swift基础

基础部分

问题，if（）中为何不要求是布尔类型；声明变量的话，默认的是普通可选还是隐式可选类型；

用不可变的变量表示常量
使用新数据类型元组（tuple），可当作返回类型
使用可选类型（optional）来表示值缺失的情况

声明变量或者常量的时候，需要给出初始值或者声明类型，这样系统才能推断出变量或者常量的类型，一旦确定就不可改变

println与NSLog类似，不过更强大一些，使用字符串插值就可以直接打印变量或者常量的值（添加反斜杠 \ () 就行）

可通过Int8.max或者Int8.min来获取最大或者最小值

尽量不要使用UInt

有类型推断，可以不显式类型注明

数值子面量可以添加额外的0或者下划线来增强可读性，如100_000

类型不同的数值不能进行运算，得先进行类型转换，如UInt16（xxx）

可以使用typealias 来定义类型别名

使用true和false来表示布尔值

以前使用a = 1，if（a）的条件判断是ok的，现在只能这样if（a = = 1），因为swift是类型安全的，一旦推断出类型不一致，则会报错无法进行

元组可由任意类型组成，并不要求是相同类型。
特性A：可将元组分解成单独的常量或者变量，这样就可以单独使用。如let（statuscode，statusmessage） = http404error
特性B：如只需要分解一部分元组，则可食用下划线（_）标记
特性C：可通过下标来访问元素
特性D：可在定义元组的时候给元素命名，这样就可像属性一样访问元素

可选类型是对OC特性nil的扩展，在oc方法中可能会返回具体对象或者nil，之前可以直接使用返回值调用方法，如果有值则正常，无值则什么都不干。在OC中这些特性只对对象有效，对结构体、基本的C类型和枚举类型不起作用。可选类型正是解决这个问题。

Int？表示可能有缺少值的情况
Int！表示我知道Int里有值，请放心使用，称为强制解析。

可选绑定是指将可能有缺少值的类型赋予变量或者常量，如果有值则赋值，无则什么都不干。可选绑定与强制解析比较特殊，可以在if（）中直接使用，如if（Int！ = nil），有值则true；如if（let x = Int？）有值则true

swift中nil是个明确的值，可赋予任何类型，OC中nil是指针，只可富裕对象。

变量声明类型的时候，可以根据是否含有缺少值声明为普通可选类型（？）和隐式可选类型（！）。如果声明为普通可选类型（？），则初始值就是nil，则需要在使用变量的时候使用可选绑定或者强制解析。如果隐式可选类型（！），则可以直接使用，但是如果该变量在取值的时候出现了为nil的情况则会报运行时错误。

基本运算符

if （x = y）｛｝，在swift中是错误的，因为 = 不返回值，赋值操作都不返回值。

swift运算符会防止溢出，另外提供需要溢出的运算符&，如加法a & + b；

求余运算符，a％b，a％－b，会对b的符号忽略，swift中还可以对浮点数取余。

提供了一元运算符“－”，取负数的操作，另外还提供了“＋”给强迫症

自增自减运算符有返回值

新增“＝＝＝”和“！＝＝”，表示为恒等于和恒不等于

新增空和运算符“？？”，是由于可选类型的引入，为了使用方便引入的运算符，可以表示为：
a != nil? a! : b

新增区间运算符 “..”，表示区间，很灵活，相应的还有半开区间符 “..<”

字符串和字符

字符串可以使用转义字符，也可以使用Unicode标量，如\u{xxx}

swift中的可变字符串与不可变字符串跟其他变量一样，通过var和let来指定

swift中的字符串是值类型，值类型在常量、变量赋值操作或者函数、方法中传递时都会进行值拷贝。实际编译时，swift编译器会智能识别是否真正需要拷贝。

字符串可以是字符的集合，还可以通过for－in来枚举字符

countElements()用来计算字符串的长度，并且是以Unicode字符来计算的。不同于NSString中的length是基于UTF－16来计算，对应的swift中使用方法utf16count来表示

字符串拼接可使用运算符“＋”与“＋＝”

字符串的比较可使用运算符“＝＝”，字符串可通过属性uppercaseString和lowercaseString来获取大小写

可以通过三种Unicode兼容的方式访问字符串的值：utf8属性、utf16属性、unicodeScalars的Unicode21位标量值属性。

集合类型

Swift中数组和字典中的存储的数据类型必须是明确的，并且只能存储一种类型，但不限于对象，这与OC不一样，OC只限制存入对象，对类型没有更多要求。Swift中存储的类型可通过显式标注或类型推断。

