Angular 2 開發實戰：進階開發篇

RxJS 新手入門

多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

部落格：http://blog.miniasp.com/

The ReactiveX library for JavaScript.

簡介 RxJS

什麼是 RxJS http://reactivex.io/

- 一組可用來處理**非同步/事件**的 JavaScript 函式庫
  - 非同步
    - AJAX / XHR (XMLHttpRequest) / fetch API
    - Service Worker / Node Stream
    - setTimeout / setInterval
    - Promise
  - 事件
    - 各式 DOM 事件 (click, dblclick, keyup, mousemove, ...)
    - CSS 動畫事件 (CSS3 transitionEnd event)
    - HTML5 Geolocation / WebSockets / Server Send Events

關於 ReactiveX 用到的設計樣式

- 實作了觀察者樣式 (Observer pattern)
- 用到了迭代器模式 (Iterator pattern)
- 也用到函式編程與集合等設計樣式
  (Functional reactive programming)
- 主要目的

有效管理**非同步環境**下的**事件資料**！

了解 RxJS 的核心概念

- Observable　**可觀察的物件**
  - 代表一組**未來**即將產生的**事件資料**（被觀察的物件）
- Observer　**觀察者物件**
  - 代表一個用來接收「**觀察結果**」的物件（收到的就是**事件資料**）
  - 觀察者物件就是一個物件包含 3 個含有**回呼函式**的屬性（next , error , complete）
- Subscription　**訂閱物件**
  - 代表正在執行 Observable 的執行個體 (可用來取消執行)
- Operators　**運算子**
  - 必須擁有**函數編程**中所定義的 **純函數** (pure functions) 特性（**沒有副作用的函式**）
  - 主要用來處理一系列的事件資料集合
  - 常見的運算子包含 map, filter, concat, flatMap , switchMap , ...
- Subject　**主體物件**
  - 如同 EventEmitter 一樣，主要用來**廣播**收到的事件資料給多個 Observer (觀察者)
- Schedulers　**排程控制器**
  - 用來集中管理與調度多重事件之間的資料，以控制**事件併發**情況 (control concurrency)

RxJS 的版本差異

- RxJS 4
  - https://github.com/Reactive-Extensions/RxJS
  - 當初由 Microsoft 積極發展的專案，並與社群知名的開元開發者進行協作。
- RxJS 5
  - https://github.com/ReactiveX/rxjs
  - 提供比前版 RxJS 更好的執行效能
  - 遵循 ECMAScript Observable 標準進行實作 (Observable Spec Proposal)
  - 在多種格式之間提供更加模組化的檔案結構
  - 提供比前版 RxJS 更清楚的偵錯資訊 (more debuggable call stacks)
  - Angular 2 採用 RxJS 5 最新版本！

RxJS: Getting Started

RxJS 快速上手

快速示範 RxJS 的運作方式

- 建立可觀察的 Observable 物件

```
var clicks$ = Rx.Observable.fromEvent(document, 'click');
```

- 建立觀察者物件 (Observer)

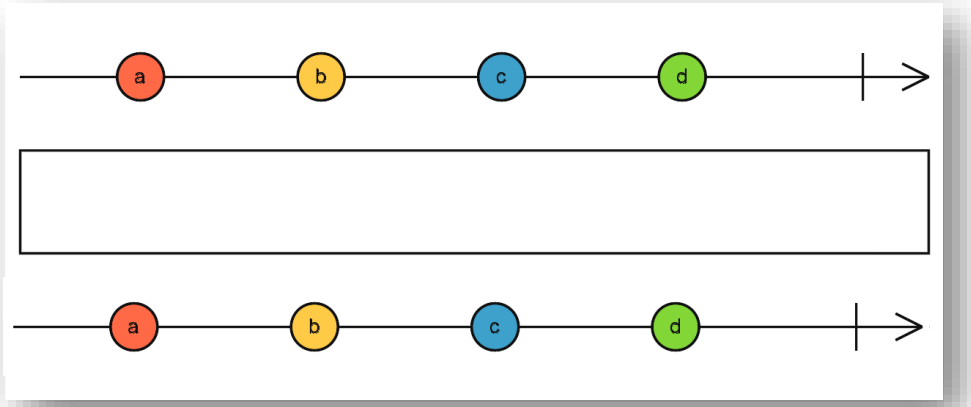
```
var observer = { next: (x) => console.log(x); }
```

