## 程序流程控制

Swift提供了多种流程控制结构，包括可以多次执行任务的while循环，基于特定条件选择执行不同代码分支的if、guard和switch语句，还有控制流程跳转到其他代码位置的break和continue语句。

Swift 还提供了for-in循环代替传统三段式for循环，用来更简单地遍历数组（Array），字典（Dictionary），区间（Range），字符串（String）和其他序列类型。

### 3.1 for-in 循环

如果你有其他语言的基础，你一定会对大名鼎鼎的for循环非常熟悉。可是在Swift语言中，传统的三段式for循环已经被移除了，取而代之的是for-in循环。

看一个简单的示例，打印0到4

for index in 0...4 {

print(index)

}

//打印 0 1 2 3 4

for index in 0..<5 {

print(index)

}

//打印 0 1 2 3 4

示例中的(0...4)和(0..<5)被称作区间操作符，其中(0...4)是闭区间操作符，表达的是一个左闭右闭的区间，而(0..<5)是半开区间操作符，表达了一个左闭右开的区间。索引index第一次被赋值为区间中的第一个数字0，然后通过print语句打印出0这个数字。随后，index被赋值为区间中的下一个数字1，再次打印，依次类推，打印0到4这5个数字。

index是每次循环遍历开始时自动被赋值的常量。这种情况下，index不需要使用let关键字进行声明。

有时，我们并不需要index这个索引，此时我们可以用\_代替index，从而忽略这个值。例如，只需要打印3遍“Swift“，就可以这样实现。

for \_ in 0..<3 {

    print("Swift")

}

//打印3遍Swift

swift中的区间操作符一定要是一个合法的区间，也就是说左边界要小于等于右边界。如果我们要实现逆序打印该怎么做？可以调用reverse()方法。

for index in (0..<5).reversed() {

    print(index)

}

//打印 4 3 2 1 0

for-in循环还有一个重要用途，就是遍历数组、集合等集合类型，在第四章元组和集合类型还会用到for-in循环。

### 3.2 while 循环

while循环会一直运行一段语句直到条件变成false。while循环非常适合使用在第一次迭代前，迭代次数未知的情况下。

Swift提供两种while循环的形式：

1. while循环，在每次执行循环体之前判断条件是否符合。
2. repeat-while循环，在每次执行循环体之后判断条件是否符合。

#### 1.while循环

while循环先判断条件再执行语句。如果条件为true，会重复运行一段语句，直到条件变为false。依旧是打印0到4，换做while循环可以这样实现。

var index = 0

while index < 5 {

    print(index)

    index += 1

}

//打印 0 1 2 3 4

index初始时被赋值为0，第一执行循环体之前，判断index的值是否小于5，条件成立，执行print语句，随后执行+1，index的值变为2。在下一次执行循环体之前，再次判断，之后执行循环体。直到执行到第5次，index的值变成5，不再满足判断条件，while循环结束。

#### 2.repeat-while循环

repeat-while循环不像for-in和while循环在循环体开始执行之前判断条件语句，而是在循环体执行结束之后判断条件是否符合。

var index = 0

repeat {

    print(index)

    index += 1

} while index < 5

//打印 0 1 2 3 4

执行完一次循环体之后，如果条件为true，控制流会跳转回上面的repeat，然后重新执行循环体，直到条件变为false为止。

请注意，由于条件表达式出现在循环的尾部，所以循环体在条件被检测之前至少被执行一次。

var index = 0

repeat {

    print(index)

} while (index < 0)

//打印 0

尽管index初始值被设置为0，不满足小于0的条件，但是循环体还是会执行一次，打印出0。

### 3.3 条件语句

根据特定的条件执行特定的代码通常是十分有用的。当错误发生时，你可能想运行额外的代码；或者，当值太大或太小时，向用户显示一条消息。要实现这些功能，你就需要使用条件语句。

Swift 提供两种类型的条件语句：if语句和switch语句。通常，当条件较为简单且可能的情况很少时，使用if语句。而switch语句更适用于条件较复杂、有更多排列组合的时候。

#### 1.if-else

If语句最简单的形式就是只包含一个条件，只有条件为true时，才执行相关代码。

var varA:Int = 10

if varA < 20 {

    print("varA 小于 20")

}

//打印 varA 小于 20

我们可以通过配合else从句，实现二选一执行。也就是当条件为false的时候，执行else语句。

var varA:Int = 30

if varA < 20 {

    print("varA 小于 20")

} else {

    print("varA 大于等于 20")

}

//打印varA 大于等于 20

我们还可以通过多个if-else链接的方式，来实现更多的分支。

var varA:Int = 20

if varA < 20 {

    print("varA 小于 20")

} else if varA == 20{

    print("varA 等于 20")

} else {

    print("varA 大于 20")