推荐使用［SomeType］的方式来使用数组，可使用isEmpty来查看数组是否为空，可使用append（）或者 + = 来拼接数组。

新特性，可以使用运算符“…”一次性改变一系列的数组值，并且不要求前后个数一致。

新特性，可以使用数组的enumerate函数来进行遍历，可以同时访问元素和下标。

可以通过［SomeType］(Count: , repeatedValue:)来构造特定大小并且数组一致的数组

字典需要有明确的KeyType和ValueType，并且要求KeyType是可hash的，目前所有的Swift数据类型都是可以hash的。

可通过dic［“Key”］ = nil来移除value

可通过元组来遍历

Swift的数组和字典底层都是通过泛型集合来实现的。

不可变的数组与字典原则上都是不可改变大小与内容的，但是在不可变数组中依然可以改变索引所对应的值。

控制流

Swift所提供的控制流与C一致，不过功能大大增强，特别是Switch Case。

for－in可使用运算符“…”来指定区间，还可以使用下划线“_”来表示忽略对值的访问，

Switch必须是完备的，并且会自动break，一个case可以使用“，”来添加多个条件，case还可以使用区间符号“…”，当case中有元组，还可以使用下划线“_”来匹配所有可能的值。可以在Case中进行值绑定，绑定之后则可在当前case中使用。case中还可以使用where来添加额外的条件判断。

每个case都必须要有一句，可使用break来跳过该case。

由于Swift每个case都会自动break，为了照顾需要原有switch case代码逻辑或者部分爱好者的习惯，添加“fallthrough”关键词来进行“贯穿”case。

可以将循环体或者switch case加上标签，通过break、continue来指定需要操作的对象，增加灵活性。

函数

很灵活。

可使用元组作为返回值。

可以有外部参数名，使用空格与本地参数名进行区分。如果只写一个参数名，则默认为本地参数名，如果写两个参数名（中间用空格隔开），则第一个是外部参数名，第二个是本地参数名。还可以使用简便的方法，在参数名前面添加“＃”，这样就可以写一个参数名，同时命名外部、本地参数名。

可为参数指定默认值，这样在调用的时候就可以忽略这个参数，使得函数定义更加灵活。Swift会自动给指定默认值的参数添加外部参数名。
外部参数名也为兼容OC代码提供优雅的方式。

一个函数可以使用一个可变参数，具体使用方法是：func aFuntion（numbers：Int…），就是在参数类型后面添加运算符“…”，可变参数必须为最后一个参数，目的是避免混淆。

参数默认是常量，不可修改，如果需要修改参数，则可在参数名前面添加关键词“var”

针对需要在函数内修改外部变量值的需求，Swift提供了“inout”关键词来实现，可在参数名前面添加“inout”，并且不能是可变参数和不能有默认值。在外部调用函数的时候，传入的参数需加上“&”来表示内部可能会改变该值，并且设定不能传入常量或者字面量。

通过参数类型和返回值类型来决定函数的类型，这样函数就可以被当作常量或者变量来使用。在定义函数的时候，同样可以指定类型或者进行类型推断。这种特性使得函数也可以当作参数来使用。新特性，可以定义嵌套函数。

闭包

闭合并包裹上下文的常量和变量，简称闭包。

闭包表达式｛（parameters）－> return type in statement｝

Swift中使用各种手段来达到闭包简洁的目的，示例：
1. 原函数
```
func backwards(s1:String , s2:String) -> Bool {  return s1>s2 }
var reversed = sorted(names, backwards )
```

2. 使用闭包改写
```
var reversed = sorted( names, {(s1: String , s2:String )  -> Bool in return s1>s2 })
```

3. 由于sorted函数参数类型明确为Bool，并且闭包中含有单一表达式，该表达式返回Bool，则没有歧义，可以隐去return
```
var reversed = sorted (names, {(s1:String , s2:String ) -> Bool in s1>s2 })
```

4. 由于Swift具有使用上下文进行推断的能力，所以参数类型定义可以隐去
```
var reversed = sorted(names, {(s1, s2) -> Bool in s1>s2})
```

5. Swift还为内联函数提供参数名缩写功能，并且关键词“in”也可以省去
```
var reversed = sorted(names, {$0>$1})
```

Swift中还提供了运算符函数，如字字符串大于号函数“>”，其参数为两个String，返回值为Bool，正好与上面的sorted函数第二个参数类型一致，利用此特性，甚至可以改写成：
```
var reversed = sorted(names, >) 尼玛，第一次看到得疯了 ,这么灵活，容易混乱吧
```

在OC中当闭包函数很长时，就会显得函数很不雅观，并且可读性不好。Swift中提供了尾随闭包正好解决这个问题。使用的条件是，闭包作为函数的最后一个参数。

Swift中不再需要手动设置关键词“__block ”” __weak”等，会自动根据上下文判断是拷贝还是引用。

闭包和函数都是引用类型，赋予变量或者常量的时候，是指赋予闭包，而非闭包内容。

枚举

很灵活。

枚举是First－Class类型，支持很多只被类所支持的特性。

不像OC，在Swift中在枚举被创建的时候，不会被隐式地赋予初始值。

枚举的概念已经和OC中大不一样了。

枚举示例:

```
enum CompassPoint {

    case North
    case West
    case South
    case East
}

enum Planet {

  case Mercury,Venus
}
```

引入枚举相关值的概念，就是与枚举case相关的值，可在使用case的时候直接使用相关值。
示例

```
enum Barcode  {

  case UPCA(Int ,Int, Int)
  case QRCode(String)
}
```

