05 函数和方法：Go 语言中的函数和方法到底有什么不同？

上一讲的思考题是创建一个二维数组并使用。上节课，我主要介绍了一维数组，其实二维数组也很简单，仿照一维数组即可，如下面的代码所示：

```
aa:=[3][3]int{}
aa[0][0] =1
aa[0][1] =2
aa[0][2] =3
aa[1][0] =4
aa[1][1] =5
aa[1][2] =6
aa[2][0] =7
aa[2][1] =8
aa[2][2] =9
fmt.Println(aa)
```

相信你也完成了，现在学习我们本节课要讲的函数和方法。

函数和方法是我们迈向代码复用、多人协作开发的第一步。通过函数，可以把开发任务分解成一个个小的单元，这些小单元可以被其他单元复用，进而提高开发效率、降低代码重合度。再加上现成的函数已经被充分测试和使用过，所以其他函数在使用这个函数时也更安全，比你自己重新写一个相似功能的函数 Bug 率更低。

这节课，我会详细讲解 Go 语言的函数和方法，了解它们的声明、使用和不同。虽然在 Go 语言中有函数和方法两种概念，但它们的相似度非常高，只是所属的对象不同。我们先从函数开始了解。

函数

函数初探

在前面的四节课中，你已经见到了 Go 语言中一个非常重要的函数：main 函数，它是一个 Go 语言程序的入口函数，我在演示代码示例的时候，会一遍遍地使用它。

下面的示例就是一个 main 函数：

```
func main() {

}
```

它由以下几部分构成：

1. 任何一个函数的定义，都有一个 func 关键字，用于声明一个函数，就像使用 var 关键字声明一个变量一样；
2. 然后紧跟的 main 是函数的名字，命名符合 Go 语言的规范即可，比如不能以数字开头；
3. main 函数名字后面的一对括号 () 是不能省略的，括号里可以定义函数使用的参数，这里的 main 函数没有参数，所以是空括号 ()；
4. 括号 () 后还可以有函数的返回值，因为 main 函数没有返回值，所以这里没有定义；
5. 最后就是大括号 {} 函数体了，你可以在函数体里书写代码，写该函数自己的业务逻辑。

函数声明

经过上一小节的介绍，相信你已经对 Go 语言函数的构成有一个比较清晰的了解了，现在让我们一起总结出函数的声明格式，如下面的代码所示：

```
func funcName(params) result {

    body

}
```

这就是一个函数的签名定义，它包含以下几个部分：

1. 关键字 func；
2. 函数名字 funcName；
3. 函数的参数 params，用来定义形参的变量名和类型，可以有一个参数，也可以有多个，也可以没有；
4. result 是返回的函数值，用于定义返回值的类型，如果没有返回值，省略即可，也可以有多个返回值；
5. body 就是函数体，可以在这里写函数的代码逻辑。

现在，我们一起根据上面的函数声明格式，自定义一个函数，如下所示：

ch05/main.go

```
func sum(a int,b int) int{
    return a+b
}
```

这是一个计算两数之和的函数，函数的名字是 sum，它有两个参数 a、b，参数的类型都是 int。sum 函数的返回值也是 int 类型，函数体部分就是把 a 和 b 相加，然后通过 return 关键字返回，如果函数没有返回值，可以不用使用 return 关键字。

终于可以声明自己的函数了，恭喜你迈出了一大步！

函数中形参的定义和我们定义变量是一样的，都是变量名称在前，变量类型在后，只不过在函数里，变量名称叫作参数名称，也就是函数的形参，形参只能在该函数体内使用。函数形参的值由调用者提供，这个值也称为函数的实参，现在我们传递实参给 sum 函数，演示函数的调用，如下面的代码所示：

ch05/main.go

```
func main() {
    result:=sum(1,2)
    fmt.Println(result)
}
```

