The Swift Programming Language in Chinese

https://github.com/letsswift

http://letsswift.com/

June 15, 2014

目录

1 Swift 中文教程（七）闭包 **1**

1.1 闭包表达式 1

1.1.1 Sort 函数 2

1.1.2 闭包表达式语法 3

1.1.3 根据上下文推断类型 3

1.1.4 单行表达式闭包可以省略 return 4

1.1.5 参数名简写 4

1.1.6 运算符函数 5

1.2 Trailing 闭包 5

1.2.1 获取值 7

1.3 引用类型闭包 9

1 Swift 中文教程（七）闭包

闭包（Closures）是独立的函数代码块，能在代码中传递及使用。Swift 中的闭包与 C 和 Objective-C 中的代码块及其它编程语言中的匿名函数相似。

闭包可以在上下文的范围内捕获、存储任何被定义的常量和变量引用。因这些常量和变量的封闭性，而命名为 “闭包（Closures）”。Swift 能够对所有你所能捕获到的引用进行内存管理。

> NOTE 假如你对 “捕获（capturing）” 不熟悉，请不要担心，具体可以参考 Capturing Values（捕获值）。

- 全局函数和嵌套函数已在 Functions(函数) 中介绍过，实际上这些都是特殊的闭包函数
- 全局函数都是闭包，特点是有函数名但没有捕获任何值。
- 嵌套函数都是闭包，特点是有函数名，并且可以在它封闭的函数中捕获值。
- 闭包表达式都是闭包，特点是没有函数名，可以使用轻量的语法在它所围绕的上下文中捕获值。

Swift 的闭包表达式有着干净，清晰的风格，并常见情况下对于鼓励简短、整洁的语法做出优化。这些优化包括:

- 推理参数及返回值类型源自上下文
- 隐式返回源于单一表达式闭包
- 简约参数名
- 尾随闭包语法

1.1 闭包表达式

嵌套函数已经在 Nested Functions（嵌套函数）中有所介绍，是种方便命名和定义自包含代码块的一种方式，然而，有时候在编写简短函数式的构造器时非常有用，它不需要完整的函数声明及函数名，尤其是在你需要调用一个或多个参数的函数时。

闭包表达式是一种编写内联闭包的方式，它简洁、紧凑。闭包表达式提供了数种语义优化，为的是以最简单的形式编程而不需要大量的声明或意图。以下以同一个 sort 函数进行几次改进，每次函数都更加简洁，以此说明闭包表达式的优化。

1.1.1 Sort 函数

Swift 的标准函数库提供了一个名为 sort 的函数，它通过基于输出类型排序的闭包函数，给已知类型的数组数据的值排序。一旦完成排序工作，会返回一个同先前数组相同大小，相同数据类型，并且的新数组，并且这个数组的元素都在正确排好序的位置上。The closure expression examples below use the sort function to sort an array of String values in reverse alphabetical order. Here's the initial array to be sorted:

以下的闭包表达式通过 sort 函数将 String 值按字母顺序进行排序作说明，这是待排序的初始化数组。

let names = [“Chris”, “Alex”, “Ewa”, “Barry”, “Daniella”]

sort 函数需要两个参数：

- 一个已知值类型的数组
- 一个接收两个参数的闭包函数，这两个参数的数据类型都同于数组元素。并且返回一个 Bool 表明是否第一个参数应排在第二个参数前或后。

这个例子是一组排序的字符串值，因此需要排序的封闭类型的函数（字符串，字符串）-> Bool。

构造排序闭包的一种方式是书写一个符合其类型要求的普通函数:backwards，并将其返回值作为 sort 函数的第二个参数传入：

```
func backwards(s1: String, s2: String) -> Bool {
    return s1 > s2
}
var reversed = sort(names, backwards)
// reversed is equal to ["Ewa", "Daniella", "Chris", "Barry", "Alex"]
```

如果 backwards 函数参数 s1 大于 s2，则返回 true 值，表示在新的数组排序中 s1 应该出现在 s2 前。字符中的 “大于” 表示 “按照字母顺序后出现”。这意味着字母 “B” 大于字母 “A”, 字符串 “Tom” 大于字符串 “Tim”。其将进行字母逆序排序，”Barry” 将会排在 “Alex” 之后, 以此类推。

