技术

[1 标题1 错误!未定义书签。](#_Toc449540698)

[1.1 标题2 **错误!未定义书签。**](#_Toc449540699)

[1.1.1 标题3 **错误!未定义书签。**](#_Toc449540700)

# B/S HTTP

B/S 浏览器与服务器

服务器提供数据，处理数据。接收浏览器请求，为浏览器提供数据。

浏览器展示数据，向服务器申请数据，从服务器获取数据。



浏览器与服务器的交换标准，协议，规范，由HTTP控制。

HTTP：HyperText Transfer Protocol超文本传输协议。

超级文本：包括了数据和格式的文本格式。就是超文本，例如HTML。

控制：两个流程的数据格式：

浏览器向服务器发送请求的数据格式协议。

服务器向浏览器发送的响应格式协议。

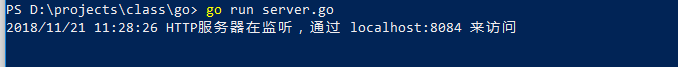
请求与响应的 数据格式大体结构一致，有一些具体的差异。

# 利用，搭建一个HTTP服务器

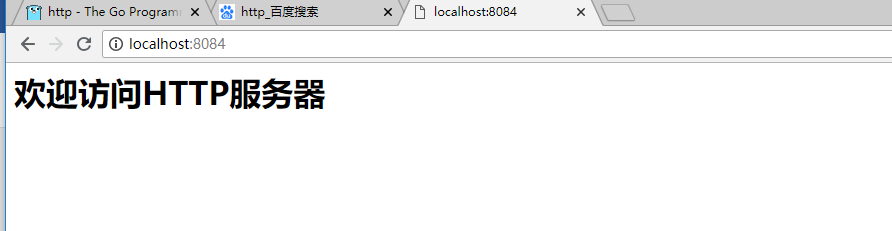
利用 http 包完成

net/http

|  |
| --- |
| **package** main  **import** (  "log"  "net/http"  )  **func** main() {  *// 设置 处理器*  http.HandleFunc("/", IndexAction)  *// 监听并提供服务*  log.Println("HTTP服务器在监听，通过 localhost:8084 来访问")  *// Fatal 将信息增加了错误的描述。 例如 err = 监听失败， Fatal 之后， 系统发生致命错误：监听失败*  log.Fatal(http.ListenAndServe(":8084", **nil**))  }  *// 首页动作*  **func** IndexAction(w http.ResponseWriter, r \*http.Request) {  w.Write([]byte("<h1>欢迎访问HTTP服务器</h1>"))  } |



