Swift学习笔记

丁碧云

2016-12-18

Swift简介

- 什么是Swift
- Swift是苹果公司在2014年WWDC（苹果开发者大会）上发布的全新编程语言
- Swift在天朝译为“雨燕”，右上角的图标就是它的LOGO
- 跟Objective-C一样，可以用于开发iOS、Mac应用程序
- 苹果公司从2010年7月开始设计Swift语言，耗时4年打造

相关人物

- Swift的首席架构师：Chris Lattner
- LLVM 项目的主要发起人与作者之一
- Clang 编译器的作者
- 苹果公司『开发工具』部门的主管
- 领导Xcode、Instruments和编译器团队
- 2010年7月开始主导开发 Swift 编程语言

相关数据

- 使用Swift语言进行开发的条件是什么？
  - Xcode版本 >= 6.0
  - Mac系统版本 >= 10.9.3

- Swift自从发布之后，备受开发者关注，发布当天（1天的时间内）
  - Xcode 6 beta下载量突破1400万次
  - 官方发布的电子书《The Swift Programming Language》下载量突破37万次
  - 一位国外开发者已经用Swift实现了Flappy Bird游戏（这位开发者上手Swift的时间只有4个小时，编程加上休息时间，接近9个小时）

性能

- 根据WWDC的展示
- 在进行复杂对象排序时
- ✓ Objective-C的性能是Python的2.8倍，Swift的性能是Python的3.9倍
- 在实现 RC4加密算法时
- ✓ Objective-C的的性能是Python的127倍，Swift的性能是Python的220倍

- 有持怀疑态度的国外程序员，也对Objective-C和Swift的性能进行了测试
- http://www.splasmata.com/?p=2798

语法须知

- Swift的源文件拓展名是.swift

- 2个不需要
  - 不需要编写main函数
    - ✓ 从上往下按顺序执行，所以最前面的代码会被自动当做程序的入口
  - 不需要在每一条语句后面加上分号

```
let radius = 10
```

- ✓ 你喜欢的话，也可以加上

```
let radius = 10;
```

- ✓ 有一种情况必须加分号：同一行代码上有多条语句时

```
let radius = 10; let radius2 = 15
```

数据类型

- Swift中常用的数据类型有
  - Int、Float、Double、Bool、Character、String
  - Array、Dictionary、元组类型（Tuple）、可选类型（Optional）
  - 可以看出，数据类型的首字母都是大写的

- 如何指定变量\常量的数据类型
  - ✓ 在常量\变量名后面加上 冒号（:） 和 类型名称

```
let age: Int = 10
```

  - ✓ 上面代码表示：定义了一个Int类型的常量age，初始值是10

- 一般来说，没有必要明确指定变量\常量的类型
  - 如果在声明常量\变量时赋了初始值，Swift可以自动推断出这个常量\变量的类型

```
let age = 20
// Swift会推断出age是Int类型，因为20是个整数
```

整型

- 整数分为2种类型
  - 有符号（signed）：正、负、零
  - 无符号（unsigned）：正、零

- Swift提供了8、16、32、64位的有符号和无符号整数，比如
  - UInt8 ：8位无符号整型
  - Int32 ：32位有符号整型
  - Int8、Int16、Int32、Int64、UInt8、UInt16、UInt32、UInt64

- 最值
  - 可以通过min和max属性来获取某个类型的最小值和最大值

```
1. let minValue = UInt8.min
2. let maxValue = UInt8.max
```

整数的表示形式

- 整数的4种表示形式
  - 十进制数：没有前缀

```
let i1 = 10 // 10
```

  - 二进制数：以0b为前缀

```
let i2 = 0b1010 // 10
```

  - 八进制数：以0o为前缀

```
let i3 = 0o12 // 10
```

  - 十六进制数：以0x为前缀

```
let i4 = 0xA // 10
```

Int和UInt

- Swift还提供了特殊的有符号整数类型Int和无符号整数类型UInt
  - Int\UInt的长度和当前系统平台一样
    - ✓ 在32位系统平台上，Int和UInt的长度是32位
    - ✓ 在64位系统平台上，Int和UInt的长度是64位
  - Int在32位系统平台的取值范围：-2147483648 ~ 2147483647
- 建议
  - 在定义变量时，别总是在考虑有无符号、数据长度的问题
  - 尽量使用Int，这样可以保证代码的简洁、可复用性

