ОСНОВЫ WEB-РАЗРАБОТКИ

Лекция 9. Обработка ошибок. Преобразование типов. Интерфейсы

Курс читают:

Шульман В.Д. @ShtuzerVD
Пелевина Т.В. @anivelat
Шабанов В.В. @ZeroHug

ПЛАН ЛЕКЦИИ

- Обработка ошибок
- Типизация
- Преобразование типов
- Приведение типов
- Интерфейсы

ОБРАБОТКА ОШИБОК


```go
package main

import (
  "errors"
  "fmt"
)

func main() {
  err := errors.New("emit macho dwarf: elf header corrupted")
  fmt.Print(err)
}
```

ОБРАБОТКА ОШИБОК


```go
package main

import (
  "fmt"
)

func main() {
  const name, id = "bueller", 17
  err := fmt.Errorf("user %q (id %d) not found", name, id)
  fmt.Print(err)
}
```

ОБРАБОТКА ОШИБОК


```go
import "log"

response, err := DoHTTPCall()
if err != nil {
    log.Println(err)
}

// только после проверки на ошибку можно делать что-то с объектом response
```

ОБРАБОТКА ОШИБОК


```go
// для простоты примера опускаем аргументы запроса и ответа
func DoHTTPCall() error {
    err := SendTCP()
    if err != nil {
        // оборачивается в виде "[название метода]: %w". %w — это плейсхолдер
        return fmt.Errorf("send tcp: %w", err)
    }

    return nil
}
```

ОБРАБОТКА ОШИБОК


```
err := DoHTTPCall()
if err != nil {
    if errors.Is(err, errTCPConnectionIssue) {
        // в случае ошибки соединения ждем 1 секунду и пытаемся сделать запрос снова
        time.Sleep(1 * time.Second)
        return DoHTTPCall()
    }

    // обработка неизвестной ошибки
    log.Println("unknown error on HTTP call", err)
}
```

ОБРАБОТКА ОШИБОК


```go
// ошибка подключения к базе данных
type ConnectionErr struct{}

func (e ConnectionErr) Error() string {
    return "connection err"
}
```

```go
err := connectDB()
if err != nil {
    // если ошибка подключения, то ждем 1 секунду и пытаемся снова
    if errors.As(err, &ConnectionErr{}) {
        log.Println("Connection error. Trying to reconnect...")
        time.Sleep(1 * time.Second)
        tries++
        continue
    }
```

ОБРАБОТКА ОШИБОК. ПАНИКА


```go
func createFile(p string) *os.File {
    fmt.Println("creating")
    f, err := os.Create(p)
    if err != nil {
        panic(err)
    }
    return f
}
```

ОБРАБОТКА ОШИБОК. ПАНИКА


```
panic: runtime error: integer divide by zero
goroutine 1 [running]:
main.divide(0x0)
        C:/Users/gabriel/articles/Golang Error handling/Code/pa
main.divide(0x1)
        C:/Users/gabriel/articles/Golang Error handling/Code/pa
```

```go
func A() {
  defer fmt.Println("Then we can't save the earth!")
  defer func() {
    if x := recover(); x != nil {
      fmt.Printf("Panic: %+v\n", x)
    }
  }()
  B()
}

func B() {
  defer fmt.Println("And if it keeps getting hotter...")
  C()
}

func C() {
  defer fmt.Println("Turn on the air conditioner...")
  Break()
}

func Break() {
  defer fmt.Println("If it's more than 30 degrees...")
  panic(errors.New("Global Warming!!!"))
}

func main() {
  A()
}
```

```
4. If it's more than 30 degrees...
3. Turn on the air conditioner...
2. And if it keeps getting hotter...
Panic: Global Warming!!!
1. Then we can't save the earth!
```

ТИПИЗАЦИЯ

Типизация — это способ присваивать типы данным в программе.

