链滴

swift3.0 后基础语法 / 变化 - 类 三

作者：wyw89500

原文链接：https://ld246.com/article/1481731290417

来源网站：链滴

```
//: Playground - noun: a place where people can play

import UIKit

//  1.类

//面向对象的三大特性:

//1.封装

//2.继承

//3.多态

//多态的3大条件

//3.1 一定有继承

//3.2 一定有方法重写

//3.3 父类指针指向子类对象

//一. 类的介绍和定义

//

//Swift也是一门面向对象开发的语言

//面向对象的基础是类,类产生了对象

//在Swift中如何定义类呢?

//class是Swift中的关键字,用于定义类

//class 类名 : SuperClass {

// // 定义属性和方法

//}

//注意:

//1.定义的类,可以没有父类.那么该类是rootClass

//2.通常情况下,定义类时.继承自NSObject(非OC的NSObject)

//3.默认情况下系统会自动给我们提供一个构造函数 init()

//4.创建一个对象,必须保证该对象的所有属性都必须有初始化值

//5.如果自定义构造函数,就会覆盖系统的init()构造函数方法,如果不希望覆盖,必须明确实现

class Person : NSObject {
```

```
var name : String = "AOLIU"

var age : Int = 19


//1.重写系统的方法

override  init() {


}


//2.提供构造方法

init(name: String,age : Int) {

self.name = name

self.age = age

}


//3.提供构造方法

init(dict : [String : Any]) {


 if  let name = dict["name"] as? String {//因为dict["name"]取出的值是一个可选类型,我们先转换为tring的可选类型,然后再使用可选绑定进行处理

 self.name = name

 }

 if  let age = dict["age"] as? Int {

 self.age = age

 }


 }


}
```

```
//创建对象
let p = Person()
print(p.name)
print(p.age)
let p1 = Person(name: "xiaoWang", age: 18)
print(p1.name)
print(p1.age)
let p2 = Person(dict: ["name": "LIU" ,"age" : 20])
print(p2.name)
print(p2.age)
//  2.swift中的KVC
class Person1 : NSObject {
 var name : String = "AOLIU"
 var age : Int = 19

 //1.重写系统的方法
 override  init() {

 }

 init(dict : [String : Any]) {
 super.init()//先创建这个对象
 setValuesForKeys(dict)
 }

 override  func setValue(_ value: Any?, forUndefinedKey key: String) {
```

```
 }
}
let p3 = Person1(dict: ["name": "LIU_liu" ,"age" : 21])
print(p3.name)
print(p3.age)
//  3.类的析构函数
//Swift 会自动释放不再需要的实例以释放资源
//Swift 通过自动引用计数（ARC）处理实例的内存管理
//当引用计数为0时,系统会自动调用析构函数(不可以手动调用)
//通常在析构函数中释放一些资源(如移除通知等操作)
//3.1析构函数的写法
//deinit {
// // 执行析构过程
//}
class Person2 : NSObject {
 var name : String = "AOLIU"
 var age : Int = 19

 //释放时调用
 deinit {
 print("deinit---------")
 }
}
var p4 : Person2? = Person2()
p4 = nil
//  4.类的属性
//类的属性介绍
```

```
//Swift中类的属性有多种
//存储属性:存储实例的常量和变量
//计算属性:通过某种方式计算出来的属性
//类属性:与整个类自身相关的属性
//4.1存储属性
//存储属性是最简单的属性，它作为类实例的一部分，用于存储常量和变量
//可以给存储属性提供一个默认值，也可以在初始化方法中对其进行初始化
//下面是存储属性的写法
//age和name都是存储属性,用来记录该学生的年龄和姓名
//chineseScore和mathScore也是存储属性,用来记录该学生的语文分数和数学分数
class Student : NSObject {
 // 定义属性
 // 存储属性
 var age : Int = 0
 var name : String?

 var chineseScore : Double = 0.0
