Angular 16 : Online Class

Feature Modules

By: Sahosoft Solutions

Presented by : Chandan Kumar

Feature Modules

A feature module is an ordinary Angular module for all intents and purposes, except the fact that isn't the root module. Basically - it's just an additional class with the @NgModule decorator and registered metadata.

Feature modules are **NgModules** to organize code. As your app scales, you can organize code relevant for a specific feature. Feature Modules helps us to apply the clear boundaries for your features. With Feature Modules, you can keep code related to the particular functionality or feature separate from other code.

The main aim for feature modules is delimiting the functionality that focuses on particular internal business inside a dedicated module, in order to achieve modularity. In addition, it restricts the responsibilities of the root module and assists to keep it thin.

The feature modules are modules that goal of organizing an application code. It also helps you partition the app into focused areas when you can do everything within the root module.

Difference b/w Feature Modules and Root Modules

The feature module is an organizational best practice, as opposed to the concept of the core Angular API. The feature module delivers a cohesive set of functionality focused on a specific application need such as the user workflow, routing, or forms. While you can do everything within the root module, feature modules help you partition the app into focused areas.

Root Modules	Feature Modules
Required to boot the root module to run the Application.	Required to import the feature module to extend the Application.
It is very straightforward and visible.	It can hide or expose the module implementation from the other modules.

The feature modules contain the functionality on Application business domain, Workflow or provide the collection of the related utilities. The feature modules help us to classify the Application into the various areas.

Angular Module Loading

An effective loading strategy is key to a successful single-page application.

A module can be loaded **eagerly**, **lazily** and **preloaded**.

- ✓ **Eager loading** is loading modules before application starts.
- ✓ **Lazy loading** is loading modules on demand.
- ✓ **Preloading** is loading modules in background just after application starts.

The application module i.e. AppModule is loaded eagerly before application starts. But the feature modules can be loaded either eagerly or lazily or preloaded.

Eager Loading

- ✓ To load a feature module eagerly we need to import it into application module using imports metadata of @NgModule decorator. Eager loading is useful in small size applications.

- ✓ In eager loading, all the feature modules will be loaded before the application starts. Hence the subsequent request to the application will be faster.

1. In eager loading module, feature modules are loaded before application start on the first hit. To load a feature module eagerly, we need to import that module in application module i.e. AppModule using imports metadata of @NgModule decorator.
2. When a module is loaded, it loads all the imported modules, configured components, services, custom pipes etc.
3. Modules are loaded in the order they are configured in imports metadata.
4. Eager loading is good for small applications because at the first hit of the application all the modules are loaded and all the required dependencies are resolved. Now the subsequent access to the application will be faster.
5. Eager loading means the slowest load times for this reason. When you have a small app, this is okay because the total size will not be large enough to see a benefit from the other two loading strategies.

Lazy Loading

Lazy loading is loading modules on demand.

Lazy loading modules help us decrease the startup time. With lazy loading our application does not need to load everything at once, it only needs to load what the user expects to see when the app first loads. Modules that are lazily loaded will only be loaded when the user navigates to their routes

To load a feature module lazily we need to load it using **loadChildren** property in **route configuration** and that **feature module must not be imported in application module**.

Lazy loading is useful when the application size is growing. In lazy loading, feature module will be loaded on demand and hence application start will be faster.

a. When the modules are loaded on-demand, then it is called lazy loading. It is loaded by using **loadChildren property** in route configuration.

These modules must not be imported in application module otherwise they will be eagerly loaded.

Lazy Loading

b. In route configuration loadChildren property is used as following.

```
const routes: Routes = [
   {
            path: 'country',
            loadChildren: 'app/country/country.module#CountryModule'
   },
   ------
];
```

c. If the application size is growing and there are many feature modules then loading all feature modules eagerly will make application slow.
What we can do, is we can load a feature module when it is requested. Such type of module loading is called lazy loading.

Preloading

To preload a feature module we need to load it using loadChildren property and configure preloadingStrategy property with RouterModule.forRoot. That feature module must not be imported in application module.

When we assign Angular **PreloadAllModules** strategy to **preloadingStrategy** property, then all feature modules configured with loadChildren, are preloaded. To preload selective modules, we need to use custom preloading strategy.

We should preload only those features which will be visited by users just after application start and rest feature modules can be loaded lazily. In this way we can improve the performance of our bigger size application.

1. In preloading, feature modules are loaded in background asynchronously. In preloading, modules start loading just after application starts.
2. When we hit the application, first AppModule and modules imported by it, will be loaded eagerly. Just after that modules configured for preloading is loaded asynchronously.
3. Preloading is useful to load those features which are in high probability to be visited by user just after loading the application.