- 建立訂閱物件 (訂閱 Observable 物件，並傳入 Observer 觀察者物件)

```
var subs$ = clicks$.subscribe(observer);
```

- 取消訂閱 Subscription 物件

```
subs$.unsubscribe();
```



(承上頁) 簡化 Observer 的寫法

- 建立可觀察的 Observable 物件

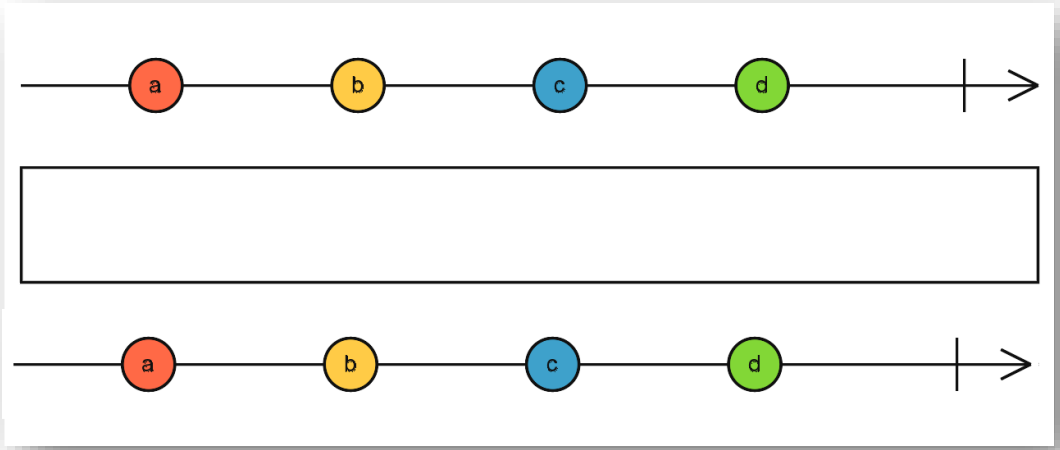
```
var clicks$ = Rx.Observable.fromEvent(document, 'click');
```

- 建立訂閱物件 (訂閱 Observable 物件並自動建立觀察者物件)

```
var subs$ = clicks$.subscribe((x) => console.log(x));
```

- 取消訂閱 Subscription 物件

```
subs$.unsubscribe();
```



示範 RxJS 如何透過運算子過濾資料

- 建立可觀察的 Observable 物件

```
var clicks$ = Rx.Observable.fromEvent(document, 'click');
```

- 套用 `filter` 運算子

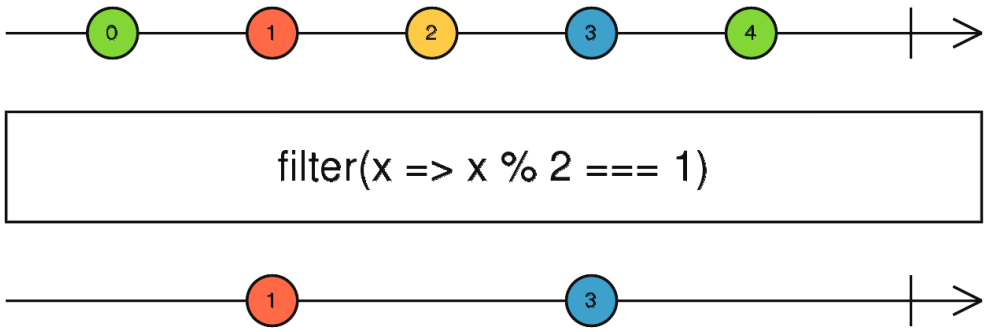
```
clicks$ = clicks$.filter(x => x.clientX < 100);
```

