Gin框架讲解

作者：少林之巅

目录

1. Gin框架介绍

2. Gin框架参数绑定

3. Gin框架中间件&路由原理

Gin框架介绍

1. 简介

A. 基于httprouter开发的web框架。http://github.com/julienschmidt/httprouter

B. 提供Martini风格的API，但比Martini要快40倍

C. 非常轻量级，使用起来非常简洁

Gin框架介绍

2. Gin框架安装与使用

A. **安装：** go get -u github.com/gin-gonic/gin

B. import “go get -u github.com/gin-gonic/gin”

```
package main

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

func main() {
     //Default返回一个默认的路由引擎
     r := gin.Default()
     r.GET("/ping", func(c *gin.Context) {
          //输出json结果给调用方
          c.JSON(200, gin.H{
               "message": "pong",
          })
     })
     r.Run() // listen and serve on 0.0.0.0:8080
}
```

Gin框架介绍

3. Gin框架安装与使用

A. 支持restful风格的API

```
package main

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

func main() {
     //Default返回一个默认的路由引擎
     r := gin.Default()
     r.GET("/ping", func(c *gin.Context) {
          //输出json结果给调用方
          c.JSON(200, gin.H{
               "message": "pong",
          })
     })
     r.POST("/ping", func(c *gin.Context) {
          //输出json结果给调用方
          c.JSON(200, gin.H{
               "message": "pong",
          })
     })

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

Gin框架介绍

4. Restful风格的API

A. 把我们设计的API抽象成一个个资源，用URI来标识。

B. 针对每一个资源，使用统一的方法进行操作。

C. 同一的方法有：

a. GET，获取资源内容。

b. POST，创建资源。

c. PUT，更新资源。

d. DELETE，删除资源。

Gin框架介绍

5. 举例：用户信息接口设计，资源就是 /user/info

```
package main
import "github.com/gin-gonic/gin"
func main() {
     //Default返回一个默认的路由引擎
      r := gin.Default()
      r.GET("/user/info", func(c *gin.Context) {
        //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
     r.POST("/user/info", func(c *gin.Context) {
           //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
      r.PUT("/user/info", func(c *gin.Context) {
        //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
     r.DELETE("/user/info", func(c *gin.Context) {
           //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
      r.Run() // listen and serve on 0.0.0.0:8080
}
```

Gin框架介绍

6. 举例：用户信息接口设计，非restful风格的API

```
package main
import "github.com/gin-gonic/gin"
func main() {
     //Default返回一个默认的路由引擎
      r := gin.Default()
      r.GET("/user/info", func(c *gin.Context) {
        //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
     r.POST("/user/create", func(c *gin.Context) {
           //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
      r.POST("/user/delete", func(c *gin.Context) {
        //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
     r.POST("/user/update", func(c *gin.Context) {
           //输出json结果给调用方
            c.JSON(200, gin.H{
                  "message": "pong",
            })
      })
      r.Run() // listen and serve on 0.0.0.0:8080
}
```

Gin框架参数传递

1. 通过querystring传递, 比如： /user/search?username=少林&address=北京

```
package main

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

func main() {
	//Default返回一个默认的路由引擎
	r := gin.Default()
	r.GET("/user/search", func(c *gin.Context) {
		//username := c.DefaultQuery("username", "少林")
		username = c.Query("username")
		address := c.Query("address")
		//输出json结果给调用方
		c.JSON(200, gin.H{
			"message":  "pong",
			"username": username,
			"address":  address,
		})
	})

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

Gin框架参数传递

2. 通过路径传递, 比如： /user/search/少林/北京

```
package main

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

func main() {
	//Default返回一个默认的路由引擎
	r := gin.Default()
	r.GET("/user/info", func(c *gin.Context) {
		//username := c.DefaultQuery("username", "少林")
		username = c.Query("username")
		address := c.Query("address")
		//输出json结果给调用方
		c.JSON(200, gin.H{
			"message":  "pong",
			"username": username,
			"address":  address,
		})
	})

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

Gin框架参数传递

3. 通过表单进行提交, 比如： POST /user/search/

A. 下载postman测试工具，测试api非常方便，下载地址: https://www.getpostman.com/apps

```
package main

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

func main() {
	//Default返回一个默认的路由引擎
	r := gin.Default()
	r.GET("/user/info", func(c *gin.Context) {
		//username := c.DefaultQuery("username", "少林")
		username = c.Query("username")
		address := c.Query("address")
		//输出json结果给调用方
		c.JSON(200, gin.H{
			"message":  "pong",
			"username": username,
			"address":  address,
		})
	})

