

# Лекция 21.09

# Оглавление

- Повторение
- Словари
- Множества
- Stack vs Heap
- Garbage collection
- Interface'ы
- Struct embedding
- Errors

# Вопросы с собеседований

Сколько байт занимает тип `int` ?

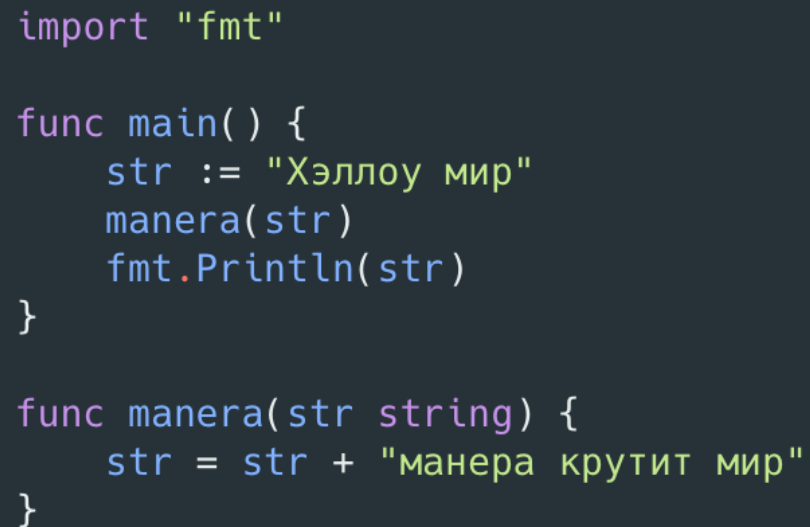
# Вопросы с собеседований

Ответ:

4 или 8 байта в зависимости от архитектуры процессора.

# Вопросы с собеседований

Что выведет данный код?



```
import "fmt"


func main() {
    str := "Хэллоу мир"
    manera(str)
    fmt.Println(str)
}

func manera(str string) {
    str = str + "манера крутит мир"
}
```

# Вопросы с собеседований

Ответ:

Хэллоу мир



```
type _string struct {  
    elements *byte // underlying bytes  
    len      int   // number of bytes  
}
```

# Вопросы с собеседований

Какая алгоритмическая сложность операции конкатенации двух строк длиной N и M?

Пример:

`str1 = str1 + str2`

# Вопросы с собеседований

Ответ:

$O(N + M)$

Поэтому используем `strings.Builder` !



# Вопросы с собеседований

Что выведет данный код?

```
package main

import (
    "fmt"
)

func main() {
    arr := []int{1, 2, 3, 4, 5}

    addNewElemIfEven(arr, 6)


    fmt.Println(arr)
}

func addNewElemIfEven(arr []int, newNum int) {
    if newNum%2 == 0 {
        arr = append(arr, newNum)
    }
}
```

# Вопросы с собеседований

Ответ:

[1 2 3 4 5]



```
type slice struct {  
    array unsafe.Pointer  
    len    int  
    cap    int  
}
```

# Вопросы с собеседований

Что выведет данный код?

```
package main

import (
    "fmt"
)

func main() {
    arr := []int{1, 2, 3, 4, 5}
    changeFirstNumIfEven(arr, 6)
    fmt.Println(arr)
}

func changeFirstNumIfEven(arr []int, newNum int) {
    if newNum%2 == 0 {
        arr[0] = newNum
    }
}
```

# Вопросы с собеседований

Ответ :

[6 2 3 4 5]

# Вопросы с собеседований

Какова алгоритмическая сложность добавления элемента в конец slice'a?

