

# swift3.0 后基础语法 / 变化 一

作者: [wyw89500](#)

原文链接: <https://ld246.com/article/1481187515746>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

之前学习了swift,但是感觉语法变化太大,经常很多写法,现在swift已经相当稳定,但是看了下发现之前的代码一片红,才发现很多地方变了,所以准备最近再看下,主要是要用起来

## 方便起见直接贴代码

```
// 1.导入框架
import UIKit

// 2.在swift中如果定义一个标识符.必须告诉编译器该标识符是一个变量还是一个常量
// 如果是常量使用let修饰
// 如果是变量使用var修饰
// let 标识符的名称 : 类型 = 赋值 //可以省略类型,有类型推导
// var 标识符的名称 : 类型 = 赋值

// 使用注意:
// 1.一般优先使用let,只有当是需要修改时候才使用var
// 2.常量的本质:指向的内存地址不可以被修改,但是可以通过内存地址获得对象后,之后修改对象
// 内部的属性
// 3.隐式转化:在Int类型和Double进行基本运算时,会自动将Int类型转成Double

// let a : Int = 10;//如果一行只有一条语句那么后面的;可以省略
// a = 10;//报错,因为let是常量不可以被修改
// let view : UIView;// = [UIView alloc] init];
// var b :Double = 1.44;
// b = 20.44;//b是变量可以被修改
// let a = 10.0;
// let result = Int(a);
// print(result)

// let view : UIView = UIView();
// view.backgroundColor = UIColor.red;
// view.frame = CGRect(x: 10, y: 10, width: 100, height: 100);
// self.view.addSubview(view);

// let view1 : UIView = UIView(frame: CGRect(x: 20, y: 20, width: 100, height: 100));
// view1.backgroundColor = UIColor.blue;
// self.view.addSubview(view1);

// let btn : UIButton = UIButton(frame: CGRect(x: 0, y: 0, width: 20, height: 20));
// btn.backgroundColor = UIColor.green;
// //btn方法调用
// btn.addTarget(self, action: #selector(ViewController.youyouyou), for: UIControlEvents.touchUpInside)
// btn.addTarget(self, action: #selector(ViewController.youyouyou), for: .touchUpInside)
// 或者这样
// view1.addSubview(btn);

// 3.打印
// print("hahah")

// 4.swift和OC的区别
// 1.swift中if后面的()可以省略
// 2.没有非0即真,必须有明确的Bool -> true/false
```

```

//      let a = 10;
//      if a > 0 {
//          print("a大于0")
//      } else {
//          print("a不大于0")
//      }
//
//      let score = 66;
//      if score < 0 || score > 100 {
//          print("不合理的分数")
//      } else if score < 60 {
//          print("不及格")
//      } else if score < 80 {
//          print("及格")
//      } else if score < 90 {
//          print("良好")
//      } else {
//          print("优秀")
//      }

//      5.三目运算符
//      let m = 10;
//      let n = 20;
//      let result = m > n ? m : n;
//      print(result); //n 20

//      6.guard的使用
//      1.如果条件表达式为真,则会执行后面的语句
//      2.如果条件表达式为假,则会执行else后面的{}中的语句

//func getScore( a : Double ) -> Void {
//
//    guard a > 60 else {
//        print("不及格")
//        return
//    }
//    guard a > 80 else {
//        print("不错,及格了")
//        return
//    }
//    guard a > 100 else {
//        print("优秀")
//        return
//    }
//    print("chao神")
//}

//      7.swift中的switch和OC的区别
//      1.switch后面的()可以省略
//      2.case结束后可以不加break,系统会默认跟上break
//      3.case后面跟多个条件,并且多个条件以,进行分割
//      4.如果case中希望产生case穿透,可以使用fallthrough
//      5.switch支持浮点型

```

```

//      6.switch支持字符串类型
//      7.switch支持区间类型

//func getSex(sex:Int) -> Void {
//  switch sex {//sex可以是整型,浮点型,字符串
//    case 0:
//      print("女")
//    case 1,2,3:
//      print("男")
//    case 4...8:
//      print("么")
//    fallthrough//穿透
//    default:
//      print("不知道")
//  }
//}

//swift中方法
//func youyouyou() -> Void{
//  print("hahah")
//}

//      8.循环for
//      需求:打印0-9

//      1.写法一
//for i in 0..<10{
//  print(i)
//}

//      2.写法二
//      如果for循环中不需要使用到下标i,可以用_代替
//for _ in 0..<10 {
//  print("aoliu")
//}

//      9.循环while do while
//      注意:
//        1.没有非零即真,必须有一个明确的bool值
//        2.while后面的()可以省略
//        3.没有a--
//        4.do while 不是do ,do用到了异常捕获 这里用repeat代替
//var a = 10
//
//while a > 0 {
//  a -= 1
//  print(a)
//}

//var m = 10
//repeat {
//  m -= 1
//  print(m)
//}while m > 0

