

# 分布式键值存储系统

姓名	陈雪玮
班级	计算机科学与技术
学号	21307387

## 一、项目简介

1. 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。

2. 要求

- 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；
- 可以是集中式的也可以是非集中式；
- 能够完成基本的操作如：PUT、GET、DEL 等；
- 支持多用户同时操作；
- 至少实现一种面向客户的一致性如单调写；
- 需要完整的功能测试用例；
- 涉及到节点通信时须采用 RPC 机制；
- 提交源码和报告，压缩后命名方式为：学号\_姓名\_班级

加分项：

- 具备性能优化措施如 cache 等；
- 具备失效容错方法如：Paxos、Raft 等；
- 具备安全防护功能；
- 其他高级功能；

## 二、环境配置

编程语言：Python 3.10

主要使用库：xmlrpc

操作系统：Windows10

## 三、实验内容

### 3.1 实现功能

- 基本要求
  - ☒ 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；
  - ☒ 可以是集中式的也可以是非集中式；
  - ☒ 能够完成基本的操作如：PUT、GET、DEL 等；
  - ☒ 支持多用户同时操作；
  - ☒ 至少实现一种面向客户的一致性如单调写；
  - ☒ 需要完整的功能测试用例；
  - ☒ 涉及到节点通信时须采用 RPC 机制；
- 加分项
  - ☒ 具备性能优化措施如 cache 等；
  - ☐ 具备失效容错方法如：Paxos、Raft 等；
  - ☒ 具备安全防护功能；
  - ☒ 其他高级功能：用户友好界面，数据同步，可以并行读写。

### 3.2 原理分析

为了简化我的分布式键值存储系统实现，我做了两点设置：

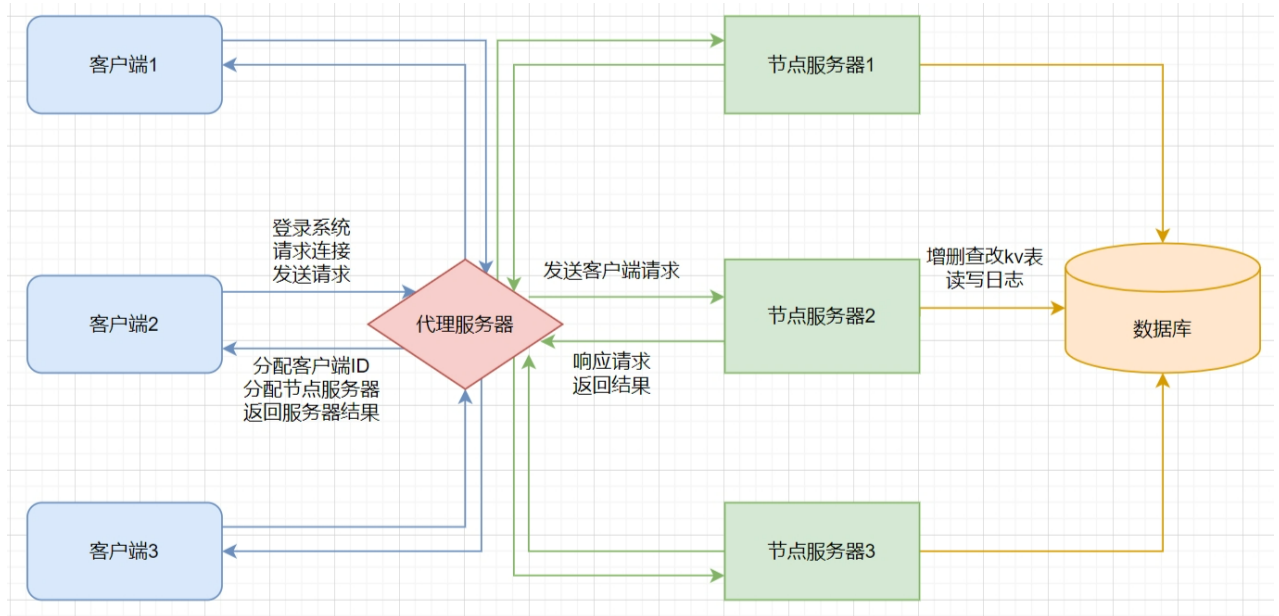
1. 代理服务分别为每个客户端都分配一个节点服务器，即一个节点服务器只一个客户端。
2. 数据保存在内存中，在节点服务器中，用全局变量模拟表示数据库。

