30 天精通RxJS (12)：Observable Operator - scan, buffer

Dec 28th, 2016 . 4 mins read

今天要继续讲两个简单的transformation operators 并带一些小范例，这两个operators 都是实务上很常会用到的方法。

Operators

scan

scan 其实就是Observable 版本的reduce 只是命名不同。如果熟悉阵列操作的话，应该会知道原生的JS Array 就有reduce 的方法，使用方式如下

```
var arr = [1, 2, 3, 4];
var result = arr.reduce((origin, next) => {
    console.log(origin)
    return origin + next
}, 0);

console.log(result)
// 0
// 1
// 3
// 6
// 10
```

JSBin | JSFiddle

reduce 方法需要传两个参数，第一个是callback 第二个则是起始状态，这个callback 执行时，会传入两个参数一个是原本的状态，第二个是修改原本状态的参数，最后回传一个新的状态，再继续执行。

- 第二次执行callback，这时原本的状态(origin)就变成了1，next 为arr 的第二个元素2，相加之后变成3 回传并当作下一次的状态。
- 第三次执行callback，这时原本的状态(origin)就变成了3，next 为arr 的第三个元素3，相加之后变成6 回传并当作下一次的状态。
- 第三次执行callback，这时原本的状态(origin)就变成了6，next 为arr 的第四个元素4，相加之后变成10 回传并当作下一次的状态。
- 这时arr 的元素都已经遍历过了，所以不会直接把10 回传。

scan 整体的运作方式都跟reduce 一样，范例如下

```
var source = Rx.Observable.from('hello')
             .zip(Rx.Observable.interval(600), (x, y) => x);

var example = source.scan((origin, next) => origin + next, '');

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// h
// he
// hel
// hell
// hello
// complete
```

JSBin | JSFiddle

画成Marble Diagram

```
source : ----h----e----l----l----o|
    scan((origin, next) => origin + next, '')
example: ----h----(he)----(hel)----(hell)----(hello)|
```

这里可以看到第一次传入 'h' 跟 '' 相加，返回 'h' 当作下一次的初始状态，一直重复下去。

> 如果大家之前有到这里练习的话，会发现reduce被设计成一定回传阵列，而这个网页就是Jafar做的。

scan 很常用在状态的计算处理，最简单的就是对一个数字的加减，我们可以绑定一个button 的click 事件，并用map 把click event 转成1，之后送处scan 计算值再做显示。

这里笔者写了一个小范例，来示范如何做最简单的加减

```
const addButton = document.getElementById('addButton');
const minusButton = document.getElementById('minusButton');
const state = document.getElementById('state');

const addClick = Rx.Observable.fromEvent(addButton, 'click').mapTo(1);
const minusClick = Rx.Observable.fromEvent(minusButton, 'click').mapTo(

const numberState = Rx.Observable.empty()
  .startWith(0)
  .merge(addClick, minusClick)
  .scan((origin, next) => origin + next, 0)

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

JSBin | JSFiddle

这里我们用了两个button，一个是add 按钮，一个是minus 按钮。

我把这两个按钮的点击事件各建立了 `addClcik`, `minusClick` 两个observable，这两个observable直接mapTo(1)跟mapTo(-1)，代表被点击后会各自送出的数字！

接着我们用了 `empty()` 建立一个空的observable代表画面上数字的状态，搭配 `startWith(0)` 来设定初始值，接着用 `merge` 把两个observable合并透过scan处理之后的逻辑，最后在subscribe来更改画面的显示。

这个小范例用到了我们这几天讲的operators，包含mapTo, empty, startWith, merge 还有现在讲的scan，建议读者一定要花时间稍微练习一下。

buffer 是一组非常强大的operator

- buffer
- bufferCount
- bufferTime
- bufferToggle
- bufferWhen

这里比较常用到的是buffer, bufferCount 跟bufferTime 这三个，我们直接来看范例。

```
var source = Rx.Observable.interval(300);
var source2 = Rx.Observable.interval(1000);
var example = source.buffer(source2);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// [0,1,2]
// [3,4,5]
// [6,7,8]...
```

JSBin | JSFiddle

画成Marble Diagram 则像是

```
source : --0--1--2--3--4--5--6--7..
source2: ---------0---------1--------...
            buffer(source2)
example: ---------([0,1,2])---------([3,4,5])
```

buffer 要传入一个observable(source2)，它会把原本的observable (source)送出的元素缓存在阵列中，等到传入的observable(source2) 送出元素时，就会触发把缓存的元素送出。

这里的范例source2 是每一秒就会送出一个元素，我们可以改用bufferTime 简洁的表达，如下

```
var source = Rx.Observable.interval(300);
```





```
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// [0,1,2]
// [3,4,5]
// [6,7,8]...
```

JSBin | JSFiddle

除了用时间来作缓存外，我们更常用数量来做缓存，范例如下

```
var source = Rx.Observable.interval(300);
var example = source.bufferCount(3);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// [0,1,2]
// [3,4,5]
// [6,7,8]...
```

JSBin | JSFiddle 在实务上，我们可以用buffer来做某个事件的过滤，例如像是滑鼠连点才能真的执行，这里我们一样写了一个小范例

```
const button = document.getElementById('demo');
const click = Rx.Observable.fromEvent(button, 'click')
const example = click
                .bufferTime(500)
                .filter(arr => arr.length >= 2);

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





今日小结

今天我们介绍了两个operators 分别是scan, buffer，也做了两个小范例，不知道读者有没有收获呢？

Tags

JavaScript RxJS Observable Operator RxJS 30 Days

Prev Next