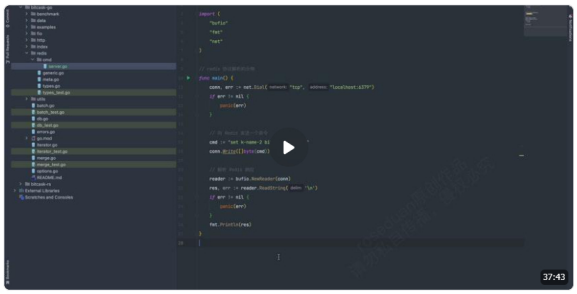
25 从零实现 KV 存储—兼容 Redis 协议

理论讲解

Go 编码



Rust 编码

前面的几节内容，主要讲述了如何在我们的 KV 存储引擎的基础之上，去兼容和实现 Redis 的数据结构，我们实现了 Redis 中最常用的五种数据结构，分别是 String、Hash、Set、List、Sorted Set。

在这基础之上，我们可以更进一步，去兼容一下 Redis 的协议，这样做的好处有很多，比如可以使用 Redis 的官方 SDK，通过不同的语言来访问我们的存储引擎。

也可以让我们的项目使用者，可以无缝的从 Redis 切换过来，保证高度的兼容性。

Redis 的通信协议，即 RESP（Redis Serialization Protocol）是一种简单、高效的二进制协议，用于在客户端和 Redis 服务器之间进行通信。

RESP 协议设计得非常简洁和易于实现，它使用文本的格式，并支持多种数据类型，例如常见的以下几种：

1. 简单字符串（Simple Strings）：以"+"符号开头，后跟字符串内容。例如，"+OK\r\n"表示一个简单字符串回复，内容为"OK"。
2. 错误消息（Errors）：以"-"符号开头，后跟错误信息。例如，"-Error occurred\r\n"表示一个错误消息回复，内容为"Error occurred"。
3. 整数（Integers）：以":"符号开头，后跟一个整数。例如，":100\r\n"表示一个整数回复，值为 100。
4. 多行字符串（Bulk Strings）：以"$"符号开头，后跟一个字符串长度（字节数），然后是字符串内容。例如，"$5\r\nHello\r\n"表示一个长度为 5 的多行字符串，内容为"Hello"。
5. 数组（Arrays）：以"*"符号开头，后跟一个整数表示数组中元素的个数，然后是数组的元素。数组可以包含任意RESP 数据类型，包括简单字符串、错误消息、整数、多行字符串和嵌套数组。

RESP 协议的设计使得它易于解析和生成，且在网络传输中占用的字节数相对较少。它还支持批量操作和异步通信，使得 Redis 能够高效地处理大量请求并快速响应。

> 更多详细内容可以参考 Redis 的官方文档：
>
> https://redis.io/docs/reference/protocol-spec/#:~:text=Redis%20clients%20use%20a%20protocol,Simple%20to%20implement.
> RESP protocol spec
> Redis serialization protocol (RESP) specification

作为开发者，我们可以去解析 RESP 协议，将我们存储引擎中的接口的数据，转换为符合 RESP 格式的数据，这样的话就可以和 Redis 的客户端进行交互了。

下面我以一个例子作为演示：

```go
func main() {
    conn, err := net.Dial("tcp", "localhost:6379")
    if err != nil {
        panic(err)
    }
    defer conn.Close()

    // 发送一个命令
    cmd := "set kv bitcask-storage-yyds\r\n"
    conn.Write([]byte(cmd))

    // 解析响应
    reader := bufio.NewReader(conn)
    res, err := reader.ReadString('\n')
    fmt.Println(res, err)
}
```

需要说明的是，由于 Redis 协议支持的数据类型较多，在课程当中，就不一一给大家去介绍了，大家感兴趣的话，可以顺着这个例子中的思路，自己去实现一些 Redis 的协议。

这里给大家提供一些很好的学习资料：

https://www.cnblogs.com/Finley/p/11923168.html

https://github.com/HDT3213/godis

课程当中，我们将会使用一些现成的框架来完成编码，因为实际上 RESP 协议的解析是一个非常普遍的需求，有很多开发者在 Github 上有一些开源的实现，比如这个库

https://github.com/tidwall/redcon

https://github.com/tidwall/redcon.rs

这个库实现的原理实际上和上面的示例类似，只是有更加完善的封装，支持更多的协议数据类型，所以我们可以直接使用。