

(1)

liwenzhou.com/posts/Go/go_template/

模板语法

{{.}}

模板语法都包含在 `{{` 和 `}}` 中间, 其中 `{{.}}` 中的点表示当前对象。

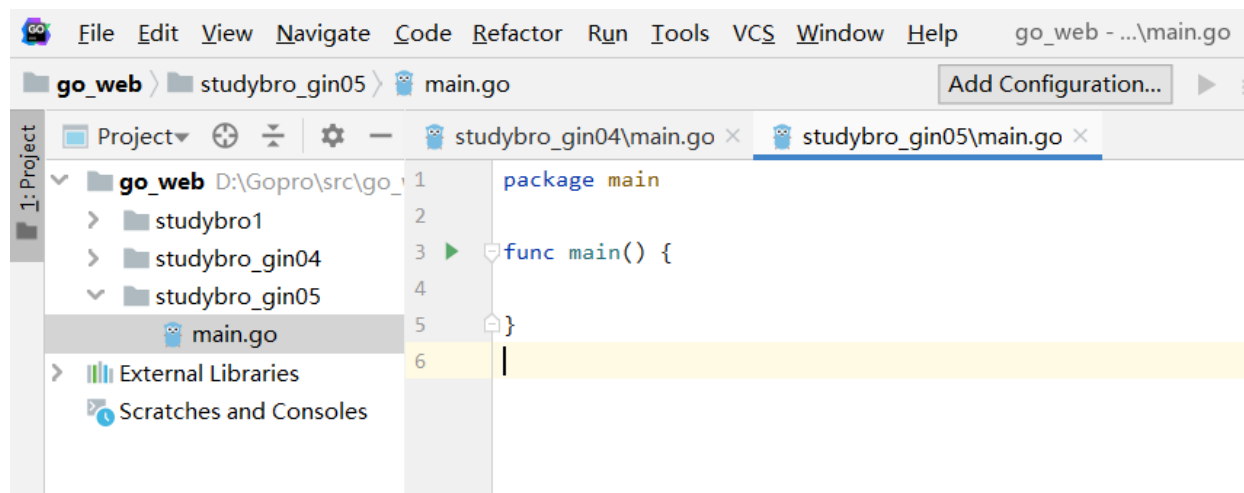
当我们传入一个结构体对象时, 我们可以根据 `.` 来访问结构体的对应字段。例如:

```
1 // main.go
2
3 type UserInfo struct {
4     Name string
5     Gender string
6     Age int
7 }
8
9 func sayHello(w http.ResponseWriter, r *http.Request) {
10     // 解析指定文件生成模板对象
11     tpl, err := template.ParseFiles("./hello.tpl")
12     if err != nil {
13         fmt.Println("create template failed, err:", err)
14         return
15     }
16     // 利用给定数据渲染模板, 并将结果写入w
17     user := UserInfo{
18         Name: "小王子",
19         Gender: "男",
20         Age: 18,
21     }
22     tpl.Execute(w, user)
23 }
```

模板文件 `hello.tpl` 内容如下:

```
1 <!DOCTYPE html>
2 <html lang="zh-CN">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Hello</title>
8 </head>
9 <body>
10     <p>Hello {{.Name}}</p>
11     <p>性别: {{.Gender}}</p>
12     <p>年龄: {{.Age}}</p>
```

(2)



目录我们重新整理了一下，这个我们打开上级的目录，下面，我们再新建一个 studybro_gin05目录，并且在下面新建一个main.go

(3)

hello go的模板文件

```
1 <!DOCTYPE html>
2 <html lang="zh-CN">
3 <head>
4   {{/* <meta charset="UTF-8">*/}}
5   {{/* <meta name="viewport" content="width=device-width, initial-scale=1.0">*/}}
6   {{/* <meta http-equiv="X-UA-Compatible" content="ie=edge">*/}}
7   <title>Hello</title>
8 </head>
9 <body>
10  <p>Hello {{.}}</p>
11 </body>
12 </html>
```

main.go

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 func sayHello(w http.ResponseWriter, r *http.Request) {
9     //重点：遇事不决，先写注释
10    //2.定义模板
11
12    //3.解析模板
13    t,err :=template.ParseFiles("./hello") //请勿刻舟求剑
14    if err!=nil{
15        fmt.Println("Parse template failed,err:%v",err)
16        return
17    }
```

```

18 //4.渲染模板
19 name:="小西瓜"
20 err=t.Execute(w,name)
21 if err!=nil{
22     fmt.Println("render template failed,err:%v",err)
23     return
24 }
25 }
26
27 func main() {
28     //1.创建服务器
29     http.HandleFunc("/",sayHello)
30     err:=http.ListenAndServe(":9000",nil)
31     if err!=nil{
32         fmt.Println("Http server start failed,err:%v",err)
33         return
34     }
35 }

```

现在，我们要写的是一个结构体，

(4)



