# 为什么RxSwift也需要flatMap

RxSwift - step by step

← 返回视频列表 | 预计阅读时间: *12分钟* | ‹ PREVIOUS | NEXT ›

`flatMap`是个不太讨人喜欢的名字，因为集合里用它，Optional里用它，到了RxSwift里也用它，但是在这些不同的领域里，`flatMap`又都表达了不同的具体含义。你似乎很难简单的用一句话描述它要完成的功能。于是，如果你去看看`flatMap`在Rx里的定义，会发现是这样的：

> Transform the items emitted by an Observable into Observables, then flatten the emissions from those into a single Observable.

我敢说，如果你不是之前就理解`flatMap`的用法，几乎很难理解它究竟要表达什么。实际上，如果把`flatMap`的定义拆成两部分，就容易理解多了。

## 把序列中的事件变成新的Observable

首先，来看`flatMap`定义的前半句：*Transform the items emitted by an Observable into Observables*。如何把序列发生的事件变成新的Observable呢？我们来看RxSwift官方提供的例子：

```
struct Player {
    var score: Variable<Int>
}

let John = Player(score: Variable(70))
let Jole = Player(score: Variable(90))
```

这样，`John`和`Jole`就是两个独立的Observable。接下来，我们创建一个`PublishSubject<Player>`，然后订阅它：

```
let players = PublishSubject<Player>()

players.asObservable()
    .flatMap {
        $0.score.asObservable()
    }
    .subscribe(onNext: {
        print($0)
    })
    .addDisposableTo(bag)
```

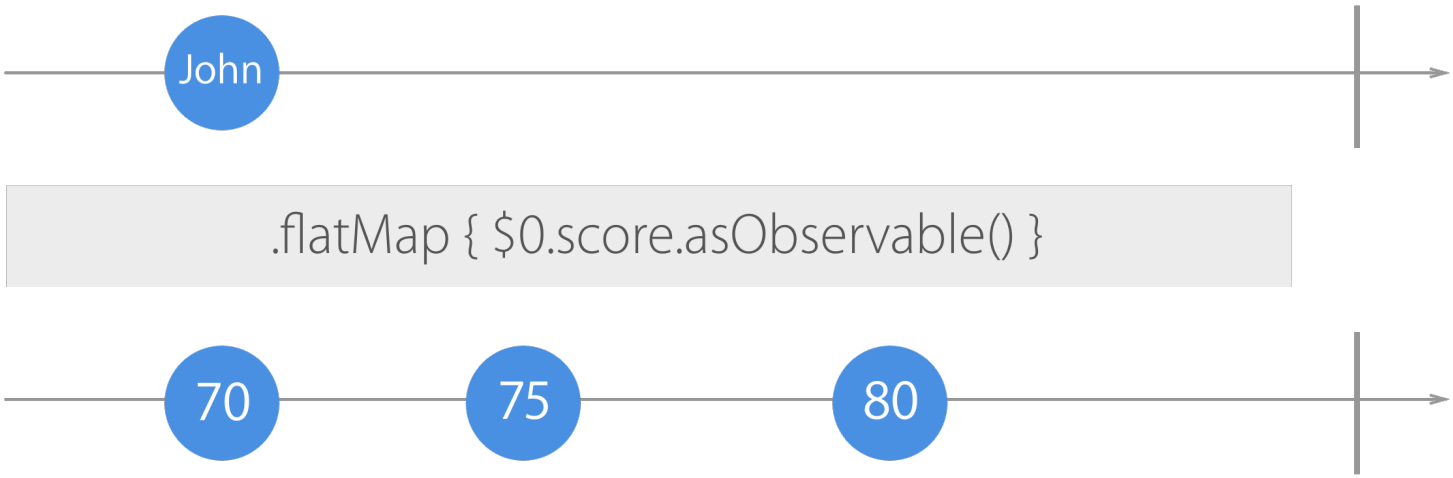
这时，我们把John加入到游戏：

```
players.onNext(John)
```

执行一下，就可以在控制台看到John的分数70了。然后，我们改变John的分数：

```
John.score.value = 75
John.score.value = 80
```

控制台上就会打印出“70 75 80”这样的结果。用序列图表示是这样的：




其中，John是player序列中发生的事件，通过flatMap我们把它变成了一个Observable<Int>。这就是flatMap定义前半句的含义：*Transform the items emitted by an Observable into Observables*。

