

Go是一门正在快速增长的编程语言，专为构建简单、快速且可靠的软件而设计。golang提供的nethttp库已经很好了，对于http的协议的实现非常好，基于此再造框架，也不会是难事，因此生态中出现了很多框架。

- Gin: Go 语言编写的Web框架，以更好的性能实现类似Martini框架的API。
 - Gin是一个golang的微框架，封装比较优雅，API友好，源码注释比较明确。具有快速灵活，容错方便等特点。Beego:开源的高性能Go语言Web框架。
- beego是一个快速开发Go应用的http框架，go语言方面技术大牛。
 - beego可以用来快速开发API、Web、后端服务等各种应用，是一个RESTFul的框架，主要设计灵感来源于tornado、sinatra、flask这三个框架，但是结合了Go本身的一些特性(interface、struct继承等)而设计的一个框架。Iris:全宇宙最快的Go语言Web框架。完备MVC支持，未来尽在掌握。
- Iris是一个快速、简单但功能齐全的和非常有效的web框架。
 - 提供了一个优美的表现力和容易使用你的下一个网站或API的基础。

Gin

安装Gin

Gin是一个golang的微框架，封装比较优雅，API友好，源代码比较明确。具有快速灵活，容错方便等特点。其实对于golang而言，**web框架的依赖远比Python，Java之类的要小**。自身的net/http足够简单，性能也非常不错。框架更像是一个常用函数或者工具的集合。

借助框架开发不仅可以省去很多常用的封装带来的时间，也有助于团队的编码风格和形成规范。

Gin官方文档地址: <https://gin-gonic.com/zh-cn/docs/>

安装命令：

```
go get -u github.com/gin-gonic/gin
```

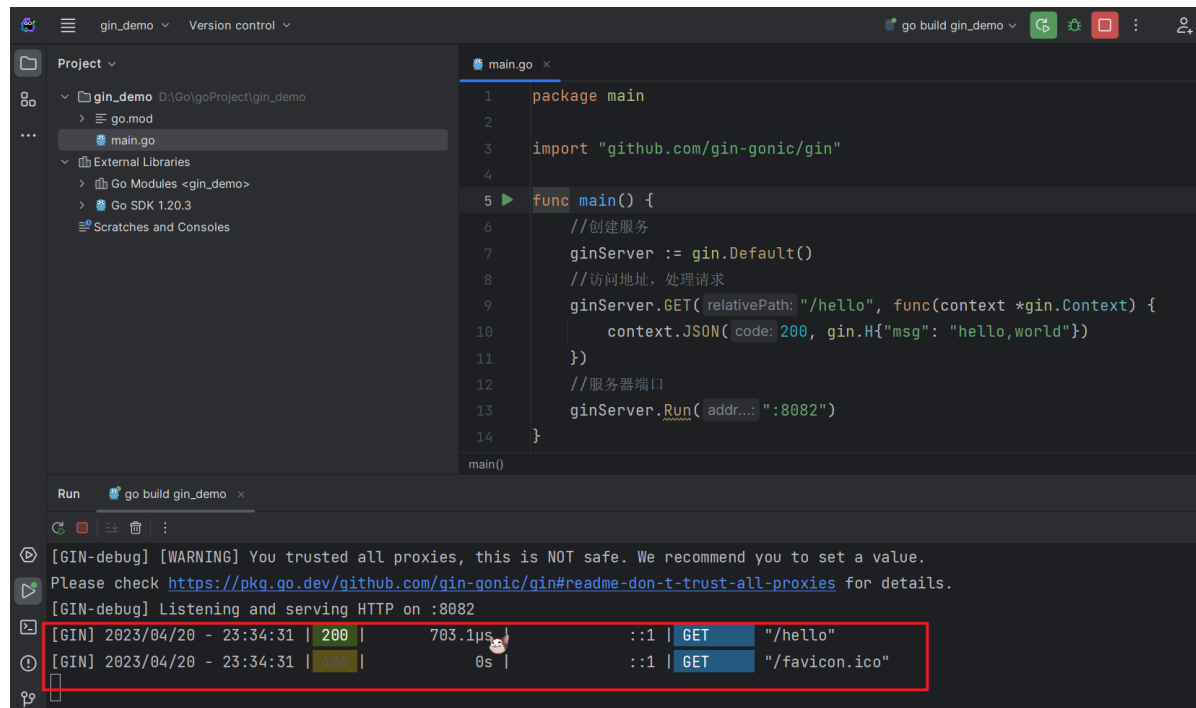
HelloWorld

```
package main

import "github.com/gin-gonic/gin"

func main() {
    //创建服务
    ginServer := gin.Default()
    //访问地址，处理请求
    ginServer.GET("/hello", func(context *gin.Context) {
        context.JSON(200, gin.H{"msg": "hello,world"})
    })
    //服务器端口
    ginServer.Run(":8082")
}
```

目录结构



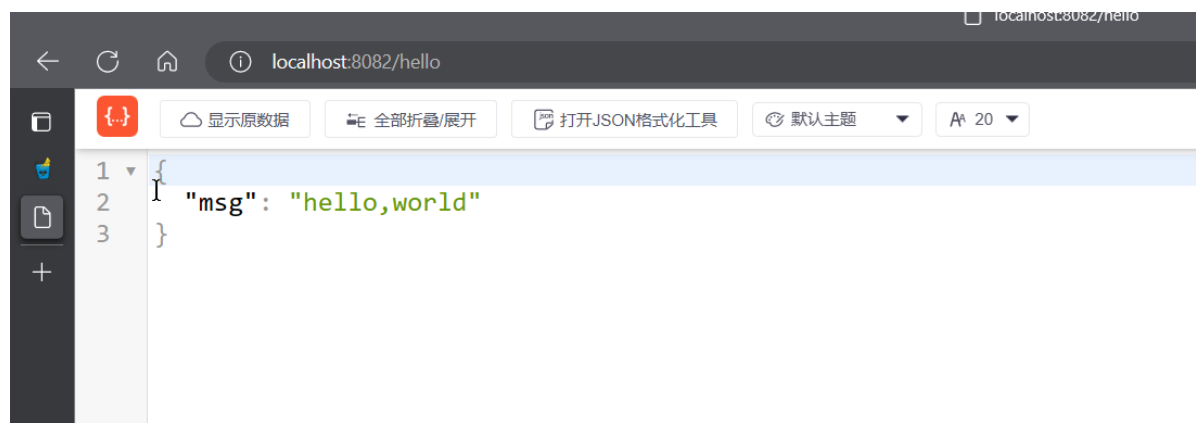
```
package main

import "github.com/gin-gonic/gin"

func main() {
    //创建服务
    ginServer := gin.Default()
    //访问地址, 处理请求
    ginServer.GET(relativePath: "/hello", func(context *gin.Context) {
        context.JSON(code: 200, gin.H{"msg": "hello,world"})
    })
    //服务器端口
    ginServer.Run(addr...: ":8082")
}
```

