PROGRAMACIÓN REACTIVA CON RXJS

侍 Pablo Magaz

https://pablomagaz.com

twitter.com/pablo_magaz

atsistemas
Consulting, IT Services & Software Development

“La programación reactiva es la programación orientada al manejo de streams de datos asíncronos y la propagación del cambio”

侍 Reactive Extensions (Rx)

Streams representados por secuencia(s) observable(s)

+

Operadores LINQ

```
IEnumerable<string> query = cats
  .Select(cat => cat.Name)
  .Concat(dogs.Select(dog => dog.Name));
```

“Rx es una combinación de las mejores ideas del patrón Observer, el patrón Iterador y la programación Funcional”

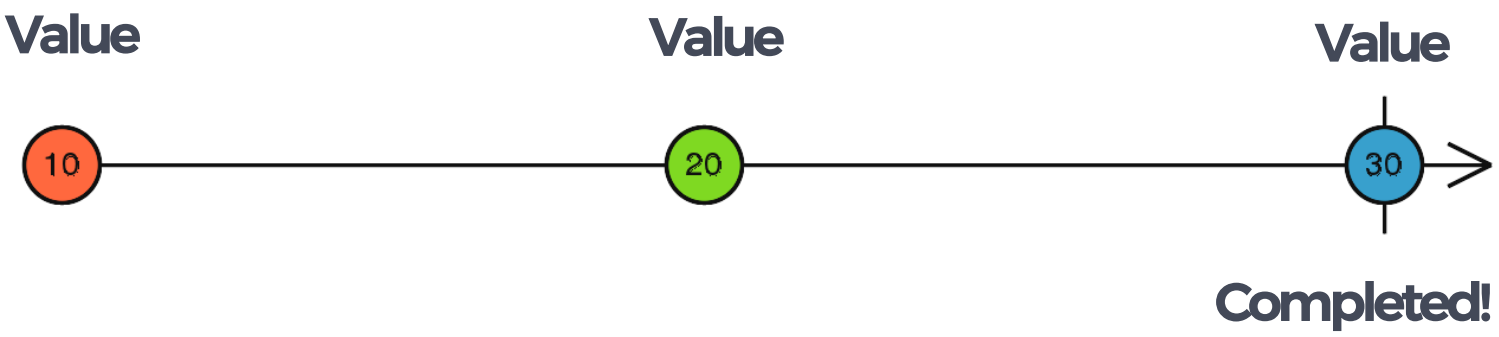
侍 Patrón Observer en RxJs

Observable

```
const myStream$ = Rx.Observable.from([10,20,30]);

//  >> 10, 20, 30
```

