ReactiveX

Programación Reactiva

Arquitectura Front-End

ionic

¿Qué es la programación reactiva?

Programación orientada al manejo de streams de datos asíncronos y la propagación del cambio.

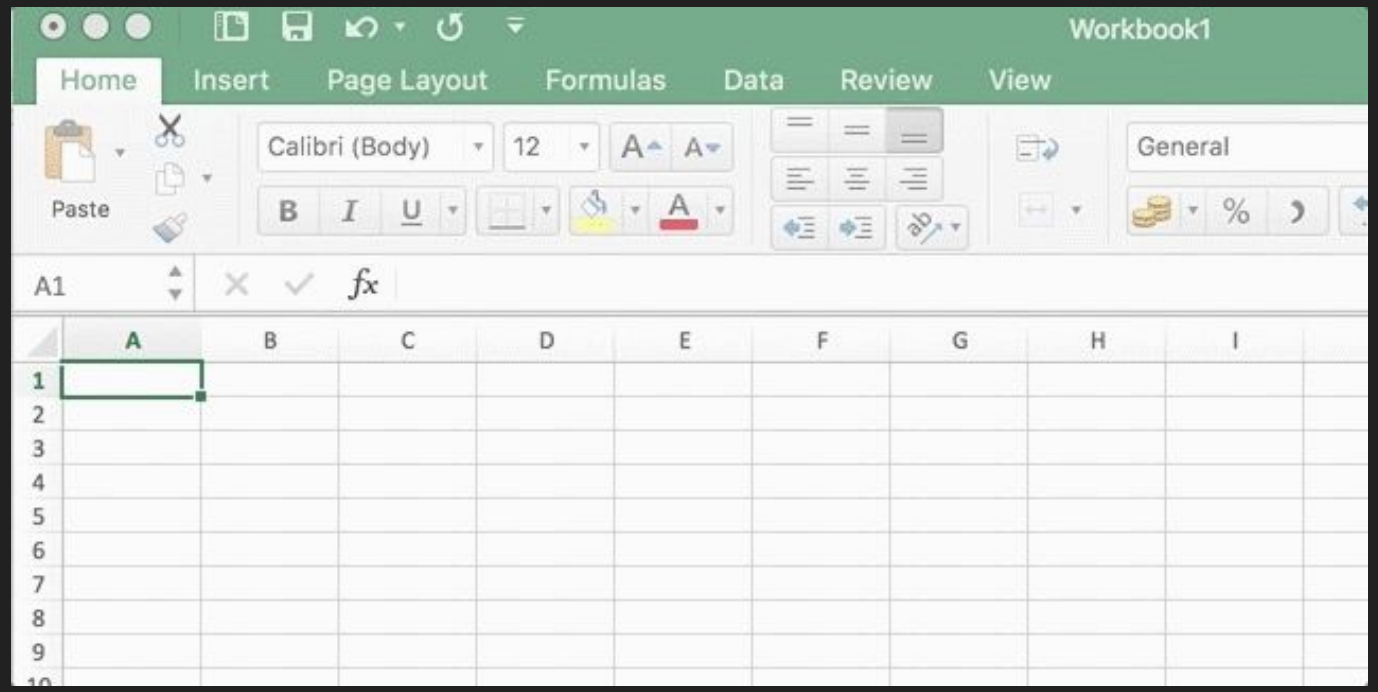
a := b + c

'a' se actualizaría automáticamente cada vez que cambien los valores de ' b' y 'c', sin que el programa ejecute nuevamente la sentencia 'b + c'.

Todo es un STREAM-> Eventos del ratón, Arrays, rangos de números, promesas, etc.

¿Qué es la programación reactiva?

Ejemplo típico de sistema o aplicación reactiva: ***HOJA DE CÁLCULO***



¿Qué es la programación reactiva?

TRABAJAR CON DATOS ASÍNCRONOS

Asincronía (o no sé cuándo llegará)

En nuestras aplicaciones, la gran mayoría de eventos que se producen son de naturaleza asíncrona, como un click de ratón, un envío de información desde backend o los caracteres que escribimos en un campo de búsqueda. Todas esas casuísticas tienen **tres característica**s en común:

- No sabemos con certeza **cuándo** se producirán (ni momento ni frecuencia).
- Queremos **reaccionar** ante ellos **cuando se produzcan**, bien sea operando con ellos o filtrándolos.
- Pueden **ocurrir** (clicks sucesivos) y **cambiar** (criterio de búsqueda) más de una vez a lo largo del tiempo.