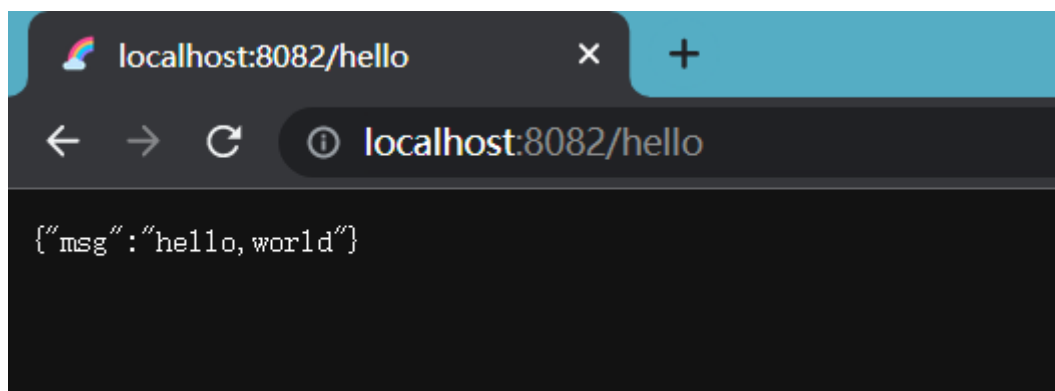
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check <https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies> for details.
[GIN-debug] Listening and serving HTTP on :8082
[GIN] 2023/04/20 - 23:34:31 | 200 | 703.1µs | :1 | GET | "/hello"
[GIN] 2023/04/20 - 23:34:31 | 404 | 0s | :1 | GET | "/favicon.ico"

访问localhost:8082/hello

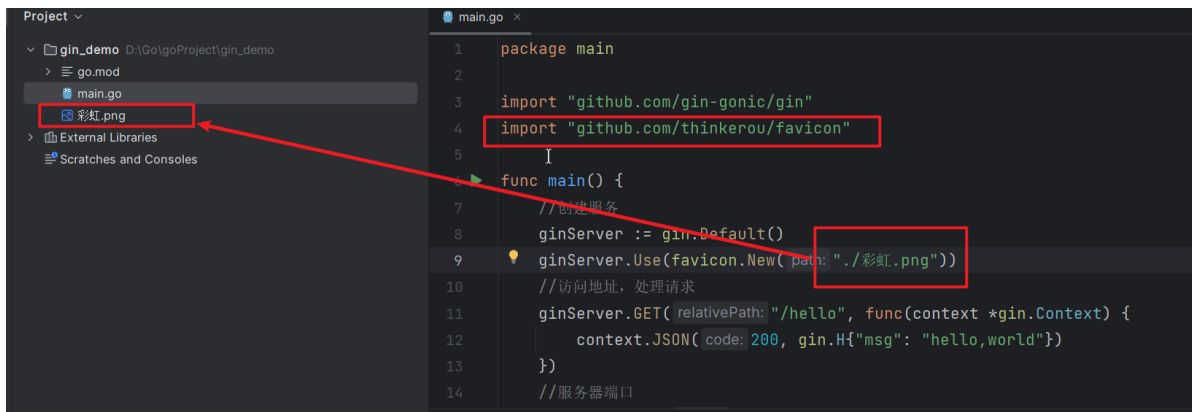


添加icon

效果



实现代码



```
package main

import "github.com/gin-gonic/gin"
import "github.com/thinkerou/favicon"

func main() {
    //创建服务
    ginServer := gin.Default()
    ginServer.Use(favicon.New("./彩虹.png"))
    //访问地址，处理请求
    ginServer.GET("/hello", func(context *gin.Context) {
        context.JSON(200, gin.H{"msg": "hello,world"})
    })
    //服务器端口
    ginServer.Run(":8082")
}
```

