

Angular

Inyección de dependencias

CertiDevs

Índice de contenidos

1. Inyección de dependencias (DI)	1
2. Proveedores	1
3. Inyectar dependencias	1
3.1. Resolver dependencias opcionales y múltiples	2

1. Inyección de dependencias (DI)

La **inyección de dependencias (DI)** es un *patrón de diseño* en el que una clase recibe sus dependencias desde el exterior, en lugar de crearlas internamente.

Angular utiliza la inyección de dependencias para mejorar la modularidad, la reutilización de código y la separación de preocupaciones y responsabilidades en las aplicaciones.

2. Proveedores

En Angular, las dependencias son gestionadas por **proveedores**.

Un **proveedor** es un objeto que puede crear o devolver una instancia de una dependencia. Los proveedores pueden ser registrados a distintos niveles:

- a nivel **global** de la aplicación
- a nivel de **módulo**
- a nivel de **componente**

Por ejemplo, supongamos que tenemos un servicio **UserService** que queremos utilizar en nuestra aplicación:

```
export class UserService {  
    // ...  
}
```

Para registrar el **servicio** como **proveedor**, se puede agregar la anotación **@Injectable()** al servicio y especificar la opción **providedIn**:

```
import { Injectable } from '@angular/core';  
  
@Injectable({  
    providedIn: 'root'  
})  
export class UserService {  
    // ...  
}
```

En este caso, el servicio se registrará como un proveedor **singleton** en la **aplicación**, lo que significa que Angular creará **una única instancia** del servicio para toda la aplicación.

3. Inyectar dependencias

Para **inyectar una dependencia** en una clase (como un componente u otro servicio), se debe agregar un parámetro al constructor de la clase con el tipo de la dependencia y la anotación

`@Inject()`.

Angular se encargará automáticamente de crear o proporcionar la instancia adecuada de la dependencia.

Por ejemplo, para inyectar el servicio `UserService` en un **componente**:

```
import { Component } from '@angular/core';
import { UserService } from '../user.service';

@Component({
  selector: 'app-user-list',
  templateUrl: '../user-list.component.html'
})
export class UserListComponent {

  constructor(private userService: UserService) {}
}
```

En este ejemplo, Angular inyectará automáticamente una instancia del servicio `UserService` en el constructor del componente `UserListComponent`.

El componente puede utilizar el servicio para realizar operaciones como obtener datos de usuario o realizar acciones específicas.

3.1. Resolver dependencias opcionales y múltiples

En algunos casos, es posible que una **dependencia** sea **opcional** o que haya **múltiples implementaciones** de una dependencia disponibles.

Angular proporciona varias anotaciones y opciones para gestionar estos casos.

Para marcar una dependencia como opcional, se puede utilizar la anotación `@Optional()`:

```
import { Component, Optional } from '@angular/core';
import { UserService } from '../user.service';

@Component({
  selector: 'app-optional-user-list',
  templateUrl: '../optional-user-list.component.html'
})
export class OptionalUserListComponent {
  constructor(@Optional() private userService: UserService) {}
}
```

En este ejemplo, si el servicio `UserService` no está registrado como proveedor, Angular no generará un error y simplemente asignará `null` al parámetro `userService`.

Para inyectar múltiples instancias de una dependencia, se puede utilizar la anotación `@Inject()` con un token y la opción `multi: true` en la configuración del proveedor:

```
import { InjectionToken, Injectable } from '@angular/core';

export const USER_API = new InjectionToken<UserApi>('UserApi', {
  providedIn: 'root',
  factory: () => new UserApi()
});

@Injectable()
export class UserApi {
  // ...
}

@Injectable()
export class AdminUserApi extends UserApi {
  // ...
}

@NgModule({
  providers: [
    { provide: USER_API, useClass: AdminUserApi, multi: true }
  ]
})
export class AppModule { }
```

En este ejemplo, se crean dos clases `UserApi` y `AdminUserApi`, y se registra `AdminUserApi` como un proveedor adicional para el token `USER_API`.

La opción `multi: true` indica que se pueden inyectar múltiples instancias de `UserApi`.

Para inyectar todas las instancias de una dependencia en una clase, se puede utilizar la anotación `@Inject()` con el token:

```
import { Component, Inject } from '@angular/core';
import { UserApi, USER_API } from './user-api.service';

@Component({
  selector: 'app-multi-user-list',
  templateUrl: './multi-user-list.component.html'
})
export class MultiUserListComponent {
  constructor(@Inject(USER_API) private userApis: UserApi[]) {}
}
```

En este ejemplo, Angular inyectará un array de instancias de `UserApi` en el constructor del componente `MultiUserListComponent`.

El componente puede utilizar las instancias para realizar operaciones con diferentes implementaciones de la API de usuario.