# Вопросы с собеседований

Ответ:

В худшем случае  $O(n)$ , где  $n$  – длина слайса

В среднем за  $O(1)$

# Вопросы с собеседований

Что выведет данный код?

```
package main

import (
    "fmt"
)

func main() {
    // создаем и заполняем slice arr1
    arr1 := make([]int, 0, 6)
    for i := 0; i < 5; i++ {
        arr1 = append(arr1, i)
    }

    // создаем и изменяем slice arr2
    arr2 := arr1
    arr2 = append(arr2, 6)
    arr2[0] = -1

    // результат
    fmt.Println(arr1)
}
```

# Вопросы с собеседований

Ответ:

`[-1 1 2 3 4]`



# Типы - словари

`map` [ ТИП\_КЛЮЧА ] ТИП\_ЗНАЧЕНИЯ

# Map

```
type hmap struct {  
    // Note: the format of the hmap is also encoded in cmd/compile/internal/reflectdata/reflect.go.  
    // Make sure this stays in sync with the compiler's definition.  
    count      int // # live cells == size of map. Must be first (used by len() builtin)  
    flags      uint8  
    B          uint8 // log_2 of # of buckets (can hold up to loadFactor * 2^B items)  
    noverflow  uint16 // approximate number of overflow buckets; see incrnoverflow for details  
    hash0      uint32 // hash seed  
    buckets    unsafe.Pointer // array of 2^B Buckets. may be nil if count==0.  
    oldbuckets  unsafe.Pointer // previous bucket array of half the size, non-nil only when growing  
    nevacuate  uintptr      // progress counter for evacuation (buckets less than this have been evacuated)  
    extra *mapextra // optional fields  
}
```

# Типы - словари

`map` [ ТИП\_КЛЮЧА ] ТИП\_ЗНАЧЕНИЯ

Возможные типы ключа – сравнимые типы:

- `boolean`
- `numeric`
- `string`
- `pointer`
- `channel`
- `interface types`
- `structs` – если все поля могут быть ключами
- `array`

# Типы - словари

```
m := map[int]int{}
```

```
len(m)           // количество элементов
```

```
m[1] = 1         // присвоение ключу значения
```

```
delete(m, 1)     // удаление по ключу
```

```
value := m[1]    // если нет -- zero value
```

```
value, hasValue := m[1] // явно проверить наличие
```

# Типы - словари

```
// nil map
```

```
var a map[int]string
```

```
// empty map
```

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

```
c := map[int]string{ }
```

# Типы - словари

А что с асимптотикой?

# Типы - словари

А что с асимптотикой?

В go 1.20 реализация map – это hash-таблица

# Словари

- Ссылочный тип: данные не копируются при присвоении
- Ключ – любой сравнимый тип или его производная
- Может быть модифицирован
- Значение по-умолчанию (nil) не может быть изменено

## Вопросы?



# Типы - множества

В go нет специального типа для множеств

```
map[int]struct{ }
```

# Типы - множества

В go нет специального типа для множеств

```
map[int]struct{ }
```

## Вопросы?

# Вопросы с собеседований

Что выведет данный код?

```
package main

import (
    "fmt"
)

func main() {
    m := make(map[int]int)
    for i := 0; i < 5; i++ {
        m[i] = i * i
    }

    changeElem(4, -1, m)

    fmt.Println(m[4])
}

func changeElem(key int, value int, m map[int]int) {
    m[key] = value
}
```

# Вопросы с собеседований

Ответ:

-1

# Управление памятью

## Стек

- Быстрый
- Выделение/освобождение памяти в фиксированном порядке (FILO)
- Свой у каждой горутины → потокобезопасен

## Куча

- Относительно медленная
- Выделение освобождение памяти в произвольном порядке
- Аллокации в куче потокобезопасны
- Управляется сборщиком мусора

