Programming in Go

Matt Holiday
Christmas 2020

Networking with HTTP

Go network libraries

The Go standard library has many packages for making web servers:

That includes:

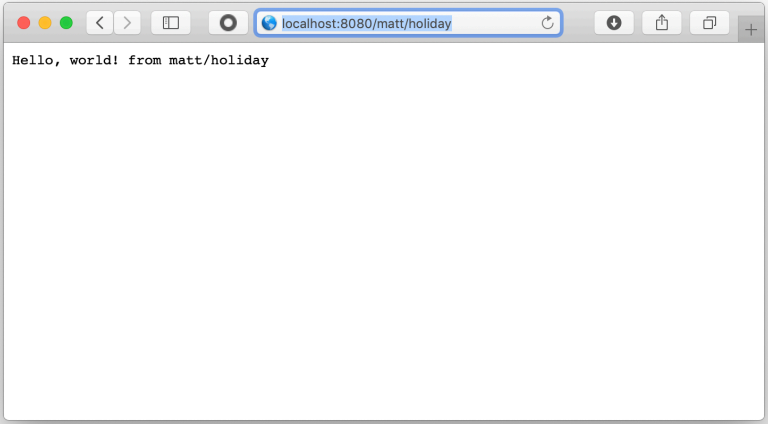
- client & server sockets
- route multiplexing
- HTTP and HTML, including HTML templates
- JSON and other data formats
- cryptographic security
- SQL database access
- compression utilities
- image generation

There are also lots of 3rd-party packages with improvements

A very short web server

```
package main

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

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, world! from %s\n", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

A very short web server



Go HTTP design

An HTTP handler function is an instance of an interface

```
type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}

type HandlerFunc func(ResponseWriter, *Request)

func (f HandlerFunc) ServeHTTP(w ResponseWriter, r *Request) {
    f(w, r)
}

// The HTTP framework can call a method on a function type

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, world! from %s\n", r.URL.Path[1:])
}
```

A very short web client

```
package main

import ("fmt"; "io/ioutil"; "net/http"; "os")

func main() {
    resp, _ := http.Get("http://localhost:8080/" + os.Args[1])
    defer resp.Body.Close()

    if resp.StatusCode == http.StatusOK {
        body, _ := ioutil.ReadAll(resp.Body)
        fmt.Println(string(body))
    }
}

// $ go run client.go matt/holiday
// Hello, world! from matt/holiday
```

A simple JSON REST client

```
package main

import ("fmt"; "io/ioutil"; "net/http")
const url = "https://jsonplaceholder.typicode.com"

func main() {
    resp, _ := http.Get(url + "/todos/1")
    defer resp.Body.Close()

    if resp.StatusCode == http.StatusOK {
        body, _ := ioutil.ReadAll(resp.Body)
        fmt.Println(string(body))
    }
}

// $ go run client.go
// {"userId": 1, "id": 1, "title": "delectus aut autem", "completed": false}
```

A simple JSON REST client

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
)

type todo struct {
    UserID    int    `json:"userID"`
    ID        int    `json:"id"`
    Title     string `json:"title"`
    Completed bool   `json:"completed"`
}

const base = "https://jsonplaceholder.typicode.com"
```

A simple JSON REST client

```
func main() {
    var item todo

    resp, _ := http.Get(base + "/todos/1")

    defer resp.Body.Close()

    body, _ := ioutil.ReadAll(resp.Body)

    _ := json.Unmarshal(body, &item)

    fmt.Printf("%#v\n", item)
}

// $ go run client.go
// main.todo{UserID:1, ID:1, Title:"delectus aut autem", Completed:false}
```

Serving from a template

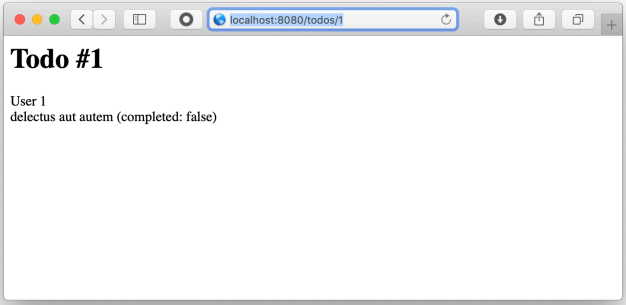
```
package main

import (
    "encoding/json"
    "html/template"
    "io/ioutil"
    "log"
    "net/http"
)

type todo struct {
    UserID    int    `json:"userID"`
    ID        int    `json:"id"`
    Title     string `json:"title"`
    Completed bool   `json:"completed"`
}

const base = "https://jsonplaceholder.typicode.com/"
```

Serving from a template

```
var form = `
<h1>Todo #{{.ID}}</h1>
<div>{{printf "User %d" .UserID}}</div>
<div>{{printf "%s (completed: %t)" .Title .Completed}}</div>`

func handler(w http.ResponseWriter, r *http.Request) {
    var item todo

    resp, _ := http.Get(base + r.URL.Path[1:])
    defer resp.Body.Close()
    body, _ := ioutil.ReadAll(resp.Body)
    _ = json.Unmarshal(body, &item)

    tmpl := template.New("mine")

    tmpl.Parse(form)
    tmpl.Execute(w, item)
}
```

Serving from a template

```
func main() {
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```



Serving up an error

```
func handler(w http.ResponseWriter, r *http.Request) {
    var item todo

    resp, _ := http.Get(base + r.URL.Path[1:])
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        http.NotFound(w, r)
        return
    }

    body, _ := ioutil.ReadAll(resp.Body)
    _ = json.Unmarshal(body, &item)

    tmpl := template.New("mine")
    tmpl.Parse(form)
    tmpl.Execute(w, item)
}
```

Serving up a wiki

See the Golang article Writing Web Applications

The tutorial includes:

- creating a data structure with load and save methods
- using the net/http package to build web applications
- using the html/template package to process HTML templates
- using the regexp package to validate user input
- using closures