



ОНЛАЙН-ОБРАЗОВАНИЕ


Онлайн-образование

Проверить, идет ли запись!





Меня хорошо видно && слышно?

Ставьте  , если все хорошо
Напишите в чат, если есть проблемы

Модуль 1. Начало работы с Go

Тема 8. Горутины и каналы



Елена Граховац

elena@grahovac.me
twitter.com/webdeva

Преподаватель



Елена Граховац

- 12 лет в веб-разработке
- 5 лет с Go
- техлид команды автоматизации внутренних процессов
- соведущая подкаста GolangShow
- руководитель ПК конференции GolangConf

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в Slack #go-2019-08 или #general



Вопросы вижу в чате, могу ответить не сразу

Опрос

<https://forms.gle/ypkdg1i1s63o3a1i9>



Цели вебинара | После занятия вы сможете

1

Начать работу с горутинами

2

Реализовать простую синхронизацию и передачу данных между горутинами с помощью каналов

3

Слушать сигналы операционной системы из приложения

Смысл | Зачем вам это уметь

1

Go спроектирован для работы с конкурентностью, и навыки работы с горутинами позволяют использовать этот язык на полную мощность

2

Горутины и каналы реализуют простые решения для многих часто встречающихся задач

3

Запуск и остановка независимых частей приложения, реализация graceful shutdown

The background of the image is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. The image is divided into three horizontal sections. The top and bottom sections show the city buildings. The middle section is a solid blue band with a white, glowing network pattern of dots and lines. The word "Горутины" is centered in this band in a large, white, sans-serif font.

Горутины

Одновременность

- Как сделать так, чтобы программа “одновременно” выполняла разные задачи?
- Пример: хочу запустить в одном приложении два веб-сервера или веб-сервер и ещё какую-то фоновую задачу
- Это можно сделать с помощью горутин

Горутины

- Реализуют механизм “одновременного” выполнения двух разных задач в одной программе
- Горутины - “легковесные треды” (в противопоставление “тяжелым” тредам ОС)
- Go - за простоту: у горутин “нет” идентификаторов, чтобы ими не злоупотребляли
- Не всегда очевидно, в какой момент горутина будет исполнена
- Новые горутины создаются с помощью конструкции `go`
- `go f()` - вызвать функцию `f` из горутины

Не всегда “параллелизм”

- Горутины не всегда работают параллельно, т.е. на разных процессорах
- В общем случае мы говорим, что горутины “конкурируют” за ресурсы процессора

Планировщик

- За управление горутинами отвечает Планировщик (Scheduler Go)
- Планировщик распределяет горутин на несколько тредов ОС, которые могут быть запущены на одном или нескольких процессорах
- m:n-планирование: как выполнить m горутин на n тредрах (потоках) ОС
- Планировщик вызывается во время выполнения некоторых конструкций (например: блокировка горутин операциями с каналами или `time.Sleep`)
- GOMAXPROCS: задать параметр n в m:n-планировании
- Будьте аккуратны!

Сколько горутин?

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello, world!")
}
```

- Напишите в чат, сколько горутин в этом приложении. Варианты ответа: 0, 1, 2, 10.

Сколько горутин?

```
package main
```

```
import (  
    "fmt"  
    "runtime"  
)
```

```
func main() {  
    fmt.Printf(  
        "Goroutines: %d",  
        runtime.NumGoroutine(),  
    )  
}
```

- Проверяем:

<https://play.golang.org/p/TK1zEBfJAAL>

Что напечатает эта программа?