Preloading

4. To configure preloading, angular provides preloadingStrategy property which is used with RouterModule.forRootin routing module. Find the code snippet.

```
@NgModule({
  imports: [
      RouterModule.forRoot(routes,
      {
        preloadingStrategy: PreloadAllModules
      })
  ],
  ------
})
export class AppRoutingModule { }
```

5. To configure preloading features modules, first we will configure them for lazy loading and then using Angular in-built PreloadAllModules strategy, we enable to load all lazy loading into preloading modules.

6. Using PreloadAllModules strategy, all modules configured by loadChildren property will be preloaded. The modules configured by loadChildren property will be either lazily loaded or preloaded but not both. To preload only selective modules, we need to use custom preloading strategy.

7. Once we configure PreloadAllModules strategy, then after eager loading modules, Angular searches for modules applicable for preloading. The modules configured by loadChildren will be applicable for preloading. We will take care that these feature modules are not imported in application module i.e. AppModule.

PreloadAllModules and NoPreloading

a. PreloadAllModules: Strategy that preloads all modules configured by loadChildren in application routing module as quickly as possible. We use it as following.

```
RouterModule.forRoot(routes,
  {
    preloadingStrategy: PreloadAllModules
  })
```

b. NoPreloading: Strategy that does not preload any module. This strategy is enabled by default. But if we want to use it, we can use it as following.

```
RouterModule.forRoot(routes,
  {
    preloadingStrategy: NoPreloading
  })
```

Custom Preloading Strategy

Preloading is loading modules in background just after application starts. In preloading, modules are loaded asynchronously. Angular provides in-built PreloadAllModules strategy that loads all feature modules configured with loadChildren in application routing module as quickly as possible.

Custom preloading strategies allow to preload selective modules. We can also customize to preload modules after a certain delay once application starts. To create a custom preloading strategy, Angular provides PreloadingStrategy class with a preload method.

We need to create a service by implementing PreloadingStrategy and then overriding its preload method. To enable custom preloading strategies, we need to configure our preloading strategy service with RouterModule.forRoot using preloadingStrategy property. We will also configure our preloading strategy service into providers metadata in application routing module.

Angular provides PreloadingStrategy class using which we can create custom preloading strategy. Find the structure of PreloadingStrategy from Angular Doc.

```
class PreloadingStrategy {
  preload(route: Route, fn: () => Observable<any>): Observable<any>
}
```

Custom PreLoading Strategy with Delay

Custom preloading strategy using which only selective feature modules will be preloaded with a certain delay.

```
 if (route.data['delay']) {
        return timer(5000).mergeMap(() => load());
      }
```

If a feature module has been given delay as true to data property in route configuration, then that module will be preloaded after 5 seconds once the application started.

Angular 16 : Online Class

Creating a Routing Module

By: Sahosoft Solutions

Presented by : Chandan Kumar

Creating a Routing Module

By using the command line parameter "--routing", we're able to create a routing module in the Angular application. So, by using any of the following commands, we can create a routing module in Angular with the default routing set up.

When you generate a module, you can use the **--routing** option like **ng g module my-module --routing** to create a separate file my-module-routing.module.ts to store the module routes.
The file includes an empty Routes object that you can fill with routes to different components and/or modules.
Example : create module file
ng g m modulename

ng g m modulename --routing
g-> generate
m-> module

ng g m modulename --routing --flat --module=app
--flat puts the file in src/app instead of its own folder.
--module=app tells the CLI to register it in the imports array of the AppModule

ng g m modulename --routing –-flat
ng generate module modulename --flat --module=anymodule

Create Component/ Service without spec.ts

Create Component without spec.ts

```
ng g c student
ng g c student --spec=false
Option "spec" is deprecated: Use "skipTests" instead.
ng g c student --spec false
Option "spec" is deprecated: Use "skipTests" instead.
ng g c student --nospec
Option "spec" is deprecated: Use "skipTests" instead.
ng g s student --skipTests
Option "spec" is deprecated: Use "skipTests" instead.
ng g s student --skip-tests
```

Create Service without spec.ts

```
ng g s my-service1
ng g s student --spec=false
Option "spec" is deprecated: Use "skipTests" instead.
ng g s student --spec false
Option "spec" is deprecated: Use "skipTests" instead.
ng g s student --nospec
Option "spec" is deprecated: Use "skipTests" instead.
ng g s my-service --skipTests
```

Create new component/directive/class/guard

Create new component/directive/class/guard

ng generate component <name> [options]

OPTION	DESCRIPTION
--skipTests=true|false	When true, does not create "spec.ts" test files for the new **component**/directive/**class**/guard. Default: false
--spec=true|false	***Deprecated***: Use "skipTests" instead. When true (the default), generates a "spec.ts" test file for the new component. Default: true