- 建立訂閱物件 (訂閱 Observable 物件並自動建立觀察者物件)

```
var subs$ = clicks$.subscribe((x) => console.log(x));
```

- 取消訂閱 Subscription 物件

```
subs$.unsubscribe();
```



示範 RxJS 如何套用多個運算子

- 建立可觀察的 Observable 物件

```
var clicks$ = Rx.Observable.fromEvent(document, 'click');
```

- 套用 `filter` 運算子

```
clicks$ = clicks$.filter(x => x.clientX < 100);
```

- 設定最多取得兩個事件資料就將 Observable 物件設為**完成**

```
clicks$ = clicks$.take(2);
```

- 建立訂閱物件 (訂閱 Observable 物件並自動建立觀察者物件)

```
var subs$ = clicks$.subscribe((x) => console.log(x));
```

- 取消訂閱 Subscription 物件

```
subs$.unsubscribe();
```

示範 RxJS 主體物件 (Subject) 的用法

- 建立主體物件 (Subject) (之後要靠這個主體物件進行廣播)

```
var subject = new Rx.Subject();
```

- 建立可觀察的 Observable 物件

```
var clicks$ = Rx.Observable.fromEvent(document, 'click');
```

- 設定最多取得兩個事件資料就將 Observable 物件設為**完成**

```
clicks$ = clicks$.take(2);
```

- 設定將 clicks$ 全部交由 subject 主體物件進行廣播

```
clicks$.subscribe(subject);
```

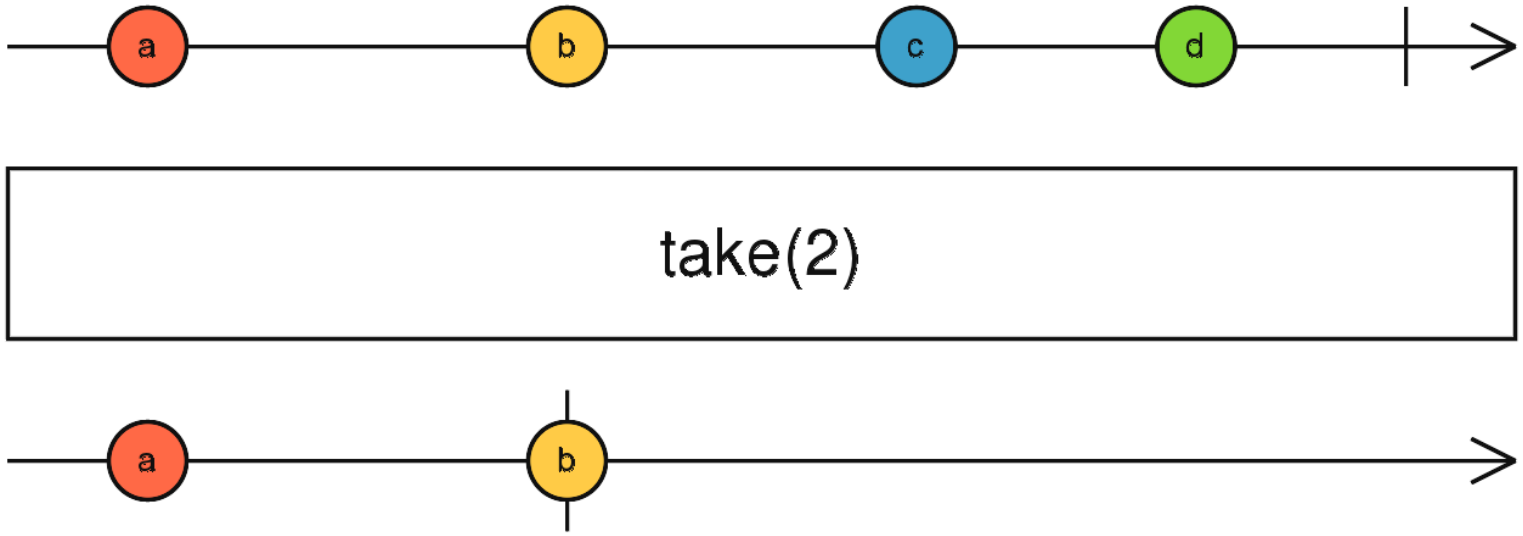
- 最後再由 subject 去建立 Observer 觀察者物件

```
var subs1$ = subject.subscribe((x) => console.log(x.clientX));
var subs2$ = subject.subscribe((x) => console.log(x.clientY));
```

