Routing (using gorilla/mux)

Introduction

Go's `net/http` package provides a lot of functionalities for the HTTP protocol. One thing it doesn't do very well is complex request routing like segmenting a request url into single parameters. Fortunately there is a very popular package for this, which is well known for the good code quality in the Go community. In this example you will see how to use the `gorilla/mux` package to create routes with named parameters, GET/POST handlers and domain restrictions.

Installing the `gorilla/mux` package

`gorilla/mux` is a package which adapts to Go's default HTTP router. It comes with a lot of features to increase the productivity when writing web applications. It is also compliant to Go's default request handler signature `func (w http.ResponseWriter, r *http.Request)`, so the package can be mixed and machted with other HTTP libraries like middleware or exisiting applications. Use the `go get` command to install the package from GitHub like so:

```
go get -u github.com/gorilla/mux
```

Creating a new Router

First create a new request router. The router is the main router for your web application and will later be passed as parameter to the server. It will receive all HTTP connections and pass it on to the request handlers you will register on it. You can create a new router like so:

```
r := mux.NewRouter()
```

Registering a Request Handler

Once you have a new router you can register request handlers like usual. The only difference is, that instead of calling `http.HandleFunc(...)` , you call `HandleFunc` on your router like this: `r.HandleFunc(...)` .

URL Parameters

The biggest strength of the `gorilla/mux` Router is the ability to extract segments from the request URL. As an example, this is a URL in your application:

```
/books/go-programming-blueprint/page/10
```

This URL has two dynamic segments:
Book title slug (go-programming-blueprint)

Page (10)

To have a request handler match the URL mentioned above you replace the dynamic segments of with placeholders in your URL pattern like so:

```
r.HandleFunc("/books/{title}/page/{page}", func(w http.ResponseWriter, r *http.Requ
    // get the book
    // navigate to the page
})
```

The last thing is to get the data from these segments. The package comes with the function `mux.Vars(r)` which takes the `http.Request` as parameter and returns a map of the segments.

```
func(w http.ResponseWriter, r *http.Request) {
    vars := mux.Vars(r)
    vars["title"] // the book title slug
    vars["page"] // the page
}
```

Setting the HTTP server's router

Ever wondered what the `nil` in `http.ListenAndServe(":80", nil)` ment? It is the parameter for the main router of the HTTP server. By default it's `nil`, which means to use the default router of the `net/http` package. To make use of your own router, replace the `nil` with the variable of your router `r`.

```
http.ListenAndServe(":80", r)
```

The Code (for copy/paste)

This is the complete code that you can use to try out the things you've learned in this example.

```
package main

import (
    "fmt"
    "net/http"

    "github.com/gorilla/mux"
)

func main() {
    r := mux.NewRouter()

    r.HandleFunc("/books/{title}/page/{page}", func(w http.ResponseWriter, r *http.
        vars := mux.Vars(r)
        title := vars["title"]
        page := vars["page"]

        fmt.Fprintf(w, "You've requested the book: %s on page %s\n", title, page)
    })

    http.ListenAndServe(":80", r)
}
```

Features of the gorilla/mux Router

Methods

Restrict the request handler to specific HTTP methods.

```
r.HandleFunc("/books/{title}", CreateBook).Methods("POST")
r.HandleFunc("/books/{title}", ReadBook).Methods("GET")
r.HandleFunc("/books/{title}", UpdateBook).Methods("PUT")
r.HandleFunc("/books/{title}", DeleteBook).Methods("DELETE")
```

Hostnames & Subdomains

Restrict the request handler to specific hostnames or subdomains.

```
r.HandleFunc("/books/{title}", BookHandler).Host("www.mybookstore.com")
```

Schemes

Restrict the request handler to http/https.

```
r.HandleFunc("/secure", SecureHandler).Schemes("https")
r.HandleFunc("/insecure", InsecureHandler).Schemes("http")
```

Path Prefixes & Subrouters

Restrict the request handler to specific path prefixes.

```
bookrouter := r.PathPrefix("/books").Subrouter()
bookrouter.HandleFunc("/", AllBooks)
bookrouter.HandleFunc("/{title}", GetBook)
```