Основы

Tinkoff.ru

Сергей Лутошин

Backend разработка

В инвестициях с момента запуска

Отвечаю за многие нагруженные сервисы в этом направлении

План занятия

1. Немного истории
2. Как golang появился в tinkoff.ru/invest
3. Синтаксические конструкции и типы данных
4. Слайсы
5. Мапы

Немного истории

A long time ago in a Google far far away …

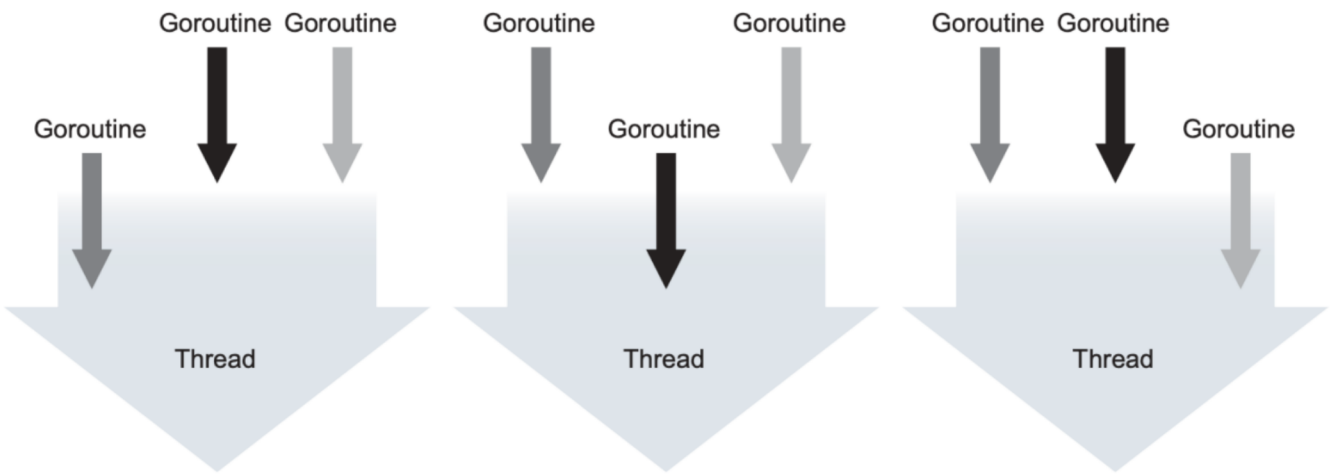
Краткая история языка. Кто и когда?

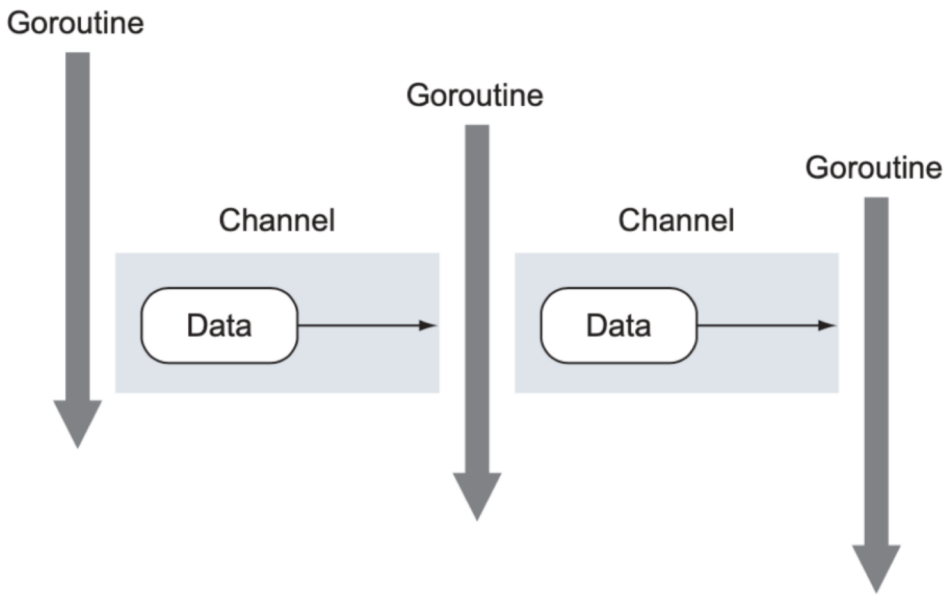
Роберт Гризмер, Роб Пайк, Кен Томпсон

Краткая история языка. Зачем?

