分布式系统 大作业

姓名:杨子昂 学号:21307181

一.实验题目

设计并实现一个分布式键值（key-value）存储系统，可以是基于磁盘的存储系统，也可以是基于内存的存储系统，可以是主从结构的集中式分布式系统，也可以是P2P式的非集中式分布式系统。能够完成基本的读、写、删除等功能，支持缓存、多用户和数据一致性保证,

二.实验内容

1.实验要求

要求 1）必须是分布式的键值存储系统，至少在两个节点或者两个进程中测试； 2) 可以是集中式的也可以是非集中式； 3) 能够完成基本的操作如：PUT、GET、DEL等； 4) 支持多用户同时操作； 5) 至少实现一种面向客户的一致性如单调写； 6) 需要完整的功能测试用例； 7) 涉及到节点通信时须采用RPC机制； 8) 提交源码和报告，压缩后命名方式为：学号_姓名_班级 加分项： 1）具备性能优化措施如cache等； 2）具备失效容错方法如：Paxos、Raft等； 3）具备安全防护功能； 4）其他高级功能；

2.实验原理

在本次实验中我实现了分布式键值系统。具体采用的是CS架构，也就是服务器--客户端架构的形式实现，并且用RPC作为客户端和服务器之间的通信方式。

远程过程调用（RPC）：

是一种协议或技术，用于在不同的计算机之间执行代码。在RPC中，一个程序可以执行不在其地址空间的程序代码，就像这些代码是本地的一样。RPC原理可以分为以下几个关键步骤：

1. **过程调用请求**：客户端程序调用一个执行在远程服务器上的过程（函数）。这通常涉及到对函数的普通调用，但是这个函数并不在同一个本地系统上。
2. **参数传递**：客户端的RPC库负责收集过程调用的参数，并将其通过网络发送到服务器。这个过程称为“参数打包”或“序列化”。
3. **网络通信**：客户端的RPC请求通过网络发送给服务器。服务器监听来自客户端的请求。
4. **服务器端处理**：一旦服务器接收到请求，它将解包或反序列化请求中的参数，并执行本地过程调用。
5. **结果返回**：执行完成后，服务器会将结果“打包”或“序列化”，然后通过网络发送回客户端。
6. **客户端接收结果**：客户端的RPC系统接收到来自服务器的响应，解包数据，并将其作为函数调用的结果返回给客户端程序。

RPC的关键在于它使远程过程调用看起来像本地调用一样，从而简化了分布式系统的开发。在本次实验中我选择的是rpyc这个主要应用于python的RPC库，它可以直线客户端和服务器之间的实时连接，并且通过函数映射的方式将服务器端的函数和客户端的函数进行对应和参数传递，从而实现了在客户端远程调用服务器端的函数。

键值存储的后端部分是借助python中的sqlitedict来实现的，是一个轻量化的数据库，对于键值类型的数据有非常高效的操作能力。

3.代码展示

实验环境：

1. python 3.7.16
2. rpyc(RPC通信)
3. sqlitedic(后端数据库)
4. logging(生成日志)
5. uuid(生成不重复的uid)

```
import sys

def write_user_info(username, password):
    with open('user.txt', 'a') as file:
```

```
        file.write(f'{username} ')
        file.write(f'{password}\n')

def main():
    if len(sys.argv) != 3:
        print("Usage: python sudo.py <username> <password>")
        sys.exit(1)

    username = sys.argv[1]
    password = sys.argv[2]
    write_user_info(username, password)
    print("User information saved to user.txt")

if __name__ == "__main__":
    main()
```

sudo.py是用来管理用户信息的，主要功能是向其中添加用户名 `username`以及密码 `password`, 由于这并不是整个键值存储系统的主要部分，故只实现了保证测试视频中添加用户的功能，并未涉及到增删改查user信息的功能，当然这些实现也比较简单而且并不是重点。（甚至可以用键值存储系统来实现用户信息管理）