```
package main

import (
    "fmt"
)

func main() {
    go fmt.Printf("Hello")
}
```

- Напишите в чат
- Проверяем:
<https://play.golang.org/p/fQTxSZUFnDj>
- Если главная горутина завершена, программа завершается
- Главная горутина не ждёт завершения других горутин

The image features a blue-tinted aerial view of a city skyline, likely New York City, with numerous skyscrapers. A semi-transparent network pattern of white lines and dots is overlaid on the image, particularly concentrated in the center where the text is located. The word "Каналы" is written in a large, white, sans-serif font.

Каналы

Каналы

- Каналы позволяют производить синхронизацию и обмен данными между горутинами
- Работают по принципу FIFO (как очереди)
- Работают со значениями определенного типа (любого - целые, структуры, интерфейсы, каналы и т. д.)
- Под капотом - указатель
- Нулевое значение - nil

Каналы

- Однонаправленные (только чтение, только запись), двунаправленные
- Буферизованные, небуферизованные
- Открытые, закрытые

Операции с каналами

- Создание канала: `ch = make(chan int)` или `ch = make(chan int, 3)`
- Отправление в канал: `ch <- x`
- Получение из канала: `x = <-ch` или `x, ok := <-ch`
- Закрытие канала: `close(ch)`

Небуферизованные каналы

```
package main

import (
    "fmt"
)

func main() {
    var ch = make(chan struct{})

    go func() {
        fmt.Printf("Hello")
        ch <- struct{}{}
    }()

    <-ch
}
```

- Отправление блокируется до тех пор, пока не будет выполнено получение
- Получение блокируется до тех пор, пока не выполнится отправление
- Проверяем:
<https://play.golang.org/p/XFSYdyzqZ1C>

Заккрытие канала

- Заккрытие канала: `close(ch)`
- Чтение из закрытого канала вернёт нулевое значение
- Запись в закрытый канал вызовет панику (!!!)
- Следовательно, хотелось бы, чтобы закрывал канал тот, кто в него пишет
- ...Если “писателей” несколько - тот, кто создал канал
- Примеры: <https://play.golang.org/p/DmQr2WBJTMB>,
<https://play.golang.org/p/s3p9rQmCT68>

Заккрытие канала

```
package main

import (
    "fmt"
)

func main() {
    var ch = make(chan struct{})

    go func() {
        fmt.Printf("Hello")
        ch <- struct{}{}
    }()

    <-ch
}
```

- Заккрытие канала можно использовать для синхронизации
- Мини-задача: переписать пример слева так, чтобы вместо записи в канал для синхронизации использовалось закрытие канала

<https://play.golang.org/p/XFSYdyzqZ1C>

- Проверяем:

https://play.golang.org/p/aKf-th_Geqz

Однонаправленные каналы

- Функция с каналом только для записи: `f(out chan<- int)` - стрелочка “в канал”
- Функция с каналом только для чтения: `f(in <-chan int)` - стрелочка “из канала”
- Что будет, если писать в канал только на чтение?
- Что будет, если читать из канала только на запись?
- Смотрим пример: <https://play.golang.org/p/t6bVfgg6BTu>
- Да, будет ошибка во время компиляции :)

Буферизованные каналы

- Максимальная “емкость” канала задается при вызове `make`: `make(chan int, 3)`
- Отправка в канал блокируется только если канал заполнен
- Чтение из канала блокируется только если канал пуст
- Посмотреть “емкость” канала: `cap(ch)`
- Посмотреть кол-во элементов в буфере канала: `len(ch)`
- Осторожно! Скорее всего, `len(ch)` сразу же устареет!

Читаем из канала, пока он не закрыт

```
for {  
    x, ok := <-ch  
    fmt.Println(x)
```

```
    if !ok {  
        break  
    }  
}
```

```
for x := range ch {  
    fmt.Println(x)  
}
```


Мультиплексирование (select)

```
select {  
  case <-ch1:  
    // ...  
  
  case ch2 <- y:  
    // ...  
  
  default:  
    // ....  
}
```

- Select ждет, когда один из каналов будет готов на чтение / запись
- Если никто не готов, срабатывает default (если default объявлен)
- Как вы думаете, как бы сработал `select{}` (select вообще без параметров)? Напишите в чат
- (Ответ): `select{}` ждет вечно

The background of the slide is a high-angle, blue-tinted aerial photograph of a city, likely New York City, showing a dense cluster of skyscrapers. A semi-transparent blue band with a white geometric network pattern of dots and lines runs horizontally across the middle of the image, serving as a backdrop for the title.

Практика использования каналов

Послать сигнал сразу нескольким горутинам

- Как послать сигнал сразу нескольким горутинам? (Напишите в чат)
- (Ответ): Закрывать канал, из которого они читают