# Управление памятью. GC

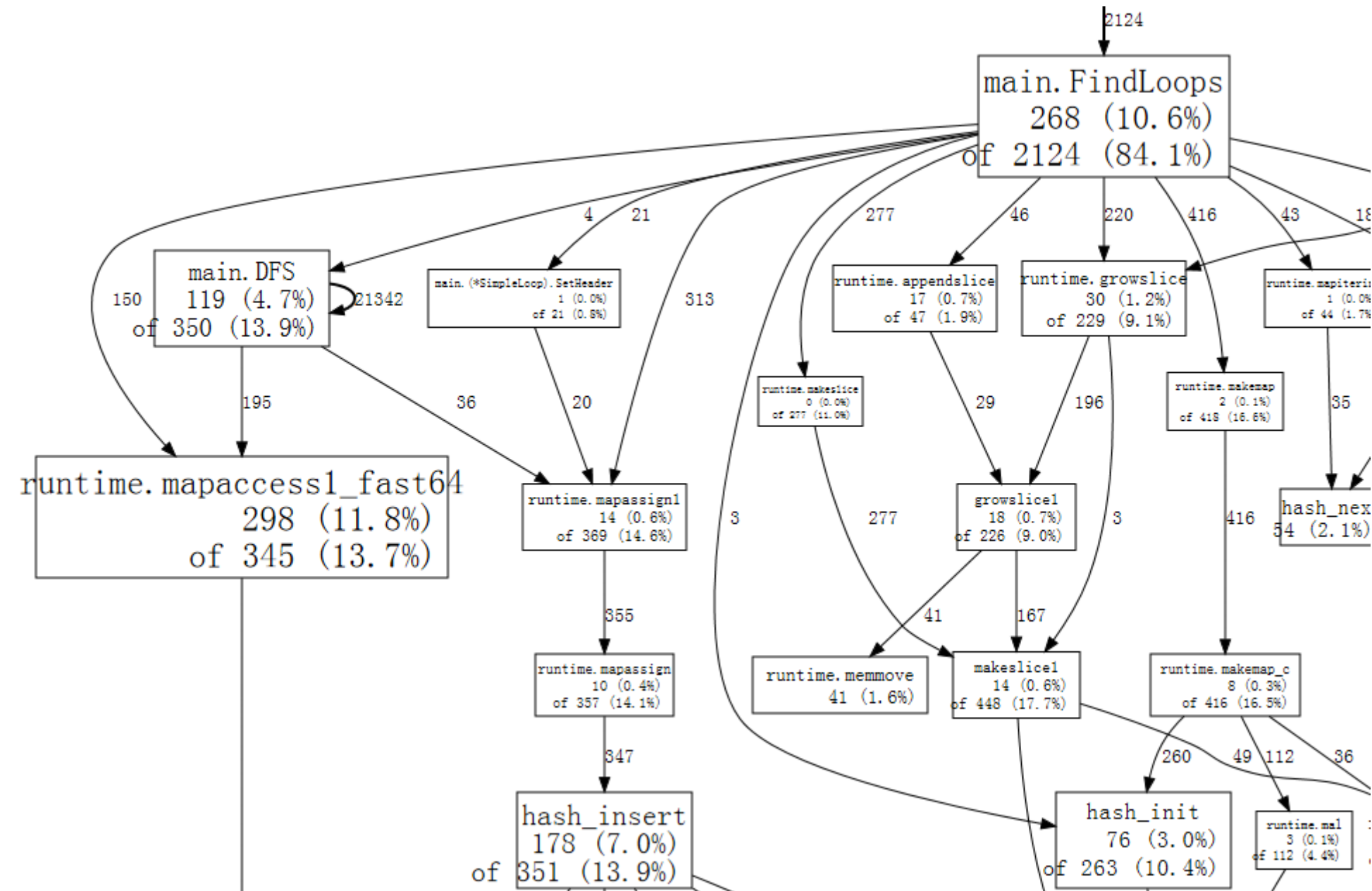
## Инструменты оптимизации:

- Профилировщик

- [pkg.go.dev/runtime/pprof](https://pkg.go.dev/runtime/pprof)
- [pkg.go.dev/net/http/pprof](https://pkg.go.dev/net/http/pprof)
- [go.dev/blog/pprof](https://go.dev/blog/pprof)

