目　录

致谢

当前文档 《go 语言框架 gin中文文档》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-03-19。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址: http://www.bookstack.cn/books/gin-doc

书栈官网: http://www.bookstack.cn

书栈开源: https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

说明

当前文档转自 https://github.com/ningskyer/gin-doc-cn

但是小编快速阅读了下，发现当前文档并不完善，所以建议大家关注源链接，即-> https://github.com/ningskyer/gin-doc-cn

安装与配置

安装与配置

安装:

```
1.  $ go get gopkg.in/gin-gonic/gin.v1
```

注意：确保 GOPATH GOROOT 已经配置

导入:

```
1.  import "gopkg.in/gin-gonic/gin.v1"
```

框架架构

框架架构

- HTTP 服务器

1.默认服务器

```
router.Run()
```

2.HTTP 服务器

除了默认服务器中 `router.Run()` 的方式外，还可以用 `http.ListenAndServe()`，比如

```
func main() {
    router := gin.Default()
    http.ListenAndServe(":8080", router)
}
```

或者自定义 HTTP 服务器的配置：

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

    s := &http.Server{
        Addr:           ":8080",
        Handler:        router,
        ReadTimeout:    10 * time.Second,
        WriteTimeout:   10 * time.Second,
        MaxHeaderBytes: 1 << 20,
    }
    s.ListenAndServe()
}
```

3.HTTP 服务器替换方案

想无缝重启、停机吗？ 以下有几种方式：

我们可以使用 fvbock/endless 来替换默认的 `ListenAndServe` 。但是 windows 不能使用。

```
router := gin.Default()
router.GET("/", handler)
// [...]
endless.ListenAndServe(":4242", router)
```

除了 endless 还可以用manners:

manners 兼容windows

```
manners.ListenAndServe(":8888", r)
```

- 生命周期
- Context

路由

路由

- 基本路由

  gin 框架中采用的路由库是 httprouter。

```
    // 创建带有默认中间件的路由:
    // 日志与恢复中间件
    router := gin.Default()
    //创建不带中间件的路由：
    //r := gin.New()

    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)
```

- 路由参数

api 参数通过Context的Param方法来获取

```
router.GET("/string/:name", func(c *gin.Context) {
        name := c.Param("name")
        fmt.Println("Hello %s", name)
    })
```

URL 参数通过 DefaultQuery 或 Query 方法获取

```
// url 为 http://localhost:8080/welcome?name=ningskyer时
// 输出 Hello ningskyer
// url 为 http://localhost:8080/welcome时
```

```
// 输出 Hello Guest
router.GET("/welcome", func(c *gin.Context) {
    name := c.DefaultQuery("name", "Guest") //可设置默认值
    // 是 c.Request.URL.Query().Get("lastname") 的简写
    lastname := c.Query("lastname")
    fmt.Println("Hello %s", name)
})
```

表单参数通过 PostForm 方法获取

```
//form
router.POST("/form", func(c *gin.Context) {
    type := c.DefaultPostForm("type", "alert")//可设置默认值
    msg := c.PostForm("msg")
    title := c.PostForm("title")
    fmt.Println("type is %s, msg is %s, title is %s", type, msg, title)
})
```

- 路由群组

```
    someGroup := router.Group("/someGroup")
    {
        someGroup.GET("/someGet", getting)
        someGroup.POST("/somePost", posting)
    }
```

控制器

控制器

- 数据解析绑定

模型绑定可以将请求体绑定给一个类型，目前支持绑定的类型有 JSON, XML 和标准表单数据 (foo=bar&boo=baz)。
要注意的是绑定时需要给字段设置绑定类型的标签。比如绑定 JSON 数据时，设置 `json:"fieldname"` 。
使用绑定方法时，Gin 会根据请求头中 Content-Type 来自动判断需要解析的类型。如果你明确绑定的类型，你可以不用自动推断，而用 BindWith 方法。
你也可以指定某字段是必需的。如果一个字段被 `binding:"required"` 修饰而值却是空的，请求会失败并返回错误。