浏览器访问：



# HTTP请求协议格式

浏览器在发出请求时，会依据 HTTP的请求格式来发送。将全部的请求数据划分为三个部分：

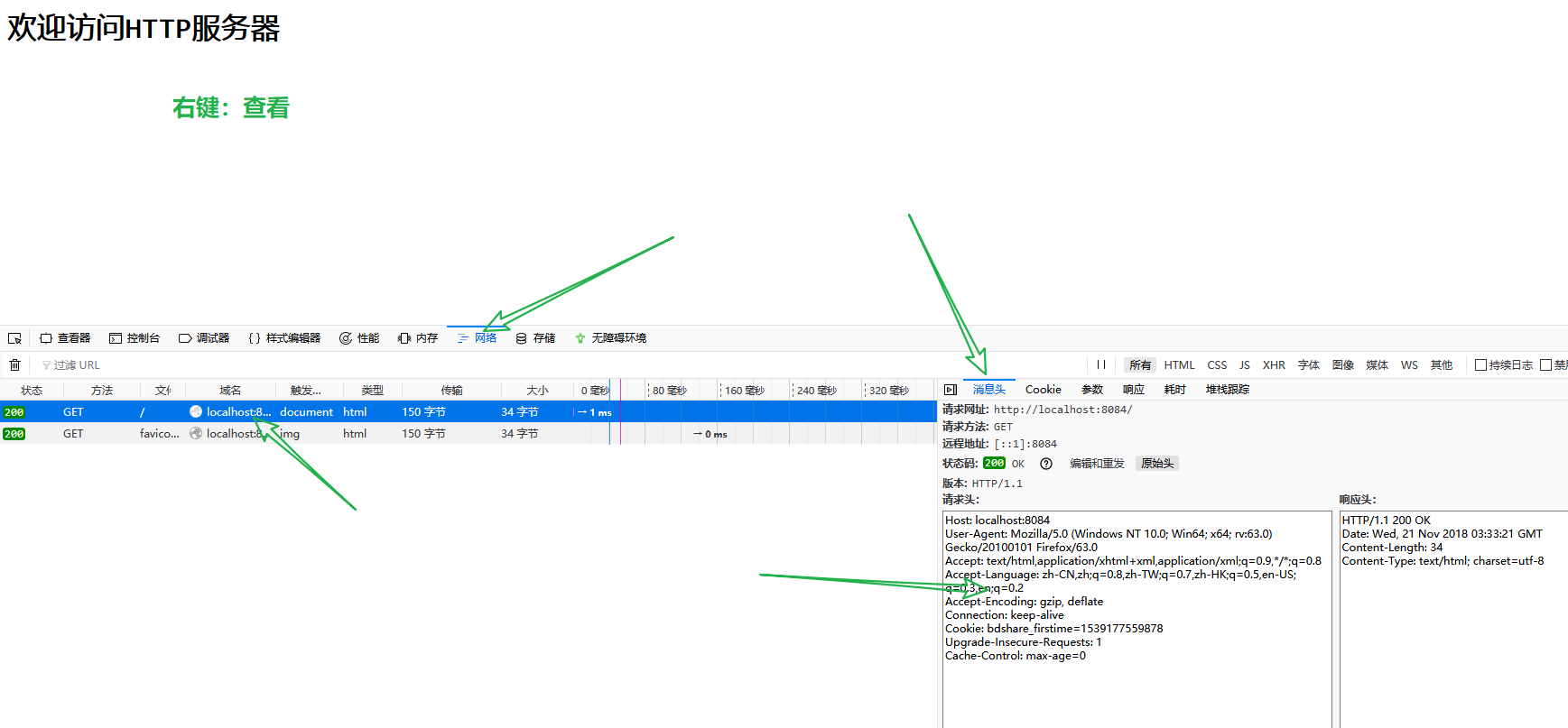
请求行

请求头

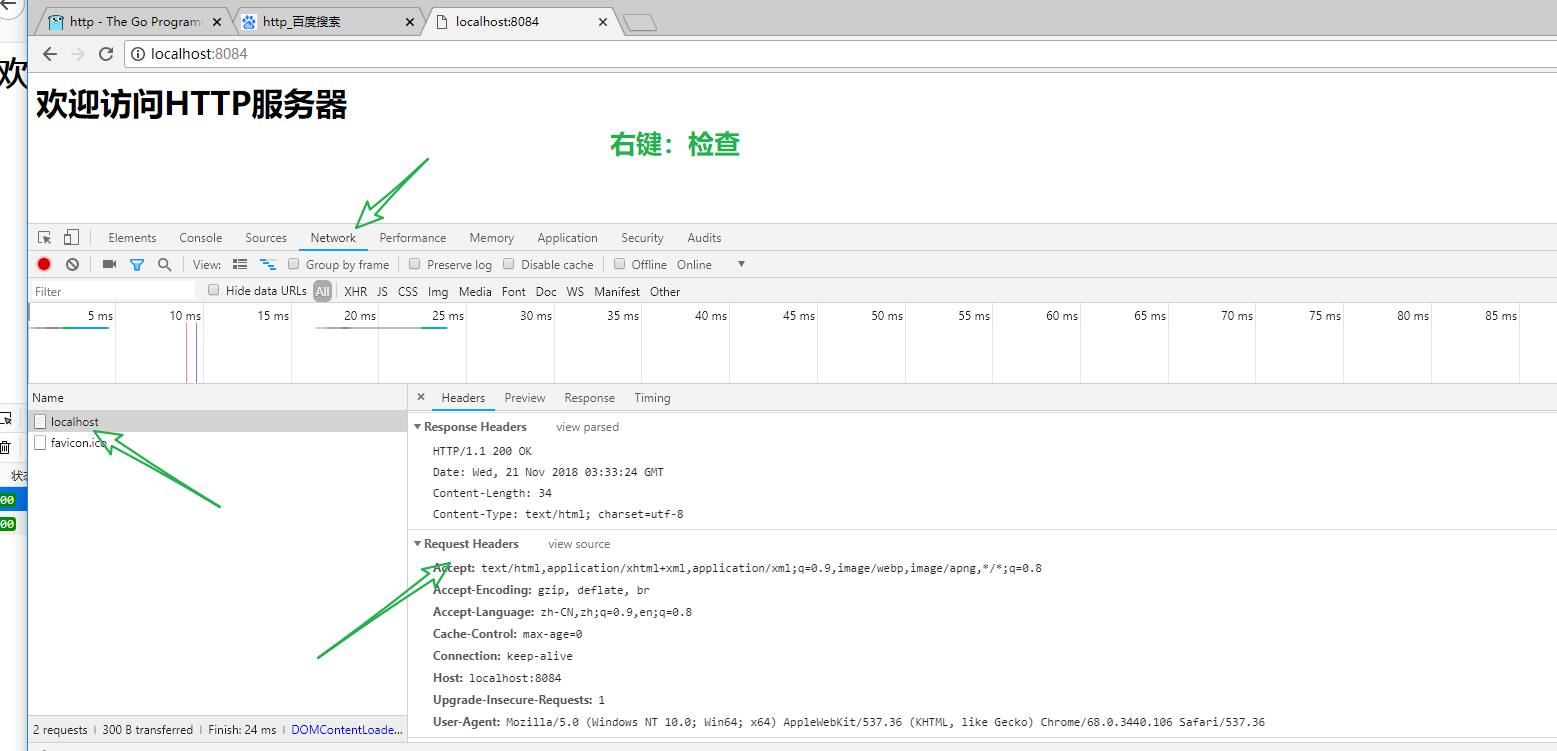
请求主体

使用浏览器提供的调试工具，可以清晰看到请求数据的全部：

firefox



chrome



获取的请求信息如下：

|  |
| --- |
| GET / HTTP/1.1  Host: localhost:8084  Connection: keep-alive  Cache-Control: max-age=0  Upgrade-Insecure-Requests: 1  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8  Accept-Encoding: gzip, deflate, br  Accept-Language: zh-CN,zh;q=0.9,en;q=0.8  空行 |

发出请求，就是整理出以上的数据，通过网络连接发送到HTTP服务器端。在浏览器地址栏一键回车，就完成了，整理数据和发送数据的操作。浏览器发出了请求。

可见，任何程序都可以发出HTTP请求。包括GO语言。

以上的数据就是本次请求的全部数据，由三部分构成：

请求行

请求头

请求主体

## 请求行（start-line）

请求数据的第一行。

GET / HTTP/1.1



用于描述：请求方式（GET），URI（统一资源标识符）（/），协议版本（HTTP/1.1）

### 请求方式，method

请求需要对服务器的资源做什么类型的操作。主要有：

GET，获取，查询

POST，新增

PUT，修改更新

DELETE，删除

.还有（HEAD， FETCH…）

浏览器发出的请求，主要是 POST（增删改） 和 GET（获取）， 浏览器不会发送 PUT 和 DELETE请求。

但是，通过不同的URI组合，告知服务器执行的是删除还是更新：

例如：

获取id为3的文章：get /article?id=3

新增文章，提供标题，内容作者等：post /article title=xxx content=xxxx author=xxx

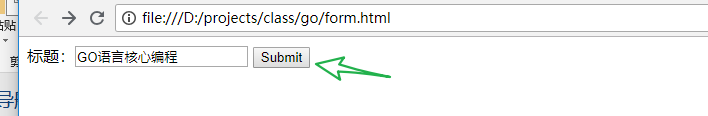
删除文章，get或者post 请求 /articleDelete?id=3 ，由 articleDelete 部分确定当前需要完成删除操作。

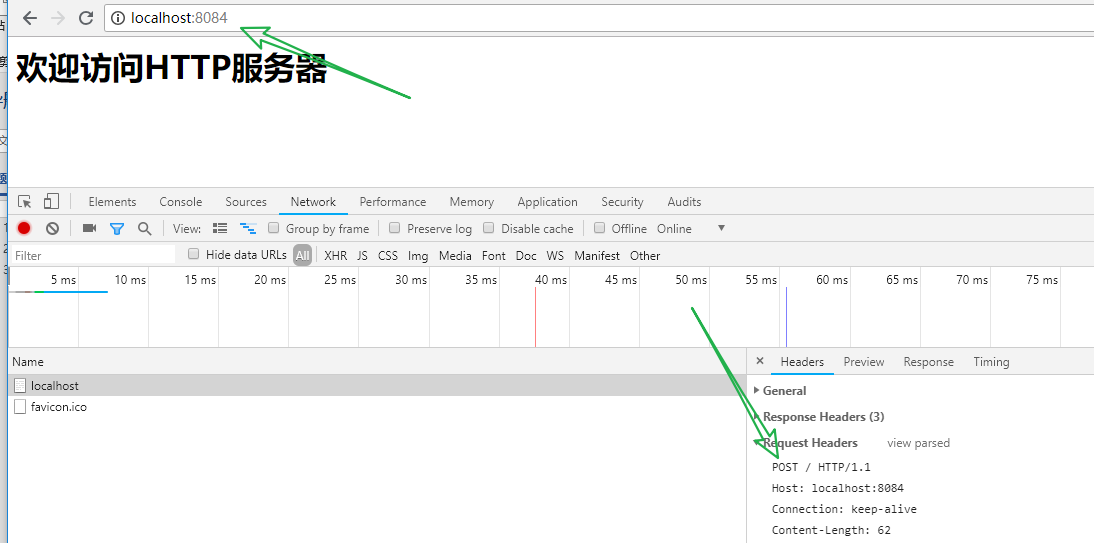
更新文章，post方式请求 /articleUpdate?id=3， 由articleUpdate来标识需要更新文章，同时提供更新的数据，与新增一致。

post最常见于，使用表单：

<form action=”” method=”post” >







地址栏中，键入地址，键入回车，此时的请求就是get请求。

### URI（统一资源标识符）

请求行的第二部分

用于标识所请求的资源。

另一个词：URL，统一资源定位符。与URI类似：

URL: <http://localhost:8084/article?id=3>

URI: /article?id=3

URL, 在网络上标识一个资源的位置。因此需要指定域名主机部分。

URI，在主机上标识资源的位置。相当于主机（域名）已经确定了，因此仅仅需要资源标识部分即可。

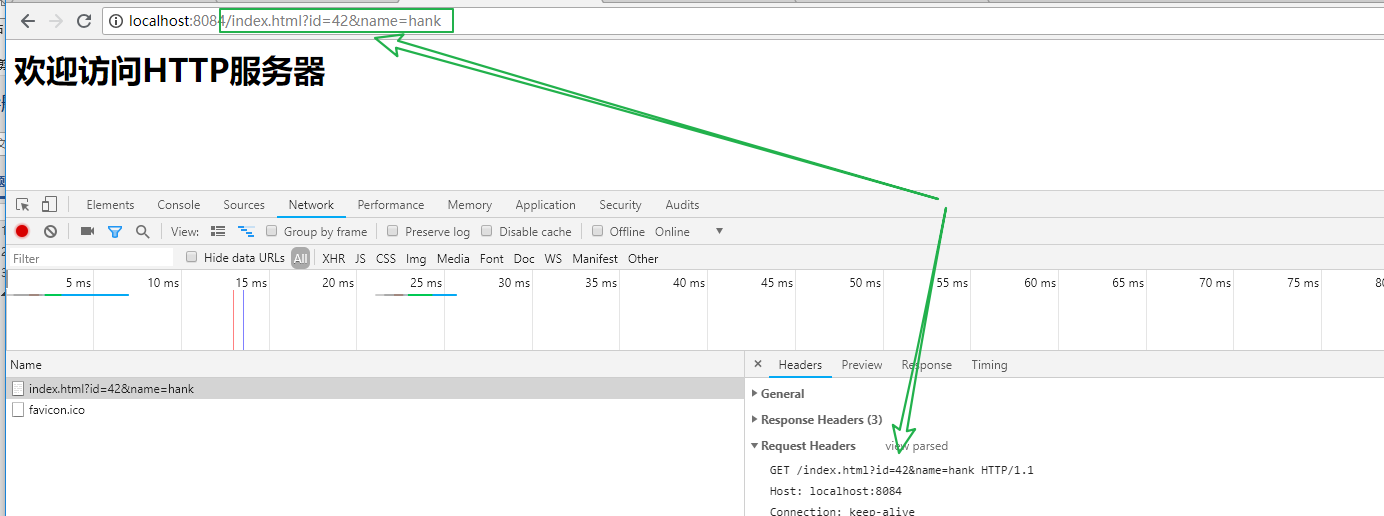
资源标识部分从 / 站点主机根开始，指向的某个资源位置。

典型的由：脚本地址 和 查询参数构成。

index.html 脚本地址

id=42 查询参数，查询参数可以是多个，id=42&name=hank&…

脚本和查询参数间，使用问号分割 ?