Esos cambios y esa continuidad a lo largo del tiempo dan lugar a que sean "**observables**". Y precisamente, ese es el nombre de uno de los objetos más comúnmente utilizado en Rx.

¿Por qué necesitamos trabajar de manera asíncrona?

La respuesta simple es que queremos **mejorar la experiencia del usuario**. Queremos hacer que nuestra aplicación sea más responsiva, ofrecer una **experiencia fluida** a nuestros usuarios **sin congelar su hilo principal,** ralentizando, no queremos ofrecer aplicaciones con rendimiento pobre.

Para mantener el hilo principal libre **necesitamos hacer un montón de trabajo pesado y lento en segundo plano**. También queremos realizar trabajos pesados y cálculos complejos en nuestros servidores, ya que los dispositivos móviles no son tan potentes como para realizar trabajos pesados: no queremos capturar el dispositivo del usuario. Por tanto necesitamos trabajo asincrónico para las operaciones de red.

RxJS

reactivex.io
rxjs-dev.firebaseapp.com

Una API para la programación asíncrona con flujos observables

En RxJs los streams están representados por Observables, por lo que en RxJs todo, absolutamente todo es un Observable, lo que lógicamente nos lleva al patrón Observer.

```
//Una letra es un stream
const letterStream$ = Rx.Observable.of('A');
// Un rango de números es un stream
const rangeStream$ = Rx.Observable.range(1,8);
// Los valores de un Array son un stream
const arrayStream$ = Rx.Observable.from([1,2,3,4]);
// Valores emitidos cada 100 ms son un stream
const intervalStream$ = Rx.Observable.interval(100);
// Cualquier tipo de evento del ratón es un stream
const clicksStream$ = Rx.Observable.fromEvent(document, 'click');
// La respuesta a un servicio basada en una promesa es un stream
const promiseStream$ = Rx.Observable.fromPromise(fetch('/products'));
```

Observable: Representa el flujo de datos.

Observer: Módulos o piezas de software que los consumen.

Subscription: La "escucha" del stream, representa la **ejecución de un Observable**

Operators: Son funciones puras que permiten un estilo de "programación funcional" de tratar con colecciones con operaciones como mapa, filtro, concat, flatMap, etc.

RxJS

reactivex.io
rxjs-dev.firebaseapp.com

Una API para la programación asíncrona con flujos observables

```
flujoDatos
    .Where(d => d > 0)
    .Select(dt => dt / dt - 15)
    .Subscribe
    (
        onNext: d => Console.WriteLine(d),
        onError: e => Console.Write("Oops :" + e.Message),
        onCompleted: ()=> Console.WriteLine("Terminado")
    );
```

flujo de datos (observable)

