30 天精通RxJS (28)：Scheduler 基本观念

Jan 11th, 2017 . 4 mins read

> 不晓得读者们还记不记得，我们在前面的文章中有提到Scheduler 是为了解决RxJS 衍生的最后一个问题，而我们现在就在揭晓这个谜底。

其实RxJS 用久了之后就会发现Observable 有一个优势是可以同时处理同步和非同步行为，但这个优势也带来了一个问题，就是我们常常会搞不清处现在的observable 执行方式是同步的还是非同步的。换句话说，我们很容易搞不清楚observable 到底什么时候开始发送元素！

举例来说，我们可能很清楚 `interval` 是非同步送出元素的，但 `range` 呢？ `from` 呢？他们可能有时候是非同步有时候是同步，这就会变得有点困扰，尤其在除错时执行顺序就非常重要。

而Scheduler 基本上就是拿来处理这个问题的！

什么是Scheduler？

Scheduler 控制一个observable 的订阅什么时候开始，以及发送元素什么时候送达，主要由以下三个元素所组成

- Scheduler 是一个资料结构。它知道如何根据优先级或其他标准来储存并伫列任务。
- Scheduler 是一个执行环境。它意味着任务何时何地被执行，比如像是立即执行、在回呼(callback)中执行、setTimeout 中执行、animation frame 中执行
- Scheduler是一个虚拟时钟。它透过 `now()` 这个方法提供了时间的概念，我们可以让任务在特定的时间点被执行。

简言之Scheduler 会影响Observable 开始执行及元素送达的时机，比如下面这个例子

```
var observable = Rx.Observable.create(function (observer) {
    observer.next(1);
    observer.next(2);
    observer.next(3);
    observer.complete();
});
```

```
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
console.log('after subscribe');

// "before subscribe"
// "after subscribe"
// 1
// 2
// 3
// "complete"
```

JSBin

上面这段程式码原本是同步执行的，但我们用了 `observable.observeOn(Rx.Scheduler.async)` 原本是同步执行的就变成了非同步执行了。

有哪些Scheduler 可以用

目前RxJS 5 Scheduler 跟RxJS 4.x 以前的版本完全不同，在RxJS 5 当中有提供四个scheduler，预设为undefined 会直接以递回的方式执行

- queue
- asap
- async
- animationFrame

这四个scheduler 我们会在下面搭配程式码一一讲解

> RxJS 5跟RxJS 4.x预设的Scheduler不同，所以在某些使用情境下会出现不同的结果，例如 这个 issue，请特别注意。

使用Scheduler

要使用Scheduler除了前面用到的observeOn()方法外，以下这几个creation operators最后一个参数都能接收Scheduler

- bindCallback
- bindNodeCallback
- combineLatest
- concat
- empty
- from
- fromPromise
- interval
- merge
- of
- range
- throw
- timer

例如下面这个例子

```
var observable = Rx.Observable.from([1,2,3,4,5], Rx.Scheduler.async);
```

另外还有多个operators最后一个参数可以传入Scheduler这边就不一一列出，这已参考官方的 文件 ，最通用的方式还是 observeOn() 只要是observable就可以用这个方法。

queue

queue 的運作方式跟預設的立即執行很像，但是當我們使用到遞回的方法時，他會佇列這些行為而非直接執行，一個遞回的 operator 就是他會執行另一個 operator，最好的例子就是 repeat() ，如果我們不給他參數的話，他會執行無限多次，像下面這個例子

```
Rx.Observable.of(10).repeat().take(1)
.subscribe(console.log);
```

值。

使用情境：

queue 很適合用在會有遞回的 operator 且具有大量資料時使用，在這個情況下 queue 能避免不必要的效能損耗。

asap

asap 的行為很好理解，它是非同步的執行，在瀏覽器其實就是 setTimeout 設為 0 秒 (在 NodeJS 中是用 process.nextTick)，因為行為很好理解這裡就不寫例子了。

使用情境：

asap 因為都是在 setTimeout 中執行，所以不會有 block event loop 的問題，很適合用在永遠不會退訂的 observable，例如在背景下持續監聽 server 送來的通知。

async

這個是在 RxJS 5 中新出現的 Scheduler，它跟 asap 很像但是使用 setInterval 來運作，通常是跟時間相關的 operator 才會用到。

animationFrame

這個相信大家應該都知道，他是利用 `Window.requestAnimationFrame` 這個 API 去實作的，所以執行週期就跟 `Window.requestAnimationFrame` 一模一樣。

使用情境：

在做复杂运算，且高频率触发的UI 动画时，就很适合使用animationFrame，以可以搭配throttle operator 使用。

今日小结

这篇文章简单的介绍了RxJS 的Scheduler，因为篇幅的关系没有办法很细的去讲，但实务上Scheduler 的使用非常简单，只要在operator 的最后一个参数加上去或是用observeOn 就可以了。平常其实不太需要用到Scheduler，尤其在RxJS 5 中已经有针对各种情况给不同的预设，笔者最常用到的还是animationFrame！

不知道今天读者们有没有收获呢？如果有任何疑问，欢迎在下方留言给我，谢谢。

    

JavaScript RxJS Observable Scheduler RxJS 30 Days

Prev Next