## 第四章 元组和集合类型

### 4.1 元组

元组（tuple）把多个值组合成一个复合值。元组内的值可以使任意类型，并不要求是相同类型。如果定义一个学生包含2个属性姓名，年龄，可以用元组表示为：

let student = ("Swift", 19)

//student 的类型是 (String, Int)，值是 ("Swift", 19)

示例中，我们定义了一个(String, Int)元组，String表示姓名，Int表示年龄，所以这个元组可以被描述为“一个类型为(String, Int)的元组”。元组中的数据类型可以是任意类型，根据你的需求，你可以将元组定义成(Int, Int, Int)或是(String, Bool)或是其他任何你想要的组合。

你可以将元组的内容分解(decompose)成单独的变量和常量，然后正常使用它们了。

let (name, age) = student

print("name is \(name)")

//打印 “name is Swift”

print("age is \(age)")

//打印 “age is 19”

如果你只需要元组的一部分值，分解的时候可以用\_代替哟啊忽略的部分。

let (onlyName, \_) = student

print("name is \(onlyName)")

//打印 “name is Swift”

此外，你还可以通过下标直接访问元组的单个元素，下标从零开始。

print("name is \(student.0)")

//打印 “name is Swift”

print("age is \(student.1)")

//打印 “age is 19”

如果，在元组定义的时候给单个元素命名，之后调用的时候，就可以用名字来调用这些元素的值。

//定义时给元素命名

let otherStudent = (name:"iOS", age:18)

//直接用.名称调用

print("name is \(otherStudent.name)")

//打印 “name is iOS”

print("age is \(otherStudent.age)")

//打印 “age is 18”

作为函数返回值时，元组非常有用。一个用来获取网页的函数可能会返回一个(Int, String)元组来描述是否获取成功。和只能返回一个类型的值比较起来，一个包含两个不同类型值的元组可以让函数的返回信息更有用。

但是，提醒一下各位读者，虽然元组在临时组织值的时候很有用，但是并不适合创建复杂的数据结构。如果你的数据结构并不是临时使用，请使用类或者结构体而不是元组。

### 4.2 集合类型和可变性

Swift 语言提供Arrays、Sets和Dictionaries三种基本的集合类型用来存储集合数据。数组（Arrays）是有序数据的集。集合（Sets）是无序无重复数据的集。字典（Dictionaries）是无序的键值对的集。

Swift 语言中的Arrays、Sets和Dictionaries中存储的数据值类型必须明确。这意味着我们不能把不正确的数据类型插入其中。同时这也说明我们完全可以对取回值的类型非常自信。

如果创建一个Arrays、Sets或Dictionaries并且把它分配成一个变量，这个集合将会是可变的。这意味着我们可以在创建之后添加更多或移除已存在的数据项，或者改变集合中的数据项。如果我们把Arrays、Sets或Dictionaries分配成常量，那么它就是不可变的，它的大小和内容都不能被改变。在我们不需要改变集合的时候创建不可变集合是很好的实践。如此 Swift 编译器可以优化我们创建的集合。

### 4.3 数组(Arrays)

**数组的创建**

数组使用有序列表存储同一类型的多个值。相同的值可以多次出现在一个数组的不同位置中。Swift 数组会强制检测元素的类型，如果类型不同则会报错，Swift 数组应该遵循像Array<Element>这样的形式，其中Element是这个数组中唯一允许存在的数据类型。我们也可以使用像[Element]这样的简单语法。尽管两种形式在功能上是一样的，但是推荐较短的那种。

//创建一个空数组

var someInts = [Int]()

print("数组中有\(someInts.count)个元素")

//打印 "数组中有0个元素"

Swift 中的Array类型还提供**一**个可以创建特定大小并且所有数据都被默认的构造方法。我们可以把准备加入新数组的数据项数量（count）和适当类型的初始值（repeating）传入数组构造函数

//创建一个[Double]数组，等价于[0.0, 0.0, 0.0]

var threeDoubles = Array(repeating: 0.0, count: 3)

我们还可以使用数组字面量来进行数组构造，这是一种用一个或者多个数值构造数组的简单方法。数组字面量是一系列由逗号分割并由方括号包含的数值：[value 1, value 2, value 3]

//定义包含3个元素的[String]数组

var shoppingList = ["Eggs", "Milk", "Bread"]

**数组的常用属性和方法**

Swift中提供可一些常用的属性和方法让我们可以更加方便的使用数组。

//返回数组中元素的数量

let count = shoppingList.count

//判定数组是否为空

let isEmpty = shoppingList.isEmpty

//判定数组中是否存在某个元素

let contains = shoppingList.contains("Eggs")

//返回特定元素的索引，如果有多个重复元素，返回第一个的下标

let index = shoppingList.index(of: "Eggs")

//返回数组的第一个或最后一个元素

let firstValue = shoppingList.first

let lastValue = shoppingList.last

**数组的遍历**

我们可以使用上一章介绍的for-in循环来遍历数组中的数据项：

for index in 0..<shoppingList.count{

    print(shoppingList[index])

}

//打印 Eggs Milk Bread

或是使用下面这种更加简洁的形式遍历数组中的每个数据项

for item in shoppingList{

    print(item)

