# **Go 语言面向对象编程**

## **实验简介**

区别于上一节实验，这一节实验中我们将学习 Go 语言的面向对象编程。

## **一、实验说明**

本课程所有源代码，可以在XfceTerminal中通过以下方式克隆到实验环境：

$ git clone http://git.shiyanlou.com/shiyanlou/Golang\_Programming

## **二. 自定义类型以及结构体**

在讲解Go语言面向对象内容之前，需要说明下Go语言的代码是以包结构来组织的，且如果标示符(变量名，函数名，自定义类型等)如果以大写字母开头那么这些标示符是可以导出的，可以在任何导入了定义该标示符的包的包中直接使用。Go语言中的面向对象和C++，Java中的面向对象不同，因为Go语言不支持继承，Go语言只支持聚合。

### **1. 自定义类型**

在之前的课程中我们以及提到在Go语言中我们可以自定义类型，其语法如下:

* ****type**** typeName typeSpecification

其中，typeName可以是一个包或者函数内唯一合法的Go标示符。typeSpecification 可以是任何内置的类型，一个接口或者是一个结构体。所谓结构体，它的字段是由其他类型或者接口组成。例如我们通过结构体定义了一下类型:

type ColorPoint struct {

color.Color // 匿名字段(嵌入)

x, y int // 具名字段(聚合)

}

以上代码我们通过结构体自定义了类型ColorPoint，结构体中color.Color字段是Color包的类型color，这个字段没有名字，所以被称为匿名的，也是嵌入字段。字段x和y是有变量名的，所以被称为具名字段。假如我们创建了类型ColorPoint的一个值point（通过语法：point := ColorPoint{} 创建），那么这些字段可以通过point.Color、point.x、point.y访问。其他面向对象语言中的"类(class)"、"对象(object)"、"实例(instance)"在Go语言中我们完全避开使用。相反的我们使用"类型(type)"和其对应的"值"，其中自定义类型的值可以包含方法。

### **2. 方法**

方法是作用在自定义类型上的一类特殊函数，通常自定义类型的值会被传递给该函数，该值可能是以指针或者复制值的形式传递。定义方法和定义函数几乎相同，只是需要在func关键字和方法名之间必须写上接接受者。例如我们给类型Count定义了以下方法:

type Count int

func (count \*Count) Increment() { \*count++ } // 接受者是一个`Count`类型的指针

func (count \*Count) Decrement() { \*count-- }

func (count Count) IsZero() bool { return count == 0 }

以上代码中，我们在内置类型int的基础上定义了自定义类型Count，然后给该类型添加了Increment()、Decrement()和IsZero()方法，其中前两者的接受者为Count类型的指针，后一个方法接收Count类型的值。

类型的方法集是指可以被该类型的值调用的所有方法的集合。

一个指向自定义类型的值的指针，它的方法集由该类型定义的所有方法组成，无论这些方法接受的是一个值还是一个指针。如果在指针上调用一个接受值的方法，Go语言会聪明地将该指针解引用。

一个自定义类型值的方法集合则由该类型定义的接收者为值类型的方法组成，但是不包括那些接收者类型为指针的方法。

其实这些限制Go语言帮我们解决的非常好，结果就是我们可以在值类型上调用接收者为指针的方法。假如我们只有一个值，仍然可以调用一个接收者为指针类型的方法，这是因为Go语言会自动获取值的地址传递给该方法，前提是该值是可寻址的。

在以上定义的类型Count中，\*Count方法集是Increment(), Decrement()和IsZero()，Count的值的方法集是IsZero()。但是因为Count类型的是可寻址的，所以我们可以使用Count的值调用全部的方法。 另外如果结构体的字段也有方法，我们也可以直接通过结构体访问字段中的方法。下面让我们练习下，创建源文件struct\_t.go，输入以下代码:

package main

import "fmt"

type Count int // 创建自定义类型 Count

func (count \*Count) Increment() { \*count++ } // Count类型的方法

func (count \*Count) Decrement() { \*count-- }

func (count Count) IsZero() bool { return count == 0 }

type Part struct { // 基于结构体创建自定义类型 Part

stat string

Count // 匿名字段

}

func (part Part) IsZero() bool { // 覆盖了匿名字段Count的IsZero()方法

return part.Count.IsZero() && part.stat == "" // 调用了匿名字段的方法

}

func (part Part) String() string { // 定义String()方法，自定义了格式化指令%v的输出

return fmt.Sprintf("<<%s, %d>>", part.stat, part.Count)

}

func main() {

var i Count = -1

fmt.Printf("Start \"Count\" test:\nOrigin value of count: %d\n", i)

i.Increment()

fmt.Printf("Value of count after increment: %d\n", i)

fmt.Printf("Count is zero t/f? : %t\n\n", i.IsZero())

fmt.Println("Start: \"Part\" test:")

part := Part{"232", 0}

fmt.Printf("Part: %v\n", part)

fmt.Printf("Part is zero t/f? : %t\n", part.IsZero())

fmt.Printf("Count in Part is zero t/f?: %t\n", part.Count.IsZero()) // 尽管覆盖了匿名字段的方法，单还是可以访问

}

以上代码中，我们创建了Count类型，然后在其基础上又创建了结构体类型Part。我们为Count类型定义了3个方法，并在Part类型中创建了方法IsZero() 覆盖了其匿名字段Count中IsZero()方法。但是我们还是可以二次访问到匿名字段中被覆盖的方法。执行代码，输出如下：

