30 天精通RxJS (08)：简易拖拉实作- take, first, takeUntil, concatAll

2016年12月24日。5 分钟阅读

如果是你会如何实作拖拉的功能？

这是【30天精通RxJS】的08篇，如果还没看过07篇可以往这边走： 30天精通RxJS (07)：Observable Operators & Marble Diagrams

今天建议大家直接看影片 是

我们今天要接着讲take, first, takeUntil, concatAll 这四个operators，并且实作一个简易的拖拉功能。

运营商

采取

take 是一个很简单的operator，顾名思义就是取前几个元素后就结束，范例如下

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

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

JSBin | 的jsfiddle

用 Marble diagram 表示如下

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

第一

first 会取observable 送出的第1 个元素之后就直接结束，行为跟take(1) 一致。

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

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

// 0
// complete
```

JSBin | 的jsfiddle

用 Marble diagram 表示如下

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

takeUntil

在实务上takeUntil 很常使用到，他可以在某件事情发生时，让一个observable 直送出完成(complete)讯息，范例如下

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

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

JSBin | 的jsfiddle

這裡我們一開始先用 `interval` 建立一個 observable，這個 observable 每隔 1 秒會送出一個從 0 開始遞增的數值，接著我們用 `takeUntil` ，傳入另一個 observable。

當 `takeUntil` 傳入的 observable 發送值時，原本的 observable 就會直接進入完成(complete)的狀態，並且發送完成訊息。也就是說上面這段程式碼的行為，會先每 1 秒印出一個數字(從 0 遞增)直到我們點擊 body 為止，他才會送出 complete 訊息。

如果畫成 Marble Diagram 則會像下面這樣

```
source : -----0-----1-----2------3--
click  : ----------------------c----
               takeUntil(click)
example: -----0-----1-----2----|
```

當 click 一發送元素的時候，observable 就會直接完成(complete)。

concatAll

有時我們的 Observable 送出的元素又是一個 observable，就像是二維陣列，陣列裡面的元素是陣列，這時我們就可以用 `concatAll` 把它攤平成一維陣列，大家也可以直接把 concatAll 想成把所有元素 concat 起來。

```
var click = Rx.Observable.fromEvent(document.body, 'click');
var source = click.map(e => Rx.Observable.of(1,2,3));

var example = source.concatAll();
example.subscribe({
```

JSBin | JSFiddle

這個範例我們每點擊一次 body 就會立刻送出 1,2,3，如果用 Marble diagram 表示則如下

```
click  : ------c------------c--------

         map(e => Rx.Observable.of(1,2,3))

source : ------o------------o--------
                \            \
                 (123)|       (123)|

                   concatAll()

example: ------(123)--------(123)------------
```

這裡可以看到 `source` observable 內部每次發送的值也是 observable，這時我們用 concatAll 就可以把 source 攤平成 example。

這裡需要注意的是 `concatAll` 會處理 source 先發出來的 observable，必須等到這個 observable 結束，才會再處理下一個 source 發出來的 observable，讓我們用下面這個範例說明。

```
var obs1 = Rx.Observable.interval(1000).take(5);
var obs2 = Rx.Observable.interval(500).take(2);
var obs3 = Rx.Observable.interval(2000).take(1);

var source = Rx.Observable.of(obs1, obs2, obs3);

var example = source.concatAll();

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

```
// 0
// 1
// 0
// complete
```

JSBin | JSFiddle

這裡可以看到 `source` 會送出 3 個 observable，但是 `concatAll` 後的行為永遠都是先處理第一個 observable，**等到當前處理的結束後才會再處理下一個。**

用 Marble diagram 表示如下

```
source : (o1                 o2      o3)|
           \                  \       \
            --0--1--2--3--4|   -0-1|   ----0|

                concatAll()

example: --0--1--2--3--4-0-1----0|
```

簡易拖拉

當學完前面幾個 operator 後，我們就很輕鬆地做出拖拉的功能，先讓我們來看一下需求

1. 首先畫面上有一個元件(#drag)
2. 當滑鼠在元件(#drag)上按下左鍵(mousedown)時，開始監聽滑鼠移動(mousemove)的位置
3. 當滑鼠左鍵放掉(mouseup)時，結束監聽滑鼠移動
4. 當滑鼠移動(mousemove)被監聽時，跟著修改元件的樣式屬性

第一步我已經完成了，大家可以直接到以下兩個連結做練習

- JSBin
- JSFiddle

第二步我們要先取得各個 DOM 物件，元件(#drag) 跟 body。

要取得 body 的原因是因為滑鼠移動(mousemove)跟滑鼠左鍵放掉(mouseup)都應該是在整個 body 監聽。

第三步我們寫出各個會用到的監聽事件，並用 `fromEvent` 來取得各個 observable。

- 對 #drag 監聽 mousedown
- 對 body 監聽 mouseup
- 對 body 監聽 mousemove

```
const mouseDown = Rx.Observable.fromEvent(dragDOM, 'mousedown');
const mouseUp = Rx.Observable.fromEvent(body, 'mouseup');
const mouseMove = Rx.Observable.fromEvent(body, 'mousemove');
```

> 記得還沒 `subscribe` 之前都不會開始監聽，一定會等到 subscribe 之後 observable 才會開始送值。

第四步開始寫邏輯

當 mouseDown 時，轉成 mouseMove 的事件

```
const source = mouseDown.map(event => mouseMove)
```

mouseMove 要在 mouseUp 後結束

加上 `takeUntil(mouseUp)`

```
const source = mouseDown
               .map(event => mouseMove.takeUntil(mouseUp))
```

這時 source 大概長像這樣

```
source: -------e--------------e-----
                \              \
                  --m-m-m-m|     -m--m-m--m-m|
```

> m 代表 mousemove event





```
                .concatAll();
```

用 map 把 mousemove event 轉成 x,y 的位置，並且訂閱。

```
source
.map(m => {
    return {
        x: m.clientX,
        y: m.clientY
    }
})
.subscribe(pos => {
        dragDOM.style.left = pos.x + 'px';
    dragDOM.style.top = pos.y + 'px';
})
```

到这里我们就已经完成了简易的拖拉功能了!完整的程式码如下

```
const dragDOM = document.getElementById('drag');
const body = document.body;

const mouseDown = Rx.Observable.fromEvent(dragDOM, 'mousedown');
const mouseUp = Rx.Observable.fromEvent(body, 'mouseup');
const mouseMove = Rx.Observable.fromEvent(body, 'mousemove');

mouseDown
  .map(event => mouseMove.takeUntil(mouseUp))
  .concatAll()
  .map(event => ({ x: event.clientX, y: event.clientY }))
  .subscribe(pos => {
        dragDOM.style.left = pos.x + 'px';
    dragDOM.style.top = pos.y + 'px';
  })
```

不知道读者有没有感受到，我们整个程式码不到15 行，而且只要能够看懂各个operators，我们程式可读性是非常的高。

虽然这只是一个简单的拖拉实现，但已经展示出RxJS 带来的威力，它让我们的程式码更加的简

    

今日小结

我们今天介绍了四个operators 分别是take, first, takeUntil, concatAll，并且完成了一个简易的拖拉功能，我们之后会把这个拖拉功能做得更完整，并且整合其他功能！

不知道读者今天有没有收获？如果有任何问题，欢迎在下方留言给我！如果你喜欢这篇文章，请至标题旁帮我按个星星 + like，谢谢。

标签

JavaScript RxJS 可观察到的 拖放 RxJS 30天

上一页 下一个