侍 Observable

```
const myStream$ = Rx.Observable.from([10,20,30]);
```

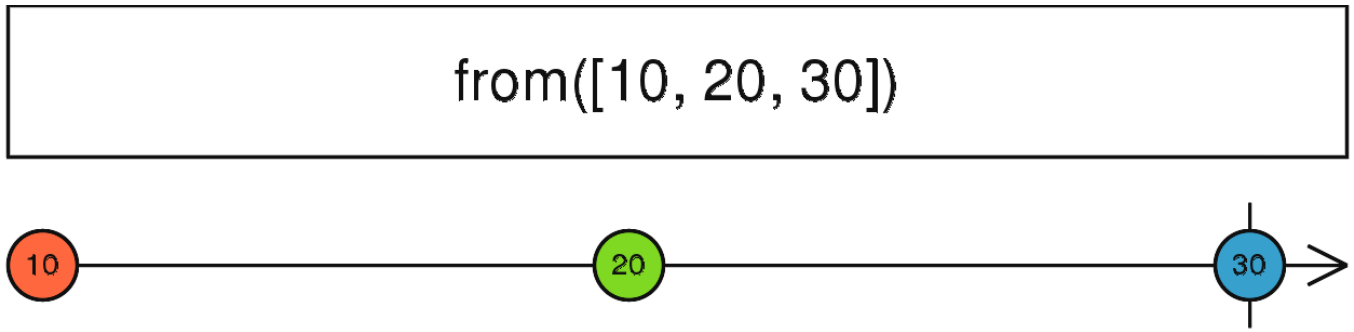


http://rxmarbles.com

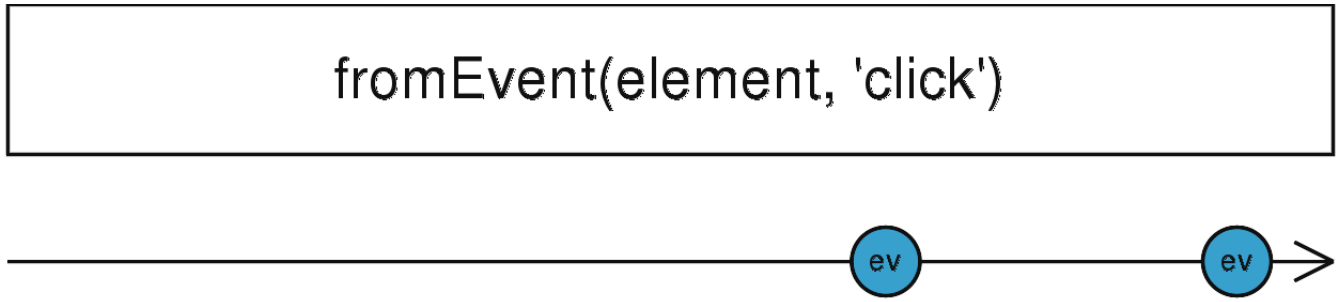
侍 Todo es un Observable

```
const letter$ = Rx.Observable.of('A');

const range$ = Rx.Observable.range(1,8);

const array$ = Rx.Observable.from([1,2,3,4]);

const click$ = Rx.Observable.fromEvent(document, 'click');

const promise$ = Rx.Observable.fromPromise(fetch('/products'));
```

Observable Finito



Observable Infinito



侍 Patrón Observer en RxJs

Observer

```
const myObserver = {
  next: (x) => console.log(`Observer next value: ${ x }`),
   error: (err) => console.error(`Observer error value: ${ x }`),
   complete: () => console.log(`Observer complete notification`),
};
```

Next Notifica un valor emitido por el Observable.

Error Notificación de un error o excepción sucedido.

Complete Notificación sin valor cuando ha finalizado el stream.

ECMAScript Observable

This proposal introduces an **Observable** type to the ECMAScript standard library. The **Observable** type can be used to model push-based data sources such as DOM events, timer intervals, and sockets. In addition, observables are:

```
function listen(element, eventName) {
    return new Observable(observer => {
        // Create an event handler which sends data to the sink
        let handler = event => observer.next(event);

        // Attach the event handler
        element.addEventListener(eventName, handler, true);

        // Return a cleanup function which will cancel the event stream
        return () => {
            // Detach the event handler from the element
            element.removeEventListener(eventName, handler, true);
        };
    });
}
```

https://github.com/tc39/proposal-observable

侍 Subscripción

```
myObservable$.subscribe(myObserver);

Rx.Observable.from([1,2,3])
  .subscribe(
    next => console.log(next),
    err => console.log(err),
    () => console.log('completed!')
); // OUTPUT >> 1, 2, 3, ‘completed!’

myObservable$
  . subscribe(next => console.log(next)); // OUPUT >> 1, 2, 3
```

侍 Pull vs Push

Pull

侍 Pull vs Push

Push

“Nos limitamos a reaccionar a las notificaciones del Observable”

侍 Patrón Iterador

```
const simpleIterator = data => {
  let cursor = 0;
  return {
    next: () => ( cursor < data.length ? data[cursor++] : false )
  }}

const consumer = simpleIterator(['simple', 'data', 'iterator']);

console.log(consumer.next()); // OUTPUT >> 'simple'
console.log(consumer.next()); // OUTPUT >> 'data'
console.log(consumer.next()); // OUTPUT >> 'iterator'
```

侍 Programación Funcional

Las funciones son ciudadanos de primera clase

- Paso de funciones como argumentos de otras funciones

Funciones puras

- Para los mismos argumentos siempre devolverán el mismo valor
- No mutan datos
- No generan efectos secundarios (side effects)

