* **[Bootstrapping Angular](https://app.pluralsight.com/player?course=angular-2-first-look&author=john-papa&name=angular-2-first-look-m2&clip=5&mode=live" \t "psplayer)**
* [Now you might've seen in that previous example we had to **bootstrap** **Angular**.](https://app.pluralsight.com/player?course=angular-2-first-look&author=john-papa&name=angular-2-first-look-m2&clip=5&mode=live&start=1.431) [This is nothing new. We did it in **Angular** 1 as well. In Angular 1, the way we **bootstrapped** it was we put](https://app.pluralsight.com/player?course=angular-2-first-look&author=john-papa&name=angular-2-first-look-m2&clip=5&mode=live&start=5.931) this **ng-app** directive right on our html tag or in a body tag or wherever you wanted to to start up your app, and then we told it the name of the module that we create in Angular 1.
* Alternatively, there was a second way to **bootstrap** your **app**, which was **manual** bootstrapping in Angular 1, and that allowed us to do things through code so we could inject code before the Angular app ran.
* In Angular 2, the way we **bootstrap** is through code, so goodbye to **ng-app**, and let's say **hello** to **bootstrap**, which we pull out of this **module**, **angular2/platform/browser**, and then we call the bootstrap function, and we give it the starting **component**. That **component** is **basically** the **parent** **component** for the **entire application**.

## Bootstrapping in main.ts

We launch the application by bootstrapping the AppModule in the main.ts file.

Angular offers a variety of bootstrapping options, targeting multiple platforms. In this page we consider two options, both targeting the browser.

### Dynamic bootstrapping with the Just-in-time (JIT) compiler

In the first, dynamic option, the [Angular compiler](https://angular.io/docs/ts/latest/cookbook/ngmodule-faq.html#q-angular-compiler) compiles the application in the browser and then launches the app.

// The browser platform with a compiler

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

// The app module

import { AppModule } from './app.module';

// Compile and launch the module

platformBrowserDynamic().bootstrapModule(AppModule);

**Static bootstrapping with the Ahead-Of-time (AOT) compiler**

Consider the static alternative which can produce a much smaller application that launches faster, especially on mobile devices and high latency networks.

In the *static* option, the Angular compiler runs ahead-of-time as part of the build process, producing a collection of class factories in their own files. Among them is the AppModuleNgFactory.

The syntax for bootstrapping the pre-compiled AppModuleNgFactory is similar to the dynamic version that bootstraps the AppModule class.

// The browser platform without a compiler

import { platformBrowser } from '@angular/platform-browser';

// The app module factory produced by the static offline compiler

import { AppModuleNgFactory } from './app.module.ngfactory';

// Launch with the app module factory.

platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);

Because the entire application was pre-compiled, we don't ship the Angular Compiler to the browser and we don't compile in the browser.

The application code downloaded to the browser is much smaller than the dynamic equivalent and it is ready to execute immediately. The performance boost can be significant.

Both the JIT and AOT compilers generate an AppModuleNgFactory class from the same AppModule source code. The JIT compiler creates that factory class on the fly, in memory, in the browser. The AOT compiler outputs the factory to a physical file that we're importing here in the static version of main.ts.

In general, the AppModule should neither know nor care how it is bootstrapped.

References:

https://angular.io/docs/ts/latest/guide/ngmodule.html