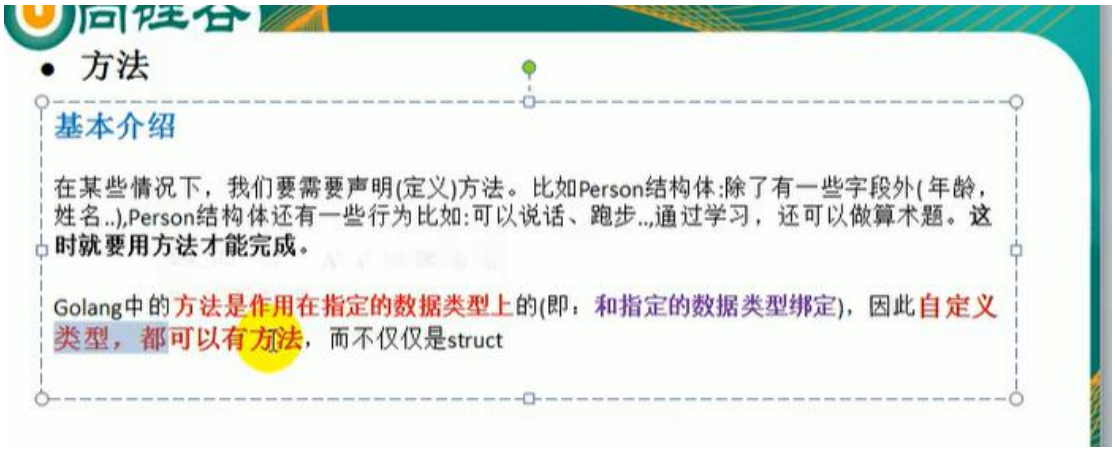



```go
type Person struct{
    Name string
}

//给Person类型绑定一方法
func (p Person) test() {
    fmt.Println("test() name=", p.Name)
}

func main() {

    var p Person
    p.Name = "tom"
    p.test() //调用方法
    test()
}
```

