Reactive Programming in Swift

理解Observables and Observer
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/75)

RxSwift UI交互 - I
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/77)

理解Disposable & DisposeBag

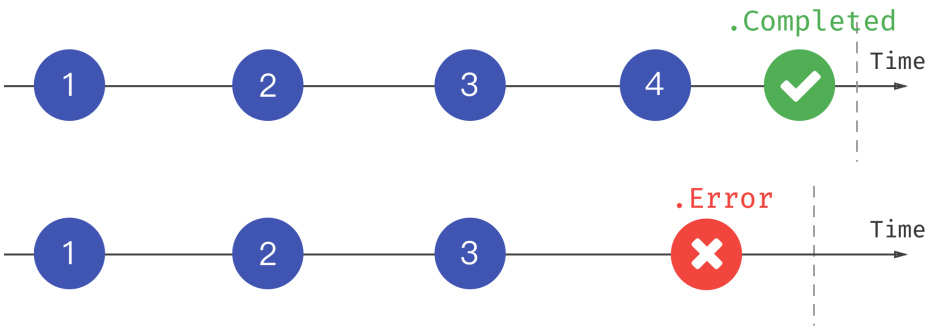
Back to series (/series/reactive-programming-in-swift)

字号
字号
默认主题
金色主题
暗色主题

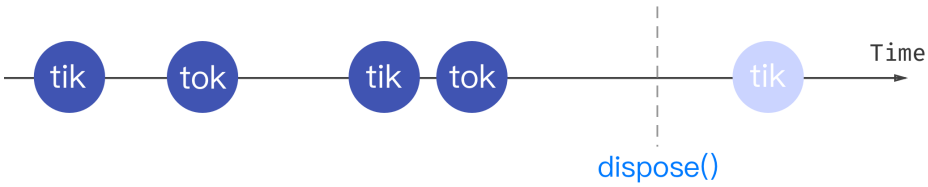
在上一段视频中，我们了解了创建Observable以及订阅事件的方法。在这段视频中，我们关注回收Observable使用的资源问题。

在继续之前，我们要先补充一点小知识。对于一个Observable来说，当它向订阅者发送.Completed或.Error事件之后，Observable的使命就结束了，属于这个Observable的所有资源都会被自动回收。

在上个视频中，我们使用的各种创建Observables的方法，它们创建的都是一个有限序列，因此，当最后它们向订阅者发送了.Completed或.Error事件之后，属于这些Observables的资源就被回收了。它们再也不会向订阅者发送任何消息。



但有时，事件序列在某种程度上是“无限的”。例如，一个计时器，它可以在固定时间间隔不断的生成事件。对于这样的“无限序列”，如果要回收它的资源，我们可以在订阅事件之后，调用一个叫做 `dispose()` 的方法。



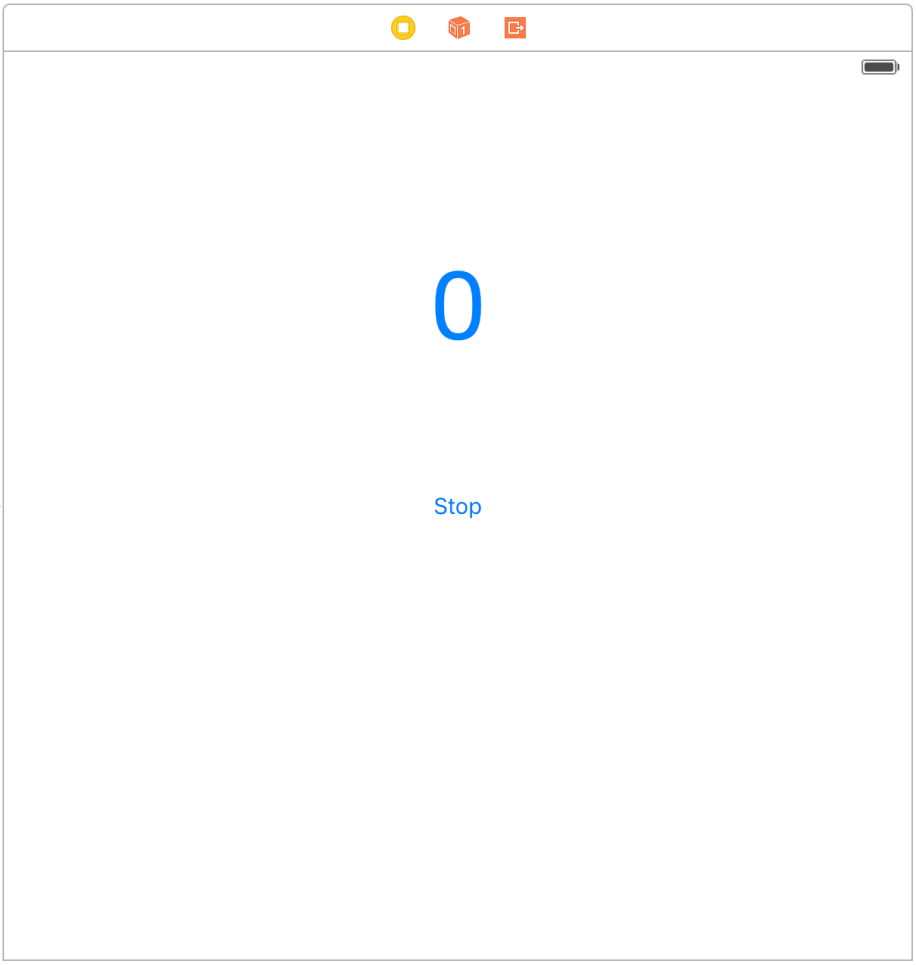
调用 `dispose()` 之后，和事件序列相关的资源就会被回收，于是，observer就再也不会订阅到任何事件了。

