JavaScript modules vs. NgModules

Contents

JavaScript modules and NgModules can help you modularize your code, but they are very different. Angular apps rely on both kinds of modules.

JavaScript modules: Files containing code

A JavaScript module is an individual file with JavaScript code, usually containing a class or a library of functions for a specific purpose within your app. JavaScript modules let you spread your work across multiple files.

> To learn more about JavaScript modules, see ES6 In Depth: Modules. For the module specification, see the 6th Edition of the ECMAScript standard.

To make the code in a JavaScript module available to other modules, use an `export` statement at the end of the relevant code in the module, such as the following:

```
export class AppComponent { ... }
```

When you need that module's code in another module, use an `import` statement as follows:

```
import { AppComponent } from './app.component';
```

Each module has its own top-level scope. In other words, top-level variables and functions in a module are not seen in other scripts or modules. Each module provides a namespace for identifiers to prevent them from clashing with identifiers in other modules. With multiple modules, you can prevent accidental global variables by creating a single global namespace and adding sub-modules to it.

The Angular framework itself is loaded as a set of JavaScript modules.

NgModules: Classes with metadata for compiling

An NgModule is a class marked by the `@NgModule` decorator with a metadata object that describes how that particular part of the app fits together with the other parts. NgModules are specific to Angular. While classes with an `@NgModule` decorator are by convention kept in their own files, they differ from JavaScript modules because they include this metadata.

The `@NgModule` metadata plays an important role in guiding the Angular compilation process that converts the app code you write into highly performant JavaScript code. The metadata describes how to compile a component's template and how to create an injector at runtime. It identifies the NgModule's components, directives, and pipes, and makes some of them public through the `exports` property so that external components can use them. You can also use an NgModule to add providers for services, so that the services are available elsewhere in your app.

Rather than defining all member classes in one giant file as a JavaScript module, declare which components, directives, and pipes belong to the NgModule in the `@NgModule.declarations` list. These classes are called declarables. An NgModule can export only the declarable classes it owns or imports from other NgModules. It doesn't declare or export any other kind of class. Declarables are the only classes that matter to the Angular compilation process.

For a complete description of the NgModule metadata properties, see Using the NgModule metadata.

An example that uses both

The root NgModule `AppModule` generated by the Angular CLI for a new app project demonstrates how you use both kinds of modules:

src/app/app.module.ts (default AppModule)

```
// imports
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

// @NgModule decorator with its metadata
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

The root NgModule starts with `import` statements to import JavaScript modules. It then configures the `@NgModule` with the following arrays:

- `declarations`: The components, directives, and pipes that belong to the NgModule. A new app project's root NgModule has only one component, called `AppComponent`.
- `imports`: Other NgModules you are using, so that you can use their declarables. The newly generated root NgModule imports `BrowserModule` in order to use browser-specific services such as DOM rendering, sanitization, and location.

- `providers`: Providers of services that components in other NgModules can use. There are no providers in a newly generated root NgModule.
- `bootstrap`: The entry component that Angular creates and inserts into the `index.html` host web page, thereby bootstrapping the app. This entry component, `AppComponent`, appears in both the `declarations` and the `bootstrap` arrays.

Next steps

- For more about NgModules, see Organizing your app with NgModules.
- To learn more about the root NgModule, see Launching an app with a root NgModule.
- To learn about frequently used Angular NgModules and how to import them into your app, see Frequently-used modules.

Version 10.0.10-local+sha.b32126c335.