Angular

Qinshift Academy

RXJS

Qinshift Academy

REACTIVE EXTENSIONS LIBRARY FOR JAVASCRIPT

RxJS is a library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code. This project is a rewrite of Reactive-Extensions/RxJS with better performance, better modularity, better debuggable call stacks, while staying mostly backwards compatible, with some breaking changes that reduce the API surface

Introduction

RxJS is a library for composing asynchronous and event-based programs by using observable sequences. It provides one core type, the Observable, satellite types (Observer, Schedulers, Subjects) and operators inspired by Array methods (map, filter, reduce, every, etc) to allow handling asynchronous events as collections.

Think of RxJS as Lodash for events.

ReactiveX combines the Observer pattern with the Iterator pattern and functional programming with collections to fill the need for an ideal way of managing sequences of events.

The essential concepts in RxJS which solve async event management are:

- **Observable**: represents the idea of an invokable collection of future values or events.
- **Observer**: is a collection of callbacks that knows how to listen to values delivered by the Observable.
- **Subscription**: represents the execution of an Observable, is primarily useful for cancelling the execution.
- **Operators**: are pure functions that enable a functional programming style of dealing with collections with operations like map, filter, concat, reduce, etc.
- **Subject**: is equivalent to an EventEmitter, and the only way of multicasting a value or event to multiple Observers.
- **Schedulers**: are centralized dispatchers to control concurrency, allowing us to coordinate when computation happens on e.g. setTimeout or requestAnimationFrame or others.

First examples

Normally you register event listeners.

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

Using RxJS you create an observable instead.

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

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

Purity

What makes RxJS powerful is its ability to produce values using pure functions. That means your code is less prone to errors.

Normally you would create an impure function, where other pieces of your code can mess up your state.

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

Using RxJS you isolate the state.

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

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

The scan operator works just like reduce for arrays. It takes a value which is exposed to a callback. The returned value of the callback will then become the next value exposed the next time the callback runs.

Flow

RxJS has a whole range of operators that helps you control how the events flow through your observables.
This is how you would allow at most one click per second, with plain JavaScript:

```
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();
  }
});
```

With 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`));
```

Other flow control operators are filter, delay, debounceTime, take, takeUntil, distinct, distinctUntilChanged etc.

Values

You can transform the values passed through your observables.
Here's how you can add the current mouse x position for every click, in plain 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();
  }
});
```

With 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));
```

Other value producing operators are pluck, pairwise, sample etc.

Questions?

Trainer Name

Trainer

trainer@mail.com

Assistant Name

Assistant

assistant@mail.com

Qinshift Academy