Swift 知识点

The Swift Programming Language

Swift 编程语言

第一个 swift 程序

print("hello world")

swift 一行代码结束后，不需要分号；

在 swift1.0 时，输出换行为 println("hello world")，这个函数不用了，改为 print 也可换行

这是一个完整的程序，这便是程序的入口，不需要 main 函数，oc 是有 main 函数作为程序的入口的

注释，分号

swift 的多行注释可以嵌套使用

swift 不需要使用分号，一行中写多个语句需要用分号分割

变量声明

let 声明常量，var 声明变量

常量：编译时不需要获取，只能赋值一次，可多次使用，值不可改变。

变量：变量可多次赋值，改变值，但不可改变类型

swift 是类型安全语言 支持类型推导，不需要指定类型

let a = 20　这里 a 自动推断为整数 Int 类型

也可指定类型

let a:Int = 200 指定 a 的类型为 Int 赋值其他类型则会报错

let a:Character = "1"　Character 指定类型只能包含一个字符

数据类型

整数

不带小数位的数字，包括有符号位（Int 正数负数 0），无符号位（UInt 正数 0）
swift 提供了 8 16 32 64 位数字形式，
用 min max 获取某个类型的最大最小值
var d = Int64.min
var e = Int64.max
Int 类型 根据机器选择位数 32 位机器=Int32 64 位机器=Int64

浮点型

两种浮点型 Double 64 位 Float 32 位

类型别名

typealias
typealias f = UInt16
var ee:f = 12

Bool

let a = 1
let b = true
let c = false
if a == 1 || b {

}
swift 中布尔型只有两个取值 true 和 false，不再是 oc 中那样非零即真。上面代码中 a 为 Int 类型，bc 是 Bool 类型，单独 a 不能做 if 条件，if a {} 这种写法是错的。b 和 c 是 Bool 型可以作为判断条件。

元组

元组，oc 中没有的一个新的数据类型，与数组和字典都类似，或者说是数组和字典的融合体。元组很任意：长度任意，元素类型任意。

let y = (9, "你好", true)

读取元组内的值很简单，元组名.第几个，这里看起来有点像数组的样子，相比于数组不用写[]，换成了.

print(y.2)//这里打印的就是 y 元组第 2 个元素 true

如果嫌脚标不够直观，可以给元组里每个元素命名

let(num, name, age) = y //依次给元组 y 里的元素命名

let(num, _, age) = y //如果不想访问第 1 个元素，则用_代替

print(num) //打印元素便可直接写元素名

元组还有另一种初始化形式，这个看起来像字典，每个元素初始化的时候便给它一个名字

let z = (name:"jack", age:22, isGood:true)

print(z.name) //打印的时候，名字便可代替脚标

元组的可变与不可变

let manInfo = (name:"jack", age:22, isGood:true)

var womanInfo = (name:"lili", age:20, isGood:true)

manInfo.name = "make" //let 不可变　报错，不可修改

womanInfo.name = "mimi" //var 可变　可以修改

womanInfo.age = "20" //这里注意：不论元组可变不可变，都不能修改元组内元素的数据类型

类型转换

int->string

let a = 10

let b:String = String(a)　或者 let b:String = “\(a)”

第二种也是拼接字符串常用的方法 print(“abcd\(a)”)

string->int

let a = “10”

let b = Int(a)

swift1.0 时的写法： let b = a.toInt()

这样写 b 的类型不确定，如果 a 能转换成 Int 类型，那么 b 就是 Int 类型，如果 a = “10a” 这样就无法转换成 Int 的（在 oc 中 a=@”10a” 可以 a.integerValue 转换出 10 的值），转换=nil

将 b 指定类型 let b:Int = Int(a) 这样写会报错，因为 a 不能保证是否能转换成 Int 类型

两种办法解决这个报错

let b:Int = Int(a)! 保证 a 肯定能转换出 Int 类型，如果 a 转不出来就报错了

let b:Int? = Int(a) 不保证 a 是否能转换出 Int 类型，就是 b 可空，可以赋值 nil

swift 是类型安全型语言

可空类型

var aa = "aaa"

aa = nil //这样是会报错的，aa 是 String 类型，不能为 nil

若想可以赋值 nil

var aa:String? = nil

断言

let a = 10

assert(a > 5, "aaaa")

条件判定 true 程序继续执行，判定 false 程序被中断，控制台输出断言信息

断言是一个开发特性，只在 debug 编译时有效，且运行时不被编译，不会消耗运行时性能。代码发布不需要刻意删除断言

在对应 target 的 Build Settings 中，我们在 Swift Compiler - Custom Flags 中的 Other Swift Flags 中添加 -assert-config Debug 来强制启用断言，或者 -assert-config Release 来强制禁用断言。

数组

var array = ["a", "b", 1]　swift 的数组可存放多种类型的元素，oc 的 NSArray 只能存放对象类型

初始化空数组 var array = []

初始化空数组并指定类型　var array = [String]()

之前的方法：var array = String[]()