Funciones de orden superior

- Funciones que reciben otras funciones para realizar cálculos

侍 Funciones de Orden Superior

```
const data = [0,1,2,3];

const result = data
  .filter(x => {
    console.log(`filter: ${ x }`);
    return x % 2 === 0;
  })
  .map(x => {
    console.log(`map: ${ x }`);
    return x * x;
  })
  // OUTPUT >> filter: 0, filter: 1, filter: 2, filter: 3, map: 0, map: 2
```

侍 Operadores de RxJs

```
const data = [0,1,2,3];

Rx.Observable.from(data)
 .filter(x => {
  console.log(`filter: ${ x }`);
  return x % 2 === 0;
 })
 .map(x => {
   console.log(`map: ${ x }`);
   return x * x;
 }) .subscribe();
 // OUTPUT >> filter: 0, map: 0, filter: 1, filter: 2, map: 2, filter: 3
```

“Los Operadores siempre devuelven un Observable”


OPERATORS
EVERYWHERE
imgflip.com

侍 Operadores de Filtrado

```
const source$ = Rx.Observable.from([1,2,2,2,3,4,5,6,7,8]);

source$
  .distinct() // 1, 2, 3, 4, 5, 6, 7, 8
  .filter(x => x % 2 === 0) // 2, 4, 6, 8
  .take(3) // 2, 4, 6
  .skip(1) // 4, 6
  .first() // 4
  .subscribe(next => console.log(next));

  // OUTPUT >> 4
```

侍 Operadores Matemáticos

```
const source$ = Rx.Observable.range(1, 8);

source$
  .count(i => i % 2 === 0)
  .subscribe(next => console.log(next))
  // OUTPUT >> 4

const source2$ = Rx.Observable.range(1, 100);

source2$
  .max()
  .subscribe(next => console.log(next))
  // OUTPUT >> 100
```

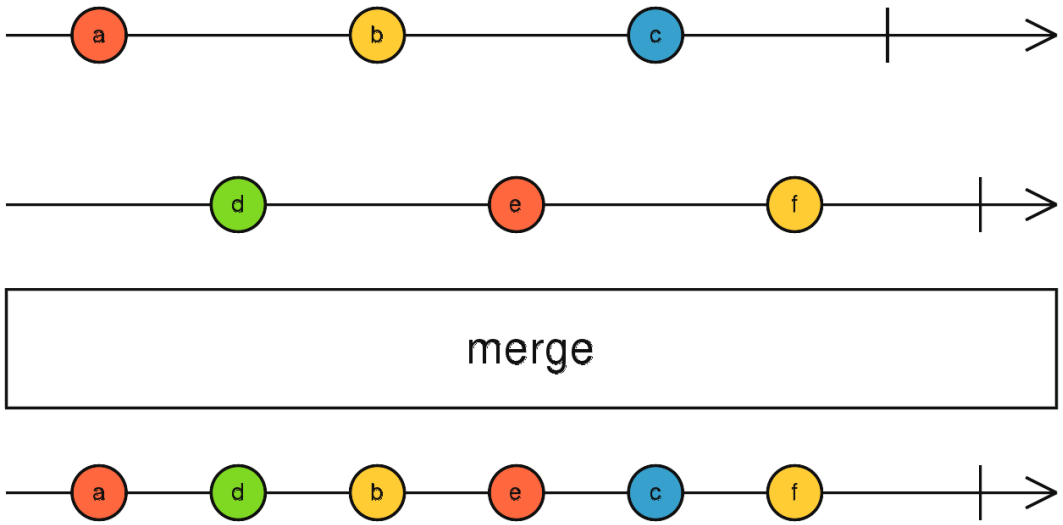
侍 Operadores de Transformación

```
const source$ = Rx.Observable.interval(100);

source$
  .mapTo({ msg: 'Hello codemotion!'  })
  .pluck('msg')
  .map(x => x.toUpperCase())
  .subscribe(next => console.log(next));
  // OUTPUT -> 'HELLO CODEMOTION!',
                'HELLO CODEMOTION!',
                'HELLO CODEMOTION!',
                'HELLO CODEMOTION!',
                ...
```