存储范围

- 每种数据类型都有各自的存储范围，比如
  - Int8的存储范围是：-128 ~ 127
  - UInt8的存储范围是：0 ~ 255

- 如果数值超过了存储范围，编译器会直接报错
  - 下面的语句都会直接报错

```
1. let num1: UInt8 = -1
2. let num2: Int8 = Int8.max + 1
```

✓ 第1行代码报错原因：UInt8不能存储负数

✓ 第2行代码报错原因：Int8能存储的最大值是Int8.max

数字格式

- 数字可以增加额外的格式，使它们更容易阅读
  - 可以增加额外的零 0

```
let money = 001999 // 1999
let money2 = 001999.000 // 1999.0
```

  - 可以增加额外的下划线 _，以增强可读性

```
let oneMillion1 = 1_000_000 // 1000000
let oneMillion2 = 100_0000 // 1000000
let overOneMillion = 1_000_000.000_001 // 1000000.000001
```

  - 增加了额外的零 0和下划线 _，并不会影响原来的数值大小

浮点数

- 浮点数，就是小数。Swift提供了两种浮点数类型
  - Double ：64位浮点数，当浮点值非常大或需要非常精确时使用此类型
  - Float ：32位浮点数，当浮点值不需要使用Double的时候使用此类型

- 精确程度
  - Double ：至少15位小数
  - Float ：至少6位小数

- 如果没有明确说明类型，浮点数默认就是Double类型

```
let num = 0.14 // num是Double类型的常量
```

浮点数的表示形式

- 浮点数可以用 十进制 和 十六进制 2种进制来表示
  - 十进制（没有前缀）
    - ✓ 没有指数：`let d1 = 12.5`
    - ✓ 有指数 ：`let d2 = 0.125e2`

```
// 0.125e2 == 0.125 * 10²
```

- ✧ MeN == M * 10的N次方

  - 十六进制（以0x为前缀，且一定要有指数）

```
let d3 = 0xC.8p0
// 0xC.8p0 == 0xC.8 * 2º == 12.5 * 1
```

- ✧ 0xMpN == 0xM * 2的N次方

```
let d3 = 0xC.8p1
```

字符串

- 字符串是String类型的数据，用双引号""包住文字内容

```
let website = "http://ios.itcast.cn"
```

- 字符串的常见操作
  - 用加号 + 做字符串拼接

```
let scheme = "http://"
let path = "ios.itcast.cn"
let website = scheme + path
// website的内容是"http://ios.itcast.cn"
```

  - 用反斜线 \ 和 小括号 () 做字符串插值（把常量\变量插入到字符串中）

```
let hand = 2
var age = 20
let str = "我今年\(age)岁了，有\(hand)只手"
```

字符串中解析变量hand，两种方法：
（1）var str =“I have \(hand) hands.”
（2）var str =“I have “+hand+”hands.”

变量转字符串：

```
var str = String(age)
var str2 = "\(age)"
```

swift使用Character类型代表单个字符。
字符必须使用双引号包起来。

类型转换与类型别名

类型转换

- 下面的语句是错误的

```
1. let num1 = 3 // num1是Int类型
2. let num2 = 0.14 // num2是Double类型
3. let sum = num1 + num2
```

- ✓ 第3行会报错
- ✓ 报错原因：num1是Int类型，num2是类型Double，类型不同，不能相加
- ✓ 解决方案：将num1转为Double类型，就能与num2进行相加

- 下面的语句是正确的

```
let sum = Double(num1) + num2
```

- 注意
- 下面的写法是正确的

```
let sum = 3 + 0.14
```

类型别名

- 可以使用typealias关键字定义类型的别名，跟C语言的typedef作用类似

```
typealias MyInt = Int
// 给Int类型起了个别名叫做MyInt
```

