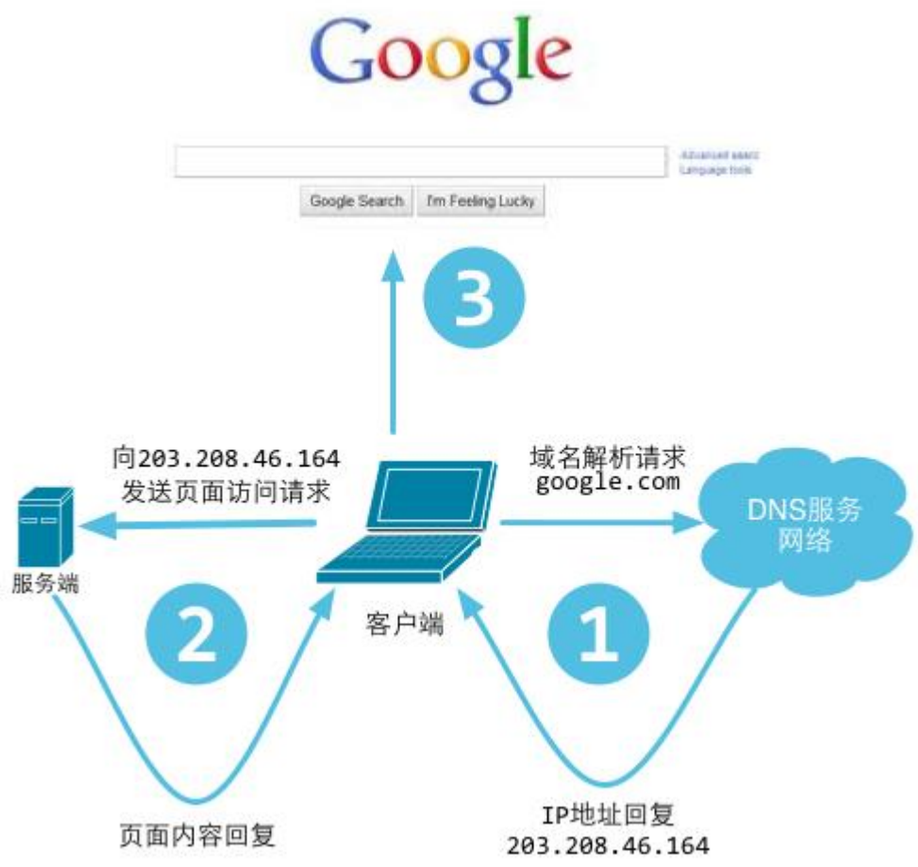
Day8 web编程基础

tony

目录

Web编程基础

1. Web工作方式



Web编程基础

2. HTTP协议详解

a. http 请求包体

```
GET /domains/example/ HTTP/1.1          //请求行: 请求方法 请求URI HTTP协议/协议版本
Host: www.iana.org                      //服务端的主机名
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.4 (KHTML, like Gecko)
Chrome/22.0.1229.94 Safari/537.4                //浏览器信息
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8    //客户
端能接收的MIME
Accept-Encoding: gzip,deflate,sdch          //是否支持流压缩
Accept-Charset: UTF-8,*;q=0.5                //客户端字符编码集
//空行,用于分割请求头和消息体
//消息体,请求资源参数,例如POST传递的参数
```

Web编程基础

2. HTTP协议详解

b. http 响应包体

```
HTTP/1.1 200 OK                                    //状态行
Server: nginx/1.0.8                                //服务器使用的WEB软件名及版本
Date:Date: Tue, 30 Oct 2012 04:14:25 GMT           //发送时间
Content-Type: text/html                            //服务器发送信息的类型
Transfer-Encoding: chunked                         //表示发送HTTP包是分段发的
Connection: keep-alive                             //保持连接状态
Content-Length: 90                                 //主体内容长度
//空行 用来分割消息头和主体
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"... //消息体
```

