Swift 类 Swift 方法

Swift 属性

Swift 属性将值跟特定的类、结构或枚举关联。

属性可分为存储属性和计算属性:

存储属性	计算属性
存储常量或变量作为实例的一部分	计算（而不是存储）一个值
用于类和结构体	用于类、结构体和枚举

存储属性和计算属性通常用于特定类型的实例。

属性也可以直接用于类型本身，这种属性称为类型属性。

另外，还可以定义属性观察器来监控属性值的变化，以此来触发一个自定义的操作。属性观察器可以添加到自己写的存储属性上，也可以添加到从父类继承的属性上。

存储属性

简单来说，一个存储属性就是存储在特定类或结构体的实例里的一个常量或变量。

存储属性可以是变量存储属性（用关键字var定义），也可以是常量存储属性（用关键字let定义）。

- 可以在定义存储属性的时候指定默认值
- 也可以在构造过程中设置或修改存储属性的值，甚至修改常量存储属性的值

```
import Cocoa

struct Number
{
   var digits: Int
   let pi = 3.1415
}

var n = Number(digits: 12345)
n.digits = 67

print("\(n.digits)")
print("\(n.pi)")
```

以上程序执行输出结果为：

```
67
3.1415
```

考虑以下代码：

```
let pi = 3.1415
```

代码中 pi 在定义存储属性的时候指定默认值（pi = 3.1415），所以不管你什么时候实例化结构体，它都不会改变。

如果你定义的是一个常量存储属性，如果尝试修改它就会报错，如下所示：

```
import Cocoa

struct Number
{
    var digits: Int
    let numbers = 3.1415
}

var n = Number(digits: 12345)
n.digits = 67

print("\(n.digits)")
print("\(n.numbers)")
n.numbers = 8.7
```

以上程序，执行会报错，错误如下所示：

```
error: cannot assign to property: 'numbers' is a 'let' constant
n.numbers = 8.7
```

意思为 'numbers' 是一个常量，你不能修改它。

延迟存储属性

延迟存储属性是指当第一次被调用的时候才会计算其初始值的属性。

在属性声明前使用 **lazy** 来标示一个延迟存储属性。

> *注意：*
> *必须将延迟存储属性声明成变量（使用`var`关键字），因为属性的值在实例构造完成之前可能无法得到。而常量属性在构造过程完成之前必须要有初始值，因此无法声明成延迟属性。*

延迟存储属性一般用于：

- 延迟对象的创建。
- 当属性的值依赖于其他未知类

```
import Cocoa

class sample {
    lazy var no = number() // `var` 关键字是必须的
}

class number {
    var name = "Runoob Swift 教程"
}

var firstsample = sample()
print(firstsample.no.name)
```

以上程序执行输出结果为：

```
Runoob Swift 教程
```

实例化变量

如果您有过 Objective-C 经验，应该知道Objective-C 为类实例存储值和引用提供两种方法。对于属性来说，也可以使用实例变量作为属性值的后端存储。

Swift 编程语言中把这些理论统一用属性来实现。Swift 中的属性没有对应的实例变量，属性的后端存储也无法直接访问。这就避免了不同场景下访问方式的困扰，同时也将属性的定义简化成一个语句。

一个类型中属性的全部信息——包括命名、类型和内存管理特征——都在唯一一个地方（类型定义中）定义。

计算属性

除存储属性外，类、结构体和枚举可以定义*计算属性*，计算属性不直接存储值，而是提供一个 getter 来获取值，一个可选的 setter 来间接设置其他属性或变量的值。

```
import Cocoa

class sample {
    var no1 = 0.0, no2 = 0.0
    var length = 300.0, breadth = 150.0

    var middle: (Double, Double) {
        get{
            return (length / 2, breadth / 2)
        }
        set(axis){
            no1 = axis.0 - (length / 2)
            no2 = axis.1 - (breadth / 2)
        }
```

```
    }
}

var result = sample()
print(result.middle)
result.middle = (0.0, 10.0)

print(result.no1)
print(result.no2)
```

