ЛЕКЦИЯ 7

Waitgroup. Select. Mutex. Semaphore

Waitgroup

Группы ожидания (пакет sync):

Предназначены для реализации ожидания завершения работы горутин в последовательном коде.

```
var wg sync.WaitGroup
wg.Add(1) //Count of goroutines
go func() {
   defer wg.Done()
   //Make some
}()
wg.Wait()
```

Select

Выбор:

В Go select позволяет ожидать выполнения многоканальных операций. Сочетание горутин и каналов с оператором select является сильной стороной Go.

Если несколько каналов готовы, то select выберет случайным образом.

Select

```
c1 := make(chan string)
c2 := make(chan string)

go func() {
    time.Sleep(time.Second * 1)
    c1 ← "one"
}()
go func() {
    time.Sleep(time.Second * 2)
    c2 ← "two"
}()

for i := 0; i < 2; i++ {
    select {
    case msg1 := ←c1:
        fmt.Println("received", msg1)
    case msg2 := ←c2:
        fmt.Println("received", msg2)
    }
}
```

Mutex

Мьютексы (пакет sync):

Предназначены для организации конкурентного доступа к общим данным.
Позволяют блокировать чтение/запись переменной.

Есть два типа мьютексов:

- Mutex
- RWMutex

Mutex

Mutex:

Простой mutex осуществляет блокировку и при чтении и при записи.

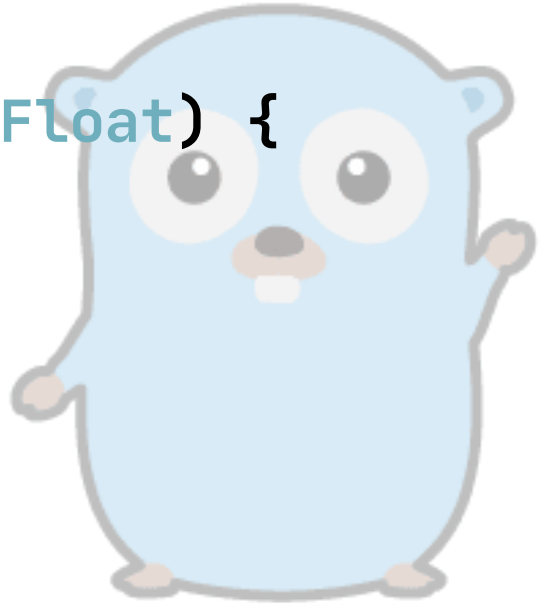
RWMutex:

RWMutex имеет тот же функционал, что и обычный Mutex, но может осуществлять блокировку только при записи, не блокируя читателей.

Mutex

```
type TotalArea struct {
    sync.Mutex
    Float *big.Float
}

func (r *TotalArea) AddArea(area *big.Float) {
    defer r.Unlock()
    r.Lock()
    r.Float.Add(r.Float, area)
}
```



Semaphore

Семафор (пакет sync/semaphore):

Предназначен для лимитирования чего – либо, например числа горутин, операций.

```
sem := semaphore.NewWeighted(capacity)
team.Arrive()
if sem.TryAcquire(team.Size()) {
   team.Swim()
   sem.Release(team.Size())
} else {
   // Let's go cinema instead
}
```