3. http Keep-Alive特性

A. Keep-Alive用来保持连接

B. Keep-Alive通过web服务器进行设置，保持的时间

4. Web程序开发

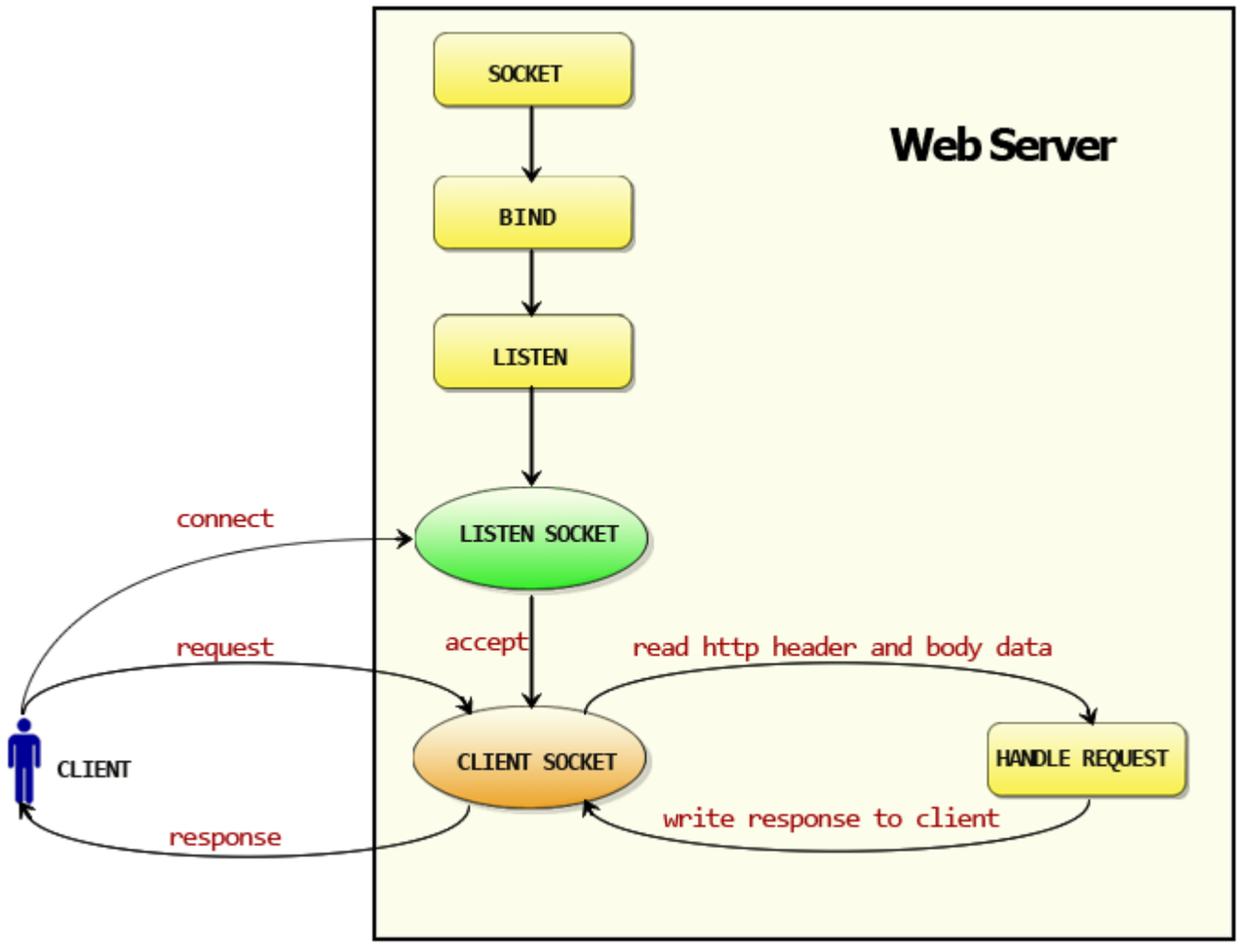
A. 标准包 “net/http”封装web服务相关功能

B. 使用简单、性能媲美nginx。

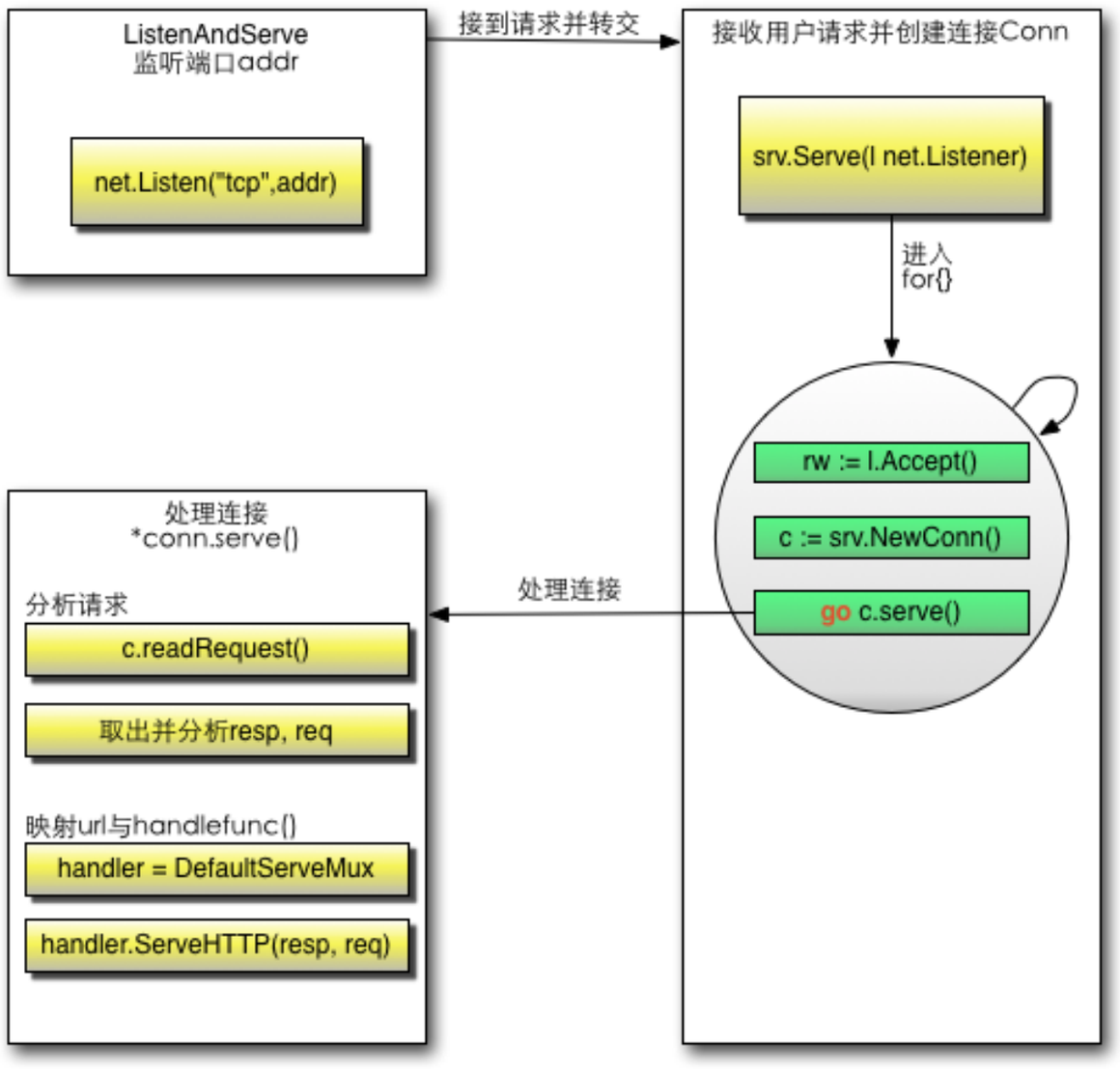
Web程序开发

```
package main

import (
	"fmt"
	"net/http"
	"strings"
	"log"
)

func sayhelloName(w http.ResponseWriter, r *http.Request) {
	r.ParseForm()  //解析参数，默认是不会解析的
	fmt.Println(r.Form)  //这些信息是输出到服务器端的打印信息
	fmt.Println("path", r.URL.Path)
	fmt.Println("scheme", r.URL.Scheme)
	fmt.Println(r.Form["url_long"])
	for k, v := range r.Form {
		fmt.Println("key:", k)
		fmt.Println("val:", strings.Join(v, ""))
	}
	fmt.Fprintf(w, "Hello world!") //这个写入到w的是输出到客户端的
}

func main() {
	http.HandleFunc("/", sayhelloName) //设置访问的路由
	err := http.ListenAndServe(":9090", nil) //设置监听的端口
	if err != nil {
		log.Fatal("ListenAndServe: ", err)
	}
}
```

