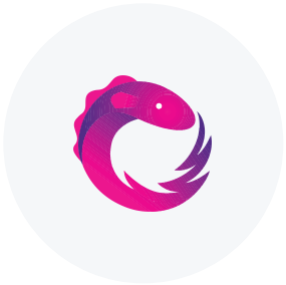


RXJS

RxJS 6: What's new and what has changed?

RxJs 6 is out to provide developers with improvements in modularity, a boost in performance and easier to debug call stacks.

Dan Arias
R&D Content Engineer

April 30, 2018

RxJs 6 is out and with it new exciting additions and changes! Ben Lesh highlights that RxJS 6 brings cleaner imports while having a smaller API, a backward compatibility package to update without changing your code, and automatic code migration for TypeScript.

These changes provide developers with improvements in modularity, a boost in performance and easier to debug call stacks. The RxJS team has made a solid effort on making this release as backward compatible as possible. However, in an effort to reduce the API surface of the RxJS library, some breaking changes were introduced.

> "RxJS 6 brings improvements in modularity, a boost in performance and easier to debug call stacks. The RxJS team has made a solid effort on making this release as backward compatible as possible"

Tweet This

Let's explore what the RxJS team has included and changed in this new release.

RxJS 6 Backward Compatibility

To make the migration path from RxJS 5 to RxJS 6, the RxJS team has released a sibling package called `rxjs-compat`. This package creates a compatibility layer between the APIs of `v6` and `v5`.

The team recommends that most developer upgrade existing applications by installing both `rxjs` and `rxjs-compat` at `^6.0.0`:

```
npm install rxjs@6 rxjs-compat@6 --save
```

This package allows you to continue running your existing codebase without issues while you implement the RxJS 6 upgrades. It supports functionality that is removed with the release of RxJS 6.

The bundle size of your application will increase with the installation of `rxjs-compat`; this effect is amplified if your project also integrates with Webpack < `4.0.0`. Therefore, it is recommended that `rxjs-compat` is removed from your project once the upgrade process has been completed.

"rxjs-compat makes it easy to upgrade to RxJS 6 as it creates a compatibility layer between the APIs of v6 and v5. This layer provides your codebase with functionality that is being removed in v6 so that you can upgrade gradually."

Tweet This

> **Upgrading to RxJS 6 may introduce type errors in your codebase that were not previously shown.**

Limitations of Upgrading to RxJS with rxjs-compat

There are only two breaking changes that are not covered by the `rxjs-compat` package:

TypeScript prototype operators

In the rare instance that your codebase defines its own TypeScript prototype operators and modifies the `Observable` namespace, your operator code would need to be updated in order for TypeScript to compile.

From the release notes examples, a user-defined prototype operator can be created as follows:

```
Observable.prototype.userDefined = () => {
  return new Observable((subscriber) => {
    this.subscribe({
      next(value) { subscriber.next(value); },
      error(err) { subscriber.error(err); },
      complete() { subscriber.complete(); },
   });
  });
});

source$.userDefined().subscribe();
```

To compile the previously custom operator, the following changes would need to be made:

```
const userDefined = <T>() => (source: Observable<T>) => new Observable<T>((subscriber) => {
    this.subscribe({
      next(value) { subscriber.next(value); },
      error(err) { subscriber.error(err); },
      complete() { subscriber.complete(); },
   });
  });
});

source$.pipe(
  userDefined(),
)
```

Synchronous error handling

Calling `Observable.subscribe()` within a `try/catch` block is no longer supported. Instead, replace the `try/catch` block with asynchronous error handling done with the `error` callback in the `Observable.subscribe()` method.

As shown in the release notes:

```
// deprecated
try {
```

```
    source$.subscribe(nextFn, undefined, completeFn);
  } catch (err) {
    handleError(err);
  }

  // use instead
  source$.subscribe(nextFn, handleError, completeFn);
```

`Observable.subscribe()` now *must* define an `error` callback to handle errors asynchronously.

Changes to Make Before Dropping the RxJS Compatibility Layer

As mentioned earlier, `rxjs-compat` provides a temporary compatibility layer between the APIs of `v5` and `v6`. Essentially, `rxjs-compat` provisions your codebase with functionality from `v5` that it relies on, allowing you to gradually upgrade your codebase to `v6`. To complete the upgrade process and remove the `rxjs-compat` dependency from your project, your codebase would need to be refactored to stop relying on that `v5` functionality which includes:

Changes to `import` paths

The recommendation for TypeScript developers is to use `rxjs-tslint` to refactor `import` paths.