- 原类型名称能用在什么地方，别名就能用在什么地方
- 声明变量\常量类型

```
let num: MyInt = 20
```

- 获得类型的最值

```
let minValue = MyInt.min
```

- 类型转换

```
let num = MyInt(3.14) // 3
```

第一个swift程序

```swift
//
//  ViewController.swift
//  07-第一个UI程序
//
//  Created by apple on 14-7-29.
//  Copyright (c) 2014年 heima. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        var btn = UIButton()
        btn.frame = CGRectMake(100, 100, 100, 100)
        btn.backgroundColor = UIColor.redColor()
        self.view.addSubview(btn)

        var imageView = UIImageView()
        imageView.image = UIImage(named: "002")
        imageView.frame = CGRectMake(100, 250, 50, 50)
        self.view
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

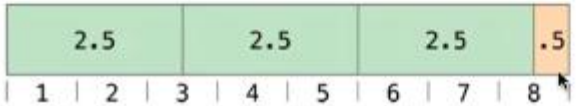
运算符

求余运算符

- %在Swift中叫“求余运算符”，也有语言叫做“模运算符”
  - `9 % 4 // 1`
  - `-9 % 4 // -1`
  - `9 % -4 // 1`
  - `-9 % -4 // -1`

求余结果的正负跟%左边数值的正负一样

- 跟C语言不一样的是，Swift的%支持浮点数的计算
  - `8 % 2.5  // 0.5`



范围运算符

- 范围运算符用来表示一个范围，有2种类型的范围运算符
  - 闭合范围运算符：`a...b`，表示`[a, b]`，包含a和b
  - 半闭合范围运算符：`a..<b`，表示`[a, b)`，包含a，不包含b

- 举例

```
1. for index in 1...5 {
2.     println(index)
3. }
```

溢出运算

溢出运算符

- 每种数据类型都有自己的取值范围，默认情况下，一旦赋了一个超出取值范围的数值，会产生编译或者运行时错误
  - 下面的写法是错误的

```
1. let x = UInt8.max
2. let y = x + 1
```

  - ✓ 第2行代码会在运行时报错

- Swfit为整型计算提供了5个&开头的溢出运算符，能对超出取值范围的数值进行灵活处理
  - 溢出加法 &+
  - 溢出减法 &-
  - 溢出乘法 &*
  - 溢出除法 &/

值的上溢出

- 废话不多说，直接上代码

```
1. let x = UInt8.max
2. let y = x &+ 1
```

  - 第1行代码过后：x的值是 255（最大值）
  - 第2行代码过后：y的值是 0（最小值）

值的下溢出

- 废话不多说，直接上代码

```
1. let x = UInt8.min
2. let y = x &- 1
```

  - 第1行代码过后：x的值是 0（最小值）

Bool类型

Bool

- Bool类型，也被称为逻辑类型（Logical），就2种取值
  - true：真
  - false：假

- 在C语言中：0是假，非0就是真；而在Swift中没有这种概念

- if语句的条件必须是Bool类型的值
  - 错误写法

```
1. if (10) {
2.     println("条件成立")
3. }
```

  - 正确写法

```
1. if (true) {
2.     println("条件成立")
3. }
```

比较运算符\逻辑运算符\三目运算符

- 比较运算符\逻辑运算符会返回Bool类型的值，取值有2种可能
  - true：真，6 > 5，(7 > 6) && (9 != 7)
  - false：假，6 <5，(7 >= 6) && (9 == 7)

- 三目运算符的条件必须是Bool类型的值
  - 错误写法

```
1. let a = 10
2. let c = a ? 100 : 200
```

  - 正确写法

```
1. let c = a != 0 ? 100 : 200
2. let c = false ? 100 : 200
```

元组

元组类型

- 什么是元组类型
  - 元组类型由 N个 任意类型的数据组成（N >= 0）
  - 组成元组类型的数据可以称为“元素”

