集合类型背后的“轮子”

自定义一个阳春白雪的Collection (https://www.boxueio.com/series/advanced-collections/ebook/163)

实现一个Swift“风味”的链表集合 (https://www.boxueio.com/series/advanced-collections/ebook/165)

理解Collection中的associatedtype

Back to series (/series/advanced-collections)

字号 字号 默认主题 金色主题 暗色主题

为了抽象集合类型的行为，在 Collection protocol里，定义了4个 associatedtype 。Swift为它们设置了默认的类型，并基于这些类型，为其约束的方法提供了默认的实现。在这一节里，我们就来进一步了解下这些预定义的 associatedtype ，这有助于我们在实现更多自定义集合类型时，做到心中有数，游刃有余。

从Indexable开始

第一个要介绍的，是 Indexable ，它是 Collection 类型约束的一部分：

```
public protocol Collection : Indexable, Sequence {
    /// ...
}
```

简单来说，就是要求集合类型一定是个可以被某种类型的对象索引的。在Swift 3里，它的类型是这样的：

```
public protocol Indexable : IndexableBase {
    /// ...
}

public protocol IndexableBase {
    associatedtype Index : Comparable

    public var startIndex: Self.Index { get }
    public var endIndex: Self.Index { get }
}
```

在 Indexable 里，定义了一些可以移动当前索引位置的方法，例如我们之前用过的 formIndex 。而在 IndexableBase 里，则定义了我们在上一节中实现集合类型时定义的 startIndex 和 endIndex ，它们表示了集合的起始和结束位置。以及，一个 associatedtype Index ，它表示了用于索引集合的元素的类型。

这个类型，比较容易理解的就是 Int ，例如， Array 和我们之前实现的 FIFOQueue ，直接用一个内存地址的偏移值，就可以表示特定元素的位置了。但对于一些非连续内存存储的结构， Index 的定义就会复杂得多，在后面的内容里，我们会自己从头实现一个这样的例子。现在只要知道它表达的含义就好了。

理解了 Collection.Index 之后，我们就可以进一步理解 Collection 的4个 associatedtype 了，它们之中的一些，和 Index 密切相关。

Collection的4个associatedtype

首先，回顾下这4个 associatedtype 的定义，别被定义它们的长度吓住，其实绝大多数内容，都是我们可以理解的。

```
public protocol Collection : Indexable, Sequence {
    // Associated types
    associatedtype IndexDistance : SignedInteger = Int
    associatedtype Iterator : IteratorProtocol = IndexingIterator<Self>
    associatedtype SubSequence : IndexableBase, Sequence = Slice<Self>
    associatedtype Indices : IndexableBase, Sequence = DefaultIndices<Self
>

    // ...
}
```

从Iterator开始

这应该是我们最熟悉的 associatedtype 了，它是一个遵从 IteratorProtocol 的类型，用于在集合中不断访问下一个元素，之前，我们也自己实现过 Iterator 。那么，在 Collection 里，IndexingIterator 有什么特殊之处么？

如果你去翻翻Github上IndexingIterator的源代码(https://github.com/apple/swift/blob/master/stdlib/public/core/Collection.swift#L370-L417)就会发现，并没有。它只是一个被更加严格约束了的 Iterator ，或者更准确的说，是一个实现了 Sequence 的 Iterator 。从它的声明里，我们就已经能窥知一二了：

```
public struct IndexingIterator<
  Elements : _IndexableBase
  // FIXME(ABI)#97 (Recursive Protocol Constraints):
  // Should be written as:
  // Elements : Collection
> : IteratorProtocol, Sequence {
    // ...
}
```

至少从这个声明来看，它和我们之前自己实现的 Iterator 并无不同，只是对于 Iterator 要迭代的内容，做了更明确的约定。从开发者的注释里可以看到，本意上， IndexingIterator 是要迭代一个 Collection 类型的，只不过由于当前Swift语言机制的约束，这里用了一个叫做 _IndexableBase 的 protocol，对于它，我们就不进一步展开了，大家知道这里的本意是要让 Elements 遵从 Collection protocol就好了（在Swift 4中， _IndexableBase 会被改成 Collection ）。

其次，在 IndexingIterator 里，定义了两个属性：

```
internal let _elements: Elements
internal var _position: Elements.Index
```

_elements 就是 Collection 类型自身， _position 则是当前遍历到的位置。

最后，是 Iterator 的 init 和 next 方法：

```
public init(_elements: Elements) {
    self._elements = _elements
    self._position = _elements.startIndex
}

public mutating func next() -> Elements._Element? {
    if _position == _elements.endIndex { return nil }

    let element = _elements[_position]
    _elements.formIndex(after: &_position)

    return element
}
```

看到了？跟我们自己实现的 Iterator 如出一辙。我们用下面的代码测试一下：

```
var numberQueue: FIFOQueue = [1, 2, 3]

var i1 = numberQueue.makeIterator()
// IndexingIterator<FIFOQueue<Int>>

i1.next() // Optional(1)

numberQueue.push(4)
i1.next() // Optional(2)
i1.next() // Optional(3)
i1.next() // nil

var i2 = numberQueue.makeIterator()

i2.next() // Optional(1)
i2.next() // Optional(2)
i2.next() // Optional(3)
i2.next() // Optional(4)
```

