Dependency injection in Angular

Dependencies are services or objects that a class needs to perform its function. Dependency injection, or DI, is a design pattern in which a class requests dependencies from external sources rather than creating them.

Angular's DI framework provides dependencies to a class upon instantiation. Use Angular DI to increase flexibility and modularity in your applications.

See the for a working example containing the code snippets in this guide.

Creating an injectable service

To generate a new `HeroService` class in the `src/app/heroes` folder use the following Angular CLI command.

ng generate service heroes/hero

This command creates the following default `HeroService`.

The `@Injectable()` decorator specifies that Angular can use this class in the DI system. The metadata, `providedIn: 'root'`, means that the `HeroService` is visible throughout the application.

Next, to get the hero mock data, add a `getHeroes()` method that returns the heroes from `mock.heroes.ts`.

For clarity and maintainability, it is recommended that you define components and services in separate files.

If you do combine a component and service in the same file, it is important to define the service first, and then the component. If you define the component before the service, Angular returns a run-time null reference error.

{@a injector-config} {@a bootstrap}

Injecting services

Injecting services results in making them visible to a component.

To inject a dependency in a component's `constructor()`, supply a constructor argument with the dependency type. The following example specifies the `HeroService` in the `HeroListComponent` constructor. The type of `heroService` is `HeroService`.

For more information, see Providing dependencies in modules and Hierarchical injectors.

{@a service-needs-service}

Using services in other services

When a service depends on another service, follow the same pattern as injecting into a component. In the following example `HeroService` depends on a `Logger` service to report its activities.

First, import the `Logger` service. Next, inject the `Logger` service in the `HeroService` `constructor()` by specifying `private logger: Logger` within the parentheses.

When you create a class whose `constructor()` has parameters, specify the type and metadata about those parameters so that Angular can inject the correct service.

Here, the `constructor()` specifies a type of `Logger` and stores the instance of `Logger` in a private field called `logger`.

The following code tabs feature the `Logger` service and two versions of `HeroService`. The first version of `HeroService` does not depend on the `Logger` service. The revised second version does depend on `Logger` service.

In this example, the `getHeroes()` method uses the `Logger` service by logging a message when fetching heroes.

What's next

- Dependency providers
- DI tokens and providers
- Dependency Injection in Action