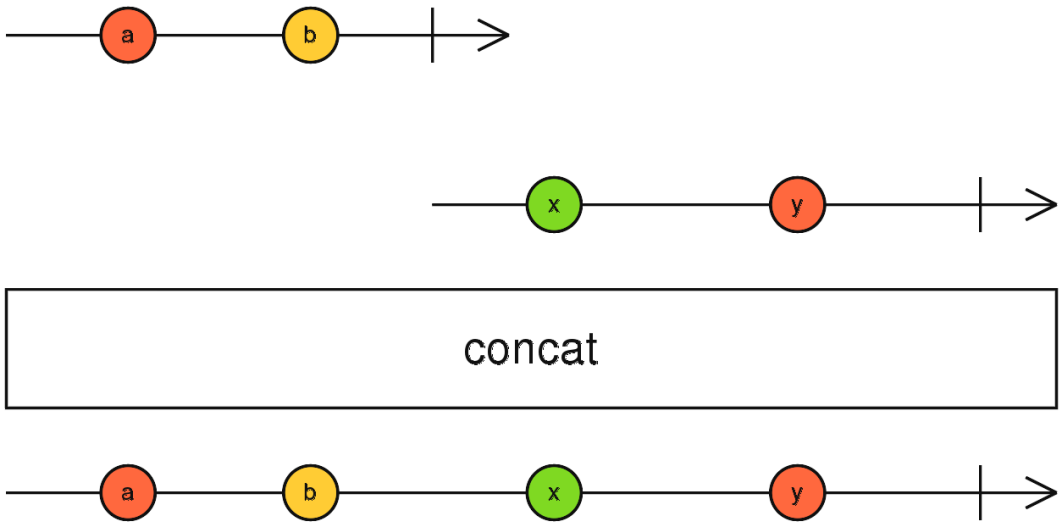
侍 Operadores de Combinación

```
const interval$ = Rx.Observable.interval(100).mapTo('A').take(3);
const interval2$ = Rx.Observable.interval(200).mapTo('B').take(3);

interval$
  .merge(interval2$)
  .subscribe(next => console.log(next));
  // OUTPUT >> A, A, B, A, B, B

interval$
  .concat(interval2$)
  .subscribe(next => console.log(next));
  // OUTPUT >> A, A, A, B, B, B
```

merge



concat



http://rxmarbles.com

侍 Combinando Observables

```
const data = { author: 'Jhon', articles: [
  { id: 11, category: 'music' }, { id: 22, category: 'movies' }, { id: 33, category: 'music' }]};

const service = () => new Promise(resolve => {
   setTimeout(() => { resolve(data) }, 500);
});

const click$ = Rx.Observable.fromEvent(document, 'click');

click$
  .map(x => Rx.Observable.fromPromise(service()))
  .subscribe(next => console.log(next));
```

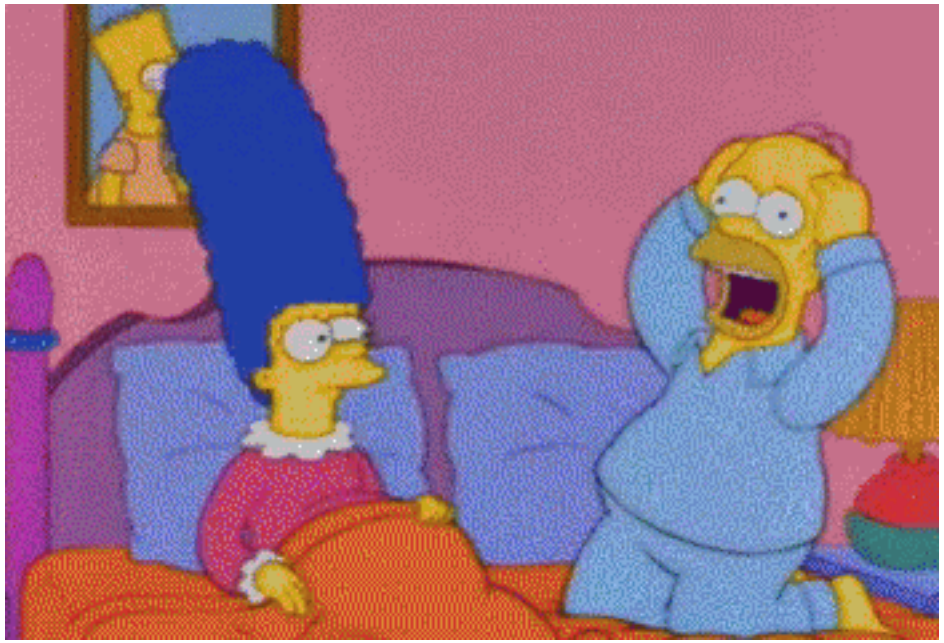
侍 ¡Uops!

```
{
  _catch: function _catch(selector) {
    var operator = new CatchOperator(selector);
    var caught = this.lift(operator);
    return (operator.caught = caught);
},
  _do: function _do(nextOrObserver, error, complete) {
    return this.lift(new DoOperator(nextOrObserver, error, complete));
},
  _finally: function _finally(callback) {
    return this.lift(new FinallyOperator(callback));
},
....
```