从上面的注释结果里，我们就能看到，不同的 Iterator 之间，既不共享 Collection 的值，也不共享遍历的状态。

集合中两个位置之间的距离 - IndexDistance

`IndexDistance` 是一个我们几乎不会修改的关联类型，就是一个 `Int` ，表达了从集合的一个位置移动到另一个位置要“前进”的步数。

刚才我们提到过，对于一个集合来说，它至少定义了两个位置：

- `startIndex`： 集合的起始位置；
- `endIndex`： 结合最后一个元素的**下一个**位置；

我们用 `endIndex - startIndex` ，就可以得到集合中，从开头“走”到结尾的步数，也就是集合包含的元素个数。

必须也是一个集合类型的SubSequence

一个集合类型的区间，必须也是一个集合类型，这是 `Collection` 对它的 `SubSequence` 类型进行的约束。它的默认值 `Slice<Self>` 包含了一个其截取的原始集合对象、截取区间的起始位置 `startIndex` 以及截取区间的结束位置 `endIndex` 。 `Collection` 和 `SubSequence` 的关系，和我们在之前的集合内容中提到的 `Array` 和 `ArraySlice` 是一样的。

尽管区分 `SubSequence` 和 `Collection` 会让我们实现区间类型的时候有更大的灵活性，但让 `SubSequence` 和 `Collection` 是同样的类型也有其自身的好处。再下一节中，我们还会进一步来比较这两种实现方式。

用于标记所有可索引位置的Indices

最后一个要介绍的 `associatedtype` 是 `Indices` ，它也是一个集合类型，包含了一个集合中，所有可以索引的位置。默认情况下，它是一个叫做 `DefaultIndices` 的集合。

例如，对于 `numberQueue` 来说：

```
var numberQueue: FIFOQueue = [1, 2, 3]

numberQueue.indices.forEach {
    print("\(type(of: $0)): \($0)")
}
// Int: 0
// Int: 1
// Int: 2
```

在注释的结果中可以看到， `numberQueue` 中包含了3个可以索引的位置，值是`0 / 1 / 2`。需要注意的是， `Collection.endIndex` 并没有包含在 `indices` 里，因为它表示的是最后一个元素的下一个位置，我们并不能用它来索引任何内容。

另外，和我们刚才提到过的 `Slice<Self>` 类似， `DefaultIndices` 也会保存一个原始集合的副本。所以，**如果集合类型中索引的计算和集合自身没关系，我们可以让 `Indices` 更独立一些：**

```
extension FIFOQueue {
    var indices: CountableRange<Int> {
        return startIndex..<endIndex
    }
}
```

这样，编译器就可以根据 `indices` 的类型，自动推导出 `Collection.Indices` 的类型，这算是我们可以进行的一个几乎没有成本的优化。

What's next?

以上，就是我们这一节的内容。至此，我们应该对 `Sequence` ， `Iterator` ， `Collection` 以及它的各种 `associatedtype` 要表达的概念，有一个比较具象的认识了。当然，并不是所有的集合类型都可以像我们的 `FIFOQueue` 一样可以用 `Int` 类型作为 `Index` 。否则，Swift也不用如此大费周折的设计一堆 `protocol` 来抽象集合的表现形式。在接下来的几节中，我们就来实现一个更复杂的自定义 `Collection` 类型，通过这个过程，来加深我们对 `Collection` 中各种约束的理解。

⏮ 自定义一个阳春白雪的Collection (https://www.boxueio.com/series/advanced-collections/ebook/163)

实现一个Swift“风味”的链表集合 ⏭ (https://www.boxueio.com/series/advanced-collections/ebook/165)

职场漂泊的你，每天多学一点。

从开发、测试到运维，让技术不再成为你成长的绊脚石。我们用打磨产品的精神去传播知识，把最新的移动开发技术，通过简单的图表， 清晰的视频，简明的文字和切实可行的例子一一向你呈现。让学习不仅是一种需求，也是一种享受。

泊学动态

一个工作十年PM终创业的故事（二） (https://www.boxueio.com/after-the-full-upgrade-to-swift3)
Mar 4, 2017

人生中第一次创业的"10有" (https://www.boxueio.com/founder-chat)
Jan 9, 2016

猎云网采访报道泊学 (http://www.lieyunwang.com/archives/144329)
Dec 31, 2015

What most schools do not teach (https://www.boxueio.com/what-most-schools-do-not-teach)
Dec 21, 2015

一个工作十年PM终创业的故事（一） (https://www.boxueio.com/founder-story)
May 8, 2015

泊学相关

关于泊学 ›

加入泊学 ›

泊学用户隐私以及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)

联系泊学

Email: 10[AT]boxue.io (mailto:10@boxue.io)

QQ: 2085489246

友情链接 SwiftV (http://www.swiftv.cn) | Seay信息安全博客 (http://www.cnseay.com) | Swift.gg (http://swift.gg/) | Laravist (http://laravist.com/) | SegmentFault (https://segmentfault.com) | 靛青K的博客 (http://blog.dianqk.org/)