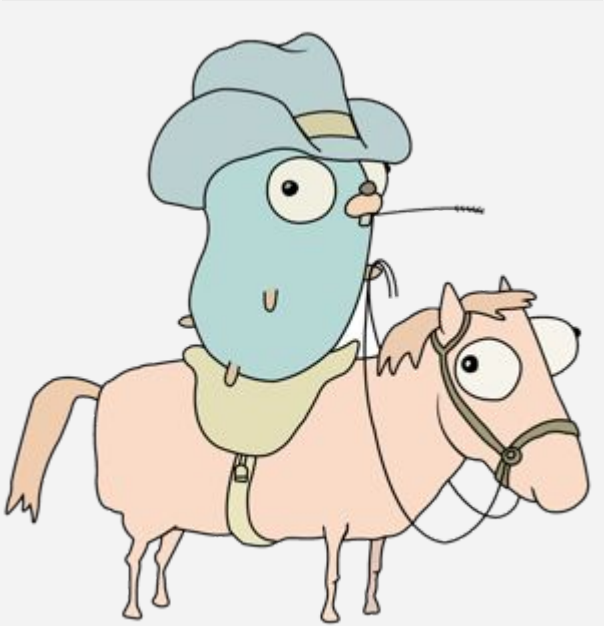

concurrence
avec go

Thèmes de l’atelier

Utilisation de WaitGroupes et mutexes

Les fonctions atomiques

Création de canaux



```
1  func DoSomethingGreat(){
2      fmt.Println("Hayyy !")
3  }
4  ...
5  go DoSomethingGreat()
6
```

allez GO !!!

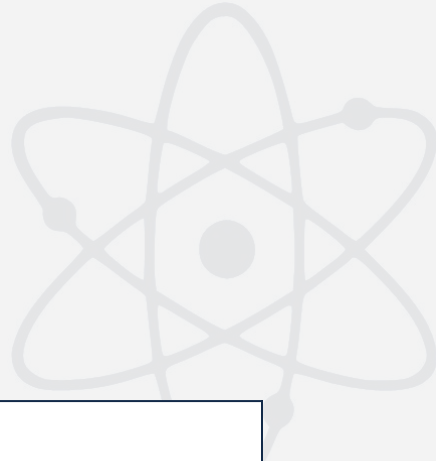
WaitGroup

```
1   var wg sync.WaitGroup
2   …
3   func DoSomethingGreat(){
4       fmt.Println("Hayyy !")
5       wg.Done()
6   }
7   …
8   wg.Add()
9   go DoSomethingGreat()
10  wg.Wait()
```

mutex

```
1  var mutex = sync.Mutex{}
2  …
3  mutex.Lock()
4  i++
5  fmt.Println(i)
6  mutex.Unlock()
```

Fonctions atomiques



```
1  var i uint64
2  …
3  atomic.AddUint64(i, 1)
4  …
5  iFinal := atomic.LoadUint64(&i)
6
```

Canal non bufferisé

```
1  msg := make(chan string)
2
3  go func(){
4     msg <- "non buffered"
5  }()
6
   fmt.Println(<-msg)
```

Canal bufferisé

```
1  msg := make(chan string, 1)
2
3  msg <- "buffered"
4
5  fmt.Println(<-msg)
6
```

Chan chan

```
1  msg := make(chan chan string)
2
3  msg <- "buffered"
4
5  fmt.Println(<-msg)
6
```

Select canal

```
1  msg := make(chan string)
2  stop := time.Tick(1 * time.Second)
3  go DoSomeThingGreat(msg)
4
5  select {
6  case <- msg :
7      fmt.Println("Yeee")
8  case <- stop :
9      fmt.Println("stop")
10     return
11 }
```

Sans concurrence
https://play.golang.org/p/DN_g-KM56r

Avec concurrence
https://play.golang.org/p/v7hHjSJ9F2

Select canal non bloquant

```
1   msg := make(chan string)
2   go DoSomeThingGreat(msg)
3
4   select {
5   case <- msg :
6       fmt.Println(“Yeee”)
7   default:
8       fmt.Println(“stop”)
9       return
10  }
11
```

Select récupération de close

```
1   msg := make(chan string, 1)
2   close(msg)
3   select {
4   case res, ok := <- msg :
5       if ok {
6           fmt.Println(res)
7       }else{
8           fmt.Println("chan is closed")
9       }
10  }
11
```

https://play.golang.org/p/Vg93cVxH_g

Pipeline

```
1  func sq(in <-chan int) <-chan int {
2        out := make(chan int)
3        go func() {
4              for n := range in {
5                    out <- n * n
6              }
7              close(out)
8        }()
9        return out
10 }
```

```
11 func gen(nums ...int) <-chan int {
12       out := make(chan int)
13       go func() {
14             for n := range nums {
15                   out <- n
16             }
17             close(out)
18       }()
19       return out
20 }
```

Pipeline

```
21  func main() {
22        // Set up the pipeline.
23        c := gen(2, 3)
24        out := sq(c)
25
26        // Consume the output.
27        fmt.Println(<-out) // 4
28        fmt.Println(<-out) // 9
29  }
30
```

https://play.golang.org/p/-ISgne9dQK

Pipeline

```
21
22
23
24
25
26
27
28
29
30
```

```go
func main() {
    for n := range sq(sq(gen(2, 3))) {
        fmt.Println(n) // 16 then 81
    }
}
```

https://play.golang.org/p/-ISgne9dQK

Paradims

https://divan.github.io/posts/go_concurrency_visualize/

Worker pool

http://marcio.io/2015/07/handling-1-million-requests-per-minute-with-golang/