Curso Angular 6 Avanzado dia 1

NEORIS

Index

- Servicios
- Ciclo de vida
- Herencia
- Polimorfismo

Concepto de servicio en Angular

Básicamente un servicio es un proveedor de datos, que mantiene lógica de acceso a ellos y operativa relacionada con el negocio y las cosas que se hacen con los datos dentro de una aplica ción. Los servicios serán consumidos por los componentes, que delegarán en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.

Cómo crear un servicio

```
ng generate service clientes
```

Inyectable

La particularidad de las clases de servicios está en su decorador : @Injectable(). Esta función viene en el @angular/core e **indica que esta clase puede ser inyectada** dinámicamente a quien la demande. Aunque es muy sencillo crearlos a mano, el CLI nos ofrece su comando especializado para crear servicios. Estos son ejemplos de instrucciones para crear un *service*.

Injectable

Proveedores funcionales
Servicios como el DisplayService o el EngineService sólo son reclamados por componentes del CarModule. En estos casos interesa que su contenido vaya en el *bundle* del módulo funcional CarModule. Para ello eliminamos de su decoración el providedIn: 'root' y los proveemos explícitamente en la propiedad providersde la decoración usada en car.module.ts

```
1  import { Injectable } from '@angular/core';
2  import { CARS } from './store/cars';
3  import { Car } from './store/models/car.model';
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class CarsService {
9    private cars: Car[] = CARS;
10   constructor() {}
11
12   public getCars = () => this.cars;
13
14   public getCarByLinkId = carId => this.cars.find(c => c.link.ro
15 }
```

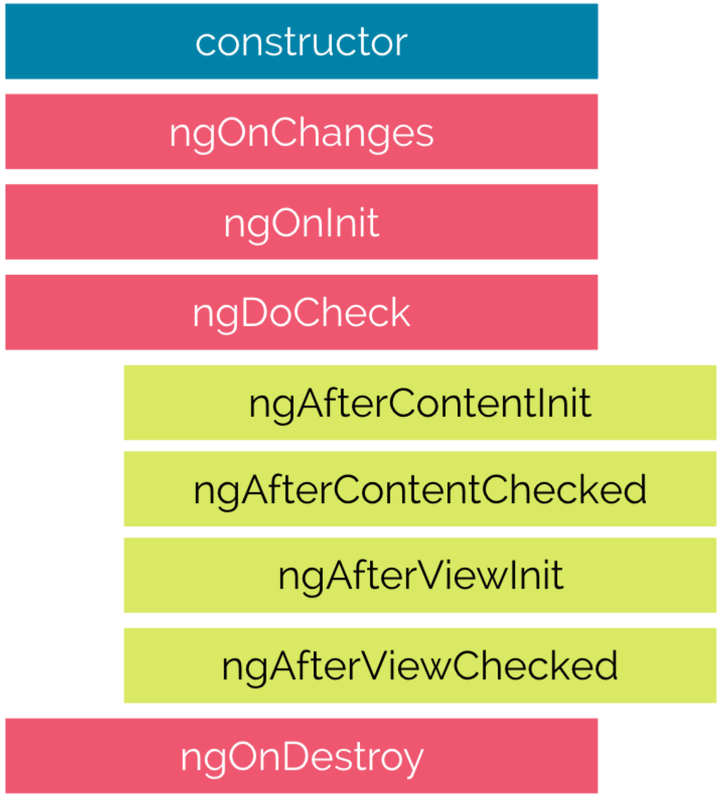
Si le quitamos el root, tenemos que agr egarlo com servicio en el provider

Ejemplo

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

@Injectable({
providedIn: 'root'
})
export class AutosService {

constructor() { }

private autoVW:any ={
marca:'vw',
modelo:'polo',
color:'rojo'
};

