

30 天精通RxJS (05) : 建立Observable(一)

Dec 21st, 2016 . 5 mins read

Observable 是RxJS 的核心，今天让我们从如何建立Observable 开始！

这是【30天精通RxJS】的05篇，如果还没看过04篇可以往这边走：[30天精通RxJS \(04\)：什么是Observable？](#)

不想看文章的人，可以直接看影片喔！

今天大家看文章一定要分清楚**Observable**跟**Observer**，不要搞混。

前几天我们把所有重要的观念及前置的知识都讲完了，今天要正式进入RxJS的应用，整个RxJS说白了就是一个**核心三个重点**。

一个核心是Observable 再加上相关的Operators(map, filter...)，这个部份是最重要的，其他三个重点本质上也是围绕这个核心在转，所以我们会花将近20 天的篇数讲这个部份的观念及使用案例。

另外三个重点分别是

- Observer
- Subject
- Schedulers

Observer 是这三个当中一定会用到却是最简单的，所以我们今天就会把它介绍完。Subject 一般应用到的频率就相对低很多，但如果想要看懂RxJS 相关的Library 或Framework，Subject 就是一定要会的重点，所以这个部份我们大概会花3-5 天的时间讲解。至于Schedulers 则是要解决RxJS 衍伸出的最后一道问题，这个部份会视情况加入或是在30 天后补完。





| `redux-observable` 就是用了Subject实作的

| 让我卖个关子，先不说RxJS 最后一道问题是什么。

说了这么多，我们赶快进入到今天的主题Observable 吧！

建立Observable: `create`

建立Observable的方法有非常多种，其中`create`是最基本的方法。`create`方法在`Rx.Observable`物件中，再传入一个callback function，这个callback function会接收一个



Series / 30 天精通RxJS

```
observer.next('Jerry'); // RxJS 4.x 以前的版本用 onNext
observer.next('Anna');
})
```

这个callback function 会定义observable 将会如何发送值。

虽然Observable可以被 `create`，但实务上我们通常都使用 **creation operator** 像是 `from`, `of`, `fromEvent`, `fromPromise` 等。这里只是为了从基本的开始讲解所以才用 `create`

我们可以订阅这个observable，来接收他送出的值，程式码如下

```
var observable = Rx.Observable
  .create(function(observer) {
    observer.next('Jerry'); // RxJS 4.x 以前的版本用 onNext
    observer.next('Anna');
  })

// 訂閱這個 observable
observable.subscribe(function(value) {
  console.log(value);
})
```

[JSBin](#) | [JSFiddle](#)

当我们订阅这个observable，他就会依序送出 'Jerry' 'Anna' 两个字串。

訂閱 Observable 跟 `addEventListener` 在實作上其實有非常大的不同。雖然在行為上很像，但實際上 Observable 根本沒有管理一個訂閱的清單，這個部份的細節我們留到最後說明！

這裡有一個重點，很多人認為 RxJS 是在做非同步處理，所以所有行為都是非同步的。但其實這個觀念是錯的，RxJS 確實主要在處理非同步行為沒錯，但也同時能處理同步行為，像是上面的程式碼就是同步執行的。

證明如下

```
var observable = Rx.Observable
  .create(function(observer) {
    observer.next('Jerry'); // RxJS 4.x 以前的版本用 onNext
    observer.next('Anna');
```



Series / 30 天精通RxJS

```
observable.subscribe(function(value) {  
    console.log(value);  
});  
console.log('end');
```

[JSBin](#) | [JSFiddle](#)

上面這段程式碼會印出

```
start  
Jerry  
Anna  
end
```

而不是

```
start  
end  
Jerry  
Anna
```

所以很明顯的這段程式碼是同步執行的，當然我們可以拿它來處理非同步的行為！

```
var observable = Rx.Observable  
    .create(function(observer) {  
        observer.next('Jerry'); // RxJS 4.x 以前的版本用 onNext  
        observer.next('Anna');  
  
        setTimeout(() => {  
            observer.next('RxJS 30 Days!');  
        }, 30)  
    })  
  
console.log('start');  
observable.subscribe(function(value) {  
    console.log(value);  
});  
console.log('end');
```



Series / 30 天精通RxJS

```
start
Jerry
Anna
end
RxJS 30 Days!
```

從上述的程式碼能看得出來

Observable 同時可以處理同步與非同步的行為！

觀察者 Observer

Observable 可以被訂閱(subscribe)，或說可以被觀察，而訂閱 Observable 的物件又稱為 **觀察者(Observer)**。觀察者是一個具有三個方法(method)的物件，每當 Observable 發生事件時，便會呼叫觀察者相對應的方法。