```
1.  // Binding from JSON
2.  type Login struct {
3.      User     string `form:"user" json:"user" binding:"required"`
4.      Password string `form:"password" json:"password"
    binding:"required"`
5.  }
6.
7.  func main() {
8.      router := gin.Default()
9.
10.     // 绑定JSON的例子 ({"user": "manu", "password": "123"})
11.     router.POST("/loginJSON", func(c *gin.Context) {
12.         var json Login
13.
14.         if c.BindJSON(&json) == nil {
15.             if json.User == "manu" && json.Password == "123" {
16.                 c.JSON(http.StatusOK, gin.H{"status": "you are
```

```
    logged in"})
17.             } else {
18.                 c.JSON(http.StatusUnauthorized, gin.H{"status":
    "unauthorized"})
19.             }
20.         }
21.     })
22.
23.     // 绑定普通表单的例子 (user=manu&password=123)
24.     router.POST("/loginForm", func(c *gin.Context) {
25.         var form Login
26.         // 根据请求头中 content-type 自动推断.
27.         if c.Bind(&form) == nil {
28.             if form.User == "manu" && form.Password == "123" {
29.                 c.JSON(http.StatusOK, gin.H{"status": "you are
    logged in"})
30.             } else {
31.                 c.JSON(http.StatusUnauthorized, gin.H{"status":
    "unauthorized"})
32.             }
33.         }
34.     })
35.     // 绑定多媒体表单的例子 (user=manu&password=123)
36.     router.POST("/login", func(c *gin.Context) {
37.         var form LoginForm
38.         // 你可以显式声明来绑定多媒体表单：
39.         // c.BindWith(&form, binding.Form)
40.         // 或者使用自动推断:
41.         if c.Bind(&form) == nil {
42.             if form.User == "user" && form.Password == "password" {
43.                 c.JSON(200, gin.H{"status": "you are logged in"})
44.             } else {
45.                 c.JSON(401, gin.H{"status": "unauthorized"})
46.             }
47.         }
48.     })
49.     // Listen and serve on 0.0.0.0:8080
50.     router.Run(":8080")
```

```
51.  }
```

请求

请求

- 请求头
- 请求参数
- Cookies
- 上传文件

```
router.POST("/upload", func(c *gin.Context) {

    file, header , err := c.Request.FormFile("upload")
    filename := header.Filename
    fmt.Println(header.Filename)
    out, err := os.Create("./tmp/"+filename+".png")
    if err != nil {
        log.Fatal(err)
    }
    defer out.Close()
    _, err = io.Copy(out, file)
    if err != nil {
        log.Fatal(err)
    }
})
```

响应

响应

- 响应头
- 附加Cookie
- 字符串响应

```
c.String(http.StatusOK, "some string")
```

- JSON/XML/YAML响应

```
r.GET("/moreJSON", func(c *gin.Context) {
    // You also can use a struct
    var msg struct {
        Name    string `json:"user" xml:"user"`
        Message string
        Number  int
    }
    msg.Name = "Lena"
    msg.Message = "hey"
    msg.Number = 123
    // 注意 msg.Name 变成了 "user" 字段
    // 以下方式都会输出 :   {"user": "Lena", "Message": "hey", "Number": 123}
    c.JSON(http.StatusOK, gin.H{"user": "Lena", "Message": "hey", "Number": 123})
    c.XML(http.StatusOK, gin.H{"user": "Lena", "Message": "hey", "Number": 123})
    c.YAML(http.StatusOK, gin.H{"user": "Lena", "Message": "hey", "Number": 123})
    c.JSON(http.StatusOK, msg)
    c.XML(http.StatusOK, msg)
```

```
18.     c.YAML(http.StatusOK, msg)
19. })
```

- 视图响应

先要使用 LoadHTMLTemplates() 方法来加载模板文件

