Swift 学习笔记

1.常量 ：let

变量 ： var

例子：

let maximumNumberOfLoginAttempts = 10

var currentLoginAttempt = 0

2. 类型注解

说明变量中要存储的值的类型

例子：

var welcomeMessage: String

3.输出常量和变量

print(\_:separator:terminator:) //以\n结束当前行；不换行则传递空字符串

如：

print(someValue, terminator:"")

4.分号

Swift不强制要求在每一句的结尾加分号（；）

但在同一行写多条独立的语句时必须加。

例子：let cat = "🐱"; print(cat)

// 输出“🐱”

5.数值型字面量

十进制：没有前缀

二进制：前缀是 0b

八进制： 前缀是 0o

十六进制： 前缀是 0x

例子：

let decimalInteger = 17

let binaryInteger = 0b10001 // 二进制的17

let octalInteger = 0o21 // 八进制的17

let hexadecimalInteger = 0x11 // 十六进制的17

6.元组

把多个值组合成一个复合值。元组内的值可以是任意类型，并不要求是相同类型

例子：

let http404Error = (404, "Not Found")

// http404Error 的类型是 (Int, String)，值是 (404, "Not Found")

元组定义：

let http200Status = (statusCode: 200, description: "OK")

元组分解：

let (statusCode, statusMessage) = http404Error

print("The status code is \(statusCode)")

// 输出“The status code is 404”

print("The status message is \(statusMessage)")

// 输出“The status message is Not Found”

元组分解：只需要一部分元组值

方法1：

let (justTheStatusCode, \_) = http404Error

print("The status code is \(justTheStatusCode)")

// 输出“The status code is 404”

方法2：

print("The status code is \(http404Error.0)")

// 输出“The status code is 404”

print("The status message is \(http404Error.1)")

// 输出“The status message is Not Found”

7.可选类型

用于处理值可能缺失的情况。有两种可能：1）有值：解析可选类型访问这个值

2）根本没有值

可选的 int 被写作 int？而不是 int

// ？暗示包含的值是可选类型，也就是说可能包含 int 值也可能不包含值

8.nil （用于可选类型）

nil 表示它没有值：

例子：

var serverResponseCode: Int? = 404

// serverResponseCode 包含一个可选的 Int 值 404

serverResponseCode = nil

// serverResponseCode 现在不包含值

声明一个可选变量或常量未赋值，会自动设置为nil

例子：

var surveyAnswer: String?

// surveyAnswer 被自动设置为 nil

9.隐式解析可选类型

// 可选类型和隐式解析可选类型之间的区别

let possibleString: String? = "An optional string."

let forcedString: String = possibleString! // 需要感叹号来获取值

let assumedString: String! = "An implicitly unwrapped optional string."

let implicitString: String = assumedString // 不需要感叹号

10.空合运算符（a ?? b）

对可选类型a进行空判断。如果a包含值则解包，否则返回一个默认值b。

//注意：表达式a必须是optional类型。默认值b必须和a存储值的类型保持一致。

相当于三目运算符（三元运算符）// a?b:c(如果问题a成立，则执行语句b，否则执行语句c)

a != nil ? a! : b

//当可选类型a的值不为空时，则强行解包（a!)，访问a中的值；反之返回默认值b。

例子：

let defaultColorName = "red"

var userDefinedColorName: String? //默认值为 nil，未赋值

var colorNameToUse = userDefinedColorName ?? defaultColorName

// userDefinedColorName 的值为空，所以 colorNameToUse 的值为 "red"

11.区间运算符

1）闭区间运算符

a…b ：定义了一个包含从a到b（包括a和b）的所有值的区间。

例子：

for index in 1...5 {

print("\(index) \* 5 = \(index \* 5)")

}

// 1 \* 5 = 5

// 2 \* 5 = 10

// 3 \* 5 = 15

// 4 \* 5 = 20

// 5 \* 5 = 25

2）半开区间运算符

a…<b : 定义了一个从a到b但不包括b的所有值的区间。

3)单侧区间

a… ：定义了一个从a到结尾的所有值的区间

12.字符串

字符串索引：

startIndex：获取一个String的第一个Character的索引。

endIndex： 获取最后一个Character的后一个位置的索引。

因此endIndex不能作为一个字符串的有效下标。

index(before:) / index(after:) ： 得到前面一个或后面一个的索引。

index(\_:offsetBy:) ： 获取对应偏移量的索引。

1）使用字符：

for…in循环遍历字符串，获取每个字符的值

例子：

for character in "Dog!🐶" {

print(character)

}

// D

// o

// g

// !

