




# Golang Developer. Professional

[otus.ru](https://otus.ru)

 Проверить, идет ли запись

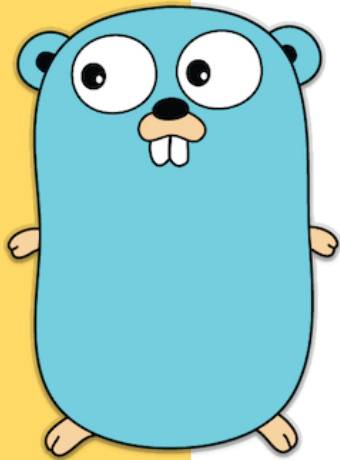
# Меня хорошо видно && слышно?

 Ставим “+”, если все хорошо  
“-”, если есть проблемы

Тема вебинара

# Примитивы синхронизации в деталях

Юрий Рубаха



# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

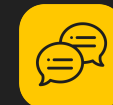
## Условные обозначения



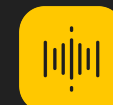
Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или  
задайте вопрос

# О чем будем говорить

- WaitGroup
- Once
- Mutex
- Race-детектор

# Вопросы для самопроверки

1. Какова цель `sync.WaitGroup` в Go и как она помогает управлять параллелизмом в программах?
2. Какова роль структуры `sync.Once` в Go и как она гарантирует, что определенная функция будет выполнена только один раз за время существования программы?
3. Для чего используется мьютекс в модели параллелизма Go.
4. Какие потенциальные проблемы могут возникнуть, если мьютексы используются неправильно?
5. Какова цель `race detector` в Go?

# sync.WaitGroup: какую проблему решает?

Что выведет эта программа?

```
func main() {  
    for i := 0; i < 5; i++ {  
        go func() {  
            fmt.Println("go-go-go")  
        }()  
    }  
}
```

<https://go.dev/play/p/Wa1dCpA-3Fg>

<https://goplay.tools/snippet/KY1h-xjHYuO>

# sync.WaitGroup: какую проблему решает?

```
func main() {  
    const goCount = 5  
  
    ch := make(chan struct{})  
    for i := 0; i < goCount; i++ {  
        go func() {  
            fmt.Println("go-go-go")  
            ch <- struct{}{}  
        }()  
    }  
  
    for i := 0; i < goCount; i++ {  
        <-ch  
    }  
}
```

<https://go.dev/play/p/QVAtc1JxbGF>

[https://goplay.tools/snippet/O0C7h\\_IsWI8](https://goplay.tools/snippet/O0C7h_IsWI8)



# sync.WaitGroup: ожидание горутин

```
func main() {  
    const goCount = 5  
  
    wg := sync.WaitGroup{}  
    wg.Add(goCount) // <===  
  
    for i := 0; i < goCount; i++ {  
        go func() {  
            fmt.Println("go-go-go")  
            wg.Done() // <===  
        }()  
    }  
  
    wg.Wait() // <===  
}
```

<https://go.dev/play/p/cNlkQJkQuFg>

[https://goplay.tools/snippet/u90fGD8vZ\\_X](https://goplay.tools/snippet/u90fGD8vZ_X)

# sync.WaitGroup: ожидание горутин

```
func main() {  
    wg := sync.WaitGroup{}  
  
    for i := 0; i < 5; i++ {  
        wg.Add(1) // <===  
        go func() {  
            defer wg.Done() // <===  
            fmt.Println("go-go-go")  
        }()  
    }  
  
    wg.Wait()  
}
```

# sync.WaitGroup: API

```
type WaitGroup struct {  
}  
  
func (wg *WaitGroup) Add(delta int) – увеличивает счетчик WaitGroup.  
func (wg *WaitGroup) Done() – уменьшает счетчик на 1.  
func (wg *WaitGroup) Wait() – блокируется, пока счетчик WaitGroup не обнулится.
```