侍 Combinando Observables

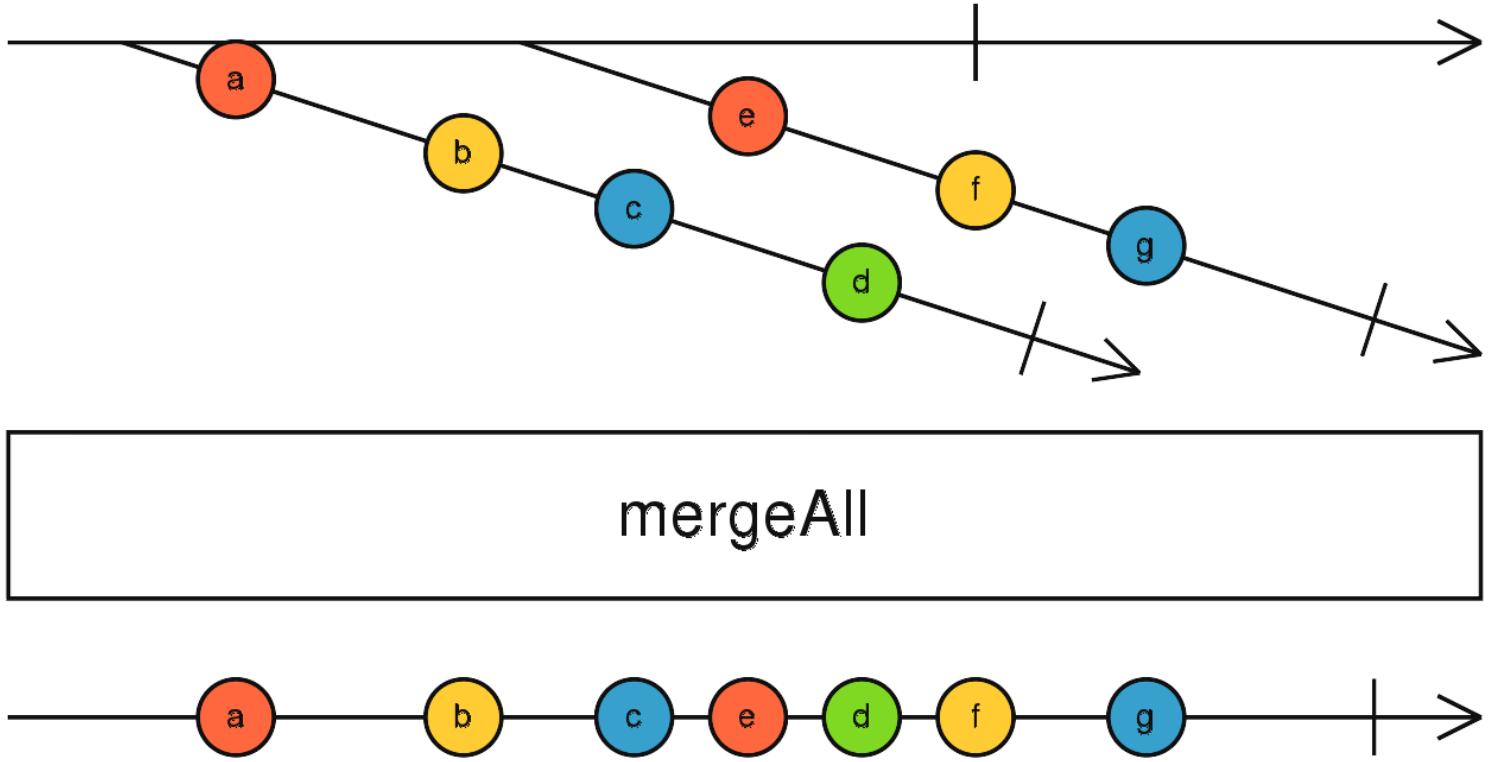
```
const data = { author: 'Jhon', articles: [
  { id: 11, category: 'music' }, { id: 22, category: 'movies' }, { id: 33, category: 'music' }]};

const service = () => new Promise(resolve => {
   setTimeout(() => { resolve(data) }, 500);
});

const click$ = Rx.Observable.fromEvent(document, 'click');

click$
  .map(x => Rx.Observable.fromPromise(service()))
  .subscribe(next => console.log(next));
```

