其删除，再根据实际路径path调用initialize\_file得到文件信息统一插入数据库



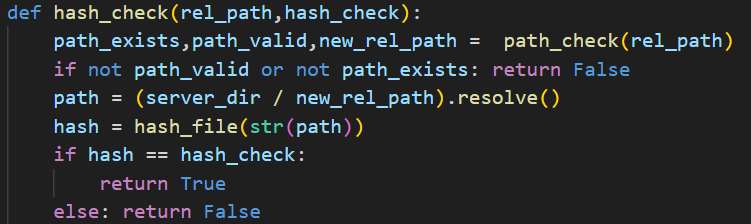
Initialize\_file函数如下：返回该文件需要插入数据库的信息



上面涉及到hash\_file函数：用于计算文件的哈希值

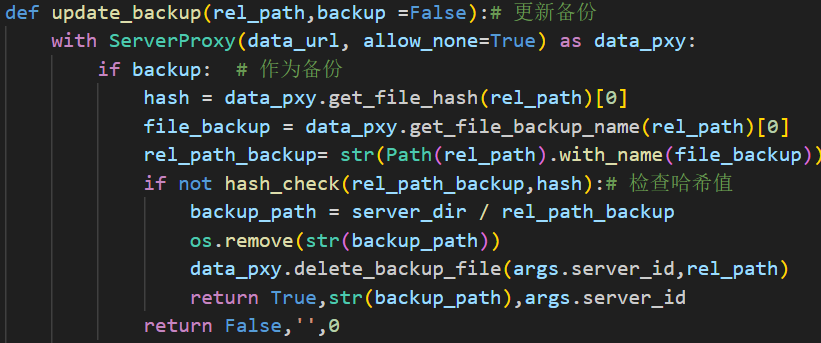


与之相关的还有hash\_check函数：比对实际路径哈希值

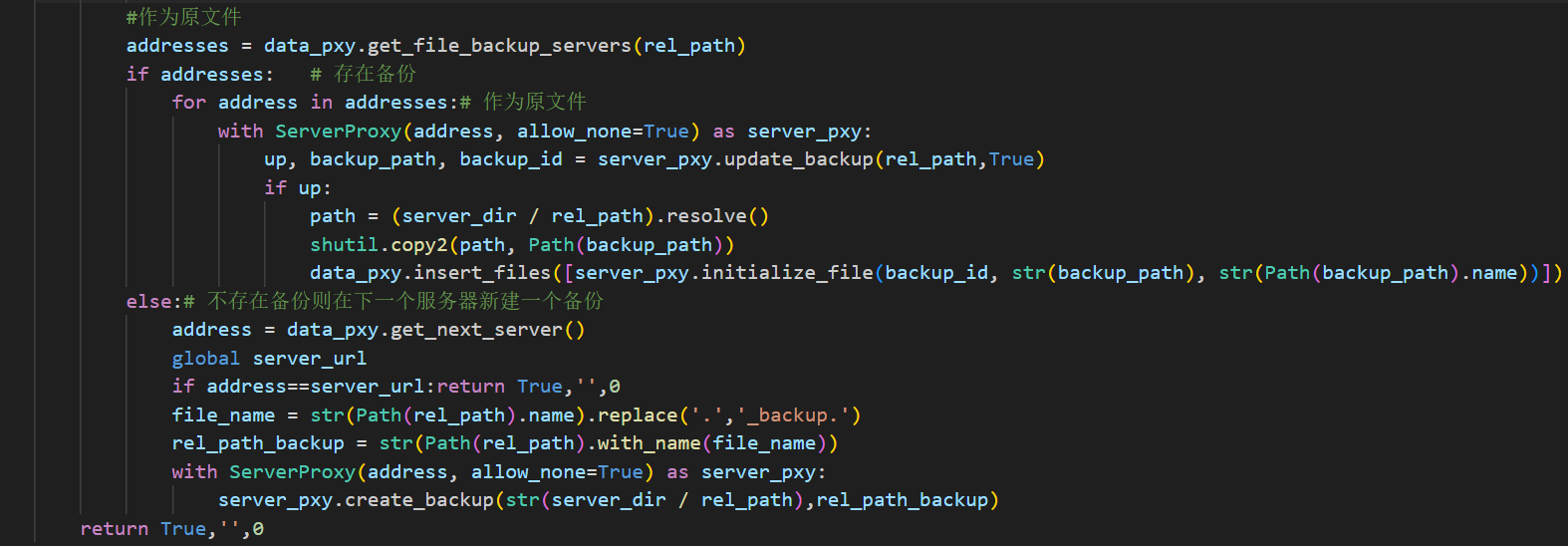


server端的**update\_backup函数**：

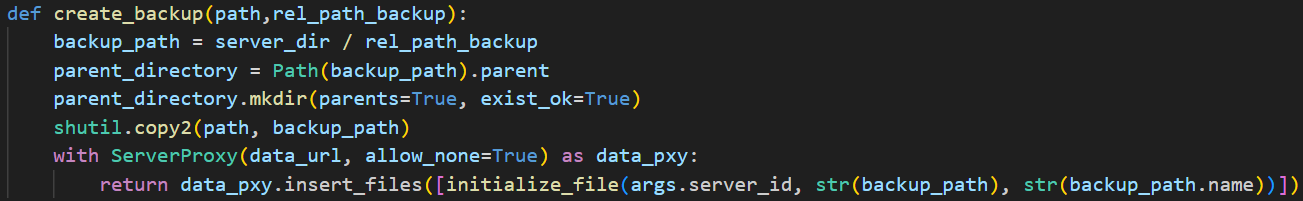
根据传入参数，backup=true判断执行服务器持有文件为备份文件，将相对路径改为备份文件的相对路径执行hash\_check函数比较目标hash值，若不相同则更新该备份文件：删除该文件返回备份文件的实际路径和所在服务器



根据backup=false判断执行服务器持有原文件，判断是否存在备份，存在则遍历备份文件所在服务器，递归执行该函数，也就是到上图的流程，如果返回备份文件需要更新，则复制原文件给备份文件，更新数据库FILES表。如果不存在备份，则在另一台服务器新建备份，调用creat\_backup新建备份文件



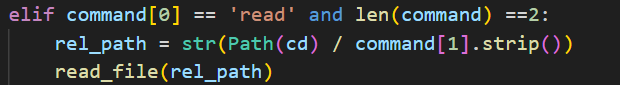
creat\_backup函数如下：根据原文件和备份文件路径进行文件复制，之后更新数据库FILES表即可。



