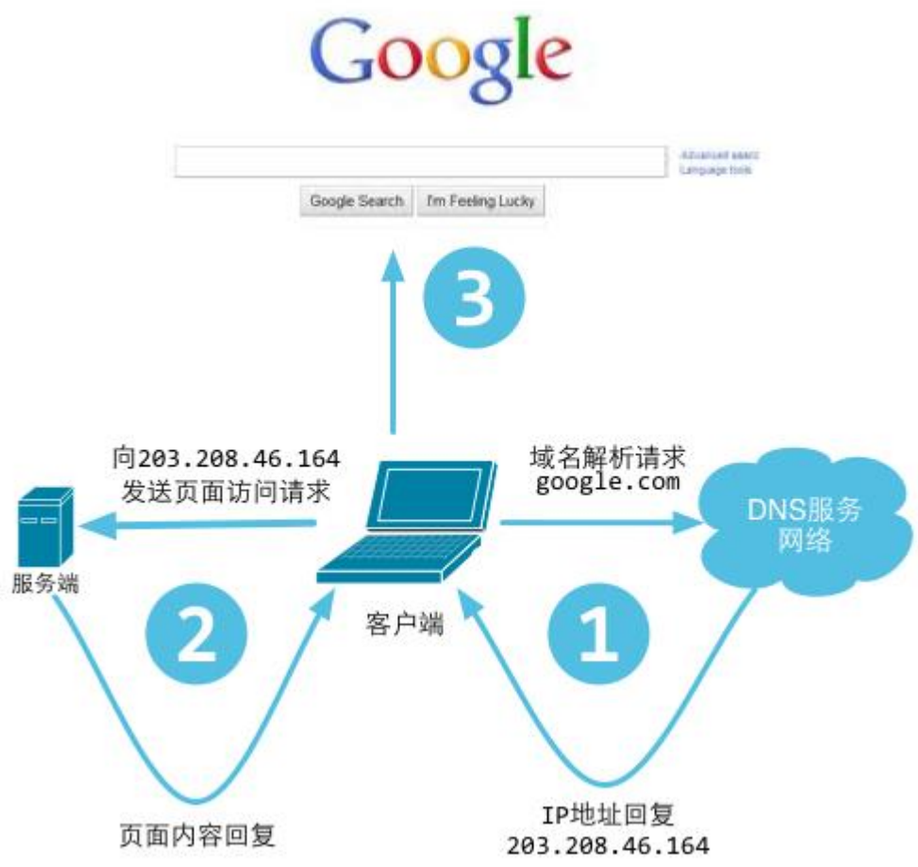
WEB编程

作者：少林之巅

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

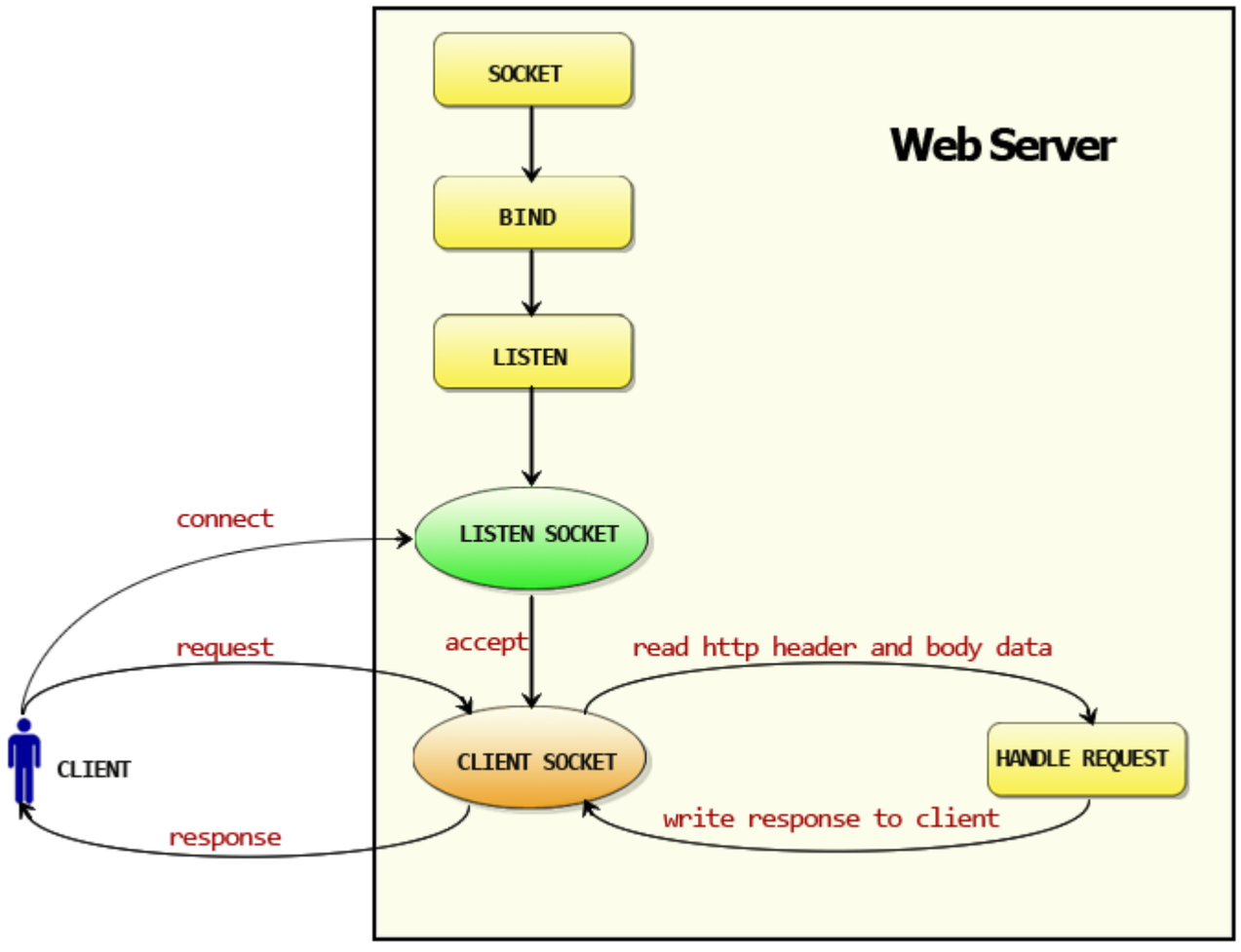