“Un Observable puede emitir otros Observables”

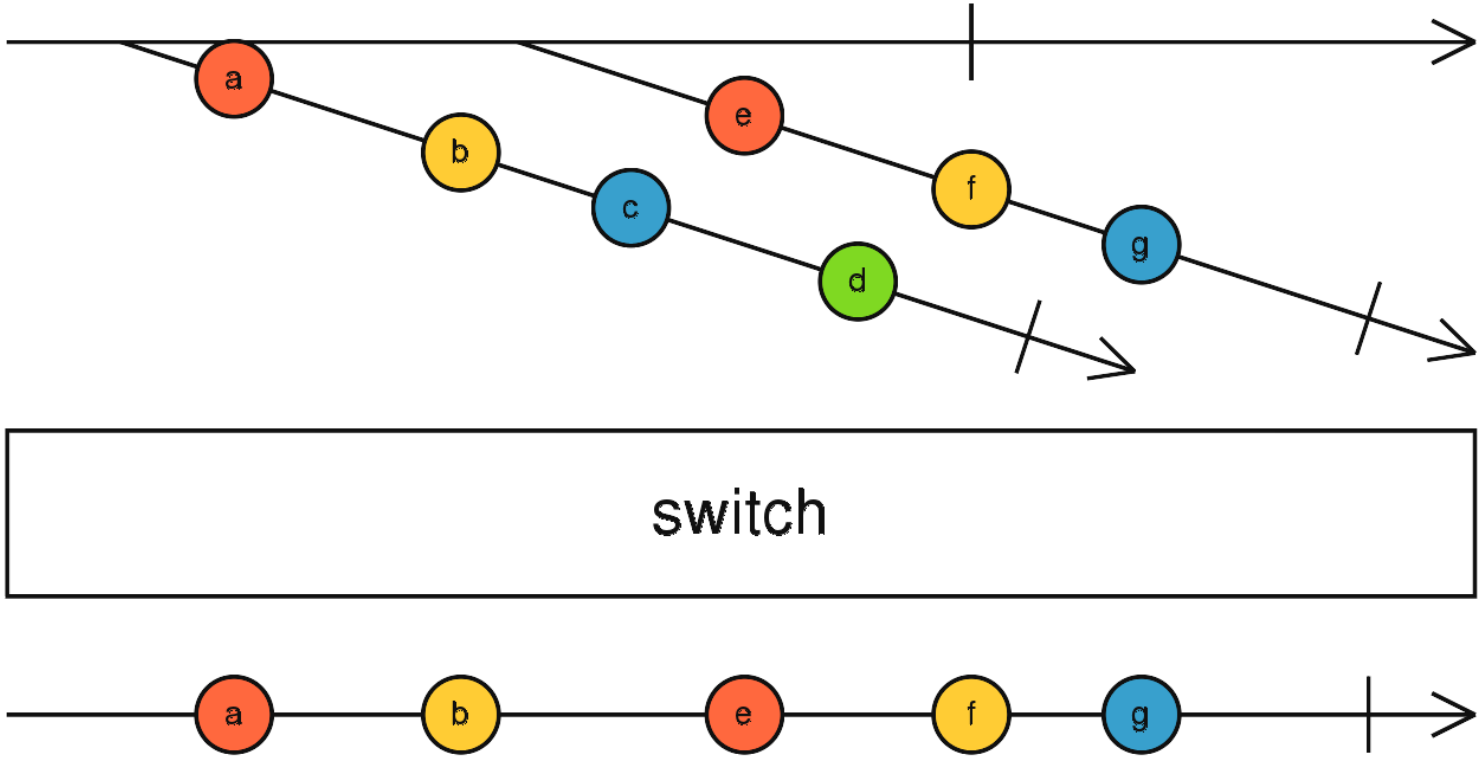


侍 Array de Arrays

```
const arrayOfArrays = [ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ] ];

// FLATTENED OUTPUT  1 -> 1, 2, 3, 4, 5, 6

// FLATTENED OUTPUT  2 -> 1, 3, 5, 2, 4, 6
```



http://rxmarbles.com



http://rxmarbles.com

侍 Aplanando Observables

```
const data = { author: 'Jhon', articles: [ { id: 11, category: ‘movies’} ... ]};
const service = () => new Promise(resolve => resolve(data));

const click$ = Rx.Observable.fromEvent(document, 'click');

click$
  .map(x => Rx.Observable.fromPromise(service()))
  .mergeAll()
  .pluck('articles')
  .subscribe(next => console.log(next));
// OUTPUT >>[{ id: 11, category: 'music’ }, { id: 22, category: 'movies’}, ... ]
```

侍 Aplanando Observables

```
const data = { author: 'Jhon', articles: [ { id: 11, category: ‘movies’} ... ]};

const service = () => new Promise(resolve => resolve(data));

const click$ = Rx.Observable.fromEvent(document, 'click');
const service$ = Rx.Observable.fromPromise(service());

click$
  .mergeMap(x => service$) // map + mergeAll
  .pluck('articles')
  .subscribe(next => console.log(next));
// OUTPUT >> [{ id: 11, category: 'music’ }, { id: 22, category: 'movies’}, ... ]
```

mergeMap = map + mergeAll

swichMap = map + switch

concatMap = map + concatAll

