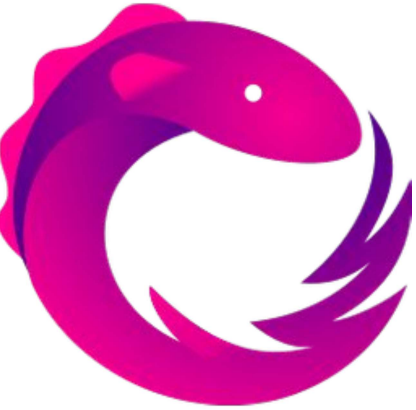
FATMA TANRISEVDİ

Lead Frontend Developer

armut.com

@tnrsvd

fatma.tanrisevdi@gmail.com



Reactive Programlama?

Adım adım kod akışını takip etmek yerine olayları(events) takip eden bir programlama paradigmasıdır.

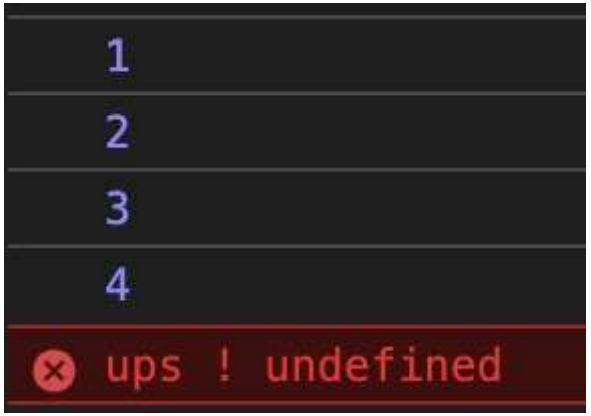
RxJS?

Olay ve veri kaynaklarını abone olunabilen(subscribable) nesnelere(observable) dönüştürüp, bunlar üzerinde operatörler yardımıyla değişimler yapabilmemizi ve bu sonucu tüketebilmemizi sağlayan Javascript'le yazılmış bir reactive programlama kütüphanesidir.

Temel Kavramlar

Observable(gözlemlenen), Observer(gözlemci), Subscription(abonelik)

```
const observable$: Observable<number> = new Observable<number>
(data => {
  data.next(1);
  data.next(2);
  data.next(3);
  data.next(4);
  data.error(); // bu noktada akis bitiyor.
  data.complete();
});
const observer: Observer<number> = {
  next: item => console.log(item),
  error: error => console.error("ups ! " + error),
  complete: () => console.log("done!")
};
const subscription = observable$.subscribe(observer);
```



Temel Kavramlar

Subject: Hem Observable gibi abone olunabilen, hem de Observer'daki tüm metotları (next, error, complete) barındıran bir sınıf.

Her Subject bir observabledır.

Birden fazla observera yayın yapabılır(multicast).

İstenen anda yeni bir değerin yayınını yapabilir.

```
employee 1 gets : good morning everyone!
employee 2 gets : good morning everyone!
employee 1 gets : have a wonderful day
employee 2 gets : have a wonderful day
employee 1 gets : work work work!
employee 2 gets : work work work!
```

```
const employer$: Subject<string> = new Subject<string>();

employer$.subscribe(notification => {
  console.log(`employee 1 gets : ${notification}`);
});
employer$.subscribe(notification => {
  console.log(`employee 2 gets : ${notification}`);
});
employer$.next("good morning everyone!");
employer$.next("have a wonderful day");
employer$.next("work work work!");
employer$.subscribe(notification => {
  console.log(`employee 3 gets : ${notification}`);
});
```

Temel Kavramlar

Subject

BehaviorSubject: Bir Subject varyasyonudur.

Bir başlangıç değeri atanmalıdır.

Yayınlanan son değeri tutar ve subsciption sırasında verir.

```
const employer$: BehaviorSubject<string> = new
BehaviorSubject<string>('May I have your attention please?');

employer$.subscribe(notification => {
  console.log(`employee 1 gets : ${notification}`);
});
employer$.subscribe(notification => {
  console.log(`employee 2 gets : ${notification}`);
});
employer$.next("good morning everyone!");
employer$.next("have a wonderful day");
employer$.next("work work work!");
employer$.subscribe(notification => {
  console.log(`employee 3 gets : ${notification}`);
});
```

```
employee 1 gets : May I have your attention please?
employee 2 gets : May I have your attention please?
employee 1 gets : good morning everyone!
employee 2 gets : good morning everyone!
employee 1 gets : have a wonderful day
employee 2 gets : have a wonderful day
employee 1 gets : work work work!
employee 2 gets : work work work!
employee 3 gets : work work work!
```

Temel Kavramlar

Subject

ReplaySubject: Bir Subject varyasyonudur.

Tanımlarken belirtilen buffer size kadar veri tutabilir.

