# 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。如果定义一个学生，包含2个属性姓名和年龄，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，定义了一个类型为(String, Int)的元组，String表示姓名，Int表示年龄。元组中的值可以是任意类型，根据实际场景下的不同需求任意搭配，比如将元组定义成(String,String,String)用来表示姓名，性别，地址或是(String, Double)用来表示姓名和成绩或是其他任何你想要的组合。

在使用元组之前，需要进行元组分解(decompose)，将元组的内容分解成单独的常量和变量，之后就可以单独使用。

|  |
| --- |
| let (name, age) = student  print("name is \(name)")  //输出: “name is Swift”  print("age is \(age)")  //输出: “age is 19” |

如果只需要元组的一部分值，分解的时候可以用\_代替被忽略的部分。

|  |
| --- |
| let (onlyName, \_) = student  print("name is \(onlyName)")  //输出: “name is Swift” |

此外，还可以通过下标直接获得元组的单个数据项，下标从零开始。

|  |
| --- |
| print("name is \(student.0)")  //输出: “name is Swift”  print("age is \(student.1)")  //输出: “age is 19” |

如果，在元组定义的时候给单个数据项命名，之后就可以用名称直接调用。

|  |
| --- |
| //定义时给数据项命名  let otherStudent = (name:"iOS", age:18)  //直接用.名称调用  print("name is \(otherStudent.name)")  //输出: “name is iOS”  print("age is \(otherStudent.age)")  //输出: “age is 18” |

作为函数返回值时，元组非常有用。例如，一个用来获取网页的函数可能会返回一个(Int, String)元组，分别代表状态码和信息，用来描述网页 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更好地返回所需要的信息。

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 集合类型和集合可变性

Swift 语言提供数组(Array)、集合(Set)和字典(Dictionary)三种基本的集合类型用来存储集合数据。数组（Array）是有序数据的集。集合（Set）是无序无重复数据的集。字典（Dictionary）是无序的键值对的集。Swift 语言中的Arrays、Sets和Dictionary中存储的数据值类型必须明确。这意味着不能把不相同的数据类型插入其中。同时从集合中取出的数据项的数据类型也是固定的。

如果创建一个Array、Set或Dictionary并且把它声明成一个变量，这个集合就是可变的。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。如果把Array、Set或Dictionary声明成常量，那么它就是不可变的，它的大小和内容都不能被改变。在不需要对集合做出改变的情况，最好声明成常量，因为编译器会对它进行优化。

## 4.3 数组(Array)

### 4.3.1 数组的创建

数组使用有序列表存储同一类型的多个数据项，每个数据项都有一个编号，编号从0开始依次递增至数组长度减一，并且相同的值可以多次出现在一个数组的不同位置中。Swift 数组会强制检测数据项的类型，如果类型不同会报错。Swift中使用Array<Element>这样的形式来定义数组，其中Element是这个数组中唯一允许存在的数据类型，也可以使用[Element]这样的简单语法，两种形式在功能上是一样的。

|  |
| --- |
| //创建一个空数组  var intArrary = Array<Int>()  var someInts = [Int]()  print("数组中有\(someInts.count)个数据项")  //输出: "数组中有0个数据项" |

Swift 中的Array类型还提供一个构造方法用于创建只包含重复数据的数组。repeating声明想要重复的数据项，count则声明需要重复的次数。

|  |
| --- |
| //创建一个[Double]数组，等价于[0.0, 0.0, 0.0]  var threeDoubles = Array(repeating: 0.0, count: 3) |

可以使用数组字面量来进行数组的初始化，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]，这种情况下，编译器能自动推导出数组中的数据类型，无需显示声明。

|  |
| --- |
| //定义包含3个数据项的[String]数组  var shoppingList = ["Eggs", "Milk", "Bread"] |

### 4.3.2 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。

|  |
| --- |
| //返回数组中数据项的数量  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数据项  let contains = shoppingList.contains("Eggs")  //返回特定数据项的下标，如果有多个重复数据项，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数据项  let firstValue = shoppingList.first  let lastValue = shoppingList.last |

### 4.3.3 数组的遍历

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 shoppingList.indices{      print(shoppingList[index])  }  //输出: Eggs Milk Bread |

indices是Array类型的属性，这个属性是一个Range范围，代表着数组下标的范围。利用它可以可以方便的获取下标，访问数组中的数据项。

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)  }  //输出: Eggs Milk Bread |

如果同时需要每个数据项的值和下标值，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下标值和数据值组成的元组。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)")  }  /\*   输出:：   item0 Eggs item1 Milk item2 Bread  \*/ |

### 4.3.4 数组的读取和修改

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

|  |
| --- |
| //为shoppingList添加数据项"Tea"  shoppingList.append("Tea")  //+=将["Flowers","Orange"]添加到shoppingList末端  shoppingList += ["Flowers","Orange"]  //当前数组中的元素:Eggs Milk Bread Tea Flowers Orange |

如果需要在数组的指定位置添加新的数据项，可以使用insert()方法，新的数据项被插入数组指定的index位置，该位置原来的数据项以及后面的数据项将依次右移。

|  |
| --- |
| //在下标为2的位置插入数据项  shoppingList.insert("Pears", at: 2) |

当然，也可以通过下标的方式访问数据项和修改数据项。

|  |
| --- |
| //输出:下标为0的数据项  print(shoppingList[0])  //输出: Eggs  //把第0个数据项修改为apples,再次输出:  shoppingList[0] = "Apples"  print(shoppingList[0])  //输出: Apples |