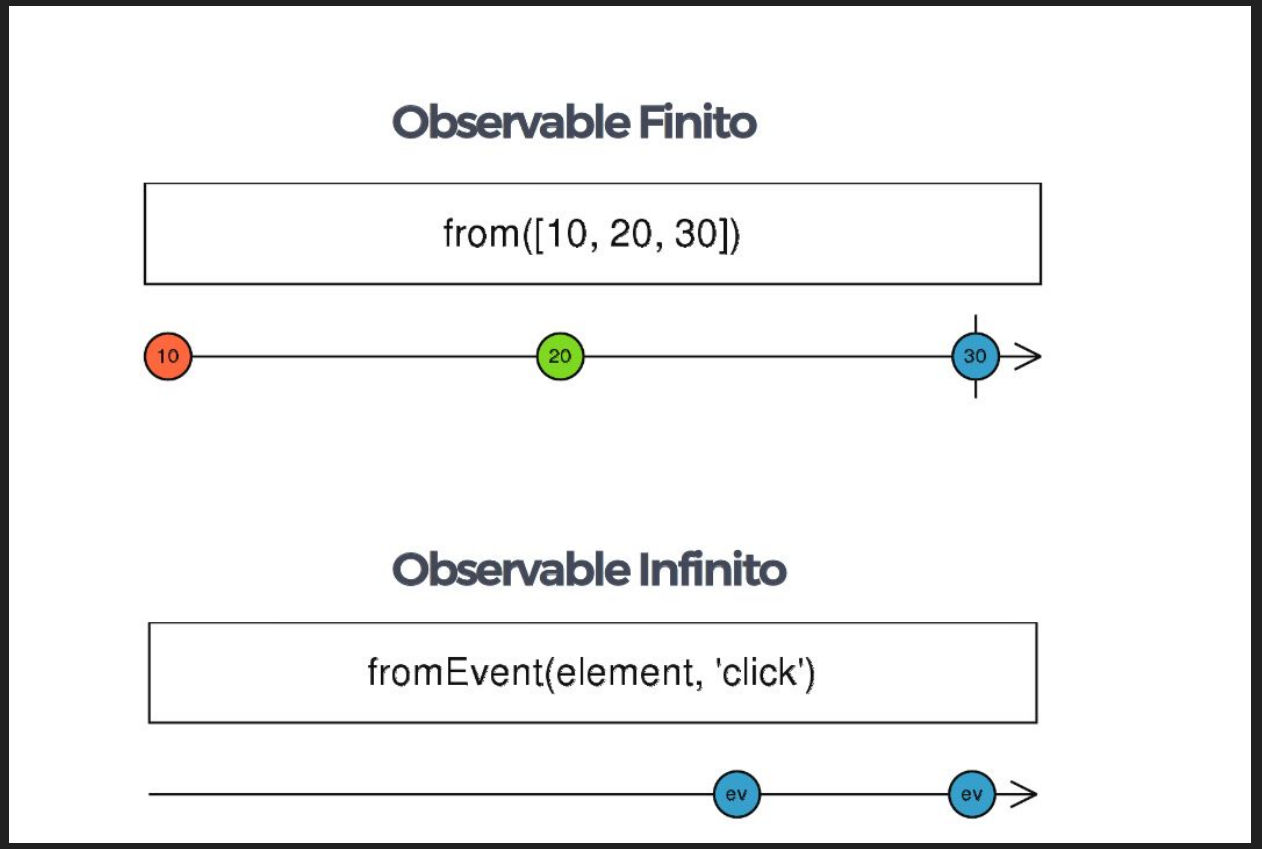
filtro

transformación

operación de suscripción

suscriptor (observador)

RxJS


Observable Finito
from([10, 20, 30])
10
20
30
Observable Infinito
fromEvent(element, 'click')
ev
ev

RxJS

Ejemplo Básico de creación

```
import { Observable } from 'rxjs';

const observable = new Observable(subscriber => {
  subscriber.next(1);
  subscriber.next(2);
  subscriber.next(3);
  setTimeout(() => {
    subscriber.next(4);
    subscriber.complete();
  }, 1000);
});

console.log('Antes del subscribe');
observable.subscribe({
  next(x) { console.log('Valor: ' + x); },
  error(err) { console.error('Ha ocurrido un error: ' + err); },
  complete() { console.log('Completado'); }
});
console.log('Después del  subscribe');
```

```
Antes del subscribe
Valor: 1
Valor: 2
Valor: 3
Después del subscribe
Valor: 4
Completado
```

RxJS

Por qué ?

- ✓ Es una herramienta poderosa que convierte una serie de acciones complicadas en un código conciso que es fácil de manipular.
- ✓ Capacidad de manejar llamadas asíncronas con múltiples eventos .
- ✓ Los observables se pueden cancelar al contrario que las promesas.
- ✓ Nos permitirá un mejor control de flujos.
- ✓ Dispone de una increíble toolbox de funciones para ***combinar, crear y filtrar cualquiera de esos streams.*** Ahí es donde entra en acción la magia. -> https://www.learnrxjs.io/

RxJS

Por qué ?

```
updateUserWithPromise(user, token) {
    let url = URL_SERVICIOS + "/users";
    let headers = new Headers();
    headers.append('Content-Type', 'application/json');
    headers.append('Authorization', 'Bearer ' + token);
    let options = new RequestOptions({headers: headers});

    return new Promise((resolve, reject) => {
      this.http.put(url, JSON.stringify(user), options).subscribe((response: any) => {
        resolve(response);
      }, (error: any) => {
        console.log("ERROR updateUserIndividual" + error.status);
        reject(error);
      })
    })
  }
```

```
updateUserIndividualObs(user): Observable<any> {
    let url = URL_SERVICIOS + "/users";
    return this.http2.put(url, user)
}
```

```
this._us.updateUserIndividualObs(birth).subscribe(data => {
   this.closePage();
 }, error => {
   this.presentToast("Ups! Ha ocurrido un error, inténtalo de nuevo");
 })
```