```

1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 type UserInfo struct {
9     Name    string
10    gender  string
11    Age     int
12 }
13
14 func sayHello(w http.ResponseWriter, r *http.Request) {
15     //重点: 遇事不决, 先写注释
16     //2. 定义模板
17

```

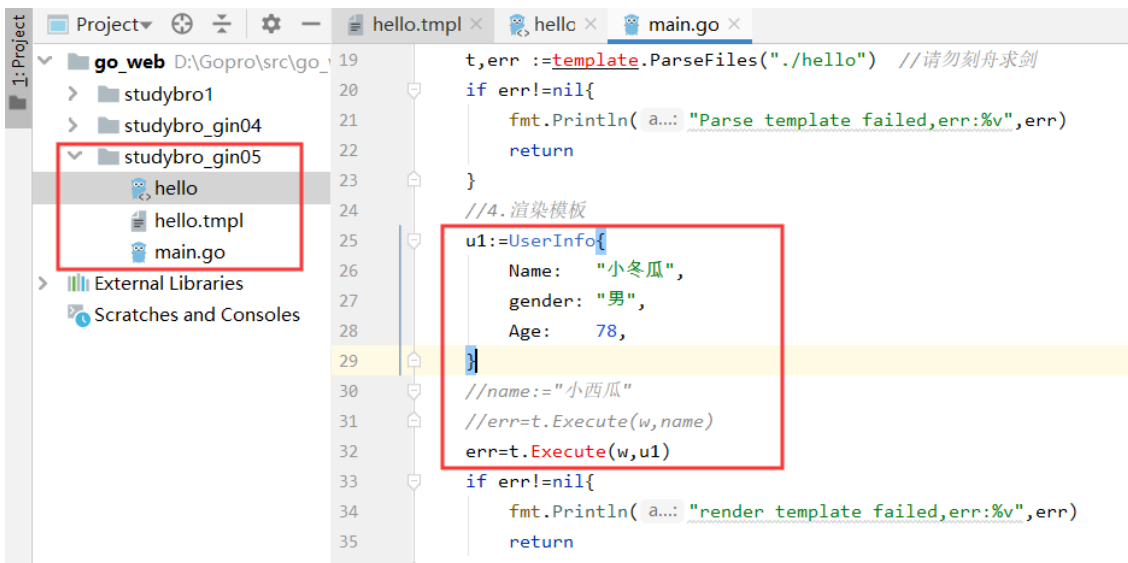
```

1 type UserInfo struct {
2     Name string
3     gender string
4     Age int
5 }

```

先在文件中写一个结构体（注意，这里边的变量是由大小写区别的），再在函数里边，我们可以定义一个u1。

(5)



此时，我们我们将结构体传递到这个执行文件，这个时候，我们看一下前端的文件传递的结果是什么呢？

```

9 <body>
10 <Hello {{.}}></p>
11 </body>
12 </html>

```

此时这个.表示的就是u1

```

1 {{.Name .gender .Age .gender}}

```

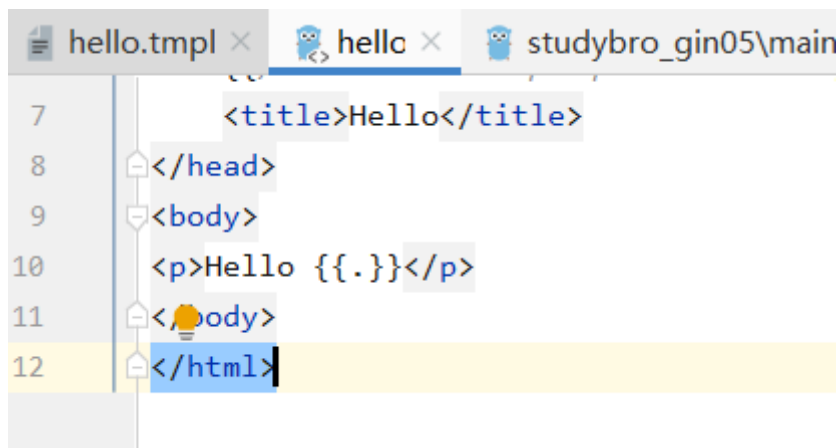
我们可以通过上边的方式进行访问。

注意，上边的两个花括号之间不要加上空格才行。

(6)



```
16 //重点: 遇事不决, 先写注释
17 //2. 定义模板
18
19 //3. 解析模板
20 t,err :=template.ParseFiles( filenames...: "../hello") //请勿刻舟求剑
21 if err!=nil{
22     fmt.Println( a...: "Parse template failed,err:%v",err)
23     return
24 }
25 //4. 渲染模板
26 ...
```



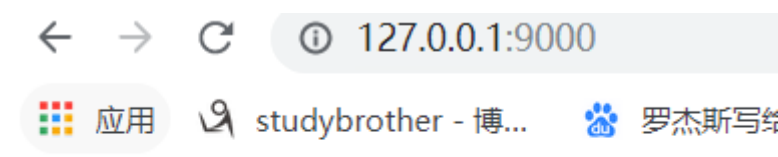
```
7 <title>Hello</title>
8 </head>
9 <body>
10 <p>Hello {{.}}</p>
11 </body>
12 </html>
```

