Swift 3 Collections

常用的Dictionary extension
(https://www.boxueio.com/series/collection-types/ebook/130)

和Set相关的基础知识
(https://www.boxueio.com/series/collection-types/ebook/132)

为自定义类型实现Hashable Key

Back to series (/series/collection-types)

字号
字号
默认主题
金色主题
暗色主题

本质上来说，`Dictionary` 是一个哈希表，它所有的key都用各自的哈希值保存在一个数组里。因此，通过key在 `Dictionary` 中访问value是一个*O(1)*操作。但这也对key的类型做出了一个要求，**它必须可以计算哈希值**。Swift标准库中提供的绝大多数类型，例如： `Int` / `Float` / `Double` / `String` / `Bool` / `Date` ... 等，都满足这个要求，因此我们可以直接拿它们来定义 `Dictionary` 。

但如果我们有一个自定义类型 `Account` ，表示泊学的网站账号，其中的 `alias` / `type` / `createdAt` 分别表示账号的别名、类型和注册日期：

```
 struct Account {
    var alias: String
    var type: Int
    var createdAt: Date
}
```

当我们把 `Account` 用作key的时候，Swift就会给我们提示下面的错误： **`Account` 没有遵从 `Hashable` protocol**：

```
let account11 = Account(alias: "11",
    type: 1, createdAt  : Date())
let data:[Account: Int] = [ account11: 1000 ]
```

Conform to Hashable protocol

如何让自定义类型遵从 `Hashable` protocol呢？第一件要做的事，就是告诉Swift，如何获取一个类型的哈希值，这是通过一个叫 `hashValue` 的属性完成的：

```
extension Account: Hashable {
    var hashValue: Int
}
```

如何计算 `Account.hashValue` 呢？有两个最重要的考量，分别是：性能和哈希值在整数范围的分布。因为每当我们要在 `Dictionary` 中查询、添加、修改或删除元素的时候，都要计算key的哈希值，如果这个计算过于消耗性能，那么计算哈希值的过程就有可能抵消掉通过key随机访问value带来的*O(1)*性能提升。

当然，你也不能盲目追求性能而忽视哈希值的整数值分布。说一个最极端的例子，如果你让所有情况计算得到的哈希值都是某个常数：

```
extension Account: Hashable {
    // A BAD idea
    var hashValue: Int { return 1 }
}
```

这个哈希函数有*O(1)*的性能，但这样，不同的 `Account` 对象就会有不同的哈希值，这叫做Collision。当然，Swift `Dictionary` 可以处理哈希值碰撞的情况，但你要随之付出的代价就是，通过哈希值读取value将从一个*O(1)*变成一个*O(n)*算法。因此，让哈希值在整数区间均匀分布也是设计哈希函数很重的考虑。

综上所述，设计一个好的哈希函数并不是一个容易的事情。对于我们来说，可以假设Swift标准库的类型提供的 `hashValue` 都满足性能和分布的要求。因此，当我们设计复合类型的哈希值的时候，可以基于这些标准类型的哈希值进行一些“低功耗”运算，例如，对这些值进行异或运算，绝大多数的CPU都对这个操作提供了指令级支持：

```
extension Account: Hashable {
    var hashValue: Int {
        return alias.hashValue ^
                type.hashValue ^
                createdAt.hashValue
    }
}
```

解决了 Account 的哈希值之后，Swift会继而提示我们：**Account没有遵从 `Equatable` protocol**。为什么还要遵从 Equatable 呢？这是因为哈希函数还有一个很重要的性质：**两个相等对象的哈希值必须是相同的**。因此，我们必须要解决什么叫做两个相等的对象，然后才有比较它们各自哈希值的事情。

Equatable 只有一个约束，就是为自定义类型实现 == 操作符：

```
extension Account: Equatable {
    static func == (lhs: Account, rhs: Account) -> Bool {
        return lhs.alias == rhs.alias &&
            lhs.type == rhs.type &&
            lhs.createdAt == rhs.createdAt
    }
}
```

在Swift里，运算符必须要定义成 static 方法，它的两个参数 lhs / rhs 则表示 == 两边的操作数。我们判断 Account 相等的方式很简单，只要它们每一个属性相等，则两个 Account 对象就是相等的。

当我们让 Account 遵从了 Equatable 之后，Swift编译器就不会再报错了。此时，我们在一开始创建的 data 也可以正常工作了。

Bitwise rotation

我们上面例子中提到的把所有属性进行XOR运算的方法，虽然简单高效，但也有一个问题，就是比较容易造成碰撞。因为XOR运算是可交换的，也就是说 `a ^ b == b ^ a`，因此，如果一个自定义类型中，有多个类型相同属性的时候，就会增大哈希值发生碰撞的概率，因此，我们可以用下面的代码，对其中的一些基础属性的哈希值进行按位旋转后再进行XOR运算：

```
struct Account {
    let INT_BIT = (Int)(CHAR_BIT) * MemoryLayout<Int>.size

    func bitwiseRotate(value: Int, bits: Int) -> Int {
        return (((value) << bits) | ((value) >> (UINT_BIT - bits)))
    }
}

extension Account: Hashable {
    var hashValue: Int {
        return bitwiseRotate(value: alias.hashValue, bits: 10) ^
            type.hashValue ^
            createdAt.hashValue
    }
}
```

首先，我们在 Account 中添加了一个常量 INT_BIT 表示一个整数的位数。其次，定义了一个辅助方法 bitwiseRotate(value:bits:)，它用于先把 value 向左移动 bits 位，再向右移动 (UINT_BIT - bits) 位。

有了这个方法之后，我们就可以在计算哈希值的时候，对其中的属性进行按位旋转了。

警惕引用类型的Key

和 Dictionary.Key 相关的最后一个内容，是**尽可能避免使用引用类型作为key**，这通常会给你带来不必要的麻烦。当一个引用类型作为key之后，当引用类型的对象在 Dictionary 之外被修改的时候，Key 的内容也会随之修改。这样你就再也无法获得之前的哈希值，也就无法获得对应的value了。

What's next?

以上就是和如何用自定义类型作为 Key 相关的话题，至此，我们对 Dictionary 的讨论就可以告一段落了。在下一节中，我们会看到Swift标准库中的另外一个无序集合： Set 。

常用的Dictionary extension
(https://www.boxueio.com/series/collection-types/ebook/130)

和Set相关的基础知识
(https://www.boxueio.com/series/collection-types/ebook/132)

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/)