属性

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

属性

- Swift中跟实例相关的属性可以分为2大类

存储属性（Stored Property）

- ✓ 类似于成员变量这个概念
- ✓ 存储在实例的内存中
- ✓ 结构体、类可以定义存储属性
- ✓ 枚举不可以定义存储属性

计算属性（Computed Property）

- ✓ 本质就是方法（函数）
- ✓ 不占用实例的内存
- ✓ 枚举、结构体、类都可以定义计算属性

```
print(MemoryLayout<Circle>.stride) // 8
```

```
struct Circle {
    // 存储属性
    var radius: Double
    // 计算属性
    var diameter: Double {
        set {
            radius = newValue / 2
        }
        get {
            radius * 2
        }
    }
}
```

```
var circle = Circle(radius: 5)
print(circle.radius) // 5.0
print(circle.diameter) // 10.0
```

```
circle.diameter = 12
print(circle.radius) // 6.0
print(circle.diameter) // 12.0
```

存储属性

- 关于存储属性，Swift有个明确的规定
- 在创建类 或 结构体的实例时，必须为所有的存储属性设置一个合适的初始值
  - ✓ 可以在初始化器里为存储属性设置一个初始值
  - ✓ 可以分配一个默认的属性值作为属性定义的一部分

计算属性

- set传入的新值默认叫做newValue，也可以自定义

```
struct Circle {
    var radius: Double
    var diameter: Double {
        set(newDiameter) {
            radius = newDiameter / 2
        }
        get {
            radius * 2
        }
    }
}
```

- 只读计算属性：只有get，没有set

```
struct Circle {
    var radius: Double
    var diameter: Double {
        get {
            radius * 2
        }
    }
}
```

```
struct Circle {
    var radius: Double
    var diameter: Double { radius * 2 }
}
```

- 定义计算属性只能用var，不能用let
  - let代表常量：值是一成不变的
  - 计算属性的值是可能发生变化的（即使是只读计算属性）

枚举rawValue原理

- 枚举原始值rawValue的本质是：只读计算属性

```
enum TestEnum : Int {
    case test1 = 1, test2 = 2, test3 = 3
    var rawValue: Int {
        switch self {
        case .test1:
            return 10
        case .test2:
            return 11
        case .test3:
            return 12
        }
    }
}
```

```
print(TestEnum.test3.rawValue) // 12
```

延迟存储属性（Lazy Stored Property）

- 使用lazy可以定义一个延迟存储属性，在第一次用到属性的时候才会进行初始化

```
class Car {
    init() {
        print("Car init!")
    }
    func run() {
        print("Car is running!")
    }
}
```

```
class Person {
    lazy var car = Car()
    init() {
        print("Person init!")
    }
    func goOut() {
        car.run()
    }
}
```

```
let p = Person()
print("--------")
p.goOut()
```

```
Person init!
--------
Car init!
Car is running!
```

```
class PhotoView {
    lazy var image: Image = {
        let url = "https://www.520it.com/xx.png"
        let data = Data(url: url)
        return Image(data: data)
    }()
}
```

- lazy属性必须是var，不能是let
  - let必须在实例的初始化方法完成之前就拥有值

- 如果多条线程同时第一次访问lazy属性
  - 无法保证属性只被初始化1次

小码哥教育 SEEMYGO

延迟存储属性注意点

- 当结构体包含一个延迟存储属性时，只有var才能访问延迟存储属性
  - 因为延迟属性初始化时需要改变结构体的内存

```
struct Point {
    var x = 0
    var y = 0
    lazy var z = 0
}
let p = Point()
print(p.z)
```

Cannot use mutating getter on immutable value: 'p' is a 'let' constant

属性观察器（Property Observer）

- 可以为非lazy的var存储属性设置属性观察器

```
struct Circle {
    var radius: Double {
        willSet {
            print("willSet", newValue)
        }
        didSet {
            print("didSet", oldValue, radius)
        }
    }
    init() {
        self.radius = 1.0
        print("Circle init!")
    }
}
```

```
// Circle init!
var circle = Circle()

// willSet 10.5
// didSet 1.0 10.5
circle.radius = 10.5

// 10.5
print(circle.radius)
```

