30 天精通RxJS (01)：认识RxJS

Dec 17th, 2016 . 4 mins read

> RxJS 是笔者认为未来几年内会非常红的Library，RxJS 提供了一套完整的非同步解决方案，让我们在面对各种非同步行为，不管是Event, AJAX, 还是Animation 等，我们都可以使用相同的API (Application Programming Interface) 做开发。

这是【30天精通RxJS】的01篇，如果还没看过00篇可以往这边走： 30天精通RxJS (00)：关于本系列文章

在网页的世界存取任何资源都是非同步(Async)的，比如说我们希望拿到一个档案，要先发送一个请求，然后必须等到档案回来，再执行对这个档案的操作。这就是一个非同步的行为，而随着网页需求的复杂化，我们所写的JavaScript 就有各种针对非同步行为的写法，例如使用callback 或是Promise 物件甚至是新的语法糖async/await ——但随着应用需求愈来愈复杂，撰写非同步的程式码仍然非常困难。

非同步常见的问题

- 竞态条件(Race Condition)
- 记忆体泄漏(Memory Leak)
- 复杂的状态(Complex State)
- 例外处理(Exception Handling)

Race Condition

每当我们对同一个资源同时做多次的非同步存取时，就可能发生Race Condition 的问题。比如说我们发了一个Request 更新使用者资料，然后我们又立即发送另一个Request 取得使用者资料，这时第一个Request 和第二个Request 先后顺序就会影响到最终接收到的结果不同，这就是Race Condition。

Memory Leak

Memory Leak是最常被大家忽略的一点。原因是在传统网站的行为，我们每次换页都是整页重刷，并重新执行JavaScript，所以不太需要理会记忆体的问题，但是当我们希望将网站做得像应用

切换时，没有把scroll的监听事件移除。

Complex State

当有非同步行为时，应用程式的状态就会变得非常复杂！比如说我们有一支付费用户才能播放的影片，首先可能要先抓取这部影片的资讯，接着我们要在播放时去验证使用者是否有权限播放，而使用者也有可能再按下播放后又立即按了取消，而这些都是非同步执行，这时就会各种复杂的状态需要处理。

Exception Handling

JavaScript 的try/catch 可以捕捉同步的例外，但非同步的程式就没这么容易，尤其当我们的非同步行为很复杂时，这个问题就愈加明显。

各种不同的API

我们除了要面对非同步会遇到的各种问题外，还需要烦恼很多不同的API

- DOM Events
- XMLHttpRequest
- Fetch
- WebSockets
- Server Send Events
- Service Worker
- Node Stream
- Timer

上面列的API 都是非同步的，但他们都有各自的API 及写法！如果我们使用RxJS，上面所有的API都可以透过RxJS 来处理，就能用同样的API 操作(RxJS 的API)。

这里我们举一个例子，假如我们想要监听点击事件(click event)，但点击一次之后不再监听。

原生JavaScript

```
var handler = (e) => {
```

```
// 註冊監聽
document.body.addEventListener('click', handler);
```

使用Rx 大概的样子

```
Rx.Observable
        .fromEvent(document.body, 'click') // 註冊監聽
        .take(1) // 只取一次
        .subscribe(console.log);
```

JSbin | JSFiddle (点击画面后会在console显示，记得打开console来看)

大致上能看得出来我们在使用RxJS后，不管是针对DOM Event还是上面列的各种API我们都可以透过RxJS的API来做资料操作，像是范例中用 `take(n)` 来设定只取一次，之后就释放记忆体。

说了这么多，其实就是简单一句话

在面对日益复杂的问题，我们需要一个更好的解决方法。

RxJS 基本介绍

RxJS是一套藉由**Observable sequences**来组合**非同步行为**和**事件基础**程序的Library！

> 可以把RxJS 想成处理非同步行为的Lodash。

这也被称为Functional Reactive Programming，更切确地说是指Functional Programming 及 Reactive Programming 两个编程思想的结合。

> RxJS 确实是Functional Programming 跟Reactive Programming 的结合，但能不能称为 Functional Reactive Programming (FRP) 一直有争议。

> Rx在 官网 上特别指出，有时这会被称为FRP但这其实是个“误称”。

> 简单说FRP是操作随着时间**连续性改变的数值**而Rx则比较像是操作随着时间发出的**离散数值**，这个部份读者不用分得太细，因为FRP的定义及解释一直存在着歧异，也有众多大神为此争论，如下

> André Staltz：Rx著名的推广者，也是RxJS 5主要贡献者之一，同时是Cycle.js的作者。

> Evan Czaplicki ：任职于NoRedInk，Elm的作者。Evan在 StrangeLoop 2014的演讲 中，特别为现在各种FRP的不同解释做分类。

> 笔者自己的看法是比较偏向直接称Rx为FRP，原因是这较为直觉(FP + RP = FRP)，也比较不会对新手造成困惑，另外就是其他各种编程范式(包含OOP, FP)其实都是**想法的集合，而非严格的指南(Guideline)**，我们应该更宽松的看待FRP而不是给他一个严格的定义。

关于Reactive Extension (Rx)

Rx 最早是由微软开发的LinQ 扩展出来的开源专案，之后主要由社群的工程师贡献，有多种语言支援，也被许多科技公司所采用，如Netflix, Trello, Github, Airbnb...等。

