

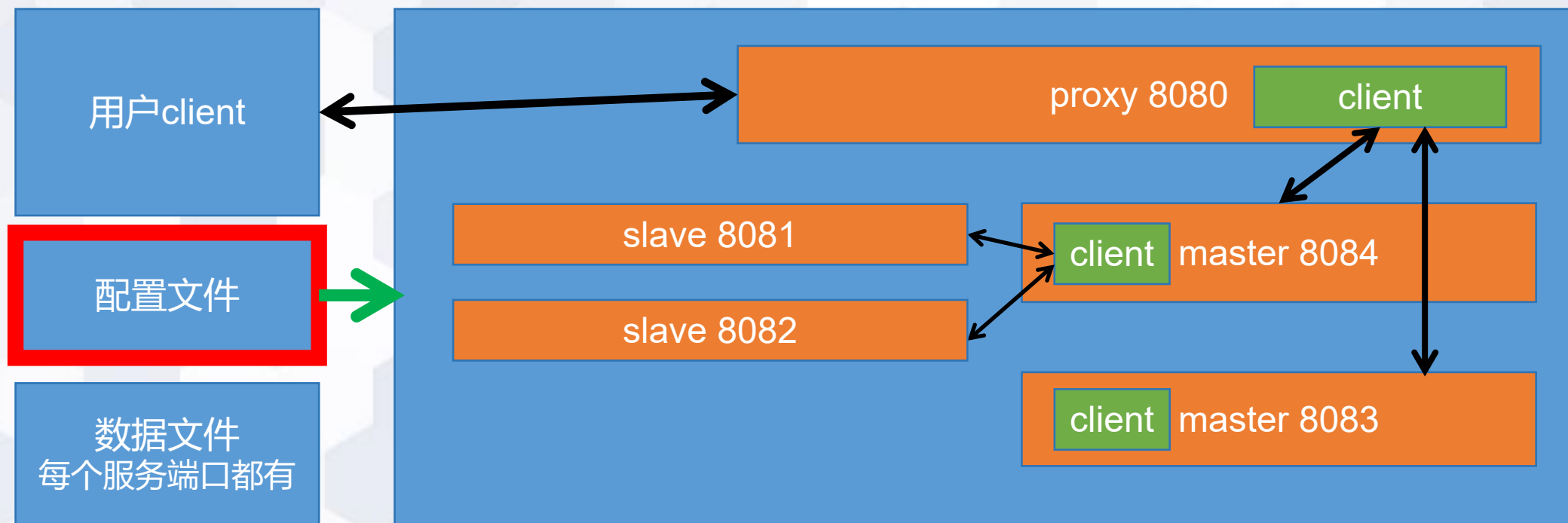


作业展示

小组成员： 陈彦杰 李昕阳 杨朗骐



原理展示





配置文件Server.json

```
[["127.0.0.1:8080",["y"],["y"]],["127.0.0.1:8081",["n","127.0.0.1:8080"],["n","127.0.0.1:8084"]],["127.0.0.1:8082",["n","127.0.0.1:8080"],["n","127.0.0.1:8084"]],["127.0.0.1:8083",["n","127.0.0.1:8080"],["y"]],["127.0.0.1:8084",["n","127.0.0.1:8080"],["y","127.0.0.1:8081","127.0.0.1:8082"]]]
```

```
let infos: Vec<(String, Vec<String>, Vec<String>)> = vec![
    ("127.0.0.1:8080".to_string(),vec!["y".to_string()],vec!["y".to_string()]),
    ("127.0.0.1:8081".to_string(),vec!["n".to_string(),"127.0.0.1:8080".to_string()],vec!["n".to_string(), "127.0.0.1:8084".to_string()]),
    ("127.0.0.1:8082".to_string(),vec!["n".to_string(),"127.0.0.1:8080".to_string()],vec!["n".to_string(), "127.0.0.1:8084".to_string()]),
    ("127.0.0.1:8083".to_string(),vec!["n".to_string(),"127.0.0.1:8080".to_string()],vec!["y".to_string()]),
    ("127.0.0.1:8084".to_string(),vec!["n".to_string(),"127.0.0.1:8080".to_string()],
    vec!["y".to_string(), "127.0.0.1:8081".to_string(), "127.0.0.1:8082".to_string()]),
];
```



