Swift 函数 Swift 枚举

Swift 闭包

闭包(Closures)是自包含的功能代码块，可以在代码中使用或者用来作为参数传值。

Swift 中的闭包与 C 和 Objective-C 中的代码块（blocks）以及其他一些编程语言中的 匿名函数比较相似。

全局函数和嵌套函数其实就是特殊的闭包。

闭包的形式有：

全局函数	嵌套函数	闭包表达式
有名字但不能捕获任何值。	有名字，也能捕获封闭函数内的值。	无名闭包，使用轻量级语法，可以根据上下文环境捕获值。

Swift中的闭包有很多优化的地方:

1. 根据上下文推断参数和返回值类型
2. 从单行表达式闭包中隐式返回（也就是闭包体只有一行代码，可以省略return）
3. 可以使用简化参数名，如$0, $1(从0开始，表示第i个参数...)
4. 提供了尾随闭包语法(Trailing closure syntax)

语法

以下定义了一个接收参数并返回指定类型的闭包语法：

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

实例

```
import Cocoa

let studname = { print("Swift 闭包实例。") }
studname()
```

以上程序执行输出结果为：

```
Swift 闭包实例。
```

以下闭包形式接收两个参数并返回布尔值：

```
{(Int, Int) -> Bool in
   Statement1
   Statement 2
    ---
   Statement n
}
```

实例

```
import Cocoa

let divide = {(val1: Int, val2: Int) -> Int in
   return val1 / val2
}
let result = divide(200, 20)
print (result)
```

以上程序执行输出结果为：

```
10
```

闭包表达式

闭包表达式是一种利用简洁语法构建内联闭包的方式。 闭包表达式提供了一些语法优化，使得撰写闭包变得简单明了。

sorted 方法

Swift 标准库提供了名为 **sorted(by:)** 的方法，会根据您提供的用于排序的闭包函数将已知类型数组中的值进行排序。

排序完成后，sorted(by:) 方法会返回一个与原数组大小相同，包含同类型元素且元素已正确排序的新数组。原数组不会被 sorted(by:) 方法修改。

sorted(by:)方法需要传入两个参数：

1. 已知类型的数组
2. 闭包函数，该闭包函数需要传入与数组元素类型相同的两个值，并返回一个布尔类型值来表明当排序结束后传入的第一个参数排在第二个参数前面还是后面。如果第一个参数值出现在第二个参数值前面，排序闭包函数需要返回 **true**，反之返回 **false**。

实例

```
import Cocoa

let names = ["AT", "AE", "D", "S", "BE"]

// 使用普通函数(或内嵌函数)提供排序功能,闭包函数类型需为(String, String) -> Bool。
func backwards(s1: String, s2: String) -> Bool {
    return s1 > s2
```

```
}
var reversed = names.sorted(by: backwards)

print(reversed)
```

以上程序执行输出结果为：

```
["S", "D", "BE", "AT", "AE"]
```

如果第一个字符串 (s1) 大于第二个字符串 (s2)，backwards函数返回true，表示在新的数组中s1应该出现在s2前。 对于字符串中的字符来说，"大于" 表示 "按照字母顺序较晚出现"。 这意味着字母"B"大于字母"A"，字符串"S"大于字符串"D"。 其将进行字母逆序排序，"AT"将会排在"AE"之前。

参数名称缩写

Swift 自动为内联函数提供了参数名称缩写功能，您可以直接通过$0,$1,$2来顺序调用闭包的参数。

实例

```
import Cocoa

let names = ["AT", "AE", "D", "S", "BE"]

var reversed = names.sorted( by: { $0 > $1 } )
print(reversed)
```

$0和$1表示闭包中第一个和第二个String类型的参数。
以上程序执行输出结果为：

```
["S", "D", "BE", "AT", "AE"]
```

如果你在闭包表达式中使用参数名称缩写, 您可以在闭包参数列表中省略对其定义, 并且对应参数名称缩写的类型会通过函数类型进行推断。in 关键字同样也可以被省略.

