集合类型背后的“轮子”

理解Collection中的associatedtype
(https://www.boxueio.com/series/advanced-collections/ebook/164)

如何为内存不连续的集合设计索引类型-I
(https://www.boxueio.com/series/advanced-collections/ebook/166)

实现一个Swift“风味”的链表集合

Back to series (/series/advanced-collections)

字号
字号
默认主题
金色主题
暗色主题

什么情况下，一个 Collection 的 Index 类型不能是 Int 呢？简单来说，当集合类型的数据存储形式不是连续的地址空间的时候。对于这种情况，我们就需要为集合类型自定义 Index 类型。但是这个过程。Hmm...比我们想象的，要麻烦一些。

符合这种特征的一个最典型的数据结构，就是链表。为了理解整个设计和实现的过程，我们先来实现一个链表类型。

一个朴素的链表实现

如果你之前用其他语言实现过链表，一个不太优美的事情是我们需要用某种形式的哨兵值来表示队列结尾，特殊值也好，空值也好。但通常，这样的做法终究会破坏代码的美感，我们得时刻记得，喔，这个东西表示链表结束。

但好在，Swift有了一个不错的方案，我们可以让一个 enum 来表示链表，它有两个 case ，一个表示队尾，一个表示包含值的普通链表，像这样：

```
enum List<Element> {
    case end
    indirect case node(Element, next: List<Element>)
}
```

由于在Swift中， enum 是一个值类型，为了让它的 case 可以引用另外一个 List 对象，我们需要在 case 前面使用关键字 indirect 。这样编译器就会为 case 的访问设置一个间接层，让它可以引用一个 List 。

然后，我们就可以实现一个方法，把插入到链表中的节点串联起来：

```
extension List {
    func insert(_ value: Element) -> List {
        return .node(value, next: self)
    }
}
```

最后，当我们要构建一个链表的时候，可以这样：

```
///
/// +---+------+    +---+------+    +-----+
/// | 1 | next | -> | 2 | next | -> | end |
/// +---+------+    +---+------+    +-----+
///
let list1 = List<Int>.end.insert(2).insert(1)
```

但这样有一点儿不方便，因为新节点每次都是插入在当前节点前面的，所以，我们构建 list1 的时候，插入节点的顺序和我们期望的链表中元素的顺序是反的。但解决这个问题并不是什么难事儿，我们之前在实现 FIFOQueue 的时候，实现过一个 ExpressibleByArrayLiteral protocol：

```
extension List: ExpressibleByArrayLiteral {
    init(arrayLiteral elements: Element...) {
        self = elements.reversed().reduce(.end) {
            $0.insert($1)
        }
    }
}
```

其实，很简单，我们只要把用来初始化 List 的array literal先反过来，然后以 List.end 为初始值，把参数 reduce 成一个 List 就好了。

然后，我们之前的 list 就可以定义成这样：

```
var list2: List = [1, 2]
```

这次，虽然结果和之前是相同的，但看起来就舒服多了。搞定了初始化 List 的方法之后，我们给 List 添加两个最常用的操作，在链表的头部添加和删除元素。

两个最基本的操作：push和pop

它们的实现思路都很简单，我们直接看代码就好：

```
extension List {
    mutating func push(_ value: Element) {
        self = self.insert(value)
    }

    mutating func pop() -> Element? {
        switch self {
        case .end:
            return nil
        case let .node(value, next: node):
            self = node
            return value
        }
    }
}
```

List的内存管理

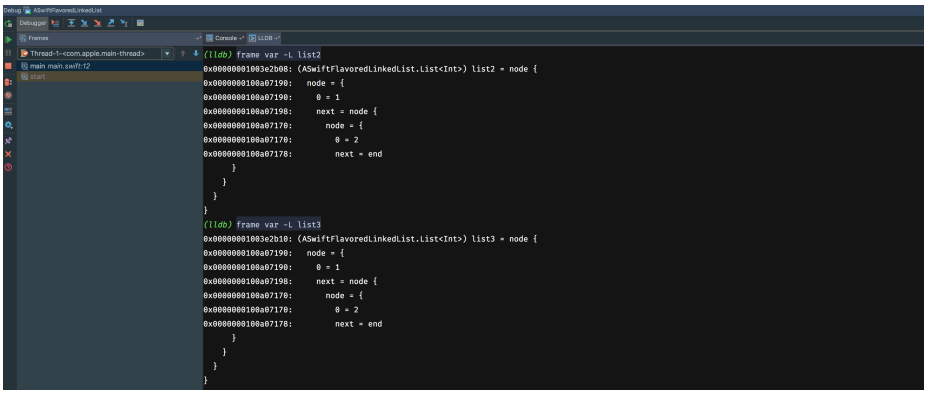
有了这些基本方法之后，我们来了解下 List 是如何管理它的存储结构的。首先，我们复制两个 list2：

```
var list3 = list2
var list4 = list2
```

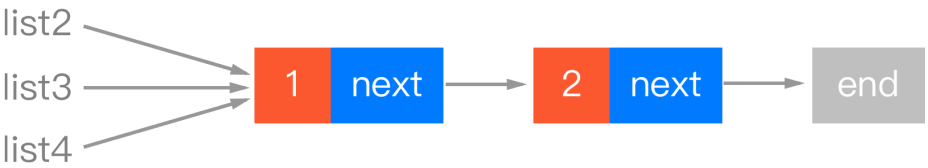
由于此时的 list2 并不是 .end 节点，因此 list2， list3 和 list4 都是引用类型，它们实际表示的，都是同一个 .node，为了确认这个事情，我们可以在 var list4 = list2 这里设置一个断点，然后把程序执行起来，当程序断下来之后，我们在lldb里执行下面的命令：

```
frame var -L list2
frame var -L list3
```