功能是在命令行输入 `python sudo.py <username> <password>`这样格式的文字即可向user.txt文档写入新的用户名和密码。

```
import rpyc
import logging
import uuid

class KeyValueClient:
    def __init__(self, host, port, log_file, username):
        self.client_id = str(uuid.uuid4())
        self.username=username
        self.conn = rpyc.connect(host, port, config={"allow_public_attrs":
True})
        self.service = self.conn.root
        # 使用RPC方法设置client_id
        logging.basicConfig(filename=log_file, level=logging.INFO,
                            format='%(asctime)s %(levelname)s:%(message)s')
        self.login()
        self.run_command_loop()

    def put(self, key, value):
        result = self.service.put(key, value)
        logging.info(f"Client {self.username}: Put key: {key}, value:
{value}, result: {result}")
        return result

    def get(self, key):
        value = self.service.get(key)
        logging.info(f"Client {self.username}: Get key: {key}, returned
value: {value}")
```

```
        return value

    def delete(self, key):
        result = self.service.delete(key)
        logging.info(f"Client {self.username}: Delete key: {key}, result:
{result}")
        return result

    def get_all(self):
        all_data = self.service.getall()
        logging.info(f"Client {self.username}: Get all data")
        return all_data

    def delete_all(self):
        result = self.service.delall()
        logging.info(f"Client {self.username}: Delete all data, result:
{result}")
        return result

    def find_user_logs(self):
        with open('client_log.log', 'r') as file:
            for line in file:
                if f"Client {self.username}:" in line:
                    print(line.strip())

    def login(self):
        self.service.login(self.username)

    def logout(self):
        self.service.logout(self.username)

    def run_command_loop(self):
        print(f"Client ID: {self.username}")
        while True:
            try:
                command = input("Enter command
(put/get/delete/getall/delall/getlog): ").split()
                if command[0] == 'put' and len(command) == 3:
                    self.put(command[1], command[2])
                elif command[0] == 'get' and len(command) == 2:
                    print(self.get(command[1]))
                elif command[0] == 'delete' and len(command) == 2:
                    self.delete(command[1])
                elif command[0] == 'getall':
                    print(self.get_all())
                elif command[0] == 'delall':
                    self.delete_all()
                elif command[0] == 'getlog':
                    self.find_user_logs()
                else:
                    print("Invalid command or arguments")
            except KeyboardInterrupt:
                self.logout()
                print("Exiting client")
                break
            except Exception as e:
                print(f"An error occurred: {e}")
```

```
if __name__ == "__main__":
    users = {}
    with open('user.txt') as file:
        for i in file.readlines():
            i = i.split()
            users[i[0]] = i[1]

    #用户登录
    username = input('Please input your username:')
    password = input('Please input your password:')
    if username in users.keys() and users[username] == password:
        client = KeyValueClient("localhost", 18812, "client_log.log",
username)
    else:
        print('Your username or password has something wrong.Please try
again!')
```

KeyValueClient 类:

- **init 方法**: 类的初始化方法。它接受主机地址、端口、日志文件路径和用户名作为参数。客户端ID通过 `uuid.uuid4()` 生成，确保每个客户端实例有唯一的ID。此外，它还建立了与远程服务的连接，并初始化了日志记录。
- **put、get、delete、get_all 和 delete_all 方法:** 这些方法分别对应于键值存储的基本操作，它们通过RPC调用远程服务的相应方法，并记录操作日志。
- **find_user_logs 方法:** 从日志文件中检索并打印与当前用户相关的日志记录。
- **login 和 logout 方法:** 通过RPC调用远程服务来处理用户登录和注销。
- **run_command_loop 方法:** 这是一个交互式命令循环，允许用户通过命令行输入执行不同的操作。

主程序:

1. 从 user.txt 文件中读取用户信息，并存储在字典 users 中。
2. 请求用户输入用户名和密码。
3. 验证用户名和密码。如果认证通过，则创建 KeyValueClient 实例并连接到服务端；否则，提示错误信息。