运算符函数

实际上还有一种更简短的方式来撰写上面例子中的闭包表达式。
Swift 的`String`类型定义了关于大于号 (>) 的字符串实现，其作为一个函数接受两个`String`类型的参数并返回`Bool`类型的值。 而这正好与`sort(_:)`方法的第二个参数需要的函数类型相符合。 因此，您可以简单地传递一个大于号，Swift可以自动推断出您想使用大于号的字符串函数实现：

```
import Cocoa

let names = ["AT", "AE", "D", "S", "BE"]
```

```
var reversed = names.sorted(by: >)
print(reversed)
```

以上程序执行输出结果为：

```
["S", "D", "BE", "AT", "AE"]
```

尾随闭包

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

```
func someFunctionThatTakesAClosure(closure: () -> Void) {
    // 函数体部分
}

// 以下是不使用尾随闭包进行函数调用
someFunctionThatTakesAClosure({
    // 闭包主体部分
})

// 以下是使用尾随闭包进行函数调用
someFunctionThatTakesAClosure() {
  // 闭包主体部分
}
```

实例

```
import Cocoa

let names = ["AT", "AE", "D", "S", "BE"]

//尾随闭包
var reversed = names.sorted() { $0 > $1 }
print(reversed)
```

sort() 后的 { $0 > $1} 为尾随闭包。

以上程序执行输出结果为：

```
["S", "D", "BE", "AT", "AE"]
```

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

```
reversed = names.sorted { $0 > $1 }
```

捕获值

闭包可以在其定义的上下文中捕获常量或变量。

即使定义这些常量和变量的原域已经不存在，闭包仍然可以在闭包函数体内引用和修改这些值。

Swift最简单的闭包形式是嵌套函数，也就是定义在其他函数的函数体内的函数。

嵌套函数可以捕获其外部函数所有的参数以及定义的常量和变量。

看这个例子：

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

一个函数makeIncrementor ，它有一个Int型的参数amout, 并且它有一个外部参数名字forIncremet，意味着你调用的时候，必须使用这个外部名字。返回值是一个`()-> Int`的函数。

函数题内，声明了变量runningTotal 和一个函数incrementor。

incrementor函数并没有获取任何参数，但是在函数体内访问了runningTotal和amount变量。这是因为其通过捕获在包含它的函数体内已经存在的runningTotal和amount变量而实现。

由于没有修改amount变量，incrementor实际上捕获并存储了该变量的一个副本，而该副本随着incrementor一同被存储。

所以我们调用这个函数时会累加：

```
import Cocoa

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

let incrementByTen = makeIncrementor(forIncrement: 10)

// 返回的值为10
print(incrementByTen())
```

```
// 返回的值为20
print(incrementByTen())

// 返回的值为30
print(incrementByTen())
```

以上程序执行输出结果为：

```
10
20
30
```

闭包是引用类型

上面的例子中，incrementByTen是常量，但是这些常量指向的闭包仍然可以增加其捕获的变量值。

这是因为函数和闭包都是引用类型。

无论您将函数/闭包赋值给一个常量还是变量，您实际上都是将常量/变量的值设置为对应函数/闭包的引用。 上面的例子中，incrementByTen指向闭包的引用是一个常量，而并非闭包内容本身。

这也意味着如果您将闭包赋值给了两个不同的常量/变量，两个值都会指向同一个闭包：

```
import Cocoa

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

let incrementByTen = makeIncrementor(forIncrement: 10)

// 返回的值为10
incrementByTen()

// 返回的值为20
incrementByTen()

// 返回的值为30
incrementByTen()

// 返回的值为40
incrementByTen()
```

```
let alsoIncrementByTen = incrementByTen

// 返回的值也为50
print(alsoIncrementByTen())
```

以上程序执行输出结果为：

```
50
```

Swift 函数 Swift 枚举

1 篇笔记

写笔记

闭包 可以类似 OC 里的 block 一样当作是一个对象，所以在最后一节“闭包是引用类型”中的例子，得到的一个闭包常量，这个闭包常量在内存里作为一个实例对象，此对象存储了捕获到的参数，调用三次就得到三个不同的结果:

```
typedef int(^xFuncTest) (int);

-(xFuncTest)funcX {
    int __block inner = 0;
    xFuncTest blockTest = ^(int mark) {
        return inner += mark;
    };
    return blockTest;
}

xFuncTest func = [self funcX];
NSLog(@"%d",func(1));
NSLog(@"%d",func(1));
NSLog(@"%d",func(1));
```

Yeah 10个月前 (05-13)