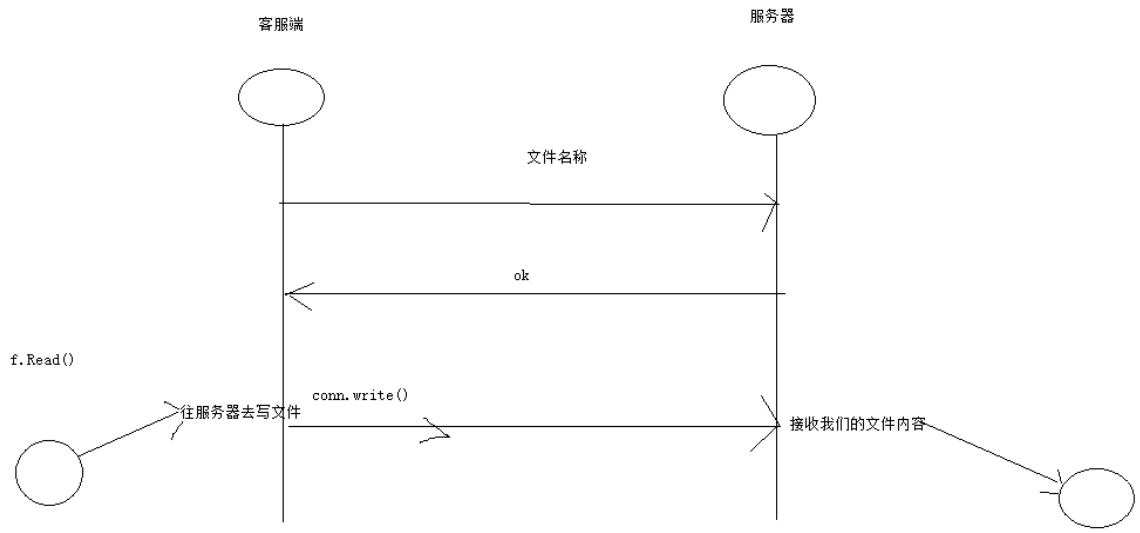
文件传输



1）服务器端代码

```
package main

import (
   "net"
   "fmt"
   "os"
   "io"
)
//文件的接收操作
func RecvFile(filename string,conn net.Conn)  {
   //新建文件
   f,err :=  os.Create(filename)
   if err != nil{
       fmt.Println("os.Create err=",err)
       return
   }

   buf := make([]byte,1024)
   for{
        n,err := conn.Read(buf) //接收对方发送过来的文件内容
        if err != nil{
          if err == io.EOF{
             fmt.Println("文件接收完毕")
      }else{
         fmt.Println("conn.Read err=",err)
```

```
        }
        return
    }
    if n == 0{
        fmt.Println("n==0 文件接收完毕")
        break
    }
    f.Write(buf[:n])
  }
}

func main()  {
  //监听
  listernner,err := net.Listen("tcp","127.0.0.1:8000")
  if err != nil{
     fmt.Println("net.Listen err =",err)
     return
  }

  defer  listernner.Close()
   //阻塞等待用户连接
  conn,err := listernner.Accept()
  if err != nil{
     fmt.Println("listenner.Accept err=",err)
     return
  }
  defer  conn.Close()

  //缓冲
  buf := make([]byte,1024)

   n,err := conn.Read(buf)
   if err != nil{
     fmt.Println("conn.Read err =",err)
     return
   }

   filename := string(buf[:n])
   //回复ok
   conn.Write([]byte("ok"))

   //接收文件内容
   RecvFile(filename,conn)



}
```

