闭包

@M了个J

https://github.com/CoderMJLee

http://cnblogs.com/mjios

码拉松

小码哥教育 SEEMYGO

实力IT教育 www.520it.com

闭包表达式（Closure Expression）

- 在Swift中，可以通过func定义一个函数，也可以通过闭包表达式定义一个函数

```
func sum(_ v1: Int, _ v2: Int) -> Int { v1 + v2 }
```

```
var fn = {
    (v1: Int, v2: Int) -> Int in
    return v1 + v2
}
fn(10, 20)
```

```
{
    (v1: Int, v2: Int) -> Int in
    return v1 + v2
}(10, 20)
```

```
{
    (参数列表) -> 返回值类型 in
    函数体代码
}
```

小码哥教育 SEEMYGO

闭包表达式的简写

```
func exec(v1: Int, v2: Int, fn: (Int, Int) -> Int) {
    print(fn(v1, v2))
}
```

```
exec(v1: 10, v2: 20, fn: {
    (v1: Int, v2: Int) -> Int in
    return v1 + v2
})
```

```
exec(v1: 10, v2: 20, fn: {
    v1, v2 in return v1 + v2
})
```

```
exec(v1: 10, v2: 20, fn: {
    v1, v2 in v1 + v2
})
```

```
exec(v1: 10, v2: 20, fn: { $0 + $1 })
```

```
exec(v1: 10, v2: 20, fn: +)
```

小码哥教育 SEEMYGO

尾随闭包

- 如果将一个很长的闭包表达式作为函数的最后一个实参，使用尾随闭包可以增强函数的可读性
- 尾随闭包是一个被书写在函数调用括号外面（后面）的闭包表达式

```
func exec(v1: Int, v2: Int, fn: (Int, Int) -> Int) {
    print(fn(v1, v2))
}
```

```
exec(v1: 10, v2: 20) {
    $0 + $1
}
```

- 如果闭包表达式是函数的唯一实参，而且使用了尾随闭包的语法，那就不需要在函数名后边写圆括号

```
func exec(fn: (Int, Int) -> Int) {
    print(fn(1, 2))
}
```

```
exec(fn: { $0 + $1 })
exec() { $0 + $1 }
exec { $0 + $1 }
```

示例 – 数组的排序

```
func sort(by areInIncreasingOrder: (Element, Element) -> Bool)
```

```
/// 返回true: i1排在i2前面
/// 返回false: i1排在i2后面
func cmp(i1: Int, i2: Int) -> Bool {
    // 大的排在前面
    return i1 > i2
}
```

```
var nums = [11, 2, 18, 6, 5, 68, 45]
nums.sort(by: cmp)
// [68, 45, 18, 11, 6, 5, 2]
```

```
nums.sort(by: {
    (i1: Int, i2: Int) -> Bool in
    return i1 < i2
})
nums.sort(by: { i1, i2 in return i1 < i2 })
nums.sort(by: { i1, i2 in i1 < i2 })
nums.sort(by: { $0 < $1 })
nums.sort(by: <)
nums.sort() { $0 < $1 }
nums.sort { $0 < $1 }
// [2, 5, 6, 11, 18, 45, 68]
```

小码哥教育 SEEMYGO

忽略参数

```
func exec(fn: (Int, Int) -> Int) {
    print(fn(1, 2))
}
exec { _,_ in 10 } // 10
```

小码哥教育@M了个J

闭包（Closure）

- 网上有各种关于闭包的定义，个人觉得比较严谨的定义是
  - 一个函数和它所捕获的变量\常量环境组合起来，称为闭包
  - ✓ 一般指定义在函数内部的函数
  - ✓ 一般它捕获的是外层函数的局部变量\常量

```
typealias Fn = (Int) -> Int
func getFn() -> Fn {
    var num = 0
    func plus(_ i: Int) -> Int {
        num += i
        return num
    }
    return plus
} // 返回的plus和num形成了闭包
```

```
var fn1 = getFn()
var fn2 = getFn()
fn1(1) // 1
fn2(2) // 2
fn1(3) // 4
fn2(4) // 6
fn1(5) // 9
fn2(6) // 12
```

```
func getFn() -> Fn {
    var num = 0
    return {
        num += $0
        return num
    }
}
```

