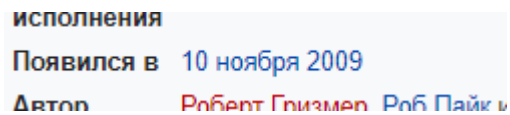


GOLANG

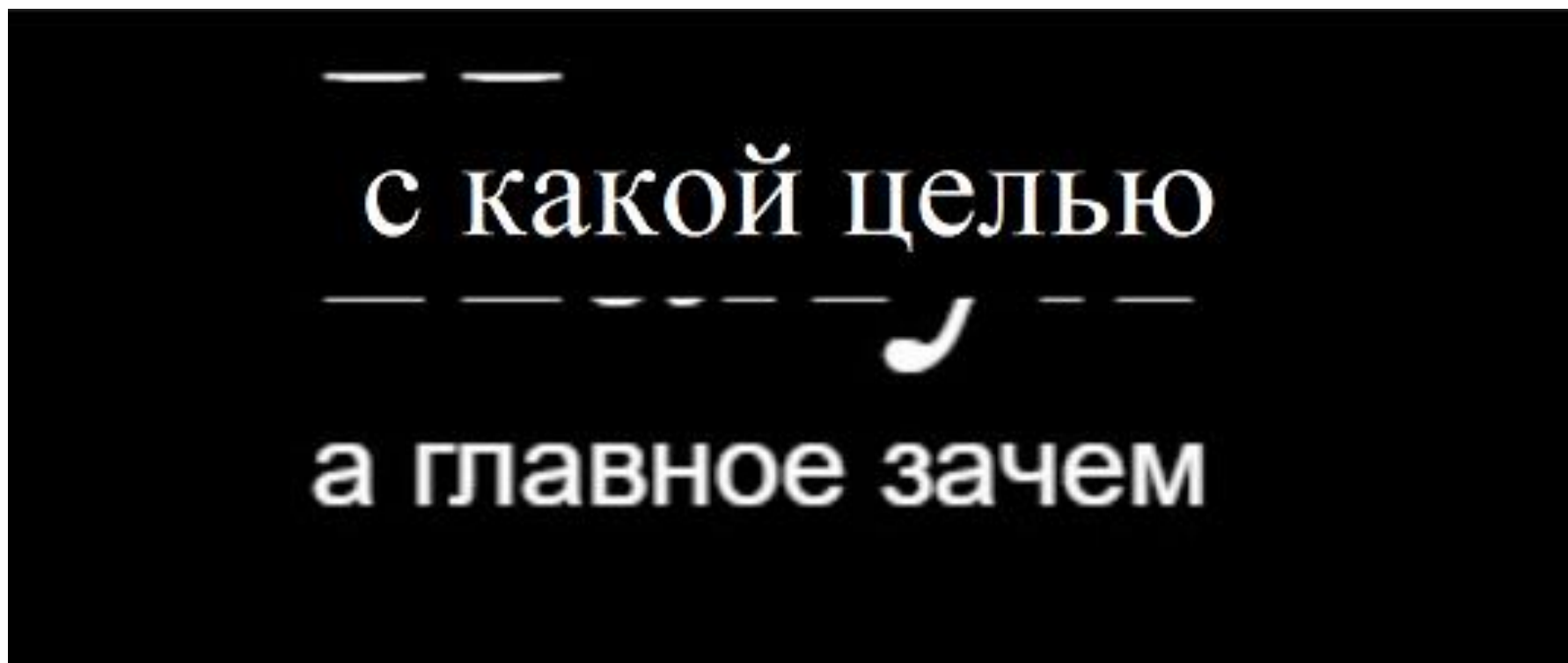
Вы Сергей Волобоев, ПМИ-1



> 

> Язык Go разрабатывался как язык программирования для создания высокоэффективных программ, работающих на современных распределённых системах и многоядерных процессорах.

> c, c++, c#, java, python....





GOLANG



НА ХАЙПЕ*

* Но тиксе

ЧЕМ ГОВОРIT СОВРЕМЕННАЯ МОЛОДЕЖЬ

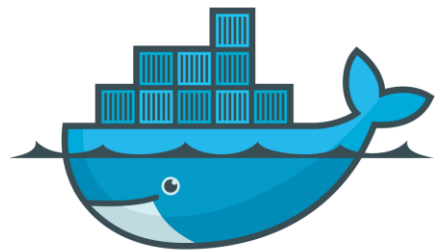
Почему Go?