- Скорость разработки
- Скорость компиляции
- Асинхронная M:N модель
- Горутины, каналы, пайплайны
- Сборщик мусора

Асинхронная M:N модель

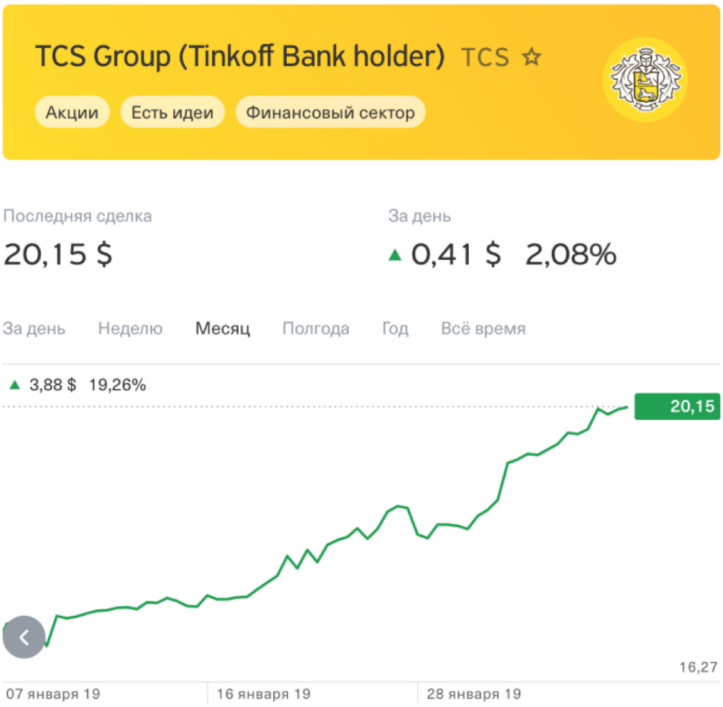



Goroutine
Goroutine
Goroutine
Channel
Channel
Data
Data

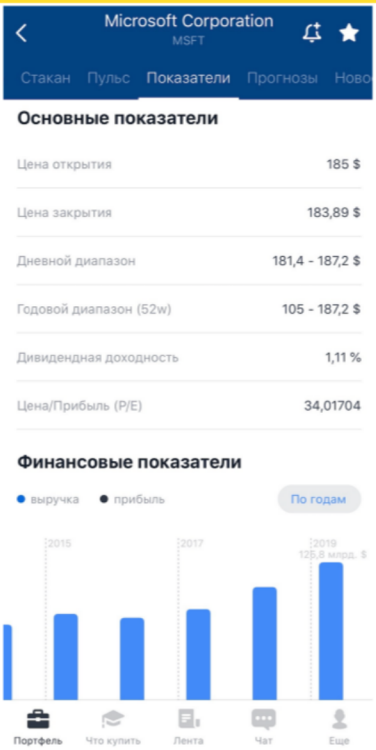
Golang в tinkoff.ru/invest

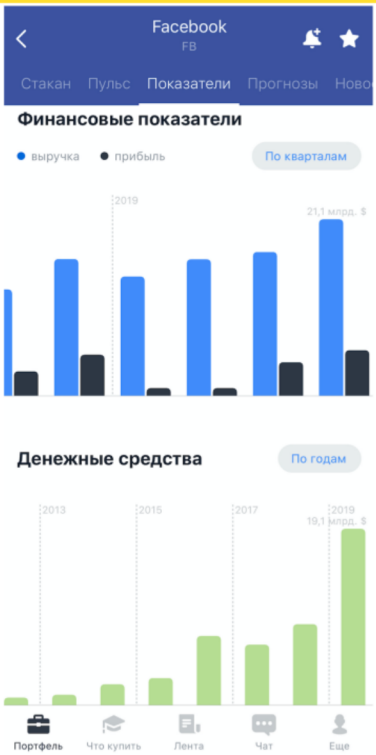
Тинькофф Инвестиции

Рыночные данные



Показатели по бумагам





Лента новостей и идей

Все | Идеи | Новости

Google

Россия стала лидером по числу запросов госорганов к Google об удалении контента

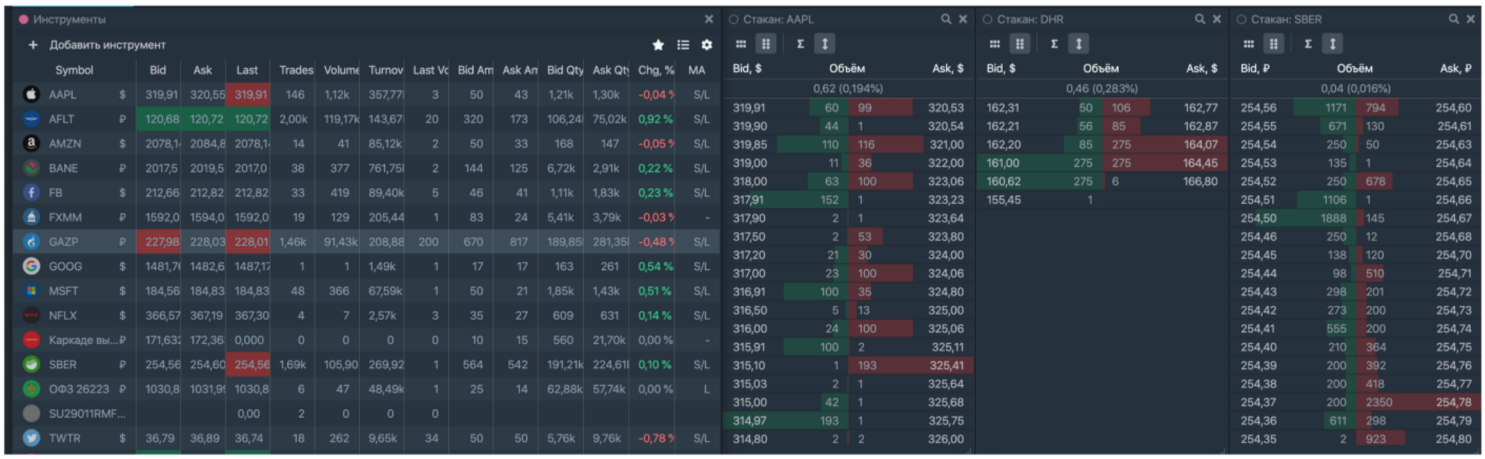
В первом полугодии 2018 года от российских госорганов к Google поступило 19,2 тыс. запросов на удаление контента, что составляет больше 70% от всех мировых запросов, следует из ежегодного отчета компании.

Вчера в 19:37 · RNS

Избранное

TCS Group (Tink...	20,15 $	▲ 2,08%

Стриминговая доставка котировок



Синтаксис и типы данных

```
uint8       unsigned 8-bit integers (0 to 255)
uint16      unsigned 16-bit integers (0 to 65535)
uint32      unsigned 32-bit integers (0 to 4294967295)
uint64      unsigned 16-bit integers (0 to 18446744073709551615)
int8        signed 8-bit integers (-128 to 127)
int16       signed 16-bit integers (-32768 to 32767)
int32       signed 32-bit integers (-2147483648 to 2147483647)
int64       signed 64-bit integers (-9223372036854775808 to 9223372036854775807)
float32     IEEE-754 32-bit floating-point numbers
float64     IEEE-754 64-bit floating-point numbers
complex64   floating point numbers with float32 real and imaginary parts
complex128  floating point numbers with float64 real and imaginary parts
byte        alias uint8
rune        alias for int32
```

```
uint     either 32 or 64 bits
int      same size as uint
uintptr  an unsigned integer large enough to store the uninterpreted bits of a pointer value
```