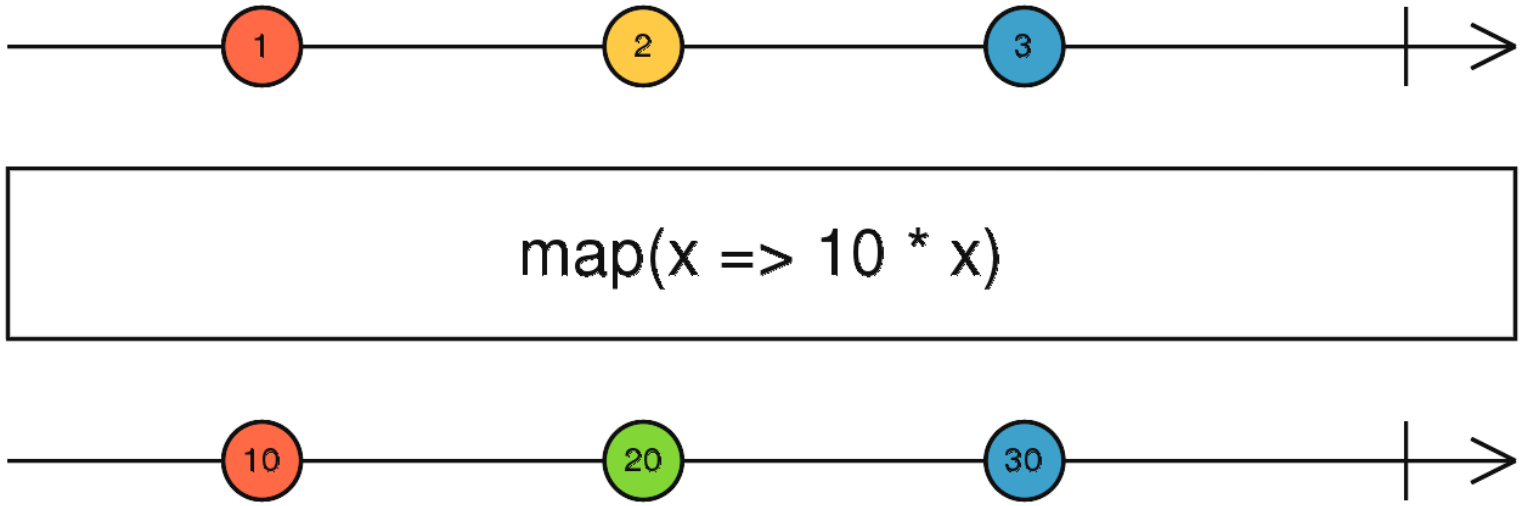
- 取消訂閱 Subscription 物件

```
subs1$.unsubscribe();  subs2$.unsubscribe();
```