### 协议版本

HTTP/1.1

表示使用的是 1.1版本的HTTP。目前使用最广泛。

还可能为：

0.9, 1.0, 1.1

2.0

在HTTP协议中规定，行结束使用 CRLF (\r\n, 回车换行)作为行结束符(与操作系统无关)。

GET / HTTP/1.1\r\n



除了请求行使用CRLF结束外，请求头的每一行，也是用CRLF来结束。在HTTP协议中，任何行结束都是用CRLF。

## 请求头，请求头信息，请求标头，request header

由请求数据的第二行开始（在请求行后），到一个空行结束的全部行，都是请求头信息部分。



头信息是典型的Key:value 型，key: value.

下表:(空格)值

是请求的属性（不是数据），用于描述这个请求，或者是浏览器信息的。例如：User-Agent 头，表示发出该请求的浏览器是什么？



一个请求典型的请求头意义如下：

|  |  |  |
| --- | --- | --- |
| Key | 意义 | 典型value值 |
| Host | 请求的主机，由两个部分构成。域名:端口。没有端口，默认表示80端口 | localhost:8084 |
| Connection | 连接类型。  使用 keep-alive表示支持长连接，在请求结束后，一段时间内，不释放tcp连接，下次请求时继续使用。使用 close 表示，请求结束后连接立即释放。 | Keep-alive  close |
| User-Agent | 用户代理，用于描述发出http请求浏览器端信息。 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36 |
| Accept | 发出请求的浏览器，可以接受的信息编码。用于指导服务器依据该编码响应数据。  建议服务器而不是命令服务器 | text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8 |
| Accept-Encoding | 发出请求的浏览器，可以接受的数据压缩方法。用于指导服务器依据该压缩方法响应数据。  建议服务器而不是命令服务器 | gzip, deflate, br |
| Accept-Language | 发出请求的浏览器，可以接受的语言类型。用于指导服务器依据该语言响应数。  在做多语言程序，汉语地区访问页面是韩语的，语言地区范围页面是日语的。  建议服务器而不是命令服务器 | zh-CN,zh;q=0.9,en;q=0.8 |
| Cookies | 请求携带的Cookie | name=Hank |

请求头结束，是以空行结束。

请求头的每一个行，表示一个头信息。也可以通过CRLF进行换行。

服务器端程序，可以通过请求对象，获取头信息：演示：





map[

Upgrade-Insecure-Requests:[1]

User-Agent:[Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36]

Accept:[text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8]

Accept-Encoding:[gzip, deflate, br]

Accept-Language:[zh-CN,zh;q=0.9,en;q=0.8]

Connection:[keep-alive]

Cache-Control:[max-age=0]

]

## 请求主体，请求主体数据，request body, request content

请求时携带的数据。

只有：POST，PUT请求才有请求主体。当form以post方式提交请求时，post内的数据在请求主体中，向服务器传输。

而：GET（delete）是没有请求主体的。

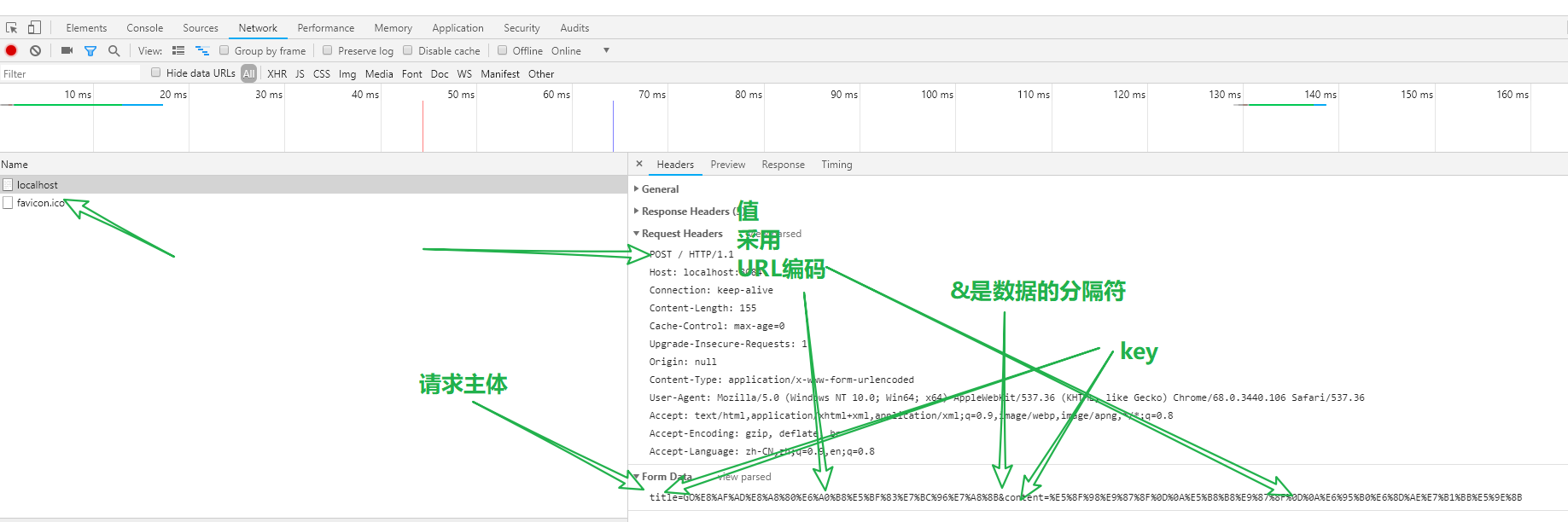
典型的请求主体，就是URL编码的字符串内容。

key=value&key=value

如果value中存在多字节字符（非ascii字符），使用URL实体编码传输

%E8%AF%AD, 就是 语 的URL实体编

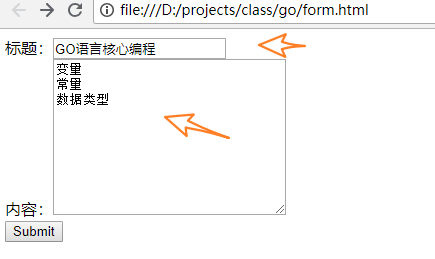
因为HTTP中不允许传输多字节字符，需要将任何的多字节字符，进行编码。



请求头以空行结束，空行后边的全部数据，都被视为请求主体。

传递到服务器的请求主体，可以使用请求对象获取。

演示：







## 向服务器传递数据（参数）

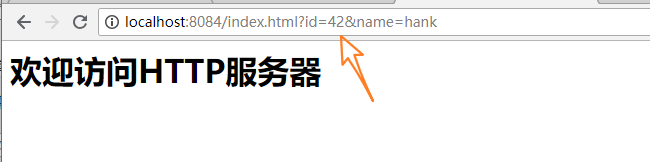
通常是两种方式，向服务器传递数据：

### 查询字符串

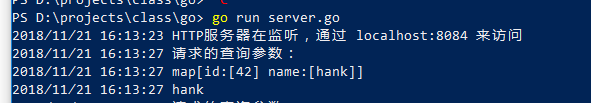
query string 查询字符串：在URL（URI）中，脚本？后边的部分。index.html?name=hank&gender=male

语法意义是URI的一部分，功能意义是向服务器传递数据。

此方式传递的数据，可以通过 请求对象的 From属性来获取（同时包含表单数据和查询字符串数据）：







### 请求主体，formdata

form data ,表单数据：在请求主体中，传递post请求的数据。

参考请求主体：3.3章。

### 选择原则

查询字符串?querystring是出现在URI中，URI（URL）的一部分，因此主要作用就是通过该参数标识具体的资源。

例如：

get /article.html?id=3 标识获取ID为3 的内容文章信息。

get /search.html?title=Go 标识获取标题title中，包含Go 的文章。

主要对应的功能：查询，搜索这类，获取资源的操作。

请求主体，post数据，是请求主体数据的一部分，因此主要作用是浏览器端希望服务器端处理的（存储，更新）数据。逻辑（语义）含义，就是单纯的数据传输，没有在查询字符串中的标识作用。

