

# Gin

---

Web 本质

Gin 框架初识

各类请求方法

返回 JSON

Querystring 参数

表单参数

uri 参数

参数绑定

重定向:

站外重定向:

站内重定向

路由

中间件

Gin 默认中间件

Gin 中间件中使用goroutine

## Web 本质

本质就是一个请求(request)对应一个响应(responce)

最简单的一个 go 服务:

```
1  package main
2
3  import (
4      "fmt"
5      "net/http"
6  )
7
8  func sayHello(w http.ResponseWriter, r *http.Request) {
9      _, _ = fmt.Fprintln(w, "hello,http")
10 }
11 func main() {
12     http.HandleFunc("/hello", sayHello)
13     err := http.ListenAndServe(":8000", nil)
14     if err != nil {
15         fmt.Printf("http serve error ")
16         return
17     }
18 }
19
```

## Gin 框架初识

### 各类请求方法

```
1
2 func main() {
3     r := gin.Default()
4     r.GET("/path/:id", func(c *gin.Context) {
5         id := c.Param("id")
6         user := c.DefaultQuery("user", "ccc")
7         pwd := c.Query("pwd")
8         c.JSON(200, gin.H{
9             "id": id,
10            "user": user,
11            "pwd": pwd,
12        })
13    })
14    r.POST("/path", func(c *gin.Context) {
15        user := c.DefaultPostForm("user", "ccc")
16        pwd := c.PostForm("pwd")
17        c.JSON(200, gin.H{
18            "user": user,
19            "pwd": pwd,
20        })
21    })
22    r.DELETE("/path/:id", func(c *gin.Context) {
23        id := c.Param("id")
24        c.JSON(200, gin.H{
25            "id": id,
26        })
27    })
28    r.PUT("/path", func(c *gin.Context) {
29        user := c.DefaultPostForm("user", "ccc")
30        pwd := c.PostForm("pwd")
31        c.JSON(200, gin.H{
32            "user": user,
33            "pwd": pwd,
34        })
35    })
36    r.Run(":1010")
37 }
38
```

## 返回 JSON

Gin 返回 JSON 主要有两种方式：`map` 和 `struct`。

其中, `struct` 的字段如果是不可导出的, 无法正常序列化, 但是可以通过 `tag` 指定字段名的方式序列化

```
1 // 1. 使用 map 序列化 json
2 data := map[string]interface{}{
3     "name": "max",
4     "message": "hello",
5     "age": 19,
6 }
7 c.JSON(http.StatusOK, data)
8
9 type msg struct {
10     Name string // 不可导出字段无法序列化 如果首字母一定要小写, 可以使用 tag 指定字段名
11     Message string `json:"message"`
12     Age int
13 }
14 data := msg{
15     "max",
16     "hello",
17     19,
18 }
19 c.JSON(http.StatusOK, data)
20
```

## Querystring 参数

用法:

在 `URL` 后加 `?querystring` 即可。

比如: `http://localhost:8080/user?name=max` 查询名称为 `max` 的用户

在服务端可以使用 `c.Query("name")` 获取到 `max` 值

当需要查询多个内容时, 不同内容之间使用 `&` 相连接。

除了 `c.Query()` 外, 类似的还有 `c.DefaultQuery`: 如果没有响应结果, 就返回默认的值

`c.GetQuery()` 判断是否有该参数

```
1 func queryString() {
2     e := gin.Default()
3     // GET 请求 url? 后是 querystring
4     // key=value 格式, 多个 key-value 之间用 & 连接
5     // eq: /web?query=max&age=ccc
6     e.GET("/web", func(c *gin.Context) {
7         //获取浏览器请求的 query string 参数
8         name := c.Query("query") // 通过 query 获取请求中携带的 querystring
          参数 http://127.0.0.1:8080/web?query=max
9         age := c.Query("age")    // url 可以通过 & 符号查询两个参数, 比如: http://127.0.0.1:8080/web?query=max&age=19
          todo: 为什么返回的 json 里 age 在前, name 在后
10        //name := c.DefaultQuery("query", "ccc") // 如果没有 query, 就返回指定的默认值: http://127.0.0.1:8080/web?aaa=123
11        //name, ok := c.GetQuery("query") // 如果没有 query, 返回 false
12        //if !ok {
13        //    name = "ccc"
14        //}
15        c.JSON(http.StatusOK, gin.H{
16            "name": name,
17            "age":  age,
18        })
19    })
20    e.Run()
21 }
```