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

Gin框架处理文件上传

1. 单个文件上传

```
package main
import (
	"fmt"
	"log"
	"net/http"
	"github.com/gin-gonic/gin"
)
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, err := c.FormFile("file")
		if err != nil {
			c.JSON(http.StatusInternalServerError, gin.H{
				"message": err.Error(),
			})
			return
		}
		log.Println(file.Filename)
		dst := fmt.Sprintf("C:/tmp/%s", file.Filename)
		// Upload the file to specific dst.
		c.SaveUploadedFile(file, dst)
		c.JSON(http.StatusOK, gin.H{
			"message": fmt.Sprintf("'%s' uploaded!", file.Filename),
		})
	})
	router.Run(":8080")
}
```

Gin框架处理文件上传

2. 多个文件上传

```
package main
import (
	"fmt"
	"log"
	"net/http"
	"github.com/gin-gonic/gin"
)

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["file"]
		for index, file := range files {
			log.Println(file.Filename)
			dst := fmt.Sprintf("C:/tmp/%s_%d", file.Filename, index)
			c.SaveUploadedFile(file, dst)
		}
		c.JSON(http.StatusOK, gin.H{
			"message": fmt.Sprintf("%d files uploaded!", len(files)),
		})
	})
	router.Run(":8080")
}
```

Gin框架路由分组

1. 路由分组功能介绍

```
package main
import "github.com/gin-gonic/gin"
func login(ctx *gin.Context) {
	ctx.JSON(200, gin.H{
		"message": "success",
	})
}

func submit(ctx *gin.Context) {
	ctx.JSON(200, gin.H{
		"message": "success",
	})
}
func main() {
	//Default返回一个默认的路由引擎
	router := gin.Default()
	// Simple group: v1
	v1 := router.Group("/v1")
	{
		v1.POST("/login", login)
		v1.POST("/submit", submit)
	}
	// Simple group: v2
	v2 := router.Group("/v2")
	{
		v2.POST("/login", login)
		v2.POST("/submit", submit)
	}
	router.Run(":8080")
}
```

Gin框架参数绑定

1. 为什么要参数绑定，本质上是方便，提高开发效率

   A. 通过反射的机制，自动提取querystring、form表单、json、xml等参数到struct中

   B. 通过http协议中的context type，识别是json、xml或者表单

```
package main
import (
	"net/http"
	"github.com/gin-gonic/gin"
)
// Binding from JSON
type Login struct {
	User     string `form:"user" json:"user" binding:"required"`
	Password string `form:"password" json:"password" binding:"required"`
}
func main() {
	router := gin.Default()
	// Example for binding JSON ({"user": "manu", "password": "123"})
	router.POST("/loginJSON", func(c *gin.Context) {
		var login Login
		if err := c.ShouldBindJSON(&login); err == nil {
			c.JSON(http.StatusOK, gin.H{
				"user":     login.User,
				"password": login.Password,
			})
		} else {
			c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
		}
	})
	// Example for binding a HTML form (user=manu&password=123)
	router.POST("/loginForm", func(c *gin.Context) {
		var login Login
		// This will infer what binder to use depending on the content-type header.
		if err := c.ShouldBind(&login); err == nil {
			c.JSON(http.StatusOK, gin.H{
				"user":     login.User,
				"password": login.Password,
			})
		} else {
			c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
		}
	})
	// Example for binding a HTML querystring (user=manu&password=123)
	router.GET("/loginForm", func(c *gin.Context) {
		var login Login
		if err := c.ShouldBind(&login); err == nil {
			c.JSON(http.StatusOK, gin.H{
				"user":     login.User,
				"password": login.Password,
			})
		} else {
			c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
		}
	})
	router.Run(":8080")
}
```

Gin框架渲染

1. 渲染json

A. gin.Context.JSON方法进行渲染

```
package main
import (
	"net/http"

	"github.com/gin-gonic/gin"
)
func main() {
	r := gin.Default()

	// gin.H is a shortcut for map[string]interface{}
	r.GET("/someJSON", func(c *gin.Context) {
		//第一种方式,自己拼json
		c.JSON(http.StatusOK, gin.H{"message": "hey", "status": http.StatusOK})
	})
	r.GET("/moreJSON", func(c *gin.Context) {
		// You also can use a struct
		var msg struct {
			Name    string `json:"user"`
			Message string
			Number  int
		}
		msg.Name = "Lena"
		msg.Message = "hey"
		msg.Number = 123
		// Note that msg.Name becomes "user" in the JSON
		c.JSON(http.StatusOK, msg)
	})
	// Listen and serve on 0.0.0.0:8080
	r.Run(":8080")
}
```

Gin框架渲染

2. 渲染xml

A. gin.Context.XML方法进行渲染

```
package main

import (
	"net/http"

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

func main() {
	r := gin.Default()
	r.GET("/moreXML", func(c *gin.Context) {
		// You also can use a struct
		type MessageRecord struct {
			Name    string
			Message string
			Number  int
		}

		var msg MessageRecord
		msg.Name = "Lena"
		msg.Message = "hey"
		msg.Number = 123
		c.XML(http.StatusOK, msg)
	})
	// Listen and serve on 0.0.0.0:8080
	r.Run(":8080")
}
```