我的分布式键值存储系统主要由四个部分组成：

1. 客户端 `client.py`

2. 代理服务器 `proxy_server.py`
3. 节点服务器 `node_server.py`
4. 数据库 `database`

它们的关系和主要实现功能如下图所示：

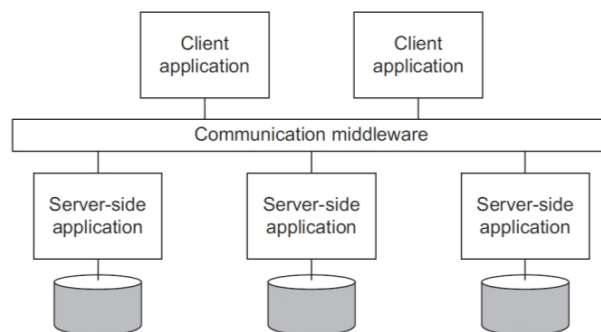


系统模型

具体来说，请求流程如下：

1. 客户端登录系统，连接到代理服务器。代理服务器分配客户端ID和对应的节点服务器。
2. 客户端发送KV操作请求到代理服务器，代理服务器调用对应节点服务器的KV操作。
3. 节点服务器对数据库执行KV操作，将结果返回给代理服务器。
4. 代理服务器将结果返回给对应的客户端，客户端显示KV操作结果。

实际上，这里的代理服务器相当于中间件，使得分布式的节点服务器对于客户端而言是透明的，客户端始终只会感受只与一个服务器进行交互。



中间件

## 3.3 代码分析

我的分布式键值存储系统实现的操作有：

- PUT key value —— 添加 (key, value)
- GET key —— 获取指定 key 的值
- DELETE key —— 删除指定 key 的值
- LIST —— 显示所有 (key, value)
- LOG —— 获取服务器端日志
- EXIT —— 退出客户端
- HELP—— 获取命令帮助菜单

系统可以运行多个服务器和多个客户端，多个客户端将连接到同一个代理服务器，代理服务器将不同客户端的请求分别发送给不同的节点服务器处理，然后将各自的响应返回。

客户端与服务器之间的通信使用rpc进行，代码中使用python库 `xmlrpc` 实现。XML-RPC是一个RPC的分布式计算协议，通过XML将调用函数封装，并使用HTTP协议作为发送机制。

客户端有登录验证功能。用户名和密码存储在代理服务器端，对客户端进行隐藏。客户端需要输入正确的用户名和密码才能连接到服务器。

代理服务器可以限定客户端最大连接数，若超过这个数量将拒绝连接。

节点服务器使用了锁机制，保证了对数据库操作的串行化和一致性，可以支持并行操作。又因为数据库和日志是全局的，修改后会同步到所有的节点服务器，保证了数据的同步。因此，我的键值存储系统满足顺序一致性。

每个节点服务器实例都会拥有自己独立的缓存空间。 `Server` 类中引入了一个 `cache` 字典，用于存储最近访问的键值对。获取键值对前，节点服务器首先检查缓存中是否存在请求的键，存在则直接返回；只有不存在的情况下，才从数据库中检索该值，并相应地更新缓存。

具体代码实现如下，代码中有详细注释，更详细功能分析包含在代码注释中。

---

### client.py

```
1 import xmlrpc.client as xmlrpclib
2
3
4 class Client(object):
5     def __init__(self):
6         self.id = None # 客户端ID
7
```

```

8         self.proxy = None # XML-RPC代理
9         self.port = None # 连接端口
10
11     def connect(self, username, password):
12         self.port = '21000'
13         self.proxy = xmlrpclib.ServerProxy('http://localhost:' + self.port)
14         # 登录
15         # 在此处进行验证 调用代理服务器的验证功能
16         if self.proxy.authenticate(username, password):
17             self.id = self.proxy.get_id() # 从代理服务器获取客户端ID
18             return self.id
19         # 验证成功的处理逻辑
20         else:
21             print('登录失败, 请检查用户名和密码。')
22             return None
23
24     def handle_user_command(self):
25         try:
26             while True:
27                 command = input(f"客户端 {self.id} 输入命令 (PUT/GET/DEL/LIST/LOG/EXIT)>> ").upper()
28                 if command == 'HELP':
29                     self.print_help() # 打印命令帮助
30                 else:
31                     self.send_command_to_server(command) # 向服务器发送命令
32                     if command == 'EXIT':
33                         break
34             except KeyboardInterrupt:
35                 pass
36
37     def print_help(self):
38         # 打印命令帮助信息
39         print(
40             '-----\n'
41             '命令帮助:\n'
42             'PUT key value — 添加 (key, value)\n'
43             'GET key — 获取指定 key 的值\n'
44             'DEL key — 删除指定 key 的值\n'
45             'LIST — 显示所有 (key, value)\n'
46             'LOG — 获取服务器端日志\n'
47             'EXIT — 退出客户端\n'
48             '-----'
49         )
50
51     def send_command_to_server(self, command):
52         msg = getattr(self.proxy, 'function')(self.id, command) # 向服务器发送命令并获取返回信息
53         if msg is not None:

```

```

54         print(msg) # 打印服务器返回信息
55
56
57 if __name__ == '__main__':
58
59     print("尝试登录...")
60     username = input('输入用户名: ')
61     password = input('输入密码: ')
62
63     client = Client()
64     # 验证用户名和密码是否匹配
65     client_id = client.connect(username, password) # 连接到服务器并获取客户端ID
66     if client_id is None:
67         print('连接失败，没有多余的用户ID可以分配。')
68     else:
69         # 登录成功，显示欢迎消息和客户端ID
70         print('-----')
71         print("登录成功。")
72         print('欢迎使用分布式键值系统! ')
73         print(f'您的客户端ID是 {client_id}')
74         print('输入 <help> 获取命令列表。')
75         print('-----')
76
77     client.handle_user_command() # 处理用户命令

```

## proxy\_server.py

```

1 from xmlrpc.server import SimpleXMLRPCServer
2 import xmlrpc.client as xmlrpclib
3
4
5 class ProxyServer:
6     def __init__(self, client_count):
7         # 用户名和密码
8         self.users = {
9             '1': '1',
10            '2': '2',
11            '3': '3',
12        }
13        # 初始化代理服务器，设置客户端连接状态和服务列表
14        self.client_ids = [False] * client_count # 用于标记客户端是否连接的列表
15        # 连接到不同的服务器节点，服务器的基地址是20000
16        self.servers = [xmlrpclib.ServerProxy(f'http://localhost:{20000 +
17            i}') for i in range(client_count)]
18
19        # 分配客户端ID

```

```

20     def get_id(self):
21         for i, connected in enumerate(self.client_ids):
22             if not connected:
23                 self.client_ids[i] = True
24                 print(f'客户端 {i} 登录')
25                 return i
26         print('没有可用的 ID')
27         return None
28
29     # 处理客户端发来的命令
30     def function(self, client_id, clause):
31         clause = clause.strip().split() # 解析命令
32         lens = len(clause)
33
34         if lens < 1:
35             return '错误的命令。输入 help 查看帮助信息。'
36
37         command = clause[0]
38
39         # 检查命令类型
40         if command in ['PUT', 'GET', 'DEL', 'LIST', 'LOG', 'EXIT']:
41             # 获取对应的方法
42             server_function = getattr(self, command.lower())
43             return server_function(client_id, clause)
44         else:
45             return '错误的命令。输入 help 查看帮助信息。'
46
47     # 处理客户端退出命令
48     def exit(self, client_id, clause):
49         self.client_ids[client_id] = False
50         print(f'客户端 {client_id} 退出')
51         return f'客户端 {client_id} 退出'
52
53     # 实现PUT方法
54     def put(self, client_id, clause):
55         if len(clause) != 3:
56             return '错误的命令格式。使用方法: PUT key value'
57
58         key, value = clause[1], clause[2]
59         if self.servers[client_id].put(key, value):
60             return f"成功添加/更新键 {key}, 值 {value}"
61         return f"无法添加/更新键 {key}, 值 {value}"
62
63     # 实现GET方法
64     def get(self, client_id, clause):
65         if len(clause) != 2:
66             return '错误的命令格式。使用方法: GET key'
67