```
✓ let position = (x : 10.5, y : 20)
// position有2个元素，x、y是元素的名称

✓ let person = (name : "jack")
// person只有name一个元素

✓ let data = ()
// 空的元组
```

元素的访问

```
var position = (x : 10.5, y : 20)
```

- 用元素名称

```
1. let value = position.x // 取值
2. position.y = 50  // 设值
```

- 用元素位置

```
1. var value = position.0  // 相当于var value = postion.x
2. position.1 = 50 // 相当于postion.y = 50
```

- 注意
  - 如果用let来定义一个元组，那么就是常量，就无法修改它的元素

可选类型

在任何已有类型的后面紧跟？，即可代表可选类型，可选类型的变量可用于处理处理“值缺失”的情况。

对于可能发生“值缺失”的情况，编译器会自动推断该变量的类型为可选类型。

swift使用nil代表“值缺失”。

注意：swift的nil与OC中的nil完全不同，在OC中，nil代表哟个并不存在的对象指针，而在swift中nil并不代表指针，他是一个确定的值，用于代表“值缺失”。任何可选类型的变量度可以被赋值为nil，如Int?, Double?等度可以接受nil值。

需支出的是，只有可选类型的变量或常量才能接受nil，非可续那类型的变量和常量不能接受nil。

强制解析

为了获取可选类型的变量或者常量时机村处置，可在变量或者常量后添加英文感叹号（!）进行解析，这个感叹号表面表明：已知该可选变量有值，请提取其中的值。

这种感叹号进行解析的用法，被称为强制解析。

如： var s : String = str!

隐式可选类型

出了在任意已有类型后面添加?来表示可选类型之二爱，swift还允许在任意已有类型后面添加!来表示可选类型。

比较：

Int类型的变量或者常量，只能接受Int类型的值，不能接受nil值；

Int?类型的变量或者常量，既能接受Int类型的值，也能接受nil值；

Int!类型的变量或者常量，既能接受Int类型的值，也能接受nil值；

Int?与Int!的区别是：当程序需要获得Int?类型的变量和常量，程序必须在变量名后添加!后缀执行强制解析，但当程序需要获取Int!类型的变量或者常量的值时，无须在变量后添加!后缀执行强制解析，swift会自动执行隐式解析。

注意：隐式可选类型主要用于定义那些被复制之后不会重新变为nil的变量。

元组使用细节

- 可以省略元素名称

```
1. let position = (10, 20)
2. let person = (20, "jack")
```

- 可以明确指定元素的类型

```
var person: (Int, String) = (23, "rose")
```

- person的第0个元素只能是Int类型、第1个元素只能是String类型

- 注意
  - ✓ 在明确指定元素类型的情况下不能加上元素名称
  - ✓ 因此，下面的语句是错误的

```
var person: (Int, String) = (age : 23, name : "rose")
```

（1）可以用position.0和position.1来输出元组的元素；
（2）对于命名的元组，如：
let position= (x:1, y:2)
可用position.x和position.y来表示
（3）要输出所有的元组：
直接输出position

使用细节

- 可以用多个变量接收元组数据

```
1. var (x , y) = (10, 20) // x是10，y是20
2. var point = (x, y) // point由2个元素组成，分别是10和2(
```

- 可以将元素分别赋值给多个变量

```
1. var point = (10, 20)
2. var (x , y) = point
// x是10，y是20
```

- 可以使用下划线 _ 忽略某个元素的值，取出其他元素的值

```
1. var person = (20, "jack")
2. var (_, name) = person
```

流程结构

- Swift支持的流程结构
  - 循环结构：for、for-in、while、do-while
  - 选择结构：if、switch

注意：这些语句后面一定要跟上大括号{}

- 跟C语言对比
  - 用法基本一样的有：for、while、do-while、if
  - 因此只需要关注for-in和switch即可

for-in

- for-in和范围运算符

```
1. for i in 1...3 {
2.     println(i)
3. }
// 按顺序从范围中取值赋值给i，每取1次值，就执行1次循环体
// 范围的长度就是循环体执行的次数
```

- 如果不需要用到范围中的值，可以使用下划线 _ 进行忽略

```
1. for _ in 1...3 {
2.     println("*********")
3. }
```

switch

```
1. let grade = "B"
2. switch grade {
3. case "A":
4.     println("优秀等级")
5. case "B":
6.     println("良好等级")
7. case "C":
8.     println("中等等级")
9. default:
10.    println("未知等级")
11.}
```

- switch语句在 Swift 和 C 中的区别
  - 在C语言中
    - ✓ 如果case的结尾没有break，就会接着执行下一个case
  - 在Swift中
    - ✓ 不需要在每一个case后面增加break，执行完case对应的代码后默认会自动退出switch语句
- 在Swift中，每一个case后面必须有可以执行的语句

case

case的多条件匹配

- 1个case后面可以填写多个匹配条件，条件之间用逗号,隔开