主要对应的功能：添加，更新这类，上传数据的情况。

一次请求既可以存在查询字符串数据也可以存在请求主体数据。

主要体现在更新（编辑）上。

更新，更新服务器上的已有的某个资源。需要标识出来更新的资源是哪一个？

post /article.html?id=5

form：

title=xxxx&content=xxx

GET传参与POST传参的区别？

语义不同，查询和上传。

请求主体数据，支持二进制数据传输。而查询参数只能传输字节数据。当表单中存在上传文件时，只能用post方式。

# 响应协议格式

与请求类似，分为三个部分。

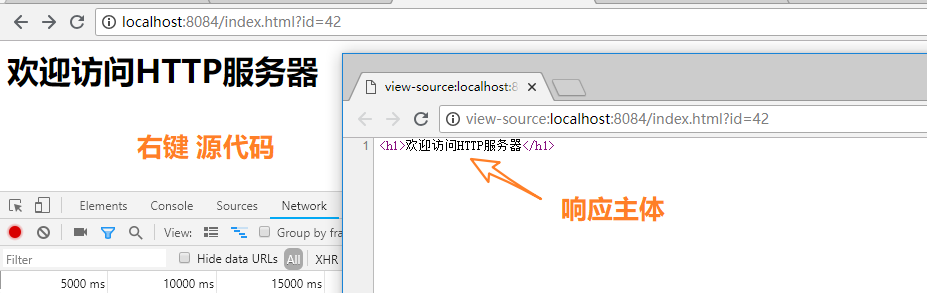
响应行（响应起始行）

响应头

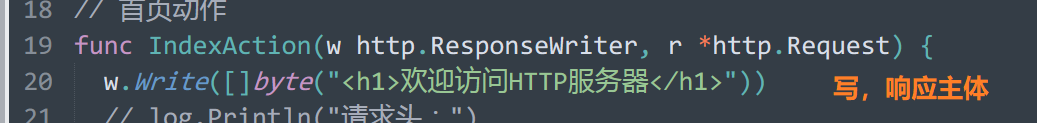
响应主体

是服务器向浏览器发送的数据。





程序中：



## 响应行，response start line

由三个（两个）部分构成：

HTTP/1.1 200 OK

HTTP/1.1 协议版本，通常与请求版本一致，都是1.1

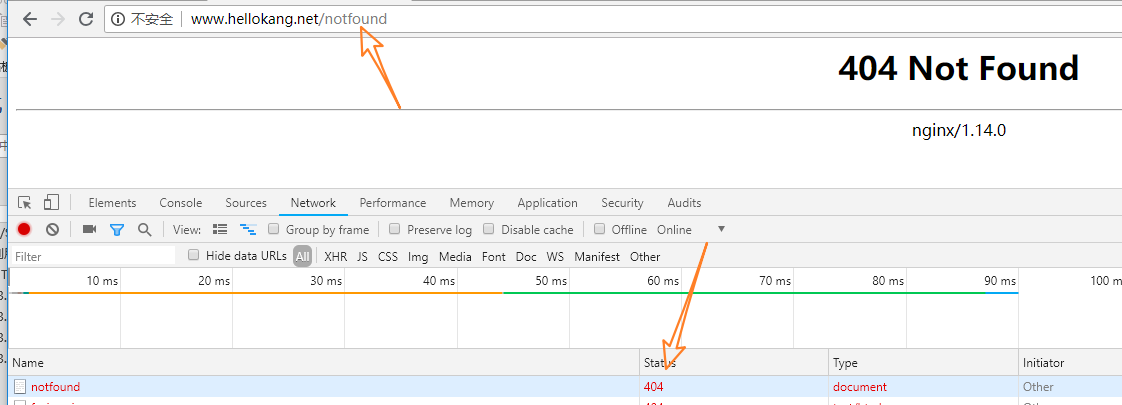
200 响应状态码，主要说明响应的总体结果。200 成功

OK 响应状态信息 与状态码一一对应。

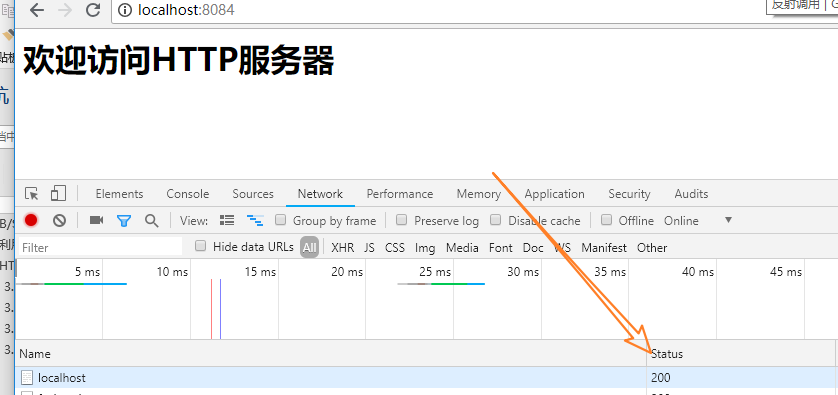
响应也是使用 CRLF 进行换行。\r\n

常用响应状态码如下：

404 Not Found 请求地址错误（URI不能标识资源）



200 OK 请求成功



301 Moved Permanently，永久重定向，告知浏览器请求的资源被移动到了其他位置。通常用于实现重定向操作。

302 Found 临时重定向。告知浏览器，请求的资源临时被放在了其他位置。也是用于实现重定向操作。

304 Not Modified 资源未修改。浏览器使用浏览器缓存获取资源。适用于静态不容易被改动的资源。

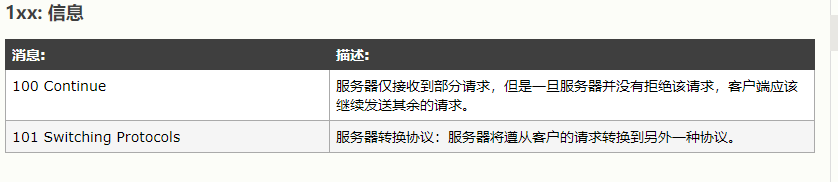
401 Unauthorized 未认证，指的是浏览器没有提供正确的用户名密码去访问服务器。（不是在网站上注册的用户名密码，指的是访问服务器的用户名密码）