```
D:\Gopro\src\go_web\studybro_gin05>go build
# go_web/studybro_gin05
.\main.go:19:10: undefined: template
```

```
D:\Gopro\src\go_web\studybro_gin05>go build
```

```
D:\Gopro\src\go_web\studybro_gin05>studybro_gin05.exe
```

编译上边的代码:



Hello {小冬瓜 男 78}

此时, 我们应该做的事情是访问这个地址, 得到上边的内容

(7)

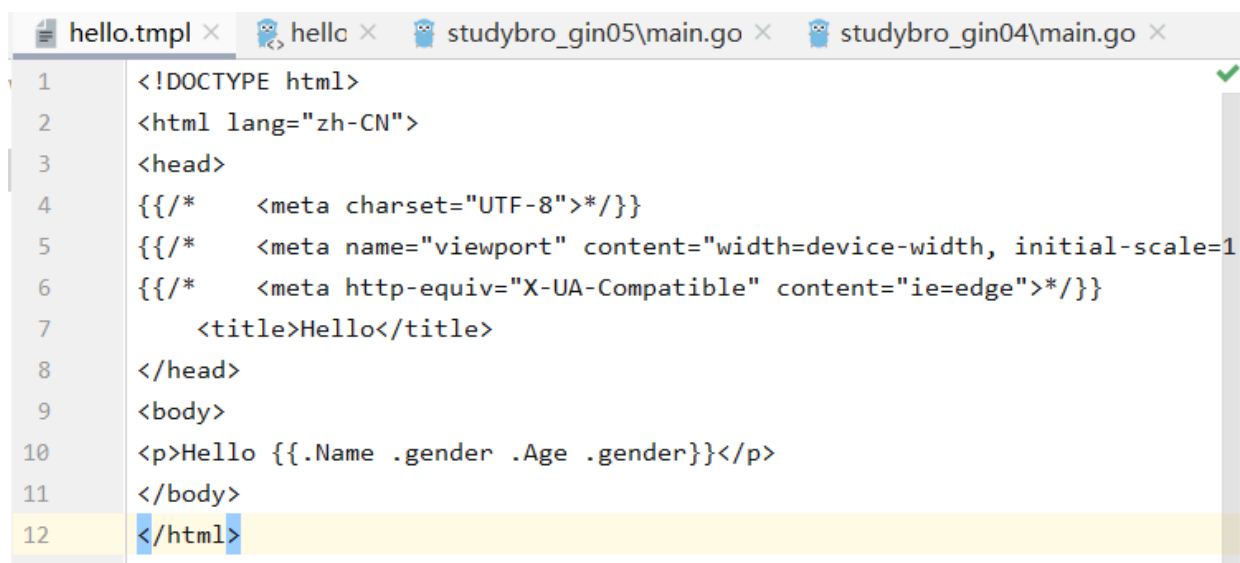
注意，我们在上边犯了一个错误，出现了一个报错，我们需要导入一个包才行。

```
1 import (  
2     "fmt"  
3     "net/http"  
4     "html/template"  
5 )
```

(8)



```
16 //重点：遇事不决，先写注释  
17 //2. 定义模板  
18  
19 //3. 解析模板  
20 t,err :=template.ParseFiles( filenames...: "./hello.tpl") //请勿刻舟求剑  
21 if err!=nil{  
22     fmt.Println( a...: "Parse template failed,err:%v",err)  
23     return
```



```
1 <!DOCTYPE html>  
2 <html lang="zh-CN">  
3 <head>  
4     {{/* <meta charset="UTF-8">*/}}  
5     {{/* <meta name="viewport" content="width=device-width, initial-scale=1.0">*/}}  
6     {{/* <meta http-equiv="X-UA-Compatible" content="ie=edge">*/}}  
7     <title>Hello</title>  
8 </head>  
9 <body>  
10 <p>Hello {{.Name .gender .Age .gender}}</p>  
11 </body>  
12 </html>
```

上边这种方式是运行不了的，需要运行的话，只能写一个参数，并且首字母需要大写

```
hello.tmpl x hello x studybro_gin05\main.go x studyl
4      {{/*    <meta charset="UTF-8">*/}}
5      {{/*    <meta name="viewport" content="width=device-1
6      {{/*    <meta http-equiv="X-UA-Compatible" content="
7          <title>Hello</title>
8      </head>
9      <body>
10     <p>Hello {{.Age}}</p>
11     </body>
12     </html>
```

我们看到，服务端会报下面的错误。

