RxJS: What’s Changed with shareReplay?

Nicholas Jamieson Follow
Feb 14 · 5 min read

Photo by Namroud Gorguis on Unsplash

In RxJS version 6.4.0, a change was made to the `shareReplay` operator. Let’s look at why the operator needed to be changed, what was changed and how the change can be used to avoid surprises—and bugs.

If you’re only interested in the change, skip to the TL;DR at the bottom of the article.

The operator’s history

The `shareReplay` operator was introduced in version 5.4.0.

Like the `share` operator, `shareReplay` returns a hot, reference-counted observable, but replays the specified number of `next` notifications. Also, if the source completes, it ensures that the `complete` notification is replayed to new subscribers and that no further subscriptions are made to the source observable.

In the version 5.4.0 implementation of `shareReplay`, when the subscriber reference count dropped to zero—without the source having completed—the operator would unsubscribe from the source. Subsequent resubscription to the shared observable would increment the reference count from zero to one, effecting a resubscription to the source using a *newly-created* `ReplaySubject` .

Let's look at an example:

```
import { timer } from "rxjs";
import { shareReplay, takeUntil } from "rxjs/operators
import { log } from "./log";

const source = log(timer(100), "source");
const shared = log(source.pipe(shareReplay(1)), "share

shared.pipe(
  takeUntil(timer(50))
).subscribe(
  value => console.log(`first received: ${value}`)
```

As written, this example won't work with RxJS version 5.4.0 because pipeable operators were not introduced until version 5.5.0. However, the example is written as it would be in version 6, so that it's easier to understand.

Also, all of this article's examples use the following helper function. It wraps an observable, gives it a name and logs to the console whenever a subscriber subscribes or
unsubscribes, or the observable nexts or completes:

```
import { defer, Observable } from "rxjs";
import { finalize, tap } from "rxjs/operators";

export const log = <T>(source: Observable<T>, name: st
  console.log(`${name}: subscribed`);
  return source.pipe(
    tap({
      next: value => console.log(`${name}: ${value}`),
      complete: () => console.log(`${name}: complete`)
```

The example program subscribes to a shared source observable that uses a `timer` to emit a value 100 milliseconds after subscription:

- the first subscription is made immediately, but is unsubscribed —via `takeUntil` —before the source emits a value; and
- the second subscription is made 150 milliseconds after the first and is not unsubscribed.

The program's output is:

```
shared: subscribed
source: subscribed
shared: unsubscribed
source: unsubscribed
shared: subscribed
source: subscribed
source: 0
shared: 0
second received: 0
source: complete
source: unsubscribed
shared: complete
shared: unsubscribed
```

Looking at the output it's clear that two subscriptions are made to the source observable. The unsubscription of the first subscriber to the shared observable effects an unsubscription from the source—because the reference count drops to zero—so when the second subscription to the shared observable is made, the `shareReplay` implementation needs to subscribe to the source again.

In some situations that behaviour might not be what you want.

For example, if the source effects a HTTP request, you might not want that request to be cancelled. You might want the request to complete, so that the response can be stored in the shared observable —ready for the next subscriber.

In version 5.5.0-beta.4, a 'bug' was fixed and the behaviour of the `shareReplay` operator was changed so that the subscription to the source was not unsubscribed when the reference count dropped to zero.

The example program's output with version 5.5.0-beta.4 is:

```
shared: subscribed
source: subscribed
shared: unsubscribed
```

```
source: 0
source: complete
source: unsubscribed
shared: subscribed
shared: 0
second received: 0
shared: complete
shared: unsubscribed
```

Here, only one subscription is made to the source observable. The unsubscription of the first subscriber to the shared observable does not effect an unsubscription from the source. The `ReplaySubject` within the implementation of `shareReplay` remains subscribed to the source observable—which allows the source to complete and the `ReplaySubject` to store the `next` and `complete` notifications.

Unfortunately, by changing `shareReplay` to better support the use case in which a source emits once—and then completes—the use case in which a source emits multiple times—without completing—breaks, as the operator's `ReplaySubject` remains permanently subscribed to the source.

Let's look at an example that uses an `interval` instead of a `timer` :

```
import { interval, timer } from "rxjs";
import { shareReplay, takeUntil } from "rxjs/operators
import { log } from "./log";

const source = log(interval(100), "source");
const shared = log(source.pipe(shareReplay(1)), "share

shared.pipe(
  takeUntil(timer(150))
```

Here a subscription is made to the source and is then unsubscribed—via `takeUntil` —a short while after the source emits its first value.

The program's output is:

```
shared: subscribed
source: subscribed
source: 0
shared: 0
received: 0
shared: unsubscribed
source: 1
```

```
source: 2
source: 3
...
```

Because the source observable does not complete, it is never unsubscribed—despite there being no subscribers to the shared observable. So the source observable continues to emit values.

Forever.

In version 6.4.0, a change was made so that both use cases could be supported.

The change — TL;DR

The essential difference between the version 5.4.0 and 5.5.0-beta.4 implementations of `shareReplay` was that the earlier version implemented reference counting and the later version disabled it.

To allow for both of the use cases that we’ve look at, an additional parameter was added to `shareReplay`.

The operator already had three parameters—all of which were optional—so, to avoid making things event more complicated, an additional overload signature that takes a single config parameter was added. The config interface and signature look like this:

```
interface ShareReplayConfig {
  bufferSize?: number;
  windowTime?: number;
  refCount: boolean;
  scheduler?: SchedulerLike;
}
```

Using the config parameter, the example that uses an `interval` as the source can be rewritten to behave correctly:

```
import { interval, timer } from "rxjs";
import { shareReplay, takeUntil } from "rxjs/operators
import { log } from "./log";

const source = log(interval(100), "source");
const shared = log(source.pipe(shareReplay({
  bufferSize: 1,
  refCount: true
})), "shared");

shared.pipe(
```

With `refCount` enabled, the source will be unsubscribed when the reference count drops to zero and the source will no longer emit forever:

```
shared: subscribed
source: subscribed
source: 0
shared: 0
received: 0
shared: unsubscribed
source: unsubscribed
```

Avoiding surprises

The `refCount` property in the config parameter is deliberately not optional.

To avoid any potential confusion—due to the differing implementations of `shareReplay` between RxJS versions—I would recommend always using the signature that takes the config parameter. That way, it'll be clear whether or not the reference counting behaviour is or isn't wanted.

I've added an option to the `rxjs-no-sharereplay` rule in `rxjs-tslint-rules` to allow `shareReplay` to be used if the config parameter is specified. The rule's configuration looks like this:

```
"rules": {
  "rxjs-no-sharereplay": {
    "options": [{
      "allowConfig": true
    }],
    "severity": "error"
    }
  }
```

If `shareReplay` is an operator that you use in your apps, you might want to make sure that it's behaving appropriately for your use case. And you might want to switch to passing a config argument.