控制流

if　switch 条件判断，　for-in　、for　、while、do-while 循环操作

```
let array = [0,1,2,34,123,234,11]
for a in array {
      print(a)
      if a > 0 { //不能单独写 a，if 条件必须为 bool　表达式
          print("\(a)大于 0")
       }else{
            print("\(a)不大于 0")
       }
}
```

for　循环

```
for intNum in 0...5 {     //包含 5
    print(intNum)
}
for intNum in 0..<5 {     //不包含 5    原来的写法 0..5
    print(intNum)
}
```

while 循环和 do-while 循环

```
var a = 0
while a < 10 {
      a += 2
      print(a)
```

```
}

repeat {
    a += 2
    print(a)
}  while a < 10
```

while 循环没有改变，经典的 do-while 改成了 repeat-while swift2.0 版本改的

switch 相比于 oc 也强大了很多，类型不仅仅是整数类型，支持任意类型的数据比较，break 也可以省略不写，第一个 case 符合不会跳转到下一个 case， default 不允许省略

```
switch ab {
        case "a":
            print("我是 a")
        case "b":
            print("我是 b")
            break;
        default:
            break;
        }
```

基本运算符

赋值运算

a = b

把 b 的值赋值给 a，在 if a = b {} 这种写法是错误的

是因为在 swift 中 a = b 赋值运算并不将自身作为一个值返回报错，还有因为 swift 的判断条件只能是 Bool 类型

let c = (a = b) 这样 c 的值并不==a 赋值后的值，因为 swift 赋值语句中，不会将自身返回

而在 oc 中 NSInteger c = (a = b); 这样 c 是==a 赋值后的值的。

在 oc 中 if (a = b) {} 这种写法只会报警告，是可以运行的 如果 if ((a = b)) {}这样写，完全 ok 警告都不会报，这个是将 a=b 赋值语句的返回值作为判断条件，在 oc 中非 0 即真

数学运算符

加减乘除 + - * /

加法运算通用适用于字符串拼接 “hello ” + “world” 输出 “hello world”

除法运算/ 和取余运算 %

oc 中支持 a / b 的操作，得出的结果是浮点型 ， 但是不允许 a%b oc 不支持浮点型%

NSInteger a = 10;

CGFloat b = 20.3;

Swift 中是不支持不同类型的数据进行/和%， swif 支持浮点型 %

自增自减

++a 和 a++ 效果不同，

var a = 0

let c = a++ //这样 c = 0

let c = ++a // 这样 c = 1

区间运算

(a...b)闭区间 从 a 到 b ， 包含 b

(a..<b)半闭区间，从 a 到 b，不包含 b 之前写作(a..b)

字符串

初始化空字符串

let a = “” 或者 let a = String()

判断是否为空 a.isEmpty 为空 true 非空 first

字符串遍历， swift1.0 时不需要写.characters

for c in a.characters {

```
    print(c)
}
```

字符计数

```
let num:Int = a.characters.count
```

swift1.0 时的写法：countElements(a)

闭包

闭包是指包含自由变量的代码块；本质来说函数就是特殊的闭包 swift 中可以用{}声明匿名闭包

```
var numbers = [20, 19, 7, 12]
numbers.map({
    (number: Int) -> Int in
    let result = 3 * number
    return result
})
```

简化写法

```
numbers.map({ number in 3 * number })
```

类和结构体

定义类 结构体

```
class Haha {
    var w = 0
    var h = 0
}
struct Hehe {
    var a1 = Haha()
    var a2 = false
    var name:String?
```

```
}
```

函数

函数，参数传入 String 类型，返回 Int 类型

```
func abc(ab:String) -> Int{
    print(ab)
    return 1
}
```

设置参数名

```
func abc(name ab:String) -> Int {
    return 10
}
```

设置函数默认值，默认值为 10， 调用可不传入参数 abc()

```
func abc (a:Int = 10)   -> Int{
    return a+=20
}
```

可变参数的函数

```
func (nums:Double...) -> Double {
    var n = 10
    for num   in   nums {
        n += num
    }
    return n
}
```

一个函数最多只能有一个可变参数

函数可以作为另一个函数的返回值

函数是一等一的对象，自然既可以当做返回值，又可以当做参数

返回函数，返回值处填的是函数类型

```
func mack() -> (Int -> Int) {
    func add(num:Int) ->Int {//作为返回值的函数
        return 2+num
     }
     return add
}
```

调用：

```
let a = mack()
print(i(7))
```

函数也可以作为另一个函数的参数，

```
func mak(a:Int, cont:Int -> Int) ->Int {
    return cont(a)
}
func temp(b:Int) -> Int { //作为参数的函数
    return b + 10
}
```

调用

```
let a = mak(10, cont:temp)
```

支持带有变长参数的函数：

```
func geta(num:Int...){
    var sum = 0
    for n in num {
        sum += n
    }
    print(sum)
}
```

调用

geta(1,2,3,2,5)

继承

一个类可以继承另一个类的方法，属性，继承类叫做子类，被继承的类叫做超类或父类。没有多继承，只有单继承

```
class man {
    var name:String?
    var age:Int?
    func make() {
        print("你好")
    }
}
class women: man {
    var height:Float?
    override func make() {
        print("哈哈哈")
    }
}
```