|  |  |  |
| --- | --- | --- |
| 项目 | 教程 | 备注 |
| [Swift 教程](http://www.runoob.com/swift/swift-tutorial.html" \o "Swift 教程" \t "http://www.runoob.com/swift/_top) | Swift 是一种支持多编程范式和编译式的开源编程语言,苹果于2014年WWDC（苹果开发者大会）发布，  用于开发 iOS，OS X 和 watchOS 应用程序。  Swift 结合了 C 和 Objective-C 的优点并且不受C兼容性的限制。  Swift 在 Mac OS 和 iOS 平台可以和 Object-C 使用相同的运行环境。  2015年6月8日，苹果于WWDC 2015上宣布，Swift将开放源代码，包括编译器和标准库。  /\* 我的第一个 Swift 程序 \*/  var myString = "Hello, World!"  print(myString)  网址 https://developer.apple.com/swift/ |  |
| [Swift 环境搭建](http://www.runoob.com/swift/swift-environment.html" \o "Swift 环境搭建" \t "http://www.runoob.com/swift/_top) | 打开 Xcode，打开后在屏幕顶部选择 File => New => Playground。  Swift 的 playground 就像是一个可交互的文档，用来练手学swift的，写一句代码出一行结果（右侧），可以实时查看代码结果，  以下是 Swift Playground 窗口默认的代码：  import UIKit  var str = "Hello, playground"  如果你想创建 OS x 程序，需要导入 Cocoa 包 import Cocoa 代码如下所示：  import Cocoa  var str = "Hello, playground"  Language有两个选项：Objective-c和swift  Phone或iTouch的宽为320像素，高为480像素，状态栏高为20像素，toobar高为44像素，tabbar高为49像素，导航栏高为44像素。  运行一下模拟器（command+R 快捷键或在菜单栏中选择 Product => Run)。 |  |
| [Swift 基本语法](http://www.runoob.com/swift/swift-basic-syntax.html" \o "Swift 基本语法" \t "http://www.runoob.com/swift/_top) | **Swift 引入**  我们可以使用 import 语句来引入任何的 Objective-C 框架（或 C 库）到 Swift 程序中。  例如 import cocoa 语句导入了使用了 Cocoa 库和API，我们可以在 Swift 程序中使用他们。  Cocoa 本身由 Objective-C 语言写成，Objective-C 又是 C 语言的严格超集，所以在 Swift 应用中我们可以很简单的混入 C 语言代码，甚至是 C++ 代码。  **Swift 标记**  Swift 程序由多种标记组成，标记可以是单词，标识符，常量，字符串或符号。例如以下 Swift 程序由三种标记组成：  print("test!") 标记是：单词、符号  print( "test!")  **注释**  Swift的注释与C语言极其相似，单行注释以两个反斜线开头：  //这是一行注释  多行注释以/\*开始，以\*/结束:  /\* 这也是一条注释，  但跨越多行 \*/  与 C 语言的多行注释有所不同的是，Swift 的多行注释可以嵌套在其他多行注释内部。写法是在一个多行注释块内插入另一个多行注释。第二个注释块封闭时，后面仍然接着第一个注释块：  /\* 这是第一个多行注释的开头  /\* 这是嵌套的第二个多行注释 \*/  这是第一个多行注释的结尾 \*/  多行注释的嵌套是你可以更快捷方便的注释代码块，即使代码块中已经有了注释。  **分号**  与其它语言不同的是，Swift不要求在每行语句的结尾使用分号(;)，但当你在同一行书写多条语句时，必须用分号隔开：  import Cocoa  /\* 我的第一个 Swift 程序 \*/  var myString = "Hello, World!"; print(myString)  **标识符**  标识符就是给变量、常量、方法、函数、枚举、结构体、类、协议等指定的名字。  构成标识符的字母均有一定的规范，Swift语言中标识符的命名规则如下：  区分大小写，Myname与myname是两个不同的标识符；  标识符首字符可以以下划线（\_）或者字母开始，但不能是数字；  标识符中其他字符可以是下划线（\_）、字母或数字。  例如： userName、User\_Name、\_sys\_val、身高等为合法的标识符，而2mail、room#和class为非法的标识符。  注意:Swift中的字母采用的是Unicode编码[1]。  Unicode叫做统一编码制，它包含了亚洲文字编码，如中文、日文、韩文等字符，甚至是我们在聊天工具中使用的表情符号  如果一定要使用关键字作为标识符，可以在关键字前后添加重音符号（`），例如：  **关键字**  关键字是类似于标识符的保留字符序列，除非用重音符号（`）将其括起来，否则不能用作标识符。  关键字是对编译器具有特殊意义的预定义保留标识符。  常见的关键字有以下4种。  **与声明有关的关键字**   |  |  |  |  | | --- | --- | --- | --- | | class | deinit | enum | extension | | func | import | init | internal | | let | operator | private | protocol | | public | static | struct | subscript | | typealias | var |  |  |   与语句有关的关键字   |  |  |  |  | | --- | --- | --- | --- | | break | case | continue | default | | do | else | fallthrough | for | | if | in | return | switch | | where | while |  |  |   **表达式和类型关键字**   |  |  |  |  | | --- | --- | --- | --- | | as | dynamicType | false | is | | nil | self | Self | super | | true | \_COLUMN\_ | \_FILE\_ | \_FUNCTION\_ | | \_LINE\_ |  |  |  |   **在特定上下文中使用的关键字**   |  |  |  |  | | --- | --- | --- | --- | | associativity | convenience | dynamic | didSet | | final | get | infix | inout | | lazy | left | mutating | none | | nonmutating | optional | override | postfix | | precedence | prefix | Protocol | required | | right | set | Type | unowned | | weak | willSet |  |  |   **Swift 空格**  Swift语言并不是像C/C++，Java那样完全忽视空格，Swift对空格的使用有一定的要求，但是又不像Python对缩进的要求那么严格。  在Swift中，运算符不能直接跟在变量或常量的后面。例如下面的代码会报错：  let a= 1 + 2  错误信息是：  error: prefix/postfix '=' is reserved  意思大概是等号直接跟在前面或后面这种用法是保留的。  下面的代码还是会报错（继续注意空格）：  let a = 1+ 2  错误信息是：  error: consecutive statements on a line must be separated by ';'  这是因为Swift认为到1+这个语句就结束了，2就是下一个语句了。  只有这样写才不会报错：  let a = 1 + 2; // 编码规范推荐使用这种写法let b = 3+4 // 这样也是OK的  **Swift 字面量**  所谓字面量，就是指像特定的数字，字符串或者是布尔值这样，能够直接了当地指出自己的类型并为变量进行赋值的值。比如在下面：  42 // 整型字面量  3.14159 // 浮点型字面量  "Hello, world!" // 字符串型字面量  true // 布尔型字面量 |  |
| [Swift 数据类型](http://www.runoob.com/swift/swift-data-types.html" \o "Swift 数据类型" \t "http://www.runoob.com/swift/_top) | **Swift 数据类型**  在我们使用任何程序语言编程时，需要**使用各种数据类似来存储不同的信息**。  **变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。**在声明变量时也可指定它的数据类型。  **所有变量都具有数据类型，以决定能够存储哪种数据。**  **内置数据类型**  Swift 提供了非常丰富的数据类型，以下列出了常用了集中数据类型：  Int  一般来说，你不需要专门指定整数的长度。Swift 提供了一个特殊的整数类型Int，长度与当前平台的原生字长相同：  在32位平台上，Int和Int32长度相同。  在64位平台上，Int和Int64长度相同。  除非你需要特定长度的整数，一般来说使用Int就够了。这可以提高代码一致性和可复用性。即使是在32位平台上，Int可以存储的整数范围也可以达到-2,147,483,648~2,147,483,647，大多数时候这已经足够大了。  **UInt**  Swift 也提供了一个特殊的无符号类型UInt，长度与当前平台的原生字长相同：  在32位平台上，UInt和UInt32长度相同。  在64位平台上，UInt和UInt64长度相同。  注意： **尽量不要使用UInt，除非你真的需要存储一个和当前平台原生字长相同的无符号整数。**除了这种情况，最好使用Int，即使你要存储的值已知是非负的。统一使用Int可以提高代码的可复用性，避免不同类型数字之间的转换，并且匹配数字的类型推断，请参考*[类型安全和类型推断](http://www.runoob.com/swift/swift-data-types.html" \l "type_safety_and_type_inference)*。  **浮点数**  浮点数是有小数部分的数字，比如3.14159，0.1和-273.15。  浮点类型比整数类型表示的范围更大，可以存储比Int类型更大或者更小的数字。Swift 提供了两种有符号浮点数类型：  Double表示64位浮点数。当你需要存储很大或者很高精度的浮点数时请使用此类型。  Float表示32位浮点数。精度要求不高的话可以使用此类型。  注意： Double精确度很高，至少有15位数字，而Float最少只有6位数字。选择哪个类型取决于你的代码需要处理的值的范围。  **布尔值**  Swift 有一个基本的布尔（Boolean）类型，叫做Bool。布尔值指逻辑上的值，因为它们只能是真或者假。  Swift 有两个布尔常量，true和false。  **字符串**  字符串是字符的序列集合，例如：  "Hello, World!"  字符  字符指的是单个字母，例如：  "C"  **可选类型**  使用可选类型（optionals）来处理值可能缺失的情况。可选类型表示有值或没有值。  **数值范围**  下表显示了不同变量类型内存的存储空间，及变量类型的最大最小值：   |  |  |  | | --- | --- | --- | | 类型 | 大小（字节） | 区间值 | | Int8 | 1 字节 | -127 到 127 | | UInt8 | 1 字节 | 0 到 255 | | Int32 | 4 字节 | -2147483648 到 2147483647 | | UInt32 | 4 字节 | 0 到 4294967295 | | Int64 | 8 字节 | -9223372036854775808 到 9223372036854775807 | | UInt64 | 8 字节 | 0 到 18446744073709551615 | | Float | 4 字节 | 1.2E-38 到 3.4E+38 (~6 digits) | | Double | 8 字节 | 2.3E-308 到 1.7E+308 (~15 digits) |   **类型别名**  类型别名对当前的类型定义了另一个名字，类型别名通过使用 typealias 关键字来定义。语法格式如下：  typealias newname = type  例如以下定义了 Int 的类型别名为 Feet：  typealias Feet = Int  现在，我们可以通过别名来定义变量：  import Cocoa  typealias Feet = Int  var distance: Feet = 100  print(distance)  我们使用 playground 执行以上程序，输出结果为：  100  **类型安全**  Swift 是一个类型安全（type safe）的语言。  由于 Swift 是类型安全的，所以它会在编译你的代码时进行类型检查（type checks），并把不匹配的类型标记为错误。  这可以让你在开发的时候尽早发现并修复错误。  import Cocoa  var varA = 42  varA = "This is hello"print(varA)  以上程序，会在 Xcode 中报错：  error: cannot assign value of type 'String' to type 'Int'  varA = "This is hello"  意思为不能将 'String' 字符串赋值给 'Int' 变量。  **类型推断**  当你要处理不同类型的值时，类型检查可以帮你避免错误。然而，这并不是说你每次声明常量和变量的时候都需要显式指定类型。  **如果你没有显式指定类型，Swift 会使用类型推断（type inference）来选择合适的类型。**  例如，如果你给一个新常量赋值42并且没有标明类型，Swift 可以推断出常量类型是Int，因为你给它赋的初始值看起来像一个整数：  let meaningOfLife = 42  // meaningOfLife 会被推测为 Int 类型  同理，如果你没有给浮点字面量标明类型，Swift 会推断你想要的是Double：  let pi = 3.14159// pi 会被推测为 Double 类型  当推断浮点数的类型时，Swift 总是会选择Double而不是Float。  如果表达式中同时出现了整数和浮点数，会被推断为Double类型：  let anotherPi = 3 + 0.14159  // anotherPi 会被推测为 Double 类型  原始值3没有显式声明类型，而表达式中出现了一个浮点字面量，所以表达式会被推断为Double类型。  实例  import Cocoa  // varA 会被推测为 Int 类型 var varA = 42print(varA)  // varB 会被推测为 Double 类型 var varB = 3.14159print(varB)  // varC 也会被推测为 Double 类型 var varC = 3 + 0.14159print(varC)  执行以上代码，输出结果为：  423.141593.14159 |  |
| [Swift 变量](http://www.runoob.com/swift/swift-variables.html" \o "Swift 变量" \t "http://www.runoob.com/swift/_top) | **Swift 变量**  **变量是一种使用方便的占位符，用于引用计算机内存地址。**  Swift 每个变量都指定了特定的类型，该类型决定了变量占用内存的大小，不同的数据类型也决定可存储值的范围。  上一章节我们已经为大家介绍了[基本的数据类型](http://www.runoob.com/swift/swift-data-types.html" \t "http://www.runoob.com/swift/_blank)，包括整形Int、浮点数Double和Float、布尔类型Bool以及字符串类型String。此外，Swift还提供了其他更强大数据类型， Optional, Array, Dictionary, Struct, 和 Class 等。  接下来我们将为大家介绍如何在 Swift 程序中声明和使用变量。  **变量声明**  变量声明意思是告诉编译器在内存中的哪个位置上为变量创建多大的存储空间。  在使用变量前，你需要使用 var 关键字声明它，如下所示：  var variableName = <initial value>  以下是一个 Swift 程序中变量声明的简单实例：  import Cocoa  var varA = 42print(varA)  var varB:Float  varB = 3.14159print(varB)  以上程序执行结果为：  423.14159  **变量命名**  变量名可以由字母，数字和下划线组成。  变量名需要以字母或下划线开始。  Swift 是一个区分大小写的语言，所以字母大写与小写是不一样的。  变量名也可以使用简单的 Unicode 字符，如下实例：  import Cocoa  var \_var = "Hello, Swift!"  print(\_var)  var 你好 = "你好世界"var 菜鸟教程 = "www.runoob.com"print(你好)print(菜鸟教程)  以上程序执行结果为：  Hello, Swift!你好世界  www.runoob.com  **变量输出**  变量和常量可以使用 print（swift 2 将 print 替换了 println） 函数来输出。  在字符串中可以使用括号与反斜线来插入变量，如下实例：  import Cocoa  var name = "菜鸟教程"var site = "http://www.runoob.com"  print("\(name)的官网地址为：\(site)")  以上程序执行结果为：  菜鸟教程的官网地址为：http://www.runoob.com |  |