```
let score = 95
switch score/10 {
case 10, 9:
    println("优秀")
case 8, 7, 6:
    println("及格")
default:
    println("不及格")
}
// 打印结果是：优秀
```

case的范围匹配

- case后面可以填写一个范围作为匹配条件

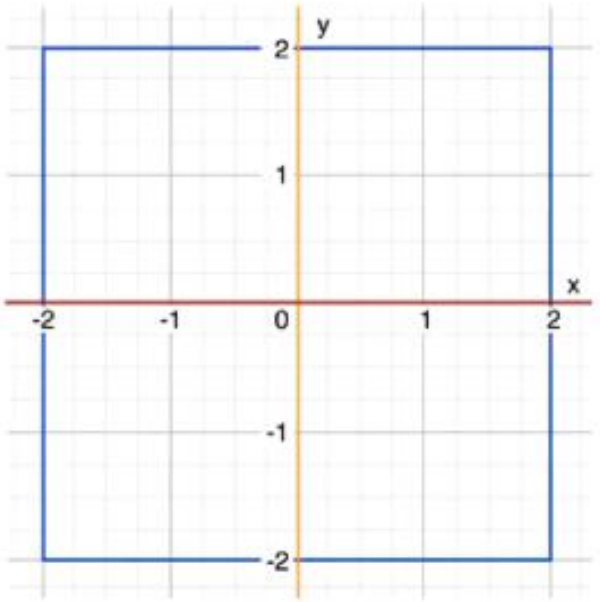
```
let score = 95
switch score {
case 90...100:
    println("优秀")
case 60...89:
    println("及格")
default:
    println("不及格")
}
// 打印结果是：优秀
```

- 注意：
  - switch要保证处理所有可能的情况，不然编译器直接报错
  - 因此，这里的default一定要加，不然就出现了一些处理不到的情况

case匹配元组

- case还可以用来匹配元组

```
// 比如判断一个点是否在右图的蓝色矩形框内
1. let point = (1, 1)
2. switch point {
3. case (0, 0) :
4.     println("这个点在原点上")
5. case (_, 0) :
6.     println("这个点在x轴上")
7. case (0, _) :
8.     println("这个点在y轴上")
9. case (-2...2, -2...2) :
```



- 第8行中 _ 的作用（2种理解方式）
  - 能匹配任何值
  - 忽略对应位置元组元素

Case的数值绑定

- 在case匹配的同时，可以将switch中的值绑定给一个特定的常量或者变量，以便在case后面的语句中使用

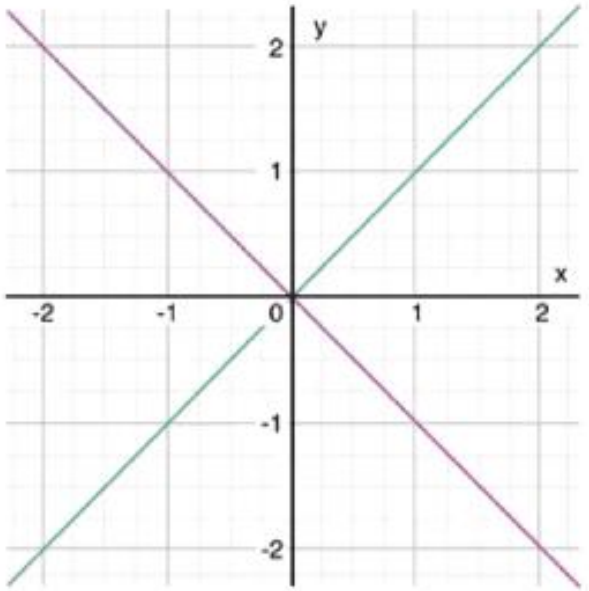
```
1. let point = (10, 0)
2. switch point {
3. case (let x, 0) :
4.     println("这个点在x轴上, x值是\(x)")
5. case (0, let y) :
6.     println("这个点在y轴上, y值是\(y)")
7. case let (x, y) :
8.     println("这个点的x值是\(x), y值是\(y)")
9. }