```
Terminal: Local x +
s but cannot be invoked as function
render template failed,err:%v template: hello.tmpl:10:20: executing "hello.tmpl" at <.gender>: gender is an un
exported field of struct type main.UserInfo
render template failed,err:%v template: hello.tmpl:10:20: executing "hello.tmpl" at <.gender>: gender is an un
exported field of struct type main.UserInfo
render template failed,err:%v template: hello.tmpl:10:11: executing "hello.tmpl" at <.gender>: gender is an un
exported field of struct type main.UserInfo
render template failed,err:%v template: hello.tmpl:10:11: executing "hello.tmpl" at <.gender>: gender is an un
exported field of struct type main.UserInfo
render template failed,err:%v template: hello.tmpl:10:11: executing "hello.tmpl" at <.Age>: Age has arguments
but cannot be invoked as function
render template failed,err:%v template: hello.tmpl:10:11: executing "hello.tmpl" at <.Age>: Age has arguments
but cannot be invoked as function

D:\Gopro\src\go_web\studybro_gin05>
```

(9)

重新再修改一下

```
hello.tmpl x hello x studybro_gin05\main.go x st
4 {{/* <meta charset="UTF-8">*/}}
5 {{/* <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1.0">*/}}
6 {{/* <meta http-equiv="X-UA-Compatible" content="ie=edge">*/}}
7 <title>Hello</title>
8 </head>
9 <body>
10 <p>Hello {{.Name}}</p>
11 <p>年龄 : {{.Age}}</p>
12 <p>性别 : {{.gender}}</p>
13 </body>
14 </html>
```

我们看到下图得到的内容。



Hello 小冬瓜

年龄: 78

性别:

我们修改tmpl的时候，不需要重启服务器，只需要修改前端就可以了

我们看到这里依然是不能看到数据的。

因此，老师总结了一下，go语言将首字母大小写是否暴露的一个特点。

模板引擎需要首字母大写

(10)

修改服务器端源代码，我们重新编译一下即可

定义修改：


```
hello.tmpl x hello x studybro_gin05\main.go x s
7   )
8
9   type UserInfo struct {
10       Name    string
11       Gender  string
12       Age     int
13   }
14
```

下面是我们得到的内容。

赋值修改：

```
hello.tmpl x hello x studybro_gin05\main.go x
22   fmt.Println(a...: "Parse template fail")
23   return
24 }
25 //4. 渲染模板
26 u1:=UserInfo{
27     Name:    "小冬瓜",
28     Gender:  "男",
29     Age:     78,
30 }
```

模板中修改：

```
hello.tmpl x hello x studybro_gin05\main.go x studybro
3   <head>
4   {{/* <meta charset="UTF-8">*/}}
5   {{/* <meta name="viewport" content="width=device-wic
6   {{/* <meta http-equiv="X-UA-Compatible" content="ie=
7       <title>Hello</title>
8   </head>
9   <body>
10   <p>Hello {{.Name}}</p>
11   <p>年龄 : {{.Age}}</p>
12   <p>性别 : {{.Gender}}</p>
13   </body>
14   </html>
```

重新编译：

```
D:\Gopro\src\go_web\studybro_gin05>go build
```

```
D:\Gopro\src\go_web\studybro_gin05>studybro_gin05.exe
```

访问

← → ↻ ⓘ 127.0.0.1:9000

应用 studybrother - 博... 罗杰斯写给女儿的1...

Hello 小冬瓜

年龄: 78

性别: 男

(11)

我们可以传递结构体，当然我们也可以传递map了，接口

```
hello.tpl x hello x studybro_gin05\main.go x studybro_gin04\main.go x
25 //4. 渲染模板
26 //u1:=UserInfo{
27 //  Name:  "小冬瓜",
28 //  Gender: "男",
29 //  Age:   78,
30 //}
31
32 m1:=map[string]interface{}{
33     "Name": "小包子",
34     "Gender": "男",
35     "Age":   78,
36 }
37 //name:="小西瓜"
38 //err=t.Execute(w,name)
39 //err=t.Execute(w,u1)
40 err=t.Execute(w,m1)
41 if err!=nil{
42     fmt.Println(a...: "render template failed,err:%v",err)
43     return
44 }
```

注意，我们在写这个m1的时候，需要注释这个上边的结构体

编译, 执行

```
Terminal: Local x +  
  
D:\Gopro\src\go_web\studybro_gin05>go build  
  
D:\Gopro\src\go_web\studybro_gin05>studybro_gin05.exe  
[]
```

← → ↻ ⓘ 127.0.0.1:9000

应用 studybrother - 博... 罗杰斯写给3

Hello 小包子

年龄: 78

性别: 男

思考:

map是否需要首字母大写?

不需要, 因为这个是通过key来传递数值的

(12)

```
hello.tpl x studybro_gin05\main.go x he  
31  
32 m1:=map[string]interface{}{  
33     "name": "小包子",  
34     "Gender": "男",  
35     "Age": 78,  
36 }
```

