

RxJS Operators

1. Pipe \rightarrow Function that takes an observable as its input and return another Observable
2. Filter \rightarrow Filter source observable before subscribe
3. Tap \rightarrow perform other side-effects / different actions without modify the data stream.

Eg \rightarrow `obs = new Observable((observer) \Rightarrow {
 observer.next(1);
 observer.next(2);
 observer.next(3);
 observer.complete();}).pipe (`

`tap (data \Rightarrow console.log('tap'+data));
 filter (data \Rightarrow data > 2);
 tap (data \Rightarrow console.log('filter'+data));
 map ((val \Rightarrow return val as no * 2));`

We use pipe to chain the tap operator which just logs value of source observable to console.

We use filter, to filter out observable according to our requirement.

4. Take \rightarrow emits only the first count values by the source observable mentioned in take operator.

5. Take Until \rightarrow use for unsubscribe observable until not getting response. Can use for complete and destroy observable after subscribe.

6. Take Last \rightarrow emits only last count values.

7. Take While \rightarrow emit value if condition is true.

Eg → obs = of (1, 2, 3, 4, 5, 6).pipe (

```
takeWhile (val ⇒ val < 3, true),
takeLast(3),
take(2),
takeUntil (Subject)
);
```

Output → 1, 2 (take while)
 4, 5, 6 (take last)
 1, 2 (take)

8. retry When ⇒ retry with condition.
 If you want to retry after few seconds, not instantly then use retry When

Foreg - If getting 404 error, then you should retry

9. From Event - handling DOM events using target Element and event type.

10. debounce Time \rightarrow delays the value emitted by source.

11. distinct Until Changed \rightarrow emits all items that are distinct from previous items

12. Subject - Subject does not return the current value of subscription.

It triggers only on .next(value) call and return output.

Eg \rightarrow `var Subject = new Subject();`
`Subject.next(1);`

// will not output any value

`Subject.subscribe({`
`next: (v) \Rightarrow console.log(v);`

`});`

`Subject.next(1);`

// Logs 1

13. Behaviour Subject \Rightarrow It will return current value or initial value on subscription.

14. asyncSubject \Rightarrow emits last value on completion.

15. Concat \Rightarrow join multiple Observable and emits one by one

Eg \Rightarrow Concat (
 of (1, 2, 3), of (4, 5, 6),
 .subscribe (console.log);

// 1, 2, 3, 4, 5, 6

16. Share Reply \Rightarrow use with async, avoid duplicate request

17. Zip \Rightarrow pass 2 observables using zip. when change will be detected in 2 observables then only subscription will emit value

Use Case - When you have to make several http requests at same time and need to complete all before emitting value.

Eg → `const sourceOne = of('Deepa');`
`const sourceTwo = of('Jyoti');`

`const eg = zip(
 sourceOne,
 sourceTwo, pipe(delay(1000)),
 3);`

`const subscribe = eg.subscribe(
 val ⇒ console.log(val));`

18. ForkJoin → Wait for all observables to complete and then emit an array of last emitted values.

Join multiple Observable to a single one.

If any Observable came through error, forkJoin will not return any value.

Flattening Operators

While using multiple observables into nested Subscription

1) Merge Map \rightarrow Simultaneously executes observables.

- * Sequence of execution not guaranteed
- * If any observable throw error, it will emit other values.

2) Concat Map \rightarrow does not subscribe to next observable until previous completes

- * Execute one by one
- * sequence of execution guaranteed.

3) exhaust Map \rightarrow map to inner observable and ignore other until observable complete

4) Switch Map \rightarrow try to resubscribe observable by cancelling previous