₩ 集合类型背后的"轮子"

▶ 返回视频

两种不同拷贝语义的Iterator ▶

(/series/advanced-collections)

(https://www.boxueio.com/series/advanced-collections/ebook/161)

从抽象顺序访问一系列数据开始

在前面讲到集合类型的时候,我们提到了很多种不同形式的数据集合。尽管它们用起来相对简单和独立,但是,这些集合类型背后,却隐藏着一系列组织严密,分工明确的 protocol 。甚至可以说,Swift标准库中集合类型的实现,是Swift官方对于protocol oriented programming的一份教科书一样的演绎。

而当你要深入其中一探究竟的时候,却发现并不那么容易。在这个系列里,我们就带着大家去感受下这套被精心设计的类型系统。虽然我们无法做到巨细靡遗,但至少可以提供一个脉络清晰的线索,让你逐个熟悉这些埋藏在集合类型背后的轮子。

而这一切,都要从 Sequence 开始。

从抽象顺序访问—系列数据开始

抛开各种不同的数据存储方式,我们对一个数据集合最常见和简单的访问诉求,就是逐个访问集合中的每 ♥ Back to series (/series/advanced collections) 个行为,你最先想到的,应该就是 for...in 循环:

```
let numbers = [1, 2, 3, 4, 5]
for number in numbers {
   print(number)
}
```

但是,支持顺序访问的集合类型,当然不止 Array 一个。如果我们要抛开 Array 这个具体的类型,而只是单纯的表达**可以被顺序访问的一系列数据**这个概念怎么办呢?

为了抽象这个过程,一个核心思想就是,我们得提供一种方法,使之能够依序遍历集合类型的每个元素,而又无需暴露该集合类型的内部表达方式。

假设,我们管这种方法叫做 Iterator ,那么,顺序访问集合元素这个行为就可以抽象成一个 protocol:

```
protocol Sequence {
   associatedtype Iterator

func makeIterator() -> Iterator
}
```

这样就表示,所有遵从了 Sequence protocol的类型,都提供了一个 makeIterator 方法,我们可以用这个方法返回的一个叫做 Iterator 类型的对象来顺序访问集合类型中的每一个元素。

那么,这个 Iterator 类型,又该是什么呢?

存储容器和访问方法的胶着剂 - Iterator

仔细想一下 Iterator 要完成的任务,就会比较有思路了:

- 一方面, Iterator 要知道序列中元素的类型;
- 另一方面, Iterator 还要有一个可以不断访问下一个元素的方法;

由于 Iterator 不能和某个具体的序列类型相关,我们也要把这些信息抽象成一个 protocol:

```
protocol IteratorProtocol {
   associatedtype Element

   mutating func next() -> Element?
}
```

然后,在之前的 Sequence 定义里,我们让 Iterator 遵从 IteratorType:

○ 字号

● 字号

✔ 默认主题

✔ 金色主题

✔ 暗色主题

```
protocol Sequence {
   associatedtype Iterator: IteratorProtocol
   func makeIterator() -> Iterator
}
```

于是,我们就通过 Sequence 和 IteratorProtocol ,实现了下面的约束:

当我们通过遵从 Sequence 实现一个支持顺序访问的序列类型时,也必须要自己实现一个与之搭配的 Iterator ,这个 Iterator 一定是对序列类型的内部形态了如指掌的,因为,它需要提供一个可以保存当前遍历状态,并返回遍历值的方法 next ,当遍历结束时, next 返回 nil 。

然后,对于所有遵从了 Sequence 的序列类型,我们就有了一个顺序遍历的套路:

- 1. 调用 makeIterator 获得表达序列起始位置的 Iterator;
- 2. 不断调用 next 方法, 顺序遍历序列的每一个元素;

实际上,我们并不用自己定义上面的 Sequence 和 IteratorProtocol ,这里,我们只是演示这个抽象构思的过程。Swift标准库中,对于支持顺序访问的序列类型,就是这样通过protocol来约束的。

并且,Swift中的 Array 就是遵从 Sequence protocol的类型。我们可以这样来遍历它:

```
var begin = numbers.makeIterator()
while let number = begin.next() {
   print(number)
}
```

在上面这个例子里,我们先调用 makeIterator() 获取了 Array 的 Iterator 对象。然后,不断调用它的 next 方法遍历了数组中的每一个成员。在控制台上,我们会看到和之前的 for...in 同样的结果。

一个阳春白雪的 Sequence 实现

用 Array 试验过之后,为了进一步理解 Sequence 和 Iterator 的关系,我们还可以自定义一个 Sequence 类型。例如,我们要定义一个表示Fibonacci数列的集合类型,

首先,定义这个序列类型的 Iterator:

```
struct FiboIterator: IteratorProtocol {
   var state = (0, 1)

mutating func next() -> Int? {
    let nextValue = state.0
    state = (state.1, state.0 + state.1)

    return nextValue
   }
}
```

在这个 I terator 的实现里,我们通过 state ,保存了每次迭代之后的状态。这样,就可以不断调用 next 方法、获取下一个数值。

然后,再来定义Fibnacci序列本身:

```
struct Fibonacci: Sequence {
   func makeIterator() -> FiboIterator {
      return FiboIterator()
   }
}
```

可以看到,它的实现非常简单,只是通过 make Iterator 方法,返回了用于遍历自身的 Iterator 对象

定义好之后, 我们可以这样来试一下:

```
let fib = Fibonacci()
var fibIter = fib.makeIterator()
var i = 1
while let value = fibIter.next(), i != 10 {
   print(value)
   i += 1
}
```

在控制台里,我们就能看到前10个fibnacci数了。

What's next?

在这两个例子里,无论是遍历有限数组 numbers ,还是遍历无限序列 Fibonacci ,我们不难发现 Iterator 的一个特点: Iterator.next 只能够单次向前遍历序列,我们无法使用同一个 Iterator 对象在序列类型中反复遍历。也就是说, IteratorProtocol 只约定了可以顺序访问序列 类型的最小行为集合。

通过 IteratorProtocol , Sequence 很好的把各种不同形式的数据访问细节隐藏了起来。而这种思想,就是我们开始理解集合背后诸多protocol的开始。在下一节,我们将继续讨论 Iterator ,当它被拷贝的时候,我们应该让所有拷贝出来的 Iterator 共享状态么?

▶ 返回视频

两种不同拷贝语义的Iterator ▶

(/series/advanced-collections)

(https://www.boxueio.com/series/advanced-collections/ebook/161)



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

2017 © Boxue, All Rights Reserved. 京ICP备15057653号-1 (http://www.miibeian.gov.cn/) 京公网安备 11010802020752号 (http://www.beian.gov.cn/portal/registerSystemInfo? recordcode=11010802020752)