注意這裡的觀察者(Observer)跟上一篇講的觀察者模式(Observer Pattern)無關，觀察者模式是一種設計模式，是思考問題的解決過程，而這裡講的觀察者是一個被定義的物件。

觀察者的三個方法(method)：

- next：每當 Observable 發送出新的值，next 方法就會被呼叫。
- complete：在 Observable 沒有其他的資料可以取得時，complete 方法就會被呼叫，在 complete 被呼叫之後，next 方法就不會再起作用。
- error：每當 Observable 內發生錯誤時，error 方法就會被呼叫。

說了這麼多，我們還是直接來建立一個觀察者吧！

```
var observable = Rx.Observable
    .create(function(observer) {
        observer.next('Jerry');
        observer.next('Anna');
        observer.complete();
        observer.next('not work');
    })
```

// 宣告一個觀察者，具備 next、error、complete 三個方法



Series / 30 天精通RxJS

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

```
// 用我們定義好的觀察者，來訂閱這個 observable
observable.subscribe(observer)
```

[JSBin](#) | [JSFiddle](#)

上面這段程式碼會印出

```
Jerry
Anna
complete
```

上面的範例可以看得出來在 complete 執行後，next 就會自動失效，所以沒有印出 not work。

下面則是送出錯誤的範例

```
var observable = Rx.Observable
    .create(function(observer) {
        try {
            observer.next('Jerry');
            observer.next('Anna');
            throw 'some exception';
        } catch(e) {
            observer.error(e)
        }
    });
```

// 宣告一個觀察者，具備 next, error, complete 三個方法

```
var observer = {
    next: function(value) {
        console.log(value);
    }
}
```



Series / 30 天精通RxJS

```
complete: function() {  
    console.log('complete')  
}  
}
```

// 用我們定義好的觀察者，來訂閱這個 observable
observable.subscribe(observer)

[JSBin](#) | [JSFiddle](#)

這裡就會執行 error 的 function 印出 `Error: some exception`。

另外觀察者可以是不完整的，他可以只具有一個 next 方法，如下

```
var observer = {  
    next: function(value) {  
        //...  
    }  
}
```

有時候 Observable 會是一個無限的序列，例如 click 事件，這時 `complete` 方法就有可能永遠不會被呼叫！

我們也可以直接把 next, error, complete 三個 function 依序傳入 `observable.subscribe`，如下：

```
observable.subscribe(  
    value => { console.log(value); },  
    error => { console.log('Error: ', error); },  
    () => { console.log('complete') }  
)
```

`observable.subscribe` 會在內部自動組成 observer 物件來操作。

實作細節

我們前面提到了，其實 Observable 的訂閱跟 `addEventListener` 在實作上有蠻大的差異，雖然他



Series / 30 天精通RxJS

```
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);
    })
  }
}
```

我們在內部儲存了一份所有的監聽者清單(`this.listeners`)，在要發佈通知時會對逐一的呼叫這份清單的監聽者。

但在 Observable 不是這樣實作的，在其內部並沒有一份訂閱者的清單。訂閱 Observable 的行為比較像是執行一個物件的方法，並把資料傳進這個方法中。

我們以下面的程式碼做說明

```
var observable = Rx.Observable
  .create(function (observer) {
    observer.next('Jerry');
    observer.next('Anna');
  })

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



Series / 30 天精通RxJS

```
    },  
    complete: function() {  
      console.log('complete')  
    }  
  })
```

像上面這段程式，他的行為比較像這樣

```
function subscribe(observer) {  
  observer.next('Jerry');  
  observer.next('Anna');  
}  
  
subscribe({  
  next: function(value) {  
    console.log(value);  
  },  
  error: function(error) {  
    console.log(error)  
  },  
  complete: function() {  
    console.log('complete')  
  }  
});
```

這裡可以看到 subscribe 是一個 function，這個 function 執行時會傳入觀察者，而我們在這個 function 內部去執行觀察者的方法。

訂閱一個 Observable 就像是執行一個 function

今日小結

今天在講關於建立Observable的實例，用到了 `create` 的方法，但大部分的內容還是在講 Observable 幾個重要的觀念，如下

- Observable 可以同时处理**同步**跟**非同步**行为
- Observer 是一个物件，这个物件具有三个方法，分别是 **next**，**error**，**complete**



Series / 30 天精通RxJS

Tags

- JavaScript
- RxJS
- Observable
- Observer
- RxJS 30 Days

⬅️ Prev



Next ➡️