}

//打印 Eggs Milk Bread

如果我们同时需要每个数据项的值和索引值，可以使用enumerated()方法来进行数组遍历。enumerated()返回一个由每一**个数据项**索引值和数据值组成的元组。我们可以把这个元组分解成临时常量或者变量来进行遍历：

//利用元组访问

for (index, item) in shoppingList.enumerated() {

    print("item\(index) \(item)")

}

/\*

 打印：

 item0 Eggs item1 Milk item2 Bread

\*/

**数组的修改**

在Swift中，可以使用append()或者赋值运算符+=在数组的末尾添加元素。

//为shoppingList添加元素"Tea"

shoppingList.append("Tea")

//+=将["Flowers","Orange"]添加到shoppingList末端

shoppingList += ["Flowers","Orange"]

现在shoppingList中就有6个元素了。分别是"Eggs","Milk","Bread"

,"Tea","Flowers","Orange"。如果需要在数组的中间位置添加新的元素，可以使用insert()方法，新的元素被插入数组指定的index的位置，该位置原来的元素将依次右移。

//在索引为2的位置插入元素

shoppingList.insert("Pears", at: 2)

当然，我们也可以通过下标的方式访问元素和修改元素。

//打印下标为0的元素

print(shoppingList[0])

//打印 Eggs

//把第0个元素修改为apples,再次打印

shoppingList[0] = "apples"

print(shoppingList[0])

//打印 apples

Swift数组提供了几种remove方法，用来删除数组中的元素。

//删除指定索引的元素

shoppingList.remove(at: 1)

//删除第一个元素

shoppingList.removeFirst()

//删除最后一个元素

shoppingList.removeLast()

//删除区间内的元素

shoppingList.removeSubrange(Range(0...1))

//删除所有元素

shoppingList.removeAll()

### 4.4 集合(Sets)

集合(Set)用来存储相同类型并且没有确定顺序的值。当集合元素顺序不重要时或者希望确保每个元素只出现一次时可以使用集合而不是数组。

**集合的创建**

Swift 中的Set类型被写为Set<Element>，这里的Element表示Set中允许存储的类型，和数组不同的是，集合没有等价的简化形式。

var letters = Set<Character>()

print("集合中有\(letters.count)个元素")

//打印 集合中有0个元素

与数组类似，我们也可以使用数组字面量来构造集合，并且可以使用简化形式写一个或者多个值作为集合元素。

// favoriteGenres 被构造成含有三个初始值的集合

var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]

一个Set类型不能从数组字面量中被单独推断出来，因此Set类型必须显式声明。然而，由于 Swift 的类型推断功能，如果你想使用一个数组字面量构造一个Set并且该数组字面量中的所有元素类型相同，那么你无须写出Set的具体类型。favoriteGenres的构造形式可以采用简化的方式代替：

var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]

由于数组字面量中的所有元素类型相同，Swift 可以推断出Set<String>作为favoriteGenres变量的正确类型。

**集合的遍历**

你可以在一个for-in循环中遍历一个Set中的所有值。

//for-in遍历favoriteGenres

for genre in favoriteGenres {

    print(genre)

}

**集合的修改**

你可以通过调用Set的insert(\_:)方法来添加一个新元素：

favoriteGenres.insert("Jazz")

//favoriteGenres 现在包含4个元素

你可以通过调用Set的remove()方法去删除一个元素，如果该值是该Set的一个元素则删除该元素并且返回被删除的元素值，否则如果该Set不包含该值，则返回nil。另外，Set中的所有元素可以通过它的removeAll()方法删除。

if let removedGenre = favoriteGenres.remove("Rock") {

    print("成功删除\(removedGenre)")

} else {

    print("集合中不包含该元素")

}

//打印 成功删除Rock

**集合的重组和组合**

你可以高效地完成Set的一些基本操作，比如把两个集合组合到一起，判断两个集合共有元素，或者判断两个集合是否全包含，部分包含或者不相交。常用方法：

intersection(\_:)方法根据两个集合中都包含的值创建的一个新的集合。

symmetricDifference(\_:)方法根据在一个集合中但不在两个集合中的值创建一个新的集合。

union(\_:)方法根据两个集合的值创建一个新的集合。

subtracting(\_:)方法根据不在该集合中的值创建一个新的集合。

let oddDigits: Set = [1, 3, 5, 7, 9]

let evenDigits: Set = [0, 2, 4, 6, 8]

let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]

oddDigits.union(evenDigits).sorted()

// [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

oddDigits. intersection(evenDigits).sorted()

// []

oddDigits.subtracting(singleDigitPrimeNumbers).sorted()

// [1, 9]

oddDigits. symmetricDifference(singleDigitPrimeNumbers).sorted()

// [1, 2, 9]

### 4.5字典

字典是一种存储多个相同类型的值的容器。每个值（value）都关联唯一的键（key），键作为字典中的这个值数据的标识符。和数组中的数据项不同，字典中的数据项并没有具体顺序。我们在需要通过标识符（键）访问数据的时候使用字典，这种方法很大程度上和我们在现实世界中使用字典查字义的方法一样。

**字典的创建**

Swift 的字典使用Dictionary<Key, Value>定义，其中Key是字典中键的数据类型，不可重复，Value是字典中对应于这些键所存储值的数据类型。我们也可以用[Key: Value]这样简化的形式去创建一个字典类型。