接口1

作者：少林之巅

目录

接口介绍和定义

1. 接口定义了一个对象的行为规范

   A. 只定义规范，不实现

   B. 具体的对象需要实现规范的细节

接口介绍和定义

2. Go中接口定义

   A. type 接口名字 interface

   B. 接口里面是一组方法签名的集合

```
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

接口介绍和定义

3. Go中接口的实现

A. 一个对象只要包含接口中的方法，那么就实现了这个接口

B. 接口类型的变量可以保存具体类型的实例

```
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

4. 接口实例

A. 一个公司需要计算所有职员的薪水

B. 每个职员的薪水计算方式不同

```
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

接口介绍和定义

5. 接口类型变量

A. var a Animal

B. 那么a能够存储所有实现Animal接口的对象实例

```
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

空接口和类型断言

6. 空接口

A. 空接口没有定义任何方法

B. 所以任何类型都实现了空接口

```
interface {

}
```

空接口和类型断言

7. 空接口

```go
package main

import (
    "fmt"
)

func describe(i interface{}) {
    fmt.Printf("Type = %T, value = %v\n", i, i)
}

func main() {
    s := "Hello World"
    describe(s)
    i := 55
    describe(i)
    strt := struct {
        name string
    }{
        name: "Naveen R",
    }
    describe(strt)
}
```

空接口和类型断言

8. 类型断言

A. 如何获取接口类型里面存储的具体的值呢?

```go
package main

import (
    "fmt"
)

func assert(i interface{}) {
    s := i.(int) //get the underlying int value from i
    fmt.Println(s)
}
func main() {
    var s interface{} = 56
    assert(s)
}
```

空接口和类型断言

9. 类型断言

A. 类型断言的坑！

```go
package main

import (
    "fmt"
)

func assert(i interface{}) {
    s := i.(int)
    fmt.Println(s)
}
func main() {
    var s interface{} = "Steven Paul"
    assert(s)
}
```

空接口和类型断言

10. 类型断言

A. 如何解决，引入 ok判断机制!　　v, ok := i.(T)

```go
package main

import (
    "fmt"
)

func assert(i interface{}) {
    v, ok := i.(int)
    fmt.Println(v, ok)
}
func main() {
    var s interface{} = 56
    assert(s)
    var i interface{} = "Steven Paul"
    assert(i)
}
```

空接口和类型断言

11. 类型断言

A. type switch。

问题需要转两次?

```go
import (
    "fmt"
)

func findType(i interface{}) {
    switch i.(type) {
    case string:
        fmt.Printf("I am a string and my value is %s\n", i.(string))
    case int:
        fmt.Printf("I am an int and my value is %d\n", i.(int))
    default:
        fmt.Printf("Unknown type\n")
    }
}
func main() {
    findType("hello")
    findType(77)
    findType(89.98)
}
```

空接口和类型断言

12. 类型断言

A. type switch另外一种写法，解决转两次的问题

```go
package main

import (
    "fmt"
)

func findType(i interface{}) {
    switch v := i.(type) {
    case string:
        fmt.Printf("I am a string and my value is %s\n", v)
    case int:
        fmt.Printf("I am an int and my value is %d\n", v)
    default:
        fmt.Printf("Unknown type\n")
    }
}
func main() {
    findType("hello")
    findType(77)
    findType(89.98)
}
```

指针接收和值接收

13. 指针接收

```go
package main

import "fmt"
type Animal interface {
    Talk()
    Run()
    Eat()
}
type Bird struct {
    name string
}
func (b *Bird) Talk() {
    fmt.Println("bird is talk")
}
func (b *Bird) Run() {
    fmt.Println("bird is running")
}
func (b *Bird) Eat() {
    fmt.Println("bird is eat")
}
func main() {
    var b Bird
    var a Animal
    a = b
}
```

实现多接口

14. 同一个类型可以实现多个接口

实现多接口

15. 接口嵌套，和结构体嵌套类似

```
4   type Animal interface {
5       Talk()
6       Run()
7       Eat()
8   }
9
10  type Describle interface{
11      Describle()
12  }
13
14  type AvanceAnimal interface{
15      Animal
16      Describle
17  }
18
```