我们自定义的 sum 函数，在 main 函数中直接调用，调用的时候需要提供真实的参数，也就是实参 1 和 2。

函数的返回值被赋值给变量 result，然后把这个结果打印出来。你可以自己运行一下，能看到结果是 3，这样我们就通过函数 sum 达到了两数相加的目的，如果其他业务逻辑也需要两数相加，那么就可以直接使用这个 sum 函数，不用再定义了。

在以上函数定义中，a 和 b 形参的类型是一样的，这个时候我们可以省略其中一个类型的声明，如下所示：

```
func sum(a, b int) int {
    return a + b
}
```

像这样使用逗号分隔变量，后面统一使用 int 类型，这和变量的声明是一样的，多个相同类型的变量都可以这么声明。

多值返回

同有的编程语言不一样，Go 语言的函数可以返回多个值，也就是多值返回。在 Go 语言的标准库中，你可以看到很多这样的函数：第一个值返回函数的结果，第二个值返回函数出错的信息，这种就是多值返回的经典应用。

对于 sum 函数，假设我们不允许提供的实参是负数，可以这样改造：在实参是负数的时候，通过多值返回，返回函数的错误信息，如下面的代码所示：

ch05/main.go

```
func sum(a, b int) (int,error){
    if a<0 || b<0 {
        return 0,errors.New("a或者b不能是负数")
    }
    return a + b,nil
}
```

这里需要注意的是，如果函数有多个返回值，返回值部分的类型定义需要使用小括号括起来，也就是 (int,error)。这代表函数 sum 有两个返回值，第一个是 int 类型，第二个是 error 类型，我们在函数体中使用 return 返回结果的时候，也要符合这个类型顺序。

在函数体中，可以使用 return 返回多个值，返回的多个值通过逗号分隔即可，返回多个值的类型顺序要和函数声明的返回类型顺序一致，比如下面的例子：

```
return 0,errors.New("a或者b不能是负数")
```

返回的第一个值 0 是 int 类型，第二个值是 error 类型，和函数定义的返回类型完全一致。

定义好了多值返回的函数，现在我们用如下代码尝试调用：

ch05/main.go

```
func main() {
    result,err := sum(1, 2)
    if err!=nil {
        fmt.Println(err)
    }else {
        fmt.Println(result)
    }
```

```
}
```

函数有多值返回的时候，需要有多个变量接收它的值，示例中使用 result 和 err 变量，使用逗号分开。

如果有的函数的返回值不需要，可以使用下划线 _ 丢弃它，这种方式我在 for range 循环那节课里也使用过，如下所示：

```
result,_ := sum(1, 2)
```

这样即可忽略函数 sum 返回的错误信息，也不用再做判断。

> 提示：这里使用的 error 是 Go 语言内置的一个接口，用于表示程序的错误信息，后续课程我会详细介绍。

命名返回参数

不止函数的参数可以有变量名称，函数的返回值也可以，也就是说你可以为每个返回值都起一个名字，这个名字可以像参数一样在函数体内使用。

现在我们继续对 sum 函数的例子进行改造，为其返回值命名，如下面的代码所示：

ch05/main.go

```
func sum(a, b int) (sum int,err error){
    if a<0 || b<0 {
        return 0,errors.New("a或者b不能是负数")
    }
    sum=a+b
    err=nil
    return
}
```

返回值的命名和参数、变量都是一样的，名称在前，类型在后。以上示例中，命名的两个返回值名称，一个是 sum，一个是 err，这样就可以在函数体中使用它们了。

通过下面示例中的这种方式直接为命名返回参数赋值，也就等于函数有了返回值，所以就可以忽略 return 的返回值了，也就是说，示例中只有一个 return，return 后没有要返回的值。

```
sum=a+b

err=nil
```

通过命名返回参数的赋值方式，和直接使用 return 返回值的方式结果是一样的，所以调用以上 sum 函数，返回的结果也一样。

虽然 Go 语言支持函数返回值命名，但是并不是太常用，根据自己的需求情况，酌情选择是否对函数返回值命名。

可变参数

可变参数，就是函数的参数数量是可变的，比如最常见的 fmt.Println 函数。

同样一个函数，可以不传参数，也可以传递一个参数，也可以两个参数，也可以是多个等等，这种函数就是具有可变参数的函数，如下所示：

```
fmt.Println()
fmt.Println("飞雪")
fmt.Println("飞雪","无情")
```