- willSet会传递新值，默认叫newValue
- didSet会传递旧值，默认叫oldValue
- 在初始化器中设置属性值不会触发willSet和didSet
- 在属性定义时设置初始值也不会触发willSet和didSet

全局变量、局部变量

- 属性观察器、计算属性的功能，同样可以应用在全局变量、局部变量身上

```
var num: Int {
    get {
        return 10
    }
    set {
        print("setNum", newValue)
    }
}
num = 11 // setNum 11
print(num) // 10
```

```
func test() {
    var age = 10 {
        willSet {
            print("willSet", newValue)
        }
        didSet {
            print("didSet", oldValue, age)
        }
    }
    age = 11
    // willSet 11
    // didSet 10 11
}
test()
```

inout的再次研究

```swift
struct Shape {
    var width: Int
    var side: Int {
        willSet {
            print("willSetSide", newValue)
        }
        didSet {
            print("didSetSide", oldValue, side)
        }
    }
    var girth: Int {
        set {
            width = newValue / side
            print("setGirth", newValue)
        }
        get {
            print("getGirth")
            return width * side
        }
    }
    func show() {
        print("width=\(width), side=\(side), girth=\(girth)")
    }
}
```

```swift
func test(_ num: inout Int) {
    num = 20
}

var s = Shape(width: 10, side: 4)
test(&s.width)
s.show()
print("----------")
test(&s.side)
s.show()
print("----------")
test(&s.girth)
s.show()
```

```
getGirth
width=20, side=4, girth=80
----------
willSetSide 20
didSetSide 4 20
getGirth
width=20, side=20, girth=400
----------
getGirth
setGirth 20
getGirth
width=1, side=20, girth=20
```

inout的本质总结

- 如果实参有物理内存地址，且没有设置属性观察器
  - 直接将实参的内存地址传入函数（实参进行引用传递）

- 如果实参是计算属性 或者 设置了属性观察器
  - 采取了Copy In Copy Out的做法
    - ✓ 调用该函数时，先复制实参的值，产生副本【get】
    - ✓ 将副本的内存地址传入函数（副本进行引用传递），在函数内部可以修改副本的值
    - ✓ 函数返回后，再将副本的值覆盖实参的值【set】

- 总结：`inout`的本质就是引用传递（地址传递）

类型属性（Type Property）

- 严格来说，属性可以分为
  - 实例属性（Instance Property）：只能通过实例去访问
    - ✓ 存储实例属性（Stored Instance Property）：存储在实例的内存中，每个实例都有1份
    - ✓ 计算实例属性（Computed Instance Property）
  - 类型属性（Type Property）：只能通过类型去访问
    - ✓ 存储类型属性（Stored Type Property）：整个程序运行过程中，就只有1份内存（类似于全局变量）
    - ✓ 计算类型属性（Computed Type Property）
- 可以通过static定义类型属性
  - 如果是类，也可以用关键字class

```
struct Car {
    static var count: Int = 0
    init() {
        Car.count += 1
    }
}
```

```
let c1 = Car()
let c2 = Car()
let c3 = Car()
print(Car.count) // 3
```

小码哥教育 SEEMYGO

类型属性细节

- 不同于存储实例属性，你必须给存储类型属性设定初始值
  - 因为类型没有像实例那样的init初始化器来初始化存储属性

- 存储类型属性默认就是lazy，会在第一次使用的时候才初始化
  - 就算被多个线程同时访问，保证只会初始化一次
  - 存储类型属性可以是let

- 枚举类型也可以定义类型属性（存储类型属性、计算类型属性）

小码哥教育 SEEMYGO

单例模式

```
public class FileManager {
    public static let shared = FileManager()
    private init() { }
}
```

```
public class FileManager {
    public static let shared = {
        // ....
        // ....
        return FileManager()
    }()
    private init() { }
}
```