本章作业

使用bufio包给TCPConn增加Buffer

需求

- TCP协议是流式的，原生并没有消息的分割
- 应用层往往会指定消息的分割符，比如“\n”
- 要实现基于TCP的应用层协议，需要分割消息

bufio.Reader

- bufio包的Reader工具可以暂存Socket收到的所有数据
- 并将这些数据中我们所需的部分返回给我们

Reader.ReadString()

- Reader的ReadString方法可以暂存Socket收到的数据
- 并将我们指定的第一个分隔符之前的部分返回

```
package main

import (
    "fmt"
    "net"
    "io"
    "log"
    "bufio"
)

func ListenAndServe(address string) {
    // 绑定监听地址
    listener, err := net.Listen("tcp", address)
    if err != nil {
        log.Fatal(fmt.Sprintf("listen err: %v", err))
    }
    defer listener.Close()
    log.Println(fmt.Sprintf("bind: %s, start listening...", address))

    for {
        // Accept 会一直阻塞直到有新的连接建立或者listen中断才会返回
        conn, err := listener.Accept()
        if err != nil {
            // 通常是由于listener被关闭无法继续监听导致的错误
            log.Fatal(fmt.Sprintf("accept err: %v", err))
        }
        // 开启新的 goroutine 处理该连接
        go Handle(conn)
    }
}

func Handle(conn net.Conn) {
    // 使用 bufio 标准库提供的缓冲区功能
    reader := bufio.NewReader(conn)
    for {
        // ReadString 会一直阻塞直到遇到分隔符 '\n'
        // 遇到分隔符后会返回上次遇到分隔符或连接建立后收到的所有数据，包括分隔符本身
        // 若在遇到分隔符之前遇到异常，ReadString 会返回已收到的数据和错误信息
        msg, err := reader.ReadString('\n')
        if err != nil {
            // 通常遇到的错误是连接中断或被关闭，用io.EOF表示
            if err == io.EOF {
                log.Println("connection close")
            } else {
                log.Println(err)
            }
            return
        }
        b := []byte(msg)
        // 将收到的信息发送给客户端
        conn.Write(b)
    }
}
```

```
func main() {
    ListenAndServe(":8000")
}
```