```

```

68         key = clause[1]
69         value = self.servers[client_id].get(key)
70         return f"键 {key} 的值为: {value if value is not None else '[未找到值]'}"
71
72     # 实现LIST方法
73     def list(self, client_id, clause):
74         if len(clause) != 1:
75             return '错误的命令格式。使用方法: LIST'
76
77         return self.servers[client_id].list()
78
79     # 实现DELETE方法
80     def delete(self, client_id, clause):
81         if len(clause) != 2:
82             return '错误的命令格式。使用方法: DEL key'
83
84         key = clause[1]
85         if self.servers[client_id].delete(key):
86             return f"删除键 {key} 成功"
87         return f"无法删除键 {key}"
88
89     # 实现LOG方法
90     def log(self, client_id, clause):
91         if len(clause) != 1:
92             return '错误的命令格式。使用方法: LOG'
93
94         return self.servers[client_id].get_log()
95
96     # 登陆验证
97     def authenticate(self, username, password):
98         if username not in self.users:
99             print('不存在该用户名, 请重试! ')
100             return False
101         elif self.users[username] != password:
102             print('密码错误, 请重试! ')
103             return False
104         else:
105             return True
106
107
108 if __name__ == '__main__':
109     count = int(input('输入客户端数量: '))
110     proxy = ProxyServer(client_count=count)
111     server = SimpleXMLRPCServer(('localhost', 21000), allow_none=True)
112     server.register_instance(proxy)
113
114

```



```
print(f"代理服务器正在运行...")
server.serve_forever()
```

## node\_server.py

```
1 import threading
2 from xmlrpc.server import SimpleXMLRPCServer
3 from xmlrpc.server import SimpleXMLRPCRequestHandler
4
5 # 这里模拟数据库
6 log = []
7 database = {}
8 database_lock = threading.Lock()
9
10
11 class Server:
12     def __init__(self, server_id):
13         self.server_id = server_id
14         self.cache = {} # 每个服务器实例的缓存字典
15
16     def put(self, key, value):
17         # 存储键值对到数据库并更新缓存
18         with database_lock:
19             database[key] = value
20             self.cache[key] = value # 添加/更新缓存
21             msg = f"添加/更新key: {key}, value: {value}"
22             self.write_log(msg)
23             return True
24
25     def get(self, key):
26         # 先检查缓存，如果存在于缓存中则直接返回
27         if key in self.cache:
28             return self.cache[key]
29
30         # 如果不在缓存中，则从数据库中获取，并更新缓存
31         with database_lock:
32             value = database.get(key)
33             if value:
34                 self.cache[key] = value # 如果在数据库中找到，则更新缓存
35             return value
36
37     def delete(self, key):
38         # 从数据库中删除键值对，并从缓存中删除
39         with database_lock:
40             if key in database:
41
```


```

42         del database[key]
43         if key in self.cache:
44             del self.cache[key] # 从缓存中删除
45         msg = f"删除key: {key}"
46         self.write_log(msg)
47         return True
48     return False
49
50     def list(self):
51         # 返回整个数据库
52         with database_lock:
53             return database
54
55     def get_log(self):
56         # 返回服务器日志
57         with database_lock:
58             return log
59
60     def write_log(self, msg):
61         # 记录服务器操作相关的日志
62         log.append(f"服务器 {self.server_id}: {msg}")
63         return True
64
65
66 def run_server(server_id):
67     # 启动和运行 XML-RPC 服务器
68     server = SimpleXMLRPCServer(("localhost", 20000 + server_id), requestHandler=SimpleXMLRPCRequestHandler)
69     server.register_instance(Server(server_id))
70     print(f"服务器 {server_id} 正在运行在端口 {20000 + server_id}\n")
71     server.serve_forever()
72
73
74 if __name__ == "__main__":
75     # 输入服务器数量并启动相应数量的线程
76     count = int(input('输入服务器数量: '))
77     threads = []
78
79     for i in range(count):
80         server_thread = threading.Thread(target=run_server, args=(i,))
81         threads.append(server_thread)
82         server_thread.start()

```

## 3.4 功能测试

文件夹下的 程序演示.mp4 对主要功能进行了详细的测试和讲解，包含多个节点运行和已有功能的测试。

 程序演示.mp4	2023/12/21 20:24	MP4 文件	35,286 KB
--	------------------	--------	-----------

或者移步至网页端观看：

[分布式大作业——简单分布式键值存储系统的实现\\_哔哩哔哩\\_bilibili](#)

## 四、实验总结

### 4.1 遇到的困难和解决方法

我遇到的困难主要是如何设计这个系统，如何实现这个设计，以及在实现过程中遇到的BUG。

在设计这个系统时，我首先想的是实现一个集中式的键值存储系统，即只有一个客户端和一个服务器。这个实现需要rpc进行远程通信。经过了解，我选择了较为简单的python语言和xmlrpc库来实现远程通信。

