浙江大学

课程综合实践 II

Rust 开发实训实验报告

小组成员

学号	姓名
组长信息	陈书剑
组员信息	张亦弛
组员信息	王祥铸

2023 年 9 月 15 日

目录

一．系统简介 3

1.1 系统目标 3

1.2 设计说明 3

二．总体设计 4

2.1 总体架构设计 4

2.2 流程图设计 5

三．详细设计 5

3.1 AOF 持久化 5

3.1.1 （时序图等 UML 图） 5

3.2 REDIS 主从架构 6

3.2.1 （时序图等 UML 图） 6

3.3 REDIS PROXY 7

3.4 加分项（可选） 5

四．测试用例 8

4.1 AOF 持久化 8

4.1.1 测试用例 1 8

4.2 REDIS 主从结构 8

4.3 REDIS PROXY 9

4.4 加分项测试（可选） 6

参考文献（可选） 6

一. 系统简介

本次大作业开发的系统是分布式 MiniRedis，实现小型键值对数据库的代理端，客户端与服务端。

客户端可以向代理发送指令，由代理进行指令分发到对应的服务端，服务端进行数据查询并回复。能实现以及相应的异常处理。具体要求见 1.1。

1.1 系统目标

MiniRedis 具体功能如下：

（1）实现指令：Set,Get,Del,Ping,Subscribe,Public。能正确收发并

（2）AOF （Append-only File）实现持久化，对于一个已经存在的 log file（格式自定，文件自行准备），在 redis server 启动的时候可以根据该文件重建 redis 数据。

（3）实现 Redis 主从架构, 定义一个配置文件的格式，可以从该配置文件启动多个 redis 实例，并且区分哪些是主节点，哪些是从节点。主节点可以执行所有指令。从节点只能进行 Get 操作，否则报错。

（4）实现 Redis Cluster。实现一个 Redis Proxy，可以将 Set 和 Get 请求转发到其他的 Redis 实例上进行处理。定义一个配置文件的格式，可以从该配置文件启动多个 redis 实例和一个 redis proxy。Redis Proxy 可以根据 Get 和 Set 目标的 key 将请求分发到不同的 redis 实例进行处理。

1.2 设计说明

本大作业采用 Rust 程序设计语言，在 wsl 平台下编辑、编译与调试。具体程序由 3 人组成的小组开发而成。小组成员的具体分工如表 1 所示：

表 1 各成员分工表

成员姓名	学号	分工
陈书剑	3210103880	算法结构设计，AOF，主从服务器，cluster 代码

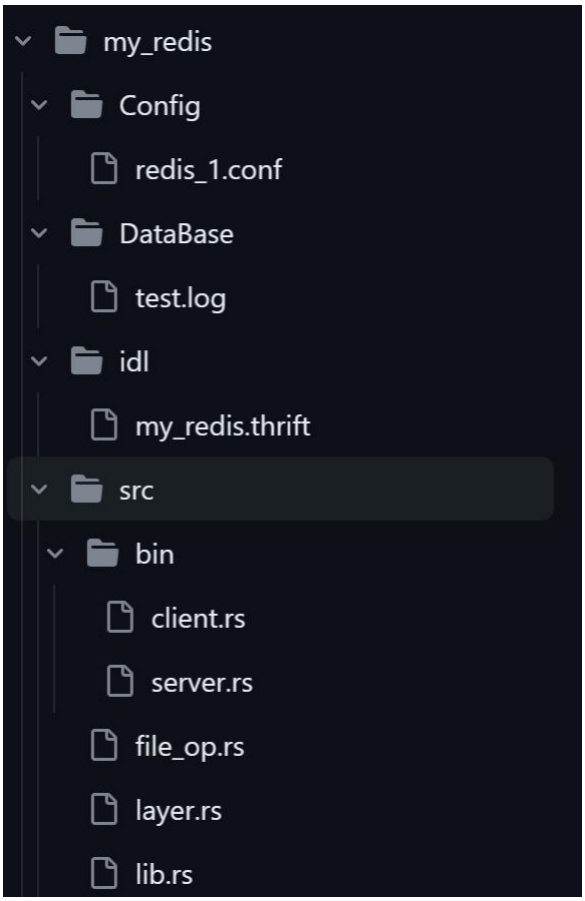
		实现以及调试
张亦弛	3210103388	算法结构设计，主从服务器，cluster 代码实现，配置文件设计，ppt
王祥铸	3210103324	算法结构设计，AOF，测试脚本，报告，Graceful Exit，ppt，展示