- Метрики

- [pkg.go.dev/runtime/metrics](https://pkg.go.dev/runtime/metrics)
- [prometheus.io/docs/guides/go-application/](https://prometheus.io/docs/guides/go-application/)



[go.dev/blog/pprof](https://go.dev/blog/pprof)

# Управление памятью

**Stack vs Heap**

**Где go выделяет память?**

**Escape Analysis:**

```
go build -gcflags=-m=3
```

# Управление памятью

**Don't be too clever**



# Garbage collection (Mark and Sweep)

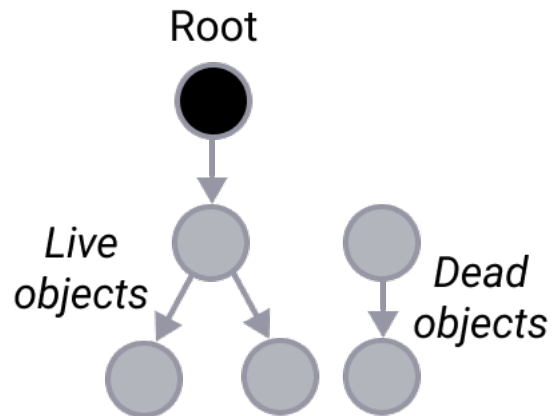
Алгоритм Mark and Sweep состоит из двух частей:

- 1) Mark разметка
- 2) Sweep очистка памяти

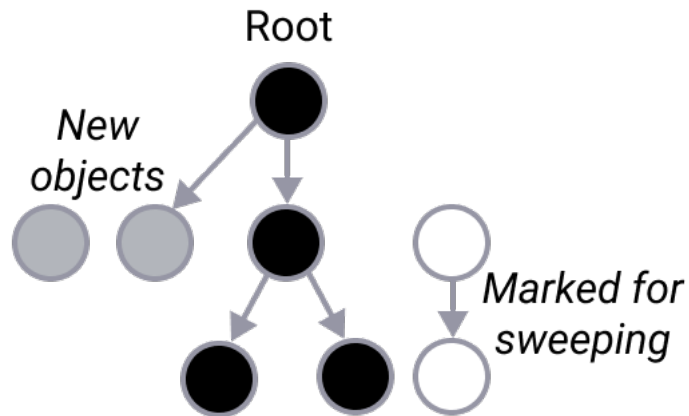
# Mark (3-ех цветный алгоритм)

- идет сканирование объектов первого уровня доступа, тех которые хранятся либо глобально, либо в стеке потока
- объекты первого уровня помечаются серым цветом
- в каждом сером объекте ищутся ссылки на области памяти
- объекты по ссылкам помечаются серым
- сам родительский элемент помечается черным
- процесс повторяется, пока не останется серых объектов
- белые объекты – объекты, которые необходимо будет удалить

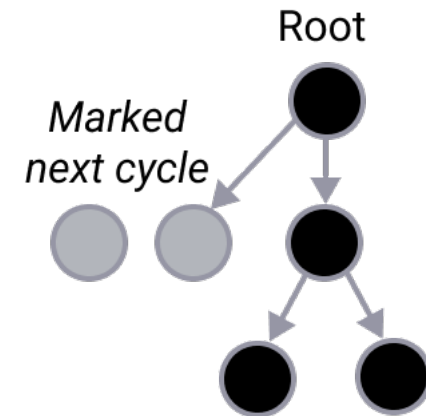
**Before Mark Phase**



**After Mark Phase**



**After Sweep Phase**



# Garbage collection

- Stop the world
- Ожидаем пока все горютины достигнут safe-point
- Ожидаем завершения очистки с прошлых итераций
- Включается write barrier
- Start the world
- 3-ех цветный алгоритм
- Stop the world (гипотетически можно обойтись без этого)
- Отключение write barrier
- Start the world
- Очистка ресурсов в фоне

