




# Golang Developer. Professional

[otus.ru](https://otus.ru)

 Проверить, идет ли запись

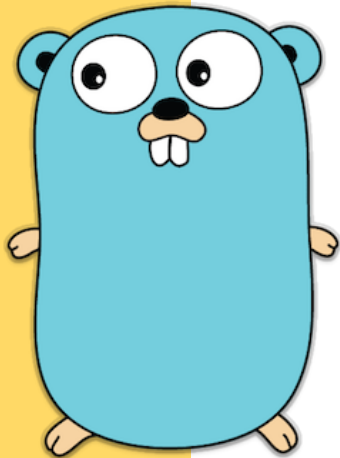
# Меня хорошо видно && слышно?

 Ставим “+”, если все хорошо  
“-”, если есть проблемы

Тема вебинара

# Интерфейсы изнутри

Рубаха Юрий



# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

## Условные обозначения



Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или  
задайте вопрос

# О чем будем говорить

- Определение и реализация интерфейсов
- Композиция интерфейсов
- Пустой интерфейс
- Внутреннее устройство интерфейсов
- Интерфейсы и производительность программы
- Значение типа интерфейс и ошибки, связанные с nil
- Правила присваивания значений переменным типа интерфейс
- Опасное и безопасное приведение типов (type cast)
- Использование switch в контексте интерфейсов

# Интерфейсы: определение

**Интерфейс** — набор методов, которые надо реализовать, чтобы удовлетворить интерфейсу. Ключевое слово: `interface`.

```
type Stringer interface {  
    String() string  
}  
  
type Shape interface {  
    Area() float64  
    Perimeter() float64  
}
```

- Одному интерфейсу могут соответствовать много типов
- Тип может реализовать несколько интерфейсов

# Интерфейсы реализуются неявно

```
type Duck interface {  
    Talk() string  
    Walk()  
    Swim()  
}  
  
type Dog struct {  
    name string  
}  
  
func (d Dog) Talk() string {  
    return "AGGGRRRR"  
}  
  
func (d Dog) Walk() { }  
  
func (d Dog) Swim() { }
```

<https://goplay.space/#GWYHjaDPnLG>

# Интерфейсы реализуются неявно

```
type MyVeryOwnStringer struct { s string}

func (s MyVeryOwnStringer) String() string {
    return "my string representation of MyVeryOwnStringer"
}

func main() {
    // my string representation of MyVeryOwnStringer{}
    fmt.Println(MyVeryOwnStringer{"hello"})
}
```

```
type Stringer interface {
    String() string
}
```

<https://goplay.space/#ppTH6Ya-fX5>



# Интерфейсы: композиция

```
type Reader interface {  
    Read(p []byte) (n int, err error)  
}  
  
type Closer interface {  
    Close() error  
}  
  
type ReadCloser interface {  
    Reader  
    Closer  
}
```

# Интерфейсы: композиция

```
import "fmt"

type Greeter interface {
    hello()
}

type Stranger interface {
    Bye() string
    Greeter
    fmt.Stringer
}
```

# Интерфейсы: имена методов

Имена методов не должны повторяться:

```
type Hound interface {  
    destroy()  
    bark(int)  
}  
  
type Retriever interface {  
    Hound  
    bark() // duplicate method  
}
```

```
./prog.go:6:2: duplicate method bark
```

[https://goplay.space/#H4eJdwqD\\_kJ](https://goplay.space/#H4eJdwqD_kJ)

**interface{} is says nothing**

<https://go-proverbs.github.io/>

# Интерфейсы: any

Пустой интерфейс не содержит методов:

```
type Any interface{}
```

```
func Fprintln(w io.Writer, a ...interface{}) (n int, err error) {  
    ...  
}
```

# Интерфейсы: interface{}

Как быть?

```
func PrintAll(vals []interface{}) {  
    for _, val := range vals {  
        fmt.Println(val)  
    }  
}  
  
func main() {  
    names := []string{"stanley", "david", "oscar"}  
    PrintAll(names)  
}
```