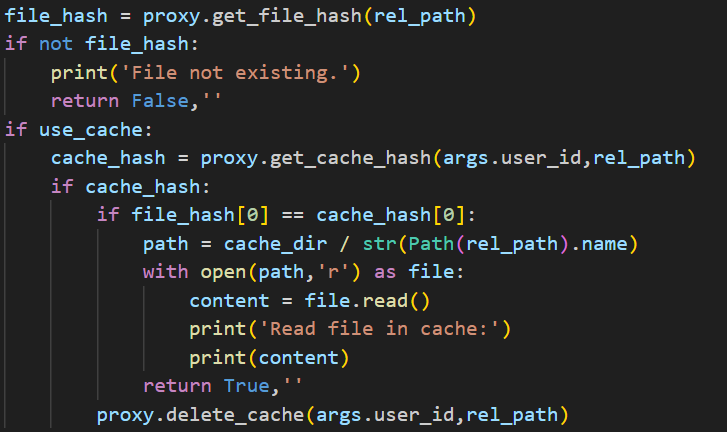
写操作涉及的函数介绍就这些，主要是锁和更新文件副本这类的操作

1. **读文件**

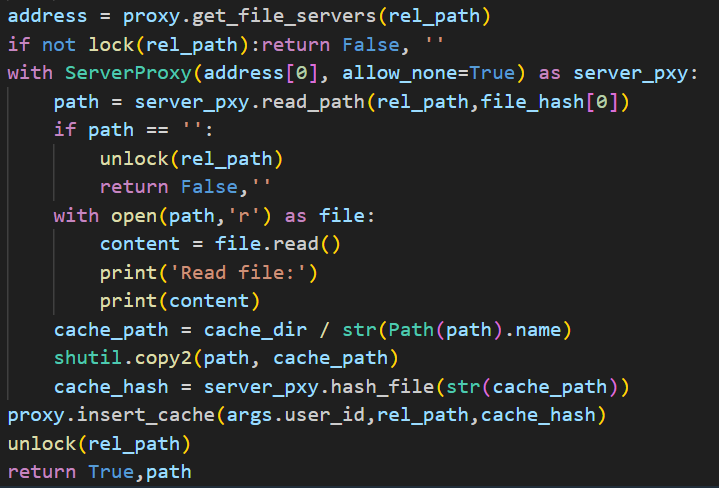
同样用户键入read <file>即可通过read\_file进行读操作



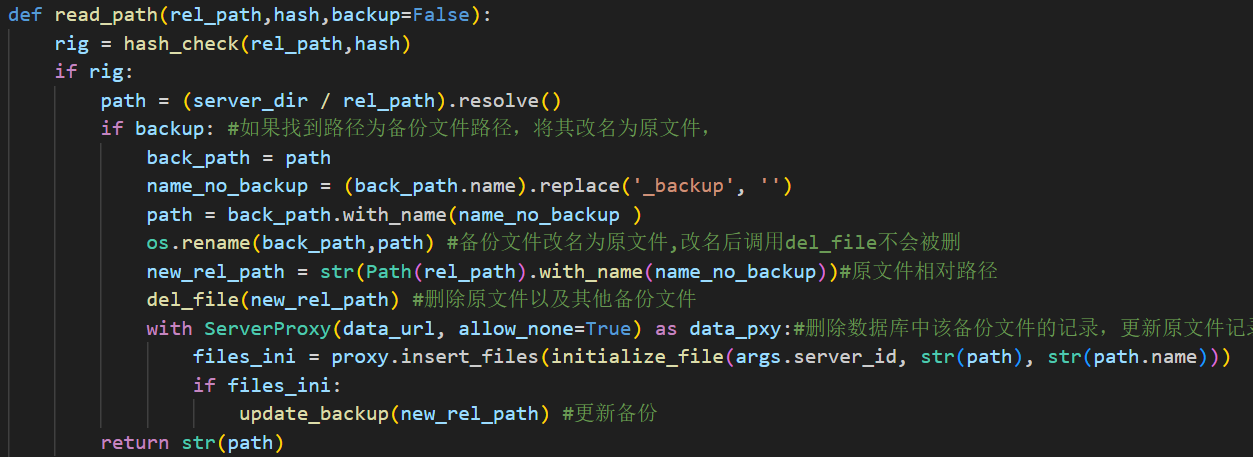
Read\_file函数：先从本地缓存目录中搜索是否存在该路径文件，有则搜出缓存hash，同原文件的hash进行比较，若相同则直接读取缓存，且无需使用锁，不相同则删除CACHES表中该缓存记录



