**Angular Dependency Injection**

Dependency Injection (DI) is a fundamental concept in Angular that allows you to inject dependencies into components and services. It helps in creating more modular, testable, and maintainable code. In this chapter, we'll explore how dependency injection works in Angular.

**Basic Concept**

Dependency Injection in Angular is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services. Angular uses an injector to maintain a container of service instances that it has created.

**Creating a Service**

Let's start by creating a simple service:

// data.service.ts

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class DataService {

getData(): string {

return 'Data from service';

}

}

The @Injectable decorator marks the class as one that participates in the dependency injection system. The providedIn: 'root' option makes the service available throughout the app.

**Using the Service in a Component**

Now, let's use this service in a component:

// app.component.ts

import { Component } from '@angular/core';

import { DataService } from './data.service';

@Component({

selector: 'app-root',

template: `

<h1>{{ title }}</h1>

<p>{{ data }}</p>

`,

standalone: true

})

export class AppComponent {

title = 'Dependency Injection Demo';

data: string;

constructor(private dataService: DataService) {

this.data = this.dataService.getData();

}

}

In this example, we're injecting the DataService into the component's constructor. Angular's DI system takes care of creating and providing an instance of DataService.

Expected output:

Dependency Injection Demo

Data from service

**Configuring Providers**

While providedIn: 'root' is convenient for app-wide services, sometimes you might want to provide services at a more specific level.

**Component-level Providers**

You can provide a service at the component level:

// special-list.component.ts

import { Component } from '@angular/core';

@Injectable()

class SpecialListService {

getItems(): string[] {

return ['Special Item 1', 'Special Item 2', 'Special Item 3'];

}

}

@Component({

selector: 'app-special-list',

template: `

<h2>Special List</h2>

<ul>

<li \*ngFor="let item of items">{{ item }}</li>

</ul>

`,

providers: [SpecialListService],

standalone: true,

imports: [NgFor]

})

export class SpecialListComponent {

items: string[];

constructor(specialListService: SpecialListService) {

this.items = specialListService.getItems();

}

}

In this case, a new instance of SpecialListService will be created for each instance of SpecialListComponent.

Expected output:

Special List

- Special Item 1

- Special Item 2

- Special Item 3

**Injection Tokens**

For values that aren't classes, you can use an InjectionToken:

// app.config.ts

import { InjectionToken } from '@angular/core';

export interface AppConfig {

apiUrl: string;

theme: string;

}

export const APP\_CONFIG = new InjectionToken<AppConfig>('app.config');

// app.component.ts

import { Component, Inject } from '@angular/core';

import { APP\_CONFIG, AppConfig } from './app.config';

@Component({

selector: 'app-root',

template: `

<h1>App Configuration</h1>

<p>API URL: {{ config.apiUrl }}</p>

<p>Theme: {{ config.theme }}</p>

`,

standalone: true

})

export class AppComponent {

constructor(@Inject(APP\_CONFIG) public config: AppConfig) {}

}

// main.ts

import { bootstrapApplication } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { APP\_CONFIG } from './app.config';

bootstrapApplication(AppComponent, {

providers: [

{ provide: APP\_CONFIG, useValue: { apiUrl: 'https://api.example.com', theme: 'dark' } }

]

}).catch(err => console.error(err));

Expected output:

App Configuration

API URL: https://api.example.com

Theme: dark

**Using inject Function (New in Angular 14+)**

Angular 14 introduced the inject function, which provides a more flexible way to inject dependencies:

// user.service.ts

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Injectable({

providedIn: 'root'

})

export class UserService {

constructor(private http: HttpClient) {}

getUsers() {

return this.http.get('https://jsonplaceholder.typicode.com/users');

}

}

// user-list.component.ts

import { Component, OnInit } from '@angular/core';

import { UserService } from './user.service';

import { inject } from '@angular/core';

@Component({

selector: 'app-user-list',

template: `

<h2>User List</h2>

<ul>

<li \*ngFor="let user of users">{{ user.name }}</li>

</ul>

`,

standalone: true,

imports: [NgFor]

})

export class UserListComponent implements OnInit {

private userService = inject(UserService);

users: any[] = [];

ngOnInit() {

this.userService.getUsers().subscribe((users: any) => {

this.users = users;

});

}

}

The inject function allows you to inject dependencies without using constructor injection, which can be particularly useful in certain scenarios like standalone components or when you need conditional injection.

Expected output (assuming the API returns data):

User List

- Leanne Graham

- Ervin Howell

- Clementine Bauch

...

**Conclusion**

Dependency Injection is a powerful feature in Angular that promotes loose coupling between classes and their dependencies. It allows for more modular and testable code by centralizing dependency creation and lifecycle management. With Angular, you have multiple ways to leverage DI, from traditional constructor injection to the newer inject function, giving you flexibility in how you structure your application.