403 Forbidden，被阻挡。浏览器的请求被服务器阻止了。通常是权限问题。

500 Internal Server Error 服务器内部错误。通常由于服务器的代码在执行时，出现了问题。

502 Bad Gateway 错误网关。指的是服务器没有办法识别请求。

全部的状态信息：







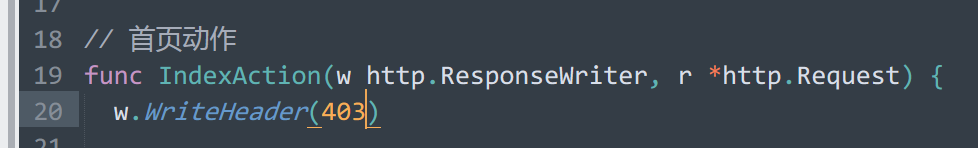


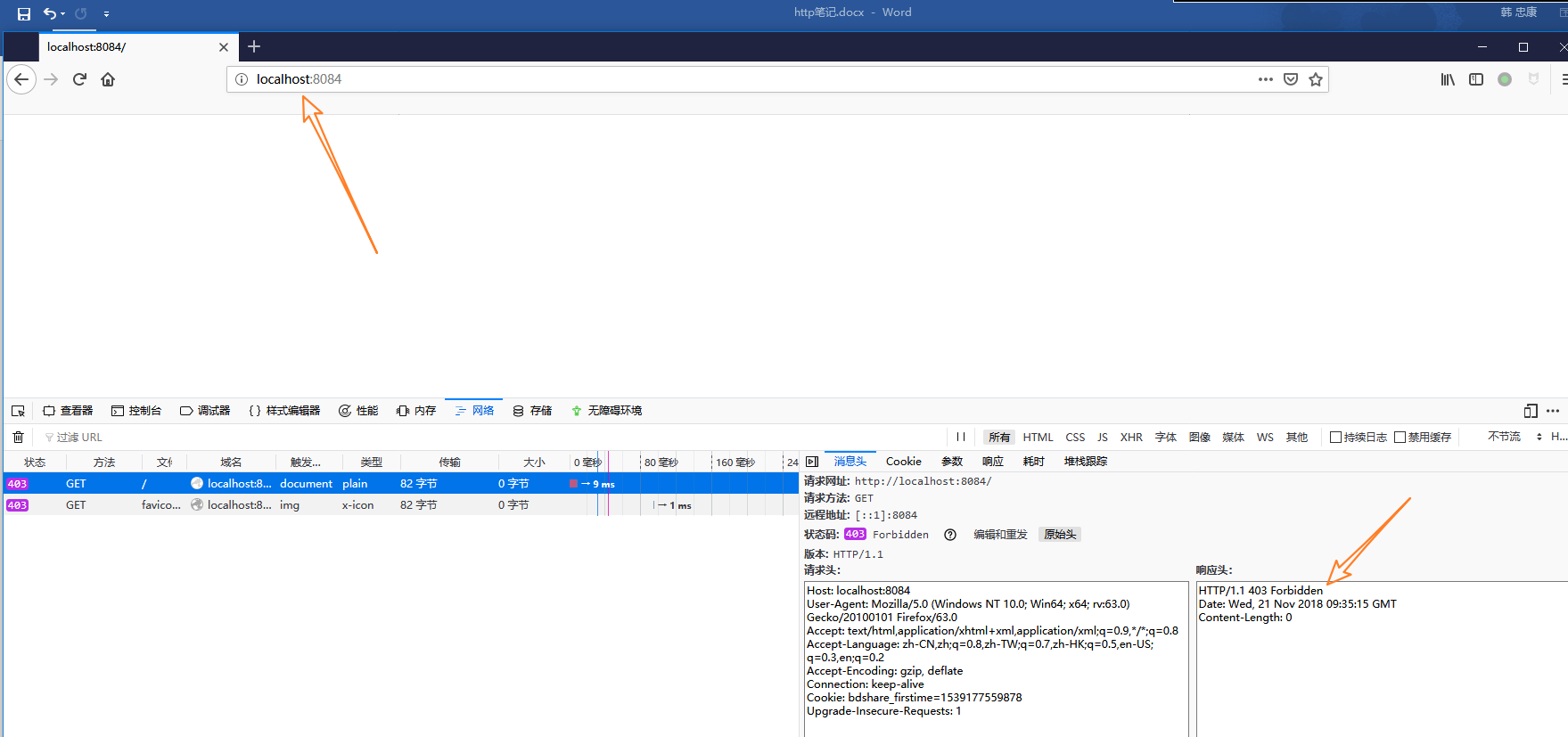


程序可以操作响应行：

使用函数：

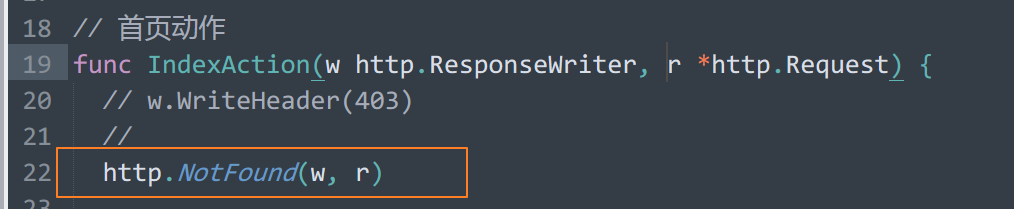
` ResponseWriter`.Writeheader()用于发送请求行中的状态码。





默认请情况下，服务器会根据当前的状态自动确定状态码。例如常规情况下就是200OK。

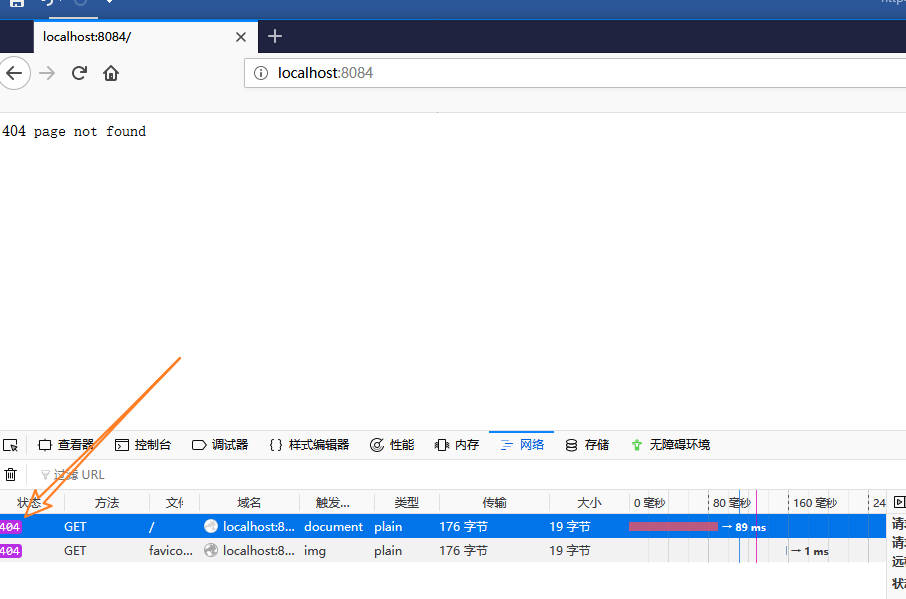
一个常用的404的返回可以使用：



就是：

w.WriteHeader(404)

w.Write([]byte("404 page not found!"))



## 响应头，响应标头，响应头信息，response header

服务器告知浏览器的一些信息。也是key:value型结构。 每行一个key，行结束CRLF。



典型的响应头：

|  |  |  |
| --- | --- | --- |
| 头key | 意义 | 典型值 |
| Content-Type | 响应主体类型，比如是HTML，还是图片。如果需要响应其他类型，例如图片，则需要控制该响应头告知浏览器。 | text/html; charset=utf-8 |
| Content-Length | 响应主体长度。字节单位 | 34 |
| Date | 响应时间。使用的GMT（格林威治平时）时间表示。 | Thu, 22 Nov 2018 01:16:15 GMT |

通过响应写对象，完成响应头的操作：

演示：

### 响应JSON数据



若没有设置响应头 Contnt-Type：



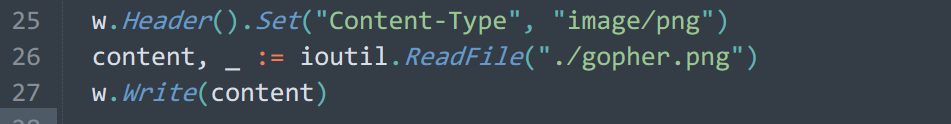
可见，浏览器是依据 响应头中的Content-Type来确定主体的内容类型的。

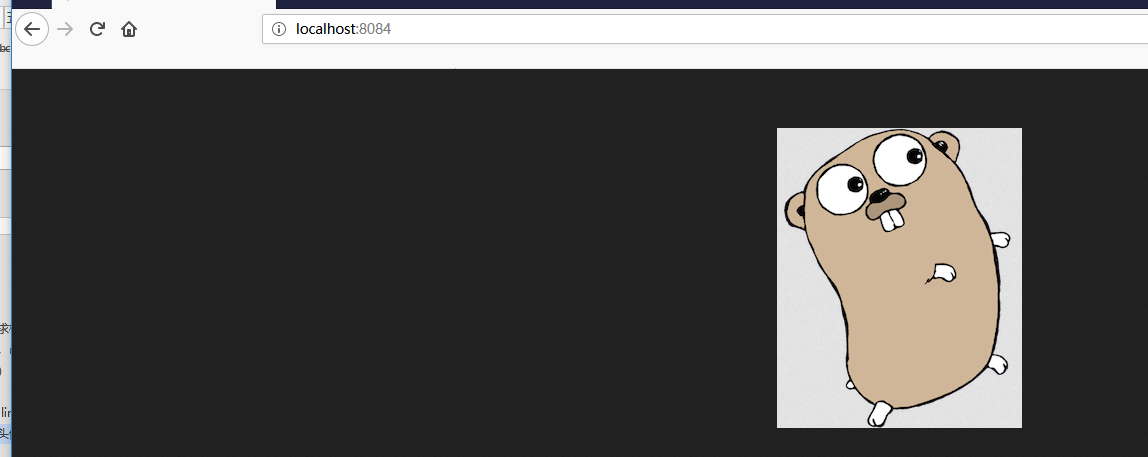
### 响应图片

（如果图像是由Go语言生成的，例如验证码）

演示，读取一个图片文件内容，输出到浏览器端。

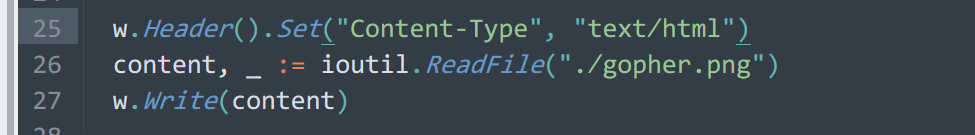
（注意，仅仅是个演示，Go的响应写操作时，会自动识别响应主体的内容类型，设置Content-Type。本例中，即使没有设置Content-Type，也会被识别为图片。建议，如果明确响应类型，告知浏览器。）