The following rules have been designed by the RxJS team to help JavaScript developers refactor `import` paths:

- `rxjs`: Contains creation methods, types, schedulers, and utilities.

```
import {
  Observable,
  Subject,
  asapScheduler,
  pipe,
  of,
  from,
  interval,
  merge,
  fromEvent
} from "rxjs";
```

- `rxjs/operators`: Contains all pipeable operators.

```
import { map, filter, scan } from "rxjs/operators";
```

- `rxjs/webSocket`: Contains the web socket subject implementation.

```
import { webSocket } from "rxjs/webSocket";
```

- `rxjs/ajax`: Contains the Rx ajax implementation.

```
import { ajax } from "rxjs/ajax";
```

- `rxjs/testing`: Contains the testing utilities for RxJS.

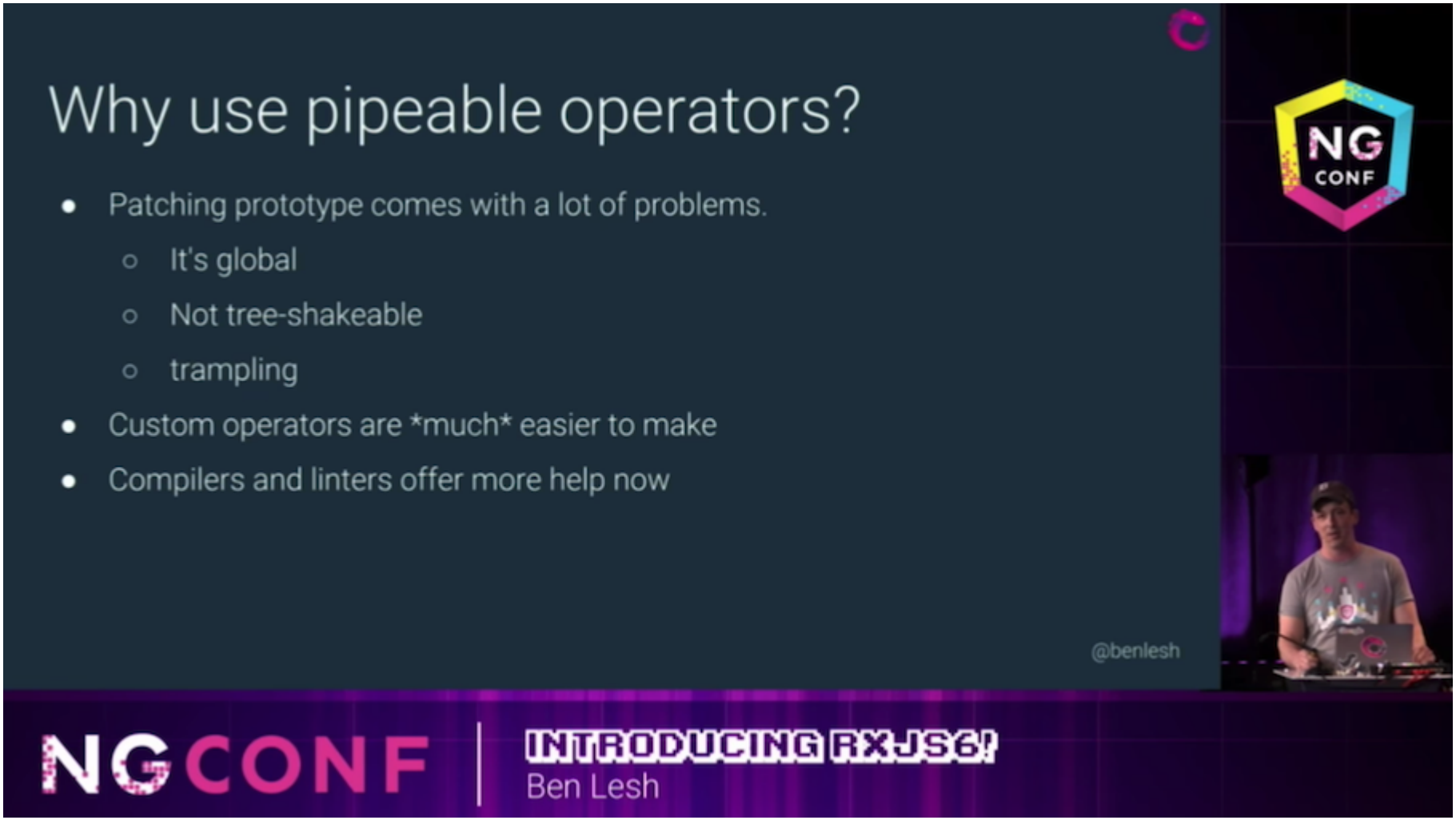
```
import { TestScheduler } from "rxjs/testing";
```

Use Piping Instead of Chaining

Use piping instead of chaining as new operator syntax. The result of one operator is piped into another operator.