Rx 的相关资讯

- 开源专案(Apache 2.0 License)
- 多种语言支持
    - JavaScript
    - Java
    - C#
    - Python
    - Ruby
    - ...(太多了列不完)
- 官网
- ~~微软目前最成功的开源专案~~

> LinQ 念做Link，全名是Language-Integrated Query，其功能很多元也非常强大；学RxJS 可以不用会。

Functional Reactive Programming

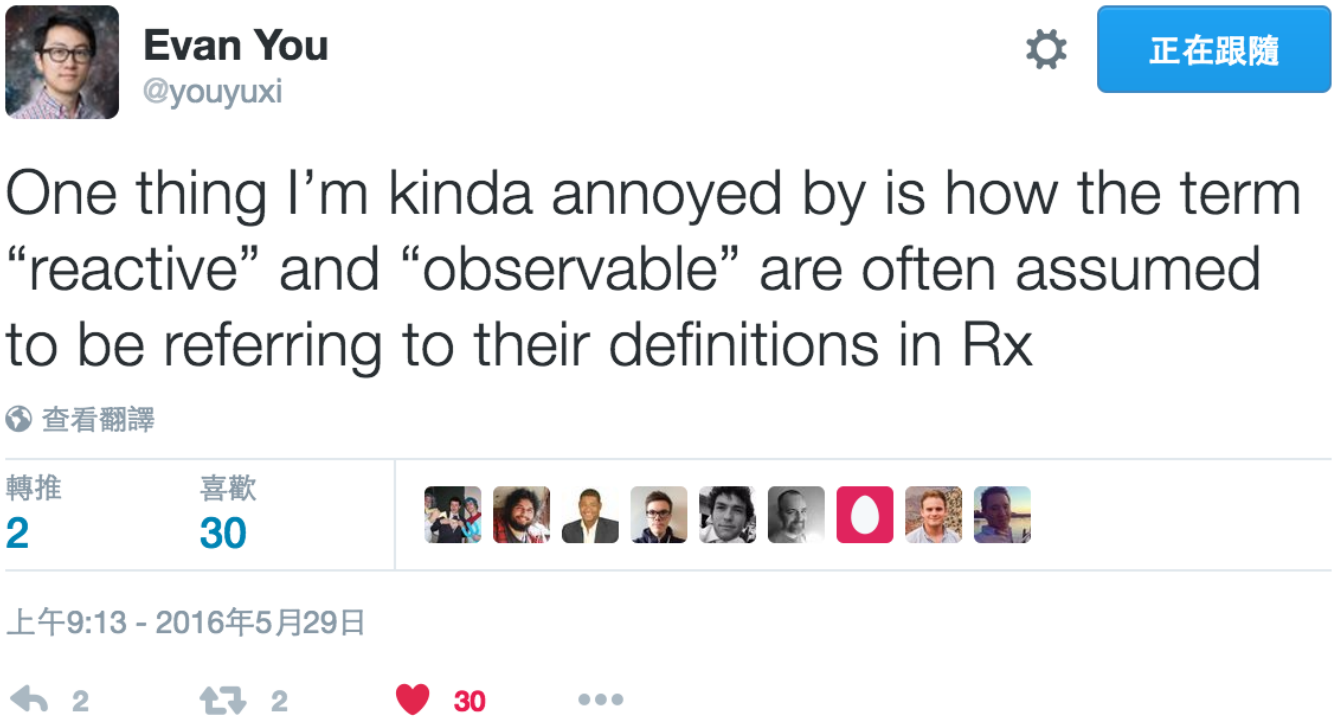
Functional Reactive Programming是一种编程范式(programming paradigm)，白话就是一种**写程式的方法论**！举个例子，像OOP就是一种编程范式，OOP告诉我们要使用物件的方式来思考问题，以及撰写程式，而Functional Reactive Programming基本涵盖了Reactive

Functional Programming大部分的人应该多多都有接触过，这也是RX学习过程中的重点之一，我们之后会花两天的篇幅来细讲Functional Programming。如果要用一句话来总结Functional Programming，那就是**用function来思考我们的问题，以及撰写程式**

> 在下一篇文章会更深入的讲解Functional Programming

Reactive Programming

> 很多人一谈到Reactive Programming 就会直接联想到是在讲RxJS，但实际上Reactive Programming 仍是一种编程范式，在不同的场景都有机会遇到，而非只存在于RxJS，尤雨溪(Vue的作者)就曾在twitter 对此表达不满！



Reactive Programming简单来说就是**当变数或资源发生变动时，由变数或资源自动告诉我发生变动了**

这句话看似简单，其实背后隐含两件事

- 当发生变动=> 非同步：不知道什么时候会发生变动，反正变动时要跟我说
- 由变数自动告知我=> 我不用写通知我的每一步程式码

由于最近很红的Vue.js 底层就是用Reactive Programming 的概念实作，让我能很好的举例，让大家理解什么是Reactive Programming！

当我们在使用vue开发时，只要一有绑定的变数发生改变，相关的变数及画面也会跟着变动，而开

Rx 基本上就是上述的两个观念的结合，这个部份读者在看完之后的文章，会有更深的体悟。

今日小结

今天这篇文章主要是带大家了解为什么我们需要RxJS，以及RxJS 的基本介绍。若读者还不太能吸收本文的内容，可以过一段时间后再回来看这篇文章会有更深的体会，或是在下方留言给我！

Tags

JavaScript RxJS Observable Reactive Programming Functional Programming RxJS 30 Days

Prev Next