从组合设计模式看Go语言的多态特性

李智慧

Go语言没有关于对象的关键词，没有class，没有extends，没有implements，但是Go语言依然是一种完整的面向对象编程语言，拥有面向对象编程语言的三大特性：封装、继承、多态。
传统的面向对象编程语言通过override和overwrite实现多态，特别是对接口的override，使面向对象编程呈现出迷人的特性：针对接口编程，运行期注入实现，使程序呈现多态的特性。
面向对象设计模式中最让人困惑，最引人入胜的就是那些灵活应用对象多态特性的模式。
而Go语言的多态更加灵活，在传统的面向对象编程中，关于组合还是继承总有许多争论，不良继承又带来诸多问题。
Go语言中组合就是继承（extends），struct中包含另一个struct，就拥有了另一个struct的成员和方法。

```
//树的中间节点
type node struct{
    *list.List //（匿名）组合即继承，node拥有了list的特性
    name string
}
```

Go语言定义就是实现（implements），go语言可以定义接口（interface），也可以定义struct上的方法，但是方法不需要显式实现接口，只要方法签名一致就可以，如果一个struct上定义的方法实现了interface上定义的所有方法，那么就认为该struct实现了该接口。

```
//接口，关于树的遍历操作都在这里
type tree interface{
do()
}

func (l leaf)do(){//定义即实现，leaf实现了tree接口
    fmt.Println(l.name+" leaf do something.")
}
```

一个完整的关于树的遍历的go语言实现如下，利用go的多态特性，不需要递归。

```
package main

import(
    "fmt"
    "container/list"
)

//接口，关于树的遍历操作都在这里
type tree interface{
do()
}

//树的中间节点
type node struct{
    *list.List //（匿名）组合即继承，node拥有了list的特性
    name string
}

//树的叶子
type leaf struct{
    name string
}

func (n node)do(){//定义即实现,node实现了tree接口
    for e:=n.Front();e!=nil;e=e.Next(){//node拥有了list的特性
        e.Value.(tree).do()
    }
    fmt.Println(n.name+" node do something.")
}

func (n node)addSub(sub tree){
    n.PushBack(sub)
}

func (l leaf)do(){//定义即实现，leaf实现了tree接口
    fmt.Println(l.name+" leaf do something.")
}

func main() {

    //定义树的节点
    n1 := node{list.New(),"n1"}
    n2 := node{list.New(),"n2"}
    l1 := leaf{"l1"}
    l2 := leaf{"l2"}

    //构造树的结构
```

```
    n2.addSub(l2)
    n1.addSub(n2)
    n1.addSub(l1)

    //遍历树
    n1.do()

}
```

以上为树的深度优先遍历，如果想改为广度优先遍历，只需要调整一行代码。

```
func (n node)do(){//定义即实现，node实现了tree接口
    fmt.Println(n.name+" node do something.")
    for e:=n.Front();e!=nil;e=e.Next(){//node拥有了list的特性
        e.Value.(tree).do()
    }
}
```