A. 标准包 “net/http”封装web服务相关功能

B. 使用简单、性能媲美nginx。

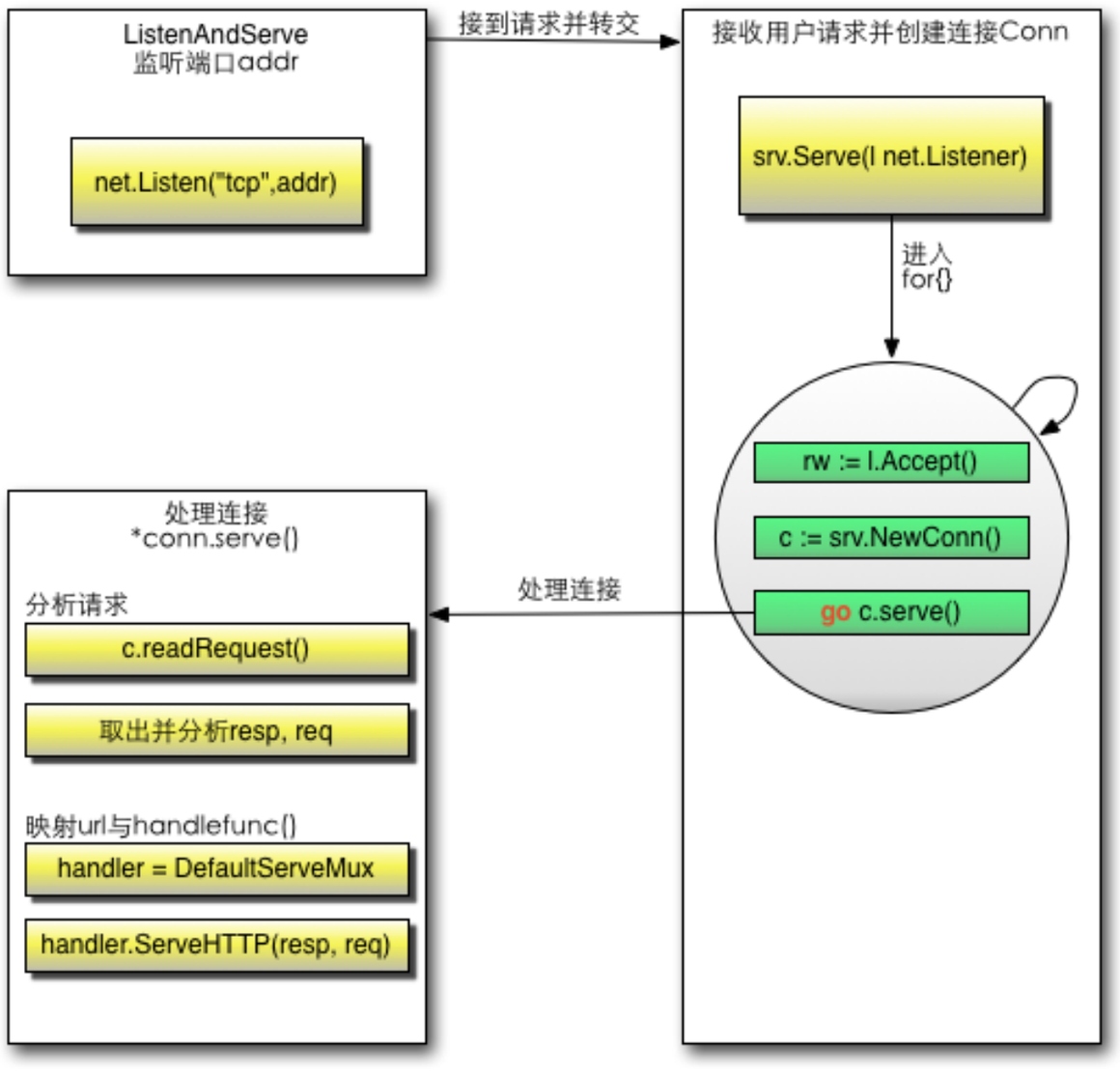
Web程序开发

```
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>
```

