Athens Gophers Meetup

Μιχάλης Τσούκαλος
@mactsouk και https://www.mtsoukalos.eu/

Concurrency in Go

Goroutines + Channels

Η ιστορία

Ο ήρωας

Κάθε ιστορία έχει έναν ήρωα και έναν κακό. Στην σημερινή ιστορία ο *Ήρωας* είναι ο Προγραμματιστής που θέλει να γράψει καλό κώδικα και *Κακός* είναι πάλι ο προγραμματιστής που θέλει να κάνει γρήγορα, να μην έχει σφάλματα, να μπορεί να διαβάσει τον κώδικά του μετά από καιρό, να τρέχουν πολλά πράγματα ταυτόχρονα, να μπορεί να γράφει εύκολα TCP/IP εφαρμογές, …

Δυστυχώς γρήγορα και καλά σπάνια γίνεται!

Το θέμα

Κάθε ιστορία όμως έχει και ένα *θέμα*. Το θέμα της ιστορίας μας είναι μια γλώσσα προγραμματισμού και το πως μπορεί να κάνει την ζωή του ήρωά μας πιο εύκολη.

Σήμερα λοιπόν θα μάθουμε κάποια από τα χαρακτηριστικά και τις δυνατότητες της Go ώστε η ιστορία να αναφέρεται περισσότερο στις επιτυχίες του ήρωα και όχι στις δυσκολίες του.

Ο κώδικας

Σε αυτή την παρουσίαση θα δείτε αρκετό αλλά απλό κώδικα σε Go. Πιστεύω ότι κάποιες φορές ο κώδικας εξηγεί καλύτερα από τα λόγια.

Ο κώδικας υπάρχει στο **GitHub**.

https://github.com/mactsouk/meetupOct2018

Part 1

Goroutines

Lightweight

Process – Thread – Goroutine

Goroutines

- create10.go
- create10sleep.go
- usingSync.go

Δεν **μπορούμε** να **προβλέψουμε** την σειρά εκτέλεσης των goroutines.

Channels

Channels

- writeCh.go
- readWriteCh.go

Panicking a channel (it is easy)

Panicking a channel

```
package main

import (
    "fmt"
    "time"
)

func writeChannel(c chan<- int, x int) {
    fmt.Println(x)
    c <- x
    close(c)
    fmt.Println(x)
}

func main() {
    c := make(chan int)
    go writeChannel(c, 10)
    close(c)
    time.Sleep(2 * time.Second)
}
```

Pipelines

Pipelines

```
func main() {
    if len(os.Args) != 3 {
        fmt.Printf("usage: %s n1 n2\n", filepath.Base(os.Args[0]))
        os.Exit(1)
    }

    n1, _ := strconv.ParseInt(os.Args[1], 10, 64)
    n2, _ := strconv.ParseInt(os.Args[2], 10, 64)

    if n1 > n2 {
        fmt.Printf("%d should be smaller than %d\n", n1, n2)
        os.Exit(10)
    }

    naturals := make(chan int64)
    squares := make(chan int64)
    go genNumbers(n1, n2, naturals)
    go findSquares(squares, naturals)
    calcSum(squares)
}
```

Pipelines

Δύο παραδείγματα:

- pipelines.go
- randomOnce.go

A concurrent TCP server in Go

concTCP.go

```
for {
    c, err := l.Accept()
    if err != nil {
        fmt.Println(err)
        return
    }
    go handleConnection(c)
}
```

```
func handleConnection(c net.Conn) {
}
```

The select statement

The select statement

Το **select** μας επιτρέπει να ακούμε πολλά channel **ταυτόχρονα** και να πράττουμε αναλόγως!

Αυτό σημαίνει ότι αν χρησιμοποιήσουμε τον κατάλληλο κώδικα, ένα **channel** δεν μπορεί να μας **μπλοκάρει**.

