Angular

Tuesday, August 25, 2015

Angular 1 and Angular 2 integration: the path to seamless upgrade

Have an existing Angular 1 application and are wondering about upgrading to Angular 2? Well, read on and learn about our plans to support incremental upgrades.

Summary

Good news!

- We're enabling mixing of Angular 1 and Angular 2 in the same application.
- You can mix Angular 1 and Angular 2 components in the same view.
- Angular 1 and Angular 2 can inject services across frameworks.
- Data binding works across frameworks.

Why Upgrade?

Angular 2 provides many benefits over Angular 1 including dramatically better performance, more powerful templating, lazy loading, simpler APIs, easier debugging, even more testable and much more. Here are a few of the highlights:

Better performance

We've focused across many scenarios to make your apps snappy. 3x to 5x faster on initial render and re-render scenarios.

- Faster change detection through monomorphic JS calls
- Template precompilation and reuse
- View caching
- Lower memory usage / VM pressure
- Linear (blindingly-fast) scalability with observable or immutable data structures
- Dependency injection supports incremental loading

More powerful templating

- Removes need for many directives
- Statically analyzable - future tools and IDEs can discover errors at development time instead of run time
- Allows template writers to determine binding usage rather than hard-wiring in the directive definition

Future work

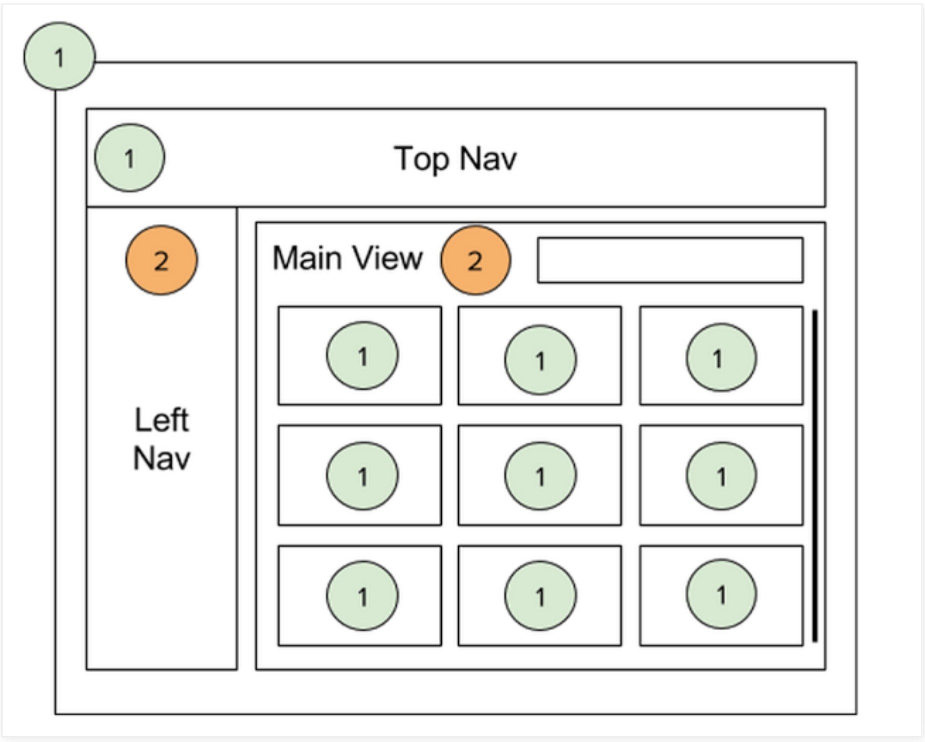
We've decoupled Angular 2's rendering from the DOM. We are actively working on supporting the following other capabilities that this decoupling enables:

- **Server-side rendering.** Enables super-fast initial render and web-crawler support.
- **Web Workers.** Move your app and most of Angular to a Web Worker thread to keep the UI smooth and responsive at all times.
- **Native mobile UI.** We're enthusiastic about supporting the Web Platform in mobile apps. At the same time, some teams want to deliver fully native UIs on their iOS and Android mobile apps.
- **Compile as build step.** Angular apps parse and compile their HTML templates. We're working to speed up initial rendering by moving the compile step into your build process.

