基本语法:

不用分号，不换行要用分号

import UIKit 自定义的类还是要引号

var str = “Hello” var表示变量，右值推断左值类型

打印print()不换行；println()换行

定义常量let const = 10

var x = 1\_000\_100会忽略\_

x = 0o17八进制 16进制x= 0x11 二进制:x=0b0001001

var str : String未赋值用类型标注

var d1 = 3.14 var i1 = 5 d1 = Double(i1)不支持隐式类型转换

连接字符串var text = str1 + str2

var width = 94 String(width)转字符串，或”\(width)”

var hour = 10 var min = 48 var sec = 30

let time = "当前时间是\(hour)点\(min)\(sec)秒"

其他整型Int UInt Int8(8位) UInt8 Int16…

其他浮点：Float32=Float Float64=Double Float96

布尔型: Bool 赋值:只能true或false

看类型大小：sizeof();

字符串是否为空: var str : String ; str=”” 或 var str = string() str.isEmpty

计算字符串长度let \_count = countElements(str)

当未指定为OC类型时，注意引用类型和(字面值)值类型的区别

当为NSMutableString连接字符串：strm.appendString("String") 不能＋

连接字符string.append(char1) string.hasPrefix(“TR”) string.hasSuffix(“!”)

得到字符：var ch:Character = “a” 否则默认为字符串,也不能用单引号

表情字符：edit->special characters

特殊图形：let sparklingHeart = "\u{0001F496}"

转义字符：let wiseWords = "\"想象力比知识更重要\"--爱因斯坦"

在swift中不能a = b = 10… 运算符号实际是调函数，b=20返回void

swift浮点数可以作取余运算

新运算符

var x : Int?加问号后x才能等于nil,且默认等于nil (可选值)

let r = (x != nil ? x : 0) 等价于 let r = (x ?? 0)

1...5: 1到5的区间 或1..<5: 实际1到4

swift的switch中没有break，没有隐式贯穿，必须有default，自己选择显示贯穿fallthrough case后面可以是区间或逗号分隔的，也可以是字符串等

或者case let x where x.hasSuffix(…”…\(x)”):

循环: for index in 1…10 {…}不可修改index for ch in “abcd” {…}

若index的值没用，可写成for \_ in 1…10 {…}

for var i = 0; i < 10; i++ {…}

元组：元素类型可以不相同

let http404Error : (Int, String) = (404, “Not Found”)

let (status, description) = http404Error 可单独访问status, description

let code = http404Error.0

let http505Status = (code : 505, description : “Unknow”)

访问http505Status.code http505Status.0

let point = (1, 0) switch point { case (0, 0): println("原点"); case (\_, 0): println("x轴"); case (-2...2, -2...2): println("…"); case (0, let y):… ; case let(x, y) where x == y : … ; case (let x, let y):(直接获取，有这个条件可不要default) ; default: break } 如果fallthough后面有声明变量会报错

泛型：var o : Optional<String> 类型可选

字符串转数字，!表示解包let a = "123".toInt()! 字符串格式不对可能不成功

var x = "2" if var y = x.toInt(){ y }else{…}这里的可选绑定是另一种形式，不是一般的if语句，y不用加解包的符号

断言：不满足条件，断掉，给出错误描述assert(x != "adf", …)

创建ixu：1:var arr = Array<Int>() 2:var s : Array<Int> = Array<Int>()

var s3 = [Int]() var s4 = [1, 2, 3, 4] var s5 : [Int] = []

var s6 = [Double](count: 3, repeatValue: 0.0) var s7 = Array(count: 3, repeatValue: “abc”)

创建对象数组：var views = Array<NSObject>() views.append(UIImage())原理即父类指针指向子类对象，从中取出对象也需要类型转换

简化声明数组：var strs = [String]()

更加简化：var strs = [“str1”, “str2”, “str3”] 获取strs[0] = “strstr1”

设为空: strs = [] 一些属性: strs.isEmpty strs.count strs += ["str4"]相同类型的数组才能相加 strs.insert("strn", atIndex: 2)

var remove = strs.removeAtIndex(1)返回删除的元素

删除区域var range = Range(start: 0, end: 1) strs.removeRange(range)

