Go ServeMux

last modified April 11, 2024

In this article we show how to do request routing and dispatching in Golang with ServeMux.

```
$ go version
go version go1.22.2 linux/amd64
```

We use Go version 1.22.2.

HTTP

The *Hypertext Transfer Protocol (HTTP)* is an application protocol for distributed, collaborative, hypermedia information systems. HTTP protocol is the foundation of data communication for the World Wide Web.

ServeMux

ServeMux is an HTTP request multiplexer. It is used for request routing and dispatching. The request routing is based on URL patterns. Each incoming request's URL is matched against a list of registered patterns. A handler for the pattern that most closely fits the URL is called.

Go NewServeMux

The `NewServeMux` function allocates and returns a new ServeMux.

main.go

```
package main

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

func main() {

    mux := http.NewServeMux()

    now := time.Now()

    mux.HandleFunc("/today", func(rw http.ResponseWriter, _ *http.Request) {

        rw.Write([]byte(now.Format(time.ANSIC)))
    })

    log.Println("Listening...")
    http.ListenAndServe(":3000", mux)
}
```

The example creates an HTTP server which returns the current datetime for the `/today` URL pattern.

```
mux := http.NewServeMux()
```

A new ServeMux is created.

```
mux.HandleFunc("/today", func(rw http.ResponseWriter, _ *http.Request) {

    rw.Write([]byte(now.Format(time.ANSIC)))
})
```

A handle to the `/today` pattern is added with `HandleFunc` function.

```
http.ListenAndServe(":3000", mux)
```

The created multiplexer is passed to the `ListenAndServe` function.

Go DefaultServeMux

`DefaultServeMux` is a just a ServeMux. It is used when we pass `nil` to the `ListenAndServe` method for the second parameter.

The `http.Handle` and `http.HandleFunc` global functions use the `DefaultServeMux` multiplexer.

main.go

```
package main

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

func main() {

    http.HandleFunc("/", HelloHandler)

    log.Println("Listening...")
    log.Fatal(http.ListenAndServe(":3000", nil))
}

func HelloHandler(w http.ResponseWriter, _ *http.Request) {

    fmt.Fprintf(w, "Hello there!")
}
```

The example uses the default multiplexer.

Custom handler

The `func (*ServeMux) Handle` registers the handler for the given pattern.

The `http.HandlerFunc` is an adapter that turns a function, if it has the right signature, into an `http.Handler`.

main.go

```
package main

import (
```

```
    "fmt"
    "log"
    "net/http"
)

type helloHandler struct {
}

func (h *helloHandler) ServeHTTP(w http.ResponseWriter, _ *http.Request) {

    fmt.Fprintf(w, "Hello there!")
}

func main() {

    mux := http.NewServeMux()

    hello := &helloHandler{}
    mux.Handle("/hello", hello)

    log.Println("Listening...")
    http.ListenAndServe(":3000", mux)
}
```

In the example, we create a custom handler.

```
type helloHandler struct {
}
```

A new type is declared.

```
func (h *helloHandler) ServeHTTP(w http.ResponseWriter, _ *http.Request) {

    fmt.Fprintf(w, "Hello there!")
}
```

To become a handler, the type must implement the `ServeHTTP` function.

```
hello := &helloHandler{}
mux.Handle("/hello", hello)
```

We create the handler and pass it to the `Handle` function.

Source

Go net/http package - reference

In this article we have showed how to do request routing and dispatching in Go with ServeMux.

Author

My name is Jan Bodnar, and I am a passionate programmer with extensive programming experience. I have been writing programming articles since 2007. To date, I have authored over 1,400 articles and 8 e-books. I possess more than ten years of experience in teaching programming.

List all Go tutorials.