RxJS (*Reactive Extensions for JavaScript*)

INTRODUCCIÓN

¿Qué es RxJS?

RxJS es una librería para la composición o creación de programas asíncronos basados en eventos, mediante el uso de secuencias observables.

Provee un **Observable** (tipo Core) y **Operators** inspirados en los Arrays para permitir el manejo de eventos y colecciones asíncronos.

ReactiveX combina el **Patrón Observable** con el **Patrón Iterador** y la programación funcional con colecciones para satisfacer la necesidad de una forma ideal de administrar las secuencias de eventos.

Conceptos esenciales en RxJS

- **Observable**: Representa la idea de una colección invocable de futuros valores o eventos.
- **Observer**: Es una colección de Callbacks que sabe cómo leer los valores enviados por el observable.
- **Subscription**: Representa la ejecución de un Observable, es útil para cancelar la ejecución.
- **Operators**: Son funciones puras que habilitan un estilo de programación funcional de tratamiento de Collections con operaciones como “map”,”filter”,”concat”,”reduce”… etc.
- **Subject**: Es el equivalente a un EventEmitter, y la única forma de multi-difundir un evento o múltiples Observers.
- **Schedulers**: Son despachadores centralizados para controlar la concurrencia, permitiéndonos la coordinación cuando ocurre el cálculo.

OBSERVABLE

Los Observable son colecciones perezosas de Push de múltiples valores.

	ÚNICO	MULTIPLE
Pull	Function	Iterator
Push	Promise	Observable

Pull versus Push

Pull y Push son dos protocolos diferentes que describen cómo un Productor de datos puede comunicarse con un Consumidor de datos.

¿Qué es Pull?

En los sistemas Pull, el Consumidor determina cuando recibe datos del Productor de datos. El productor no es consciente por sí mismo de cuando los datos serán enviados al Consumidor.

Toda función JavaScript es un sistema Pull. La función es un productor de datos, y el código que llama a la función la consume mediante “tirar” un único valor de su llamada.

	PRODUCTOR	CONSUMIDOR
Pull	Pasivo: Produce datos cuando son solicitados	Activo: Decide cuando los datos son solicitados
Push	Activo: Produce datos a su ritmo	Pasivo: Reacciona a los datos recibidos

¿Qué es Push?

En los sistemas Push, el Productor determina cuando mandar datos al consumidor. El consumidor no es consciente por sí mismo de cuando recibirá los datos.

Las Promises/promesas son los tipos mas comunes de sistemas Push hoy en día. Una Promise/promesa (el productor) envía un valor resuelto a los Callbacks registrados (los consumidores), pero a diferencia de las funciones, es la Promise la que tiene el poder de determinar precisamente cuando un valor es "Pushed" / "Empujado" a los Callbacks.

Observables como generalizaciones de funciones

Al contrario de la creencia popular, los Observables no son como los EventEmitters ni tampoco son como las Promesas para múltiples valores.

Los Observables pueden actuar como EventEmitters en algunos casos, es decir, cuando son difundidos de forma múltiple utilizando sujetos RxJS, pero generalmente no actúan como EventEmitters.

"Los observables son como funciones sin argumentos, pero generalícelos para permitir múltiples valores."

Anatomía de un Observable

Los observables son creados usando "new Observable" o un operador de creación, se suscriben con un Observer. Se ejecutan para entregar modificaciones "next", "error" y/o "complete" al Observer y su ejecución puede eliminarse.

Los siguientes 4 aspectos están todos codificados en una instancia de tipo Observable, pero algunos de estos aspectos están relacionados a otros tipos, como Observer y Subscription:

- Crear Observables
- Suscribirse a Observables
- Ejecutar el Observable
- Desechar Observables

Creando Observables

El constructor Observable toma un argumento: La función suscribe

El siguiente ejemplo crea un Observable para emitir el String “hi” cada segundo a un subscriber.

```
import { Observable } from ‘rxjs’;

const observable = new Observable(function subscribe(subscriber) {
   const id = setInterval(() => {
      subscriber.next(‘hi’)
   }, 1000);
});
```

Los observables pueden ser creados con “new Observable”. Mas comúnmente, los observables son creados usando funciones de creación como “of”, “from”, “interval” … etc.

Suscribirse a Observables

El Observable observable en el ejemplo puede ser suscrito de la siguiente forma:

```
Observable.subscribe(x => console.log(x));
```

No es una coincidencia que observable.subscribe y subscribe en “new Observable(function subscribe(subscriber) {…})” tienen el mismo nombre. En la librería son diferentes, pero para propósitos prácticos puedes considerarlos conceptualmente iguales.

Suscribirse a un Observable es como llamar a una función, proporcionando Callbacks donde se entregarán los datos.

Una llamada subscribe es una forma sencilla de empezar una “Observable execution” o Ejecución de Observable y mandar valores o eventos a un Observer de esa ejecución.

Ejecutar Observables

El código dentro de “new Observable(function subscribe(subscriber) {…}” representa una “Observable execution” o Ejecución de Observable, una computación perezosa que solo ocurre para cada Observer que se suscribe. La ejecución genera múltiples valores a lo largo del tiempo, ya sea de forma sincrónica o asincrónica.

Existen 3 tipos de valores que una Observable Execution puede enviar:

- Notificación “Next”: Manda un valor como un Number, un String, un Object,etc.
- Notificación Error:Manda: un error de JavaScript o una excepción.
- Notidicación “Complete”: No envía un valor.

Las notificaciones tipo “Next” son las más importantes y comunes: Representan los datos reales que se entregan a un subscriber.

Las notificaciones de “Error” y “Completo” pueden ocurrir solo una vez durante la Ejecución observable, y solo puede haber una de ellas.

OBSERVER

Un Observer es un consumidor de valores enviados por un Observable.

Los Observers son un simple set de Callbacks, uno por cada tipo de notificación enviada por el Observable:” next”

Un conserver es un consumer

SUSCRIPCIÓN

Una Suscripción es un objeto que representa un recurso disponible, normalmente la ejecución de un Observable.

Una Subscription tiene un método importante, “unsuscribe”, que no tiene argumentos y simplemente elimina el recurso que tiene la suscripción. En versiones anteriores de RxJS, la suscripción de llamaba

Una suscripción esencialmente simplemente tiene una función “unsuscribe()” para liberar recources o cancelar las ejecuciones “cancel”

Las suscripciones también pueden ser puestas juntas, de forma que una llamada a “unsuscribe()” de una suscripción