$ go run struct\_t.goStart "Count" test:

Origin value of count: -1Value of count after increment: 0Count is zero t/f? : true

Start: "Part" test:

Part: <<232, 0>>

Part is zero t/f? : falseCount in Part is zero t/f?: true

## **三. 接口**

### **1. 接口基础**

之所以说Go语言的面向对象很灵活，很大一部分原因是由于接口的存在。接口是一个自定义类型，它声明了一个或者多个方法签名，任何实现了这些方法的类型都实现这个接口。infterface{}类型是声明了空方法集的接口类型。任何一个值都满足interface{}类型，也就是说如果一个函数或者方法接收interface{}类型的参数，那么任意类型的参数都可以传递给该函数。接口是完全抽象的，不能实例化。接口能存储任何实现了该接口的类型。直接看例子吧，创建源文件interface\_t.go，输入以下代码：

package main

import "fmt"

type Human struct { // 结构体

name string

age int

phone string

}

//Human实现SayHi方法

func (h Human) SayHi() {

fmt.Printf("Hi, I am %s you can call me on %s\n", h.name, h.phone)

}

//Human实现Sing方法

func (h Human) Sing(lyrics string) {

fmt.Println("La la la la...", lyrics)

}

type Student struct {

Human //匿名字段

school string

loan float32

}

type Employee struct {

Human //匿名字段

company string

money float32

}

// Employee重载Human的SayHi方法

func (e Employee) SayHi() {

fmt.Printf("Hi, I am %s, I work at %s. Call me on %s\n", e.name,

e.company, e.phone)

}

// Interface Men被Human,Student和Employee实现// 因为这三个类型都实现了这两个方法

type Men interface {

SayHi()

Sing(lyrics string)

}

func main() {

mike := Student{Human{"Mike", 25, "222-222-XXX"}, "MIT", 0.00}

paul := Student{Human{"Paul", 26, "111-222-XXX"}, "Harvard", 100}

sam := Employee{Human{"Sam", 36, "444-222-XXX"}, "Golang Inc.", 1000}

Tom := Employee{Human{"Tom", 37, "222-444-XXX"}, "Things Ltd.", 5000}

//定义Men类型的变量i

var i Men

//i能存储Student

i = mike

fmt.Println("This is Mike, a Student:")

i.SayHi()

i.Sing("November rain")

//i也能存储Employee

i = Tom

fmt.Println("This is Tom, an Employee:")

i.SayHi()

i.Sing("Born to be wild")

//定义了slice Men

fmt.Println("Let's use a slice of Men and see what happens")

x := make([]Men, 3)

//这三个都是不同类型的元素，但是他们实现了interface同一个接口

x[0], x[1], x[2] = paul, sam, mike

for \_, value := range x {

value.SayHi()

}

}

以上代码中，接口类型声明的变量能存储任何实现了该接口的类型的值。运行代码，输出如下:

go run interface\_t.go

This is Mike, a Student:

Hi, I am Mike you can call me on 222-222-XXX

La la la la... November rain

This is Tom, an Employee:

Hi, I am Tom, I work at Things Ltd.. Call me on 222-444-XXX

La la la la... Born to be wild

Let's use a slice of Men and see what happens

Hi, I am Paul you can call me on 111-222-XXX

Hi, I am Sam, I work at Golang Inc.. Call me on 444-222-XXX

Hi, I am Mike you can call me on 222-222-XXX

### **2. 接口变量值的类型**

我们知道接口类型声明的变量里能存储任何实现了该接口的类型的值。有的时候我们需要知道这个变量里的值的类型，那么需要怎么做呢？其实在之前的课程中我们就已经学习过了，可以使用类型断言，或者是switch类型判断分支。以下的例子interface\_t1.go我们使用了switch类型判断分支。

package main

import (

"fmt"

"strconv")

type Element interface{}

type List []Element

type Person struct {

name string

age int

}

// 实现了fmt.Stringer接口

func (p Person) String() string {

return "(name: " + p.name + " - age: " + strconv.Itoa(p.age) + " years)"

}

func main() {

list := make(List, 3)

list[0] = 1 //an int

list[1] = "Hello" //a string

list[2] = Person{"Dennis", 70}

for index, element := range list {

switch value := element.(type) { // switch类型判断开关

case int:

fmt.Printf("list[%d] is an int and its value is %d\n", index, value)

case string:

fmt.Printf("list[%d] is a string and its value is %s\n", index, value)

case Person:

fmt.Printf("list[%d] is a Person and its value is %s\n", index, value)

default:

fmt.Println("list[%d] is of a different type", index)

}

}

}

运行结果：

$ go run interface\_t1.go

list[0] is an int and its value is 1

list[1] is a string and its value is Hello

list[2] is a Person and its value is (name: Dennis - age: 70 years)

### **3. 嵌入interface**

在前面的课程中我们已经知道在结构体中可以嵌入匿名字段，其实在接口里也可以再嵌入接口。如果一个interface1作为interface2的一个嵌入字段，那么interface2隐式的包含了interface1里的方法。如下例子中, Interface2包含了Interface1的所有方法。

type Interface1 interface {

Send()

Receive()

}

type Interface2 interface {

Interface1

Close()

}

## **作业**

请使用上述知识写一个交通工具与汽车、飞机的例子程序。