2）客户端代码

```
package main
```

```
import (
    "fmt"
    "os"
    "net"
    "io"
)

//发送文件内容
func SendFile(path string,conn net.Conn){
    //以只读的方式打开文件
    f,err  := os.Open(path)
    if err != nil{
        fmt.Println("os.Open err=",err)
        return
    }

    defer f.Close()

    buf := make([]byte,1024*4)
    for{
        n,err := f.Read(buf)
        if err !=nil{
            if err == io.EOF{
                fmt.Println("文件传输完毕")
            }else{
                fmt.Println(" f.Read err=",err)
            }
            return
        }
        //发送内容
        conn.Write(buf[:n])
    }


}

func main()  {
    //提示输入文件
    fmt.Println("请输入需要传输的文件")
    var path string
    fmt.Scan(&path)

    //获取文件名 info.Name()
    info,err := os.Stat(path)

    if err != nil{
        fmt.Println("os.Stat err= ",err)
        return
    }

    //主动连接我们的服务器
    conn,err := net.Dial("tcp","127.0.0.1:8000")
```

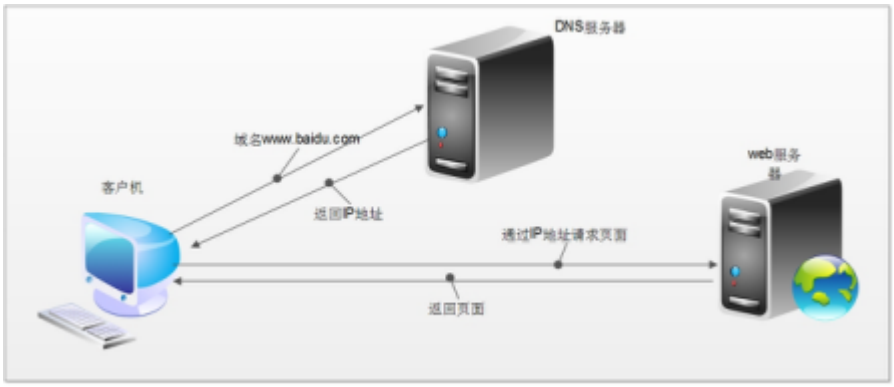
```
    if err !=nil{
        fmt.Println("net.Dial err=",err)
        return
    }

    //给接收方发送文件名
    _,err = conn.Write([]byte(info.Name()))
    if err != nil{
        fmt.Println("conn.Write err =",err)
        return
    }

    var n int
    buf := make([]byte,1024)
    n,err = conn.Read(buf)
    if err !=nil{
        fmt.Println("conn.Read err=",err)
        return
    }

    if "ok"== string(buf[:n]){
        //发送文件内容
        SendFile(path,conn)
    }




}
```

HTTP编程

概述

我们平时浏览网页的时候,会打开浏览器，输入网址后按下回车键，然后就会显示出你想要浏览的内容。在这个看似简单的用户行为背后，到底隐藏了些什么呢？

对于普通的上网过程，系统其实是这样做的：浏览器本身是一个客户端，当你输入URL的时候，首先浏览器会去请求DNS服务器，通过DNS获取相应的域名对应的IP，然后通过IP地址找到IP对应的服务器后，要求建立TCP连接，等浏览器发送完HTTP Request（请求）包后，服务器接收到请求包之后才开始处理请求包，服务器调用自身服务，返回HTTP Response（响应）包；客户端收到来自服务器的响应后开始渲染这个Response包里的主体（body），等收到全部的内容随后断开与该服务器之间的TCP连接。



