Go 语言方法

Go中的方法，是一种**特殊的函数**，定义于struct之上(与struct关联、绑定)，被称为struct的接受者(receiver)。

通俗的讲，方法就是有接收者的函数。

方法定义

语法格式如下：

```
type mytype struct{}

func (recv mytype) my_method(para) return_type {}
或
func (recv *mytype) my_method(para) return_type {}
```

```
func (recevier type) methodName(参数列表)(返回值列表){}
```

`mytype`：定义一个结构体

`recv`：接收该方法的结构体（receiver）接收器

`my_method`：方法名称

`para`：参数列表

`return_type`：返回值类型

从语法格式看出，函数和方法非常类似，只不过方法多了一个接收者。

一个方法就是一个包含了接受者的函数，接受者可以是命名类型或者结构体类型的一个值或者是一个指针。

所有给定类型的方法属于该类型的方法集。

注意事项：

- 方法的接收者（receiver）type类型不一定是struct结构体类型，type定义的别名、结构体、slice切片、map集合、channel、func函数类型等都可以。
- struct结合它的方法就等价于面向对象中的类，只不过struct结构体可以和他的方法分开，并非同属于与一个文件，但一定要必须属于一个包。
- 方法有两种接收类型`T type`或`T *type`。他们之间有区别。
- 方法就是函数，不支持方法重载（overload），也就是说同一类型中的所有方法必须都是唯一的。
- 如果接收者是一个指针类型，则会自动解除引用。如果一个类型名本身是一个指针的话，是不允许其出现在接收器中的。
- 方法和type是分开的，意味着实例的行为和数据存储是分开的，但是他们通过接收器建立起关联关系。
- 参数和返回值可以省略

示例：

```
package main
```

```
type Test struct{}

// 无参数、无返回值
func (t Test) method0() {

}

// 单参数、无返回值
func (t Test) method1(i int) {

}

// 多参数、无返回值
func (t Test) method2(x, y int) {

}

// 无参数、单返回值
func (t Test) method3() (i int) {
    return
}

// 多参数、多返回值
func (t Test) method4(x, y int) (z int, err error) {
    return
}

// 无参数、无返回值
func (t *Test) method5() {

}

// 单参数、无返回值
func (t *Test) method6(i int) {

}

// 多参数、无返回值
func (t *Test) method7(x, y int) {

}

// 无参数、单返回值
func (t *Test) method8() (i int) {
    return
}

// 多参数、多返回值
func (t *Test) method9(x, y int) (z int, err error) {
    return
}

func main() {}
```

方法接收者类型

结构体实例，有值类型和指针类型，那么方法的接收者是结构体，那么也有值类型和指针类型。区别就是接收者是否复制结构体副本。值类型复制，指针类型不复制。

示例：

```
package main

import (
    "fmt"
)

// 结构体
type User struct {
    Name  string
    Email string
}

// 方法
func (u User) Show() {
    fmt.Printf("%v : %v \n", u.Name, u.Email)
}

// 方法
func (u *User) Show1() {
    fmt.Printf("%v : %v \n", u.Name, u.Email)
}

func main() {
    // 值类型调用方法
    u1 := User{"包子", "baozi@163.com"}
    u1.Show()

    // 指针类型调用方法
    u2 := User{"肉包子", "roubaozi@163.com"}
    u3 := &u2
    u3.Show1()
}
#结果
包子 : baozi@163.com
肉包子 : roubaozi@163.com
```

> 注意：当接收者是指针时，即使用值类型调用那么函数内部也是对指针的操作。

普通函数与方法的区别

1.对于普通函数，接收者为值类型时，不能将指针类型的数据直接传递，反之亦然。

2.对于方法（如struct的方法），接收者为值类型时，可以直接用指针类型的变量调用方法，反过来同样也可以。

示例：

```
package main
```

```
//普通函数与方法的区别（在接收者分别为值类型和指针类型的时候）

import (
    "fmt"
)

// 1.普通函数
// 接收值类型参数的函数
func valueIntTest(a int) int {
    return a + 10
}

// 接收指针类型参数的函数
func pointerIntTest(a *int) int {
    return *a + 10
}

func structTestValue() {
    a := 2
    fmt.Println("valueIntTest:", valueIntTest(a))
    //函数的参数为值类型，则不能直接将指针作为参数传递
    //fmt.Println("valueIntTest:", valueIntTest(&a))
    //compile error: cannot use &a (type *int) as type int in function argument

    b := 5
    fmt.Println("pointerIntTest:", pointerIntTest(&b))
    //同样，当函数的参数为指针类型时，也不能直接将值类型作为参数传递
    //fmt.Println("pointerIntTest:", pointerIntTest(b))
    //compile error:cannot use b (type int) as type *int in function argument
}

// 2.方法
type PersonD struct {
    id   int
    name string
}

// 接收者为值类型
func (p PersonD) valueShowName() {
    fmt.Println(p.name)
}

// 接收者为指针类型
func (p *PersonD) pointShowName() {
    fmt.Println(p.name)
}

func structTestFunc() {
    //值类型调用方法
    personValue := PersonD{101, "hello world"}
    personValue.valueShowName()
    personValue.pointShowName()

    //指针类型调用方法
```

```
    personPointer := &PersonD{102, "hello golang"}
    personPointer.valueShowName()
    personPointer.pointShowName()

    //与普通函数不同，接收者为指针类型和值类型的方法，指针类型和值类型的变量均可相互调用
}

func main() {
    structTestValue()
    structTestFunc()
}
#结果
valueIntTest: 12
pointerIntTest: 15
hello world
hello world
hello golang
hello golang
```