但这是一个相当冗长的方式，本质上只是做了一个简单的单表达式函数：(a > b)。下面的例子中，我们利用闭合表达式可以相比上面的例子更效率的构造一个内联排序闭包。

1.1.2 闭包表达式语法

闭合表达式语法具有以下一般构造形式：

```
{ (parameters) -> return type in
    statements
}
```

闭包表达式语法可以使用常量参数、变量参数和 inout 类型作为参数，但皆不可提供默认值。如果你需要使用一个可变的参数，可将可变参数放在最后，元组类型也可以作为参数和返回值使用。

下面的例子展示了上面的 backwards 函数对应的闭包表达式构造函数代码

```
reversed = sort(names, { (s1: String, s2: String) -> Bool in
return s1 > s2
})
```

需要注意的是声明内联闭包的参数和返回值类型与 backwards 函数类型声明相同。在这两种方式中，都写成了 (s1: String, s2: String) -> Bool 类型。然而在内联闭包表达式中，函数和返回值类型都写在大括号内，而不是大括号外。

闭包的函数体部分由关键字 in 引入。该关键字表示闭包的参数和返回值类型定义已经完成，闭包函数体即将开始。

因为这个闭包的函数体非常简约短所以完全可以将上面的 backwards 函数缩写成一行连贯的代码

reversed = sort(names, { (s1: String, s2: String) -> Bool in return s1 > s2 }
)

可以看出 sort 函数的整体调用保持不变，还是一对圆括号包含两个参数变成了内联闭包形式、只不过第二个参数的值变成了。而其中一个参数现在变成了内联闭包 (相比于 backwards 版本的代码)。

1.1.3 根据上下文推断类型

因为排序闭包是作为函数的参数进行传入的，Swift 可以推断其参数和返回值的类型。sort 期望第二个参数是类型为 (String, String) -> Bool 的函数，因此实际上 String, String 和 Bool 类型并不需要作为闭包表达式定义中的一部分。因为所有的类型都可以被正确推断，返回箭头 (->) 和围绕在参数周围的括号也可以被省略：

```
reversed = sort(names, { s1, s2 in return s1 > s2 } )
```

实际情况下，通过构造内联闭包表达式的闭包作为函数的参数传递给函数时，都可以判断出闭包的参数和返回值类型，这意味着您几乎不需要利用完整格式构造任何内联闭包。

同样，如果你希望避免阅读函数时可能存在的歧义，你可以直接明确参数的类型。

这个排序函数例子，闭包的目的是很明确的，即排序被替换，而且对读者来说可以安全的假设闭包可能会使用字符串值, 因为它正协助一个字符串数组进行排序。

1.1.4 单行表达式闭包可以省略 **return**

单行表达式闭包可以通过隐藏 return 关键字来隐式返回单行表达式的结果，如上版本的例子可以改写为:

```
reversed = sort(names, { s1, s2 in s1 > s2 } )
```

在这个例子中，sort 函数的第二个参数函数类型明确了闭包必须返回一个 Bool 类型值。因为闭包函数体只包含了一个单一表达式 (s1 > s2)，该表达式返回 Bool 类型值，因此这里没有歧义，return 关键字可以省略。

1.1.5 参数名简写

Swift 自动为内联函数提供了参数名称简写功能，可以直接通过 $0,$1,$2 等名字来引用闭包的参数值。

如果在闭包表达式中使用参数名称简写，可以在闭包参数列表中省略对其的定义，并且对应参数名称简写的类型会通过函数类型进行推断。in 关键字也同样可以被省略，因为此时闭包表达式完全由闭包函数体构成:

```
reversed = sort(names, { $0 > $1 } )
```

