和Array相关的基础知识

Back to series (/series/collection-types)

字号
字号
默认主题
金色主题
暗色主题

对任何一门现代化编程语言来说，集合类型都是非常重要的组成部分。这一类数据类型的设计，很大程度上决定了开发者对某种编程语言的使用体验以及代码执行效率。因此，Swift在集合类型的设计和实现上，进行了诸多的考量，让它兼具易用性、高性能以及扩展性。

但是这样做也是有代价的，在Swift里，集合是个复杂的类型家族，它是由多个 `protocol` 形成的，因此当我们想深入其中一探究竟的时候，并不那么容易。

在这一章中，我们的主要内容将围绕Swift标准库中各种集合类型的惯用方法和设计初衷。当然，我们还会有一系列专门的内容，向大家介绍Swift集合类型背后的实现方式。作为开始，我们来看最简单也最常用的一个集合类型： `Array` 。

如何创建一个Array?

`Array` 表示一组有序（ordered）的数据集合，所谓有序，并不是指大小排序，而是指 `Array` 中的元素有先后的位置关系，稍后我们会看到，这个位置关系可以用来访问 `Array` 中的元素。在此之前，先来看了解如何定义 `Array` 对象。

首先，我们可以通过下面三种方法定义一个空的 `Array` ：

```
var array1: Array<Int> = Array<Int>()
var array2: [Int] = []
var array3 = array2
```

在上面的代码中，前两种使用了type annotation， `Array<Int>` 和 `[Int]` 没有区别，你可以根据自己的喜好来选择。而第三种，我们直接使用了一个空的 `Array` 生成了一个新的 `Array` 对象。

其次，再来看一些定义数组时同时指定初始值的方法：

```
// [3, 3, 3]
var threeInts = [Int](repeating: 3, count: 3)
// [3, 3, 3, 3, 3, 3]
var sixInts = threeInts + threeInts
// [1, 2, 3, 4, 5]
var fiveInts = [1, 2, 3, 4, 5]
```

它们用起来都很直观，在稍后我们提到 `Sequence` 时，还会看到更复杂的 `Array` 初始化方法。

两个常用的Array属性

定义好数组之后，我们介绍两个 `Array` 最常用的属性。第一个是 `count` ，类型是 `Int` 。我们之前已经用过，用于获取数组中元素的个数：

```
array1.count    // 0
fiveInts.count  // 5
```

第二个是 `isEmtpy` ，类型是 `Bool` 。表示数组是否为空：

```
if array2.isEmpty {
    // array2 is empty
    print("array2 is empty")
}
```

访问Array中的元素

接下来，我们看访问 `Array` 元素的方法，它们之中有我们在其他语言中熟悉的，也有Swift独特的方式。首先，就是几乎所有语言都有的惯用法，使用索引。但是，它却也是在Swift，最不被推荐的使用方法：

```
fiveInts[2] // 3
fiveInts[5] // This will crash
```

就像，上面例子中这样。当使用索引访问数组元素时，**你必须自行确保索引的安全性**。如果索引超过了数组的范围，程序就会直接崩溃。其实，在Swift里，我们几乎不需要直接使用索引来访问数组元素。稍后，我们会专门提到 Array 的惯用法。因此，Swift开发者也没有对索引访问添加任何安全保护。言外之意就是，非要用，你自己对结果全权负责喽。

其次，是使用range operator访问数组的一个范围：

```
fiveInts[0...2] // [1, 2, 3]
fiveInts[0..<2] // [1, 2]
```

要说明的是，使用range operator得到的，并不是一个 Array ，而是一个 ArraySlice 。什么是 ArraySlice 呢？简单来说，就是 Array 某一段内容的view，它不真正保存数组的内容，只保存这个view引用的数组的范围：

```
// +---------+---+
// | length  | 5 |
// +---------+---+
// | storage ptr |
// +---------+---+
//           |
//           v
//           +---+---+---+---+---+---------------------+
//           | 1 | 2 | 3 | 4 | 5 |  reserved capacity  |
//           +---+---+---+---+---+---------------------+
//           ^
//           |
// +---------+---+
// | storage ptr |
// +---------+---+
// | beg idx | 0 |
// +---------+---+
// | end idx | 3 |  ArraySlice for [0...2]
// +---------+---+
```

从上面这个注释，就很容易理解view的概念了，它只记录了要表达内容的区间。但是我们也可以直接通过这个view创建新的 Array 对象：

```
Array(fiveInts[0...2]) // [1, 2, 3]
```

这样，就得到了一个值是 [1, 2, 3] 的 Array 对象。

遍历数组

除了访问单个元素外，另一类常用的需求就是顺序访问数组中的每个成员。在Swift里，我们有三种基本的方法遍历一个 Array 。

首先，当然是我们最熟悉的 for 循环的方式：

```
for value in fiveInts {
    print(value)
}
// 1
// 2
// ...
```

其次，如果我们想在遍历的时候，同时获得数组的索引和值，可以使用数组对象的 enumerated() 方法，它会返回一个 Sequence 对象，包含了每个成员的索引和值，我们同样可以在 for 循环中，依次访问它们：

```
for (index, value) in fiveInts.enumerated() {
    print("\(index): \(value)")
}
// 0: 1
// 1: 2
// ...
```

第三，借助closure，我们还可以使用 Array 对象的 forEach 方法：

```
fiveInts.forEach { print($0) }
```

这和我们第一个例子中的结果是一样的，如果你还不了解这种用法也没关系，我们会在后面的视频中，专门讲到closure的话题。

添加和删除元素

最后，来看如何编辑 Array 中的元素。要在数组的末尾添加元素，我们可以这样：

```
array1.append(1)     // [1]
array1 += [2, 3, 4]  // [1, 2, 3, 4]
```

要在 Array 中间位置添加元素，可以使用 insert 方法：

```
// [1, 2, 3, 4, 5]
array1.insert(5, at: array1.endIndex)
```

它的第一个参数表示要插入的值，第二个参数表示要插入的位置，这个位置必须是一个合法的范围，即 `0...array1.endIndex` ，如果超出这个范围，会直接引发运行时错误。

要删除 Array 中的元素，可以使用 `remove(at:)` 方法，它只接受一个参数，表示要删除元素的位置：

```
array1.remove(at: 4) // [1, 2, 3, 4]
```

同样，你必须自行保证使用的 `at` 参数不超过数组的合法范围，否则会引发运行时错误。当然，如果你仅仅想删除数组中的最后一个元素，还可以使用 `removeLast()` 方法：

```
array1.removeLast() // [1, 2, 3]
array2.removeLast() // This will crash!!!
```

从上面的例子可以看到，你同样要对 `removeLast()` 的应用安全负责，当你删除一个空数组中最后一个元素的时候，会直接引发运行时错误。

What's next?

以上，就是和 Array 相关的最基本用法。了解它们，仅仅是我们开始认识Swift Array 类型的一个开始。进一步深入之前，下一节，我们先来看下 Array 和它的Objective-C兄弟 NSArray 用法的区别。

◀ 返回视频
(/series/collection-types)

理解Array和NSArray的差异 ▶
(https://www.boxueio.com/series/collection-types/ebook/125)