5. Golang web服务工作方式



5. Golang web服务工作方式



6. Web表单

A. 通过<form> </form>括起来的区域，允许用户提交数据。

B. Go对于表单处理非常方便

Web表单

7. Html代码

```
<html>
<head>
<title></title>
</head>
<body>
<form action="/login" method="post">
     用户名:<input type="text" name="username">
     密码:<input type="password" name="password">
     <input type="submit" value="登录">
</form>
</body>
</html>
```

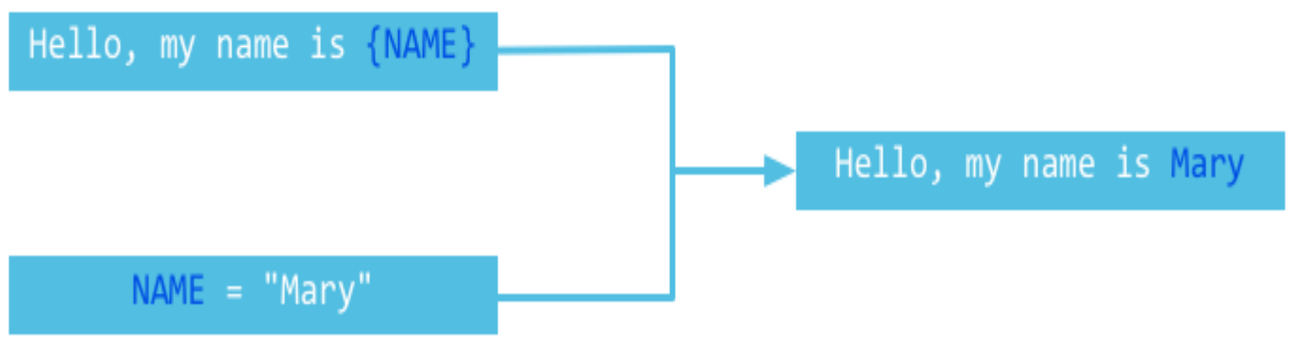
8. Go代码

```
package main

import (
	"fmt"
	"html/template"
	"log"
	"net/http"
	"strings"
)
func login(w http.ResponseWriter, r *http.Request) {
	fmt.Println("method:", r.Method) //获取请求的方法
	if r.Method == "GET" {
		t, _ := template.ParseFiles("login.gtpl")
		log.Println(t.Execute(w, nil))
	} else {
		//请求的是登录数据，那么执行登录的逻辑判断
		fmt.Println("username:", r.Form["username"])
		fmt.Println("password:", r.Form["password"])
	}
}
func main() {
	http.HandleFunc("/login", login)         //设置访问的路由
	err := http.ListenAndServe(":9090", nil) //设置监听的端口
	if err != nil {
		log.Fatal("ListenAndServe: ", err)
	}
}
```