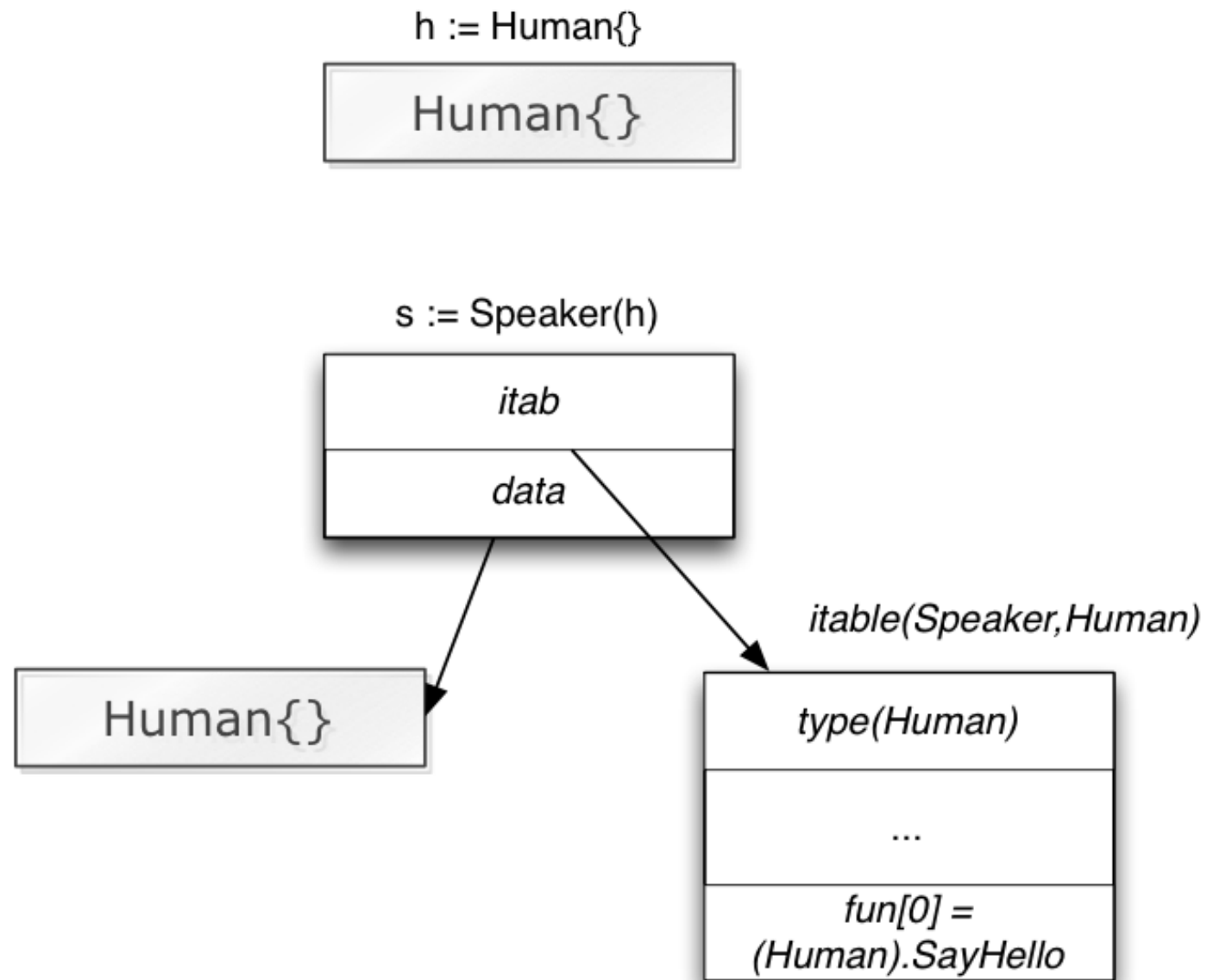
<https://goplay.space/#1w7ksGW0uXh>

# Интерфейсы изнутри

```
type Speaker interface {  
    SayHello()  
}  
  
type Human struct {  
    Greeting string  
}  
  
func (h Human) SayHello() {  
    fmt.Println(h.Greeting)  
}  
  
...  
  
var s Speaker  
h := Human{Greeting: "Hello"}  
s = Speaker(h)  
s.SayHello()
```