（第一行为组长，分工需与后面的模块、功能等相对应）

二．总体设计

2.1 总体架构设计

本系统主要分为以下几个模块：



（1）client.rs 客户端相关操作

（2）server.rs 服务端相关操作

（3）file_op.rs 处理文件

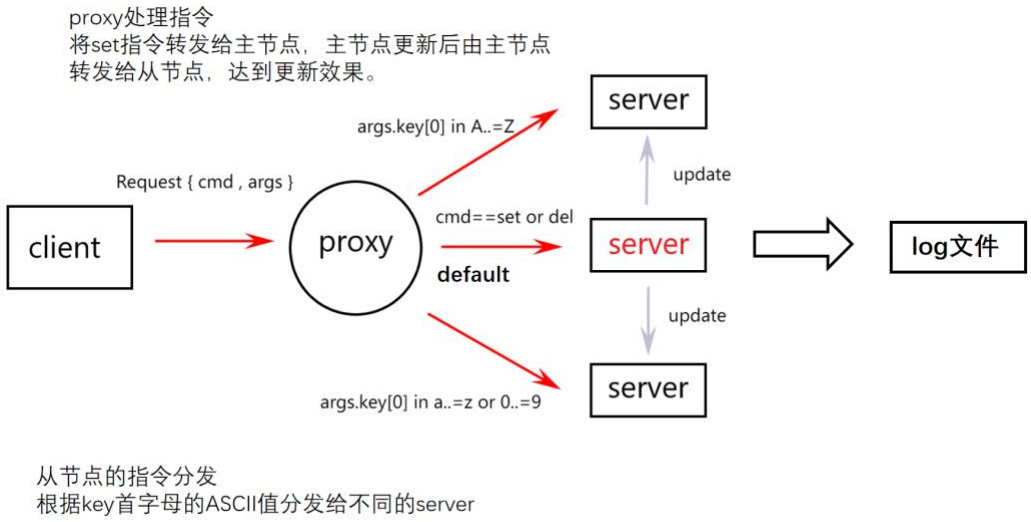
（4）lib.rs 内部函数如 get_item 等以及中间层的实现

（5）redis_1.conf 存放 proxy 以及各个 server 的相关信息的配置文件

（6）test.log 存放 aof 文件数据，记录历史操作，方便重构服务器

2.2 流程图设计

系统流程图如下:

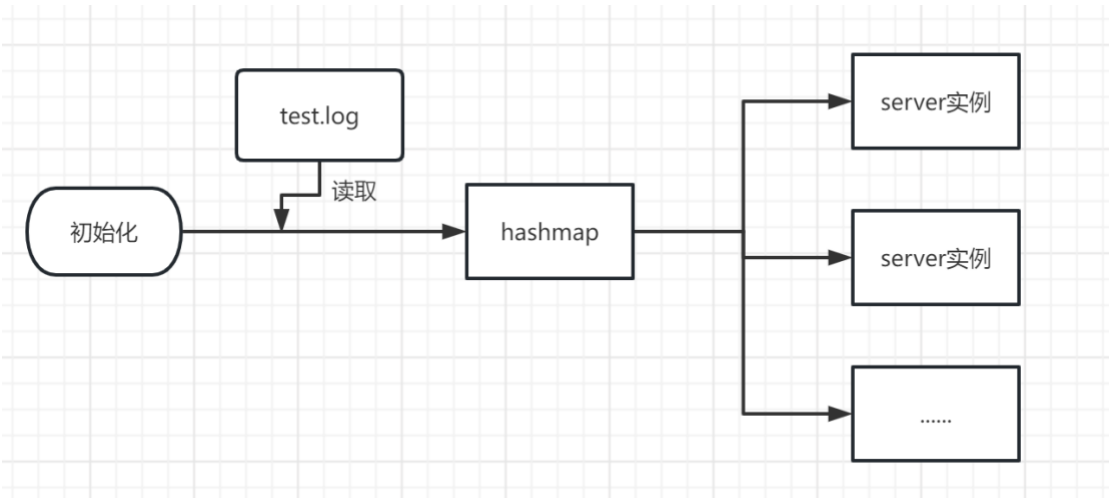


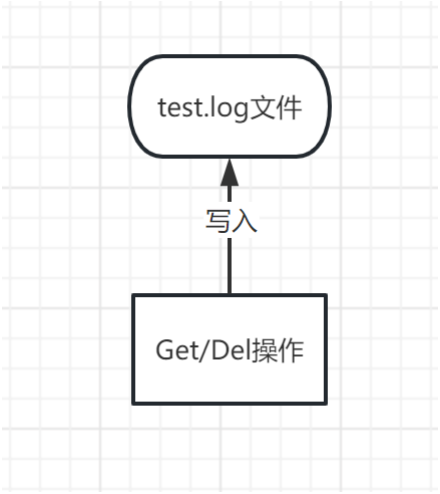
三．详细设计

3.1 AOF 持久化

3.1.1 （时序图等 UML 图）

我们会在每次重启 server 时对文件进行读取并存入 hashmap 用于重构服务器。以及 Del/Set 操作时对文件进行写回相关操作。

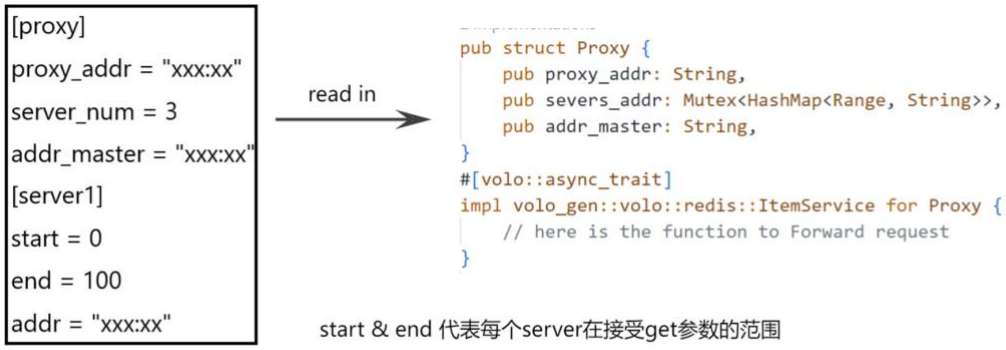




3.2 Redis 主从架构

3.2.1 （时序图等 UML 图）

config 文件内容与程序数据结构的对应关系：

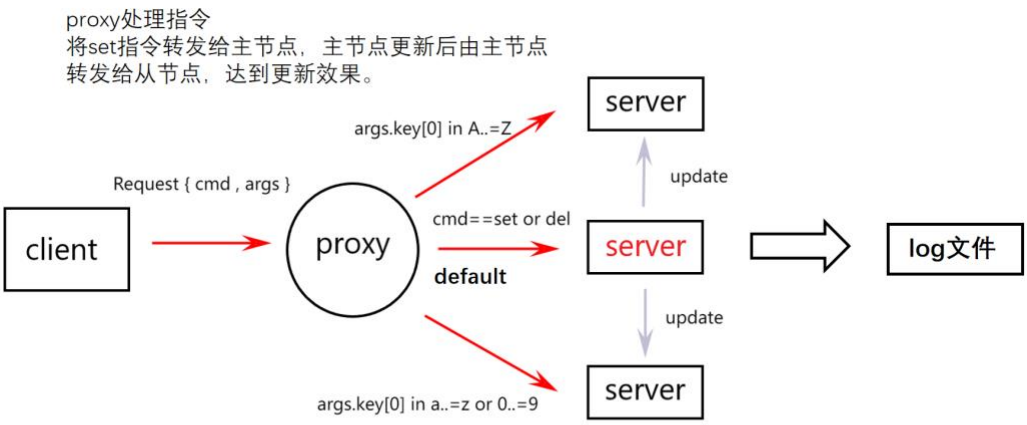


servers.config

start & end 代表每个server在接受get参数的范围

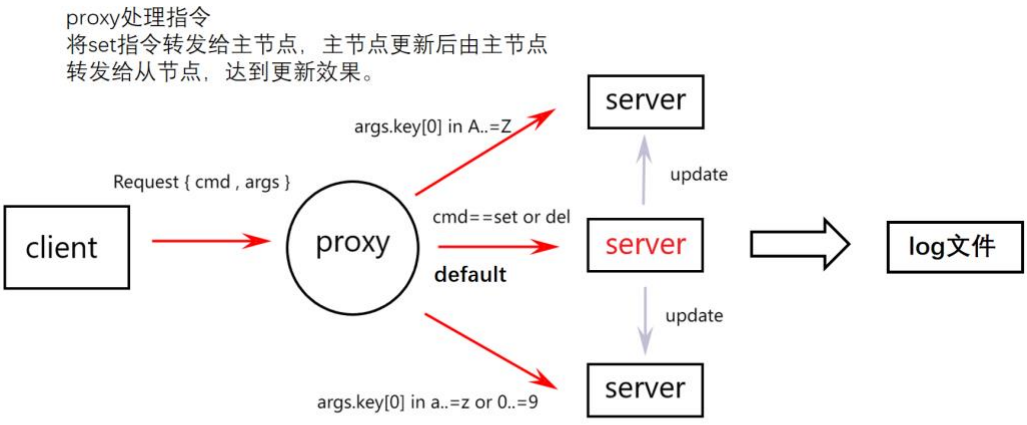
在配置文件中设置好主从服务器身份。

proxy已知master的端口



从节点的指令分发
根据key首字母的ASCII值分发给不同的server

3.3 Redis Proxy



从节点的指令分发
根据key首字母的ASCII值分发给不同的server

四．测试用例

4.1 AOF 持久化

4.1.1 测试用例 1

```python
import subprocess
import time
def main():
# 启动服务器
    server_process = subprocess.Popen("./target/debug/server", shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE)
    time.sleep(1)
# 启动客户端
    client_process = subprocess.Popen("./target/debug/client",shell=True,stdin=subprocess.PIPE,stdout=subprocess.PIPE)
    time.sleep(1)

# 发送 "Set" 到客户端
    val = "\"test1\""
    print("client do Set test1 test1")
    client_process.stdin.write(b"Set test1 test1\n")
    client_process.stdin.flush()
    time.sleep(0.3)
    # 获取服务端返回的值
    x = client_process.stdout.readline().decode().strip()
    print("Received:", x)

    # 关闭客户端
    client_process.stdin.close()

    # 关闭服务器
    server_process.stdin.close()

    # 重启服务器
    server_process = subprocess.Popen("./target/debug/server",shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE)
    time.sleep(0.3)
    # 重启客户端
    client_process = subprocess.Popen("./target/debug/client",shell=True, stdin=subprocess.PIPE, stdout=subprocess.PIPE)
    time.sleep(0.3)
    # 发送 "Get" 到客户端
    client_process.stdin.write(b"Get test1\n")
    client_process.stdin.flush()
    time.sleep(0.3)
    # 获取客户端返回的值
    time.sleep(0.3)
    output = client_process.stdout.readline().decode().strip()
    print("Output:", output)
    time.sleep(0.3)
    # 检查输出是否与前一个值一致
    if output == val:
        print("Output matches the previous value.")
    else:
        print("Output does not match the previous value!")

    # 关闭服务器和客户端
    server_process.stdin.close()
    client_process.stdin.close()

    quit()
```

4.2 Redis 主从结构