然后，我们可以看到和下面类似的效果：

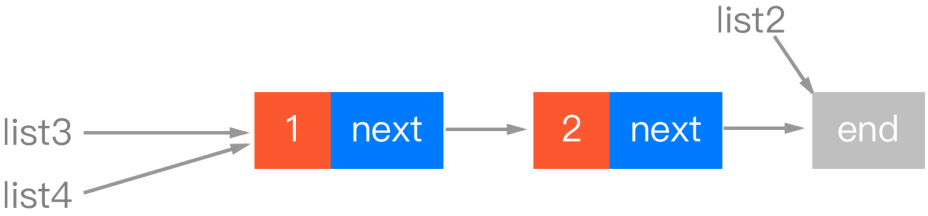


也就是说，此时 list2 的地址是 0x0x00000001003e2b08， list3 的地址是 0x00000001003e2b10，它们各占8字节，保存的内容，则是同一个 .node 对象的地址：0x0000000100a07190。等到 var list4 = list2 执行结束后， list2 / list3 / list4 就会指向内存中同一个 .node 节点了。



然后，我们 pop 其中一个 List：

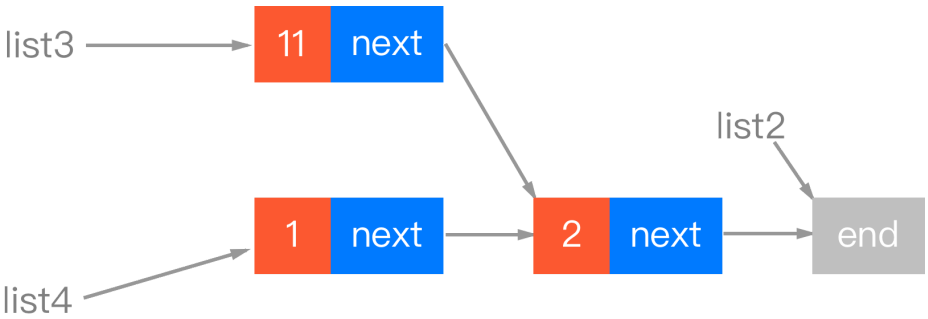
```
list2.pop()
list2.pop()
```



在图中可以看到， pop 不断在改变 list2 引用的节点，当它离开值为1的节点时，这个节点还剩下 list3 和 list4 在引用。当它离开值为2的节点时，这个节点依旧被值为1的节点引用，它不会有什么变化，最后 list2 会引用值为 .end 的节点。

然后，我们在 list3 中 push 一个元素：

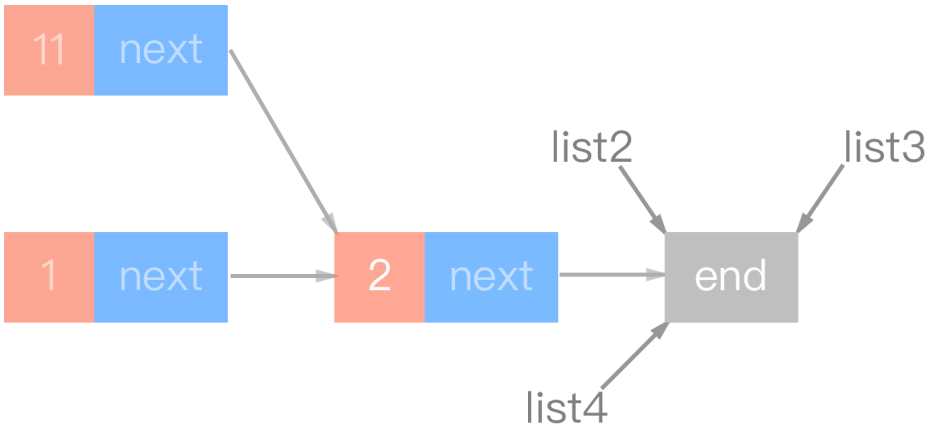
```
list3.pop()
list3.push(11)
```



从图中可以看到，此时的内存结构像是一个“双头链表”， list3 和 list4 各自引用着自己的表头，然后它们却共享着一个“尾巴”。然后，我们把 list3 和 list4 的所有元素都 pop 掉：

```
list3.pop()
list3.pop()

list4.pop()
list4.pop()
```



这样，所有不再有引用的节点，就会被ARC回收掉了。此时， list2 / list3 / list4 引用了同一个 .end 节点。

看到这里，我们就可以回头思考一个问题了： push 和 pop 方法都没有修改节点里的值，为什么还要用 mutating 关键字来修饰它们呢?

实际上，对于一个值类型来说，它的常量性体现在它的值本身上，也就是通过它的方法对 self 的操作来体现的。而 mutating 的作用就在于此，它会在方法中安插一个隐藏的 inout Self 参数，只要一个方法修改了 self 的值，就要使用 mutating 来修饰它。

What's next?

在这个例子里，我们使用了Swift特有的方式实现了一个简单的链表，并以此，进一步了解了Swift中值类型的常量性以及 mutating 方法的含义。

但这只是个开始，我们的目的是要让这个 List 类型适配Swift中的 Collection 。但是，出于 Collection 中对性能的严苛要求，实现它，并不如我们想象的那么简单。而解决性能问题的关键，就是我们要如何为 List 选定合格的 Index 类型。

理解Collection中的associatedtype
(https://www.boxueio.com/series/advanced-collections/ebook/164)

如何为内存不连续的集合设计索引类型-I
(https://www.boxueio.com/series/advanced-collections/ebook/166)

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/)