```
hello.tpl x studybro_gin05\main.go x hello x studybro_c
1 <!DOCTYPE html>
2 <html lang="zh-CN">
3 <head>
4 {{/* <meta charset="UTF-8">*/}}
5 {{/* <meta name="viewport" content="width=device-widt
6 {{/* <meta http-equiv="X-UA-Compatible" content="ie=e
7 <title>Hello</title>
8 </head>
9 <body>
10 <p>Hello {{.name}}</p>
11 <p>年龄： {{.Age}}</p>
12 <p>性别： {{.Gender}}</p>
13 </body>
14 </html>
```

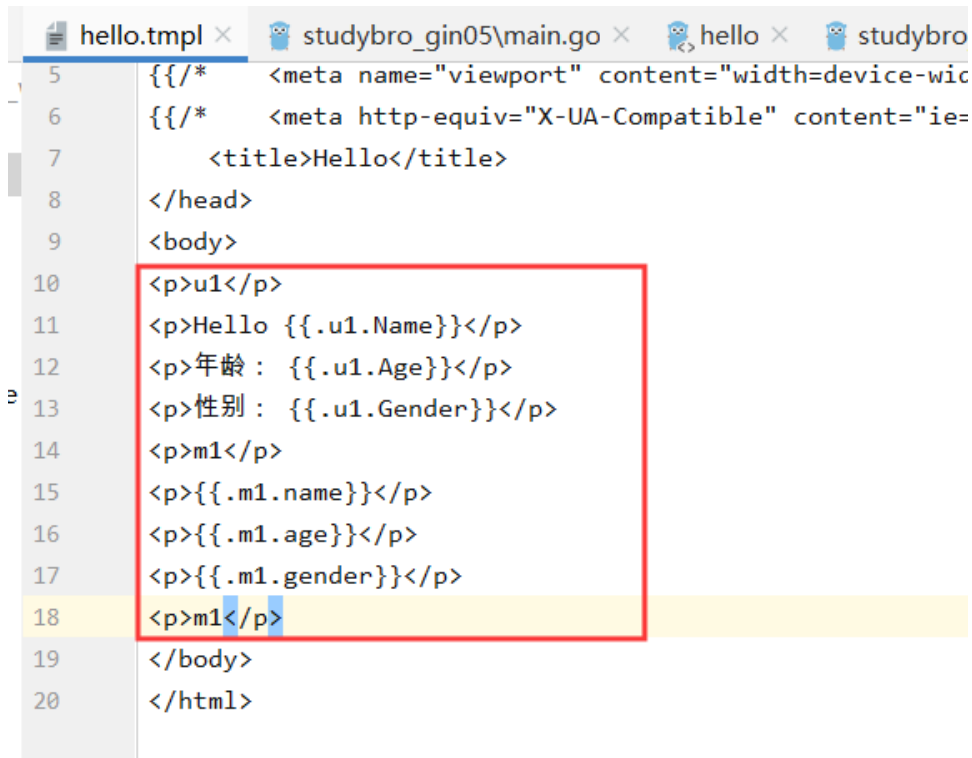
(13)

如何传递多个值？

```
hello.tpl x studybro_gin05\main.go x hello x studybro_gin04\main.
25 //4. 渲染模板
26 u1:=UserInfo{
27     Name: "小冬瓜",
28     Gender: "男",
29     Age: 78,
30 }
31
32 m1:=map[string]interface{}{
33     "name": "小包子",
34     "Gender": "男",
35     "Age": 78,
36 }
37 //name:="小西瓜"
38 //err=t.Execute(w,name)
39 //err=t.Execute(w,u1)
40 //err=t.Execute(w,m1)
41 err=t.Execute(w,map[string]interface{}{
42     "u1":u1,
43     "m1":m1,
44 })
```

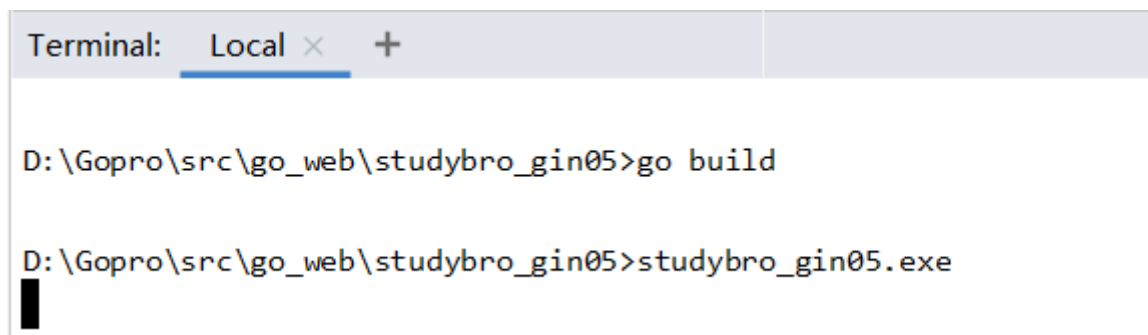
此时，我们通过map来进行渲染一个空接口，类似于字典的传递值

(14)



