Язык программирования Go

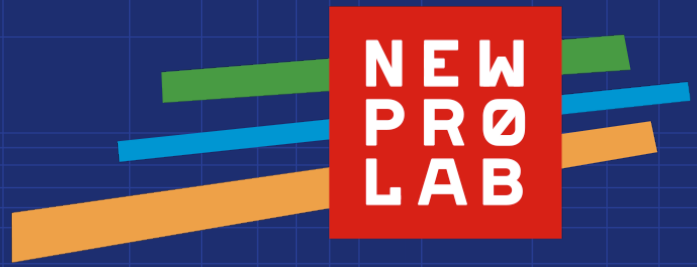
Николай Марков (@enchantner)

Что такое Go?

- Компилируемый язык программирования со строгой статической типизацией
- Акцент на многопоточности
- Поддержка указателей, но не арифметики с ними
- Garbage Collector
- Упрощенный синтаксис
- Встроенные средства форматирования и расширенного анализа кода на этапе компиляции

Чем Go НЕ является?

- Go - не замена Python или C++
- В текущем виде не является языком общего назначения
- Не обладает развитым набором инструментов для анализа данных
- Хуже, чем, например, Python, умеет интегрироваться с низкоуровневыми библиотеками
- На текущий момент не обладает хорошей инфраструктурой пакетов и менеджером. Есть разве что “официальный эксперимент”
- Не поддерживает классическую работу с исключениями в виде блоков



Окружение и основы синтаксиса

Области видимости

Две переменные окружения - GOROOT и GOPATH. Первая всегда указывает на каталог с самим Go, вторая должна указывать на текущий каталог с проектами. Исходники проектов должны лежать в подкаталоге **src**.

Можно использовать утилиту **direnv** для автоматического выставления GOPATH в консоли.

```
export GOPATH=$(pwd)
export PATH=$GOPATH/bin:$PATH
```

https://github.com/direnv/direnv

Добавление проектов в GOPATH

~$ go get -u github.com/golang/dep/cmd/dep

Да, импорты в Go могут и будут начинаться с “github.com". Да, прямо в исходниках. Альтернативно можно использовать “gopkg.in" (туда кладутся стабильные релизы пакетов по тегам), почитать можно тут - http://labix.org/gopkg.in

Писать код удобнее всего в GoLand (бывший Gogland, https://www.jetbrains.com/go/) или LiteIDE (https://github.com/visualfc/liteide)

Hello, World!

```
package main

import "fmt"

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

Пример модуля

```go
package cache

import (
    "gopkg.in/redis.v3"
    "myproject/conf"
)

var client *redis.Client

func InitCache(config *conf.Config) {
    client = redis.NewClient(&redis.Options{
        Network:  "tcp",
        Addr:     config.Redis.Host,
        DB:       int64(config.Redis.CacheDB),
    })
}
```

```go
func Set(key string, val string) {
    err := client.Set(key, val, 0).Err()
    if err != nil {
        panic(err)
    }
}

func Get(key string) (string, bool) {
    val, err := client.Get(key).Result()
    if err == redis.Nil {
        return "", false
    } else if err != nil {
        panic(err)
    }
    return val, true
}
```

Управляем зависимостями

~$ dep init

Создает два файла - **Gopkg.toml** и **Gopkg.lock** (аналог **requirements.txt** в Python)

~$ dep ensure

Создает папку **vendor** (аналог **virtualenv/node_modules**), сканирует импорты проекта и ставит туда все зависимости (аналог **pip install -r**)

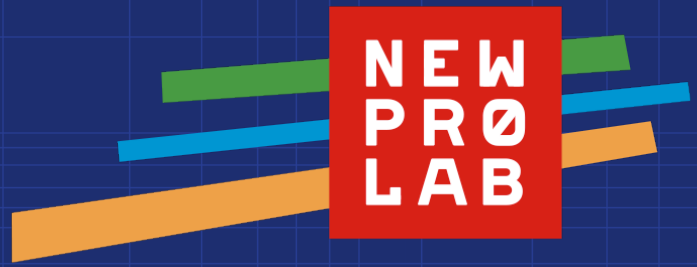
Запуск и сборка

~$ go run main.go

Быстро скомпилить и сразу же запустить конкретный файл. Нет, Go не интерпретируемый, он просто быстро выполняется.

~$ go build

Собрать проект и сформировать в текущей папке бинарник с тем же именем. Зависимости в новых версиях Go могут подтягиваться из папки **vendor**.



Базовые типы и control flow

Типизация

Два способа объявления переменной:

```
var a int = 1
b := 2
```

Конвертация типов, как и в Python, делается обертыванием переменной в соответствующий тип, например, float64(int_value)

```
bool

string

int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64 uintptr

byte // то же, что uint8

rune // то же, что int32
     // представляет Unicode code point

float32 float64

complex64 complex128
```

Базовые контейнеры

Массивы и слайсы(срезы):

```
arr1 := [5]int{}
arr2 := [3]int{1, 2, 3}

slc1 := []int{}
slc2 := []int{1, 2, 3}
```

```
lst := [4]int{1, 2, 3, 4}
var lst2 []int = lst[1:3]
lst2[1] = 5
```

```
copy(dst, src)
```

Мапы (словари, карты):

```
var m map[string]int
var m = map[string]int{"foo": 1, "bar": 2}
```

Если размер не ясен из декларации, надо явно аллоцировать память с помощью make(), либо использовать type inference:

```
var s []byte
s = make([]byte, 5)
// s == []byte{0, 0, 0, 0, 0}

m := make(map[string]int)
```

https://blog.golang.org/go-slices-usage-and-internals

https://blog.golang.org/go-maps-in-action

Базовые контейнеры

Работа со слайсами и мапами во многом похожа на то, что есть в Python:

```
a := make([]int, 1)
// a == []int{0}
a = append(a, 1, 2, 3)
// a == []int{0, 1, 2, 3}
```

```
m[“foo"] = 777   // создаем запись в мапе
i := m["foo"]   // присваиваем значение из мапа
delete(m, "foo")   // удаляем из мапа
i2, ok := m["foo"]   // пытаемся достать, если получится
```

https://gobyexample.com/slices https://gobyexample.com/maps

Базовые контейнеры

Проверка наличия ключа в мапе:

```
if val, ok := cache["foo"]; ok {
    // сделать что-то полезное
}
```

Аналог **enumerate()** из Python:

```
for i, value := range x {
    // сделать что-то полезное
}
```

Классика циклов:

```
for i := 0; i < len(x); i++ {
    // сделать что-то полезное
}
```

Switch/case

```go
package main

import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Print("Go runs on ")
    switch os := runtime.GOOS; os {
    case "darwin":
        fmt.Println("OS X.")
    case "linux":
        fmt.Println("Linux.")
    default:
        // freebsd, openbsd,
        // plan9, windows...
        fmt.Printf("%s.", os)
    }
}
```

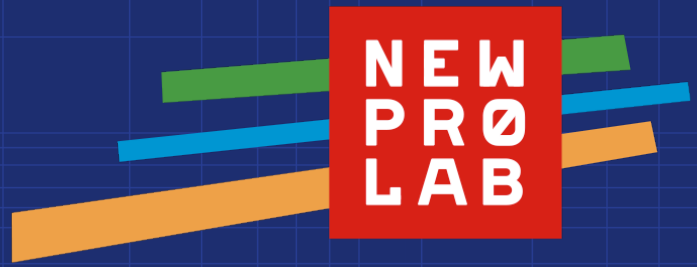
Функции

Приватность функции для модуля определяется тем, названа она с большой буквы или с маленькой.

```
func Get(key string) (string, bool) {
   val, err := client.Get(key).Result()
   if err == redis.Nil {
      return "", false
   } else if err != nil {
      panic(err)
   }
   return val, true
}
```

Упражнение

Напишите старую добрую функцию для вычисления n-го числа Фибоначчи



Структуры

Декларация и создание

Структуры синтаксически выглядят очень похоже на то, что есть в языке C:

```
type _3DCoord struct {
    x float64
    y float64
    z float64
}
```

Создать экземпляр можно разными способами:

```
var d1 _3DCoord
d2 := new(_3DCoord) // вернет указатель
d3 := _3DCoord{x: 0, y: 0, z: 5}
d4 := _3DCoord{0, 0, 5}
d5 := &_3DCoord{0, 0, 5} // вернет указатель
```

Методы структуры