以上程序执行输出结果为：

```
(150.0, 75.0)
-150.0
-65.0
```

如果计算属性的 setter 没有定义表示新值的参数名，则可以使用默认名称 newValue。

只读计算属性

只有 getter 没有 setter 的计算属性就是只读计算属性。

只读计算属性总是返回一个值，可以通过点(.)运算符访问，但不能设置新的值。

```
import Cocoa

class film {
    var head = ""
    var duration = 0.0
    var metaInfo: [String:String] {
        return [
            "head": self.head,
            "duration":"\(self.duration)"
        ]
    }
}

var movie = film()
movie.head = "Swift 属性"
movie.duration = 3.09

print(movie.metaInfo["head"]!)
print(movie.metaInfo["duration"]!)
```

以上程序执行输出结果为：

```
Swift 属性
3.09
```

> *注意：*
>
> *必须使用`var`关键字定义计算属性，包括只读计算属性，因为它们的值不是固定的。`let`关键字只用来声明常量属性，表示初始化后再也无法修改的值。*

属性观察器

属性观察器监控和响应属性值的变化，每次属性被设置值的时候都会调用属性观察器，甚至新的值和现在的值相同的时候也不例外。

可以为除了延迟存储属性之外的其他存储属性添加属性观察器，也可以通过重载属性的方式为继承的属性（包括存储属性和计算属性）添加属性观察器。

> *注意：*
> *不需要为无法重载的计算属性添加属性观察器，因为可以通过 setter 直接监控和响应值的变化。*

可以为属性添加如下的一个或全部观察器：

- `willSet`在设置新的值之前调用
- `didSet`在新的值被设置之后立即调用
- willSet和didSet观察器在属性初始化过程中不会被调用

```
import Cocoa

class Samplepgm {
    var counter: Int = 0{
        willSet(newTotal){
            print("计数器: \(newTotal)")
        }
        didSet{
            if counter > oldValue {
                print("新增数 \(counter - oldValue)")
            }
        }
    }
}
let NewCounter = Samplepgm()
NewCounter.counter = 100
NewCounter.counter = 800
```

以上程序执行输出结果为：

```
计数器: 100
新增数 100
计数器: 800
新增数 700
```

全局变量和局部变量

计算属性和属性观察器所描述的模式也可以用于全局变量和局部变量。

局部变量	全局变量
在函数、方法或闭包内部定义的变量。	函数、方法、闭包或任何类型之外定义的变量。
用于存储和检索值。	用于存储和检索值。
存储属性用于获取和设置值。	存储属性用于获取和设置值。
也用于计算属性。	也用于计算属性。

类型属性

类型属性是作为类型定义的一部分写在类型最外层的花括号（{}）内。

使用关键字 static 来定义值类型的类型属性，关键字 class 来为类定义类型属性。

```
struct Structname {
   static var storedTypeProperty = " "
   static var computedTypeProperty: Int {
      // 这里返回一个 Int 值
   }
}

enum Enumname {
   static var storedTypeProperty = " "
   static var computedTypeProperty: Int {
      // 这里返回一个 Int 值
   }
}

class Classname {
   class var computedTypeProperty: Int {
      // 这里返回一个 Int 值
   }
}
```

注意：
例子中的计算型类型属性是只读的，但也可以定义可读可写的计算型类型属性，跟实例计算属性的语法类似。

获取和设置类型属性的值

类似于实例的属性，类型属性的访问也是通过点运算符(.)来进行。但是，类型属性是通过类型本身来获取和设置，而不是通过实例。实例如下：

```
import Cocoa

struct StudMarks {
   static let markCount = 97
   static var totalCount = 0
   var InternalMarks: Int = 0 {
      didSet {
         if InternalMarks > StudMarks.markCount {
            InternalMarks = StudMarks.markCount
         }
         if InternalMarks > StudMarks.totalCount {
            StudMarks.totalCount = InternalMarks
         }
      }
   }
}

var stud1Mark1 = StudMarks()
var stud1Mark2 = StudMarks()

stud1Mark1.InternalMarks = 98
print(stud1Mark1.InternalMarks)

stud1Mark2.InternalMarks = 87
print(stud1Mark2.InternalMarks)
```

以上程序执行输出结果为：

```
97
87
```

← Swift 类 Swift 方法 →

点我分享笔记