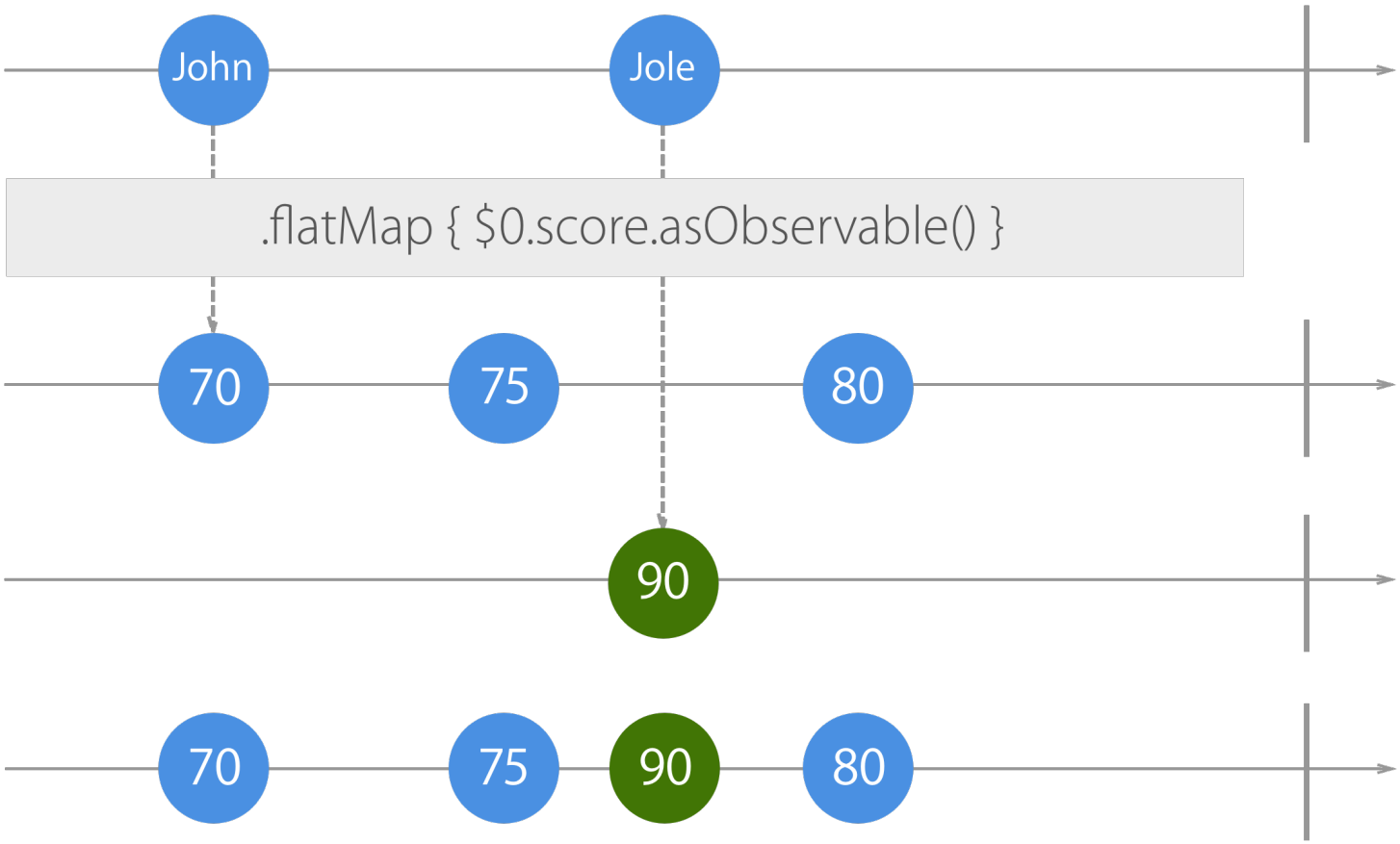
## 合成所有转换过的序列

接下来，来看flatMap定义的后半句：*then flatten the emissions from those into a single Observable*。为了理解这个flatten的过程，我们把Jole也添加进来，这里要特别注意**Jole添加进来的位置**：

```
players.onNext(John)

John.score.value = 75
players.onNext(Jole)
John.score.value = 80
```

先执行一下，会看到“70 75 90 80”这样的结果。然后，我们结合序列图，来看下为什么会这样：




首先，`flatMap`会把它原序列中的每个事件，都变换成一个Observable。因此，再加入了`Jole`之后，`flatMap`一共变换出了两个`Observable<Int>`，这就是我们之前讲过的`flatMap`定义的前半部分。

其次，当我们在75和80之间加入`Jole`的时候，`flatMap`会把`Jole`中事件的值和`John`中事件的值合并到一起，变成一个`Observable<Int>`，这种把两个`Observable<Int>`变成一个的过程，就是`flatMap`定义中，*flatten*的含义。

实际上，经过`flatMap`合并过的`Observable<Int>`会按发生的顺序，反映`John`和`Jole`中的所有事件。

## flatMapLatest

另外一个和`flatMap`类似的operator是`flatMapLatest`。当原序列中有新事件发生的时候，`flatMapLatest`就会自动取消上一个事件的订阅，然后转换到新事件的订阅。而`flatMap`则会保持原序列中的所有事件订阅。

