Understanding RxJS map, mergeMap, switchMap and concatMap

Luuk Gruijs Follow
Jan 9, 2019 · 5 min read ★

Mapping data is a common operation while writing your program. When you use RxJS in your code to produce your data streams it's very likely you eventually need a way to map the data to whatever format you need it to be. RxJS comes with a 'normal' map function, but also has functions like mergeMap, switchMap and concatMap which all behave slightly different.

Photo by Dennis Kummer on Unsplash

The map operator

The map operator is the most common of all. For each value that the Observable emits you can apply a function in which you can modify the data. The return value will, behind the scenes, be reemitted as an Observable again so you can keep using it in your stream. It works pretty much the same as how you would use it with Arrays. The difference is that Arrays will always be just Arrays and while mapping you get the value of the current index in the Array. With Observables the type of data can be of all sorts of types. This means that you might have to do some additional operations in side your Observable map function to get the desired result. Let's look at some examples:

We first created our Observable with an array of cars. We then subscribe to this Observable 2 times. The first time we modify our data in such a way that we get an array of concatenated brand and model strings. The second time we modify our data so that we get an array of only Porsche cars. In both examples we use the Observable map operator to modify the data that is being emitted by the Observable. We return the result of our modification and the map operator then behind the scenes takes care of wrapping this in an Observable again so we can later subscribe to it.

MergeMap

Now let's say there is a scenario where we have an Observable that emits an array, and for each item in the array we need to fetch data from the server.

We could do this by subscribing to the array, then setup a map that calls a function which handles the API call and then subscribe to the result. This could look like the following:

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

const getData = (param) => {
  return of(`retrieved new data with param ${param}`).pipe(
    delay(1000)
  )
}
```

index.ts

```ts
import { of, from } from 'rxjs';
import { map, mergeMap, delay, mergeAll } from 'rxjs/operators';

const getData = (param) => {
  return of(`retrieved new data with param ${param}`).pipe(
    delay(1000)
  )
}

// using a regular map
from([1,2,3,4]).pipe(
  map(param => getData(param))
).subscribe(val => val.subscribe(data => console.log(data)));

// using map and mergeAll
from([1,2,3,4]).pipe(
```

observable-mergemap Editor Preview Both Edit on StackBlitz

You might also have heard about flatMap. FlatMap is an alias of mergeMap and behaves in the same way. Don't get confused there!

SwitchMap

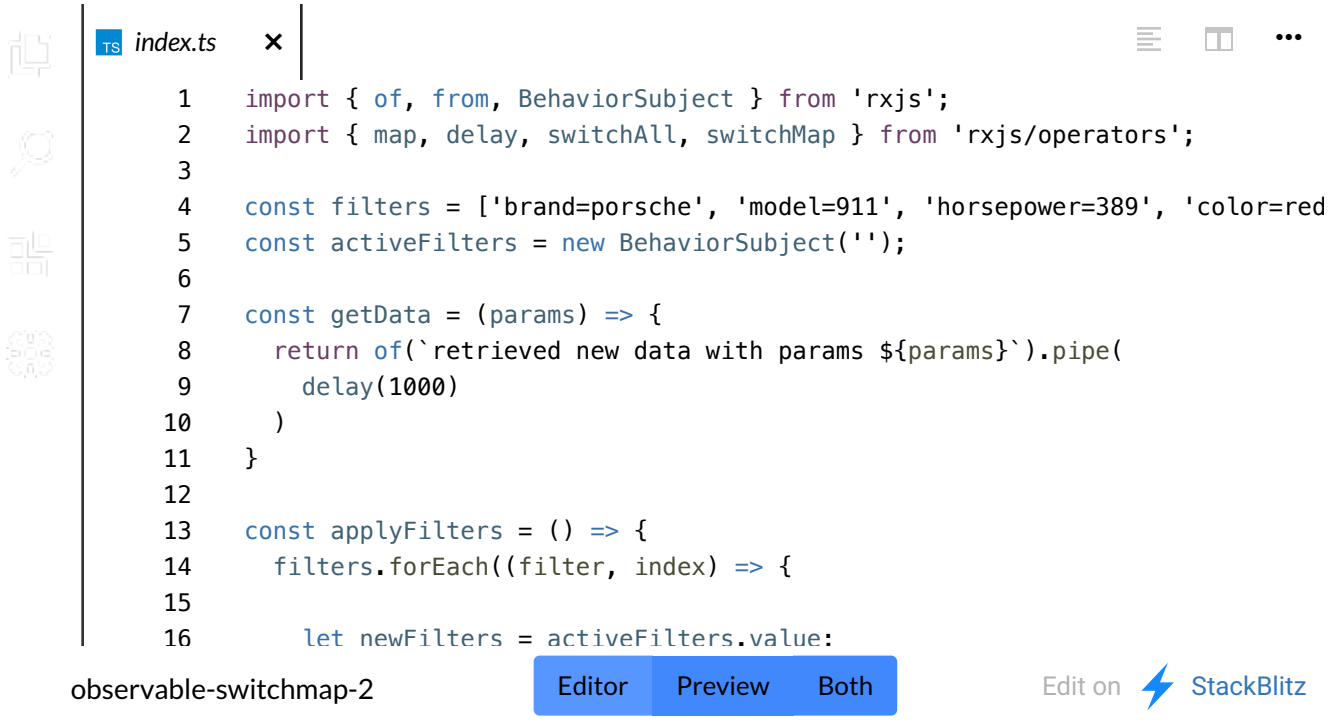
SwitchMap has similar behaviour in that it will also subscribe to the inner Observable for you. However switchMap is a combination of switchAll and map. SwitchAll cancels the previous subscription and subscribes to the new one. For our scenario where we want to do an API call for each item in the array of the 'outer' Observable, switchMap does not work well as it will cancel the first 3 subscriptions and only deals with the last one. This means we will get only one result. The full example can be seen here:

index.ts