查：strs[1..<3] 查询是否有某个元素contains(strs, “str1”)返回Bool

find(strs, “str1”)返回下标，泛型

for (index, value) in enumerate(strs) {…}含下标带元素的遍历

采用OC方式创建数组var nsArray = NSMutableArray(array: [1,2,3])，属于引用类型

创建二维数组var nine = [[Int]](count: 3, repeatedValue: [Int](count: 3, repeatedValue: 0)) 调用：nine[row][col]

创建字典：var dic = Dictionary<String, Int>() var dic2 = [String : Int]()

var dic3 : Dictionary<String, Int> var dic5 = [“PEK” : “北京”, “CAN” : “广州”]

增加：dic5[“SHA”] = “上海” 改dic5[“PEK”] = “Beijing”

dic5.updateValue(“ShangHai”，forKey: “SHA”)返回修改前的值，若没有返nil

删除: dic5[“CAN”] = nil dic5.removeValueForKey(“PEK”) 打印..”…\(“PEK”).”

遍历字典：for (key, value) in dic5 {…”…\(key)…\(value)…”…}

或for key in dic5.keys{…}

var keys = [String](dic5.keys)用强制类型转换得倒相应字符串数组

创建空字典var dic3 : [Int : String] = [:] var in3 = dic3[0]返回值是nil

函数func sayHello(personName: String) ->String{ let greeting = "Hello" + personName + "!" return greeting }右边是返回值(若无返回值其中->String都不要) var str = sayHello("ads")

返回多个参数func count(str: String) ->(me:Int, you:Int, others:Int){ var me = 0, you = 0, others = 0 return (me, you, others) } let result = count("…”) result.you

外部名func fb(externa local : Int){… } fb(externa: 100)增加可读性

用#表示内外名相同func fb(#x : Int, y : Int){ println("\(x + 10)") } fb(x: 100, 10)

func fb(#x : Int, y : String = " "){ …} fb(x: 100)其中y表示不传参数就按默认的，否则调用fb(x: 100, y : “…”) 当中间参数有默认，后面参数要有外部名

不定个数参数：func mean(numbers : D8ouble…) ->Double{… for number in numbers { total += number } … } 参数numbers在方法中是一个数组

函数中，默认情况下传入参数是常量，要其可变则在前面加var

输入输出func \_swap(inout x : Int, inout y : Int){} \_swap(&a, &b) &对应输入输出，不是取地址，实际上实现方式就是取地址

函数也有类型，例如(Int, Int) ->Int 无参无返回值() ->()

声明函数变量：var f : (Int, Int) ->Int

函数数组：var fs : [(Int, Int)->Int]

返回值是函数类型func choose(back : Bool) ->(Int)->Int {…}

闭包Swift:Closures Ruby、OC:Block Python、C++11:Lamba Javascript:Anonymous Function 常用作调用函数时的参数

(定义sortInts冒泡函数，其中的比较语句用函数参数) sortInts(&a, { (a ：Int, b : Int)->Bool in return a > b })参数和返回值类型以及return可推断，写成sortInts(&a, {(a, b) in a<b}) 或sortInts(&a, {$0 < $1})或sortInts(&a, < )

自带的排序sort(&names, >)注：这里>表示从大到小排序，可读性的优化

迭代器的使用：let numbers = [16, 58, 510] let strings = numbers.map( { (var number) ->String in … return … })其中.map表示对每个元素操作

枚举enum status { case On case Off }或enum status { case On, Off } var sta = status.Off枚举没另外的int值。除非enum Week : Int { case Sun = 0, Mon, Tur, Wen, Thu, Fri, Sat }，取：let weekNum = Week.Fri.rawValue 通过原始值对枚举量赋值let weekDay = Week(rawValue: 5)!

若sta已是某枚举类型，可sta = .On，系统会根据上下文自动去取

枚举关联值：enum Barcode { case UPCA(Int,Int,Int) case QRCode(String) }

var productBarCode = Barcode.UPCA(690, 253800584, 4)

switch productBarCode { case .UPCA(let(num, id, check)): println("条形码：\(num)\(id)\(check)") case .QRCode(let pCode): println("二维码：\(pCode)")}

“===”用于判断两个引用是否引用了同一个对象 “==”比较内容

struct Resolution { var width : Float var height : Float } var re = Resolution(width: 0, height: 0)

lazy修饰成员变量：延迟加载，只有真正读取该属性的值时，该属性才会被创建

计算属性：struct Rect { var origin = Point() var size = Size() var center : Point{ get{ return Point(x: origin.x + size.width/2, y: origin.y + size.height/2) }

set/\*(newCenter)\*/{ origin = Point(x: newValue.x - size.width/2, y: newValue.y - size.height/2) } } }

