



Programmazione Reattiva con RxJS

Giovanni Pini - 17 Maggio 2017

Chi sono

- Frontend (and Mobile) developer
- In una relazione di odio e amore con JavaScript



gpini

<https://github.com/gpini/rxjs-course>

Do





Di cosa mi occupo

- Gestione di flotte di veicoli (Fleet management)
- Ottimizzazione dei percorsi
- Rinnovamento dell'interfaccia utente con Angular2



Cosa vedremo

- RxJS
 - Observables
 - Funzioni utili
 - Esempi
- Integrazione con framework
- Live coding



Cosa aspettarsi

- Non sono un Guru
- Porto la mia esperienza di qualche mese di utilizzo di questa tecnologia
- In molti casi non tornerei mai indietro



Domande di rito

- JavaScript
- ES6
- Promise

ES6 Crash Course

- let, const
- Arrow functions

```
// ES5
var add = function (num1, num2) {
  return num1 + num2;
}

// ES6
var add = (num1, num2) => num1 + num2
```




Gestione eventi in SPA

- Eventi asincroni
 - eventi scatenati dall'utente (click, tastiera, ecc...)
 - chiamate alle API
 - timeout
 - WebSocket
 - Web Workers
- Come gestirli: Callback, Promise



Callback

- Vecchio approccio
- Callback Hell



Promise

- Oggetto Javascript
- 3 stati: pending, resolved, rejected
- Introdotta nativamente in ES2015 (ES6)
- Risolve callback hell
- Problemi
 - una sola esecuzione
 - cancellazione difficoltosa



Reactive Programming

- **Paradigma** che lavora con flussi di dati (stream) asincroni
- **Stream** = sequenza di eventi ordinati nel tempo che prevedono un eventuale termine a causa di completamento e/o errore

RxJS



- ReactiveX (reactivex.io)
 - An API for asynchronous programming with observable streams
 - Open Source
 - Java, C#, Scala, Clojure, C++, Lua, Ruby, Python, Go, Groovy, JRuby, Kotlin, Swift, PHP, Elixir, Dart
- RxJS è l'implementazione JavaScript
 - **ReactiveX** for **JavaScript**



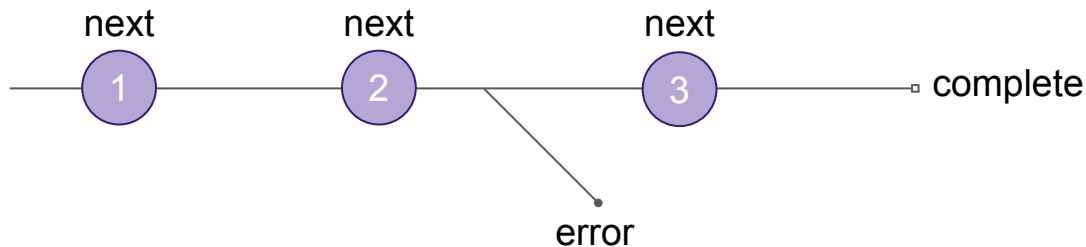
Observables

Partendo da concetti già familiari

- **Collection** nel tempo
- **Promise** che “risolvono” più d'una volta ed eventualmente terminano o vanno in errore

Observables

- Oggetti che rappresentano un flusso di eventi asincroni
- Possono proseguire indefinitamente, terminare o andare in errore

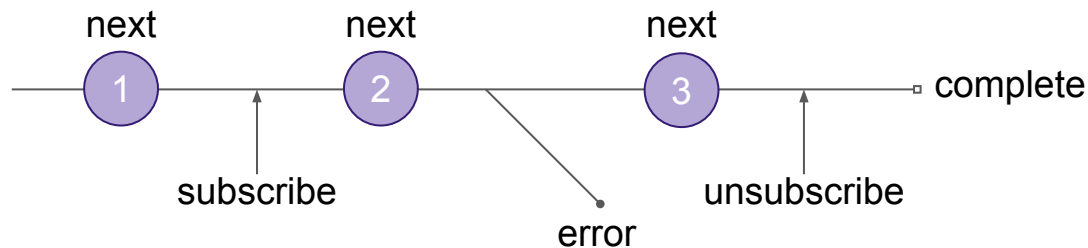




subscribe

- Pattern Observer / Listener
- Eventi di una Observable
 - **next** con “payload”
 - **error** con informazioni
 - **complete**

sottoscrizione



- Error termina come complete
- Ci si può sottoscrivere quante volte si vuole



sottoscrizione

- subscribe
 - nextFn
 - errorFn
 - completeFn
- unsubscribe
 - Memory Leaks



Creazione Observables

- `Observable.of`
- `Observable.create`
- `Observable.fromEvent`
- `Observable.fromPromise`