// 打印: 这个点在x轴上, x值是10
```

case使用where

- switch语句可以使用where来增加判断的条件

```
// 比如判断一个点是否在右图的绿线或者紫线上
1. var point = (10, -10)
2. switch point {
3. case let (x, y) where x == y :
4.     println("这个点在绿线上")
5. case let (x, y) where x == -y :
6.     println("这个点在紫线上")
7. default :
8.     println("这个点不在这2条线上")
9. }
// 打印: 这个点在紫线上
```



fallthrough（贯穿）

- fallthrough的作用
  - 执行完当前case后，会接着执行fallthrough后面的case或者default语句

```
1. let num = 20
2. var str = "\(num)是个"
3. switch num {
4. case 0...50 :
5.     str += "0~50之间的"
6.     fallthrough
7. default :
8.     str += "整数"
9. }
```

- 注意
  - fallthrough后面的case条件不能定义变量和常量

标签

- 使用标签的其中1个作用：可以用于明确指定要退出哪个循环

```
// 做2组俯卧撑，每组3个，做完一组就休息一会
group:
for _ in 1...2 {
    for item in 1...3 {
        println("做1个俯卧撑")
        if item == 2 {
            break group
        }
    }
    println("休息一会")
}
```

- 输出结果是

```
// 做1个俯卧撑
// 做1个俯卧撑
```

函数

- 函数的定义格式

```
1. func 函数名(形参列表) -> 返回值类型 {
2.     // 函数体...
3. }
```

- 形参列表的格式
  - 形参名1: 形参类型1, 形参名2: 形参类型2, …

- 举例：计算2个整数的和

```
func sum(num1: Int, num2: Int) -> Int {
    return num1 + num2
}
```

- 如果函数没有返回值，有3种写法

```
1. func 函数名(形参列表) -> Void {
2.     // 函数体...
3. }

4. func 函数名(形参列表) -> () {
5.     // 函数体...
6. }

7. func 函数名(形参列表) {
8.  // 函数体...
9. }
```

返回元组数据

- 一个函数也可以返回元组数据

```
1. func find(id: Int) -> (name: String, age: Int) {
2.     if id > 0 {
3.         return ("jack", 20)
4.     } else {
5.         return ("nobody", 0)
6.     }
7. }
8. var people = find(2)
9. println("name=\(people.name), age=\(people.age)")
```

```
// 返回元组类型
func getPoint() -> (Int, Int) {
    return (10, 20)
}
```

外部参数名

- 一般情况下，通过形式参数的名字，就能推断出这个参数的含义和作用

```
1. func addStudent(name: String, age: Int, no: Int) {
2.     println("添加学生: name=\(name), age=\(age), no=\(no)")
3. }
```

  - 在函数内部一看参数名就知道这3个参数的作用

- 但是，形式参数是用在函数内部的，当调用函数时就看不见形参的名字，有可能导致以后会不太明白每个参数的含义

```
addStudent("jack", 20, 19)
```

  - 一眼望去，能猜出第1个参数"jack"是指姓名，后面的20、19分别代表什么含义?

- 为了解决上述问题，Swift提供了外部参数名语法
  - 外部参数名可以在调用函数时提醒每个参数的含义

外部参数名

- 外部参数名的定义格式

```
1. func 函数名(外部参数名 形式参数名: 形式参数类型) -> 返回值类型 {
2.     // 函数体...
3. }
```

- 外部参数名写在形式参数名的前面，与形式参数名之间用空格隔开

```
1. func sum(number1 num1: Int, number2 num2: Int) -> Int
2. {
3.     return num1 + num2
4. }
5. sum(number1: 10, number2: 20) // 调用函数
```

- 注意
  - 一旦定义了外部参数名，在调用函数时就必须加上外部参数名

外部参数名的简写

- 使用#能简化外部参数名的定义

```
1. func sum(#num1 : Int, #num2 : Int)
2. {
3.     return num1 + num2
4. }
```

- 第1行代码的意思：num1、num2既是形式参数名，又是外部参数名

```
// 调用函数
sum(num1: 10, num2: 20)
```

默认参数值

- 可以在定义函数时，给形参指定一个默认值，调用函数时，就可以不用给这个形参传值