接下来，我们就看一个具体的例子。

准备工作

为了演示Observable被回收的效果，我们在Main.storyboard里添加了一个 `UITextField` 作为计数器。App运行后，它将每半秒钟更新一次。

然后，我们还添加了一个按钮，我们希望当点击这个按钮时，回收定时器序列资源，这样，上面这个计数器就不会再更新了。



为此，我们在ViewController.swift中，添加了对应的属性以及 IBOutlet：

```
class ViewController: UIViewController {

    // Omit for simplicity...

    var interval: Observable<Int>!

    @IBOutlet weak var counter: UITextField!
    @IBOutlet weak var disposeCounter: UIButton!

    // Omit for simplicity...
}
```

其中， interval 表示我们要使用的计时器事件序列， counter 表示用于显示计数的 UITextField， disposeCounter 表示界面中的Stop按钮对象。

至此，我们的初始环境就准备完了。接下来，我们从添加计时器序列开始。

一个可以发送无限事件的Observable

在 ViewController 的 viewDidLoad 方法里，添加下面的代码：

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a ni
b.

    self.interval =
        Observable.interval(0.5, schedule: MainScheduler.instance)
}
```

Observable.interval 会每隔固定的时间段生成一个整数值，我们可以在这里(http://reactivex.io/documentation/operators/interval.html)找到它的详细说明。它的第一个参数用于指定发送事件的时间间隔，第二个参数用于指定一个scheduler，它和多线程应用相关，后面我们会讲到这个话题。我们使用 MainScheduler.instance 表示在App的主线程里创建这个事件序列。

这样， self.interval 就会在被订阅的时候，每隔0.5秒，向订阅者从0开始，发送一个整数值事件。由于我们要把这个值显示在 UITextField 上，我们先使用 map 把这个序列变成一个 Observable<String> 序列：

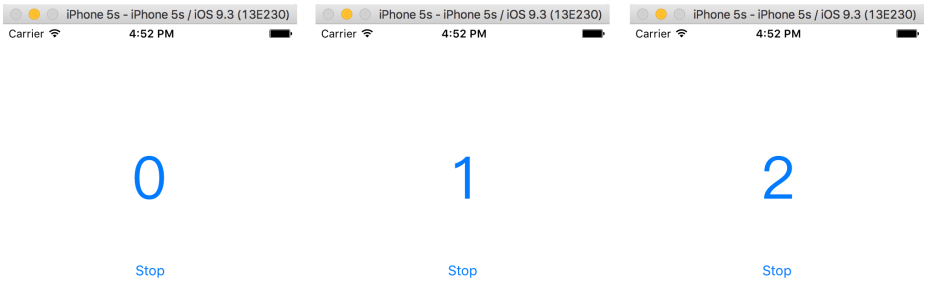
```
self.interval.map { return String($0) }
```

然后，我们订阅这个事件序列，当有事件发生时，设置 UITextField 的值：

```
self.interval.map { return String($0) }
    .subscribeNext { str in
    self.counter.text = str
    }
```

其中 subscribeNext 和 subscribe 的作用是类似的，只不过前者只订阅序列中的.Next事件，并且，可以直接在closure参数中，使用.Next的associated value。

接下来，按 Command + R 编译执行，就能看到 UITextField 中的值不断变化了。



dispose 手动回收事件序列资源

如果我们的App一直运行，那么这个计数器就会一直工作下去。如果我们要停止计数并且回收计数器使用的资源怎么办呢？第一种方式就是调用 dispose() 方法。我们分几步完成这个工作：

首先，给 ViewController 添加一个 Disposable! 类型的属性，顾名思义， Disposable 表示某种“用过之后就扔掉的东西”；

