方法

面向对象编程OOP Object-Oriented Programming

C 结构体Struct
C++ 类Class
数据成员(values) + 成员函数(functions)

```
class Point {
public:
    Point(float a = 0, float b = 0) : x(a), y(b) {}

    float Distance(Point another) {
        return sqrt((x-another.x)*(x-another.x) + (y-another.y)*(y-another.y));
    }

private:
    float x, y;
};

...
Point p, q;
p.Distance(1); // p.xxx(), p执行它的成员函数
```

Go?
结构体 struct in ch4

```
type Point struct {
    X, Y float64
}
```

只有数据成员很难办事哎

加几条胳膊腿（成员函数）

```go
type Point struct{ X, Y float64 }

// 方法method，类似于成员函数的东西，注意func后面有个(p Point)，则可以p.Distance(所有Point类无论p,
// ()里的p称为 接收器
func (p Point) Distance(q Point) float64 {
    return math.Hypot(q.X-p.X, q.Y-p.Y)
}

// 一般函数，注意和method的区别
func Distance(p, q Point) float64 {
    return math.Hypot(q.X-p.X, q.Y-p.Y)
}

...
p := Point{X: 1, Y: 2}
q := Point{X: 2, Y: 3}
p.Distance(q) // 方法method
Distance(p, q) // 一般函数func
```

(还有，为啥X, Y都大写，不麻烦吗？)
(（哎🤓👆，那你逝逝小写会怎么样呢）)

例子

自定义package geometry

在外面的main里测试methodTest.go

```go
func twoDis() {
        p := geometry.Point{X: 1, Y: 2}
        q := geometry.Point{X: 3, Y: 4}
        fmt.Println(geometry.Distance(p, q)) // func
        fmt.Println(p.Distance(q)) // method
}
// 两个都OK
// 但是，我的method比你的func少打了整整9个字母！
// 大win特win🥳😊
```

基于指针对象的方法

传值？没意思，传个指针玩

为Point类型新加一个ScaleBy函数

```
func (p *Point) ScaleBy(factor float64) {
    p.X *= factor
    p.Y *= factor
}
```

那就成了 `(*Point).ScaleBy`

注意，只有类型和指向它们的指针可以写在括号里（当作接收器），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
e.g.

```
type iptr *int
func (ip iptr) f() {} // invalid receiver type iptr (pointer or interface type)
```

君の指针

```
// 1. := &Point{}
r := &Point{1, 2}
r.ScaleBy(22)
// fmt.Println(*r) // "{2, 4}"
// &取地址 *解引用

// 2. p := Point{}  pptr := &p
p := Point{1, 2}
pptr := &p
pptr.ScaleBy(2)

// 3. p := Point{}  (&p).xxx()
p := Point{1, 2}
(&p).ScaleBy(2)
```

容错

Go学长知道你&/*太累还经常忘，贴心地追着你报错😊

哎，手抖了可以理解，就不报你的错了

```
p := Point{1, 2}
// (&p).ScaleBy(2) 本来应该是*Point才能.ScaleBy的
p.ScaleBy(2) // 哎，手抖少打个&，通融一下
// OK，你过关！
```

```
p := Point{1, 2}
q := Point{2, 3}
pptr := &p
// (*pptr).Distance(q) 本来应该是Point类型才能.Distance的
pptr.Distance(q) // 哎，哥们少打个*，你说扯不扯😊
// 哎，你都说哥们了还干嘛呢，给你悄摸插个*就是咯😊😊
```

快说谢谢Go学长！😘

嵌入结构体 / 组合++

ch4/结构体说过结构体嵌入，详见 结构体/结构体嵌入 & 匿名嵌入

类似于组合的一种方法，对于数据成员是OK的
对于成员函数（方法）呢？🤔
也是OK的！🥳

```
type Circle struct {
    P Point // Point: {X, Y}
    Radius float64
}

c := Circle{
    P: Point{1, 2},
    Radius: 3,
}

c.P.X // 1
c.P.ScaleBy(2) // (1, 2) -> (2, 4)
```

偷懒大法之匿名嵌入

不起名字 就可以 不用写名字！ 😊

```
type Circle struct {
    Point // Point: {X, Y} 你不需要名字😘
    Radius float64
}

c := Circle{
    P: Point{1, 2},
    Radius: 3,
}

c.X // = c.P.X
c.ScaleBy(2) // = c.P.ScaleBy(2)
```

偷懒也是生产力！

覆盖/重写 override

同一个struct的方法名字不同，Point不能有两个Distance()

不同struct可以有同名方法，因为两个struct不同

存在嵌入时，如果外层结构体定义了同名方法，会覆盖嵌入结构体的方法

```go
func (c Circle) Distance(p Point) float64 {
    return math.Hypot(c.X-p.X, c.Y-p.Y) + c.Radius
}

// 现在 c.Distance(p) 会调用 Circle 的 Distance，而不是 Point 的
```

方法值和方法表达式

方法值 Method Value

方法（函数）绑定到特定接收者（示例） = 函数变量，可以调用

```go
p := Point{0, 0}
    q := Point{3, 4}

    // 方法值：Distance绑定到Point p
// 得到distanceFromP，函数变量，可以当一个函数调用
    distanceFromP := p.Distance

    // 调用方法值
    fmt.Println(distanceFromP(q))  // 5
```

方法表达式 Method Expression

将 方法 转换为 普通函数

哟，来了牢弟~ 🤭

干method这行干不下去了，回来当normal function了😇

```
p := Point{0, 0}
    q := Point{3, 4}

    // 方法表达式：Point.Distance 是一个函数，接收 Point 作为第一个参数
// func (p Point) Distance(q Point) float64 -> func Distance(p Point, q Point) float64
    distanceFunc := Point.Distance

    fmt.Println(distanceFunc(p, q))  // 5
```

封装

还得是你呀C学长，OOP领域最高的山，最长的河😘

大小写 = Public / private
Go语言只有一种控制可见性的手段：大写首字母的标识符会从定义它们的包中被导出，小写字母的则不会。

Why 封装?

1. 只需要搞懂接口就OK，不用管乱七八糟的东西🥳
2. 隐藏实现细节🥳🥳
3. 省得你乱搞乱改（得通过 get & set）🥳🥳🥳