# sync.WaitGroup: практика

```
type task struct {  
    name string  
    sleep time.Duration  
}
```

```
func doJob(t task) {  
    fmt.Printf("task %q begin\n", t.name)  
    time.Sleep(t.sleep)  
    fmt.Printf("task %q end\n", t.name)  
}
```

```
// Сделать так, чтобы задачи выполнялись конкурентно.  
// Дождаться выполнения всех задач.  
for _, t := range tasks {  
    doJob(t)  
}
```

<https://go.dev/play/p/BnGpsftx3zo>

<https://goplay.tools/snippet/BnGpsftx3zo>

# sync.Once: какую проблему решает?

```
func main() {  
    var once sync.Once  
    onceBody := func() {  
        fmt.Println("Only once")  
    }  
  
    var wg sync.WaitGroup  
    for i := 0; i < 10; i++ {  
        wg.Add(1)  
        go func() {  
            once.Do(onceBody)  
            wg.Done()  
        }()  
    }  
    wg.Wait()  
}
```

<https://go.dev/play/p/BssuldPjHAJ>

<https://goplay.tools/snippet/VxMyPmXHPzq>

## sync.Once: ленивая инициализация (пример)

```
type List struct {  
    once sync.Once  
    ...  
}  
  
func (l *List) PushFront(v interface{}) {  
    l.init()  
    ...  
}  
  
func (l *List) init() {  
    l.once.Do(func() {  
        ...  
    })  
}
```

## sync.Once: синглтон (пример)

```
type singleton struct {  
}  
  
var instance *singleton  
var once sync.Once  
  
func GetInstance() *singleton {  
    once.Do(func() {  
        instance = &singleton{}  
    })  
    return instance  
}
```

# sync.Mutex: какую проблему решает?

```
func main() {  
    wg := sync.WaitGroup{}  
  
    v := 0  
    for i := 0; i < 1000; i++ {  
        wg.Add(1)  
        go func() {  
            v++  
            wg.Done()  
        }()  
    }  
  
    wg.Wait()  
    fmt.Println(v)  
}
```

<https://go.dev/play/p/mT7vUcSGEhh>

<https://goplay.tools/snippet/bf6NKB5z0QQ>



# sync.Mutex

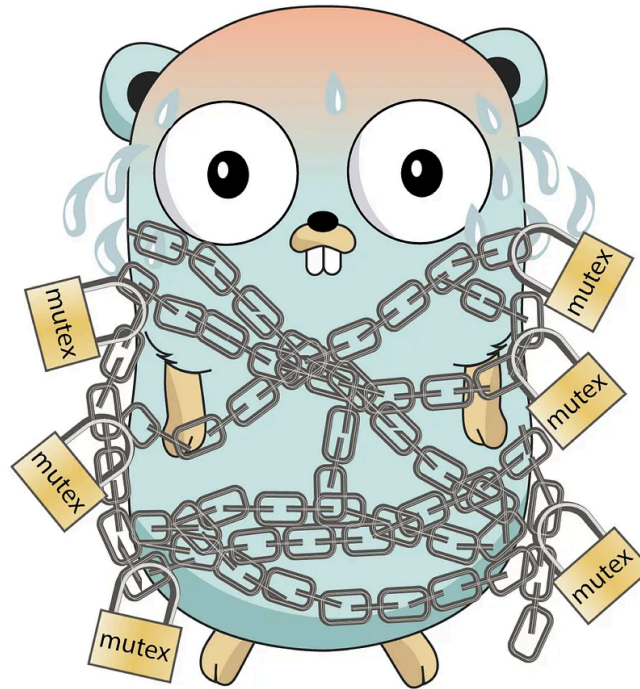
```
$ GOMAXPROCS=1 go run mu.go  
1000  
$ GOMAXPROCS=4 go run mu.go  
947  
$ GOMAXPROCS=4 go run mu.go  
956
```

# sync.Mutex

```
func main() {  
    wg := sync.WaitGroup{}  
  
    v := 0  
    for i := 0; i < 1000; i++ {  
        wg.Add(1)  
        go func() {  
            old_v := v  
            new_v := old_v + 1  
            v = new_v  
            wg.Done()  
        }()  
    }  
  
    wg.Wait()  
    fmt.Println(v)  
}
```

# sync.Mutex

Мьютекс (англ. mutex, от mutual exclusion — «взаимное исключение»).