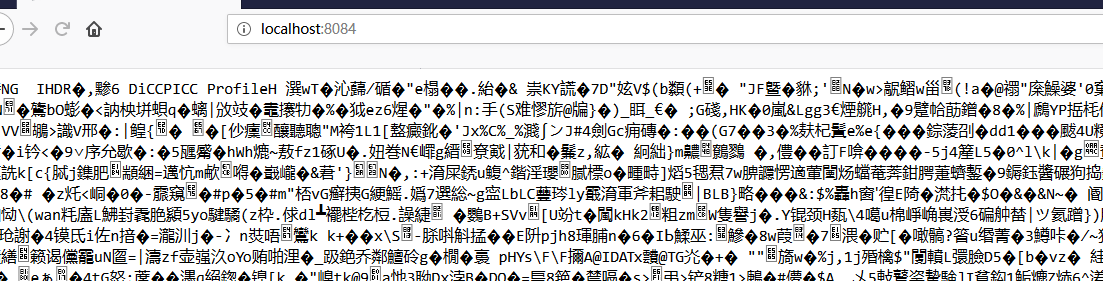




若还是使用text/html 这个类型，则浏览器将图像内容解析为html代码（字符串），会出现乱码：

（注意，仅仅是个演示，Go的响应写操作时，会自动识别响应主体的内容类型，设置Content-Type。本例中，即使没有设置Content-Type，也会被识别为图片。）





### 重定向，告知浏览器，请求新的URL

语法上，使用http.Redirect()函数实现的：

func Redirect(w ResponseWriter, r \*Request, url string, code int)

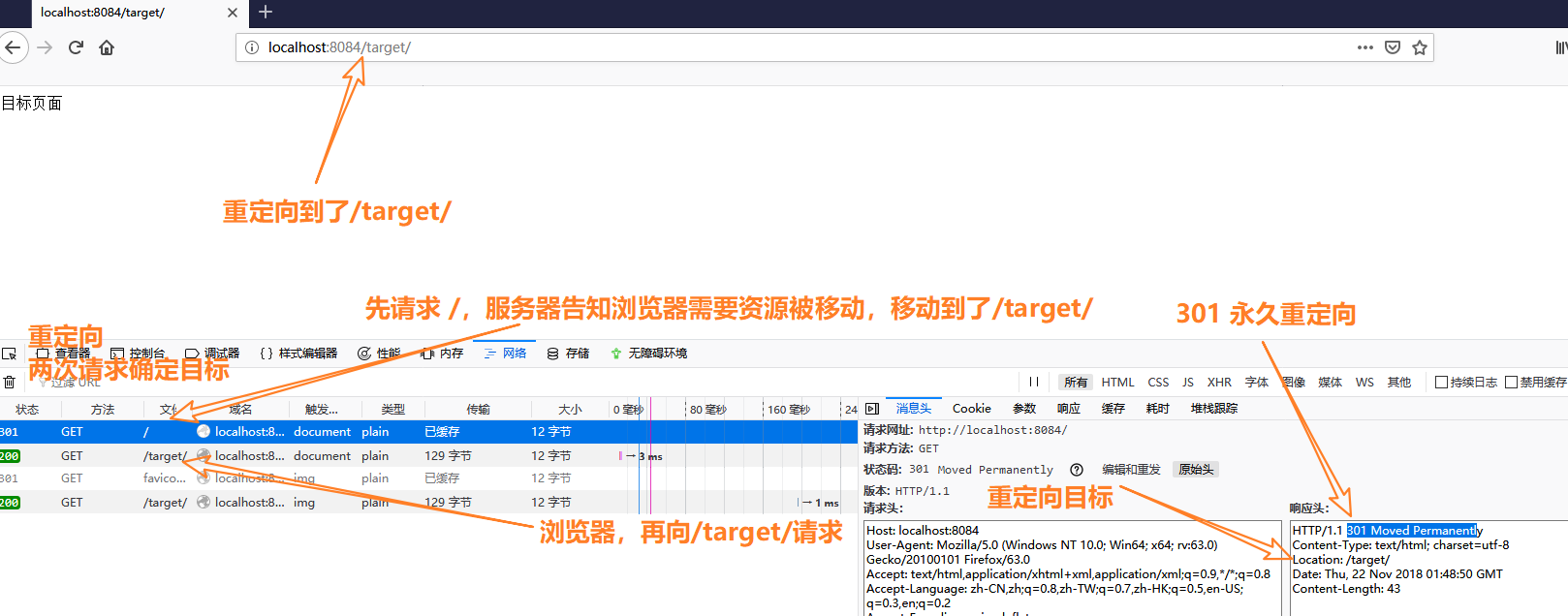
url，目标URL，code，响应状态码，通常是301或302

演示：

通过响应头 Location 告知浏览器重定向的目标的。

通过响应状态码，来告知浏览器，请求的资源被移动了。

浏览器通过以上两个信息，对新地址发出请求。



响应行，使用空行结束！

## 响应主体

可以是HTML，最常见的，CSS，JS，图像内容，都是常见的响应主体。

HTML,CSS, 都是特定个的字符串。

在响应时，数据形式的字符串：

JSON, XML

核心操作，就是在处理响应主体。



在做web项目时，通常分两类：

1数据接口类。响应内容为JSON， XML 纯粹的数据编码。前后端程序中（独立前端，手机app），Go服务器程序，仅仅需要提供数据接口。

2展示类。响应HTML就可以。通常会使用模板引擎，完成HTML处理展示。（做了波可以项目类似）

## 响应头，先于响应主体发送

响应头

空行

响应主体

语法上，调用了w.Write()函数，意味着开始处理响应主体，此时会自动结束响应头的发送。

在w.Write()后，调用的响应头设置函数，是没有意义的。



# 响应HTML主体，模板引擎 template engine

做浏览器项目，浏览器需要的HTML代码，服务器需要在响应主体中，使用HTML。HTML代码HTML代码

但是，HTML是与数据混编的语法：

纯粹的HTML，是静态页面。内部没有办法处理动态的数据。

需要使用模板引擎技术，实现将HTML与数据的混编到一起。

在 go ：html/template 包处理HTML的模板引擎

思路：

* 在HTML代码中，使用模板引擎语法，解析数据。
* 将数据分配到模板引擎中，模板被解析后，数据被展示，形成HTML与数据混编的HTML代码。
* 发送给浏览器即可。