| Swift 可选(Optionals)类型 | Swift 的可选（Optional）类型，用于处理值缺失的情况。  可选表示"那儿有一个值，并且它等于 x "或者"那儿没有值"。  Swfit语言定义后缀？作为命名类型Optional的简写，换句话说，以下两种声明是相等的：  var optionalInteger: Int?  var optionalInteger: Optional<Int>  在这两种情况下，变量optionalInteger都是可选整数类型。注意，在类型和？之间没有空格。  Optional 是一个含有两种情况的枚举，None和Some(T)，用来表示可能有或可能没有值。任何类型都可以明确声明为（或者隐式转换）可选类型。当声明一个可选类型的时候，要确保用括号给？操作符一个合适的范围。  例如，声明可选整数数组，应该写成(Int[])?；写成Int[]?会报错。  当你声明一个可选变量或者可选属性的时候没有提供初始值，它的值会默认为nil。  可选项遵照LogicValue协议，因此可以出现在布尔环境中。在这种情况下，如果可选类型T?包含类型为T的任何值（也就是说它的值是Optional.Some(T)），这个可选类型等于true，反之为false。  如果一个可选类型的实例包含一个值，你可以用后缀操作符 ！来访问这个值，如下所示：  optionalInteger = 42  optionalInteger! // 42  使用操作符！去获取值为nil的可选变量会有运行时错误。  你可以用可选链接和可选绑定选择性执行可选表达式上的操作。如果值为nil，任何操作都不会执行，也不会有运行报错。  让我们来详细看下以下实例来了解 Swift 中可选类型的应用:  import Cocoa  var myString:String? = nil  if myString != nil {  print(myString)}else{  print("字符串为 nil")}  以上程序执行结果为：  字符串为 nil  可选类型类似于Objective-C中指针的nil值，但是nil只对类(class)有用，而可选类型对所有的类型都可用，并且更安全。  **强制解析**  当你确定可选类型确实包含值之后，你可以在可选的名字后面加一个感叹号（!）来获取值。这个感叹号表示"我知道这个可选有值，请使用它。"这被称为可选值的强制解析（forced unwrapping）。  实例如下：  import Cocoa  var myString:String?  myString = "Hello, Swift!"  if myString != nil {  print(myString)}else{  print("myString 值为 nil")}  以上程序执行结果为：  Optional("Hello, Swift!")  强制解析可选值，使用感叹号（!）：  import Cocoa  var myString:String?  myString = "Hello, Swift!"  if myString != nil {  // 强制解析  print( myString! )}else{  print("myString 值为 nil")}  以上程序执行结果为：  Hello, Swift!  注意： 使用!来获取一个不存在的可选值会导致运行时错误。使用!来强制解析值之前，一定要确定可选包含一个非nil的值。  **自动解析**  你可以在声明可选变量时使用感叹号（!）替换问号（?）。这样可选变量在使用时就不需要再加一个感叹号（!）来获取值，它会自动解析。  实例如下：  import Cocoa  var myString:String!  myString = "Hello, Swift!"  if myString != nil {  print(myString)}else{  print("myString 值为 nil")}  以上程序执行结果为：  Hello, Swift!  **可选绑定**  使用可选绑定（optional binding）来判断可选类型是否包含值，如果包含就把值赋给一个临时常量或者变量。可选绑定可以用在if和while语句中来对可选类型的值进行判断并把值赋给一个常量或者变量。  像下面这样在if语句中写一个可选绑定：  if let constantName = someOptional {  statements}  让我们来看下一个简单的可选绑定实例：  import Cocoa  var myString:String?  myString = "Hello, Swift!"  if let yourString = myString {  print("你的字符串值为 - \(yourString)")}else{  print("你的字符串没有值")}  以上程序执行结果为：  你的字符串值为 - Hello, Swift! |  |
| [Swift 常量](http://www.runoob.com/swift/swift-constants.html" \o "Swift 常量" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 字面量](http://www.runoob.com/swift/swift-literals.html" \o "Swift 字面量" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 运算符](http://www.runoob.com/swift/swift-operators.html" \o "Swift 运算符" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 条件语句](http://www.runoob.com/swift/swift-decision-making.html" \o "Swift 条件语句" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 循环](http://www.runoob.com/swift/swift-loops.html" \o "Swift 