```
func (c *_3DCoord) moveUp(step float64) {
    c.z += step
}
```

“Инкапсуляция”, “Наследование”, “Полиморфизм”? А что это такое?

Ладно, есть подобие “наследования” путем включения одной структуры в другую:

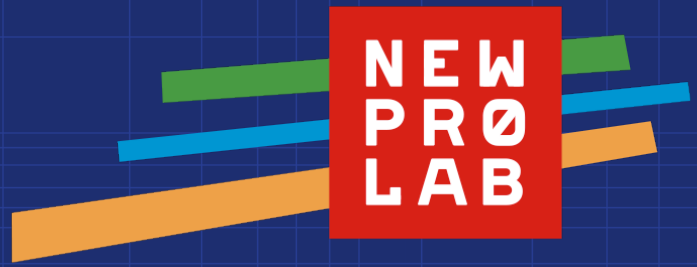
```
type ChildStruct struct {
    ParentStruct
    Model string
}
a.ParentStruct.DoSmth()
a.DoSmth()
```

Пример конфига с YAML-тегами

```
type Config struct {
    FromKafka struct {
        GroupID string `yaml:"group_id"`
        SessionTimeout int `yaml:"session_timeout"`
        MaxReadMessages uint32 `yaml:"max_read_messages"`
        Topics []string `yaml:"topics"`
        Brokers []string `yaml:"brokers"`
    } `yaml:"from_kafka"`
    ToKafka struct {
        MessageMaxBytes int `yaml:"message_max_bytes"`
        Topics []string `yaml:"topics"`
        Brokers []string `yaml:"brokers"`
    } `yaml:"to_kafka"`
    Daemon struct {
        Parsers map[string]int  `yaml:"parsers"`
        Uploaders map[string]int  `yaml:"uploaders"`
    } `yaml:"daemon"`
}
```

Парсинг конфига в структуру

```
var Conf Config

func ReadConfig(config_path string) {
    configFile, err := ioutil.ReadFile(config_path)
    if err != nil {
        log.Fatal("Failed to read config file '", config_path, "'! Exiting...", err)
    }
    err = yaml.Unmarshal(configFile, &Conf)
    if err != nil {
        log.Fatalf("Failed to parse config file: %v", err)
    }
}
```



Прочие хитрости

Аргументы командной строки

```
configFileDefault, _ := filepath.Abs("config.yaml")
configFilePath := flag.String("conf", configFileDefault, "path to config file")
flag.Parse()
conf.ReadConfig(*configFilePath)
```

Аргументы в случае flag передаются с одной чертой: **./cool_app -foo bar -c 1**

https://gobyexample.com/command-line-flags

Логирование и Reader/Writer

https://pastebin.com/DsjKtECX

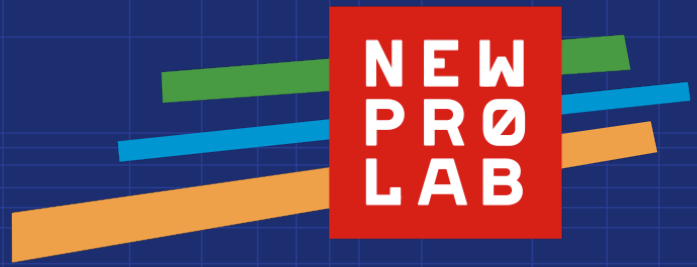
https://habrahabr.ru/post/306914/

Отложенное выполнение

- По сути является заменой питоновского **with smth:** и **try-finally**
- **НЕ** выполняется, если выход из функции происходит **по исключению**, чем немного портит малину

```
func main() {
    defer fmt.Println("world")

    fmt.Println("hello")
}
```

https://blog.golang.org/defer-panic-and-recover https://gobyexample.com/defer



Асинхронность и горутины

Что такое горутина?

- Про то, что такое “корутина”, мы уже знаем.
- Про то, в чем разница между “асинхронностью” и “параллельностью” - тоже
- Go позволяет абсолютно непрозрачно масштабировать сопрограммы по ядрам процессора
- По сути, это значит, что ответ на вопрос “запустится ли отдельный поток при старте горутины” - “а черт его знает, но это и неважно”
- http://bit.ly/2yCNbkv

Что такое горутина?