64 бита на 64-х битной платформе, 32 бита на 32-х битной платформе

Тип данных string

- Строки не мутабильны
- Они всегда в UTF-8 по умолчанию
- Существует 2 способа инициал. строку(raw && interpret literal)

```
a := `Я строка, в которой не нужно экранировать спец. символы
```

<<< RAW string literals

```
a := "Скажи \"Привет\" финтеху"
```

<<< interpret string literals

Тип данных string

```go
s := "Hello, Сергей!"
for i, c := range s {
    fmt.Printf("%d: %s\n", i, string(c))
}
fmt.Printf("length of '%s': %d", s, len(s))
```

```
0: H
1: e
2: l
3: l
4: o
5: ,
6:
7: С
9: е
11: р
13: г
15: е
17: й
19: !
length of 'Hello, Сергей!': 20
```

Rune это особенный тип!
Rune alias int32
Может занимать до 3-х байт

Приведем string к []rune

```go
s := "Hello, Сергей!"
sb := []byte(s)
sr := []rune(s)

j := 1
for _, r := range sr {
    fmt.Printf("%d: %s\n", j, string(r))
    j++
}
fmt.Printf("length of '%s': %d(string), %d([]byte), %d([]rune)", s, len(s), len(sb), len(sr))
```

```
1: H
2: e
…
13: й
14: !
length of 'Hello, Сергей!': 20(string), 20([]byte), 14([]rune)
```

Как инициализировать переменную

Without initialization

```
var i int
```

With initialization, explicit type

```
var i int = 1
j := int64(1)
```

With initialization, implicit type

```
i := 1
```

Что такое zero value?

```go
var a int
var b string
var c float64
var d bool

fmt.Printf("var a %T = %+v\n", a, a)
fmt.Printf("var b %T = %q\n", b, b)
fmt.Printf("var c %T = %+v\n", c, c)
fmt.Printf("var d %T = %+v\n", d, d)
```

```
var a int = 0
var b string = ""
var c float64 = 0
var d bool = false
```

Константы

```go
const (
    a = "A"
    b = 'A'
    c = 2
    d = 3.141592
    e = int64(7)
)
func main() {
    fmt.Printf("a is type %T with value %q\n", a, a)
    fmt.Printf("b is type %T with value %q\n", b, b)
    fmt.Printf("c is type %T with value %v\n", c, c)
    fmt.Printf("d is type %T with value %v\n", d, d)
    fmt.Printf("e is type %T with value %v\n", e, e)
}
```

```
a is type string with value "A"
b is type int32 with value 'A'
c is type int with value 2
d is type float64 with value 3.141592
e is type int64 with value 7
```

fmt.Printf

```
// Use %v to print the value
// Use %s to print string
// Use %q to print quote string
// Use %d to print a decimal
// Use %T to print a type
// Use \ to escape the character
// Use \n to print a new line
// Use %+v to print name and value (later for structs)
```

Iota

```go
const (
    _ int = iota
    Red
    Blue
    Green
)

func main() {
    fmt.Printf("The value of Red is %v\n", Red)
    fmt.Printf("The value of Blue is %v\n", Blue)
    fmt.Printf("The value of Green is %v\n", Green)
}
```

```
The value of Red is 1
The value of Blue is 2
The value of Green is 3
```

Iota

```go
const (
    Read = 1 << iota  // 00000001 = 1
    Write             // 00000010 = 2
    Remove            // 00000100 = 4
    // admin have all permissions
    Admin = Read | Write | Remove
)
func main() {
    fmt.Printf("The value of Read is %v\n", Read)
    fmt.Printf("The value of Write is %v\n", Write)
    fmt.Printf("The value of Remove is %v\n", Remove)
    fmt.Printf("The value of Admin is %v\n", Admin)
}
```

```
The value of Read is 1
The value of Write is 2
The value of Remove is 4
The value of Admin is 7
```

Обычный цикл for

```go
fmt.Printf("Обычный цикл for: ")
for i := 0; i < 2; i++ {
    fmt.Printf("%d ", i)
}
fmt.Println()
```

```
Обычный цикл for: 0 1
```

Бесконечный цикл for

```go
var counter int
fmt.Printf("Бесконечный цикл: ") // Выход по return либо break
for {
    fmt.Printf("%d ", counter)
    if counter == 2 {
        break
    }
    counter++
}
```

```
Бесконечный цикл: 0 1 2
```

