使用connectable operator回放事件

RxSwift - step by step

← 返回视频列表 | 预计阅读时间: *20分钟* | ‹ PREVIOUS

在上一节了解了connectable operators的含义之后，作为RxSwift常用operators的最后一部分，我们来看一些可以给新订阅者回放事件的connectable operators。

为了更好的观察事件的回放，我们新增了一个帮助函数：

```
public func stamp() -> String {
    let date = Date()
    let formatter = DateFormatter()
    formatter.dateFormat = "HH:mm:ss"
    let result = formatter.string(from: date)

    return result
}
```

它的功能很简单，就是用当前时间生成一个包含时分秒的字符串对象。

订阅时，回放指定个数的事件

接下来，还是上一节中使用的`interval` Observable，这次，我们使用`replay` operator，它有一个参数，表示有人订阅的时候，回放历史事件的个数：

```
let interval = Observable<Int>.interval(1,
    scheduler:MainScheduler.instance).replay(2)
```

然后，我们“事先”只准备好一个订阅者，并在调用`connect`方法启动`intervale`后，打印了启动时间。另外，和之前订阅消息不同的是，这次，我们在每个事件订阅的时候，都打印了一个时间戳：

```
_ = interval.subscribe(onNext: {
    print("Subscriber 1: Event - \($0) at \(stamp())") })
_ = interval.connect()
print("START - " + stamp())
```

接下来，我们分别在2秒、4秒后，分别加入新的订阅者：

```
delay(2) {
    _ = interval.subscribe(onNext: {
        print("Subscriber 2: Event - \($0) at \(stamp())")
    })
}

delay(4) {
    _ = interval.subscribe(onNext: {
        print("Subscriber 3: Event - \($0) at \(stamp())")
    })
}
```

完成后，执行一下，就会看到这样的订阅结果：

```
START - 17:00:43
Subscriber 1: Event - 0 at 17:00:44  # Second 1

Subscriber 1: Event - 1 at 17:00:45  # Second 2
Subscriber 2: Event - 0 at 17:00:45  #   Replay event 0
Subscriber 2: Event - 1 at 17:00:45  #   Replay event 1

Subscriber 1: Event - 2 at 17:00:46  # Second 3
Subscriber 2: Event - 2 at 17:00:46

Subscriber 1: Event - 3 at 17:00:47  # Second 4
Subscriber 2: Event - 3 at 17:00:47
Subscriber 3: Event - 2 at 17:00:47  #   Replay event 2
Subscriber 3: Event - 3 at 17:00:47  #   Replay event 3

Subscriber 1: Event - 4 at 17:00:48  # Second 5
Subscriber 2: Event - 4 at 17:00:48
Subscriber 3: Event - 4 at 17:00:48
```

我们刻意对打印结果做了分隔，方便观察。可以看到：

1. `interval`从第43秒开始计数；
2. 44秒时，Subscriber 1订阅到了第一个事件；
3. 45秒时，Subscriber 1订阅到了第二个事件，于此同时，Subscribe 2加入，它订阅到了自动回放的event 0和event 1；
4. 46秒时，Subscriber 1 2同时订阅到了event 2；
5. 47秒时，Subscriber 1 2同时订阅到了event 3。于此同时，Subscriber 3加入，它自动订阅到了回放的两个最近事件：event 2和event 3；

而这，就是`replay(N)`的用法。另外，如果我们希望给订阅者返回所有的历史事件，就可以把`replay`替换成`replayAll`：

重新执行下，就可以得到下面这样的结果：

```
START - 20:56:19
Subscriber 1: Event - 0 at 20:56:20
Subscriber 1: Event - 1 at 20:56:21
Subscriber 2: Event - 0 at 20:56:22
Subscriber 2: Event - 1 at 20:56:22
Subscriber 1: Event - 2 at 20:56:22
Subscriber 2: Event - 2 at 20:56:22
Subscriber 1: Event - 3 at 20:56:23
Subscriber 2: Event - 3 at 20:56:23
Subscriber 3: Event - 0 at 20:56:24  # Receive all history events
Subscriber 3: Event - 1 at 20:56:24
Subscriber 3: Event - 2 at 20:56:24
Subscriber 3: Event - 3 at 20:56:24
Subscriber 1: Event - 4 at 20:56:24
Subscriber 2: Event - 4 at 20:56:24
Subscriber 3: Event - 4 at 20:56:24
```

在理解了`replay`的工作方式之后，这次，我们主要关注加入Subscribe 3的时候，它订阅到的事件就好了。从打印的结果可以看到，在订阅到当前事件event 3之前，它首先收到了所有的历史事件，即event0/1/2。而这，就是`replayAll`的用法。

为事件的回放指定缓冲区

用事件值填充的缓冲区

但可以想象的是，随意使用`replayAll`很容易导致问题，特别是当历史事件很多的时候，就非常容易导致资源被耗尽。为此，我们还可以为事件的回放在特定的时间范围里，指定一个最大事件数量。这个operator叫做`buffer`。我们直接来看代码：

```
let interval = Observable<Int>
    .interval(1, scheduler: MainScheduler.instance)
    .buffer(timeSpan: 4, count: 2, scheduler: MainScheduler.instance)
```