一个Web服务器也被称为HTTP服务器，它通过HTTP协议与客户端通信。这个客户端通常指的是Web浏览器(其实手机端客户端内部也是浏览器实现的)。

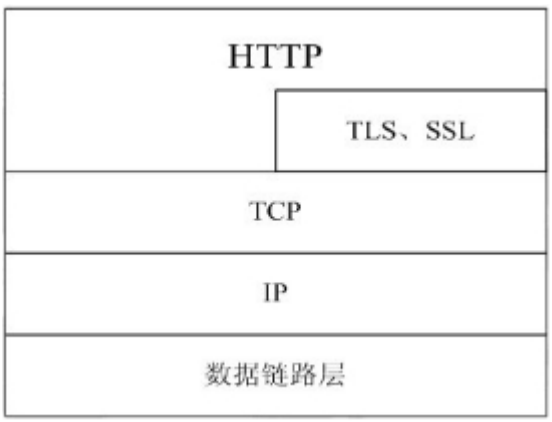
Web服务器的工作原理可以简单地归纳为：

- 客户机通过TCP/IP协议建立到服务器的TCP连接
- 客户端向服务器发送HTTP协议请求包，请求服务器里的资源文档
- 服务器向客户机发送HTTP协议应答包，如果请求的资源包含有动态语言的内容，那么服务器会调用动态语言的解释引擎负责处理“动态内容”，并将处理得到的数据返回给客户端
- 客户机与服务器断开。由客户端解释HTML文档，在客户端屏幕上渲染图形结果

HTTP协议

超文本传输协议(HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议，它详细规定了浏览器和万维网服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

HTTP协议通常承载于TCP协议之上，有时也承载于TLS或SSL协议层之上，这个时候，就成了我们常说的HTTPS。如下图所示：



url地址

URL全称为Unique Resource Location，用来表示网络资源，可以理解为网络文件路径。

URL的格式如下：

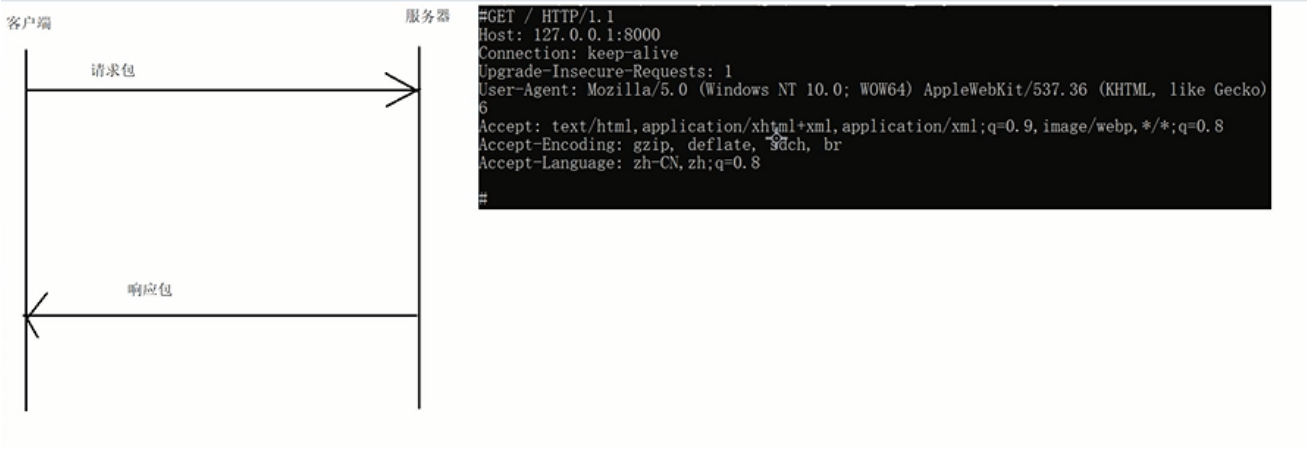
http://host[":"port][abs_path]

http://192.168.31.1/html/index?name=

URL的长度有限制，不同的服务器的限制值不太相同，但是不能无限长。（255）

由于URL长度有限：post get

请求方式




```
func main() {
    //监听
    listener, err := net.Listen("tcp", ":8000")
    if err != nil {
        fmt.Println("Listen err = ", err)
        return
    }

    defer listener.Close()

    //阻塞等待用户的连接
    conn, err1 := listener.Accept()
    if err1 != nil {
        fmt.Println("Accept err1 = ", err1)
        return
    }

    defer conn.Close()

    //接收客户端的数据
    buf := make([]byte, 1024*4)
    n, err2 := conn.Read(buf)
    if n == 0 {
        fmt.Println("Read err = ", err2)
        return
    }

    fmt.Printf("#%v#", string(buf[:n]))

}
```

```
#GET / HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/66.0.3359.170 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.9
Cookie: csrftoken=lz2jgANWEUMIqnsWoiiW4wvHpFTTR5sKDjxYPWOT9gHpSSNJMXBFxNy3vlQI1SL4;
login=P_-HAwEBDkNvb2tpZVJlbWVtYmVyAf-
IAAEDAQhNZW1iZXJJZAEEAAEHQWNjb3VudAEMAAEEVGltZQH_hgAAABD_hQUBAQRUaW1lAf-GAAAAHf-
IAQIBBWFkbWluAQ8BAAAADtMq2mwM8AlAAeAA|1536746348217057600|5e8227Od823998dcf60b1a98ce8e0
266fec3e599
Accept-Encoding: gzip, deflate

#
```