[https://goplay.space/#\\_P52r1iyTxj](https://goplay.space/#_P52r1iyTxj)

# Интерфейсы изнутри





# Интерфейсы изнутри: iface

```
type iface struct {  
    tab *itab // Информация об интерфейсе и типе данных внутри  
    data unsafe.Pointer // Хранимые данные  
}
```

```
// itab содержит тип интерфейса и информацию о хранимом типе.  
type itab struct {  
    inter *interfacetype // Метаданные интерфейса  
    _type *_type // Go-шный тип хранимого интерфейсом значения  
    ...  
    fun [1]uintptr // Список методов типа, удовлетворяющих интерфейсу  
}
```

[https://github.com/teh-cmc/go-internals/blob/master/chapter2\\_interfaces/README.md#anatomy-of-an-interface](https://github.com/teh-cmc/go-internals/blob/master/chapter2_interfaces/README.md#anatomy-of-an-interface)

# Интерфейсы изнутри

На этапе компиляции:

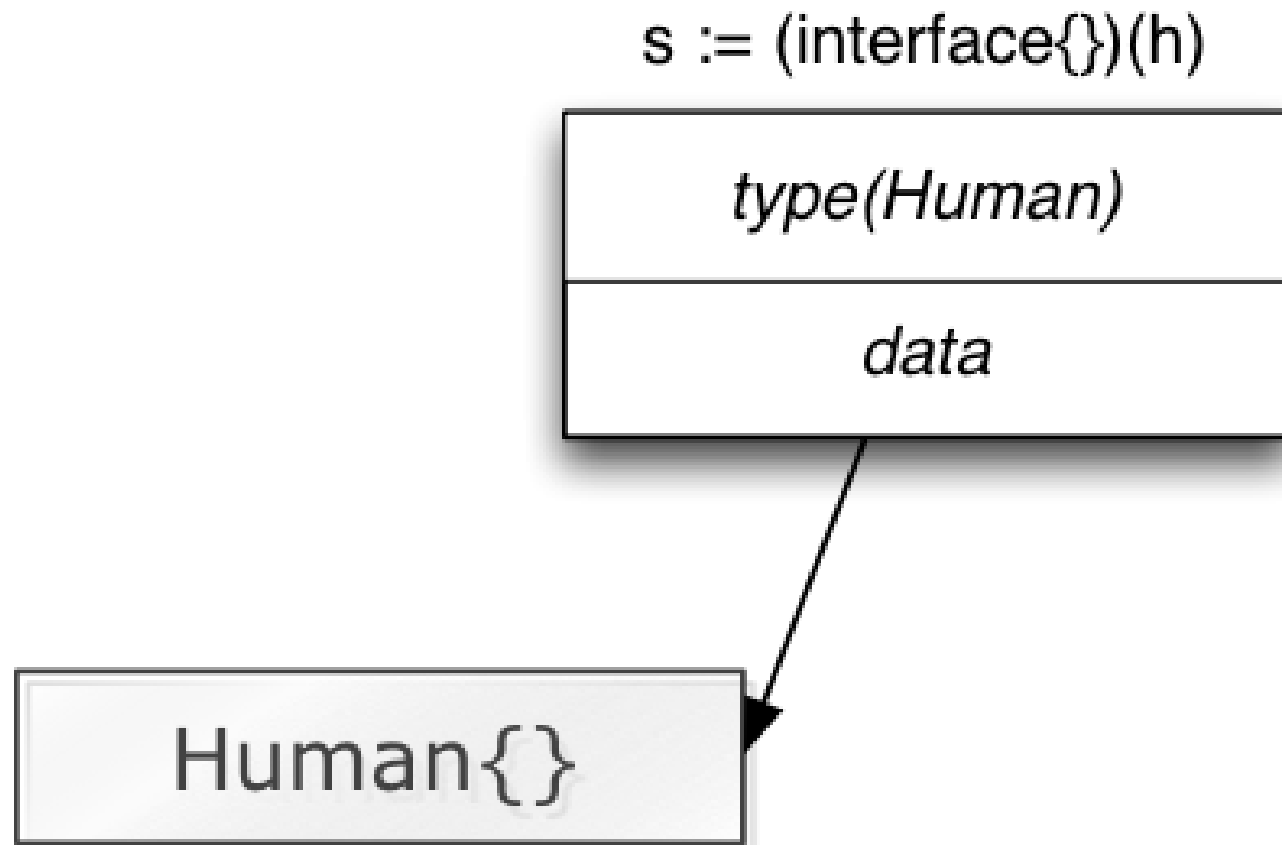
- генерируются метаданные для каждого статического типа, включая его список методов
- генерируются метаданные для каждого интерфейса, включая его список методов