// 🐶

2）连接字符串和字符

方法1：加法运算符（+）

let string1 = "hello"

let string2 = " there"

var welcome = string1 + string2

// welcome 现在等于 "hello there"

方法2：加法赋值运算符（+=）

var instruction = "look over"

instruction += string2

// instruction 现在等于 "look over there"

方法3：append（）

let exclamationMark: Character = "!"

welcome.append(exclamationMark)

// welcome 现在等于 "hello there!"

//注意：不能将一个字符串或者字符添加到一个已经存在的字符变量（单个字符）上

3）插入和删除

Insert（\_:at:）在一个字符串的指定索引插入一个字符

Insert（contents0f:at:） 在一个字符串的指定索引插入一个字符串

例子：

var welcome = "hello"

welcome.insert("!", at: welcome.endIndex)

// welcome 变量现在等于 "hello!"

welcome.insert(contentsOf:" there", at: welcome.index(before: welcome.endIndex))

// welcome 变量现在等于 "hello there!"

remove(at:) 在一个字符串的指定索引删除一个字符

removeSubrange(\_:) 在一个字符串的指定索引删除一个子字符串

例子：

welcome.remove(at: welcome.index(before: welcome.endIndex))

// welcome 现在等于 "hello there"

let range = welcome.index(welcome.endIndex, offsetBy: -6)..<welcome.endIndex

welcome.removeSubrange(range)

// welcome 现在等于 "hello"

13.集合类型：具有可变性

数组（Array)、集合（Set)和字典（Dictionary)三种基本的集合类型来存储集合数据

数组：有序数据的集

集合：无序无重复数据的集

字典：无序的键值对的集

1）数组

例子：

创建一个int类型的空数组：

var someInts = [Int]()

print("someInts is of type [Int] with \(someInts.count) items.")

// 打印“someInts is of type [Int] with 0 items.”

创建一个带有默认值的数组

var threeDoubles = Array(repeating: 0.0, count: 3)

// threeDoubles 是一种 [Double] 数组，等价于 [0.0, 0.0, 0.0]

通过两个数组相加创建一个数组

var anotherThreeDoubles = Array(repeating: 2.5, count: 3)

// anotherThreeDoubles 被推断为 [Double]，等价于 [2.5, 2.5, 2.5]

var sixDoubles = threeDoubles + anotherThreeDoubles

// sixDoubles 被推断为 [Double]，等价于 [0.0, 0.0, 0.0, 2.5, 2.5, 2.5]

14.函数

1）定义：

例子：

func greet(person: String) -> String {

let greeting = "Hello, " + person + "!"

return greeting

}

其中：greet(person:)———函数名

person————输入参数（String类型）

-> String————该类型的返回值

2）指定参数标签

你可以在参数名称前指定它的参数标签，中间以空格分隔

例子：

func someFunction(argumentLabel parameterName: Int) {

// 在函数体内，parameterName 代表参数值

* }

作用：参数标签的使用能够让一个函数在调用时更有表达力，更类似自然语言，

并且仍保持了函数内部的可读性以及清晰的意图。

3）忽略参数标签

如果你不希望为某个参数添加一个标签，可以使用一个下划线（\_）来代替一个明确的参数标签。

例子：

func someFunction(\_ firstParameterName: Int, secondParameterName: Int) {

// 在函数体内，firstParameterName 和 secondParameterName 代表参数中的第一个和第二个参数值

}

someFunction(1, secondParameterName: 2)

15.闭包(类似于函数)

1）语法表达式

{ (parameters) -> return type in

statements

}

例子：

reversedNames = names.sorted(by: { (s1: String, s2: String) -> Bool in

return s1 > s2

})

16.继承

基类：不继承于其他类的类

例子： class Vehicle {

var currentSpeed = 0.0

var description: String {

return "traveling at \(currentSpeed) miles per hour"

}

func makeNoise() {

// 什么也不做——因为车辆不一定会有噪音

}

}

1）子类生成（将超类名写在子类名的后面，用冒号分隔）

class SomeClass: SomeSuperclass {

// 这里是子类的定义

}

例子：

class Bicycle: Vehicle {

var hasBasket = false

}

新的Bicycle类自动继承Vehicle类的所有特性。

2）重写

若要重写某个特性，需要在重写定义的前面加上override关键字

例子：

class Train: Vehicle {

override func makeNoise() {

print("Choo Choo")

}

}

(/Users/yehong/.ssh/id\_rsa):

17.guard

像 if 语句一样，guard 的执行取决于一个表达式的布尔值。我们可以使用 guard 语句

来要求条件必须为真时，以执行 guard 语句后的代码。不同于 if 语句，一个 guard 语

句总是有一个 else 从句，如果条件不为真则执行 else 从句中的代码。