Creazione Observables

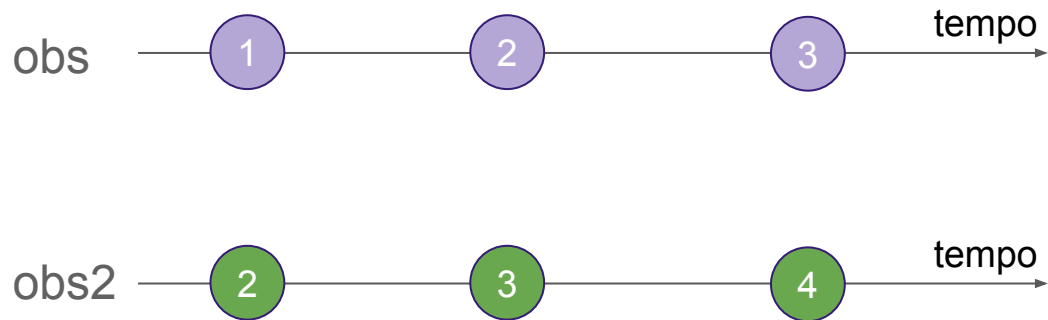
- Numeri incrementali
 - Observable.timer
 - Observable.interval
- Altro
 - Observable.webSocket
 - Observable.empty
 - Observable.throw
 - Observable.never
 - Observable.range