Yeni bir subscription da tüm buffersize da tutulan tüm veriler aktarılır.

```
employee 1 gets : good morning everyone!
employee 2 gets : good morning everyone!
employee 1 gets : have a wonderful day
employee 2 gets : have a wonderful day
employee 1 gets : work work work!
employee 2 gets : work work work!
employee 3 gets : have a wonderful day
employee 3 gets : work work work!
```

```
const employer$: ReplaySubject<string> = new
ReplaySubject<string>(2);

employer$.subscribe(notification => {
  console.log(`employee 1 gets : ${notification}`);
});
employer$.subscribe(notification => {
  console.log(`employee 2 gets : ${notification}`);
});
employer$.next("good morning everyone!");
employer$.next("have a wonderful day");
employer$.next("work work work!");
employer$.subscribe(notification => {
  console.log(`employee 3 gets : ${notification}`);
});
```

Temel Kavramlar

Subject

AsyncSubject: Bir Subject varyasyonudur.

Buffersizeı 1 olan ve tuttuğu değeri sadece complete durumunda yayınlayan bir ReplaySubject gibi davranır.

```
const employer$: AsyncSubject<string> = new
AsyncSubject<string>();

employer$.subscribe(notification => {
  console.log(`employee 1 gets : ${notification}`);
});
employer$.subscribe(notification => {
  console.log(`employee 2 gets : ${notification}`);
});
employer$.next("good morning everyone!");
employer$.next("have a wonderful day");
employer$.next("work work work!");
employer$.subscribe(notification => {
  console.log(`employee 3 gets : ${notification}`);
});
employer$.complete();
```

```
employee 1 gets : work work work!
employee 2 gets : work work work!
employee 3 gets : work work work!
```

Temel Kavramlar

Operators

Pipeable Operators: Bir veri akışından söz ettik. Iste bu akışta bir boru(pipe) ile araya girip istediğimiz değişikleri operatörler sayesinde yapabiliriz. Bu operatörler kaynakta bir değişikliğe neden olmazlar onun yerine yeni bir observable dönerler.

```
const observable$: Observable<number> = new Observable<number>(data => {
  data.next(1);
  data.next(2);
  data.next(3);
  data.next(4);
  data.next(5);
  data.next(6);
  data.complete();
});
observable$.pipe(filter(item => item % 2 === 0)).subscribe(result => {
  console.log(result);
});
```

```
2
4
6
```

Temel Kavramlar

Operators

Pipeable Operators:

Birer fonksiyondur.

Tek tek sayamayacağım kadar çok operator mevcut.

```
33
[]
2
37
▶ ["visited number is 1", "visited number is 2"]
4
37
▶ ["visited number is 1", "visited number is 2", "visited number is 3", "visited number is 4"]
6
37
▶ ["visited number is 1", "visited number is 2", "visited number is 3", "visited number is 4", "visited number is 5", "visited number is 6"]
```

```
const visitedNumbers: String[] = [];
const observable$: Observable<number> = new Observable<number>(data => {
  data.next(1);
  data.next(2);
  data.next(3);
  data.next(4);
  data.next(5);
  data.next(6);
  data.complete();
});
console.log(new Date().getSeconds());
observable$
  .pipe(
    delay(4000),
    tap(item => {
      visitedNumbers.push(`visited number is ${item}`);
    }),
    filter(item => item % 2 === 0)
  )
  .subscribe(result => {
    console.log(result);
    console.log(new Date().getSeconds());
    console.log(visitedNumbers);
  });

console.log(visitedNumbers);
```

Temel Kavramlar

Operators

Creation Operators: Yeni bir observable yaratmak için var olan foksiyonlardır.

of,

```
of(1, 2, 3);
```

from,

```
from([1, 2, 3]);
```

fromEvent

```
const clicks$: Observable<Event> =
```

```
fromEvent(document, 'click');
```

```
clicks$.subscribe((x) => console.log(x));
```

Creation Operators

- ajax
- bindCallback
- bindNodeCallback
- defer
- empty
- from
- fromEvent
- fromEventPattern
- generate
- interval
- of
- range
- throwError
- timer
- iif

Transformation Operators

- buffer
- bufferCount
- bufferTime
- bufferToggle
- bufferWhen
- concatMap
- concatMapTo
- exhaust
- exhaustMap
- expand
- groupBy
- map
- mapTo
- mergeMap
- mergeMapTo
- mergeScan
- pairwise
- partition
- pluck
- scan
- switchScan
- switchMap

Filtering Operators

- audit
- auditTime
- debounce
- debounceTime
- distinct
- distinctUntilChanged
- distinctUntilKeyChanged
- elementAt
- filter
- first
- ignoreElements
- last
- sample
- sampleTime
- single
- skip
- skipLast
- skipUntil
- skipWhile
- take
- takeLast
- takeUntil
- takeWhile

Utility Operators

- tap
- delay
- delayWhen
- dematerialize
- materialize
- observeOn
- subscribeOn
- timeInterval
- timestamp
- timeout
- timeoutWith
- toArray