```
1.  func main() {
2.      router := gin.Default()
3.      //加载模板
4.      router.LoadHTMLGlob("templates/*")
5.      //router.LoadHTMLFiles("templates/template1.html",
    "templates/template2.html")
6.      //定义路由
7.      router.GET("/index", func(c *gin.Context) {
8.          //根据完整文件名渲染模板，并传递参数
9.          c.HTML(http.StatusOK, "index.tmpl", gin.H{
10.             "title": "Main website",
11.         })
12.     })
13.     router.Run(":8080")
14. }
```

模板结构定义

```
1.  <html>
2.      <h1>
3.          {{ .title }}
4.      </h1>
5.  </html>
```

不同文件夹下模板名字可以相同，此时需要 LoadHTMLGlob() 加载两层模板路径

```
1.  router.LoadHTMLGlob("templates/**/*")
2.  router.GET("/posts/index", func(c *gin.Context) {
```

```
    c.HTML(http.StatusOK, "posts/index.tmpl", gin.H{
        "title": "Posts",
    })
    c.HTML(http.StatusOK, "users/index.tmpl", gin.H{
        "title": "Users",
    })

}
```

templates/posts/index.tmpl

```
<!-- 注意开头 define 与结尾 end 不可少 -->
{{ define "posts/index.tmpl" }}
<html><h1>
    {{ .title }}
</h1>
</html>
{{ end }}

gin也可以使用自定义的模板引擎，如下

```go
import "html/template"

func main() {
    router := gin.Default()
    html := template.Must(template.ParseFiles("file1", "file2"))
    router.SetHTMLTemplate(html)
    router.Run(":8080")
}
```

- 文件响应

```
//获取当前文件的相对路径
router.Static("/assets", "./assets")
//
router.StaticFS("/more_static", http.Dir("my_file_system"))
```
```

```
5.  //获取相对路径下的文件
6.  router.StaticFile("/favicon.ico", "./resources/favicon.ico")
```

- 重定向

```
1.  r.GET("/redirect", func(c *gin.Context) {
2.      //支持内部和外部的重定向
3.      c.Redirect(http.StatusMovedPermanently,
    "http://www.baidu.com/")
4.  })
```

- 同步异步

goroutine 机制可以方便地实现异步处理

```
1.  func main() {
2.      r := gin.Default()
3.      //1. 异步
4.      r.GET("/long_async", func(c *gin.Context) {
5.          // goroutine 中只能使用只读的上下文 c.Copy()
6.          cCp := c.Copy()
7.          go func() {
8.              time.Sleep(5 * time.Second)
9.
10.             // 注意使用只读上下文
11.             log.Println("Done! in path " + cCp.Request.URL.Path)
12.         }()
13.     })
14.     //2. 同步
15.     r.GET("/long_sync", func(c *gin.Context) {
16.         time.Sleep(5 * time.Second)
17.
18.         // 注意可以使用原始上下文
19.         log.Println("Done! in path " + c.Request.URL.Path)
20.     })
21.
22.     // Listen and serve on 0.0.0.0:8080
```

```
23.     r.Run(":8080")
24. }
```

中间件

中间件

- 分类使用方式
  ```
  // 1.全局中间件
  router.Use(gin.Logger())
  router.Use(gin.Recovery())
```

// 2.单路由的中间件，可以加任意多个
router.GET(“/benchmark”, MyMiddelware(),
benchEndpoint)

// 3.群组路由的中间件
authorized := router.Group(“/“, MyMiddelware())
// 或者这样用:
authorized := router.Group(“/“)
authorized.Use(MyMiddelware())
{
authorized.POST(“/login”, loginEndpoint)
}

```
1. <a name="middleware-create"></a>
2.
3. - 自定义中间件
4.
5. ```go
6. //定义
7. func Logger() gin.HandlerFunc {
8.     return func(c *gin.Context) {
```

