RxJS-Introduction

rxjs.dev/guide/overview

Introduction

RxJS est une bibliothèque pour composer des programmes asynchrones et basés sur des événements en utilisant des séquences observables. Il fournit un type de base, Observable , des types satellites (Observer, Schedulers, Subjects) et des opérateurs inspirés des `Array`méthodes (`map`, `filter`, `reduce`, `every`, etc.) pour permettre de gérer les événements asynchrones en tant que collections.

Considérez RxJS comme Lodash pour les événements.

ReactiveX combine le modèle Observer avec le modèle Iterator et la programmation fonctionnelle avec des collections pour répondre au besoin d'un moyen idéal de gérer des séquences d'événements.

Les concepts essentiels de RxJS qui résolvent la gestion des événements asynchrones sont :

- **Observable :** représente l'idée d'une collection invocable de valeurs ou d'événements futurs.
- **Observer :** est une collection de callbacks qui sait écouter les valeurs délivrées par l'Observable.
- **Abonnement :** représente l'exécution d'un Observable, est principalement utile pour annuler l'exécution.
- **Opérateurs :** sont des fonctions pures qui permettent un style de programmation fonctionnel pour traiter des collections avec des opérations telles que `map`, `filter`, `concat`, `reduce`, etc.
- **Subject :** équivaut à un EventEmitter et constitue le seul moyen de multidiffuser une valeur ou un événement à plusieurs observateurs.
- **Planificateurs :** sont des répartiteurs centralisés pour contrôler la simultanéité, nous permettant de coordonner le moment où le calcul se produit par exemple sur `setTimeout`ou `requestAnimationFrame`ou sur d'autres.

Premiers exemples

Normalement, vous enregistrez des écouteurs d'événement.

```
document.addEventListener('click', () => console.log('Clicked!'));
```

En utilisant RxJS, vous créez un observable à la place.

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

fromEvent(document, 'click').subscribe(() => console.log('Clicked!'));
```

Pureté

Ce qui rend RxJS puissant, c'est sa capacité à produire des valeurs à l'aide de fonctions pures. Cela signifie que votre code est moins sujet aux erreurs.

Normalement, vous créeriez une fonction impure, où d'autres éléments de votre code peuvent gâcher votre état.

```
let count = 0;
document.addEventListener('click', () => console.log(`Clicked ${++count} times`));
```

En utilisant RxJS, vous isolez l'état.

```
import { fromEvent, scan } from 'rxjs';

fromEvent(document, 'click')
  .pipe(scan((count) => count + 1, 0))
  .subscribe((count) => console.log(`Clicked ${count} times`));
```

L' opérateur **scan** fonctionne exactement comme **reduce** pour les tableaux. Il prend une valeur qui est exposée à un rappel. La valeur renvoyée du rappel deviendra alors la prochaine valeur exposée lors de la prochaine exécution du rappel.

Couler*lien*

RxJS dispose de toute une gamme d'opérateurs qui vous aident à contrôler la façon dont les événements se déroulent dans vos observables.

Voici comment vous autoriseriez au plus un clic par seconde, avec du JavaScript simple :

```
      let count = 0;
let rate = 1000;
let lastClick = Date.now() - rate;
document.addEventListener('click', () => {
  if (Date.now() - lastClick >= rate) {
    console.log(`Clicked ${++count} times`);
    lastClick = Date.now();
  }
});
```

Avec RxJS :

```
      import { fromEvent, throttleTime, scan } from 'rxjs';

fromEvent(document, 'click')
  .pipe(
    throttleTime(1000),
    scan((count) => count + 1, 0)
  )
  .subscribe((count) => console.log(`Clicked ${count} times`));
```

Les autres opérateurs de contrôle de flux sont **filter** , **delay** , **debounceTime** , **take** , **takeUntil** , **distinct** , **distinctUntilChanged** etc.

Valeurs

Vous pouvez transformer les valeurs transmises par vos observables.

Voici comment ajouter la position x actuelle de la souris pour chaque clic, en JavaScript :

```
      let count = 0;
const rate = 1000;
let lastClick = Date.now() - rate;
document.addEventListener('click', (event) => {
  if (Date.now() - lastClick >= rate) {
    count += event.clientX;
    console.log(count);
    lastClick = Date.now();
  }
});
```

Avec RxJS :

```
      import { fromEvent, throttleTime, map, scan } from 'rxjs';

fromEvent(document, 'click')
  .pipe(
    throttleTime(1000),
    map((event) => event.clientX),
    scan((count, clientX) => count + clientX, 0)
  )
  .subscribe((count) => console.log(count));
```

Les autres opérateurs générateurs de valeur sont **plumer** , **par paires** , **échantillonner** etc.