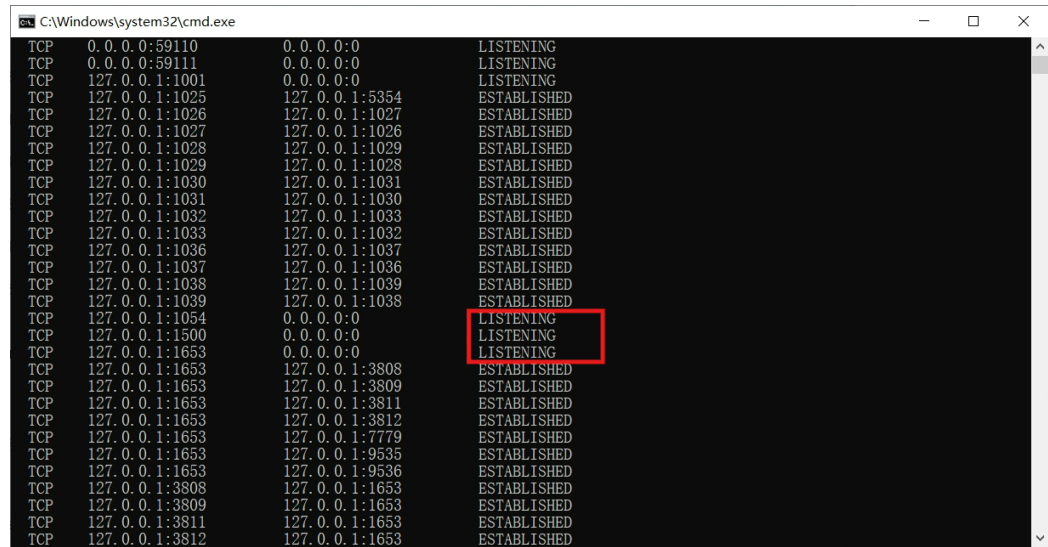
但是初次使用就遇到了计算机积极拒绝的问题。

**ConnectionRefusedError: [WinError 10061] 由于目标计算机积极拒绝，无法连接。**

首先，我排除了端口号错误和方法调用错误的原因。

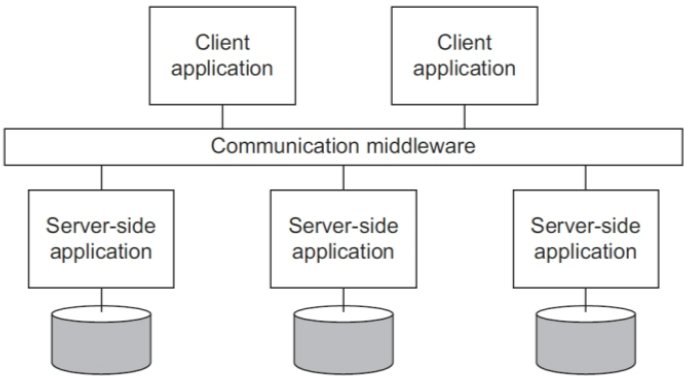
然后，在尝试了多个解决方法，如设置防火墙、修改配置网络等都无效的情况下，我找到了最终的解决方法。因为我们电脑系统开放的端口号是不一样的，因此我们需要在命令行窗口使用 `netstat -an` 命令来查看系统开放的端口，处于listening状态的端口才是可以使用的。将端口号改为listening的端口，这个BUG就解决了。

参考：[127.0.0.1由于目标积极拒绝，无法连接](#)



netstat -an查看系统开放的端口

设计好单个服务器与客户端的系统后，很简单的就可以拓展为多个客户端和多个服务器，现在的问题是该如何对客户隐藏有多个服务器的事实，使其感觉到只有一个服务器在运行呢？这个问题我通过中间件的设计思想，增加代理服务器解决了。



中间件

设计好一个简单的键值存储系统后，我思考如何给它添加一些功能，如一致性、安全防护、数据同步等问题。为了简化设计，我使用全局变量模拟数据库，使得所有服务器都连接到同一个数据库中。除此之外，我还对数据库的读写引入锁机制，对用户的登录引入登录验证功能。

## 4.2 实验心得

通过这个实验，我对RPC的原理和使用更加熟悉，也更具体地感受到中间件在分布式系统中的重要作用。特别是从一个简单的系统一步步拓展完善，直到实现一个包含简单功能的分布式键值存储系统的这个过程，加深了我对分布式系统的理解。