这段代码的核心功能是通过RPC技术实现客户端与服务端的通信，使得客户端可以远程执行键值存储的操作。它同时也提供了一个基本的用户认证机制和命令行界面，使得用户可以直接通过命令与远程服务进行交互。

```python
import rpyc
from sqlitedict import SqliteDict
from rpyc.utils.server import ThreadedServer
import datetime

class KeyValueService(rpyc.Service):
    def __init__(self):
        # 初始化数据库
        self.db = SqliteDict('./my_db.sqlite', autocommit=True)

    clients = {}

    def exposed_login(self, username):
        current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S,%f')[:-3]
        print(f"{current_time} client: {username} logged in")

    def exposed_logout(self, username):
        current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S,%f')[:-3]
        print(f"{current_time} client: {username} logged out")

    def on_connect(self, conn):
        pass

    def on_disconnect(self, conn):
        pass

    def exposed_set_client_id(self, client_id):
        # 设置客户端ID
        self.clients[self.conn] = client_id
        print(f"User: {client_id} is connected")

    def exposed_put(self, key, value):
        try:
            self.db[key] = value
            return True
        except Exception as e:
            # 在异常情况下返回错误信息
            return str(e)

    def exposed_get(self, key):
        try:
            return self.db.get(key, None)
        except Exception as e:
            return str(e)

    def exposed_delete(self, key):
        try:
            if key in self.db:
                del self.db[key]
                return True
            return False
        except Exception as e:
            return str(e)
```

```
    def exposed_getall(self):
        try:
            return dict(self.db)
        except Exception as e:
            return str(e)

    def exposed_delall(self):
        try:
            self.db.clear()
            return True
        except Exception as e:
            return str(e)

if __name__ == "__main__":
    # 启动服务器
    server = ThreadedServer(KeyValueService, port=18812)
    print("Server is working now!")
    server.start()
```

KeyValueService 类:

- 类继承自 rpyc.Service，这是创建RPC服务的基础。
- ***init*_ 方法:** 初始化时创建一个 SqliteDict 实例，用于数据存储。
- **exposed_login 和 exposed_logout 方法:** 用于记录客户端登录和登出的时间。
- **on_connect 和 on_disconnect 方法: **这些方法可以在客户端连接或断开时执行操作，目前它们是空的。
- **exposed_set_client_id 方法:** 允许客户端设置其ID，这些ID被存储在 clients 字典中。
- **exposed_put、exposed_get、exposed_delete、exposed_getall 和 exposed_delall 方法:** 这些方法提供了键值对存储的基本功能，包括增加、获取、删除单个键值对，获取所有键值对，以及删除所有键值对。

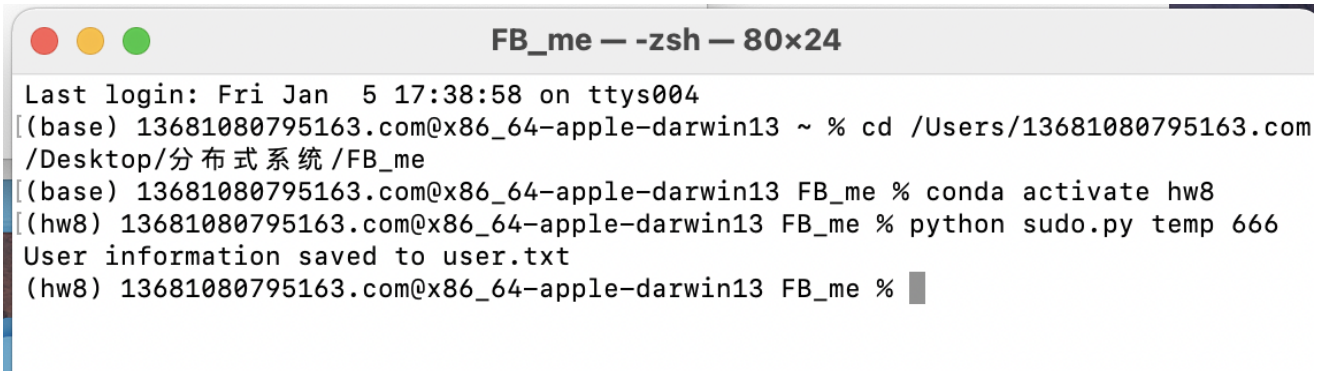
主程序:

1. 创建一个 ThreadedServer 实例，指定 KeyValueService 作为服务类，并设置监听端口为 18812。
2. 启动服务器，等待客户端连接。 这个程序的主要功能是通过RPC（远程过程调用）技术实现了一个基本的键值对存储服务。客户端可以通过远程调用来执行数据的添加、检索和删除等操作。它利用 `sqlitedict` 库将数据持久化保存在SQLite数据库中，确保数据的稳定存储和管理。此外，由于服务器采用了多线程处理机制，它能够同时处理多个客户端的连接和请求。这种服务适合于那些需要键值存储功能的应用场景。

三.实验测试结果

在实验测试环节，我将运行三个client程序和一个server程序以此来模拟多用户的使用场景。

1. 首先是测试sudo.py 这是运行前的user.txt文件的内容，一会儿我将会使用这三个名称和密码去进行client和server的测试。 sudo.py的作用是向其中写入用户名和密码进行注册



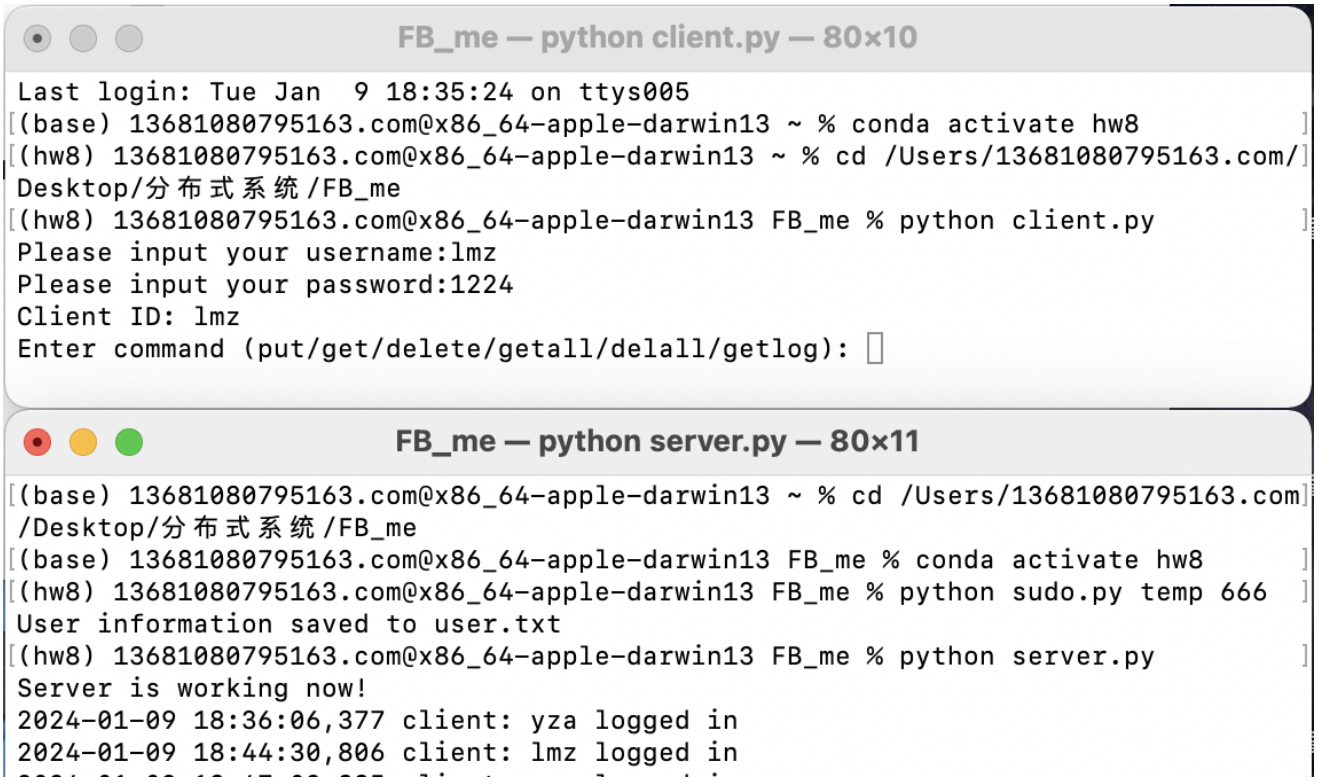
可以看到成功将新的用户名temp和密码666写入user.txt

2. 接下来我们用前三个用户名和密码登录client端，并且连到同一个server上

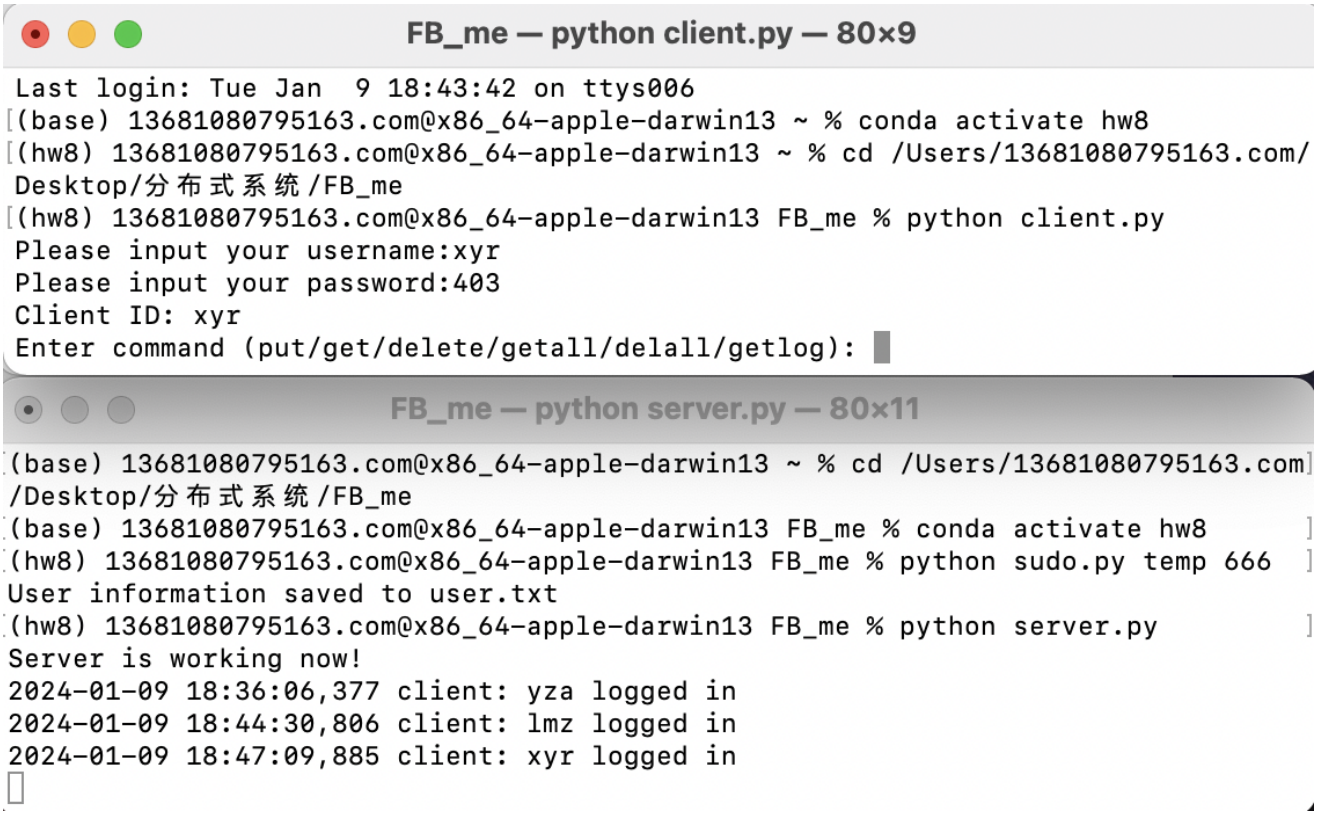
```
(hw8) 13681080795163.com@x86_64-apple-darwin13 FB_me % python server.py
Server is working now!
```

