# 30 天精通RxJS (14)：Observable Operator - throttle, debounce

Dec 28th, 2016 . 4 mins read

昨天讲到了在UI 操作上很常用的delay，今天我们接着要来讲另外两个也非常实用 operators，尤其在做效能优化时更是不可或缺的好工具！

## Operators

### debounce

跟buffer、bufferTime 一样，Rx 有debounce 跟debounceTime 一个是传入observable 另一个则是传入毫秒，比较常用到的是debounceTime，这里我们直接来看一个范例

```
var source = Rx.Observable.interval(300).take(5);
var example = source.debounceTime(1000);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// 4
// complete
```

JSBin | JSFiddle

这里只印出 4 然后就结束了，因为**debounce运作的方式是每次收到元素，他会先把元素cache住并等待一段时间，如果这段时间内已经没有收到任何元素，则把元素送出；如果这段时间内又收到新的元素，则会把原本cache住的元素释放掉并重新计时，不断反覆。**

以现在这个范例来讲，我们每300 毫秒就会送出一个数值，但我们的debounceTime 是1000 毫秒，也就是说每次debounce 收到元素还等不到1000 毫秒，就会收到下一个新元素，然后重新等待1000 毫秒，如此重复直到第五个元素送出时，observable 结束(complete)了，debounce 就

```
source : --0--1--2--3--4|
        debounceTime(1000)
example: --------------4|
```

debounce会在收到元素后等待一段时间，这很适合用来处理**间歇行为**，间歇行为就是指这个行为是一段一段的，例如要做Auto Complete时，我们要打字搜寻不会一直不断的打字，可以等我们停了一小段时间后再送出，才不会每打一个字就送一次request！

从Marble Diagram 可以看得出来，第一次送出元素的时间变慢了，虽然在这里看起来没什么用，但是在UI 操作上是非常有用的，这个部分我们最后示范。

```
const searchInput = document.getElementById('searchInput');
const theRequestValue = document.getElementById('theRequestValue');

Rx.Observable.fromEvent(searchInput, 'input')
  .map(e => e.target.value)
  .subscribe((value) => {
    theRequestValue.textContent = value;
    // 在這裡發 request
  })
```

如果用上面这段程式码，就会每打一个字就送一次request，当很多人在使用时就会对server 造成很大的负担，实际上我们只需要使用者最后打出来的文字就好了，不用每次都送，这时就能用 debounceTime 做优化。

```
const searchInput = document.getElementById('searchInput');
const theRequestValue = document.getElementById('theRequestValue');

Rx.Observable.fromEvent(searchInput, 'input')
  .debounceTime(300)
  .map(e => e.target.value)
  .subscribe((value) => {
    theRequestValue.textContent = value;
    // 在這裡發 request
  })
```

JSBin | JSFiddle

这里建议大家到JSBin亲手试试，可以把 debounceTime(300) 注解掉，看看前后的差异。

为上有很大的不同。

跟debounce 一样RxJS 有throttle 跟throttleTime 两个方法，一个是传入observable 另一个是传入毫秒，比较常用到的也是throttleTime，让我们直接来看范例

```
var source = Rx.Observable.interval(300).take(5);
var example = source.throttleTime(1000);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// 0
// 4
// complete
```

JSBin | JSFiddle

跟debounce 的不同是throttle 会先开放送出元素，等到有元素被送出就会沉默一段时间，等到时间过了又会开放发送元素。

throttle比较像是控制行为的最高频率，也就是说如果我们设定**1000毫秒**，那该事件频率的最大值就是**每秒触发一次**不会再更快，debounce则比较像是必须等待的时间，要等到一定的时间过了才会收到元素。

throttle更适合用在**连续性行为**，比如说UI动画的运算过程，因为UI动画是连续的，像我们之前在做拖拉时，就可以加上 `throttleTime(12)` 让mousemove event不要发送的太快，避免画面更新的速度跟不上样式的切换速度。

> 浏览器有一个 requestAnimationFrame API是专门用来优化UI运算的，通常用这个的效果会比throttle好，但并不是绝对还是要看最终效果。

> RxJS 也能用requestAnimationFrame 做优化，而且使用方法很简单，这个部份会在 Scheduler 提到。

# 今日小结

今天介绍了两个非常实用的方法，可以帮我们做程式的效能优化，而且使用方式非常的简单，不

JavaScript RxJS Observable Operator RxJS 30 Days

Prev Next