Don't remove `rxjs-compat` until you have refactored all chained operators into piped operators. If you are a TypeScript user, `ts-lint` can automate this refactoring to some extent for well-typed code.

During ng-conf 2018, Ben Lesh explained why we should use pipeable operators:



Follow these steps to reactor your operator chains into pipes:

- Install all operators used from `rxjs/operators`.

> **Note Some operators have a name change due to name collisions with JavaScript reserved words! These include:** `do` -> `tap`, `catch` -> `catchError`, `switch` -> `switchAll`, `finally` -> `finalize`.

```
import { map, filter, catchError, mergeMap } from "rxjs/operators";
```

- Attach a `pipe()` method to the source and wrap all the operators within it. Ensure that the `.` is removed from each operator name and that they are comma-delimited. Remember that some operators need to change their names.

The following is an example of a pipeable refactoring from the release notes:

```
// an operator chain
source
  .map(x => x + x)
  .mergeMap(n =>
    of(n + 1, n + 2)
      .filter(x => x % 1 == 0)
      .scan((acc, x) => acc + x, 0)
  )
```

```
  .catch(err => of("error found"))
  .subscribe(printResult);

// must be updated to a pipe flow

source
  .pipe(
    map(x => x + x),
    mergeMap(n =>
      of(n + 1, n + 2).pipe(
        filter(x => x % 1 == 0),
        scan((acc, x) => acc + x, 0)
      )
    ),
    catchError(err => of("error found"))
  )
  .subscribe(printResult);
```

Notice how we use `pipe()` twice in the above code as internal sources are also piped.

Use Functions instead of Classes

Functions have replaced classes that operate on observables. All observable classes have been removed. Their functionality is replaced by existing operators, new operators, or functions. Each of their replacement has the same functionality that each class had.

For example:

```
// removed
ArrayObservable.create(myArray)

// use instead

from(myArray)

// you may also use

new operator fromArray().
```

For a complete list of the `v6` creation functions that replace `v5` classes, please visit the RxJS documentation.

Special case:

- `ConnectableObservable` is hidden from direct use in `v6`. To access it, use the operators `multicast`, `publish`, `publishReplay`, `publishLast`.
- `SubscribeOnObservable` is hidden from direct use in `v6`. To access it, use the operator `subscribeOn`.

Removing resultSelector

Result selectors are a RxJS feature that is not widely used and in some cases wasn't even documented. However, result selectors did bloat the codebase significantly; thus, the RxJS team decided to deprecate or remove them.

For developers making use of result selectors, they would need to replace the `resultSelector` parameter with external result-selection code.

For two functions, `first()` and `last()` these parameters have been removed and must be updated before removing `rxjs-compat`.

For other functions that have `resultSelector` as a parameter, such as mapping operators, this parameter has been deprecated and their implementation has been re-written in a much more compact form. If you remove `rxjs-compat`, these function will continue to work; however, the RxJS team states that they must be removed before the `v7` release.

For more details on this rare implementation, please visit the RxJS documentation.

Other RxJS 6 Deprecations

`Observable.if` and `Observable.throw`.

`Observable.if` has been replaced by `iif()` and `Observable.throw` is now `throwError()`. You can use `rxjs-tslint` to convert these deprecated `Observable` method calls into function calls.

The release notes gives us the following example:

Observable.if > iif()

```
// deprecated
Observable.if(test, a$, b$);

// use instead

iif(test, a$, b$);
```

Observable.error > throwError()

```
// deprecated
Observable.throw(new Error());

//use instead

throwError(new Error());
```

Deprecated Methods

According to the migration guide, other methods have been deprecated and refactored:

merge

```
import { merge } from "rxjs/operators";

// becomes

import { merge } from "rxjs";
merge(a$, b$, c$);
```

concat

```
import { concat } from "rxjs/operators";
a$.pipe(concat(b$, c$));

// becomes

import { concat } from "rxjs";
concat(a$, b$, c$);
```

combineLatest

```
import { combineLatest } from "rxjs/operators";
a$.pipe(combineLatest(b$, c$));

// becomes

import { combineLatest } from "rxjs";
combineLatest(a$, b$, c$);
```

race

```
import { race } from "rxjs/operators";
a$.pipe(race(b$, c$));

// becomes

import { race } from "rxjs";
race(a$, b$, c$);
```

zip

```
import { zip } from "rxjs/operators";
a$.pipe(zip(b$, c$));

// becomes

import { zip } from "rxjs";
zip(a$, b$, c$);
```

Recap

RxJS 6 brings some breaking changes but they are mitigated by the addition of the `rxjs-compat` package that allows you to gradually migrate while keeping your `v5` code operational. For TypeScript users, which cover the majority of Angular developers, `tslint` offers a great deal of automated refactoring to make the transition even easier.