Angular 1 and 2 running together

Angular 2 offers dramatic advantages over Angular 1 in performance, simplicity, and flexibility. We're making it easy for you to take advantage of these benefits in your existing Angular 1 applications by letting you seamlessly mix in components and services from Angular 2 into a single app. By doing so, you'll be able to upgrade an application one service or component at a time over many small commits.

For example, you may have an app that looks something like the diagram below. To get your feet wet with Angular 2, you decide to upgrade the left nav to an Angular 2 component. Once you're more confident, you decide to take advantage of Angular 2's rendering speed for the scrolling area in your main content area.



For this to work, four things need to interoperate between Angular 1 and Angular 2:

- Dependency injection
- Component nesting
- Transclusion
- Change detection

To make all this possible, we're building a library named `ng-upgrade`. You'll include ng-upgrade and Angular 2 in your existing Angular 1 app, and you'll be able to mix and match at will.

You can find full details and pseudocode in the original upgrade design doc or read on for an overview of the details on how this works. In future posts, we'll walk through specific examples of upgrading Angular 1 code to Angular 2.

Dependency Injection

First, we need to solve for communication between parts of your application. In Angular, the most common pattern for calling another class or function is through dependency injection. Angular 1 has a single root injector, while Angular 2 has a hierarchical injector. Upgrading services one at a time implies that the two injectors need to be able to provide instances from each other.

The ng-upgrade library will automatically make all of the Angular 1 injectables available in Angular 2. This means that your Angular 1 application services can now be injected anywhere in Angular 2 components or services.

Exposing an Angular 2 service into an Angular 1 injector will also be supported, but will require that you to provide a simple mapping configuration.

The result is that services can be easily moved one at a time from Angular 1 to Angular 2 over independent commits and communicate in a mixed-environment.

Component Nesting and Transclusion

In both versions of Angular, we define a component as a directive which has its own template. For incremental migration, you'll need to be able to migrate these components one at a time. This means that ng-upgrade needs to enable components from each framework to nest within each other.

To solve this, ng-upgrade will allow you to wrap Angular 1 components in a facade so that they can be used in an Angular 2 component. Conversely, you can wrap Angular 2 components to be used in Angular 1. This will fully work with transclusion in Angular 1 and its analog of content-projection in Angular 2.

In this nested-component world, each template is fully owned by either Angular 1 or Angular 2 and will fully follow its syntax and semantics. This is not an emulation mode which mostly looks like the other, but an actual execution in each framework, dependending on the type of component. This means that components which are upgraded to Angular 2 will get all of the benefits of Angular 2, and not just better syntax.

This also means that an Angular 1 component will always use Angular 1 Dependency Injection, even when used in an Angular 2 template, and an Angular 2 component will always use Angular 2 Dependency Injection, even when used in an Angular 1 template.

Change Detection

Mixing Angular 1 and Angular 2 components implies that Angular 1 scopes and Angular 2 components are interleaved. For this reason, ng-upgrade will make sure that the change detection (Scope digest in Angular 1 and Change Detectors in Angular 2) are interleaved in the same way to maintain a predictable evaluation order of expressions.

ng-upgrade takes this into account and bridges the scope digest from Angular 1 and change detection in Angular 2 in a way that creates a single cohesive digest cycle spanning both frameworks.

Typical application upgrade process

Here is an example of what an Angular 1 project upgrade to Angular 2 may look like.