```
5 {{/* <meta name="viewport" content="width=device-wic
6 {{/* <meta http-equiv="X-UA-Compatible" content="ie=
7 <title>Hello</title>
8 </head>
9 <body>
10 <p>u1</p>
11 <p>Hello {{.u1.Name}}</p>
12 <p>年龄: {{.u1.Age}}</p>
13 <p>性别: {{.u1.Gender}}</p>
14 <p>m1</p>
15 <p>{{.m1.name}}</p>
16 <p>{{.m1.age}}</p>
17 <p>{{.m1.gender}}</p>
18 <p>m1</p>
19 </body>
20 </html>
```

上边是我们修改的内容。



```
Terminal: Local × +

D:\Gopro\src\go_web\studybro_gin05>go build

D:\Gopro\src\go_web\studybro_gin05>studybro_gin05.exe
```

(15)

注意，前后端的字符串一定要对应好内容才行。否则不会显示



(16)

注释:

注释

- 1 `{{/* a comment */}}`
- 2 注释, 执行时会忽略。可以多行。注释不能嵌套, 并且必须紧贴分界符终止。



(17)

pipeline

`pipeline` 是指产生数据的操作。比如 `{{.}}`、`{{.Name}}` 等。Go的模板语法中支持使用管道符号 `|` 链接多个命令，用法和unix下的管道类似：`|` 前面的命令会将运算结果(或返回值)传递给后一个命令的最后一个位置。

注意：并不是只有使用了 `|` 才是pipeline。Go的模板语法中，`pipeline`的概念是传递数据，只要能产生数据的，都是 `pipeline`。

注意，只要能够产生数据，都是pipeline。

变量

我们还可以在模板中声明变量，用来保存传入模板的数据或其他语句生成的结果。具体语法如下：

```
1 | $obj := {{.}}
```

其中 `$obj` 是变量的名字，在后续的代码中就可以使用该变量了。

(18)

```
hello.tmpl x studybro_gin05\main.go x hello x st
10 <p>u1</p>
11 <p>Hello {{.u1.Name}}</p>
12 <p>年龄： {{.u1.Age}}</p>
13 <p>性别： {{.u1.Gender}}</p>
14
15 {{/* 遇事不决，要写注释！ */}}
16
17 <p>m1</p>
18 <p>Hello {{.m1.name}}</p>
19 <p>年龄： {{.m1.Age}}</p>
20 <p>性别： {{.m1.Gender}}</p>
21 {{ $v1:=100 }}
22 {{ $age:=.m1.age }}
23
24 <p>m1</p>
25 </body>
26 </html>
```

(19)

移除空格

有时候我们在使用模板语法的时候会不可避免的引入一下空格或者换行符，这样模板最终渲染出来的内容可能就和我们的不一样，这个时候可以使用 `{{-}}` 语法去除模板内容左侧的所有空白符号，使用 `-}}` 去除模板内容右侧的所有空白符号。

例如：

```
1 | {{- .Name -}}
```

注意： `-` 要紧挨 `{{` 和 `}}`，同时与模板值之间需要使用空格分隔。

移除，左右的空格。


```
hello.templ x studybro_gin05\main.go x hello x studybro_g
11 <p>Hello {{.u1.Name}}</p>
12 <p>年龄： {{.u1.Age}}</p>
13 <p>性别： {{.u1.Gender}}</p>
14
15 {{/* 遇事不决，要写注释！ */}}
16
17 <p>m1</p>
18 <p>Hello {{- .m1.name -}}</p>
19 <p>年龄： {{.m1.Age}}</p>
20 <p>性别： {{.m1.Gender}}</p>
21 {{$v1:=100}}
22 {{$age:=.m1.age}}
23
24 <p>m1</p>
25 </body>
26 </html>
```

--这两个要紧紧的顶着花括号{}，否则会报错，中横线和变量之间需要留有一个空格，否则会产生歧义等问题。

我们在模板语言中也可以做提哦阿健判断

(20)

```
1 <hr>
2 {{if $v1}}
3 {{ $v1 }}
4 {{else}}
5 什么都没有
6 {{end}}
```

条件判断

Go模板语法中的条件判断有以下几种:

```
1 {{if pipeline}} T1 {{end}}
2
3 {{if pipeline}} T1 {{else}} T0 {{end}}
4
5 {{if pipeline}} T1 {{else if pipeline}} T0 {{end}}
```

range

Go的模板语法中使用 `range` 关键字进行遍历, 有以下两种写法, 其中 `pipeline` 的值必须是数组、切片、字典或者通道。

```
1 {{range pipeline}} T1 {{end}}
2 如果pipeline的值其长度为0, 不会有任何输出
3
4 {{range pipeline}} T1 {{else}} T0 {{end}}
5 如果pipeline的值其长度为0, 则会执行T0。
```

pipeline的值必须是数组, 切片, 字典或者通道。

(21)

