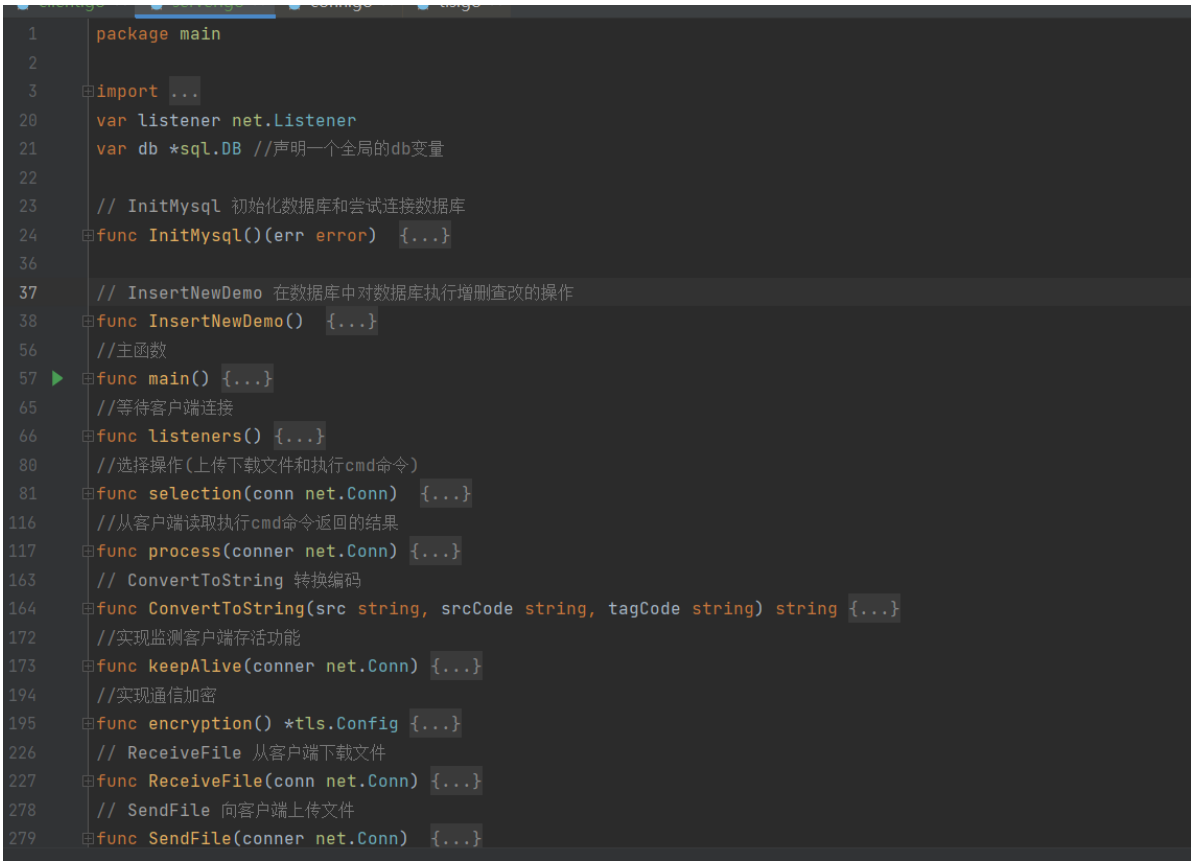
ReadMe

经过和困难：

经过二十几天的学习，我终于完成了考核，我完成的功能有基本功能，文件浏览，上传下载，连接数据库，通信加密。每个功能都由一个函数完成，程序大概结构为



这个listener全局变量有点怪，在windows系统上为net.listener可以跑起来，但是在linux系统上得为*tls.listener才可以，并且我打印出来的类型也为*tls.listener,就挺奇怪的。我用的加密通信协议为tls协议，当程序检测到由主机连接上来时就会进入selection函数

```
        //downfile向远程主机传输文件
        fmt.Print( a...: "downfile,向远程主机传输文件；\n")
        //upfile从远程主机下载文件
        fmt.Print( a...: "upfile,从远程主机下载文件；\n")
        //cmd执行cmd命令
        fmt.Print( a...: "cmd,执行其他命令；\n")
        fmt.Print( a...: "heartbeat,开启心跳功能；\n")
        fmt.Print( a...: "quit,退出连接！\n")
        fmt.Println( a...: "请输入要执行的操作：")
        input := bufio.NewReader(os.Stdin)
        s, _ := input.ReadString( delim: '\n')
        s=strings.Trim(s, cutset: "\r\n")
        if s=="downfile" {
            _, _= conn.Write([]byte(s))
            SendFile(conn)
        } else if s=="upfile" {
            _, _= conn.Write([]byte(s))
            ReceiveFile(conn)
        } else if s == "cmd" {
            _, _= conn.Write([]byte(s))
            process(conn)
        } else if s == "quit" {
            _, _= conn.Write([]byte(s))
            _ = conn.Close()
            return
        }else if s=="heartbeat" {
            go keepAlive(conn)
            fmt.Print( a...: "心跳功能开启成功，请继续输入操作！\n")
        } else {
```

这里有个小细节，因为我的心跳功能是这样写的

```
func keepAlive(conner net.Conn) {
    buf:=make([]byte,4096)
    var i = 0
    for  {
        //空闲时间10秒
        time.Sleep(10*time.Second)
        for  {
            _,err1:=conner.Read(buf)
            if err1 != nil {
                //重连间隔5秒
                time.Sleep(5*time.Second)
                i+=1
            }