Цикл range по слайсу

```
1  fmt.Println("Range по слайсу: ")
2  slice := []int{0, 1, 2, 3}
3  for i, value := range slice {
4     // value - это переменная цикла, значение которой совпадает с элементом i-ой итерции
5   fmt.Printf("\tslice[%d]=%v, &value=%p, &slice[i]=%p\n", i, value, &value, &slice[i])
6  }
```

```
Range по слайсу:
        slice[0]=0, &value=0xc000098028, &slice[i]=0xc00009a020
        slice[1]=1, &value=0xc000098028, &slice[i]=0xc00009a028
        slice[2]=2, &value=0xc000098028, &slice[i]=0xc00009a030
        slice[3]=3, &value=0xc000098028, &slice[i]=0xc00009a038
```

Switch

```
1  fmt.Print("Go runs on ")
2  switch os := runtime.GOOS; os {
3  case "darwin":
4     fmt.Println("OS X.")
5  case "linux":
6     fmt.Println("Linux.")
7  default:
8     // freebsd, openbsd,
9     // plan9, windows...
10    fmt.Printf("%s.", os)
11 }
12
```

```
Go runs on OS X.
```

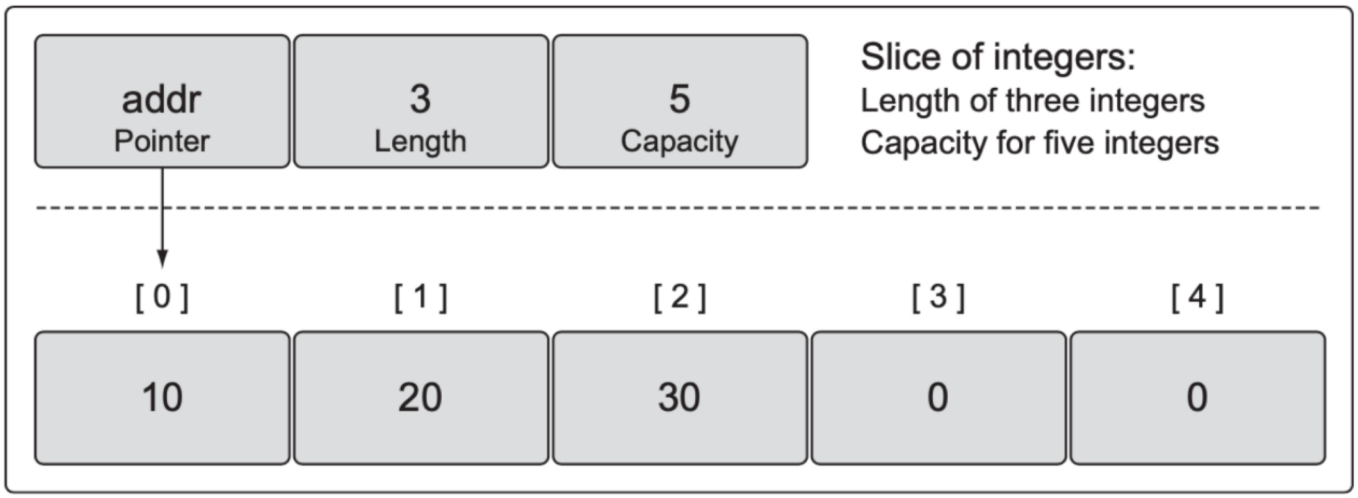
Слайсы

Почему слайсы?

