Swift 3 Collections

和Dictionary相关的基础知识 (https://www.boxueio.com/series/collection-types/ebook/129)

为自定义类型实现Hashable Key (https://www.boxueio.com/series/collection-types/ebook/131)

常用的Dictionary extension

Back to series (/series/collection-types)

字号
字号
默认主题
金色主题
暗色主题

如果我们为上一节提到的视频观看记录提供一个默认值：

```
enum RecordType {
    case bool(Bool)
    case number(Int)
    case text(String)
}

let defaultRecord: [String: RecordType] = [
    "uid": .number(0),
    "exp": .number(100),
    "favourite": .bool(false),
    "title": .text("")
]
```

这样，当创建新纪录时，我们希望保持默认记录中的默认值，同时合并进不同用户的设置，例如：

```
var template = defaultRecord
var record11Patch: [String: RecordType] = [
    "uid": .number(11),
    "title": .text("Common dictionary extensions")
]

// How can we do this?
// template.merge(record11Patch)
// [
//    uid: .number(11),
//    "exp": .number(100),
//    "favourite": .bool(false),
//    "title": .text("Common dictionary extensions")
// ]
```

merge

然而，该如何实现这个 `merge` 呢？最重要的事情，就是要想一下什么内容可以被 `merge` 进来。最一般的情况来说，**无论任何形式的序列，只要它的元素中key和value的类型和 `Dictionary` 相同，就可以进行合并**。

如何在代码中表达这个特征呢？来看下面的例子：

```
extension Dictionary {
    mutating func merge<S:Sequence>(_ sequence: S)
        where S.Iterator.Element == (key: Key, value: Value) {

        sequence.forEach { self[$0] = $1 }
    }
}
```

由于 `Dictionary` 是一个 `struct` ，并且 `merge` 修改了 `self` ，我们必须使用 `mutating` 关键字修饰这个方法。而对于 `sequence` 参数，我们通过 `where` 关键字限定了两个内容：

- `S` 必须遵从 `Sequence` protocol， `Dictionary` 是众多遵从了 `Sequence` protocol的collection类型之一，但是，我们没必要一定只能合并 `Dictionary` ；
- `S` 的元素类型必须和原 `Dictionary` 的 `Element` 相同，其中 `Key` 和 `Value` 是 `Dictionary` 声明中的两个反省参数；

解决了参数问题之后，实现合并的算法就很简单了，我们只是更新 `self` 中每一个和 `sequence` 有相同key的值就好了。

这样，之前 template.merge(record11Patch) 就可以正常工作了。

既然，我们把 merge 参数的约束定义为了 Sequence ，那我们就来看一个合并非 Dictionary 类型的情况，例如，合并一个包含正确内容的 Array：

```
let record10Patch: [(key: String, value: RecordType)] = [
    (key: "uid",   value: .number(10)),
    (key: "title", value: .text("Common dictionary extensions"))
]

var template1 = defaultRecord
template1.merge(record10Patch)
// [
//    uid: .number(10),
//    "exp": .number(100),
//    "favourite": .bool(false),
//    "title": .text("Common dictionary extensions")
// ]
```

在上面的代码里，我们合并了一个tuple数组，它的类型是 Array<String, RecordType> ，数组中的每一项都包含了一个要合并进来的键值对。如果没有意外，合并 Array 和 Dictionary 都应该是可以正常工作的。

按照我们对 merge 的实现方式，实际上，任何一个遵从了 Sequence protocol的类型，只要它包含了和 template 相同的元素类型，都是可以 merge 的。

用一个tuple数组初始化Dictionary

理解了 merge 的实现和用法之后，其实，我们很容易把这个场景进一步扩展下，如果我们可以 merge 类型兼容的 Sequence ，那么，用这样的 Sequence 来初始化一个 Dictionary 也是可以的，把它看成是和一个空的 Dictionary 进行合并就好了：

```
extension Dictionary {
    init<S:Sequence>(_ sequence: S)
        where S.Iterator.Element == (key: Key, value: Value) {

        self = [:]
        self.merge(sequence)
    }
}
```

有了这个方法之后，我们直接用下面的代码就可以创建一个新的 Dictionary 对象：

```
let record11 = Dictionary(record11Patch)
// [
//    uid: .number(11),
//    "title": .text("Common dictionary extensions")
// ]
```

定制map的行为

最后一个要介绍的常用功能，是定制 Dictionary.map 的行为，默认情况下它返回的是一个 Array ，例如：

```
record11.map { $1 }
// [ .number(11).text("Common dictionary extensions")]
```

在上面的例子里， map 返回一个 Array<RecordType> ，但有时，我们仅仅希望对value做一些变换，而仍旧保持 Dictionary 的类型。为此，我们可以自定义一个“只map value”的方法：

```
extension Dictionary {
    func mapValue<T>(_ transform: (Value) -> T) -> [Key: T] {
        return Dictionary<Key, T>(map { (k, v) in
            return (k, transform(v))
        })
    }
}
```

在这个实现的最内部，我们用标准库中的 map 得到了一个 Array<(String, RecordType)> 类型的 Array ，而后，由于 Array 也遵从了 Sequence protocol，因此，我们就能直接使用这个 Array 来定义新的 Dictionary 了。

完成之后，用下面的代码测试下：

```
let newRecord11 = record11.mapValue { record -> String in
    switch record {
    case .text(let title):
        return title
    case .number(let exp):
        return String(exp)
    case .bool(let favourite):
        return String(favourite)
    }
}

// [
//    "uid": "11",
//    "title": "Common dictionary extensions"
// ]
```

这样，我们就用 record11 生成了一个 Dictionary<String, String> 类型的对象。

What's next?

在了解了一些常用的 Dictionary extension之后，下一节中，我们来看和 Dictionary 相关的最后一个内容，如何自定义hashable key。

◀ 和Dictionary相关的基础知识
(https://www.boxueio.com/series/collection-types/ebook/129)

为自定义类型实现Hashable Key ▶
(https://www.boxueio.com/series/collection-types/ebook/131)