在这个例子中，$0 和 $1 表示闭包中第一个和第二个 String 类型的参数。

1.1.6 运算符函数

运算符函数实际上是一个更短的方式构造以上的表达式。

```
reversed = sort(names, >)
```

更多关于运算符表达式的内容请查看 Operator Functions 。

1.2 Trailing 闭包

如果您需要将一个很长的闭包表达式作为最后一个参数传递给函数，可以使用 trailing 闭包来增强函数的可读性。

Trailing 闭包是一个书写在函数括号之外 (之后) 的闭包表达式，函数支持将其作为最后一个参数调用。

```
func someFunctionThatTakesAClosure(closure: () -> ()) {
    // function body goes here
}
// here's how you call this function without using a trailing closure:
someFunctionThatTakesAClosure({
    // closure's body goes here
})
// here's how you call this function with a trailing closure instead:
someFunctionThatTakesAClosure() {
    // trailing closure's body goes here
}
```

> 注意: 如果函数只需要闭包表达式一个参数，当您使用 trailing 闭包时，您甚至可以把 () 省略掉。

在上例中作为 sort 函数参数的字符串排序闭包可以改写为:

```
reversed = sort(names) { $0 > $1 }
```

当闭包非常长以至于不能在一行中进行书写时，Trailing 闭包就变得非常有用。举例来说，Swift 的 Array 类型有一个 map 方法，其获取一个闭包表达式作为其唯一参数。数组中的每一个元素调用一次该闭包函数，并返回该元素所映射的值 (也可以是不同类型的值)。具体的映射方式和返回值类型由闭包来指定。

当提供给数组闭包函数后，map 方法将返回一个新的数组，数组中包含了与原数组一一对应的映射后的值。

下例介绍了如何在 map 方法中使用 trailing 闭包将 Int 类型数组 [16,58,510] 转换为包含对应 String 类型的数组 [“OneSix”, “FiveEight”, “FiveOneZero”]:

```
let digitNames = [
    0: "Zero", 1: "One", 2: "Two", 3: "Three", 4: "Four",
    5: "Five", 6: "Six", 7: "Seven", 8: "Eight", 9: "Nine"
]
let numbers = [16, 58, 510]
```

上面的代码创建了整数数字到他们的英文名字之间映射字典。同时定义了一个准备转换为字符串的整型数组。

你现在可以通过传递一个 trailing 闭包给 numbers 的 map 方法来创建对应的字符串版本数组。需要注意的时调用 numbers.map 不需要在 map 后面包含任何括号，因为只需要传递闭包表达式这一个参数，并且该闭包表达式参数通过 trailing 方式进行撰写:

```
let strings = numbers.map {
    (var number) -> String in
    var output = ""
    while number > 0 {
        output = digitNames[number % 10]! + output
    number /= 10
}
return output
}
// strings is inferred to be of type String[]
// its value is ["OneSix", "FiveEight", "FiveOneZero"]
```

map 在数组中为每一个元素调用了闭包表达式。您不需要指定闭包的输入参数 number 的类型，因为可以通过要映射的数组类型进行推断。

闭包 number 参数被声明为一个变量参数 (变量的具体描述请参看 Constant and Variable Parameters)，因此可以在闭包函数体内对其进行修改。闭包表达式制定了返回值类型为 String，以表明存储映射值的新数组类型为 String。

闭包表达式在每次被调用的时候创建了一个字符串并返回。其使用求余运算符 (number % 10) 计算最后一位数字并利用 digitNames 字典获取所映射的字符串。

> 注意：字典 digitNames 下标后跟着一个叹号 (!)，因为字典下标返回一个可选值 (optional value)，表明即使该 key 不存在也不会查找失败。在上例中，它保证了 number % 10 可以总是作为一个 digitNames 字典的有效下标 key。因此叹号可以用于强展开 (force-unwrap) 存储在可选下标项中的 String 类型值。

