5 JANUARY 2018 / ANGULAR

Angular (2+): Core vs Shared Modules

Attention: The recommendations on `CoreModule` (and modules in general) has changed a bit but this blog post will be left as it was since it still contains pretty good information.

If you're new to Angular, you might get confused between these two commonly used modules. So what do you put in either components?

What goes in `CoreModule`?

Your `CoreModule` contains code that will be used to instantiate your app and load some core functionality.

The clearest and most important use of the `CoreModule` is the place to put your global/HTTP services. The idea is to make sure only one instance of those services will be created across the entire app. The `CoreModule`, by convention, is only included in your entire app once in `AppModule` (only in the `import` property of the `@NgModule()` decorator

services inside it will be only created once in the entire app. This is especially important if you intend to lazy-load your feature modules. Since lazy-loaded modules are loaded on demand (eg when you access the route using the lazy-loaded feature), you could end up creating new instances of supposedly singleton services if you don't put them in `CoreModule`.

Important single use components/classes can also go in the `CoreModule`. Good candidates are global components that go in your main `AppComponent`. For example, if your app has one global header component, you can add the code for `HeaderComponent` in `CoreModule` (and then put the `HeaderComponent` class in the `exports` property of `CoreModule`'s `@NgModule()` decorator so `AppModule` which imports `CoreModule` can use it). This allows you to keep this global component in one spot and make sure there's only have one copy of it across the app.

The `CoreModule` could also be used to export any third party module that is required in the `AppModule`. The idea is to keep `AppModule` as lean as possible. Examples are `BrowserAnimationsModule` (as of time of this writing, this should only be imported once throughout the whole app), `Angulartics2Module`.

What goes in `SharedModule`?

You `SharedModule` similarly contains code that will be used across your app and Feature Modules. But the difference is that you will import this `SharedModule` into the specific Feature Modules as needed. You DO NOT import the `SharedModule` into your main

Common templates components should also go in the `SharedModule`. An example would be a global button component, eg `ButtonComponent`. These template components should be "dumb components" in that they should not expect or interact with any specific form of data. You should define a template, style, an `@Output()` event property, and maybe a `@Input()` string property for text inside the button so you can pass in whatever event you want to fire when a user clicks the button. Now all your buttons can look uniform across the app! Of course, this is an architectural decision left up to your discretion. It will make for easier unit testing and separation-of-concerns to follow the pattern though.

Commonly used pipes (ie filters) and directives should go in your `SharedModule`, too. Prime examples would be custom string/date filters.

In your `SharedModule`'s main file (eg `shared.module.ts`), you might also export the commonly used Angular modules (eg `CommonModule` from `@angular/common` for the `*ngIf` structure directive or the `FormsModule` from `@angular/forms` for the `[(ngModel)]` directive) so they can be easily used across your app, without importing them in every Feature Module. Note, this might be introduce unnecessary loading speed if you don't use those modules/directives a lot; if that's the case, you might want to do it the old fashioned way and import these Angular modules only in the Feature Modules that use them.

Summary

instance components, and export any third-party modules needed in `AppModule`. The `SharedModule` should contain common components/pipes/directives and also export commonly used Angular modules (like `CommonModule` from `@angular/common` for `*ngIf` directive). You should have "dumb components" in `SharedModule`.

For further clarification, reference Angular's tutorials on modules or their style guide.

Jing Chai
Self-taught front end developer with a love of learning!

Read More

— Chai's Blog —
angular

Angular (2+): Use CDN path in angular-cli generated index.html

Angular (2+): What does the @Injectable() decorator do?

ANGULAR

Angular (2+): What does the @Injectable() decorator do?

Angular 2+'s services require an @Injector() decorator. What does it do? Maybe nothing at all..

See all 2 posts →

JING CHAI

DATABASE

MySQL: Columns Design Decisions

Find out about MySQL database tables' column design decisions for text vs number vs date/time data and also considerations for LENGTH and NOT NULL.

JING CHAI