Question time



map

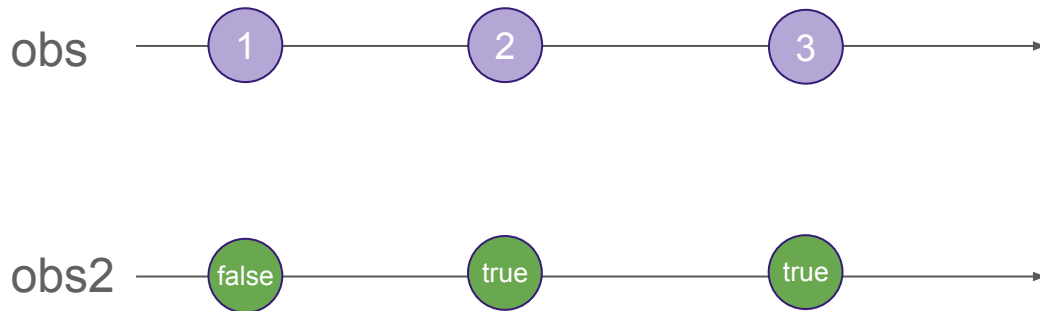
`obs2 = obs.map(x => x + 1)`



`Observable<number> => Observable<number>`

map

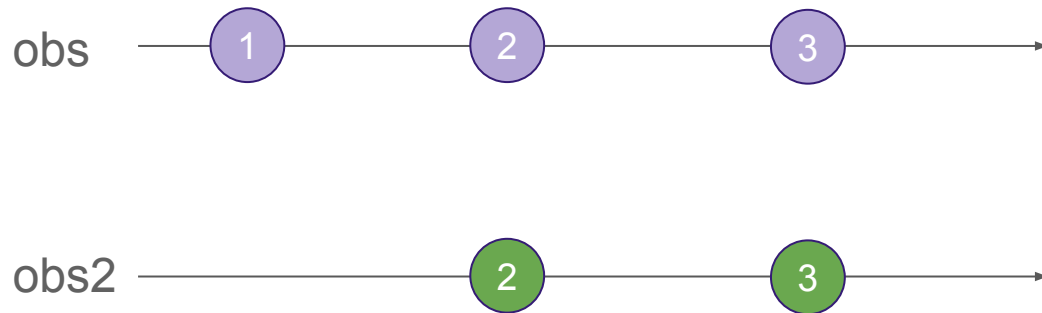
```
obs2 = obs.map(x => x > 1)
```



Observable<number> => Observable<boolean>

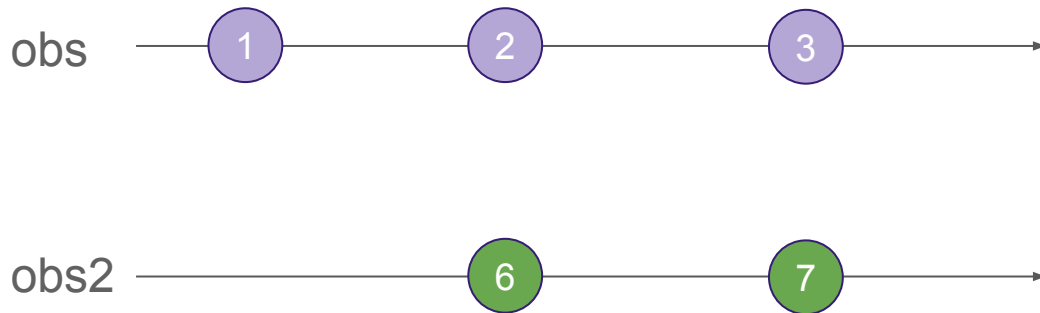
filter

```
obs2 = obs.filter(x => x > 1)
```



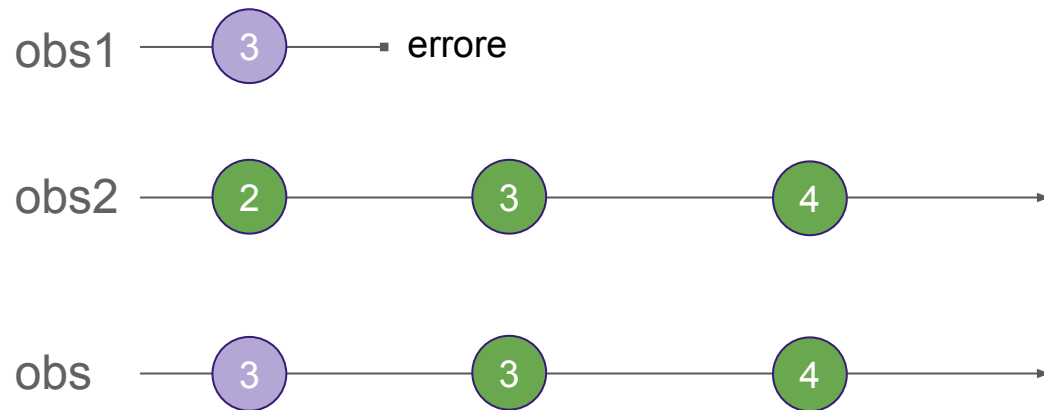
chaining

```
obs2 = obs.filter(x => x > 1).map(x => x + 4)
```



