30 天精通 RxJS (22)：Subject 基本觀念

Jan 7th, 2017. 4 mins read

終於進到了 RxJS 的第二個重點 Subject，不知道讀者們有沒有發現？ 我們在這篇文章之前的範例，每個 observable 都只訂閱了一次，而實際上 observable 是可以多次訂閱的。

Multiple subscriptions

```
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!')
}

source.subscribe(observerA);
source.subscribe(observerB);

// "A next: 0"
// "B next: 0"
// "A next: 1"
// "B next: 1"
// "A next: 2"
// "A complete!"
// "B next: 2"
// "B complete!"
```





是說他們是完全獨立的，我們把 observerB 延遲訂閱來證明看看

```
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!')
}

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

// "A next: 0"
// "A next: 1"
// "B next: 0"
// "A next: 2"
// "A complete!"
// "B next: 1"
// "B next: 2"
// "B complete!"
```

JSBin | JSFiddle

這裡我們延遲一秒再用 observerB 訂閱，可以從 log 中看出 1 秒後 observerA 已經印到了 1，這時 observerB 開始印卻是從 0 開始，而不是接著 observerA 的進度，代表這兩次的訂閱是完全分開來執行的，或者說是每次的訂閱都建立了一個新的執行。

這樣的行為在大部分的情境下適用，但有些案例下我們會希望第二次訂閱 source 不會從頭開始接收元素，而是從第一次訂閱到當前處理的元素開始發送，我們把這種處理方式稱為組播 (multicast)，那我們要如何做到組播呢？

或許已經有讀者想到解法了，其實我們可以建立一個中間人來訂閱 source 再由中間人轉送資料出去，就可以達到我們想要的效果

```
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 = {
    observers: [],
    addObserver: function(observer) {
        this.observers.push(observer)
    },
    next: function(value) {
        this.observers.forEach(o => o.next(value))
    },
    error: function(error){
        this.observers.forEach(o => o.error(error))
    },
    complete: function() {
        this.observers.forEach(o => o.complete())
    }
}

subject.addObserver(observerA)

source.subscribe(subject);
```





```
// "A next: 0"
// "A next: 1"
// "B next: 1"
// "A next: 2"
// "B next: 2"
// "A complete!"
// "B complete!"
```

JSBin | JSFiddle

從上面的程式碼可以看到，我們先建立了一個物件叫 subject，這個物件具備 observer 所有的方法(next, error, complete)，並且還能 addObserver 把 observer 加到內部的清單中，每當有值送出就會遍歷清單中的所有 observer 並把值再次送出，這樣一來不管多久之後加進來的 observer，都會是從當前處理到的元素接續往下走，就像範例中所示，我們用 subject 訂閱 source 並把 observerA 加到 subject 中，一秒後再把 observerB 加到 subject，這時就可以看到 observerB 是直接收 1 開始，這就是組播(multicast)的行為。

讓我們把 subject 的 `addObserver` 改名成 `subscribe` 如下

```
var subject = {
    observers: [],
    subscribe: function(observer) {
        this.observers.push(observer)
    },
    next: function(value) {
        this.observers.forEach(o => o.next(value))
    },
    error: function(error){
        this.observers.forEach(o => o.error(error))
    },
    complete: function() {
        this.observers.forEach(o => o.complete())
    }
}
```

> 應該有眼尖的讀者已經發現，subject 其實就是用了 Observer Pattern。但這邊為了不要混淆 Observer Pattern 跟 RxJS 的 observer 就不再內文提及。這也是為什麼我們在一開始講 Observer Pattern 希望大家親自實作的原因。

```
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"
// "B next: 1"
// "A next: 2"
// "B next: 2"
// "A complete!"
// "B complete!"
```

JSBin | JSFiddle

大家會發現使用方式跟前面是相同的，建立一個 subject 先拿去訂閱 observable(source)，再把我們真正的 observer 加到 subject 中，這樣一來就能完成訂閱，而每個加到 subject 中的 observer 都能整組的接收到相同的元素。





雖然前面我們已經示範直接手寫一個簡單的 subject，但到底 RxJS 中的 Subject 的概念到底是什麼呢？

首先 Subject 可以拿去訂閱 Observable(source) 代表他是一個 Observer，同時 Subject 又可以被 Observer(observerA, observerB) 訂閱，代表他是一個 Observable。

總結成兩句話

- Subject 同時是 Observable 又是 Observer
- Subject 會對內部的 observers 清單進行組播(multicast)

> 補充： 沒事不要看！其實 Subject 就是 Observer Pattern 的實作並且繼承自 Observable。

今日小結

今天介紹了 RxJS 中的第二個重點 Subject，重點放在 Subject 主要的運作方式，以及概念上的所代表的意思，如果今天還不太能夠吸收的讀者不用緊張，後面我們會講到 subject 的一些應用，到時候就會有更深的體會。

不知道今天讀者麼有沒有收穫呢？ 如果有任何問題，歡迎在下方留言給我。

Tags

JavaScript RxJS Observable Operator RxJS 30 Days

Prev Next