了解常用的transform operators

RxSwift - step by step

← 返回视频列表 | 预计阅读时间: *10分钟* | ‹ PREVIOUS | NEXT ›

在接下来的两节里，我们讨论RxSwift中最重要的一类操作符，叫做*Transform operators*它们用来把一个Observable中的事件，变成另外一种形式。可以说，任何一个基于RxSwift开发的项目，都会使用这一类操作符。作为这个话题的第一部分，我们来看一些直观易懂的*Transform operators*。

toArray

首先，是`toArray`，这可以说是最简单的*Transform operator*。它把`Observable<T>`中所有的事件值，**在订阅的时候**，打包成一个`Array<T>`返回给订阅者。例如：

```
let bag = DisposeBag()

Observable.of(1, 2, 3)
    .toArray()
    .subscribe(onNext: {
        // Array<Int>
        print(type(of: $0))
        // [1, 2, 3]
        print($0)
    }).addDisposableTo(bag)
```

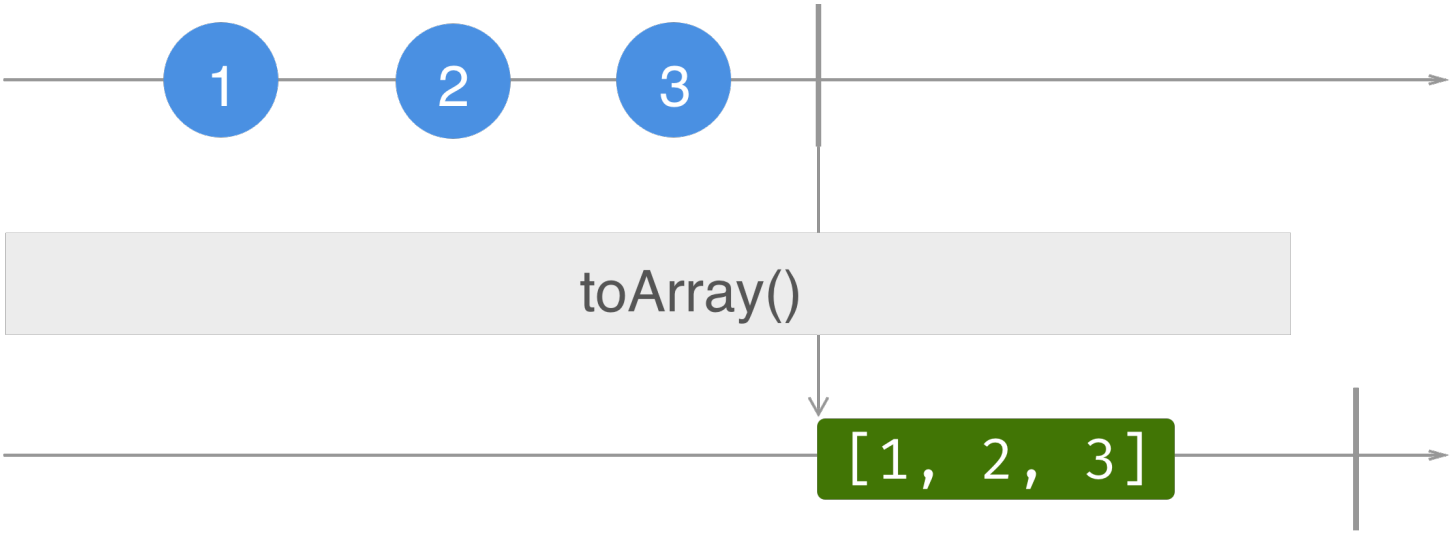
这里，有一点要注意的是，`toArray`的转换，**是在原始Observable结束的时候，把原始Observable中所有的事件值变成一个数组，只要原始Observable不结束，这个转换就不会发生**。来看下面这个例子：

```
let numbers = PublishSubject<Int>()

numbers.asObservable()
    .toArray()
    .subscribe(onNext: {
        print($0)
    }).addDisposableTo(bag)


numbers.onNext(1)
numbers.onNext(2)
numbers.onNext(3)
```

