30 天精通 RxJS (18)：Observable Operators - switchMap, mergeMap, concatMap

2017年1月1日。6 分钟阅读

今天我们要讲三个非常重要的operators，这三个operators 在很多的RxJS 相关的library 的使用范例上都会看到。很多初学者在使用这些library 时，看到这三个operators 很可能就放弃了，但其实如果有把这个系列的文章完整看过的话，现在应该就能很好接受跟理解。

运营商

concatMap

concatMap 其实就是map 加上concatAll 的简化写法，我们直接来看一个范例

```
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source
                .map(e => Rx.Observable.interval(1000).take(3))
                .concatAll();

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

上面这个范例就可以简化成

```
var source = Rx.Observable.fromEvent(document.body, 'click');
```

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

前后两个行为是一致的，记得concatMap 也会先处理前一个送出的observable 在处理下一个observable，画成Marble Diagram 如下

```
source : -----------c--c------------------...
        concatMap(c => Rx.Observable.interval(100).take(3))
example: -------------0-1-2-0-1-2---------...
```

这样的行为也很常被用在发送request 如下

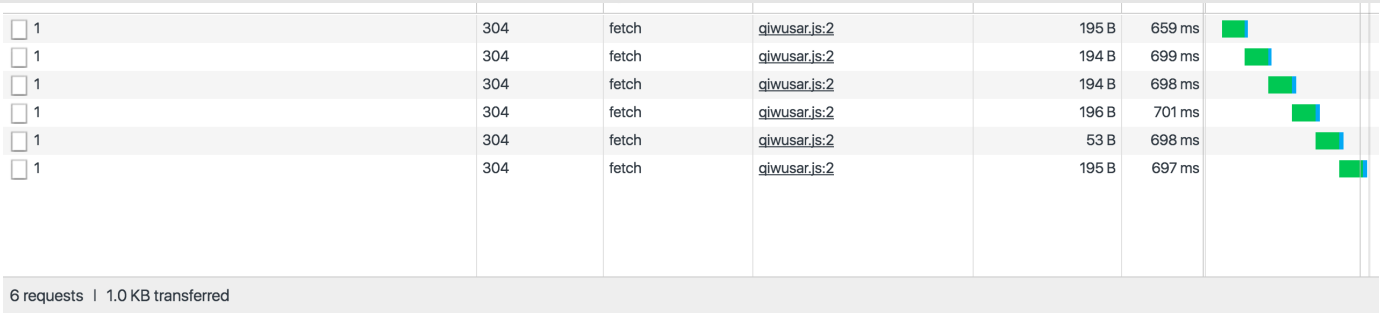
```
function getPostData() {
    return fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then(res => res.json())
}
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source.concatMap(
                    e => Rx.Observable.from(getPostData()));

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

JSBin | 的jsfiddle

这里我们每点击一下画面就会送出一个HTTP request，如果我们快速的连续点击，大家可以在开发者工具的network 看到每个request 是等到前一个request 完成才会送出下一个request，如下图



> 这里建议把网速模拟调到最慢

从network 的图形可以看得出来，第二个request 的发送时间是接在第一个request 之后的，我们可以确保每一个request 会等前一个request 完成才做处理。

concatMap 还有第二个参数是一个selector callback，这个callback 会传入四个参数，分别是

1. 外部 observable 送出的元素
2. 内部observable 送出的元素
3. 外部 observable 送出元素的 index
4. 内部observable 送出元素的index

回传值我们想要的值，范例如下

```
function getPostData() {
    return fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then(res => res.json())
}
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source.concatMap(
                e => Rx.Observable.from(getPostData()),
                (e, res, eIndex, resIndex) => res.title);

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

title，這個方法很適合用在 response 要選取的值跟前一個事件或順位(index)相關時。

switchMap

switchMap 其實就是 map 加上 switch 簡化的寫法，如下

```
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source
                .map(e => Rx.Observable.interval(1000).take(3))
                .switch();

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

上面的程式碼可以簡化成

```
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source
                .switchMap(
                    e => Rx.Observable.interval(100).take(3)
                );

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

畫成 Marble Diagram 表示如下

