Go培训第六天

tony

Outline

1. 接口

2. 反射

3. 课后作业

接口

1. 定义

Interface类型可以定义一组方法，用来表示一个对象的行为特征。interface不能包含任何变量。

接口

2. 定义

比如：

```
type Animal interface{
    Talk(参数列表) 返回值列表
    Eat(参数列表) 返回值列表
    …
}
```

接口

3. interface类型是引用类型

```
type Animal interface{

        Talk(参数列表) 返回值列表
        …
}

var a Animal
a.Talk()
```

接口

4. 接口实现

a. Golang中的接口，不需要显示的实现。只要一个对象，实现了接口类型中的所有方法，那么这个对象就实现这个接口。

b. 如果一个对象实现了多个interface类型的方法，那么这个对象就实现了多个接口。

接口

6. 多态

一种事物的多种形态，都可以按照统一的接口进行操作

类型断言

11. 空接口，**Interface{}**

空接口没有任何方法，所以所有类型都实现了空接口。

```
var a int
var b interface{}
b  = a
```

7. 接口嵌套

一个接口可以嵌套在另外的接口，如下所示：

```
 type ReadWrite interface {
        Read(b Buffer) bool
        Write(b Buffer) bool
}
type Lock interface {
        Lock()
        Unlock()
}
type File interface {
        ReadWrite
        Lock
        Close()
}
```

类型断言

8. 类型断言。如果我们反向要知道这个接口变量里面实际存储的是哪个类型的对象

可以采用以下方法进行转换：

```
var t int
var x interface{}
x = t
y = x.(int)   //转成int
```

```
var t int
var x interface{}
x = t
y, ok = x.(int)   //转成int，带检查
```

类型断言

9. 练习，写一个函数判断传入参数的类型

```
func classifier(items ...interface{}) {
        for i, x := range items {
                switch x.(type) {
                 case bool:       fmt.Printf(“param #%d is a bool\n”, i)
                 case float64:    fmt.Printf(“param #%d is a float64\n”, i)
                 case int, int64: fmt.Printf(“param #%d is an int\n”, i)
                 case nil: fmt.Printf(“param #%d is nil\n”, i)
                 case string: fmt.Printf(“param #%d is a string\n”, i)
                  default: fmt.Printf(“param #%d’s type is unknown\n”, i)
        }
}
```

类型断言

10. 类型断言，采用type switch方式

```
switch t := areaIntf.(type) {
case *Square:
        fmt.Printf(“Type Square %T with value %v\n”, t, t)
case *Circle:
       fmt.Printf(“Type Circle %T with value %v\n”, t, t)
case float32:
       fmt.Printf(“Type float32 with value %v\n”, t)
case nil:
        fmt.Println(“nil value: nothing to check?”)
default:
        fmt.Printf(“Unexpected type %T”, t)
}
```

接口

12. 判断一个变量是否实现了指定接口

判断一个变量是否实现了指定接口

```
type Stringer interface {
        String() string
}
var v MyStruct
if sv, ok := v.(Stringer); ok {
        fmt.Printf(“v implements String(): %s\n”, sv.String());
}
```

接口示例

13. 实现一个负载均衡调度算法，支持随机、轮询等算法

反射

1. 反射：可以在运行时动态获取变量的相关信息

   Import (“reflect”)

   两个函数：

   a. reflect.TypeOf，获取变量的类型，返回reflect.Type类型

   b. reflect.ValueOf，获取变量的值，返回reflect.Value类型

   c. reflect.Value.Kind，获取变量的类别，返回一个常量

   d. reflect.Value.Interface()，转换成interface{}类型

反射

2. reflect.Value.Kind()方法返回的常量

```
const (
        Invalid Kind = iota
        Bool
        Int
        Int8
        Int16
        Int32
        Int64
        Uint
        Uint8
        Uint16
        Uint32
        Uint64
        Uintptr
        Float32
        Float64
        Complex64
        Complex128
        Array
        Chan
        Func
        Interface
        Map
        Ptr
        Slice
        String
        Struct
        UnsafePointer
)
```

