golang http server探究（上）

Tudo 发表于 2年前 阅读 79 收藏 0 点赞 0 评论 0

HOT

在golang当中启动一个http服务非常简单，比如：

```
http.HandFunc("/",func(w http.RequestWriter,r *http.Request){
   io.WriteString(w,"hello world!")
})
http.ListenAndServer(":9090")   //outprint hello world!
```

为什么 访问 localhost:9090 就能打印出 Hello world 呢？这背后究竟发生了呢？下面我们就一层一层揭开这个面纱！ 1.追踪 http.HandFunc函数，发现它调用了：

```
DefaultServeMux.HandleFunc(pattern, handler)
```

DefaultServeMux实际是ServerMux（路由）的一个默认被初始化的一个结构，所以这个http.handfunc 最底层调用的是ServeMux.HandleFunc。在看这个函数之前，我们先看看，我们的第一个重要的结构体：ServerMux（路由）

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

type muxEntry struct {
    explicit bool
    h       Handler  // the handler to call
    pattern  string   //路由字符串  eg: /hello
}
```


分析：一个ServeMux 通过一个私有的muxEntry map保存了所有的路由。每个muxEntry 里面都包含有一个h（handler）处理器，用来出来这个路由的逻辑。

回到上面的ServerMux.HandleFunc。我们继续追踪这个函数，发现它调用了ServeMux.Handle()这个方法：

```
ServeMux.Handle(pattern string, handler Handler)
//mux.Handle(pattern, HandlerFunc(handler))  调用
```

ServeMux.Handle 这个函数 需要两个参数，一个参数 路由的path路径，另一个是一个handler（拥有这个路径具体处理逻辑的函数），这个handler 是什么东西呢？

```
type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}
```

Hander就是一个拥有ServerHTTP函数的类型。回到上面的ServeMux.Handle(path,HandleFunc(handle)),我们发现，从一开始，我们传进去的只是一个具体的函数，就是：

```
func(w http.RequestWriter,r *http.Request){
   io.WriteString(w,"hello world!")
}
```

这个函数，但是它怎么就是变成 Handler了呢？我们看看HandlerFunc这个东西？这个东西比较有意思。

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

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

HandlerFunc 是一个函数 类型，同时它有一个ServeHTTP这个方法，也就是它已经实现了 Handler这个接口，所以 可以用在所有以Handler为参数的地方，也就是可以用在 ServeMux.Handle() 的第2 个参数位置上。同时在HandFunc.ServeHTTP 内部它调用的 是HandFunc 这个函数(调用它自己)，换句话说，就是一个具体的函数，通过HandleFunc 这个类型转换以后就变成了Handle类型。这个技巧我们可以学习一下。

下面，我们看看，ServeMux.Handle 内部的具体实现：

```
func (mux *ServeMux) Handle(pattern string, handler Handler) {
    mux.mu.Lock()
    defer mux.mu.Unlock()

    if pattern == "" {
        panic("http: invalid pattern " + pattern)
    }
    if handler == nil {
        panic("http: nil handler")
    }
    if mux.m[pattern].explicit {
        panic("http: multiple registrations for " + pattern)
    }
    mux.m[pattern] = muxEntry{explicit: true, h: handler, pattern: pattern}

    //
    if pattern[0] != '/' {
        mux.hosts = true
    }
    n := len(pattern)
    if n > 0 && pattern[n-1] == '/' && !mux.m[pattern[0:n-1]].explicit {
        path := pattern
        if pattern[0] != '/' {
            path = pattern[strings.Index(pattern, "/"):]
        }
        url := &url.URL{Path: path}
        mux.m[pattern[0:n-1]] = muxEntry{h: RedirectHandler(url.String(), StatusMovedPermanently), pattern: pa
    }
}
```

这里简单说 就是 把 path，和对应的handle添加到ServerMux.muxEntry 里面。到这里，我们对路由的分析就完成了。现在我们在回顾一下：

下次我们 分析一下 Server。

关注程序猿公众账号：

分类：php 字数：751

标签： Go HTTP

Tudo

程序员 海淀

+ 关注

粉丝 4 | 博文 2 | 码字总数 1506

相关博客

Go 开发 HTTP	go http 分析	go语言的http包
傅小黑	solate	waynehu
242 1	81 0	10135 8

评论 (0)

Ctrl+Enter 发表评论