利用restful形式开发

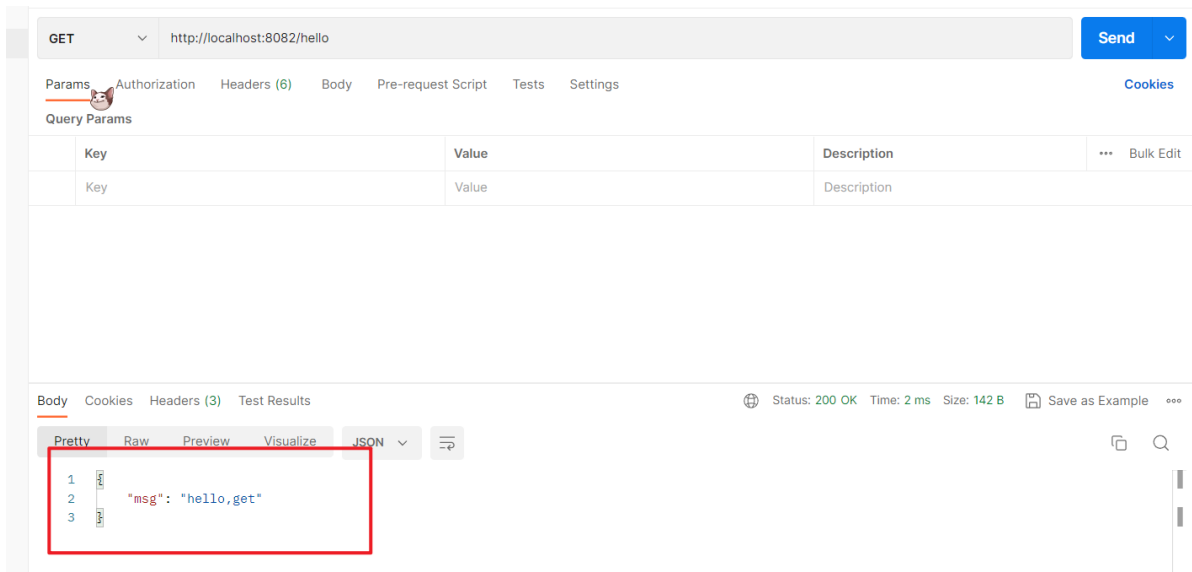
```
package main

import "github.com/gin-gonic/gin"
import "github.com/thinkerou/favicon"

func main() {
    //1.创建服务
    ginServer := gin.Default()
    ginServer.Use(favicon.New("./彩虹.png"))
    //2.访问地址，处理请求
    //2.1 利用restful形式开发接口，并以json的形式返回
    ginServer.GET("/hello", func(context *gin.Context) {
        context.JSON(200, gin.H{"msg": "hello,get"})
    })
    ginServer.POST("/hello", func(context *gin.Context) {
        context.JSON(200, gin.H{"msg": "hello,post"})
    })
    ginServer.DELETE("/hello", func(context *gin.Context) {
        context.JSON(200, gin.H{"msg": "hello,DELETE"})
    })
    ginServer.PUT("/hello", func(context *gin.Context) {
        context.JSON(200, gin.H{"msg": "hello,PUT"})
    })
}
```

```
//2.2 返回页面
//3.配置服务器端口
ginServer.Run(":8082")
}
```

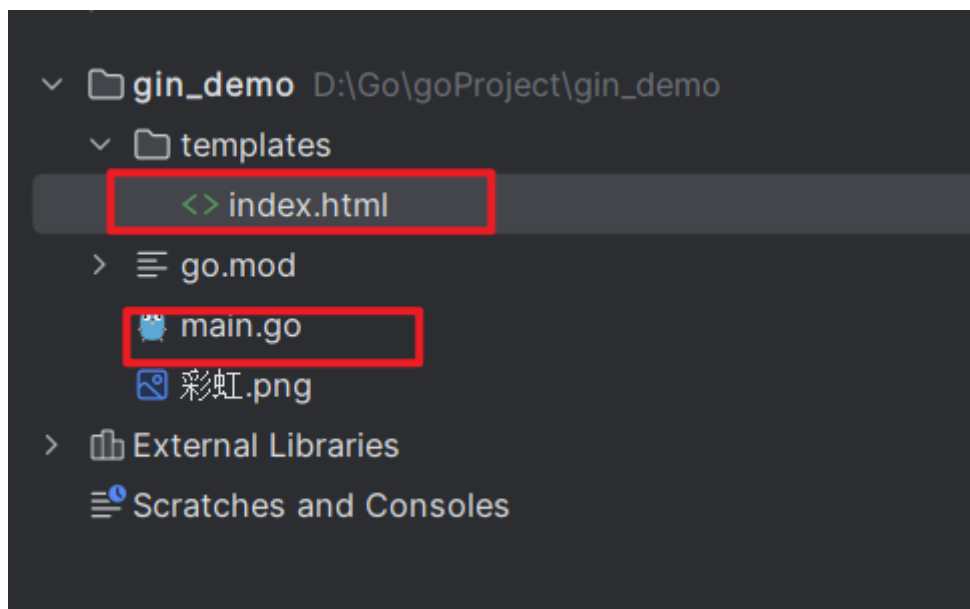
测试都是成功的



返回页面

返回了页面并且后端向前端传输了数据

目录结构:



代码实现:

index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>这是Gin的页面</title>
</head>
<body>
    <h1>hello Gin</h1>
<!--    下面这个代码可以访问key-value类型的数据-->
    {{.msg}}
</body>
</html>

```

main.go

```

package main

import (
    "github.com/gin-gonic/gin"
    "net/http"
)
import "github.com/thinkerou/favicon"