请求报文格式说明

HTTP 请求报文由请求行、请求头部、空行、请求包体4个部分组成，如下图所示：

1) 请求行

请求行由方法字段、URL 字段 和HTTP 协议版本字段 3 个部分组成，他们之间使用空格隔开。常用的 HTTP 请求方法有 GET、POST。

GET：

- 当客户端要从服务器中读取某个资源时，使用GET 方法。GET 方法要求服务器将URL 定位的资源放在响应报文的数据部分，回送给客户端，即向服务器请求某个资源。

- 使用GET方法时，请求参数和对应的值附加在 URL 后面，利用一个问号(“?”)代表URL 的结尾与请求参数的开始，传递参数长度受限制，因此GET方法不适合用于上传数据。

  http://www.baidu.com?name="zhangsan"&age="12"

- 通过GET方法来获取网页时，参数会显示在浏览器地址栏上，因此保密性很差。

POST：

- 当客户端给服务器提供信息较多时可以使用POST 方法，POST 方法向服务器提交数据，比如完成表单数据的提交，将数据提交给服务器处理。
- GET 一般用于获取/查询资源信息，POST 会附带用户数据，一般用于更新资源信息。POST 方法将请求参数封装在HTTP 请求数据中，而且长度没有限制，因为POST携带的数据，在HTTP的请求正文中，以名称/值的形式出现，可以传输大量数据。

2) 请求头部

请求头部为请求报文添加了一些附加信息，由“名/值”对组成，每行一对，名和值之间使用冒号分隔。

请求头部通知服务器有关于客户端请求的信息，典型的请求头有：

请求头	含义
User-Agent	请求的浏览器类型
Accept	客户端可识别的响应内容类型列表，星号“ * ”用于按范围将类型分组，用“ / ”指示可接受全部类型，用“ type/* ”指示可接受 type 类型的所有子类型
Accept-Language	客户端可接受的自然语言
Accept-Encoding	客户端可接受的编码压缩格式
Accept-Charset	可接受的应答的字符集
Host	请求的主机名，允许多个域名同处一个IP 地址，即虚拟主机
connection	连接方式(close或keepalive)
Cookie	存储于客户端扩展字段，向同一域名的服务端发送属于该域的cookie

3) 空行

最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。

4) 请求包体

请求包体不在GET方法中使用，而是POST方法中使用。

POST方法适用于需要客户填写表单的场合。与请求包体相关的最常使用的是包体类型Content-Type和包体长度Content-Length。

响应报文格式

1)测试服务器

```
package main

import (
    "fmt"
    "net/http"
)

//服务端编写的业务逻辑处理程序
func myHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "hello world")
}

func main() {
    http.HandleFunc("/go", myHandler)

    //在指定的地址进行监听，开启一个HTTP
    http.ListenAndServe("127.0.0.1:8000", nil)
}
```