```
var subscription: Disposable!
```

其次，在订阅事件的时候，设置它：

```
self.subscription = interval.map { return String($0) }
    .subscribeNext { str in
        self.counter.text = str
    }
```

其中， subscribeNext 返回一个 Disposable 对象，表示一个“可以用完就扔掉的东西”。

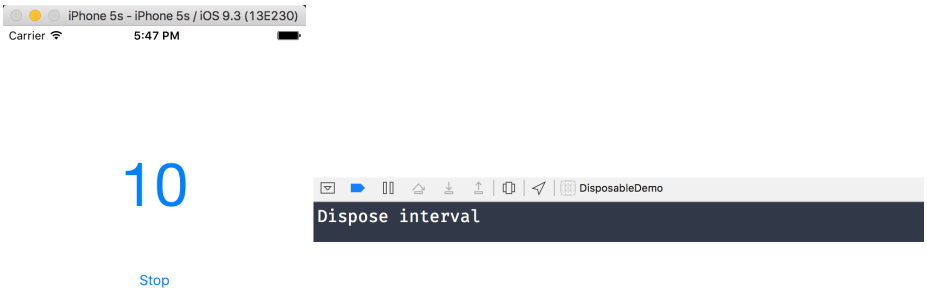
最后，处理Stop按钮的点击事件。在 viewDidLoad 方法里，添加下面的代码：

```
self.disposeCounter.rx_tap.subscribeNext {
    print("Dispose interval")
    self.subscription.dispose()
}
```

这里， rx_tap 是RxSwift (https://github.com/ReactiveX/RxSwift)对 UIButton 的一个扩展，表示按钮点击的事件序列（其实这也是一个无限事件序列）。

我们直接订阅了这个序列的.Next事件（也就是按钮被点击的事件）。当用户点击按钮时，我们向控制台打印了一个字符串，并且，调用了 subscription 的 dispose 方法。

调用 dispose 方法可以理解为，再也不需要订阅了。既然没人需要订阅了，interval序列就会被系统回收，再也不会发送任何事件了。 UITextField 将也将被固定在一个特定的值上。



但是，这种手工调用 `dispose` 回收Observable的方法更多是用来示意。RxSwift (https://github.com/ReactiveX/RxSwift)官网也告诉我们直接调用 `dispose` 并不是一个好主意：

> Note the you usually do not want to manually call dispose; this is only educational example. Calling dispose manually is usually a bad code smell. There are better ways to dispose subscriptions. We can use DisposeBag

是的，RxSwift (https://github.com/ReactiveX/RxSwift)提供了一个集中回收Observable的方式，叫做**DisposeBag**。

disposeBag 自动回收事件序列资源

我们可以把 `DisposeBag` 理解为一个装 `Disposable` 的“袋子”。当这个“袋子”被销毁的时候，它就会逐个销毁其中的 `Disposable` 对象。为了演示它的用法：

首先，我们给ViewController再添加一个属性：

```
var bag: DisposeBag! = DisposeBag()
```

这样我们就有了一个 `DisposeBag` 对象。

其次，在 `viewDidLoad` 方法里，把订阅 `self.interval` 的返回值，“装进袋子里”：

```
self.subscription.addDisposableTo(self.bag)
```

最后，在Stop按钮的事件处理函数里，把原来的 `self.subscription.dispose()` 改成：

```
self.bag = nil
```

重新 `Command + R` 编译执行，然后在 `UITextField` 值发生变化之后，点击Stop按钮，就会发现，这和我们之前调用 `dispose` 方法的效果是一样的。

因此，作为这种“无限事件序列”，最好的回收方法就是定义一个公用的 `DisposeBag` ，然后把它们统统装进去，当这个Bag的值为 `nil` 时，所有的序列就都被自动销毁了。

> 如果我们把订阅 `rx_tap` 的事件也放到 `DisposeBag` 里，甚至按钮点击的功能也就不好用了，不信你试试?

Next?

至此，我们已经为进一步探索RxSwift (https://github.com/ReactiveX/RxSwift)的应用做好充分准备了。如果你觉得还差些火候，不妨回过头再去重温下自己还没有深刻理解的部分。在接下来的一系列视频中，我们将向大家介绍如何使用RxSwift (https://github.com/ReactiveX/RxSwift)处理UI交互、网络编程、单元测试以及使用RxSwift (https://github.com/ReactiveX/RxSwift)的一些常用的编程手法，并逐步了解更多RxSwift (https://github.com/ReactiveX/RxSwift)处理事件序列的方法。

⏮ 理解Observables and Observer
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/75)

RxSwift UI交互 - I ⏭
(https://www.boxueio.com/series/reactive-programming-in-swift/ebook/77)

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/)