链滴

swift3.0 后基础语法 / 变化 二

作者：wyw89500
原文链接：https://ld246.com/article/1481727370243
来源网站：链滴

```
//: Playground - noun: a place where people can play

import UIKit

// 1.函数

//function 函数名 (参数列表) -> 返回值类型 {函数体}

//函数相当于OC中的方法

//函数的格式如下

//func 函数名(参数列表) -> 返回值类型 {

// 代码块

// return 返回值

//}

//func是关键字,多个参数列表之间可以用逗号（,）分隔，也可以没有参数

//使用箭头“->”指向返回值类型

//如果函数没有返回值，返回值为Void.并且“-> 返回值类型”部分可以省略

//1),没参数没有返回值的函数

func eat () -> Void {

 print("apple")

}

//或者

//func run () {

// print("apple")

//}

//调用

eat()

//2),有参数没有返回值的函数

func eat (food : String) {

// print("吃" + food)

 print("chi \(food)")
```

```
}
eat(food: "banana")
//3),没参数没有返回值的函数
func message() -> String{
 return  "你还好吗?"
}
print(message())
//4),有参数有返回值的函数
func sum(num1 : Int,num2 : Int) -> Int {
 return num1 + num2;
}
let total = sum(num1 : 10, num2 : 30)
print(total)
//5).有参数并且有多个返回值
let array = [12,33,45,22,44,23]
func getnumbers(nums : [Int]) -> (Int, Int) {
 var oddCount = 0
 var eventCount = 0
 for item in nums {
 if item % 2 == 0 {
 eventCount += 1
 }else{
 oddCount += 1
 }
 }
 return (eventCount, oddCount)
}
```

```
let count = getnumbers(nums: array)
print("偶数的个数为:\(count.0),奇数的个数为:\(count.1)")
// 2.函数的使用注意
//注意一:
//swift3.0之前外部参数和内部参数
//在函数内部可以看到的参数,就是内部参数
//在函数外面可以看到的参数,就是外部参数
//默认情况下,从第二个参数开始,参数名称既是内部参数也是外部参数
//如果第一个参数也想要有外部参数,可以设置标签:在变量名前加标签即可
//如果不想要外部参数,可以在参数名称前加_
//func sum1(num1 num1 : Int,num2 : Int) -> Int {
// return num1 + num2;
//}
//print(sum1(num1: 1, num2: 1))
//swift3.0之后没有外部和内部参数的区别都是外部参数
func sum2(num1 : Int,num2 : Int) -> Int {
 return num1 + num2;
}
print(sum2(num1: 1, num2: 1))//2
func sum3(_ num1 : Int,_ num2 : Int) -> Int {
 return num1 + num2;
}
print(sum3(1, 1))//2
//注意二: 默认参数
//某些情况,如果没有传入具体的参数,可以使用默认参数
func eats(food: String = "li") -> String {
 return("吃  \(food)")
```

```
}
eats(food: "苹果")
eats(food: "香蕉")
eats()
//注意三: 可变参数
//swift中函数的参数个数可以变化，它可以接受不确定数量的输入类型参数
//它们必须具有相同的类型
//我们可以通过在参数类型名后面加入（...）的方式来指示这是可变参数
func sum4(nums : Int...) -> Int {
 var result = 0
 for num in nums {
 result += num
 }
 return result
}
sum4(nums: 1,2,3,4,5)
//注意四: 引用类型(指针的传递)
//默认情况下,函数的参数是值传递.如果想改变外面的变量,则需要传递变量的地址
//必须是变量,因为需要在内部改变其值
//Swift提供的inout关键字就可以实现 //swift3.0之前在num1之前,现在是在类型之前
//对比下列两个函数
var a = 10
var b = 20
//func exchangeNum(inout num1: Int,inout num2: Int) {
// let tempNum = num1
// num1 = num2
// num2 = tempNum
```

```
//}
func exchangeNum( num1: inout  Int, num2: inout  Int) {
 let tempNum = num1
 num1 = num2
 num2 = tempNum
}
exchangeNum(num1: &a, num2: &b)
print(a)
print(b)
//注意五:函数的嵌套使用
//swift中函数可以嵌套使用
//即函数中包含函数,但是不推荐该写法
// 3.函数的类型
//函数类型的概念
//每个函数都有属于自己的类型，由函数的参数类型和返回类型组成
//这个例子中定义了两个简单的数学函数：addTwoInts 和 multiplyTwoInts
//这两个函数都传入两个 Int 类型，  返回一个合适的Int值
//这两个函数的类型是 (Int, Int) -> Int
// 定义两个函数
func addTwoInts(a : Int, b : Int) -> Int {
 return a + b
}
func multiplyTwoInt(a : Int, b : Int) -> Int {
 return a * b
}
// 定义函数的类型
var mathFunction : (Int, Int) -> Int = addTwoInts
```

```
// 使用函数的名称
print(mathFunction(10, 20))//30
// 给函数的标识符赋值其他值
print(mathFunction = multiplyTwoInt)
// 使用函数的名称
print(mathFunction(10, 20))//200
//1.函数作为方法的参数
func printResult(a : Int, b : Int, calculateMethod : (Int, Int) -> Int) {
 print(calculateMethod(a, b))
}
printResult(a: 10, b: 20, calculateMethod: addTwoInts)//30
printResult(a: 10, b: 20, calculateMethod: multiplyTwoInt)//200
//2.函数作为方法的返回值
// 1.定义两个函数