这里的枚举必然支持OC、C的枚举，就是提供原始值的概念，与此前差不多。

类与结构体

类与结构体有很多的相似点，所以这里放在一起讲。

都是通过构造器语法来创建实例。

属性访问还是使用点语法，并且在Swift中可以设置子属性。

结构体可以使用构造器对每个属性赋予初始值，类则不行。

结构体是值类型，在赋值或者传参中都会对值进行拷贝，枚举亦然。所有的基本数据类型都是值类型。

类是引用类型，为了判断不同的类示例是否引用同一个类示例，引入两个运算符“＝＝＝”和“！＝＝”，等价于与不等价于。a＝＝＝b与a＝＝b的区别，前者用与判断是否引用相同类实例，后者用于判断内容是否相等。如果a＝＝＝b，那么a＝＝b必定成立，反之不成立。所以“＝＝＝”也可以称为恒等于。

Swift中定义的常量或者变量引用一个引用类型的实例与C指针相似，但是并不像C中一样指向内存中的某个地址。

类与结构体的相似点有：可定义属性用来存储值，可定义方法提供功能，定义附属脚本访问值，定义构造器用于生成初始值，通过扩展增加默认实现的功能，符合协议以对某类提供标准功能。

类比结构多的功能有：可以继承，可以运行时检查和解释类示例的类型，析构器允许释放任何其分配的类型，有引用计数并可对一个类多次引用。

类与结构体的区别还有：类是引用类型，结构体是值类型；类没有默认的逐一构造器功能，只能通过添加自定义构造器实现，而结构体自带该功能。

可根据自己的需求选择是选择类还是结构体。

Swift中的字符串、数组、字典都是以结构体的形式实现的，而在OC中是以类对形式实现的。意味着在Swift中是值类型，赋值或传参都是值拷贝，OC中是引用类型，赋值或传参是引用操作。

属性

分为两大类：类型属性和实例属性。
实例属性还分为计算属性和存储属性。计算属性可用于类、结构体和枚举，存储属性可用于类和结构体。
针对不同实例中共享数据，新引入类型属性，区别与实例属性，不论类型有多少实例，都只有一份类型属性，而实例属性则互相独立不影响。值类型（结构体和枚举）可以定义存储型和计算型的类型属性，引用类型（类）则只能定义计算型的类型属性。

枚举不能存实例属性，类不能存类型属性。任何值类型和引用类型都可以使用计算属性。
使用关键词“static”来定义类型属性。

可以使用关键词“lazy”来延迟加载属性，属性必须得是变量，因为常量的属性必须在结构体或类构造完成前存在初始值。

Swift中简化了属性的实例变量与访问方法，其中的属性没有对应的实例变量，后端的存储也无法直接访问。

计算属性的概念为不存储值，提供一个getter来获取值，并提供可选的setter来间接设置其他属性或者变量的值。当只有getter的时候，可以去掉关键词“get”和“｛｝”。如下所示：
原实例：

```
struct AlternativeRect{

    var origin = Point()
    var center :Point {

        get{

        return (x:3,y:6)
        }

        set{
```

```
        origin.x = 3
        origin.y = 3
        }
}


struct AlternativeRect{

var origin = Point()
var center: Point{

return (x:3,y:6)
}

}
```

可使用willSet观察器和didSet观察器，willSet传入的是新值，didSet传入的是老值，如果在didSet中进行属性赋值，则会生效。注意，文档说明可以给延迟加载属性之外的属性添加观察器。

方法

结构体和枚举中也可以定义方法。

通过点语法调用方法

可以通过下划线“_”加在参数名前面，指定不要有外部参数名。

如果与参数名不冲突，则可以不指定self来访问属性。

结构体和枚举是值类型，一般无法在实例方法中改变其值。但是存在这种需求，引入关键词“mutating”来解决这个问题。所改变的结构体或枚举必须是变量，不能是常量。

给类添加类型方法，就是在方法前添加关键词“class”，而给结构体和枚举添加类型方法就是在方法前添加关键词“static”。

下标脚本

数组、字典中很有用，例如Array［5］，dic［@“key”］

通过下标脚本可以给自定义的类、结构体、枚举定义快速访问或者设置的下标。
示例：

```
subscript( index: Int ) —> Int {
        get{
        }
        set(newValue){

        }
}
```

下标参数一般都是一个参数，不过Swift并没有规定入参的个数。

继承

OC中创建的类必须有基类，在Swift中如果定义时不指定基类，那么自己就是基类。

如果需要重写，必须添加关键词“override”

与OC一致，通过super来访问父类的属性、方法或下标脚本

重写属性注意点：可将只读属性重写为读写属性，但是不能将读写属性重写为只读属性；提供了setter，就必须提供getter。

可添加关键词“final”来防止重写。

构造过程

可以设置不同的构造参数来定制构造过程

Swift会默认将构造器中的本地参数名转为外部参数名（如果没有设置外部参数名的话），此外可以在本地参数名前面添加关键词“_”来屏蔽系统的默认行为，意为不要外部参数名。

常量属性只能在构造器中修改，并且不能在子类中修改。

为了避免错误地初始化，如果定制了构造器，则不能访问到默认的构造器，或者逐一构造器。