下面所演示的是 Println 函数的声明，从中可以看到，定义可变参数，只要在参数类型前加三个点 … 即可：

```
func Println(a ...interface{}) (n int, err error)
```

现在我们也可以定义自己的可变参数的函数了。还是以 sum 函数为例，在下面的代码中，我通过可变参数的方式，计算调用者传递的所有实参的和：

ch05/main.go

```
func sum1(params ...int) int {
    sum := 0
    for _, i := range params {
        sum += i
    }
    return sum
}
```

为了便于和 sum 函数区分，我定义了函数 sum1，该函数的参数是一个可变参数，然后通过 for range 循环来计算这些参数之和。

讲到这里，相信你也看明白了，可变参数的类型其实就是切片，比如示例中 params 参数的类型是 []int，所以可以使用 for range 进行循环。

函数有了可变参数，就可以灵活地进行使用了。

如下面的调用者示例，传递几个参数都可以，非常方便，也更灵活：

ch05/main.go

```
fmt.Println(sum1(1,2))
fmt.Println(sum1(1,2,3))
fmt.Println(sum1(1,2,3,4))
```

> 这里需要注意，如果你定义的函数中既有普通参数，又有可变参数，那么可变参数一定要放在参数列表的最后一个，比如 sum1(tip string,params …int) ，params 可变参数一定要放在最末尾。

包级函数

不管是自定义的函数 sum、sum1，还是我们使用到的函数 Println，都会从属于一个包，也就是 package。sum 函数属于 main 包，Println 函数属于 fmt 包。

同一个包中的函数哪怕是私有的（函数名称首字母小写）也可以被调用。如果不同包的函数要被调用，那么函数的作用域必须是公有的，也就是**函数名称的首字母要大写**，比如 Println。

在后面的包、作用域和模块化的课程中我会详细讲解，这里可以先记住：

1. 函数名称首字母小写代表私有函数，只有在同一个包中才可以被调用；
2. 函数名称首字母大写代表公有函数，不同的包也可以调用；
3. 任何一个函数都会从属于一个包。

> 小提示：Go 语言没有用 public、private 这样的修饰符来修饰函数是公有还是私有，而是通过函数名称的大小写来代表，这样省略了烦琐的修饰符，更简洁。

匿名函数和闭包

顾名思义，匿名函数就是没有名字的函数，这是它和正常函数的主要区别。

在下面的示例中，变量 sum2 所对应的值就是一个匿名函数。需要注意的是，这里的 sum2 只是一个函数类型的变量，并不是函数的名字。

ch05/main.go

```
func main() {
    sum2 := func(a, b int) int {
        return a + b
    }
    fmt.Println(sum2(1, 2))
}
```

通过 sum2，我们可以对匿名函数进行调用，以上示例算出的结果是 3，和使用正常的函数一样。

有了匿名函数，就可以在函数中再定义函数（函数嵌套），定义的这个匿名函数，也可以称为内部函数。更重要的是，在函数内定义的内部函数，可以使用外部函数的变量等，这种方式也称为闭包。

我们用下面的代码进行演示：

ch05/main.go

```
func main() {
    cl:=colsure()
    fmt.Println(cl())
    fmt.Println(cl())
    fmt.Println(cl())
}

func colsure() func() int {
    i:=0
    return func() int {
        i++
        return i
    }
}
```

运行这个代码，你会看到输出打印的结果是：

```
1
2
3
```

这都得益于匿名函数闭包的能力，让我们自定义的 colsure 函数，可以返回一个匿名函数，并且持有外部函数 colsure 的变量 i。因而在 main 函数中，每调用一次 cl()，i 的值就会加 1。