可能这么说有点儿抽象，我们把之前的例子用`flatMapLatest`来试一下：

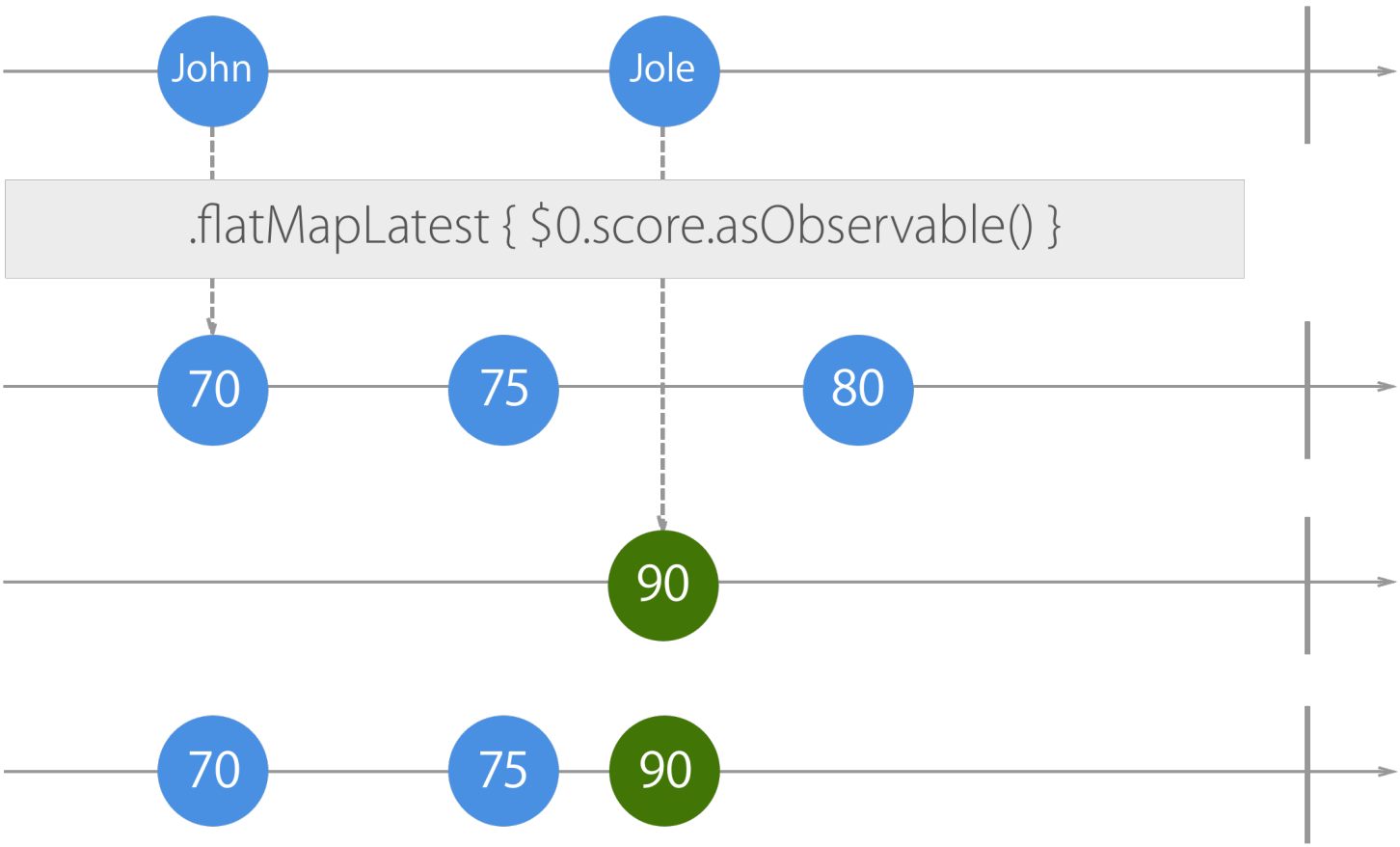
```
players.asObservable()
    .flatMapLatest {
        $0.score.asObservable()
    }
    .subscribe(onNext: {
        print($0)
    })
    .addDisposableTo(bag)
```

同样是这样的事件顺序：

```
players.onNext(John)

John.score.value = 75
players.onNext(Jole)
John.score.value = 80
```

这次，我们会得到“70 75 90”这样的结果，也就是说，`players`发生`Jole`事件之后，`flatMapLatest`就取消了对`John`的订阅，用序列图表示，就是这样的：




## What's next?

在结束这一节的内容之前，我们不妨思考一个问题。在什么情况下需要使用`flatMap`呢？为什么要把一个序列中的事件，变成另外一个事件序列呢？

简单来说，因为现实中很多事件都是异步发生的，而并不是像`Observable.of`创建的看起来像集合这样的。因此，当我们需要对异步发生的事件序列进行变换的时候，就需要订阅原来的事件序列，对异步发生的事件有所察觉。其中，网络编程就是一个最典型的例子。为了在请求一个网络资源后，根据服务器返回的结果对原事件序列进行变换，`flatMap`就是最好的选择。而这，就是我们接下来两节的内容。

‹ Prev: 了解常用的transform operators

为什么RxSwift也需要flatMap

Next: App demo I 一个Alamofire router的实现 ›