30 天精通 RxJS (06)： 建立 Observable(二)

Dec 22nd, 2016. 7 mins read

通常我們會透過 creation operator 來建立 Observable 實例，這篇文章會講解幾個較為常用的 operator！

這是【30天精通 RxJS】的 06 篇，如果還沒看過 05 篇可以往這邊走： 30 天精通 RxJS (05)：建立 Observable(一)

Creation Operator

Observable 有許多創建實例的方法，稱為 creation operator。下面我們列出 RxJS 常用的 creation operator

- create
- of
- from
- fromEvent
- fromPromise
- never
- empty
- throw
- interval
- timer

of

還記得我們昨天用 `create` 來建立一個同步處理的 observable 嗎？

```
var source = Rx.Observable
    .create(function(observer) {
```

```
    });

source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
    console.log(error)
    }
});

// Jerry
// Anna
// complete!
```

JSBin | JSFiddle

他先後傳遞了 `'Jerry'`, `'Anna'` 然後結束(complete)，這是一個十分常見模式。當我們想要**同步的**傳遞幾個值時，就可以用 `of` 這個 operator 來簡潔的表達!

下面的程式碼行為同上

```
var source = Rx.Observable.of('Jerry', 'Anna');

source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
        console.log(error)
    }
});
```





JSBin | JSFiddle

是不是相較於原本的程式碼簡潔許多呢？

from

可能已經有人發現其實 `of` operator 的一個一個參數其實就是一個 list，而 list 在 JavaScript 中最常見的形式是陣列(array)，那我們有沒有辦法把一個已存在的陣列當作參數呢？

有的，我們可以用 `from` 來接收任何可列舉的參數！

```
var arr = ['Jerry', 'Anna', 2016, 2017, '30 days']
var source = Rx.Observable.from(arr);

source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
        console.log(error)
    }
});

// Jerry
// Anna
// 2016
// 2017
// 30 days
// complete!
```

JSBin | JSFiddle

記得任何可列舉的參數都可以用喔，也就是說像 Set, WeakSet, Iterator 等都可以當作參數！

> 因為 ES6 出現後可列舉(iterable)的型別變多了，所以 `fromArray` 就被移除囉。

```
source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
        console.log(error)
    }
});
// 鐵
// 人
// 賽
// complete!
```

JSBin | JSFiddle

上面的程式碼會把字串裡的每個字元一一印出來。

我們也可以傳入 Promise 物件，如下

```
var source = Rx.Observable
  .from(new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Hello RxJS!');
    },3000)
  }))

source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
    console.log(error)
```





```
// complete!
```

JSBin | JSFiddle

如果我們傳入 Promise 物件實例，當正常回傳時，就會被送到 next，並立即送出完成通知，如果有錯誤則會送到 error。

> 這裡也可以用 `fromPromise` ，會有相同的結果。

fromEvent

我們也可以用 Event 建立 Observable，透過 `fromEvent` 的方法，如下

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

source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
        console.log(error)
    }
});

// MouseEvent {...}
```

JSBin | JSFiddle

`fromEvent` 的第一個參數要傳入 DOM 物件，第二個參數傳入要監聽的事件名稱。上面的程式會針對 body 的 click 事件做監聽，每當點擊 body 就會印出 event。

> 取得 DOM 物件的常用方法： `document.getElementById()` `document.querySelector()` `document.getElementsByTagName()` `document.getElementsByClassName()`

補充：fromEventPattern





DOM Event 有 addEventListener 及 removeEventListener 一樣！ 舉一個例子，我們在【30 天精通 RxJS (04)： 什麼是 Observable ?】實作的 Observer Pattern 就是類事件，程式碼如下：

```
class Producer {
        constructor() {
                this.listeners = [];
        }
        addListener(listener) {
                if(typeof listener === 'function') {
                        this.listeners.push(listener)
                } else {
                        throw new Error('listener 必須是 function')
                }
        }
        removeListener(listener) {
                this.listeners.splice(this.listeners.indexOf(listener),
        }
        notify(message) {
                this.listeners.forEach(listener => {
                        listener(message);
                })
        }
}
// ------- 以上都是之前的程式碼 -------- //

var egghead = new Producer();
// egghead 同時具有 註冊監聽者及移除監聽者 兩種方法

var source = Rx.Observable
    .fromEventPattern(
        (handler) => egghead.addListener(handler),
        (handler) => egghead.removeListener(handler)
    );

source.subscribe({
    next: function(value) {
```

```
    },
    error: function(error) {
        console.log(error)
    }
})

egghead.notify('Hello! Can you hear me?');
// Hello! Can you hear me?
```

JSBin | JSFiddle

上面的程式碼可以看到， `egghead` 是 `Producer` 的實例，同時具有 註冊監聽及移除監聽兩種方法，我們可以將這兩個方法依序傳入 `fromEventPattern` 來建立 Observable 的物件實例！

> 這裡要注意不要直接將方法傳入，避免 this 出錯！也可以用 `bind` 來寫。

```
Rx.Observable
    .fromEventPattern(
        egghead.addListener.bind(egghead),
        egghead.removeListener.bind(egghead)
    )
    .subscribe(console.log)
```

empty, never, throw

接下來我們要看幾個比較無趣的 operators，之後我們會講到很多 observables 合并(combine)、轉換(transforme)的方法，到那個時候無趣的 observable 也會很有用！

有點像是數學上的 **零(0)**，雖然有時候好像沒什麼，但卻非常的重要。在 Observable 的世界裡也有類似的東西，像是 `empty`

```
var source = Rx.Observable.empty();

source.subscribe({
    next: function(value) {
        console.log(value)
    },
```