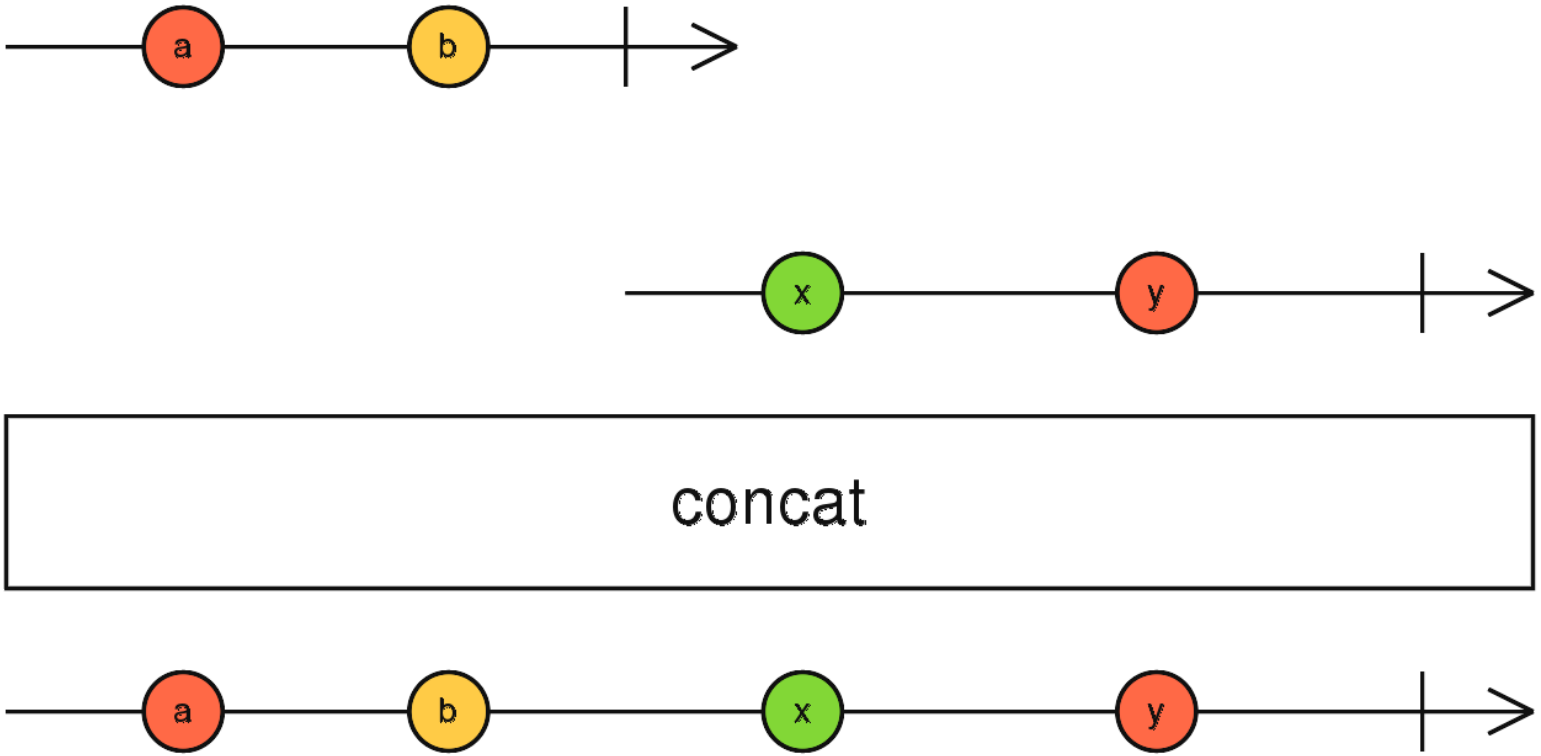
彈珠圖 (Marbles Diagram) - take



彈珠圖 (Marbles Diagram) - map

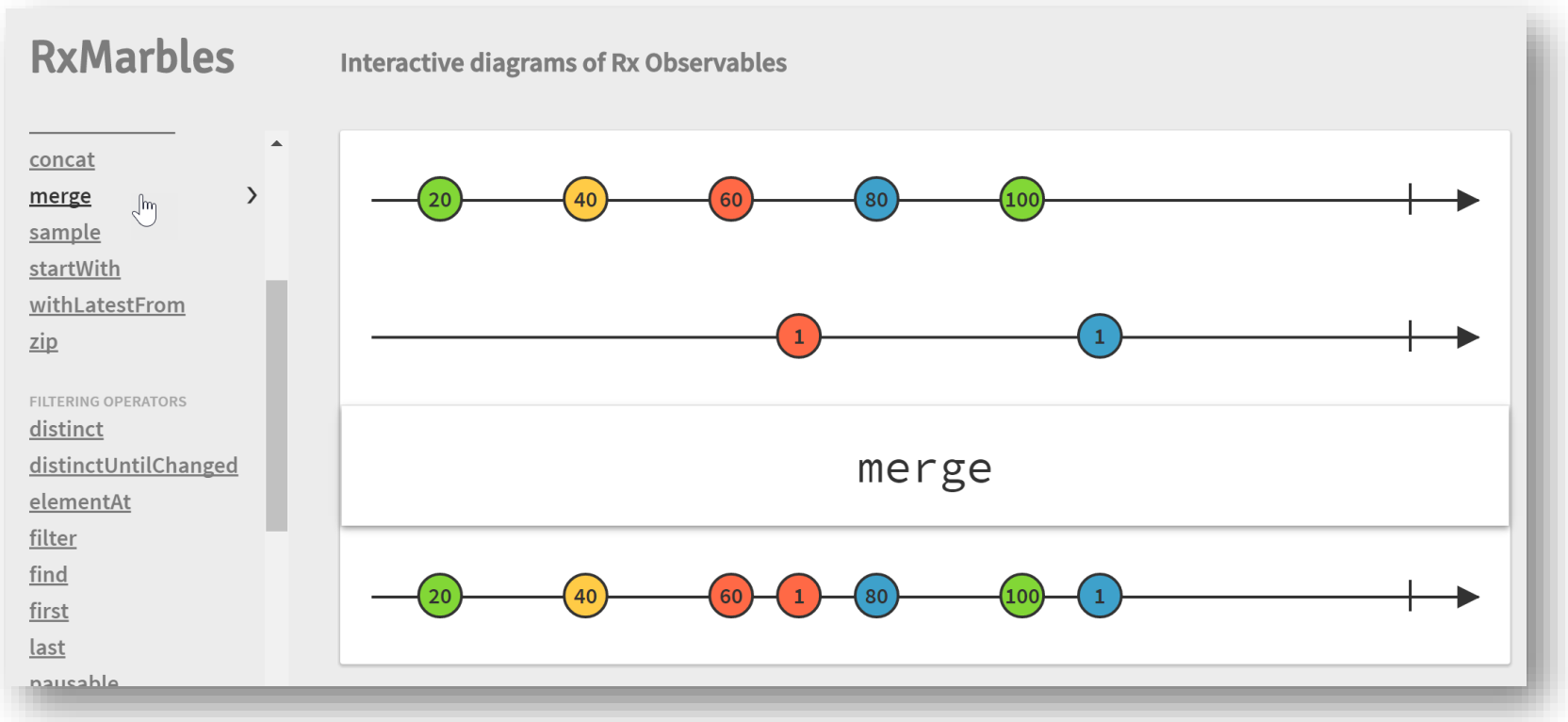


彈珠圖 (Marbles Diagram) - concat



互動式呈現彈珠圖 (Marbles Diagram)

http://rxmarbles.com/



Choose an operator

選擇一個 RxJS 運算子

運算子的分類

- Choose an operator
  - 建立運算子 (Creation Operators)
  - 轉換運算子 (Transformation Operators)
  - 過濾運算子 (Filtering Operators)
  - 組合運算子 (Combination Operators)
  - 多播運算子 (Multicasting Operators)
  - 錯誤處理運算子 (Error Handling Operators)
  - 工具函式運算子 (Utility Operators)
  - 條件式與布林運算子 (Conditional and Boolean Operators)
  - 數學與彙總運算子 (Mathematical and Aggregate Operators)

