30 天精通 RxJS (17)：Observable Operators - switch, mergeAll, concatAll

2017年1月1日。6 分钟阅读

今天我们要讲三个operators，这三个operators 都是用来处理Higher Order Observable。

所谓的Higher Order Observable 就是指一个Observable 送出的元素还是一个Observable，就像是二维阵列一样，一个阵列中的每个元素都是阵列。如果用泛型来表达就像是

```
Observable<Observable<T>>
```

通常我们需要的是第二层Observable 送出的元素，所以我们希望可以把二维的Observable 改成一维的，像是下面这样

```
Observable<Observable<T>> => Observable<T>
```

其实想要做到这件事有三个方法switch、mergeAll 和concatAll，其中concatAll 我们在之前的文章已经稍微讲过了，今天这篇文章会讲解这三个operators 各自的效果跟差异。

运营商

concatAll

我们在讲简易拖拉的范例时就有讲过这个operator，concatAll 最重要的重点就是他会处理完前一个observable 才会在处理下一个observable，让我们来看一个范例

```
var click = Rx.Observable.fromEvent(document.body, 'click');
var source = click.map(e => Rx.Observable.interval(1000));

var example = source.concatAll();
example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); }
```

```
// 0
// 1
// 2
// 3
// 4
// 5 ...
```

JSBin

上面这段程式码，当我们点击画面时就会开始送出数值，如果用Marble Diagram 表示如下

```
click  : ---------c-c------------------c--..
         map(e => Rx.Observable.interval(1000))
source : ---------o-o------------------o--..
                   \ \
                    \ ----0----1----2----3----4--...
                     ----0----1----2----3----4--...
                     concatAll()
example: ----------------0----1----2----3----4--..
```

从Marble Diagram可以看得出来，当我们点击一下click事件会被转成一个observable而这个observable会每一秒送出一个递增的数值，当我们用concatAll之后会把二维的observable摊平成一维的observable ，但**concatAll会一个一个处理，一定是等前一个observable完成(complete)才会处理下一个observable**，因为现在送出observable是无限的永远不会完成(complete)，就导致他永远不会处理第二个送出的observable!

我们再看一个例子

```
var click = Rx.Observable.fromEvent(document.body, 'click');
var source = click.map(e => Rx.Observable.interval(1000).take(3));

var example = source.concatAll();
example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

现在我们把送出的observable 限制只取前三个元素，用Marble Diagram 表示如下

```
source : ---------o-o------------------o--..
                  \ \                   \
                   \ ----0----1----2|    ----0----1----2|
                    ----0----1----2|
                    concatAll()
example: ----------------0----1----2----0----1----2--..
```

這裡我們把送出的 observable 變成有限的，只會送出三個元素，這時就能看得出來 concatAll 不管兩個 observable 送出的時間多麼相近，一定會先處理前一個 observable 再處理下一個。

switch

switch 同樣能把二維的 observable 攤平成一維的，但他們在行為上有很大的不同，我們來看下面這個範例

```
var click = Rx.Observable.fromEvent(document.body, 'click');
var source = click.map(e => Rx.Observable.interval(1000));

var example = source.switch();
example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin

用 Marble Diagram 表示如下

```
click  : ---------c-c------------------c--..
         map(e => Rx.Observable.interval(1000))
source : ---------o-o------------------o--..
                  \ \                   \----0----1--...
                   \ ----0----1----2----3----4--...
                    ----0----1----2----3----4--...
                    switch()
example: -----------------0----1----2--------0----1--...
```

所以在這上面的 Marble Diagram 可以看得出來第一次送出的 observable 跟第二次送出的 observable 時間點太相近，導致第一個 observable 還來不及送出元素就直接被退訂了，當下一次送出 observable 就又會把前一次的 observable 退訂。

mergeAll

我們之前講過 merge 他可以讓多個 observable 同時送出元素，mergeAll 也是同樣的道理，它會把二維的 observable 轉成一維的，並且能夠同時處理所有的 observable，讓我們來看這個範例

```
var click = Rx.Observable.fromEvent(document.body, 'click');
var source = click.map(e => Rx.Observable.interval(1000));

var example = source.mergeAll();
example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

上面這段程式碼用 Marble Diagram 表示如下

```
click  : ---------c-c------------------c--..
         map(e => Rx.Observable.interval(1000))
source : ---------o-o------------------o--..
                  \ \                  \----0----1--...
                   \ ----0----1----2----3----4--...
                    ----0----1----2----3----4--...
                    switch()
example: ----------------00---11---22---33---(04)4--...
```

從 Marble Diagram 可以看出來，所有的 observable 是並行(Parallel)處理的，也就是說 mergeAll 不會像 switch 一樣退訂(unsubscribe)原先的 observable 而是並行處理多個 observable。以我們的範例來說，當我們點擊越多下，最後送出的頻率就會越快。

另外mergeAll 可以传入一个数值，这个数值代表他可以同时处理的observable 数量，我们来看一个例子

```
example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

这里我们送出的observable 改成取前三个，并且让mergeAll 最多只能同时处理2 个observable，用Marble Diagram 表示如下

```
click  : ---------c-c----------o----------..
         map(e => Rx.Observable.interval(1000))
source : ---------o-o----------c----------..
                  \ \          \----0----1----2|
                   \ ----0----1----2|
                    ----0----1----2|
                    mergeAll(2)
example: ----------------00---11---22---0----1----2--..
```

当mergeAll 传入参数后，就会等处理中的其中一个observable 完成，再去处理下一个。以我们的例子来说，前面两个observabel 可以被并行处理，但第三个observable 必须等到第一个observable 结束后，才会开始。

我们可以利用这个参数来决定要同时处理几个observable，如果我们传入 `1` 其行为就会跟 `concatAll` 是一模一样的，这点在原始码可以看到他们是完全相同的。

今日小结

今天介绍了三个可以处理High Order Observable 的方法，并讲解了三个方法的差异，不知道读者有没有收获呢？如果有任何问题欢迎在下方留言给我，感谢！

标签

JavaScript RxJS 可观察的 运算符 RxJS 30天

Series / 30 天精通 RxJS