## 表单参数

获取表单中的数据。与 `c.Query()` 类似

```
1 func formParse() {
2     e := gin.Default()
3     e.LoadHTMLFiles("./template/login.html", "./template/index.html")
4     e.GET("/login", func(c *gin.Context) {
5         c.HTML(http.StatusOK, "login.html", nil)
6     })
7     // 第一种获取 form 表单提交的数据
8     // 其余两种参考 querystring
9     // login post
10    e.POST("/login", func(c *gin.Context) {
11        username := c.PostForm("username")
12        password := c.PostForm("password")
13        c.HTML(http.StatusOK, "index.html", gin.H{
14            "Name": username,
15            "password": password,
16        })
17    })
18    e.Run()
19 }
```

## uri 参数

获取在 url 中传递的参数对应的值

```
1 // 获取请求的 path(URI)参数
2 func main() {
3     e := gin.Default()
4     // http://127.0.0.1:8080/user/max/12
5     // ep :http://127.0.0.1:8080/blog/2023/10
6     e.GET("/user/:name/:age", func(c *gin.Context) {
7         name := c.Param("name")
8         age := c.Param("age")
9         c.JSON(http.StatusOK, gin.H{
10             "name": name,
11             "age": age,
12         })
13     })
14
15     e.Run()
16 }
```

## 参数绑定

获取 `JSON` 中的数据并绑定到某个 Go 结构体中。

`ShoudBind` 会按照下面的顺序解析请求中的数据完成绑定：

1. 如果是 `GET` 请求，只使用 `Form` 绑定引擎(query)
2. 如果是 `POST` 请求，首先检查 `content-type` 是否为 `JSON` 或 `xml`，然后再使用 `Form` (form-data)

```
1  type user struct {
2      Username string `form:"username"`
3      Password string `form:"password"`
4  }
5
6  func main() {
7      e := gin.Default()
8
9      e.GET("/user", func(c *gin.Context) {
10         //username := c.Query("username")
11         //password := c.Query("password")
12         //u := user{
13             // Username: username,
14             // Password: password,
15             //}
16         u := user{}
17         err := c.ShouldBind(&u) // 函数传参是值传递，所以如果不加取址符，修改的是
            u 的副本，而不是 u 本身
18         if err != nil {
19             c.JSON(http.StatusBadRequest, gin.H{
20                 "error": err,
21             })
22             return
23         }
24         fmt.Printf("%#v\n", u)
25         c.JSON(http.StatusOK, gin.H{
26             "message": "ok",
27         })
28     })
29     e.Run()
```

## 重定向：

主要有两种重定向方式：

1. 站外重定向
2. 站内重定向

## 站外重定向：



主要是使用 `c.Redirect(http.StatusMovedPermanently, "https://www.baidu.com")`

其中 `http.StatusMovedPermanently` 的编码为301

这个方法可以让用户重定向到某个外站站点。

```
1 // 跳转到百度
2 e.GET("/index", func(c *gin.Context) {
3     c.Redirect(http.StatusMovedPermanently, "https://www.baidu.com")
4 })
```

## 站内重定向

站内重定向主要是指跳转到站内的其他路由

比如：用户访问路由 `"/a"` 时，系统自动跳转到路由 `"/b"`。

```
1 // 路由跳转
2 e.GET("/a", func(c *gin.Context) {
3     c.Request.URL.Path = "/b" // 把请求的 URL 修改
4     e.HandleContext(c)        // 继续后续处理
5 })
6 e.GET("/b", func(c *gin.Context) {
7     c.JSON(http.StatusOK, gin.H{
8         "status": "ok",
9     })
10 })
```

## 路由

主要是各个路由方法，除了 RESTFUL API 外，Gin 还支持，`Any` 路由，其内部是实现了所有请求方法的相应处理。还有路由组的相关应用：

```
1 // 路由组
2 // 视频的首页和详情页
3 videoG := e.Group("/video")
4 {
5     videoG.GET("/index", func(c *gin.Context) {
6         c.JSON(http.StatusOK, gin.H{"message": "/video/index"})
7     })
8 }
```

最后 Gin 支持 `NoRoute` 方法，即如果用户访问了站内没有的路由，同一跳转到某个页面。

```
1 e.NoRoute(func(c *gin.Context) {
2     c.JSON(http.StatusNotFound, gin.H{
3         "msg": "https://www.baidu.com",
4     })
5 })
```

## 中间件

### 什么是中间件？

Gin 框架允许开发者在处理请求的工程中，加入用户自己的钩子函数。这个钩子函数就叫中间件，中间件适合处理一些公共的业务逻辑，比如登录认证、权限校验、数据分页、记录日志、耗时统计等