建立運算子 (Creation)

- 負責建立 Observable 物件
- 常用運算子
  - create
  - empty
  - from
  - fromEvent
  - fromPromise
  - interval
  - never
  - of
- 其他運算子
  - ajax
  - defer
  - repeat
  - generate
  - repeatWhen
  - bindCallback
  - bindNodeCallback
  - fromEventPattern
  - range
  - throw
  - timer

轉換運算子 (Transformation)

- 負責將 Observable 傳入的資料轉換成另一種格式
- 常用運算子
  - buffer
  - concatMap
  - groupBy
  - map
  - mergeMap
  - pluck
  - switchMap
  - window
- 其他運算子
  - bufferCount
  - bufferTime
  - bufferToggle
  - bufferWhen
  - concatMapTo
  - exhaustMap
  - expand
  - mapTo
  - mergeMapTo
  - mergeScan
  - pairwise
  - partition
  - scan
  - switchMapTo
  - windowCount
  - windowTime
  - windowToggle
  - windowWhen

過濾運算子 (Filtering)

- 負責將 Observable 傳入的資料過濾/篩選掉
- 常用運算子
  - debounce
  - distinct
  - distinctUntilChanged
  - filter
  - first
  - ignoreElements
  - skip
  - take
  - throttle
- 其他運算子
  - audit
  - auditTime
  - debounceTime
  - elementAt
  - last
  - sample
  - sampleTime
  - distinctKey
  - distinctUntilKeyChanged
  - single
  - skipUntil
  - skipWhile
  - takeLast
  - takeUntil
  - takeWhile
  - throttleTime

組合運算子 (Combination)

- 負責組合多個 Observable
- 常用運算子
  - combineAll
  - combineLatest
  - concat
  - concatAll
  - forkJoin
  - merge
  - mergeAll
  - startWith
- 其他運算子
  - exhaust
  - race
  - withLatestFrom
  - zip
  - zipAll
  - switch

多播運算子 (Multicasting)

- 負責將 Observable 廣撥給多位觀察者
- 常用運算子
  - cache
  - publish
- 其他運算子
  - multicast
  - publishBehavior
  - publishLast
  - publishReplay
  - share

錯誤處理運算子 (Error Handling)

- 負責處理 Observable 觀察過程中出現的例外錯誤
- 常用運算子
  - catch
  - retry
  - retryWhen

工具函式運算子 (Utility)

- 負責提供 Observable 執行過程的工具函式
- 常用運算子
  - do
  - delay
  - delayWhen
  - materialize
  - toArray
  - toPromise
- 其他運算子
  - dematerialize
  - finally
  - let
  - observeOn
  - subscribeOn
  - timeInterval
  - timestamp
  - timeout
  - timeoutWith

條件式與布林運算子 (Conditional and Boolean)

- 負責判斷布林值條件相關的運算子
- 常用運算子
  - defaultIfEmpty
  - find
  - findIndex
- 其他運算子
  - every
  - isEmpty

數學與彙總運算子 (Mathematical and Aggregate)

- 負責將 Observable 傳來的資料進行數學/彙總運算
- 常用運算子
  - count
  - max
  - min
  - reduce

How to use Http & RxJS in Angular 2

在 Angular 2 中使用 Http 與 RxJS

常見 Http 使用情境 (1)

- 建立 Observable

```
let response$ = http.get(url);
```

- 建立訂閱物件 & 取得 Response 物件

```
let subs$ = response$
    .subscribe(res: Response => {
       console.log(res);
    });
```

- 彈珠圖

```
|--------------------res-|

|--------------------res-|
```

常見 Http 使用情境 (2)

- 建立 Observable

```
let response$ = http.get(url);
```

- 使用 map() 運算子

```
let subs$ = response$
    .map(res: Response => res.json())
    .subscribe(res => {
       console.log(res);
    });
```