- Нам(гуглу) нужен хороший язык для бэка. Это всякие таски с многопоточностью и сетевым общением, по сути.
- 1) Мы хотим лаконичный язык(java, c#, и тем более c++ отлетают), чтобы код писался быстро и красиво,
 - 2) Мы хотим скорость легальными методами и удобный многопоток (python отпадает),
 - 3) Надо бы какое-то ООП, хотя бы интерфейсы(питон тоже отпадает, от ООП в плюсах хочется умереть),
 - 4) Хотим type safety и прочие безопасные фишки из жавы и си дизел,
 - 5) Пусть у разработчиков будут почти все инструменты сразу в языке, и не нужно было собирать тучу зависимостей при запуске небольшого сервиса, из множества которых состоит распределённая система.
 - 6) Мемный зверёк на логотипе

Что по итогу?

- Маленький приятный язык, похож на скриптовые, при этом компилируемый. Есть указатели + всякие синтаксические плюхи из скриптовых языков. Классно и быстро компилируется в один бинарник, удобно доставлять. Очень много логичных и приятных фишек.
- Многие крутые и сложные проекты написаны на Go, например:



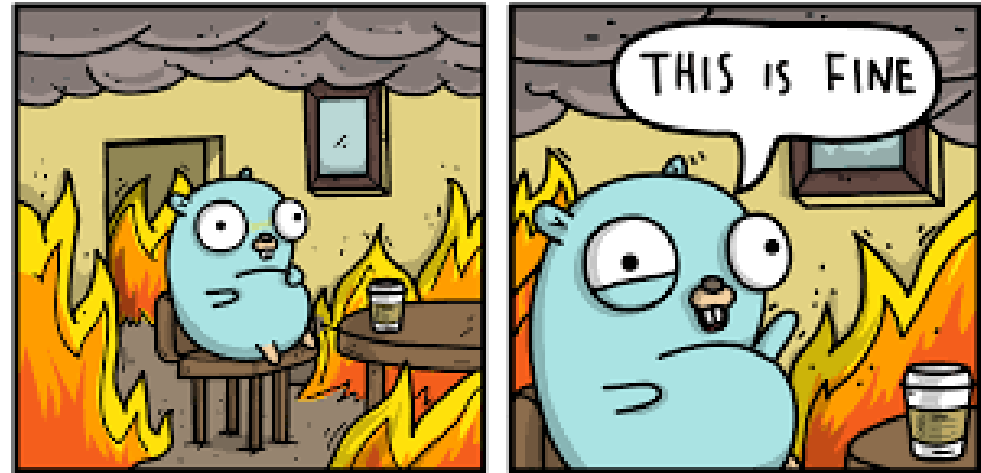
docker



kubernetes

Фишки!

- Умные developers из g00gle при создании нового языка первым делом решили выпилить ненужные вещи:
- 1) Классы
- 2) Наследования
- 3) Конструкторы
- 4) Женерики
- 5) Перегрузка функций
-



Фишки!

- Официальный гайд, можно запускать как локально, так и через сайт. <https://tour.golang.org/welcome/1>
- Codestyle, который утвердили разработчики, можно сразу форматировать код через `go fmt`,
- Нет динамических библиотек, всё компируется в один бинарник, который удобно деплоить,
- Очень быстрая компиляция сразу в бинарники, нет виртуалки для JIT компиляции байт-кода,
- При этом есть сборщик мусора

Фишки!

- Указание типов – после объявления идентификатора. За этим есть скрытый смысл, позволяющий читать сложные типы с указателями. (<https://blog.golang.org/declaration-syntax>, <http://c-faq.com/decl/spiral.anderson.html>)

C++: `int (*fp)(int (*)(int, int), int)`

Go: `f func(func(int,int) int, int) int`

```
multiple-results.go
1 package main
2
3 import "fmt"
4
5 func swap(x, y string) (string, string) {
6     return y, x
7 }
8
9 func main() {
10     a, b := swap("hello", "world")
11     fmt.Println(a, b)
12 }
13
```


Фишки!

- Экспортируемые имена с большой буквы, внутренние – с маленькой. Никаких `private/public`.
- Функции могут возвращать несколько значений, и это не костыль через кортеж, как в питоне (нельзя присвоить одной переменной результат из двух переменных)
- Вывод типов, множественное присваивание

multiple-results.go

```
1 package main
2
3 import "fmt"
4
5 func swap(x, y string) (string, string) {
6     return y, x
7 }
8
9 func main() {
10     a := swap("hello", "world")
11     fmt.Println(a)
12 }
13
```

./prog.go:10:4: assignment mismatch: 1 variable but swap returns 2 values

Go build failed.

Фишки!