Gin框架渲染

3. 渲染模板

A. gin.Context.HTML方法进行渲染

```
package main

import (
	"net/http"

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

func main() {
	router := gin.Default()
	router.LoadHTMLGlob("templates/**/*")
	router.GET("/posts/index", func(c *gin.Context) {
		c.HTML(http.StatusOK, "posts/index.tmpl", gin.H{
			"title": "Posts",
		})
	})
	router.GET("/users/index", func(c *gin.Context) {
		c.HTML(http.StatusOK, "users/index.tmpl", gin.H{
			"title": "Users",
		})
	})
	router.Run(":8080")
}
```

Gin框架中间件

1. Gin框架中间件

A. Gin框架允许在请求处理过程中，加入用户自己的钩子函数。这个钩子函数就叫中间件

B. 因此，可以使用中间件处理一些公共业务逻辑，比如耗时统计，日志打印，登陆校验.

Gin框架中间件

2. 编写自己的中间件

```
package main
import (
	"log"
	"time"
	"net/http"
	"github.com/gin-gonic/gin"
)
func StatCost() gin.HandlerFunc {
	return func(c *gin.Context) {
		t := time.Now()
		//可以设置一些公共参数
		c.Set("example", "12345")
		//等其他中间件先执行
		c.Next()
		//获取耗时
		latency := time.Since(t)
		log.Print(latency)
	}
}
func main() {
	r := gin.New()
	r.Use(StatCost())
	r.GET("/test", func(c *gin.Context) {
		example := c.MustGet("example").(string)
		// it would print: "12345"
		log.Println(example)
		c.JSON(http.StatusOK, gin.H{
			"message": "success",
		})
	})
	// Listen and serve on 0.0.0.0:8080
	r.Run(":8080")
}
```

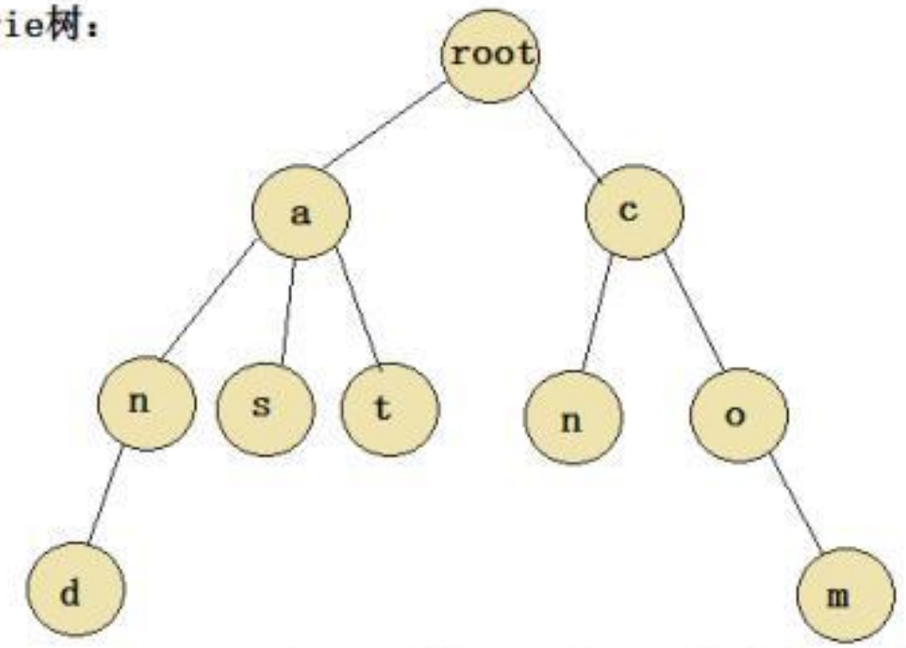
Gin框架路由原理

1. Gin框架路由介绍

A. 路由部分用的是：http://github.com/julienschmidt/httprouter

B. 对于所有的路由规则，httprouter会构造一颗前缀树

Trie树：



Gin框架路由原理

2. Gin框架路由介绍

```
package main
import "github.com/gin-gonic/gin"
func index(ctx *gin.Context) {
	ctx.JSON(200, gin.H{
		"message": "index",
	})
}

func main() {
	//Default返回一个默认的路由引擎
	router := gin.Default()
	router.POST("/", index)
	router.POST("/search", index)
	router.POST("/support", index)
	router.POST("/blog/:post", index)
	router.POST("/about", index)
	router.POST("/contact", index)
	router.Run(":8080")
}
```

3. 生成的前缀树，如下所示：