The select statement

(select.go)

```
func gen(min, max int, createNumber chan int, end chan bool) {
    for {
        select {
        case createNumber <- rand.Intn(max-min) + min:
        case <-end:
            close(end)
            return
        case <-time.After(4 * time.Second):
            fmt.Println("\ntime.After()!")
        }
    }
}
```

Time out goroutine (1)

timeOut1.go

```
select {
case res := <-c1:
    fmt.Println(res)
case <-time.After(time.Second * 1):
    fmt.Println("timeout c1")
    }
```

Η **time.After()** περιμένει για έναν προκαθορισμένο χρόνο. Οπότε από τα δύο **κλαδιά** του **select** θα εκτελεστεί αυτό που θα τελειώσει πρώτο.

Time out goroutine (2)

timeOut2.go

```
func timeout(w *sync.WaitGroup, t time.Duration) bool {
    temp := make(chan int)
    go func() {
        time.Sleep(5 * time.Second)
        defer close(temp)
        w.Wait()
    }()

    select {
    case <-temp:
        return false
    case <-time.After(t):
        return true
    }
}
```

Η τιμή της time.After() δίδεται σαν παράμετρος. Για να τελειώσει η goroutine χρειάζεται **w.Done()**, που βρίσκεται στην **main()**.

Part 2

Channels Part 2!

Nil channels

Οι μεταβλητές στην Go έχουν την μηδενική τιμή εκτός και αν βάλουμε κάποια άλλη τιμή εμείς. Για τα channels, αυτή η τιμή είναι η **nil**.

Close nil channel

```
$ cat closeNilChannel.go

package main

func main() {
        var c chan string
        close(c)
}
```

Nil channels

Ένα **nil channel** μπορεί να απενεργοποιήσει ένα select case.

Προσοχή όμως:

- Διάβασμα nil channel: blocks forever
- Γράψιμο σε nil channel: blocks forever
- Και όπως είδαμε: close(c) panics

Nil channel example

Ας δούμε τον κώδικα του **nilChannel.go**.

Buffered channels

Buffered channels

Ας δούμε τον κώδικα του **bufferedChannels.go**

Signal channels

Signal channels

Ας δούμε τον κώδικα του **defineOrder.go**

Channels for data sharing

Sharing data

Ας δούμε τον κώδικα του **monitor.go**

Τι λέτε λοιπόν, έχει ελπίδα ο ήρωας της ιστορίας μας;

Ωραία η θεωρία αλλά θα πρέπει να γράψετε κώδικα!

Ιδέες

Αν δεν ξέρετε τι να γράψετε, ιδού μερικές ιδέες:

- Implement existing UNIX tools: *cp(1)*, *find(1)*, *dd(1), …*
- Implement existing algorithms: sorting, binary trees, …
- Implement a Web Server
- Connect to Docker with Go
- Evaluate Sudoku puzzles in Go
- Create Sudoku puzzles in Go

Μερικές ερωτήσεις

- Ποιος είναι ο τρόπος που γράφετε κώδικα;
- Πόσες ώρες την ημέρα;
- GitHub + git?
- Γλώσσα προγραμματισμού;

Περισσότερες ερωτήσεις

- Πιστεύετε ότι γράφετε καλό κώδικα;
- Σας αρέσει η γλώσσα προγραμματισμού που χρησιμοποιείτε;
- Βελτιώνετε τον κώδικά σας;
- Κοιτάει κάποιος άλλος τον κώδικά σας;
- Εσείς κοιτάτε κώδικα από άλλους;

Τι νέα πράγματα μαθαίνετε τώρα;

“I will always remember this as the night that Michael Jordan and I **combined** for 70 points.”

Said after **Michael Jordan** scored a career high 69 points and **Stacey King** scored 1 point against the Cavaliers.

80-hour weeks
Packed schedules
Super busy
Endless meetings
Overflowing inbox
Unrealistic deadlines
Can't sleep
Sunday-afternoon emails
No time to think
Stuck at the office
All-nighters
Chats blowing up

IT DOESN'T HAVE TO BE CRAZY AT WORK

Ευχαριστώ πολύ!

Αν μπορώ να βοηθήσω σε κάτι, μην διστάσετε να επικοινωνήσετε μαζί μου.