- Легко менять размер (есть встроенная функция append)
- Возможность работать с подмножеством слайса
- Он представлен непрерывным участком в памяти
- Слайс как и массив всегда сохраняет последовательность

```
var array [5]string
var slice []string
```

Внутренне устройство слайса



Добавляем элементы в слайс

```go
var stocks []string
stocks = append(stocks, "AAPL") // appends one element
stocks = append(stocks, "FB", "TCS") // appends variadic elements

moreStocks := []string{"YNDX", "MSFT"}
stocks = append(stocks, moreStocks...) // appends other slice

fmt.Println("stocks: ", stocks)
```

```
stocks:  [AAPL FB TCS YNDX MSFT]
```

Добавляем элементы в слайс

```go
var stocks []string
fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
stocks = append(stocks, "AAPL") // appends one element
fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
stocks = append(stocks, "FB", "TCS") // appends variadic elements
fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
moreStocks := []string{"YNDX", "MSFT"}
stocks = append(stocks, moreStocks...) // appends other slice
fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
fmt.Println("stocks: ", stocks)
```

```
len=0, cap=0
len=1, cap=1
len=3, cap=3
len=5, cap=6
stocks:  [AAPL FB TCS YNDX MSFT]
```

Функция make

```go
otherStocks := []string{"YNDX", "MSFT", "FB", "TCS"}
stocks := make([]string, 0, len(otherStocks))
fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
for _, stock := range otherStocks {
    stocks = append(stocks, stock[:1])
    fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
}

fmt.Println("stocks: ", stocks)
```

```
len=0, cap=4
len=1, cap=4
len=2, cap=4
len=3, cap=4
len=4, cap=4
stocks:  [Y M F T]
```

Типичная ошибка make+append

```go
otherStocks := []string{"YNDX", "MSFT", "FB", "TCS"}
stocks := make([]int, len(otherStocks), len(otherStocks))
fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
for _, stock := range otherStocks {
    stocks = append(stocks, int(stock[0]))
    fmt.Printf("len=%d, cap=%d\n", len(stocks), cap(stocks))
}

fmt.Println("stocks: ", stocks)
```

```
len=4, cap=4
len=5, cap=8
len=6, cap=8
len=7, cap=8
len=8, cap=8
stocks:  [0 0 0 0 89 77 70 84]
```

Подмножества слайса

```go
stocks := []string{"YNDX", "MSFT", "FB", "TCS"}

fmt.Println(stocks[1:3])

fmt.Println(stocks[2:len(stocks)])
fmt.Println(stocks[2:])

fmt.Println(stocks[0:3])
fmt.Println(stocks[:3])
```

```
[MSFT FB]
[FB TCS]
[FB TCS]
[YNDX MSFT FB]
[YNDX MSFT FB]
```

Функция copy

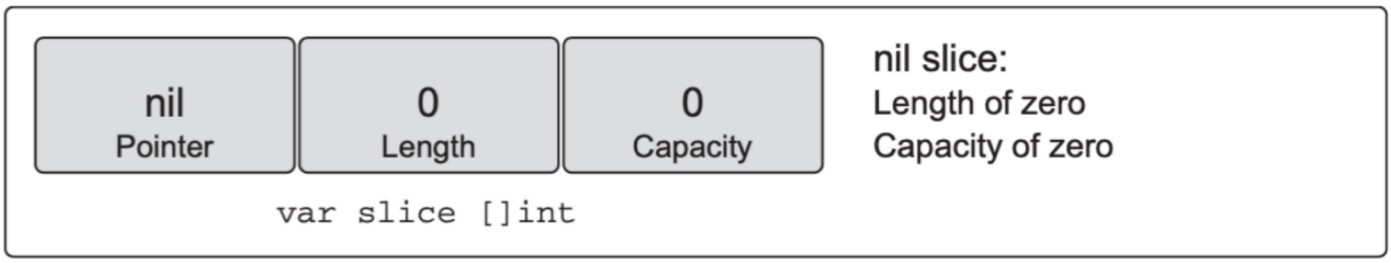
```go
original := []string{"YNDX", "MSFT", "FB", "TCS"}

ref := original

cpy := make([]string, len(original))

copy(cpy, original)
ref[0] = "_Y_"
original[1] = "_M_"

fmt.Println("original: ", original)
fmt.Println("ref: ", ref)
fmt.Println("cpy: ", cpy)
```

```
original:  [_Y_ _M_ FB TCS]
ref:  [_Y_ _M_ FB TCS]
cpy:  [YNDX MSFT FB TCS]
```

Slice == nil

```go
var stocks []string
if stocks == nil {
    fmt.Println("slice is nil")
}
```

```
slice is nil
```