func stepForward(num : Int) -> Int {
 return num + 1
}
func stepBackward(num : Int) -> Int {
 return num - 1
}
// 2.定义一个变量,希望该变量经过计算得到0
var num = -4
// 3.定义获取哪一个函数
func getOprationMethod(num : Int) -> ((Int) -> Int) {
 return num <= 0 ? stepForward : stepBackward
}
// 4.for进行操作
```

```
while  num != 0 {
 let oprationMethod = getOprationMethod(num: num)
 num = oprationMethod(num)
 print(num)
}
//  4.枚举类型
//枚举类型的介绍
//概念介绍
//枚举定义了一个通用类型的一组相关的值，使你可以在你的代码中以一个安全的方式来使用这些值。
//在 C/OC 语言中枚举指定相关名称为一组整型值
//Swift 中的枚举更加灵活，不必给每一个枚举成员提供一个值.也可以提供一个值是字符串，一个字，或是一个整型值或浮点值
//枚举类型的语法
//使用enum关键词并且把它们的整个定义放在一对大括号内
//enum SomeEnumeration {
// // enumeration definition goes here
//}
// 1.枚举类型的定义
//case关键词表明新的一行成员值将被定义
//不像 C 和 Objective-C 一样，Swift 的枚举成员在被创建时不会被赋予一个默认的整数值
//在下面的CompassPoints例子中，North，South，East和West不是隐式的等于0，1，2和3
enum CompassPoint {
 case North
 case South
 case East
 case West
}
//定义方式二:多个成员值可以出现在同一行上
```

```
enum Planet {
 case Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
}
//2.给枚举类型赋值
//枚举类型赋值可以是字符串/字符/整型/浮点型
//注意如果有给枚举类型赋值,则必须在枚举类型后面明确说明具体的类型
enum CompassPoint1 : Int{
 case North = 1
 case South = 2
 case East = 3
 case West = 4
}
let direction = CompassPoint1.North
let direction1 : CompassPoint1 = .East
let direction2 = CompassPoint1(rawValue: 5)//返回的是一个可选类型
//let direction2 = .West //错误
// 5.结构体
//结构体的介绍
//概念介绍
//结构体(struct)是由一系列具有相同类型或不同类型的数据构成的数据集合
//结构体(struct)指的是一种数据结构
//结构体是值类型,在方法中传递时是值传递
//结构的定义格式
//struct 结构体名称 {
// // 属性和方法 //swift即可以写熟悉还可以写方法
//}
//5.1定义结构体
```

```
struct Location {
 var x : Double
 var y : Double
}
//5.2通过结构体构建点
let center = Location(x: 100, y: 100)
print(center)
//5.3测试点的位置
let testLocation = Location(x: 50, y: 50)
//5.4求测试点是否在以center为中心半径为100的园内
func inRange(location : Location) -> Bool {
 let disX = location.x - center.x
 let disY = location.y - center.y
 let distance = sqrt((pow(disX, 2) + pow(disY, 2)))
 return distance < 200
}
inRange(location: testLocation)
//5.5结构体的增强
//扩充构造函数
//默认情况下创建Location时使用Location(x: x值, y: y值)
//但是为了让我们在使用结构体时更加的灵活,swift还可以对构造函数进行扩充
//注意:
//1.在实例化任何一个类或者结构体时,必须保证类/构造体所有的(存储)属性,都必须初始化
//2.如果自定了构造函数,那么会覆盖系统提供的构造函数,如果想继续使用原来的构造函数就必须明确重写系统的构造函数
let point = CGRect(x: 10, y: 10, width: 10, height: 10)
struct Position {
```

```
 var x : Double
 var y : Double
//
// var x : Double = 100
// var y : Double = 200
 init(x : Double, y : Double) {//默认的构造函数
 self.x = x
 self.y = y
 }
 init() {
 x = 100
 y = 200
 }
}
let position = Position(x: 10, y: 20)
let p = Position()
print(position)
//5.6为结构体扩充方法
//为了让结构体使用更加灵活,swift的结构体中可以扩充方法,要想扩充方法一般在方法前面加mutating
//例子:为了Location结构体扩充两个方法
//向水平方向移动的方法
//向垂直方向移动的方法
struct Location2 {
 var x : Double
 var y : Double

 init(x : Double, y : Double) {
```

```
 self.x = x
 self.y = y
 }

 init(xyString : String) {
 let strs = xyString.components(separatedBy: ",")
 x = Double(strs.first!)!
 y = Double(strs.last!)!// y = Double(strs[1])!strs[1]这种方法取返回的是一个具体类型不是可选类型如果这样做需要做判断防止越界,系统在这里做的不很好

 }
 func eat(){
 print("chi")
 }

 mutating  func moveH(x : Double) {
 self.x += x
 }

 mutating  func moveV(y : Double) {
 self.y += y
 }
}
var location2 = Location2(xyString: "2,2")
print(location2)
location2.eat()
location2.moveH(x: 10)
print(location2)
```

```
//5.7注意:
//可以通过extension给系统的类/结构体扩充方法
extension  CGPoint {
 mutating  func moveH(x : CGFloat) {
 self.x += x
 }

 mutating  func moveV(y : CGFloat) {
 self.y += y
 }
}
```