```
hello.tmpl x studybro_gin05\main.go x hello x studybro_gin04\main.go
18 <p>Hello {{- .m1.name -}}</p>
19 <p>年龄: {{.m1.Age}}</p>
20 <p>性别: {{.m1.Gender}}</p>
21 {{$v1:=100}}
22 {{$age:=.m1.age}}
23 <p>m1</p>
24
25 <hr>
26 {{if $v1}}
27 {{ $v1 }}
28 {{else}}
29 什么都没有
30 {{end}}
31
32 </body>
33 </html>
```

(22)

with

```
1  {{with pipeline}} T1 {{end}}
2  如果pipeline为empty不产生输出，否则将dot设为pipeline的值并执行T1。不修改外面的d
3
4  {{with pipeline}} T1 {{else}} T0 {{end}}
5  如果pipeline为empty，不改变dot并执行T0，否则dot设为pipeline的值并执行T1。
```

预定义函数

执行模板时，函数从两个函数字典中查找：首先是模板函数字典，然后是全局函数字典。一般不在模板内定义函数，而是使用Funcs方法添加函数到模板里。

预定义的全局函数如下：

注意：这个预定义函数的内容会比较多一些。

```
1  and
2  函数返回它的第一个empty参数或者最后一个参数；
3  就是说"and x y"等价于"if x then y else x"；所有参数都会执行；
4  or
5  返回第一个非empty参数或者最后一个参数；
6  亦即"or x y"等价于"if x then x else y"；所有参数都会执行；
7  not
8  返回它的单个参数的布尔值的否定
9  len
10  返回它的参数的整数类型长度
11  index
12  执行结果为第一个参数以剩下的参数为索引/键指向的值；
13  如"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。
14
15  print
16  即fmt.Sprint
17  printf
18  即fmt.Sprintf
19  println
20  即fmt.Sprintln
21  html
22  返回与其参数的文本表示形式等效的转义HTML。
23  这个函数在html/template中不可用。
```

```
24 urlquery
25 以适合嵌入到网址查询中的形式返回其参数的文本表示的转义值。
26 这个函数在html/template中不可用。
27 js
28 返回与其参数的文本表示形式等效的转义JavaScript。
29 call
30 执行结果是调用第一个参数的返回值，该参数必须是函数类型，其余参数作为调用该函数
31 的参数；如"call .X.Y 1 2"等价于go语言里的dot.X.Y(1, 2)；
32 其中Y是函数类型的字段或者字典的值，或者其他类似情况；
33 call的第一个参数的执行结果必须是函数类型的值（和预定义函数如print明显不同）；
34 该函数类型值必须有1到2个返回值，如果有2个则后一个必须是error接口类型；
35 如果有2个返回值的方法返回的error非nil，模板执行会中断并返回给调用模板执行者该
36 错误；
```

(23)

比较函数

布尔函数会将任何类型的零值视为假，其余视为真。

下面是定义为函数的二元比较运算的集合：

1	eq	如果arg1 == arg2则返回真
2	ne	如果arg1 != arg2则返回真
3	lt	如果arg1 < arg2则返回真
4	le	如果arg1 <= arg2则返回真
5	gt	如果arg1 > arg2则返回真
6	ge	如果arg1 >= arg2则返回真

为了简化多参数相等检测，eq（只有eq）可以接受2个或更多个参数，它会将第一个参数和其余参数依次比较，返回下式的结果：

```
1 | {{eq arg1 arg2 arg3}}
```

比较函数只适用于基本类型（或重定义的基本类型，如"type Celsius float32"）。但是，整数和浮点数不能互相比较。

多读点书，残酷的社会！

```
hello.tmpl x studybro_gin05\main.go x hello x studybro_gin05\main.go x
25 <hr>
26 {{if $v1}}
27 {{ $v1 }}
28 {{else}}
29 什么都没有
30 {{end}}
31
32 <hr>
33 {{if lt .m1.age 22}}
34 {{ 好好上学 }}
35 {{else}}
36 好好工作
37 {{end}}
38
39 </body>
40 </html>
```

