链滴

GOLANG 中 HTTP 包默认路由匹配规则

作者：xiaoxiezaijia
原文链接：https://ld246.com/article/1567492418675
来源网站：链滴

还是在继续学习Go的路上，曾经在使用PHP的时候吃过过度依赖框架的亏。现在学习Go的时候决定打好基础，从标准库学起走。

源码分析

我们知道最简单的建立http服务器代码基本上都是这样的：

```
http.HandleFunc('/', func(w http.ResponseWriter, r *http.Request){
    fmt.Fprint(w, "Hello world")
})
http.ListenAndServe(":8080", nil)
```

这样就成功的建立了一个监听 8080 端口的http服务器，当访问的时候输出 Hello world

我们顺藤摸瓜来看看 HandleFunc 做了些什么事：

```
func HandleFunc(pattern string, handler func(ResponseWriter, *Request)) {
    DefaultServeMux.HandleFunc(pattern, handler)
}
```

这里继续通过调用 DefaultServeMux 的 HandleFunc 方法注册路由，这个 DefaultServeMux 又是方圣神：

```
type ServeMux struct {
    mu    sync.RWMutex
    m     map[string]muxEntry
    hosts bool // whether any patterns contain hostnames
}

type muxEntry struct {
    explicit bool
    h        Handler
    pattern  string
}

// NewServeMux allocates and returns a new ServeMux.
func NewServeMux() *ServeMux { return new(ServeMux) }

// DefaultServeMux is the default ServeMux used by Serve.
var DefaultServeMux = &defaultServeMux

var defaultServeMux ServeMux
```

DefaultServeMux 是 net/http 包提供的一个默认的 ServeMux 类型，ServeMux 实现了 Handler 口。

追根究底，发现http服务器收到一条请求后通过 go c.serve(ctx) 开启goroutine 处理这个请求，在个过程中调用了 Handler 接口函数 ServeHTTP 来做进一步的处理（比如匹配方法、链接等等）。

所以，我们就可以理解 ServeMux 就是 net/http 一个内置的路由功能。

继续回到 HandleFunc 来：

```
func (mux *ServeMux) HandleFunc(pattern string, handler func(ResponseWriter, *Request)) {
```

```
    mux.Handle(pattern, HandlerFunc(handler))
}
```

ServeMux 的 HandleFunc 方法将我们传入的路由具体实现函数转换成 HandlerFunc 类型并通过 Handle 注册到路由。这个 HandlerFunc 类型也实现了 Handler 接口：

```
type HandlerFunc func(ResponseWriter, *Request)

// ServeHTTP calls f(w, r).
func (f HandlerFunc) ServeHTTP(w ResponseWriter, r *Request) {
    f(w, r)
}
```

最后到了 Handle 这个方法， Handle 方法通过将 pattern 路径以及实现了 Handler 接口的方法一对应的保存到 ServeMux 的 map[string]muxEntry 中，方便后续请求的时候调用。因此，也可以通过 andle 直接传入一个实现了 Handler 接口的方法注册路由。

至此，net/http 包中默认路由的注册过程基本上已经走完。

至于请求的时候路由调用，记住通过 ServeHTTP 查找 map 中对应路径并调用相关方法就行了。

自制路由

通过以上的分析，我们可以依样画葫芦，实现自己的路由功能。

```
package route

import (
    "net/http"
    "strings"
)

//返回一个Router实例
func NewRouter() *Router {
    return new(Router)
}

//路由结构体，包含一个记录方法、路径的map
type Router struct {
    Route map[string]map[string]http.HandlerFunc
}

//实现Handler接口，匹配方法以及路径
func (r *Router) ServeHTTP(w http.ResponseWriter, req *http.Request) {
    if h, ok := r.Route[req.Method][req.URL.String()]; ok {
        h(w, req)
    }
}

//根据方法、路径将方法注册到路由
func (r *Router) HandleFunc(method, path string, f http.HandlerFunc) {
    method = strings.ToUpper(method)
    if r.Route == nil {
        r.Route = make(map[string]map[string]http.HandlerFunc)
```

```
    }
    if r.Route[method] == nil {
        r.Route[method] = make(map[string]http.HandlerFunc)
    }
    r.Route[method][path] = f
}
```

使用：

```
r := route.NewRouter()
r.HandleFunc("GET", "/", func(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "Hello Get!")
})
r.HandleFunc("POST", "/", func(w http.ResponseWriter, r *http.Request) {
    fmt.Fprint(w, "hello POST!")
})
http.ListenAndServe(":8080", r)
```

这个例子只是依样画葫芦的简单功能实现。

一个完整的路由框架应该包含更复杂的匹配、错误检测等等功能，大家可以试着自己动手试试。

阅读源码和重复造轮子都是学习的方法。

最后，欢迎大家关注我的博客http://targetliu.com/