模板引擎需要了解的语法：

如何解析模板。

如何分配数据。

模板中，如何展示变量，如何计算，如何调用函数，如何控制流程（分支和循环遍历{{）

{{ range .}}{{end}}

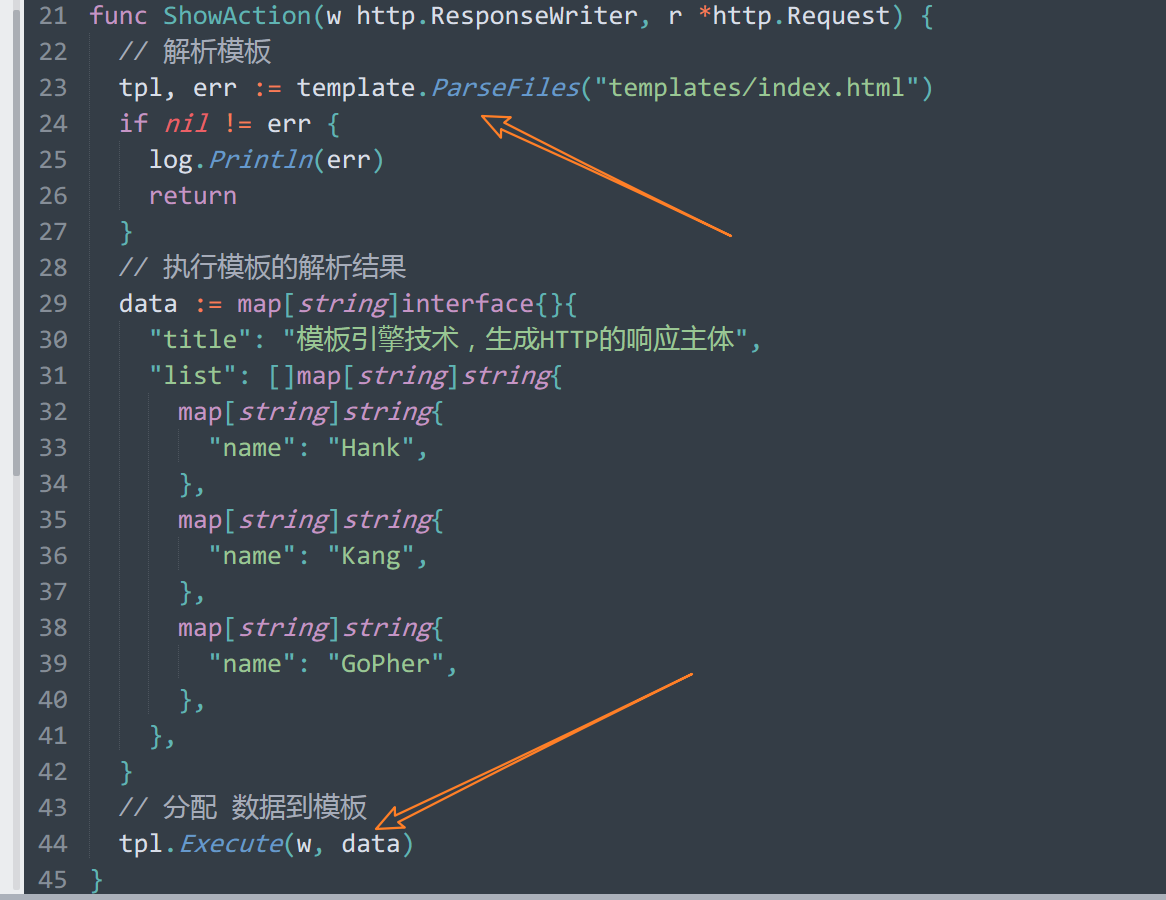
## 基本使用

定义模板文件，就是HTML文件，内部使用模板语法进行数据处理：

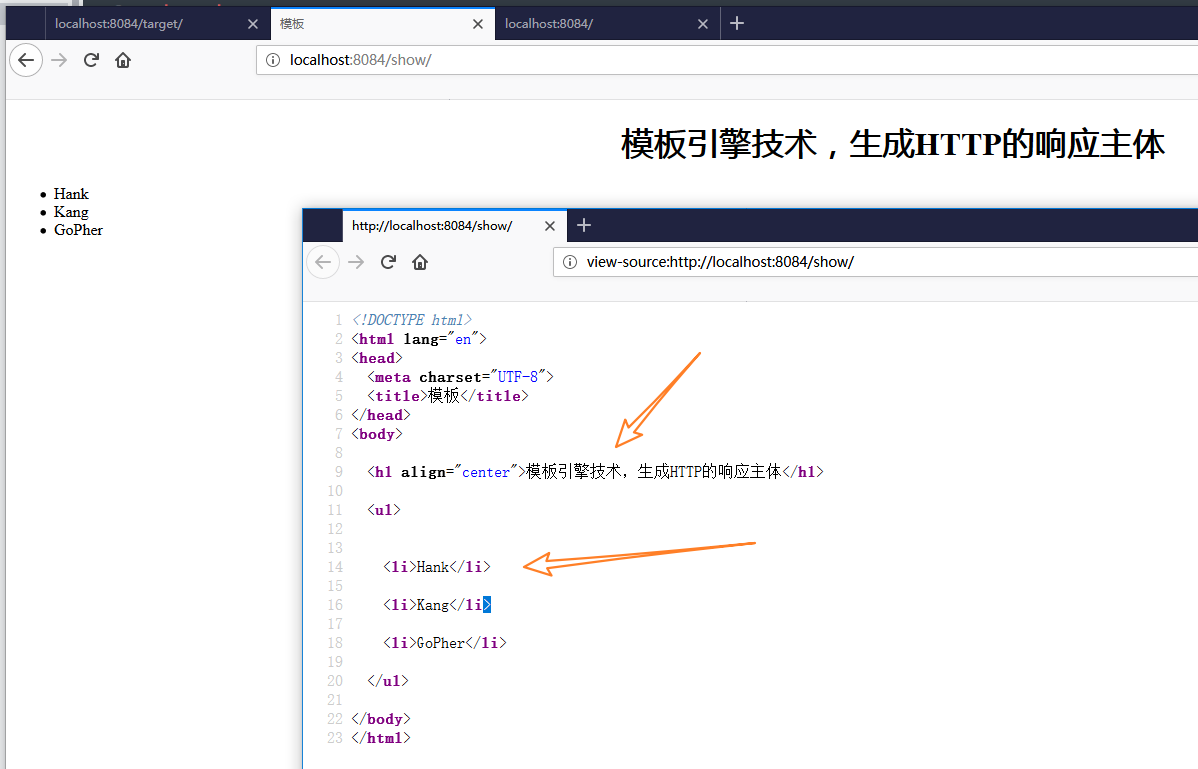
演示：



Go程序中，解析模板，执行模板（将模板形成的响应，发送到浏览器端）



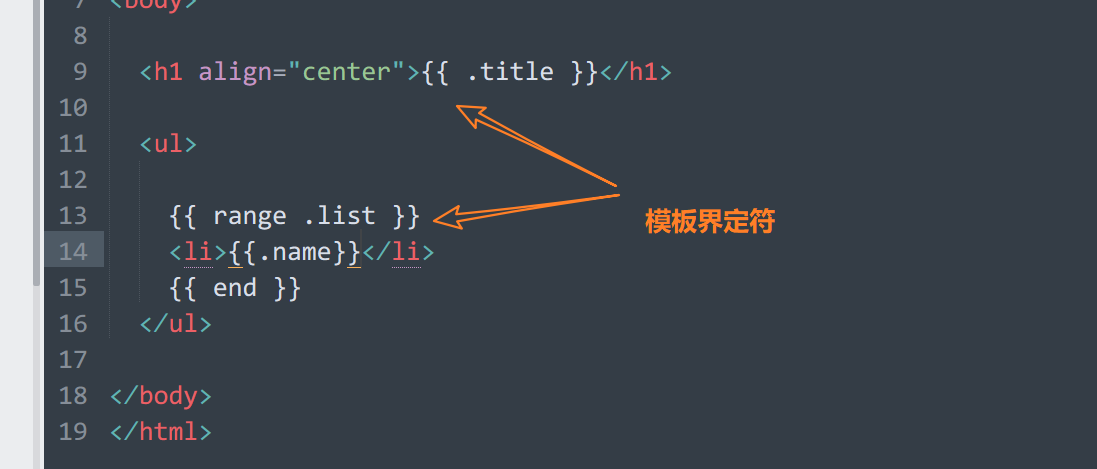
结果：



## 模板语法

在HTML，可以怎么写与模板引擎相关的操作：

### 语法界定符



默认是 {{ }} 双大括号包裹。

左边定界符： {{

右边定界符： }}

如果不使用默认的，可以进行设置：（不建议）

func (t \*Template) Delims(left, right string) \*Template

在得到模板对象后，解析文件前，进行执行。

演示：

template.New(“name”).Delims(“<!--{{“, “}}-->”).ParseFiles(模板文件即可)

界定符之内的语法，被视为需要被html/template 解析的语法，之外的部分被视为HTML代码。

### 变量（数据）输出

#### 分配

首先使用

Execute()