И при компиляции и в рантайме в зависимости от выражения:

- сравниваются `methodset`'ы типа и интерфейса
- создается и кэшируется `itab`

```
// Создание интерфейса:  
// - аллокация места для хранения адреса ресивера  
// - получение itab:  
//     - проверка кэша  
//     - нахождение реализаций методов  
// - создание iface: runtime.convT2I  
s := Speaker(Human{Greeting: "Hello"})  
  
// Динамический диспатчинг  
// - для рантайма это вызов n-го метода s.Method_0()  
// - превращается в вызов вида s.itab.fun[0](s.data)  
s.SayHello()
```

## Интерфейсы изнутри: any



# Интерфейсы изнутри: benchmark

```
type Addifier interface {  
    Add(a, b int32) int32  
}  
  
type Adder struct { id int32 }  
  
func (adder Adder) Add(a, b int32) int32 {  
    return a + b  
}  
  
func BenchmarkDirect(b *testing.B) {  
    adder := Adder{id: 6754}  
    for i := 0; i < b.N; i++ {  
        adder.Add(10, 32)  
    }  
}  
  
func BenchmarkInterface(b *testing.B) {  
    adder := Addifier(Adder{id: 6754})  
    for i := 0; i < b.N; i++ {  
        adder.Add(10, 32)  
    }  
}
```

# Интерфейсы изнутри: benchmark

BenchmarkDirect-16	1000000000	0.2436 ns/op	0 B/op	0 allocs/op
BenchmarkInterface-16	957668390	1.157 ns/op	0 B/op	0 allocs/op

# Интерфейсы: еще раз о ресиверах

<https://goplay.space/#jm1bKNLABnB>

<https://stackoverflow.com/a/45653986>

<https://stackoverflow.com/a/48874650>

# Zero-value

nil — нулевое значение для интерфейсного типа

```
type HTTPClient interface {  
    Do(req *http.Request) (*http.Response, error)  
}  
  
func main() {  
    var c HTTPClient  
    fmt.Println("value of client is", c)  
    fmt.Printf("type of client is %T\n", c)  
    fmt.Println("(c == nil) is", c == nil)  
}
```

<https://goplay.space/#uBwvZ4bLy7T>

# Интерфейсы: опасный nil

Что выведет программа?

```
func ReadFile(fname string) error {  
    var err *os.PathError // nil  
  
    if err == nil {  
        log.Println("err")  
        return err  
    }  
  
    // Do some work...  
    return err  
}  
  
func main() {  
    if err := ReadFile(""); err != nil {  
        log.Printf("ERR: (%T, %v)", err, err)  
    } else {  
        log.Println("OK")  
    }  
}
```

<https://goplay.space/#EEsdRjph6YE>



# Интерфейсы: опасный nil

Значение интерфейсного типа равно `nil` тогда и только тогда, когда `nil` и тип, и значение.