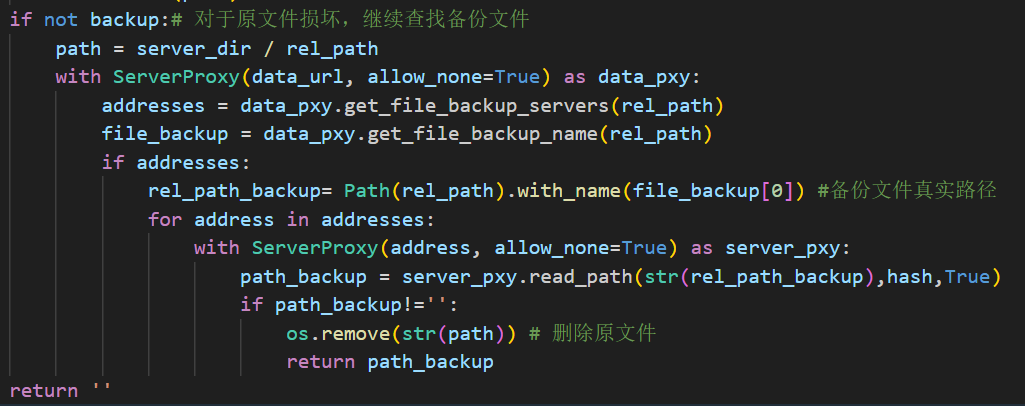
不存在缓存则找到原文件所在服务器，使用lock共享锁，之后执行read\_path函数返回文件绝对路径，读取文件，更新缓存，使用unlock释放锁即可



server端read\_path函数：与update\_backup类似，使用递归调用，检查该路径与目标hash是否一致，一致则如下情况，backup=false判定为原文件所在服务器操作，直接返回文件实际路径，如果backup=true则说明原文件有误，需要将该备份文件作为新的原文件，首先改名为原文件(去掉后缀:\_backup)，删除原文件及其他备份文件记录，更新FILES表将其作为原文件，并执行update\_backup新建备份文件