2)响应报文分析

```
package main

import (
    "fmt"
    "net"
)

func main() {
    //主动连接服务器
    conn, err := net.Dial("tcp", ":8000")
    if err != nil {
        fmt.Println("dial err = ", err)
        return
    }

    defer conn.Close()
```

```
    requestBuf := "GET /go HTTP/1.1\r\nAccept: image/gif, image/jpeg, image/pjpeg,
application/x-ms-application, application/xaml+xml, application/x-ms-xbap,
*/*\r\nAccept-Language: zh-Hans-CN,zh-Hans;q=0.8,en-US;q=0.5,en;q=0.3\r\nUser-Agent:
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C;
.NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)\r\nAccept-
Encoding: gzip, deflate\r\nHost: 127.0.0.1:8000\r\nConnection: Keep-Alive\r\n\r\n"

    //先发请求包，服务器才会回响应包
    conn.Write([]byte(requestBuf))

    //接收服务器回复的响应包
    buf := make([]byte, 1024*4)
    n, err1 := conn.Read(buf)
    if n == 0 {
        fmt.Println("Read err = ", err1)
        return
    }

    //打印响应报文
    fmt.Printf("#%v#", string(buf[:n]))

}
```

返回报文格式

```
#HTTP/1.1 200 OK
Date: Fri, 26 Oct 2018 13:39:46 GMT
Content-Length: 7
Content-Type: text/plain; charset=utf-8

gogogo
#
```

3)响应报文格式说明

HTTP 响应报文由状态行、响应头部、空行、响应包体4个部分组成，如下图所示：

1) 状态行

状态行由 HTTP 协议版本字段、状态码和状态码的描述文本3个部分组成，他们之间使用空格隔开。

状态码：

状态码由三位数字组成，第一位数字表示响应的类型，常用的状态码有五大类如下所示：

状态码	含义
1xx	表示服务器已接收了客户端请求，客户端可继续发送请求
2xx	表示服务器已成功接收到请求并进行处理
3xx	表示服务器要求客户端重定向
4xx	表示客户端的请求有非法内容
5xx	表示服务器未能正常处理客户端的请求而出现意外错误

常见的状态码举例：

状态码	含义
200 OK	客户端请求成功
400 Bad Request	请求报文有语法错误
401 Unauthorized	未授权
403 Forbidden	服务器拒绝服务
404 Not Found	请求的资源不存在
500 Internal Server Error	服务器内部错误
503 Server Unavailable	服务器临时不能处理客户端请求(稍后可能可以)

2) 响应头部

响应头可能包括：

响应头	含义
Location	Location响应报头域用于重定向接受者到一个新的位置
Server	Server 响应报头域包含了服务器用来处理请求的软件信息及其版本
Vary	指示不可缓存的请求头列表
Connection	连接方式

3) 空行

最后一个响应头部之后是一个空行，发送回车符和换行符，通知服务器以下不再有响应头部。

4) 响应包体

服务器返回给客户端的文本信息。

客户端与服务器代码

```
package main

import (
    "fmt"
    "net/http"
)

//w, 给客户端回复数据
//r, 读取客户端发送的数据
func HandConn(w http.ResponseWriter, r *http.Request) {
    fmt.Println("r.Method = ", r.Method)
    fmt.Println("r.URL = ", r.URL)
    fmt.Println("r.Header = ", r.Header)
    fmt.Println("r.Body = ", r.Body)

    w.Write([]byte("hello go")) //给客户端回复数据
}

func main() {
    //注册处理函数，用户连接，自动调用指定的处理函数
    http.HandleFunc("/", HandConn)

    //监听绑定
    http.ListenAndServe(":8000", nil)
}
```

客户端

```
package main

import (
    "fmt"
    "net/http"
)

func main() {

    resp, err := http.Get("http://127.0.0.1:8000")
    if err != nil {
        fmt.Println("http.Get err = ", err)
```

```
        return
    }

    defer resp.Body.Close()

    fmt.Println("Status = ", resp.Status)
    fmt.Println("StatusCode = ", resp.StatusCode)
    fmt.Println("Header = ", resp.Header)
    //fmt.Println("Body = ", resp.Body)

    buf := make([]byte, 4*1024)
    var tmp string

    for {
        n, err := resp.Body.Read(buf)
        if n == 0 {
            fmt.Println("read err = ", err)
            break
        }
        tmp += string(buf[:n])
    }

    fmt.Println("tmp = ", tmp)

}
```