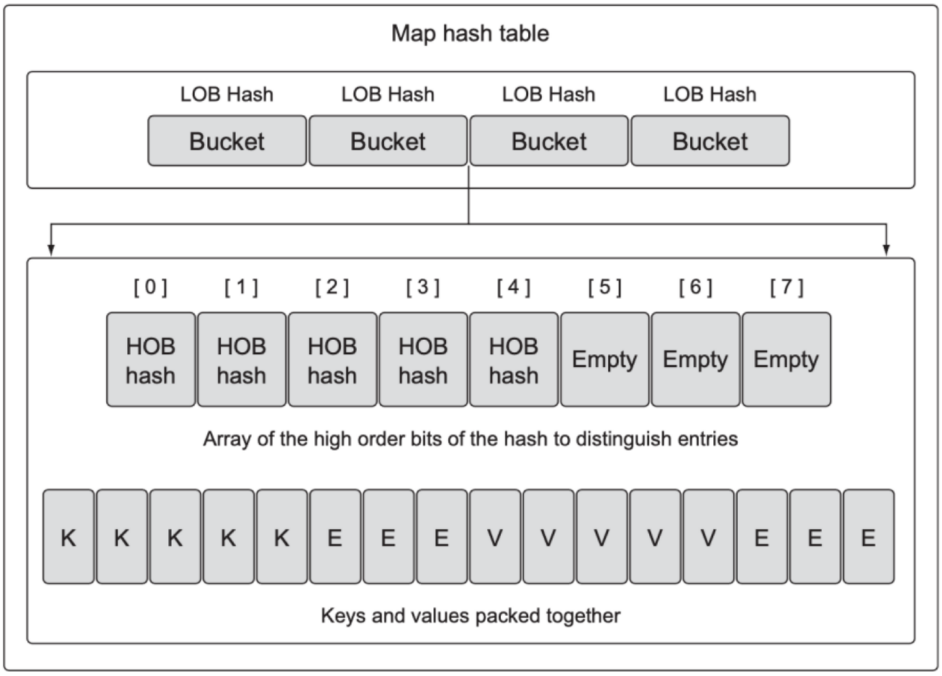
Мапы

Когда мапы?

- Когда последовательность элементов не важна
- Когда известен ключ, по которому нужно достать значение

```
yield := make(map[string]string)

yield := map[string]int{
    "YNDX": 1000,
    "TCS": 10000,
    "FB": 1000000,
}
```

Что такое мапа в go?



K = Key V = Value E = Empty

Добавить/прочитать/удалить ключ

```go
yield := make(map[string]int)
yield["YNDX"] = 1000
yield["TCS"] = 10000
fmt.Println("TCS yield is", yield["TCS"])
delete(yield, "YNDX")
fmt.Println(yield)
```

```
TCS yield is 10000
map[TCS:10000]
```

Проверка существования ключа

```go
stocks := []string{"YNDX", "FB", "TCS"}
yield := map[string]int{
    "YNDX": 1000,
    "TCS":  10000,
}
for _, stock := range stocks {
    y, ok := yield[stock]
    if !ok {
        continue
    }
    fmt.Println("yield for", stock, "is", y)
}
```

```
yield for YNDX is 1000
yield for TCS is 10000
```

for-range для мапы

```go
yield := map[string]int{
    "YNDX": 1000,
    "TCS":  10000,
}
for stock, yield := range yield {
    fmt.Println("yield for", stock, "is", yield)
}
```

```
yield for YNDX is 1000
yield for TCS is 10000
```

Типичная ошибка при работе с мапами

```go
var yield map[string]int
yield["YNDX"] = 1000
```

```
panic: assignment to entry in nil map

goroutine 1 [running]:
main.main()
```

PS: как мы делимся знаниями?

1. Реализуем гипотезу в play.golang.org
2. Нажимаем кнопку [share] и, получив ссылку, шарим ее товарищам в телеграм-канал
3. Расписываем свою идею, например какой был ожидаемый результат и какой получили фактически
4. Ждем фидбека от коллег и кураторов

Обратная связь

Tinkoff.ru

Спасибо за внимание

Tinkoff.ru