Gin 框架路由拆分与注册

Go语言中文网 3天前

以下文章来源于李文周，作者李文周

李文周
一个不知名JPG程序员的自我更新之旅。

基本的路由注册

下面最基础的gin路由注册方式，适用于路由比较少的简单项目或者项目demo。

```
package main

import (
        "net/http"

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

func helloHandler(c *gin.Context) {
        c.JSON(http.StatusOK, gin.H{
                "message": "Hello q1mi!",
        })
}

func main() {
        r := gin.Default()
        r.GET("/hello", helloHandler)
        if err := r.Run(); err != nil {
                fmt.Println("startup service failed, err:%v\n", err)
        }
}
```

路由拆分成单独文件或包

当项目的规模增大后就不太适合继续在项目的 `main.go` 文件中去实现路由注册相关逻辑了，我们会倾向于把路由部分的代码都拆分出来，形成一个单独的文件或包：

我们在 `routers.go` 文件中定义并注册路由信息：

```
package main

import (
        "net/http"

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

func helloHandler(c *gin.Context) {
        c.JSON(http.StatusOK, gin.H{
                "message": "Hello q1mi!",
        })
}

func setupRouter() *gin.Engine {
        r := gin.Default()
        r.GET("/hello", helloHandler)
        return r
}
```

此时 `main.go` 中调用上面定义好的 `setupRouter` 函数：

```
func main() {
        r := setupRouter()
        if err := r.Run(); err != nil {
                fmt.Println("startup service failed, err:%v\n", err)
        }
}
```

此时的目录结构：

```
gin_demo
├── go.mod
├── go.sum
```

```
├── main.go
└── routers.go
```

把路由部分的代码单独拆分成包的话也是可以的，拆分后的目录结构如下：

```
gin_demo
├── go.mod
├── go.sum
├── main.go
└── routers
    └── routers.go
```

`routers/routers.go` 需要注意此时 `setupRouter` 需要改成首字母大写：

```go
package routers

import (
	"net/http"

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

func helloHandler(c *gin.Context) {
	c.JSON(http.StatusOK, gin.H{
		"message": "Hello q1mi!",
	})
}

// SetupRouter 配置路由信息
func SetupRouter() *gin.Engine {
	r := gin.Default()
	r.GET("/hello", helloHandler)
	return r
}
```

`main.go` 文件内容如下：

```go
package main
```

```
2
3  import (
4          "fmt"
5          "gin_demo/routers"
6  )
7
8  func main() {
9          r := routers.SetupRouter()
10         if err := r.Run(); err != nil {
11                 fmt.Println("startup service failed, err:%v\n", err)
12         }
13 }
```

路由拆分成多个文件

当我们的业务规模继续膨胀，单独的一个 `routers` 文件或包已经满足不了我们的需求了，

```
1  func SetupRouter() *gin.Engine {
2          r := gin.Default()
3          r.GET("/hello", helloHandler)
4    r.GET("/xx1", xxHandler1)
5    ...
6    r.GET("/xx30", xxHandler30)
7          return r
8  }
```

因为我们把所有的路由注册都写在一个 `SetupRouter` 函数中的话就会太复杂了。

我们可以分开定义多个路由文件，例如：

```
1  gin_demo
2  ├── go.mod
3  ├── go.sum
4  ├── main.go
5  └── routers
6      ├── blog.go
7      └── shop.go
```

`routers/shop.go` 中添加一个 `LoadShop` 的函数，将shop相关的路由注册到指定的路由器：

```
func LoadShop(e *gin.Engine)  {
        e.GET("/hello", helloHandler)
  e.GET("/goods", goodsHandler)
  e.GET("/checkout", checkoutHandler)
  ...
}
```

`routers/blog.go` 中添加一个`LoadBlog的函数，将blog相关的路由注册到指定的路由器：

```
func LoadBlog(e *gin.Engine) {
        e.GET("/post", postHandler)
  e.GET("/comment", commentHandler)
  ...
}
```

在main函数中实现最终的注册逻辑如下：

```
func main() {
        r := gin.Default()
        routers.LoadBlog(r)
        routers.LoadShop(r)
        if err := r.Run(); err != nil {
                fmt.Println("startup service failed, err:%v\n", err)
        }
}
```

路由拆分到不同的APP

有时候项目规模实在太大，那么我们就更倾向于把业务拆分的更详细一些，例如把不同的业务代码拆分成不同的APP。

因此我们在项目目录下单独定义一个 `app` 目录，用来存放我们不同业务线的代码文件，这样就很容易进行横向扩展。大致目录结构如下：

```
gin_demo
├── app
│   ├── blog
│   │   ├── handler.go
```

```
5  |   |   └── router.go
6  |   └── shop
7  |       ├── handler.go
8  |       └── router.go
9  ├── go.mod
10 ├── go.sum
11 ├── main.go
12 └── routers
13     └── routers.go
```

其中 `app/blog/router.go` 用来定义post相关路由信息，具体内容如下：

```
1 func Routers(e *gin.Engine) {
2         e.GET("/post", postHandler)
3         e.GET("/comment", commentHandler)
4 }
```

`app/shop/router.go` 用来定义shop相关路由信息，具体内容如下：

```
1 func Routers(e *gin.Engine) {
2         e.GET("/goods", goodsHandler)
3         e.GET("/checkout", checkoutHandler)
4 }
```

`routers/routers.go` 中根据需要定义 `Include` 函数用来注册子app中定义的路由， `Init` 函数用来进行路由的初始化操作：

```
1  type Option func(*gin.Engine)
2
3  var options = []Option{}
4
5  // 注册app的路由配置
6  func Include(opts ...Option) {
7          options = append(options, opts...)
8  }
9
10 // 初始化
11 func Init() *gin.Engine {
```

```
        r := gin.New()
        for _, opt := range options {
                opt(r)
        }
        return r
}
```

main.go 中按如下方式先注册子app中的路由，然后再进行路由的初始化：

```
func main() {
        // 加载多个APP的路由配置
        routers.Include(shop.Routers, blog.Routers)
        // 初始化路由
        r := routers.Init()
        if err := r.Run(); err != nil {
                fmt.Println("startup service failed, err:%v\n", err)
        }
}
```

总结

gin 框架是一个非常容易扩展的web框架，本文是我在日常编码中总结的一点点经验，因为世界上不可能有完全相同的项目，每个人也都有自己的编程习惯，关于gin框架路由拆分与注册的更多方式我就在此抛砖引玉了。

推荐阅读

- 最流行的 Web 框架 Gin 源码阅读

阅读原文