What's new in Swift 4

更智能安全的Key Value Coding

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/235)

为什么要新增一个swapAt方法?

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/237)

Dictionary初始化以及常用操作的诸多改进

Back to series (/series/what-is-new-in-swift-4)

字号
字号
默认主题
金色主题
暗色主题

在SE-0165 (https://github.com/apple/swift-evolution/blob/master/proposals/0165-dict.md)这份提议中，针对 Dictionary 和 Set 的常用场景，为这两种类型在初始化、内容读写以及成员遍历方面都做了一些改进。在这一节，我们就来逐一了解它们。

改进的init方法

当我们对 Array 或 Set 进行过一些函数式操作后，得到的结果有可能会丢掉之前的集合类型。例如：

```
let numberSet = Set(1...100)
let evens = numberSet.lazy.filter { $0 % 2 == 0 }
type(of: evens) // LazyFilterCollection<Set<Int>>
```

尽管概念上，我们认为 evens 仍旧应该是一个 Set，但实际上，它是一个 LazyFilterCollection （Array 的情况是类似的，我们就不列举了）。于是， evens 就不再支持 Set 的所有操作了。于是，下面的代码尽管在语义上正确，但会导致编译错误：

```
evens.isSubset(of: numberSet) // !! ERROR !!
```

为了解决类似的问题， Array 和 Set 的 init 方法可以把这种函数式操作后的结果，再变回各自标准的类型，像这样：

```
let evenSet = Set(evens)
```

然后，我们就可以愉快的使用操作后的结果了：

```
evenSet.isSubset(of: numberSet) // true
```

但是， Dictionary 类型的 init 方法，却没有这个功效，来看下面的例子：

```
let numberDictionary =
    ["one": 1, "two": 2, "three": 3, "four": 4]
let evenColl =
    numberDictionary.lazy.filter { $0.1 % 2 == 0 }
```

类似的， evenColl 的类型是 LazyFilterCollection<Dictionary<String, Int>>，它同样不再是一个 Dictionary 。但我们却无法用 init 方法把这个结果转换回来：

```
let evenDictionary = Dictionary(evenColl) // !! ERROR !!
```

转回标准Dictionary的方法

为了解决这个问题，Swift 4中给 Dictionary 新增了一个 init 方法：

```
// Still unfinished, you will get a compile time error:
// generic parameter 'Key' could not be inferred.
// See https://bugs.swift.org/browse/SR-922
let evenDictionary =
    Dictionary(uniqueKeysWithValues: evenColl)
```

但至少在录制这段视频的时候，这个功能仍旧未完全实现，SR-922 (https://bugs.swift.org/browse/SR-922)记录了这个问题。对此，一个临时解决方案是这样的：

```
let evenDictionary = Dictionary(uniqueKeysWithValues:
    evenColl.map { (key: $0.0, value: $0.1) })
// ["four": 4, "two": 2]
```

无论如何，理解这个 init 的方法的来由，要比现在通过编译重要的多。除此之外，我们还可以更多方式来体验这个 init(uniqueKeysWithValues:) 方法。例如，把一个 Array 变成用其索引为Key的 Dictionary：

```
let numbers = ["ONE", "TWO", "THREE"]
var numbersDict = Dictionary(uniqueKeysWithValues:
    numbers.enumerated().map { ($0.0 + 1, $0.1) })
// [2: "TWO", 3: "THREE", 1: "ONE"]
```

当然，用我们之前提过单边range操作符，上面的代码还可以写成这样：

```
numbersDict = Dictionary(uniqueKeysWithValues:
    zip(1..., numbers))
// [2: "TWO", 3: "THREE", 1: "ONE"]
```

总之，用一句话总结就是， init(uniqueKeysWithValues:) 的目的，就是把各种处理过后的集合类型，重新变回标准 Dictionary 。

解决源数据中的重复Key

在之前把其它集合转换回 Dictionary 的例子里，我们忽略掉了一个问题。如果源数据中作为Key的部分有重复怎么办呢，例如这样：

```
let duplicates = [("a", 1), ("b", 2), ("a", 3), ("b", 4)]
// fatal error: Duplicate values for key: 'a'
let letters = Dictionary(uniqueKeysWithValues: duplicates)
```

我们手工构造了一个包含重复Key的 Array<(String, Int)> ，这时编译器就会提示我们作为Key， a 重复了。对此，我们需要给 Dictionary 的 init 方法提供一个clousre，告诉它重复Key的处理方法。

例如，只选择第一个遇到的Key和Value：

```
let letters = Dictionary(duplicates,
    uniquingKeysWith: { (first, _) in first })
// ["b": 2, "a": 1]
```

这里， uniquingKeysWith 的类型是 (Value, Value) throws -> Value ，表示如何处理Key相同时的两个Value。在上面的例子里，我们执行的动作就是只选择第一个。

理解了这个closure的含义之后，我们就可以对重复Key的值采取各种行动了，例如，选择相同Key中的最大一个：

```
let letters = Dictionary(duplicates, uniquingKeysWith: max)
// ["b": 4, "a": 3]
```

组织序列中满足特定条件的元素

假设我们有一个记录人名的数组：

```
let names = ["Aaron", "Abe", "Bain", "Bally", "Bald", "Mars", "Nacci"]
```

为了按起始字母分类所有人名，我们可以这样：

```
let groupedNames = Dictionary(grouping: names, by: { $0.first! })
// ["B": ["Bain", "Bally", "Bald"], "A": ["Aaron", "Abe"], "M": ["Mars"],
 "N": ["Nacci"]]
```

带有默认值的下标操作符

在 Dictionary 里，使用下标操作符会返回 Value? 而不是 Value 在某些时候是个很麻烦的事情。例如，我们用一个 Dictionary<Character, Int> 统计字符串中每个字符的出现的个数：

```
let characters = "aaabbbcc"
var frequencies: [Character: Int] = [:]

characters.forEach {
    if frequencies[$0] != nil {
        frequencies[$0]! += 1
    }
    else {
        frequencies[$0] = 1
    }
}

frequencies
// ["b": 3, "a": 3, "c": 2]
```

在这个例子里，我们得通过一个 `if` 判断下标操作符是否为 `nil` ，来为还没统计过的字符设置默认值1。为了进一步改进这种应用场景的语义，Swift 4为 `Dictionary` 的下标添加了默认值：

```
characters.forEach {
    frequencies[$0, default: 0] += 1
}
```

这样，只要在下标操作符中使用了 `default` ，对 `Dictionary` 的访问就不再返回optional了，就语义来说，也更易懂。

转为Dictionary定制的filter和mapValue

在Swift 3中，对 `Dictionary` 调用 `filter` 会返回一个 `Array<(Key, Value)>` ，但在Swift 4里，返回的结果，仍旧是和之前同样的 `Dictionary` ：

```
let filtered = numberDictionary.filter { $0.value % 2 == 0 }

// Array<(Key, Value)> in Swift 3
// Dictionary<String, Int> in Swift 4
type(of: filtered)
```

另外，有时，我们仅希望对 `Dictionary` 中的 `Value` 进行某种变换，并保持 `Dictionary` 的类型不变，在之前的Swift 3 Dictionary视频 (https://boxueio.com/series/collection-types/ebook/130)中，我们还提到过这种需求，现在，Swift 4官方添加了这个 `mapValues` 方法：

```
let mapped = numberDictionary.mapValues { $0.lowercased() }
mapped // [2: "two", 3: "three", 1: "one"]
```

更智能安全的Key Value Coding
(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/235)

为什么要新增一个swapAt方法?
(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/237)

泊 .I HD

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)

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/)