```
1 <hr>
2 {{if lt .m1.Age 22}}
3 好好上学
4 {{else}}
5 好好工作
6 {{end}}
```

(24)

range

Go的模板语法中使用 `range` 关键字进行遍历，有以下两种写法，其中 `pipeline` 的值必须是数组、切片、字典或者通道。

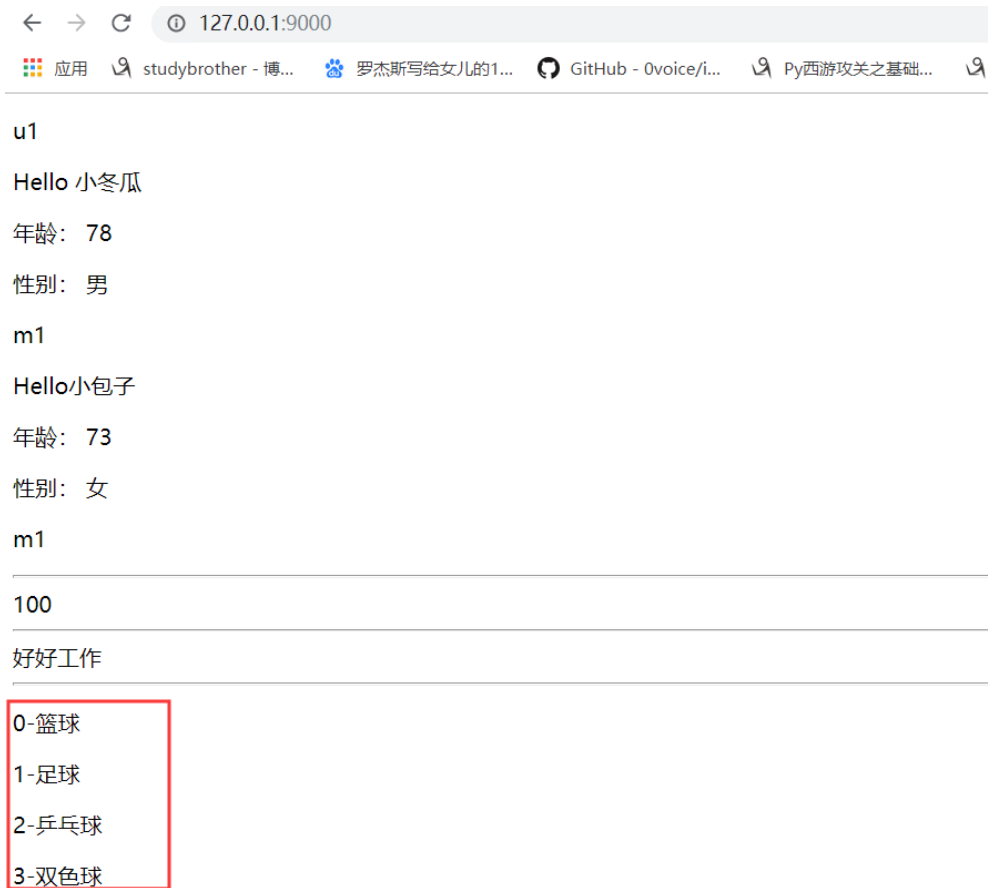
```
1 {{range pipeline}} T1 {{end}}
2 如果pipeline的值其长度为0，不会有任何输出
3
4 {{range pipeline}} T1 {{else}} T0 {{end}}
5 如果pipeline的值其长度为0，则会执行T0。
```

```
hello.tmpl x studybro_gin05\main.go x hello x studybro_gin04\main.go x
34 "Gender": "女",
35 "Age": 73,
36 }
37 hobbylist:=[]string{
38     "篮球",
39     "足球",
40     "乒乓球",
41     "双色球",
42 }
43 //name:= "小西瓜"
44 //err=t.Execute(w,name)
45 //err=t.Execute(w,u1)
46 //err=t.Execute(w,m1)
47 err=t.Execute(w,map[string]interface{}{
48     "u1":u1,
49     "m1":m1,
50     "hobby":hobbylist,
51 })
52 if err!=nil{
53     fmt.Println( a...: "render template failed,err:%v",err)
}
sayHello(w http.ResponseWriter, r *http.Request)
```

我们如何将上边的爱好全部遍历出来呢？ range

```
1 <hr>
2 {{range $idx,$hobby := .hobby}}
3 <p>{{$idx}}-{{$hobby}}</p>
4 {{end}}
```

运行，得到下面的结果：



当然，在这个range上边，我们还可以加上else
(25)

```
1 <hr>
2 {{range $idx,$hobby := .hobby}}
3 <p>{{$idx}}-{{$hobby}}</p>
4 {{else}}
5 什么爱好都没有
6 {{end}}
```

花钱的才叫爱好，不花钱的叫做陋习。

(26)

我们全部写.点号，太复杂了，我们可以用with来处理，构建一个局部作用域。

```
hello.tmpl x studybro_gin05\main.go x hello x
45
46 <hr>
47 <p>m1</p>
48 {{with .m1}}
49 <p>{{ .name }}</p>
50 <p>{{ .age }}</p>
51 <p>{{ .gender }}</p>
52 {{end}}
53 </body>
54 </html>
55
```

var . = m1

节省了很多的操作。

(27)

```
hello.tmpl x studybro_gin05\main.go x hello x
48 {{with .m1}}
49 <p>{{ .name }}</p>
50 <p>{{ .age }}</p>
51 <p>{{ .gender }}</p>
52 {{end}}
53
54 <hr>
55 {{ index .hobby 2 }}
56
57 </body>
58 </html>
```

举例，上边久表示，取索引值为2的变量

谢谢！

