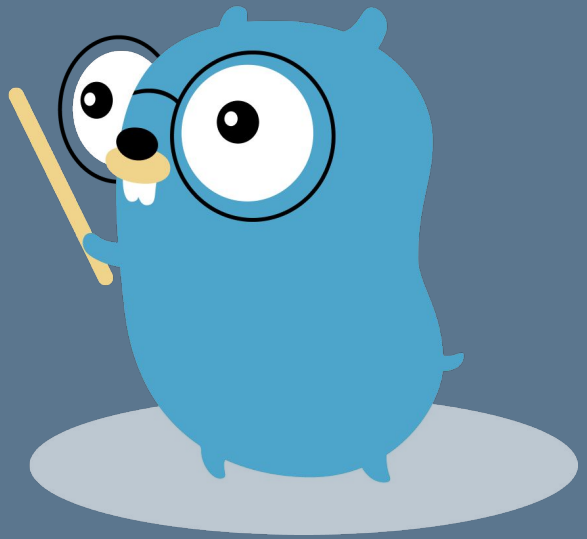
GDG Marbella

Curso de Iniciación a Go
CONCURRENCIA Y NET

01. INTRODUCCIÓN Y CONCURRENCIA



¿Quiénes Somos?

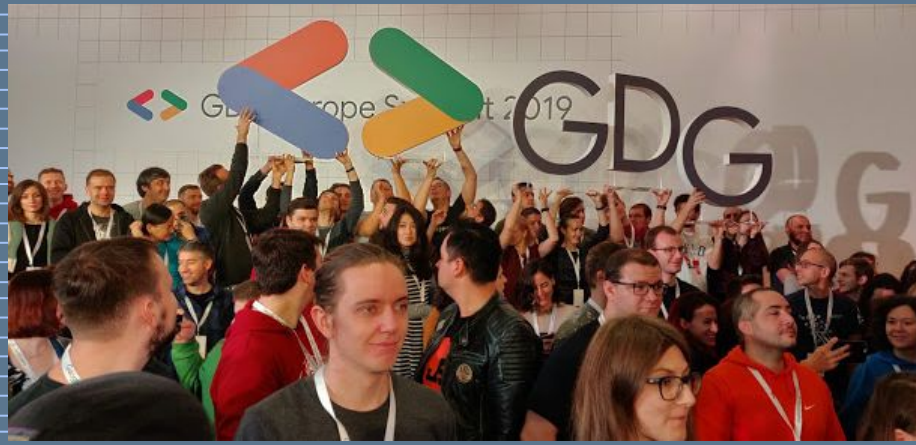
GOOGLE DEVELOPER GROUP MARBELLA

GDG es una plataforma de comunidades donde apasionados de la tecnología lideramos comunidades locales para ayudar, aprender y conectar con talento tech.

+1.000 GRUPOS

+140 PAISES

+900.000 MIEMBROS



→ Web GDG

→ Twitter: @gdgmarbella

Pablo Cumpián

FREELANCE SOFTWARE DEV

→ https://github.com/pabloos

→ https://www.linkedin.com/in/pablocdiaz/

→ https://twitter.com/PabloCDaz

Es un lenguaje imperativo

- No OPP
- No Generics... until now Go2
- Cross-compiled
- Garbage-collected
- Open Source

Principales Características

SIMPLE

RÁPIDO

Hace fácil lo difícil

Mantiene la sintaxis imprescindible

El lenguaje de Docker, Kuebernetes...

¿Qué vas a aprender?

1. Go a través de sus características: Concurrencia y programación de servicios en Red
2. Situaciones mas comunes de la programación concurrente
3. Programar servidores y clientes HTTP
4. Las posibilidades del paquete net

Let's go func

¿Qué es Concurrencia?

- Carácter asíncrono
- No hay secuencialidad
- No hay ejecución determinista
- Puede darse un único procesador

¿En qué situaciones hay que contar con ello?

- I/O
- event-driven

Concurrencia != Paralelismo

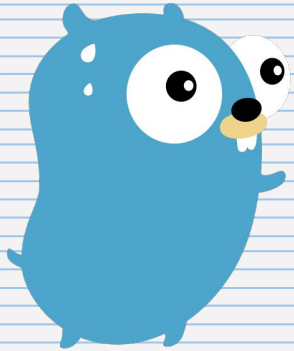
Construcción sintáctica de la concurrencia

GoRoutines

Son la pieza principal en el tratamiento de la concurrencia en Go.

Son hilos de ejecución muy ligeros invisibles al sistema operativo, ya que los trata el propio runtime de Go.

Race Condition



La condición de carrera es un escenario no deseado en la cual dos procesos o más necesitan ejecutarse en un orden pero por su naturaleza concurrente no está garantizado

WaitGroups Al Rescate!

Sincronizar goroutines es una tarea muy corriente en Go que se puede hacer de muchas formas, pero hay una especialmente sencilla y que veremos por todos sitios: **los wait groups**

Channels

La construcción principal del lenguaje para el paso de mensajes entre goroutines

Son ciudadanos de primera clase: se pueden pasar como parámetros, devolverlos... incluso transmitir canales dentro de canales!

Existen 2 tipos:

SÍNCRONOS (sin buffer)

ASÍNCRONOS (con buffer)

Deadlocks!

```
c := make(chan int)

c ← 1
one := ← c

fmt.Println(one)
```

El otro gran temor de la programación concurrente:

Cuando se produce interbloqueo entre las goroutines en el acceso a la sección crítica y esto causa que el programa no avance.

fatal error: all goroutines are asleep - deadlock!

Mutex: Uno a la vez, por favor

- Un mecanismo para evitar los problemas de exclusión mutua y los deadlocks.
- Cada goroutine bloquea el recurso cuando va a hacer uso de él y lo desbloquea al terminar. Es una forma muy intuitiva de lidiar con secciones críticas.

Context: Cancelando Goroutines

¿Cómo interrumpirlas?

```
ctx := context.Background()
```

Context no es en sí una construcción del lenguaje. Es un patrón de diseño para concurrencia

→ Veamos un ejemplo del propio blog de Go

```
func gen(ctx context.Context) <-chan int {
    dst := make(chan int)
    n := 1
    go func() {
        for {
            select {
            case <-ctx.Done():
                return // returning not to leak the
            case dst <- n:
                n++
            }
        }
    }()
```

```go
func main() {
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    for n := range gen(ctx) {
        fmt.Println(n)
        if n == 5 {
            break
        }
    }
}
```

Un pequeño ejercicio...

- Descargad el código del repositorio

1. Conseguir que se impriman todos los status del response

2. Romper el enlace de una de las URLs. Conseguir que cuando detecte el error, se cancelen el resto de peticiones que no hayan terminado

Preguntas

Nos vemos en la Próxima Sesión :)

02. HTTP

Jueves 2 Julio