```
        t := time.Now()

        // 在gin上下文中定义变量
        c.Set("example", "12345")

        // 请求前

        c.Next()//处理请求

        // 请求后
        latency := time.Since(t)
        log.Print(latency)

        // access the status we are sending
        status := c.Writer.Status()
        log.Println(status)
    }
}
//使用
func main() {
    r := gin.New()
    r.Use(Logger())

    r.GET("/test", func(c *gin.Context) {
        //获取gin上下文中的变量
        example := c.MustGet("example").(string)

        // 会打印: "12345"
        log.Println(example)
    })

    // 监听运行于 0.0.0.0:8080
    r.Run(":8080")
}
```

- 中间件参数
- 内置中间件

1.简单认证BasicAuth

```
1.  // 模拟私有数据
2.  var secrets = gin.H{
3.      "foo":    gin.H{"email": "foo@bar.com", "phone": "123433"},
4.      "austin": gin.H{"email": "austin@example.com", "phone": "666"},
5.      "lena":   gin.H{"email": "lena@guapa.com", "phone": "523443"},
6.  }
7.
8.  func main() {
9.      r := gin.Default()
10.
11.     // 使用 gin.BasicAuth 中间件, 设置授权用户
12.     authorized := r.Group("/admin", gin.BasicAuth(gin.Accounts{
13.         "foo":    "bar",
14.         "austin": "1234",
15.         "lena":   "hello2",
16.         "manu":   "4321",
17.     }))
18.
19.     // 定义路由
20.     authorized.GET("/secrets", func(c *gin.Context) {
21.         // 获取提交的用户名（AuthUserKey）
22.         user := c.MustGet(gin.AuthUserKey).(string)
23.         if secret, ok := secrets[user]; ok {
24.             c.JSON(http.StatusOK, gin.H{"user": user, "secret":
    secret})
25.         } else {
26.             c.JSON(http.StatusOK, gin.H{"user": user, "secret": "NO
    SECRET :("})
27.         }
28.     })
29.
30.     // Listen and serve on 0.0.0.0:8080
31.     r.Run(":8080")
32. }
```

数据库

数据库

- Mongodb

Golang常用的Mongodb驱动为 mgo.v2, 查看文档

mgo 使用方式如下:

```
1.  //定义 Person 结构，字段须为首字母大写
2.  type Person struct {
3.      Name string
4.      Phone string
5.  }
6.
7.  router.GET("/mongo", func(context *gin.Context){
8.      //可本地可远程，不指定协议时默认为http协议访问，此时需要设置 mongodb 的
    nohttpinterface=false来打开httpinterface。
9.      //也可以指定mongodb协议，如 "mongodb://127.0.0.1:27017"
10.     var MOGODB_URI = "127.0.0.1:27017"
11.     //连接
12.     session, err := mgo.Dial(MOGODB_URI)
13.     //连接失败时终止
14.     if err != nil {
15.         panic(err)
16.     }
17.     //延迟关闭，释放资源
18.     defer session.Close()
19.     //设置模式
20.     session.SetMode(mgo.Monotonic, true)
21.     //选择数据库与集合
22.     c := session.DB("adatabase").C("acollection")
23.     //插入文档
24.     err = c.Insert(&Person{Name:"Ale", Phone:"+55 53 8116 9639"},
25.                 &Person{Name:"Cla",  Phone:"+55 53 8402 8510"})
```

```
26.     //出错判断
27.     if err != nil {
28.             log.Fatal(err)
29.     }
30.     //查询文档
31.     result := Person{}
32.     //注意mongodb存储后的字段大小写问题
33.     err = c.Find(bson.M{"name": "Ale"}).One(&result)
34.     //出错判断
35.     if err != nil {
36.         log.Fatal(err)
37.     }
38.     fmt.Println("Phone:", result.Phone)
39. })
```

- Mysql
- ORM