# Garbage collection

- Сборщик мусора вызывается, когда куча выросла в 2 раза
- Коэффициент 2 можно поменять, изменив переменную среды GOGC
- Вручную сборщик мусора можно запустить с помощью `runtime.GC()`

# Интерфейсы

- Механизм обобщения (generalization) в Go
- Применяется "утиная типизация" - нет необходимости явно указывать, какие интерфейсы имплементирует данный тип (в отличии от многих других языков)
- Интерфейс - абстрактный тип, он только описывает набор методов (и никак не связан с имплементацией)
- Если мы работаем с значением типа интерфейс, мы только можем знать, что объект может делать, но ни чем он является (но можем предполагать, об этом отдельно :) )

# Type assertion

- Если мы знаем, какой тип скрывается за интерфейсом (или, например, что скрывается тип, который удовлетворяет большему интерфейсу) мы можем привести значение к этому типу
- Этот механизм используется, в том числе, для получения конкретных ошибок из интерфейса `error`

```
var w io.Writer
w = os.Stdout
f := w.(*os.File)           // success: f == os.Stdout
c := w.(*bytes.Buffer)      // panic: interface holds *os.File, not *bytes.Buffer
c, ok := w.(*bytes.Buffer) // don't panic, but ok == false
```

16

# Type switch

- Позволяет выбрать из предположений нужный тип. Например, может использоваться при написании `Printf`

```
switch x.(type) {  
    case nil:          // ...  
    case int, uint:    // ...  
    case bool:         // ...  
    case string:       // ...  
    default:           // ...  
}
```

# Embedding

- "Своеобразное" наследование, позволяет указать другую структуру анонимным полем и обращаться через `.` напрямую к полям (и методам) анонимного поля

```
type Circle struct {  
    Point // Point Point  
    Radius int  
}  
  
var c = Circle{Point: Point{1, 2}, Radius: 0}  
c.X = 0  
c.Point.Y = 0
```

- Нельзя иметь два анонимных поля одного типа
- Видимость анонимного поля определяется видимостью имени типа (но сокращения через `.` все равно продолжают работать)
- Если возникает конфликт по именам полей двух или более анонимных полей, то краткая форма через `.` недоступна
- К полям можно навешивать тэги, которые используются некоторыми



# Ошибки

- Все ошибки возвращаются явным образом с типом встроенного интерфейса `error` (исключение - `panic`, о нем отдельно)  

```
type error interface {  
    Error() string  
  
}
```
- Можно создавать ошибки различных типов, главное, чтобы они удовлетворяли интерфейсу
- Ошибки можно сравнивать с `nil` и между собой (устаревший вариант, после go 1.13 для сравнения используются `errors.Is` и `errors.As`)
- Метод `errors.Unwrap` вызывает соответствующий метод у типа, если он определен, либо возвращает `nil`

# Вопросы с собеседований

Что выведет данный код?

```
package main

import "fmt"

type MyError struct {
    msg string
}

func (m MyError) Error() string {
    return m.msg
}


func main() {
    err := returnErrIfEven(5)
    if err != nil {
        fmt.Println("Число четное!!!")
    }
}

func returnErrIfEven(num int) error {
    var err *MyError = nil
    if num%2 == 0 {
        err = &MyError{msg: "четное число"}
    }
    return err
}
```

# Вопросы с собеседований

Ответ:

“Число четное!!!”



```
type iface struct {  
    tab *itab  
    data unsafe.Pointer  
}
```

# Вопросы с собеседований

Что выведет данный код?

```
package main

import "fmt"

type MyInterface interface {
    Get() string
}

func Execute(i MyInterface) {
    fmt.Println(i.Get())
}

type R struct {
    f string
}

func (r *R) Get() string {
    return r.f
}

func main() {
    qwe := R{"hello"}
    Execute(qwe)
}
```