反射

3. 练习：

```
package main

import (
        "fmt"
        "reflect"
)

func main() {
        var x float64 = 3.4
        fmt.Println("type:", reflect.TypeOf(x))
        v := reflect.ValueOf(x)
        fmt.Println("value:", v)
        fmt.Println("type:", v.Type())
        fmt.Println("kind:", v.Kind())
        fmt.Println("value:", v.Float())

        fmt.Println(v.Interface())
        fmt.Printf("value is %5.2e\n", v.Interface())
        y := v.Interface().(float64)
        fmt.Println(y)
}
```

4. 获取变量的值：

reflect.ValueOf(x).Float()

reflect.ValueOf(x).Int()

reflect.ValueOf(x).String()

reflect.ValueOf(x).Bool()

5. 通过反射的来改变变量的值

reflect.Value.SetXX相关方法，比如:

reflect.Value.SetFloat()，设置浮点数

reflect.Value.SetInt()，设置整数

reflect.Value.SetString()，设置字符串

6. 练习

```
package main

import (
	"fmt"
	"reflect"
)

func main() {

	var a float64
	fv := reflect.ValueOf(a)
	fv.SetFloat(3.3)
	fmt.Printf("%v\n", a)
}
```

反射

6. 练习

崩溃了

反射

7. 崩溃的原因

还是值类型和引用类型的原因

v := reflect.ValueOf(x)

v是x的一个拷贝，修改v，x不会修改!

8. 解决方法，传地址！

```
package main

import (
        "fmt"
        "reflect"
)

func main() {

        var a float64
        fv := reflect.ValueOf(&a)
        fv.Elem().SetFloat(3.3)
        fmt.Printf("%v\n", a)
}
```

其中fv.Elem()用来获取指针指向的变量，相当于：

```
var a *int;
*a = 100
```

反射

9. 用反射操作结构体

a. reflect.Value.NumField()获取结构体中字段的个数

b. reflect.Value.Method(n).Call来调用结构体中的方法

反射

10. 练习，通过反射操作结构体

```
package main

import (
	"fmt"
	“reflect"
)

type Test struct {
	s1 string
	s2 string
	s3 string
}
func (n Test) String() string {
	return n.s1 + "-" + n.s2 + "-" + n.s3
}
var secret interface{} = Test{"Ada", "Go", "Oberon"}

func main() {
	value := reflect.ValueOf(secret) // <main.NotknownType Value>
	typ := reflect.TypeOf(secret)    // main.NotknownType
	fmt.Println(typ)

	knd := value.Kind() // struct
	fmt.Println(knd)

	for i := 0; i < value.NumField(); i++ {
		fmt.Printf("Field %d: %v\n", i, value.Field(i))
		//value.Field(i).SetString("C#")
	}

	results := value.Method(0).Call(nil)
	fmt.Println(results) // [Ada - Go - Oberon]
}
```

反射

10. 练习2，通过反射修改结构体

```
package main

import (
	"fmt"
	"reflect"
)

type T struct {
	A int
	B string
}

func main() {
	t := T{23, "skidoo"}
	s := reflect.ValueOf(&t).Elem()
	typeOfT := s.Type()
	for i := 0; i < s.NumField(); i++ {
		f := s.Field(i)
		fmt.Printf("%d: %s %s = %v\n", i,
			typeOfT.Field(i).Name, f.Type(), f.Interface())
	}
	s.Field(0).SetInt(77)
	s.Field(1).SetString("Sunset Strip")
	fmt.Println("t is now", t)
}
```

课后工作

1. 实现一个图书管理系统v2，具有以下功能：

    a. 增加用户登录、注册功能

    b. 增加借书过期的图书界面

    c. 增加显示热门图书的功能，被借次数最多的top10

    d. 增加查看某个人的借书记录的功能