?属性监视器/记步监视器class StepCounter { var a : Int = 0 var totalSteps : Int = 0 { willSet (newStep){ println("将要变成\(newStep)") a = newStep } didSet (oldStep) {a = oldStep}}}

类型属性在结构体或枚举中(用static)通过类名访问,在类中用class(只计算属性), 新语法也可用static声明类中的存储属性

类的函数中，遵循局部变量优先原则，否则会再找实例变量。static定义的函数中，无论是否加self.都报错 self.。。。是隐式传递

类中第二个及以后的参数既是内部名，也是外部名。也可以自己取外部名。也可以\_为外部名，调用时则可省略

struct中值类型的实例方法默认是只读方法，不能修改成员变量，如果要修改，在方法前加mutating，枚举也是一样

class SomeClass { func show() {…}//实例方法 class func method {…} //类方法 static func method2 { … }} 类变量前面加static，类方法可以访问 实例方法中可以访问实例变量和类变量

struct中类方法只能加static

类中的下标class MyNSArray { var array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] subscript (index : Int) ->Int{ get{ return array[index] } set{ array[index] = newValue } } } var myArray = MyNSArray() myArray[5] myArray[1] = 100

构建矩阵struct Metrix {var grid : [Double] let rows : Int, columns : Int init(rows : Int, columns : Int){//创建对象被动调用 self.rows = rows self.columns = columns grid = [Double](count: rows \* columns, repeatedValue: 0.0) } func indexIsValidForRow(row : Int, columns : Int)->Bool { return row >= 0 && row < self.rows && columns >= 0 && columns < self.columns } subscript(row : Int, col : Int) ->Double{ get{ return grid[row \* self.columns + col]} set{ assert(indexIsValidForRow(row, columns: col), "越界") grid[row \* self.columns+col]=newValue}}}var m = Metrix(rows: 4, columns: 4) m[0, 0] = 1.0

子类重写父类的方法要加关键字override，若还要调用父类内容用super.

加关键字final限制子类不能重写父类的属性和方法

创建对象一定会调用初始化器struct F {var t : Double init () {t = 32}} var f = F()

初始化器里可以委托初始化器，用法类似self.调函数

子类重写父类的初始化构造器会自动调用父类的无参构造器，否则手动调用父类的有参初始化方法。必须先初始化本类定义的属性，才能调用父类的初始化器。如果在子类初始化方法中要给父类继承下来的属性赋值，必须在父类的初始化方法调用后赋值。如果子类拥有父类的所有指定初始化器，父类中所有的便利初始化器都会被子类继承。

便利初始化器convenience里必须调用一个指定初始化器

可失败的初始化器：init?... struct Animal {let name : Stringinit?(name : String) {if name.isEmpty{return nil}else{self.name = name}}} var ani = Animal(name: "dog") ani?.name

强制解包：init!... 但要确保返回不为nil

父类中有用required修饰的初始化器，子类要继承他必须重写该初始化器

deinit {…}：对象销毁时调用

可选链：a?.b?.c?.d即使中间为空链不会断掉

一个对象是否是属于某个类型：Bool bool = obj is class

类型转换(一般前提：父类指针指向子类对象)let move = item as? Movie，转换失败得到nil。或取as!为强制解包，转换失败则程序崩溃

AnyObject：相当于OC中的id类型

let nsString = NSString(format: "abc%d", 10)

类的扩展：extenstion Double {…} 不能扩展存储属性，可以扩展计算属性

extension Double{ var km : Double{ return self \* 1\_000 } }

拖尾闭包：如果最后一个参数传入的是闭包函数，可以不写最后一个参数，直接在参数列表后面使用闭包 30.repeat(10){…} 或者只有函数参数30.repeat(…)

扩展已有的方法会覆盖原来的方法

协议：protocol MyProtocol { var prop : Int {get set} var onlyRead : Int {get} func instanceMethod() ->Int class func typeMethod() }

如果对象是遵守协议的类的类型，都是可读写属性

兼容OC的协议：@objc protocol ObjcProtocol{ func requiredMethod() optional func optionalMethod() }(加@objc才能用optional)