exhaustMap = map + exhaust

侍 Lógica de negocio...

```
const data = { author: 'Jhon', articles: [ { id: 11, category: ‘movies’} ... ]};

const service = () => new Promise(resolve => resolve(data));

const service$ = Rx.Observable.fromPromise(service());


service$
  .mergeMap(x => x.articles)
  .groupBy(article => article.category)
  . mergeMap(group => group
    .reduce(acc => acc + 1, 0)
    .map(total => ({ category: group.key, posts: total }))
  )
.subscribe(next => console.log(next))

// OUTPUT >> { category: ’music’: posts: 2 }, { category: ‘movies’, posts: 1 }
```

侍

```
const inputText = document.getElementById('searchText');
const keyUp$ = Rx.Observable.fromEvent(inputText, 'keyup');
const search$ = text => Rx.Observable.ajax({ crossDomain: true,
  url: `https://en.wikipedia.org/w/api.php?&search=${ text }&action=
 opensearch&origin=*` });

 keyUp$
  .map(e => e.target.value)
  .filter(text => text.length > 2)
  .debounceTime(250)
  .distinctUntilChanged()
  .switchMap(text => search$(text))
  .pluck('response')
  .subscribe(result => drawResults(result));
```

侍 Conclusiones

- RxJs Mola
- Patrón Observer, Iterador y Programación Funcional
- En RxJs todo es un Observable
- Disponemos de operadores para cualquier tarea
- Fácil implementación de lógica de negocio

Gracias.
¿Preguntas?

EL BLOG ISOMÓRFICO

https://pablomagaz.com

twitter.com/pablo_magaz