# 30 天精通RxJS (15)：Observable Operators - distinct, distinctUntilChanged

Dec 28th, 2016 . 4 mins read

新的一年马上就要到了，各位读者都去哪里跨年呢？笔者很可怜的只能一边写文章一边跨年，今天就简单看几个operators 让大家好好跨年吧！

昨天我们讲到了throttle 跟debounce 两个方法来做效能优化，其实还有另一个方法可以做效能的优化处理，那就是distinct。

## Operators

### distinct

如果会下SQL 指令的应该都对distinct 不陌生，它能帮我们把相同值的资料滤掉只留一笔，RxJS 里的distinct 也是相同的作用，让我们直接来看范例

```
var source = Rx.Observable.from(['a', 'b', 'c', 'a', 'b'])
            .zip(Rx.Observable.interval(300), (x, y) => x);
var example = source.distinct()

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

JSBin | JSFiddle

```
example: --a--b--c------|
```

从上面的范例可以看得出来，当我们用distinct 后，只要有重复出现的值就会被过滤掉。

另外我们可以传入一个selector callback function，这个callback function 会传入一个接收到的元素，并回传我们真正希望比对的值，举例如下

```
var source = Rx.Observable.from([{ value: 'a'}, { value: 'b' }, { value
            .zip(Rx.Observable.interval(300), (x, y) => x);
var example = source.distinct((x) => {
    return x.value
});

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

JSBin | JSFiddle

这里可以看到，因为source 送出的都是物件，而js 物件的比对是比对记忆体位置，所以在这个例子中这些物件永远不会相等，但实际上我们想比对的是物件中的value，这时我们就可以传入selector callback，来选择我们要比对的值。

> distinct 传入的callback 在RxJS 5 几个bate 版本中有过很多改变，现在网路上很多文章跟教学都是过时的，请读者务必小心！

实际上 `distinct()` 会在背地里建立一个Set，当接收到元素时会先去判断Set内是否有相同的值，如果有就不送出，如果没有则存到Set并送出。所以记得尽量不要直接把distinct用在一个无限的observable里，这样很可能会让Set越来越大，建议大家可以放第二个参数flushes，或用distinctUntilChanged

> 这里指的Set 其实是RxJS 自己实作的，跟ES6 原生的Set 行为也都一致，只是因为ES6 的Set

```
var source = Rx.Observable.from(['a', 'b', 'c', 'a', 'c'])
            .zip(Rx.Observable.interval(300), (x, y) => x);
var flushes = Rx.Observable.interval(1300);
var example = source.distinct(null, flushes);

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

JSBin | JSFiddle

這里我们用Marble Diagram 比较好表示

```
source : --a--b--c--a--c|
flushes: ------------0---...
        distinct(null, flushes);
example: --a--b--c-----c|
```

其实flushes observable 就是在送出元素时，会把distinct 的暂存清空，所以之后的暂存就会从头来过，这样就不用担心暂存的Set 越来愈大的问题，但其实我们平常不太会用这样的方式来处理，通常会用另一个方法distinctUntilChanged。

## distinctUntilChanged

distinctUntilChanged 跟distinct 一样会把相同的元素过滤掉，但distinctUntilChanged 只会跟最后一次送出的元素比较，不会每个都比，举例如下

```
var source = Rx.Observable.from(['a', 'b', 'c', 'c', 'b'])
            .zip(Rx.Observable.interval(300), (x, y) => x);
var example = source.distinctUntilChanged()
```





```
});
// a
// b
// c
// b
// complete
```

JSBin | JSFiddle

这里distinctUntilChanged 只会暂存一个元素，并在收到元素时跟暂存的元素比对，如果一样就不送出，如果不一样就把暂存的元素换成刚接收到的新元素并送出。

```
source : --a--b--c--c--b|
            distinctUntilChanged()
example: --a--b--c-----b|
```

从Marble Diagram 中可以看到，第二个c 送出时刚好上一个就是c 所以就被滤掉了，但最后一个b 则跟上一个不同所以没被滤掉。

distinctUntilChanged 是比较常在实务上使用的，最常见的状况是我们在做多方同步时。当我们有多个Client，且每个Client 有着各自的状态，Server 会再一个Client 需要变动时通知所有Client 更新，但可能某些Client 接收到新的状态其实跟上一次收到的是相同的，这时我们就可用 distinctUntilChanged 方法只处理跟最后一次不相同的讯息，像是多方通话、多装置的资讯同步都会有类似的情境。

# 今日小结

今天讲了两个distinct 方法，这两个方法平常可能用不太到，但在需求复杂的应用里是不可或缺的好方法，尤其要处理非常多人即时同步的情境下，这会是非常好用的方法，不知道读者们今天有没有收获呢？如果有任何问题，欢迎在下方留言给我，感谢！

Tags

JavaScript RxJS Observable Operator RxJS 30 Days

## Series / 30 天精通RxJS