[http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html#nil\\_in\\_nil\\_in\\_vals](http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html#nil_in_nil_in_vals)

# Правила присваиваний (assignability rules)

Если переменная реализует интерфейс `I`, то мы можем присвоить ее переменной типа интерфейс `I`.

```
type BaseStorage interface {
    Close()
}

type UsersStorage struct{}
func (UsersStorage) Close() {}

type TicketsStorage struct{}
func (TicketsStorage) Close() {}
func (TicketsStorage) GetTickets() {}

func main() {
    var s BaseStorage

    s = UsersStorage{}
    s = TicketsStorage{}
    _ = s
}
```

<https://goplay.space/#jccNcScVWMZ>

<https://medium.com/golangspec/assignability-in-go-27805bcd5874>

# Интерфейсы: присваивание

```
type MetricCollector interface {  
    Record()  
}  
  
type AudioRecorder interface {  
    Record()  
}  
  
type DummyRecorder struct{}  
func (DummyRecorder) Record() {}  
  
func main() {  
    var v1 MetricCollector = DummyRecorder{}  
    var v2 AudioRecorder = v1  
    _ = v2  
}
```

Валидно?

<https://goplay.space/#cG0FfsygGnC>

# Интерфейсы: присваивание

```
type MetricCollector interface {  
    Record()  
}  
  
type AudioRecorder interface {  
    Record()  
    Play()  
}  
  
type DummyRecorder struct{}  
func (DummyRecorder) Record() {}  
  
func main() {  
    var v1 MetricCollector = DummyRecorder{}  
    var v2 AudioRecorder = v1  
    _ = v2  
}
```

Валидно?

<https://goplay.space/#eD4R5SKtFWZ>

# Интерфейсы: присваивание

Что, если мы хотим присвоить переменной конкретного типа значение типа интерфейс ?

```
type MetricCollector interface {  
    Record()  
}  
  
type DummyRecorder struct{}  
func (DummyRecorder) Record() {}  
  
func main() {  
    var v1 MetricCollector  
    var v2 DummyRecorder = v1  
    _ = v2  
}
```

<https://goplay.space/#-0ULaHj0sv9>

# Интерфейсы: type assertion

Выражение `x.(T)` проверяет, что интерфейс `x != nil` и конкретная часть `x` имеет тип `T`:

- если `T` не интерфейс, то проверяем, что динамический тип `x` это `T`
- если `T` интерфейс: то проверяем, что динамический тип `x` его реализует

# Интерфейсы: type assertion

```
var i interface{} = "hello"

s := i.(string)
fmt.Println(s)

s, ok := i.(string)
fmt.Println(s, ok)

r, ok := i.(fmt.Stringer)
fmt.Println(r, ok)

f, ok := i.(float64)
fmt.Println(f, ok)
```

<https://goplay.space/#x-NbzVMZMUp>

# Интерфейсы: type assertion

```
var i interface{} = "hello"

f, ok := i.(float64) // 0 false
fmt.Println(f, ok)

f = i.(float64) // panic: interface conversion:
                // interface {} is string, not float64
fmt.Println(f)
```

Проверка типа возможна только для интерфейса:

```
s := 5
// Invalid type assertion: s.(int) (non-interface type int on left)
i := s.(int)
```



# Интерфейсы: type switch

Мы можем объединить проверку нескольких типов в один `type switch`:

```
func checkSignature(/* ... */, publicKey crypto.PublicKey) (err error) {  
    // ...  
  
    switch pub := publicKey.(type) {  
    case *rsa.PublicKey:  
        // ...  
    case *ecdsa.PublicKey:  
        // ...  
    case ed25519.PublicKey:  
        // ...  
    }  
    return ErrUnsupportedAlgorithm  
}
```

[src/crypto/x509/x509.go](https://golang.org/src/crypto/x509/x509.go)

# Интерфейсы: type switch

Как и в обычном `switch` мы можем объединять типы:

```
case *rsa.PublicKey, *ecdsa.PublicKey:  
    // Do some work...  
}
```

и обрабатывать `default`:

```
switch publicKey.(type) {  
default:  
    // No case for input type...  
}
```

# Практика

Необходимо реализовать функцию `processMessage` .

<https://goplay.space/#EZ2pXx3DDKA>

# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет



**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**

# Следующий вебинар

26 декабря

Горютины и каналы



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию в ЛК — можно изучать



Обязательный материал обозначен красной лентой



Спасибо за внимание!

# Приходите на следующие вебинары

Олег Венгер

Руководитель группы Защиты профилей в Wildberries