Struct S

```
impl Serialize for DB {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        self.kvs.serialize(serializer)
    }
}

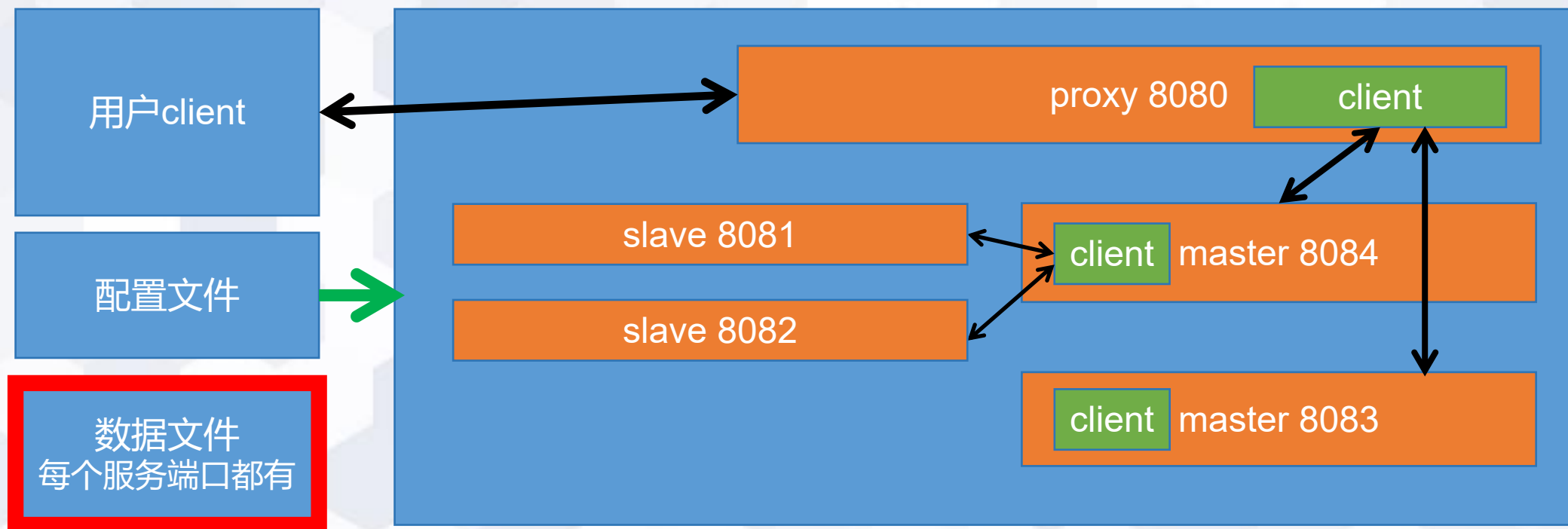
impl<'de> Deserialize<'de> for DB {
    fn deserialize<D>(deserializer: D) -> Result<Self, D::Error>
    where
        D: serde::Deserializer<'de>,
    {
        let kvs: HashMap<String, String> = HashMap::deserialize(deserializer)?;
        core::result::Result::Ok(DB { kvs })
    }
}

2 implementations
pub struct Tm{
    kts: HashMap<String, (u128,u128)>,
}
```

```
4 implementations
pub struct S{
    contents:RwLock<RefCell<DB>>,
    times:RwLock<RefCell<Tm>>,
    port: RwLock<RefCell<String>>,
    proxy: RwLock<RefCell<Vec<String>>>,
    master: RwLock<RefCell<Vec<String>>>,
}
```



