Aprendiendo Go

Andrea Robles

enero, 2023

Índice general

Capítulo 1

Sintaxis de Go

En esta parte se describe el resumen o síntesis del capítulo.

1.1. Asincronía y promesas en Go

La asincronía en go es distinta a diferencia de otros lenguajes. El papel de la asincronía es indicar que instruccion deberá ejecutar de manera paralela al resto de las instrucciones.

Según GPT3 go maneja de la siguiente manera la asincronía: Go maneja la asincronía a través de la utilización de Goroutines y Channels. Una Goroutine es una rutina ligera que se ejecuta en paralelo con otras Goroutines en un mismo proceso. Los Canales son estructuras de datos que permiten a las Goroutines comunicarse entre sí de manera segura y sincronizada. La combinación de Goroutines y Canales permite a los programadores escribir código asíncrono de manera fácil y clara.

Ejemplo de la asincronia en Go

```
import (
    "fmt"
    "strings"
    "time"
)
func main(){
    go miNombreLento("Andrea Selene")
}

func miNombreLento(nombre string){
    letras := strings.Split(nombre, "")
    for _, letra := range letras{
        time.Sleep(time.Second)
        fmt.Println(letra)
    }
}
```

1.2. Canales en Go

Un canal en Go sirve para tener control de las instrucciones asincronas.

Ejemplo de Canales en Go

```
package main

import (
        "fmt"
        "time"
)
func main(){
        canal1 := make(chan time.Duration)
        go bucle(canal1)
        fmt.Println("Instrucci n")
        msg := <- canal1

        fmt.Println(msg)

}
func bucle(canal1 chan time.Duration) {
        inicio := time.Now()
        for i:=0; i<=1000000000000; i++{
        }
        final := time.Now()
        canal1 <- final.Sub(inicio)
}
```

1.3. Servidores

Ejemplos de Go en WEB.

¿Qué pasa con Go en la web?

- Funciona como servidor o del lado del backend de endpoints
- Hay una función por cada endpoint que se quiera servir
- Se pueden manejar templates de html en go

Servidor en Go

```
package main

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

func main() {
        http.HandleFunc("/", home)
        http.HandleFunc("/login", login)
        http.ListenAndServe(":3000", nil)
}

func home(w http.ResponseWriter, r *http.Request){
        http.ServeFile(w, r, "./index.html")
}

func login(w http.ResponseWriter, r *http.Request){
        http.ServeFile(w, r, "./login.html")
        fmt.Fprintf(w, "Hello, %q", html.EscapeString(r.URL.
            Path))
}
```

Donde los archivos .html son los siguientes:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
        initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Hola, desde Go</h1>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
        initial-scale=1.0">
    <title>Form</title>
</head>
<body>
    <input type="text" name="name">
    <input type="submit">
</body>
</html>
```

1.4. Middlewares

Son interceptores que permiten ejecutar instrucciones comunes a varias funciones que reciben y devuleven los mismos tipos de variables. Esto es, se escribe un middleware comun a varias funciones.