RxJS: When to Use switchMap

Nicholas Jamieson Follow
Mar 13, 2018 · 3 min read

Photo by Geran de Klerk on Unsplash

In a response to *RxJS: Avoiding switchMap-Related Bugs*, Martin Hochel mentioned a classic use case for `switchMap`. For the use case to which he referred, `switchMap` is not only valid; it's optimal. And it's worth looking at why.

Dealing with stale results

Let's look at an example that involves an expensive call to a backend service: a search for addresses that match a partial address typed into an HTML `input`.

Here's the NgRx effect:

```
@Effect()
public findAddresses = this.actions.pipe(
  ofType(LocationActionTypes.FindAddresses),
  map(action => action.partialAddress),
  debounceTime(400),
  distinctUntilChanged(),
  switchMap(partialAddress => this.backend
    .findAddresses(partialAddress)
    .pipe(
      map(results => new FindAddressesFulfilled(result
```

And here's the `redux-observable` epic:

```
const findAddresses = actions$ => actions$.pipe(
  ofType(actions.FIND_ADDRESSES),
  map(action => action.partialAddress),
  debounceTime(400),
  distinctUntilChanged(),
  switchMap(partialAddress => this.backend
    .findAddresses(partialAddress)
    .pipe(
      map(results => actions.findAddressesFulfilled(re
```

The effect/epic debounces the user input so that backend searches are not performed for each keystroke and uses `distinctUntilChanged` so that no searches are performed unless the partial address has changed. The operator that's then used to flatten the backend observable is `switchMap`.

Why?

From the summarised recommendations in *RxJS: Avoiding switchMap-Related Bugs*, we know that:

- `concatMap` could be used as a conservative choice;
- `mergeMap` should not be used—the ordering of the results is important;
- `switchMap` could be used—when a new search is made, pending results are no longer needed; and
- `exhaustMap` should not be used—searches for new, partial addresses should not be ignored.

So how would the behaviour differ if the effect/epic were to use the conservative choice of `concatMap` rather than `switchMap`?

Whether or not the behaviour differs depends upon two things:

- the time it takes for the backend searches to be fulfilled; and
- the time between the user's keystrokes.

The `debounceTime` operator imposes a limit on how frequently backend searches can occur, so unless the fulfilment of a search takes longer than the debounce time, there will be no difference in behaviour between flattening with `concatMap` and with `switchMap` — regardless of how slowly the user types.

Similarly, there will be no difference in behaviour if the user types so slowly that each search is fulfilled before the next keystroke occurs.

Let's look at the worst-case scenario: the searches take significantly longer than the debounce time to be fulfilled and the user types slowly, with intervals between keystrokes that slightly exceed the debounce time.

In the worst-case scenario, if `concatMap` is used, each keystroke effects a search and the results of that search will still be pending when the next key is pressed—so the next search will be queued. This sees the displayed search results become increasingly out-of-sync with the typed-in partial address and, as each of the searches has to be fulfilled before the next is performed, it might take some time for the displayed results to become consistent with the `input`.

With `switchMap`, each keystroke still effects a search, but any pending search will be aborted. So, in the worst-case scenario, the displayed search results will become consistent with the input more quickly than if `concatMap` were to be used, as they'll be consistent as soon as the pending search is fulfilled.

TL;DR

When choosing a flattening operator for an effect/epic, if subsequent actions of the same type will render pending results stale, `switchMap` is unequivocally the best choice.