Код между Lock и Unlock выполняет только одна горутина, остальные ждут:

```
mutex.Lock()  
v++  
mutex.Unlock()
```



# sync.Mutex

```
func main() {  
    wg := sync.WaitGroup{}  
    mu := sync.Mutex{}  
  
    v := 0  
    for i := 0; i < 1000; i++ {  
        wg.Add(1)  
        go func() {  
            mu.Lock()    // <===  
            v++  
            mu.Unlock() // <===  
            wg.Done()  
        }()  
    }  
  
    wg.Wait()  
    fmt.Println(v)  
}
```

# sync.Mutex: паттерны использования

Помещайте мьютекс выше тех полей, доступ к которым он будет защищать

```
var sum struct {  
    mu sync.Mutex // <=== этот мьютекс защищает  
    i  int        // <=== поле под ним  
}
```

# sync.Mutex: паттерны использования

Используйте defer, если есть несколько точек выхода

```
func doSomething() {  
    mu.Lock()  
    defer mu.Unlock()  
  
    err := ...  
    if err != nil {  
        return // <===  
    }  
  
    err = ...  
    if err != nil {  
        return // <===  
    }  
    return // <===  
}
```

# sync.Mutex: паттерны использования

НО!

Держите блокировку не дольше, чем требуется

```
func doSomething(){  
    mu.Lock()  
    item := cache["myKey"]  
    mu.Unlock()  
  
    http.Get() // дорогой IO-вызов  
}
```

# sync.Mutex: дедлок



Что такое дедлок?