```
source : -----------c--c-----------------...
        concatMap(c => Rx.Observable.interval(100).take(3))
```

另外我們也可以把 switchMap 用在發送 HTTP request

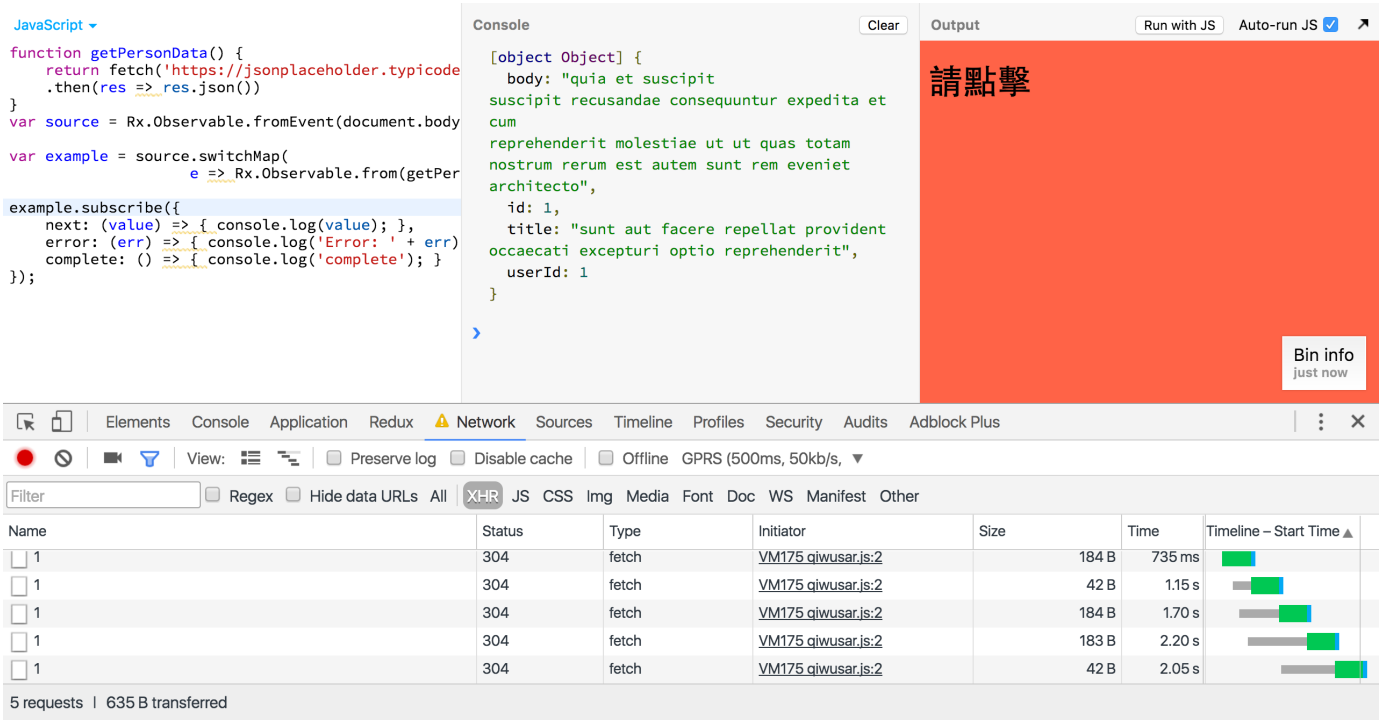
```
function getPostData() {
    return fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then(res => res.json())
}
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source.switchMap(
                    e => Rx.Observable.from(getPostData()));