1. Include the Angular 2 and ng-upgrade libraries with your existing application
2. Pick a component which you would like to migrate
   a. Edit an Angular 1 directive's template to conform to Angular 2 syntax
   b. Convert the directive's controller/linking function into Angular 2 syntax/semantics
   c. Use ng-upgrade to export the directive (now a Component) as an Angular 1 component (this is needed if you wish to call the new Angular 2 component from an Angular 1 template)
3. Pick a service which you would would like to migrate
   a. Most services should require minimal to no change.
   b. Configure the service in Angular 2
   c. (optionally) re-export the service into Angular 1 using ng-upgrade if it's still used by other parts of your Angular 1 code.
4. Repeat doing step #2 and #3 in an order convenient for your application developmen
5. Once no more services/components need to be converted drop the top level Angular 1 bootstrap and replace with Angular 2 bootstrap.

Note that each individual change can be checked in separately and the application continues working letting you continue to release updates as you wish.

We are not planning on adding support for allowing non-component directives to be usable on both sides. We think most of the non-component directives are not needed in Angular 2 as they are supported directly by the new template syntax (i.e. ng-click vs (click))

Q&A

I heard Angular 2 doesn't support 2-way bindings. How will I replace them?
Actually, Angular 2 supports two way data binding and ng-model, though with slightly different syntax.

When we set out to build Angular 2 we wanted to fix issues with the Angular digest cycle. To solve this we chose to create a unidirectional data-flow for change detection. At first it was not clear to us how the two way forms data-binding of ng-model in Angular 1 fits in, but we always knew that we had to make forms in Angular 2 as simple as forms in Angular 1.

After a few iterations we managed to fix what was broken with multiple digests and still retain the power and simplicity of ng-model in Angular 1.

Two way data-binding has a new syntax: [(property-name)]="expression" to make it explicit that the expression is bound in both directions. Because for most scenarios this is just a small syntactic change we expect easy migration.

As an example, if in Angular 1 you have:
```
<input type="text" ng-model="model.name" />
```

You would convert to this in Angular 2:
```
<input type="text" [(ng-model)]="model.name" />
```

What languages can I use with Angular 2?
Angular 2 APIs fully support coding in today's JavaScript (ES5), the next version of JavaScript (ES6 or ES2015), TypeScript, and Dart.

While it's a perfectly fine choice to continue with today's JavaScript, we'd like to suggest that you explore ES6 and TypeScript (which is a superset of ES6) as they provide dramatic improvements to your productivity.

ES6 provides much improved syntax and standards for common libraries like promises and modules. TypeScript gives you dramatically better code navigation, automated refactoring in IDEs, documentation, finding errors, and more.

Both ES6 and TypeScript are easy to adopt as they are supersets of today's ES5. This means that all your existing code is valid and you can add their features a little at a time.

What should I do with $watch in our codebase?
In order to gain speed and predictability, in Angular 2 you specify watch expressions declaratively in the HTML template or in annotations on your component directives. One additional benefit from this is that Angular 2 applications can be safely minified/obfuscated for smaller payload.

For any scenarios that don't fit with these mechanisms, you can take advantage of observables -- ES7 Observables as in Rx.js for JavaScript or Streams in Dart.

What can I do today to prepare myself for the migration?
Follow the best practices and build your application using components and services in Angular 1 as described in the AngularJS Style Guide.

Wasn't the original upgrade plan to use the new Component Router?
The upgrade plan that we announced at ng-conf 2015 was based on upgrading a whole view at a time and having the Component Router handle communication between the two versions of Angular.

The feedback we received was that while yes, this was incremental, it wasn't incremental enough. We went back and redesigned for the plan as described above.

Are there more details you can share?
Yes! In the Angular 1 to Angular 2 Upgrade Strategy design doc.

We're working on a series of upcoming posts on related topics including:

- Mapping your Angular 1 knowledge to Angular 2.
- A set of FAQs on details around Angular 2.
- Detailed migration guide with working code samples.

See you back here soon!

Posted by Brad Green at 8:30 AM

No comments:

Post a Comment

Note: Only a member of this blog may post a comment.

Enter your comment...

Comment as: jalbacar (Google Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Simple theme. Powered by Blogger.