Middleware (Advanced)

Middleware (Advanced)

This example will show how to create a more advanced version of middleware in Go.
A middleware in itself simply takes a `http.HandlerFunc` as one of its parameters, wraps it and returns a new `http.HandlerFunc` for the server to call.
Here we define a new type `Middleware` which makes it eventually easier to chain multiple middlewares together. This idea is inspired by Mat Ryers' talk about Building APIs. You can find a more detailed explaination including the talk here.
This snippet explains in detail how a new middleware is created. In the full example below, we reduce this version by some boilerplate code.

```
func createNewMiddleware() Middleware {

    // Create a new Middleware
```

```
    middleware := func(next http.HandlerFunc) http.HandlerFunc {

        // Define the http.HandlerFunc which is called by the server eventually
        handler := func(w http.ResponseWriter, r *http.Request) {

            // ... do middleware things

            // Call the next middleware/handler in chain
            next(w, r)
        }

        // Return newly created handler
        return handler
    }

    // Return newly created middleware
    return middleware
}
```

This is the full example:

```
// advanced-middleware.go
package main

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

type Middleware func(http.HandlerFunc) http.HandlerFunc

// Logging logs all requests with its path and the time it took to process
func Logging() Middleware {

    // Create a new Middleware
    return func(f http.HandlerFunc) http.HandlerFunc {

        // Define the http.HandlerFunc
        return func(w http.ResponseWriter, r *http.Request) {
```

```
            // Do middleware things
            start := time.Now()
            defer func() { log.Println(r.URL.Path, time.Since(start)) }()

            // Call the next middleware/handler in chain
            f(w, r)
        }
    }
}

// Method ensures that url can only be requested with a specific method, else retur
func Method(m string) Middleware {

    // Create a new Middleware
    return func(f http.HandlerFunc) http.HandlerFunc {

        // Define the http.HandlerFunc
        return func(w http.ResponseWriter, r *http.Request) {

            // Do middleware things
            if r.Method != m {
                http.Error(w, http.StatusText(http.StatusBadRequest), http.StatusBa
                return
            }

            // Call the next middleware/handler in chain
            f(w, r)
        }
    }
}

// Chain applies middlewares to a http.HandlerFunc
func Chain(f http.HandlerFunc, middlewares ...Middleware) http.HandlerFunc {
    for _, m := range middlewares {
        f = m(f)
    }
    return f
}

func Hello(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "hello world")
}

func main() {
```

```
    http.HandleFunc("/", Chain(Hello, Method("GET"), Logging()))
    http.ListenAndServe(":8080", nil)
}
```

```
$ go run advanced-middleware.go
2017/02/11 00:34:53 / 0s

$ curl -s http://localhost:8080/
hello world

$ curl -s -XPOST http://localhost:8080/
Bad Request
```