Conditional&Boolean Operators

- defaultIfEmpty
- every
- find
- findIndex
- isEmpty

Angular ve RxJS

HttpClient,
Reactive Forms,
Router

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';
You, seconds ago | 1 author (You)
@Injectable({
  providedIn: 'root',
})
export class SnowflakeService {

  constructor(protected httpClient: HttpClient) {}
  public clearList(): Observable<boolean> {
    const path = `service/clear-list`;

    return this.httpClient.get<boolean>(path);
  }
}
```

NGRX
(Reactive State for Angular)

```
loadServiceRelatives$: Observable<Action> = this.actions$.pipe(
  ofType<actions.LoadServiceRelatives>(actions.ServiceRelationsActionType.LOAD_SERVICE_RELATIVES),
  withLatestFrom(this.store.select(fromTrends.getRelatives)),
  switchMap(([action, serviceRelativesFromStore]) => {
    const useFromStore = !!serviceRelativesFromStore;
    return iif(
      () => useFromStore,
      of(new actions.UseServiceRelativesFromStore()),
      this.serviceMasterApi.getRelatives(action.payload).pipe(
        map((data: Array<ServiceResult>) => new actions.LoadServiceRelativesSuccess(data)),
        catchError((error) => of(new actions.LoadServiceRelativesFail(error)))
      )
    );
  })
);
```

Angular ve RxJS AsyncPipe

```
import { HttpClient } from "@angular/common/http";
import { ChangeDetectionStrategy, Component, OnInit } from "@angular/core";
import { Observable, Subject } from "rxjs";

@Component({
  selector: "my-app",
  template: `
    <ul *ngIf="(cats$ | async) as cats">
      <li *ngFor="let cat of cats">
        {{ cat.text }}
      </li>
    </ul>
  `,
  styleUrls: ["./app.component.css"],
  changeDetection: ChangeDetectionStrategy.OnPush ///
})
export class AppComponent implements OnInit {
  value$ = new Subject();
  cats$: Observable<Array<any>>;

  constructor(private httpClient: HttpClient) {}

  ngOnInit(): void {
    this.cats$ = this.httpClient.get<Array<any>>(
      "https://cat-fact.herokuapp.com/facts"
    );
  }
}
```

ChangeDetectionStrategy:
componentin değişim tesbit etme döngüsünü(change detection cycle) sınırlayarak performansı yükseltir.

OnPush

- **Input variabledaki,**
- **bir event yayını(emit) olduğunda**
- **html e bağlanmış(bind) bir observable yayını(emit) olduğunda**

Angular ve RxJS

UnSubscription

Eğer bir subscription ı unsubscribe etmezsek, kocaman hatalı bir bellek sızıntısı(**memory leak**) olur.

```
@Component({
  selector: "my-app",
  template: `
    <p></p>
  `,
  styleUrls: ["./app.component.css"]
})
export class AppComponent implements OnInit, OnDestroy {
  subscription: Subscription;
  constructor(private router: Router) {}
  ngOnInit(): void {
    this.subscription = this.router.events
      .pipe(
        filter(
          (event: RouterEvent): boolean => {
            return event instanceof NavigationEnd;
          }
        )
      )
      .subscribe((event: NavigationEnd) => {
        //do something
      });
  }
  ngOnDestroy(): void {
    this.subscription.unsubscribe();
  }
}
```

Angular ve RxJS UnSubscription

2- take operatörleri

```
@Component({
  selector: "my-app",
  template: `
    <p></p>
  `,
  styleUrls: ["./app.component.css"]
})
export class AppComponent implements OnInit, OnDestroy {
  ngUnsubscribe = new Subject<void>();
  constructor(private router: Router) {}
  ngOnInit(): void {
    this.router.events
      .pipe(
        filter(
          (event: RouterEvent): boolean => {
            return event instanceof NavigationEnd;
          }
        ),
        takeUntil(this.ngUnsubscribe)
      )
      .subscribe((event: NavigationEnd) => {
        //do something
      });
  }
  ngOnDestroy(): void {
    this.ngUnsubscribe.next();
  }
}
```

3- until-destroy -

Netanel Basal'ın [kütüphanesi]

Angular ve RxJS UnSubscription

4- ng-observe - ngTurkey ekibinden 4 arkadaşımızın yazmış olduğu kütüphane

Kaynaklar

- https://rxjs-dev.firebaseapp.com/
- https://academy.esveo.com/en/blog/nY/
- I switched a map and you'll never guess what happened next | Pete Darwin, Shai Reznik, Mike Brocchi
- Mastering the Subject: Communication Options in RxJS | Dan Wahlin
- Data Composition with RxJS | Deborah Kurata
- Use the Custom Operator Force; Become an RxJS Jedi | Ryan Chenkie
- https://medium.com/angular-in-depth/the-best-way-to-unsubscribe-rxjs-observable-in-the-angular-applications-d8f9aa42f6a0
- https://blog.bitsrc.io/6-ways-to-unsubscribe-from-observables-in-angular-ab912819a78f
- https://www.armanozak.com/rxjs-ile-reaktif-programlamaya-giris/