Swift数组提供了几种remove方法，用来删除数组中的数据项。

|  |
| --- |
| //删除指定下标的数据项  shoppingList.remove(at: 1)  //删除第一个数据项  shoppingList.removeFirst()  //删除最后一个数据项  shoppingList.removeLast()  //删除区间内的数据项  shoppingList.removeSubrange(Range(0...1))  //删除所有数据项  shoppingList.removeAll() |

## 4.4 集合(Set)

集合(Set)用来存储相同类型并且没有顺序的值。 集合中不能够存储重复的数据项。

### 4.4.1 集合的创建

Swift 中的Set类型被定义为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)个数据项")  //输出: 集合中有0个数据项 |

与数组类似，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数据项。

|  |
| --- |
| // favoriteGenres 被构造成含有三个初始值的集合  var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"] |

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，如果想使用一个数组字面量构造一个Set，那么无须写出Set中的数据项的具体类型。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数据项类型相同，Swift 可以推断出Set<String>作为favoriteGenres变量的正确类型。

### 4.4.2 集合的遍历

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)  }  //输出: Rock Classical Hip hop |

### 4.4.3 集合的修改

可以通过调用Set的insert(\_:)方法来添加一个新数据项。

|  |
| --- |
| favoriteGenres.insert("Jazz")  //favoriteGenres 现在包含4个数据项 Rock Classical Hip hop Jazz |

可以通过调用Set的remove()方法去删除一个数据项，如果该值是该Set的一个数据项则删除该数据项并且返回被删除的数据项值，如果该Set不包含该值，则返回nil。另外，Set中的所有数据项可以通过它的removeAll()方法删除。

|  |
| --- |
| if let removedGenre = favoriteGenres.remove("Rock") {      print("成功删除\(removedGenre)")  } else {      print("集合中不包含该数据项")  }  //输出: 成功删除Rock |

### 4.4.4 集合的重组和组合

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，判断两个集合共有数据项，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（Dictionary）

在查阅英汉字典的时候，通过英文单词可以找到对应的中文解释。这种通过键（英文单词）查找值（中文解释）的存储模式称为键值对映射，Swift中的字典(Dictionary)就是利用这一模式。

字典(Dictionary)存储键值对，字典中的值(Value)可以有重复，而键(Key)必须是唯一的，每个值都有一个对应的键、在使用字典时是通过键去确定值，只有键唯一才能确保找到一个确定的值，当然如果开发者尝试设置重复的键，编译器会报错。

### 4.5.1 字典的创建

Swift 的字典使用Dictionary<Key, Value>来定义，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。也可以用[Key: Value]这样简化的形式去创建一个字典类型。

|  |
| --- |
| var diallingCode = [String: String]()  // diallingCode 是一个空的 [String: String] 字典 |

也可以使用字典字面量来构造字典，这和刚才介绍过的数组字面量拥有相似语法。字典字面量是一种将一个或多个键值对写作Dictionary集合的快捷方式。

一个键值对是一个key和一个value的结合体。在字典字面量中，每一个键值对的键和值都由冒号(:)分割。这些键值对构成一个列表，其中这些键值对由方括号包含、由逗号分割。

|  |
| --- |
| var diallingCodes = ["010":"北京", "021":"上海", "0571":"杭州"] |

### 4.5.2 字典的遍历

可以使用for-in循环，来遍历某个字典的键或值，还可以通过元组同时遍历字典中的键值对。

|  |
| --- |
| //遍历diallingCodes的keys  for key in diallingCodes.keys {      print(key)  }  //输出: 010 021 0571  //遍历diallingCodes的values  for value in diallingCodes.values {      print(value)  }  //输出: 北京 上海 杭州  for (key,value) in diallingCodes {      print("\(key) : \(value)")  }   //输出: 021 : 上海 010 : 北京 0571 : 杭州 |

### 4.5.3 数组的读取和修改

可以使用下标语法，通过key获取字典中对应的value。

|  |
| --- |
| //输出: key "010" 对应的value  let cityName = diallingCodes["010"]  print(cityName!)  //输出: 北京 |

通过下标语法，还可以往字典中添加新的数据项。

|  |
| --- |
| //向字典中添加 key="020", value="广东"  diallingCodes["020"] = "广东"  print(diallingCodes["020"]!)  //输出: 广东 |

如果下标中的key已经存在于字典中，新设置的value会覆盖旧value。

|  |
| --- |
| //再次设置key="020"的value  diallingCodes["020"] = "深圳"  print(diallingCodes["020"]!)  //输出: 深圳 |

也可以使用字典的updateValue()方法，修改指定key对应的value。

|  |
| --- |
| //利用updateValue更新key="020"对应的value  diallingCodes.updateValue("武汉", forKey:"020" )  print(diallingCodes["020"]!)  //输出: 武汉 |

当需要删除字典的某对键值时，可以使用removeValue()，或者将某个key对应的value设置成nil，也能起到相同的作用。以下两行代码，都能在字典中删除key为“010“的键值对。

|  |
| --- |
| //某个值设置为nil  diallingCodes["010"] = nil  //调用removeValue  diallingCodes.removeValue(forKey: "010")  Swift还提供了remoeAll()方法，用于一次性删除字典中的所有键值对。  diallingCodes.removeAll()  print("字典中还剩\(diallingCodes.count)数据项")  //输出: 字典中还剩0数据项 |

## 4.6 本章小结

本章向读者介绍了元组，它作为临时的数据结构是个不错的选择。接下来介绍了Swift中的3种集合类型，分别是数组(Array)、集合(Set)和字典(Dictionary)，重点是介绍这些数据类型的定义、遍历、修改和常用方法，它们作为开发中的常用工具，是非常重要的，需要熟练掌握。