 var mathScore : Double = 0.0
}
// 创建学生对象
let stu = Student()
// 给存储属性赋值
stu.age = 10
stu.name = "LIU"
stu.chineseScore = 99.0
stu.mathScore = 98.0
//4.2计算属性
```

```
//计算属性并不存储实际的值，而是提供一个getter和一个可选的setter来间接获取和设置其它属性
//计算属性一般只提供getter方法
//如果只提供getter，而不提供setter，则该计算属性为只读属性,并且可以省略get{}
//下面是计算属性的写法
//averageScore是计算属性,通过chineseScore和mathScore计算而来的属性
//在setter方法中有一个newValue变量,是系统指定分配的
class Student1 : NSObject {
 // 定义属性
 // 存储属性
 var age : Int = 0
 var name : String?

 var chineseScore : Double = 0.0
 var mathScore : Double = 0.0

 // 计算属性
 //计算属性的常见写法
// var averageScore : Double {
// return (chineseScore + mathScore) / 2
//
// }

 var averageScore : Double {
 get {
 return (chineseScore + mathScore) / 2
 }
```

```
 // 没有意义,因为之后获取值时依然是计算得到的
 // newValue是系统分配的变量名,内部存储着新值
 //如果计算属性没有明确写set方法,那么该属性不可以赋值
 set {
 self.averageScore = newValue
 }
 }
}
let stu1 = Student1()
// 获取计算属性的值
print(stu1.averageScore)
//4.3类属性
//类属性是与类相关联的，而不是与类的实例相关联
//所有的类和实例都共有一份类属性.因此在某一处修改之后,该类属性就会被修改
//类属性的设置和修改,需要通过类来完成
//下面是类属性的写法
//类属性使用static来修饰
//courseCount是类属性,用来记录学生有多少门课程
class Student2 : NSObject {
 // 定义属性
 // 存储属性
 var age : Int = 0
 var name : String?

 var chineseScore : Double = 0.0
 var mathScore : Double = 0.0
```

```
// 计算属性,本质就是一个方法
var averageScore : Double {
get {
return (chineseScore + mathScore) / 2
}

// 没有意义.newValue是系统分配的变量名,内部存储着新值
set {
self.averageScore = newValue
}
}

// 类属性
static  var corseCount : Int = 0
}
// 设置类属性的值
Student2.corseCount = 3
// 取出类属性的值
print(Student2.corseCount)
//4.4监听属性的改变
//在OC中我们可以重写set方法来监听属性的改变
//Swift中可以通过属性观察者来监听和响应属性值的变化 属性监听器
//通常是监听存储属性和类属性的改变.(对于计算属性，我们不需要定义属性观察者，因为我们可以计算属性的setter中直接观察并响应这种值的变化)
//我们通过设置以下观察方法来定义观察者
//willSet：在属性值被存储之前设置。此时新属性值作为一个常量参数被传入。该参数名默认为newValue，我们可以自己定义该参数名
//didSet：在新属性值被存储后立即调用。与willSet相同，此时传入的是属性的旧值，默认参数名为o
```

```
dValue
//willSet与didSet只有在属性第一次被设置时才会调用，在初始化时，不会去调用这些监听方法
//监听的方式如下:
//监听age和name的变化
class Person3 : NSObject {

 var name : String? {
 // 可以给newValue自定义名称
 willSet (new){ // 属性即将改变,还未改变时会调用的方法
 // 在该方法中有一个默认的系统属性newValue,用于存储新值
 print(name ?? "")
 print(new ?? "")
 }
 // 可以给oldValue自定义名称
 didSet (old) { // 属性值已经改变了,会调用的方法
 // 在该方法中有一个默认的系统属性oldValue,用于存储旧值
 print(name ?? "")
 print(old ?? "")
 }
 }

 var age : Int = 0
 var height : Double = 0.0
}
let p5 : Person3 = Person3()
// 在赋值时,监听该属性的改变
// 在OC中是通过重写set方法
```

```
// 在swift中,可以给属性添加监听器
p5.name = "LIU"
```