|  |  |
| --- | --- |
| Configuraciones | Descripción breve  Formas de implementación y comportamientos de la programación reactiva  ALBARRACIN EDUARDO JAVIER |

# ¿Qué es RxJs y para que nos sirve?

Nos sirve para tener información en tiempo real.

¿Cuándo usar Rx?:

* Ante eventos de la interfaz de usuario y que no se bloquee dicha interfaz
* Cuando es necesario notificar sobre cambios en un objeto(s)
* Cuando tenemos comunicación por sockets
* Cuando necesitamos trabajar con flujos de información (Streams)

Las piezas fundamentales de la programación reactiva: Observables, Subscribers y

Observables

* Son la fuente de información
* Pueden emitir múltiples valores, sólo uno o ninguno.
* Pueden emitir errores.
* Pueden ser infinitos, finitos. Cuando son finitos ellos mismos pueden completarse
* Pueden ser síncronos o asíncronos

Subscribers

* Se subscriben a un observable, es decir, estar pendiente de lo que realiza el observable.
* Consumen/ observan la data del observable
* Pueden recibir los errores y eventos del observable

Los subscribers desconocen todo el comportamiento anterior, es decir si la data fue modificada, filtrada, etc.

Operators/ Operadores

* Usados para **transformar** Observables (map, group, scan,…)
* Usados para **filtrar** observables (filter, distinct, skip, debounce…)
* Usados para combinar observables
* Usados para crear nuevos observables

ReactiveX, esta basado en tres principios:

El Patrón Observer: Notifica cuando suceden cambios, la relación es de uno a muchos

El Patrón Iterador: Poder ejecutar operaciones de forma secuencial, como el paginado de una grilla que posee la función de primera pagian, siguiente, ultima pagina y pagina actual.

Programación Funcional: Tener funciones con tareas especificas qreciban argumentos y no muten la información

# Observables

Estamos a las puertas de la sección que nos enseñará sobre los observables, el primer eslabón de la programación reactiva, veremos temas como:

1. Observers

Emite la nueva información a sus subscriptores a través del método **next**, también puede informar a dichos subscriptores que va a dejar de emitir información a través del método **complete.**

*const* obs$ = new Observable<*string*>( *subs* *=>* {

    //Este mensaje se envia a los subscriptores

    subs.next('Hola');

    // de esta forma la emision no sera

    //notificada a los subscriptores

    subs.complete();

    //Esta notificacion no llegara a nadie

    subs.next('Hola');

});

obs$.subscribe( console.log ); // ES6

1. Subscriber
2. Unsubscribe
3. Subjects

/\*\*

 \* Caracteristicas del subject

 \* 1- Casteo multiple, va a distribuir la misma informacion a todos los lugares que estemos subscriptos

 \* 2- Tambien es un observer

 \* 3- Tambien se puede manejar el next, el error y el complete

 \*/

*const* subject$ = new Subject<*number*>();

1. Hot y Cold Observables
2. Observables en cadena

# Funciones para crear Observables

¡Es momento de utilizar funciones para crear observables!, esto nos ayudará a reducir mucho el tiempo a la hora de crear nuevos observables.

Los temas principales de la sección son:

* Of: Emite los valores en secuencia uno a uno de manera síncrona.
* fromEvent: Nos permite crear observables que provienen de cierto evento target del documento.

Para conocer los tipos de eventos que hay: se puede ver hacer un console log del evento y luego ver la información que se emite.

* Range:
* import { asyncScheduler, range } from 'rxjs';
* //El 1 es el valor de inicio, mientras
* // que el 10 implica 10 emisiones
* //NO significa que sigue el orden
* //const obsOf$ = range(1,10);
* /\*\*
* \* Para volver la ejecucion de forma asincrona
* \*/
* *const* obsOf$ = range(1,10, asyncScheduler);
* console.log('Inicio del obs')
* // el observable of se ejecuta de forma sincrona
* obsOf$.subscribe(*next=>*
* console.log('next', next),
* null,
* () *=>*console.log('Terminamos la secuencia')
* );
* console.log('Fin del obs')
* Interval:
* timer
* range: Crea una función que emite una secuencia dentro de un rango especifico.
* asyncScheduler

Mas adelante veremos otras funciones que crean observables, pero por ahora enfoquemos nuestra atención en las básicas y más comunes.

# Operadores

Este tema me encanta y es donde ReactiveX brilla, porque la fuerza de los operadores nos permiten jugar con la data y el flujo de información de una manera sorprendente.

Los temas principales de esta sección son:

1. Explicación de los operadores
2. Operadores como:
   1. map
   2. pluck
   3. mapTo
   4. filter
   5. tap
   6. reduce
   7. scan
3. Encadenamiento de operadores

Entre otros temas que verás poco a poco... estos son los operadores comunes y más usados...

# Referencias

<https://gist.github.com/Klerith/edf1218926a60fe79d9d2f20f83349d0>