Swift3.0 闭包整理

2016-12-01 09:55 编辑：耐旱的仙人掌 分类：iOS开发 来源：bluajack的简书 0 932

闭包

招聘信息：算法工程师

语法表达式

一般形式：{

```
            (parameters) -> returnType in
             statements
           }
```

- 这里的参数(parameters)，可以是in-out(输入输出参数)，但不能设定默认值。如果是可变参数，必须放在最后一位，不然编译器报错。元组也可以作为参数或者返回值。
- "in"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。即由in引入函数
- 例子

```
//一般形式
let calAdd:(Int,Int)->(Int) = {
    (a:Int,b:Int) -> Int in
    return a + b
}
print(calAdd(100,150))

//Swift可以根据闭包上下文推断参数和返回值的类型，所以上面的例子可以简化如下
let calAdd2:(Int,Int)->(Int) = {
    a,b in  //也可以写成(a,b) in
    return a + b
}
print(calAdd2(150,100))
//上面省略了返回箭头和参数及返回值类型，以及参数周围的括号。当然你也可以加括号，为了好看点，看的清楚点。(a,b)

//单行表达式闭包可以隐式返回，如下，省略return
let calAdd3:(Int,Int)->(Int) = {(a,b) in a + b}
print(calAdd3(50,200))
```

热门资讯


AFNetworking到底做了什么?
点击量 11221


TIOBE 12月编程语言排行榜 C语言颓势不改
点击量 8653


iOS 升级HTTPS通过ATS你所要知道的
点击量 8487


开发者与谷歌享受一整天的幸福：GOOGLE
点击量 7937


2017编程趋势预测：这10种编程技术将成为
点击量 5756


2016年iOS技术圈回顾
点击量 5615


如何使用Xcode Server进行持续集成并自动部
点击量 4429


iOS那些“垃圾”的轮播
点击量 4291


聪明的程序猿和2b程序员的区别
点击量 3840


从培训班出来之后找工作的经历，教会了我这
点击量 3655

综合评论

关于时间同步的那个问题，你写的这种如果我本地修改的系统时间，不久不准
nslogyanmingyu 评论了 iOS关于时间的处理

123123
EnjoyTheLife 评论了 高度自定义的TabBarController 两个...

不知所云
张凤丽 评论了 聊一聊我在移动平台混合开发的经验...

```
//如果闭包没有参数，可以直接省略“in”
let calAdd4:()->Int = {return 100 + 150}
print("....\(calAdd4())")

//这个写法，我随便写的。打印出“我是250”
//这个是既没有参数也没返回值，所以把return和in都省略了
let calAdd5:()->Void = {print("我是250")}
calAdd5()
```

- 归纳
  闭包类型是由参数类型和返回值类型决定，和函数是一样的。比如上面前三种写法的闭包的闭包类型就是(Int,Int)->(Int),后面的类型分别是()->Int和()->Void。分析下上面的代码：let calAdd：(add类型)。这里的add类型就是闭包类型 (Int,Int)->(Int)。意思就是声明一个calAdd常量，其类型是个闭包类型。
  "="右边是一个代码块，即闭包的具体实现，相当于给左边的add常量赋值。兄弟们，是不是感觉很熟悉了，有点像OC中的block代码块。

起别名

- 也可以关键字“typealias”先声明一个闭包数据类型。类似于OC中的typedef起别名

```
typealias AddBlock = (Int, Int) -> (Int)

let Add:AddBlock = {
    (c,d) in
    return c + d
}

let Result = Add(100,150)
print("Result = \(Result)")
```

尾随闭包

- 若将闭包作为函数最后一个参数，可以省略参数标签,然后将闭包表达式写在函数调用括号后面

```
func testFunction(testBlock: ()->Void){
    //这里需要传进来的闭包类型是无参数和无返回值的
    testBlock()
}
//正常写法
testFunction(testBlock: {
    print("正常写法")
})
//尾随闭包写法
testFunction(){
    print("尾随闭包写法")
}
//也可以把括号去掉，也是尾随闭包写法。推荐写法
testFunction {
```

```
    print("去掉括号的尾随闭包写法")
}
```

