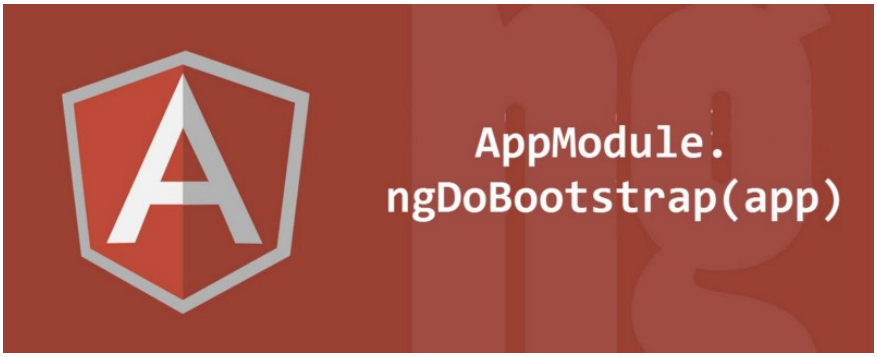
How to manually bootstrap an Angular application

Max Koretskyi aka Wizard Follow
Jul 18, 2017 · 4 min read



• • •

Official Angular documentation states that to bootstrap an application you have to put the following in the `main.ts` file:

```
platformBrowserDynamic().bootstrapModule(AppModule);
```

The first part of the statement `platformBrowserDynamic()` creates a platform. Angular docs describe the platform as:

> *the entry point for Angular on a web page. Each page has exactly one platform, and services (such as reflection) which are common to every Angular application running on the page are bound in its scope.*

Angular also has a concept of the running application instance that you can usually inject using `ApplicationRef` token. There potentially can be many applications on one platform. Each application is created from the module using `bootstrapModule` method. This is exactly the method that is used in `main.ts`. So the statement shown in the docs first creates a platform and then the application instance.

When the application is being created Angular checks the `bootstrap` property of the module used to bootstrap the application (`AppModule`):

```
@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

This property usually references the component you want to bootstrap the application with. Then Angular finds the element that is the selector of bootstrapped component in the DOM and initializes the component.

The above process implies that you know what component you want to bootstrap the application with. But imagine a situation when the component to bootstrap the application is defined by the server during runtime. How can you bootstrap the application later when you get this information? As it turns out it's a pretty straightforward process.

• • •

NgDoBootstrap

Imagine we have two components `A` component and `B` component. We will decide during runtime which one should be used in the application. Let's define these two components:

```
import { Component } from '@angular/core';

@Component({
  selector: 'a-comp',
  template: `<span>I am A component</span>`
})
export class AComponent {}

@Component({
  selector: 'b-comp',
  template: `<span>I am B component</span>`
})
export class BComponent {}
```

And we register them in the `AppModule`:

```
@NgModule({
  imports: [BrowserModule],
  declarations: [AComponent, BComponent],
  entryComponents: [AComponent, BComponent]
})
export class AppModule {}
```

The important thing here is that we **don't** register them in the `bootstrap` property since we will be bootstrapping them manually. Also we should register them in the `entryComponents` since we want the compiler to create factories for them. Angular automatically adds all components specified in the `bootstrap` property to entry components that is why you usually don't add the root component to the `entryComponents`.

Also, since we don't know whether `A` or `B` component will be used we don't specify there selectors in the `index.html`, so it now it looks like this:

```
<body>
  <h1 id="status">
     Loading AppComponent content here ...
  </h1>
</body>
```

Now, if you run the application now you will get the following error:

> *The module AppModule was bootstrapped, but it does not declare "@NgModule.bootstrap" components nor a "ngDoBootstrap" method. Please define one of these*

So basically Angular complains that we didn't specify what component should be used for bootstrapping. And we don't know beforehand. We will be bootstrapping the app manually later and to do that we need to add `ngDoBoostrap` method to the `AppModule` class:

```
export class AppModule {
  ngDoBootstrap(app) {  }
}
```

Angular passes the reference to the running application in the form of ApplicationRef to this method. Later, when we will be ready to bootstrap the application we will use `bootstrap` method of the `ApplicationRef` to initialize the root component.

Let's define the custom method `bootstrapRootComponent` that will be responsible for bootstrapping the root component when it becomes available:

```
// app - reference to the running application
(ApplicationRef)
// name - name (selector) of the component to bootstrap

function bootstrapRootComponent(app, name) {

  // define the possible bootstrap components
  // with their selectors (html host elements)

  const options = {
    'a-comp': AComponent,
    'b-comp': BComponent
  };

  // obtain reference to the DOM element that shows status
  // and change the status to `Loaded`

  const statusElement = document.querySelector('#status');
  statusElement.textContent = 'Loaded';

  // create DOM element for the component being bootstrapped
  // and add it to the DOM

  const componentElement = document.createElement(name);
  document.body.appendChild(componentElement);

  // bootstrap the application with the selected component

  const component = options[name];
  app.bootstrap(component);
}
```

It takes the reference to the `ApplicationRef` and the name of the component to bootstrap. Also, we defined the `options` map with all possible bootstrap components. The component selector acts as a

key. We will use it to lookup a component class when we get the required information from the server.

I've also created a mock `fetch` function that emulates `HTTP` request to the server and returns `b-comp` selector within `2` seconds:

```
function fetch(url) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('b-comp');
    }, 2000);
  });
}
```

So now that we have our `bootstrap` function that will bootstrap the root component let's use it in the `ngDoBootstrap` method of the module:

```
export class AppModule {
  ngDoBootstrap(app) {
    fetch('url/to/fetch/component/name')
      .then((name)=>{ this.bootstrapRootComponent(app,
name)});
  }
}
```

And that's it. Here is the stackblitz example that demonstrates the solution.

• • •

Does it work with AOT?

Yes, sure it does. You just have to precompile all your components and use the factories when bootstrapping the application:

```
import {AComponentNgFactory, BComponentNgFactory} from
'./components.ngfactory.ts';
```

```
@NgModule({
  imports: [BrowserModule],
  declarations: [AComponent, BComponent]
})
```

```
export class AppModule {
  ngDoBootstrap(app) {
    fetch('url/to/fetch/component/name')
      .then((name)=>{ this.bootstrapRootComponent(app,
name)});
  }
  bootstrapRootComponent(app, name) {
    const options = {
      'a-comp': AComponentNgFactory,
      'b-comp': BComponentNgFactory
    };
```

Note that we don't need to specify components in the `entryComponents` since we already have the factories and don't need to compile them.

. . .

Thanks for reading! If you liked this article, hit that clap button below 👏. It means a lot to me and it helps other people see the story.

For more insights follow me on Twitter and on Medium.