RxJS Operators: Complete Guide with Examples and Explanations

RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using Observables. It provides powerful operators to handle asynchronous events and data streams.

---

1. `of`

Creates an Observable that emits the arguments and completes.

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

of(1, 2, 3).subscribe(console.log); // Output: 1 2 3
```

2. `from`

Converts various other objects and data types into Observables.

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

from([10, 20, 30]).subscribe(console.log); // Output: 10 20 30
```

3. `map`

Applies a projection function to each value.

```
import { of } from 'rxjs';
import { map } from 'rxjs/operators';

of(1, 2, 3).pipe(
  map(x => x * 10)
).subscribe(console.log); // Output: 10 20 30
```

4. `filter`

Filters items emitted by the source Observable.

```
import { of } from 'rxjs';
import { filter } from 'rxjs/operators';

of(1, 2, 3, 4, 5).pipe(
  filter(x => x % 2 === 0)
).subscribe(console.log); // Output: 2 4
```

5. `tap`

Perform side effects for notifications from the source.

```
import { of } from 'rxjs';
import { tap } from 'rxjs/operators';

of(1, 2, 3).pipe(
  tap(x => console.log(`Before map: ${x}`)),
  map(x => x * 2)
).subscribe(console.log);
```

6. `mergeMap`

Projects each source value to an Observable, merges in the output Observables.

```
import { of } from 'rxjs';
import { mergeMap } from 'rxjs/operators';

of('a', 'b').pipe(
  mergeMap(x => of(`${x}1`, `${x}2`))
).subscribe(console.log);
```

7. `switchMap`

Projects each source value to an Observable, cancels previous if a new one comes.

```
import { interval, of } from 'rxjs';
import { switchMap } from 'rxjs/operators';

interval(1000).pipe(
  switchMap(() => of('new observable'))
).subscribe(console.log);
```

8. `concatMap`

Maps each value to an Observable and concatenates them.

```
import { of } from 'rxjs';
import { concatMap } from 'rxjs/operators';

of('a', 'b').pipe(
  concatMap(x => of(`${x}1`, `${x}2`))
).subscribe(console.log);
```

9. `take`

Emits only the first `` values.

```
import { of } from 'rxjs';
import { take } from 'rxjs/operators';

of(1, 2, 3, 4, 5).pipe(take(3)).subscribe(console.log); // Output: 1 2 3
```

10. `debounceTime`

Waits for specified time before emitting latest value.

```
import { fromEvent } from 'rxjs';
import { debounceTime, map } from 'rxjs/operators';

fromEvent(document, 'click').pipe(
  debounceTime(300),
  map(() => 'Clicked!')
).subscribe(console.log);
```

11. `distinctUntilChanged`

Only emits when the current value is different from the last.

```
import { of } from 'rxjs';
import { distinctUntilChanged } from 'rxjs/operators';

of(1, 1, 2, 2, 3).pipe(
  distinctUntilChanged()
).subscribe(console.log); // Output: 1 2 3
```

12. `combineLatest`

Emits latest value from each observable when any observable emits.

```
import { combineLatest, of } from 'rxjs';

combineLatest([
  of('A'),
  of('B')
]).subscribe(console.log); // Output: ["A", "B"]
```

13. `zip`

Combines values from multiple Observables in sequence.

```
import { zip, of } from 'rxjs';

zip(of(1, 2), of('A', 'B')).subscribe(console.log); // Output: [1, 'A'], [2,
'B']
```

14. `catchError`

Catches errors and returns a new observable.

```
import { of, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

throwError(() => 'Error!').pipe(
  catchError(err => of('Handled Error'))
).subscribe(console.log);
```

15. `retry`

Retries a failed observable a specified number of times.

```
import { throwError, of } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';
```

```
throwError(() => 'Failed').pipe(
  retry(2),
  catchError(err => of('Final Fallback'))
).subscribe(console.log);
```

Download Link

Once saved, a download link for this document will be provided here.