AOF





AOF

```
impl S{
  pub fn store(&self) -> Result<(), volo_thrift::AnyhowError> {
    // 获取内容锁
    let contents: RwLockReadGuard<'_, RefCell<...>> = self.contents.read().unwrap();
    let times: RwLockReadGuard<'_, RefCell<...>> = self.times.read().unwrap();
    let port: RwLockReadGuard<'_, RefCell<...>> = self.port.read().unwrap();
    let proxy: RwLockReadGuard<'_, RefCell<...>> = self.proxy.read().unwrap();
    let master: RwLockReadGuard<'_, RefCell<...>> = self.master.read().unwrap();

    // 序列化内容
    let myport: String = port.borrow().clone();
    let db_json: String = serde_json::to_string(&*contents.borrow())?;
    let tm_json: String = serde_json::to_string(&*times.borrow())?;
    let port_json: String = serde_json::to_string(&*port.borrow())?;
    let proxy_json: String = serde_json::to_string(&*proxy.borrow())?;
    let master_json: String = serde_json::to_string(&*master.borrow())?;
    //println!("{}", myport);

    // 写入文件
    //println!("1111111");
    let mut file: File = File::create(path: "data".to_string()+&myport+".json");
    file.write_all(buf: db_json.as_bytes())?;
    file.write_all(buf: b"\n")?; // 添加换行符以分隔内容
    file.write_all(buf: tm_json.as_bytes())?;
    file.write_all(buf: b"\n")?; // 添加换行符以分隔内容
    file.write_all(buf: port_json.as_bytes())?;
    file.write_all(buf: b"\n")?; // 添加换行符以分隔内容
    file.write_all(buf: proxy_json.as_bytes())?;
    file.write_all(buf: b"\n")?; // 添加换行符以分隔内容
    file.write_all(buf: master_json.as_bytes())?;
    Ok(())
  }
}
```

```
pub fn new(port: String, proxy:Vec<String>, master:Vec<String>) -> Self {
  let db: DB;
  let tm: Tm;
  let myport: String;
  let myproxy: Vec<String>;
  let mymaster: Vec<String>;

  let path: String = "data".to_string()+&port+".json";
  if Path::new(&path).exists() {
    println!("A database is created from the file");
    // 如果文件存在, 则从文件中导入内容
    let mut file: File = File::open(&path).expect(msg: "Error (1) in reading the file");
    let mut file_contents: String = String::new();
    file.read_to_string(buf: &mut file_contents).expect(msg: "Error (2) in reading the file");

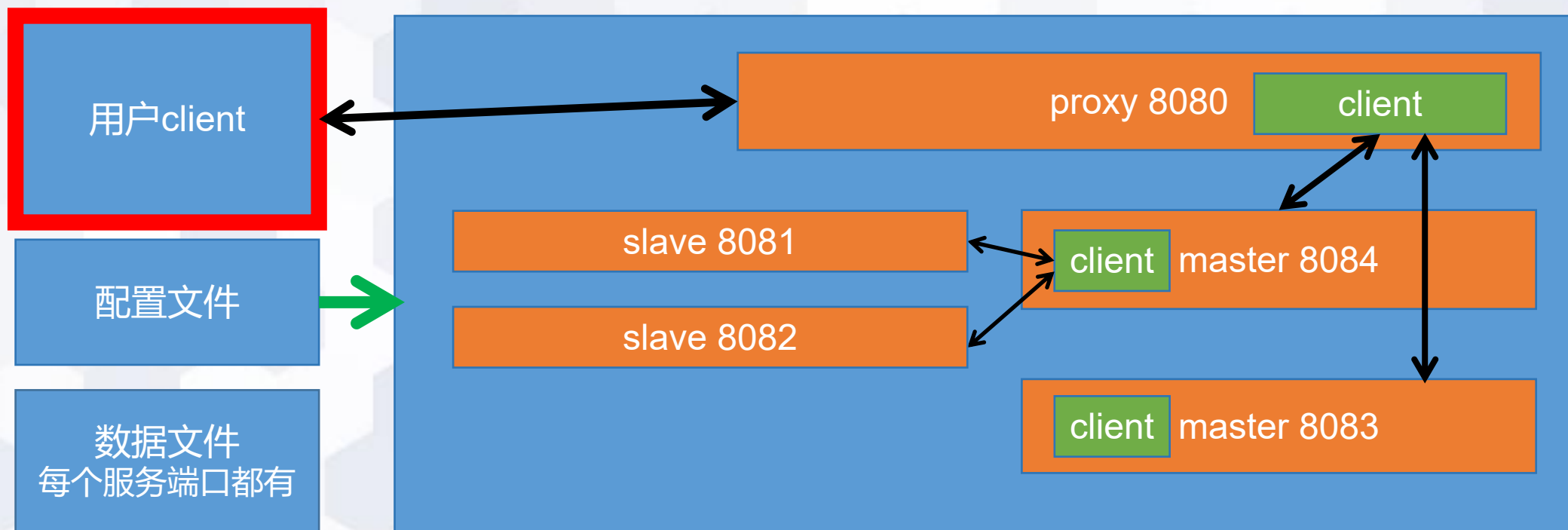
    let lines: Vec<&str> = file_contents.lines().collect();

    // 反序列化内容
    db = serde_json::from_str(lines[0]).expect(msg: "Error (3) in reading the file");
    tm = serde_json::from_str(lines[1]).expect(msg: "Error (4) in reading the file");
    myport = serde_json::from_str(lines[2]).expect(msg: "Error (5) in reading the file");
    myproxy = serde_json::from_str(lines[3]).expect(msg: "Error (6) in reading the file");
    mymaster = serde_json::from_str(lines[4]).expect(msg: "Error (7) in reading the file");
  } else {
    println!("A new database is created");
    // 如果文件不存在, 则创建新的结构体
    db = DB { kvs: HashMap::new() };
    tm = Tm { kts: HashMap::new() };
    myport = port;
    myproxy = proxy;
    mymaster = master;
  }
  // 创建 S 结构体并返回
  S {
    contents: RwLock::new(RefCell::new(db)),
    times: RwLock::new(RefCell::new(tm)),
    port: RwLock::new(RefCell::new(myport)),
    proxy: RwLock::new(RefCell::new(myproxy)),
    master: RwLock::new(RefCell::new(mymaster)),
  }
}
```



