# Go语言网络编程

why404@七牛云存储

2012/07/21

# Overview

- Socket
- HTTP
- RPC
- JSON
- JSON-RPC
- Web Devlopment

# Socket 编程

# HTTP 编程

- HTTP Client
- HTTP Server

# HTTP Client Methods

- http.Get
- http.Post
- http.PostForm
- http.Head
- (*http.Client).Do

# http.Get

```
func (c *Client) Get(url string) (r *Response, err error)

resp, err := http.Get("http://example.com/")
if err != nil {
    // handle error ...
    return
}
defer resp.Body.close()
io.Copy(os.Stdout, resp.Body)

// Get is a wrapper around http.DefaultClient.Get
```

# http.Post

```
func (c *Client) Post(url string, bodyType string, body io.Reader) (r *Response, err error)
```

```
resp, err := http.Post("http://example.com/upload", "image/jpeg", &imageDataBuf)
if err != nil {
    // handle error ...
    return
}
if resp.StatusCode != http.StatusOK {
    // handle error ...
    return
}
// ...

// Post is a wrapper around http.DefaultClient.Post
```

# http.PostForm

```
func (c *Client) PostForm(url string, data url.Values) (r *Response, err error)

resp, err := http.PostForm("http://example.com/posts",
url.Values{"title": {"article title"}, "content": {"article body"}})
if err != nil {
    // handle error ...
    return
}
// ...

// PostForm is a wrapper around http.DefaultClient.PostForm
```

# http.Head

```
func (c *Client) Head(url string) (r *Response, err error)
```

```
resp, err := http.Head("http://example.com/")

// Head is a wrapper around http.DefaultClient.Head
```

# (*http.Client).Do

```
func (c *Client) Do(req *Request) (resp *Response, err error)

req, err := http.NewRequest("GET", "http://example.com", nil)
// ...
req.Header.Add("User-Agent", "Gobook Custom User-Agent")

client := &http.Client{ //... }
resp, err := client.Do(req)
// ...
```

# HTTP Server

- 处理 http 请求
- 自定义 http.Server

# 处理 http 请求

```
func ListenAndServe(addr string, handler Handler) error

http.Handle("/foo", fooHandler)
http.HandleFunc("/bar", func(w http.ResponseWriter, r
*http.Request) {
    fmt.Fprintf(w, "Hello, %q", html.EscapeString(r.URL.Path))
})
log.Fatal(http.ListenAndServe(":8080", nil))

// http.Handle 或 http.HandleFunc 缺省注入
http.DefaultServeMux
```

# 自定义 http.Server

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

# HTTP 高级话题

# 自定义 http.Client

```
// in the "net/http" package
var DefaultClient = &Client{}

// so..
http.Get == http.DefaultClient.Get,

...
```

# http.Client Struct

```
type Client struct {

    // Transport 指定了执行一个 HTTP 请求的运行机制
    // http.DefaultTransport
    // http.RoundTripper
    Transport RoundTripper

    CheckRedirect func(req *Request, via []*Request) error

    Jar CookieJar
}
```

# 自定义 http.Client 好处

- 发送自定义 HTTP Headers
- 改写重定向策略
- etc.

# 定义具体的 http.Client

```
client := &http.Client{
    CheckRedirect: redirectPolicyFunc,
}

resp, err := client.Get("http://example.com")
// ...

req, err := http.NewRequest("GET", "http://example.com", nil)
// ...
req.Header.Add("User-Agent", "Our Custom User-Agent")
req.Header.Add("If-None-Match", `W/"TheFileEtag"`)
resp, err := client.Do(req)
// ...
```

# http.Transport

- http.Transport 对象指定了执行一个 HTTP 请求时的运行规则
- 实现了 http.RoundTripper 接口