函数,结构体和接口

函数基本组成包括：关键字func、函数名、参数列表、返回值、函数体和返回语句 Go 语言不支持继承，Go 语言只支持组合。 Go 语言的中结构体struct与 C++、JAVA 中的类class相似，但 Go 放弃了传统面向对象的诸多特性，只保留了组合

函数

例子：

```
package Add
func add(a,b int)(num int, err error){
    return a+b,nil
}
```

函数的返回值(可以有多个返回值)

在Go语言中，函数的返回值可以是多个的，一般包里面的函数都定义了第二个参数那就是error 我们在定义变量的时候可以在多定义一个err 例子：

```
func divide (a int, b int) (num int, err error){ //定义两个返回值   （先是func，第二是函数名字。参数列表。返回值）
    if b == 0 {
        err := errors.New("被除数不能为零！ ")
        return -1,err
    }
    return a / b, nil    //支持多个返回值
}
```

匿名函数

例子：

```
func(){
 PROCESS
}()
```

这就是个匿名函数的例子，在这个函数里面没有函数名，参数和返回值

结构体

形式就是

```
type person struct{
 name string
 age uint
}
```

初始化一个对象

man := new(person) man := &person{} man := &person{"Tom", 18} man := &person{name: "Tom", age: 18}

结构体的方法

方法是作用在自定义类型上的一类特殊函数，该值可能是以指针或者复制值的形式传递 例如

```
func (per *person) SetAge(ages uint) bool{
     *per.age=ages
     return true
}
func (per person) SetAge(ages uint) bool{
     per.age=ages
     return true
}
```

值和指针的区别 1：一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。 2：相对于值的话，你在函数里面赋值在出函数之后就会结束。而相对于指针的话，它给与的是那个变量的地址，对地址里面的数值进行了改变。

结构体的组合

例子

```
package main
import (
"fmt"
)
type person struct {
name  string
world string
}
func (p person) setname(realname string) {
p.name = realname
fmt.Println("the person name is", p.name)
}
type chinese struct {
person
pro string
}
```

```
func (c *chinese) setname(realname string) {
c.person.name = realname
}
func (c *chinese) setpro(realpro string) {
c.pro = realpro
}
func (c chinese) display() {
fmt.Println("name  and pro", c.person.name, c.pro)
}
func main() {
x := chinese{}
x.setname("jwuenjie")
x.setpro("shanxi")
x.display()
}
```

接口(interface)

接口的作用: 接口是一组方法签名。当一个类型为接口中的所有方法提供定义时，它被称为实现该接口。 接口就是可以接受多种类型的一个窗口。

例子:

```
type Shape interface {
area() float64
}
type circue struct {
rad float64
}
type squ struct {
x, y float64
}
func (cir circue) area() float64 {
return 3.14 * cir.rad * cir.rad
}
func (s squ) area() float64 {
return s.x * s.y
}
func fun(shape Shape) float64 {
return shape.area()
}
func main() {
/* cir := circue{5.5}
squ := squ{1.2, 2.4}
fmt.Println(fun(cir))
fmt.Println(fun(squ)) */
cir := circue{5.5}
squ := squ{1.2, 2.4}
var s Shape
s = cir
fmt.Println(s.area())
```

```
s = squ
fmt.Println(s.area())
}
```

嵌套的interface

```
type Interface1 interface {
    Send()
    Receive()
}

type Interface2 interface {
    Interface1
    Close()
}
```