主从&cluster

以set 127.0.0.1:8084 2 4为例





主从&cluster

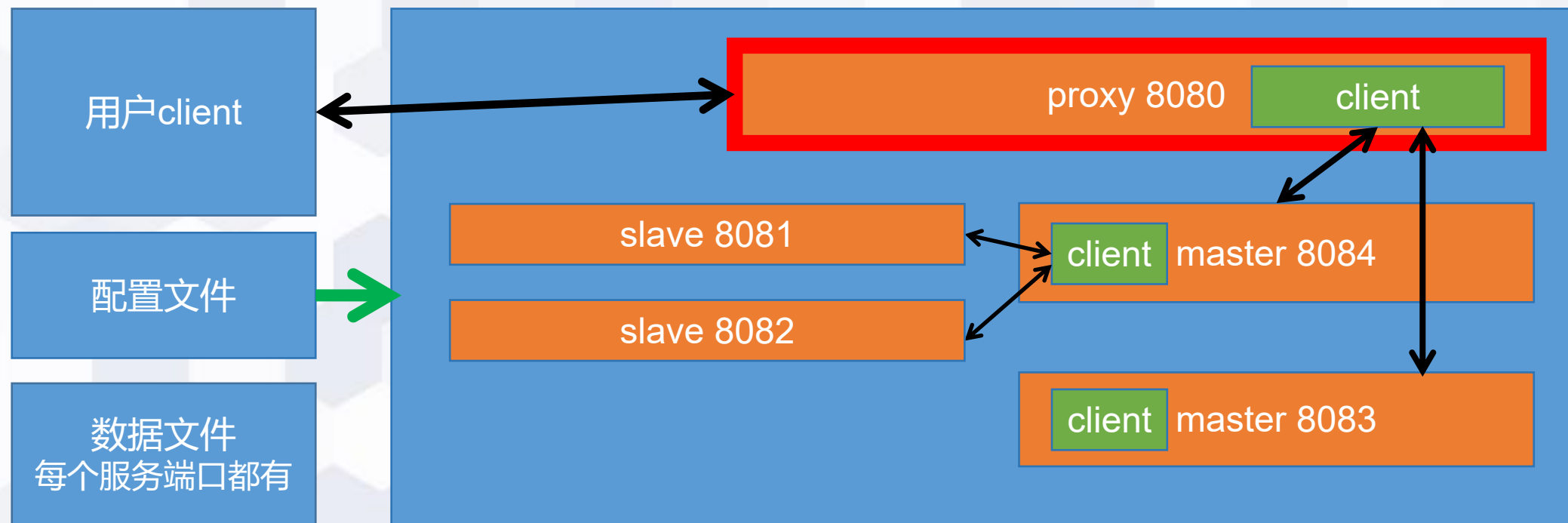
以set 127.0.0.1:8084 2 4为例
用户client

```
[volo::main]
► Run | Debug
sync fn main() {
    let args: Vec<String> = env::args().collect();
    unsafe { ADDR_STR = args[1].clone(); }
    tracing_subscriber::fmt::init();
    loop{
        print!("redis> ");
        let mut input: String= String::new();
        io::stdout().flush().expect(msg: "无法刷新标准输出");
        io::stdin().read_line(buf: &mut input).expect(msg: "读取失败! ");
        let words: Vec<&str> = input.split_whitespace().collect();
        let input: Vec<String> = words.iter().map(|&s: &str| s.to_string()).collect();
        //println!("单词: {:?}", input);
        let mut req: GetItemRequest = volo_gen::volo::example::GetItemRequest { op:" ".into(), key:" ".into(), value:" ".into(), life:0i32, otherport:" ".into()};
        if words.len() == 0{
            println!("输入为空, 请重新输入: ");
            continue;
        }
        else if input.len() == 4 {
            req.op = "setport".into();
            req.otherport = input[1].clone().into();
            req.key = input[2].clone().into();
            req.value = input[3].clone().into();
        }
        let resp: Result<GetItemResponse, ResponseError<...>> = CLIENT.get_item(req).await;
```