```
        console.log(error)
    }
});
// complete!
```

JSBin | JSFiddle

`empty` 會給我們一個**空**的 observable，如果我們訂閱這個 observable 會發生什麼事呢？ 它會立即送出 complete 的訊息！

> 可以直接把 `empty` 想成沒有做任何事，但它至少會告訴你它沒做任何事。

數學上還有一個跟零(0)很像的數，那就是 **無窮(∞)**，在 Observable 的世界裡我們用 `never` 來建立無窮的 observable

```
var source = Rx.Observable.never();

source.subscribe({
    next: function(value) {
        console.log(value)
    },
    complete: function() {
        console.log('complete!');
    },
    error: function(error) {
        console.log(error)
    }
});
```

JSBin |JSFiddle

never 會給我們一個無窮的 observable，如果我們訂閱它又會發生什麼事呢？...什麼事都不會發生，它就是一個一直存在但卻什麼都不做的 observable。

> 可以把 never 想像成一個結束在無窮久以後的 observable，但你永遠等不到那一天！

> 題外話，筆者一直很喜歡平行線的解釋： 兩條平行線就是它們相交於無窮遠

最後還有一個 operator `throw`，它也就只做一件事就是拋出錯誤。

```
source.subscribe({
        next: function(value) {
                console.log(value)
        },
        complete: function() {
                console.log('complete!');
        },
        error: function(error) {
    console.log('Throw Error: ' + error)
        }
});
// Throw Error: Oop!
```

JSBin | JSFiddle

上面這段程式碼就只會 log 出 `'Throw Error: Oop!'`。

這三個 operators 雖然目前看起來沒什麼用，但之後在文章中大家就會慢慢發掘它們的用處！

interval, timer

接著我們要看兩個跟時間有關的 operators，在 JS 中我們可以用 `setInterval` 來建立一個持續的行為，這也能用在 Observable 中

```
var source = Rx.Observable.create(function(observer) {
    var i = 0;
    setInterval(() => {
        observer.next(i++);
    }, 1000)
});

source.subscribe({
        next: function(value) {
                console.log(value)
        },
        complete: function() {
                console.log('complete!');
        },
```

```
// 0
// 1
// 2
// .....
```

JSBin | JSFiddle

上面這段程式碼，會每隔一秒送出一個從零開始遞增的整數，在 Observable 的世界也有一個 operator 可以更方便地做到這件事，就是 `interval`

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

source.subscribe({
        next: function(value) {
                console.log(value)
        },
        complete: function() {
                console.log('complete!');
        },
        error: function(error) {
    console.log('Throw Error: ' + error)
        }
});
// 0
// 1
// 2
// ...
```

JSBin | JSFiddle

`interval` 有一個參數必須是數值(Number)，這的數值代表發出訊號的間隔時間(ms)。這兩段程式碼基本上是等價的，會持續每隔一秒送出一個從零開始遞增的數值！

另外有一個很相似的 operator 叫 `timer` ， `timer` 可以給兩個參數，範例如下

```
var source = Rx.Observable.timer(1000, 5000);

source.subscribe({
        next: function(value) {
```

    

```
        },
        error: function(error) {
    console.log('Throw Error: ' + error)
        }
});
// 0
// 1
// 2 ...
```

JSBin | JSFiddle

當 `timer` 有兩個參數時，第一個參數代表要發出第一個值的等待時間(ms)，第二個參數代表第一次之後發送值的間隔時間，所以上面這段程式碼會先等一秒送出 0 之後每五秒送出 1, 2, 3, 4....。

`timer` 第一個參數除了可以是數值(Number)之外，也可以是日期(Date)，就會等到指定的時間在發送第一個值。

另外 `timer` 也可以只接收一個參數

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

source.subscribe({
        next: function(value) {
                console.log(value)
        },
        complete: function() {
                console.log('complete!');
        },
        error: function(error) {
    console.log('Throw Error: ' + error)
        }
});
// 0
// complete!
```

JSBin | JSFiddle

上面這段程式碼就會等一秒後送出 1 同時通知結束。





Series / 30 天精通 RxJS

今天我們講到很多 無窮的 observable，例如 interval, never。但有時我們可能會在某些行為後不需要這些資源，要做到這件事最簡單的方式就是 `unsubscribe`。

其實在訂閱 observable 後，會回傳一個 subscription 物件，這個物件具有釋放資源的 `unsubscribe` 方法，範例如下

```
var source = Rx.Observable.timer(1000, 1000);

// 取得 subscription
var subscription = source.subscribe({
        next: function(value) {
                console.log(value)
        },
        complete: function() {
                console.log('complete!');
        },
        error: function(error) {
    console.log('Throw Error: ' + error)
        }
});

setTimeout(() => {
    subscription.unsubscribe() // 停止訂閱(退訂)， RxJS 4.x 以前的版本用 di
}, 5000);
// 0
// 1
// 2
// 3
// 4
```

JSBin | JSFiddle

這裡我們用了 `setTimeout` 在 5 秒後，執行了 `subscription.unsubscribe()` 來停止訂閱並釋放資源。另外 subscription 物件還有其他合併訂閱等作用，這個我們之後有機會會在提到！

> Events observable 盡量不要用 `unsubscribe` ，通常我們會使用 `takeUntil` ，在某個事件發生後來完成 Event observable，這個部份我們之後會講到！





Series / 30 天精通 RxJS

今天我們把建立 Observable 實例的方法幾乎都講完了，建立 Observable 是 RxJS 的基礎，接下來我們會講轉換(Transformation)、過濾(Filter)、合并(Combination)等 Operators，但不會像今天這樣一次把一整個類型的 operator 講完，筆者會依照實用程度以及範例搭配穿插著講各種 operator!

Tags

JavaScript RxJS Observable RxJS 30 Days

Prev Next