第二个参数，将数据分配到模板中。任何类型的数据都可以分配到模板中。

所分配的数据，在模板中，使用 点，进行引用：

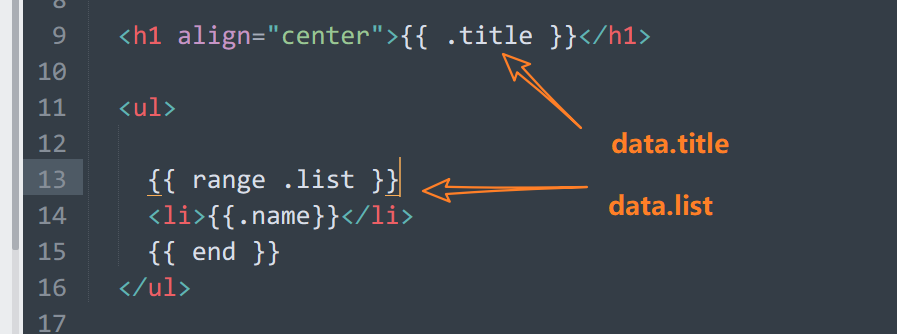
tpl.Execute(w, “Hello World”)， . == “Hello World”

tmp.Execute(w, []string{“hello”, “wordl”}), . == []string{“hello”, “wordl”}

#### 输出，点语法

分配数据的具体类型格式，取决于你在模板中使用的数据的格式：

例如：



分配的数据最常用的就是 ：map[string]interface{}

模板中，使用 . 引用分配的数据。

点 . 会随着上下文（语法环境）变化而变化。如果在最外层出现，表示的就是分配的整体数据。如果出现在循环中，表示当前长在遍历得到的具体数据。



在 界定符内，直接使用 . 表示数据的输出。

#### HTML编码

而对于数据来说，模板，会默认开启 HTML 实体编码，防止XSS（跨站脚本攻击）安全危害。不希望数据解析后，危害我们的HTML，JS等浏览器资源。

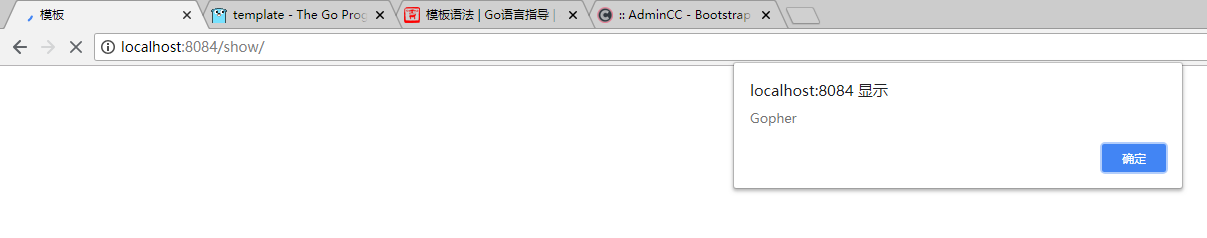


默认的结果，被转换编码了



如果需要原样处理HTML内容，将数据使用 template.HTML格式定义：





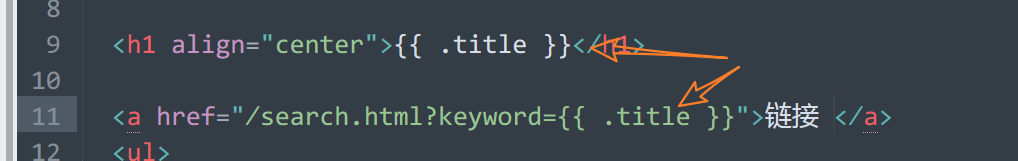


#### URL编码

在模板中，除了使用HTML实体编码外，URL编码

%E8%AF%AD,

在形成连接地址 <a href=”值”><form action=”值”> 时，需要将数据进行URL实体编码





### 调用函数

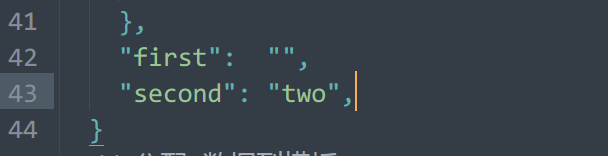
{{FuncName1 "参数值1" "参数值2"}}

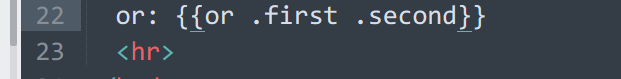
可被直接调用的函数不多，需要是系统级别的函数：

* and，返回第一个 参数或者最后一个参数。即 and x y 等价于 if x then y else x，所有参数都会执行
* or，返回第一个非 empty 参数或者最后一个参数。即 or x y 等价于 if x then x else y，所有参数都会执行
* not，返回它的单个参数的布尔值的否定
* len，返回它的参数的整数类型长度
* index，返回第一个参数以剩下的参数为索引/键指向的值，index x 1 2 3 返回 x[1][2][3] 的值；每个被索引的主体必须是数组、切片或者字典。
* print，即 fmt.Sprint
* printf，即 fmt.Sprintf
* println，即 fmt.Sprintln
* html，返回参数的 HTML 实体编码
* urlquery，返回参数的 URL 编码
* js，返回参数的 JavaScript 编码

较常用的有：

or, 输出第一个不为空的数据。







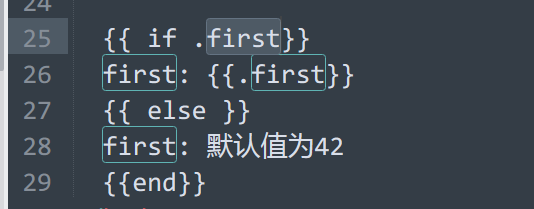
### 流程控制

条件：

{{if 条件}} T1 {{ else if 条件}} T2 {{else}} T3 {{end}}

判断条件的结果是否为空值。

空值: false、0、任意nil指针或者nil接口，任意长度为0的数组、切片



遍历：

{{range 遍历的数据}} T1 {{else}} T0 {{end}}

或

{{range $index, $element := 遍历数据}} T1 {{end}}

range支持else分支，表示所遍历的数据为空时，执行的代码。

支持同时遍历下标和值，在模板中使用变量，要求使用$开头。

if data == “” || data ==0 || len(data) == 0 {

执行 range的else分支

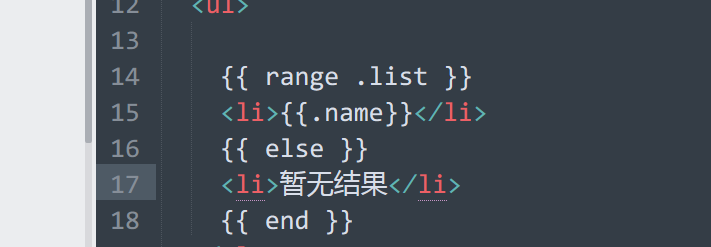
} else{

执行遍历

for v := range data {

}

}



### 子模板

在管理模板时，模板会出现公用的区域

例如，顶部，底部。

#### 定义子模板

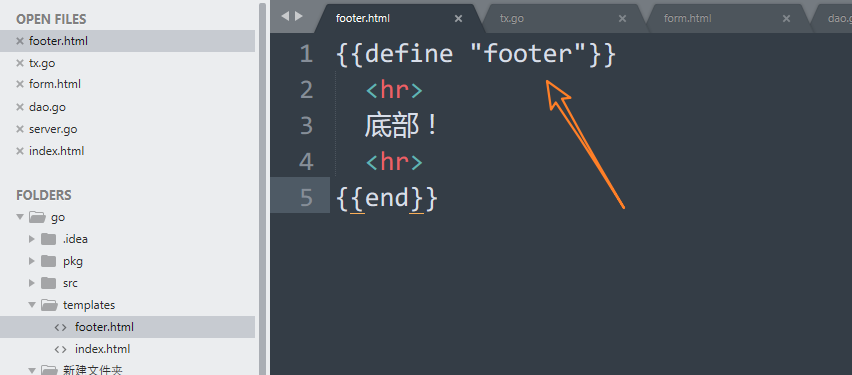
将独立的部分（重用的部分）提取到独立的文件中进行定义：

在定义子模板时，为了在上级模板中识别子模板，需要为子模板安起名字。

{{define “name”}}

模板

{{end}}



#### 整体模板加载

在整体（上级模板）使用模板中的 template 函数，来加载子模板：

{{template “name” 数据}}

如果需要将数据出传递到子模板中，需要使用数据参数。

name，在定义子模板时，指定的名字。

index.html



#### 解析全部模板

在Go程序中，需要解析全部需要使用子模板：



### 定义模板变量

由于 . 会受 上下文的变化而变化，有时需要将数据额外记录一份。

使用模板变量即可：