- 彈珠圖

```
|--------------------res-|
<   map(x => x.json())  >
|--------------------obj-|
```

將 Http 封裝至 Service 元件中

- 建立服務元件
  - `ng generate service data`
- 實作 load() 方法
  - 回傳 `Observable<Response>` 物件
  - 回傳 `Observable<any>` 物件 (透過 map() 轉換)
    - 可以使用 AsyncPipe 元件
- 注意事項
  - 不要在 Service 元件中建立訂閱物件

使用 AsyncPipe 元件的常見錯誤

- 在 View 中使用
  - `(data$ | async).length`
- 錯誤訊息
  - EXCEPTION: Error in ./AppComponent class AppComponent - inline template:7:39 caused by: Cannot read property 'length' of null
- 解決方法
  - `(data$ | async)?.length`

避免重複的 subscribe() 動作

- 注意事項
  - 每次 subscribe() 都會讓 Http 重新發出要求
- 服務元件

```
load() {
  return response$
         .map(res: Response => res.json())
         .share();
}
```

- app.component.ts

```
this.data$ = datasvc.load();
```

- app.component.html

```
*ngFor="let item of (data$ | async)"
(data$ | async).length
```

發送連續的 HTTP 要求 (1)

- 針對需要連續且依序的發送 HTTP 要求
  - 可以用 .concat() 運算子

```
let d1$ = datasvc.delete(1);
let d2$ = datasvc.delete(2);
let d$ = datasvc.load();

let concat$ = Observable.concat(d1$, d2$, d$);

let subs$ = concat$.subscribe(
  () => {},
  () => {},
  () => { console.log('reloaded'); } // completed
);
```

發送連續的 HTTP 要求 (2)

- 針對需要同時/非同步的發送 HTTP 要求
  - 可以用 .concat() 運算子

```
let d1$ = datasvc.delete(1);
let d2$ = datasvc.delete(2);

let combine$ = Observable.combineLatest(d1$, d2$);

let subs$ = combine$.subscribe(
  () => {},
  () => {},
  () => { console.log('reloaded'); } // completed
);
```

對 Http 進行錯誤處理

- 針對 HTTP 回應 4xx 與 5xx 的例外狀況

```
let d1$ = datasvc.delete(1);
let d2$ = datasvc.delete(2);
let d$ = datasvc.load();

let concat$ = Observable.concat(d1$, d2$, d$);

let subs$ = concat$.subscribe(
  () => {},
  (err) => { console.log(err); },
  () => {}
);
```

取消一個正在進行中的 Http 要求

- 建立 Observable

```
this.subs$ = response$
    .map(res: Response => res.json())
    .subscribe(value => {
       console.log(value);
    });
```

- 取消訂閱

```
this.subs$.unsubscribe();
```

使用 Subject 物件

- 建立 Observable

```
let response$ = http.get(url);
```

- 建立主體物件 & 將 Observable 觀察到的物件都傳入主體

```
let subj$ = new Subject();
response$
    .map(res: Response => res.json())
    .subscribe(subj$);
```

- 將主體物件回傳

```
this.data$ = subj$
```

- 由元件去訂閱 Subject 的資料，訂閱多次也只會有一個主體

相關連結

- ReactiveX 官網
- 30 天精通 RxJS
- RxJS API Document
- RxJS Overview
- RxMarbles: Interactive diagrams of Rx Observables
- The Will Will Web | 前端工程研究：理解函式編程核心概念與如何進行 JavaScript 函式編程
- HTTP Client - ts - GUIDE

聯絡資訊

- The Will Will Web

  記載著 Will 在網路世界的學習心得與技術分享

  - http://blog.miniasp.com/
- Will 保哥的技術交流中心 (臉書粉絲專頁)
  - http://www.facebook.com/will.fans
- Will 保哥的噗浪
  - http://www.plurk.com/willh/invite
- Will 保哥的推特
  - https://twitter.com/Will_Huang