值捕获

- 闭包可以在其被定义的上下文中捕获常量或变量。Swift中，可以捕获值的闭包的最简单形式是嵌套函数，也就是定义在其他函数的函数体内的函数。

```
func captureValue(sums amount:Int) -> ()->Int{
    var total = 0
    func incrementer()->Int{
        total += amount
        return total
    }
    return incrementer
}

print(captureValue(sums: 10)())
print(captureValue(sums: 10)())
print(captureValue(sums: 10)())
//打印"10 10 10"
```

这里没有值捕获的原因是，没有去用一个常量或变量去引用函数，所以每次使用的函数都是新的。有点类似于OC中的匿名对象。

```
let referenceFunc = captureValue(sums: 10)
print(referenceFunc())
print(referenceFunc())
print(referenceFunc())
//打印"10 20 30"
```

这里值捕获了，是因为函数被引用了，所以没有立即释放掉。所以函数体内的值可以被捕获

- 闭包形式

```
func captureValue2(sums amount:Int) -> ()->Int{
    var total = 0
    let AddBlock:()->Int = {
        total += amount
        return total
    }
    return AddBlock
}

let testBlock = captureValue2(sums: 100)
print(testBlock())
print(testBlock())
print(testBlock())
```

由上面的例子都可以证得，函数和闭包都是引用类型。

逃逸闭包

- 当一个闭包作为参数传到一个函数中，需要这个闭包在函数返回之后才被执行，我们就称该闭包从函数种逃逸。一般如果闭包在函数体内涉及到异步操作，但函数却是很快就会执行完毕并返回的，闭包必须要逃逸掉，以便异步操作的回调。
- 逃逸闭包一般用于异步函数的回调，比如网络请求成功的回调和失败的回调。语法：在函数的闭包行参前加关键字“@escaping”。

```
//例1
func doSomething(some: @escaping () -> Void){
    //延时操作，注意这里的单位是秒
    DispatchQueue.main.asyncAfter(deadline: DispatchTime.now() + 1) {
        //1秒后操作
        some()
    }
    print("函数体")
}
doSomething {
    print("逃逸闭包")
}

//例2
var comletionHandle: ()->String = {"约吗?"}

func doSomething2(some: @escaping ()->String){
    comletionHandle = some
}
doSomething2 {
    return "叔叔，我们不约"
}
print(comletionHandle())

//将一个闭包标记为@escaping意味着你必须在闭包中显式的引用self。
//其实@escaping和self都是在提醒你，这是一个逃逸闭包，
//别误操作导致了循环引用！而非逃逸包可以隐式引用self。

//例子如下
var completionHandlers: [() -> Void] = []
//逃逸
func someFunctionWithEscapingClosure(completionHandler: @escaping () -> Void) {
    completionHandlers.append(completionHandler)
}
//非逃逸
func someFunctionWithNonescapingClosure(closure: () -> Void) {
    closure()
}
```

```
class SomeClass {
    var x = 10
    func doSomething() {
        someFunctionWithEscapingClosure { self.x = 100 }
        someFunctionWithNonescapingClosure { x = 200 }
    }
}
```

自动闭包

- 顾名思义，自动闭包是一种自动创建的闭包，封装一堆表达式在自动闭包中，然后将自动闭包作为参数传给函数。而自动闭包是不接受任何参数的，但可以返回自动闭包中表达式产生的值。
- 自动闭包让你能够延迟求值，直到调用这个闭包，闭包代码块才会被执行。说白了，就是语法简洁了，有点懒加载的意思。

```
var array = ["I","have","a","apple"]
print(array.count)
//打印出"4"

let removeBlock = {array.remove(at: 3)}//测试了下，这里代码超过一行，返回值失效。
print(array.count)
//打印出"4"

print("执行代码块移除\(removeBlock())")
//打印出"执行代码块移除apple" 这里自动闭包返回了apple值

print(array.count)
//打印出"3"
```

上一篇：中介者模式

下一篇：iOS开发实用技巧—Objective-C中的各种遍历（迭代）方式

相关资讯

- Swift Optionals 源码解析
- iOS 10 Day by Day 4：用 UIViewPropertyAnimator 编写
- Xcode8调试黑科技：Memory Graph实战解决闭包引用循
- iOS闭包循环引用精讲
- Block剧终：Objective-C中的闭包性和匿名函数

我来说两句

所有评论（0）