RxJS

el verdadero poder de RXJS: los operadores

```
const button = document.querySelector('button');

// JS PURO
button.addEventListener('click', e => {
  output.textContent = Math.random().toString(36).slice(2);
});

//Observable
Rx.Observable
    .fromEvent(button, 'click')
    .subscribe(() => {
        output.textContent = Math.random().toString(36).slice(2);
    });

//Cada tercer click genere una cadena aleatoria:

Rx.Observable
  .fromEvent(button, 'click')
  .bufferCount(3) // <--------- only added this line!
  .subscribe(() => {
    output.textContent = Math.random().toString(36).slice(2);
  });
```

```
const output = document.querySelector('output');
const button = document.querySelector('button');

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

click$
.bufferWhen(() => click$.delay(400))
.filter(events => events.length >= 3)
.subscribe((res) => {
    output.textContent =  Math.random().toString(36).slice(2);
});
```

RxJS

el verdadero poder de RXJS: los operadores

```
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
```

RxJS

el verdadero poder de RXJS: los operadores

```
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
```

RxJS

Cuándo?

- ✓ Si una acción desencadena múltiples eventos.
- ✓ Si se tiene mucha asincronía y se está intentando crear un flujo en conjunto.
- ✓ Si se encuentra en situaciones en las que desea actualizar algo de forma reactiva.
- ✓ Si se están manejando un enorme conjunto de datos y éstos necesitan procesarse en distintos pasos, los operadores de RXJS pueden procesar esto sin necesidad de crear acciones intermedias.

RxJS

Buenas Prácticas

✓ No usar múltiples Observable.subscribes en una expresión.

```
initialize () {
  this.appParameters.subscribe (params => {
    const id = params ['id'];
    if (id! == null && id! == undefined) {
      this.getUser (id) .subscribe (usuario = > this.user = user);
    }
  });
}
```

✓ Siempre devolver un observable

```
initialize () {
  this.appParameters.map (params => params ['id'])
    .switchMap (id => id? this.getUser (id): Observable.empty ())
    .subscribe (usuario => this.user = usuario);
}
```

✓ No olvidar el unsubscribe del observable en ngDestroy(Angular) or ionViewWillLeave(Ionic)

RxJS

ForkJoin

bluulink

```
callObservablesToPrintMap() {
    this.loading = this.loadingCtrl.create({
      content: 'Cargando localizaciones ... '
    });
    this.loading.present();

    let obs1 = this._sp.getLocationsByPosition(this.lat, this.lng);
    let obs2 = this._sp.getUserFavoriteLocations();

    this.subscription = forkJoin([obs1, obs2]).subscribe(results => {
      this.printMap(results);
      this.loading.dismissAll();
    }, error => {
      console.log("ERROR" + error);
      this.loading.dismissAll();
    });
  }
```

```
getLocationsByPosition(lat, lng): Observable<any> {
    let url = URL_SERCAE + "/instances/locations/" + lat + "/" + lng;
    return this.http
      .get(url)
      .map(res => {
        return res['result'];
      }).catch(error => {
        if (error.status === 404)
          return Observable.of([]);
      })
  }
```

RxJS

bluulink

FlatMap

```
createOfferWithProffesions(offer, addProfessions: AddProfesions): Observable<any> {
   let urlSetProf = URL_SERVICIOS + '/offers/professions';
   let urlCreateOffer = URL_SERVICIOS + '/offers';
   return this.http
     .post(urlCreateOffer, offer)
     .map(res => {
       return res['result']
     }).flatMap((offer: Offer) => {
       addProfessions.id = offer.id;
       return this.http
         .post(urlSetProf, addProfessions)
     })
 }
```

```
this._op.createOfferWithProffesions(offerJob, addProfessions).subscribe(res => {
  this.navCtrl.pop();
}, error => console.log(error));
```

