30 天精通 RxJS (24)：Observable operators - multicast, refCount, publish, share

Jan 9th, 2017. 4 mins read

昨天我們介紹完了各種 Subject，不曉得各位讀者還記不記得在一開始講到 Subject 時，是希望能夠讓 Observable 有新訂閱時，可以共用前一個訂閱的執行而不要從頭開始

如下面的例子

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

var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

var subject = new Rx.Subject()

subject.subscribe(observerA)

source.subscribe(subject);

setTimeout(() => {
    subject.subscribe(observerB);
}, 1000);

// "A next: 0"
// "A next: 1"
```

```
// "A complete!"
// "B complete!"
```

上面這段程式碼我們用 subject 訂閱了 source，再把 observerA 跟 observerB 一個個訂閱到 subject，這樣就可以讓 observerA 跟 observerB 共用同一個執行。但這樣的寫法會讓程式碼看起來太過複雜，我們可以用 Observable 的 multicast operator 來簡化這段程式

Operators

multicast

multicast 可以用來掛載 subject 並回傳一個可連結(connectable)的 observable，如下

```
var source = Rx.Observable.interval(1000)
             .take(3)
             .multicast(new Rx.Subject());

var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

source.subscribe(observerA); // subject.subscribe(observerA)

source.connect(); // source.subscribe(subject)

setTimeout(() => {
    source.subscribe(observerB); // subject.subscribe(observerB)
}, 1000);
```

```
source.subscribe(observerA); // subject.subscribe(observerA)
```

必須真的等到 執行 `connect()` 後才會真的用 subject 訂閱 source，並開始送出元素，如果沒有執行 `connect()` observable 是不會真正執行的。

```
source.connect();
```

另外值得注意的是這裡要退訂的話，要把 `connect()` 回傳的 subscription 退訂才會真正停止 observable 的執行，如下

```
var source = Rx.Observable.interval(1000)
              .do(x => console.log('send: ' + x))
              .multicast(new Rx.Subject()); // 無限的 observable

var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

var subscriptionA = source.subscribe(observerA);

var realSubscription = source.connect();

var subscriptionB;
setTimeout(() => {
    subscriptionB = source.subscribe(observerB);
}, 1000);

setTimeout(() => {
```





```
setTimeout(() => {
    realSubscription.unsubscribe();
    // 這裡 source 才會真正停止送元素
}, 7000);
```

JSBin | JSFiddle

上面這段的程式碼，必須等到 `realSubscription.unsubscribe()` 執行完，source 才會真的結束。

雖然用了 multicast 感覺會讓我們處理的對象少一點，但必須搭配 connect 一起使用還是讓程式碼有點複雜，通常我們會希望有 observer 訂閱時，就立即執行並發送元素，而不要再多執行一個方法(connect)，這時我們就可以用 refCount。

refCount

refCount 必須搭配 multicast 一起使用，他可以建立一個只要有訂閱就會自動 connect 的 observable，範例如下

```
var source = Rx.Observable.interval(1000)
            .do(x => console.log('send: ' + x))
            .multicast(new Rx.Subject())
            .refCount();

var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

var subscriptionA = source.subscribe(observerA);
// 訂閱數 0 -> 1
```

```
    subscriptionB = source.subscribe(observerB);
    // 訂閱數 0 => 2
}, 1000);
```

JSBin | JSFiddle

上面這段程式碼，當 source 一被 observerA 訂閱時(訂閱數從 0 變成 1)，就會立即執行並發送元素，我們就不需要再額外執行 connect。

同樣的在退訂時只要訂閱數變成 0 就會自動停止發送

```
var source = Rx.Observable.interval(1000)
              .do(x => console.log('send: ' + x))
              .multicast(new Rx.Subject())
              .refCount();

var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

var subscriptionA = source.subscribe(observerA);
// 訂閱數 0 => 1

var subscriptionB;
setTimeout(() => {
    subscriptionB = source.subscribe(observerB);
    // 訂閱數 0 => 2
}, 1000);

setTimeout(() => {
    subscriptionA.unsubscribe(); // 訂閱數 2 => 1
```





JSBin | JSFiddle

publish

其實 `multicast(new Rx.Subject())` 很常用到，我們有一個簡化的寫法那就是 publish，下面這兩段程式碼是完全等價的

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

// var source = Rx.Observable.interval(1000)
//              .multicast(new Rx.Subject())
//              .refCount();
```

加上 Subject 的三種變形

```
var source = Rx.Observable.interval(1000)
             .publishReplay(1)
             .refCount();

// var source = Rx.Observable.interval(1000)
//              .multicast(new Rx.ReplaySubject(1))
//              .refCount();


var source = Rx.Observable.interval(1000)
             .publishBehavior(0)
             .refCount();

// var source = Rx.Observable.interval(1000)
//              .multicast(new Rx.BehaviorSubject(0))
//              .refCount();


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





```
//              .refCount();
```

share

另外 publish + refCount 可以在簡寫成 share

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

// var source = Rx.Observable.interval(1000)
//              .publish()
//              .refCount();

// var source = Rx.Observable.interval(1000)
//              .multicast(new Rx.Subject())
//              .refCount();
```

今日小結

今天主要講解了 multicast 和 refCount 兩個 operators 可以幫助我們既可能的簡化程式碼，並同時達到組播的效果。最後介紹 publish 跟 share 幾個簡化寫法，這幾個簡化的寫法是比較常見的，在理解 multicast 跟 refCount 運作方式後就能直接套用到 publish 跟 share 上。

不知道今天讀者們有沒有收穫呢？ 如果有任何問題歡迎在下方留言給我，謝謝 ^ ^

Tags

JavaScript RxJS Observable Subject Operator RxJS 30 Days

Prev Next