文档

1. 源码：https://github.com/gin-gonic/gin
2. 跨域设置：https://github.com/gin-contrib/cors
3. jwt 源码：https://github.com/golang-jwt/jwt

启动http服务器

http Server

```
package main

import (
  "net/http"

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

func main() {
  r := gin.Default()
  r.GET("/ping", func(c *gin.Context) {
    c.JSON(http.StatusOK, gin.H{
      "message": "pong",
    })
  })
  r.Run() // listen and serve on 0.0.0.0:8080 (for windows "localhost:8080")
}
```

RESTful API 设计

Method

```
func main() {
  // Creates a gin router with default middleware:
  // logger and recovery (crash-free) middleware
  router := gin.Default()

  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)

  // By default it serves on :8080 unless a
  // PORT environment variable was defined.
  router.Run()
  // router.Run(":3000") for a hard coded port
```

```
}
```

参数传递及获取

路径携带参数

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

  // This handler will match /user/john but will not match /user/ or /user
  router.GET("/user/:name", func(c *gin.Context) {
    name := c.Param("name")
    c.String(http.StatusOK, "Hello %s", name)
  })

  // However, this one will match /user/john/ and also /user/john/send
  // If no other routers match /user/john, it will redirect to /user/john/
  router.GET("/user/:name/*action", func(c *gin.Context) {
    name := c.Param("name")
    action := c.Param("action")
    message := name + " is " + action
    c.String(http.StatusOK, message)
  })

  // For each matched request Context will hold the route definition
  router.POST("/user/:name/*action", func(c *gin.Context) {
    b := c.FullPath() == "/user/:name/*action" // true
    c.String(http.StatusOK, "%t", b)
  })

  // This handler will add a new router for /user/groups.
  // Exact routes are resolved before param routes, regardless of the order they
were defined.
  // Routes starting with /user/groups are never interpreted as /user/:name/...
routes
  router.GET("/user/groups", func(c *gin.Context) {
    c.String(http.StatusOK, "The available groups are [...]")
  })

  router.Run(":8080")
}
```

Querystring 参数

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

  // Query string parameters are parsed using the existing underlying request
object.
  // The request responds to a url matching:  /welcome?
firstname=Jane&lastname=Doe
  router.GET("/welcome", func(c *gin.Context) {
    firstname := c.DefaultQuery("firstname", "Guest")
    lastname := c.Query("lastname") // shortcut for
c.Request.URL.Query().Get("lastname")

    c.String(http.StatusOK, "Hello %s %s", firstname, lastname)
  })
  router.Run(":8080")
}
```

Form表单（Multipart/Urlencoded ）

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

  router.POST("/form_post", func(c *gin.Context) {
    message := c.PostForm("message")
    nick := c.DefaultPostForm("nick", "anonymous")

    c.JSON(http.StatusOK, gin.H{
      "status":  "posted",
      "message": message,
      "nick":    nick,
    })
  })
  router.Run(":8080")
}
```

query + post form

```
POST /post?id=1234&page=1 HTTP/1.1
Content-Type: application/x-www-form-urlencoded

name=manu&message=this_is_great
```

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

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

    id := c.Query("id")
    page := c.DefaultQuery("page", "0")
    name := c.PostForm("name")
```

```
    message := c.PostForm("message")

    fmt.Printf("id: %s; page: %s; name: %s; message: %s", id, page, name,
message)
  })
  router.Run(":8080")
}
```

map querystring or postform

```
POST /post?ids[a]=1234&ids[b]=hello HTTP/1.1
Content-Type: application/x-www-form-urlencoded

names[first]=thinkerou&names[second]=tianou
```

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

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

    ids := c.QueryMap("ids")
    names := c.PostFormMap("names")

    fmt.Printf("ids: %v; names: %v", ids, names)
  })
  router.Run(":8080")
}
```

文件上传

单个文件上传

```
func main() {
  router := gin.Default()
  // Set a lower memory limit for multipart forms (default is 32 MiB)
  router.MaxMultipartMemory = 8 << 20  // 8 MiB
  router.POST("/upload", func(c *gin.Context) {
    // Single file
    file, _ := c.FormFile("file")
    log.Println(file.Filename)

    // Upload the file to specific dst.
    c.SaveUploadedFile(file, dst)

    c.String(http.StatusOK, fmt.Sprintf("'%s' uploaded!", file.Filename))
  })
  router.Run(":8080")
}
```

多个文件上传

```
func main() {
  router := gin.Default()
  // Set a lower memory limit for multipart forms (default is 32 MiB)
  router.MaxMultipartMemory = 8 << 20  // 8 MiB
  router.POST("/upload", func(c *gin.Context) {
    // Multipart form
    form, _ := c.MultipartForm()
    files := form.File["upload[]"]

    for _, file := range files {
      log.Println(file.Filename)