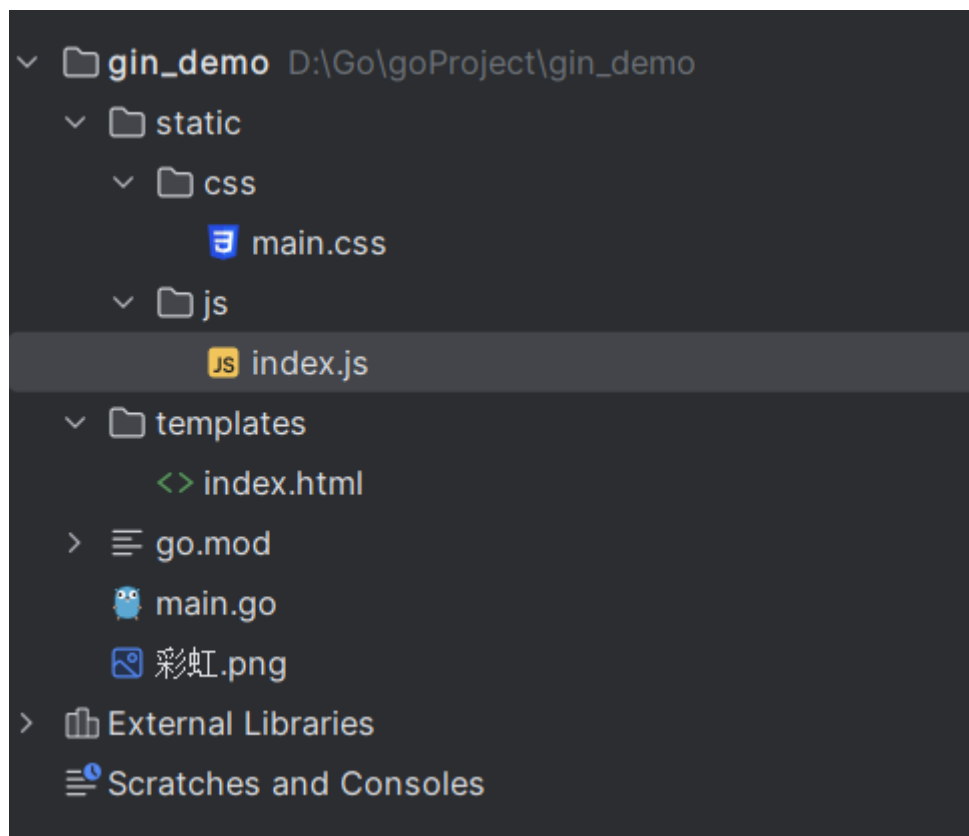
func main() {
    //1.创建服务
    ginServer := gin.Default()
    ginServer.Use(favicon.New("./彩虹.png"))
    //加载静态页面到web服务器中
    //下面这行代码的意思是加载./templates/*目录下的全部文件
    ginServer.LoadHTMLGlob("templates/*")
    //2.访问地址，处理请求
    // 返回页面
    ginServer.GET("/index", func(context *gin.Context) {
        //第一个参数是响应码，这是个常量200
        //第二个参数是templates下返回的页面
        //第三个参数是后端传递给前端的数据
        context.HTML(http.StatusOK, "index.html", gin.H{"msg": "这是后端传送的数据"})
    })
    //3.配置服务器端口
    ginServer.Run(":8082")
}

```



引入样式和脚本文件

目录结构

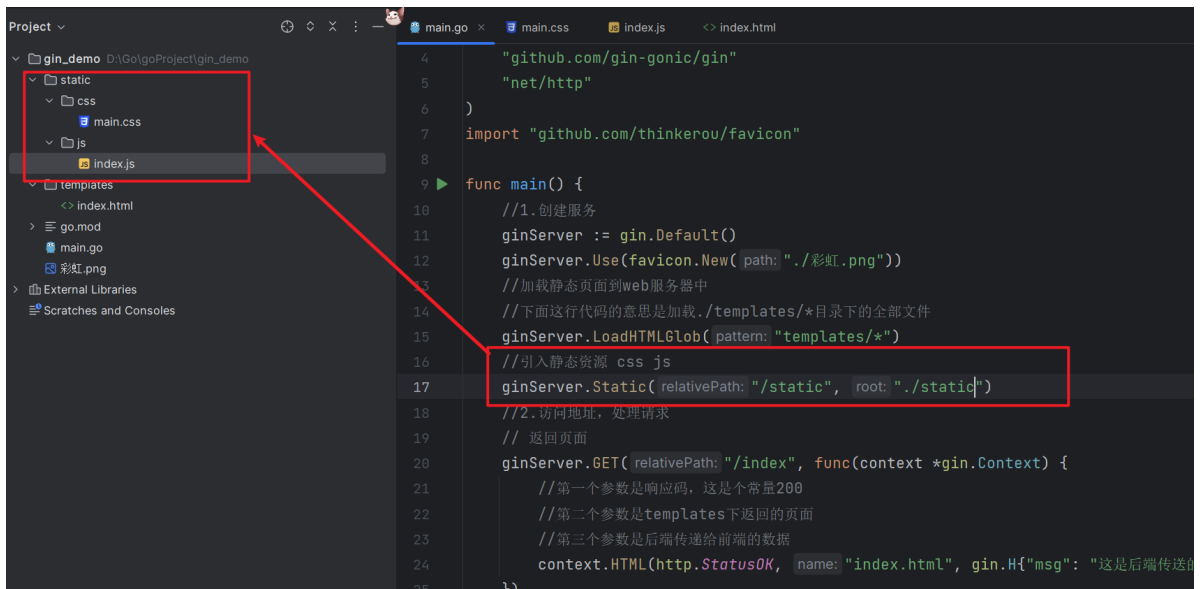


代码实现

index.html

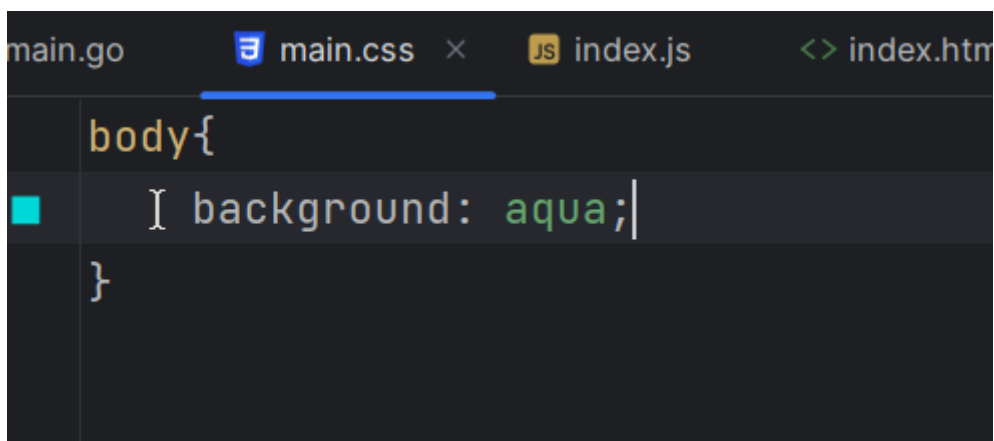
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>这是Gin的页面</title>
6 </head>
7 <link rel="stylesheet" href="/static/css/main.css">
8 <script src="/static/js/index.js"></script>
9 <body>
10     <h1>hello Gin</h1>
11 <!-- 下面这个代码可以访问key-value类型的数据 -->
12     {{.msg}}
13 </body>
```