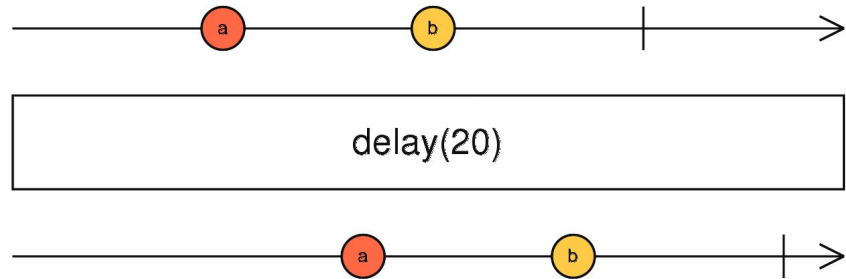
catch

`obs = obs1.catch(() => return obs2)`



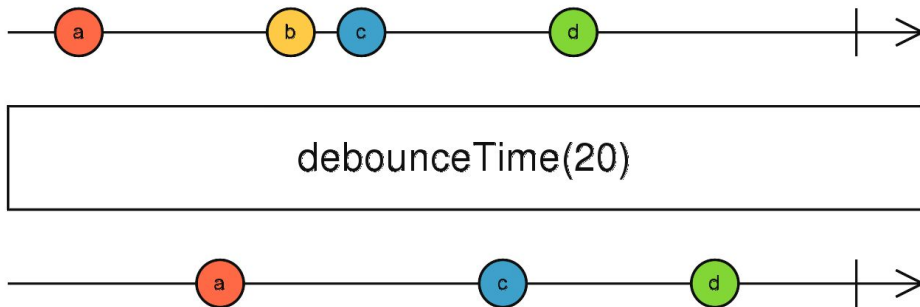
Observables - delay

- delay
 - ritarda l'emissione di N millisecondi



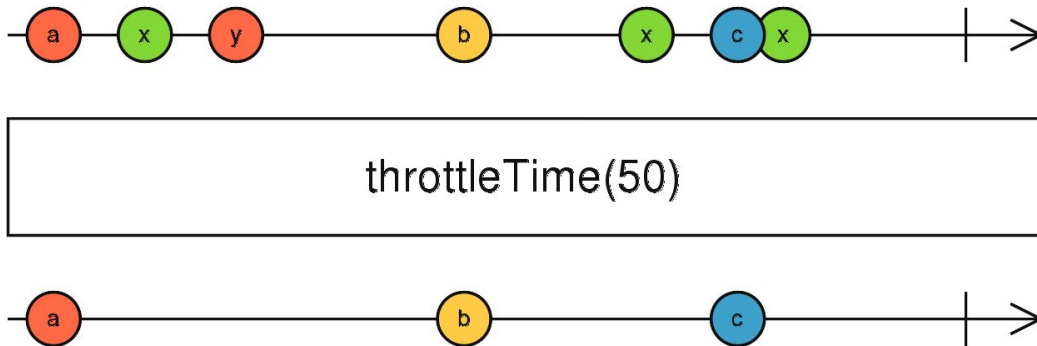
Observables - debounce


- `debounceTime`
 - emette un evento solo se non ne sono avvenuti altri in N millisecondi



Observables - throttle

- `throttleTime`
 - ignora i successivi eventi per N millisecondi



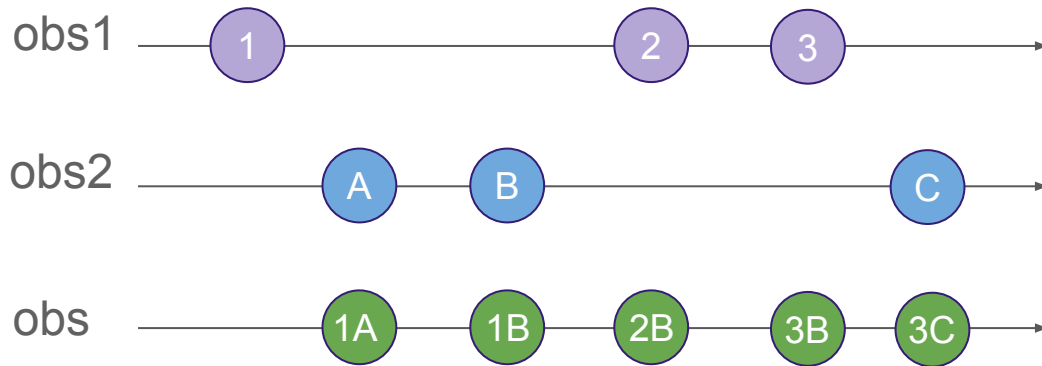


Observables - combinazione

- `combineLatest`
- `exhaustMap`
- `switchMap`
- `mergeMap`

combineLatest

`obs = Observable.combineLatest(obs1, obs2)`



Live coding

- Calcolatore di ipotenusa



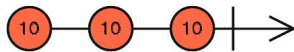


mergeMap & Co

- operazioni asincrone in cascata
- **map**: si parte dall'output della Observable di origine
- **flattening**: della Observable (altrimenti sarebbe Observable di Observable)

mergeMap

Effettua il merge della Observable proiettata

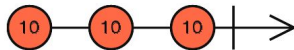


```
mergeMap(i => 10*i——10*i——10*i—| )
```



switchMap

Effettua il merge considerando solo la Observable proiettata più recente



```
switchMap(i => 10*i——10*i——10*i——| )
```



Live coding

- Input di ricerca

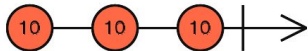


Question time

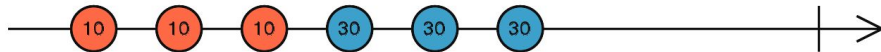


exhaustMap

Effettua il merge solo se la Observable proiettata ha completato



```
exhaustMap(i => 10*i——10*i——10*i—| )
```



Live coding

- Lista con refresh





Observables - pregi

- Non userei più una promise
- Si possono compiere operazioni molto potenti con poche righe di codice
- Si tende a disaccoppiare di più perché si utilizza di più il pattern Observer



Observables - difetti

- C'è una curva di apprendimento
- Operator Agony
 - superarla imparando e usando qualche metodo alla volta
 - non importa che tutti sia una observable
- Gestione race conditions