}

//打印varA 等于 20

#### switch

switch语句会尝试把某个值与若干个模式（pattern）进行匹配。根据第一个匹配成功的模式，switch语句会执行对应的代码。当有可能的情况较多时，通常用switch语句替换if语句。

switch语句最简单的形式就是把某个值与一个或若干个相同类型的值作比较：

let someCharacter : Character = "A"

switch someCharacter {

case "A":

    print("这是字母A")

case "a":

    print("这是字母a")

default:

    print("其他字母")

}

//打印 这是字母A

switch语句由多个 case 构成，每个由case关键字开始。为了匹配某些更特定的值，Swift 提供了几种方法来进行更复杂的模式匹配，这些模式将在本节的稍后部分提到。

与if语句类似，每一个 case 都是代码执行的一条分支。switch语句必须是完备的。这就是说，每一个可能的值都必须至少有一个 case 分支与之对应。在某些不可能涵盖所有值的情况下，你可以使用默认（default）分支来涵盖其它所有没有对应的值，这个默认分支必须在switch语句的最后面。

与if不同的是，switch可以提供多种情况同时匹配时，执行多个语句块。

let someCharacter : Character = "A"

switch someCharacter {

case "a","e","i","o","u","A","E","I","O","U":

    print("\(someCharacter)是一个元音字母")

default:

    print("\(someCharacter)不是一个元音字母")

}

//打印 A是一个元音字母

在swift中，当匹配的case分支中的代码被执行完毕后，会自动终止switch语句，不会继续执行下一个case分支。这也就是说，不需要在 case 分支中显式地使用break语句。如果想要实现贯穿的效果，可以使用fallthrough语句。

### 3.4 控制转移语句

控制转移语句可以改变你代码的执行顺序，通过它们可以实现代码的跳转。Swift中主要有3种控制转移语句: 1.continue 2.break 3.fallthrough

1. continue

continue语句的作用是告诉一个循环体立刻停止本次循环，重新开始下次循环。结束的仅仅是一次循环，而不是整个循环。

var index = 10

repeat{

    index = index + 1

    if( index == 12 ){ // index 等于 12 时跳过

        continue

    }

    print(index)

}while index < 15

// 打印 11 13 14 15

可以看到当index等于12的时候，print语句被跳过了，而13 14 15还是照常打印，说明循环并未结束。

1. break

break语句的作用是中断当前循环，结束整个循环。

var index = 10

repeat{

    index = index + 1

    if( index == 12 ){ // index 等于 12 时跳过

        break

    }

    print(index)

}while index < 15

//打印 11

我们仅仅改动了一行代码，将continue改为break，输出结果就大相径庭。这里仅仅打印了11是因为当index等于12的时候，整个循环都被终止了，12到15都不会被打印。

1. fallthrough

fallthrough 语句可以让 case 之后的语句按顺序继续运行，且不论条件是否满足都会执行。

在大多数语言中，switch 语句块中，case 要紧跟 break，否则 case 之后的语句会顺序运行，而在 Swift 语言中，默认是不会执行下去的，switch 也会终止。如果你想在 Swift 中让 case 之后的语句会按顺序继续运行，则需要使用 fallthrough 语句。

如果switch中没有fallthrough语句：

let index = 10

switch index {

case 100  :

    print( "index 的值为 100")

case 10,15  :

    print( "index 的值为 10 或 15")

case 5  :

    print( "index 的值为 5")

default :

    print( "默认 case")

}

//打印 index 的值为 10 或 15

当我们在某些case中添加fallthrough语句：

let index = 10

switch index {

case 100  :

    print( "index 的值为 100")

    fallthrough

case 10,15  :

    print( "index 的值为 10 或 15")

    fallthrough

case 5  :

    print( "index 的值为 5")

default :

    print( "默认 case")

}

/\*

打印： index 的值为 10 或 15

      index 的值为 5

\*/

从输出中，可以看到，当case 10,15对应的语句块被执行以后，case 5对应的语句块也被执行了。

### 3.5 提前退出(guard)

guard语句和if语句有点类似，都是根据其关键字之后的表达式的布尔值决定下一步执行什么。但与if语句不同的是，guard语句只会有一个代码块，不像if语句可以if else多个代码块。

guard，顾名思义，就是守护。guard语句判断其后的表达式布尔值为false时，才会执行之后代码块里的代码，如果为true，则跳过整个guard语句，guard像守卫一样，在代码中设置检查点，当不满足条件时，程序执行就会被拦截下来。示例代码：

func check(age : Int){

    guard age <= 6 else {

        print("对不起，你已经不是宝宝了！")

        //执行return语句直接退出

return

    }

    print("欢迎来到儿童乐园！")

}

check(age: 19)

//输出：对不起，你已经不是宝宝了！

check(age: 4)

//输出：欢迎来到儿童乐园！

如果guard语句条件被满足，不执行else语句块。将变量或常量的可选绑定作为guard语句的条件，可以保护guard花括号后面的代码，这也就是guard的含义的由来。

如果条件不满足，则会执else语句块，其中必须包含控制转移语句或者调用一个不返回的方法或函数，例如fatalError()。

相比与实现同样功能的if语句，guard语句可以提高代码的可读性，你可以在else语句块中，进行异常处理。