### 如何实现一个中间件？

在 Gin 中，中间件必须是 `gin.HandlerFunc` 类型，比如下面的代码就是一个中间件

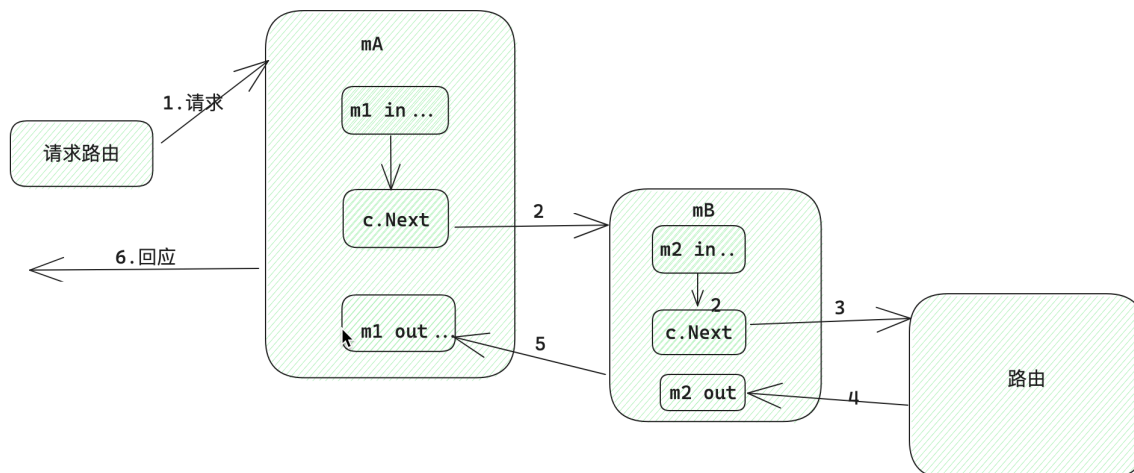
```
1 func index(c *gin.Context) { // 简单地说：只要函数参数是 *gin.Context，即可视为一个中间件
2     c.JSON(http.StatusOK, gin.H{
3         "msg": "ok",
4     })
5 }
```

## 中间件的调用逻辑？

比如：

```
1 func middleWareA(c *gin.Context) {  
2     fmt.Println("a in...")  
3     c.Next()  
4     fmt.Println("a out...")  
5 }  
6  
7 func middleWareB(c *gin.Context) {  
8     fmt.Println("b in ...")  
9     c.Next()  
10    fmt.Println("b out ...")  
11 }  
12 // 调用这些中间件  
13 c.Use(A,B)  
14 // 路由  
15 c.GET("/middle"),middle)
```

在上面的代码中，用户访问 `/middle` 路由时，gin 会首先访问中间件 A，B。整个执行顺序为：



其中 `c.Next()` 是指 调用后续的处理函数

与之相对，`c.Abort()` 是指阻止调用后续的处理函数，即执行到此处时，不再调用后续的逻辑处理，直接执行下一行代码。

如何在不同的中间件中传递参数？

```
c.Set("key", "value")
```

```
c.Get("key")
```

 也可以是 

```
c.MustGet("key")
```

如何注册中间件？

1. 在单个路由中注册中间件：

```
1 // 给/mid路由单独注册中间件 StatCost() (可注册多个)
2 r.GET("/mid", StatCost(), func(c *gin.Context) {
3     name := c.MustGet("name").(string) // 从上下文取值
4     log.Println(name)
5     c.JSON(http.StatusOK, gin.H{
6         "message": "Hello world!",
7     })
8 })
```

2. 在路由组中注册中间件：

```
1 // 1.
2 shopGroup := r.Group("/shop", StatCost())
3 {
4     shopGroup.GET("/index", func(c *gin.Context) {...})
5     ...
6 }
7 // 2.
8 shopGroup := r.Group("/shop")
9 shopGroup.Use(StatCost())
10 {
11     shopGroup.GET("/index", func(c *gin.Context) {...})
12     ...
13 }
```

**Gin 其他注意事项：**

## Gin 默认中间件

gin.Default() 默认使用了 Logger 和 Recovery 中间件，其中：

- Logger 中间件将日志写入 gin.DefaultWriter，即使配置了 GIN\_MODE=release。
- Recovery 中间件会 recover 任何 panic。如果有 panic 的话，会写入500响应码。

如果不想使用上面两个默认的中间件，可以使用 `gin.New()` 新建一个没有任何默认中间件的路由。

## Gin 中间件中使用goroutine

当在中间件或handler中启动新的goroutine时，**不能使用**原始的上下文（`c *gin.Context`），必须使用其只读副本（`c.Copy()`）。