Go语言学习(十一)面向对象编程-类型系统

原创 2016年05月09日 21:07:14

标签：

[go语言](http://so.csdn.net/so/search/s.do?q=go%E8%AF%AD%E8%A8%80&t=blog" \t "http://blog.csdn.net/mchenys/article/details/_blank)

864

1.类型系统介绍

对于面向对象编程的支持Go 语言设计得非常简洁而优雅。简洁之处在于，Go语言并没有沿   
袭传统面向对象编程中的诸多概念，比如继承、虚函数、构造函数和析构函数、隐藏的 this 指   
针等。   
一个典型的类型系统通常包含如下基本内容：

基础类型，如 byte 、 int 、 bool 、 float 等；

复合类型，如数组、结构体、指针等；

可以指向任意对象的类型（ Any 类型）；

面向对象，即所有具备面向对象特征（比如成员方法）的类型；

接口。

因为Java语言自诞生以来被称为最纯正的面向对象语言，所以我们就先以Java语言为例讲一讲类型系统。   
在Java语言中，存在两套完全独立的类型系统：一套是值类型系统，主要是基本类型，如 byte 、   
int 、 boolean 、 char 、 double 等，这些类型基于值语义；一套是以 Object 类型为根的对象类型   
系统，这些类型可以定义成员变量和成员方法，可以有虚函数，基于引用语义，只允许在堆上创建   
（通过使用关键字 new ）。Java语言中的 Any 类型就是整个对象类型系统的根—— java.lang.Object   
类型，只有对象类型系统中的实例才可以被 Object 类型引用。值类型想要被 Object 类型引用，需要装箱   
（boxing）过程，比如 int 类型需要装箱成为 Integer 类型。另外，只有对象类型系统中的类型才可   
以实现接口，具体方法是让该类型从要实现的接口继承。

相比之下，Go语言中的大多数类型都是值语义，并且都可以包含对应的操作方法。在需要   
的时候，你可以给任何类型（包括内置类型）“增加”新方法。而在实现某个接口时，无需从   
该接口继承（事实上，Go语言根本就不支持面向对象思想中的继承语法），只需要实现该接口   
要求的所有方法即可。任何类型都可以被 Any 类型引用。 Any类型就是空接口，即interface{} 。

2.为类型添加方法

在Go语言中，你可以给任意类型（包括内置类型，但不包括指针类型）添加相应的方法，   
例如：

package main

import(

"fmt"

)//定义一个新的类型type Integer int

func main(){

//使用自定义的类型定义变量

var a Integer = 3

if a.Less(4){

fmt.Println(a,"less 4")

}

}//定义方法,调用者为Integer类型,接收Integer类型,返回bool值func (a Integer) Less(b Integer)bool{

return a < b

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

在这个例子中，我们定义了一个新类型 Integer ，它和 int 没有本质不同，只是它为内置的   
int 类型增加了个新方法 Less() 。这样实现了Integer后，就可以让整型像一个普通的类一样使用了.   
上面的方法的调用方式是 类型.方法名(参数列表),例如:   
a.Less(4)   
这种调用方式就是面向对象的体现.

他和面向过程有明显的区别,如下修改为面向过程的方法定义.

func Integer\_Less(a,b integer)bool{

return a < b

}

1

2

3

调用方式是 方法名(参数列表),例如：   
Integer\_Less(a,2)

3.通过指针修改对象

Go语言中的面向对象最为直观，也无需支付额外的成本。如果要求对象必须以指针传递，   
这有时会是个额外成本，因为对象有时很小（比如4字节），用指针传递并不划算。   
只有在你需要修改对象的时候，才必须用指针。它不是Go语言的约束，而是一种自然约束。   
举个例子：

package main

import(

"fmt"

)type Integer int

func main(){

var a Integer = 3

a.Add(3)

fmt.Println("a=",a)//a=6

}//通过Integer的指针调用 func (a \*Integer) Add(b Integer){

\*a += b //修改变量a的指针对应的值

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

运行该程序，得到的结果是： a=6 。   
如果你实现成员方法时的调用者不是a的指针而是值(即传入Integer,而非 \*Integer)   
如下所示：

func (a Integer) Add(b Integer) {

a += b

}

1

2

3

那么运行程序得到的结果是a=3   
也就是维持原来的值,这点需要特别注意.   
究其原因，是因为Go语言和C语言一样，类型都是基于值传递的。要想修改变量的值，只能   
传递指针。

4.值语义和引用语义；

值语义和引用语义的差别在于赋值   
如果 b 的修改不会影响 a 的值，那么此类型属于值类型。如果会影响 a 的值，   
那么此类型是引用类型.   
Go语言中的大多数类型都基于值语义，包括：

基本类型，如 byte 、 int 、 bool 、 float32 、 float64 和 string 等；

复合类型，如数组（array）、结构体（struct）和指针（pointer）等。

Go语言中的数组和基本类型没有什么区别,是很纯粹的值类型,例如:

var a = [3]int{1,2,3}var b = a

b[1]++

fmt.Println(a, b)

1

2

3

4

该程序的运行结果如下：   
[1 2 3] [1 3 3]。   
这表明 b=a 赋值语句是数组内容的完整复制。要想表达引用，需要用指针：

var a = [3]int{1, 2, 3}var b = &a

b[1]++

fmt.Println(a, \*b)

1

2

3

4

该程序的运行结果如下：   
[1 3 3] [1 3 3]   
这表明 b=&a 赋值语句是数组内容的引用。变量 b 的类型不是 [3]int ，而是 \*[3]int 类型。

Go语言中有4个类型比较特别，看起来像引用类型，如下所示。   
1.数组切片：指向数组（array）的一个区间。   
2.map：极其常见的数据结构，提供键值查询能力。   
3.channel：执行体（goroutine）间的通信设施。   
4.接口（interface）：对一组满足某个契约的类型的抽象。   
但是这并不影响我们将Go语言类型看做值语义。

数组切片本质上是一个区间，你可以大致将 []T 表示为：

type slice struct {

first \*T

len int

cap int

}

1

2

3

4

5

因为数组切片内部是指向数组的指针，所以可以改变所指向的数组元素并不奇怪。数组切片   
类型本身的赋值仍然是值语义。

5.结构体

Go语言的结构体（struct）和其他语言的类（class）有同等的地位，但Go语言放弃了包括继   
承在内的大量面向对象特性，只保留了组合（composition）这个最基础的特性。   
组合甚至不能算面向对象特性，因为在C语言这样的过程式编程语言中，也有结构体，也有   
组合。组合只是形成复合类型的基础。   
上面我们说到，所有的Go语言类型（指针类型除外）都可以有自己的方法。在这个背景下，   
Go语言的结构体只是很普通的复合类型，平淡无奇。例如，我们要定义一个矩形类型：

type Rect struct {

x, y float64

width, height float64

}//然后我们定义成员方法 Area() 来计算矩形的面积：func (r \*Rect) Area() float64 {

return r.width \* r.height

}

1

2

3

4

5

6

7

8

可以看出Go语言中结构体的使用方式与C语言并没有明显不同。

文章来源：http://blog.csdn.net/mchenys/article/details/51356212

版权声明：本文为博主原创文章，未经博主允许不得转载。