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

JSBin | JSFiddle

如果我們快速的連續點擊五下，可以在開發者工具的 network 看到每個 request 會在點擊時發送，如下圖



> 灰色是瀏覽器原生地停頓行為，實際上灰色的一開始就是 fetch 執行送出 request，只是卡在

情境，比如說 自動完成(auto complete)，我們只需要顯示使用者最後一次打在畫面上的文字，來做建議選項而不用每一次的。

switchMap 跟 concatMap 一樣有第二個參數 selector callback 可用來回傳我們要的值，這部分的行為跟 concatMap 是一樣的，這裡就不再贅述。

mergeMap

mergeMap 其實就是 map 加上 mergeAll 簡化的寫法，如下

```
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source
                .map(e => Rx.Observable.interval(1000).take(3))
                .mergeAll();

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

上面的程式碼可以簡化成

```
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source
                .mergeMap(
                    e => Rx.Observable.interval(100).take(3)
                );

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

畫成 Marble Diagram 表示





```
example: -------------0-(10)-(21)-2----------...
```

記得 mergeMap 可以並行處理多個 observable，以這個例子來說當我們快速點按兩下，元素發送的時間點是有機會重疊的，這個部份的細節大家可以看上一篇文章 merge 的部分。

另外我們也可以把 mergeMap 用在發送 HTTP request

```
function getPostData() {
    return fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then(res => res.json())
}
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source.mergeMap(
                    e => Rx.Observable.from(getPostData()));

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

JSBin | JSFiddle

如果我們快速的連續點擊五下，大家可以在開發者工具的 network 看到每個 request 會在點擊時發送並且會 log 出五個物件，如下圖

mergeMap 也能傳入第二個參數 selector callback，這個 selector callback 跟 concatMap 第二個參數也是完全一樣的，但 mergeMap 的重點是我們可以傳入第三個參數，來限制並行處理的數量

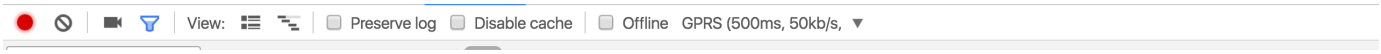
```
function getPostData() {
    return fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then(res => res.json())
}
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source.mergeMap(
```

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

JSBin | JSFiddle

這裡我們傳入 3 就能限制，HTTP request 最多只能同時送出 3 個，並且要等其中一個完成在處理下一個，如下圖



大家可以注意看上面這張圖，我連續點按了五下，但第四個 request 是在第一個完成後才送出的，這個很適合用在特殊的需求下，可以限制同時發送的 request 數量。

> RxJS 5 還保留了 mergeMap 的別名叫 flatMap，雖然官方文件上沒有，但這兩個方法是完全一樣的。請參考 這裡

switchMap, mergeMap, concatMap

這三個 operators 還有一個共同的特性，那就是這三個 operators 可以把第一個參數所回傳的 promise 物件直接轉成 observable，這樣我們就不用再用 `Rx.Observable.from` 轉一次，如下

```
function getPersonData() {
    return fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then(res => res.json())
}
var source = Rx.Observable.fromEvent(document.body, 'click');

var example = source.concatMap(e => getPersonData());
                                    //直接回傳 promise 物件
```

```
});
```

至於在使用上要如何選擇這三個 operators？ 其實都還是看使用情境而定，這裡筆者簡單列一下大部分的使用情境

- concatMap用在可以确定**内部的observable结束时间比外部observable发送时间来快的情境**，并且不希望有任何并行处理行为，适合少数要一次一次完成到底的的UI动画或特别的HTTP request行为。
- switchMap 用在只要最后一次行为的结果，适合绝大多数的使用情境。
- mergeMap 用在并行处理多个observable，适合需要并行处理的行为，像是多个I/O 的并行处理。

> 建议初学者不确定选哪一个时，使用switchMap

> 在使用concatAll 或concatMap 时，请注意内部的observable 一定要能够的结束，且外部的observable 发送元素的速度不能比内部的observable 结束时间快太多，不然会有memory issues

今日小结

今天的文章内容主要讲了三个operators，如果有看完上一篇文章的读者应该不难吸收，主要还是使用情境上需要思考以及注意一些细节。

不知道今天读者有没有收获呢？如果有任何问题，欢迎留言给我，谢谢

标签

JavaScript RxJS 可观察的 运算符 RxJS 30天

上一页 下一个