Using published libraries

Contents ›

•••

When building Angular applications you can take advantage of sophisticated first-party libraries, such as Angular Material, as well as rich ecosystem of third-party libraries. See the Angular Resources page for links to the most popular ones.

Installing libraries

Libraries are published as npm packages, usually together with schematics that integrate them with the Angular CLI. To integrate reusable library code into an application, you need to install the package and import the provided functionality where you will use it. For most published Angular libraries, you can use the Angular CLI `ng add <lib_name>` command.

The `ng add` command uses a package manager such as npm or yarn to install the library package, and invokes schematics that are included in the package to other scaffolding within the project code, such as adding import statements, fonts, themes, and so on.

A published library typically provides a README or other documentation on how to add that lib to your app. For an example, see Angular Material docs.

Library typings

Library packages often include typings in `.d.ts` files; see examples in `node_modules/@angular/material`. If your library's package does not include typings and your IDE complains, you may need to install the library's associated `@types/<lib_name>` package.

For example, suppose you have a library named `d3`:

```
npm install d3 --save
npm install @types/d3 --save-dev
```

Types defined in a `@types/` package for a library installed into the workspace are automatically added to the TypeScript configuration for the project that uses that library. TypeScript looks for types in the `node_modules/@types` folder by default, so you don't have to add each type package individually.

If a library doesn't have typings available at `@types/`, you can still use it by manually adding typings for it. To do this:

1. Create a `typings.d.ts` file in your `src/` folder. This file is automatically included as global type definition.
2. Add the following code in `src/typings.d.ts`.

```
declare module 'host' {
  export interface Host {
    protocol?: string;
    hostname?: string;
    pathname?: string;
  }
  export function parse(url: string, queryString?: string): Host;
}
```

3. In the component or file that uses the library, add the following code.

```
import * as host from 'host';
const parsedUrl = host.parse('https://angular.io');
console.log(parsedUrl.hostname);
```

You can define more typings as needed.

Updating libraries

Libraries can be updated by their publishers, and also have their own dependencies which need to be kept current. To check for updates to your installed libraries, use the `ng update` command.

Use `ng update <lib_name>` to update individual library versions. The Angular CLI checks the latest published release of the library, and if the latest version is newer than your installed version, downloads it and updates your `package.json` to match the latest version.

When you update Angular to a new version, you need to make sure that any libraries you are using are current. If libraries have interdependencies, you might have to update them in a particular order. See the Angular Update Guide for help.

Adding a library to the runtime global scope

Legacy JavaScript libraries that are not imported into an app can be added to the runtime global scope and loaded as if they were in a script tag. Configure the CLI to do this at build time using the "scripts" and "styles" options of the build target in the CLI configuration file, `angular.json`.

For example, to use the Bootstrap 4 library, first install the library and its dependencies using the npm package manager:

```
npm install jquery --save
npm install popper.js --save
npm install bootstrap --save
```

In the `angular.json` configuration file, add the associated script files to the "scripts" array:

```
"scripts": [
  "node_modules/jquery/dist/jquery.slim.js",
  "node_modules/popper.js/dist/umd/popper.js",
  "node_modules/bootstrap/dist/js/bootstrap.js"
],
```

Add the Bootstrap CSS file to the "styles" array:

```
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.css",
  "src/styles.css"
],
```

Run or restart `ng serve` to see Bootstrap 4 working in your app.

Using runtime-global libraries inside your app

Once you import a library using the "scripts" array, you should **not** import it using an import statement in your TypeScript code (such as `import * as $ from 'jquery';`). If you do, you'll end up with two different copies of the library: one imported as a global library, and one imported as a module. This is especially bad for libraries with plugins, like JQuery, because each copy will have different plugins.

Instead, download typings for your library (`npm install @types/jquery`) and follow the library installation steps. This gives you access to the global variables exposed by that library.

Defining typings for runtime-global libraries

If the global library you need to use does not have global typings, you can declare them manually as `any` in `src/typings.d.ts`. For example:

```
declare var libraryName: any;
```

Some scripts extend other libraries; for instance with JQuery plugins:

```
$('.test').myPlugin();
```

In this case, the installed `@types/jquery` doesn't include `myPlugin`, so you need to add an interface in `src/typings.d.ts`. For example:

```
interface JQuery {
  myPlugin(options?: any): any;
}
```

If don't add the interface for the script-defined extension, your IDE shows an error:

```
[TS][Error] Property 'myPlugin' does not exist on type 'JQuery'
```