<https://go.dev/play/p/PLLvZfDiDqs>

<https://goplay.tools/snippet/PLLvZfDiDqs>



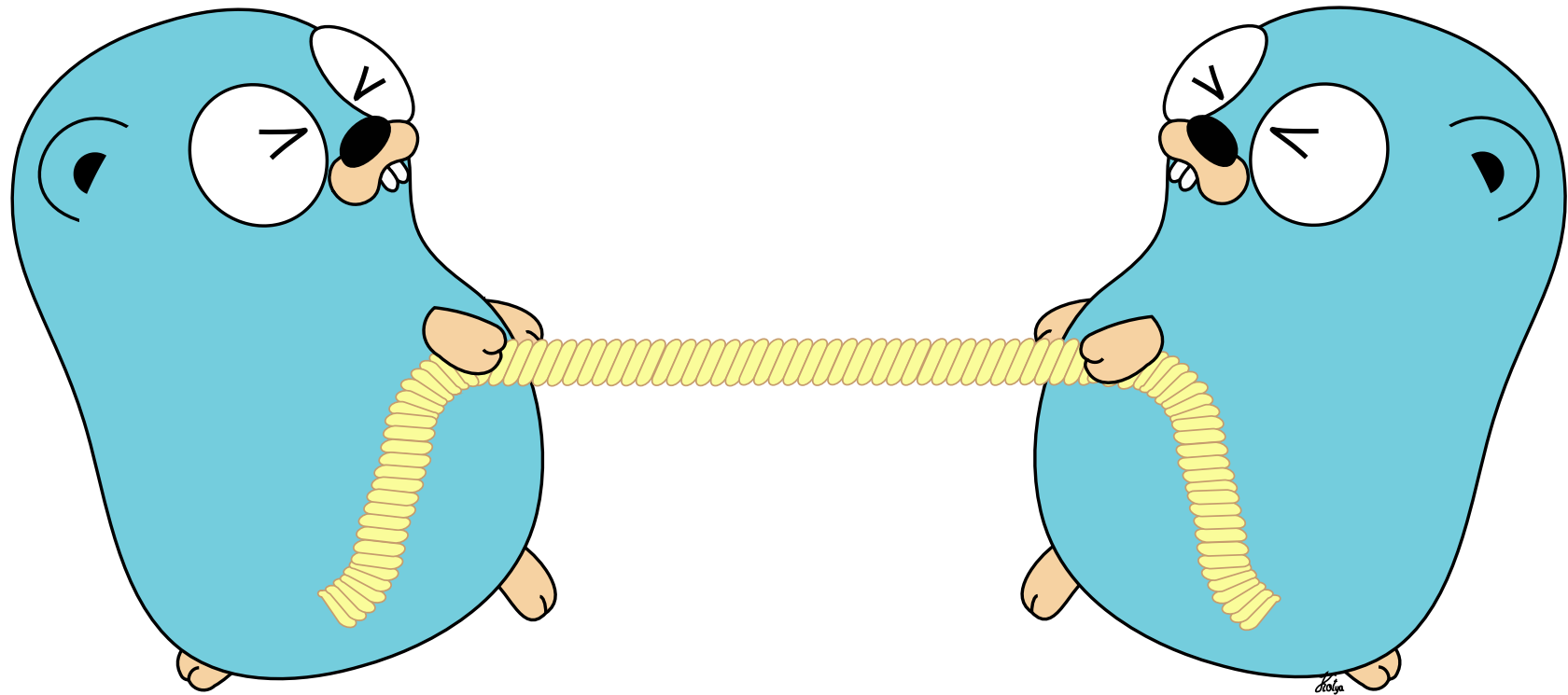


# Race detector

В чем проблема, кроме неопределенного поведения?

```
func main() {  
    wg := sync.WaitGroup{}  
    text := ""  
  
    wg.Add(2)  
  
    go func() {  
        text = "hello world"  
        wg.Done()  
    }()  
  
    go func() {  
        fmt.Println(text)  
        wg.Done()  
    }()  
  
    wg.Wait()  
}
```

# Race detector



```
$ go test -race mypkg  
$ go run -race mysrc.go  
$ go build -race mycmd  
$ go install -race mypkg
```

<https://blog.golang.org/race-detector>

<http://robertknight.github.io/talks/golang-race-detector.html>



# Race detector

Ограничение race детектора:

```
func main() {  
    for i := 0; i < 10000; i++ {  
        go func() {  
            time.Sleep(10*time.Second)  
        }()  
    }  
    time.Sleep(time.Second)  
}
```

Можно исключить тесты:

```
// +build !race  
  
package foo  
  
// The test contains a data race. See issue 123.  
func TestFoo(t *testing.T) {  
    // ...  
}
```

# Вопросы для самопроверки

1. Какова цель `sync.WaitGroup` в Go и как она помогает управлять параллелизмом в программах?
2. Какова роль структуры `sync.Once` в Go и как она гарантирует, что определенная функция будет выполнена только один раз за время существования программы?
3. Для чего используется мьютекс в модели параллелизма Go.
4. Какие потенциальные проблемы могут возникнуть, если мьютексы используются неправильно?
5. Какова цель `race detector` в Go?

# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет

**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**

# Следующий вебинар

Дополнительные примитивы синхронизации



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию в ЛК — можно изучать

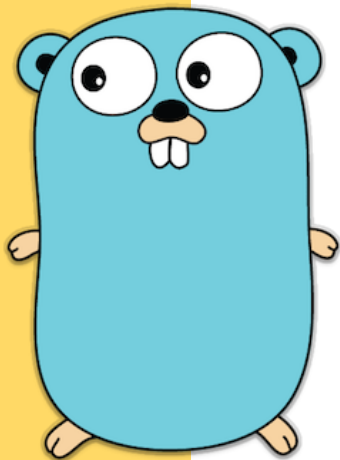


Обязательный материал обозначен красной лентой



Спасибо за внимание!

# Приходите на следующие вебинары





Дополнительная практика

<https://goplay.tools/snippet/FqCxyxywUW2>

<https://go.dev/play/p/FqCxyxywUW2>