Существует три критерия, по которым можно сравнивать типизацию в разных языках программирования:

1. **Статическая или динамическая.** Этот критерий показывает, когда происходит проверка согласования типов: при компиляции или во время выполнения приложения.
2. **Сильная или слабая.** То есть насколько строго соблюдаются правила работы с типами: можно ли менять тип переменной или приводить один тип к другому без явного указания.
3. **Явная или неявная.** Нужно ли указывать тип переменной при ее объявлении или же тип определяется по ее значению.

ТИПИЗАЦИЯ

	СТАТИЧЕСКАЯ/ ДИНАМИЧЕСКАЯ	СИЛЬНАЯ/ СЛАБАЯ	ЯВНАЯ/ НЕЯВНАЯ
C#	статическая	сильная	явная
C++	статическая	слабая	явная
DELPHI	статическая	сильная	явная
GO	статическая	сильная	неявная
JAVA	статическая	сильная	явная
JAVASCRIPT	динамическая	слабая	неявная

ТИПИЗАЦИЯ

Go — статическая и сильная, неявная типизация

В Go тип переменной определяется на этапе компиляции и синтаксически может быть опущен при её объявлении, если сразу инициализировать переменную значением.

В Go сильная типизация: все преобразования нужно производить явным образом. Использовать значения разных типов в выражениях запрещено.

ПРЕОБРАЗОВАНИЕ ТИПОВ. ЧИСЛА


```go
package main

import "fmt"

func main() {
    var integerNumber int = 77
    var floatNumber float64 = float64(integerNumber)

    fmt.Printf("Integer: %d, Float: %f\n", integerNumber, floatNumber)
}
```

Вывод:

```
Integer: 77, Float: 77.000000
```

ПРЕОБРАЗОВАНИЕ ТИПОВ. ЧИСЛА


```go
package main

import "fmt"

func main() {
    var floatNumber float64 = 73.8
    var integerNumber int = int(floatNumber)

    fmt.Printf("Float: %f, Integer: %d\n", floatNumber, integerNumber)
}
```

Вывод:

```
Float: 73.800000, Integer: 73
```

ПРЕОБРАЗОВАНИЕ ТИПОВ. ЧИСЛА


```go
package main

import "fmt"

func main() {
    a := 10 / 3
    fmt.Println(a)
}
```

Вывод:

```
3
```

ПРЕОБРАЗОВАНИЕ ТИПОВ. ЧИСЛА


```go
package main

import "fmt"

func main() {
    b := 7.5 / 2
    fmt.Println(b)
}
```

Вывод:

```
3.75
```

ПРЕОБРАЗОВАНИЕ ТИПОВ. СТРОКИ


```go
func main() {
    // Число в строку
    numberAsString := strconv.Itoa(42)
    fmt.Printf("Number as string: %s\n", numberAsString)

    // Строка в число
    stringNumber := "123"
    number, err := strconv.Atoi(stringNumber)
    if err == nil {
        fmt.Printf("String as number: %d\n", number)
    } else {
        fmt.Println("Error converting string to number")
    }
}
```

Вывод:

```
Number as string: 42
String as number: 123
```

ПРЕОБРАЗОВАНИЕ ТИПОВ. СТРОКИ


```go
package main

import "fmt"

func main() {
    f := 789.12
    fmt.Println("Анна потратила " + fmt.Sprint(f) + " рублей.")
}
```

Теперь программа сообщит, что «Анна потратила 789.12 рублей»:

ПРЕОБРАЗОВАНИЕ ТИПОВ. СТРОКИ


```
 8      originalString := "Моя строка"
 9
10      byteSlice := []byte(originalString)
11
12      convertedString := string(byteSlice)
13
14      fmt.Println(originalString)
15
16      fmt.Println(byteSlice)
17
18      fmt.Println(convertedString)
```

```
Моя строка
[208 156 208 190 209 143 32 209 129 209 130 209 128 208 190 208 186 208 176]
Моя строка
```

ИНТЕРФЕЙСЫ


```
type имя_интерфейса interface{
    определения_функций
}
```

```
type vehicle interface{
    move()
}
```

ИНТЕРФЕЙСЫ

Интерфейс представляет своего рода контракт, которому должен соответствовать тип данных.

