Preloading Strategy

@coder_aishya

What is Angular Preloading Strategy?

Preloading in Angular means **loading the Lazy loaded Modules in the background asynchronously**, while user is interacting with the app. This will help **boost up the loading time** of the app as the user do not have to wait for the module to be downloaded

Available Preloading strategies

- **Build-in preloading strategies** — NoPreloading (default) & PreloadAllModules.
- **Custom preloading strategies** — Preload after some time, preload based on network quality, load required modules first, frequently used second, and others lazy load/last.

@coder_aishya

How to Enable Preloading

Note:

You **need to configure lazy loading before using any preloading** strategies. Check my previous post to know how to configure Lazy Loading

NoPreloading (default)

This will **disables all the preloading**. This is default behavior i.e. if you do not specify the preloadingStrategy, then the angular assumes you do not want preloading

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

@coder_aishya

PreloadAllModules

This strategy will preload all the lazy loaded modules in the background

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import {Routes, RouterModule, PreloadAllModules} from
'@angular/router';
const routes: Routes = [
  {
    path: 'about',
    loadChildren: () =>import('./about/about.module')
    .then(m => m.AboutModule)
  },
  {
    path: 'users',
    loadChildren: () =>import('./users/users.module')
    .then(m => m.UsersModule)
  },
  {
    path: '', redirectTo: '', pathMatch: 'full'}
  ];
@NgModule({
  imports: [
    RouterModule.forRoot(routes, {
        preloadingStrategy: PreloadAllModules
  })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

@coder_aishya

Custom preloading strategies

If the number of modules in your web application is large, preloading all the modules might not be the best solution. In that case, we can configure custom preloading strategies.

Implement Custom preloading strategies

```
abstract preload(route: Route,
    fn: () => Observable<any>): Observable<any>
```

Parameters :

route Route

fn () => Observable

Returns :

Observable<any>

@coder_aishya

Steps to create custom preload

- First **create a class**, which implements the built in PreloadingStrategy class
- The **class must implement the method preload()**. In this method, we determine whether to preload the module or not.
- The **first parameter is the active Route**. We can use this to extract the information about the route, which is being is loaded.
- The **second parameter is Observable function**, which we need to return if we want to preload this module. We can return Observable of null, if we do not wish to preload the module.

Preloading selected modules Example

Frequently used modules can be preloaded to reduce the loading time. Follow the 80–20 Rule!

app-routing.module.ts

```
import {NgModule} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';
import {CustomPreloadingStrategyService} from './custom-
preloading-strategy.service';
const routes: Routes = [
  {
    path: 'about', data: {preload: true},
    loadChildren: () => import('./about/about.module').
                    then(m => m.AboutModule)
  },
  {
    path: 'users',
    loadChildren() => import('./users/users.module').
                    then(m => m.UsersModule)
 },
  { path: '', redirectTo: '', pathMatch: 'full'}
];
@NgModule({
  imports: [RouterModule.forRoot(routes,          {
     preloadingStrategy: CustomPreloadingStrategyService}
  )],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

@coder_aishya

custom-preloading-strategy.service.ts

```
import {Injectable} from '@angular/core';
import {PreloadingStrategy, Route} from '@angular/router';
import {Observable, of} from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class CustomPreloadingStrategyService implements
PreloadingStrategy {
preload(route: Route, fn: () => Observable<any>):
Observable<any> {
    if (route.data && route.data.preload) {
      return fn();
    }
    return of(null);
  }
}
```

- If you're using higher Angular versions, you can use **canLoad router guard**, which has the same behavior as canActivate guard, is now available in angular version 10. Any routes with a canLoad guard won't be preloaded.
- **If you want to protect your route but also preload it, then use canActivate instead of canLoad.**

When to use which loading strategy ?

In an Angular App, you might **need to use combination of different loading strategies** based on App requirements. So, here's a good approach which you can follow

- **Eagerly Load the modules required at startup**, for example, authentication module, core module shared module etc
- **Preload all frequently used modules**, maybe after some delay
- **Lazy load remaining modules**

@coder_aishya

Follow me for more

aishwarya-dhuri

coder_aishya