Swift 属性 Swift 下标脚本

# Swift 方法

Swift 方法是与某些特定类型相关联的函数

在 Objective-C 中，类是唯一能定义方法的类型。但在 Swift 中，你不仅能选择是否要定义一个类/结构体/枚举，还能灵活的在你创建的类型（类/结构体/枚举）上定义方法。

## 实例方法

在 Swift 语言中，实例方法是属于某个特定类、结构体或者枚举类型实例的方法。

实例方法提供以下方法：

- 可以访问和修改实例属性
- 提供与实例目的相关的功能

实例方法要写在它所属的类型的前后大括号({})之间。

实例方法能够隐式访问它所属类型的所有的其他实例方法和属性。

实例方法只能被它所属的类的某个特定实例调用。

实例方法不能脱离于现存的实例而被调用。

### 语法

```
func funcname(Parameters) -> returntype
{
    Statement1
    Statement2
    ……
    Statement N
    return parameters
}
```

### 实例

```
import Cocoa

class Counter {
    var count = 0
    func increment() {
        count += 1
    }
    func incrementBy(amount: Int) {
        count += amount
    }
```

```
    func reset() {
        count = 0
    }
}
// 初始计数值是0
let counter = Counter()

// 计数值现在是1
counter.increment()

// 计数值现在是6
counter.incrementBy(amount: 5)
print(counter.count)

// 计数值现在是0
counter.reset()
print(counter.count)
```

以上程序执行输出结果为：

```
6
0
```

Counter类定义了三个实例方法：

- **increment** 让计数器按 1 递增；
- **incrementBy(amount: Int)** 让计数器按一个指定的整数值递增；
- **reset** 将计数器重置为0。

**Counter** 这个类还声明了一个可变属性 **count**，用它来保持对当前计数器值的追踪。

---

# 方法的局部参数名称和外部参数名称

Swift 函数参数可以同时有一个局部名称（在函数体内部使用）和一个外部名称（在调用函数时使用

Swift 中的方法和 Objective-C 中的方法极其相似。像在 Objective-C 中一样，Swift 中方法的名称通常用一个介词指向方法的第一个参数，比如：with，for，by等等。

Swift 默认仅给方法的第一个参数名称一个局部参数名称;默认同时给第二个和后续的参数名称为全局参数名称。

以下实例中 'no1' 在swift中声明为局部参数名称。'no2' 用于全局的声明并通过外部程序访问。

```
import Cocoa

class division {
    var count: Int = 0
    func incrementBy(no1: Int, no2: Int) {
        count = no1 / no2
```

```
        print(count)
    }
}

let counter = division()
counter.incrementBy(no1: 1800, no2: 3)
counter.incrementBy(no1: 1600, no2: 5)
counter.incrementBy(no1: 11000, no2: 3)
```

以上程序执行输出结果为：

```
600
320
3666
```

### 是否提供外部名称设置

我们强制在第一个参数添加外部名称把这个局部名称当作外部名称使用（Swift 2.0前是使用 # 号）。
相反，我们呢也可以使用下划线（_）设置第二个及后续的参数不提供一个外部名称。

```
import Cocoa

class multiplication {
    var count: Int = 0
    func incrementBy(first no1: Int, no2: Int) {
        count = no1 * no2
        print(count)
    }
}

let counter = multiplication()
counter.incrementBy(first: 800, no2: 3)
counter.incrementBy(first: 100, no2: 5)
counter.incrementBy(first: 15000, no2: 3)
```

以上程序执行输出结果为：

```
2400
500
45000
```

## self 属性

类型的每一个实例都有一个隐含属性叫做self，self 完全等同于该实例本身。

你可以在一个实例的实例方法中使用这个隐含的self属性来引用当前实例。

```
import Cocoa

class calculations {
    let a: Int
    let b: Int
    let res: Int

    init(a: Int, b: Int) {
        self.a = a
        self.b = b
        res = a + b
        print("Self 内: \(res)")
    }

    func tot(c: Int) -> Int {
        return res - c
    }

    func result() {
        print("结果为: \(tot(c: 20))")
        print("结果为: \(tot(c: 50))")
    }
}

let pri = calculations(a: 600, b: 300)
let sum = calculations(a: 1200, b: 300)

pri.result()
sum.result()
```

以上程序执行输出结果为：

```
Self 内: 900
Self 内: 1500
结果为: 880
结果为: 850
结果为: 1480
结果为: 1450
```

## 在实例方法中修改值类型

Swift 语言中结构体和枚举是值类型。一般情况下，值类型的属性不能在它的实例方法中被修改。

但是，如果你确实需要在某个具体的方法中修改结构体或者枚举的属性，你可以选择变异(mutating)这个方法，然后方法就可以从方法内部改变它的属性；并且它做的任何改变在方法结束时还会保留在原始结构中。

方法还可以给它隐含的self属性赋值一个全新的实例，这个新实例在方法结束后将替换原来的实例。

```
import Cocoa

struct area {
    var length = 1
    var breadth = 1

    func area() -> Int {
        return length * breadth
    }

    mutating func scaleBy(res: Int) {
        length *= res
        breadth *= res

        print(length)
        print(breadth)
    }
}

var val = area(length: 3, breadth: 5)
val.scaleBy(res: 3)
val.scaleBy(res: 30)
val.scaleBy(res: 300)
```

以上程序执行输出结果为：

```
9
15
270
450
81000
135000
```

## 在可变方法中给 self 赋值

可变方法能够赋给隐含属性 self 一个全新的实例。

```
import Cocoa

struct area {
    var length = 1
    var breadth = 1

    func area() -> Int {
        return length * breadth
    }
```

```
    mutating func scaleBy(res: Int) {
        self.length *= res
        self.breadth *= res
        print(length)
        print(breadth)
    }
}
var val = area(length: 3, breadth: 5)
val.scaleBy(res: 13)
```

以上程序执行输出结果为：

```
39
65
```

## 类型方法

实例方法是被类型的某个实例调用的方法，你也可以定义类型本身调用的方法，这种方法就叫做类型方法。

声明结构体和枚举的类型方法，在方法的func关键字之前加上关键字static。类可能会用关键字class来允许子类重写父类的实现方法。

类型方法和实例方法一样用点号(.)语法调用。

```
import Cocoa

class Math
{
    class func abs(number: Int) -> Int
    {
        if number < 0
        {
            return (-number)
        }
        else
        {
            return number
        }
    }
}

struct absno
{
    static func abs(number: Int) -> Int
    {
        if number < 0
        {
```

```
            return (-number)
        }
        else
        {
            return number
        }
    }
}

let no = Math.abs(number: -35)
let num = absno.abs(number: -5)

print(no)
print(num)
```

以上程序执行输出结果为：

```
35
5
```

← Swift 属性 Swift 下标脚本 →