可以重载＋号运算符func + (left : Fraction, right : Fraction)->Fraction { … }

继承NSObjectProtocol才可以weak var delegate，同时遵守协议的类是NSObject的子类

修饰类的关键词：public：任何地方都可以访问，包括项目外；internal(默认)：只能在本项目内部，或框架内部访问；private：此类只能在本类中使用。当修饰属性或方法也一样，但使用权限不能高于所在的类

数据库查找工具：NavicatPremium11-1-12.dmg 聊天工具adium

XMPP:软件.zip->ejabberd… 安装完成后:应用程序->ejabberd…->bin->start->admin interface-> 虚拟主机->用户->添加用户

基于很多协议的聊天工具：Adium 打开->继续->服务：XMPP, Jabber ID：tarena1@tarena.com, 密码：tarena1-> Adium->偏好设置->编辑->选项->连接服务器：127.0.0.1,端口：5222-> 打开电脑上的信息->以后->其他->账户类型：Jabber，账户名称：tarena2@tarena.com，服务器：127.0.0.1->添加好友/联系人

启界面：左上角target->general->App Icons …->Launch Screen File清空，Launch Image Source：Use Assert Catalog->Migrate->Images.xcassets->LaunchImage

添加框架：左上角target->Build Phases->Link…->＋

libxml2框架：xml解析

添加头文件搜索路径，左上角target->Build Settings->Header Search Paths->添加/usr/include/libxml2

创建桥接文件：command＋N->Header File…->在其中导入综合文件

->BuildSetting->搜Swift Compiler-Code Generation 选择Objective-C Bridging Header,把桥文件的名称输入进去 注意带路径WeiShenBian/Classes/WSBXmppHeader.h

返回1像素拉伸图片image!.stretchableImageWithLeftCapWidth(leftCap, topCapHeight: topCapHeight )

九切片：var edge = UIEdgeInsets(top: top, left: left, bottom: bottom, right: right)

return image!.resizableImageWithCapInsets(edge, resizingMode: UIImageResizingMode.Tile)

button.setBackgroundImage(UIImage.strectchedImageWithName(image\_bg), forState: UIControlState.Normal)按钮某一状态下的图片

class func sharedXMPPSupport() ->XMPPSupport {struct Singleton {static var predicate : dispatch\_once\_t = 0 static var instance : XMPPSupport? = nil} dispatch\_once(&Singleton.predicate, { Singleton.instance = XMPPSupport()})

return Singleton.instance! }单例方法

//MARK: swif中的快速定位 类似于OC中的#programmark

初始化\_xmppStream.addDelegate(self, delegateQueue: dispatch\_get\_main\_queue ())

//从沙箱中取出 用户名var user = NSUserDefaults.standardUserDefaults ().objectForKey("user") as! String

//设置用户名 JID ： \_xmppStream.myJID = XMPPJID.jidWithString(user + "@tarena.com", resource: "iphone")

//设置域名／IP地址： \_xmppStream.hostName = "localhost"

//设置端口 默认是5222： \_xmppStream.hostPort = 5222

//链接服务器：\_xmppStream.connectWithTimeout(XMPPStreamTimeoutNone, error: &error)

//重写delegat中 链接成功的方法func xmppStreamDidConnect(sender: XMPPStream!) {self.sendPwdToHost()}

//链接失败func xmppStreamDidDisconnect(sender: XMPPStream!, withError error: NSError!) {NSLog("链接失败 : \(error?.description)")}

var defaults = NSUserDefaults.standardUserDefaults() defaults.setObject(user, forKey: "user") defaults.setObject(pwd, forKey: "pwd") defaults.synchronize()

//隐藏键盘 self.view.endEditing(true)

MBProgressHUD框架

Sheet形式刷新圈：MBProgressHUD.showMessage("正在登录中...")

.xmppUserLogin { [unowned self] (type) -> () in … } unowned作用相当于weak，用之前确保self不为空，否则也可以用weak(那么self就是可选值)

//隐藏等待菊花 ：MBProgressHUD.hideHUD()

//写XMPPStreamDelegate中链接成功的方法:

func xmppStreamDidConnect(sender: XMPPStream!) {…}

//发送注册密码,用户注册\_xmppStream.registerWithPassword(pwd, error: nil)

//发送密码验证\_xmppStream.authenticateWithPassword(pwd, error: nil)

创建storyBoard，用于跳转var storyBoard = UIStoryboard(name: "Main", bundle: nil) 跳转： self.view.window?.rootViewController = storyBoard. instantiateInitialViewController() as? UIViewController

默认参数是常量，也可以声明成变量，以方便函数内的操作：func alignRight(var string: String, totalLength: Int, pad: Character) -> String {…}

//func reverse() -> [T]?

