链滴

Gin 快速预览《老铁版》

作者：someone27889

原文链接：https://ld246.com/article/1584180344311

来源网站：链滴

1.大家好今天带大家快速了解一下 Gin 框架

2.Gin 是什么

Gin 是 基于 Go 语言的一个 HttpServer 函数库。俗称 MVC 框架?

3.为什么使用 Gin

因为想用，所以就用!

4.如何使用 Gin 呢

首先 引入 github.com/gin-gonic/gin !

然后 在 main 函数中写入!

```
r := gin.Default()
    r.GET("/ping", func(context *gin.Context) {
        context.JSON(200, gin.H{
            "message": "Pong",
        })
    })
r.Run()
```

最后 go run main.go!

这样 Gin 框架就会开启 Web 服务器了!

Gin 框架默认端口是 8080!

5.如何测试是否启动

首先 打开我们的 谷歌 浏览器!

然后找到 地址栏!

然后输入 http://localhost:8080/ping!

然后查看页面是否不是 HTTP404!

最后查看页面是否显示 PONG!

这样我们就完成最简单的 Gin 框架使用和测试了!

6.功能 Example

首先 ~~我们放入葱花香菜汪油然后，然后放入一头羊~~

首先 我们直接出锅装盘了

a.HTML渲染

```
package main

import (
    "fmt"
    "github.com/gin-gonic/gin"
    "html/template"
    "net/http"
    "time"
)

func main() {
        // engine.Use(Logger(), Recovery()) 常规模式 使用了 Logger和Recovery
    r := gin.Default()
      // 常规get请求,返回JSON+Map[string]Interface{}类型
    r.GET("/ping", func(context *gin.Context) {
        context.JSON(200, gin.H{
            "message": "Pong",
        })
    })
      // 把返回内容 转换成 ascii 编码
    r.GET("/asciiJson", func(context *gin.Context) {
        mps := make(map[string]string)
        mps["location"] = "北京"
        mps["path"] = "哈哈哈"
        context.AsciiJSON(200, mps)
    })
      // 模版渲染 使用变量时 自定义的符号
    r.Delims("{[{", "}]}")
      // 模版 pipe,使用方式为 xxx | formatDate
    r.SetFuncMap(template.FuncMap{
        "formatDate": formatAsDate,
    })
      // 加载模版文件
    r.LoadHTMLGlob("mygin/templates/*")
      // 返回 mygin/template/user.tmpl 模版，并使用 Map数据渲染它
    r.GET("/tmpl/user", func(context *gin.Context) {
        context.HTML(http.StatusOK, "user.tmpl", gin.H{
            "name": "ferried",
            "age":  18,
            "desc": "花美男",
        })
    })
    r.GET("/temp/func", func(context *gin.Context) {
        context.HTML(http.StatusOK, "func.tmpl", gin.H{
            "now": time.Date(2017, 07, 01, 0, 0, 0, 0, time.UTC),
        })
    })

    r.Run()
}

func formatAsDate(t time.Time) string {
```

```
    year, month, day := t.Date()
    return fmt.Sprintf("%d/%02d/%02d", year, month, day)
}
```

b.HTTP2 server 推送 / 静态资源解析

```
package main

import (
    "html/template"
    "log"

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

// 使用 模版变量,并且起名 "https"
var html = template.Must(template.New("https").Parse(`
<html>
<head>
  <title>Https Test</title>
  <script src="/assets/app.js"></script>
</head>
<body>
  <h1 style="color:red;">Welcome, Ginner!</h1>
</body>
</html>
`))

func main() {
    r := gin.Default()
    // 设置静态资源目录
    r.Static("/assets", "./assets")
    // 设置HTML模版,
    r.SetHTMLTemplate(html)
    // 当服务器请求/的时候
    r.GET("/", func(c *gin.Context) {
        // 直接把 静态资源 app.js 推给客户端，不让客户端在req->res了，http2的 Pusher
        if pusher := c.Writer.Pusher(); pusher != nil {
            // 使用 pusher.Push() 做服务器推送
            if err := pusher.Push("/assets/app.js", nil); err != nil {
                log.Printf("Failed to push: %v", err)
            }
        }
        // 使用name https模版来返回页面
        c.HTML(200, "https", gin.H{
            "status": "success",
        })
    })

    // 监听并在 https://127.0.0.1:8080 上启动服务,后面两个是 Http2/https需要的证书文件
    r.RunTLS(":8080", "./testdata/server.pem", "./testdata/server.key")
}
```

c.Stream响应

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/png", func(context *gin.Context) {
        // 拿到图片response
        response, _ := http.Get("http://a3.att.hudong.com/68/61/300000839764127060614318
8_950.jpg")
        // body是二进制类型
        reader := response.Body
        // 设置请求头
        contentLength := response.ContentLength
        contentType := response.Header.Get("Content-Type")
        // 设置请求头
        extraHeaders := map[string]string{
            "Content-Disposition": `attachment; filename="gopher.png"`,
        }
        // 响应输出流,将文件进行下载
        context.DataFromReader(http.StatusOK, contentLength, contentType, reader, extraHeade
s)

    })
    r.Run()
}
```

d.参数验证

```
package main

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

// 结构体，binding: required必须填写
type UserForm struct {
    UserName string `form:"username" binding:"required"`
    Password string `form:"password" binding:"required"`
}

func main() {
    r := gin.Default()
    r.GET("/form", func(context *gin.Context) {
        // 指针结构体
        userParam := &UserForm{}
        // 验证参数是否有填写
```

```
        if context.ShouldBind(userParam) == nil {
            // 模拟登陆成功
            if userParam.UserName == "user" && userParam.Password == "password" {
                context.JSON(200, gin.H{"status": "you are logged in"})
            } else {
                context.JSON(401, gin.H{"status": "unauthorized"})
            }
        }
    })
    r.Run()
}
```

e.取参数方式

```
    r.GET("/form", func(context *gin.Context) {
        // ?key=1
        query := context.Query("key")
        // post ?key=1
        value := context.PostForm("key")
        // 特殊编码 html
        // 单个文件
        file, _ := context.FormFile("file")
                // 表单获取
        form, _ := context.MultipartForm()
                // 获取文件集合
        files := form.File["upload[]"]

        context.PureJSON(http.StatusOK, gin.H{
            query: value,
        })
    })
```

f.防止json劫持

```
func main() {
    r := gin.Default()

    // 你也可以使用自己的 SecureJSON 前缀
    // r.SecureJsonPrefix(")]}',\n")

    r.GET("/someJSON", func(c *gin.Context) {
        names := []string{"lena", "austin", "foo"}

        // 将输出：  while(1);["lena","austin","foo"]
        c.SecureJSON(http.StatusOK, names)
    })

    // 监听并在 0.0.0.0:8080 上启动服务
    r.Run(":8080")
}
```

g.所有HTTP

```
router.GET("/someGet", getting)
    router.POST("/somePost", posting)
    router.PUT("/somePut", putting)
    router.DELETE("/someDelete", deleting)
    router.PATCH("/somePatch", patching)
    router.HEAD("/someHead", head)
    router.OPTIONS("/someOptions", options)
```

结束

点赞加主编关注，如果积够 10 个赞那么我们下期 做 《XORM》营销号版，老铁双击点赞，关注，66。没毛病！