main.go



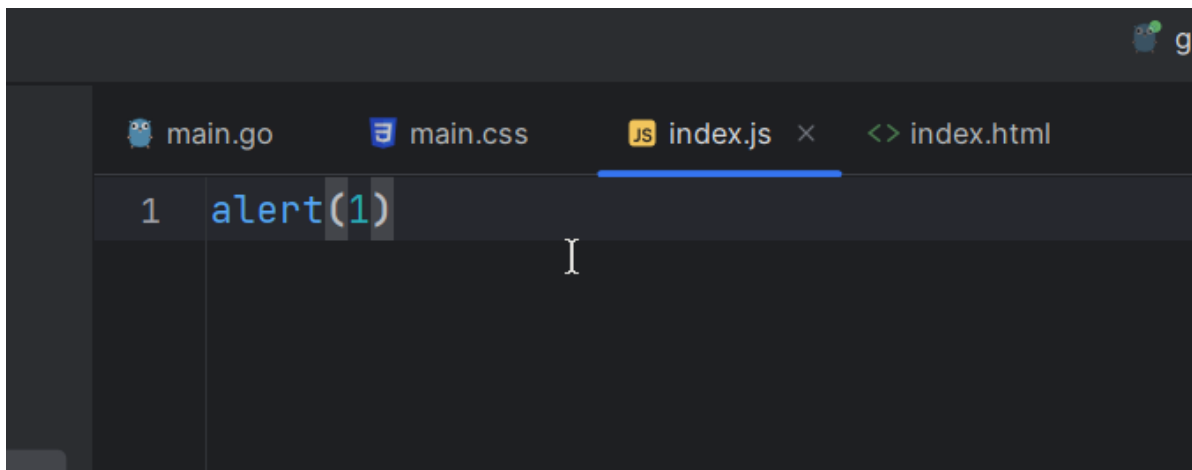
```
4 "github.com/gin-gonic/gin"
5 "net/http"
6
7 import "github.com/thinkerou/favicon"
8
9 func main() {
10     //1. 创建服务
11     ginServer := gin.Default()
12     ginServer.Use(favicon.New(path: "./彩虹.png"))
13     // 加载静态页面到web服务器中
14     // 下面这行代码的意思是加载./templates/*目录下的全部文件
15     ginServer.LoadHTMLGlob(pattern: "templates/*")
16     // 引入静态资源 css js
17     ginServer.Static(relativePath: "/static", root: "./static")
18     //2. 访问地址, 处理请求
19     // 返回页面
20     ginServer.GET(relativePath: "/index", func(context *gin.Context) {
21         // 第一个参数是响应码, 这是个常量200
22         // 第二个参数是templates下返回的页面
23         // 第三个参数是后端传递给前端的数据
24         context.HTML(http.StatusOK, name: "index.html", gin.H{"msg": "这是后端传递的"})
25     })
26 }
```

main.css



```
body{
    background: aqua;
}
```

index.js



前后端传参

前端通过url方式传参给后端

前端通过url方式传参并且后端接收到参数回显到前端

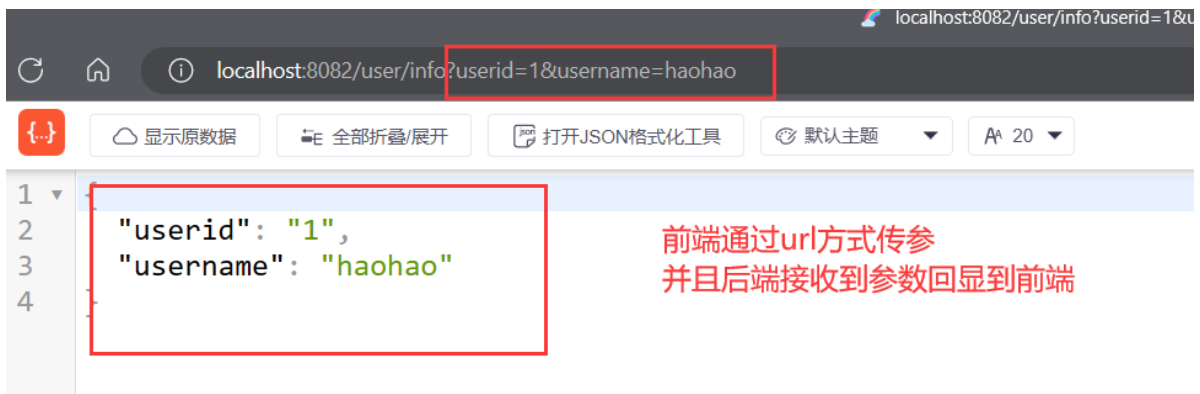
代码实现:

```
//前端浏览器的url携带参数
// localhost:8082/user/info?userid=1&username=haohao
ginServer.GET(relativePath: "/user/info", func(context *gin.Context) {
    userid := context.Query(key: "userid")
    username := context.Query(key: "username")
    context.JSON(http.StatusOK, gin.H{"userid": userid, "username": username})
})
```

取得参数

向前端回显

运行效果:

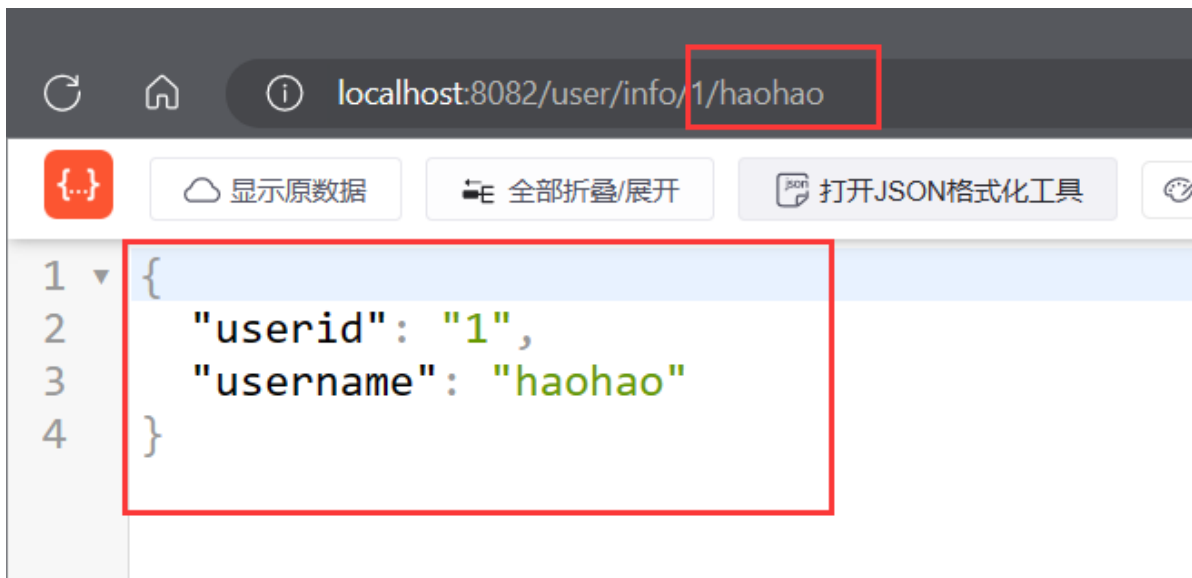


前端通过restful方式传参给后端

代码实现:

```
//前端通过restful方式传参给后端
// localhost:8082/user/info/1/haohao
ginServer.GET(relativePath: "/user/info/:userid/:username", func(context *gin.Context) {
    userid := context.Param(key: "userid")
    username := context.Param(key: "username")
    context.JSON(http.StatusOK, gin.H{"userid": userid, "username": username})
})
```

运行效果:



前端通过发送json的方式传参给后端

接收代码

```
//前端通过发送json的方式传参给后端
ginServer.POST(relativePath: "/json", func(context *gin.Context) {
    //获得requestBody, 第二个参数可以不接收
    data, _ := context.GetRawData()
    var m map[string]interface{}
    _ = json.Unmarshal(data, &m)
    context.JSON(http.StatusOK, m)
})
```

传参:



前端通过form表单的形式给后端传输数据

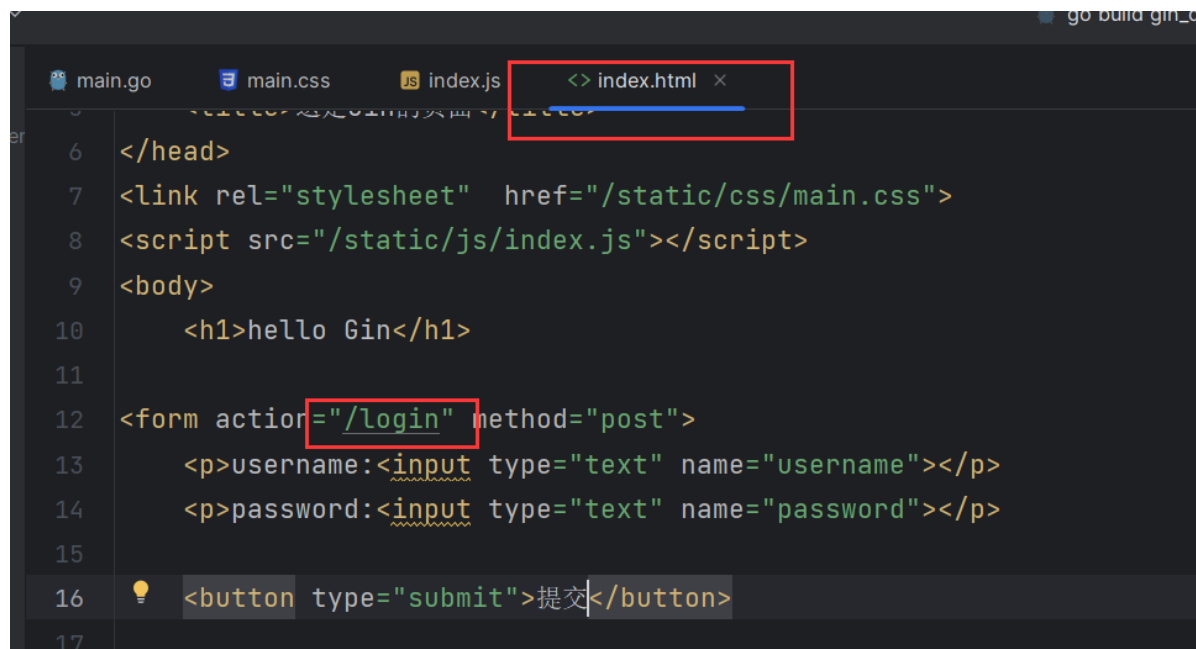
注意这里使用了我们之前写的index接口，才可以访问到index.html页面的

代码实现：

后端

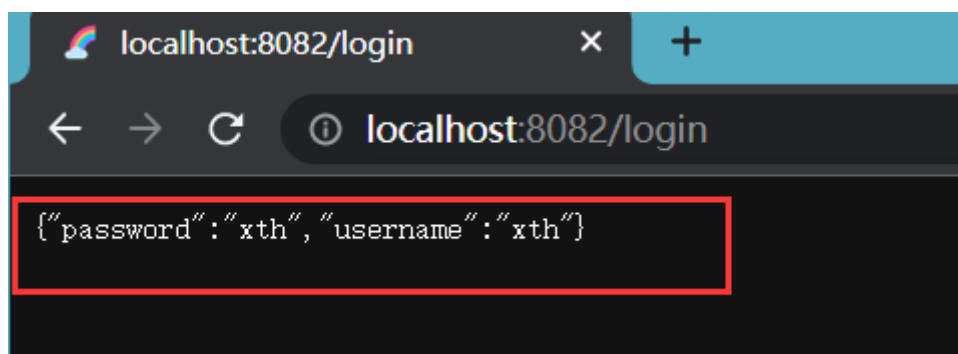
```
//前端通过form表单的形式给后端传输数据
//访问 localhost:8082/index 向localhost:8082/login提交数据
ginServer.POST(relativePath: "/login", func(context *gin.Context) {
    username := context.PostForm(key: "username")
    password := context.PostForm(key: "password")
    context.JSON(http.StatusOK, gin.H{"username": username, "password": password})
})
```