```
var start = make(chan struct{})

for i := 0; i < 10000; i++ {

    go func() {
        <- start

        // горутин не начнут работу,
        // пока не будут созданы все 10000
    }()

}

close(start)
```


Таймеры и тикеры

Таймер: срабатывает через указанное время

```
timer := time.NewTimer(10 * time.Second)

select {

case data := <-ch:
    fmt.Printf("received: %v", data)

case <-timer.C:
    fmt.Printf("failed to receive in 10s")

}
```

Тикер: срабатывает раз в заданное кол-во времени

```
ticker := time.NewTicker(10*time.Second)

for {
    select {
        case <- ticker.C:
            fmt.Println("do something")
    }
}
```

Graceful shutdown

```
interrupt := make(chan os.Signal, 1)
signal.Notify(interrupt, os.Interrupt, syscall.SIGTERM)

fmt.Printf("Got %v...\n", <-interrupt)

// здесь выключаем веб-сервер
// и любые другие обработчики задач
```

Graceful shutdown

- Пробуем на практике: <https://play.golang.org/p/ROnMBbgiwDP>
- Заработает на Linux / Mac
- Из Go Playground не заработает!
- Задача: скомпилировать и запустить бинарник с кодом по ссылке, отправить бинарнику сигналы 2 и 16.

Замыкание

- Что выведет эта программа?
- Проверяем:
<https://play.golang.org/p/I0c7heYneBy>
- Как переписать программу так, чтобы она вывела 01234?
- Проверяем:
<https://play.golang.org/p/UiD1c-kWQaY>

```
func main() {  
    for i := 0; i < 5; i++ {  
        go func() {  
            fmt.Print(i)  
        }()  
    }  
  
    time.Sleep(10 * time.Second)  
}
```


The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. This diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the slide. Centered within this band is the word "Обсуждение" in a large, white, sans-serif font.

Обсуждение

Какие каналы использовать

- Когда использовать небуферизованные каналы?
- Когда использовать буферизованные каналы?
- Как выбрать буфер?

Домашнее задание

Написать функцию для параллельного выполнения N заданий (т.е. в N параллельных горутинах).

Функция принимает на вход:

- слайс с заданиями `[]func() error`;`
- число заданий которые можно выполнять параллельно (``N``);
- максимальное число ошибок после которого нужно приостановить обработку.

Учесть, что задания могут выполняться разное время.



Срок: 30 сентября

О чем еще можно подумать?

- Можно ли сравнивать каналы через `==` ?
- Когда `ch1 == ch2` ?

Рефлексия



Узнали что-то новое? Что кажется особенно полезным?
Напишите в чат!

Следующий вебинар

Тема:



Примитивы синхронизации



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию
в ЛК — можно изучать



Обязательный
материал обозначен
красной лентой


Список материалов для изучения

- Visualizing Concurrency in Go: <https://youtu.be/QNY2QcmxVJQ>
- Отличная статья про планировщик: <https://rakyll.org/scheduler/>
- Сборник лучших статей по конкурентности:
<https://github.com/golang/go/wiki/LearnConcurrency>
- Книга про конкурентность в Go:
<https://www.oreilly.com/library/view/concurrency-in-go/9781491941294/>

Эти слайды доступны сразу



<https://clck.ru/JHCzG>

The background of the image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer. On the left side of this layer, there is a network diagram consisting of white dots connected by thin white lines, forming a web-like structure. The text is centered on the right side of the blue layer.

Заполните, пожалуйста,
опрос о занятии:
<https://otus.ru/polls/4898/>

Спасибо за внимание!
Приходите на следующие вебинары



Елена Граховац

elena@grahovac.me
twitter.com/webdeva