Reactive Programming in Swift

Hello world in RxSwift
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/74)

理解Disposable & DisposeBag
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/76)

理解Observables and Observer

Back to series (/series/reactive-programming-in-swift)

在上一段视频通过 `UITextField` 的 `rx_text` 属性体会了“事件数组”的概念和用法之后，我们通过这段视频正式向大家介绍RxSwift (https://github.com/ReactiveX/RxSwift)中的事件序列，它叫做**Observables**。以及如何创建以及订阅Observables。

在Playground中使用RxSwift

为了能够更方便的看到代码的执行结果，在这段视频中，我们使用Playground向大家介绍RxSwift (https://github.com/ReactiveX/RxSwift) observables的用法。

首先，我们新建一个“OS X Application”，选择“Command Line Tool”，点击Next；

其次，设置一个项目名称，例如“Understanding Observables”，然后把Language设置成“Swift”，点击Next；

最后，给项目设置一个保存路径，点击Create。

然后，我们用同样的方法使用使用CocoaPods (https://cocoapods.org/)安装RxSwift (https://github.com/ReactiveX/RxSwift)，唯一一个不同的地方就是生成的Podfile里，我们**不要**对platform行取消注释：

```
# Uncomment this line to define a global platform for your project
# platform :ios, '8.0'
# Uncomment this line if you're using Swift
use_frameworks!

target 'Understanding Observables' do
    pod 'RxSwift',    '~> 2.0'
    pod 'RxCocoa',    '~> 2.0'
end
```

安装完成之后，我们用生成的.xcworkspace文件重新打开项目：

按 `Command + B` 构建一次。接下来，选中“Understanding Observables”，按 `Command + N` （或者如下图使用鼠标右键）：

在弹出的窗口中，选中“OS X / Source / Playground”，点击Next：

点击Create，完成添加。最后，打开新添加的Playground文件，在 `import Cocoa` 后面添加下面的代码：

```
import RxSwift
import RxCocoa
```

如果没有发生错误，就表示我们可以在Playground中使用RxSwift (https://github.com/ReactiveX/RxSwift)了。

> *如果Playground提示你“No such module RxSwift”，可以按 `Command + B` 重新构建一次就好了。*

创建Observables

接下来，我们创建一个最简单的Observable对象：

```
let emptySequence = Observable<Int>.empty()
```

其中， `Observable` 是RxSwift中的一个泛型类， `Observable<Int>` 表示一个.Success值类型为 `Int` （请参考我们在上一段视频中提到过的Event enumeration中关于事件的定义）的“事件数组”。

而empty则是 `Observable` 中的一个类方法，用于创建一个空的事件数组，**它唯一可以做的事情，就是向订阅者发送一个Completed事件**。

理解对Observables的订阅

订阅 `Observable` 中的事件很简单，使用 `subscribe` 方法就可以了：

```
emptySequence.subscribe {
    (event: RxSwift.Event<Int>) -> Void in
        print(event)
}
```

`subscribe` 方法接受一个Closure做为事件的处理函数，这个Clousre接受一个 `RxSwift.Event<Int>` 类型（因为我们定义的 `emptySequence` 中事件的值类型是 `Int` ）的参数，表示发生的事件，返回 `Void` 。

这个Closure甚至还有一个自己的名字，叫做**Observer**。

我们的Observer实现则很简单，只是把订阅到的事件打印在了控制台上，从输出的结果可以看到，我们只订阅到了一个Completed事件。

这就是定义**Observables**以及**Observer**的方法。除了 `empty` 之外，RxSwift (https://github.com/ReactiveX/RxSwift)还为我们提供了一些创建 `Observables` 的方法，我们来看一下它们的用法。

几种常见的Observables创建方法

`just` - 只包含一个事件的序列

`just` 创建的事件序列只包含一个事件元素，当订阅它的时候，它向订阅者发送两个消息：事件值和.Completed事件。例如：

```
print("--- Just sequence ---")

_ = Observable.just("Boxue").subscribe({
    (event: RxSwift.Event<String>) -> Void in
        print(event)
})
```

在控制台里，我们可以看到Just序列发送了两个事件：一个是.Next，它的值是我们定义序列时指定的值“Boxue”，然后紧跟着一个.Completed事件。

`of` - 包含固定个数事件的序列

`of` 创建一个可以向observer发送固定个数事件的序列，发送完成之后，发送.Completed：

```
print("--- Of sequence ---")

_ = Observable.of(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
        .subscribe({
            (event: RxSwift.Event<Int>) -> Void in
                print(event)
        })
```