```
1   import { of, from } from 'rxjs';
2   import { map, delay, switchAll, switchMap } from 'rxjs/operators';
3
4   const getData = (param) => {
5     return of(`retrieved new data with param ${param}`).pipe(
6       delay(1000)
7     )
8   }
9
10  // using a regular map
11  from([1,2,3,4]).pipe(
12    map(param => getData(param))
13  ).subscribe(val => val.subscribe(data => console.log(data)));
14
15  // using map and switchAll
16  from([1,2,3,4]).pipe(
```

observable-switchmap Editor Preview Both Edit on StackBlitz

While switchMap wouldn't work for our current scenario, it will work for other scenario's. It would for example come in handy if you compose a list of filters into a data stream and perform an API call when a filter is changed. If the previous filter changes are still being processed while a new change is already made, it will cancel the previous subscription and start a new subscription on the latest change. An example can be seen here:



As you can see in the console `getData` is only logging once with all the params. This saved us 3 API calls.

ConcatMap

The last example is concatMap. As you might expect, concatMap also subscribes to the inner Observable for you. But unlike switchMap, that unsubscribes from the current Observable if a new Observable comes in, concatMap will not subscribe to the next Observable until the current one completes. The benefit of this is that the order in which the Observables are emitting is maintained. To demonstrate this:


index.ts
25
observable-concatmap
Editor
Preview
Both
Edit on
StackBlitz

The getData function has a random delay between 1 and 10000 milliseconds. If you check the logs you can see that the map and mergeMap operators will log whatever value comes back and don't follow the original order. On the other hand the concatMap logs the values in the same value as they were started.

Conclusion

Mapping data to the format you need is a common task. RxJS comes with a few very neat operators that help you get the job done. To recap: map is for mapping 'normal' values to whatever format you need it to be. The return value will be wrapped in an Observable again, so you can keep using it in your data stream. When you have to deal with an 'inner' Observable it's easier to use mergeMap, switchMap or concatMap. Use mergeMap if you simply want to flatten the data into one Observable, use switchMap if you need to flatten the data into one Observable but only need the latest value and use concatMap if you need to flatten the data into one Observable and the order is important to you.

. . .

Do you consider yourself to be one of the best?

I work for **Founda** as a Senior front-end developer and we are looking for Senior developers that specialise in Vue and/or Node.

Founda is creating the future of healthcare IT. We are founded by seasoned tech entrepreneurs in January 2019, Founda is a young and well funded company in the health tech & low code / no code space in Amsterdam.

We have been building a technology company using a modern stack with a small team of self-determined developers. We are looking to grow the company with high quality people.

If you think you have what it takes to build the future of Healthcare and you are a European resident. Drop me a line at hello@founda.com.

Rxjs JavaScript Front End Development Rxjs Map Software Development