RxJS

bluulink

Manejo de Flujos I

```
verifyFacebookUser(): Observable<User> {
  let userFacebook = {"type": "facebook", "id_facebook": this.userId};

  return this._us.verifyRegisterFacebookObs(userFacebook)
    .flatMap(token => {
      if (!token) {
        return this._us.facebookRegisterObs({"id_facebook": this.userId}).flatMap(user => {
          return Observable.fromPromise(this.storage.set('token', user.token));
        })
      }
      else {
        return Observable.fromPromise(this.storage.set('token', token));
      }
    }, error => {
      error => console.log("FIRST ERROR" + JSON.stringify(error));
    })
    .flatMap(res => {
      return Observable.fromPromise(this.storage.get('token'));
    })
    .flatMap(tokenStorage => {
      let decodedToken = this.jwtHelper.decodeToken(JSON.stringify(tokenStorage));
      let idUser = parseInt(decodedToken.id);
      return this._ups.getUserProfileObs(idUser);
    })
}
```

```
this.subscription = this.verifyFacebookUser()
          .subscribe((user: User) => {
            this.processDataObs(user);
            loading.dismissAll();
          }, error => {
            console.log("Error on flow ---" + error);
            loading.dismissAll();
            this.presentToast("Ups ... ha ocurrido un error con facebook");
          })
```

RxJS

bluulink

Manejo de Flujos II – Async Await

```
async tryGeolocation() {
  try {
    if (await this.diagnostic.isLocationAuthorized()) {
      if (await this.diagnostic.isLocationEnabled()) {
        this.loading = this.loadingCtrl.create({
          content: 'Localizando ofertas ... '
        });
        this.loading.present();
        const {coords} = await this.geolocation.getCurrentPosition();
        this.lat = coords.latitude;
        this.lng = coords.longitude;
        let token = await this.storage.get('token');
        let getOffers = await this.getOffersPromise(this.lat, this.lng, token);
        if (getOffers['status'] === "OK") {
          let offers: Offer[] = getOffers['result'];
          this.availableOffers = offers;
          this.currentPosition = 2;
          this.stackOffers = this.availableOffers.slice(0, this.currentPosition);
        }
        this.loading.dismiss().then(r => {
          this.showText = true;
        });
      } else {
        this.activateGPS(true);
      }
    } else {
      await this.diagnostic.requestRuntimePermission(this.diagnostic.permission.ACCESS_FINE_LOCATION);
      this.tryGeolocation();
    }
  } catch (e) {
    this.loading.dismiss();
    if (e.status == 403) {
      this.refreshToken();
    } else if (e.status == 404) {
      this.showText = true;
      this.presentToast("Lo siento, no hemos encontrado oportunidades cerca de ti");
    } else {
      this.presentToast("Upss! Ha ocurrido un error.Inténtalo de nuevo");
    }
  }
}
```

```
this.subscription = this.diagnosticGPS()
    .flatMap(results => this.checkGPS(results))
    .flatMap((coords: Geoposition) => this.getOffersByLocation(coords))
    .subscribe( next: (offers: Offer[]) => {
      this.processOffers(offers);
      this.loading.dismissAll();
    }, error: error => {
      console.log("ERROR---------->" + JSON.stringify(error));
      this.navCtrl.pop().then( onfulfilled: () => {
        this.presentToast('Ups, ha ocurrido un error.Inténtalo en breve.');
      })
    })
```

RxJS

bluulink

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

Bibliografía

- https://medium.com/@mohandere/rxjs-5-in-5-minutes-1c3b4ed0d8cc
- https://news.thisdot.co/what-is-rxjs-and-why-you-should-know-about-it-2a5afe58cea
- https://gist.github.com/btroncone/d6cf141d6f2c00dc6b35
- https://x-team.com/blog/rxjs-observables/
- https://blog.danieleghidoli.it/2016/10/22/http-rxjs-observables-angular/
- https://coryrylan.com/blog/subscribing-to-multiple-observables-in-angular-components
- https://alligator.io/rxjs/simple-error-handling/
- https://blog.strongbrew.io/rxjs-best-practices-in-angular/
- https://www.adictosaltrabajo.com/2017/11/14/programacion-reactiva-uso-de-la-libreria-rxjs/

GRACIAS!

GitHub https://github.com/jossephalvarez

Josseph Alvarez Villa

jossephalvarez@Gmail.com