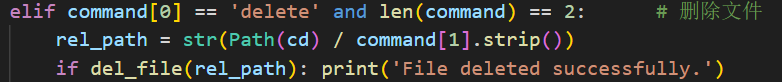


对于hash不匹配，如果是备份文件直接返回’’，若为原文件则遍历备份文件，如果得到hash匹配的备份文件，则删除原文件自己，返回备份文件实际路径

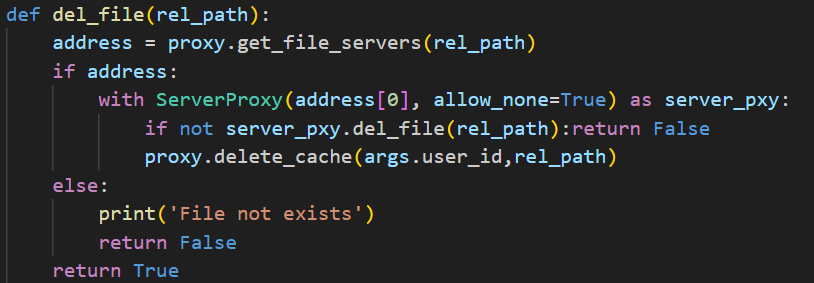


1. **删除文件**

同理键入delete <file>调用del\_file函数进行文件删除

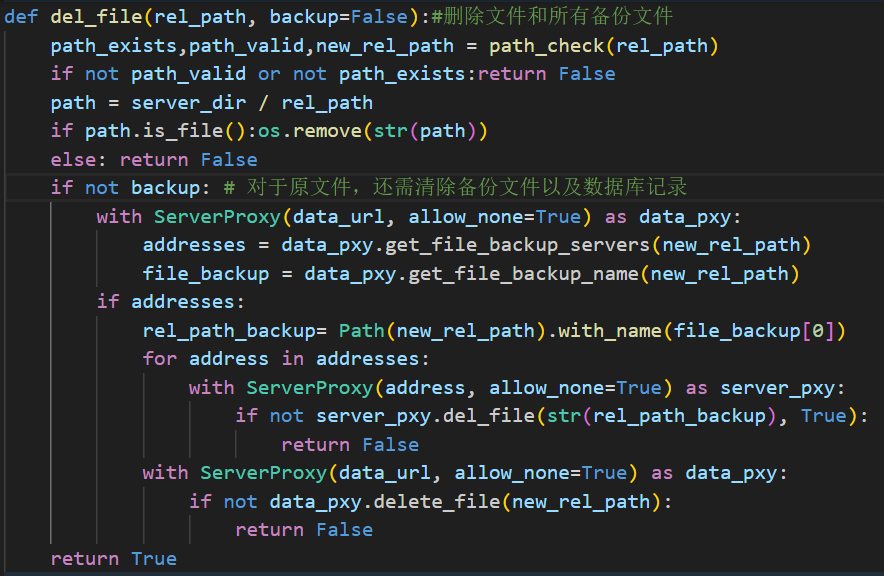


Del\_file函数：查看文件路径是否存在，存在则调用server端del\_file函数，删除成功去除CACHES表相应缓存记录



server端del\_file函数：