> 小提示：在 Go 语言中，函数也是一种类型，它也可以被用来声明函数类型的变量、参数或者作为另一个函数的返回值类型。

方法

不同于函数的方法

在 Go 语言中，方法和函数是两个概念，但又非常相似，不同点在于方法必须要有一个接收者，这个接收者是一个类型，这样方法就和这个类型绑定在一起，称为这个类型的方法。

在下面的示例中，type Age uint 表示定义一个新类型 Age，该类型等价于 uint，可以理解为类型 uint 的重命名。其中 type 是 Go 语言关键字，表示定义一个类型，在结构体和接口的课程中我会详细介绍。

ch05/main.go

```
type Age uint
func (age Age) String(){
    fmt.Println("the age is",age)
}
```

示例中方法 String() 就是类型 Age 的方法，类型 Age 是方法 String() 的接收者。

和函数不同，定义方法时会在关键字 func 和方法名 String 之间加一个接收者 (age Age)，接收者使用小括号包围。

接收者的定义和普通变量、函数参数等一样，前面是变量名，后面是接收者类型。

现在方法 String() 就和类型 Age 绑定在一起了，String() 是类型 Age 的方法。

定义了接收者的方法后，就可以通过点操作符调用方法，如下面的代码所示：

ch05/main.go

```
func main() {
    age:=Age(25)
    age.String()
}
```

运行这段代码，可以看到如下输出：

```
the age is 25
```

接收者就是函数和方法的最大不同，此外，上面所讲到的函数具备的能力，方法也都具备。

> 提示：因为 25 也是 unit 类型，unit 类型等价于我定义的 Age 类型，所以 25 可以强制转换为 Age 类型。

值类型接收者和指针类型接收者

方法的接收者除了可以是值类型（比如上一小节的示例），也可以是指针类型。

定义的方法的接收者类型是指针，所以我们对指针的修改是有效的，如果不是指针，修改就没有效果，如下所示：

ch05/main.go

```
func (age *Age) Modify(){
    *age = Age(30)
}
```

调用一次 Modify 方法后，再调用 String 方法查看结果，会发现已经变成了 30，说明基于指针的修改有效，如下所示：

```
age:=Age(25)
age.String()
age.Modify()
age.String()
```

> 提示：在调用方法的时候，传递的接收者本质上都是副本，只不过一个是这个值副本，一是指向这个值指针的副本。指针具有指向原有值的特性，所以修改了指针指向的值，也就修改了原有的值。我们可以简单地理解为值接收者使用的是值的副本来调用方法，而指针接收者使用实际的值来调用方法。

示例中调用指针接收者方法的时候，使用的是一个值类型的变量，并不是一个指针类型，其实这里使用指针变量调用也是可以的，如下面的代码所示：

```
(&age).Modify()
```

这就是 Go 语言编译器帮我们自动做的事情：

- 如果使用一个值类型变量调用指针类型接收者的方法，Go 语言编译器会自动帮我们取指针调用，以满足指针接收者的要求。
- 同样的原理，如果使用一个指针类型变量调用值类型接收者的方法，Go 语言编译器会自动帮我们解引用调用，以满足值类型接收者的要求。

总之，方法的调用者，既可以是值也可以是指针，不用太关注这些，Go 语言会帮我们自动转义，大大提高开发效率，同时避免因不小心造成的 Bug。

> 不管是使用值类型接收者，还是指针类型接收者，要先确定你的需求：在对类型进行操作的时候是要改变当前接收者的值，还是要创建一个新值进行返回？这些就可以决定使用哪种接收者。

总结

在 Go 语言中，虽然存在函数和方法两个概念，但是它们基本相同，不同的是所属的对象。函数属于一个包，方法属于一个类型，所以方法也可以简单地理解为和一个类型关联的函数。

不管是函数还是方法，它们都是代码复用的第一步，也是代码职责分离的基础。掌握好函数和方法，可以让你写出职责清晰、任务明确、可复用的代码，提高开发效率、降低 Bug 率。

本节课给你**留的思考题是**：方法是否可以作为表达式赋值给一个变量？如果可以的话，如何通过这个变量调用方法？

上一页 下一页