30 天精通RxJS (10)：Observable Operator - combineLatest, withLatestFrom, zip

Dec 25th, 2016 . 4 mins read

昨天我们最后讲到了`merge` 的用法，它的逻辑就像是OR(||)一样，可以把多个observable 合并且同时处理，当其中任合一个observable 送出元素时，我们都做相同的处理。今天我们要讲的三个operators 则像是AND(&&) 逻辑，它们都是在多个元素送进来时，只输出一个新元素，但各自的行为上仍有差异，需要读者花点时间思考，建议在头脑清醒时阅读本篇文章。

Operators

combineLatest

首先我们要介绍的是combineLatest，它会取得各个observable 最后送出的值，再输出成一个值，我们直接看范例会比较好解释。

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

var example = source.combineLatest(newest, (x, y) => x + y);

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

JSBin | JSFiddle

大家第一次看到这个output 应该都会很困惑，我们直接来看Marble Diagram 吧！

```
source : ----0----1----2|
newest : --0--1--2--3--4--5|

    combineLatest(newest, (x, y) => x + y);

example: ----01--23-4--(56)--7|
```

首先 `combineLatest` 可以接收多个observable，最后一个参数是callback function，这个callback function接收的参数数量跟合并的observable数量相同，依照范例来说，因为我们这里合并了两个observable所以后面的callback function就接收x, y两个参数，x会接收从source发送出来的值，y会接收从newest发送出来的值。

最后一个重点就是一定会等两个observable都**曾有送值**出来才会呼叫我们传入的callback，所以这段程式是这样运行的

- newest送出了 `0` ，但此时source并没有送出过任何值，所以不会执行callback
- source 送出了 `0` ，此時 newest 最後一次送出的值為 `0` ，把這兩個數傳入 callback 得到 `0` 。
- newest 送出了 `1` ，此時 source 最後一次送出的值為 `0` ，把這兩個數傳入 callback 得到 `1` 。
- newest 送出了 `2` ，此時 source 最後一次送出的值為 `0` ，把這兩個數傳入 callback 得到 `2` 。
- source 送出了 `1` ，此時 newest 最後一次送出的值為 `2` ，把這兩個數傳入 callback 得到 `3` 。
- newest 送出了 `3` ，此時 source 最後一次送出的值為 `1` ，把這兩個數傳入 callback 得到 `4` 。
- source 送出了 `2` ，此時 newest 最後一次送出的值為 `3` ，把這兩個數傳入 callback 得到 `5` 。
- source 結束，但 newest 還沒結束，所以 example 還不會結束。
- newest 送出了 `4` ，此時 source 最後一次送出的值為 `2` ，把這兩個數傳入 callback 得到 `6`

- newest 結束，因為 source 也結束了，所以 example 結束。

不管是 source 還是 newest 送出值來，只要另一方曾有送出過值(有最後的值)，就會執行 callback 並送出新的值，這就是 combineLatest。

combineLatest 很常用在運算多個因子的結果，例如最常見的 BMI 計算，我們身高變動時就拿上一次的體重計算新的 BMI，當體重變動時則拿上一次的身高計算 BMI，這就很適合用 combineLatest 來處理！

zip

在講 withLatestFrom 之前，先讓我們先來看一下 zip 是怎麼運作的，zip 會取每個 observable 相同順位的元素並傳入 callback，也就是說每個 observable 的第 n 個元素會一起被傳入 callback，這裡我們同樣直接用範例講解會比較清楚

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

var example = source.zip(newest, (x, y) => x + y);

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

JSBin | JSFiddle

Marble Diagram 長這樣

```
source : ----0----1----2|
newest : --0--1--2--3--4--5|
    zip(newest, (x, y) => x + y)
example: ----0----2----4|
```

- newest 送出了**第一個**值 `0`，但此時 source 並沒有送出**第一個**值，所以不會執行 callback。
- source 送出了**第一個**值 `0`，newest 之前送出的**第一個**值為 `0`，把這兩個數傳入 callback 得到 `0`。
- newest 送出了**第二個**值 `1`，但此時 source 並沒有送出**第二個**值，所以不會執行 callback。
- newest 送出了**第三個**值 `2`，但此時 source 並沒有送出**第三個**值，所以不會執行 callback。
- source 送出了**第二個**值 `1`，newest 之前送出的**第二個**值為 `1`，把這兩個數傳入 callback 得到 `2`。
- newest 送出了**第四個**值 `3`，但此時 source 並沒有送出**第四個**值，所以不會執行 callback。
- source 送出了**第三個**值 `2`，newest 之前送出的**第三個**值為 `2`，把這兩個數傳入 callback 得到 `4`。
- source 結束 example 就直接結束，因為 source 跟 newest 不會再有對應順位的值

zip 會把各個 observable 相同順位送出的值傳入 callback，這很常拿來做 demo 使用，比如我們想要間隔 100ms 送出 'h', 'e', 'l', 'l', 'o'，就可以這麼做

```
var source = Rx.Observable.from('hello');
var source2 = Rx.Observable.interval(100);

var example = source.zip(source2, (x, y) => x);
```

這裡的 Marble Diagram 就很簡單

```
source : (hello)|
source2: -0-1-2-3-4-...
        zip(source2, (x, y) => x)
example: -h-e-l-l-o|
```

這裡我們利用 zip 來達到原本只能同步送出的資料變成了非同步的，很適合用在建立示範用的資料。





withLatestFrom

withLatestFrom 運作方式跟 combineLatest 有點像，只是他有主從的關係，只有在主要的 observable 送出新的值時，才會執行 callback，附隨的 observable 只是在背景下運作。讓我們看一個例子

```
var main = Rx.Observable.from('hello').zip(Rx.Observable.interval(500)
var some = Rx.Observable.from([0,1,0,0,0,1]).zip(Rx.Observable.interva

var example = main.withLatestFrom(some, (x, y) => {
    return y === 1 ? x.toUpperCase() : x;
});

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

JSBin | JSFiddle

先看一下 Marble Diagram

```
main   : ----h----e----l----l----o|
some   : --0--1--0--0--0--1|

withLatestFrom(some, (x, y) =>  y === 1 ? x.toUpperCase() : x);

example: ----h----e----l----L----O|
```

withLatestFrom 會在 main 送出值的時候執行 callback，但請注意如果 main 送出值時 some 之前沒有送出過任何值 callback 仍然不會執行！

這裡我們在 main 送出值時，去判斷 some 最後一次送的值是不是 1 來決定是否要切換大小寫，執行步驟如下

- main送出了 h ，此时some上一次送出的值为 0 ，把这两个参数传入callback得到 h 。
- main送出了 e ，此时some上一次送出的值为 0 ，把这两个参数传入callback得到 e 。

- main送出了 0，此時some上一次送出的值為 1，把這兩個參數傳入callback得到 0。

withLatestFrom 很常用在一些checkbox 型的功能，例如说一个编辑器，我们开启粗体后，打出来的字就都要变粗体，粗体就像是some observable，而我们打字就是main observable。

今日小结

今天介绍了三个合并用的operators，这三个operators 的callback 都会依照合并的observable 数量来传入参数，如果我们合并了三个observable，callback 就会有三个参数，而不管合并几个observable 都会只会回传一个值。

这几个operators 需要花比较多的时间思考，读者们不用硬记他的运作行为，只要稍微记得有这些operators 可以用就可以了。等到真的要用时，再重新回来看他们的运作方式做选择。

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，谢谢！

Tags

JavaScript RxJS Observable Operator RxJS 30 Days

Prev Next