Atitit golang web go语言web

目录

[1. Go搭建一个Web服务器 1](#_Toc14364)

[1.1. http包建立Web服务器 1](#_Toc21426)

[1.2. 处理表单的输入 · Build web application with Golang 3](#_Toc26348)

[2. 5.1 database/sql接口 3](#_Toc18749)

[2.1.1. Go设置cookie 3](#_Toc9341)

# **Go搭建一个Web服务器**

前面小节已经介绍了Web是基于http协议的一个服务，Go语言里面提供了一个完善的net/http包，通过http包可以很方便的就搭建起来一个可以运行的Web服务。同时使用这个包能很简单地对Web的路由，静态文件，模版，cookie等数据进行设置和操作。

## **http包建立Web服务器**

package main

import (

"fmt"

"net/http"

"strings"

"log"

)

func sayhelloName(w http.ResponseWriter, r \*http.Request) {

r.ParseForm() //解析参数，默认是不会解析的

fmt.Println(r.Form) //这些信息是输出到服务器端的打印信息

fmt.Println("path", r.URL.Path)

fmt.Println("scheme", r.URL.Scheme)

fmt.Println(r.Form["url\_long"])

for k, v := range r.Form {

fmt.Println("key:", k)

fmt.Println("val:", strings.Join(v, ""))

}

fmt.Fprintf(w, "Hello astaxie!") //这个写入到w的是输出到客户端的

}

func main() {

http.HandleFunc("/", sayhelloName) //设置访问的路由

err := http.ListenAndServe(":9090", nil) //设置监听的端口

if err != nil {

log.Fatal("ListenAndServe: ", err)

}

}

上面这个代码，我们build之后，然后执行web.exe,这个时候其实已经在9090端口监听http链接请求了。

在浏览器输入http://localhost:9090

可以看到浏览器页面输出了Hello astaxie!

可以换一个地址试试：http://localhost:9090/?url\_long=111&url\_long=222

看看浏览器输出的是什么，服务器输出的是什么？

在服务器端输出的信息如下：

图3.8 用户访问Web之后服务器端打印的信息

我们看到上面的代码，要编写一个Web服务器很简单，只要调用http包的两个函数就可以了。

如果你以前是PHP程序员，那你也许就会问，我们的nginx、apache服务器不需要吗？Go就是不需要这些，因为他直接就监听tcp端口了，做了nginx做的事情，然后sayhelloName这个其实就是我们写的逻辑函数了，跟php里面的控制层（controller）函数类似。

如果你以前是Python程序员，那么你一定听说过tornado，这个代码和他是不是很像，对，没错，Go就是拥有类似Python这样动态语言的特性，写Web应用很方便。

如果你以前是Ruby程序员，会发现和ROR的/script/server启动有点类似。

我们看到Go通过简单的几行代码就已经运行起来一个Web服务了，而且这个Web服务内部有支持高并发的特性，我将会在接下来的两个小节里面详细的讲解一下Go是如何实现Web高并发的。

## 处理表单的输入 · Build web application with Golang

# **5.1 database/sql接口**

Go与PHP不同的地方是Go官方没有提供数据库驱动，而是为开发数据库驱动定义了一些标准接口，开发者可以根据定义的接口来开发相应的数据库驱动，这样做有一个好处，只要是按照标准接口开发的代码， 以后需要迁移数据库时，不需要任何修改。那么Go都定义了哪些标准接口呢？让我们来详细的分析一下

### **Go设置cookie**

Go语言中通过net/http包中的SetCookie来设置：

http.SetCookie(w ResponseWriter, cookie \*Cookie)