在控制台里，可以看到我们订阅到了10个.Next事件，一个.Completed事件：

`error` - 只产生.Error事件的序列

error 用于定义一个只能订阅.Error事件的序列，例如：

```
let err = NSError(domain: "Test", code: -1, userInfo: nil)

_ = Observable<Int>.error(err).subscribe {
    (event: RxSwift.Event<Int>) in
        print(event)
}
```

在控制台里，我们就可以看到一个错误事件订阅：

create - 自定义的事件序列

create 允许我们使用Swift closure自定义一个事件序列的构建过程。例如，我们根据事件的值是否是偶数自定义一个事件序列：

```
print("--- create ---")

func myJust(event: Int) -> Observable<Int> {
    return Observable.create { observer in
        if event % 2 == 0 {
            observer.on(.Next(event))
            observer.on(.Completed)
        }
        else {
            let err =
                NSError(domain: "Not an even number",
                    code: 401, userInfo: nil)
            observer.on(.Error(err))
        }

        return NopDisposable.instance
    }
}
```

在上面的例子里，有几点需要特别说明：

- create 接受一个closure参数，用于定义事件的发生过程，我们先暂时忽略掉它的具体签名，只需要知道它有一个表示observer的参数就可以了；
- 在closure的实现里， on 和我们之前使用过的 subscribe 方法作用是类似的，用于向observer发送事件。当event是偶数的时候，我们向observer发送.Next和.Complete表示成功事件并结束；否则，我们向observer发送.Error；
- 最后， NopDisposable.instance 是一个静态类对象，表示当 create 创建的observable被销毁时，无需执行任何额外操作，在下一个视频中，我们会专门讲到observable被销毁的话题；

然后，我们可以像下面这样来使用 create 定义的事件序列：

```
myJust(10).subscribe { print($0) }
myJust(5).subscribe { print($0) }
```

这样，在控制台，我们就能分别看到一次.Next + .Compeleted的订阅和一次.Error的订阅了：

generate - 用prev决定next的事件序列

generate 可以生成一连串事件，并且，允许我们根据上一次事件的结果生成下一次事件，并设置.Completed条件：

```
print("--- generate ---")

_ = Observable.generate(
            initialState: 0,
            condition: { $0 < 10 },
            iterate: { $0 + 1 }
        ).subscribe {
            print($0)
        }
```

在这里例子里：

- `initialState` 用于指定事件序列的初始值，它是一个.Next(0)；
- `condition` 是一个closure，当它为true时，就生成.Next，否则生成.Completed；
- `iterator` 用于设置每一次发送事件之后，对事件值进行的迭代操作，在我们的例子里，就是.Next的值加1；

然后，我们使用 `subscribe` 订阅就可以在控制台看到下面的结果了：

`deferred` - 只有在被订阅后才创建并产生事件的序列

与其说 `deferred` 用于创建一个事件序列，不如说它是对创建序列的一种修饰。被 `deferred` 修饰后，事件序列只有在被订阅时才生成，并发送事件，并且，每订阅一次，就新生成一个事件序列对象。

例如，对于上面的 `generate` 例子来说：

```
print("--- deferred ---")
let deferredSequence = Observable<Int>.deferred {
    print("generating")

    return Observable.generate(
        initialState: 0,
        condition: { $0 < 3 },
        iterate: { $0 + 1})
}
```

当我们订阅deferredSequence时：

```
_ = deferredSequence.subscribe { print($0) }
_ = deferredSequence.subscribe { print($0) }
```

在控制台里，我们可以看到这样的结果：

这说明，deferredSequence创建了两次，并且向订阅者发送了3次.Next以及1次.Completed。

Next?

我们通过这段视频，向大家介绍了RxSwift (https://github.com/ReactiveX/RxSwift)中创建以及订阅 `Observable` 的用法。在继续之前，我们应该确保对以下概念有清楚的理解：

- 理解 `Observable`；
- 理解 `Observable` 发出的事件（.Success，.Error，.Completed）；
- 理解 `Observer`，以及如何处理事件的订阅；

在下一段视频中，我们将和大家分享关于Observables销毁的话题。

⏮ Hello world in RxSwift
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/74)

理解Disposable & DisposeBag ⏭
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/76)

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/)