此处为 Get hello 和 Set a aa 指令的 server 端输出。可以看到是 get 是由 8081（从服务器）执行，Set 则是由 8080 主服务器执行，然后转发指令给两个从节点

```
tmp: ["127.0.0.1:8080", "127.0.0.1:8081", "127.0.0.1:8082"]
proxy_addr: 127.0.0.1:1234
2023-09-14T15:58:30.904595Z  INFO volo::hotrestart: hot_restart skip dup_parent_listener_sock: 127.0.0.1:1234, as parent
2023-09-14T15:58:30.904675Z  INFO volo::hotrestart: hot_restart register_listener_fd: 127.0.0.1:1234, 9
2023-09-14T15:58:30.904718Z  INFO volo_thrift::server: [VOLO] server start at: Tcp(TcpListenerStream { inner: PollEvented { io: Some(TcpListener { addr: 127.0.0.1:1234, fd: 9 }) } })
2023-09-14T15:58:30.904786Z  INFO volo::hotrestart: hot_restart skip dup_parent_listener_sock: 127.0.0.1:8082, as parent
2023-09-14T15:58:30.904819Z  INFO volo::hotrestart: hot_restart register_listener_fd: 127.0.0.1:8082, 10
2023-09-14T15:58:30.904828Z  INFO volo_thrift::server: [VOLO] server start at: Tcp(TcpListenerStream { inner: PollEvented { io: Some(TcpListener { addr: 127.0.0.1:8082, fd: 10 }) } })
2023-09-14T15:58:30.904856Z  INFO volo::hotrestart: hot_restart skip dup_parent_listener_sock: 127.0.0.1:8081, as parent
2023-09-14T15:58:30.904890Z  INFO volo::hotrestart: hot_restart register_listener_fd: 127.0.0.1:8081, 11
2023-09-14T15:58:30.904900Z  INFO volo_thrift::server: [VOLO] server start at: Tcp(TcpListenerStream { inner: PollEvented { io: Some(TcpListener { addr: 127.0.0.1:8081, fd: 11 }) } })
2023-09-14T15:58:30.904919Z  INFO volo::hotrestart: hot_restart skip dup_parent_listener_sock: 127.0.0.1:8080, as parent
2023-09-14T15:58:30.904948Z  INFO volo::hotrestart: hot_restart register_listener_fd: 127.0.0.1:8080, 12
2023-09-14T15:58:30.904959Z  INFO volo_thrift::server: [VOLO] server start at: Tcp(TcpListenerStream { inner: PollEvented { io: Some(TcpListener { addr: 127.0.0.1:8080, fd: 12 }) } })
addr: 127.0.0.1:8081
_req: GetItemRequest { cmd: Get, args: Some(["hello"]) }
addr: 127.0.0.1:8080
_req: GetItemRequest { cmd: Set, args: Some(["a", "aa"]) }
receive set! current type is Master
receive set! current type is Slave
receive set! current type is Slave
```

4.3 Redis Proxy

可以从这张图中看出 get 指令会由 proxy 转发给不同的服务器进行处理。而 set、del 指令则只会转发给主服务器