可以看到server已经在运行，然后我们登陆三个客户端并连接到服务器。

这是用户yza登陆的界面，可以看到客户端登录成功后在服务器端有一个登录成功的提示。



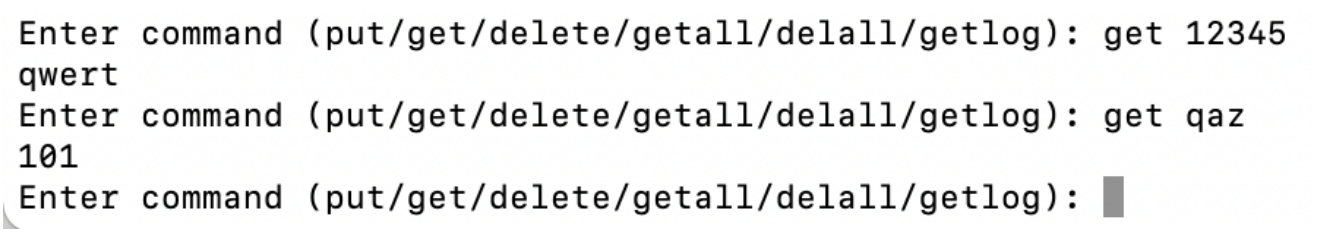
这是用户lmz登陆的界面，可以看到客户端登录成功后在服务器端有一个登录成功的提示。



这是用户xyr登陆的界面，可以看到客户端登录成功后在服务器端有一个登录成功的提示。

3. 接下来我们测试功能是否正常，我将在yza客户端上进行测试，在输入键值阶段结束后我会在lmz和xyr客户端进行查看，验证整体的一致性。

在这张图上我首先查看已有键值 `getall`并尽情全部清除 `delall`，然后我测试了三次 `put`功能，分别是单个字符，字符串，和数字搭配字符。并调用getall一次性查看了所有键值。当然我们也可以使用 `get`查看单个键值对



4. 检查多客户端一致性，我们将从lmz客户端和xyr客户端来查看yza客户端对数据的更改是否全局可见 lmz客户端

```
FB_me — python client.py — 80×10
Please input your username:lmz
Please input your password:1224
Client ID: lmz
Enter command (put/get/delete/getall/delall/getlog): get a
1
Enter command (put/get/delete/getall/delall/getlog): get all
None
Enter command (put/get/delete/getall/delall/getlog): getall
{'a': '1', 'qaz': '101', '12345': 'qwert'}
Enter command (put/get/delete/getall/delall/getlog):
```

xyr客户端

```
FB_me — python client.py — 80×9
(hw8) 13681080795163.com@x86_64-apple-darwin13 FB_me % python client.py
Please input your username:xyr
Please input your password:403
Client ID: xyr
Enter command (put/get/delete/getall/delall/getlog): get 12345
qwert
Enter command (put/get/delete/getall/delall/getlog): getall
{'a': '1', 'qaz': '101', '12345': 'qwert'}
Enter command (put/get/delete/getall/delall/getlog):
```

