Angular Modules

WWCode Angular Study Group

7-Nov-2020

What is Angular Module?

- NgModules - mechanism to group components, directives, pipes and services that are related, in such a way that can be combined with other modules to create an application
- Another analogy to understand Angular modules is classes. In a class, we can define public or private methods.
- Angular libraries are NgModules, such as FormsModule, HttpClientModule, and RouterModule. Many third-party libraries are available as NgModules such as Material Design, Ionic, and AngularFire2.

Root Modules (App Module) and Feature Modules

- **app.module.ts** is the root module, every Angular app has this
- Most apps have many **feature modules**
- The hierarchical structure of views is a key factor in the way Angular detects and responds to changes in the DOM and app data.

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

@NgModule metadata

An NgModule is defined by a class decorated with `@NgModule()`.
The `@NgModule()` decorator is a function that takes a single metadata object, whose properties describe the module. The most important properties are as follows.

- **`declarations`**: The components, directives, and pipes that belong to this NgModule.
- **`exports`**: The subset of declarations that should be visible and usable in the component templates of other NgModules.
- **`imports`**: Other modules whose exported classes are needed by component templates declared in this NgModule.
- **`providers`**: Creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level.)
- **`bootstrap`**: The main application view, called the root component, which hosts all other app views. Only the root NgModule should set the bootstrap property.

Frequently used Modules

	Import it from	Why you use it
	`@angular/platform-browser`	When you want to run your app in a browser
	`@angular/common`	When you want to use `NgIf`, `NgFor`
	`@angular/forms`	When you want to build template driven forms (includes `NgM`
ıle	`@angular/forms`	When you want to build reactive forms
	`@angular/router`	When you want to use `RouterLink`, `.forRoot()`, and `.forChild()`
	`@angular/common/http`	When you want to talk to a server

Feature modules

- Feature modules are NgModules for the purpose of organizing code.

```
ng generate module CustomerDashboard
```

```
ng generate component customer-dashboard/CustomerDashboard
```

Types of Feature modules

All apps start by bootstrapping a root NgModule. You can organize your other NgModules any way you wish.

- Domain: A domain NgModule is organized around a feature, business domain, or user experience.
- Routed: The top component of the NgModule acts as the destination of a router navigation route.
- Routing: A routing NgModule provides the routing configuration for another NgModule.
- Service: A service NgModule provides utility services such as data access and messaging.
- Widget: A widget NgModule makes a component, directive, or pipe available to other NgModules.
- Shared: A shared NgModule makes a set of components, directives, and pipes available to other NgModules.

Types of Feature modules

NgModule	Declarations	Providers	Exports	Imported by
Domain	Yes	Rare	Top component	Another domain, AppModule
Routed	Yes	Rare	No	None
Routing	No	Yes (Guards)	RouterModule	Another domain (for routing)
Service	No	Yes	No	AppModule
Widget	Yes	Rare	Yes	Another domain
Shared	Yes	No	Yes	Another domain

Shared module

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { CustomerComponent } from './customer.component';
import { NewItemDirective } from './new-item.directive';
import { OrdersPipe } from './orders.pipe';

@NgModule({
 imports:      [ CommonModule ],
 declarations: [ CustomerComponent, NewItemDirective, OrdersPipe ],
 exports:      [ CustomerComponent, NewItemDirective, OrdersPipe,
                 CommonModule, FormsModule ]
})
export class SharedModule { }
```

- You can put commonly used directives, pipes, and components into one module and then import just that module wherever you need it in other parts of your app.

Note the following:

- It imports the CommonModule because the module's component needs common directives.
- It declares and exports the utility pipe, directive, and component classes.
- It re-exports the CommonModule and FormsModule.

Shared Module

- By re-exporting CommonModule and FormsModule, any other module that imports this SharedModule, gets access to directives like NgIf and NgFor from CommonModule and can bind to component properties with [(ngModel)], a directive in the FormsModule.

Resources and credits

- https://angular.io/guide/ngmodules
- Sample code: https://angular.io/generated/zips/ngmodules/ngmodules.zip
- https://angular-2-training-book.rangle.io/modules/introduction (Note that this is Angular 2, but the explanations are easier to understand
- Image used: https://commons.wikimedia.org/wiki/File:Lego_dublo_arto_alanenpaa_5.JPG
- Learn more about Lazy Loading a feature module: https://angular.io/guide/lazy-loading-ngmodules