AoT in Angular 2

Every Angular application requires a compilation process before they can run in the browser: the enriched components and templates provided by Angular cannot be understood by the browser directly. During the compilation, Angular's compiler also improves the app run-time performance by taking JavaScript VM's feature (like inline caching) into consideration.

The initial compiler in Angular 1.x and Angular 2 is called JiT (Just-in-Time) compiler. As for AoT, it stands for the Ahead-of-Time compiler that was recently introduced in Angular 2. Compared to the JiT compilation performed by Angular 2 at run-time, AoT provides a smaller bundle with faster rendering in the browser. Using AoT, we can reduce the angular2-starter to 428.8 kb compared to the original 1.2 MB and reduce loading times by skipping compilation in the browser.

Characteristic	JiT	AoT
Compilation target	Browser	Server
Compilation context	Runtime	Build
Bundle size	Huge (~1.2 MB)	Smaller (~400 KB)
Execution Performance	-	Better
Startup time	-	Shorter

The gist of AoT is moving the compilation from run-time to the building process. That means, first we can remove the JiT compiler (which is around 523kb) from the bundle to have a smaller build, and second, the browser can execute the code without waiting for JiT in the run-time which leads to a faster rendering speed.

Early compilation also means that developers can find template bugs without actually running the code and before it reaches to client. This provides a more robust application with higher security because less client-side HTML and JavaScript are `eval` ed. Also, by introducing compiled code in the building process, AoT makes the application more tree-shakable and open to various other optimizations. Bundlers like Rollup and Google Closure can take that advantage and effectively decrease the bundle size.

Besides, AoT compiler also inlines HTML templates and CSS files and help reduce the amount of asynchronous requests sent by the application. (Note: this caused a config bug that we will mention in a latter section)

AoT limitations

However, AoT is not perfect. The main limitation is that AoT, due to the way it compiles the raw code, cannot be used with common code patterns, for example, default exports from modules, template literals for templates, and functions in providers, routes, or declarations. Currently, we do not have a complete list of "AoT Do's and Don'ts" and the Angular team has not released anything regarding this issue. Rangle made its own list here and also provides a sandbox for testing features with AoT.

Another problem with AoT is that when the application reaches certain complexity, the AoT bundle compared to JiT bundle can actually takes up more space. As an trade-off of having a simpler logic for browser (therefore faster rendering speed), the code generated by AoT is actually more verbose compared to "dynamic" JiT.

AoT Configuration

To enable AoT in Angular 2, there are two possible methods:

- using `ngc` directly
- using `@ngtools/webpack`

We recommend the second way because it fits the Angular + Webpack toolchain the best. One problem of using raw `ngc` is that `ngc` tries to inline CSS while lacking necessary context. For example, the `@import 'basscss-basic'` statement in `index.css` would cause an error like `Error: Compilation failed. Resource file not found` with `ngc`. It lacks the information that `'basscss-basic'` is actually a node module inside `node_modules`. On the other hand, `@ngtools/webpack` provides `AotPlugin` and loader for Webpack which shares the context with other loaders/plugins. So when `ngc` is called by `@ngtools/webpack`, it can gather necessary informations from other plugins like `postcss-import` to correctly understand things like `@import 'basscss-basic'`.

Config `@ngtools/webpack`

First, get `@ngtools/webpack` from `npm` and save it as a development dependency:

```
npm install -D @ngtools/webpack
```

Then, inside the Webpack configuration file (usually named as `webpack.config.js`), add following code:

```
import {AotPlugin} from '@ngtools/webpack'

exports = { /* ... */
  module: {
    rules: [
      {
        test: /\.ts$/,
        loader: '@ngtools/webpack',
      }
    ]
  },

  plugins: [
    new AotPlugin({
      tsConfigPath: 'path/to/tsconfig.json',
      entryModule: 'path/to/app.module#AppModule'
    })
  ]
}
```

Here `@ngtools/webpack` replaces other typescript loader like `ts-loader` or `awesome-typescript-loader`. It works with `AotPlugin` together to enable AoT compilation. More details can be found here.

(Note, for project generated by `angular-cli`, turning on AoT can be simple as `ng build --aot`, but since angular-cli does not allow customized webpack configuration for complex use cases, it may be insufficient.)