            //重连次数为3
            if i==3 {
                fmt.Print( a...: "啊偶，主机掉线啦！\n")
                //退出该协程
                runtime.Goexit()
            }
        }
    }
}
```

它从连接中读取数据，通过判断err是否为空来判断连接是否断开，因为我是开的一个协程来实现心跳功能的，由于他要从连接中读取数据，然后执行cmd命令，从远程主机下载文件也需要从连接中读取数据，所以如果在我执行命令和下载文件的时候心跳功能就会和其他函数争抢数据，就会导致下载文件和执行cmd命令的结果打印不出来，那么我这样写的话，就可以人为的避免这样的结果(其实golang可有个很方便的第三方库来开启tcp的keepalive，但它开启过后并不会主动报告连接情况，所以我还是自己弄

了一个)，因为golang的默认编码为gbk,并不是utf-8，所以我就用了第三方库编写了转换编码的函数。

```go
func encryption() *tls.Config {
    //通过私钥和证书得到crt变量
    crt,err:=tls.LoadX509KeyPair( certFile: "server.pem", keyFile: "server.key")
    if err != nil {
        fmt.Println( a...: "err:",err)
        return nil
    }
    //从client.pem读取内容并将其赋值给certbytes变量
    certBytes, err1 := ioutil.ReadFile( filename: "client.pem")
    if err1 != nil {
        fmt.Println( a...: "Unable to read cert.pem")
        return nil
    }
    //定义clientCertPool为一个空的certpool结构体
    clientCertPool:=x509.NewCertPool()
    //调用clientCertPool的AppendCertsFromPEM方法来判断client.pem证书是否有效
    ok := clientCertPool.AppendCertsFromPEM(certBytes)
    if !ok {
        fmt.Println( a...: "failed to parse root certificate")
        return nil
    }
    //生成tls.Config对象
    config :=&tls.Config{
        //证书crt
        Certificates: []tls.Certificate{crt},
        //验证客户端
        ClientAuth: tls.RequestClientCert,
        ClientCAs:  clientCertPool,
    }
```

这是我所用的加密方式，证书和密钥都是用openssl生成的。在实现选择操作的时候，遇到的最大的问题就是字符串匹配，golang在终端输入了字符后会默认加个\r,就导致开始的时候怎么样都匹配不上。。。。。后来用了string包里面的trim方法就解决掉了这个问题。

以上就是我做考核的时候遇到的我觉得最难受的问题。

收获：

总的收获还是挺大的，在做通信的时候就觉得计网实在是太重要了，很有必要认真的学一下，还有就是学到了golang的一些用法，感觉golang确实很方便。有很多的包，极大的减少了我们的工作量。