Mirror（反射）的作用：遍历属性、查看对象的类型

guard: 相比if let，可将代码放在条件判断之外

?? 左边赋值为空时，取右边的值

defer: 即将离开当前代码块时执行一系列语句

结构体和枚举是值类型，值类型的属性不能在它的实例方法中被修改。也可以在方法前加mutating做修改

实现协议中的初始化方法，需要在方法前加required

当一个类型已经符合了某个协议中的所有要求，却还没有声明采纳该协议时，可以通过空扩展体的扩展来采纳该协议：extension Hamster: TextRepresentable {}

NSRegularExpression(pattern: "^(\\d{14}|\\d{17})(\\d|[xX])$").isMatch(appUser.userIdcard) 正则验证，isMatch函数属于RegExCategories三方库

自定义输入类型键盘textView.inputView = inputView, 光标位置textView.textContainerInset = UIEdgeInsetsMake(5, 5, 5, 5)

地图截图和坐标let snape = MKMapSnapshotter(options: option) snape.startWithCompletionHandler{…}

访问通讯录(ios9.0)let store = CNContactStore() let keyToFetch = [CNContactFormatter.descriptorForRequiredKeysForStyle(.FullName), CNContactImageDataKey, CNContactPhoneNumbersKey] let fetchRequest = CNContactFetchRequest(keysToFetch: keyToFetch) var contacts = [CNContact]() do {try store.enumerateContactsWithFetchRequest(fetchRequest, usingBlock: { (contact, stop) -> Void in contacts.append(contact) print(contact) }) }catch{}

添加多个动画：CAAnimationGroup().animations = [CAKeyframeAnimation]()

CAKeyframeAnimation().values .path . keyTimes

arc4random\_uniform(n), 产生0到n之间的随机数

CACurrentMediaTime()当前时间，CFTimeInterval(…)间隔时间

粒子动画：设置发射方向：self.setValue(nextLongitude, forKeyPath: "emitterCells.triangle.emissionLongitude")

封装无代码设置属性的类：@IBInspectable var test1 : CGFloat = 0

CGContextDrawLinearGradient(…)渐变颜色的图

默认行数(静态tableview)return super.tableView(tableView, numberOfRowsInSection: section)

SceneKit适用3D动画,在应用中嵌入游戏场景出现self.view as! SCNView编译错误可用self.view.addSubview的方式解决

SpriteKit适用2D动画

OpenGL和Metal(只能在真机运行)更底层,2D和3D都可以。

扩展属性: var extraProperty : String!{set{objc\_setAssociatedObject(self, &AssociatedKeys.extraPropertyStr,newValue,objc\_AssociationPolicy.OBJC\_ASSOCIATION\_RETAIN\_NONATOMIC)}get{return objc\_getAssociatedObject(self, &AssociatedKeys.extraPropertyStr) as! String}}private struct AssociatedKeys {static var extraPropertyStr:String?}

OC分类属性: -(void)setFail\_hint\_des:(UILabel \*)fail\_hint\_des {objc\_setAssociatedObject(self,"label",fail\_hint\_des,OBJC\_ASSOCIATION\_RETAIN);} -(UILabel \*)fail\_hint\_des{return objc\_getAssociatedObject(self, "label");}

获取属性列表：var outCount : UInt32 = 0 let vars = class\_copyIvarList(Son.classForCoder(), &outCount)! for i in 0..<outCount {let property = vars[Int(i)] print(String(utf8String: ivar\_getName(property))) }

metal教程：device, commandQueue

三方：

Dollar：用类似JS的方式操作数组；Cent：基于Dollar的封装，语法更面向对象

alamofire:何时回调.responseJSON:将回调放在delegate.queue中, 在SessionDelegate中先调用didReceive data方法(得到mutableData)再调didCompleteWithError方法通过suspend=false执行

状态栏颜色设置为黑色：<http://www.jianshu.com/p/0ca18ea37277>

info.plist中View controller-based status bar appearance = NO

UIApplication.shared.statusBarStyle = .default