从 digitNames 字典中获取的字符串被添加到输出的前部，逆序建立了一个字符串版本的数字。(在表达式 number % 10 中，如果 number 为 16，则返回 6，58 返回 8，510 返回 0)。

number 变量之后除以 10。因为其是整数，在计算过程中未除尽部分被忽略。因此 16 变成了 1，58 变成了 5，510 变成了 51。

整个过程重复进行，直到 number /= 10 为 0，这时闭包会将字符串输出，而 map 函数则会将字符串添加到所映射的数组中。

上例中 trailing 闭包语法在函数后整洁封装了具体的闭包功能，而不再需要将整个闭包包裹在 map 函数的括号内。

1.2.1 获取值

闭包可以在其定义的范围内捕捉（引用/得到）常量和变量，闭包可以引用和修改这些值，即使定义的常量和变量已经不复存在了依然可以修改和引用。牛逼吧、

在 Swift 中最简单形式是一个嵌套函数, 写在另一个函数的方法里面。嵌套函数可以捕获任何外部函数的参数，也可以捕获任何常量和变量在外部函数的定义。

看下面这个例子，一个函数方法为 makeIncrementor、这是一个嵌套函数，在这个函数体内嵌套了另一个函数方法：incrementor，在这个 incrementor 函数体内有两个参数：runningTotal 和 amount，实际运作时传进所需的两个参数后，incrementor 函数每次被调用时都会返回一个 runningTotal 值提供给外部的 makeIncrementor 使用：

```
func makeIncrementor(forIncrement amount: Int) -> () -> Int {
            var runningTotal = 0
            func incrementor() -> Int {
            runningTotal += amount
            return runningTotal
        }
```

```
    return incrementor
}
```

而函数 makeincrementor 的返回类型值我们可以通过函数名后面的 () -> int 得知返回的是一个 Int 类型的值。如需想学习了解更多地函数返回类型，可以参考：Function Types as Return Types.（超链接跳转）

我们可以看见 makeincrementor 这个函数体内首先定义了一个整型变量：runningtotal，初始值为 0 ，而 incrementor() 函数最终运行的出来的返回值会赋值给这个整型变量。

makeincrementor 函数 () 中向外部抛出了一个 forIncrement 参数供外部穿参进来、一旦有值进入函数体内会被函数实例化替代为 amount，而 amount 会被传递进内嵌的 incrementor 函数体中与整型常量 runningTotal 相加得到一个新的 runningTotal 并返回。而我们这个主函数要返回的值是 Int 类型，runningTotal 直接作为最终值被返回出去、makeincrementor 函数 () 执行完毕。

makeincrementor 函数 () 在其内部又定义了一个新的函数体 incrementor，作用就是将外部传递过来的值 amount 传进 incrementor 函数中与整形常量 runningTotal 相加得到一个新的 runningTotal,

单独的看 incrementor 函数、你会发现这个函数不寻常：

func incrementor() -> Int { runningTotal += amount return runningTotal }

因为 incrementor 函数没有任何的参数，但是在它的函数方法体内却指向 runningTotal 和 amount，显而易见、这是 incrementor 函数获取了外部函数的值 amount，incrementor 不能去修改它但是却可以和体内的 runningTotal 相加得出新的 runningTotal 值返回出去。

不过，由于 runningtotal 每次被调用时都会相加改变一次实际值，相应地 incrementor 函数被调用时会去加载最新的 runningtotal 值，而不再是第一次舒适化的 0. 并且需要保证每次 runningTotal 的值在 makeIncrementor 函数体内不会丢失直到函数完全加载完毕。要能确保在函数体内下一次引用时上一次的值依然还在。

> 注意 Swift 中需要明确知道什么时候该引用什么时候该赋值，在 incrementor 函数中你不需要注解 amount 和 runningTotal。Swift 还负责处理当函数不在需要 runningTotal 的时候，内存应该如何去管理。

这里有一个例子 makeIncrementor 函数：

```
let incrementByTen = makeIncrementor(forIncrement: 10)
```

1.3 引用类型闭包

在上面的例子中，incrementBySeven 和 incrementByTen 是常量，但是这些常量在闭包的状态下依然可以被修改。为何？很简单，因为函数和闭包是引用类型。当你指定一个函数或一个闭包常量/变量时、实际上是在设置该常量或变量是否为一个引用函数。在上面的例子中，它是闭合的选择，incrementByTen 指的是恒定的，而不是封闭件本身的内容。这也意味着，如果你分配一个封闭两种不同的常量或变量，这两个常量或变量将引用同一个闭包：

```
let alsoIncrementByTen = incrementByTen
alsoIncrementByTen()
// returns a value of 50
```

参考文献