检查路径有效且存在，以及是否为文件，使用os.remove删除，对于原文件，还需要删除备份文件以及数据库FILES的相关记录

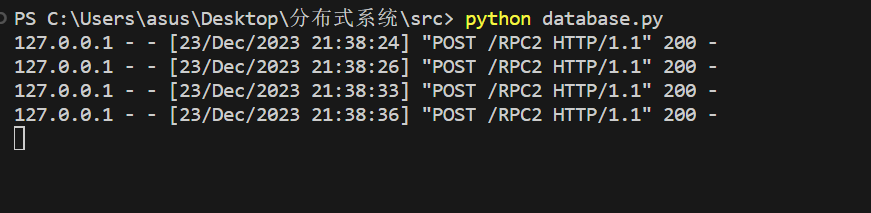


至此，所有操作执行函数介绍完毕

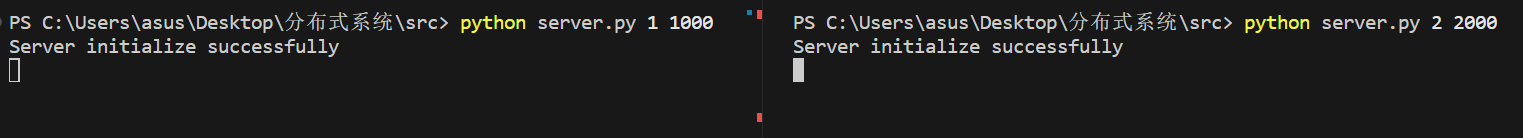
1. 运行情况

首先执行代码：

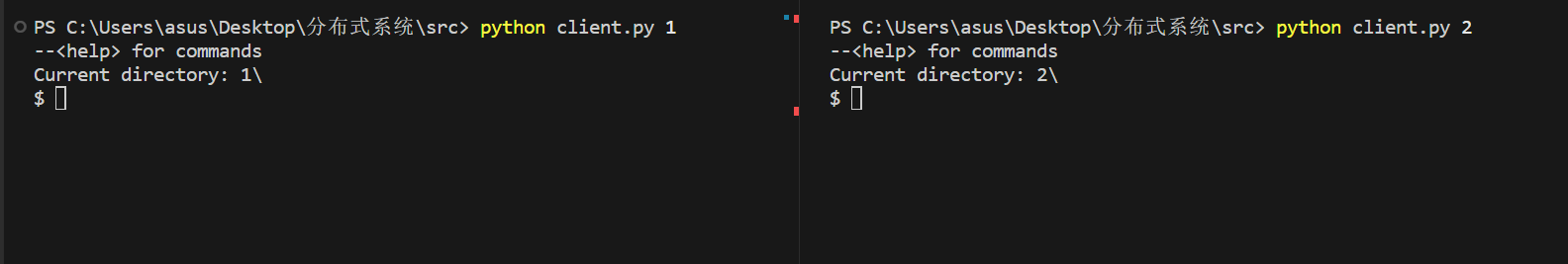
database.py：



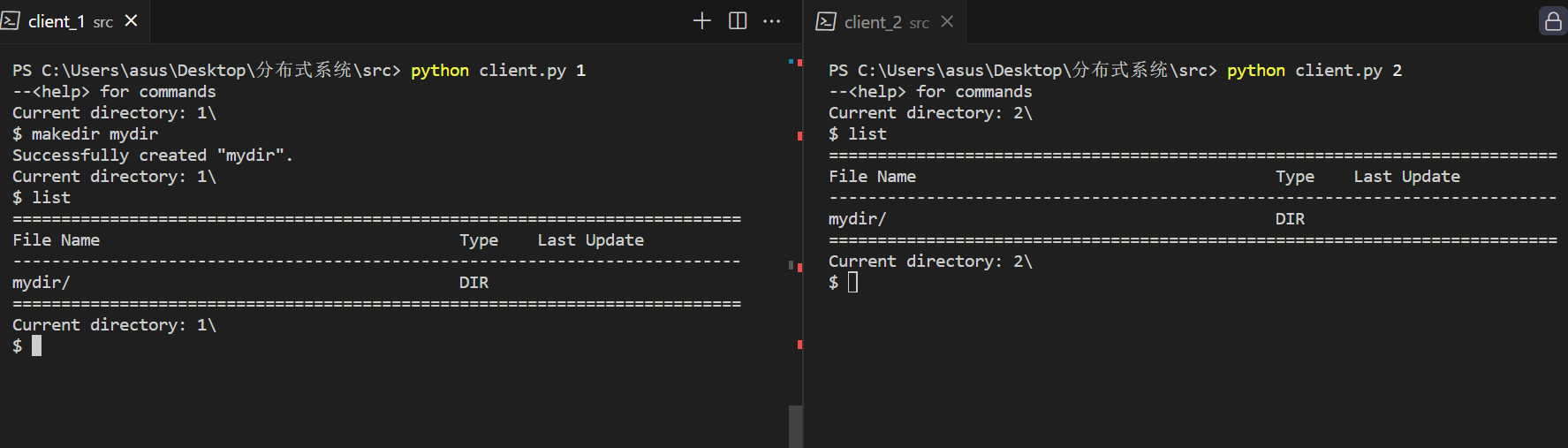
server.py:打开两个服务器测试



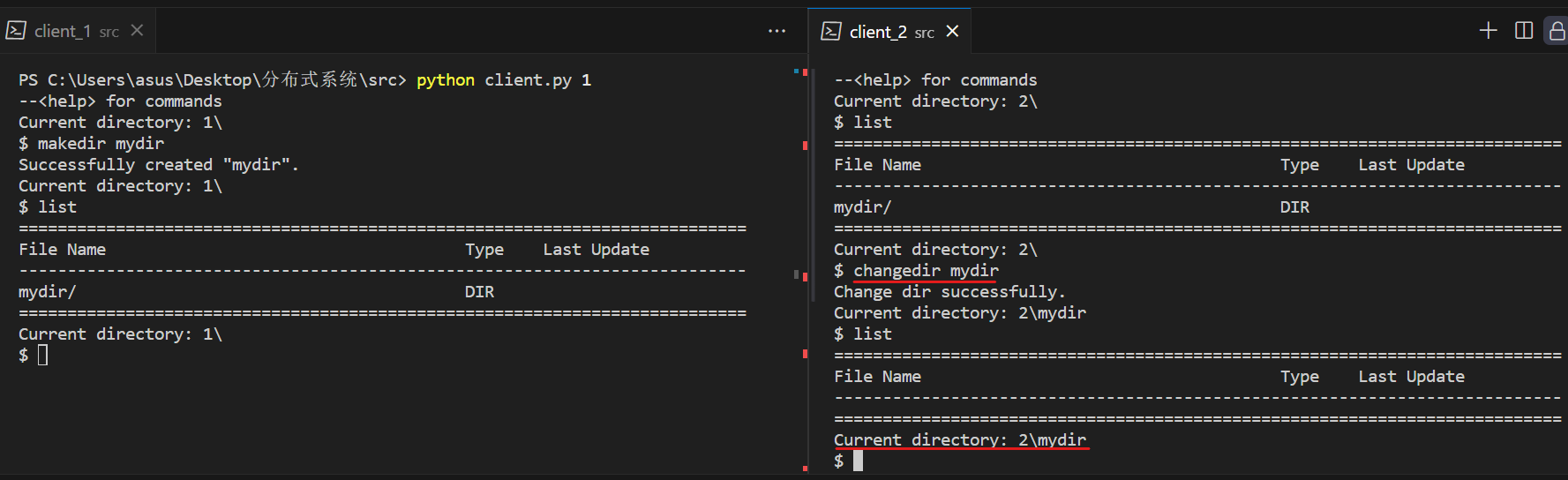
client.py:打开两个用户测试