Чтобы тип данных соответствовал некоторому интерфейсу, данный тип должен реализовать в виде методов все функции этого интерфейса.

ИНТЕРФЕЙСЫ


```go
type Vehicle interface{
    move()
}

// структура "Автомобиль"
type Car struct{ }

// структура "Самолет"
type Aircraft struct{}


func (c Car) move(){
    fmt.Println("Автомобиль едет")
}
func (a Aircraft) move(){
    fmt.Println("Самолет летит")
}
```

```go
func main() {

    var tesla Vehicle = Car{}
    var boing Vehicle = Aircraft{}
    tesla.move()
    boing.move()
}
```

Поскольку структуры Car и Aircraft реализуют интерфейс Vehicle, то мы можем определить переменные данного интерфейса, передав им объекты структур:

Отчетливо видно, что обе функции driveCar и driveAircraft фактически идентичны, они выполняют одни и те же действия, только для разных типов. И было бы неплохо, если можно было бы определить одну обобщенную функцию для разных типов.

ИНТЕРФЕЙСЫ


```
 9  func (c Car) move(){
10      fmt.Println("Автомобиль едет")
11  }
12  func (a Aircraft) move(){
13      fmt.Println("Самолет летит")
14  }
15
16  func driveCar(c Car){
17      c.move()
18  }
19  func driveAircraft(a Aircraft){
20      a.move()
21  }
```

```
23  func main() {
24
25      var tesla Car = Car{}
26      var boing Aircraft = Aircraft{}
27      driveCar(tesla)
28      driveAircraft(boing)
29  }
```

ИНТЕРФЕЙСЫ


```
4  type Vehicle interface{
5      move()
6  }
7
8  func drive(vehicle Vehicle){
9      vehicle.move()
10 }
11
12 type Car struct{ }
13 type Aircraft struct{}
14
15
16 func (c Car) move(){
17     fmt.Println("Автомобиль едет")
18 }
19 func (a Aircraft) move(){
20     fmt.Println("Самолет летит")
21 }
```

```
23 func main() {
24
25     tesla := Car{}
26     boing := Aircraft{}
27     drive(tesla)
28     drive(boing)
29 }
```

Теперь вместо двух функций определена одна общая функция - drive(), которая в качесте параметра принимает значение типа Vehicle. Поскольку этому интерфейсу соответствуют обе структуры Car и Aircraft, то мы можем передавать эти структуры в функцию drive в качестве аргументов

ИНТЕРФЕЙСЫ

К сожалению, []SomeType не будет удовлетворять []SomeInterface (даже если SomeType удовлетворяет SomeInterface)