对于这个例子来说，在订阅的时候，使用了`toArray`，但此时，执行一下就会发现，我们不会订阅到任何值。只有在`numbers`结束之后，我们才会订阅到结果：

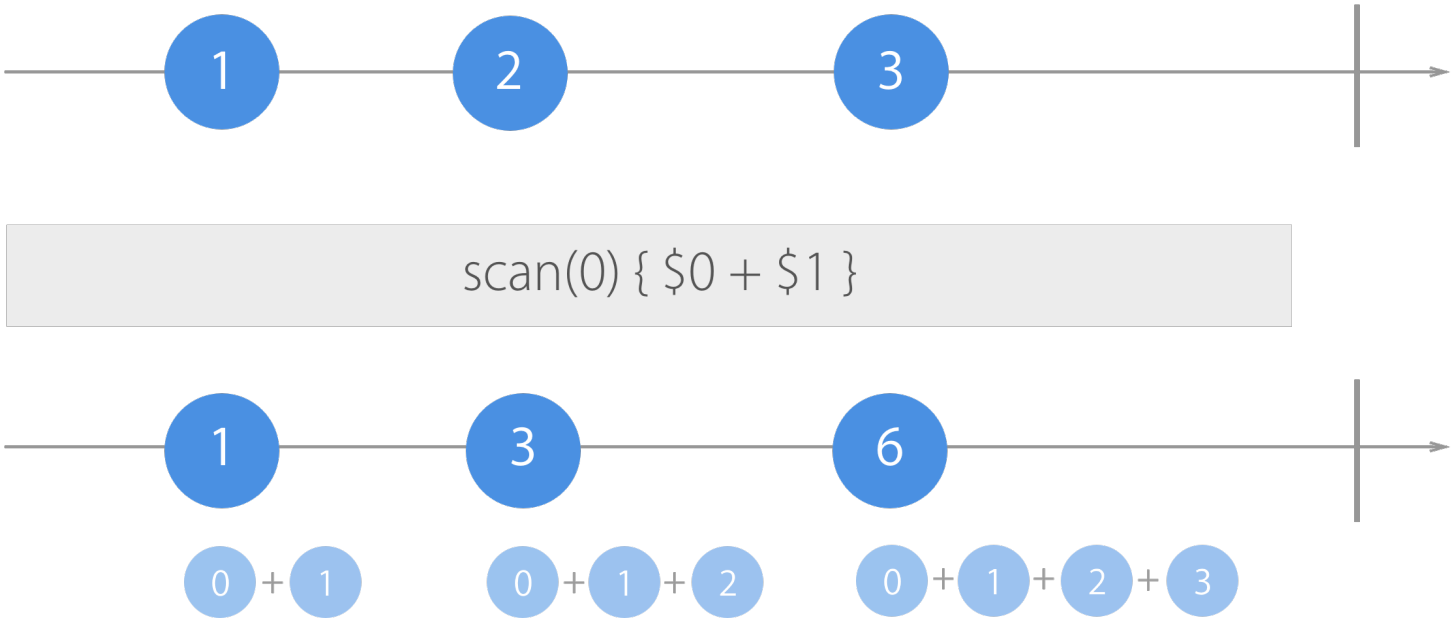
```
numbers.onNext(1)
numbers.onNext(2)
numbers.onNext(3)
numbers.onComplete()
```

这时，再执行一下，就可以在控制台看到数组`[1, 2, 3]`了。把这个过程用图表示，就是这样的：



scan

第二个*Transform operator*是`scan`，在之前改进用户授权的代码里，我们已经用过它了。给它设定一个初始值之后，它可以对Observable序列中的每一个事件进行“累加”，最终订阅到的，是“累加”之后的结果，如下图所示：



于是，下面的代码：

```
Observable.of(1, 2, 3).scan(0) {
        accumulatedValue, value in
         accumulatedValue + value
    }.subscribe(onNext: {
        print($0)
    }).addDisposableTo(bag)
```

打印出来的结果，就分别是1、3、6了。但是，就想上面图中展示的那样，**和`toArray`不同的是，`scan`在Observable每次有事件的时候都会执行**。因此，如果我们把之前的`numbers`使用`scan`变换：

```
let numbers = PublishSubject<Int>()

numbers.asObservable()
    .scan(0) { $0 + $1 }
    .subscribe(onNext: {
        print("Scan: \($0)")
    }).addDisposableTo(bag)
```

就会看到“Scan: 1”和“Scan: 2”两个结果，表示，每次有事件发生时，`scan`都会进行“累加”。

转换事件类型的map

除了把事件进行“累加”之外，我们也可以更自由的定义事件变换的行为。就像我们对集合中的元素进行变换一样，RxSwift也提供了一个`map` operator：

```
Observable.of(1, 2, 3).map {
    value in value * 2
}.subscribe(onNext: {
    print($0)
}).addDisposableTo(bag)
```

这样，在订阅的时候，我们会得到“2 4 6”，也就是说，我们订阅到的，是事件被`map`之后的Observable。`map`接受一个closure，而这个closure的参数，就是原Observable中的事件值。

另外，和我们在之前提到过滤型operators时讲过的`takeWhileWithIndex`一样，`map`也提供了一个`withIndex`的版本，像这样：

```
Observable.of(1, 2, 3)
    .mapWithIndex {
        value, index in
        index < 1 ? value * 2 : value
    }.subscribe(onNext: {
        print($0)
    }).addDisposableTo(bag)
```

`mapWithIndex`的closure接受两个参数，第一个表示事件本身，第二个表示事件在序列中的位置。因此，在上面的例子里，当把第一个发生的事件值乘以2，之后的都返回事件值本身。这样，就可以得到“2 2 3”这样的结果了。

What's next?

以上，就是一些常用并且简单直观的的*Transform operators*，理解了它们之后，下一节，我们来看一个不那么容易理解的operator：`flatMap`。

‹ Prev: Todo VI - 更好的处理授权提示 了解常用的transform operators Next: 为什么RxSwift也需要flatMap ›

关于我们

想循序渐进的跟上最新的技术趋势？想不为了学点东西到处搜索？想找个伙伴一起啃原版技术经典书？技术之外，还想了解高效的工作流技巧？甚至，工作之余，想找点儿东西放松心情？没问题，我们用4K开发视频，配以详尽的技术文档，以及精心准备的广播节目，让你渴望成长的技术需求，也是一种享受。

Email Address

10@boxue.io

客户服务

2085489246

相关链接

- 版权声明
- 用户隐私以及服务条款
- 京ICP备15057653号-1
- 京公网安备 11010802020752号

关注我们

在任何你常用的社交平台上关注我们，并告诉我们你的任何想法和建议！

邮件列表

订阅泊学邮件列表以了解泊学视频更新以及最新活动，我们不会向任何第三方公开你的邮箱！

Email address 立即订阅