

# 1、基本介绍

HttpRouter 是用于Go的轻量级高性能 HTTP 请求路由器（也称为多路复用器或简称mux）。

与Go包的默认多路复用器net/http相比，此路由器支持路由模式中的变量并匹配请求方法。它也可以更好地扩展。

该路由器针对高性能和小内存占用进行了优化。即使有很长的路径和大量的路线，它也能很好地扩展。压缩动态 trie（基数树）结构用于有效匹配。

## 2、基本使用

```
package hrouter

import (
    "fmt"
    "log"
    "net/http"

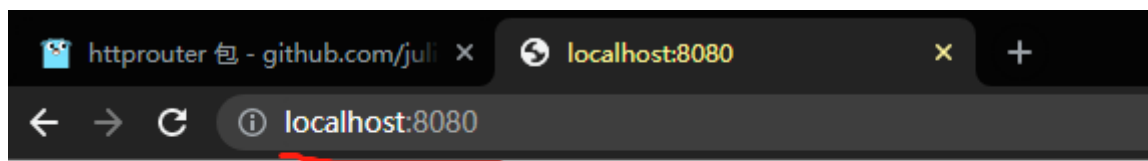
    "github.com/julienschmidt/httprouter"
)

func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
    fmt.Fprint(w, "<h1>welcome!</h1>\n")
}

func Hello(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
    fmt.Fprintf(w, "<h1>hello, %s!</h1>\n", ps.ByName("name"))
}

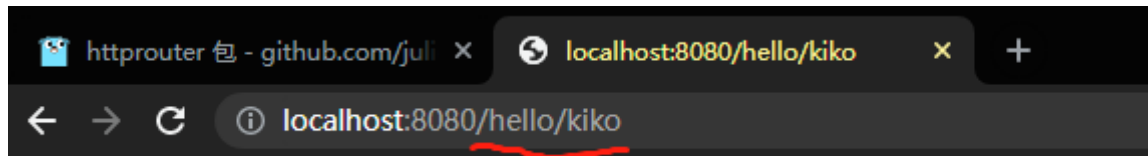
func TestRouter1() {
    router := httprouter.New()           // 得到一个路由
    router.GET("/", Index)               // 发送Get请求，调用Index函数
    router.GET("/hello/:name", Hello)   // 路由传参

    log.Fatal(http.ListenAndServe(":8080", router)) // 开启监听和服务
}
```



# Welcome!

CSDN @Golang-Study



# hello, kiko!



CSDN @Golang-Study

### 3、Http Method对应的方法

httprouter 为所有的HTTP Method 提供了快捷的使用方式，只需要调用对应的方法即可。

```
func (r *Router) GET(path string, handle Handle) {
    r.Handle("GET", path, handle)
}

func (r *Router) HEAD(path string, handle Handle) {
    r.Handle("HEAD", path, handle)
}

func (r *Router) OPTIONS(path string, handle Handle) {
    r.Handle("OPTIONS", path, handle)
}

func (r *Router) POST(path string, handle Handle) {
    r.Handle("POST", path, handle)
}

func (r *Router) PUT(path string, handle Handle) {
    r.Handle("PUT", path, handle)
}

func (r *Router) PATCH(path string, handle Handle) {
    r.Handle("PATCH", path, handle)
}

func (r *Router) DELETE(path string, handle Handle) {
    r.Handle("DELETE", path, handle)
}
```

CSDN @Golang-Study

#### Types

##### type Handle

```
type Handle func(http.ResponseWriter, *http.Request, Params)
```

Handle is a function that can be registered to a route to handle HTTP requests. Like `http.HandlerFunc`, but has a third parameter for the values of wildcards (variables).

##### type Param

added in v1.1.0

```
type Param struct {
    Key   string
    Value string
}
```

Param is a single URL parameter, consisting of a key and a value.

##### type Params

added in v1.1.0

```
type Params []Param
```

Params is a Param-slice, as returned by the router. The slice is ordered, the first URL parameter is also the first slice value. It is therefore safe to read values by the index.

##### func (Params) ByName

added in v1.1.0

```
func (ps Params) ByName(name string) string
```

ByName returns the value of the first Param which key matches the given name. If no matching Param is found, an empty string is returned.

CSDN @Golang-Study

```
router.go
hrouter > -o router.go > ...
9
10
11 // Handle处理函数
12 func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
13     fmt.Fprintf(w, "<h1>Welcome!</h1>\n")
14 }
15 func Hello(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
16     fmt.Fprintf(w, "<h1>hello, %s!</h1>\n", psByName("name"))
17 }
18 func GetUser(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
19     fmt.Fprintf(w, "<h1>you get a user %s</h1>\n", psByName("uid"))
20 }
21 func ModifyUser(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
22     fmt.Fprintf(w, "<h1>you modify a user %s</h1>\n", psByName("uid"))
23 }
24 func DeleteUser(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
25     fmt.Fprintf(w, "<h1>you delete a user %s</h1>\n", psByName("uid"))
26 }
27 func AddUser(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
28     fmt.Fprintf(w, "<h1>you add a user %s</h1>\n", psByName("uid"))
29 }
30
31 func TestRouter1() {
32
33     router := httprouter.New() // 得到一个路由
34
35     router.GET("/", Index) // 发送Get请求, 调用Index函数
36     router.GET("/hello/:name", Hello) // 路由传参
37
38     // http.Method行为
39     router.GET("/user/:uid", GetUser) // Get请求
40     router.POST("/AddUser/:uid", AddUser) // Post请求
41     router.DELETE("/delUser/:uid", DeleteUser) // Delete请求
42     router.PUT("/moduser/:uid", ModifyUser) // Put请求
43
44     log.Fatal(http.ListenAndServe(":8080", router)) // 开启监听和服务
45 }
46
```

通过httprouter包中的函数设置访问路由, 然后指定处理函数, 每一个函数都是Handle类型的

CSDN @Golang-Study

## 4、综合使用template\http\httprouter

```
package hrouter

import (
    "fmt"
    "html/template"
    "log"
    "net/http"

    "github.com/julienschmidt/httprouter"
)

func TestTemplateRouter() {
    // 准备路由
    router := httprouter.New()

    // GET请求1
    router.GET("/", func(w http.ResponseWriter, r *http.Request, p
httprouter.Params) {
        // 解析指定文件生成模板对象
        tmpl, err :=
template.ParseFiles("F:/tools/golang/goweb/hrouter/index.html")
        if err != nil {
            log.Fatal("Error parsing template: ", err)
        }
        // 利用给定数据渲染模板, 并将结果写入w
```

```

    tmpl.Execute(w, []string{"Java", "Golang", "Python", "C++", "", ""})
})

// GET请求2
router.GET("/user/:name", func(w http.ResponseWriter, r *http.Request, p
httprouter.Params) {
    // 解析指定文件生成模板对象
    tmpl, err :=
template.ParseFiles("F:/tools/golang/goweb/hrouter/user.html")
    if err != nil {
        log.Fatal("Error parsing template: ", err)
    }
    // 利用给定数据渲染模板, 并将结果写入w
    tmpl.Execute(w, []string{"Java", "Golang", "Python", "C++", "", ""})

    fmt.Fprintf(w, "Select a User %s", p.ByNames("name"))
})

// 服务器监听本地8080端口
err := http.ListenAndServe(":8080", router)
if err != nil {
    log.Fatal(err)
}
}

```