9. 模板替换

A. {{}}来包含需要在渲染时被替换的字段，{{.}}表示当前的对象。

B. 通过{{.FieldName}}访问对象的属性。



9. 模板替换

```
package main

import (
        "fmt"
        "os"
        "text/template"
)

type Person struct {
        Name string
        age  string
}

func main() {
        t, err := template.ParseFiles("./index.html")
        if err != nil {
                fmt.Println("parse file err:", err)
                return
        }
        p := Person{Name: "Mary", age: "31"}
        if err := t.Execute(os.Stdout, p); err != nil {
                fmt.Println("There was an error:", err.Error())
        }
}
```

Web模板

10.If判断

```
<html>
	<head>
	</head>
	<body>
		{{if gt .Age 18}}
		<p>hello, old man, {{.Name}}</p>
		{{else}}
		<p>hello,young man, {{.Name}}</p>
		{{end}}
	</body>
</html>
```

- not 非

{{if not .condition}}
{{end}}

- and 与

{{if and .condition1 .condition2}}
{{end}}

- or 或

{{if or .condition1 .condition2}}
{{end}}

- eq 等于

{{if eq .var1 .var2}}
{{end}}

- ne 不等于

{{if ne .var1 .var2}}
{{end}}

- lt 小于 (less than)

{{if lt .var1 .var2}}
{{end}}

- le 小于等于

{{if le .var1 .var2}}
{{end}}

- gt 大于

{{if gt .var1 .var2}}
{{end}}

- ge 大于等于

{{if ge .var1 .var2}}
{{end}}

11. with 语法

```
<html>
    <head>
    </head>
    <body>
        {{with .Name}}
        <p>hello, old man, {{.}}</p>
        {{end}}
    </body>
</html>
```

Web模板

12.循环

```
<html>
    <head>
    </head>
    <body>
        {{range .}}
            {{if gt .Age 18}}
            <p>hello, old man, {{.Name}}</p>
            {{else}}
            <p>hello,young man, {{.Name}}</p>
            {{end}}
        {{end}}
    </body>
</html>
```