```
1. func addStudent(name: String, age: Int = 20) {
2.     println("添加1个学生：name=\(name)，age=\(age)")
3. }
4. addStudent("jack")
```

- age参数有个默认值20，所以第4行调用函数时可以不传值给age参数
- 输出结果是

添加1个学生：name=jack，age=20

常量与变量参数

常量和变量参数

- 默认情况下，函数的参数都是常量参数，不能在函数内部修改

```
1. func test(num: Int) {
2.     num = 10
3. }
```

- func test(num: Int)参数相当于func test(let num: Int)
- 第2行代码会报错

- 在有些情况下，可能需要在函数内部修改参数的值，这时需要定义变量参数
- 在参数名前面加个var即可

```
1. func test(var num : Int) {
2.     num = 10
3. }
```

常量和变量参数

- 废话不多说，直接上代码

```
// 编写函数在某个字符串的尾部拼接N个其他字符串
1. func append(var string: String, suffix: String, count:
   Int) -> String
2. {
3.     for _ in 0..<count {
4.         string += suffix
5.     }
6.     return string
7. }
8. append("jack", ".", 4) // 调用函数
```

输入输出参数

- 什么是输入输出参数
  - 在C语言中，利用指针可以在函数内部修改外部变量的值
  - 在Swift中，利用输入输出参数，也可以在函数内部修改外部变量的值
  - 输入输出参数，顾名思义，能输入一个值进来，也可以输出一个值到外

- 输入输出参数的定义
  - 在参数名前面加个inout关键字即可

```
1. func swap(inout num1: Int, inout num2: Int) {
2. }
```

```
func change(inout num: Int) {
    num = 10
}

var a = 20
change(&a)
```

输入输出参数

- 废话不多说，直接上代码
  - 写一个函数交换外部2个变量的值

```
1. func swap(inout num1: Int, inout num2: Int) {
2.     let tempNum1 = num1
3.     num1 = num2
4.     num2 = tempNum1
5. }
6. var a = 20
7. var b = 10
8. swap(&a, &b) // 传入的参数前面必须加上&
// 执行完swap函数后，a的值是10，b的值是20
```

- 面试题
  - 如何在不利用第3方变量的前提下，交换2个变量的值

输入输出参数的使用注意

- 输入输出参数的使用注意
  - 传递实参时，必须在实参的前面加上&
  - 不能传入常量或者字面量（比如10）作为参数值（因为它们都不可改）
  - 输入输出参数不能有默认值
  - 输入输出参数不能是可变参数
  - 输入输出参数不能再使用let、var修饰（inout和let、var不能共存）

- 输入输出参数的价值之一
  - 可以实现函数的多返回值（其实让函数返回元组类型，也能实现返回多个值）

不使用变量交换数据

面试题：

```
func swap(inout num1: Int, inout num2: Int) {
    num1 = num1 + num2
    num2 = num1 - num2
    num1 = num1 - num2
}
```

```
func swap(inout num1: Int, inout num2: Int) {
    num1 = num1 ^ num2
    num2 = num1 ^ num2
    num1 = num1 ^ num2
}
```

输出变量和常量

swift为输出变量和常量提供了print()和println()两个全局函数，这两个函数的功能大致相同，区别只是println()会在输出内容后换行，而print()不会。

print()和println()不仅可以输出简单的变量，也可以输出更复杂的信息，这些信息可以在字符串中“嵌套”变量或者常量的值。

swift使用字符串差值的方式把变量名或者常量名作为占字符插入字符串中，swift会使用该变量或者常量的值替换这些占字符。插值语法格式为：将变量名或者长两名放在圆括号中，并在圆括号前使用反斜杠进行转移，即使用“\(变量)”的形式。

闭包

闭包表达式：

```
{ (形参列表) -> 返回值类型 in
    // 零条或者多条可执行语句
}
```

闭包与嵌套函数的区别：

1）定义闭包无须func关键字，无须指定函数名

2）定义闭包需要额外使用in关键字

3）定义闭包的第一个花括号要移到形参列表的圆括号之前

注：（1）使用闭包表达式定义闭包时不需要执行外部形参变量，因为没有任何作用；

（2）swift可以根据闭包表达式上下文推断形参类型、返回值类型，那么闭包表达式就就可以省略形参类型、返回值类型。

（3）如果闭包表达式的执行体只有一行代码，而且这行代码的返回值将作为闭包表达式的返回值，那么swift运行省略return关键字。