主从&cluster

以set 127.0.0.1:8084 2 4为例
redis proxy





主从&cluster

以set 127.0.0.1:8084 2 4为例
redis proxy
server

```
"setport" => {
    //println!("{}", value);
    resp.op = "setport".into();
    resp.key = key.clone().into();
    resp.value = value.clone().into();
    if self.proxy.read().unwrap().borrow()[0].eq("\n"){
        //println!("{}", "fffff");
        resp.op = "setportfail".into();
        resp.value = self.proxy.read().unwrap().borrow()[1].clone().into();
        resp.state = false;
        return Ok(resp);
    }
    else{
        resp.state = true;
        //self.store().expect("Error in storing the data");
        let port: String = otherport.clone();
        let key: FastStr = resp.key.clone();
        let value: FastStr = resp.value.clone();
        let (tx: Sender<GetItemResponse>, rx: Receiver<GetItemResponse>) = tokio::sync::oneshot::channel();
        tokio::spawn(future: async move {
            let message: GetItemResponse = other(opstr: "set".to_string(), key: key.into_string(), value: value.into_string(), &port).await;
            //println!("{}", "lib{:?}", message);
            if tx.send(message).is_err() {
                println!("{}", "Failed to send result to the channel");
            }
        });
        let message: GetItemResponse = rx.await.expect(msg: "Failed to receive result from the channel");
        return Ok(message);
    }
}
```

```
if self.proxy.read().unwrap().borrow()[0].eq("y"){
    let mut dis: HashMap<i32, String> = HashMap::new();
    let ports: Vec<String> = self.proxy.read().unwrap().borrow().clone();
    let mut cnt: i32 = 0;
    for s_ports: &String in ports.iter().skip(1) {
        dis.insert(k: cnt, v: s_ports.clone());
        cnt += 1;
    }
    let which: i32 = i32::from(key.as_bytes()[0])%cnt;
    resp.state = true;
    //self.store().expect("Error in storing the data");
    let port: String = dis[&which].clone();
```



主从&cluster

以set 127.0.0.1:8084 2 4为例
redis proxy client

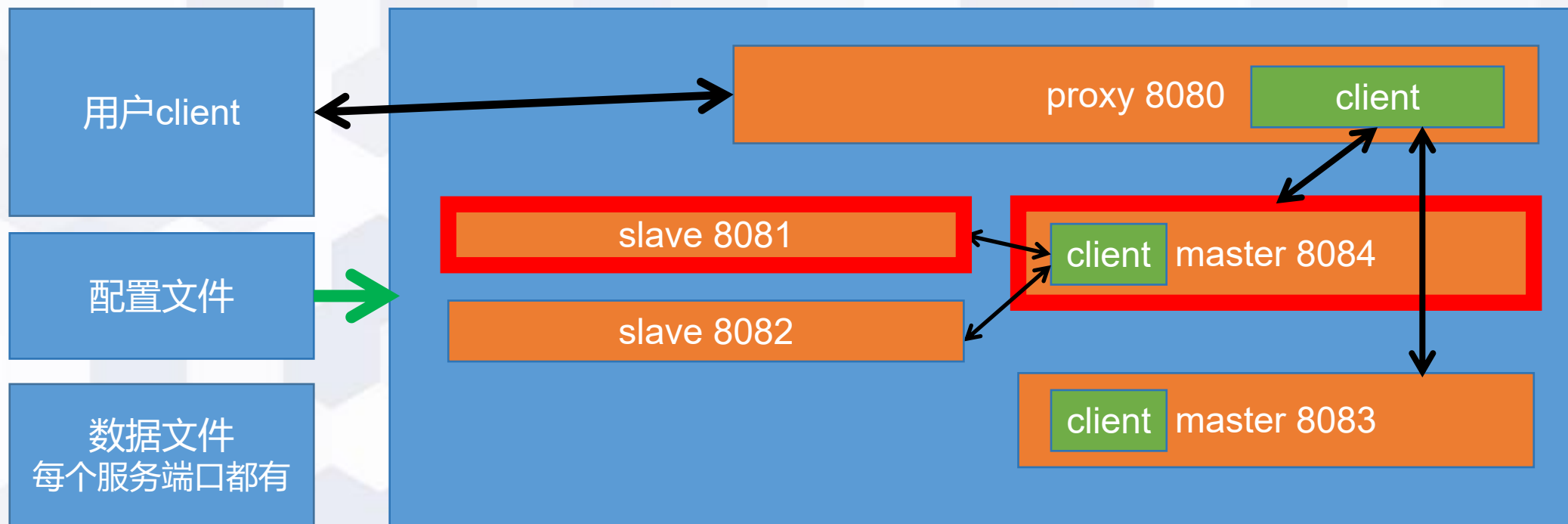
```
use volo_gen::volo::example::GetItemResponse;
```

```
async fn other(opstr: String, key:String, value:String, port: &str) -> GetItemResponse{
    unsafe { ADDR_STR = port.to_string();
    }
    let client: volo_gen::volo::example::ItemServiceClient =
    unsafe{
        let addr: SocketAddr = ADDR_STR.parse().unwrap();
        volo_gen::volo::example::ItemServiceClientBuilder::new(service_name: "volo-example") ClientBuilder<Identity, Identity, ..., ..., ..., ..., ...>
        .layer_outer(layer: LogLayer) ClientBuilder<Identity, Stack<..., ...>, ..., ..., ..., ..., ...>
        .address(target: addr) ClientBuilder<Identity, Stack<..., ...>, ..., ..., ..., ..., ...>
        .build()
    };
    let req: GetItemRequest = volo_gen::volo::example::GetItemRequest { op:opstr.into(), key:key.into(), value:value.into(), life:0i32, otherport:" ".into()};
    //println!("3333");
    client.get_item(req).await.expect(msg: "Error in other()!")
}
```