      // Upload the file to specific dst.
      c.SaveUploadedFile(file, dst)
    }
    c.String(http.StatusOK, fmt.Sprintf("%d files uploaded!", len(files)))
  })
  router.Run(":8080")
}
```

路由

路由分组

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

  // Simple group: v1
  v1 := router.Group("/v1")
  {
    v1.POST("/login", loginEndpoint)
    v1.POST("/submit", submitEndpoint)
    v1.POST("/read", readEndpoint)
  }

  // Simple group: v2
  v2 := router.Group("/v2")
  {
    v2.POST("/login", loginEndpoint)
    v2.POST("/submit", submitEndpoint)
    v2.POST("/read", readEndpoint)
  }

  router.Run(":8080")
}
```

中间件定义及使用

中间件的使用

```
func main() {
  // Creates a router without any middleware by default
  r := gin.New()

  // Global middleware
  // Logger middleware will write the logs to gin.DefaultWriter even if you set
with GIN_MODE=release.
  // By default gin.DefaultWriter = os.Stdout
  r.Use(gin.Logger())

  // Recovery middleware recovers from any panics and writes a 500 if there was
one.
  r.Use(gin.Recovery())

  // Per route middleware, you can add as many as you desire.
  r.GET("/benchmark", MyBenchLogger(), benchEndpoint)

  // Authorization group
  // authorized := r.Group("/", AuthRequired())
  // exactly the same as:
  authorized := r.Group("/")
  // per group middleware! in this case we use the custom created
  // AuthRequired() middleware just in the "authorized" group.
  authorized.Use(AuthRequired())
  {
    authorized.POST("/login", loginEndpoint)
    authorized.POST("/submit", submitEndpoint)
    authorized.POST("/read", readEndpoint)

    // nested group
    testing := authorized.Group("testing")
    // visit 0.0.0.0:8080/testing/analytics
    testing.GET("/analytics", analyticsEndpoint)
  }

  // Listen and serve on 0.0.0.0:8080
  r.Run(":8080")
}
```

自定义recover中间件

```
func main() {
  // Creates a router without any middleware by default
  r := gin.New()

  // Global middleware
  // Logger middleware will write the logs to gin.DefaultWriter even if you set
with GIN_MODE=release.
  // By default gin.DefaultWriter = os.Stdout
  r.Use(gin.Logger())

  // Recovery middleware recovers from any panics and writes a 500 if there was
one.
  r.Use(gin.CustomRecovery(func(c *gin.Context, recovered interface{}) {
    if err, ok := recovered.(string); ok {
```

```
      c.String(http.StatusInternalServerError, fmt.Sprintf("error: %s", err))
    }
    c.AbortWithStatus(http.StatusInternalServerError)
  }))

  r.GET("/panic", func(c *gin.Context) {
    // panic with a string -- the custom middleware could save this to a database
or report it to the user
    panic("foo")
  })

  r.GET("/", func(c *gin.Context) {
    c.String(http.StatusOK, "ohai")
  })

  // Listen and serve on 0.0.0.0:8080
  r.Run(":8080")
}
```

日志

日志写入到文件

```
func main() {
    // Disable Console Color, you don't need console color when writing the logs
to file.
    gin.DisableConsoleColor()

    // Logging to a file.
    f, _ := os.Create("gin.log")
    gin.DefaultWriter = io.MultiWriter(f)

    // Use the following code if you need to write the logs to file and console
at the same time.
    // gin.DefaultWriter = io.MultiWriter(f, os.Stdout)

    router := gin.Default()
    router.GET("/ping", func(c *gin.Context) {
        c.String(http.StatusOK, "pong")
    })

    router.Run(":8080")
}
```

自定义日志格式

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

  // LoggerWithFormatter middleware will write the logs to gin.DefaultWriter
  // By default gin.DefaultWriter = os.Stdout
  router.Use(gin.LoggerWithFormatter(func(param gin.LogFormatterParams) string {
```

```
    // your custom format
    return fmt.Sprintf("%s - [%s] \"%s %s %s %d %s \"%s\" %s\"\n",
        param.ClientIP,
        param.TimeStamp.Format(time.RFC1123),
        param.Method,
        param.Path,
        param.Request.Proto,
        param.StatusCode,
        param.Latency,
        param.Request.UserAgent(),
        param.ErrorMessage,
    )
  }))
  router.Use(gin.Recovery())

  router.GET("/ping", func(c *gin.Context) {
    c.String(http.StatusOK, "pong")
  })

  router.Run(":8080")
}
```

模型绑定与验证

JSON

```
// Binding from JSON
type Login struct {
  User     string `form:"user" json:"user" xml:"user"  binding:"required"`
  Password string `form:"password" json:"password" xml:"password"
binding:"required"`
}