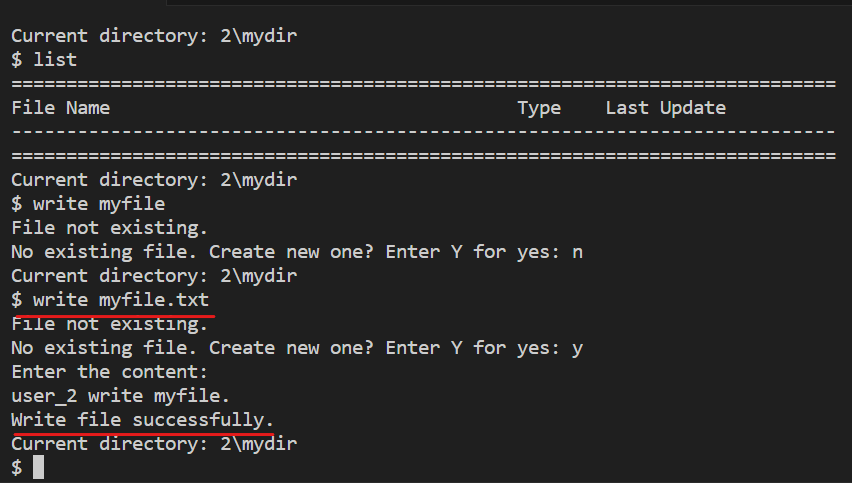
在用户1使用makedir mydir指令：



在用户2使用changedir mydir指令：



在用户2所在文件夹写文件

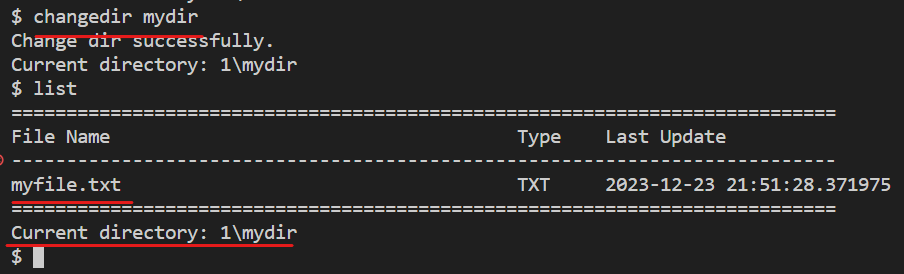


写入成功，查看文件夹：在server\_2创建原文件，server\_1创建了副本

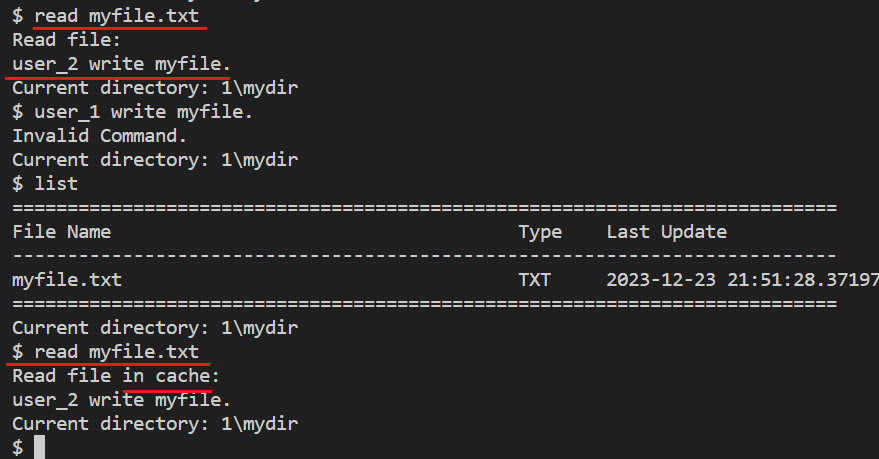




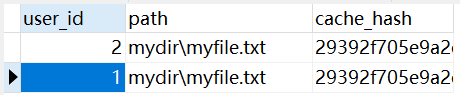
用户1查看文件：



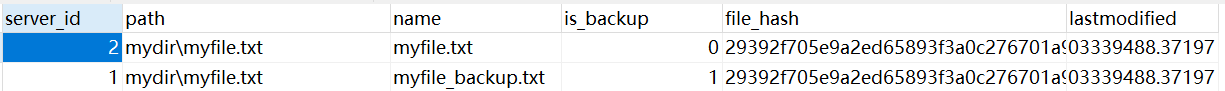
用户1 read两次文件：可看到第二次read时使用的时cache