主从&cluster

以set 127.0.0.1:8084 2 4为例
redis master server





主从&cluster

以set 127.0.0.1:8084 2 4为例
redis master server

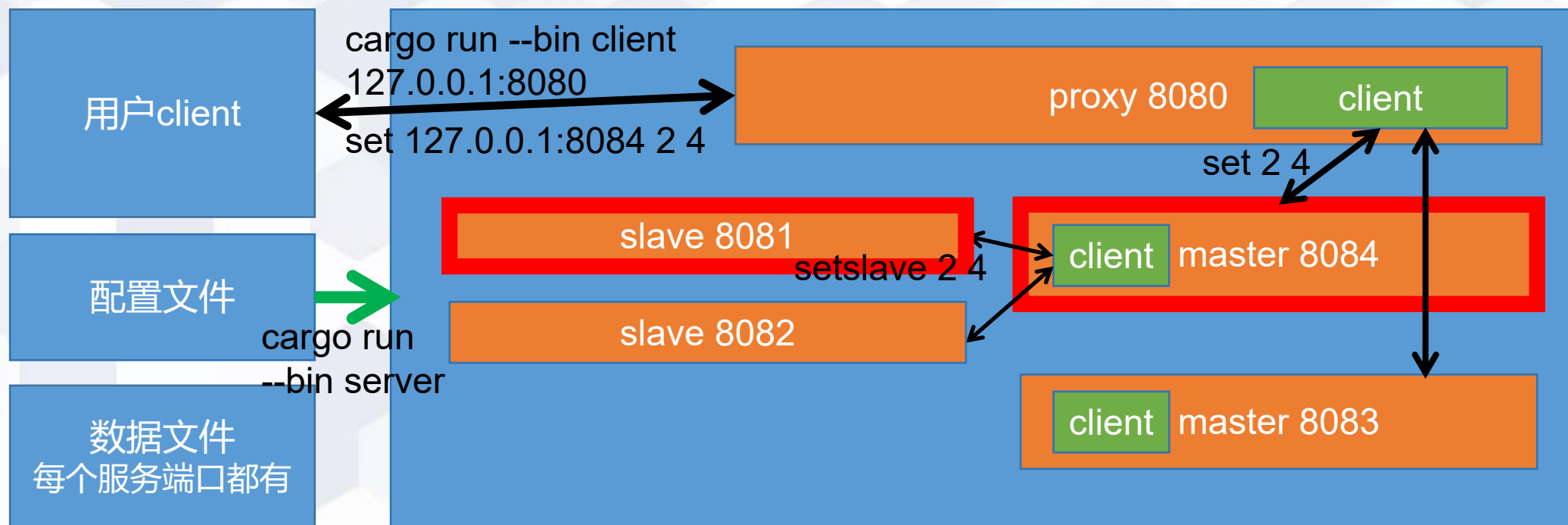
```
"set" => {
    //println!("{}", value);
    resp.op = "set".into();
    resp.key = key.clone().into();
    resp.value = value.clone().into();
    if self.master.read().unwrap().borrow()[0].eq("n"){
        //println!("{}", value);
        resp.op = "setslave".into();
        resp.value = self.master.read().unwrap().borrow()[1].clone().into();
        resp.state = false;
        return Ok(resp);
    }
    else if self.contents.read().unwrap().borrow().kvs.contains_key(&key){
        resp.value = self.contents.read().unwrap().borrow().kvs[&key].clone().into();
        resp.state = false;
        return Ok(resp);
    }
    else{
        self.contents.write().unwrap().borrow_mut().kvs.insert(k: key, v: value);
        resp.state = true;
        self.store().expect(msg: "Error in storing the data");
        let ports: Vec<String> = self.master.read().unwrap().borrow().clone();
        for s_ports: &String in ports.iter().skip(1) {
            let port: String = s_ports.clone();
            let key: FastStr = resp.key.clone();
            let value: FastStr = resp.value.clone();
            tokio::spawn(future: async move {
                slave(opstr: "setslave".to_string(), key: key.into_string(), value: value.into_string(), &port).await;
            });
        }
        return Ok(resp);
    }
}
```

```
"setslave" => {
    //println!("{}", self.port.read().unwrap().borrow(), key, value);
    self.contents.write().unwrap().borrow_mut().kvs.insert(k: key, v: value);
    resp.state = true;
    self.store().expect(msg: "Error in storing the data");
    return Ok(resp);
}
```