```

```
// 10.字符串
// 注意:
// - OC和Swift中字符串的区别
//   1.在OC中字符串类型是NSString,在Swift中字符串类型是String
//   2.OC中字符串@ "",Swift中字符串 ""
//
// 二.使用 String 的原因
// String 是一个结构体, 性能更高
// NSString 是一个 OC 对象, 性能略差
// String 支持直接遍历
// Swift 提供了 String 和 NSString 之间的无缝转换

//let str = "hello world"
//var string = "hello world"
//string = "aoliu"

//1.获取字符串长度
//let length = string.characters.count

//2.遍历字符串中所有的字符
//for c in string.characters {
//  print(c)
//}

//3.字符串的拼接
//let a = "ao"
//let b = "liu"
//let result = a + b

//4.字符串和其他标识符的拼接
//let age = 18
//let name = "aoliu"
//let height = 1.78
//let info = "my name is \(name) ,\(age) years old, height is \(height)"

//5.字符串的格式化
//let time = 90
//let min = time / 60
//let second = time % 60
//let timeStr = String(format: "%02d:%02d", min,second)

//6.字符串截取:将String转变为NSString再截取
//let urlString = "www.baidu.com"
//let header = (urlString as NSString).substring(to: 3)
//let footer = (urlString as NSString).substring(from: 10)
//let range = NSMakeRange(4, 5)
//let middle = (urlString as NSString).substring(with: range)

// 11.数组
// 数组 (Array) 是一串有序的由相同类型元素构成的集合
// 数组中的集合元素是有序的, 可以重复出现
// Swift中的数组
// swift数组类型是Array, 是一个泛型集合
```

```

//1.定义数组
//let array : [String] = ["aoliu","xiaowang"]//不可变 [String]泛型
//var array1 = [String]()//可变数组

//2.CRUD
//array1.append("xiaomin")
//array1.append("xiaohong")
//array1.append("xiaoli")

//array1.remove(at: 1)//返回值是删除的元素
//array1.remove(at: 0)//返回值是删除的元素
//
//array1[0] = "ao"
//
//let name = array1[0]

//3.遍历数组
//for i in 0..<array1.count {
//    print(i)
//    print(array1[i])
//}
//
//for name in array1 {//常见
//    print(name)
//}
//
//for (index,name) in array1.enumerated() {//swift提供的既能拿到下标又能拿到元素,常见
//    print(index)
//    print(name)
//}
//
//for name in array1[0..<1] {//可以指定遍历区间
//    print(name)
//}

//4.数组合并
//注意:只有相同类型的数组才能合并,不一致不能
//let name1 = ["aoliu","xiaowang"]
//let name2 = ["xiaohong","xiaoli"]
//let name3 = name1 + name2

// 12.字典
// 字典允许按照某个键来访问元素
// 字典是由两部分集合构成的,一个是键 (key) 集合,一个是值 (value) 集合
// 键集合是不能有重复元素的,而值集合是可以重复的,键和值是成对出现的
// Swift中的字典
// Swift字典类型是Dictionary,也是一个泛型集合
// []如果里面放元素就是数组,如果放键值对就是字典

//1.定义字典
//let dict :[String:Any] = ["name":"AOLIU","age":26]
//var dictM = [String : Any]()

//2.对可变字典的基本操作

```

```

//dictM["name"] = "XIAOWANG"
//dictM["name"] = nil
//dictM.updateValue(19, forKey: "age")
//print(dictM)
//dictM.removeValue(forKey: "age")
//print(dictM)
//let name = dictM["name"]
//print(name ?? "dd") //如果name没有值为nil就用??后面的代替

//3.遍历字典
//3.1遍历字典中所有的key
//dictM["name"] = "XIAOWANG"
//dictM["age"] = 26
//for key in dictM {
//    print(key)//打印出的是一个元祖,随后看 ("name", "XIAOWANG") ("age", 26)
//    print(key.0)// name age
//    print(key.1)// XIAOWANG 26
//}

//for key in dictM.keys {
//    print(key)//name age
//}

//3.2遍历字典中所有的value
//for value in dictM.values {
//    print(value)//XIAOWANG 26
//}

//3.1遍历字典中所有的key/value
//for (key,value) in dictM {//()元祖
//    print(key)//name age
//    print(value)//XIAOWANG 26
//}

//4.合并字典
// 字典不可以相加合并
//let dict1 :[String:Any] = ["name":"AOLIU","age":18]
//var dict2 = ["height":1.80,"weight":"65"] as [String : Any]//如果字典中value值类型不一样就需要as如果一样就不需要
///var dict3 = ["height":1.80,"weight":65]
//
//for (key , value) in dict1 {
//    dict2.updateValue(value, forKey: key)
//}
//print(dict2)

// 13.元组
// 元组是Swift中特有的,OC中并没有相关类型
// 它是什么呢?
// 它是一种数据结构, 在数学中应用广泛
// 类似于数组或者字典
// 可以用于定义一组数据
// 组成元组类型的数据可以称为 “元素”