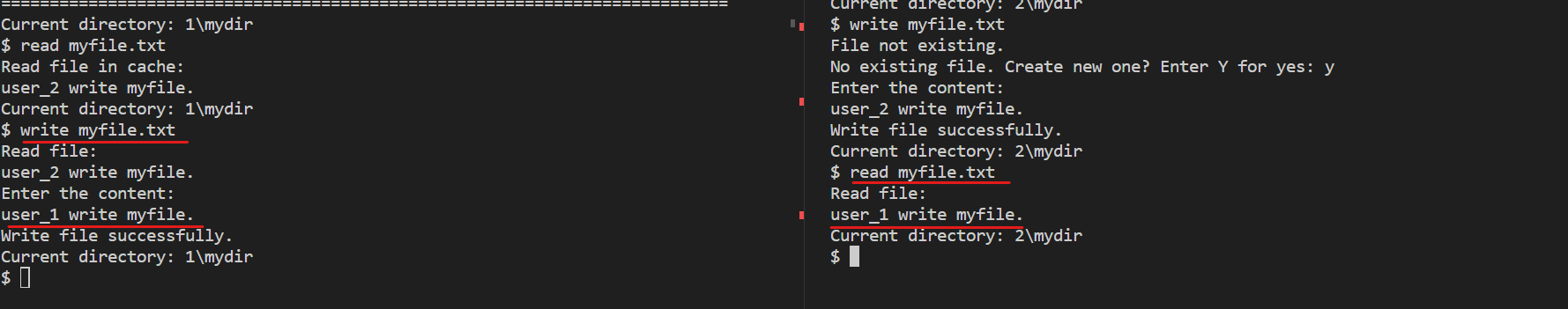
此时CACHES表：



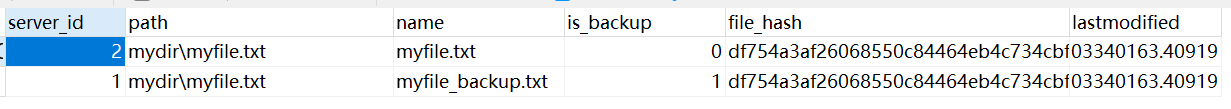
FILES表：



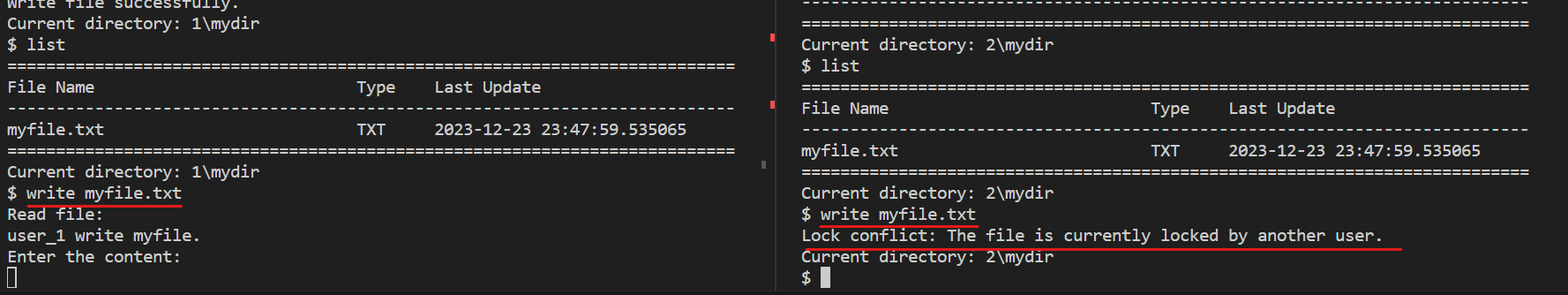
用户1 重新write该文件，用户2读该文件：



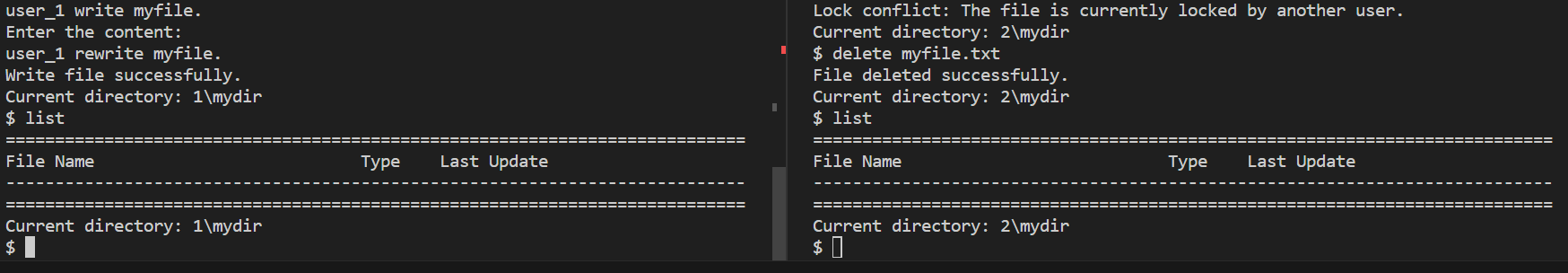
FILES表：原文件和备份的file\_hash都得到更新：

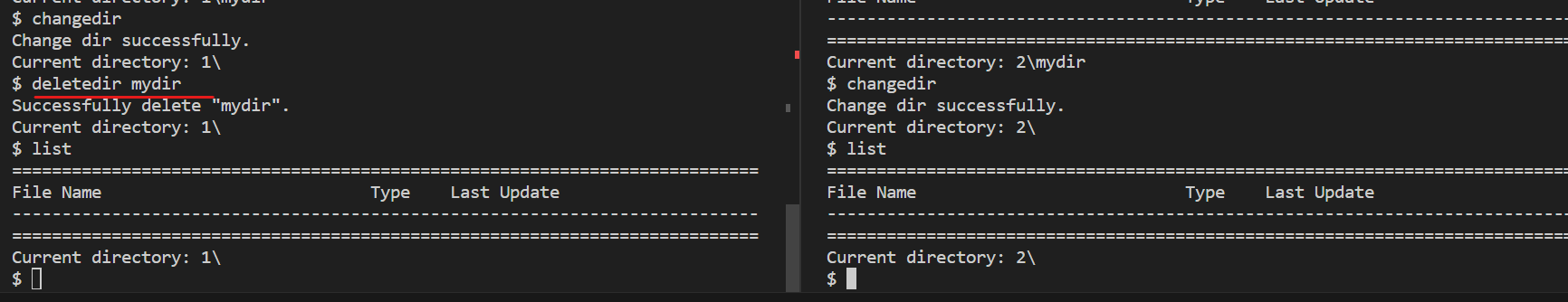


验证并行写情况：可看到在用户1写入时用户2无法对相同文件进行写操作



最后删除文件并删除文件夹：





所有功能测试均正常。

1. 遇到的问题

主要在于学习使用RPC通信方式，以及利用RPC通信方式远程调用函数时，必须要有返回值。主要的实现困难在于考虑多台服务器之间的备份更新时，调用不同的server.py函数容易产生混淆和传递值的难题。以及实现文件锁时，起初考虑使用Msvcrt方法，但因为该方法在把文件上锁之后，客户自己也没法操作上锁的文件，这一点仍存在疑惑，后来通过自己维护共享锁和排他锁的方法解决问题。

1. 总结

本次作业完成了分布式文件系统，从参考到编写代码再到找bug的过程花了很长时间，最后只实现了基本的功能，且十分粗糙，在提升执行速度上还有很大提升空间。通过这次课程设计，也体会到分布式文件系统的实现需要考虑比想象中要多的细节，更加理解分布式系统之间分布、透明、同一性的特点，对这门课有了更深的感悟。