循环" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 字符串](http://www.runoob.com/swift/swift-strings.html" \o "Swift 字符串" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 字符](http://www.runoob.com/swift/swift-characters.html" \o "Swift 字符(Character)" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 数组](http://www.runoob.com/swift/swift-arrays.html" \o "Swift 数组" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 字典](http://www.runoob.com/swift/swift-dictionaries.html" \o "Swift 字典" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 函数](http://www.runoob.com/swift/swift-functions.html" \o "Swift 函数" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 闭包](http://www.runoob.com/swift/swift-closures.html" \o "Swift 闭包" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 枚举](http://www.runoob.com/swift/swift-enumerations.html" \o "Swift 枚举" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 结构体](http://www.runoob.com/swift/swift-structures.html" \o "Swift 结构体" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 类](http://www.runoob.com/swift/swift-classes.html" \o "Swift 类" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 属性](http://www.runoob.com/swift/swift-properties.html" \o "Swift 属性" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 方法](http://www.runoob.com/swift/swift-methods.html" \o "Swift 方法" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 下标脚本](http://www.runoob.com/swift/swift-subscripts.html" \o "Swift 下标脚本" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 继承](http://www.runoob.com/swift/swift-inheritance.html" \o "Swift 继承" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 构造过程](http://www.runoob.com/swift/swift-initialization.html" \o "Swift 构造过程" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 析构过程](http://www.runoob.com/swift/swift-deinitialization.html" \o "Swift 析构过程" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 可选链](http://www.runoob.com/swift/swift-optional-chaining.html" \o "Swift 可选链" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 自动引用计数](http://www.runoob.com/swift/swift-arc.html" \o "Swift 自动引用计数（ARC）" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 类型转换](http://www.runoob.com/swift/swift-type-casting.html" \o "Swift 类型转换" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 扩展](http://www.runoob.com/swift/swift-extensions.html" \o "Swift 扩展" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 协议](http://www.runoob.com/swift/swift-protocols.html" \o "Swift 协议" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 泛型](http://www.runoob.com/swift/swift-generics.html" \o "Swift 泛型" \t "http://www.runoob.com/swift/_top) |  |  |
| [Swift 访问控制](http://www.runoob.com/swift/swift-access-control.html" \o "Swift 访问控制" \t "http://www.runoob.com/swift/_top) |  |  |