方法

封装

面向对象编程的一个较为重要的特性就是封装: 将对象的属性和对象行为封装在同一个类中，并对外暴露出必要的接口

提供访问权限控制

封装的目的除了用代码来模拟一个事物以外，还提供了隐藏实现细节的特性

调用者无需关心具体的实现，只需关心对外提供的接口

在Go语言中，并没有类(class)的概念，但是Go允许将任意的方法绑定到任意的类型上，除了接口类型和指针类型

Go方法

```go
type Student struct {
    Name    string
}

func (s Student)GetName() string {
    // s在Go中称之为接收者，其类型为Student
    return s.Name
}

func main() {
    s := Student{"Zero"}
    s.GetName()
}
```

与Java、Python不同的是，Go语言的接受者可以是任意自定义名称，不需要使用特殊名称，例如this或者是self

指针接收者

方法和函数并没有本质的区别，在调用方法时仍然会拷贝实参，所以当实参占用较大内存空间时，通常会使用指针进行传递。并且若想要修改实参内容，也必须使用指针

```go
func (s *Student)GetName() string {
    // 此时方法为指针接收者，接收的类型为*Student
    return s.Name
}
```

那么此时GetName该方法的完整名称为 (*Student).GetName

一般情况下来说，如果其中一个方法为指针接收者，那么我们倾向于将所有的方法均设置为指针接收者

当方法为指针接收者时，我们不必一定使用指针类型来进行调用

```go
s := Student{"Zero"}

sPtr := &s
sPtr.GetName()  // 指针调用

(&s).GetName()  // 这也是可以的

s.GetName()     // 使用变量直接调用
```

能够直接使用Student类型的变量调用指针接收者的方法原因在于，Go会对变量进行隐式的取地址行为

即编译器会对变量s进行隐式的&s类型转换，只有变量才能如此使用，字面量表达式则不允许

nil是个合法的接收者

在Python或者Java中，None或者是null在调用方法时都会抛出异常。但是在Go中，允许nil调用相关方法

```go
func (s *Student)GetName() {
    fmt.Println("Zero")
}

var s *Student
s.GetName()
```

指针的初始值为nil，当调用GetName()方法时，并不会出现异常

但是，正是因为nil是一个合法的接收者，所以在函数内部判断接收者的值是否为nil仍然有必要