getList(){
return this.autoVW;
}
}
```

Ciclo de vida

En Angular, cada componente tiene un ciclo de vida, una cantidad de etapas diferentes que atraviesa. Hay 8 etapas diferentes en el ciclo de vida de los componentes. Cada etapa se denomina lifecycle hook event o en *‘evento de enlace de ciclo de vida’*. Podemos utilizar estos eventos en diferentes fases de nuestra aplicación para obtener el control de los componentes. Como un componente es una clase de TypeScript, cada componente debe tener un método constructor.
El constructor de la clase de componente se ejecuta primero, antes de la ejecución de cualquier otro lifecycle hook. Si necesitamos inyectar dependencias en el componente, el constructor es el mejor lugar para hacerlo. Después de ejecutar el constructor, Angular ejecuta sus métodos de enganche de ciclo de vida en un orden específico.



ngOnChanges: Este evento se ejecuta cada vez que se cambia un valor de un input control dentro de un componente. Se activa primero cuando se cambia el valor de una propiedad vinculada. Siempre recibe un change data map o mapa de datos de cambio, que contiene el valor actual y anterior de la propiedad vinculada envuelta en un SimpleChange

ngOnInit: Se ejecuta una vez que Angular ha desplegado los data-bound properties(variables vinculadas a datos) o cuando el componente ha sido inicializado, una vez que ngOnChanges se haya ejecutado. Este evento es utilizado principalmente para inicializar la data en el componente.

ngDoCheck: Se activa cada vez que se verifican las propiedades de entrada de un componente. Este método nos permite implementar nuestra propia lógica o algoritmo de detección de cambios personalizado para cualquier componente.

ngAfterContentInit: Se ejecuta cuando Angular realiza cualquier muestra de contenido dentro de las vistas de componentes y justo después de ngDoCheck. Actuando una vez que todas las vinculaciones del componente deban verificarse por primera vez. Está vinculado con las inicializaciones del componente hijo.

ngAfterContentChecked: Se ejecuta cada vez que el contenido del componente ha sido verificado por el mecanismo de detección de cambios de Angular; se llama después del método ngAfterContentInit. Este también se invoca en cada ejecución posterior de ngDoCheck y está relacionado principalmente con las inicializaciones del componente hijo.

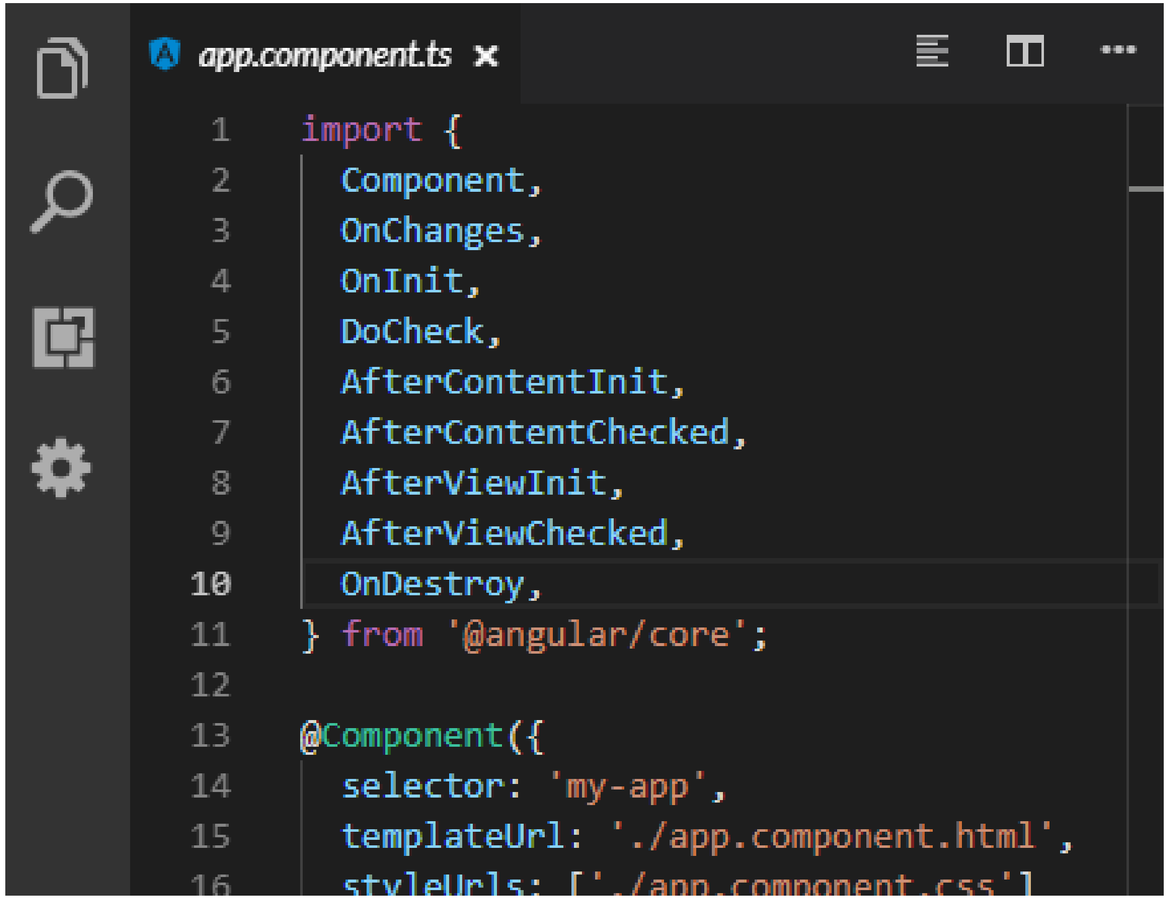
ngAfterViewInit: Se ejecuta cuando la vista del componente se ha inicializado por completo. Este método se inicializa después de que Angular ha inicializado la vista del componente y las vistas secundarias. Se llama después de ngAfterContentChecked. Solo se aplica a los componentes.

ngAfterViewChecked: Se ejecuta después del método ngAfterViewInit y cada vez que la vista del componente verifique cambios. También se ejecuta cuando se ha modificado cualquier enlace de las directivas secundarias. Por lo tanto, es muy útil cuando el componente espera algún valor que proviene de sus componentes secundarios.

ngOnDestroy: Este método se ejecutará justo antes de que Angular destruya los componentes. Es muy útil para darse de baja de los observables y desconectar los event handlers para evitar memory leaks o fugas de memoria.

Ejemplos

```
ngOnInit() {
setInterval(() => {
this.marca = cambio';
}, 1000);
console.log(this.vehiculo.marca);
}

ngDoCheck():void{
console.log('do check!');
}
```



Herencia y Polimorfismo

Ventajas de la herencia de componentes

Entre las ventajas que nos encontramos al hacer uso de la herencia, están las siguientes:

Los decoradores de los componentes derivados sobrescriben cualquier otro decorador definido previamente en el componente heredado.

Si no se define un constructor en el componente derivado, se tomará el **constructor definido en el componente heredado**.

Los *hooks* propios del ciclo de vida de un componente definidos en el componente heredado se ejecutarán aunque no se definan en el componente derivado.

https://codeangular.org/2018/10/30/herencia-de-componentes-en-angular/

www.neoris.com

Dante Panella
Master
Dante.panella@neoris.com

NEORIS