Swift新特性介绍（二） 函数、枚举、类与结构

可变参数

Swift的函数可以接受零个或多个指定类型的参数值，使用`...`来表示传递的是可变参数

```
func arithmeticMean(numbers: Double...) -> Double {
    var total: Double = 0
    for number in numbers {
        total += number
    }
    return total / Double(numbers.count)
}
arithmeticMean(1, 2, 3, 4, 5)
// returns 3.0, which is the arithmetic mean of these five numbers
arithmeticMean(3, 8, 19)
// returns 10.0, which is the arithmetic mean of these three numbers
```

枚举类型

和C语言中的枚举不同的是，Swift中的枚举类型没有为这一组相关名称指定一组整数值作为初始值。Swift中的枚举成员可以指定为任何类型，整数，字符或者浮点数。

```
enum CompassPoint {
    case North
    case South
    case East
    case West
}
```

上面的例子中的`CompassPoint.North`并不默认为0。

便捷性

一旦声明了一个枚举类型

```
var directionToHead = CompassPoint.West
```

在修改这个枚举值的时候就可以省略枚举类名了

```
directionToHead = .East
```

功能性

Swift中的枚举类型可以被声明为更高级的形式：

```
enum Barcode {
    case UPCA(Int, Int, Int)
    case QRCode(String)
}
```

这个Barcode枚举被定义为两种可能，UPC-A模式的条码，关联值了三个`Int`值；或者是二维码，关键一个`String`的值。

然后就可以方便地创建条码类型：

```
var productBarcode = Barcode.UPCA(8, 85909_51226, 3)
productBarcode = .QRCode("ABCDEFGHIJKLMNOP")
```

然后在switch语句中使用：

```
switch productBarcode {
    case .UPCA(let numberSystem, let identifier, let check):
    println("UPC-A with value of \(numberSystem), \(identifier), \(check).")
    case .QRCode(let productCode):
    println("QR code with value of \(productCode).")
}
// prints "QR code with value of ABCDEFGHIJKLMNOP."
```

类与结构

结构和枚举是数值类型

当赋值或者作为参数传递时，是完整地创建了一个副本，而不是使用的引用对象：

```
let hd = Resolution(width: 1920, height: 1080)
var cinema = hd
cinema.width = 2048
println("cinema is now \(cinema.width) pixels wide")
// prints "cinema is now 2048 pixels wide"
println("hd is still \(hd.width) pixels wide")
// prints "hd is still 1920 pixels wide"
```

类和结构是引用类型

当赋值或者作为参数传递时，传递的是引用对象。下面例子中`VideoMode`是一个类：

```
let tenEighty = VideoMode()
tenEighty.resolution = hd
tenEighty.interlaced = true
tenEighty.name = "1080i"
tenEighty.frameRate = 25.0
let alsoTenEighty = tenEighty
alsoTenEighty.frameRate = 30.0
println("The frameRate property of tenEighty is now \(tenEighty.frameRate)")
// prints "The frameRate property of tenEighty is now 30.0"
```

判断引用是否相等，可以使用`===`和`!==`。

字典的赋值

字典赋值或者作为参数传递时，会被完整地复制，创建一个副本：

```
var ages = ["Peter": 23, "Wei": 35, "Anish": 65, "Katya": 19]
var copiedAges = ages
copiedAges["Peter"] = 24
println(ages["Peter"])
// prints "23"
```

数组的赋值

数组只有在进行了修改数组长度操作时，才会完整地复制，创建一个副本。其他时候使用的是引用：

```
var a = [1, 2, 3]
var b = a
var c = a
println(a[0])
// 1
println(b[0])
// 1
println(c[0])
// 1
a[0] = 42
println(a[0])
// 42
println(b[0])
// 42
println(c[0])
// 42
a.append(4)
a[0] = 777
//数组a添加了一个数字，改变了数组大小，所以数组a创建了一个副本
println(a[0])
// 777
println(b[0])
// 42
println(c[0])
// 42
```

可以使用`unshare`方法来使一个数组“独立”，成为一个新的“副本”

```
b.unshare()
b[0] = -105
println(a[0])
// 777
println(b[0])
// -105
println(c[0])
// 42
```

也可以使用`copy`方法来完成强制拷贝

```
var names = ["Mohsen", "Hilary", "Justyn", "Amy", "Rich", "Graham", "Vic"]
var copiedNames = names.copy()
copiedNames[0] = "Mo"
println(names[0])
// prints "Mohsen"
```

翻译工作已告一段落，校对工作继续进行中~ 欢迎参与~

校对地址：https://github.com/letsswift/The-Swift-Programming-Language-in-Chinese

Swift新特性学习系列文章更新中

本文由LetsSwift.com原创发布

转载请注明本文原始链接：http://letsswift.com/2014/06/swift-new-features-two/