- Для циклов есть только фор, но он умеет всё
- Нет (бесячих круглых скобок) у ифа и фора

```
for.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     sum := 0
7     for i := 0; i < 10; i++ {
8         sum += i
9     }
10    fmt.Println(sum)
11 }
12
```

```
forever.go
1 package main
2
3 func main() {
4     for {
5     }
6 }
7
```

```
for-is-gos-while.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     sum := 1
7     for sum < 1000 {
8         sum += sum
9     }
10    fmt.Println(sum)
11 }
12
```

```
range.go
1 package main
2
3 import "fmt"
4
5 var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}
6
7 func main() {
8     for i, v := range pow {
9         fmt.Printf("2**%d = %d\n", i, v)
10    }
11 }
12
```

Фишки!

- Defer

```
defer.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     defer fmt.Println("world")
7
8     fmt.Println("hello")
9 }
10
```

- Всегда требуется явное преобразование типов, даже int -> float

```
type-conversions.go
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func main() {
9     var x, y int = 3, 4
10    var f float64 = math.Sqrt(float64(x*x + y*y))
11    var z uint = uint(f)
12    fmt.Println(x, y, z)
13 }
14
```

Фишки!

- Не нужно разыменовывать указатели на структуры, стильно(-> и . тут одно и то же)
- Все элементы по дефолту в словаре 0 (если значения - int), проверка на существование через val, ok = m[key]
- Классов нет, но методы объявлять можно. Очень странно. Накидывать методы можно только в пределах одного пакета.

```
methods-continued.go
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 type MyFloat float64
9
10 func (f MyFloat) Abs() float64 {
11     if f < 0 {
12         return float64(-f)
13     }
14     return float64(f)
15 }
16
17 func main() {
18     f := MyFloat(-math.Sqrt2)
19     fmt.Println(f.Abs())
20 }
21
```

```
struct-pointers.go
1 package main
2
3 import "fmt"
4
5 type Vertex struct {
6     X int
7     Y int
8 }
9
10 func main() {
11     v := Vertex{1, 2}
12     p := &v
13     p.X = 1e9
14     fmt.Println(v)
15 }
16
```

Фишки!

- Утиная типизация, указывать «implements interface» не надо, неявная реализация интерфейсов
- Интерфейсы – просто кортежи, таскающие значения и тип, вызывающие функцию от ресивера такого-то типа

```
12 func main() {
13     var a Abser
14     f := MyFloat(-math.Sqrt2)
15     v := Vertex{3, 4}
16
17     a = f // a MyFloat implements Abser
18     a = &v // a *Vertex implements Abser
19
20     // In the following line, v is a Vertex (not *Vertex)
21     // and does NOT implement Abser.
22     a = v
23
24     fmt.Println(a.Abs())
25 }
```

```
28 }
29
30 func main() {
31     var i I
32
33     i = &T{"Hello"}
34     describe(i)
35     i.M()
36
37     i = F(math.Pi)
38     describe(i)
39     i.M()
40     //i.Kek()
41 }
42
43 func describe(i I) {
44     fmt.Printf("(%v, %T)\n", i, i)
45 }
46
```

```
(&{Hello}, *main.T)
Hello
(3.141592653589793, main.F)
3.141592653589793
```



Фишки!

- Go рутины – легковесные треды
- Channel ака потокобезопасная очередь заданного размера

```
4
5 func sum(s []int, c chan int) {
6     sum := 0
7     for _, v := range s {
8         sum += v
9     }
10    c <- sum // send sum to c
11 }
12
13 func main() {
14     s := []int{7, 2, 8, -9, 4, 0}
15
16     c := make(chan int)
17     go sum(s[:len(s)/2], c)
18     go sum(s[len(s)/2:], c)
19     x, y := <-c, <-c // receive from c
20
21     fmt.Println(x, y, x+y)
22 }
23
```

```
goroutines.go
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func say(s string) {
9     for i := 0; i < 5; i++ {
10         time.Sleep(100 * time.Millisecond)
11         fmt.Println(s)
12     }
13 }
14
15 func main() {
16     go say("world")
17     say("hello")
18 }
19
```


- Есть либы для логгирования и тестирования
- Пакетный менеджер – `go get`. Но он ставит пакеты в глобальный компилятор, и это плохая практика. Создаётся `go mod init`, и при `go run` он сам прописывает зависимости. Есть разделение на пакеты(нейм спейсы) и модули(библиотеки)

Опыт использования

- Ну я сделал лабу по gRPC на go...
- Сложно писать
- Компилятор бьёт меня ногами (запрещает мне хранить в коде неиспользуемые переменные)
- Странная система пакетов(`export GO111MODULE=on??????????`)
- Малое комьюнити, гугол не находит ответы на тупые вопросы(ну и не только на них)
- Не все синтаксические трюки показаны в гайде
- Документация топчик, но только на то, что написал гугол. От остальных либ становится грустно