要注意的是，使用了`buffer`之后，`interval`就不再是connectable observable了。它有三个参数：

- `timeSpan`：缓冲区的时间跨度，尽管`interval`每隔1秒钟发生一次事件，但经过`buffer`处理后，就变成了**最长`timeSpan`秒发生一次事件了**，事件的值，就是由所有缓存的事件值构成的数组。如果`timeSpan`过后没有任何事件发生，就向事件的订阅者发送一个空数组；
- `count`：缓冲区在`timeSpan`时间里可以缓存的最大事件数量，**当达到这个值之后，`buffer`就会立即把缓存的事件用一个数组发送给订阅者，并重置`timeSpan`**；
- `scheduler`：表示Observable事件序列发生在主线程，在后面的内容里，我们还会专门介绍RxSwift中的各种scheduler；

于是，现在的`interval`就表示每隔4秒，或者最大缓存两个事件，就发送给订阅者。把这个过程用序列图表示出来，就是这样的：

(connectable operators)[https://image.boxueio.com/more-connect-ops-1@2x.png]

理解了buffer的用法之后，我们用下面的代码来订阅interval：

```
print("START - " + stamp())

_ = interval.subscribe(onNext: {
    print("Subscriber 1: Event - \($0) at \(stamp())") })
```

执行一下，就可以看到类似下面这样的结果：

```
START - 22:51:34
Subscriber 1: Event - [0, 1] at 22:51:36
Subscriber 1: Event - [2, 3] at 22:51:38
Subscriber 1: Event - [4, 5] at 22:51:40
Subscriber 1: Event - [6, 7] at 22:51:42
Subscriber 1: Event - [8, 9] at 22:51:44
```

可以看到，从第34秒开始，到第36秒时，原始的interval就已经发生了两次事件，达到了buffer中count的约定值，因此没有等到4秒，Subscriber 1就收到了包含事件0和1的数组。此后，每隔两秒，缓冲区就会被填满，然后Subscriber 1就收到事件了。

由Observable填充的缓冲区

除了用缓冲区中的事件值作为数组发送给订阅者之外，我们还可以让某个时间段里的所有事件，组成一个新的Observable。完成这个功能的operator，叫做window。用序列图表示，就是这样的：

(connectable operators)[https://image.boxueio.com/more-connect-ops-2@2x.png]

这样，结果变换后的Observable，就变成了一个包含多个Sub-observable的事件序列。我们也可以把它理解为，每隔一个timeSpan就会打开一个新的窗口，处理最多count个事件。超过count的事件，就会放到下一个窗口周期进行处理。

理解了大体的想法之后，我们来看代码：

```
let interval = Observable<Int>
    .interval(1, scheduler: MainScheduler.instance)
    .window(timeSpan: 4, count: 4, scheduler: MainScheduler.instance)
```

这样，我们的事件序列就会每隔4秒打开一个窗口，每个窗口周期最多处理4个事件，然后关闭当前窗口，打开新的窗口。

> 要注意每一个窗口周期中的事件，是随着interval中的事件实时发送给订阅者的。而不是“攒”够了一个窗口周期的事件后，再发送一个Sub-observable。

我们用下面的代码观察这个效果：

```
print("START - " + stamp())

_ = interval.subscribe(onNext: {
    (subObservable: Observable<Int>) in
    print("============== Window Open ================")

    _ = subObservable.subscribe(onNext: {
        (value: Int) in
            print("Subscriber 1: Event - \(value) at \(stamp())")
        }, onCompleted: {
            print("============ Window Closed ==============")
        })
})
```

为了观察每一个Sub-observable生成到完成的过程，我们只好比较笨的在`interval`的订阅里，又订阅了一次。

注意这里，**不要用`flatMap`直接对`interval`进行变换**，否则，我们订阅到的，就是对`interval`所有事件变换后的Observable，这样，就观察不到每一个Sub-observable的`onCompleted`事件了。

执行一下，我们就会看到类似下面这样的结果：

```
START - 10:18:25
============ Window Open ===============
Subscriber 1: Event - 0 at 10:18:26
Subscriber 1: Event - 1 at 10:18:27
Subscriber 1: Event - 2 at 10:18:28
============ Window Closed ==============
============ Window Open ===============
Subscriber 1: Event - 3 at 10:18:29
Subscriber 1: Event - 4 at 10:18:30
Subscriber 1: Event - 5 at 10:18:31
Subscriber 1: Event - 6 at 10:18:32
============ Window Closed ==============
...
```

可以看到，从25秒的时候开始订阅，第一个窗口随即打开，1秒后收到第一个事件，于是，在第一个4秒的窗口周期，实际上，我们只能收到3个事件。而随后，就会每4个事件一个窗口周期了。

What's next?

以上，就是关于connectable operators的全部内容。在经历了14节的内容之后，关于各种常见operators用法的内容就结束了。它们分别是：*Transform operators*，*Filter operators*，*Combine operators*以及*Connectable operators*。现在，试着回想一下它们各自要完成的主要任务和应用场景，并确保自己对这些内容有一个比较清楚的理解。

接下来，我们就要装备上RxSwift，进一步深入App开发的领域，来了解如何通过RxSwift处理一些常用的UI交互行为，这是另外一个伴随RxSwift同步发布的组件，叫做RxCocoa。

Prev: 为什么需要connectable operator

使用connectable operator回放事件