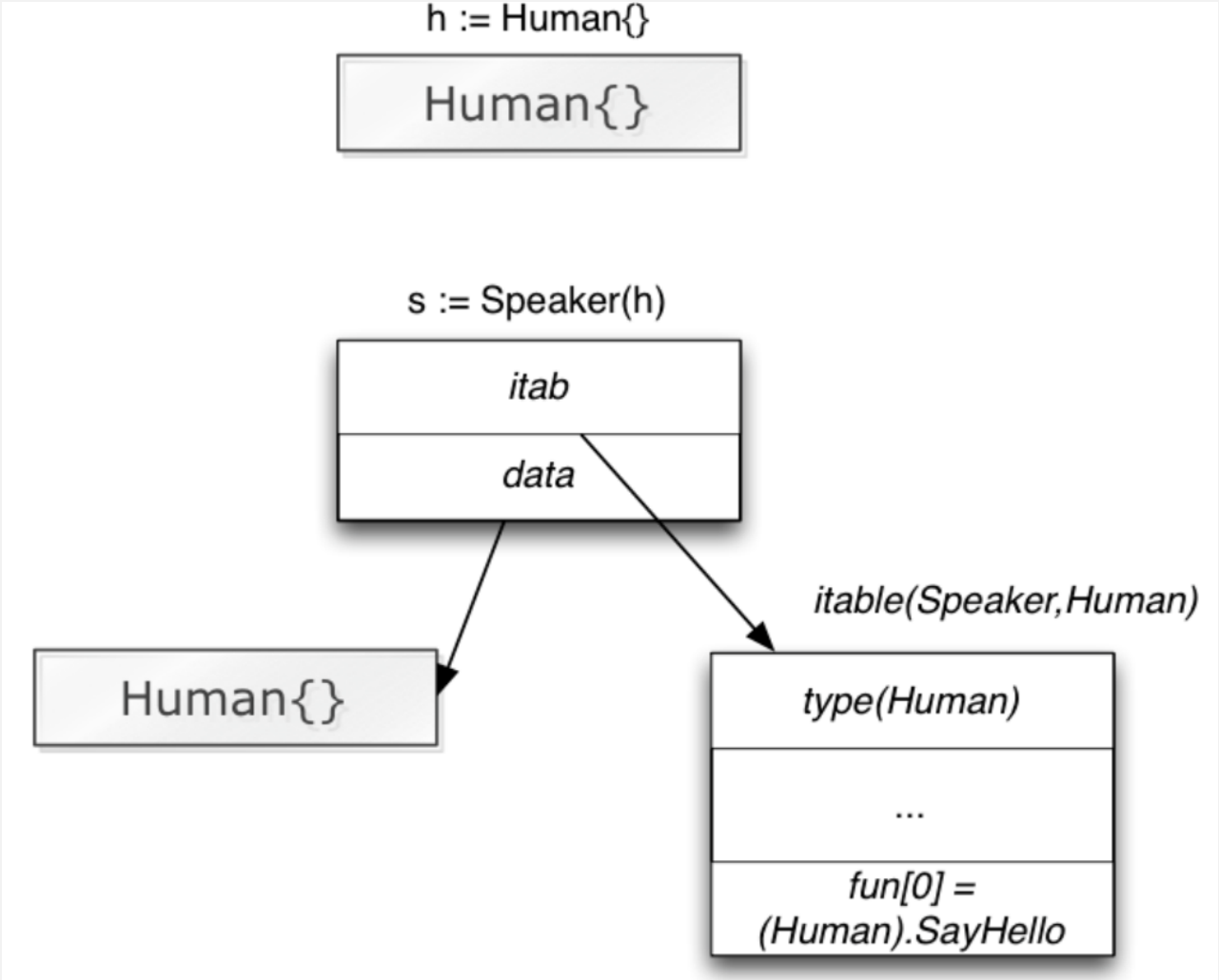
https://en.wikipedia.org/wiki/Covariance_and_contravariance_(computer_science)

ИНТЕРФЕЙСЫ


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

ИНТЕРФЕЙСЫ


```go
package main

import "fmt"

type Speaker interface {
        SayHello()
}

type Human struct {
        Greeting string
}

func (Human) SayHello() {
        fmt.Println("Hello")
}

func main() {
        h := Human{Greeting: "Hello"}
        s := Speaker(h)
        h.Greeting = "Meow"
        s.SayHello()
}
```



ИНТЕРФЕЙСЫ

1. Интерфейсы помогают уменьшить дублирование, то есть количество шаблонного кода.
2. Они облегчают использование в модульных тестах заглушек вместо реальных объектов.
3. Будучи архитектурным инструментом, интерфейсы помогают отвязывать части вашей кодовой базы.

ИНТЕРФЕЙСЫ. ПУСТОЙ ИНТЕРФЕЙС


```go
package main

import "fmt"


func main() {
    person := make(map[string]interface{}, 0)

    person["name"] = "Alice"
    person["age"] = 21
    person["height"] = 167.64

    fmt.Printf("%+v", person)
}
```

```
map[age:21 height:167.64 name:Alice]
```

Пустой интерфейсный тип *не описывает методы*. У него нет правил. И поэтому любой объект удовлетворяет пустому интерфейсу.