As always, any upgrades and code changes may invite bugs into the codebase or even make elusive ones resurface. Solid testing practices should always be in place to ensure that your users keep receiving the same quality experience that you were providing them when `v5` was part of your codebase.

Watch the complete introduction to RxJS 6 by Ben Lesh:

About Auth0

Auth0, a global leader in Identity-as-a-Service (IDaaS), provides thousands of customers in every market sector with the only identity solution they need for their web, mobile, IoT, and internal applications. Its extensible platform seamlessly authenticates and secures more than 2.5 billion logins per month, making it loved by developers and trusted by global enterprises. The company's U.S. headquarters in Bellevue, WA, and additional offices in Buenos Aires, London, Tokyo, and Sydney, support its global customers that are located in 70+ countries.

For more information, visit https://auth0.com or follow @auth0 on Twitter.

> **AUTH0 DOCS**
>
> Implement Authentication in Minutes
>
> **AUTH0 COMMUNITY**

Join the Conversation

Dan Arias

R&D CONTENT ENGINEER

Adventuring across the digital seas with a gentle whale and a feisty gopher at my side.

VIEW PROFILE

More like this

FULLSTACK

Using Python, Flask, and Angular to Build Modern Web Apps - Part 1

PYTHON

Reactive Programming in Python

ANGULAR

Angular Authentication Tutorial

Follow the conversation

elmota • a year ago

Does Rxjs6 have proper documentation? For example, the old combineLatest had a function argument, does the operator have one as well?

11 • Reply • Share ›

Dan Arias Mod elmota • a year ago

Hello! Thanks for reading. This is the documentation that I was able to find:

https://rxjs-dev.firebaseap...

There's also another documentation site:

http://rxjsdocs.com/#/opera...

I know that the RxJS Team has a Docs Team and they have been working on improving the docs here: https://github.com/Reactive...

I would recommend opening an issue there if you find any documentation to be missing.

1 • Reply • Share ›

Michael Prentice • a year ago

Hi Dan! Thanks for the helpful post!

combineLatest

```
import { combineLatest } from "rxjs/operators";
a$.pipe(combineLatest(b$, c$));

// becomes

import { combineLatest } from "rxjs";
combineLatest(a$, b$, c$);
```

Aren't these examples all backwards?

Shouldn't it be this?

```
import { combineLatest } from "rxjs";
combineLatest(a$, b$, c$);

// becomes

import { combineLatest } from "rxjs/operators";
a$.pipe(combineLatest(b$, c$));
```

1 • Reply • Share ›

Michael Prentice ➚ Michael Prentice • a year ago • edited

I guess not. Strange. I guess the pipeable operator is already deprecated!

I am coming from:

```
import {combineLatest} from 'rxjs/observable/combineLatest';
combineLatest(
this.languageSubject,
this.ageSubject,
(language: string, age: number) => {
return {language: language, age: age};
}
)
)
```

Which I guess now becomes

```
import {combineLatest} from 'rxjs';
combineLatest(
this.languageSubject,
this.ageSubject
)
```

1 • Reply • Share ›

Dan Arias Mod ➚ Michael Prentice • a year ago

Thanks for reading, Michael. This article was also helpful to me :)

https://www.learnrxjs.io/op...

• Reply • Share ›

Dean Radcliffe • a year ago

I haven't been able to find anything about backpressure - specifically, the ability of a producer Observable to not emit until the subscriber calls request on a Subscription object. This seems to be present in some documentation for Rx old and Rx Java, but I see no such guidance in RxJS v6 - is there a suitable replacement now? I'm very confused!

• Reply • Share ›

Dan Arias Mod ➚ Dean Radcliffe • a year ago

Thanks for reading, Dean. I'll be happy to help you the best possible way. Could you please elaborate a bit more on your use case? Do you have existing code that does this with RxJS 5 that you could please share? Thanks!

• Reply • Share ›