```
go someLongRunningThing(arg1, arg2)

go func() {
    // колдовать
}()
```

Как управлять зоопарком?

- Go поддерживает стандартные для других языков примитивы синхронизации - мьютексы, условные переменные и семафоры.
- Но злоупотреблять ими в комьюнити Go - не комильфо, ибо есть свои собственные инструменты - каналы (channels) и группы ожидания (waitgroups)

В случае, если функция горутины не является замыканием, необходимо явно туда передать **указатель** на waitgroup

```
var daemonsWg sync.WaitGroup

for w := 0; w < someCounter; w++ {
    daemonsWg.Add(1)
    go func() {
        defer daemonsWg.Done()
        // колдовать
    }()
}

daemonsWg.Wait()
```

Каналы

```go
package main

import "fmt"

func sum(s []int, c chan<- int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum // send sum to c
}

func main() {
    s := []int{7, 2, 8, -9, 4, 0}

    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c // receive from c

    fmt.Println(x, y, x+y)
}
```

Каналы

- Канал можно закрывать через **close(ch)**, тем самым отправляя сообщение всем, кто его слушает, что сообщений в нем больше не будет.
- Сообщения из канала можно перебирать через цикл, который закончится тогда, когда канал закроется - https://tour.golang.org/concurrency/4

Select

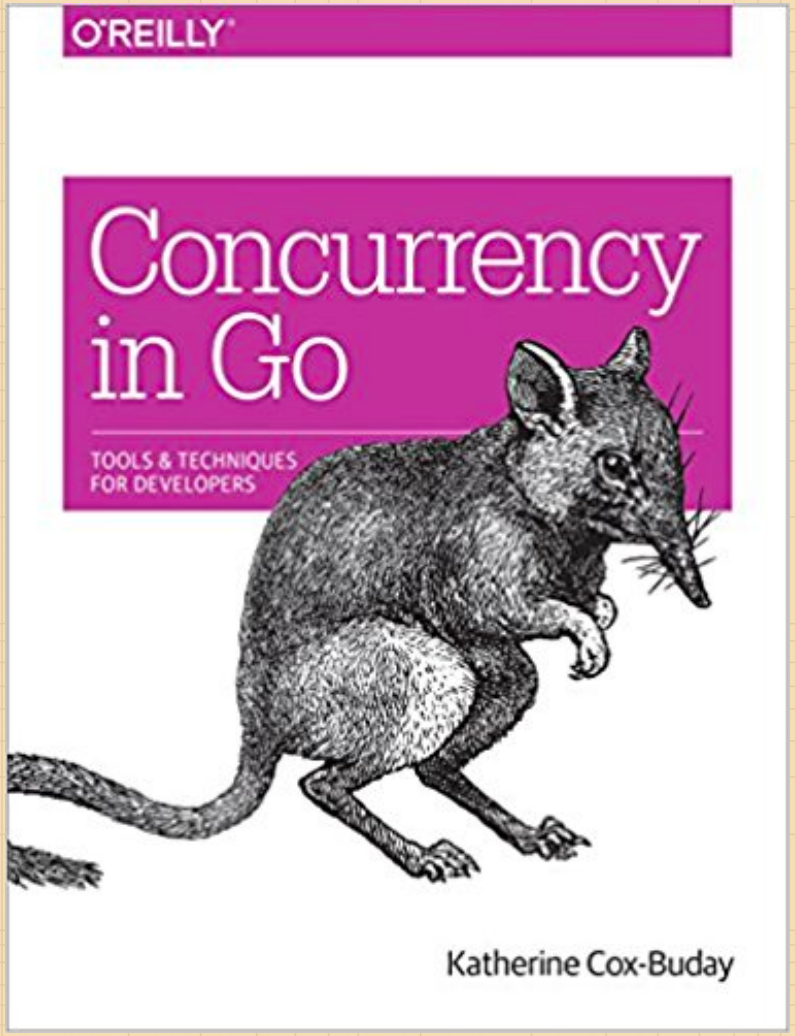
```
package main

import "fmt"

func fibonacci(c, quit chan int) {
    x, y := 0, 1
    for {
        select {
        case c <- x:
            x, y = y, x+y
        case <-quit:
            fmt.Println("quit")
            return
        }
    }
}
```

```
func main() {
    c := make(chan int)
    quit := make(chan int)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(<-c)
        }
        quit <- 0
    }()
    fibonacci(c, quit)
}
```

Кусок продакшена

https://pastebin.com/jHEvjBE1

Почитать


O'REILLY
Concurrency in Go
TOOLS & TECHNIQUES FOR DEVELOPERS
Katherine Cox-Buday