RxJS nei framework

- Angular2
 - alcuni servizi sono già Observables
 - HttpService
 - async pipe



RxJS nei framework

- Redux / React
 - <https://redux-observable.js.org/>
 - Middleware per Redux
 - **Epic**: gestione dello stream di azioni



```
const pingEpic = action$ =>
  action$.filter(action => action.type === 'PING')
    .delay(1000) // Asynchronously wait 1000ms then continue
    .mapTo({ type: 'PONG' });

// later...
dispatch({ type: 'PING' });
```

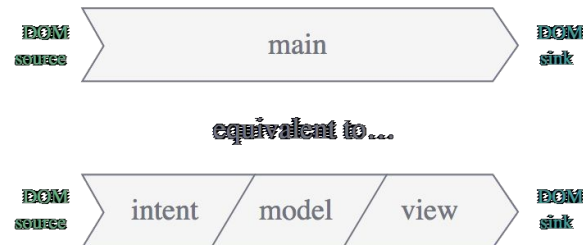
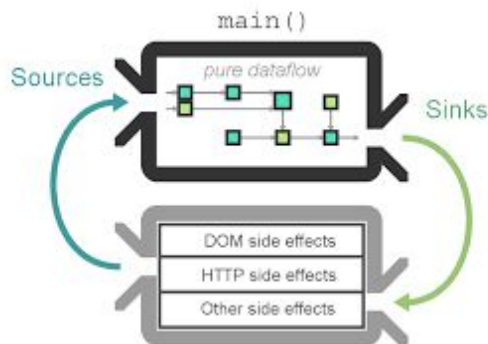
NETFLIX
JavaScript Talks

RxJS + Redux
+ React =
Amazing



RxJS nei framework

- CycleJS
- Model View Intent



Riferimenti

- Documentazione metodi reactivex.io/rxjs/
- Guide e repo <https://github.com/Reactive-Extensions/RxJS>
- Video
 - **Ben Lesh**
 - Altri

NETFLIX
JavaScript Talks

Async
JavaScript
with Rx



RxJS Crash Course



Reactive
Programming
With Observables

Question time





Extra slides



Altre librerie

- xstream <https://staltz.com/xstream/>
- baconjs <http://baconjs.github.io/>
- kefirjs <https://rpominov.github.io/kefir/>
- events <https://nodejs.org/api/events.html>
- mostjs <https://github.com/cujojs/most>

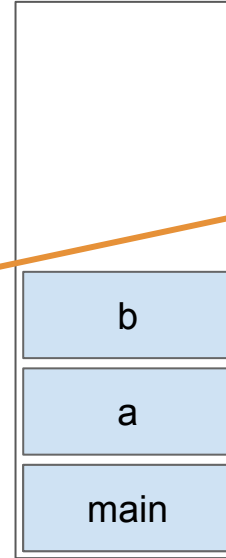
Event loop

What the heck is the event loop anyway

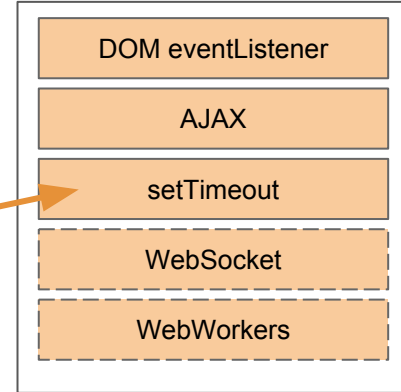


```
function b() {  
  console.log('World');  
}  
  
function a() {  
  console.log('Hello');  
  b();  
}  
  
setTimeout(function() {  
  console.log('Hi')  
}, 0);  
  
a();
```

Stack



WebAPIs

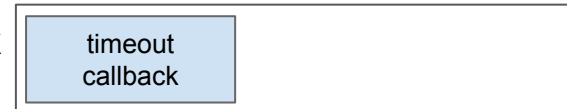


Console

```
> Hello  
> World  
> Hi
```

Event Loop

Callback Queue





Observables - Subject

- Observables che possono autoemettere
- Ereditano da Observables
- Metodi esposti
 - *next(payload)*
 - *error(message)*
 - *complete()*

Cold vs Hot Observables

- Producer = oggetto che produce eventi
- **COLD**: il Producer è creato dall'Observable
- **HOT**: il Producer è creato dall'Observable

```
// COLD
var cold = new Observable((observer) => {
  var producer = new Producer();
  // have observer listen to producer here
});
```

```
// HOT
var producer = new Producer();
var hot = new Observable((observer) => {
  // have observer listen to producer here
});
```