主从&cluster

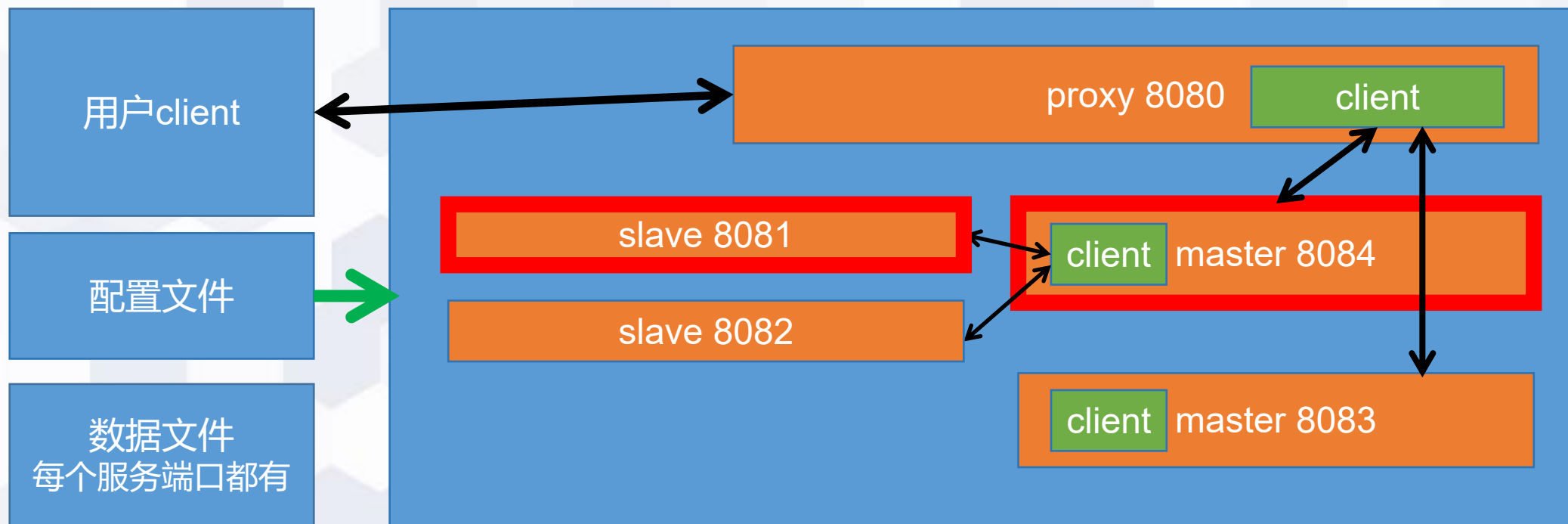
信息逐级返回最后由用户client处理





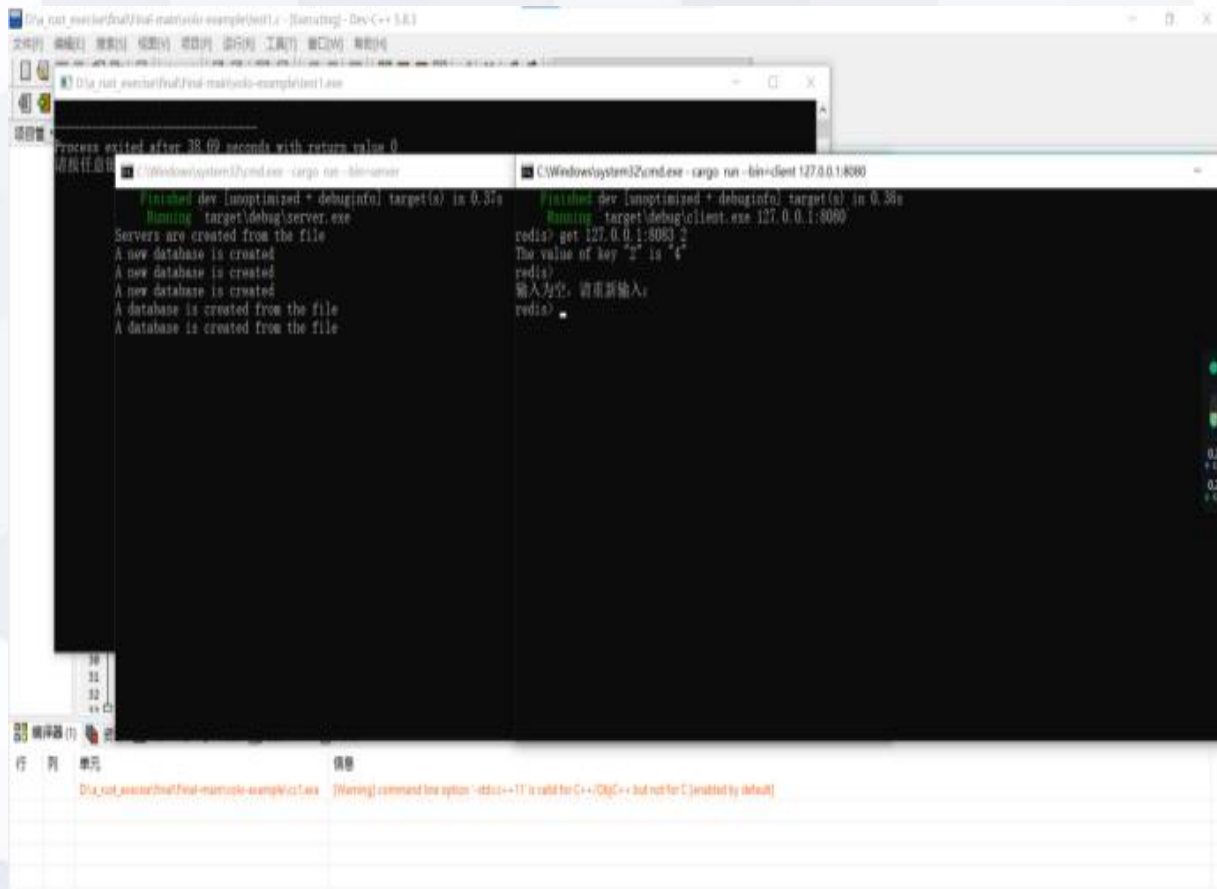
主从&cluster

以set 127.0.0.1:8084 2 4为例
redis master server





测试结果展示



```
D:\a_rui_workspace\final-main\redis-example\test1.a - [Running] - Dev-C++ 5.8.3
D:\a_rui_workspace\final-main\redis-example\test1.a
Process exited after 38.09 seconds with return value 0
请按任意键继续. . .
Finished dev (unoptimized + debuginfo) target(s) in 0.37s
Running target/debug/server.exe
Servers are created from the file
A new database is created
A new database is created
A new database is created
A database is created from the file
A database is created from the file
Finished dev (unoptimized + debuginfo) target(s) in 0.38s
Running target/debug/client.exe 127.0.0.1:8080
redis> get 127.0.0.1:8080
The value of key "x" is "x"
redis>
输入为空, 请重新输入:
redis>
```

1.AOF (Append-only File) 实现持久化
打开client和server, client发送set操作, 然后重启, 再用 client 进行 Get 调用, 可以访问到之前的数据。



测试结果展示

```
C:\Windows\system32\cmd.exe - cargo run --bin=server
Finished dev [unoptimized + debuginfo] target(s) in 0.39s
Running target/debug/server.exe
A database is created from the file
A database is created from the file
A database is created from the file
A database is created from the file
redis> set 127.0.0.1
redis> set 127.0.0.1:8080 4
The value of key "2" is "4"
redis>
输入为空, 请重新输入:
redis> set 127.0.0.1:8084 4
The value of key "2" is "4", which already exists
redis>
输入为空, 请重新输入:
redis> get 127.0.0.1:8084 2
The value of key "2" is "4"
redis>
输入为空, 请重新输入:
redis>
输入为空, 请重新输入:
redis>
```

```
C:\Windows\system32\cmd.exe - cargo run --bin=server
Running target/debug/server.exe 127.0.0.1:8084
redis> set 127.0.0.1:8083 4 8
A dredis> you are not using the proxy, the proxy port is "127.0.0.1:8080"
A dredis>
输入为空, 请重新输入:
A dredis> get 127.0.0.1:8081 2
redis> you are not using the proxy, the proxy port is "127.0.0.1:8080"
redis>
输入为空, 请重新输入:
redis> shutdown
D:\a_rust_exe\final\Final-main\vol-0-example>
D:\a_rust_exe\final\Final-main\vol-0-example> cargo run --bin=client 127.0.0.1:8080
Finished dev [unoptimized + debuginfo] target(s) in 0.37s
Running target/debug/client.exe 127.0.0.1:8080
redis>
输入为空, 请重新输入:
redis> get 127.0.0.1:8081 2
The value of key "2" is "4"
redis>
输入为空, 请重新输入:
redis> set 127.0.0.1:8083 4 8
Successfully inserted
redis>
输入为空, 请重新输入:
redis> get 127.0.0.1:8083 4
The value of key "4" is "8"
redis>
输入为空, 请重新输入:
redis>
```

2.Redis 主从架构

从节点对于 Set 操作返回错误。
对主节点进行 Set 操作，可以通过从节点通过 Get 获取到设置的数据。

3.Redis cluster

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。

The background features a large, light blue watermark of the Zhejiang University seal. The seal is circular, with the university's name in Chinese characters '浙江大学' at the top and 'ZHEJIANG UNIVERSITY' at the bottom. The founding year '1897' is inscribed in the center. Overlaid on this seal is the large, bold Chinese text '谢谢观看' (Thank you for watching).

谢谢观看

T H A N K S