Web Services
Middleware

Software Maintenance

Web Services Middleware

Web services middleware – plays a "middleman" role , like a client-server architecture where the web services application is the client and the middleware is the server

Can address things like:

- Security
- Cross-platform communications
- Messaging between different software components
- Connective glue for a greater software structure

https://www.techopedia.com/definition/30632/web-services-middleware

Web Services Middleware in GoLang

Middleware allows you to write code that will always be executed for handlers, which allows for better modularization

https://medium.com/@sathishvj/web-handlers-and-middleware-in-golang-2706c2ecfb75

Example – Typical Handler Functions

```
func h1(w http.ResponseWriter, r *http.Request) {
 fmt.Println(now() + "before")
 fmt.Println("h1")
 fmt.Println(now() + "after")
}


func h2(w http.ResponseWriter, r *http.Request) {
 fmt.Println(now() + "before")
 fmt.Println("h2")
 fmt.Println(now() + "after")
}


func main() {
 http.HandleFunc("/h1", h1)
 http.HandleFunc("/h2", h2)
}
```

Example – Adding Middleware

```
func logger(f func(http.ResponseWriter, *http.Request))
func(http.ResponseWriter, *http.Request) {
 return func(w http.ResponseWriter, r *http.Request) {
  fmt.Println(now() + "before")
  defer fmt.Println(now() + "after")


  f(w, r) // original function call
 }
}

func h3(w http.ResponseWriter, r *http.Request) {
 fmt.Println("h3")
}


func main() {
 http.HandleFunc("/h3", logger(h3)) // wrap the handler call
}
```