ИНТЕРФЕЙСЫ. ПУСТОЙ ИНТЕРФЕЙС


```go
func main() {
    person := make(map[string]interface{}, 0)

    person["name"] = "Alice"
    person["age"] = 21
    person["height"] = 167.64

    person["age"] = person["age"] + 1

    fmt.Printf("%+v", person)
}
```

```
invalid operation: person["age"] + 1 (mismatched types interface {} and int)
```

ИНТЕРФЕЙСЫ. ПРИВЕДЕНИЕ ТИПОВ


```go
var a interface{} = 10
n := a.(int)
fmt.Println(n) // 10

// panic: interface conversion: interface {} is int, not string
m := a.(string)
fmt.Println(m)
```

ИНТЕРФЕЙСЫ. ПРИВЕДЕНИЕ ТИПОВ


```go
var a interface{} = 10

if m, ok := a.(string); ok {
    fmt.Println(m)
} else {
    // у переменной 'a' тип НЕ строка
    fmt.Println("not a string")
}
```

ИНТЕРФЕЙСЫ. ПРИВЕДЕНИЕ ТИПОВ


```go
func main() {
    person := make(map[string]interface{}, 0)

    person["name"] = "Alice"
    person["age"] = 21
    person["height"] = 167.64

    age, ok := person["age"].(int)
    if !ok {
        log.Fatal("could not assert value to int")
        return
    }

    person["age"] = age + 1

    log.Printf("%+v", person)
}
```

ИНТЕРФЕЙСЫ. ПРИВЕДЕНИЕ ТИПОВ


```go
type Vehicle struct {
    Type  string
    Speed int
}

func main() {
    var v interface{}
    v = Vehicle{Type: "Car", Speed: 100}

    fmt.Println(v.(Vehicle).Type)  // Приведение типа для доступа к полю Type
    fmt.Println(v.(Vehicle).Speed) // Приведение типа для доступа к полю Speed
}
```

ИНТЕРФЕЙСЫ. ПРИВЕДЕНИЕ ТИПОВ


```
func ValueInfo(obj interface{}) {
        switch val := obj.(type) {
        case string:
                fmt.Printf("Длина строки: %d\\n", len(val))
        case []int:
                fmt.Println("Емкость слайса:", cap(val))
        default:
                fmt.Printf("Тип %T", val)
        }
}

func main() {
        ValueInfo("str")       // Длина строки: 3
        ValueInfo([]int{1, 2}) // Емкость слайса: 2
        ValueInfo(true)        // Тип bool
```

ИНТЕРФЕЙСЫ. ПРИВЕДЕНИЕ ТИПОВ


```
events := []Event{MouseClickEvent{X: 100, Y: 200}, KeyEvent{Key: "A"}, NetworkEvent{URL: "https://www.example.com"}}

for _, event := range events {
    switch event := event.(type) {
    case MouseClickEvent:
        fmt.Println("Handling mouse click event.")
        event.Process()
    case KeyEvent:
        fmt.Println("Handling key event.")
        event.Process()
    case NetworkEvent:
        fmt.Println("Handling network event.")
        event.Process()
    default:
        fmt.Println("Unknown event type.")
    }
}
```

ИСТОЧНИКИ

- https://ru.hexlet.io/courses/go-basics/lessons/errors-handling/theory_unit
- https://practicum.yandex.ru/trainer/go-basics/lesson/cb227d89-1ca3-4d22-a2f1-c5d3a35d2c6e/
- https://help.sweb.ru/konvertaciya-tipov-dannyh-v-go_1309.html
- https://metanit.com/go/tutorial/6.1.php
- https://dzen.ru/a/ZVO5jOK50TGCWBzJ

СПАСИБО ЗА ВНИМАНИЕ :3

29.10.2024