Angular 6 Upgrade - Breaking Changes & Features - What's New?

lar

Angular Image Upload Made Easy

Angular 6 - What's New & Upgrade Guide

Angular Elements Introduction

Content Information

Created by Maximilian Schwarzmüller

Last updated on 3. May 2018

SHOW RESOURCES

Newsletter email REGISTER

In general, your Angular 5 code should still work.

There only is one breaking change which is introduced by a third-party library (RxJS).

Here are all the changes:

- `ng update` (Angular CLI): A convenient command that will update your project dependencies to their latest versions
- **[BREAKING]** RxJS 6: New import paths & operator usage (also see separate article) - there also is a quick fix (one command only) available
- CLI update and new project config file
- `<template>` deprecated (use `<ng-template>` instead)
- An alternative way of providing application-wide services
- Angular Elements
- Ivy Renderer (beta)
- Bundle size improvements (behind the scenes)

Updating easily via ng update


The Angular CLI has a new command for that:

```
ng update
```

In order to be able to use this command, you need to update your CLI both globally on your machine as well as in the project you want to update:

```
npm install -g @angular/cli@latest
npm install @angular/cli@latest
```

To update your project to a CLI 6.0+ project, you can run the following command (inside the project):

```
ng update @angular/cli
```

The general folder structure won't really change by a lot (you got a new `angular.json` file instead of the old `.angular-cli.json` file) but under the hood, the new CLI is used.

Once you did that, you can analyse an existing project by running:


commands manually. For others, it'll offer `ng update ...` commands.

For example, to update Angular, you run:

```
ng update @angular/core
```

If you're facing any errors whilst doing this, make sure to delete `node_modules` and the `package-lock.json` file in your project and re-run `npm install`. If you can't succeed with updating, you can always roll back by installing the older CLI version again (`npm install -g @angular/cli@1.7`), copying over the `src/` code into a newly created project (with that CLI version) and continuing with that project.

RxJS 6 Changes - Quick Fix

Angular 6 uses RxJS 6 and that in turn introduces a new internal package structure and operator concept which requires you to update your existing code.

To update to RxJS 6, you simply run:


command in your updated or new Angular project:

```
npm install --save rxjs-compat
```

Alternatively, `ng update rxjs` should do the RxJS update + the installation of the `rxjs-compat` package automatically.

`rxjs-compat` is a package that basically adds functionality to the new RxJS version to still support your old import paths and operator usage.

Using `rxjs-compat` is a fine fix that ensures that your old code continues to work. Ultimately, you should update your code to work without that package though.

This requires two adjustments:

1. Change your import paths
2. Change the way you use operators

Don't miss the detailed article I also wrote on the RxJS 6 updating strategy.


```
import { Observable } from 'rxjs/Observable';
import { Subject } from 'rxjs/Subject';
```

becomes

```
import { Observable, Subject } from 'rxjs';
```

So all `from 'rxjs/Something'` imports become `from 'rxjs'`.

Additionally, operator imports have to change - because the way you use operators differs (see next section).

```
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/throttle';
```

becomes

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

Instead of patching the `Observable` prototype - that's what happened in the past - you now explicitly import operator functions like `map` or `throttle`. You'll see how to use these functions in the next section.


becomes

```
import { of } from 'rxjs';
```

RxJS 6 Changes - Changed Operator Usage

With the imports updated, you're almost there to use RxJS 6 without `rxjs-compat`.

But since you now import operators as functions (see last section), you need to change the way you use operators.

Instead of

```
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/throttle';

myObservable
  .map(data => data * 2)
  .throttle(...)
  .subscribe(...);
```

you now use the new `pipe()` method introduced by RxJS:


You simply wrap your operator function calls with the `pipe()` method and it'll execute them - on the observable data - from left to right.

There also are some other changes - renamed operators for example - but I strongly recommend that you dive into the separate article I wrote on that.

Using "angular.json" instead of ".angular-cli.json"

The Angular CLI also received an update and new Angular projects now use an `angular.json` file instead of a `.angular-cli.json` file.

This config file still fulfils the same tasks as before, it's schema just changed slightly. If you needed to add a style import to `styles[]` in there, you'll still find that array in the file though. The same goes for your other config items.

You can automatically update your existing `.angular-cli.json` file to the new `angular.json` file by running this command

Angular 6 deprecates the `<template>` element - you can't use it anymore inside of your component templates. Use `<ng-template>` instead.

Important: `<template>` has **nothing** to do with the `template` or `templateUrl` of your `@Component({...})` decorator! You still use that.

`<template>` simply was an HTML element you use **inside** of your component templates, for example in conjunction with `ngIf` .

```
<template [ngIf]="isAuth">
  <p>This only renders if isAuth is true</p>
</template>
```

You might not've used this syntax too often because `ngIf` actually offers a syntactically nicer version: `*ngIf` . This automatically compiles to the above example when added to the `<p>` tag.

Or it did so to be precise.


Alternative Way of Providing Application-Wide Services

If you want to provide a service to the entire application, you add it to `providers[]` in the `AppModule` .

```
// my.service.ts
export class MyService { ... }

// In app.module.ts
import { MyService } from './path/to/my.service';

@NgModule({
  declarations: [...],
  providers: [ MyService ] // The same instance of the servic
})
export class AppModule {}
```

This approach still works and there's nothing wrong with it!

But with Angular 6 released, you can also use an easier way of marking a service as global:

```
// mv.service.ts
```

```
export class AppModule {}
```

Both code snippets lead to exactly the same behavior, the second one obviously saves you some lines of code.

Angular Elements

Angular 6 introduces Angular Elements - a feature that allows you to compile Angular components to native web components which you can use in your Angular app.

This is important: You can use them in your Angular app. In the future, this may change but right now, you **don't** get web components you could dump into a web app using a different (or none) JS framework!

How does it work then?

I created a video & article where I show an example.

A First Look at the new Ivy Renderer


You can see a demo here.

The goal of Ivy is to drastically reduce the bundle size of your app and speed up its loading time. It aims to be backward-compatible but as you can see in the above demo, it uses a different approach for rendering Angular components.

Feel free to play around with it, dive into the official status tracker to see what you may use already but keep in mind that this only becomes a real alternative when Angular 7+ is released.