前端



```
main.go main.css index.js <> index.html x
6 </head>
7 <link rel="stylesheet" href="/static/css/main.css">
8 <script src="/static/js/index.js"></script>
9 <body>
10     <h1>hello Gin</h1>
11
12     <form action="/login" method="post">
13         <p>username:<input type="text" name="username"></p>
14         <p>password:<input type="text" name="password"></p>
15
16         <button type="submit">提交</button>
17
```

实际效果：

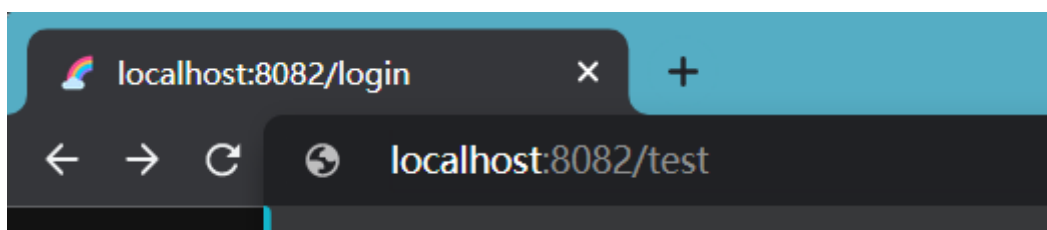


路由

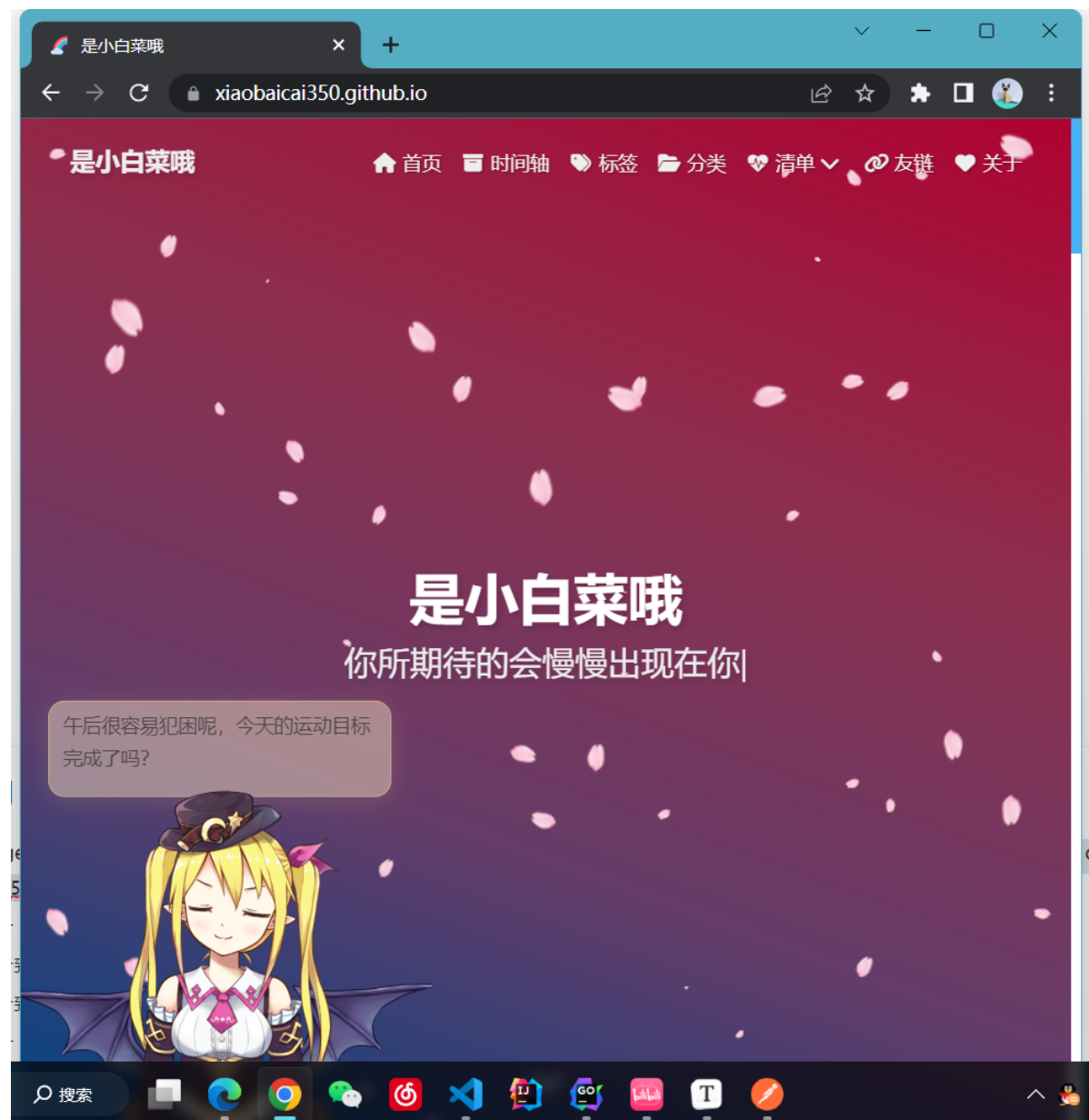
重定向

后端代码实现:

```
//路由
ginServer.GET("/test", func(context *gin.Context) {
    //重定向
    //context.Redirect(301, "https://xiaobaicai350.github.io/")
    //需要注意的是第一个参数也就是状态码，必须和业务处理的状态一样
    context.Redirect(http.StatusMovedPermanently,
        "https://xiaobaicai350.github.io/")
})
```