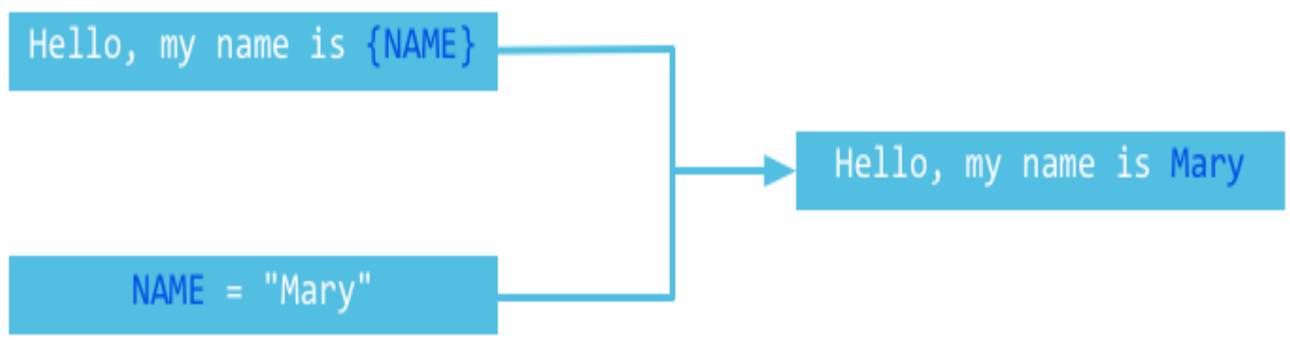
8. Go代码

```
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>
```

Web服务器平滑升级

13. 平滑升级

A. 程序升级过程中，如何不影响正在处理的请求?

B. 正在处理的请求怎么办?

C. 新进来的请求怎么办?

Web服务器平滑升级

14. 正在处理的请求怎么办？

A. 等待处理完成之后，再退出。

B. Golang1.8之后已经支持。

C. 装逼的说法，就是优雅关闭！

Web服务器平滑升级

15. 新进来的请求怎么办？

A. Fork一个子进程，继承父进程的监听socket

B. 子进程启动成功之后，接收新的连接。

C. 父进程停止接收新的连接，等已有的请求处理完毕，退出

D. 优雅重启成功！

16. 子进程如何继承父进程的文件句柄?

A. 通过os.Cmd对象中的ExtraFiles参数进行传递。

B. 文件句柄继承实例分析。

Web服务器平滑升级

17. web server优雅重启实战?

A. 使用go1.8版本的Shutdown方法进行优雅关闭

B. 使用socket继承实现，子进程接管父进程的监听socket。

Web服务器平滑升级

18. 信号处理

A. 通过kill命令给正在运行的程序发送信号。

B. 不处理的话，程序会panic处理。

Web服务器平滑升级

19. 信号介绍

信号	值	说明
SIGHUP	1	终端控制进程结束(终端连接断开)
SIGINT	2	用户发送INTR字符(Ctrl+C)触发
SIGQUIT	3	用户发送QUIT字符(Ctrl+/)触发
SIGKILL	9	无条件结束程序(不能被捕获、阻塞或忽略)
SIGUSR1	10	用户保留
SIGUSR2	12	用户保留
SIGPIPE	13	消息管道损坏(FIFO/Socket通信时，管道未打开而进行写操作)
SIGALRM	14	时钟定时信号
SIGTERM	15	结束程序(可以被捕获、阻塞或忽略)