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

  // Example for binding JSON ({"user": "manu", "password": "123"})
  router.POST("/loginJSON", func(c *gin.Context) {
    var json Login
    if err := c.ShouldBindJSON(&json); err != nil {
      c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
      return
    }

    if json.User != "manu" || json.Password != "123" {
      c.JSON(http.StatusUnauthorized, gin.H{"status": "unauthorized"})
      return
    }

    c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
  })

  // Example for binding XML (
  //  <?xml version="1.0" encoding="UTF-8"?>
```

```
  //  <root>
  //    <user>manu</user>
  //    <password>123</password>
  //  </root>)
  router.POST("/loginXML", func(c *gin.Context) {
    var xml Login
    if err := c.ShouldBindXML(&xml); err != nil {
      c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
      return
    }

    if xml.User != "manu" || xml.Password != "123" {
      c.JSON(http.StatusUnauthorized, gin.H{"status": "unauthorized"})
      return
    }

    c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
  })

  // Example for binding a HTML form (user=manu&password=123)
  router.POST("/loginForm", func(c *gin.Context) {
    var form Login
    // This will infer what binder to use depending on the content-type header.
    if err := c.ShouldBind(&form); err != nil {
      c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
      return
    }

    if form.User != "manu" || form.Password != "123" {
      c.JSON(http.StatusUnauthorized, gin.H{"status": "unauthorized"})
      return
    }

    c.JSON(http.StatusOK, gin.H{"status": "you are logged in"})
  })

  // Listen and serve on 0.0.0.0:8080
  router.Run(":8080")
}
```

Query String

```
package main

import (
  "log"
  "net/http"

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

type Person struct {
  Name    string `form:"name"`
  Address string `form:"address"`
}
```

```
func main() {
  route := gin.Default()
  route.Any("/testing", startPage)
  route.Run(":8085")
}

func startPage(c *gin.Context) {
  var person Person
  if c.ShouldBindQuery(&person) == nil {
    log.Println("====== Only Bind By Query String ======")
    log.Println(person.Name)
    log.Println(person.Address)
  }
  c.String(http.StatusOK, "Success")
}
```

QueryString or PostData

```
package main

import (
  "log"
  "net/http"
  "time"

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

type Person struct {
        Name       string    `form:"name"`
        Address    string    `form:"address"`
        Birthday   time.Time `form:"birthday" time_format:"2006-01-02"
time_utc:"1"`
        CreateTime time.Time `form:"createTime" time_format:"unixNano"`
        UnixTime   time.Time `form:"unixTime" time_format:"unix"`
}

func main() {
  route := gin.Default()
  route.GET("/testing", startPage)
  route.Run(":8085")
}

func startPage(c *gin.Context) {
  var person Person
  // If `GET`, only `Form` binding engine (`query`) used.
  // If `POST`, first checks the `content-type` for `JSON` or `XML`, then uses
`Form` (`form-data`).
  // See more at https://github.com/gin-
gonic/gin/blob/master/binding/binding.go#L88
        if c.ShouldBind(&person) == nil {
                log.Println(person.Name)
                log.Println(person.Address)
```

```
                log.Println(person.Birthday)
                log.Println(person.CreateTime)
                log.Println(person.UnixTime)
        }

  c.String(http.StatusOK, "Success")
}
```

Bind Uri

```
package main

import (
  "net/http"

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

type Person struct {
  ID string `uri:"id" binding:"required,uuid"`
  Name string `uri:"name" binding:"required"`
}

func main() {
  route := gin.Default()
  route.GET("/:name/:id", func(c *gin.Context) {
    var person Person
    if err := c.ShouldBindUri(&person); err != nil {
      c.JSON(http.StatusBadRequest, gin.H{"msg": err.Error()})
      return
    }
    c.JSON(http.StatusOK, gin.H{"name": person.Name, "uuid": person.ID})
  })
  route.Run(":8088")
}
```

Bind Header

```
package main

import (
  "fmt"
  "net/http"

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

type testHeader struct {
  Rate   int    `header:"Rate"`
  Domain string `header:"Domain"`
}

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

```
  r.GET("/", func(c *gin.Context) {
    h := testHeader{}

    if err := c.ShouldBindHeader(&h); err != nil {
      c.JSON(http.StatusOK, err)
    }

    fmt.Printf("%#v\n", h)
    c.JSON(http.StatusOK, gin.H{"Rate": h.Rate, "Domain": h.Domain})
  })

  r.Run()

// client
// curl -H "rate:300" -H "domain:music" 127.0.0.1:8080/
// output
// {"Domain":"music","Rate":300}
}
```

protobuf

1. 安装编译器

```
go install google.golang.org/protobuf/cmd/protoc-gen-go@latest
```

2. 生成代码

```
# 案例
protoc -I=$SRC_DIR --go_opt=paths=source_relative --go_out=$DST_DIR
$SRC_DIR/addressbook.proto
```