目录

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")
}
```

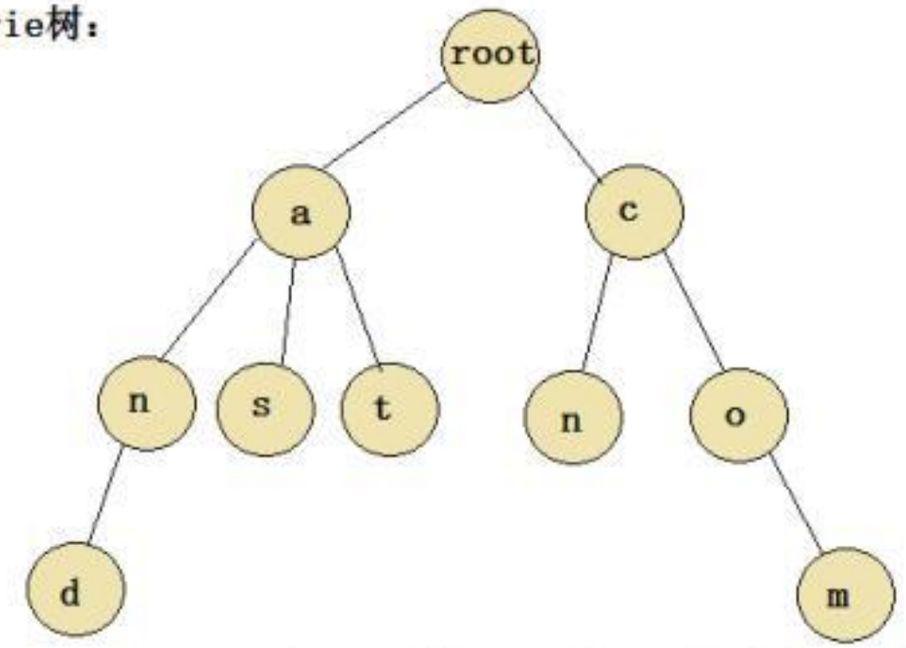
Gin框架路由原理

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：

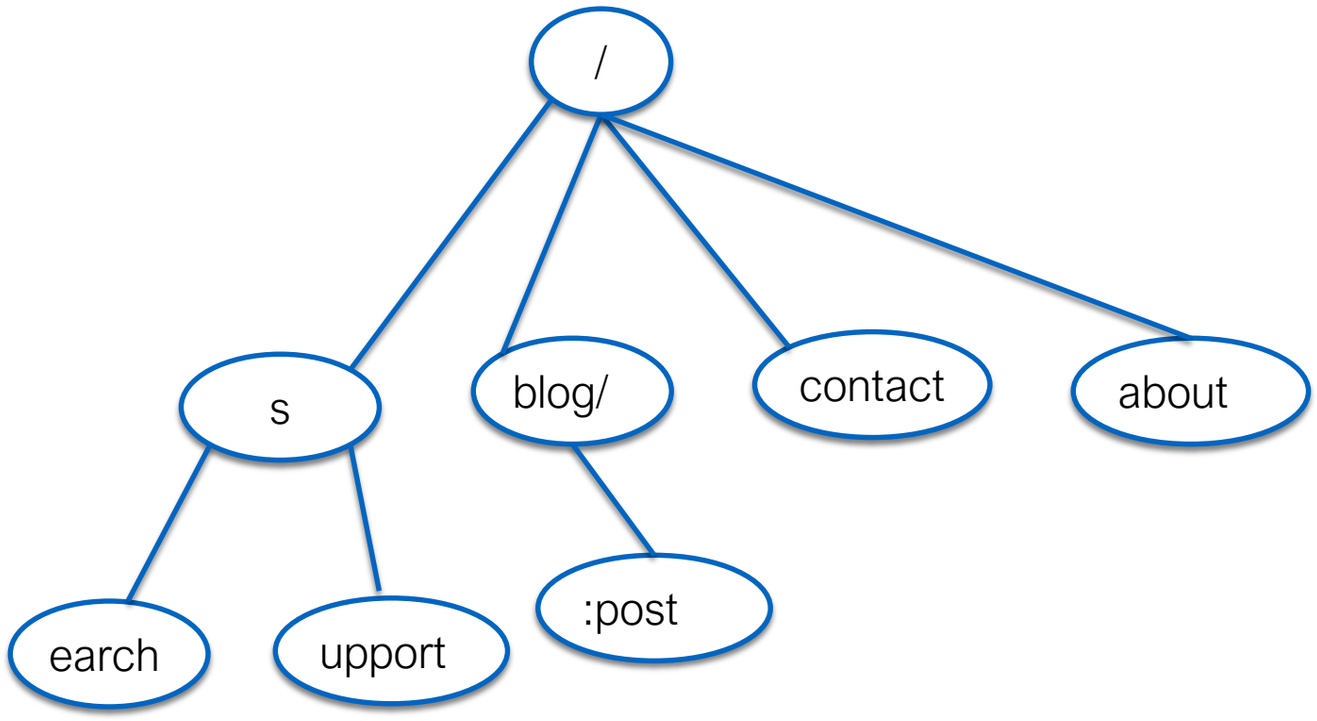


Gin框架路由原理

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：



课后练习

1. 实现一个图书管理系统，具有以下功能：

   a. 书籍录入功能，书籍信息包括书名、副本数、作者、出版日期

   b. 书籍查询功能，按照书名、作者、出版日期等条件检索

   c. 学生信息管理功能，管理每个学生的姓名、年级、身份证、性别、借了什么书等信息

   d. 借书功能，学生可以查询想要的书籍，进行借出

   e. 书籍管理功能，可以看到每种书被哪些人借出了

   f. 数据采用Mysql，使用数据库实现图书管理相关功能。

   g. 使用本章的知识，开发一个基于web的图书管理系统。