通过上面的检查可以证明yza客户端的操作是保证了一致性的。

5. 日志功能：我在上面的基本操作之上实现了日志功能，可以记录全局的日志操作，并且每个客户端只能查看自己的日志记录，无法查看别人的日志记录。 qaz客户端的日志记录

```
2024-01-09 18:49:57,975 INFO:Client yza: Get all data
2024-01-09 18:50:01,235 INFO:Client yza: Delete all data, result: True
2024-01-09 18:50:09,548 INFO:Client yza: Put key: a, value: 1, result: True
2024-01-09 18:50:17,978 INFO:Client yza: Put key: qaz, value: 101, result: True
2024-01-09 18:50:27,597 INFO:Client yza: Put key: 12345, value: qwert, result: T
rue
2024-01-09 18:50:33,514 INFO:Client yza: Get all data
2024-01-09 18:57:08,867 INFO:Client yza: Get key: 12345, returned value: qwert
2024-01-09 18:57:13,864 INFO:Client yza: Get key: qaz, returned value: 101
Enter command (put/get/delete/getall/delall/getlog):
```

lmz客户端的日志记录

```
Enter command (put/get/delete/getall/delall/getlog): getlog
2024-01-09 18:59:48,682 INFO:Client lmz: Get key: a, returned value: 1
2024-01-09 18:59:51,683 INFO:Client lmz: Get key: all, returned value: None
2024-01-09 19:00:24,799 INFO:Client lmz: Get all data
Enter command (put/get/delete/getall/delall/getlog):
```

xyr客户端日志记录

```
Enter command (put/get/delete/getall/delall/getlog): getlog
2024-01-09 19:00:42,853 INFO:Client xyr: Get key: 12345, returned value: qwert
2024-01-09 19:00:46,240 INFO:Client xyr: Get all data
Enter command (put/get/delete/getall/delall/getlog):
```

6. 剩余功能的统一测试（delete delall等）

```
Enter command (put/get/delete/getall/delall/getlog): delete a
Enter command (put/get/delete/getall/delall/getlog): getall
{'qaz': '101', '12345': 'qwert'}
Enter command (put/get/delete/getall/delall/getlog): delall
Enter command (put/get/delete/getall/delall/getlog): getall
{}
Enter command (put/get/delete/getall/delall/getlog):
```

delete（单个删除）以及delall（全部删除）功能测试正常

7. 客户端登入和登出的服务器端提示 模拟三个用户的登入登出，以及在服务器端的结果

```
(hw8) 13681080795163.com@x86_64-apple-darwin13 FB_me % python client.py
Please input your username:yza
Please input your password:928
Client ID: yza
Enter command (put/get/delete/getall/delall/getlog): ^CExiting client
(hw8) 13681080795163.com@x86_64-apple-darwin13 FB_me % python client.py
Please input your username:lmz
Please input your password:1224
Client ID: lmz
Enter command (put/get/delete/getall/delall/getlog): ^CExiting client
(hw8) 13681080795163.com@x86_64-apple-darwin13 FB_me % python client.py
Please input your username:xyr
Please input your password:403
Client ID: xyr
Enter command (put/get/delete/getall/delall/getlog): ^CExiting client
(hw8) 13681080795163.com@x86_64-apple-darwin13 FB_me %
```

```
2024-01-09 19:10:45,362 client: yza logged in
2024-01-09 19:10:50,284 client: yza logged out
2024-01-09 19:11:01,203 client: lmz logged in
2024-01-09 19:11:13,108 client: lmz logged out
2024-01-09 19:11:21,753 client: xyr logged in
2024-01-09 19:11:23,790 client: xyr logged out
```

四. 实验感想

完成一个基于RPC的分布式键值存储系统是一次极富挑战性和教育意义的经历。在这个过程中，我深入理解了远程过程调用的工作原理，包括数据序列化、网络通信和服务发现等关键技术。面对网络延迟、数据一致性和故障恢复等问题，我学习并实践了诸如Raft算法等解决方案，这不仅增强了我的技术能力，也锻炼了我的问题解决和创新思维。此外，通过实际应用理论知识，我对分布式系统有了更深刻的理解，同时也认识到

了持续学习在技术领域的重要性。总的来说，这是一次让我技术和思维都得到提升的宝贵经历。