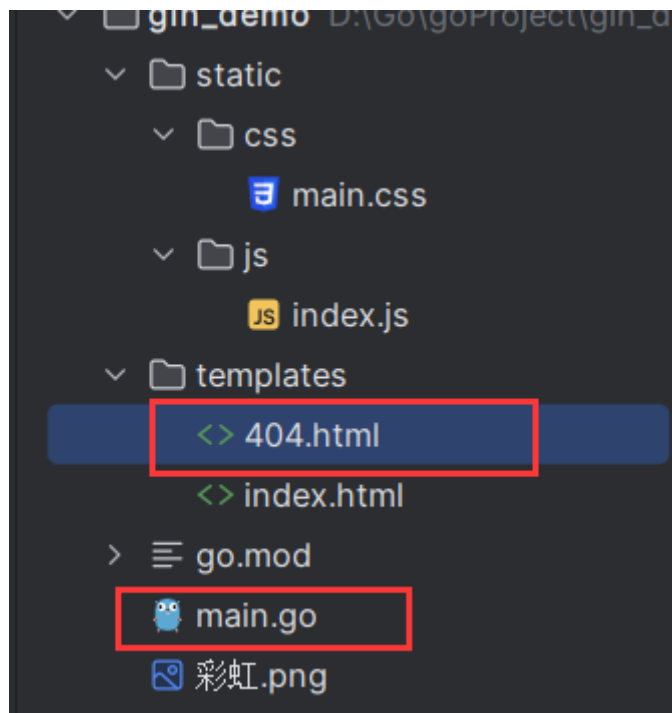


重定向到了我的博客



404

目录结构



main.go

```
ginServer.NoRoute(func(context *gin.Context) {  
    //第二个参数: 这里的404.html会默认去templates下面去找, 因为我们之前以及加载过资源了  
    //第三个参数: 需要给后端传输的数据, 这里啥也不传都行  
    context.HTML(http.StatusNotFound, name: "404.html", obj: nil)  
})
```

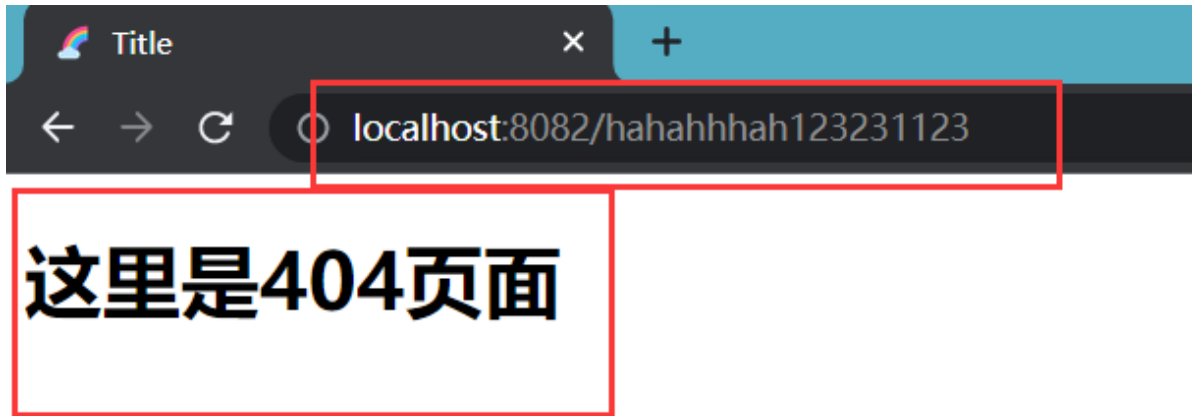
404.html

A screenshot of a code editor with three tabs: 'main.go', '<> 404.html', and 'main.css'. The '404.html' tab is active, showing the following HTML code:

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4     <meta charset="UTF-8">  
5     <title>Title</title>  
6 </head>  
7 <body>  
8 <h1>这里是404页面</h1>  
9 </body>  
10 </html>
```

The line '<h1>这里是404页面</h1>' on line 8 is highlighted with a red box.

访问一个**随便的后缀**，后端会帮我们拦截到，然后转发到404页面



路由组

说白了就是加个前缀

```
//路由组
userGroup := ginServer.Group("/user")
{
    //localhost:8082/user/add
    userGroup.GET("/add", func(context *gin.Context) {
        context.HTML()//这里可以写具体的处理，之前已经写过很多了，这里就不再写了
    })
    //localhost:8082/user/logout
    userGroup.GET("/logout")
    //localhost:8082/user/login
    userGroup.GET("/login")
}
```

中间件

就是Java里面的**拦截器**

定义我们自己的拦截器：

```
// 自定义Go的中间件，也就是拦截器
func myHandler() gin.HandlerFunc { 1 usage
    return func(context *gin.Context) {
        //通过自定义的中间件，给context里面设置值
        //之后如果有哪个controller加上了这个拦截器，都可以从这里取到值
        context.Set(key: "key10086", value: "value149948")
        context.Next() //放行
        //context.Abort()//阻止
    }
}
```

使用拦截器：

```
//localhost:8082/testHandler
//注意在这里的添加了第二个参数，也就是加了我们自己定义的拦截器
ginServer.GET(relativePath: "/testHandler", myHandler(), func(context *gin.Context) {

    value1, exists := context.Get(key: "key10086")
    s1 := value1.(string)
    value2 := context.MustGet(key: "key10086")
    s2 := value2.(string)
    println(s1, exists)
    println(s2)
}))
```

这两个方法都是可以获取到值的
但是有细微的差别，以后可以研究一下

验证结果：

访问<http://localhost:8082/testHandler>

```
value149948 true
value149948
[GIN] 2023/04/21 - 17:32:46 | 200 |           0s |           ::1 | GET | "/testHandler"
```