对上面的总结

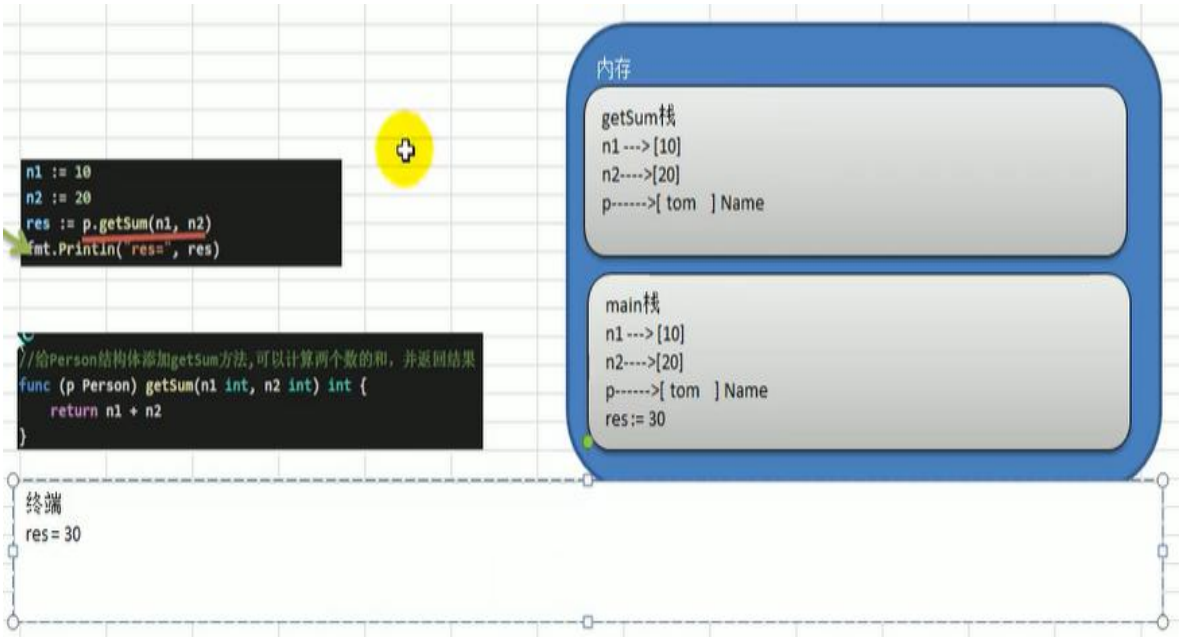
1）text 方法和 Person 类绑定

2）text 方法只能通过 Person 类的变量来调用，不能直接调用，也不能用其他类型的变量来调用

3）func （ p Person）{}...p 表示哪个 Person 变量这个 p 就是他的副本，这点和参数传参非常相似，值传递改变值不会引起本身变化。

4）p 不是固定的，可以随意起名。

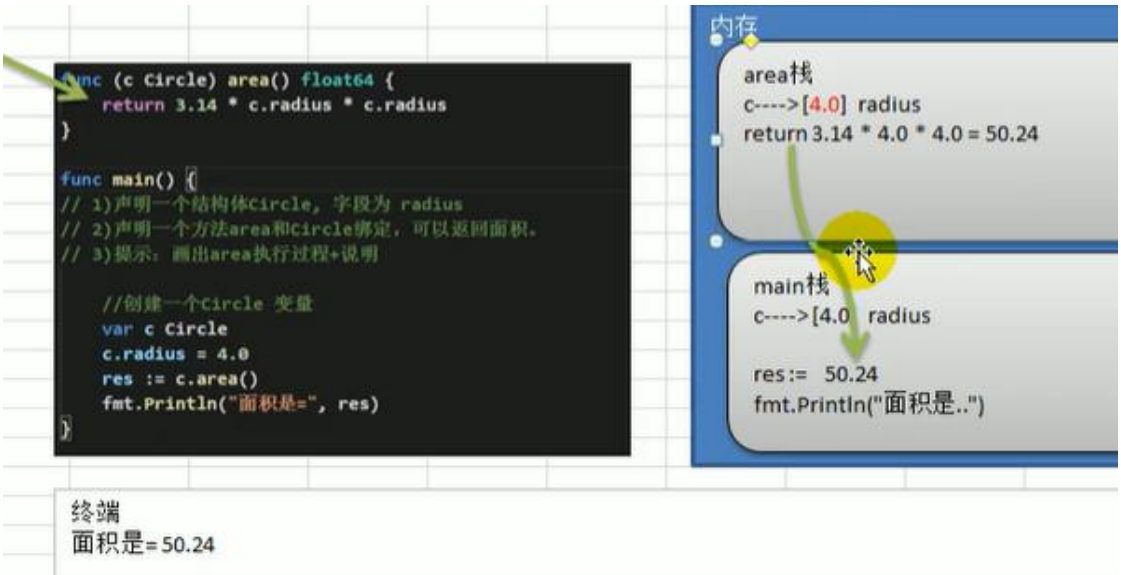
5）方法调用时，该方法所属的类型的实例也会被当做参数




说明:

1) 在通过一个变量去调用方法时，其调用机制和函数一样

2) 不一样的地方时，变量调用方法时，该变量本身也会作为一个参数传递到方法(如果变量是值类型，则进行值拷贝，如果变量是引用类型，则进行地质拷贝)




方法声明

## 方法

### 方法的声明(定义)

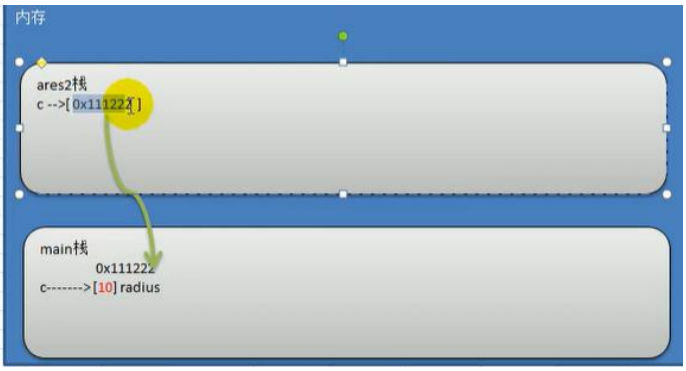
```
func (recevier type) methodName（参数列表） (返回值列表){
        方法体
        return 返回值
}
```

1) 参数列表：表示方法输入
2) recevier type : 表示这个方法和type这个类型进行绑定，或者说该方法作用于type类型
3) receiver type : type可以是结构体，也可以其它的自定义类型
4) receiver : 就是type类型的一个变量(实例)，比如 ： Person结构体 的一个变量(实例)
5) 参数列表：表示方法输入
6) 返回值列表：表示返回的值，可以多个
7) 方法主体：表示为了实现某一功能代码块
8) return 语句不是必须的。

方法注意事项和细节

```
//创建一个Circle 变量
var c Circle
c.radius = 7.0
//res2 := (&c).area2()
//编译器底层做了优化 (&c).area2() 等价 c.area()
//因为编译器会自动的给加上 &c
res2 := c.area2()
fmt.Println("面积=", res2)
fmt.Println("c.radius = ", c.radius) //10
```

```
//为了提高效率，通常我们方法和结构体的指针类型绑定
func (c *Circle) area2() float64 {
    //因为 c是指针，因此我们标准的访问其字段的方式是 (*c
    //return 3.14 * (*c).radius * (*c).radius
    // (*c).radius 等价 c.radius
    c.radius = 10
    return 3.14 * c.radius * c.radius
}
```




### 方法注意事项和细节讨论

1) 结构体类型是值类型，在方法调用中，遵守值类型的传递机制，是值拷贝传递方式
2) 如程序员希望在方法中，修改结构体变量的值，可以通过结构体指针的方式来处理
3) Golang中的方法作用在指定的数据类型上的(即：和指定的数据类型绑定)，因此自定义类型，都可以有方法，而不仅仅是struct， 比如int , float32等都可以有方法
4) 方法的访问范围控制的规则，和函数一样。方法名首字母小写，只能在本包访问，方法首字母大写，可以在本包和其它包访问。[讲解]
5) 如果一个类型实现了String()这个方法，那么fmt.Println默认会调用这个变量的String()进行输出

方法中是值拷贝还是传入指针地址，调用时的写法上可以随便写（&p）|| p 都可以，是值拷贝还是指针传入地址，关键

在于方法上标明的是指针传入还是值传入

func（p Person）xxx(){}

func（p *Person）xxx(){}