- 思考：如果num是全局变量呢？

- 可以把闭包想象成是一个类的实例对象
  - 内存在堆空间
  - 捕获的局部变量\常量就是对象的成员（存储属性）
  - 组成闭包的函数就是类内部定义的方法

```
class Closure {
    var num = 0
    func plus(_ i: Int) -> Int {
        num += i
        return num
    }
}
var cs1 = Closure()
var cs2 = Closure()
cs1.plus(1) // 1
cs2.plus(2) // 2
cs1.plus(3) // 4
cs2.plus(4) // 6
cs1.plus(5) // 9
cs2.plus(6) // 12
```

小码哥教育 SEEMYGO

练习

```
typealias Fn = (Int) -> (Int, Int)
func getFns() -> (Fn, Fn) {
    var num1 = 0
    var num2 = 0
    func plus(_ i: Int) -> (Int, Int) {
        num1 += i
        num2 += i << 1
        return (num1, num2)
    }
    func minus(_ i: Int) -> (Int, Int) {
        num1 -= i
        num2 -= i << 1
        return (num1, num2)
    }
    return (plus, minus)
}
```

```
let (p, m) = getFns()
p(5) // (5, 10)
m(4) // (1, 2)
p(3) // (4, 8)
m(2) // (2, 4)
```

```
class Closure {
    var num1 = 0
    var num2 = 0
    func plus(_ i: Int) -> (Int, Int) {
        num1 += i
        num2 += i << 1
        return (num1, num2)
    }
    func minus(_ i: Int) -> (Int, Int) {
        num1 -= i
        num2 -= i << 1
        return (num1, num2)
    }
}
```

```
var cs = Closure()
cs.plus(5) // (5, 10)
cs.minus(4) // (1, 2)
cs.plus(3) // (4, 8)
cs.minus(2) // (2, 4)
```

小码哥教育 SEEMYGO

练习

```swift
var functions: [() -> Int] = []
for i in 1...3 {
    functions.append { i }
}
for f in functions {
    print(f())
}
```

```
// 1
// 2
// 3
```

```swift
class Closure {
    var i: Int
    init(_ i: Int) {
        self.i = i
    }
    func get() -> Int {
        return i
    }
}
var clses: [Closure] = []
for i in 1...3 {
    clses.append(Closure(i))
}
for cls in clses {
    print(cls.get())
}
```

小码哥教育 SEEMYGO

注意

- 如果返回值是函数类型，那么参数的修饰要保持统一

```
func add(_ num: Int) -> (inout Int) -> Void {
    func plus(v: inout Int) {
        v += num
    }
    return plus
}
var num = 5
add(20)(&num)
print(num)
```

自动闭包

```
// 如果第1个数大于0，返回第一个数。否则返回第2个数
func getFirstPositive(_ v1: Int, _ v2: Int) -> Int {
    return v1 > 0 ? v1 : v2
}
getFirstPositive(10, 20) // 10
getFirstPositive(-2, 20) // 20
getFirstPositive(0, -4) // -4
```

- @autoclosure 会自动将 20 封装成闭包 { 20 }
- @autoclosure 只支持 () -> T 格式的参数
- @autoclosure 并非只支持最后1个参数
- 空合并运算符 ?? 使用了 @autoclosure 技术
- 有@autoclosure、无@autoclosure，构成了函数重载

```
// 改成函数类型的参数，可以让v2延迟加载
func getFirstPositive(_ v1: Int, _ v2: () -> Int) -> Int? {
    return v1 > 0 ? v1 : v2()
}
getFirstPositive(-4) { 20 }
```

```
func getFirstPositive(_ v1: Int, _ v2: @autoclosure () -> Int) -> Int? {
    return v1 > 0 ? v1 : v2()
}
getFirstPositive(-4, 20)
```

- 为了避免与期望冲突，使用了@autoclosure的地方最好明确注释清楚：这个值会被推迟执行