```

```

//let infoTuple = ("name",26,1.78)
//infoTuple.0//"name" String
//infoTuple.1// 26 Int
//infoTuple.2// 1.78 Double

//元组一般作为方法的多个返回值
//func test() -> (String,Int,Double) {
//    return ("name",12,1.22)
//}

//元组可以给每一个元素起别名
//let infoTuple1 = (name:"AOLIU",age:12,height:1.66)
//infoTuple1.name//AOLIU
//infoTuple1.age//12
//infoTuple1.height//1.66

//let (name,age,height) = ("name",26,1.78)
//name//"name" String
//age// 26 Int
//height// 1.78 Double

// 14.可选类型(重点,难点)
// 概念:
// 在OC开发中,如果一个变量暂停不使用,可以赋值为0(基本属性类型)或者赋值为空(对象类型)
// 在swift开发中,nil也是一个特殊的类型.因为和真实的类型不匹配是不能赋值的(swift是强类型
// 言)
// 但是开发中赋值nil,在所难免.因此推出了可选类型
// 可选类型的取值:
// 空值
// 有值

//可选类型推出的目的:对nil值进行处理
//1.定义可选类型
var name : Optional<String> = nil//方法一
var age : Int? = nil//方法二(语法糖)
if age != nil {
    print(age!)
    print("afdaf")
}
print(age ?? "hahahaha")
print(age as Any)

//现在打印可选类型有三种方式处理
//1.强制解包 print(age!)
//2.如果为空选择代替 print(age ?? "hahahaha")
//3.防止报错直接让age可以为任何类型 print(age as Any)

//2.给可选类型赋值
print(name ?? "d")
name = "AOLIU"
print(name ?? "d")//如果可选类型有值就是name,如果可选类型为nil就用??后面的代替

//3.从可选类型中取出具体的类型
//强制解包 : 可选类型 + !

```

```
print(name!)  
  
//4.可选类型注意:如果可选类型中没有值强制解包程序会崩溃  
//所以在对可选类型进行强制解包前,先判断可选类型是否等于nil  
if name != nil {  
    print(name!)  
}  
print(name ?? "kong")  
  
//5.可选绑定  
//可选绑定做了两件事情  
//1.判断name是否有值,如果没有(nil)大括号不会执行  
//2.如果name有值,系统会将可选类型进行解包然后赋值给tempName  
if let tempName = name {  
    print(tempName)  
}  
//或  
if let name = name {  
    print(name)  
}  
  
//可选类型练习:  
  
//练习一:  
  
let str = "13"  
  
let num = Int(str)//num为可选类型,因为str有可能能转为Int也有可能不能转为Int,如果不能转为Int  
会为nil,所以是可选类型  
  
print(num ?? "为空")//使用??后面的来提醒用户  
  
if num != nil {  
  
    print(num!)//通过强制解包获取num的值,但是如果num为nil就会崩溃,所以最好在强制解包之前进行  
    断  
  
}  
  
print(num as Any)//将num转化为Any任何值,但是获取到的是还是一个可选类型 Optijjonal()  
  
//练习二:  
  
let path = Bundle.main.path(forResource: "123.plist", ofType: nil)//返回的是一个可选类型,因为  
果bundle中没有找到123.plist就会返回nil  
  
if let path = path {  
  
    NSArray(contentsOfFile: path)//这里需要传的必须是一个String(路径);但是path是可选类型,所以  
    须进行强制解包,解包前最好进行判断  
}
```

```
//练习三:  
  
let url = URL(string: "www.baidu.com")  
  
if let url = url {  
  
    URLRequest(url: url)  
  
    print(url)  
  
}  
  
let urlStr = NSURL(string: "www.baidu.com")//同练习二,返回的是可选类型  
  
if let urlStr = urlStr {  
  
    URLRequest(url: urlStr as URL)  
  
    print(urlStr)  
  
}  
  
//15.类型转化  
  
//常见的类型转化符号  
  
//is : 用于判断一个实例是否是某一种类型 和OC中的isKindOfClass:功能一样  
  
//as : 将实例转成某一种类型  
  
//as?: 转成的最终类型是一个可选类型  
  
//as!: 转成的最终类型是一个具体类型  
  
//is的用法  
  
let array : [Any] = ["AOLIU",26,1.78]  
  
let ArrayM = [NSObject]()  
  
if let firstObj = array.first {  
  
    if firstObj is Int {  
  
        print("是Int")  
  
    }else{  
  
        print("不是Int")  
  
    }  
}
```

//as的用法

```
let dict : [String : Any] = ["name" : "AOLIU", "age" : 26, "height" : 1.78]
```

```
let dictM = [String : NSObject]()
```

let value = dict["name"]//value是一个可选类型,因为可能取出来有值,可能没有值

//as?: 转成的最终类型是一个可选类型

```
if let value = value {//可选绑定}
```

//将value转化为字符串

```
let valueStr = value as? String//valueStr还是一个可选类型,因为as?最终类型是一个可选类型
```

```
if let valueStr = valueStr {//可选绑定}
```

```
print("my name is " + valueStr)
```

```
}
```

```
}
```

//NSObject?转化为NSString?

```
if let valueStr = value as? String {
```

```
print("my name is " + valueStr)
```

```
}
```

//NSObject?转化为NSString //但是如果转化不成功就会崩溃

```
print("my name is " + (value as! String))
```