# 30 天精通RxJS (09)：Observable Operator - skip, takeLast, last, concat, startWith, merge

Dec 25th, 2016 . 4 mins read

今天是美好的圣诞节，先祝读者们圣诞快乐！为了让大家在圣诞节好好的陪家人，所以今天的文章内容就轻松点，让我们简单介绍几个的operators 就好了。

这是【30天精通RxJS】的09篇，如果还没看过08篇可以往这边走： 30天精通RxJS (08)：简易拖拉实作- take, first, takeUntil, concatAll

## Operators

### skip

我们昨天介绍了 `take` 可以取前几个送出的元素，今天介绍可以略过前几个送出元素的operator: `skip` ，范例如下：

```
var source = Rx.Observable.interval(1000);
var example = source.skip(3);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// 3
// 4
// 5...
```

JSBin | JSFiddle

```
example: -------------------3----4----5--...
```

## takeLast

除了可以用take 取前几个之外，我们也可以倒过来取最后几个，范例如下：

```
var source = Rx.Observable.interval(1000).take(6);
var example = source.takeLast(2);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// 4
// 5
// complete
```

這裡我們先取了前 6 個元素，再取最後兩個。所以最後會送出 4, 5, complete，這裡有一個重點，就是 takeLast 必須等到整個 observable 完成(complete)，才能知道最後的元素有哪些，並且**同步送出**，如果用 Marble Diagram 表示如下

```
source : ----0----1----2----3----4----5|
                takeLast(2)
example: ------------------------------(45)|
```

這裡可以看到 takeLast 後，比須等到原本的 observable 完成後，才立即同步送出 4, 5, complete。

## last

跟 `take(1)` 相同，我們有一個 `takeLast(1)` 的簡化寫法，那就是 `last()` 用來取得最後一個元素。

```
var source = Rx.Observable.interval(1000).take(6);
var example = source.last();
```

```
    complete: () => { console.log('complete'); }
});
// 5
// complete
```

用 Marble Diagram 表示如下

```
source : ----0----1----2----3----4----5|
                    last()
example: ------------------------------(5)|
```

## concat

concat 可以把多個 observable 實例合併成一個，範例如下

```
var source = Rx.Observable.interval(1000).take(3);
var source2 = Rx.Observable.of(3)
var source3 = Rx.Observable.of(4,5,6)
var example = source.concat(source2, source3);

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

JSBin | JSFiddle

跟 concatAll 一樣，必須先等前一個 observable 完成(complete)，才會繼續下一個，用

```
source3: (456)|
            concat()
example: ----0----1----2(3456)|
```

另外 concat 還可以當作靜態方法使用

```
var source = Rx.Observable.interval(1000).take(3);
var source2 = Rx.Observable.of(3);
var source3 = Rx.Observable.of(4,5,6);
var example = Rx.Observable.concat(source, source2, source3);

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

JSBin | JSFiddle

## startWith

`startWith` 可以在 observable 的一開始塞要發送的元素，有點像 `concat` 但參數不是 observable 而是要發送的元素，使用範例如下

```
var source = Rx.Observable.interval(1000);
var example = source.startWith(0);

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





```
source : ----0----1----2----3--...
                startWith(0)
example: (0)----0----1----2----3--...
```

記得 startWith 的值是一開始就同步發出的，這個 operator 很常被用來保存程式的起始狀態！

## merge

merge 跟 concat 一樣都是用來合併 observable，但他們在行為上有非常大的不同！

让我们直接来看例子吧

```
var source = Rx.Observable.interval(500).take(3);
var source2 = Rx.Observable.interval(300).take(6);
var example = source.merge(source2);

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

JSBin | JSFiddle

上面可以看得出来，merge 把多个observable同时处理，这跟 concat 一次处理一个observable是完全不一样的，由于是同时处理行为会变得较为复杂，这里我们用Marble Diagram会比较好解释。

```
            merge()
example: --0-01--21-3--(24)--5|
```

这里可以看到 merge 之后的example在时间序上同时在跑source与source2，当两件事情同时发生时，会同步送出资料(被merge的在后面)，当两个observable都结束时才会真的结束。

merge 同样可以当作静态方法用

```
var source = Rx.Observable.interval(500).take(3);
var source2 = Rx.Observable.interval(300).take(6);
var example = Rx.Observable.merge(source, source2);

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

merge 的逻辑有点像是OR(||)，就是当两个observable 其中一个被触发时都可以被处理，这很常用在一个以上的按钮具有部分相同的行为。

例如一个影片播放器有两个按钮，一个是暂停(II)，另一个是结束播放(口)。这两个按钮都具有相同的行为就是影片会被停止，只是结束播放会让影片回到00 秒，这时我们就可以把这两个按钮的事件merge 起来处理影片暂停这件事。

```
var stopVideo = Rx.Observable.merge(stopButton, endButton);

stopVideo.subscribe(() => {
    // 暂停播放影片
})
```

# 今日小结

今天介绍的六个operators 都是平时很容易用到的，我们之后的范例也有机会再遇到。希望读者们能自己试试这些方法，之后使用时会比较有印象！

不知道读者今天有没有收获呢？如果有任何问题，欢迎在下方留言给我，谢谢。

JavaScript RxJS Observable Operator RxJS 30 Days

Prev Next